Prácticas de estadística con R

José María Sarabia
Faustino Prieto
Vanesa Jordá



PIRÁMIDE

Prácticas de estadística con R

JOSÉ MARÍA SARABIA
FAUSTINO PRIETO
VANESA JORDÁ
DEPARTAMENTO DE ECONOMÍA. UNIVERSIDAD DE CANTABRIA

Prácticas de estadística con R

EDICIONES PIRÁMIDE

COLECCIÓN «ECONOMÍA Y EMPRESA»

Director:
Miguel Santesmases Mestre
Catedrático de la Universidad de Alcalá

Edición en versión digital

Está prohibida la reproducción total o parcial de este libro electrónico, su transmisión, su descarga, su descompilación, su tratamiento informático, su almacenamiento o introducción en cualquier sistema de repositorio y recuperación, en cualquier forma o por cualquier medio, ya sea electrónico, mecánico, conocido o por inventar, sin el permiso expreso escrito de los titulares del copyright.

© José María Sarabia, Faustino Prieto y Vanesa Jordá, 2018
© Primera edición electrónica publicada por Ediciones Pirámide (Grupo Anaya, S. A.), 2018
Para cualquier información pueden dirigirse a piramide_legal@anaya.es
Juan Ignacio Luca de Tena, 15. 28027 Madrid
Teléfono: 91 393 89 89
www.edicionespiramide.es
ISBN: 978-84-368-3874-9

A Cori, José María y Belén (JMS).

Dedico este libro a mis padres,
Faustino Prieto Peña y Manuela Mendoza Alcaraz (FP).

A mis padres, Carlos y Pilar (VJ).

Índice

Prólogo 13

1. Manejo de información estadística con R 15

1.1. Instalación de R 15
1.2. Primera sesión de trabajo con R 16
1.3. Tipos de datos en R 18
1.4. Creando nuestra primera base de datos: vectores en R 22
1.5. Matrices en R 28
1.6. Objetos «list» y «dataframe» 36
1.7. Variables cualitativas en R: objeto «factor» 40
1.8. Lectura y escritura de bases de datos 41
1.9. Gráficos en R 43

2. Estadística descriptiva 49

2.1. Conceptos básicos 49
2.2. Medidas de posición 52
2.3. Medidas de dispersión 55
2.4. Variable tipificada 59
2.5. Medidas de forma 61
2.6. Gráficos de estadística descriptiva 65
2.7. Estadística descriptiva y finanzas 73
2.8. Ejercicios propuestos 76

3. Análisis de la desigualdad económica 77

3.1. La curva de Lorenz y el índice de Gini 77

© Ediciones Pirámide

3.2. Otras medidas de desigualdad relativa 84
3.3. Medidas de desigualdad absoluta 88
3.4. Descomposición de medidas de desigualdad 92
3.5. Desigualdad de educación 98
3.6. Curvas de concentración 101
3.7. Ejercicios propuestos 107

4. Distribuciones de probabilidad 109

4.1. Introducción 109
4.2. Distribuciones discretas 110
4.2.1. Distribución binomial 110
4.2.2. Distribución de Poisson 114
4.2.3. Distribución geométrica 116
4.2.4. Distribución binomial negativa 118
4.2.5. Distribución hipergeométrica 120
4.2.6. La ley de Benford 123
4.3. Distribuciones continuas 126
4.3.1. Distribución uniforme continua 126
4.3.2. Distribución normal 129
4.3.3. Distribución lognormal 139
4.3.4. Distribución exponencial 142
4.3.5. Distribución gamma 144
4.4. Ejercicios propuestos 148

5. Inferencia estadística 151

5.1. Introducción 151
5.2. Distribución en el muestreo 152
5.3. Intervalos de confianza 157
5.4. Contrastes de hipótesis 163
5.5. Ejercicios propuestos 180

6. Modelos de regresión 183

6.1. Introducción 183
6.2. Modelo de regresión lineal simple 184
6.3. Modelo de regresión lineal múltiple 196
6.4. Diagnosis del modelo 199
6.5. Multicolinealidad 204
6.6. Predicción 205
6.7. Variable dependiente binaria: regresión logística 207
6.8. Ejercicios propuestos 213

© Ediciones Pirámide

7. Análisis de la varianza 215

7.1. Introducción 215
7.2. Análisis de la varianza con un factor 215
7.3. Análisis de la varianza con dos factores 228
7.4. Análisis de la varianza con dos factores e interacción 234
7.5. Ejercicios propuestos 239

8. Métodos de análisis multivariante 241

8.1. Introducción 241
8.2. Distribución normal bidimensional 241
8.3. Análisis descriptivo de datos multivariantes 249
8.4. Análisis de componentes principales 255
8.5. Ejercicios propuestos 267

9. Análisis Cluster 269

9.1. Introducción 269
9.2. Medidas de asociación 274
9.3. Métodos no jerárquicos de análisis Cluster 284
9.3.1. Método de *k*-medias *(k-means)* 285
9.3.2. Método de *k*-medoides *(k-medoids)* 288
9.4. Métodos jerárquicos de análisis Cluster 290
9.4.1. Método de Ward 291
9.4.2. Método de distancia mínima *(linkage simple)* 302
9.4.3. Método de distancia máxima *(linkage completo)* 303
9.4.4. Método del promedio antre grupos 305
9.4.5. Método del centroide 305
9.4.6. Método del medoide 306
9.5. Ejercicios propuestos 307

10. Redes complejas 309

10.1. Introducción 309
10.2. Creación de redes con *igraph* 311
10.3. Creación de redes con *network* 316
10.4. Creación de redes de topología específica. Conversión *igraph-network* 322
10.5. Medidas de centralidad en redes complejas 325
10.6. Medidas de conectividad en redes complejas 330
10.7. Medidas de agrupamiento en redes complejas 332
10.8. Modelos de formación de redes aleatorias 334
10.9. Ejercicios propuestos 338

Bibliografía 341

© Ediciones Pirámide

Prólogo

Este libro es algo más que un manual para el aprendizaje en R; es la suma de un conocimiento conjunto fruto de nuestro propio aprendizaje. En las siguientes páginas, los usuarios encontrarán una guía fácil y accesible para el manejo de este increíble *software* libre.

