Universidad de Salamanca
Facultad de Ciencias
Grado en Estadística

Curso 2017/2018

Informática II

Ejercicios

Natalia Alonso Moreda
Pilar Franco Martín
Lucia Moraleja Alonso

ÍNDICE GENERAL

1

EJERCICIOS DE INTRODUCCIÓN A R

1.1 EJERCICIO 1

La función mean calcula la media aritmética de una serie de valores. ¿Cuáles son los argumentos de dicha función?

Función genérica para la media aritmética (recortada).

mean(x , . . .)

Sus argumentos son:

- **x**
  Un objeto R Actualmente existen métodos para vectores numéricos / lógicos y objetos de fecha, fecha, hora e intervalo de tiempo. Los vectores complejos solo están permitidos para trim = 0.
- **trim**
  La fracción (0 a 0.5) de las observaciones que se recortarán desde cada extremo de x antes de calcular la media. Los valores de compensación fuera de ese rango se toman como el punto final más cercano.
- **na.rm**
  Un valor lógico que indica si los valores de NA deben eliminarse antes de que continúe el cálculo.
- **...**
  Otros argumentos pasados a/o desde otros métodos.

1.2 EJERCICIO 2

Dado un conjunto de valores, sabemos que existe una función en R que calcula el máximo, pero no recordamos su nombre. ¿Serías capaz de encontrarla a través de la ayuda?

```
apropos("max")
[1] "cummax"      "max"         "max.col"     "pmax"
"pmax.int"    "promax"      "varimax"     "which.max"

help(max)
```

La función que buscamos es **max**, la cual tiene los siguientes argumentos:

- **...**
  Argumentos numéricos o de caracteres.
- **na.rm**
  Un valor lógico que indica si los valores faltantes deben eliminarse.

1.3 EJERCICIO 3

Queremos utilizar el comando help.search para obtener información sobre la función plot. ¿Cómo lo harías?

```
help.search("plot")

# O abreviadamente
?? "plot"
```

1.4 EJERCICIO 4

Hemos visto que la función apropos nos permite realizar búsquedas de objetos.[1] Siguiendo esos ejemplos:

a) Busca los objetos cuyo nombre empiece por var.

```
apropos("^var")
```

b) Busca los objetos cuyo nombre tenga de 5 a 7 caracteres

```
apropos("^.{5,7}$")
```

1.5 EJERCICIO 5

Las edades de un grupo de amigos son 27, 23, 29, 24 y 31 años. Crea un vector edades con estos datos y calcula su media, de forma que la salida se guarde en un fichero llamado amigos.

```
edades <- c(27,23,29,24,31)
media <- mean(edades)
write(media, file="amigos")
```

Vuelve a calcularla pero de manera que ahora el resultado salga por pantalla.

```
edades <- c(27,23,29,24,31)
media <- mean(edades)
print(media)
```

1.6 EJERCICIO 6

¿Cómo obtienes el archivo de texto con los últimos comandos ejecutados?

Hay dos opciones:

- Manualmente, pulsando en la ventana history y pulsando el símbolo de guardar.
- Mediante un comando:

```
#Con la extension de R para verlo como un historial
savehistory("rutaAlDirectorio/nombreDelArchivo.Rhistory")

#Con la extension deseada, puede omitirse la extension
savehistory("rutaAlDirectorio/nombreDelArchivo.extensionElegida")
```

1.7 EJERCICIO 7

Hemos visto diferentes formas de definir vectores con R. Supongamos que queremos definir el vector x = (1, 2, 3, 4, 5). Decláralo de tres formas equivalentes.

```
x <- c(1,2,3,4,5)
x <- seq(from=1, to=5)
x <- 1:5
assign("x", c(1,2,3,4,5))
assign("x", seq(from=1, to=5))
assign("x", 1:5)
```

1.8 EJERCICIO 8

En muchas ocasiones nos interesa hacer referencia a determinadas componentes de un vector. Hemos visto que para ello utilizaremos los corchetes [ ] . Crea el vector x = (2, −5, 4, 6, −2, 8). A partir de dicho vector define:
a) y = (2, 4, 6, 8).
b) z = (−5, −2).
c) v = (−5, 4, 6, −2, 8).
d) w = (2, 4, −2).

```
x <- c(2,-5,4,6,-2,8)
y <- x[which(x > 0)]
z <- x[which(x < 0)]
v <- x[2:6] # Tambien vale x[seq(2,length(x))] o x[2:length(x)]
w <- x[seq(1,length(x),2)]
```

