Índice

Capítulo 1

Introducción al R

1.1. Introducción

R es un software que comenzó como un *clon* del paquete (no gratuito) S-Plus, que era un compendio de aplicaciones estadísticas que utilizaban el lenguaje S, lenguaje diseñado por la compañía AT&T's Bell Laboratories, en principio, para su uso interno. S-Plus consiguió una gran difusión en los últimos años del siglo XX y fueron dos profesores de la Universidad de Auckland (Nueva Zelanda), Ross Ihaka y Robert Gentleman los que elaboraron una versión reducida de S para tareas docentes. La R, inicial del nombre de pila de ambos profesores, sirvió de denominación al nuevo paquete estadístico.

En 1995 Martin Maechler les convenció para su distribución gratuita, estando disponible las primeras versiones piloto (denominadas con un 0 en el primer dígito) en 1999. Hoy en día, existen numerosas aportaciones (todas de libre distribución) programadas en R, las cuales se pueden obtener en la página web de donde se obtuvo R.

No estará de más hacer una advertencia y es que nadie se responsabiliza de los resultados obtenidos con R, dado el carácter de libre distribución del software. No obstante, nosotros sí nos responsabilizamos de los cálculos que aparecen en este texto ya que han sido verificados por el autor.

Una última observación: al abrir R aparecerá la *línea de comandos*, que es aquella que comienza con el símbolo $>$, y será en esta línea en donde deberemos *teclear* las instrucciones que queramos sean ejecutadas por R. En los ejemplos del libro incluiremos este símbolo, el cual, lógicamente no debe ser tecleado si queremos reproducirlos.

Como dijimos en el Prólogo del libro, dado que vamos a analizar todos los ejemplos con el Paquete R, es conveniente empezar conociendo este software en profundidad. A ello dedicaremos este primer capítulo. Es muy interesante que, mientras lo va leyendo, tenga abierto R para ir reproduciendo las instrucciones

Si queremos conocer el modo o la longitud de un vector deben usarse las funciones `mode` y `length`.

La forma más sencilla de crear un vector es mediante la función `c`. Por ejemplo, para crear el vector `x` formado por los números 1, 2 y 5, y conocer su modo y longitud ejecutaríamos la secuencia,

```
> x<-c(1,2,5)
> mode(x)
[1] "numeric"
> length(x)
[1] 3
```

Si los elementos de un vector son del modo *carácter*, debemos incluir dichos elementos no numéricos entre comillas. Así, ejecutando la expresión

```
> y<-c("Pepe","Juan","Luis Alfredo")
```

crearemos el vector `y` el cual tendrá tres elementos.

Obsérvese que los elementos de un vector pueden ser otros vectores. Por ejemplo, podríamos crear ahora el vector `z` formado por cinco elementos no numéricos de la forma

```
> z<-c("Felipe","Miguel Angel",y)
```

y ahora podemos hacer

```
> z
[1] "Felipe"   "Miguel Angel"   "Pepe"   "Juan"   "Luis Alfredo"

> mode(z)
[1] "character"

> length(z)
[1] 5
```

Una forma de crear un vector de números enteros consecutivos entre, por ejemplo, -7 y 7, es ejecutar `(1)`. Ejecutando `(2)` vemos que lo hemos hecho bien. Si queremos obtener (o renombrar) uno o varios elementos de un vector, bastaría con ejecutar `(3)` o `(4)`, dependiendo si queremos obtener los elementos 4 y 5 del vector, o renombrar, con el valor $0'5$, el tercero.

```
> x<-c(-7:7)                                                    (1)
> x                                                             (2)
 [1] -7 -6 -5 -4 -3 -2 -1  0  1  2  3  4  5  6  7
> x[4:5]                                                        (3)
[1] -4 -3
> x[3]<-0.5                                                     (4)
> x
 [1] -7.0 -6.0  0.5 -4.0 -3.0 -2.0 -1.0  0.0  1.0  2.0  3.0  4.0  5.0  6.0  7.0
```

Se pueden ejecutar funciones a los vectores y éstas se aplicarán a todos los elementos del vector. Por ejemplo:

```
> x<-c(2:6)
> x
[1] 2 3 4 5 6
> sqrt(x)
[1] 1.414214 1.732051 2.000000 2.236068 2.449490
```

Se pueden multiplicar, sumar, etc., vectores de la misma longitud, obteniendo así un nuevo vector en donde se han multiplicado, sumando, etc., los elementos de cada uno de los vectores:

```
> x1<-c(0:4)
> x2<-c(2:6)
> x1
[1] 0 1 2 3 4
> x2
[1] 2 3 4 5 6
> x1*x2
[1] 0 3 8 15 24
```

