Tema 2:

Algunas clases de objetos comunes

2.1. Vector

El primer tipo de objeto que puede manejar es el vector. Por ejemplo, el símbolo de dos puntos, situado entre dos números, construye un vector de modo sencillo, tanto en orden ascendente como descendente:

```
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10
> 10:1
[1] 10 9 8 7 6 5 4 3 2 1
> 1:3.5
[1] 1 2 3
> 1.9:3.5
[1] 1.9 2.9
```

El vector comienza en el primer número suministrado y finaliza en el segundo o en un número anterior sin sobrepasarlo, tanto en orden ascendente como descendente.

Es ilustrativo observar que el vector **1:1** es precisamente el número 1, por lo que se puede afirmar que, en realidad, todos los números son considerados como vectores, lo que ayuda a comprender las operaciones relacionadas con vectores.

Pero los vectores no están limitados a ser sucesiones equiespaciadas de números, ni siquiera a serlo de números.

Podrá construir vectores fácilmente con la función de concatenación, **c**. Verá sus propiedades con más detalle posteriormente, pero ahora le basta con saber que admite como argumento varios vectores (recuerde que un sólo número es equivalente a un vector) y los concatena en uno solo. Así

```
> c(2,7,1,6)
[1] 2 7 1 6
> c(1:10)
[1] 1 2 3 4 5 6 7 8 9 10
> c(1:10,2,7,1,6,2:-5)
[1] 1 2 3 4 5 6 7 8 9 10 2
[12] 7 1 6 2 1 0 -1 -2 -3 -4 -5
> c()
NULL
```

En el último caso, el resultado es el objeto NULL. Si los elementos van entrecomillados, serán cadenas de
caracteres y obtendrá un vector de este tipo. Estos vectores son útiles, por ejemplo, para dar nombres a las
variables utilizadas en un análisis o dar nombre a los meses del año\footnote{La función \code{month.name} contiene estos nombres en inglés.}, como hacemos a continuación.

```
> nombre.mes = c("Enero","Febrero","Marzo","Abril","Mayo"
```

```
,"Junio","Julio","Agosto","Septiembre","Octubre"
,"Noviembre","Diciembre")
> c("Hola", "Uno", "Dos")
[1] "Hola" "Uno" "Dos"
```

Un vector está formado por elementos del mismo tipo, así pues, no pueden mezclarse números y cadenas de
caracteres, ya que se transformarán en cadenas de caracteres, como ocurre en el siguiente ejemplo:

```
> c("Hola", "Uno", "Dos",3)
[1] "Hola" "Uno" "Dos" "3"
```

Del mismo modo, si mezcla números reales y complejos, se convertirán todos a complejos:

```
> c(1,1+2i)
[1] 1+0i 1+2i
```

Puede asignar nombre a los elementos de un vector mediante la función **names**. De hecho, esta función permite dar nombre a los elementos de cualquier objeto. Aplicada a un vector sería así:

```
> x<-1:5
> x
[1] 1 2 3 4 5
> names(x)<-c("I","II","III","IV","V")
> x
I II III IV V
1 2 3 4 5
```

La operación **:** es un caso particular de la función **seq** que permite construir vectores que son sucesiones equiespaciadas. Consultando la ayuda, **help(seq)**, verá que posee los siguientes argumentos:

- **from** Valor inicial de la sucesión.
- **to** Valor final de la sucesión.
- **by** Espaciado entre los valores.
- **length** Longitud del valor resultante.
- **along** Un objeto cuya longitud se usará para el objeto a construir.

Es un error utilizar todos los argumentos simultáneamente. Sólo debe dar los necesarios, el resto se calculan a partir de los dados. Si no indica ninguno, todos valen 1.

