Apunts
Programació 2
Grau en Estadística

Marta Fairén
Edelmira Pasarella

10 d'abril de 2017

Índex

Capítol 1

Vectors (recordatori)

Comencem per recordar que un **vector** v és un objecte **estructurat** composat per un nombre finit d'objetes N anomenats components o elements. Tots els components d'un vector són del mateix tipus T. El tipus T és el **tipus base del vector**. És a dir, els vectors són objectes homogenis. De manera genèrica, el vector v del que estem parlant és de la forma:

t_1	t_2	t_3	t_4	t_5	t_6	$\ldots$	t_N

on $\forall i \in \{1, \ldots, N\}$, $t_i \in T$. És a dir, t_i es un valor del tipus base T. Un vector d'enters, de dimensió $N = 8$ podria ser aquest de sota:

1	2	3	4	-4	-3	-2	-1

Mentre que un vector de booleans, de dimensió $N = 4$ pot ser per exemple:

$TRUE$	$TRUE$	$FALSE$	$TRUE$

Pel que fa a requeriment de memòria de les variables de tipus vector, els seus components s'emmagatzemen de manera contigua a un bloc de memòria de mida igual a la dimensió del vector. Per a ser precisos hem d'aclarir que, en realitat, per a una variable vector es reserva espai a memòria equivalent a la dimensió del vector multiplicat per l'espai requerit per emmagatzemar un objecte del tipus base. El tipus base no sempre és un dels tipus base del lleguatge. Això vol dir que no sempre tenim vectors d'enters, booleans, reals i caracters. Podem tenim vectors de tipus estructurats també. En particular, els vector amb tipus base vectors (vectors de vectors) són coneguts com matrius i d'això en parlarem al Capítol 2.

1.1 Creació de vectors al llenguatge R

Al llenguatge R tenim varies funcions que són constructors de vectors.

1. La funció `c()`

   Aquest constructor ens permet crear vectors per enumeració dels seus components. El vector resultant tindrà dimensió igual al nombre d'elements que s'han fet servir a la crida de la funció `c()`.

```
> v <- c(0.4, 0.1)            # N = 2
> v
[1] 0.4 0.1
>
> v <- c(1, 2, 3, 4, 5, 6, 7)       # N = 7
> v
[1] 1 2 3 4 5 6 7
>
> v <- c(TRUE, FALSE)         # N = 2
> v
[1]  TRUE FALSE
>
> v <- c("a", "b", "c","d","e")     # N = 5
> v
[1] "a" "b" "c" "d" "e"
>
```

   S'ha de notar que poden crear vectors buits:

```
> v <- c()
> v
NULL
>
```

2. La funció `vector(Tipus_Base, Dimensió)`

   Aquest altre constructor ens permet crear vectors d'un tipus base i d'una dimensió donada. Si el Tipus_Base no es dona, per defecte, crea un vector de booleans inicialitzat en FALSE.

```
> v<-vector(length=6)
> v
[1] FALSE FALSE FALSE FALSE FALSE FALSE
>
```

3. L'operador `:`

   Aquest operador permet crear vectors d'un rang de números.

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
> v <- 1:10
> v[2]
```

```
[1] 2
> v[2] <- 4
> v
 [1]  1  4  3  4  5  6  7  8  9 10
>
> v <- 4:9
> v
[1] 4 5 6 7 8 9
>
> v <- 12:7
> v
[1] 12 11 10  9  8  7
>
```

4. Altres operadors que ens permeten construïr vectors: `seq()` and `rep()`,

```
> v <- seq(from=16,to=24, by=2)
> v
[1] 16 18 20 22 24
>
> v <- seq(from=16,to=4, by=-3)
> v
[1] 16 13 10  7  4
>
> v<- seq(8)
> v
[1] 1 2 3 4 5 6 7 8
>
> v <- rep(1,5)
> v
[1] 1 1 1 1 1
>
> v <- rep(15,3)
> v
[1] 15 15 15
>
```

I també podem combinar els operadors:

```
> v <- c(10,rep(2,5),9)
> v
[1] 10  2  2  2  2  2  9
>
> v <- c(seq(5))
> v
[1] 1 2 3 4 5
> v <- c(13:20)
> v
[1] 13 14 15 16 17 18 19 20
>
> v <- c(seq(from=15, to=25, by=3))
```

```
> v
[1] 15 18 21 24
>
```

Un cop s'ha creat un vector v, no es pot canviar la seva dimensió. Malgrat això, podem "redimensionar" un vector con segueix:

```
> v <- 1:4
> v
[1] 1 2 3 4
> v <- c(v,10)
> v
[1]  1  2  3  4 10
> v <- c(v,10)
> v <- c(10,v)
> v
[1] 10  1  2  3  4 10 10
> v <- c(10:20,v)
> v
 [1] 10 11 12 13 14 15 16 17 18 19 20 10  1  2  3  4
    10 10
```

Amb les instruccions de dalt sembla que estem redimensionant el vector v. En realitat, aquestes instruccions són força costoses perquè cada vegada que s'afegeix un o més components, es crea un nou vector a un altre lloc de memoria i el vector original s'ha de relocalitzar totalment. Llavors, s'ha de fer servir amb molta cura aquest tipus d'instruccions.

1.2 Accés als components d'un vector

Mitjaçant un **operador d'accés directe**, que té com a paràmetre un enter que correspon a un índex que indica la posició que es desitja visitar, es pot accedir a qualsevol component del vector de manera directa. Matemàticament l'especificació d'aquesta funció és la següent:

$$[\ \ ] : vector \times enter \to T$$

Llavors, donat un vector v i una expressió de tipus enter exp, fem servir la crida

$$v[exp]$$

amb

$$1 \leq avaluacio(exp) \leq N \qquad (1)$$

per referir-nos al component del vector que es troba a la posició que resulti d'avaluar exp.

Aleshores, no cal visitar els components anteriors a un component donat per arribar-hi.

```
> v <- c(0.4, 0.1)
> v[1]
[1] 0.4
```

```
>
> v <- c(1, 2, 3, 4, 5, 6, 7)
> v[5]
[1] 5
>
> v <- c(TRUE, FALSE)
> v[2]
[1] FALSE
> v <- c("a", "b", "c","d","e")
> i <- 2
> v[i]
[1] "b"
> v[i+1]
[1] "c"
> v[i*2]
[1] "d"
> j <- 3
> v[i+j]
[1] "e"
> v[2*i+1]
[1] "e"
>
```

Si assignem un valor numèric a qualsevol component d'un vector creat amb l'operador `vector()`, el tipus base canvia.

```
> v<-vector(length=6)
> v
[1] FALSE FALSE FALSE FALSE FALSE FALSE
>
> v[4] <- 55
> v
[1]  0  0  0 55  0  0
>
```

És molt important vigilar que es satisfaci la restricció (1). En cas que no la respectem tindrem un problema en intentar accedir a una posiciò no definida dins del vector. En aquests casos, l'R ens respon amb `NA` (Not Available):

```
> v <- c("a", "b", "c","d","e")
> v[6]
[1] NA
>
```

En particular, quan el vector és buit, tindrem:

```
> v <- c()
> v
NULL
> v[1]
NULL
>
```

S'ha de tenir en compte que, segons la definició de vectors, $v[exp] = t_{avaluacio(exp)} \in T$ i que per tant, aquest component es comporta com una variable del tipus base i podem fer servir qualsevol dels seus operadors.

```
> w <- rep(8,7)
> w
[1] 8 8 8 8 8 8 8
> v <- 5:10
> v
[1]  5  6  7  8  9 10
> cat(v[5]+w[6],"\n")
17
> y <- 5*v[3]
> y
[1] 35
>
```

Ara que hem vist la necessitat que els accessos als components d'un vector siguin posicions permeses, surt la necessitat de conèixer la dimensió d'un vector. Per fer-ho disposem a l'R d'una funció especificada com segueix:

$$length : vector \rightarrow enter$$

Així doncs, si la dimensió d'un vector v és N, la crida

$$length(v)$$

ens tornarà N.