Una situación muy habitual es que tengamos nuestros datos en un fichero *ascii*, llamado, por ejemplo, `datos.txt`. En ese caso, con objeto de crear un vector (no una matriz) debemos utilizar la función `scan`, direccionando el fichero. Así, si nuestros datos estuvieran, por ejemplo, en el USB `e:`, deberíamos ejecutar la expresión

```
> valores<-scan("e:\\datos.txt")
```

con lo que habríamos creado el vector de datos denominado `valores`. Lógicamente, si el *device* en donde están los datos no es `e:`, sustituiríamos éste por el correspondiente. También deberíamos direccionar los datos si están en un

subdirectorio. Por ejemplo, si estuvieran en el subdirectorio `curso` y los vamos a incorporar desde el *device* `d:`, deberíamos ejecutar

```
> valores<-scan("d:\\curso\\datos.txt")
```

Si queremos poner nombres a los elementos de un vector, podemos utilizar la función `names`, creando así un *vector de nombres* de la misma longitud que el vector. Por ejemplo, si queremos crear un vector formado por el 7, el 4 y el 3, luego asignarle a esos elementos los nombres *Primer examen*, *Segundo examen* y *Tercer examen* y, por último, comprobar el resultado, tendríamos que ejecutar la siguiente secuencia de comandos

```
> z<-c(7,4,3)
> names(z)<-c("Primer examen","Segundo examen","Tercer examen")
> z
 Primer examen Segundo examen  Tercer examen
             7              4              3
```

Si sólo queremos conocer los nombres del vector `z`, ejecutaríamos la expresión `names(z)`.

En ocasiones tenemos matrices de datos en donde los valores de una variable de tipo cualitativo vienen codificados de manera que R podría confundirlos con valores de una variable cuantitativa. Por ejemplo, si tenemos un matriz de datos como la siguiente que hemos denominado `datos2.txt`,

```
Peso Sexo
79  001
83  001
56  101
```

en donde hemos codificado el Sexo con dos números, la incorporamos a R ejecutando

```
> datos2<-read.table("e:\\datos2.txt",header=T,colClasses=c("numeric","factor"))
> datos2
  Peso Sexo
1   79  001
2   83  001
3   56  101
```

Como vemos, con el argumento `colClasses` especificamos a R si la columna es numérica o es un factor.

1.3.2. Factores

El *factor* es un vector de datos no numéricos formado por datos procedentes de categorías; por ejemplo, datos obtenidos al anotar si el individuo es *hombre* o *mujer*. A la hora de ejecutar Métodos Estadísticos, no existe diferencia práctica entre un Factor y un vector de modo *character* aunque sí podría haberla dentro de un *data frame*. De momento, no obstante, no haremos distinción entre estos dos tipos de datos.

1.3.3. Matrices

Como dijimos antes, una matriz es una disposición bidimensional en donde, al igual que ocurría con los vectores, todos los elementos deben ser del mismo *modo*.

Para crear una matriz utilizaremos la función `matrix` con dos argumentos, la función `c`, la cual tendrá a su vez como argumentos los datos a introducir, y el número de columnas que deberá tener la matriz. La matriz se construirá por columnas.

Por ejemplo, si queremos convertir en el objeto-matriz `ejemplo` la matriz

```
  2  33  22  6
  8  19  16  4
```

ejecutaríamos la expresión

```
> ejemplo<-matrix(c(2, 8, 33, 19, 22, 16, 6, 4),ncol=4)
```

Ahora podemos comprobar que si lo hemos hecho bien ejecutando la expresión o sentencia que hemos marcado como `(5)`

```
> ejemplo                                                    (5)
     [,1] [,2] [,3] [,4]
[1,]    2   33   22    6
[2,]    8   19   16    4
```

Podemos utilizar, en lugar del argumento `ncol`, el argumento `nrow`, el cual asigna el número de filas que deberá tener la matriz. No obstante, ésta se seguirá formando por columnas.