Puede crear un vector con la función **vector**. Su forma es

```
vector(mode="logical", length=0)
```

donde **mode** indica el modo o tipo del vector y **length** es su longitud.

mode devuelve o modifica el tipo de un objeto. Los modos posibles incluyen los siguientes:

logical, **numeric**, **complex**, **character**, **null**, **list**, **function**, **graphics**, **expression**, **name**, **frame**, **raw** y **unknown**. El tipo **numeric** engloba los diferentes tipos numéricos correspondientes a números reales.

```
> x<-1:5
> mode(x)
[1] "numeric"
> mode(x)<-"complex"
> x
[1] 1+0i 2+0i 3+0i 4+0i 5+0i
> x<-vector("numeric",5)
> x
[1] 0 0 0 0 0
```

2.1.1. Operaciones con vectores

Puede realizar operaciones con vectores directamente. Así,

```
> (1:10)*2
[1] 2 4 6 8 10 12 14 16 18 20
> (1:10)*(1:10)
[1] 1 4 9 16 25 36 49 64 81 100
```

donde observará, en primer lugar, el uso de paréntesis, que garantiza que las operaciones se realizan en el orden deseado (Si no los utiliza, \R\ sigue un orden de precedencias en las operaciones. Compruebe \verb|(1:10)^2| y \verb|1:10^2|.) y, en segundo lugar, que la multiplicación se realiza componente a componente.
Del mismo modo actuaría cualquier otra operación. En realidad, la primera de las dos operaciones anteriores, que parece más intuitiva a primera vista, lo es menos. Lo que \R\ ha hecho es replicar el vector 2 (2:2) tantas veces como ha hecho falta (10) para que la operación coincidiese con una como la segunda. Si en esa replicación hubiese llegado a sobrar algún elemento se hubiese indicado mediante un mensaje. Los dos ejemplos siguientes ilustran este concepto, que será muy utilizado, por lo que debe asimilarlo correctamente.

```
> (1:10)+(1:5)
[1] 2 4 6 8 10 7 9 11 13 15
> (1:10)+(2:5)
[1] 3 5 7 9 7 9 11 13 11 13
Warning message:
longer object length is not a multiple
of shorter object length in: (1:10) + (2:5)
```

Esto es, **(1:10)^2** equivale a **(1:10)^(2:2)**

```
> (1:10)^(2:2)
[1] 1 4 9 16 25 36 49 64 81 100
```

Compare con **(1:10)^(1:3)**

```
> (1:10)^(1:3)
```

```
[1] 1 4 27 4 25 216 7 64 729 10
Warning message:
longer object length is not a multiple of
shorter object length in: (1:10)^(1:3)
```

La longitud de un vector puede obtenerse con la función **length**, como puede ser **length (1:10)**. Puede consultar la ayuda sobre esta función con la función **help(length)** donde encontrará que sirve también para *definir* la longitud del vector, haciéndolo más largo o más corto en su caso. En caso de hacerlo más largo, se completará con **NA** (Iniciales de Not Available). También encontrará en otros casos el valor **NaN**, Not a Number, en casos en que el resultado no puede ser calculado, esto es con valores faltantes.

```
> kk<-1
> kk
[1] 1
> length(kk)<-10
> kk
[1] 1 NA NA NA NA NA NA NA NA NA
> kk<-"Pepe"
> kk
[1] "Pepe"
> length(kk)<-10
> kk
[1] "Pepe" NA NA NA NA NA NA NA NA NA
```

Puede referirse al elemento que ocupa la posición **i** de un vector, **x**, mediante **x[i]**. También puede referirse a un subconjunto mediante un subconjunto de subíndices. Si los subiíndices son números naturales, hacen referencia a las posiciones dentro del vector. Si los subíndices son valores lógicos, se corresponden con los mismos elementos del vector, pero sólo hacen referencia a los que tienen un subíndice con valor lógico **TRUE**.

```
> x<-(1:10)^2
> x
[1] 1 4 9 16 25 36 49 64 81 100
> x[3]
[1] 9
> x[c(3,5,9)]
[1] 9 25 81
> x[seq(1,9,by=2)]
[1] 1 9 25 49 81
```