```
> v<-c()
> length(v)
[1] 0
>
> v <- c("a", "b", "c","d","e")
> length(v)
[1] 5
>
> v <- c(TRUE, FALSE)
> length(v)
[1] 2
>
```

1.3 Recorregut

Hi ha molts problemes on s'han de processar tots i cadascun dels components d'una seqüència que, aleshores, pot estar representada per un vector. Aquesta classe de problemas se'ls anomena "Recorreguts". Per exemple, per calcular la mitjana d'una seqüència de notes d'estudiants, per avaluar un polinomi donat els seus coeficients, hem de fer un recorregut. Hi ha molts problemes que requereixen que tots els elemnts de l'entrada es processin. Llavors, per fer recorreguts d'un vector, tindrem un esquema genèric com aquest:

```
A
for (i in 1:length(v)){
   processar(v[i])
}
B
```

On `A` correspon a un bloc d'instruccions de preprocessament, per exemple, la inicialització d'algunes variables, i `B` correspon a un bloc d'instruccions de postprocessament, per exemple, imprimir el resultat.

Un primer exemple molt simple és l'escriptura dels components d'un vector:

```
escriure_vector <- function(v){
   for (i in 1:length(v)){
      cat(v[i],"\n")
   }
}
```

I es pot utilitzar d'aquesta forma:

```
> v<- 2:15
> escriure_vector(v)
2
3
4
5
6
7
8
9
10
11
12
13
14
15
>
```

Suposem que necessitem una funció per llegir un vector. Aquesta funció pot ser con segueix

```
llegir_vector <- function(v){
   for (i in 1:length(v)){
      v[i] <- scan(n=1,quiet=TRUE)
   }
   return (v)
}
```

I la podem fer servir de la següent manera:

```
> v <- vector(length = 8)
> v <- llegir_vector(v)
1: 2
1: 4
1: 6
```

```
1: 8
1: 10
1: 12
1: 14
1: 16
> v
[1]  2  4  6  8 10 12 14 16
>
```

Amdós exemples previs, no requereixen cap preprocessament o postprocesament. Considerem ara el problema de trobar el valor màxim dins d'un vector no buit (aquesta és la precondició del problema):

```
maxim <- function(v){
   max<- v[1]
   for(i in 2:length(v)){
      if (v[i] > max) max <- v[i]
   }
   return (max)
}
```

```
> v<-c(12,5,6,7,10,100,4,5, 56,2,20)
> maxim(v)
[1] 100
> v<-c(12,5,6,7,10,100,4,5, 56,2,200)
> maxim(v)
[1] 200
> v<-c(1200,5,6,7,10,100,4,5, 56,2,20)
> maxim(v)
[1] 1200
>
```

Un altre exemple és la funció que calcula el producte escalar de dos vectors de la mateixa dimensió (precondició):

```
producte_escalar <- function(v,w){
   p <- 0
   for (i in 1:length(v)){
      p <- p + v[i]*w[i]
   }
   return (p)
}
```

```
> v <- rep(2,3)
> v
[1] 2 2 2
> w <- 1:3
> w
[1] 1 2 3
> producte_escalar(v,w)
[1] 12
>
```

1.4 Cerca

Hi ha una classe de problemes on no cal visitar o processar tots els elements, és a dir, fer un recorregut, perquè ens demanen confirmar un predicat, és a dir, una propietat, `P(s)`, sobre la seqüència `s` representada amb un vector `v`, com per exemple:

- `P(s)` = "la seqüència `s` té múltiples de 3".
  Aquesta propietat vol dir el seguent: $\exists i : \ 1 \leq i \leq n : \ v[i] \, mod \, 3 = 0$ i això s'ha de comprovar. Anirem processant cadascun dels components del vector i si trobem un que sigui múltiple de 3, hem d'aturar la comprovació i donar una resposta afirmativa. En cas que no hi hagi cap múltiple de 3 al vector, acabarem fent un recorregut i la resposta seria negativa. Un possible script a l'R és el seguent:

```
multiploe3 <- function(v){
   i <- 1
   trobat <- FALSE
   while (i <= length(v) && !trobat){
      trobat <- v[i] %% 3 == 0
      i <- i + 1
   }
   return (trobat)
}
```

- `P(s)` = "la seqüència `s` té un cim".
  Per a aquest problema és necesari conèixer la definició de cim. Atès que la seqüència `s` es representa com un vector `s`, un cim és un component $v[i]$, $2 \leq i \leq n-1$, de v tal que $v[i-1] < v[i] < v[i+1]$. Si, mentre es fa el processament del vector es troba un component que satisfaci aquesta propietat s'ha d'aturar el recorregut perquè ja podem donar una resposta afirmativa. En cas contrari, si haguéssim de fer el processament de tots els components del vector, és a dir, el recorregut del vector, obtindríem una resposta negativa. Obviament, per comprobar si un vector té un cim, es requereix que tingui com a mínim tres elements. Llavors, aquesta és la precondició del problema: la dimensió de v ha de ser més gran o igual a 3. Un possible script a l'R és el seguent:

```
#Precondicio: length(v) >= 3
cim <- function(v){
   ant <- 1
   act <- 2
   suc <- 3
   trobat <- FALSE
   while (suc <= length(v) && !trobat){
      trobat <- v[ant] < v[act] && v[act] > v[suc]
      ant <- act
      act <- suc
      suc <- suc + 1
   }
   return (trobat)
}
```

- `P(s)` = "la seqüência `s` és creixent"
  Si analitzem aquesta propietat, ens adonarem que en realitat vol dir el seguent: $\forall i: 1 \leq i \leq n-1: v[i] \leq v[i+1]$ i això és el que realment hem de comprovar. En aquest cas, la cerca consisteix en trobar un contraexemple d'aquesta propietat. És a dir, si trobem un parell de components consecutius del vector tal que $1 \leq i \leq n-1: v[i] > v[i+1]$ hem d'aturar la comprovació de la propietat i donar una resposta negativa. En aquest cas, si féssim el recorregut, la resposta seria afirmativa. S'ha de notar que, segons la definició, tota seqüència buida o que tingui un sol element és creixent. Un possible script en R és el seguent:

```
creixent <- function(v){
   i <- 1
   continuar <- TRUE
   while (i < length(v) && continuar){
      continuar <- v[i] < v[i+1]
      i <- i + 1
   }
   return (continuar)
}
```

- `P(s)` = "la seqüència `s` no té cap element parell".
  Si analitzem aquesta propietat, ens adonarem que en realitat vol dir el seguent: $\forall i: 1 \leq i \leq n: v[i] \; mod \; 2 \neq 0$ no és parell i això és el que realment hem de comprovar. Un altre cop, el problema consisteix en trobar un contraexemple de la propietat. És a dir, si trobem un component parell del vector que representa la seqüència, hem d'aturar la comprovació i donar una resposta negativa. Si féssim el recorregut, la resposta seria afirmativa. Un possible script a l'R és el seguent:

```
noparells <- function(v){
   i <- 1
   continuar <- TRUE
   while (i <= length(v) && continuar){
      continuar <- v[i] %% 2 != 0
      i <- i + 1
   }
   return (continuar)
}
```

1.5 Sorting

Una família molt important d'algorismes és la família d'algorismes d'ordenament d'un vector. El problema pot ser vist de la següent manera: donat un vector, s'han de reorganitzar els seus components de forma tal que el vector arribi a representar una seqüència creixent (també pot ser decreixent). Això es pot formalitzar com segueix:

$$sort(v, v') = permutacio(v, v') \text{ i } \forall i: 1 \leq i \leq n-1: v'[i] \leq v'[i+1]$$

On el predicat $permutacio(v, v')$ diu que v' és una permutació de v. Els algorismes bàsics d'ordenament són el `SelectSort` (ordenament per selecció), l'`InsertSort` (ordenament per inserció) i el `BubbleSort` (ordenament per intercanvi o mètode de la bombolla). La dificultat que tenen aquests algorismes és que el vector s'ha d'ordenat sense fer servir cap emmagazatment addicional. És a dir, només s'ha de fer servir l'espai a memòria del vector v. Podeu trobar més detalls d'aquests algorismes a [1]. A sota presentem una implementació a l'R del `SelectSort`.

```
1  buscarmin <- function(v,inicio,final){
2     pos_min <- inicio
3     i <- inicio+1
4     while (i <= final){
5        if (v[i] < v[pos_min]) pos_min <- i
6        i <- i+1
7     }
8     return (pos_min)
9  }
10
11 SelectSort <- function(v){
12    n <- length(v)
13    i <- 1
14    #Durant la resta de l'execucio es mante que
15    #el segment v[1..i-1] esta ordenat
16    while (i <= n-1){
17       #Busca el component minim a v[i..n]
18       pos_min <- buscarmin(v,i,n)
19       #Intercanvia v[i] amb v[pos_min]
20       aux <- v[i]
21       v[i] <- v[pos_min]
22       v[pos_min] <- aux
23       i <- i+1
24    }
25    return (v)
26 }
```

La funció sort a l'R. El llenguatge R disposa d'una funció que ordena un vector de manera creixent (opció per defecte) o decreixent. La discusió en relació a l'algorisme d'ordenament que implementa aquesta funció està fora de l'abast d'aquest document.

