

Manipulation de données avec R

par

Odile Wolber

1. Les objets

R manipule des objets. Ainsi, lorsque l'on importe un fichier dans R, on obtient dans R un objet nommé data frame. Les variables, les données, les fonctions, les résultats d'analyses sont stockés dans des objets. Il existe plusieurs types d'objets : vecteurs, facteurs, etc. Les principaux objets sont présentés dans cette partie.

Les objets sont caractérisés par leur nom, leur contenu et des attributs qui vont spécifier le type de données représenté par l'objet.

Le nom d'un objet doit commencer par une lettre et peut comporter des lettres, des chiffres, des points (.) et des espaces soulignés (_). R distingue, pour les noms des objets, les majuscules des minuscules, c'est-à-dire que x et X nommeront des objets distincts.

Les objets ont tous au moins deux attributs : le mode et la longueur. Le mode est le type des éléments d'un objet. Il en existe quatre principaux : numérique, caractère, complexe, et logique (FALSE ou TRUE). Pour connaître le mode et la longueur d'un objet, on utilise respectivement les fonctions mode() et length().

1.1. Objets basiques

L'objet le plus basique est une constante, qui peut être numérique, complexe, caractère, ou logique.

On affecte directement une valeur à un objet. L'objet n'a pas besoin d'être déclaré.

Par exemple, on saisit sur la console :

```
n = 8.
```

On tape ensuite n pour afficher sa valeur. On obtient le résultat suivant :

```
[1] 8
```

Le symbole [1] indique que l'affichage commence au premier élément de n.

On aurait également pu affecter une valeur à l'objet n en utilisant le signe <- (signe moins accolé à un crochet) ou -> :

```
n <- 8
```

```
n
```

```
[1] 8
```

```
8 -> n
```

```
n
```

```
[1] 8
```

Dans la suite du document, on utilisera le symbole «=».

Quelques exemples d'objets basiques :

```
x = 1
```

```
x
```

```
[1] 1
```

Si on affecte une valeur à un objet existant, sa valeur précédente est effacée :

```
x = 10
```

```
x
```

```
[1] 10
```

```
y = 10 + 2
y
[1] 12
```

On utilise «;» pour séparer des commandes distinctes sur la même ligne :

```
w = 8; name = "Wikistat"; dicton = "Aide-toi, le ciel t'aidera";
w ; name ; dicton
[1] 8
[1] "Wikistat"
[1] "Aide-toi, le ciel t'aidera"
```

1.2. Autres objets

1.1.1. Les objets et leurs attributs

On crée des objets en utilisant l'opérateur = ou -> ou <-. Le mode et le type de l'objet ainsi créé sont généralement déterminés de façon implicite. Il est possible de créer un objet en précisant son mode, sa longueur, son type, etc. On peut, par exemple, créer un vecteur 'vide' puis modifier successivement ses éléments, ce qui est beaucoup plus efficace que de rassembler ces éléments avec c(). On utilisera alors l'indexation. On peut aussi créer des objets à partir d'autres objets.

Le tableau suivant indique les modes possibles pour les objets vecteur, facteur, array, matrice, data.frame, ts et liste (tableau tiré du document R pour les débutants d'Emmanuel Paradis).

Objet	Modes	Plusieurs modes possibles dans le même objet
vecteur	num, car, comp, log	Non
facteur	num, car	Non
array	num, car, comp, log	Non
matrice	num, car, comp, log	Non
data.frame	num, car, comp, log	Oui
ts	num, car, comp, log	Oui
liste	num, car, comp, log, fonction, expression	Oui

num = numérique, car = caractère, comp = complexe, log = logique

Nous n'aborderons pas les array (tableaux à k dimensions, la matrice au cas particulier où k=2), ni les ts (données de type séries temporelles).

Les valeurs manquantes sont représentées par NA (Not Available) quel que soit l'objet ou le mode.