El ejemplo anterior crea un objeto, **x**, que contiene los cuadrados de los números naturales de 1 a 10. Luego se ha referido, en primer lugar, al tercer elemento; en segundo lugar, a los elementos tercero, quinto y noveno; y, por último, a los elementos impares. En el siguiente caso se hace referencia a los elementos que cumplen cierta condición.

```
> sexo
Error: Object "sexo" not found
> sexo<-c("H","H","M","M","H")
> sexo
[1] "H" "H" "M" "M" "H"
```

```
> nombre
Error: Object "nombre" not found
> nombre<-c("Pedro","Juan","Ana","Lola","Rafael")
> nombre
[1] "Pedro" "Juan" "Ana" "Lola" "Rafael"
> sexo=="H"
[1] T T F F T
> nombre[sexo=="H"]
[1] "Pedro" "Juan" "Rafael"
```

La referencia a los elementos impares de \textbf{x} puede hacerla también así:

```
> x[1:10%%2==1]
[1] 1 9 25 49 81
```

2.2. Factor

Un **factor** es un vector de cadenas de caracteres que se interpreta de modo especial mediante el atributo **levels** que indica los valores numéricos posibles, cada uno de los cuales se asocia a cada cadena diferente.

La forma de creación es mediante la función factor, cuya utilización es

```
factor(x, levels=, labels=, exclude= , ordered=)
```

siendo los argumentos los siguientes:

- **x** son los datos a partir de los cuales se genera el factor
- **levels** es un vector de niveles opcional. Los valores de **x** que no correspondan a uno de los indicados se sustituyen por **NA**. El valor predeterminado de este parámetro es la lista ordenada de valores distintos de **x**.
- **labels** es un vector de valores que se usará como etiqueta de los niveles. El valor predeterminado es **as.character(levels)**.
- **exclude** es un vector de valores que deben excluirse de la formación de niveles y sustituirse por **NA**.
- **ordered** es un valor lógico que indica si lo es el factor.

Un factor corresponde a una variable nominal u ordinal, dividida en categorías, y que se utiliza, por ejemplo, para dividir una población en grupos. Es un concepto muy importante que debe asimilar a través de la práctica.

Ejemplo:

```
> factor(c("A","B","A"))
[1] A B A
Levels: A B
> levels(factor(c("A","B","A")))
[1] "A" "B"
```

2.3. Matriz

El siguiente tipo de objeto que puede utilizar es la matriz, considerada como una variable indexada mediante dos índices. Para crearlas puede utilizar la función \code{matrix}. Los parámetros de esta función son:

- **data** Vector que contiene los valores que formarán la matriz. Debe tener en cuenta que si no es suficientemente grande, se repetirá las veces que sea necesario.
- **nrow** Número de filas. Si especifica el número de columnas, y no da el de filas, se calcula este mediante **ceiling(length(data)/ncol)**. Si no especifica ninguno, se toma **ncol=1**.
- **ncol** Número de columnas. Si especifica el número de filas, y no da el de columnas, se calcula este mediante **ceiling(length(data)/nrow)**.
- **byrow** Variable lógica que indica si la matriz debe construirse por filas o por columnas. El valor predeterminado es **F**.
- **dimnames** Lista (concepto que se verá a continuación) de longitud 2 con los nombres de las filas y las columnas. Cada componente de la lista debe tener longitud 0 o ser un vector de cadenas de caracteres con la misma longitud que la dimensión correspondiente de la matriz.

Ejemplos:

```
matriz <- matrix(0, 4, 5)
```

Crea una matriz de 4 por 5 cuyos elementos son ceros (ya que el cero se repite tantas veces como haga falta para crearla).

