Programación Estadística: Manejo de Datos

Jocelyn Simmonds (jsimmond@dcc.uchile.cl)

Departamento de Ciencias de la Computación

Texto

Se puede usar la función `which()` para encontrar las posiciones de los elementos de un vector que cumplen una condición:

```
> x <- 1:10
> which(x %% 2 == 0)
[1]  2  4  6  8 10
>
> x2 <- runif(10, 2, 4.5)     # genera numeros al azar (dist. uniforme)
> x2
 [1] 3.966691 4.420197 3.243075 3.012766 3.617560 3.299832 2.524505 2.098100
 [9] 2.113189 4.041362
> which(x2 >= mean(x2))
[1]  1  2  3  5  6 10
```

Texto

Se usan las comillas para crear secuencias de caracteres:

```
1   > x <- "hola"
2   > x
3   [1] "hola"
4   > y <- 'chao'
5   > y
6   [1] "chao"
7   > z <- "hola y chao"
8   > z
9   [1] "hola y chao"
10  > c(x, "y", y)
11  [1] "hola" "y"    "chao"
12  > paste(x, "y", y)
13  [1] "hola y chao"
```

Texto

La función `paste()` se puede usar para generar los rótulos de un gráfico:

```
> labels <- paste("Libro", 1:10)
> labels
 [1] "Libro 1"  "Libro 2"  "Libro 3"  "Libro 4"  "Libro 5"  "Libro 6"
 [7] "Libro 7"  "Libro 8"  "Libro 9"  "Libro 10"
> labels2 <- paste("x", 1:10, 10:4)
> labels2
 [1] "x 1 10" "x 2 9"  "x 3 8"  "x 4 7"  "x 5 6"  "x 6 5"  "x 7 4"  "x 8 10"
 [9] "x 9 9"  "x 10 8"
> labels3 <- paste("Lab", 1:10, sep="")
> labels3
 [1] "Lab1"  "Lab2"  "Lab3"  "Lab4"  "Lab5"  "Lab6"  "Lab7"  "Lab8"  "Lab9"
[10] "Lab10"
```

Si alguno de los argumentos es más corto que el resto, se "repiten" los elementos según sea necesario.

Pruébenlo

Creen los siguientes vectores:

1. “x 1” “y 2” “x 3” “y 4” “x 5”
2. “x 1 10” “y 2 9” “x 3 8” “y 4 7” “x 5 10”

Objetos

Tipos de objetos

- vector: secuencia de elementos, todos del mismo tipo
- factor: vector especial para manejar datos categóricos
- arreglo: colección de elementos de datos, organizados en varias dimensiones (p.ej., cubo de $3 \times 3 \times 3$)
- matriz: arreglo de dos dimensiones
- lista: secuencia de elementos, pueden ser de cualquier tipo (incluyendo listas)
- data frame: lista de vectores del mismo largo

Las funciones también son objetos en R, pero las veremos en la siguiente clase.

Propiedades de objetos

Todo objeto `x` tiene al menos dos propiedades:

1. `mode(x)`: indica el "tipo" del objeto. Ejemplos: numeric, logical, character, etc.
2. `length(x)`: indica el "tamaño" del objeto (no siempre es útil)

```
1  > x <- 1:10
2  > mode(x)
3  [1] "numeric"
4  > length(x)
5  [1] 10
6  >
7  > x <- "hola"
8  > mode(x)
9  [1] "character"
10 > length(x)
11 [1] 1
```

Cambio de mode()

Existen varias funciones as.xxxxx(objeto), que permiten cambiar el mode() de un objeto.

```
> z <- 0:9
> digitos <- as.character(z)
> digitos
 [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9"
>
> d <- as.integer(digitos)
> d
 [1] 0 1 2 3 4 5 6 7 8 9
> d == z
 [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

También existen varias funciones is.xxxxx(objeto), que retornan TRUE si el objeto es del tipo en el nombre de la función.

Pruébenlo

1. Creen un vector de números decimales, vean que pasa cuando aplican `as.integer()`
2. ¿Qué pasa si le aplican `as.integer()` a un vector de texto?
3. . . . y a un vector de valores lógicos?

Vectores

Cambiando el tamaño de un vector

Podemos asignar valores en posiciones mayores que el largo actual:

```
> x <- integer(15)
> x
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
>
> e <- integer()      # vector de largo 0
> e[3] <- 17
> e
[1] NA NA 17          # ojo que rellena con NA en vez de 0
> e[10] <- 3
> e
 [1] NA NA 17 NA NA NA NA NA NA  3
```

Cambiando elementos del vector

Podemos cambiar elementos usando el complemento:

```
1   > e
2    [1] NA NA 17 NA NA NA NA NA NA  3
3   > e[-2] <- 4          # asignar 4 a todo menos la posicion 2
4   > e
5    [1]  4 NA  4  4  4  4  4  4  4  4
6   > e[2] <- 1.3
7   > e
8    [1] 4.0 1.3 4.0 4.0 4.0 4.0 4.0 4.0 4.0 4.0
9   > mode(e)
10  [1] "numeric"
11  > e[c(1, 5, 8)] <- "abc"
12  > e
13   [1] "abc" "1.3" "4"   "4"   "abc" "4"   "4"   "abc" "4"   "4"
14  >
```

Noten que el vector pasa a ser de tipo `numeric` por la asignación en la linea 6, y `character` por la asignación en la linea 11.

Cambiando el tamaño de un vector

También podemos cambiar el valor de `length()`:

```
1   > f
2   [1]  4  4 NA NA NA
3   >
4   > length(f) <- 3
5   > f
6   [1]  4  4 NA
7   >
8   > length(f) <- 13
9   > f
10   [1]  4  4 NA NA NA NA NA NA NA NA NA NA NA
```

En general, no recomiendo hacer esto . . .

Concatenación de elementos

Podemos agregar nuevos elementos a la cola del vector usando `c()`:

```
1   > x <- NULL
2   > x <- c(x, 2)
3   > x
4   [1] 2
5   > x <- c(x, 4)
6   > x
7   [1] 2 4
8   > x <- c(x, 6)
9   > x
10  [1] 2 4 6
11  > x <- c(x, 8)
12  > x
13  [1] 2 4 6 8
14  >
```

Pruébenlo

Creen el vector `x` de largo 15:

1. asignen el valor 1.4 a las posiciones 2, 7 y 9
2. asignen el valor 2.8 a las posiciones 4, 8 y 12
3. . . . y 1 al resto.

Factores

Factores

Podemos crear un `factor` de un vector de datos categóricos:

```
> x <- c("rojo", "azul", "rojo", "rojo", "azul")
> fact_x <- factor(x)
> fact_x
[1] rojo azul rojo rojo azul
Levels: azul rojo           # siempre se muestran en orden lexicografico
>
> levels(fact_x)
[1] "azul" "rojo"
>
> y <- c("rojo1", "azul", "rojo2", "rojo11", "azul")
> fact_y <-factor(y)
> fact_y
[1] rojo1  azul   rojo2  rojo11 azul
Levels: azul rojo1 rojo11 rojo2
```

Los `Levels` de un factor son las categorías que R identifico en los datos.

Factores: uso

Los factores se usan para calcular funciones como el promedio o suma en forma agregada por categoría:

```
> fact_x
[1] rojo azul rojo rojo azul
Levels: azul rojo
> datos <- 1:5
> tapply(datos, fact_x, sum)
azul rojo
   7    8
> tapply(datos, fact_x, mean)
    azul     rojo
3.500000 2.666667
> tapply(datos, fact_x, max)
azul rojo
   5    4
```

Ojo: `datos` y el vector usado para crear `fact_x` debe ser del mismo largo.

Combinando factores

La función `tapply()` recibe como segundo parámetro una lista de factores:

```
> fact_x
[1] rojo azul rojo rojo azul
Levels: azul rojo
> datos <- 1:5
>
> genero <- c("H", "H", "F", "F", "F")
> fact_g <- factor(genero)
> tapply(datos, list(fact_x, fact_g), sum)
      F H
azul 5 2
rojo 7 1
```

Pruébenlo

Creemos de nuevo el vector de edades de los compañeros del curso.
Colectemos además los siguientes datos:

1. preferencias de películas: drama, comedia, acción o misterio?
2. sabor favorito de helado: frutilla, vainilla o chocolate?

Hagan el cruce de datos usando `tapply()`

Arreglos y matrices

Creando un arreglo

En R, un arreglo es un vector con "dimensiones":