Tan importante es crear nuevo conocimiento como hacerlo comprensible y fácilmente accesible a todo el mundo. Por eso, además de acercar R a los nuevos usuarios, abrimos nuevos horizontes para que se despierte su interés en su aplicación sobre temas tan novedosos como la «desigualdad económica» y la «teoría de redes».

El libro incluye también los temas clásicos de estadística, que conforman el cuerpo teórico-práctico de esta disciplina. Tras comenzar con los fundamentos básicos del programa R, continuamos con la parte de estadística descriptiva. El análisis descriptivo de un conjunto de datos supone el primer acercamiento a sus diferentes características y singularidades. Tras el estudio de la desigualdad económica, continuamos con el análisis de los modelos probabilísticos. La especificación de un modelo de probabilidad es el primer aspecto a tener en cuenta para el correcto análisis estadístico de un conjunto de datos. En este sentido, el programa R dispone de una amplia y creciente gama de modelos probabilísticos que sirven de patrón para nuestros datos. La facilidad con la que R maneja las diferentes distribuciones lo hacen superior a muchos de los programas disponibles en el mercado. Los temas de estimación y contraste son tratados en el capítulo de inferencia estadística. Seguimos con los modelos de regresión, tanto lineal simple y múltiple como de variable dependiente binaria. Como continuación se estudian los diversos modelos del análisis de la varianza. Las metodologías básicas del análisis multivariante (distribución normal bivariada, análisis descriptivo multivariante, análisis de componentes principales y análisis Cluster) son tratadas en los dos temas siguientes. El libro termina con el estudio de las redes complejas.

© Ediciones Pirámide

Confiamos en que este manual no solo permitirá un acceso sencillo y práctico a R, sino también una oportunidad para explorar numerosas teorías y escenarios estadísticos de gran interés y aplicación a varias disciplinas, como la economía, la medicina, la ingeniería e incluso el derecho.

Nuestra vocación investigadora va aparejada a la docente —tanto o más importante— porque es capaz de inculcar un sentimiento fundamental en cualquier persona ávida de saber, y es que todos somos capaces de aprender todo lo que nos propongamos con la ayuda y las herramientas adecuadas.

Estamos convencidos de que este manual es una de esas herramientas adecuadas; solo queda la dedicación y la constancia de los usuarios en este viaje a lo largo de más de 300 páginas con las que —al terminar— serán capaces de crear y desarrollar cualquier escenario estadístico posible. Comienza el viaje...

José María Sarabia
Faustino Prieto
Vanesa Jordá

© Ediciones Pirámide

1 Manejo de información estadística con R

En este capítulo estudiaremos cómo manejar la información estadística con el programa R. Comenzaremos con una breve sección dedicada a la instalación del programa, para seguir con los tipos de datos en R. A continuación se introducen las bases de datos: vectores, matrices y otros objetos. En la sección siguiente estudiaremos la lectura y escritura de datos y terminaremos el capítulo con algunos tipos de gráficos.

1.1. INSTALACIÓN DE R

Como primer paso, vamos a instalar R en nuestro ordenador. R es software libre, por lo que es posible instalarlo y utilizarlo libremente (e incluso es posible estudiarlo, modificarlo y redistribuirlo) sin coste alguno. Además, R está disponible para Windows, OS X (Mac) y Linux.

Dicha instalación es sencilla y rápida, y podemos realizarla de la siguiente forma:

1. Accedemos al siguiente enlace: https://www.r-project.org/ y aparecerá una pantalla como en la figura 1.1.

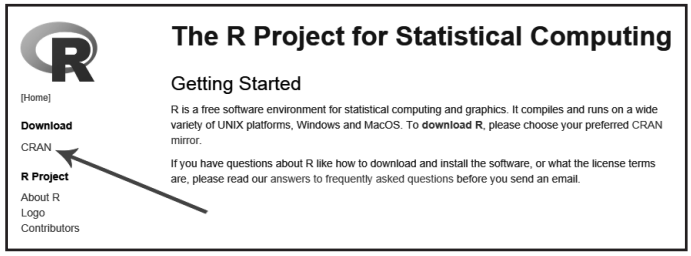


Figura 1.1. Pantalla de inicio de R.

