Data frames
- Creación de un *data frame*
- Acceder al contenido de un *data frame*
- Agregar filas y columnas a un *data frame*
- Nombres de filas y columnas
- *Data frames* y la escalabilidad

Listas
- Creación de una lista
- Acceso a los elementos de una lista
- Asignación de nombres a los elementos

3. Tipos de datos (II)

Los tipos de datos descritos en el capítulo previo comparten una característica común: todos los datos que contienen han de ser del mismo tipo. En un vector o una matriz, por ejemplo, no es posible guardar un número, una cadena de caracteres y un valor lógico. Todos los valores serán convertidos a un tipo común, habitualmente `character`. Esto dificulta la realización de operaciones sobre los valores de otros tipos.

En este capítulo conoceremos dos estructuras de datos más avanzadas de R: los *data frame* y las listas.

3.1 Data frames

El *data frame* es seguramente el tipo de dato más utilizado en R, implementándose internamente en forma de lista que tiene una determinada estructura. Un *data frame* está compuesto de múltiples columnas y filas, como una matriz, pero cada columna puede ser un tipo distinto. Al igual que las matrices, las columnas y filas pueden tener nombres, lo cual simplifica el acceso a la información como se explicará a continuación.

3.1.1 Creación de un data frame

La función encargada de crear objetos de este tipo es `data.frame()`. El resultado obtenido es un objeto clase `data.frame`, en el que cada columna aparecerá como una variable y cada fila como una observación. Todas las columnas han de tener el mismo número de filas.

Sintáxis 3.1 `data.frame(vector1, ..., vectorN [, row.names= nombresFilas, stringsAsFactors=TRUE|FALSE])`

Genera un nuevo data frame a partir de los datos contenidos en los vectores entregados como parámetros. Todos ellos deben tener el mismo número de filas. Los nombres de las columnas se establecerán a partir de los nombres de los vectores. Opcionalmente pueden facilitarse nombres para las filas con el parámetro `row.names`. Habitualmente `data.frame()` convertirá las cadenas de caracteres en

factors. Este comportamiento puede controlarse mediante el parámetro `stringsAsFactors`, asignándole el valor `FALSE` si deseamos preservar las cadenas como tales.

Las funciones `length()`, `ncol()` y `nrow()`, que usábamos en el capítulo previo con matrices, también se utilizan con *data frames*. En este caso, no obstante, la primera y la segunda son equivalentes, devolviendo el número de columnas. En el siguiente ejercicio se muestra cómo generar un *data frame* con tres columnas, llamadas `Dia`, `Estimado` y `Lectura`, conteniendo datos de un vector creado antes y dos creados dinámicamente mediante repetición y la función `rnorm()`:

Ejercicio 3.1 Creación de un *data frame*, visualización y obtención de su estructura

```
> df <- data.frame(Dia = fdias[1:20],
+                  Estimado = rep(c(T,F),10),
+                  Lectura = rnorm(20,5))
> head(df)

  Dia Estimado  Lectura
1 Dom     TRUE 3.845764
2 Mié    FALSE 5.986513
3 Jue     TRUE 2.547441
4 Jue    FALSE 5.714854
5 Dom     TRUE 6.426501
6 Lun    FALSE 5.417223

> c(length(df), ncol(df), nrow(df))

[1]  3  3 20
```

Podemos crear un *data frame* vacío, conteniendo únicamente el nombre de las columnas y sus respectivos tipos, facilitando a `data.frame()` exclusivamente esa información. Opcionalmente pueden indicarse un número de filas entre paréntesis, tomando estas un valor por defecto. Por ejemplo:

Ejercicio 3.2 Creación de *data frames* inicialmente vacios

```
> df2 <- data.frame(Dia = numeric(),
+                   Estimado = logical(),
+                   Lectura = numeric())
> df2

[1] Dia      Estimado Lectura
<0 rows> (or 0-length row.names)

> df3 <- data.frame(Dia = numeric(10),
+                   Estimado = logical(10),
+                   Lectura = numeric(10))
> df3
```

```
   Dia Estimado Lectura
1    0    FALSE       0
2    0    FALSE       0
3    0    FALSE       0
4    0    FALSE       0
5    0    FALSE       0
6    0    FALSE       0
7    0    FALSE       0
8    0    FALSE       0
9    0    FALSE       0
10   0    FALSE       0
```

■

También puede crearse un *data frame* a partir de una matriz y otros tipos de datos, mediante la función `as.data.frame()`. Para comprobar si un cierto objeto es o no un *data frame* podemos usar la función `is.data.frame()`.