Otra posibilidad es utilizar ambos. En el caso de que queramos definir la matriz `z` con solamente cuatro de los ocho datos que teníamos en la matriz `ejemplo` anterior, ejecutaríamos la siguiente expresión

```
> z<-matrix(ejemplo,nrow=2,ncol=2)
```

mediante la cual extraemos una parte de la matriz dada. Para comprobar el resultado obtenido ejecutaríamos

```
> z
```

obteniendo

```
> z
     [,1] [,2]
[1,]    2   33
[2,]    8   19
```

Observemos que R crea las matrices por columnas. Es decir, con los valores aportados por la función `c` va completando columnas. Si quisiéramos que los completara por filas, utilizaríamos el argumento `byrow=T`. Así, podemos ejecutar

```
> matrix(c(2,8,33,19,22,16,6,4),ncol=4,byrow=T)
     [,1] [,2] [,3] [,4]
[1,]    2    8   33   19
[2,]   22   16    6    4
```

Las matrices pueden ser de caracteres pero recordemos que en ellas, todos los elementos tienen que ser del mismo modo. Por ejemplo, el dato `personas`, formado por los seis individuos

```
Juan       Alfredo
Lupita     Enriqueta
Ernesto    Teodiselo
```

se obtendría mediante la secuencia

```
> personas <- matrix(c("Juan", "Lupita", "Ernesto", "Alfredo",
+ "Enriqueta", "Teodiselo"), ncol=2)
```

como comprobamos ejecutando `(6)`

```
> personas                                                    (6)
     [,1]       [,2]
[1,] "Juan"     "Alfredo"
[2,] "Lupita"   "Enriqueta"
[3,] "Ernesto"  "Teodiselo"
```

Si tenemos dos o más vectores del mismo *modo* (es decir, numéricos o de caracteres) y además tienen la misma longitud, se pueden combinar para formar una matriz utilizando la función `cbind`.

Así por ejemplo, si tenemos un vector `x` con los consumos de veinte coches y un vector `y` con los kilómetros recorridos por esos mismos vehículos, se puede formar la matriz `w` de dimensión 20 $\times$2 mediante la expresión

```
> w <-cbind(x,y)
```

La función `cbind` une los vectores por columnas. De forma análoga se podría utilizar la función `rbind`, la cual los uniría por filas.

Además del *modo*, el otro atributo más importante de una matriz es su dimensión. Se puede averiguar mediante la función `dim`. Así, para averiguar la dimensión de la matriz `personas`, ejecutaríamos la expresión

```
> dim(personas)
[1] 3 2
```

que nos indica que es 3$\times$2.

Otra cuestión de interés en la construcción de matrices de datos es el nombre de las filas y columnas. Para poner nombre a las filas y columnas de una matriz se utiliza, dentro de la función `matrix`, el argumento `dimnames` el cual debe ser una lista de exactamente dos componentes, la primera de las cuales da los nombres de las filas de la matriz y la segunda la de los componentes. Así, si queremos poner nombres a las filas y las columnas de la matriz `ejem`

```
     2  33  22  5
     8  19  16  4
```

ejecutaríamos la expresión

```
> ejem<-matrix(c(2, 8, 33, 19, 22, 16, 5, 4), ncol=4,
+ dimnames=list(c("Individuo 1","Individuo 2"),
+ c("Hermanos","Edad","Peso","Escolaridad")))
```

Si queremos comprobar la operación realizada podemos ejecutar el nombre del nuevo objeto creado, obteniendo

```
> ejem
            Hermanos Edad Peso Escolaridad
Individuo 1        2   33   22           5
Individuo 2        8   19   16           4
```

También es posible poner nombres a las filas y columnas de matrices ya creadas; por ejemplo, a la matriz anteriormente creada `z`

```
     2  33
     8  19
```

le podemos asignar nombres ejecutando la expresión

```
> dimnames(z) <- list(c("Individuo 1","Individuo 2"),
+ c("Hermanos","Edad"))
```

con lo que ejecutando `z` obtendríamos

```
> z
            Hermanos Edad
Individuo 1        2   33
Individuo 2        8   19
```

Alternativamente podríamos utilizar las funciones `rownames` o `colnames` para poner sólo los nombres a las filas o columnas de una matriz, por ejemplo, de la matriz `fuma` ejecutando:

```
> fuma<-matrix(c(13,17,18,87,83,82),ncol=2)
> colnames(fuma)<-c("fumadores","no fumadores")
> rownames(fuma)<-c("A","B","C")
> fuma
  fumadores no fumadores
A        13           87
B        17           83
C        18           82
```

Si queremos formar una matriz `A` (por ejemplo de 2 columnas) a partir de los datos de un fichero denominado `datos.txt` que se encuentre en `e:`, ejecutaríamos el siguiente comando

```
> A<-matrix(scan("e:\\datos.txt"),ncol=2)
```