© Ediciones Pirámide

2. A continuación, hacemos «click» en Download CRAN (figura 1.2).

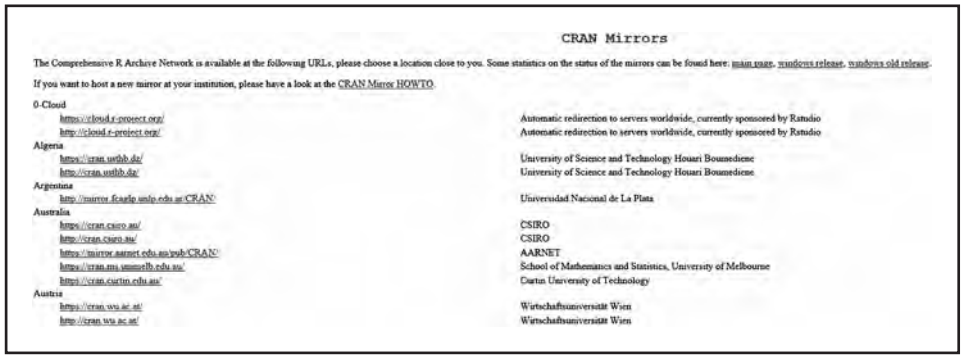


Figura 1.2. Pantalla https://cran.r-project.org/mirrors.html

3. Seleccionamos uno de los CRAN Mirror disponibles (figura 1.3)

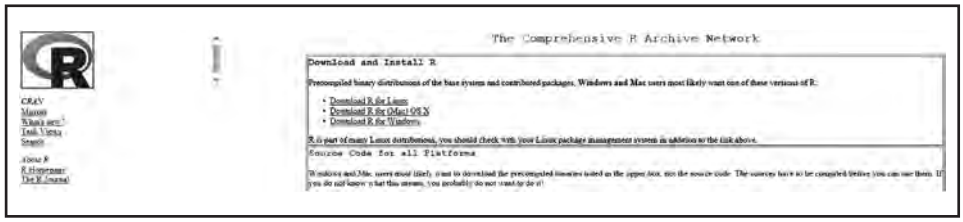


Figura 1.3. Pantalla para cargar R.

4. Finalmente, hacemos «click» en «Download R for ...», en función del sistema operativo que tengamos en nuestro ordenador, y seguimos los pasos que nos indiquen en cada caso.

Tras dicha instalación, tendremos operativa la versión básica de R, cuyas funcionalidades podremos ampliar según nuestras necesidades, tal como veremos en los siguientes capítulos.

1.2. PRIMERA SESIÓN DE TRABAJO CON R

Una vez tengamos instalado R en nuestro ordenador, comenzaremos a utilizarlo. Para ello, realizaremos una primera sesión de trabajo sencilla que nos va a permitir familiarizarnos con la consola de R. Tras hacer «click» en el icono de R, creado en nuestro escritorio durante su instalación, la pantalla que nos apa-

© Ediciones Pirámide

recerá se denomina «R Console», y en ella podremos ver inicialmente el siguiente símbolo:

```
>
```

Si a continuación de este símbolo tecleamos 2+2 y pulsamos la tecla «Enter», obtenemos el resultado:

```
> 2+2
[1] 4
```

A continuación, vamos a realizar en la consola de R las siguientes operaciones: 9−7; 14*4; 5.4/7; 3^4; $\sqrt{36}$; $\sqrt[9]{7^2}$ (tomado el punto como símbolo decimal):

```
> 9-7
[1] 2
> 14*4
[1] 56
> 5.4/7
[1] 0.7714286
> 3^4
[1] 81
> sqrt(36)
[1] 6
> 5^(2/9)
[1] 1.429969
```

Para finalizar esta primera toma de contacto con R, realizaremos las siguientes operaciones: log (1000); $\log_{10}$ (1000); exp (1); e^{-7}; 3*π + 5; 0!; 4!; $\binom{5}{3}$; 7/0:

```
> log(1000)
[1] 6.907755
> log(1000,10)
[1] 3
> exp(1)
[1] 2.718282
```

© Ediciones Pirámide

```
> exp(-7)
[1] 0.000911882
> 3*pi+5
[1] 14.42478
> factorial(0)
[1] 1
> factorial(4)
[1] 24
> choose(5,3)
[1] 10
> 7/0
[1] Inf
```

Como puede comprobarse, por defecto R considera `log` como el logaritmo neperiano. El comando `log(1000,10)` indica el logaritmo decimal. El número combinatorio:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!'}$$

se obtiene con el comando `choose(n,k)`. El símbolo infinito se denota como `Inf`.

1.3. TIPOS DE DATOS EN R

El programa R permite manejar diferentes tipos de datos: «numeric» (números reales), «character» (datos tipo texto), «logical» (verdadero o falso), etc. En esta sección veremos varios ejemplos de ello.

Podemos, por ejemplo, estar interesados en asignar a la variable continua x (altura de un jugador de baloncesto) el valor de 2,13 metros, el cual es un número real, es decir, un dato del tipo «numeric». Dicha asignación la realizaremos con uno de los comandos más utilizados en R, el comando `<-` (equivalente al comando =), escribiendo primero el nombre de la variable, luego dicho comando `<-` y finalmente el dato que queremos asignarle.

Tras ejecutar dicha asignación, si queremos comprobar que se ha realizado correctamente, podemos escribir directamente el nombre de la variable x y pulsar «Enter», y R nos devolverá como resultado el valor guardado en dicha variable:

```
> x<-2.13
> x
[1] 2.13
```


© Ediciones Pirámide

En el ejemplo anterior, hemos utilizado un dato tipo «numeric». Si deseamos asignar un dato tipo «character» o «logical», procederemos de igual forma:

```
> nombre<-"Luis"
> nombre
[1] "Luis"
> desempleado<-F
> desempleado
[1] FALSE
```

Hemos asignado a la variable `nombre` (por ejemplo, nombre de un habitante de nuestra localidad) un dato del tipo «character» (que puede contener uno o más caracteres, y que hay que introducir entre comillas), y hemos asignado a la variable `desempleado` (si dicha persona está desempleada o no) un dato tipo «logical» (en este caso `FALSE`, abreviado como `F`).