```
1  > v <- c(9,3,5,1,7,9,5) sort(v) [1] 1 3 5 5 7 9 9
2  > sort(v,decreasing=TRUE) [1] 9 9 7 5 5 3 1
```

Capítol 2

Matrius

Una **matriu** m és un objecte **estructurat**, bidimensional, que pot ser vist com una taula. Des del punt de vista d'estructures de dades, una matriu és un vector de vectors. És a dir, un vector on cada component és un vector. Aleshores, al igual que els vectors, tots els components d'una matriu són del mateix tipus base T. Llavors, una matriu és un objecte homogeni, composat per un nombre finit de files F (entrades horitzontals) i un nombre finit de columnes C (entrades verticals) anomenats components o elements. De manera genèrica, la matriu $m_{F\times C}$ de la que estem parlant és de la forma:

$$\overbrace{\begin{bmatrix} m_{1,1} & m_{1,2} & \dots & m_{1,C-1} & m_{1,C} \\ m_{2,1} & m_{2,2} & \dots & m_{2,C-1} & m_{2,C} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{F-1,1} & m_{F-1,2} & \dots & m_{F-1,C-1} & m_{F-1,C} \\ m_{F,1} & m_{F,2} & \dots & m_{F,C-1} & m_{F,C} \end{bmatrix}}^{C\ columnes}$$
$$F\ files$$

Com podem veure, la matriu m té dimensió $F \times C$ i per tant té $F \times C$ components on $\forall i \in \{1, \dots, F\}$ $\forall j \in \{1, \dots, C\}$, $m_{i,j} \in T$. És a dir, $m_{i,j}$ és un valor del tipus base T. Una matriu d'enters, de dimensió 4×4 podria ser aquesta:

$$m_{4\times 4} = \begin{bmatrix} 1 & 2 & 3 & 7 \\ 4 & 4 & 3 & 2 \\ 2 & 1 & 6 & 1 \\ 3 & 1 & 5 & 1 \end{bmatrix}$$

Mentre que una matriu de booleans, de dimensió 3×2 pot ser per exemple:

$$m_{3\times 2} = \begin{bmatrix} TRUE & TRUE \\ FALSE & TRUE \\ TRUE & FALSE \end{bmatrix}$$

2.1 Creació de matrius al llenguatge R

Al llenguatge R tenim la següent funció per construir matrius.

Funció `matrix()`

Aquest constructor ens permet crear matrius per enumeració dels seus components, i.e. mitjançant un vector.

1. El nombre de files, $matrix(c(e_1, ..., e_n), nrow = F)$

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10, 11,12), nrow
    =4)
> m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
>
```

2. El nombre de columnas, $matrix(c(e_1, ..., e_n), ncol = C)$

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10, 11,12), ncol
    =4)
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
>
```

3. Ambdos, $matrix(c(e_1, ..., e_n), nrow = F, ncol = C)$

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10, 11,12), nrow
    =4,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
>
```

4. Cap modalitat $matrix(c(e_1, ..., e_n))$. En aquest cas els elements es colůloquen en una matriu d'una única columna.

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10, 11,12))
> m
      [,1]
 [1,]    1
 [2,]    2
 [3,]    3
 [4,]    4
 [5,]    5
 [6,]    6
 [7,]    7
 [8,]    8
 [9,]    9
[10,]   10
[11,]   11
[12,]   12
>
```

IMPORTANT Els components d'una matriu creada amb la funció `matrix()` són emmagatzemats per columnes.

La matriu resultant tindrà dimensió igual al nombre d'elements que s'han fet servir a la crida de la funció `matrix()`, és a dir n. Aleshores, n ha de ser múltiple de les files i de les columnes.

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), nrow=5)
```

```
Mensajes de aviso perdidos
In matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 5) :
la longitud de los datos [12] no es un submúltiplo o múltiplo del número
de filas [5] en la matriz
```

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10, 11,12), ncol=7)
```

```
Mensajes de aviso perdidos
In matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), ncol = 7) :
la longitud de los datos [12] no es un submúltiplo o múltiplo del número
de columnas [7] en la matriz
```

```
> m <- matrix(c(1,2,3,4,5,6,7,8,9,10, 11,12), nrow=2)
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11
[2,]    2    4    6    8   10   12
>
```

De tota manera, si volem emmagatzemar els components per files podem fer servir la modalitat `matrix(c(...),nrow=F, byrow=TRUE)`

Com al cas dels vectors, es poden crear matrius "buides". En realitat són matrius on els seus components tenen el valor `NA`.

```
> m <- matrix(nrow=4,ncol=3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
```

```
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
> m <- matrix(nrow=4)
> m
     [,1]
[1,]   NA
[2,]   NA
[3,]   NA
[4,]   NA
> m <- matrix(ncol=3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
>
> m <- matrix()
> m
     [,1]
[1,]   NA
>
```

A diferència dels vectors, després d'assignar un valor d'un tipus T a algún dels components de la matriu, la resta dels component mantenen el valor `NA`.

```
> m <- matrix(nrow=4,ncol=3)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
[4,]   NA   NA   NA
> m[3,2] <- 100
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA  100   NA
[4,]   NA   NA   NA
> m <- matrix(nrow=4,ncol=3)
> m[3,2] <- TRUE
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA TRUE   NA
[4,]   NA   NA   NA
>
```

2.2 Accés als components d'una matriu

S'accedeix als components d'una matriu mitjaçant un **operador d'accés directe**, que té com a paràmetres dos enters que corresponen a un índex per a la fila i un índex per a la columna que es desitja accedir, respectivament. És a dir, indicant la posició que es desitja visitar es pot accedir a qualsevol component del vector de manera directa. Matemàticament l'especificació d'aquesta funció és la següent:

$$[\ \ ] : matriu \times enter \times enter \rightarrow T$$

Llavors, donada una matriu m i un parell d'expressions de tipus enter, exp_f i exp_c, fem servir la crida

$$m[exp_f, exp_c]$$

amb

$$1 \leq avaluacio(exp_f) \leq F \qquad (2)$$

$$1 \leq avaluacio(exp_c) \leq C \qquad (3)$$

per referir-nos al component de la matriu que es troba a la posició $\langle exp_f, exp_c \rangle$.

Aleshores, com al cas dels vectors, tenim accés directe per visitar els components d'una matriu.

```
> m
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> m[3,2]
[1] 7
> m[1,3]
[1] 9
> i <- 2
> m[2*i,i]
[1] 8
> j <- 2
> m[i+j,3]
[1] 12
>
```

Una de les facilitas per manipular matrius que ens ofereix l'R és que podem accedir no només als components de les matrius si no també a submatrius. En particular, podem agafar una fila o una columna sencera.

```
> m
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> m[,2]       #accedeix a la columna 2
[1] 4 5 6
> m[2,]      #accedeix a la fila 2
[1] 2 5
```

```
> m[1:2,]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
>
```

És molt important vigilar que es satisfacin les restriccions (2) i (3). En cas que no es respectin aquestes restriccions, rebrem un missatge d'error en intentar accedir a una posició ilůlegal dins de la matriu considerada.

```
> m
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> m[3,7]
Error en m[3, 7] :  subíndice fuera de los límites
```

Semblant als vectors, els components d'una matriu pertanyen al tipus base T, llavors $m[exp_f, exp_c] \in T$ i que per tant, aquest component es comporta com una variable del tipus T i podem fer servir qualsevol dels seus operadors.

```
> b
      [,1]  [,2]
[1,]  TRUE  TRUE
[2,] FALSE  TRUE
[3,] FALSE FALSE
[4,]  TRUE FALSE
> b[1,2] && b[3,1]
[1] FALSE
> b[1,2] && b[4,1]
[1] TRUE
> m
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> cat(m[2,2]+m[3,1],"\n")
8
>
```

A més dels operadors mencionats a dalt, podem conèixer la dimensió d'una matriu i el número de components que té mitjançant els següents operadors:

$$dim : matriu \to enter \times enter$$

$$length : matriu \to enter$$

$$nrow : matriu \times enter \to enter$$

$$ncol : matriu \times enter \to enter$$

```
> m <- matrix(c(1,2,3,4,5,6),ncol=2)
> m
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> dim(m)
[1] 3 2
> c <- dim(m)
> c
[1] 3 2
> length(m)
[1] 6
> nrow(m)
[1] 3
> ncol(m)
[1] 2
>
```

2.3 Recorregut

Hi ha molts problemes on s'han de processar tots i cadascun dels components d'una matriu. Com ja hem dit al capítol 1, aquesta classe de problemas se'ls anomena "Recorreguts". Per recórrer tots els elements d'una matriu hem d'accedir a tots ells. Això ho podem fer per files o per columnes. El següent esquema és un recorregut per files d'una matriu:

```
A
for (i in 1:nrow(m)){
   for (j in 1:ncol(m)){
      processar(m[i,j])
   }
}
B
```

On `A` correspon a un bloc d'instruccions de preprocessament, per exemple, la inicialització d'algunes variables, i `B` correspon a un bloc d'instruccions de postprocessament, per exemple, imprimir el resultat.

També es pot fer el recorregut d'una matriu per columnes amb l'esquema de sota:

```
A
for (j in 1:ncol(m)){
   for (i in 1:nrow(m)){
      processar(m[i,j])
   }
}
B
```

Com a exemple podem veure l'escriptura dels components d'una matriu m:

```
escriure_matriu <- function(m){
   for (i in 1:nrow(m)){
      for (j in 1:ncol(m)){
         cat(m[i,j],"\n")
      }
   }
}
```

I aquesta funció es pot utilitzar d'aquesta forma:

```
> m
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> escriure_matriu(m)
1
2
3
4
5
6
>
```

També podem donar format de matriu a aquesta sortida fent la funció com segueix:

```
escriure_matriu <- function(m){
   for (i in 1:nrow(m)){
      for (j in 1:ncol(m)){
         cat(m[i,j]," ")
      }
                  cat(m[i,j],"\n")
   }
}
```

I això donarà la sortida següent:

```
> m
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
> escriure_matriu(m)
1 2
3 4
5 6
>
```

Suposem que necessitem una funció per llegir (per files) una matriu. Aquesta funció pot ser con segueix

```
llegir_matriu <- function(F,C){
   m <- matrix(nrow=F,ncol=C)
```

```
3     for (i in 1:F){
4        for (j in 1:C){
5           m[i,j] <- scan(n=1,quiet=TRUE)
6        }
7     }
8     return (m)
9  }
```

I la podem fer servir de la següent manera:

```
1  > m <- llegir_matriu(3,2)
2  1: 1
3  1: 2
4  1: 3
5  1: 4
6  1: 5
7  1: 6
8  > m
9        [,1] [,2]
10 [1,]    1    2
11 [2,]    3    4
12 [3,]    5    6
13 >
```

2.4 Cerca

Com al cas dels vectors, tenim una classe de problemes on no cal visitar o processar tots els elements, és a dir, fer un recorregut, perquè ens demanen confirmar un predicat, és a dir, una propietat, `P(m)`, sobre la matriu `m` . Per resoldre els problemes de cerca, s'han de fer servir les mateixes tècniques que hem explicat en la Secció 1.4. Per exemple:

`P(s)` = "els components de la matriu $m_{F\times C}$ formen una seqüencia estrictament creixent quan es recorren per fila".
Si analitzem aquesta propietat, ens adonarem que en realitat vol dir el següent: $\forall i \forall j : 1 \leq i \leq F : 1 < j \leq C : m[i,(j-1)] \leq m[i,j]$ i $\forall i : 1 < i \leq F : m[(i-1),C] \leq m[i,1]$. Llavors, això és el que realment hem de comprovar. En aquest cas, la cerca consisteix en trobar un contraexemple d'aquesta propietat. És a dir, si trobem un parell de components consecutius del recorregut per files de la matriu tal que el primer sigui més gran que el segon hem d'aturar la comprovació de la propietat i donar una resposta negativa. S'ha de notar que, segons la definició, tota seqüència buida o que tingui un sol element és creixent. Un possible script en R és el seguent:

```
1  creixent <- function(m){
2     ant <- m[1,1]
3     for (i in 2:nrow(m)){
4        for (j in 1:ncol(m)){
5           if (m[i,j] <= ant) return (FALSE)
6           else ant <- m[i,j]
```

```
      }
    }
    return (TRUE)
}

> m <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
>
> creixent(m)
[1] TRUE
> m <- matrix(c(1,2,3,3,4,5,5,5,6),nrow=3,byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    3    4    5
[3,]    5    5    6
> creixent(m)
[1] FALSE
>
> m <- matrix(c(1,2,3,3,4,5,5,5,6),nrow=3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    5
[3,]    3    5    6
> creixent(m)
[1] FALSE
>
> m <- matrix(c(3,5,6,7,9,11),nrow=3,byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    9    2    3
[2,]    3    4    5
[3,]    5    5    6
> creixent(m)
[1] TRUE
>
> m <- matrix(c(1,2,3,3,4,5,5,5,-1),nrow=3,byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    3    4    5
[3,]    5    5   -1
> creixent(m)
[1] FALSE
>
```

Deixem com a exercici al lector fer la implementació corresponent a verificar si una matriu m és estrictament creixent quan es recorre (a) per columnes, (b) en ziga-zaga horitzontal (començant des de la posició $m[1,1]$ d'esquerra a dreta) i (c) en ziga-zaga vertical (començant des de la posició $m[1,1]$ d'adalt a baix).

Capítol 3

Lists

Un **list** l és un objecte **estructurat** composat per objetes anomenats components o elements. A diferència d'un vector, els components d'un list poden ser de tipus diferents. Aleshores, els lists són objectes heterogenis. De manera genèrica, un list l és de la forma:

t_1	t_2	t_3	t_4	t_5	t_6	$\ldots$	t_N

on $\forall i \in \{1, \ldots, N\}$, $t_i \in T_i$. És a dir, t_i és un valor del tipus T_i i $\forall i \neq j$, $i, j \in \{1, \ldots, N\}$ no necessàriament $T_i = T_j$.

Per exemple, considerem el següent list:

$Enric$	$Canals$	21	H	1.77	$FALSE$	66543211	$93 - 4444446$

En aquest exemple el list té components de diferents tipus: cadena de caracteres, enter, real i booleà.

Els components poden rebre un nom que facilita l'accés als components i permet un nivell d'abstracció més alt.

nom_1	nom_2	nom_3	nom_4	nom_5	nom_6	$\ldots$	nom_N
t_1	t_2	t_3	t_4	t_5	t_6	$\ldots$	t_N

En aquest list, $\forall i \in \{1, \ldots, N\}$, $t_i \in T_i$ i nom_i és l'identificador del i-éssim component. Com veurem a l'apartat 3.1, els noms d'un list poden ser dinàmicament assignats. A sota podem veure el list de l'exemple anterior amb noms per als components:

Nom	$Cognom$	$Edat$	$Sexe$	$Alçada$	$Treballa$	$Mobil$	$Telefon$
$Enric$	$Canals$	21	H	1.77	$FALSE$	66543211	$93 - 4444446$

En aquest exemple el list té components anomenats “Nom” i “Cognom” de tipus cadena de caràcters, “Edat” de tipus enter, “Sexe” de tipus caràcter (’H’/’D’), “Alçada” de tipus real, “Treballa” de tipus booleà (TRUE/FALSE) i, “Mobil” i “Telefon” que podrien ser de tipus cadena de caràcters també.

3.1 Creació de lists

Per a construir un list tenim la següent funció constructora:
`list()`

Aquest constructor ens permet crear lists buits, lists per enumeració dels seus components i lists amb *tags* o noms per als seus components, com segueix:

1. list buit

```
> l <- list()
> l
list()
>
```

2. list per enumeració

```
> l <- list("Enric", "Canals",21,"H",1.77,FALSE
    ,66543211,"93 - 4444446")
> l
[[1]]
[1] "Enric"

[[2]]
[1] "Canals"

[[3]]
[1] 21

[[4]]
[1] "H"

[[5]]
[1] 1.77

[[6]]
[1] FALSE

[[7]]
[1] 66543211

[[8]]
[1] "93 - 4444446"
>
```

3. list amb noms per als components

```
> l <- list(Nom="Enric", Cognom="Canals",Edat=21,
    Sexe="H",Alcada=1.77,Treballa=FALSE,Mobil
    =66543211,Telefon="93 - 4444446")
> l
$Nom
[1] "Enric"

$Cognom
[1] "Canals"

$Edat
[1] 21

$Sexe
[1] "H"

$Alcada
[1] 1.77

$Treballa
[1] FALSE

$Mobil
[1] 66543211

$Telefon
[1] "93 - 4444446"
```

Addicionalment, podem consultar els noms dels components d'un list mitjançant la funció `names(l)` que donat un list `l` ens retorna un vector amb els noms dels seus components:

```
> names(l)
[1] "Nom"       "Cognom"    "Edat"      "Sexe"
"Alcada"    "Treballa" "Mobil"     "Telefon"
```

De vegades és necessari donar-li noms als components d'un list creat prèviament per enumeració. Això es pot fer assignant a `names(l)` el vector amb els noms que es vol:

```
> l<-list("Maria Perez","FIB",2011)
> l
[[1]]
[1] "Maria Perez"

[[2]]
[1] "FIB"