1.1.2. Vecteurs

La fonction `vector()` a deux arguments : le mode des éléments qui compose le vecteur et la longueur du vecteur.

```
a = vector("numeric", 5)
b = vector("character", 5)
c = vector("logical", 5)
d = vector("complex", 5)
```

équivalent à

```
a = numeric(5)
b = character(5)
c = logical(5)
d = complex(5)
```

Si on tape sur la console `a;b;c;d`, le résultat suivant s'affiche :

```
[1] 0 0 0 0 0
[1] "" "" "" "" ""
[1] FALSE FALSE FALSE FALSE FALSE
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
```

On peut également construire un vecteur à l'aide de la fonction `c()`
`vecteur = c(5, 7.2, 3.6, 4.9); vecteur`
`[1] 5.0 7.2 3.6 4.9`

1.1.3. Facteurs

La fonction `factor()` crée des variables qualitatives nominales.

```
factor( x,  
       levels = sort(unique(x), na.last = TRUE),  
       labels = levels,  
       exclude = NA, ordered = is.ordered(x))
```

`levels` spécifie quels sont les niveaux possibles du facteur (par défaut les valeurs uniques du vecteur `x`), i.e. les valeurs que peuvent prendre les éléments du facteur.

`labels` définit les noms des niveaux

`exclude` les valeurs de `x` à ne pas inclure dans les niveaux

`ordered` argument logique spécifiant si les niveaux du facteur sont ordonnés.

```
factor(1:3)  
[1] 1 2 3  
Levels: 1 2 3
```

```
factor(1:3, levels = 1:5)  
[1] 1 2 3  
Levels: 1 2 3 4 5
```

```
factor(1:3, levels = 1:5, labels = c("A", "B", "C", "D", "E"))  
[1] A B C  
Levels: A B C D E
```

```
factor(c(5,7,8,9), levels=5 :8)  
[1] 5 7 8 <NA>  
Levels : 5 6 7 8
```

9 ne fait pas partie des levels (ensemble de définition). Le chiffre 9 est donc codé comme valeur manquante.

Exemple : 10 personnes ont répondu à une enquête de satisfaction auto-administrée. Le degré de satisfaction est recueilli à partir d'une variable comportant les 4 modalités suivantes :

1 – très satisfait

2 – satisfait

3 – insatisfait

4 – pas du tout satisfait

Voici les réponses des répondants : 1 1 2 3 1 0 4 2 3 4

Définissez un facteur dans R pour coder ces réponses.

```
factor(c(1,1,2,3,1,0,4,2,3,4),levels=1:4)  
[1] 1 1 2 3 1 <NA> 4 2 3 4  
Levels: 1 2 3 4
```

La réponse 0 ne fait pas partie du domaine de définition. Aussi, elle est codée comme valeur manquante `<NA>`.

1.1.4. Matrices

Une matrice est un vecteur qui possède un argument supplémentaire qui définit les dimensions de la matrice. Tous les éléments d'une matrice doivent être de même mode.

```
matrix( data = NA, nrow = 1, ncol = 1,  
       byrow = FALSE,  
       dimnames = NULL)
```

byrow indique si les valeurs données par data doivent remplir successivement les colonnes (FALSE, par défaut) ou les lignes (si TRUE).

dimnames permet de donner des noms aux lignes et colonnes. On peut aussi donner des noms aux colonnes ou aux lignes de la matrice grâce aux fonctions `rownames()` et `colnames()`.

```
matrix(0, 5, 7)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  0    0    0    0    0    0    0
[2,]  0    0    0    0    0    0    0
[3,]  0    0    0    0    0    0    0
[4,]  0    0    0    0    0    0    0
[5,]  0    0    0    0    0    0    0
```

```
x = 1:20
```

```
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12
[13] 13 14 15 16 17 18 19 20
```

```
mat1 = matrix(x, 4, 5)
```

```
> mat1
      [,1] [,2] [,3] [,4] [,5]
[1,]  1    5    9   13   17
[2,]  2    6   10   14   18
[3,]  3    7   11   15   19
[4,]  4    8   12   16   20
```

```
mat2 = matrix(x, 4, 5, byrow = TRUE)
```

```
> mat2
      [,1] [,2] [,3] [,4] [,5]
[1,]  1    2    3    4    5
[2,]  6    7    8    9   10
[3,] 11   12   13   14   15
[4,] 16   17   18   19   20
```

