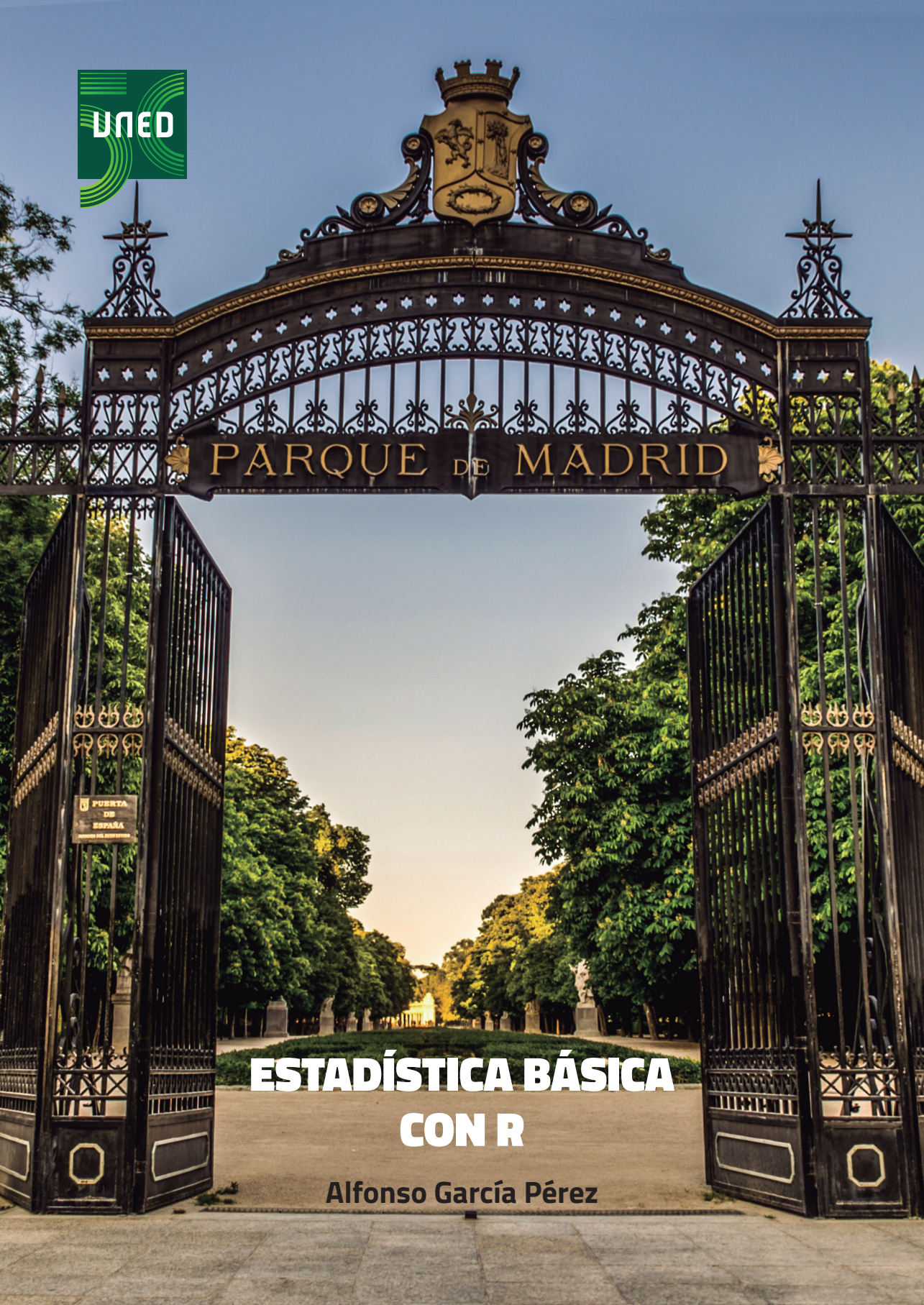

ESTADÍSTICA BÁSICA CON R

Alfonso García Pérez

ESTADÍSTICA BÁSICA CON R

Alfonso García Pérez

ESTADÍSTICA BÁSICA CON R

www.uned.es/publicaciones

ISBN electrónico: 978-84-362-7992-4

Edición digital: marzo de 2023

Aquí podrá encontrar información
adicional y actualizada de esta publicación

Dedicado a *Claudia*,
la alegría y felicidad de una
familia.

Índice

Prólogo

Este texto es una introducción a los principales conceptos de la Estadística, término genérico que habitualmente se utiliza para englobar tanto a la *Estadística Descriptiva*, como al *Cálculo de Probabilidades*, como a la *Inferencia Estadística*.