1.9 EJERCICIO 9

Sabemos que para sumar vectores éstos deben tener la misma longitud. Sin embargo R trabaja de manera distinta. Define los vectores x = (1, 2, 3, 4,5, 6), y = (7, 8), z = (9, 10, 11, 12).

```
x <- 1:6
y <- 7:8
z <- 9:12
```

Calcula:
a) x + x

```
print(x+x)
[1]  2  4  6  8 10 12
```

Ha sumado cada posición consigo misma.
b) x + y. ¿Qué ha hecho R?

```
print(x+y)
[1]  8 10 10 12 12 14
```

Como y tiene longitud inferior a x, ha sumado a cada posición de x la posición de y correspondiente, y una vez no hay más elementos del vector y para sumar con el vector x (se ha llegado al final de y), continúa sumando con el siguiente elemento de x, pero empieza de nuevo con el primer elemento de y, así hasta llegar al último elemento de x.
c) x + z. Ahora R da un warning pero aun así nos da un resultado. ¿Cómo lo ha calculado?

```
print(x+z)
[1] 10 12 14 16 14 16
Warning message:
In x + z :
  longitud de objeto mayor no es multiplo de la longitud de uno menor
```

Ha hecho el mismo preceso que en el apartado b, pero ahora como la longitud de los sumandos no son múltiplos, da un warning ya que algunos elementos de uno de los sumandos se ha sumado menos veces que los demás, en concreto en este caso los 2 primeros elementos de z se han sumado con 2 elementos de x, pero los dós últimos solo se han sumado con 1 elemento de x.

1.10 EJERCICIO 10

Define el vector x = (1, 2, 3, 4, 5, 6). A partir de dicho vector se han construido las matrices m1, m2, m3, m4:

¿m1 [,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
¿m2 [,1] [,2]
[1,] 1 4
[2,] 2 5
[3,] 3 6
¿m3 [,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6
¿m4 [,1] [,2] [,3]
[1,] 1 4 1
[2,] 2 5 2
[3,] 3 6 3

Todas las matrices se han definido a partir de matrix(x,...). Intenta reproducir el código necesario para obtener cada una de ellas.

```
m1 <- matrix(x, nrow=2, ncol=3)
# o tambien matrix(x, nrow=2, ncol=3, byrow=FALSE)

m2 <- matrix(x, nrow=3, ncol=2)
# o tambien matrix(x, nrow=3, ncol=2, byrow=FALSE)

m3 <- matrix(x, nrow=2, ncol=3, byrow=TRUE)
```

```
m4 <- matrix(c(x,x[1:3]), nrow=3, ncol=3)
# o tambien matrix(c(x,x[1:3]), nrow=3, ncol=3, byrow=FALSE)
# o m4 <- matrix(c(x,x), nrow=3, ncol=3)
# byrow es FALSE por defecto, por eso en algunas
# podemos eliminar su declaracion.

# De otra manera:
m1 <- rbind(x[seq(1,length(x),2)], x[seq(2,length(x),2)])
#Tambien vale cbind(x[1:2], x[3:4], x[5:6])
m2 <- cbind(x[1:3], x[4:6])
#Tambien vale rbind(x[seq(1,length(x),4)], x[seq(2,length(x),4)],
# x[seq(3,length(x),4)])
m3 <- rbind(x[1:3], x[4:6]) #Tambien vale t(m2)
#Tambien vale cbind(x[seq(1,length(x),4)], x[seq(2,length(x),4)],
# x[seq(3,length(x),4)])
m4 <- cbind(x[1:3], x[4:6], x[1:3]) #Tambien vale cbind(m2, x[1:3])
```

1.11 EJERCICIO 11

¿Cuál es la diferencia entre *, %* % y outer() ? Compruébalo con las matrices:
A= (2,3 B=(3,8)
1,4)