La fonction `paste()` est utile dans une situation où on souhaite nommer les lignes et / ou les colonnes d'une matrice. En effet, la fonction `paste()` permet de concaténer des objets.

```
nom_var = paste("V", 1:5, sep = "")
```

```
> nom_var
[1] "V1" "V2" "V3" "V4" "V5"
```

```
nom_ind = paste("I", 1:4, sep = "")
```

```
> nom_ind
[1] "I1" "I2" "I3" "I4"
```

```
colnames(mat2) = nom_var
```

```
rownames(mat2) = nom_ind
```

```
(ou bien dimnames(mat2) = list(nom_ind, nom_var))
```

```
mat2
```

```
> mat2
      V1  V2  V3  V4  V5
I1    1   2   3   4   5
I2    6   7   8   9  10
I3   11  12  13  14  15
I4   16  17  18  19  20
```

1.1.5. Data frames

Un tableau de données est créé de façon implicite par la fonction `read.table`. On peut également créer un tableau de données avec la fonction `data.frame`.

Les éléments d'une `data.frame` ne doivent pas nécessairement avoir le même mode. Les éléments de mode caractères sont considérés comme des facteurs.

Tous les éléments de la `data.frame` doivent être de la même longueur. Dans le cas contraire, l'élément le plus court est « recyclé » un nombre entier de fois.

Pour créer une `data.frame` à partir d'un vecteur numérique et d'un vecteur caractère, on procède de la manière suivante :

```
a = c(1, 2, 3) ; a ; mode(a)
[1] 1 2 3
[1] "numeric"

b = c("a", "b", "c") ; b ; mode(b)
[1] "a" "b" "c"
[1] "character"

df = data.frame(a, b)
> df
```

	a	b
1	1	a
2	2	b
3	3	c

Pour créer avec une seule instruction une `data.frame` avec une variable numérique et une variable caractère :

```
df2 = data.frame(a = 1:6, b = letters[1:6]);
> df2
```

	a	b
1	1	a
2	2	b
3	3	c
4	4	d
5	5	e
6	6	f

Si on juxtapose un vecteur de longueur 6 et un vecteur de longueur 3, le deuxième vecteur est dupliqué :

```
a = c(1, 2, 3, 4, 5, 6)
b = c("a", "b", "c")
df = data.frame(a, b)
>df
```

	a	b
1	1	a
2	2	b
3	3	c
4	4	d
5	5	e
6	6	f

Il faut donc que la longueur de l'un des vecteurs soit un multiple de la longueur de l'autre vecteur :

```
a = c(1, 2, 3, 4, 5)
b = c("a", "b", "c")
df = data.frame(a, b)
Error in data.frame(a, b) : arguments imply differing number of rows: 5, 3
```

1.1.6. Listes

Une liste se crée de la même manière qu'une data.frame. Tout comme les data.frame, les éléments qui la compose ne sont pas nécessairement du même mode. Ils ne sont pas nécessairement de la même longueur, *ainsi que l'illustre l'exemple suivant* :

```
a = c(1, 2, 3, 4, 5)
b = c("a", "b", "c")
liste1 = list(a, b)
> liste1
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "a" "b" "c"
```

On peut nommer les éléments d'une liste de la manière suivante :

```
names(liste1) = c("L1", "L2"); liste1

$L1
[1] 1 2 3 4 5

$L2
[1] "a" "b" "c"

liste2 = list(L1 = a, L2 = b)
liste2
$L1
[1] 1 2 3 4 5

$L2
[1] "a" "b" "c"
```

2. Conversions

2.1. Conversion de modes

Dans de nombreuses situations pratiques, il est utile de convertir le mode d'un objet en un autre. Une telle conversion sera possible grâce à une fonction de la forme : as.mode (as.numeric, as.logical, as.character, ...).

conversion en	fonction	règles
numérique	as.numeric	FALSE → 0 TRUE → 1 "1", "2", ... → 1, 2, ... "A", ... → NA
logique	as.logical	0 → FALSE autres nombres → TRUE "FALSE" → FALSE "TRUE" → TRUE autres caractères → NA
caractère	as.character	1, 2, ... → "1", "2", ... FALSE → "FALSE" TRUE → "TRUE"

(tableau tiré du document R pour les débutants d'Emmanuel Paradis)

Considérons quelques exemples simples :

Premier cas : On souhaite convertir des objets de mode logical en mode numeric

```
logique = c(FALSE, FALSE, TRUE, TRUE, FALSE, TRUE)
conversion_numerique = as.numeric(logique)
conversion_numerique
[1] 0 0 1 1 0 1
```