En la *Estadística Descriptiva* se deja que los *datos hablen por sí mismos*, es decir se ordenan, representan, etc., de manera que puedan sugerirnos estructuras o modelos que los expliquen. Sirve además, como introducción de algunos conceptos de Cálculo de Probabilidades y de Inferencia Estadística.

En el *Cálculo de Probabilidades* se define y maneja la Probabilidad como medida de la incertidumbre que presentan los fenómenos aleatorios, fenómenos que son el objeto de estudio de la Estadística; se proponen, además, modelos que pueden regir estas experiencias aleatorias.

La *Inferencia Estadística* es, sin duda, la parte más interesante puesto que con ella se pueden obtener conclusiones de donde se extrajeron los datos, midiendo los posibles errores en términos de probabilidades.

Al estudio de estas tres partes es a lo que dedicaremos el texto pero, hoy en día, el uso del ordenador se convierte en indispensable, por lo que teníamos que incluir algún paquete estadístico que sirviera de ayuda a nuestro estudio. Elegimos el *Paquete Estadístico R* porque pensamos que es el idóneo en la ejecución de los Métodos Estadísticos. Además es gratuito. Al final del libro indicamos la dirección de Internet de dónde obtener R y damos indicaciones de cómo instalarlo en su ordenador.

Para estudiar el texto sólo se requiere una formación elemental de Matemáticas como la impartida en la Enseñanza Media.

El texto está pensado para un cuatrimestre y como introducción a las partes antes mencionadas con objeto de que, al final de su estudio, el lector haya entendido los fundamentos y sepa utilizar los Métodos Estadísticos analizados.

Para que el texto sea lo más conciso posible, hemos publicado al margen del libro la adenda *Fórmulas y Tablas Estadísticas* (citada como ADD) que sirve de resumen de algunas fórmulas utilizadas, incluyendo también tablas de

distribuciones, aunque las probabilidades asociadas a éstas se podrán calcular con R y evitar utilizar la adenda ADD.

Aparecen al final de los capítulos algunos Ejercicios de Autoevaluación, cuya solución está al final de este libro, con objeto de que el lector valore si ha adquirido los contenidos del capítulo. En los textos *Problemas Resueltos de Estadística Básica* PREB, y *Ejercicios de Estadística Aplicada* EEA, aparecen numerosos Ejercicios de Autoevaluación resueltos con R, casi todos con datos reales o siendo *nonfiction problems*, es decir, ejercicios que podrían llegar a ser reales. Es muy recomendable ejercitarse en el uso de R con estas dos colecciones de problemas resueltos.

Comenzamos el libro con un capítulo introductorio a R ya que en el resto del libro se irán resolviendo ejemplos con la ayuda de este software. De hecho, entendemos que una buena manera de aprender R es utilizarlo en situaciones concretas. Advertimos, no obstante, que no pretendemos enseñar *programación R* sino utilizar este paquete en la ejecución de Métodos Estadísticos. Si el lector está interesado en profundizar en este software, le recomendamos algunos textos al final del primer capítulo.

De igual manera, si el lector desea estudiar más Métodos Estadísticos de los que hay en el libro, le recomendamos los textos que se citan por TA, *Estadística Aplicada Avanzada con R*, y MR, *Estadística Aplicada Robusta con R*. Además, al final de cada capítulo aparecerán Lecturas Recomendadas por si quiere profundizar en los temas tratados en el capítulo, pero su lectura no es necesaria para seguir el desarrollo de este texto. Al final del libro tiene una bibliografía general.

Comentando el contenido de este texto digamos que, a continuación del capítulo inicial sobre R se estudian tres capítulos sobre temas básicos: uno de Estadística Descriptiva y dos de Cálculo de Probabilidades.

En el Capítulo 5 se inicia verdaderamente la Inferencia Estadística con el estudio de los principales estadísticos a utilizar en las diversas situaciones planteadas. Los Intervalos de Confianza se estudian en el Capítulo 6 y los conceptos elementales de Tests de Hipótesis (la herramienta estadística más empleada, sin ninguna duda), se estudian en el Capítulo 7.

Los tests de hipótesis estudiados en el Capítulo 7 requieren habitualmente de la normalidad de los datos para poder ser utilizados. En el Capítulo 8 estudiamos tests, denominados no paramétricos, que no requieren de esta suposición.

Los dos últimos capítulos abordan las aplicaciones más comunes de la Inferencia Estadística; se trata del Análisis de la Varianza, estudiado en el Capítulo 9, y del Análisis de la Regresión, analizado en el Capítulo 10.