Sintáxis 3.2 `as.data.frame(objeto [, stringsAsFactors=TRUE|FALSE])`

Facilita la conversión de objetos de otros tipos a tipo `data.frame`. Cada tipo de dato puede aportar su propia versión de esta función, con una implementación específica del proceso de conversión.

Sintáxis 3.3 `is.data.frame(objeto)`

Comprueba si el objeto facilitado como parámetro es o no de tipo `data.frame`, devolviendo `TRUE` o `FALSE` según corresponda.

3.1.2 Acceder al contenido de un data frame

A pesar de que en un `data.frame` cada columna puede contener datos de un tipo distinto, su estructura es similar a la de una matriz al ser una estructura de datos bidimensional, compuesta de filas y columnas. Por ello el método de acceso a su contenido, mediante el operador `[]`, sigue el mismo patrón:

Ejercicio 3.3 Acceso al contenido de un *data frame*

```
> df[5,3]  # Tercera columna de la quinta fila

[1] 6.426501

> df[5,]   # Quinta fila completa

  Dia Estimado  Lectura
5 Dom     TRUE 6.426501

> df[,3]   # Tercera columna completa

 [1] 3.845764 5.986513 2.547441 5.714854 6.426501 5.417223
 [7] 6.952092 3.963256 5.608792 4.580951 5.793826 4.644867
[13] 4.846451 6.433905 4.745405 6.325569 5.618622 4.464703
[19] 6.854527 4.002581
```

```
> df[c(-3,-6),] # Todo menos filas 3 y 6

   Dia Estimado  Lectura
1  Dom     TRUE 3.845764
2  Mié    FALSE 5.986513
4  Jue    FALSE 5.714854
5  Dom     TRUE 6.426501
7  Jue     TRUE 6.952092
8  Sáb    FALSE 3.963256
9  Mié     TRUE 5.608792
10 Mié    FALSE 4.580951
11 Jue     TRUE 5.793826
12 Mié    FALSE 4.644867
13 Mié     TRUE 4.846451
14 Dom    FALSE 6.433905
15 Vie     TRUE 4.745405
16 Jue    FALSE 6.325569
17 Jue     TRUE 5.618622
18 Sáb    FALSE 4.464703
19 Jue     TRUE 6.854527
20 Mar    FALSE 4.002581
```

■

Al trabajar con *data frames* es habitual utilizar la terminología de SQL para referirse a las operaciones de acceso al contenido de la estructura de datos. Así, al filtrado de filas se le llama normalmente *selección*, mientras que el filtrado de columnas es conocido como *proyección*.

Las columnas de un *data frame* son directamente accesibles mediante la notación `dataFrame$columna`[1], tal y como se muestra en el siguiente ejemplo:

Ejercicio 3.4 Acceso al contenido de un *data frame*

```
> df$Lectura

 [1] 3.845764 5.986513 2.547441 5.714854 6.426501 5.417223
 [7] 6.952092 3.963256 5.608792 4.580951 5.793826 4.644867
[13] 4.846451 6.433905 4.745405 6.325569 5.618622 4.464703
[19] 6.854527 4.002581
```

■

Proyección y selección pueden combinarse a fin de poder ejecutar consultas más complejas sobre el contenido de un *data frame*. En el sigueinte ejercicio se proponen dos ejemplos de esta técnica:

[1] Esta es la notación genérica que se usa en R para acceder a cualquier atributo de un objeto. Los *data frames* son objetos en los que cada columna es definida como un atributo.