Deuxième cas : On souhaite convertir des objets de mode character en mode numeric

```
caractere = c("1", "2", "3", "A", "/", "T", "%", "-")
conversion_numerique = as.numeric(caractere)
Warning message:
NAs introduced by coercion
conversion_numerique
[1] 1 2 3 NA NA NA NA NA
```

Troisième cas : On souhaite convertir des objets de mode numeric en mode logical

```
numerique = 0:5
conversion_logique1 = as.logical(numerique)
conversion_logique1
[1] FALSE TRUE TRUE TRUE TRUE TRUE
```

Quatrième cas : On souhaite convertir des objets de mode character en mode logical

```
caractere = c("FALSE", "TRUE", "F", "T", "false", "t", "A", "(")
conversion_logique2 = as.logical(caractere)
conversion_logique2
[1] FALSE TRUE FALSE TRUE FALSE NA NA NA
```

Cinquième cas : On souhaite convertir des objets de mode numeric en mode character

```
numerique = 1:8
conversion_caractere1 = as.character(numerique)
conversion_caractere1
[1] "1" "2" "3" "4" "5" "6" "7" "8"
```

Sixième cas : On souhaite convertir des objets de mode logical en mode character

```
logique = c(TRUE, FALSE)
conversion_caractere2 = as.character(logique)
conversion_caractere2
[1] "TRUE" "FALSE"
```

2.2. Conversion d'objets

Dans de nombreuses situations pratiques, il est utile de convertir un objet en un autre. Une telle conversion sera possible grâce à une fonction de la forme : `as.objet` (`as.matrix`, `as.data.frame`, `as.list`, `as.factor`, ...).

On souhaite convertir la matrice a en une data.frame b :

```
> a = matrix(1:25, nrow = 5, ncol = 5);a
      [,1] [,2] [,3] [,4] [,5]
[1,]  1   6  11  16  21
[2,]  2   7  12  17  22
[3,]  3   8  13  18  23
[4,]  4   9  14  19  24
[5,]  5  10  15  20  25
```

```
b = as.data.frame(a);b;
  V1  V2  V3  V4  V5
1   1   6  11  16  21
2   2   7  12  17  22
3   3   8  13  18  23
4   4   9  14  19  24
5   5  10  15  20  25
```

La conversion d'un facteur en numérique se fait en deux étapes. En effet, R convertit avec le codage numérique des niveaux du facteur.

Exemples :

```
facteur = factor(c(1, 5, 10));facteur
[1] 1 5 10
Levels: 1 5 10
```

```
facteur_numerique1 = as.numeric(facteur);facteur_numerique1;
[1] 1 2 3
```

Sur l'exemple suivant, le facteur comporte les modalités Male (level 2) et Female (level 1). La conversion en numérique donne donc le facteur 2 1.

```
fac2 = factor(c("Male", "Female"))
fac2
[1] Male Female
Levels: Female Male
as.numeric(fac2)
[1] 2 1
```

Pour conserver un facteur (en numérique) en conservant les niveaux tels qu'ils sont spécifiés, on convertira d'abord en caractère puis en numérique.

```
facteur_caractere = as.character(facteur);facteur_caractere
[1] "1" "5" "10"
facteur_numerique = as.numeric(facteur_caractere);facteur_numerique
[1] 1 5 10
```

3. Accéder aux données

3.1. Lister les objets en mémoire

Supposons qu'on ait les objets suivants en mémoire :

```
x = 10 ; X = 5
w = 8
name = "Wikistat"
dicton = "Aide-toi, le ciel t'aidera"
y = 10 + 2
```

La fonction `ls()` permet d'afficher la liste des objets en mémoire (seul le nom des objets est affiché) :

```
ls()
[1] "X" "dicton" "name" "w" "x" "y"
```

Si on souhaite lister uniquement les objets contenant un caractère donné dans leur nom, on utilisera l'option `pat` (abréviation de `pattern`) :

```
ls(pat = "n")
[1] "dicton" "name"
```

Pour restreindre la liste aux objets qui commence par un caractère donné on utilisera l'option « ^ » :

```
ls(pat = "^n")  
[1] "name"
```