[[3]]
[1] 2011
```

```
11
12 #Fins aqui els components del list l no tenen
       noms. Ara li afegim els noms:
13
14 > names(l) <- c("Nom","Centre","Ingres")
15 > l
16 $Nom
17 [1] "Maria Perez"
18
19 $Centre
20 [1] "FIB"
21
22 $Ingres
23 [1] 2011
```

3.2 Accés als components d'un list

Existeixen diferents maneres d'accedir directament a cada component d'un list. En efecte, mitjaçant **operadors d'accés directe**, que tenen com a paràmetre o bé el nom d'un component o bé un índex que indica la posició del component que es desitja visitar, es pot accedir a qualsevol component del list de manera directa. Les especificacions d'aquests operadors són les següents:

1. $\$: list \times id \rightarrow T_{id}$

   Essent l un list, id el nom d'un component de l i T_{id} el tipus del component amb nom id. La crida a aquest operador té la següent sintaxi

   ```
   l$id
   ```

   ```
   1 > l<-list(Nom="Maria Perez",Centre="FIB",Ingres
         =2011)
   2 > l$Nom
   3 [1] "Maria Perez"
   4 > l$Centre
   5 [1] "FIB"
   6 > l$Ingres
   7 [1] 2011
   8 >
   ```

2. $[[\quad]] : list \times enter \rightarrow T$

   En aquest operador d'accés, els lists són tractats de manera semblant als vectors i fem servir la crida

   ```
   l[[exp]]
   ```

   per referir-nos al component del list l que es troba a la posició que resulti d'avaluar l'expressió entera exp. Cal notar, però, una diferència important

amb els vectors, i és que amb el list, l'operador és un doble gafet (`[[ ]]`) i cal parar atenció a aquesta diferència.

De la mateixa manera que passava amb els vectors, els accessos als components d'un list han de ser necessàriament posicions permeses en el list, i per tant, cal tenir la possibilitat de conèixer la dimensió d'un list. Per fer-ho disposem d'una funció especificada com segueix:

$$length : list \rightarrow enter$$

Així doncs, si un list l té N components, la crida

$$length(l)$$

ens tornarà N.

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres
    =2011)
> l
$Nom
[1] "Maria Perez"

$Centre
[1] "FIB"

$Ingres
[1] 2011

> length(l)
[1] 3
>
> l1 <-list()
> length(l1)
[1] 0
```

Ara podem demanar que

$$1 \leq avaluacio(exp) \leq length(l) \qquad (2)$$

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres
    =2011)
> l[[1]]
[1] "Maria Perez"
> l[[2]]
[1] "FIB"
> l[[3]]
[1] 2011
> l[[1+1]]
[1] "FIB"
> l[[2*1]]
[1] "FIB"
```

Com en el cas dels vectors, és molt important vigilar que es satisfaci la restricció (2). En cas que no la respectem tindrem un problema en intentar accedir a una posiciò il·legal dins del list. En aquests casos, l'R ens respòn con segueix

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres
    =2011)
> l[[4]]
Error en l[[4]] : subindice fuera de  los limites
>
```

S'ha de tenir en compte que existeix una bijecció del conjunt de noms d'un list l i la posició en que aquest nom apareix al list. Per exemple, al list l obtingut fent la instrucció

```
l<-list(Nom="Maria Perez",Centre="FIB",Ingres
    =2011):
```

tindríem la bijecció següent:

		posicio
Nom	$\rightarrow$	1
$Centre$	$\rightarrow$	2
$Ingres$	$\rightarrow$	3

3. Entre les dues opcions anteriors per accedir als components d'un list l, també existeix una altra manera d'indexar-lo amb la crida següent:

```
l[[quote(id)]]
```

on $quote(id)$ és el nom corresponent al id dins de doble cometes:

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres
    =2011)
> l[["Centre"]]
[1] "FIB"
>
```

Com a conseqüencia, tenim tres maneres d'accedir al mateix component:

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres
    =2011)
>
> l[["Centre"]]
[1] "FIB"
>
> l$Centre
[1] "FIB"
>
> l[[2]]
[1] "FIB"
>
```

3.3 Afegir i eliminar elements d'un list

Els lists poden créixer i decréixer dinàmicament. És a dir, podem afegir nous components a un list i eliminar qualsevol dels components que existeixin.

Per a afegir un nou element directament li assignem un valor al component i això ja crea el component:

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres=2011)
> l
$Nom
[1] "Maria Perez"

$Centre
[1] "FIB"

$Ingres
[1] 2011

# Afegim l'element "Universitat":

> l$Universitat <- "UPC"
> l
$Nom
[1] "Maria Perez"

$Centre
[1] "FIB"

$Ingres
[1] 2011

$Universitat
[1] "UPC"
>
```

Per a eliminar un element el que farem és assignar-li el valor `NULL` al component que volem eliminar:

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres=2011,
    Universitat="UPC")
> l
$Nom
[1] "Maria Perez"

$Centre
[1] "FIB"

$Ingres
[1] 2011

$Universitat
```

```
[1] "UPC"

# Eliminem l'element "Centre":

> l$Centre<-NULL
> l
$Nom
[1] "Maria Perez"

$Ingres
[1] 2011

$Universitat
[1] "UPC"
>
```

També es pot preguntar si un element existeix en un list o no mitjançant la funció *is.null*:

$$is.null : component\_del\_list \rightarrow bool$$

```
> is.null(l$Centre)
[1] TRUE

> is.null(l[["Ingres"]])
[1] FALSE
>
```

3.4 Accés a sublists

Per obtenir un sublist a partir d'un list, es fa servir la següent forma d'indexada:

$$[\ \ ] : list \times interval \rightarrow list$$

on *interval* és un interval de la forma $[exp]$ o $[exp1 : exp2]$. La crida es fa com segueix:

```
l[exp]
```

```
> l<-list(Nom="Maria Perez",Centre="FIB",Ingres =2011,
    Universitat="UPC")
> l[1]
$Nom
[1] "Maria Perez"
>
> l[[1]]
[1] "Maria Perez"
>
```

S'ha de notar la diferència de significat quan es fa servir gafet simple i quan es fa servir doble gafet. En el primer cas torna un list mentre que en el segon cas torna l'ièsim component.

També podem fer un sublist amb més d'un element del list original usant un subrang per a l'índex:

```
l[exp1:exp2]
```

```
> l[1:3]
$Nom
[1] "Maria Perez"

$Centre
[1] "FIB"

$Ingres
[1] 2011

> l[3:4]
$Ingres
[1] 2011

$Universitat
[1] "UPC"
>
> l[3:1]
$Ingres
[1] 2011

$Centre
[1] "FIB"

$Nom
[1] "Maria Perez"
>
> l[(2+1):(3+1)]
$Ingres
[1] 2011

$Universitat
[1] "UPC"
>
```

3.5 Recorreguts i cerques

Hi ha molts problemes on s'han de recórrer els components d'un list i també problemes on és necessari fer servir l'esquema de cerca. Aquestes dues famílies de problemes les hem caracteritzat al Capítol 1, quan parlàvem de vectors, a les seccions 1.3 i 1.4. Com a exemple, aquí podem veure una funció que escriu un list:

```
escriure_list <- function(l){
   n <- names(l)
   for (i in 1:length(l)){
      cat(n[i],": ",l[[i]],"\n")
   }
}
```

que fa un recorregut del list i que es pot utilitzar d'aquesta forma:

```
> l <- list(Nom="Joan",Cognom="Martinez",Edat=23,Grau=
   "Informatica",Universitat="UPC")
> escriure_list(l)
Nom : Joan
Cognom : Martinez
Edat : 23
Grau : Informatica
Universitat : UPC
>
```

Capítol 4

Data Frames

Un **data frame** *df* és un objecte **estructurat**, de dues dimensions, on les columnes (també anomenades *variables*) són dades d'un mateix tipus i les files (o *individus*) són un conjunt de dades de diferent tipus. Un data frame es pot veure com un list de vectors i també com una matriu no heterogènia. Vist com una matriu, un data frame és un objecte composat per un nombre finit de files F (entrades horitzontals) i un nombre finit de columnes C (entrades verticals) anomenats components o elements. De manera genèrica, podem considerar que un data frame *df* és de la forma:

$$
\begin{array}{c}
\\
F\ files
\end{array}
\overbrace{\begin{bmatrix}
df_{1,1} & df_{1,2} & \cdots & df_{1,C-1} & df_{1,C} \\
df_{2,1} & df_{2,2} & \cdots & df_{2,C-1} & df_{2,C} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
df_{F-1,1} & df_{F-1,2} & \cdots & df_{F-1,C-1} & df_{F-1,C} \\
df_{F,1} & df_{F,2} & \cdots & df_{F,C-1} & df_{F,C}
\end{bmatrix}}^{C\ columnes}
$$

Com podem veure, en aquest cas el data frame *df* té F files i C columnes i per tant té $F \times C$ components on es compleix

$$\forall i \in \{1, \ldots, F\}, \forall j \in \{1, \ldots, C\}, df_{i,j} \in T_j$$

Dit d'una altra manera, amb el data frame *df* hom pot associar la dimensió $F \times C$. En un data frame, cada fila i és un objecte de tipus *list* amb dades no necessàriament homogènies mentre que cada columna j és un objecte de tipus vector i, per tant, homogeni. És a dir, $df_{i,j}$ es un valor del tipus base T_j. Per fixar la idea, si considerem una enquesta, un data frame és el tipus d'objecte de l'R que permet representar les dades recollides mitjançant l'enquesta. Vist d'aquesta manera, cada columna correspon als valors d'una variable estadística (i possiblement el seu nom és a la capçalera) i cada fila correspon a una resposta donada per una persona enquestada.