Si ahora escribe:

```
> matrix(1:10,5)
[,1] [,2]
[1,] 1 6
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10
```

R crea la matriz que aparece, que como ha ordenado tiene cinco filas y por tanto (10/5) dos columnas, y se rellena por columnas. El mismo resultado obtendría con

```
matrix(data=1:10, ncol=2, nrow=5, byrow=F)
```

Si quiere dar nombres a las columnas (o a las filas) puede hacerlo asignando valores al parámetro **dimnames**

```
> matrix(1:10,5,dim=list(c(),c("Primero","Segundo")))
Primero Segundo
[1,] 1 6
```

```
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10
```

Cuando las longitudes no son las requeridas exactamente, se repetirá el vector si es necesario, pudiendo recibir un mensaje de advertencia si alguna de las veces no se utiliza completo.

```
> matrix(1:10,3)
[,1] [,2] [,3] [,4]
[1,] 1 4 7 10
[2,] 2 5 8 1
[3,] 3 6 9 2
Warning message:
data length [10] is not a sub-multiple or multiple
of the number of rows [3] in matrix
```

Puede hacer referencia a una submatriz, en particular a un elemento de la matriz, sin más que indicar los índices a que hace referencia. Si dispone de los datos de peso, altura y edad de un grupo de personas puede escribir:

```
> datos<-c(
77 , 1.63 , 23,
58 , 1.63 , 23,
89 , 1.85 , 26,
55 , 1.62 , 23,
47 , 1.60 , 26,
60 , 1.63 , 26,
54 , 1.70 , 22,
58 , 1.65 , 23,
75 , 1.78 , 26,
65 , 1.70 , 24,
82 , 1.77 , 28,
85 , 1.83 , 42,
75 , 1.74 , 25,
65 , 1.65 , 26)
> matriz<-matrix(datos,ncol=3,
+ dimnames=list(c(),c("Peso","Altura","Edad")), byrow=T)
> matriz[,1]
[1] 77 58 89 55 47 60 54 58 75 65 82 85 75 65
> matriz[,"Peso"]
[1] 77 58 89 55 47 60 54 58 75 65 82 85 75 65
> matriz[4,3]
Edad
23
> matriz[1,]
Peso Altura Edad
77 1.63 23
```

Las operaciones realizadas, en primer lugar, guardan en el objeto **matriz** la matriz de datos

construida a partir del vector de datos. Puede hacer referencia a la columna (vector) de pesos mediante **matriz[,1]** o **matriz[,"Peso"]**, a un elemento concreto con **matriz[4,3]**, que da el valor de la tercera variable en el cuarto individuo, o a un individuo concreto, como **matriz[1,]** que da las observaciones del primer individuo.

Por otra parte, además de valores naturales, los subíndices pueden tomar también valores lógicos. En ese caso, se está haciendo referencia sólamente a las posiciones cuyo subíndice tome el valor **TRUE**.

```
> x=1:3
> x
[1] 1 2 3
> x[c(T,F,T)]
[1] 1 3
```

2.3.1. Operaciones con Matrices

Los operadores aritméticos pueden utilizarse sin problemas. Recuerde que su uso es análogo al caso de los vectores, esto es, componente a componente.

```
> A1<-matrix(1:10,2,5)
> A1
[,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
> A2<-matrix((1:10)^2,2,5)
> A2
[,1] [,2] [,3] [,4] [,5]
[1,] 1 9 25 49 81
[2,] 4 16 36 64 100
> A1*A2
[,1] [,2] [,3] [,4] [,5]
[1,] 1 27 125 343 729
[2,] 8 64 216 512 1000
> A2/A1
[,1] [,2] [,3] [,4] [,5]
[1,] 1 3 5 7 9
[2,] 2 4 6 8 10
> A1%%3
[,1] [,2] [,3] [,4] [,5]
[1,] 1 0 2 1 0
[2,] 2 1 0 2 1
> A1*(2:4)
[,1] [,2] [,3] [,4] [,5]
[1,] 2 12 15 14 36
[2,] 6 8 24 24 20
> A1*(2:6)
[,1] [,2] [,3] [,4] [,5]
[1,] 2 12 30 21 45
[2,] 6 20 12 32 60
```

