ED1

Dereck Amesquita

10/1/2021

Vectores en R

- `c()`: para definir un vector
- `scan()`: para definir un vector introduciendo datos
- `fix(x)`: para modificar visualmente el vector x
- `rep(a,n)`: para definir un vector constante que contiene el dato a repetido n veces

```
x<-c(1,2,3,4,5)
y<-rep("Dereck", 5)
x
```

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

```
y
```

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

Tipos de Caracteres

- `logical`: lógicos (`TRUE` o `FALSE`)
- `integer`: números enteros, $\mathbb{Z}$
- `numeric`: números reales, $\mathbb{R}$
- `complex`: números complejos, $\mathbb{C}$
- `character`: palabras Cada caracter va en orden de importancia Si en un vector usamos palabras y numeros enteres, todos se considerarn como el mas importante, en este caso todo serian palabras y estarian dentro de comillas.

Secuencias

Se utiliza la funcion "seq" Y se utiliza seq(a,b,by=c). Donde: a es el numero inicial b es el numero final c es el salto de numero en numero

```
seq(0,10,by=2)
```

```
## [1]  0  2  4  6  8 10
```

Si quisieramos un numero exacto de valores en nuestra matriz usamos lenght.out

```r
seq(0,20, length.out = 9)
```

```
## [1]  0.0  2.5  5.0  7.5 10.0 12.5 15.0 17.5 20.0
```

Aplicar funciones a un vector

En R es sumamente sencillo transformar un vector

```r
y= seq(1,10)
z=2+10*y
z
```

```
##  [1]  12  22  32  42  52  62  72  82  92 102
```

```r
u=1:5
elevado=function(x){x^x}
elevado(u)
```

```
## [1]    1    4   27  256 3125
```

```r
#Otra forma
sapply(u, FUN=elevado)
```

```
## [1]    1    4   27  256 3125
```

###Operaciones con vectores

```r
y=1:10
x=11:20
z=x*y
z
```

```
##  [1]  11  24  39  56  75  96 119 144 171 200
```

La funcion cumsum nos da la suma acumulada de un vector La funcion sort nos ordena la matriz La funcion rev invierte el orden de la matriz La funcion cummax nos da el numero mas grande encontrado hasta el momento La funcion cummin nos da el numero mas pequeño encontrado hasta el momento La funcion diff nos da la primera diferencia

```r
x=1:10
cumsum(x)
```

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

```r
y=c(1,4,6,8,2,3)
sort(y)
```

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

```r
cummax(x)
```

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

```r
cummin(x)
```

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

```r
diff(x)
```

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

Dividir vectores

La funcion length nos da el tamaño vector[lenght(vector)] es igua a decir vector[tamaño del vector] por tanto nos dara el ultimo elemento. Se pueden utilizar operadores lógicos:

- ‘‘‘==‘‘‘: =
- ‘‘‘!=‘‘‘: $\neq$
- ‘‘‘>=‘‘‘: $\ge$
- ‘‘‘<=‘‘‘: $\le$
- ‘‘‘<‘‘‘: $<$
- ‘‘‘>‘‘‘: $>$
- ‘‘‘!‘‘‘: NO lógico
- ‘‘‘&‘‘‘: Y lógico
- ‘‘‘|‘‘‘: O lógico

```r
v<-c(5,4,9,8)
v[2]
```

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

```r
v[-c(2:3)] #Con el -c podemos eliminar dicho rango. En este caso eliminamos de la 2 a la 3
```

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

```r
v[v!=8 & v>5] # El unico numero diferente de 8 y mayor a 5 es 9
```

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

Condicionantes

```r
# Onbtener numeros pares
v<- seq(1,50, by=1.5)
v[v%%2==0]
```

```
## [1]  4 10 16 22 28 34 40 46
```

```r
#Obtener numero en posicion par
v[seq(2,50,by=2)]
```

```
##  [1]  2.5  5.5  8.5 11.5 14.5 17.5 20.5 23.5 26.5 29.5 32.5 35.5 38.5 41.5 44.5
## [16] 47.5   NA   NA   NA   NA   NA   NA   NA   NA   NA
```

```r
# Obtener valores booleanos (Debebemos olvidar "[]" y R nos dara los valores T y F)
v>20
```