Considerem per exemple el data frame següent:

$$df = \begin{bmatrix} Edat & Sexe & Alçada & Hobby \\ 18 & H & 1.81 & Fútbol \\ 20 & M & 1.67 & Lectura \\ 15 & M & 1.60 & Dança \\ 19 & H & 1.76 & Bowling \\ 23 & M & 1.70 & Música \\ 18 & H & 1.65 & Lectura \end{bmatrix}$$

Interpretant aquest exemple com les respostes a una enquesta, les variables estadístiques serien `Edat, Sexe, Alçada` i `Hobby` on els seus valors serien els que es troben a la columna que encapçala cadascuna. Addicionalment, cada fila seria una possible resposta a l'enquesta. Com hem dit abans, cada fila pot ser vista com un *list*, com ara ⟨`23, M, 1.70, Música`⟩ i cada columna com a un vector, per exemple el vector de seqüència de caràcters ⟨`"Fútbol", L·Lectura", "Dança", "Bowling", "Música", L·Lectura"`⟩.

4.1 Creació de data frames al llenguatge R

El llenguatge R ofereix diverses formes de crear data frames. Ací en veurem dos: la creació directa i la creació a partir de fitxers.

4.1.1 Creació directa

La creació directa d'un data frame utilitza el constructor

`data.frame()`

Aquest constructor rep com a paràmetres les columnes del data frame a crear. Com en el cas dels *list*, hom pot assignar un nom a cadascuna de les columnes (variables) del data frame. Per exemple podem construir un data frame fent:

```
data.frame(nomcol_1 = valcol_1, ..., nomcol_k = valcol_k, stringsAsFactors=FALSE)
```

Un exemple concret de construcció d'un data frames és

```
> df <- data.frame(Nom=c("Maria","Kim","Pep"),
                   Edat=c(18,20,23),
                   stringsAsFactors=FALSE)
> df
   Nom Edat
1  Maria   18
2    Kim   20
3    Pep   23
>
```

Però a diferència del *list*, en aquest cas també poden usar-se els noms nom de les variables d'R per a donar nom als camps del data frame. Així doncs podem fer el següent:

```
> Edat <- c(18,20,15,19,23,18)
> Sexe <- c("H","M","M","H","M","H")
> Alcada <- c(1.81,1.67,1.60,1.76,1.70,1.65)
```

```
> Hobby <- c("Futbol","Lectura","Danca","Bowling","
    Musica","Lectura")
> df <- data.frame(Edat,Sexe,Alcada,Hobby,
    stringsAsFactors=FALSE)
> df
  Edat Sexe Alcada   Hobby
1   18    H   1.81  Futbol
2   20    M   1.67 Lectura
3   15    M   1.60   Danca
4   19    H   1.76 Bowling
5   23    M   1.70  Musica
6   18    H   1.65 Lectura
>
```

Recordem que el paràmetre `stringsAsFactors` defineix la manera en la que R interpretarà les cadenes de caràcters. Si no es defineix de manera explícita el paràmetre `stringsAsFactors=FALSE`, el valor que pren per defecte és `TRUE` i això vol dir que les cadenes de caràcters seran tractades d'una forma diferent a la que estem acostumats. L'explicació en detall d'aquest paràmetre està fora de l'abast d'aquest document i per tant, sempre que calgui crear data frames amb columnes on hi hagi cadenes de caràcters s'inclourà l'assignació `stringsAsFactors=FALSE`.

També es poden crear data frames buits. Per exemple, si a l'intèrpret R introduïm l'assignació

```
> df1 <- data.frame()
```

i tot seguit demanem pel valor de la variable `df1`, s'obté

```
> df1
data frame with 0 columns and 0 rows
>
```

4.1.2 Creació de data frames a partir de fitxers

Generalment els data frames són objectes grans i interessarà construir-los a partir de les dades d'un fitxer CSV, format en el qual les dades estan usualment separades per un espai. En aquest cas, R ofereix un operador de lectura de fitxers `read.table` que s'utilitza com segueix:

```
read.table(quote(nomfitxer), header=TRUE)
```

El paràmetre `header = TRUE` defineix que la primera línia conté les capçaleres del data frame, és a dir, els noms de les variables. Si no s'explicita pren el valor per defecte `FALSE`. Per exemple, el codi R

```
> df <- read.table("mydata.txt", header=TRUE)
```

crea el data frame `df` llegint les dades del fitxer de nom `"mydata.txt"` amb capçaleres.

4.2 Operadors del tipus *list* i data frames

Ja hem explicat que un data frame es pot veure com un *list*, per tant se li podran aplicar tots els operadors estudiats al capítol anterior.

4.2.1 Accés a una columna

En tot el que segueix, la variable `df` denotarà un data frame. suposem el data frame de la Secció 4.1.1, la primera columna corresponent al nom `Edat` es pot seleccionar usant el valor enter de líndex

```
> df[[1]
[1] 18 20 15 19 23 18
```

El nom de de la columna amb l'operador $

```
> df$Edat
[1] 18 20 15 19 23 18
```

O bé emprant el nom com a índex

```
> df[["Hobby"]]
[1] "Futbol"  "Lectura" "Danca"   "Bowling" "Musica"
    "Lectura"
```

4.2.2 Obtenir la longitud

Podem demanar la longitud del data frame amb la funció `length()` i ens retornarà el nombre de columnes (variables) del data frame:

```
> length(df)
[1] 4
>
```

4.2.3 Noms

Podem accedir als noms de les variables:

names(): Retorna un vector amb els noms de les variables o columnes.

colnames(): Retorna un vector amb els noms de les variables. És equivalent a `names()`.

o també als noms de les files:

rownames(): Retorna un vector amb els noms de les files.

En tots els casos, igual que passa en les estructures *list*, també es poden assignar a aquestes funcions el vector de noms que hom vulgui definir. Exemples d'utilització d'aquestes operacions aplicades al data frame definit al llistat 4.1.1 són:

```
> names(df)
[1] "Edat"   "Sexe"   "Alcada" "Hobby"
> colnames(df)
[1] "Edat"   "Sexe"   "Alcada" "Hobby"
> rownames(df)
[1] "1" "2" "3" "4" "5" "6"
>
```

Considerem el data frame del llistat de la Secció 4.1.1. Si ara hom defineix el conjunt de noms de files amb

```
> rownames(df) <- c("Joan","Anna","Maria","Albert","
    Jana","Josep")
>
```

el data frame `df` resultant serà

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> length(df)
[1] 4
>
```

4.3 Operadors del tipus matriu i data frames

Com ha estat explicat més amunt, un data frame també es pot veure com una matriu, per tant també es podem aplicar els operadors definits per a les matriu i accedir als elements del data frame de forma matricial, és a dir, variables amb dos índexs. Recordem-ne algunes d'aquestes operacions.

nrow(): Retorna el nombre de files.

ncol(): Retorna el nombre de columnbes.

dim(): Retorna el nombre de files i columnes, és a dir, les dimensions de la matriu.

Considerem ara el data frame `df` del llistat de la Secció 4.2.3. Aleshores tindrem

```
> nrow(df)
[1] 6
> ncol(df)
[1] 4
> dim(df)
[1] 6 4
>
```

Important En particular, quan s'usen operadors d'indexat, cal assegurar-se'n que el valor resultant de l'avaluació de l'expressió de l'índex caigui dins d'un rang de valors [1..max] on max és el valor màxim de files o columnes del data frame. En general cal tenir molta cura amb el fet que les funcions s'apliquin sobre data frames que estiguin correctament definits. Un exemple concret de situació en la qual una funció s'aplica de manera errònia és el següent

```
> df2
data frame with 0 columns and 0 rows
> nrow(df2)
[1] 0
> ncol(df2)
[1] 0
> df[1,2]
NULL
```

4.3.1 Accés a una columna sencera

Recordem que cada columna d'un data frame és un vector, per tant, un objecte amb informació homogènia. Considerem un altre cop el data frame `df` del llistat de la Secció 4.2.3. Aleshores tindrem

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> df[,2]
[1] "H" "M" "M" "H" "M" "H"
> df[,3]
[1] 1.81 1.67 1.60 1.76 1.70 1.65
>
```

4.3.2 Accés a una fila sencera

Recodem que cada fila és un list per tant, un objecte heterogeni. Fixeu-vos que en fer la petició d'una única fila en realitat ens retorna un data frame amb només aquesta fila.