También es posible realizar el producto matricial mediante el operador **%*%**.

```
> B<-matrix(1:10,ncol=2)
> B
[,1] [,2]
[1,] 1 6
[2,] 2 7
[3,] 3 8
[4,] 4 9
[5,] 5 10
> A1%*%B
[,1] [,2]
[1,] 95 220
[2,] 110 260
```

La función **crossprod** devuelve el producto matricial cruzado de dos matrices, esto es, la traspuesta de la primera matriz, multiplicada por la segunda. Si no especifica segunda matriz, **R** la toma igual a la primera, con lo que obtiene el conocido producto **X'X**.

```
> crossprod(A1)
[,1] [,2] [,3] [,4] [,5]
[1,] 5 11 17 23 29
[2,] 11 25 39 53 67
[3,] 17 39 61 83 105
[4,] 23 53 83 113 143
[5,] 29 67 105 143 181
```

La función **outer** realiza el producto exterior de dos matrices (o vectores) sobre una función dada. La forma de uso generalizada es

```
outer(X, Y, FUN="*", ...)
```

donde **X** e **Y** son los argumentos de la función **FUN**, que es cualquier función que acepte al menos dos vectores como argumentos y devuelva un valor único. Después de la función pueden pasarse otros argumentos adicionales que sean necesarios para la función, lo que se indica mediante los puntos suspensivos.

El resultado es una variable multiindexada (*array*), cuyo vector de dimensiones es la concatenación de los vectores de dimensión de **X** e **Y** y donde **FUN(X[i,j,k,...], Y[a,b,c,...])** es el valor del elemento **[i,j,k,...,a,b,c,...]**.

También puede usarse en forma de operador mediante **%o%**, en cuyo caso la función utilizada es el producto.

```
> x<-1:3
> y<-1:4
> outer(x ,y)
[,1] [,2] [,3] [,4]
[1,] 1 2 3 4
[2,] 2 4 6 8
[3,] 3 6 9 12
```

t calcula la matriz traspuesta.

```
> t(B)
[,1] [,2] [,3] [,4] [,5]
[1,] 1 2 3 4 5
[2,] 6 7 8 9 10
```

solve invierte una matriz, si sólo se le da un argumento, y también resuelve sistemas lineales de ecuaciones, cuando se le dan dos. El primer argumento es una matriz que debe ser cuadrada no singular, y el segundo argumento, opcional, es un vector o matriz de coeficientes.

Las siguientes órdenes crean una matriz cuadrada, **B'B**, la invierten, y comprueban que el producto de la matriz y su inversa es la matriz identidad (salvo errores de redondeo).

```
> crossprod(B)
[,1] [,2]
[1,] 55 130
[2,] 130 330
> solve(crossprod(B))
[,1] [,2]
[1,] 0.264 -0.104
[2,] -0.104 0.044
> solve(crossprod(B))%*%crossprod(B)
[,1] [,2]
[1,] 1.000000e+00 8.604228e-15
[2,] -3.538836e-15 1.000000e+00
```

Para resolver el sistema de ecuaciones:

3x+2y=5, x-y=0

que en forma matricial se escribe

$$\begin{pmatrix} 3 & 2 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ 0 \end{pmatrix}$$

escriba

```
> solve(matrix(c(3,2,1,-1),ncol=2,byrow=T),c(5,0))
[1] 1 1
```

siendo la solución, por tanto, **x=1, y=1**.

eigen calcula los autovalores y autovectores de una matriz cuadrada. El primer argumento de esta función, **x**, es la matriz. El segundo, optativo, es **symmetric**. Es de tipo lógico y se utiliza para indicar que la matriz es simétrica (o hermitiana en el caso complejo), en caso

contrario se inspecciona para comprobarlo. El tercer argumento, optativo, es **only.values=F**. También es de tipo lógico y en caso de ser **T** no se calculan los autovectores.