Ejercicio 3.5 Ejemplos de proyección y selección de datos en un *data frame*

```
> df$Estimado==F

 [1] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
[11] FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE

> # Obtener el día y lectura de todas las filas en las que no se
> # haya estimado
> df[df$Estimado == F, c('Dia','Lectura')]

   Dia  Lectura
2  Mié 5.986513
4  Jue 5.714854
6  Lun 5.417223
8  Sáb 3.963256
10 Mié 4.580951
12 Mié 4.644867
14 Dom 6.433905
16 Jue 6.325569
18 Sáb 4.464703
20 Mar 4.002581

> # Filtrar también las filas cuya lectura sea <= que 3
> df[df$Estimado == F & df$Lectura > 3, c('Dia','Lectura')]

   Dia  Lectura
2  Mié 5.986513
4  Jue 5.714854
6  Lun 5.417223
8  Sáb 3.963256
10 Mié 4.580951
12 Mié 4.644867
14 Dom 6.433905
16 Jue 6.325569
18 Sáb 4.464703
20 Mar 4.002581
```

■

Además de para recuperar el contenido del *data frame*, las anteriores notaciones pueden también ser utilizadas para modificar cualquier dato. En el ejemplo siguiente se muestra cómo cambiar el mismo dato usando dos notaciones diferentes de las antes explicadas:

Ejercicio 3.6 Modificar el contenido de un *data frame*

```
> df[15,1] <- 'Vie'    # Acceso al mismo dato usando
> df$Dia[15] <- 'Vie'  # dos notaciones distintas
> df[12:17,]
```

```
   Dia Estimado  Lectura
12 Mié    FALSE 4.644867
13 Mié     TRUE 4.846451
14 Dom    FALSE 6.433905
15 Vie     TRUE 4.745405
16 Jue    FALSE 6.325569
17 Jue     TRUE 5.618622

>
```

3.1.3 Agregar filas y columnas a un data frame

Añadir e insertar nuevas filas y columnas en un *data frame* con contenido son operaciones que pueden efectuarse de diversas formas. La longitud de cualquier vector o matriz puede extenderse directamente con el operador `[]`, usando como índice el valor siguiente a la actual longitud. Esto también es válido para los *data frame*, pero hemos de tener en cuenta cómo afectará la operación a los tipos de las columnas.

Adición de nuevas filas

En el siguiente ejemplo se muestra cómo agregar una nueva fila con esta técnica básica. Antes y después de hacerlo se usa la función `str()` para obtener información básica de la estructura del *data frame*. Hemos de prestar atención a los tipos de cada variable:

Ejercicio 3.7 Agregar nuevas filas a un *data frame*

```
> str(df)

'data.frame':        20 obs. of  3 variables:
 $ Dia     : Factor w/ 7 levels "Dom","Jue","Lun",..: 1 5 2 2 1 3 2 6 5 5 ...
 $ Estimado: logi  TRUE FALSE TRUE FALSE TRUE FALSE ...
 $ Lectura : num  3.85 5.99 2.55 5.71 6.43 ...

> # Cuidado, se pierden los tipos de las columnas y todas pasan a ser character
> df[nrow(df)+1,] <- c('Vie', FALSE, 5)

> str(df)

'data.frame':        21 obs. of  3 variables:
 $ Dia     : Factor w/ 7 levels "Dom","Jue","Lun",..: 1 5 2 2 1 3 2 6 5 5 ...
 $ Estimado: chr  "TRUE" "FALSE" "TRUE" "FALSE" ...
 $ Lectura : chr  "3.84576411" "5.98651328" "2.54744080" "5.71485449" ...
```

La función `c()` que utilizamos para facilitar los datos de la nueva fila crea un vector temporal. Como ya sabemos, los vectores son estructuras en las que todos los elementos han de ser del mismo tipo, razon por la que `FALSE` y `5` se convierten al tipo `character`, al ser este el único compatible para los tres elementos. Al agregar la nueva fila el *data frame* establece como tipos de las columnas los correspondientes a los nuevos datos.

Para crear la nueva fila debemos usar la función `data.frame()`, de forma que lo que haríamos seria concatenar un nuevo *data frame* al final del ya existente. En lugar de usar la notación `df[nrow(df)+1,]`, que es completamente válida, podemos recurrir a la función `rbind()`.

Sintáxis 3.4 `rbind(objeto1, ..., objetoN)`

Concatena los objetos facilitados como argumentos por filas. Los objetos pueden ser vectores, matrices o `data.frames`. El tipo del resultado dependerá de los tipos de los objetos.