`R` distingue entre mayúsculas y minúsculas. Como ejemplo de ello, los posibles valores de los datos tipo «logical», `TRUE` o `FALSE`, se pueden abreviar respectivamente como `T` o `F`, pero no pueden sustituirse por «True», «true», «False» ni «false», en cuyo caso obtendremos un mensaje de error, tal como puede verse en el siguiente ejemplo:

```
> desempleado<-FALSE
> desempleado
[1] FALSE
> desempleado<-False
Error: objeto 'False' no encontrado
> desempleado<-false
Error: objeto 'false' no encontrado
```

A continuación, vamos a describir una serie de comandos útiles en el manejo de datos. En primer lugar, en relación con los datos tipo «numeric»:

```
> # Ejemplo de manejo de datos tipo "numeric"
> a<-2.34
> b<-3.25
> a+b
[1] 5.59
> sum(a,b)
[1] 5.59
```

© Ediciones Pirámide

Hemos sumado los valores reales tipo «numeric», asignados a las variables `a` y `b` de dos formas diferentes, mediante + y mediante el comando `sum`, y hemos introducido un «comentario» (texto precedido por el símbolo #, que `R` interpreta que no es un comando y por tanto no ejecuta) cuyo fin es únicamente aportar información útil al que lee el código de `R`.

En relación con los datos tipo «character», suele ser habitual tener que unir cadenas de texto. Dicha acción es muy fácil de llevar a cabo en `R` con el comando `paste`, escribiendo entre paréntesis y separados por comas todos los textos que queramos unir, en el orden que deseemos que se unan, e indicando al final, con la opción `sep`, la separación que necesitamos que haya entre dichos textos. En el ejemplo siguiente, uniremos tres datos tipo «character», que asignaremos previamente a tres variables, con un espacio de separación entre ellos, y comprobaremos al final que dicha unión se ha realizado correctamente. Para ello escribimos el nombre de la variable donde hayamos guardado el resultado de dicha unión y pulsamos «Enter»:

```
> # Ejemplo de manejo de datos tipo character
> nombre<-“Luis”
> apellido1<-“Pérez”
> apellido2<-“García”
> nombrecompleto<-paste(nombre,apellido1,apellido2,sep=“ ”)
> nombrecompleto
[1] “Luis Pérez García”
```

Con respecto a los datos tipo «logical», es interesante dedicar un comentario a lo que se conoce como «condiciones», «operadores de comparación» y «operadores lógicos», y cómo se introducen dichas instrucciones en `R`.

Un ejemplo de condición: si asignamos a la variable `a` el valor 5, y a la variable `b` el valor 7, la condición «*a* es mayor que *b*» es falsa. Para introducirla en `R` utilizaremos el operador de comparación > (mayor) y el resultado que obtendremos es `FALSE` (resultado tipo «logical»):

```
> a<-5
> b<-7
> a>b
[1] FALSE
```

Otros dos ejemplos de condiciones serían: «*a* es mayor que 3», «*b* es mayor o igual que 7», «*a* es menor que 5», «*a* es menor o igual que *b*», «*a* es igual a 5», «*b* es igual a 3» y «*a* es distinto de *b*»:

© Ediciones Pirámide

```
> a<-5
> b<-7
> a>3
[1] TRUE
> b>=7
[1] TRUE
> a<5
[1] FALSE
> a<=b
[1] TRUE
> a==5
[1] TRUE
> b==3
[1] FALSE
> a!=b
[1] TRUE
```

El comando de R == se utiliza como operador de comparación «igual a», y el símbolo != en R se utiliza como operador de comparación «distinto de».

Podemos considerar condiciones más complejas a las anteriores utilizando el operador lógico «AND», el cual podemos introducir en R como &. Este operador permite valorar si se cumplen dos condiciones simultáneamente. Por ejemplo, con los mismos valores de a y b de los dos ejemplos anteriores la condición (a>3)&(b>=2) es verdadera («*a* es mayor que 3 y *b* es mayor o igual que 2 simultáneamente»), y el resultado que obtendremos es TRUE:

```
> a<-5
> b<-7
> (a>3)&(b>=2)
[1] TRUE
```

Además de «AND», otro operador lógico muy utilizado es «OR», el cual podemos introducir en R como | o bien «OR», y permite valorar si de dos condiciones al menos se cumple una. Un ejemplo se presenta a continuación:

```
> a<-10
> b<-4
> resultado1<-(a!=7)&(b<5)  # TRUE si se cumple a la vez que a
> resultado1                # es distinto de 7 y b menor que 5
[1] TRUE
```

© Ediciones Pirámide

```
> resultado2<-(a>=3)|(b>8)  # TRUE si a es mayor o igual que 3
> resultado2                # o b es mayor que 8 (al menos una)
[1] TRUE
```

El resultado en los dos casos es el valor TRUE, al ser verdaderas tanto la condición establecida con «AND» como la establecida con «OR». La tabla 1.1 incluye un resumen con los operadores de comparación y operadores lógicos utilizados en R.

TABLA 1.1
Operaciones de comparación y lógicos en R

Operador	Significado
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual
==	Igual a
!=	Distinto
&	AND
|	OR

1.4. CREANDO NUESTRA PRIMERA BASE DE DATOS: VECTORES EN R

Normalmente, las bases de datos de nuestro interés pueden tener diferentes estructuras.

Un tipo de estructura común es la de una secuencia de datos del mismo tipo. Este sería el caso de una muestra de n valores: x_1, x_2, ..., x_n de una variable X. Si, por ejemplo, X es el «peso en kilogramos de los miembros de un club de tenis», una muestra de valores de dicha variable es: 80.1, 73.5, 92.0, 65.4, 73.9, 77.2, 81.8, 84.1, 76.7 y 70.3. En este caso el tamaño de muestra es $n = 10$ y todos los datos son de tipo «numeric» (números reales).

En R, estas estructuras de datos ya se encuentran predefinidas en forma de lo que se conoce como «objetos», y en particular en un tipo concreto de objeto conocido como «vector». Por tanto, cuando tengamos una muestra de datos del mismo tipo disponible como secuencia de datos, R nos permitirá manejarla en forma de un objeto predefinido denominado «vector». En caso de que nuestra base de datos tenga una estructura diferente (datos de distinto tipo, datos

© Ediciones Pirámide

disponibles en forma de matriz, etc), recurriremos a otros objetos predefinidos en R («matriz», «list», «dataframe», etc.), que analizaremos en apartados siguientes.