Existen vídeos de introducción de los capítulos del libro en YouTube aunque su visionado no exime de su lectura detallada a la hora de aprender a manejar los métodos que en este texto se enseñan. Si quiere recibir los ficheros de datos

y los comandos de R que utilizamos en el libro, así como su Suplemento, las direcciones de los vídeos de YouTube, e incluso comentarios o aclaraciones del autor, puede ponerse en contacto con él, en el correo electrónico

estadistica.aplicada.robusta@gmail.com

Alfonso García Pérez
Catedrático de Estadística e I.O.
https://www2.uned.es/experto-metodos-avanzados/Garcia-Perez.htm

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

que aquí se indican.

Como ocurre con todos los paquetes estadísticos, R también utiliza un lenguaje propio. Toda instrucción o comando que pueda ser ejecutada desde la línea de comandos se denomina *expresión*. Las expresiones se ejecutan con Enter.

Éstas pueden tener una longitud de más de una línea. Cuando se presiona Enter después de una expresión sintácticamente incompleta, ésta no se ejecuta ni se producen mensajes de error; aparece el *prompt* `+` al comienzo de una nueva línea de comandos invitándonos así a completar la expresión no concluida.

Los elementos básicos de R son los *objetos* y, por tanto, a ellos se referirán las expresiones de R. De hecho, los objetos son ficheros capaces de ser editados y, en su caso, ejecutados.

Un *objeto-dato*, o simplemente un *objeto*, es el resultado de ejecutar una expresión en la que aparece el operador `<-` . Dicho de otra forma, una expresión que nos interese guardar (por ejemplo para volver a utilizarla en otra ocasión o porque va a ser parte de otra expresión) puede ser *salvada* con el operador `<-` , también denominado operador *asignación*.

Por ejemplo, si queremos denominar `a` al número `1`, ejecutaríamos la expresión

```
> a<-1
```

pudiendo hacer ahora, por ejemplo,

```
> a+a
[1] 2
```

Apuntemos que el número entre corchetes que aparece, `[1]`, indica solamente el lugar que ocupa el primer valor del resultado de ejecutar la expresión que le precede. Esto es especialmente útil cuando manejemos un número elevado de datos.

El nombre asignado a un objeto debe empezar por una letra y puede incluir cualquier combinación de letras mayúsculas o minúsculas, números y puntos. Por ejemplo, dos nombres de objetos podrían ser `X` y `datos.nuevos`

Como dijimos, los objetos obtenidos como resultado de ejecutar una expresión pueden ser utilizados como parte de una nueva expresión.

Los dos tipos de objetos más utilizados son el *dato*, sobre el que volveremos más adelante, y la *función*. Las funciones constan de un nombre seguido de dos paréntesis, `nombre()`; entre los paréntesis se incluyen sus *argumentos*. Si

sólo ejecutamos su nombre obtendremos su definición y si ejecuta `?nombre` obtendrá ayuda sobre su utilización.

Como el papel que juegan las funciones es ciertamente diferente del de los otros objetos R, no volveremos a referirnos a ellas como objetos sino como funciones.

Su importancia es tal que, de hecho, puede decirse que R es un *lenguaje funcional*, en el sentido de que sus *programas* se presentan como funciones escritas en su lenguaje. Pero lo más interesante es que, dada la flexibilidad de R, podemos crear nuestras propias funciones, añadiéndolas a las ya existentes, respondiendo así a nuestras necesidades particulares; éstas podrán combinar distintas expresiones R que deseemos sean ejecutadas de forma conjunta.

Una de las funciones más sencillas y por medio de la cual *salimos* del programa es

```
> q()
```

Al ejecutarla, el ordenador nos preguntará si queremos conservar los cálculos que hayamos realizado en la sesión, mediante la pregunta *Guardar imagen de área de trabajo?* (*Save workspace image?* si eligió el idioma inglés en la instalación). Si respondemos *Sí*, al comenzar la sesión siguiente podremos volver a utilizar los resultados de la sesión recién finalizada. En otro caso, todo lo ejecutado en la sesión se perderá. En ocasiones no es mala idea utilizar esta opción de contestar *No* si no queremos modificar nada de lo realizado en una sesión en la que nos hayamos equivocado en cosas de mucha importancia, o no queremos conservar.