Podemos obtener la traspuesta de una matriz con la función `t`. Por ejemplo, con la matriz antes creada `ejem`, podemos ejecutar

```
> t(ejem)
            Individuo 1 Individuo 2
Hermanos              2           8
Edad                 33          19
Peso                 22          16
Escolaridad           5           4
```

También podemos multiplicar matrices o vectores numéricos (que son un caso particular de matrices aunque aquí considerados como *vector columna* en un sentido algebraico y no como un *vector fila*, cuando era considerado un tipo de dato de R), si tienen las dimensiones adecuadas para poder ser multiplicadas, utilizando la secuencia de símbolos

```
%*%
```

Veamos el siguiente ejemplo:

```
> A<-matrix(c(2,3,1,-1,1,0,0,-2,-1),ncol=3)
> B<-matrix(c(0,1,0),ncol=1)
> A
     [,1] [,2] [,3]
[1,]    2   -1    0
[2,]    3    1   -2
[3,]    1    0   -1
> B
     [,1]
[1,]    0
[2,]    1
[3,]    0
> A%*%B
     [,1]
[1,]   -1
[2,]    1
[3,]    0
```

Se puede también invertir una matriz (cuadrada) mediante la función `solve`

```
> A
     [,1] [,2] [,3]
[1,]    2   -1    0
[2,]    3    1   -2
[3,]    1    0   -1

> solve(A)
           [,1]      [,2]       [,3]
[1,]  0.3333333 0.3333333 -0.6666667
[2,] -0.3333333 0.6666667 -1.3333333
[3,]  0.3333333 0.3333333 -1.6666667
```

Lógicamente, su producto con la matriz originaria será la matriz identidad:

```
> A%*%solve(A)
     [,1]         [,2] [,3]
[1,]    1 0.000000e+00    0
[2,]    0 1.000000e+00    0
[3,]    0 5.551115e-17    1
```

1.3.4. Estructuras de datos

Los objetos-matrices que acabamos de ver tienen una limitación importante: todos sus datos deben ser del mismo *modo*. Es decir, podemos tener *matrices numéricas*, o *matrices de caracteres* o *matrices lógicas*. No obstante, en la mayoría de las situaciones, la *matriz de datos* asociada al experimento aleatorio que estemos considerando, contendrá datos de varios *modos* en diferentes columnas. En este caso, no podremos utilizar objetos-matrices, sino objetos-estructuras de datos.

Para crear una *estructura de datos* o *data frame* podemos utilizar dos funciones: Una, la función `data.frame`, la cual une al igual que la función `matrix`, objetos de varias clases, también por columnas.

Por otro lado, para leer datos procedentes de un fichero externo debemos utilizar la función `read.table`. Por ejemplo, si tenemos la siguiente matriz de datos en el fichero `e:oviedo.txt`

```
Peso  Talla  Sexo  Edad  EstaCivil
65    1.65   F     45     Casado
75    1.80   M     55     Casado
80    1.95   M     47     Casado
67    1.75   F     34     Soltero
```

podemos convertirlo en una estructura de datos denominada `oviedo` ejecutando la siguiente expresión

```
> oviedo<-read.table("e:\\oviedo.txt",header=T)
> oviedo
  Peso Talla Sexo Edad EstaCivil
1   65  1.65    F   45    Casado
2   75  1.80    M   55    Casado
3   80  1.95    M   47    Casado
4   67  1.75    F   34   Soltero
```

El argumento `header=T` es para indicar que la primera línea es la cabecera.

Si los datos estuvieran en otro formato que no fuera `txt`, podríamos utilizar otra función para importarlos. Por ejemplo, si estuvieran en un fichero Excel en formato csv, ejecutaríamos

```
> oviedo<-read.csv("e:\\oviedo.csv",header=T)
```

Este tipo de datos (el *data frame*) es el más habitual utilizado en R y el que se obtiene por defecto utilizando R-commander que es una librería muy recomendable para principiantes en el uso de R (ver la última sección del capítulo).

Aunque no es muy habitual, es posible exportar de R un *data frame* mediante la función `write.table`. Por ejemplo, si `oviedo` es un *data frame* en R, y lo queremos exportar al dispositivo `e:` como `prueba.txt`, debemos ejecutar

```
> write.table(oviedo,file="e:\\prueba.txt",row.names=F)
```

Selección de subconjuntos de un data frame

Dado que, como dijimos más arriba, habitualmente la matriz de datos va a venir en formato *data frame*, es muy útil saber cómo elegir subconjuntos de la matriz de datos aunque, en muchas ocasiones, habrá más de una forma de conseguir los subconjuntos que deseemos. Recomendamos, además, que el lector aprenda a manejar R ejecutando R. No obstante, damos a continuación algunos ejemplos de extracción de parte de un *data frame* considerando el antes construido, `oviedo`.