A continuación, vamos a presentar los comandos básicos utilizados en el manejo de vectores.

En primer lugar, creamos la base de datos en R dándole el nombre `misdatos`, escribiendo en la consola de R los siguientes dos comandos:

```
> misdatos<-c(80.1,73.5,92.0,65.4,73.9,77.2,81.8,84.1,76.7,70)
> misdatos
[1] 80.1 73.5 92.0 65.4 73.9 77.2 81.8 84.1 76.7 70.0
```

En el script anterior aparece el código: «`c(...)`», comando que permite crear un vector compuesto por los datos que se incluyen entre paréntesis. Se comprueba, tras escribir a continuación `misdatos`, que dicho vector se ha construido correctamente.

Como el requisito para utilizar el objeto «vector» es que los datos sean del mismo tipo, vamos a crear un vector con datos tipo «character» y otro con datos tipo «logical»:

```
> misdatos2<-c("Javier","Luisa","Isabel","Jose")
> misdatos2
[1] "Javier" "Luisa" "Isabel" "Jose"
> misdatos3<-c(TRUE, FALSE, FALSE,T,F)
> misdatos3
[1] TRUE FALSE FALSE TRUE FALSE
```

Por otro lado, si intentamos crear un vector con datos de distinto tipo, puede verse que R no lo va a permitir:

```
> misdatos4<-c(TRUE,1.2)
> misdatos4
[1] 1.0 1.2
> misdatos5<-c(F,1.2,"hola")
> misdatos5
[1] "FALSE" "1.2" "hola"
```

Podemos ver que al intentar crear un vector `misdatos4` con un dato tipo «logical» y otro de tipo «numeric», R toma los dos como «numeric», asignando

© Ediciones Pirámide

al dato lógico TRUE el valor real 1.0, y también que al intentar crear un vector misdatos5 con un dato tipo «logical», otro de tipo «numeric» y un tercero de tipo «character», R toma los tres como tipo «character», asignando al dato lógico F (FALSE) el texto «FALSE» y al valor real 1.2 el texto «1.2».

A continuación vamos a ver cómo crear en R un vector que contenga una secuencia de valores reales, en el primer caso con una separación de una unidad, y en el segundo caso con una separación diferente:

```
> misdatos6<-10:25
> misdatos6
 [1] 10 11 12 13 14 15 16 17 18 19 20
[12] 21 22 23 24 25
> misdatos7<-1.2:5.2
> misdatos7
[1] 1.2 2.2 3.2 4.2 5.2
> misdatos8<-seq(3,6,by=0.1)
> misdatos8
 [1] 3.0 3.1 3.2 3.3 3.4 3.5 3.6 3.7
 [9] 3.8 3.9 4.0 4.1 4.2 4.3 4.4 4.5
[17] 4.6 4.7 4.8 4.9 5.0 5.1 5.2 5.3
[25] 5.4 5.5 5.6 5.7 5.8 5.9 6.0.1 9.2 9.3 9.4 9.5
[67] 9.6 9.7 9.8 9.9 10.0
```

En la secuencia anterior misdatos6 y misdatos7 son dos vectores creados con el comando «:», el cual genera una secuencia de valores entre los límites indicados a ambos lados del comando (entre 10 y 25 y entre 1.2 y 5.2, respectivamente) y con una separación de una unidad. En el caso de misdatos8, se trata de un vector creado con el comando seq, el cual genera una secuencia de valores a partir de los tres datos escritos entre paréntesis: el primer valor corresponde al límite inferior, el segundo valor corresponde al límite superior y el tercer valor (by=) es la separación entre cada uno de los valores de dicha secuencia.

Hasta este momento, en todos los ejemplos, los resultados que hemos obtenido han aparecido en la consola de R acompañados con el símbolo [1]. En éste último ejemplo, puede verse que esto ha cambiado. En misdatos6 aparece la numeración [1] y [12], y en misdatos8 aparece [1], [9], [17], [25] y [67]. Con estos símbolos, R indica la posición dentro del vector del elemento que se encuentra justo a su lado derecho, y viene a ayudarnos a situar los elementos en aquellos casos en los que no todos entran en la misma línea en la consola de R, cuya anchura hemos decidido nosotros (si la anchura de la consola fuese diferente, podremos obtener una numeración distinta).

Continuamos con los comandos que permiten extraer datos de una base de datos (base de datos con la estructura en R de vector), en primer lugar a partir

© Ediciones Pirámide

de su posición en el vector, y en segundo lugar a partir del cumplimiento o no de las condiciones que indiquemos.

Para conocer cómo extraer datos a partir de su posición, consideramos `misdatos9` como un vector que contiene los números impares menores de 20. A continuación seleccionamos:

— El número impar que ocupa la quinta posición.
— Todos los números impares desde la posición 5 hasta la posición 8, ambas inclusive.
— Los datos con la primera y tercera posiciones.
— Los datos situados en la primera posición y desde la tercera hasta la quinta.

El código se presenta a continuación:

```
> misdatos9<-c(1,3,5,7,9,11,13,15,17,19)
> misdatos9[5]
[1] 9
> misdatos9[5:8]
[1] 9 11 13 15
> misdatos9[c(1,3)]
[1] 1 5
> misdatos9[c(1,3:5)]
[1] 1 5 7 9
```

Cuando son más de una las posiciones de los datos que nos interesan, escribiremos entre corchetes un vector de posiciones creado a partir de los comandos vistos anteriormente. En el caso de posiciones consecutivas, en `misdatos9[5:8]` utilizamos el vector de posiciones `5:8`. En el caso de posiciones no consecutivas, en `misdatos9[c(1,3)]]` utilizamos el vector de posiciones `c(1,3)`. En el caso de una combinación de posiciones consecutivas y no consecutivas, en `misdatos9[c(1,3:5)]` utilizamos el vector de posiciones `c(1,3:5)`.