En el siguiente ejercicio se ofrecen dos ejemplos en los que se añade una nueva fila al final de las ya existentes:

Ejercicio 3.8 Agregar nuevas filas a un *data frame*

```
> df[nrow(df)+1,] <- data.frame('Vie', F, 5)
> str(df)

'data.frame':        21 obs. of  3 variables:
 $ Dia     : Factor w/ 7 levels "Dom","Jue","Lun",..: 1 5 2 2 1 3 2 6 5 5 ...
 $ Estimado: logi  TRUE FALSE TRUE FALSE TRUE FALSE ...
 $ Lectura : num  3.85 5.99 2.55 5.71 6.43 ...

> tail(df)

   Dia Estimado  Lectura
16 Jue    FALSE 6.325569
17 Jue     TRUE 5.618622
18 Sáb    FALSE 4.464703
19 Jue     TRUE 6.854527
20 Mar    FALSE 4.002581
21 Vie    FALSE 5.000000

> df <- rbind(df, data.frame(
+   Dia = fdias[1],
+   Estimado = T,
+   Lectura = 3.1415926))
> str(df)

'data.frame':        22 obs. of  3 variables:
 $ Dia     : Factor w/ 7 levels "Dom","Jue","Lun",..: 1 5 2 2 1 3 2 6 5 5 ...
 $ Estimado: logi  TRUE FALSE TRUE FALSE TRUE FALSE ...
 $ Lectura : num  3.85 5.99 2.55 5.71 6.43 ...

> tail(df)

   Dia Estimado  Lectura
17 Jue     TRUE 5.618622
18 Sáb    FALSE 4.464703
19 Jue     TRUE 6.854527
20 Mar    FALSE 4.002581
21 Vie    FALSE 5.000000
22 Dom     TRUE 3.141593
```

■

Inserción de filas

En caso de que las nuevas filas no hayan de añadirse al final de las ya existentes en el *data frame*, sino insertarse en una posición concreta, habremos de partir el *data frame* original en dos partes y colocar entre ellas la nueva fila. Para ello recurriremos nuevamente a la función `rbind()`, tal y como se aprecia en el siguiente ejercicio:

Ejercicio 3.9 Insertar filas en un *data frame*

```
> nuevaFila <- data.frame(Dia = fdias[1],
+                         Estimado = F,
+                         Lectura = 4242)
> df <- rbind(df[1:9,], nuevaFila, df[10:nrow(df),])
> df[8:14,]

    Dia Estimado     Lectura
8   Sáb    FALSE    3.963256
9   Mié     TRUE    5.608792
10  Dom    FALSE 4242.000000
101 Mié    FALSE    4.580951
11  Jue     TRUE    5.793826
12  Mié    FALSE    4.644867
13  Mié     TRUE    4.846451
```

■

Observa en el ejercicio previo cómo el número asociado a la antigua fila se ha cambiado, para evitar repeticiones. El nuevo identificador no es consecutivo, es decir, no se renumeran las filas del *data frame*.

Adición de nuevas columnas

Para agregar una nueva columna a un *data frame* hemos de facilitar un vector con la información. Dicho vector debería tener tantos elementos como filas haya actualmente en el *data frame*. La nueva columna puede añadirse usando directamente la notación `objeto$columna` o bien usando la función `cbind()`.

Sintáxis 3.5 `cbind(objeto1, ..., objetoN)`

Concatena los objetos facilitados como argumentos por columnas. Los objetos pueden ser vectores, matrices o `data.frames`. El tipo del resultado dependerá de los tipos de los objetos.

El siguiente ejercicio muestra cómo agregar dos nuevas columnas a nuestro anterior *data frame*. La primera se llamará `Lectura` y sería de tipo numérico, mientras que segunda tendrá el nombre `Fecha` y contendra una fecha.

Ejercicio 3.10 Agregar nuevas columnas a un *data frame*

```
> df$Ajustado <- df$Lectura + rnorm(nrow(df), 2)
> df <- cbind(df, Fecha = date())
> head(df)

  Dia Estimado  Lectura  Ajustado                    Fecha
1 Dom     TRUE 3.845764  6.623803 Wed Aug 20 11:02:46 2014
2 Mié    FALSE 5.986513  7.016683 Wed Aug 20 11:02:46 2014
3 Jue     TRUE 2.547441  4.659956 Wed Aug 20 11:02:46 2014
4 Jue    FALSE 5.714854  7.738610 Wed Aug 20 11:02:46 2014
```

```
5 Dom     TRUE 6.426501 10.375874 Wed Aug 20 11:02:46 2014
6 Lun    FALSE 5.417223  6.175288 Wed Aug 20 11:02:46 2014
```