- **A*B** Este operador multiplica cada posición [i,j] de la matriz A por la posición [i,j] de la matriz B.

```
> A*A
     [,1] [,2]
[1,]    4    9
[2,]    1   16

> B*B
     [,1] [,2]
[1,]    9   64
```

  Las multiplicaciones A*B y B*A no es posible realizarlas ya que las matrices no tienen la misma dimensión.

```
Error in A * B : arreglos de dimenson no compatibles
```

- **A %* %B** Este operador realiza la multiplicación de matrices, por lo que el número de columnas de A tiene que ser igual al número de filas de B. No es posible realizar esta

multiplicación de matrices en el caso A %* %B, ya que A tiene 2 columnas y B una sola fila.

```
Error in A %* %B : argumentos no compatibles
```

Pero si es posible realizar esta multiplicación en el orden inverso, B %* %A:

```
> B %* %A
     [,1] [,2]
[1,]   14   41
```

- **outer(A,B)** Esta operación representa el producto externo de dos matrices.
  El producto externo de las matrices X e Y es la matriz A con la dimensión c (dim (X), dim (Y)) donde el elemento A [c (arrayindex.x, arrayindex.y)] = FUN (X [arrayindex.x], Y [arrayindex.y], ...). La función por defecto es * (FUN="*").
  Se puede establecer otra función diferente a la multiplicación en el producto externo, con el tercer argumento de la función FUN.

```
> outer(A,A)
, , 1, 1

     [,1] [,2]
[1,]    4    6
[2,]    2    8

, , 2, 1

     [,1] [,2]
[1,]    2    3
[2,]    1    4

, , 1, 2

     [,1] [,2]
[1,]    6    9
[2,]    3   12

, , 2, 2

     [,1] [,2]
[1,]    8   12
[2,]    4   16
```

```
> outer(B,B)
, , 1, 1

     [,1] [,2]
[1,]    9   24

, , 1, 2

     [,1] [,2]
[1,]   24   64
```

```
# Cambiando la operacion por defecto:
> outer(B,B, FUN="/")
, , 1, 1

     [,1]     [,2]
[1,]    1 2.666667

, , 1, 2

      [,1] [,2]
[1,] 0.375    1

> outer(A,B)
, , 1, 1

     [,1] [,2]
[1,]    6    9
[2,]    3   12

, , 1, 2

     [,1] [,2]
[1,]   16   24
[2,]    8   32
```

```
> outer(B,A)
, , 1, 1

     [,1] [,2]
[1,]    6   16

, , 2, 1

     [,1] [,2]
[1,]    3    8

, , 1, 2

     [,1] [,2]
[1,]    9   24

, , 2, 2

     [,1] [,2]
[1,]   12   32
```

1.12 EJERCICIO 12

Un grupo de amigos está formado por Ana de 23 años, Luis de 24 años, Pedro de 22, Juan de 24, Eva de 21 y Jorge de 22 años. Crea los vectores correspondientes a nombre, edad y sexo. (Usa la codificación M=mujer, H=hombre). Convierte el vector sexo en un factor sexf. ¿Cuáles son los niveles de dicho factor?

```
nombres <- c("Ana", "Luis", "Pedro", "Juan", "Eva", "Jorge")
edades <- c(23,24,22,24,21,22)
sexos <- c("M", "H", "H", "H", "M", "H")
sexf <- as.factor(sexos)
```

```
> table(sexf)
sexf
H M
4 2
```

Con la función table(...) analizamos los niveles del factor sexf, y obtenemos que H tiene un nivel 4, y M tiene un nivel 2.

1.13 EJERCICIO 13

Con los datos anteriores crea un dataframe que se llame amigos.

```
amigos <- data.frame(nombre=nombres, edad=edades, sexo=sexos)
```

```
> amigos
  nombre edad sexo
1    Ana   23    M
2   Luis   24    H
3  Pedro   22    H
4   Juan   24    H
5    Eva   21    M
6  Jorge   22    H
```

Obtenemos un dataframe con 6 observaciones y 3 atributos para cada observación.

BIBLIOGRAFÍA

[1] REGULAR EXPRESSIONS R, *A collect of regular expressions at R*, Consultado el 10/03/2018, Disponible en `http://www.diegocalvo.es/expresiones-regulares-en-r/`