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> df[2,]
       Edat Sexe Alcada   Hobby
Anna     20    M   1.67 Lectura
> df[4,]
       Edat Sexe Alcada   Hobby
Albert   19    H   1.76 Bowling
```

4.3.3 Accés a una component d'un data frame

De manera anàloga al que passava en el càlcul matricial, tot sovint interessa tractar cada element d'un data frame de manera individual. Considerant el data frame `df` del llistat de la Secció 4.2.3 tindrem

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> df[1,3]
[1] 1.81
> df[3,4]
[1] "Danca"
> df[2,5]
NULL
> df[7,3]
NULL
>
```

Noteu que, quan es tracta d'accedir a elements del data frame que no estan definits, l'operador retorna la constant `NULL`, és a dir, un valor invàlid. Recordeu el que ha estat dit més amunt respecte d'haver d'usar amb cura els índexs.

També es pot accedir a un element determinat usant com a índex el nom de la columna

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> df[1,"Alcada"]
[1] 1.81
> df[3,"Hobby"]
[1] "Danca"
>
```

4.4 Subdata frames

De la mateixa manera que de les matrius es poden extreure submatrius, dels data frames es poden extreure subdata frames. Les diverses extracccions es poden fer per rang d'índex, amb filtres o selecció de files

4.4.1 Seleccció per rang d'índex

La seleccció de subdata frames per rang d'índex segueix la mateixa sintaxi que en el cas de l'extracció de submatrius

- L'absència del segon índex, `[i, ]` designa la fila `i`-èssima.
- L'absència del primer índex, `[, j]` designa la columna `j`-èssima.
- La presència d'un rang, `[min:max]` indica el conjunt de files o columnes tals que la primera és la corresponent a l'índex `min` i la darrera és la corresponent a l'índex `max`.

Vegem tot seguit alguns exemples.

```
> df
       Edat  Sexe  Alcada   Hobby
Joan     18     H   1.81  Futbol
Anna     20     M   1.67 Lectura
Maria    15     M   1.60   Danca
Albert   19     H   1.76 Bowling
Jana     23     M   1.70  Musica
Josep    18     H   1.65 Lectura
>
> df[2:4,]
       Edat  Sexe  Alcada   Hobby
Anna     20     M   1.67 Lectura
Maria    15     M   1.60   Danca
Albert   19     H   1.76 Bowling
>
> df[,1:3]
       Edat  Sexe  Alcada
Joan     18     H   1.81
Anna     20     M   1.67
Maria    15     M   1.60
Albert   19     H   1.76
Jana     23     M   1.70
Josep    18     H   1.65
>
> df[2:4,1:3]
       Edat  Sexe  Alcada
Anna     20     M   1.67
Maria    15     M   1.60
Albert    19     H   1.76
>
> df[2:4,3]
[1] 1.67 1.60 1.76
```

Noteu que el resultat de la darrera línea és un vector.

Com hem vist, l'accés a les dades d'una única columna del data frame retorna les dades de la columna en un vector. Si el que volem és que el resultat retornat sigui una columna del data frame expressada com un altre data frame cal afegir el paràmetre `drop = FALSE` com segueix:

```
> df[2:4,3, drop=FALSE]
       Alcada
Anna     1.67
Maria    1.60
Albert   1.76
```

4.4.2 Filtres

Moltes vegades el que es vol és obtenir un subdata frame segons alguna condició o propietat que es pot expressar com una expressió lògica repecte d'una columna. Si el data frame és `df`, la sentència seria:

```
df[condició(df$colnom, valor), rangcol]
```

per exemple

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> df[df$Alcada < 1.70,]
       Edat Sexe Alcada   Hobby
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Josep    18    H   1.65 Lectura
> df[df$Alcada > 1.70, 2:4]
      Sexe Alcada   Hobby
Joan     H   1.81  Futbol
Albert   H   1.76 Bowling
>
```

4.4.3 Selecció de files

Si hom vol seleccionar un subdata frame que contingui totes les columnes del data frame original però només inclogui les files que compleixin una determinada condició, el llenguatge R ofereix la funció `subset`. La sintaxi és

```
subset(df, cond(colnoms, valors))
```

Exemples d'utilització són

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
>
```

```
> subset(df, Sexe == "H")
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Albert   19    H   1.76 Bowling
Josep    18    H   1.65 Lectura
> subset(df, Sexe == "H" & Alcada >= 1.65)
      Edat Sexe Alcada   Hobby
Joan    18    H   1.81  Futbol
Albert  19    H   1.76 Bowling
Josep   18    H   1.65 Lectura
>
```

4.5 Extensió de data frames

Els data frame es poden extendre afegint tant files com columnes usant els operadors específics del llenguatge R.

4.5.1 Afegir una fila

Afegir una fila a un data frame significa afegir un individu, és a dir un valor concret per a cadascuna de les variables. La funció corresponent és

```
rbind()
```

Per a poder afegir un individu (una fila) al data frame, com que les dades a afegir són heterogènies, possiblement de diferents tipus, cal afegir un altre *data frame* o un *list*. Aquest nou element cal que tingui necessàriament el mateix nombre de columnes que el data frame que es vol augmentar. Un exemple que il.lustra aquesta funció és:

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> rbind(df, data.frame(Edat=22,Sexe="M",Alcada=1.70,
    Hobby="Tenis",
                       stringsAsFactors=FALSE))
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
7        22    M   1.70   Tenis
>
```

Fixeu-vos que, afegint data frame, cal tenir cura de definir l'opció `stringsAsFactors=FALSE` si algun dels camps del data frame és una cadena de caràcters.

Afegir una nova fila a un data frame expressada com un *list* s'aconsegueix amb la sintaxi

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
> NovaFila <- list(Edat=22, Sexe="M", Alcada=1.70,
    Hobby="Tenis")
> rbind(df, NovaFila)
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
7        22    M   1.70   Tenis
>
```

També es poden afegir diverses files, és a dir, es pot afegir un data frame a un altre data frame.

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
>
> df1 <- data.frame(Edat=c(17,21,15),Sexe=c("H","M","H
    "),
                          Alcada=c(1.70,1.71,1.65),
                             Hobby=c("Bowling","Danca",
                             "Musica"),
                          stringsAsFactors=FALSE)
>
> df1
  Edat Sexe Alcada   Hobby
1   17    H   1.70 Bowling
2   21    M   1.71   Danca
3   15    H   1.65  Musica
>
```

```
> rbind(df, df1)
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
7        17    H   1.70 Bowling
8        21    M   1.71   Danca
9        15    H   1.65  Musica
>
```

4.5.2 Afegir una nova columna

Afegir una columna a un data frame significa afegir una variable, és a dir, afegir un nou valor a tots i cadascun dels individus del data frame. La funció és

```
cbind()
```

Per a poder afegir una variable o columna al data frame, les dades poden estar emmagatzemades directament en un vector, perquè són homogènies. L'única restricció que cal tenir en compte és que el nombre de components del vector ha de ser igual al nombre de files del data frame.

```
> Ciutat <- c("Paris","Barcelona","Barcelona","Roma","
    Caracas","Barcelona","Paris")
> df <- rbind(df, list(22,"M",1.70,"Tenis"))
> df <- cbind(df, Ciutat)
>
> df
       Edat Sexe Alcada   Hobby    Ciutat
Joan     18    H   1.81  Futbol     Paris
Anna     20    M   1.67 Lectura Barcelona
Maria    15    M   1.60   Danca Barcelona
Albert   19    H   1.76 Bowling      Roma
Jana     23    M   1.70  Musica   Caracas
Josep    18    H   1.65 Lectura Barcelona
7        22    M   1.70   Tenis     Paris
>
```

També es pot afegir una columna al data frame tal i com es feia en el cas del *list*, és a dir, aplicant directament l'operador "$":

```
> df
       Edat Sexe Alcada   Hobby
Joan     18    H   1.81  Futbol
Anna     20    M   1.67 Lectura
Maria    15    M   1.60   Danca
Albert   19    H   1.76 Bowling
Jana     23    M   1.70  Musica
Josep    18    H   1.65 Lectura
```

```
> df$CodiPostal <- c
    (08027,08003,08014,08034,08034,08006)
> df
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H   1.81  Futbol       8027
Anna     20    M   1.67 Lectura       8003
Maria    15    M   1.60   Danca       8014
Albert   19    H   1.76 Bowling       8034
Jana     23    M   1.70  Musica       8034
Josep    18    H   1.65 Lectura       8006
>
```

Notis que, a diferència de la funció `cbind()`, afegir directament columnes modifica el data frame.