Para conocer cómo extraer datos a partir de su valor (utilizando una condición) creamos `misdatos10` como vector que contiene los números pares menores de 30. En esta ocasión lo construimos con el comando `seq` en vez de con el comando `c(...)`. A continuación, seleccionamos:

— Los números pares con un valor inferior a 13.
— Los números pares con un valor superior o igual a 18.
— Los números pares distintos a 12.
— Los números pares que se encuentren entre el 12 y el 18 sin que ambos estén incluidos.

© Ediciones Pirámide

La solución se presenta en el siguiente código:

```
> misdatos10<-seq(2,30,by=2)
> misdatos10
 [1] 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30
> misdatos10[misdatos10<13]
[1] 2 4 6 8 10 12
> misdatos10[misdatos10>=18]
 [1] 18 20 22 24 26 28 30
> misdatos10[misdatos10!=12]
 [1] 2 4 6 8 10 14 16 18 20 22 24 26 28 30
> misdatos10[(misdatos10>12)&(misdatos10<18)]
[1] 14 16
```

Puede verse en estos últimos ejemplos que, entre corchetes, lo que figura no es un vector de posiciones, sino una condición, la condición que han de cumplir los datos que deseamos extraer de la base de datos. Dicha condición podrá contener o no operadores lógicos, según el caso.

El programa R permite manejar todos los datos de un vector a la vez, de forma simultánea, lo cual nos facilitará en muchos momentos la programación de las tareas que necesitamos que R realice con los datos. Podemos, por ejemplo, sumar un mismo valor a todos los datos de un vector a la vez, para, por ejemplo, contemplar un cambio de origen de una variable: $X \rightarrow a + X$:

```
> misdatos11<-c(1,4,7.8,10)
> a<-5.4
> misdatos12<-a+misdatos11
> misdatos12
[1] 6.4 9.4 13.2 15.4
```

Podemos, igualmente, multiplicar todos los datos a la vez por un mismo valor, para, por ejemplo, realizar un cambio de escala en una variable $X \rightarrow b\ X$:

```
> misdatos13<-c(8,5,9.0,8.88,0.57)
> b<-3.5674
> misdatos14<-b*misdatos13
> misdatos14
[1] 28.539200 17.837000 32.106600 31.678512 2.033418
```

De una forma equivalente, se puede sumar o multiplicar, elemento a elemento, todos los datos de dos vectores de igual dimensión (con igual número de elementos):

© Ediciones Pirámide

```
> misdatos15<-c(0,1,0,2,0,3,0,4,0,5)
> misdatos16<-c(1,2,3,4,5,6,7,8,9,10)
> misdatos17<-misdatos15+misdatos16
> misdatos17
[1] 1 3 3 6 5 9 7 12 9 15
> misdatos18<-misdatos15*misdatos16
> misdatos18
[1] 0 2 0 8 0 18 0 32 0 50
```

El producto escalar de dos vectores se obtiene con el símbolo `%*%`:

```
> misdatos15<-c(0,1,0,2,0,3,0,4,0,5)
> misdatos16<-c(1,2,3,4,5,6,7,8,9,10)
> misdatos15%*%misdatos16
     [,1]
[1,] 110
```

A continuación, vamos a crear una base de datos tipo vector con un tamaño especificado para utilizar con un determinado tipo de datos, en principio, vacía de contenido. Es decir, vamos a ver cómo podemos crear dicha estructura y darle un nombre, y todo ello sin incluir datos en la misma.

La utilidad de dicha acción es la de crear un espacio vacío previo que podremos ir rellenando con datos de un determinado tipo. Por ejemplo, vamos a crear un vector vacío, con 12 posiciones, para datos tipo «numeric», que llamaremos `misdatos17`; por otro lado, un vector vacío, con 8 posiciones, de datos tipo «character», que llamaremos `misdatos18`, y, finalmente, un vector vacío, con 9 posiciones, para datos tipo «logical», que llamaremos `misdatos19`:

```
> misdatos17<-numeric(12)
> misdatos17
[1] 0 0 0 0 0 0 0 0 0 0 0 0
> misdatos18<-character(8)
> misdatos18
[1] "" "" "" "" "" "" "" ""
> misdatos19<-logical(9)
> misdatos19
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Los resultados que se obtienen varían en función del tipo de datos para el que hayamos creado el vector.

© Ediciones Pirámide

Para finalizar este apartado sobre vectores incluimos dos comandos de uso habitual: `sum` y `length`. El comando `sum` permite sumar todos los componentes de un vector, mientras que el comando `length` permite conocer el número de componentes del vector (dimensión del vector). Presentamos a continuación un ejemplo de estos comandos:

```
> misdatos20<-c(1,2,1,3,1,4,1,5,1,6)
> sum(misdatos20)
[1] 25
> length(misdatos20)
[1] 10
```

1.5. MATRICES EN R

Un segundo tipo de estructura habitual en las bases de datos es el de una matriz de datos, todos del mismo tipo:

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{np} \end{pmatrix}$$

donde el primer subíndice indicaría el número de fila y el segundo el número de columna al que el dato pertenece. Se trata de una matriz de dimensión $n \times p$.

A modo de ejemplo, la siguiente matriz representa las variables (columnas): peso en kilogramos X_1, altura en metros X_2 y precio en euros de su raqueta X_3, para cuatro individuos (filas de la matriz) del mismo club de tenis:

$$\begin{pmatrix} 80.1 & 1.70 & 229 \\ 73.5 & 1.65 & 48.5 \\ 92.0 & 1.78 & 110 \\ 65.4 & 1.60 & 87 \end{pmatrix}$$

Todos los elementos de la matriz son números reales de tipo «numeric». En `R`, este tipo de estructuras de datos también se encuentran predefinidas en forma de un objeto denominado «matrix». En este apartado vamos a estudiar los principales comandos relacionados con en el manejo de matrices.

En primer lugar, creamos la base de datos anterior en `R` dándole el nombre de `mimatriz`, escribiendo los siguientes comandos:

© Ediciones Pirámide