```
> z <- numeric(1500)
> z
   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
...
[1444] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[1481] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
>
> dim(z)
NULL
```

Creando un arreglo

```
> dim(z) <- c(3,5,100)
> z
...
, , 99

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

, , 100

     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
> dim(z) <- c(3,6,100)
Error in dim(z) <- c(3, 6, 100) :
  dims [product 1800] do not match the length of object [1500]
```

Creando un arreglo

Puede redefinir las dimensiones si es consistente con número de elementos:

```
> dim(z) <- c(30,50)
> z
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
 [1,]    0    0    0    0    0    0    0    0    0     0     0     0     0
 [2,]    0    0    0    0    0    0    0    0    0     0     0     0     0
...
[30,]    0    0    0    0    0    0    0    0    0     0     0     0     0
...
...
      [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49]
 [1,]     0     0     0     0     0     0     0     0     0     0     0     0
 [2,]     0     0     0     0     0     0     0     0     0     0     0     0
...
[30,]     0     0     0     0     0     0     0     0     0     0     0     0
      [,50]
 [1,]     0
 [2,]     0
...
[30,]     0
```

Orden de elementos

Los elementos de un arreglo están ordenados por columnas. En otras palabras, la matriz

$$M = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$$

se crea de la siguiente forma:

```
> M <- c(11, 21, 12, 22, 13, 23)
> dim(M) <- c(2, 3)
> M
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   21   22   23
>
```

Acceder a elementos por posición

Usaremos [] para acceder a los elementos de un arreglo por posición:

```
> z <- numeric(1500)
> dim(z) <- c(3,5,100)
> z[1,1,99] <- 1
> z[,,99]                    # se muestra la matriz completa del nivel 99
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
```

Si omitimos una posición, R incluye la columna/fila completa.

Acceder a elementos por posición

```
> z[1,1,99] <- 1
> z[,,99]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    0    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
> z[1,3,99] <- 3
> z[,,99]
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    0    3    0    0
[2,]    0    0    0    0    0
[3,]    0    0    0    0    0
> z[1,,98:100]
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]    0    0    0
[3,]    0    3    0
[4,]    0    0    0
[5,]    0    0    0
>
```

Acceder a elementos por posición

Una vez que hayamos indicado las dimensiones de un arreglo, podemos usar `dim()` para ver las dimensiones actuales del arreglo:

```
> dim(z)
[1]   3   5 100
```

De todas formas, podemos acceder a los elementos del arreglo directamente por su posición en el vector de datos:

```
> z[1,1,99] <- 1
> z[1]
[1] 0
> z[1470]       # 3 x 5 x 98
[1] 0
> z[1471]
[1] 1
```

Otra forma de crear un arreglo

También podemos crear un arreglo en un solo paso, usando la función `array(vector de datos, dim = vector de dimensiones)`:

```
> x <- array(1:20, dim=c(4,5))
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Ojo: si el vector de datos es más corto que el número de elementos especificados por las dimensiones, el vector de datos se "reusa" hasta completar el arreglo.

Arreglos de posiciones

Podemos crear un "arreglo de posiciones" para modificar un arreglo:

```
> x <- array(1:20, dim=c(4,5))
> i <- array(c(1, 2, 3, 4, 1, 3), dim=c(3, 2))
> i
     [,1] [,2]
[1,]    1    4
[2,]    2    1
[3,]    3    3
> x[i] <- 0
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9    0   17
[2,]    0    6   10   14   18
[3,]    3    7    0   15   19
[4,]    4    8   12   16   20
```

Ojo: posiciones negativas generan un error; posiciones con algún 0 son ignorados.