Par défaut, ls.str affiche les détails de tous les objets contenus en mémoire :

```
> ls.str()  
dicton : chr "Aide-toi, le ciel t'aidera"  
name : chr "Wikistat"  
w : num 8  
x : num 10  
X : num 5  
y : num 12
```

Une option utile de ls.str est max.level qui spécifie le niveau de détails de l'affichage des objets composites. On évite d'afficher tous les détails avec l'option max.level = -1 :

```
print(ls.str(), max.level = -1)
```

Exemple : on veut imprimer à l'écran le dictionnaire des variables de la data.frame paysniv3. On utilise l'instruction suivante :

```
ls.str(pattern="paysniv3")
```

On obtient le listing suivant (tronqué) :

```
paysniv3 : `data.frame`: 173 obs. of 25 variables:  
$ NOM : Factor w/ 173 levels "AFGANISTHAN",...: 3 47 88 102 146 157 15 22 27 36 ...  
$ POP87: num 23.5 51.9 3.8 24.4 23.5 ...  
$ NAT : num 42 37 39 36 45 32 51 48 35 46 ...  
...  
$ CONTI: num 1 1 1 1 1 1 1 1 1 1 ...
```

On a non seulement la liste des variables et leurs attributs, mais également les valeurs prises par ces variables pour les premières observations.

3.2. Accéder par le système d'indexation

L'indexation est un moyen efficace et flexible d'accéder de façon sélective aux éléments d'un objet. L'indexation se fait à l'aide de crochets [], les parenthèses étant réservés pour les arguments d'une fonction.

Les vecteurs

Soit le vecteur x suivant :

```
x = 1:20 ; x  
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Pour accéder au cinquième élément du vecteur x, il suffit de taper la commande suivante :

```
x[5]  
[1] 5
```

Pour les vecteurs, matrices et tableaux, il est possible d'accéder aux valeurs de ces éléments à l'aide d'une expression de comparaison :

```
x = c(1.1, 5.3, 9, 4.2, 3.6, 7.2, 8.4, 1.6, 8.8, 3.5);x  
[1] 1.1 5.3 9.0 4.2 3.6 7.2 8.4 1.6 8.8 3.5
```

Si on saisit $x < 5$ sur la console, les valeurs suivantes s'affichent :

```
[1] TRUE FALSE FALSE TRUE TRUE FALSE FALSE TRUE FALSE TRUE
```

En effet, $1.1 < 5$ (l'expression $x < 5$ est vérifiée, donc TRUE), $5.3 > 5$ (FALSE)...

Le vecteur y ne comprend que les éléments de x qui sont inférieurs à 5 :

```
y = x[x < 5];y  
[1] 1.1 4.2 3.6 1.6 3.5
```

Le vecteur z comprend le 2^{ème} et le 6^{ème} éléments du vecteur x :

```
z = x[c(2, 6)];z  
[1] 5.3 7.2
```

On accède aux éléments supérieurs à 10 du vecteur x par la commande suivante :

```
x = 1 :20  
x[x > 10]  
[1] 11 12 13 14 15 16 17 18 19 20
```

On remplace les éléments de x supérieurs à 10 par la valeur 20 par la commande suivante :

```
x[x > 10] = 20 ; x  
[1] 1 2 3 4 5 6 7 8 9 10 20 20 20 20 20 20 20 20 20 20
```

On remplace les éléments égaux à 20 par la valeur 0 par la commande suivante :

```
x[x==20] = 0; x  
[1] 1 2 3 4 5 6 7 8 9 10 0 0 0 0 0 0 0 0 0 0
```

Notons l'usage du « == » qui n'est pas une affectation mais un test d'égalité.

Les matrices

Soit la matrice a :

```
a = matrix(1:25, 5, 5);  
a  
      [,1] [,2] [,3] [,4] [,5]  
[1,]  1   6  11  16  21  
[2,]  2   7  12  17  22  
[3,]  3   8  13  18  23  
[4,]  4   9  14  19  24  
[5,]  5  10  15  20  25
```

On accède à la valeur de la ième ligne et jème colonne par x[i,j]. On peut accéder aux valeurs de la jème colonne de la matrice X par la commande X[, j]. De la même manière, on accède aux valeurs de la ième ligne de la matrice X par la commande X[i,].

On accède à l'élément de la ligne 2, colonne 3 par la commande a[2,3]