4.6 Modificació de data frames

Hi ha alguns problemes on cal modificar parcialment els data frames, és a dir, cal modificar els valors d'algunes variables. En aquest cas, la funció de l'R que permet fer aquestes modificacions és

`transform()`

A continuació podem veure alguns exemples d'ús d'aquesta funció.

```
> df
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H   1.81  Futbol       8027
Anna     20    M   1.67 Lectura       8003
Maria    15    M   1.60   Danca       8014
Albert   19    H   1.76 Bowling       8034
Jana     23    M   1.70  Musica       8034
Josep    18    H   1.65 Lectura       8006
> transform(df, Alcada=Alcada*100)
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H    181  Futbol       8027
Anna     20    M    167 Lectura       8003
Maria    15    M    160   Danca       8014
Albert   19    H    176 Bowling       8034
Jana     23    M    170  Musica       8034
Josep    18    H    165 Lectura       8006
> transform(df,Alcada=Alcada%/%2.54) #passem cm a
    polzades
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H  71.26  Futbol       8027
Anna     20    M  65.75 Lectura       8003
Maria    15    M  62.99   Danca       8014
Albert   19    H  69.29 Bowling       8034
Jana     23    M  66.93  Musica       8034
Josep    18    H  64.96 Lectura       8006
> transform(df,Alcada=Alcada%/%12,CodiPostal=
    CodiPostal%%100) #passem polzades a peus, reduim cp
```

```
        Edat Sexe Alcada   Hobby CodiPostal
Joan      18    H   5.94  Futbol         27
Anna      20    M   5.48 Lectura          3
Maria     15    M   5.25   Danca         14
Albert    19    H   5.77 Bowling         34
Jana      23    M   5.58  Musica         34
Josep     18    H   5.41 Lectura          6
>
```

4.7 Ordenació de data frames

Els data frames poden ser ordenats segons una o diverses columnes. La funció que cal utilitzar és

```
order()
```

L'ordenació es fa de manera creixent o decreixent en funció del paràmetre `decreasing` que pren valor cert o fals. Per defecte pren valor `FALSE` i l'ordenació és fa segons valors creixents. Per exemple, hom pot ordenar el data frame `df` de més amunt segons la variable `Edat` i en ordre creixent com segueix

```
> df
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H   1.81  Futbol        8027
Anna     20    M   1.67 Lectura        8003
Maria    15    M   1.60   Danca        8014
Albert   19    H   1.76 Bowling        8034
Jana     23    M   1.70  Musica        8034
Josep    18    H   1.65 Lectura        8006
>
> df[order(df$Edat),]
       Edat Sexe Alcada   Hobby CodiPostal
Maria    15    M   1.60   Danca        8014
Joan     18    H   1.81  Futbol        8027
Josep    18    H   1.65 Lectura        8006
Albert   19    H   1.76 Bowling        8034
Anna     20    M   1.67 Lectura        8003
Jana     23    M   1.70  Musica        8034
>
```

Notis la coma que apareix al darrere de la columna seleccionada `$Edat),`. L'ordenació decreixent seria

```
> df
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H   1.81  Futbol        8027
Anna     20    M   1.67 Lectura        8003
Maria    15    M   1.60   Danca        8014
Albert   19    H   1.76 Bowling        8034
Jana     23    M   1.70  Musica        8034
Josep    18    H   1.65 Lectura        8006
>
```

```
> df[order(df$Edat, decreasing=TRUE),]
       Edat Sexe Alcada   Hobby CodiPostal
Jana     23    M   1.70  Musica       8034
Anna     20    M   1.67 Lectura       8003
Albert   19    H   1.76 Bowling       8034
Joan     18    H   1.81  Futbol       8027
Josep    18    H   1.65 Lectura       8006
Maria    15    M   1.60   Danca       8014
>
```

Una ordenació creixent simultàniament segons `Edat` i `Alcada` seria

```
> df
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H   1.81  Futbol       8027
Anna     20    M   1.67 Lectura       8003
Maria    15    M   1.60   Danca       8014
Albert   19    H   1.76 Bowling       8034
Jana     23    M   1.70  Musica       8034
Josep    18    H   1.65 Lectura       8006
> df[order(df$Edat, df$Alcada),]
       Edat Sexe Alcada   Hobby CodiPostal
Maria    15    M   1.60   Danca       8014
Josep    18    H   1.65 Lectura       8006
Joan     18    H   1.81  Futbol       8027
Albert   19    H   1.76 Bowling       8034
Anna     20    M   1.67 Lectura       8003
Jana     23    M   1.70  Musica       8034
>
```

També podríem afegir, alhora, una selecció per columnes i mostrar només un subrang d'elles, com per exemple:

```
> df
       Edat Sexe Alcada   Hobby CodiPostal
Joan     18    H   1.81  Futbol       8027
Anna     20    M   1.67 Lectura       8003
Maria    15    M   1.60   Danca       8014
Albert   19    H   1.76 Bowling       8034
Jana     23    M   1.70  Musica       8034
Josep    18    H   1.65 Lectura       8006
> df[order(df$Edat, df$Alcada), 2:4]
       Sexe Alcada   Hobby
Maria     M   1.60   Danca
Josep     H   1.65 Lectura
Joan      H   1.81  Futbol
Albert    H   1.76 Bowling
Anna      M   1.67 Lectura
Jana      M   1.70  Musica
>
```

4.8 Fusió de data frames

El llenguatge R permet construir data frames a partir de la fusió de dos data frames donats. La fusió es fa a partir d'una variable (columna) que els data frames tinguin en comú. El data frame resultant estarà compost per

1. La unió de les columnes dels dos data frames. Notis que això vol dir que no es repeteixen les que tinguin en comú.

2. Les files dels dos data frames tals que tenen el mateix índex de fila i el valor de la columna que tenen en comú els data frames són iguals.

La sintaxi de la funció és

```
merge(df1, df2)
```

Un exemple de data frame resultant de la fusió dels data frames `df1` i `df2` és

```
> df1
  Var1 Var2 Var3
1   A1   B4   C1
2   A2   B2   C2
3   A3   B1   C3
4   A4   B2   C1
5   A2   B7   C3
6   A4   B3   C4
7   A5   B6   C3
8   A6   B1   C3
>
> df2
  Var2 Var4
1   B1   D1
2   B2   D2
3   B3   D1
4   B4   D3
5   B5   D5
>
>
> merge(df1, df2)
>
  Var2 Var1 Var3 Var4
1   B1   A3   C3   D1
2   B1   A6   C3   D1
3   B2   A2   C2   D2
4   B2   A4   C1   D2
5   B3   A4   C4   D1
6   B4   A1   C1   D3
>
```

També podem fer servir el paràmetre `all` per indicar que el *merge* inclogui totes les files malgrat que hi hagi valors de la columna compartida que no siguin iguals. Per exemple

```
> df1
  Var1 Var2 Var3
1   A1   B4   C1
2   A2   B2   C2
3   A3   B1   C3
4   A4   B2   C1
5   A2   B7   C3
6   A4   B3   C4
7   A5   B6   C3
8   A6   B1   C3
>
> df2
>
  Var2 Var4
1   B1   D1
2   B2   D2
3   B3   D1
4   B4   D3
5   B5   D5
>
> merge(df1, df2, all = TRUE)
>
  Var2 Var1 Var3 Var4
1   B1   A3   C3   D1
2   B1   A6   C3   D1
3   B2   A2   C2   D2
4   B2   A4   C1   D2
5   B3   A4   C4   D1
6   B4   A1   C1   D3
7   B5 <NA> <NA>   D5
8   B6   A5   C3 <NA>
9   B7   A2   C3 <NA>
>
```

Notis que quan a les files on hi ha valors no compartits en la columna `Var2` prenen per valor `<NA>`.

Quan hi ha columnes (variables) que tenen valors comuns a dos data frames però els seus noms no coincideixin es pot aplicar l'operador `merge` fent servir els paràmetres `by.x` i `by.y` per indicar el nom de la variable a considerar tan a l'operador `x` com a l'operador `y`.

```
> df1
  Var1 Var2 Var3
1   A1   B4   C1
2   A2   B2   C2
3   A3   B1   C3
4   A4   B2   C1
5   A2   B7   C3
6   A4   B3   C4
7   A5   B6   C3
8   A6   B1   C3
```

```
> df2
>
  Dif2 Var4
1   B1   D1
2   B2   D2
3   B3   D1
4   B4   D3
5   B5   D5
>
> merge(df1, df2, by.x = "Var2", by.y = "Dif2")
>
  Var2 Var1 Var3 Var4
1   B1   A3   C3   D1
2   B1   A6   C3   D1
3   B2   A2   C2   D2
4   B2   A4   C1   D2
5   B3   A4   C4   D1
6   B4   A1   C1   D3
>
```

Bibliografia

[1] Wirth, Niklaus.: Algorithms and data structures. London et al.: Prentice-Hall, 1986.