Matrices

Matrices son arreglos de 2 dimensiones, pueden ser incluidas en expresiones aritméticas:

```
> C <- array(c(4, 3, 3, 2), dim=c(2,2))
> C
     [,1] [,2]
[1,]    4    3
[2,]    3    2
>
> C + 2
     [,1] [,2]
[1,]    6    5
[2,]    5    4
>
> C * C           # ojo! * no es multiplicacion de matrices!
     [,1] [,2]
[1,]   16    9
[2,]    9    4
```

Operaciones sobre matrices

Sean A, D matrices, b, x vectores y n una constante:

- `t(A)`: retorna A traspuesta
- `nrow(A)`: retorna el número de filas de A
- `ncol(A)`: retorna el número de columnas de A
- `A %*% B`: retorna el producto de las matrices A, B
- `det(A)`: retorna el determinante de A
- `diag(A)`: retorna un vector con la diagonal principal de A
- `diag(x)`: retorna una matriz diagonal, donde los elementos de la diagonal son los valores de x
- `diag(n)`: retorna una matriz identidad de $n \times n$
- `solve(A, b)`: resuelve el sistema de ecuaciones lineares $A \cdot x = b$
- `solve(A)`: retorna la matriz inversa de A

Pruébenlo

Resuelvan el siguiente sistema de ecuaciones lineales:

$$\begin{aligned} 2x + y - 2z &= 10 \\ 3x + 2y + 2z &= 1 \\ 5x + 4y + 3z &= 4 \end{aligned}$$

Listas y Data frames

Listas: elementos de cualquier tipo

`ex` es una lista de tres elementos: uno de tipo `character`, uno de tipo `integer` y uno de tipo `vector`

```
1  > ex <- list(nombre="Juan", num.hijos=2, edades.hijos=c(4, 7))
2  > ex
3  $nombre
4  [1] "Juan"
5
6  $num.hijos
7  [1] 2
8
9  $edades.hijos
10 [1] 4 7
```

Listas: elementos de cualquier tipo

```
> ex <- list(nombre="Juan", num.hijos=2, edades.hijos=c(4, 7))
> ex[3]
$edades.hijos
[1] 4 7

> ex[["num.hijos"]]
[1] 2
>
> ex[[3]]
[1] 4 7
>
> ex[[3]][1]
[1] 4
>
> ex$edades.hijos
[1] 4 7
>
> ex$edades.hijos[2]
[1] 7
```

Listas: elementos de cualquier tipo

R no revisa que los nombres de los elementos de una lista sean únicos, lo que puede causar errores a futuro:

```
> ex2 <- list(x=1, y=2, x=3)
> ex2
$x
[1] 1

$y
[1] 2

$x
[1] 3

> ex2[["x"]]   # toma el primer elemento con nombre = x
[1] 1
> ex2[[1]]
[1] 1
> ex2[[3]]
[1] 3
```

Editando una listas

Pueden usar `c()` para unir listas:

```
> ex3 <- c(ex, ex2)
> ex3
$nombre
[1] "Juan"

$num.hijos
[1] 2

$edades.hijos
[1] 4 7

$x
[1] 1

$y
[1] 2

$x
[1] 3
```

Editando una listas

Tambien pueden usar `length()` para acortar una lista:

```
> length(ex3)
[1] 6
> length(ex3) <- 3
> ex3
$nombre
[1] "Juan"

$num.hijos
[1] 2

$edades.hijos
[1] 4 7
```

Data frame

Un data frame es una lista donde todas las columnas son del mismo largo:

```
> mtcars
                   mpg cyl  disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
...
```

Un data frame incluye nombres de filas y nombres de columnas.

R trae varios data frames de prueba, pueden listarlos usando la función `data()`.

Creando un data frame

Pueden crear un data frame de las siguientes formas:

- de forma manual, usando la función `data.frame()`,
- convirtiendo una lista de listas, usando la función `as.data.frame()`,
- ...o cargando los datos de un archivo externo, usando la función `read.table()`

Leyendo datos de un archivo de texto

Para usar `read.table()`, los datos deben estar en el siguiente formato:

- la primera fila del archivo tiene los nombres de las n columnas
- el resto de las filas contienen los datos de los registros ($n+1$ datos porque incluye un identificador de fila)