■

Inserción de columnas

Al igual que ocurría con las filas, para insertar una columna en una posición concreta es necesario dividir el actual *data frame*, uniendo las partes para formar el nuevo mediante la función `cbind()`. El siguiente ejercicio demuestra cómo reconstruir el *data frame* de forma que las dos primeras columnas sean las antiguas primera y tercera. Estan irían seguidas de una nueva columna, tras la cual aparecería la antigua segunda. El resultado solamente se muestra por la consola, sin llegar a almacenarse en la variable:

Ejercicio 3.11 Insertar nuevas columnas en un *data frame*

```
> head(cbind(df[,c(1,3)],
+      Ajustado = df$Lectura + rnorm(nrow(df),2), df$Estimado))

  Dia  Lectura Ajustado df$Estimado
1 Dom 3.845764 7.378324        TRUE
2 Mié 5.986513 9.469816       FALSE
3 Jue 2.547441 4.657718        TRUE
4 Jue 5.714854 7.508201       FALSE
5 Dom 6.426501 9.779609        TRUE
6 Lun 5.417223 8.106187       FALSE
```

■

3.1.4 Nombres de filas y columnas

Al igual que las matrices, las filas y columnas de un *data frame* pueden tener asignados nombres. Estos se obtienen y modifican con las funciones `colnames()` y `rownames()` que conocimos en el capítulo previo, a las que hay que sumar la función `names()` que, en este contexto, sería equivalente a `colnames()`.

Ejercicio 3.12 Nombres de columnas y filas en un *data frame*

```
> names(df)

[1] "Dia"      "Estimado" "Lectura"  "Ajustado" "Fecha"

> colnames(df)

[1] "Dia"      "Estimado" "Lectura"  "Ajustado" "Fecha"

> rownames(df)

 [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
[11] "101" "11"  "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"
[21] "20"  "21"  "22"
```

3.1.5 Data frames y la escalabilidad

Como se apuntó anteriormente, el *data frame* es posiblemente el tipo de dato más usado en R. Esto, sin embargo, no significa que sea el más apropiado en todos los casos. Dependiendo de la cantidad de información a manejar, y de las operaciones a efectuar sobre ella, probablemente nos encontremos con problemas de escalabilidad. Leer en un *data frame* cientos de miles de filas, agregándolas una a una, puede requerir un tiempo considerable, ya que cada adición implica crear un nuevo *data frame* como resultado y descargar el anterior, dando más trabajo al recolector de basura[2].

Si se conoce de antemano el número de filas que tendrá el *data frame*, siempre puede reservarse toda la memoria que será necearia al principio, asignando después los valores a fila tal y como se demuestra en el siguiente ejercicio:

Ejercicio 3.13 Creación de un *data frame* reservando memoria al inicio

```
> n <- 15
> df <- data.frame(Lectura = numeric(n),
+                  Fecha = character(n),
+                  stringsAsFactors = FALSE)
> for(idx in 1:n) {
+   df$Lectura[idx] <- rnorm(1,10)
+   df$Fecha[idx] <- as.character(Sys.Date())
+ }
> head(df)

    Lectura      Fecha
1  9.860742 2014-08-20
2  9.570793 2014-08-20
3 10.829137 2014-08-20
4  9.619860 2014-08-20
5 11.259605 2014-08-20
6  8.705983 2014-08-20
```

Una alternativa, válida no solo para cargar datos más rápido sino en general para obtener un mejor rendimiento mientras trabajamos con grandes volúmenes de datos, es el tipo `data.table` ofrecido en el paquete del mismo nombre. Puedes instalar dicho paquete y usar la función `vignette()` para acceder a una introducción a su uso. En *Handling big data in R* [Pau13a] puedes encontrar algunos ejemplos y comparaciones de rendimiento interesantes.

[2] R cuenta con un GC (*Garbage collector*) o recolector de basura, encargado de liberar la memoria de los objetos que ya no son necesarios, por ejemplo tras haberlos destruido con `rm()`. También podemos invocarlo explícitamente mediante la función `gc()`

3.2 Listas

Es el tipo de dato más polifacético con que cuenta R. Una lista puede contener elementos de cualquier tipo, sin una estructura predefinida. Esto nos permite almacenar cualquier información e interpretarla como nos convenga en cada caso.