El resultado es una lista: El primer elemento es **values**, que es el vector de autovalores, y el segundo es **vectors**, que es la matriz de autovectores por columnas. A partir de los autovalores puede escribir una función para calcular fácilmente el determinante de la matriz:

determinante <- function(x) prod(eigen(x)$values)

No es necesario crear esta función, ya que **R** suministra las funciones **det** y **determinant** que lo realizan directamente.

2.4. Variable multiindexada

La generalización de los vectores y matrices son las variables multiindexadas, denominadas **arrays**, y de las cuales son casos particulares los vectores y matrices. Para crear una variable multiindexada se utiliza la función

array(data, dim, dimnames)

donde **dim** es un vector de dimensiones.

Así, por ejemplo, la siguiente orden crea una variable triindexada. Puede hacer referencia a cualquier subconjunto de la misma, de modo similar a las matrices.

```
> Tri <- array(1:(2*3*5),c(2,3,5))
> Tri

, , 1
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

, , 2
[,1] [,2] [,3]
[1,] 7 9 11
[2,] 8 10 12

, , 3
[,1] [,2] [,3]
[1,] 13 15 17
[2,] 14 16 18

, , 4
[,1] [,2] [,3]
[1,] 19 21 23
[2,] 20 22 24

, , 5
[,1] [,2] [,3]
```

```
[1,] 25 27 29
[2,] 26 28 30
> Tri[1,1,1]
[1] 1
> Tri[1,,]
[,1] [,2] [,3] [,4] [,5]
[1,] 1 7 13 19 25
[2,] 3 9 15 21 27
[3,] 5 11 17 23 29
```

aperm permite trasponer una variable multiindexada indicando la propia variable como primer argumento y el vector de permutaciones de índices como segundo.

```
> x <- array(1:6, 2:3)
> x
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
> xt <- aperm(x, c(2,1))
> xt
[,1] [,2]
[1,] 1 2 [2,] 3 4
[3,] 5 6
> xt <- aperm(x, c(2,1),F)
> xt
[,1] [,2] [,3]
[1,] 1 5 4
[2,] 3 2 6
```

2.5. Lista

Los tipos anteriores corresponden a variables de un solo tipo, aunque indexadas. Si necesita disponer de variables de varios tipos, puede recurrir a las listas. Una lista se construye con la función **list** que devuelve un objeto de tipo lista con tantos componentes como argumentos se le suministren y es el más utilizado para devolver el resultado de una función.

```
> list(A=1:10,B=nombre.mes)
$A:
[1] 1 2 3 4 5 6 7 8 9 10

$B:
[1] "Enero" "Febrero" "Marzo" "Abril"
[5] "Mayo" "Junio" "Julio" "Agosto"
[9] "Septiembre" "Octubre" "Noviembre" "Diciembre"
```

Puede referirse a cada uno de los elementos de la lista de dos formas distintas: Si tiene nombre, como en este caso, mediante el nombre de la lista, el símbolo **$**, y el nombre del elemento. En cualquier caso, siempre puede referirse a él mediante el índice de posición entre dobles corchetes.

```
> list(A=1:10,B=nombre.mes)$A
[1] 1 2 3 4 5 6 7 8 9 10
> list(A=1:10,B=nombre.mes)[[1]]
[1] 1 2 3 4 5 6 7 8 9 10
```

La segunda forma pone más de relieve la estructura de la lista. Además, si el elemento referido es un objeto indexado podrá de nuevo hacerse referencia a índices para designar sus componentes.

```
> lista1<-list(A=1:10,B=nombre.mes)
> lista2<-list(lista1,C="Otro nombre",Matriz=matriz)
> lista2
[[1]]:
[[1]]$A:
[1] 1 2 3 4 5 6 7 8 9 10

[[1]]$B:
[1] "Enero" "Febrero" "Marzo" "Abril"
[5] "Mayo" "Junio" "Julio" "Agosto"
[9] "Septiembre" "Octubre" "Noviembre" "Diciembre"

$C:
[1] "Otro nombre"

$Matriz:
Peso Altura Edad
[1,] 77 1.63 23
[2,] 58 1.63 23
[3,] 89 1.85 26
[4,] 55 1.62 23
[5,] 47 1.60 26
[6,] 60 1.63 26
[7,] 54 1.70 22
[8,] 58 1.65 23
[9,] 75 1.78 26
[10,] 65 1.70 24
[11,] 82 1.77 28
[12,] 85 1.83 42
[13,] 75 1.74 25
[14,] 65 1.65 26

> lista2[[3]][2,3]
Edad
23
> lista2[[1]][[1]][2]
[1] 2
```