```
a[2, 3] = 2; a  
a  
      [,1] [,2] [,3] [,4] [,5]  
[1,]  1   6  11  16  21  
[2,]  2   7   2  17  22  
[3,]  3   8  13  18  23  
[4,]  4   9  14  19  24  
[5,]  5  10  15  20  25
```

On souhaite accéder aux valeurs de la troisième colonne de la matrice a

```
a[, 3]  
[1] 11 2 12 14 15
```

Le dernier résultat est un vecteur et non une matrice. Par défaut, R retourne un objet de la plus petite dimension possible. Ceci peut être modifié avec l'option drop dont le défaut est TRUE :

```
> a[, 3, drop = FALSE]  
      [,1]  
[1,]  11  
[2,]   2  
[3,]  13
```

```
[4,] 14
[5,] 15
```

On peut également affecter des valeurs à des lignes ou colonnes de matrice.

```
a[, 3] = 3; a
a
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6    3   16   21
[2,]    2    7    3   17   22
[3,]    3    8    3   18   23
[4,]    4    9    3   19   24
[5,]    5   10    3   20   25
```

```
a[3, ] = 3; a
a
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6    3   16   21
[2,]    2    7    3   17   22
[3,]    3    3    3    3    3
[4,]    4    9    3   19   24
[5,]    5   10    3   20   25
```

```
a[, 3] = 11:15; a
a
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    3   13    3    3
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

Le système d'indexation sert aussi à supprimer des lignes ou colonnes. On peut supprimer la $j^{\text{ème}}$ colonne de la matrice X par la commande X[, -j]. De la même manière, on peut supprimer la $i^{\text{ème}}$ ligne de la matrice X par la commande X[-i,].

On supprimer la troisième ligne de la matrice a avec la commande suivante :

```
a[-3, ];
```

```
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

Une utilisation pratique de cette indexation est, par exemple, la possibilité de sélectionner les éléments pairs d'une variable entière :

```
x = 1:10
```

Avec la commande x%%2, on teste si x est divisible par 2 et on obtient le résultat suivant :

```
[1] 1 0 1 0 1 0 1 0 1 0
```

Si maintenant on teste que x est divisible par 2, on obtient une suite d'objets logiques :

```
x%%2 == 0
```

```
[1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE
```

La commande suivante sélectionne uniquement les éléments du vecteur pour lesquels (x%%2==0)=FALSE :

```
xpair = x[x%%2 == 0];xpair
```

```
[1] 2 4 6 8 10
```

Ce système d'indexation utilise donc des valeurs logiques retournées par des opérateurs de comparaison. Ces valeurs logiques peuvent être calculées au préalable, elles seront éventuellement recyclées.

```
temp = c(FALSE, TRUE)
x[temp]
[1] 2 4 6 8 10
c(FALSE, TRUE) prend les éléments pairs
c(TRUE, FALSE) prend les éléments impairs.
```

Les data.frames

L'indexation peut-être également utilisée avec les data.frames, mais avec la difficulté que les différentes colonnes du data.frame peuvent-être de mode différents.

Soit la data.frame construite de la manière suivante :

```
a = c(1, 2, 3)
b = c("a", "b", "c")
mat1=matrix(13:18,nr=3,nc=2)
df = data.frame(a, b,mat1)
```

```
      a  b X1 X2
[1,] 1  a 13 16
[2,] 2  b 14 17
[3,] 3  c 15 18
```

En tapant la commande `df[2]`, on obtient la deuxième colonne `b` de la data.frame :

```
df[2]
      b
[1,]  a
[2,]  b
[3,]  c
```

Pour obtenir la matrice `mat1`, on tape la commande :

```
df[3 :4]
      X1 X2
[1,] 13 16
[2,] 14 17
[3,] 15 18
```

Pour accéder au 3^{ème} élément de la 2^{ème} colonne de `df`, on utilise la commande suivante :

```
df[3,2]
[1] c
Levels: a b c
```

On obtient le même résultat avec les deux commandes suivantes :

```
df[2][3, 1]
df[2][3, ]
```

Remarquons une nouvelle fois que, par défaut, le mode `character` dans une `data.frame` est considéré comme un mode `factor`.

On accède à la troisième ligne de la matrice `mat1` avec la commande suivante :