3.2.1 Creación de una lista

La función encargada de crear una nueva lista es `list()`. Podemos saber cuántos elementos contiene una lista con la función `length()`, que ya hemos usado en casos previos.

Sintáxis 3.6 `list(objeto1, ..., objetoN)`

Crea una nueva lista introduciendo como elementos los objetos entregados como parámetros.

Los objetos alojados en una lista pueden ser de cualquier tipo, incluyendo otras listas. Esto también significa que podemos incluir *data frames* como elementos de una lista. No hay más límite para la cantidad y complejidad de la información almacenada que la memoria disponible en el sistema.

Ejercicio 3.14 Creación de nuevas listas

```
> lst1 <- list(3.1415927, 'Hola', TRUE, fdias[4])
> lst2 <- list(fdias[1:10], mes, df)
> length(lst1)

[1] 4

> lst1

[[1]]
[1] 3.141593

[[2]]
[1] "Hola"

[[3]]
[1] TRUE

[[4]]
[1] Jue
Levels: Dom Jue Lun Mar Mié Sáb Vie

> length(lst2)

[1] 3

> lst2

[[1]]
 [1] Dom Mié Jue Jue Dom Lun Jue Sáb Mié Mié
```

```
Levels: Dom Jue Lun Mar Mié Sáb Vie

[[2]]
        Lun Mar Mié Jue Vie Sáb Dom
Semana1   1   2   3   4   5   6   7
Semana2   8   9  10  11  12  13  14
Semana3  15  16  17  18  19  20  21
Semana4  22  23  24  25  26  27  28
Semana5  29  30  31  32  33  34  35

[[3]]
     Lectura      Fecha
1   9.860742 2014-08-20
2   9.570793 2014-08-20
3  10.829137 2014-08-20
4   9.619860 2014-08-20
5  11.259605 2014-08-20
6   8.705983 2014-08-20
7  11.711551 2014-08-20
8   7.472963 2014-08-20
9  10.626633 2014-08-20
10  9.567412 2014-08-20
11  9.242191 2014-08-20
12 11.192863 2014-08-20
13 10.303550 2014-08-20
14  8.053791 2014-08-20
15 10.192173 2014-08-20
```

3.2.2 Acceso a los elementos de una lista

Al trabajar con listas, el habitual operador `[]` que hemos usado con matrices, vectores y *data frames* no devuelve el contenido de un elemento, sino una lista con el elemento o elementos desgignados por los índices. Para acceder al contenido propiamente dicho tenemso que utilizar el operador `[[]]`. El siguiente ejercicio muestra la diferencia entre ambos:

Ejercicio 3.15 Acceso a los elementos de una lista

```
> lst1[2]   # Una lista con el segundo elemento

[[1]]
[1] "Hola"

> lst1[[2]] # El contenido del segundo elemento

[1] "Hola"

> lst1[c(2,3)] # Una sublista
```

```
[[1]]
[1] "Hola"

[[2]]
[1] TRUE

> lst2[[3]][1] # Un elemento del dato contenido en un elemento

    Lectura
1  9.860742
2  9.570793
3 10.829137
4  9.619860
5 11.259605
6  8.705983
7 11.711551
8  7.472963
9 10.626633
10 9.567412
11 9.242191
12 11.192863
13 10.303550
14 8.053791
15 10.192173
```

Mediante la función `unlist()` podemos convertir una lista en un vector, facilitando así el acceso a su contenido. Esto tiene sentido especialmente en listas cuyos elementos son datos simples: números, cadenas, valores lógicos, etc. Si existen elementos complejos el resultado puede resultar difícil de tratar.

Sintáxis 3.7 `unlist(lista[, recursive=TRUE|FALSE])`

Genera un vector a partir del contenido de una lista. Si esta contuviese elementos complejos, tales como otras listas, el parámetro `recursive` determinará si también ha de aplicarse el proceso de simplificación a ellos.