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

```r
# y/o en matrices
v[v<5|v>45]
```

```
## [1]  1.0  2.5  4.0 46.0 47.5 49.0
```

La funcion which

Lo que hace es darnos la ubicacion del elemento

```r
z<- seq(1,50, by=1.5)
which(z>20)
```

```
##  [1] 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
```

```r
#Desde la posicion 14 hasta la 33 son mayores a 20
z[which(z>20)]
```

```
##  [1] 20.5 22.0 23.5 25.0 26.5 28.0 29.5 31.0 32.5 34.0 35.5 37.0 38.5 40.0 41.5
## [16] 43.0 44.5 46.0 47.5 49.0
```

```r
z[14:33]
```

```
##  [1] 20.5 22.0 23.5 25.0 26.5 28.0 29.5 31.0 32.5 34.0 35.5 37.0 38.5 40.0 41.5
## [16] 43.0 44.5 46.0 47.5 49.0
```

```r
z[z>20]
```

```
##  [1] 20.5 22.0 23.5 25.0 26.5 28.0 29.5 31.0 32.5 34.0 35.5 37.0 38.5 40.0 41.5
## [16] 43.0 44.5 46.0 47.5 49.0
```

```r
which.min(z) #Nos la da pocision del elemento mas pequeño
```

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

Comprobaciones

R puede resolver cualquier expresion y expresarnos y es correcto o no

```r
2*5==5
```

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

```r
#R puede modificar una matriz
y=1:10
y[5]=8 #R remplazara el numero de la posicion 5 con el 8
y[15]=47 #Incluso podemos agregar un elemento mas
length(y)
```

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

```r
is.na(y) #is.na nos devuelve como TRUE las valores que son NA
```

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

```r
which(is.na(y)) #El which nos da las posiciones de los valores TRUE los cuales son NA
```

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

```r
x[which(is.na(y))] #Asi comprobamos que son NA
```

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

```r
mean(y, na.rm=TRUE) #na.rm borra los NA
```

```
## [1] 9.545455
```

Convertiremos los valores ausentes en la media de y

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

```
##  [1]  1.000000  2.000000  3.000000  4.000000  8.000000  6.000000  7.000000
##  [8]  8.000000  9.000000 10.000000  9.545455  9.545455  9.545455  9.545455
## [15] 47.000000
```

Podemos calcular la media sin usar na.rm

Quitar NA de nuestra matriz

Podemos quitar los NA de nuestra matriz mediante "!"

```r
s=1:10
s[15]=20
s[!is.na(s)] #Son los valores que no son NA
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 20
```

```r
mean(s[!is.na(s)])
```

```
## [1] 6.818182
```

Otro recurso "muy forzado" es usar la funcion na.omit. R simplemente nos la matriz sin los valores na. Lo recomendable seria usar !is.na() y ponerlo dentro de una nueva matriz.

```r
s
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 NA NA NA NA 20
```

```r
na.omit(s)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 20
## attr(,"na.action")
## [1] 11 12 13 14
## attr(,"class")
## [1] "omit"
```

```r
p = s[!is.na(s)]
p
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 20
```

Factores en R

¿Que es un Factor?

Es como un vector, pero con una estructura interna más rica que permite usarlo para clasificar observaciones

- `levels`: atributo del factor. Cada elemento del factor es igual a un nivel. Los niveles clasifican las entradas del factor. Se ordenan por orden alfabético
- Para definir un factor, primero hemos de definir un vector y trasformarlo por medio de una de las funciones `factor()` o `as.factor()`.

```r
autos<-c("Kia","Audi","Audi","Nissan","Audi","Kia")
autos.factor=factor(autos)
autos.factor
```

```
## [1] Kia    Audi   Audi   Nissan Audi   Kia
## Levels: Audi Kia Nissan
```

```r
#Son devuelve levels lo cual son los niveles, y son elementos unicos del factor.
```

La función factor()

- `factor(vector,levels=...)`: define un factor a partir del vector y dispone de algunos parámetros que permiten modificar el factor que se crea:
  - `levels`: permite especificar los niveles e incluso añadir niveles que no aparecen en el vector
  - `labels`: permite cambiar los nombres de los niveles
- `levels(factor)`: para obtener los niveles del factor