Una observación que merece destacarse es que, desde esa línea de comandos podemos utilizar R como una potente calculadora matemática. Así, desde allí se pueden realizar las habituales operaciones matemáticas:

```
> 9*8
[1] 72
```

u obtener el valor de las funciones más conocidas como la potencial, la exponencial, la raíz cuadrada,

```
> 3^2
[1] 9
> exp(2)
[1] 7.389056
> sqrt(16)
[1] 4
```

También se pueden resolver sistemas de ecuaciones o hacer integración numérica y otras muchas aplicaciones matemáticas (y, como no, estadísticas) que el lector irá aprendiendo al manejar R mientras lee este libro o los otros textos del autor que aparecen en la Bibliografía, ya que son miles las funciones que contiene R. Algunas de ellas no están en la versión que ha instalado sino en librerías adicionales que se pueden instalar y sobre las que hablaremos más adelante.

Dos funciones que queremos ya destacar son, la función `objects`, mediante la cual podemos listar los objetos R existentes

```
> objects()
```

Si queremos que liste solamente los objetos que, por ejemplo, contengan `x`, ejecutaríamos

```
> objects(pattern="x")
```

Y la función `rm`, utilizada para suprimir objetos; para ello debemos utilizar como argumentos suyos, los objetos a eliminar. Así, si queremos suprimir los objetos `dato1` y `función3`, ejecutaríamos la expresión

```
> rm(dato1,función3)
```

Finalizamos esta primera toma de contacto con R señalando que, para recuperar alguna instrucción ejecutada anteriormente, basta con pulsar la tecla ↑ tantas veces como sea necesario hasta que la instrucción aparezca. Luego deberemos ejecutarla pulsando Enter.

1.2. El editor de objetos R

Para crear o modificar objetos o funciones R, debemos utilizar las denominadas *funciones editoras* `edit` y, preferentemente, `fix` . Al ejecutarlas entraremos en el editor propio de R, o en el *Bloc de Notas* de Windows, o en el que hayamos establecido en las Preferencias del programa. Si los objetos a editar son datos, suele abrirse el Editor de Datos.

También pueden crearse funciones desde la línea de comandos. Si queremos crear una nueva función, por ejemplo $f(x) = 2x$, teclearemos desde la línea de comandos,

```
> f<-function(x){2*x}
```

habiendo tecleado entre las llaves la definición de la función f. Ahora, si queremos saber cuál es su valor en, por ejemplo, 12, ejecutaremos `f(12)`

```
> f(12)
[1] 24
```

Cualquier asignación que hagamos eliminará asignaciones previas con el mismo nombre. Es decir, si ahora denominamos `f` a otro objeto de R, la anterior asignación habrá sido eliminada. Por esta razón, si queremos modificar una función ya existente conservando a la vez ésta, deberemos crear primero la nueva con la expresión ya conocida

```
> nueva<-antigua
```

luego ejecutar

```
> fix(nueva)
```

la cual, una vez modificada, será salvada al salir del editor.

Un conjunto de instrucciones de R se denomina *script*, el cual puede ser guardado en el propio programa. Aquella parte que marquemos puede ser ejecutada con las teclas Ctrl-R.

Es habitual en R ir creando poco a poco un script para obtener un gráfico o ejecutar un conjunto de instrucciones, guardarlas estas en un fichero txt para copiarlo entero más tarde y pegarlo en la línea de comandos, ejecutando de esta forma y de una sola vez, todas las instrucciones así guardadas.

1.3. Datos en R

El concepto de *dato* en R es más amplio que el utilizado en Estadística. De hecho, lo que en R se denomina dato es el resultado de ejecutar una expresión R, es decir, un tipo de objeto.

Así por ejemplo, un dato podrá ser una matriz en donde como primera columna aparecen los nombres de los individuos en los que se han observado las variables cuyos valores aparecen en las restantes columnas.

Cada tipo de dato tiene asociados determinados *atributos*; el más importante es su *modo*. Consideraremos cuatro clases de *modos*:

- *logical* (lógico): Modo binario en donde los valores posibles son `T` ó `F` (Verdadero o Falso).

- *numeric* (numérico): Modo en donde los valores posibles son números reales.

- *complex* (complejo): Modo en donde los valores posibles son números complejos.

- *character* (carácter): Modo en donde los valores posibles son caracteres separados por comillas.

Dado que en este curso no veremos números complejos y que los modos lógicos serán utilizados básicamente sólo como valores de argumentos de funciones de R, únicamente tendremos los habituales datos de tipo cuantitativo y de tipo cualitativo o caracteres. Estos podrían aparecer, en principio, de cinco formas diferentes:

- *vector* (vector): Conjunto de elementos en un orden específico. Todos los elementos de un vector deben ser del mismo modo. Los más utilizados son los *vectores numéricos*, es decir, vectores cuyos elementos son números.

- *matrix* (matriz): Disposición bidimensional de elementos de un mismo *modo*.