El siguiente ejercicio muestra la diferencia entre aplicar recursivamente el proceso de conversión a cada elemento o no hacerlo:

Ejercicio 3.16 Conversión de listas a vectores

```
> unlist(lst2)

        "1"         "5"         "2"
        "2"         "1"         "3"
        "2"         "6"         "5"
        "5"         "1"         "8"
```

```
             "15"              "22"              "29"

              "2"               "9"              "16"

             "23"              "30"               "3"

             "10"              "17"              "24"

             "31"               "4"              "11"

             "18"              "25"              "32"

              "5"              "12"              "19"

             "26"              "33"               "6"

             "13"              "20"              "27"

             "34"               "7"              "14"

             "21"              "28"              "35"
          Lectura1          Lectura2          Lectura3
"9.86074237245708" "9.57079288767732" "10.8291371095133"
          Lectura4          Lectura5          Lectura6
"9.61985988287166"  "11.259605075469" "8.70598333620988"
          Lectura7          Lectura8          Lectura9
"11.7115514267809" "7.47296302377809" "10.6266327189315"
         Lectura10         Lectura11         Lectura12
 "9.5674122142443" "9.24219085599938" "11.1928627509596"
         Lectura13         Lectura14         Lectura15
"10.3035496828115" "8.05379071914909" "10.1921726943101"
            Fecha1            Fecha2            Fecha3
      "2014-08-20"      "2014-08-20"      "2014-08-20"
            Fecha4            Fecha5            Fecha6
      "2014-08-20"      "2014-08-20"      "2014-08-20"
            Fecha7            Fecha8            Fecha9
      "2014-08-20"      "2014-08-20"      "2014-08-20"
           Fecha10           Fecha11           Fecha12
      "2014-08-20"      "2014-08-20"      "2014-08-20"
           Fecha13           Fecha14           Fecha15
      "2014-08-20"      "2014-08-20"      "2014-08-20"

> unlist(lst2, recursive = FALSE)

[[1]]
[1] 1

[[2]]
[1] 5
```

```
[[3]]
[1] 2

[[4]]
[1] 2

[[5]]
[1] 1

[[6]]
[1] 3

[[7]]
[1] 2

[[8]]
[1] 6

[[9]]
[1] 5

[[10]]
[1] 5

[[11]]
[1] 1

[[12]]
[1] 8

[[13]]
[1] 15

[[14]]
[1] 22

[[15]]
[1] 29

[[16]]
[1] 2

[[17]]
[1] 9

[[18]]
[1] 16

[[19]]
```

```
[1] 23

[[20]]
[1] 30

[[21]]
[1] 3

[[22]]
[1] 10

[[23]]
[1] 17

[[24]]
[1] 24

[[25]]
[1] 31

[[26]]
[1] 4

[[27]]
[1] 11

[[28]]
[1] 18

[[29]]
[1] 25

[[30]]
[1] 32

[[31]]
[1] 5

[[32]]
[1] 12

[[33]]
[1] 19

[[34]]
[1] 26

[[35]]
[1] 33
```

```
[[36]]
[1] 6

[[37]]
[1] 13

[[38]]
[1] 20

[[39]]
[1] 27

[[40]]
[1] 34

[[41]]
[1] 7

[[42]]
[1] 14

[[43]]
[1] 21

[[44]]
[1] 28

[[45]]
[1] 35

$Lectura
 [1]  9.860742  9.570793 10.829137  9.619860 11.259605  8.705983
 [7] 11.711551  7.472963 10.626633  9.567412  9.242191 11.192863
[13] 10.303550  8.053791 10.192173

$Fecha
 [1] "2014-08-20" "2014-08-20" "2014-08-20" "2014-08-20"
 [5] "2014-08-20" "2014-08-20" "2014-08-20" "2014-08-20"
 [9] "2014-08-20" "2014-08-20" "2014-08-20" "2014-08-20"
[13] "2014-08-20" "2014-08-20" "2014-08-20"
```

3.2.3 Asignación de nombres a los elementos

Los elementos de una lista pueden tener asignados nombres. Estos se obtienen y establecen mediante la función `names()` que ya conocemos. El uso de nombres facilita el acceso a los elementos, especialmente cuando se usa la notación `lista$nombre` ya que permite prescindir del operador `[[]]` tal y como se aprecia en el siguiente ejercicio.

Ejercicio 3.17 Uso de nombres con listas

```
> names(lst1) <- c('PI','Mensaje','Activado','Inicio')
> lst1[[1]]

[1] 3.141593

> lst1[['PI']]

[1] 3.141593

> lst1$PI

[1] 3.141593
```

■