```
df[3:4][3, ]
      X1 X2
[3,] 15 18
```

On souhaite assigner la valeur 1 à la deuxième colonne de `df` :

```
df[2]= 1;df
      a  b X1 X2
[1,] 1  1 13 16
[2,] 2  1 14 17
[3,] 3  1 15 18
```

On assigne la valeur 999 à la 3^{ème} ligne de la 2^{ème} colonne :

```
df[3,2] = 999;df
```

```
      a    b  X1  X2
[1,]  1    1  13  16
[2,]  2    1  14  17
[3,]  3  999  15  18
```

On assigne la valeur 999 aux 3^{èmes} lignes des colonnes 2, 3 et 4 :

```
df[2:4][3, ] = 999;df
```

```
      a    b  X1  X2
[1,]  1    1  13  16
[2,]  2    1  14  17
[3,]  3  999  999  999
```

Les listes

Pour les listes, l'accès aux différents éléments de la liste se fait avec des crochets doubles. Par exemple, pour accéder aux troisième éléments de la liste nommé L, il suffira de taper la commande L[[3]]. Le résultat pourra ensuite être indexé de la même façon que pour les vecteurs, les matrices.

Exemple :

```
liste = list(a = 1:6, b = matrix(1:24, 6, 4));liste
```

```
liste[[1]]
```

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

```
liste[[2]]
```

```
      [,1] [,2] [,3] [,4]
[1,]   1   7  13  19
[2,]   2   8  14  20
[3,]   3   9  15  21
[4,]   4  10  16  22
[5,]   5  11  17  23
[6,]   6  12  18  24
```

```
liste[[1]][c(TRUE, FALSE)]
```

```
[1] 1 3 5
```

```
liste[[2]][ ,3]
```

```
[1] 13 14 15 16 17 18
```

3.3. Accéder par le nom

On ne peut accéder aux valeurs d'éléments par le nom que dans le cadre de data.frame ou de liste. L'appel se fait par le symbole \$.

```
liste = list(a = 1:6, b = matrix(1:24, 6, 4));liste
```

```
liste$a
```

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

```
liste$b
```

```
      [,1] [,2] [,3] [,4]
[1,]   1   7  13  19
[2,]   2   8  14  20
[3,]   3   9  15  21
[4,]   4  10  16  22
[5,]   5  11  17  23
[6,]   6  12  18  24
```

```
liste$b[1, ]
```

```
[1] 1 7 13 19
```

```
data_frame = data.frame(a = 1:6, b = letters[1:3])
data_frame$a
[1] 1 2 3 4 5 6
data_frame$b
[1] a b c a b c
Levels: a b c
```

3.4. Accéder par l'éditeur de données

Il est possible d'utiliser un éditeur graphique de style tableur pour éditer un objet contenant des données grâce à la fonction `data.entry()`.

```
matrice = matrix(1:9, 3, 3)
data.entry(matrice)
```

La fenêtre suivante s'ouvre :



Par exemple, on remplace le 4 par un 0 et on ferme l'éditeur de données..

Si on tape ensuite la commande `matrice` dans la fenêtre de commande, on obtient le résultat suivant :

```
matrice
      var1 var2 var3
[1,]    1    0    7
[2,]    2    5    8
[3,]    3    6    9
```

4. Opérateurs et fonctions

4.1. Les opérateurs

Il y a trois principaux types d'opérateurs dans R :

Arithmétique	Comparaison	Logique
+ addition	< inférieur à	! x NON logique
- soustraction	> supérieur à	x & y ET logique
* multiplication	<= inférieur ou égal à	x && y idem
/ division	>= supérieur ou égal à	x y OU logique
^ puissance	== égal	x y idem
%% modulo	!= différent	xor(x, y) OU exclusif
%/% division entière		

Les opérateurs < ET > et < OU > existent sous deux formes : la forme simple opère sur chaque élément des objets et retourne autant de valeurs logiques que de comparaisons effectuées ; la forme double opère sur le premier élément des objets.

On utilisera l'opérateur < ET > pour spécifier une inégalité du type $0 < x < 1$ qui sera codée :

```
0 < x & x < 1.
```

L'expression $0 < x < 1$ est valide mais ne donnera pas le résultat escompté : les deux opérateurs de cette expression étant identiques, ils seront exécutés successivement de la gauche vers la droite. L'opération $0 < x$ sera d'abord réalisée retournant une valeur logique qui sera ensuite comparée à 1 ($TRUE$ ou $FALSE < 1$) : dans ce cas la valeur logique sera convertie implicitement en numérique (1 ou $0 < 1$).