- *factor* (factor): Vector cuyos elementos son valores procedentes de un número finito de *categorías*.

- *data frame* (estructura de datos): Disposición bidimensional de elementos cuyas columnas pueden estar formadas por elementos de distinto *modo*.

- *list* (lista): Expresión más general de dato, la cual puede contener colecciones arbitrarias de datos.

R tiene incorporado un Editor de Datos que permite visualizarlos y, en cierta medida, manejarlos de una forma algo más cómoda, pero siempre debemos tener presente que R considera diferentes tipos de datos que iremos describiendo en los siguientes apartados. Los más utilizados son el *vector*, la *matriz* y el *data frame*.

1.3.1. Vectores

El *vector* es el tipo de dato más utilizado en R, junto con el *data frame*, especialmente como argumento de funciones.

Como antes dijimos, todos los elementos de un vector deben ser del mismo modo. El otro atributo considerado en un vector es su *longitud*.

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
```

```
2 18  rojo
3 25 verde
4 27  rojo
5 33  azul
```

sabemos que si quisiéramos extraer los individuos (filas) correspondientes al "valor" de B = rojo, hay que ejecutar

```
> datos[datos$B=="rojo",]
   A    B
1 12 rojo
2 18 rojo
4 27 rojo
```

pero, si quisiéramos extraer los de valores rojo y verde, habría que ejecutar

```
> datos[is.element(B, c("rojo","verde")),]
   A     B
1 12  rojo
2 18  rojo
3 25 verde
4 27  rojo
```

aunque es más sencillo acudir al suceso complementario ejecutando

```
> datos[datos$B != "azul",]
   A     B
1 12  rojo
2 18  rojo
3 25 verde
4 27  rojo
```

Transformar datos en R es muy simple. Basta con aplicar una función a los datos de la variable original, pero es interesante saber que se puede añadir una nueva variable (una nueva columna) a un *data frame* ya existente. Por ejemplo,

```
> oviedo$nueva<-log(oviedo$Peso)
> oviedo
  Peso Talla Sexo Edad EstaCivil    nueva
1   65  1.65    F   45    Casado 4.174387
2   75  1.80    M   55    Casado 4.317488
3   80  1.95    M   47    Casado 4.382027
4   67  1.75    F   34   Soltero 4.204693
```

Recuerde que si asignamos el mismo nombre, el nuevo objeto R reemplaza al anterior.

1.3.5. Listas

El objeto-dato de R más flexible es la *lista*, el cual puede admitir datos de diferentes *modos* y de diferentes longitudes, e inclusive otras listas. Por esta razón, las listas son, habitualmente, el tipo de objeto-dato final en donde se van incorporando los otros tipos de datos.

La mayoría de las funciones R que realizan un análisis estadístico, presentan sus resultados en una lista. Así, una lista podría estar constituida por los datos originales, los valores ajustados, los residuos, los estadísticos de contraste, el p-valor e inclusive hasta por el método empleado. Todo esto se combina en un sólo elemento: una lista.

Para crear una lista se utiliza la función `list` en donde cada uno de sus argumentos se convierte en una componente de la lista.

Por ejemplo, si queremos crear una lista formada por la estructura de datos `oviedo`, más arriba creada, y un vector formado por dos números, el 4 y el 5, ejecutaríamos `list(oviedo, c(4,5))` y R respondería

```
[[1]]:
     Peso  Talla Sexo Edad EstaCivil
Ind1 "65" "1.65"  "F" "45"  "Casado"
Ind2 "75" "1.80"  "M" "55"  "Casado"
Ind3 "80" "1.95"  "M" "47"  "Casado"
Ind4 "67" "1.75"  "F" "34" "Soltero"

[[2]]:
[1] 4 5
```

1.4. Gráficos

La ventana de gráficos se abre de forma automática al ejecutar alguna función que los realice. Estas funciones se dividen en funciones gráficas de *alto nivel* y de *bajo nivel*. Con las primeras se obtiene un gráfico nuevo, mientras que con las segundas podemos hacer modificaciones de un gráfico ya existente. Además podemos controlar aspectos específicos de nuestros gráficos usando parámetros gráficos adicionales.

1.4.1. Funciones gráficas de alto nivel

Como antes dijimos, las funciones gráficas de alto nivel producen un gráfico totalmente nuevo, incluidos los ejes y sus etiquetas, borrando previamente el gráfico que pudiera existir en la ventana de gráficos.

Las funciones gráficas de alto nivel se dividen en varios grupos dependiendo, fundamentalmente, de lo que queramos representar. Si queremos representar