```
> datos<-c(80.1,73.5,92,65.4,1.7,1.65,1.78,1.6,229,48.5,110,87)
> mimatriz<-matrix(datos,4,3)
> mimatriz
     [,1] [,2]  [,3]
[1,] 80.1 1.70 229.0
[2,] 73.5 1.65  48.5
[3,] 92.0 1.78 110.0
[4,] 65.4 1.60  87.0
```

Hemos creado el vector `datos` con el comando `c(...)`, tal como vimos en el apartado anterior. A continuación hemos definido la matriz `mimatriz` con el comando `matrix`, para lo cual hemos introducido, entre paréntesis y separados por comas, los datos, el número de filas y el número de columnas, en este orden. Finalmente, hemos comprobado que la base de datos se ha creado correctamente.

Como puede verse en el resultado obtenido, junto con los datos aparecen numerados entre corchetes el número de fila y el de columna. Si quisiéramos dar un nombre a dichas filas y columnas, se puede hacer de la siguiente manera:

```
> variables<-c("peso(Kg)","altura(m)","precioRaqueta(euros)")
> socios<-c("socio1", "socio2", "socio3", "socio4")
> colnames(mimatriz)<-variables
> row.names(mimatriz)<-socios
> mimatriz
       peso(Kg) altura(m) precioRaqueta(euros)
socio1     80.1      1.70                229.0
socio2     73.5      1.65                 48.5
socio3     92.0      1.78                110.0
socio4     65.4      1.60                 85.8
```

Hemos utilizado los comandos `colnames` y `row.names` para referirnos a las columnas y filas de la matriz `mimatriz`.

Si ahora quisiéramos volver a tener de nuevo las filas y columnas numeradas, podríamos hacerlo de la siguiente forma:

```
> colnames(mimatriz)<-NULL
> row.names(mimatriz)<-NULL
> mimatriz
     [,1] [,2]  [,3]
[1,] 80.1 1.70 229.0
[2,] 73.5 1.65  48.5
[3,] 92.0 1.78 110.0
[4,] 65.4 1.60  87.0
```

© Ediciones Pirámide

A continuación, vamos a crear la misma matriz anterior, `mimatriz`, de dos formas distintas. Si partimos de los datos de cada uno de los cuatro socios, que introducimos en R como cuatro vectores diferentes, y los combinamos como filas mediante el comando `rbind`, obtendremos el mismo resultado que antes:

```
> socio1<-c(80.1,1.7,229)
> socio2<-c(73.5,1.65,48.5)
> socio3<-c(92,1.78,110)
> socio4<-c(65.4,1.6,87)
> mimatriz<-rbind(socio1,socio2,socio3,socio4)
> variables<-c("peso(Kg)","altura(m)","precioRaqueta(euros)")
> colnames(mimatriz)<-variables
> mimatriz
       peso(Kg) altura(m) precioRaqueta(euros)
socio1     80.1      1.70                229.0
socio2     73.5      1.65                 48.5
socio3     92.0      1.78                110.0
socio4     65.4      1.60                 85.8
```

Si partimos de los valores de las tres variables (introducidos como tres vectores diferentes), podemos combinarlos en columnas mediante el comando `cbind`, obteniéndose el mismo resultado:

```
> variable1<-c(80.1,73.5,92,65.4)
> variable2<-c(1.7,1.65,1.78,1.6)
> variable3<-c(229,48.5,110.9,85.8)
> mimatriz<-cbind(variable1,variable2,variable3)
> variables<-c("peso(Kg)","altura(m)","precioRaqueta(euros)")
> socios<-c("socio1", "socio2", "socio3", "socio4")
> colnames(mimatriz)<-variables
> row.names(mimatriz)<-socios
> mimatriz
       peso(Kg) altura(m) precioRaqueta(euros)
socio1     80.1      1.70                229.0
socio2     73.5      1.65                 48.5
socio3     92.0      1.78                110.0
socio4     65.4      1.60                 85.8
```

Los datos de un objeto de tipo «matrix» han de ser todos del mismo tipo, al igual que sucedía con el objeto de tipo «vector». Si probamos a crear un objeto «matriz» con datos de distinto tipo, R los convertirá al mismo tipo. Por ejemplo, si de tres socios conocemos su nombre (dato tipo «character»), su peso en kilo-

© Ediciones Pirámide

gramos (dato tipo «numeric») y si ha pagado la cuota de socio (dato tipo «logical»), y creamos un objeto «matrix» con dichos datos, R convertirá todos los datos a tipo «character»:

```
> socio1<-c("Luis",80.1,FALSE)
> socio2<-c("Isabel",65.4,TRUE)
> socio3<-c("Javier",77.7,TRUE)
> mimatriz2<-rbind(socio1,socio2,socio3)
> variables<-c("nombre","peso(Kg)","pagadaCuota")
> colnames(mimatriz2)<-variables
> mimatriz2
       nombre   peso(Kg) pagadaCuota
socio1 "Luis"     "80.1"     "FALSE"
socio2 "Isabel"   "65.4"     "TRUE"
socio3 "92"       "1.78"     "110"
```

Para conocer cómo extraer datos de una base de datos con estructura en R de «matrix» utilizaremos la matriz `mimatriz` creada en los ejemplos anteriores. Realizaremos las siguientes operaciones de tipo extracción. Se trata de extraer:

- El dato situado en la primera fila y tercera columna.
- Toda la primera fila.
- Toda la segunda columna.
- La submatriz formada por las tres primeras filas y dos últimas columnas de `mimatriz`.
- Una submatriz formada con la primera y tercera filas y la primera columna de `mimatriz`.

Las operaciones anteriores se presentan en el siguiente código:

```
> a<-mimatriz[1,3]
> b<-mimatriz[1,]
> c<-mimatriz[,2]
> d<-mimatriz[1:3,1:2]
> e<-mimatriz[c(1,3),1]
> a
[1] 229
> b
   peso(Kg) altura(m) precioRaqueta(euros)
       80.1       1.7                229.0
```

© Ediciones Pirámide

```
> c
socio1 socio2 socio3 socio4
  1.70   1.65   1.78   1.60
> d
       peso(Kg) altura(m)
socio1     80.1      1.70
socio2     73.5      1.65
socio3     92.0      1.78
> e
socio1 socio3
  80.1   92.0
```

En aquellas matrices en las que hayamos incluido un nombre a las filas y a las columnas, como es el caso de `mimatriz`, la extracción de datos a partir de su posición también se puede realizar utilizando dichas denominaciones, en vez del número de fila y columna que corresponda. Con esta información, el ejercicio anterior se puede realizar de la siguiente forma:

```
> a<-mimatriz["socio1","precioRaqueta(euros)"]
> b<-mimatriz["socio1",]
> c<-mimatriz[,"altura(m)"]
> d<-mimatriz[c("socio1","socio2","socio3"),c("peso(Kg)","altu
ra(m)")]
> e<-mimatriz[c("socio1","socio3"),"peso(Kg)"]
> a
[1] 229
> b
   peso(Kg) altura(m) precioRaqueta(euros)
       80.1       1.7                229.0
> c
socio1 socio2 socio3 socio4
  1.70   1.65   1.78   1.60
> d
       peso(Kg) altura(m)
socio1     80.1      1.70
socio2     73.5      1.65
socio3     92.0      1.78
> e
socio1 socio3
  80.1   92.0
```

En una matriz, también es posible extraer aquellos elementos que cumplan una determinada condición. Por ejemplo:

© Ediciones Pirámide

— El peso (en kilogramos) de los socios que compraron una raqueta con un precio superior a los 100 euros.
— Lo que pagaron por su raqueta los socios que miden menos de 1,75 m:

```
> a1<-mimatriz[,1][mimatriz[,3]>100]
> a2<-mimatriz[,"peso(Kg)"][mimatriz[,"precioRaqueta(euros)"]>
100]
> b1<-mimatriz[,3][mimatriz[,2]<1.75]
> b2<-mimatriz[,"precioRaqueta(euros)"]
[mimatriz[,"altura(m)"]<1.75]
> a1
socio1 socio3
  80.1   92.0
> a2
socio1 socio3
  80.1   92.0
> b1
socio1 socio2 socio4
 229.0   48.5   85.8
> b2
socio1 socio2 socio4
 229.0   48.5   85.8
```

En el código anterior, las matrices `a1` y `a2`, `b1` y `b2` proporcionan un mismo resultado, con la diferencia de cómo se identifica la posición de los datos en la matriz (mediante su numeración o su denominación).

Al igual que vimos con los vectores, es posible manejar todos los datos de un objeto «matrix» de forma simultánea. Como ejemplo, creamos una matriz con 2 filas y 5 columnas cuyos elementos sean iguales a 5, para, a continuación, sumar y multiplicar dichos elementos por un mismo valor, 7, y finalmente sumar y multiplicar (elemento a elemento) las dos matrices resultantes:

```
> matriz3<-matrix(5,2,5)
> matriz3
     [,1] [,2] [,3] [,4] [,5]
[1,]    5    5    5    5    5
[2,]    5    5    5    5    5
> matriz4<-matriz3+7
> matriz4
     [,1] [,2] [,3] [,4] [,5]
[1,]   12   12   12   12   12
[2,]   12   12   12   12   12
```

© Ediciones Pirámide

```
> matriz5<-matriz3*7
> matriz5
     [,1] [,2] [,3] [,4] [,5]
[1,]   35   35   35   35   35
[2,]   35   35   35   35   35
> matriz6<-matriz4+matriz5
> matriz6
     [,1] [,2] [,3] [,4] [,5]
[1,]   47   47   47   47   47
[2,]   47   47   47   47   47
> matriz7<-matriz4*matriz5
> matriz7
     [,1] [,2] [,3] [,4] [,5]
[1,]  420  420  420  420  420
[2,]  420  420  420  420  420
```

En caso de que necesitemos calcular el determinante de una matriz, obtener la matriz inversa, o transpuesta, o el producto de dos matrices (no confundir con el producto elemento a elemento visto en la práctica anterior), los comandos necesarios serán, respectivamente: `det(A)`, `solve(A)`, `t(A)` y `A%*%B`. La tabla 1.2 muestra las principales operaciones con matrices.

A continuación, crearemos dos matrices cuadradas de orden 3 (`matriz8` y `matriz9`) que multiplicaremos (`matriz10`) y compararemos el resultado obtenido con la multiplicación elemento a elemento (`matriz11`). Asimismo, obtendremos el determinante, la inversa y la transpuesta de la matriz resultante de restar las matrices: `matriz10` y `matriz11`:

```
> matriz8<-matrix(1:9,3,3)
> matriz8
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> matriz9<-matrix(seq(2,18,2),3,3)
> matriz9
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> matriz10<-matriz8%*%matriz9
> matriz10
     [,1] [,2] [,3]
[1,]   60  132  204
```


© Ediciones Pirámide