La diferencia fundamental entre las tres formas, **[**, **[[** y **$** es que la primera permite seleccionar varios elementos, en tanto que las dos últimas solo permiten seleccionar uno. Además, **$** no permite utilizar índices calculados. El operador **[[** necesita que se le indiquen todos los índices (ya que debe seleccionar un sólo elemento) en tanto que **[** permite obviar

índices, en cuyo caso se seleccionan todos los valores posibles.

Si se aplican a una lista, **[[** devuelve el elemento de la lista especificado y **[** devuelve una lista con los elementos especificados.

2.6. Hoja de datos

La adaptación de la matriz de datos al uso habitual en Estadística es el objeto **data.frame**, que se puede traducir como *Hoja de datos*, aunque puede ser preferible mantener el nombre original por no existir una traducción estándar. La diferencia fundamental con la matriz de datos es que este objeto no tiene por qué estar compuesto de elementos del mismo tipo. Los objetos pueden ser vectores, factores, matrices, listas e incluso hojas de datos. Las matrices, listas y hojas de datos, contribuyen con tantas variables como columnas tengan. Los vectores numéricos y los factores se incluyen directamente y los vectores no numéricos se fuerzan como factores. Si no desea la expansión o la conversión, debe utilizar la función **I**. En caso de duda, debe tener en cuenta que el cambio se hace utilizando la función **as.data.frame**. Puede referirse a cualquier elemento de la hoja mediante dos índices, de modo similar a una matriz.

Ejemplo:

Las órdenes siguientes crean una hoja de datos con dos vectores, a continuación crean una nueva hoja uniendo la anterior con un vector de caracteres que representa el sexo.

```
> hoja1<-data.frame(Peso=c(90,87,60), Altura=c(1.85,1.87,1.63))
> hoja1
Peso Altura
1 90 1.85
2 87 1.87
3 60 1.63
> hoja2<-data.frame(hoja1,Sexo=c("H","H","M"))
> hoja2
Peso Altura Sexo
1 90 1.85 H
2 87 1.87 H
3 60 1.63 M
> codes(hoja2[,3])
[1] 1 1 2
```

Este vector, como se ha indicado, se transforma en un factor, como puede comprobar pidiendo sus códigos. Adviértase que el uso de corchetes individuales o dobles implica diferencia en los resultados.

```
> is.vector(hoja1)
[1] FALSE
> is.matrix(hoja1)
[1] FALSE
> is.data.frame(hoja1)
[1] TRUE
> is.factor(hoja2[3])
[1] FALSE
```

```
> is.factor(hoja2[[3]])
[1] TRUE
> hoja2[3]
Sexo
1 H
2 H
3 M
> hoja2[[3]]
[1] H H M
Levels: H M
> hoja2[,3]
[1] H H M
Levels: H M
> hoja2[[,3]]
Error in "[[.data.frame"(hoja2, , 3) : invalid subscript type
```

Para introducir un vector de nombres como tales, sin transformarlo en factores, debe utilizar la función **I**, como se muestra a continuación:

```
> Nombres=c("Pepe","Juan","Antonia")
> Nombres
[1] "Pepe" "Juan" "Antonia"
> Hoja3=data.frame(hoja2,Nombre=I(Nombres))
> Hoja3
Peso Altura Sexo Nombre
1 90 1.85 H Pepe
2 87 1.87 H Juan
3 60 1.63 M Antonia
```

Las estructuras de datos que más se utilizan, son precisamente las más complejas: La hoja de datos es la más apropiada para describir unos datos a analizar (ver **attach**), y la lista es la más apropiada para devolver datos desde una función, ya que permite devolver datos de varios tipos y, además, de modo estructurado, lo que facilita utilizar el resultado de una función como entrada de otra.