Ejemplo: `houses.data`

```
1  Price Floor Area Rooms Age Cent.heat
2  01 52.00 111.0 830 5 6.2 no
3  02 54.75 128.0 710 5 7.5 no
4  03 57.50 101.0 1000 5 4.2 no
5  04 57.50 131.0 690 6 8.8 no
6  05 59.75 93.0 900 5 1.9 yes
```

Ahora podemos cargar los datos:

```
1  > # archivo debe estar dentro del workspace
2  > prices <- read.table("houses.data")
```

Acceso a los componentes de un data frame

Usaremos el signo $ para indicar la componente que nos interesa:

```
> prices <- read.table("houses.data")
> prices
   Price Floor Area Rooms Age Cent.heat
01 52.00   111  830     5 6.2        no
02 54.75   128  710     5 7.5        no
03 57.50   101 1000     5 4.2        no
04 57.50   131  690     6 8.8        no
05 59.75    93  900     5 1.9       yes
> prices$Floor
[1] 111 128 101 131  93
> prices[[1]]
[1] 52.00 54.75 57.50 57.50 59.75
> prices["03", "Rooms"]
[1] 5
> prices["03", 1:5]
   Price Floor Area Rooms Age
03  57.5   101 1000     5 4.2
```

Editando un data frame

R provee una interfaz para editar los elementos de un data frame:

```
1  > prices_new <-edit(prices)
```

```
> prices
   Price Floor Area Rooms Age Cent.heat
01 52.00   111  830     5 6.2        no
02 54.75   128  710     5 7.5        no
03 57.50   101 1000     5 4.2        no
04 57.50   131  690     6 8.8
05 59.75    93  900     5 1.9
> prices$Floor
[1] 111 128 101 131  93
> prices[[1]]
[1] 52.00 54.75 57.50 57.50 59.7
> prices["03", Rooms]
Error in `[.data.frame`(prices,
> prices["03", "Rooms"]
[1] 5
> prices["03", 1:5]
   Price Floor Area Rooms Age
03  57.5   101 1000     5 4.2
> prices_new <-edit(prices)
```

R Data Editor

Copy Paste Quit

	row.names	Price	Floor	Area	Rooms	Age	Cent.heat
1	01	52	111	830	5	6.2	no
2	02	54.75	128	710	5	7.5	no
3	03	57.5	101	1000	5	4.2	no
4	04	57.5	131	690	6	8.8	no
5	05	59.75	93	900	5	1.9	yes
6	06	45	76	680	4	1.5	yes
7							
8							
9							
10							
11							

Editando un data frame

Pueden editar el data frame original en forma directa:

```
> fix(prices)
```

O pueden crear un nuevo data frame:

```
> nuevo <- edit(data.frame())
```

Pruébenlo

Usen los datos de peliculas y sabores de helados para crear su propio archivo de datos. Carguen estos datos en un data frame usando `read.table()`

Creando un data frame en forma manual

Pueden crear un data frame a partir de vectores existentes:

```
> nombres <- c("Ana Maria", "Doris", "Paula", "Pedro", "Juan")
> apellidos <- c("Jerez", "Said", "Soto", "Perez", "Soto")
> edades <- c(23, 65, 44, 45, 33)
> personas <- as.data.frame(list(nombres, apellidos, edades))
> personas
  c..Ana.Maria....Doris....Paula....Pedro....Juan..
1                                        Ana Maria
2                                            Doris
3                                            Paula
4                                            Pedro
5                                             Juan
  c..Jerez....Said....Soto....Perez....Soto.. c.23..65..44..45..33.
1                                      Jerez                      23
2                                       Said                      65
3                                       Soto                      44
4                                      Perez                      45
5                                       Soto                      33
```

Falta crear los nombres de las componentes.

Creando un data frame en forma manual

Pueden usar `names()` para definir los nombres de las componentes de un data frame:

```
> names(personas) <- c("Nombre", "Apellido", "Edad")
> personas
      Nombre Apellido Edad
1 Ana Maria    Jerez   23
2      Doris     Said   65
3      Paula     Soto   44
4      Pedro    Perez   45
5       Juan     Soto   33
> personas$Apellido
[1] Jerez Said  Soto  Perez Soto
Levels: Jerez Perez Said Soto
> personas$Edad
[1] 23 65 44 45 33
```