Si queremos obtener los datos de una columna, digamos la segunda, ejecutaríamos la sentencia

```
> oviedo[,2]
[1] 1.65 1.80 1.95 1.75
```

que sería equivalente a pedirle a R que nos diera los datos de la variable `Talla` de la forma

```
> oviedo$Talla
[1] 1.65 1.80 1.95 1.75
```

De hecho, añadir un dólar al final de un objeto de R implica seleccionar el subconjunto correspondiente a lo que aparece detrás de ese símbolo. Por ejemplo, es muy habitual que después de obtener una gran cantidad de resultados al aplicar una función, sólo deseemos una parte de ellos (quizás para poder utilizarla como argumento de otra función en una *composición de funciones*), parte que se obtiene añadiendo al final de la sentencia relativa a la función, el nombre de lo que queramos obtener con un dólar delante.

Un solo valor es fácilmente extraído indicando su fila y columna. Por ejemplo, el valor del individuo de la segunda fila y segunda columna es

```
> oviedo[2,2]
[1] 1.8
```

Siguiendo con los ejemplos de extracción de datos de un *data frame*, si queremos obtener las columnas tercera y cuarta, ejecutaríamos

```
> oviedo[,3:4]
  Sexo Edad
1    F   45
2    M   55
3    M   47
4    F   34
```

Si queremos extraer las filas segunda, tercera y cuarta, ejecutaríamos

```
> oviedo[2:4,]
  Peso Talla Sexo Edad EstaCivil
2   75  1.80    M   55    Casado
3   80  1.95    M   47    Casado
4   67  1.75    F   34   Soltero
```

O, si las filas que queremos extraer son la primera y la tercera, o las columnas primera y tercera ejecutaríamos, respectivamente,

```
> sele<-c(1,3)
> sele
[1] 1 3

> oviedo[sele,]
  Peso Talla Sexo Edad EstaCivil
1   65  1.65    F   45    Casado
3   80  1.95    M   47    Casado

> oviedo[,sele]
  Peso Sexo
1   65    F
2   75    M
3   80    M
4   67    F
```

Si quisiéramos quitarlas, ejecutaríamos las sentencias con un signo menos delante de la forma:

```
> oviedo[,-sele]
  Talla Edad EstaCivil
1  1.65   45    Casado
2  1.80   55    Casado
3  1.95   47    Casado
4  1.75   34   Soltero

> oviedo[-sele,]
  Peso Talla Sexo Edad EstaCivil
2   75  1.80    M   55    Casado
4   67  1.75    F   34   Soltero
```

También podemos utilizar el nombre de las variables en esta selección ejecutando, por ejemplo,

```
> oviedo[,c("Peso","Edad")]
  Peso Edad
1   65   45
2   75   55
3   80   47
4   67   34
```

Estas instrucciones se pueden combinar, por ejemplo, para conseguir las primeras 3 filas de las variables `Peso` y `Edad` ejecutando:

```
> oviedo[1:3,c("Peso","Edad")]
  Peso Edad
1   65   45
2   75   55
3   80   47
```

Si queremos extraer todos los individuos (todas las filas) para los que alguna variable en particular tome, por ejemplo, un valor mayor o igual (o menor) que algún número determinado ejecutaríamos

```
> oviedo[oviedo$Talla >= 1.80,]
  Peso Talla Sexo Edad EstaCivil
2   75  1.80    M   55    Casado
3   80  1.95    M   47    Casado
```

o igual a algún valor concreto (cualitativo)

```
> oviedo[oviedo$Sexo == "M",]
  Peso Talla Sexo Edad EstaCivil
2   75  1.80    M   55    Casado
3   80  1.95    M   47    Casado
```

o cuantitativo

```
> oviedo[oviedo$Talla == 1.80,]
  Peso Talla Sexo Edad EstaCivil
2   75   1.8    M   55    Casado
```

o combinar varias variables. Por ejemplo, extrayendo todas las mujeres (`Sexo = F`) de `Talla` mayor que $1'65$,

```
> oviedo[oviedo$Talla > 1.65 & oviedo$Sexo=="F",]
  Peso Talla Sexo Edad EstaCivil
4   67  1.75    F   34   Soltero
```

Si queremos combinar varios "valores" de una variable cualitativa podemos actuar de dos formas. Por ejemplo, si tenemos el *data frame*

```
> datos
   A     B
1 12  rojo
```