Si desea seleccionar un subconjunto de una hoja de datos, puede hacerlo con la función **subset**, función que puede utilizar también en vectores.

La sintaxis en vectores es **subset(x, subset, ...)** y en hojas de datos **subset(x, subset, select, drop = FALSE, ...)** donde los argumentos son

- **x** El objeto al que se aplica la selección, generalmente una hoja de datos.
- **subset** La condición que se aplica para seleccionar un subconjunto. En una hoja de datos pueden utilizarse los nombres de las variables (columnas) de la hoja.
- **select** Columnas que se desea conservar en una hoja de datos.
- **drop** Este argumento se pasa al método de indexación de hojas de datos.

Para realizar modificaciones en una hoja de datos es muy útil la función **tansform** que es de la forma **tansform(x, ...)**, donde **x** es la hoja de datos en la que se van a realizar transformaciones y el resto de argumentos son transformaciones posibles definidas de la

forma **variable=expresión**. Si el nombre de la variable coincide con una de las columnas de la hoja de datos se almacena allí la transformación, en caso contrario se añade una columna con el nombre de variable indicado.

Los siguientes ejemplos son ilustrativos de las situaciones comentadas.

```
> Hoja3
Peso Altura Sexo Nombre
1 90 1.85 H Pepe
2 87 1.87 H Juan
3 60 1.63 M Antonia
> subset(Hoja3,select=c(Sexo,Nombre))
Sexo Nombre
1 H Pepe
2 H Juan
3 M Antonia
> subset(Hoja3,subset=(Sexo=="H"))
Peso Altura Sexo Nombre
1 90 1.85 H Pepe
2 87 1.87 H Juan
> transform(Hoja3,Peso=log(Peso))
Peso Altura Sexo Nombre
1 4.499810 1.85 H Pepe
2 4.465908 1.87 H Juan
3 4.094345 1.63 M Antonia
> transform(Hoja3,LogPeso=log(Peso))
Peso Altura Sexo Nombre LogPeso
1 90 1.85 H Pepe 4.499810
2 87 1.87 H Juan 4.465908
3 60 1.63 M Antonia 4.094345
```

2.7. Comprobación y cambio de tipos de objetos

Hay una serie de funciones que permiten comprobar si un objeto es de un tipo determinado, todas comienzan por **is.**, o cambiarlo a un tipo concreto, que comienzan por **as.**. Así podrá (en un programa, claro está) querer dilucidar si un objeto es un vector o no. Puede hacerlo mediante la función **is.vector** que, aplicada al objeto, indica si es o no un vector, esto es, si es cierto (**T**) o falso (**F**) que es un vector. Del mismo modo puede comprobar si un número es entero, si es un valor lógico, etc.

Si necesita que el objeto sea de un tipo concreto puede transformarlo (si se puede) a ese tipo. Así, un vector puede transformarse en una hoja de datos mediante la función **as.data.frame**.

Utilizando la ayuda puede encontrar todas las funciones que comienzan con las palabras **is.** y **as.** utilizando la función **apropos**. Así, las siguientes:

```
is.R is.array is.atomic is.call is.character is.complex is.data.frame is.dou
is.element is.empty.model is.environment is.expression is.factor is.fin
is.function is.infinite is.integer is.language is.list is.loaded is.logical is.matrix is.
is.na.data.frame is.name is.nan is.null is.numeric is.object is.ordered is.pairl
```

is.qr is.real is.recursive is.single is.symbol is.ts is.vector

Ejemplo:

```
> x<-1:5
> is.vector(x)
[1] T
> is.data.frame(x)
[1] F
> x<-as.data.frame(x)
> is.data.frame(x)
[1] T
```