```
> x <- 0.5
> 0 < x < 1
[1] FALSE
```

Les opérateurs de comparaison opèrent sur chaque élément des deux objets qui sont comparés, et retournent donc un objet de même taille. Pour effectuer une comparaison « globale » de deux objets, deux fonctions sont disponibles : [identical](#) et [all.equal](#).

Exemples :

```
X=data.frame(Z)
identical(Z,X)
[1] FALSE
```

Dans l'exemple précédent, X et Z contiennent les mêmes données, mais sont des objets différents. Aussi, la comparaison par la fonction identical donne le résultat FALSE.

```
x=TRUE
y=!x
y
[1] FALSE
```

```
x=c(TRUE,FALSE,FALSE,TRUE,TRUE)
y=c(FALSE,FALSE,TRUE,TRUE,FALSE)
z=x&y
z
[1] FALSE FALSE FALSE TRUE FALSE
u=x&&y
u
[1] FALSE
t=x | y
t
[1] TRUE FALSE TRUE TRUE TRUE
s=x||y
> s
[1] TRUE
```

4.2. Calculs arithmétiques et fonctions simples

Les fonctions disponibles dans R pour les manipulations de données sont nombreuses. Nous n'en citons ci-dessous que quelques-unes.

<code>sum(x)</code>	somme des éléments de x
<code>prod(x)</code>	produit des éléments de x
<code>max(x)</code>	maximum des éléments de x
<code>min(x)</code>	minimum des éléments de x
<code>which.max(x)</code>	retourne l'indice du maximum des éléments de x
<code>which.min(x)</code>	retourne l'indice du minimum des éléments de x
<code>range(x)</code>	idem que <code>c(min(x), max(x))</code>
<code>length(x)</code>	nombre d'éléments dans x
<code>mean(x)</code>	moyenne des éléments de x
<code>median(x)</code>	médiane des éléments de x
<code>var(x) ou cov(x)</code>	variance des éléments de x (calculée sur n - 1) ; si x est une matrice ou un tableau de données, la matrice de variance-covariance est calculée

<code>cor(x)</code>	matrice de corrélation si <code>x</code> est une matrice ou un tableau de données (1 si <code>x</code> est un vecteur)
<code>var(x, y)</code> ou <code>cov(x, y)</code>	covariance entre <code>x</code> et <code>y</code> , ou entre les colonnes de <code>x</code> et de <code>y</code> si ce sont les matrices ou des tableaux de données
<code>cor(x, y)</code>	corrélation linéaire entre <code>x</code> et <code>y</code> , ou matrice de corrélations si ce sont des matrices ou des tableaux de données

Ces fonctions retournent une valeur simple (donc un vecteur de longueur 1), sauf `range` qui retourne un vecteur de longueur 2, et `var`, `cov` et `cor` qui peuvent retourner une matrice.

Les fonctions suivantes retournent des résultats plus complexes.

<code>round(x, n)</code>	arrondit les éléments de <code>x</code> à <code>n</code> chiffres après la virgule
<code>rev(x)</code>	inverse l'ordre des éléments de <code>x</code>
<code>sort(x)</code>	trie les éléments de <code>x</code> dans l'ordre ascendant ; pour trier dans l'ordre descendant : <code>rev(sort(x))</code>
<code>rank(x)</code>	rangs des éléments de <code>x</code>
<code>log(x, base)</code>	calcule le logarithme à base <code>base</code> de <code>x</code>
<code>scale(x)</code>	si <code>x</code> est une matrice, centre et réduit les données ; pour centrer uniquement ajouter l'option <code>center=FALSE</code> , pour réduire uniquement <code>scale=FALSE</code> (par défaut <code>center=TRUE</code> , <code>scale=TRUE</code>)
<code>pmin(x,y,...)</code>	un vecteur dont le <i>i</i> ème élément est le minimum entre <code>x[i]</code> , <code>y[i]</code> , . . .
<code>pmax(x,y,...)</code>	idem pour le maximum
<code>cumsum(x)</code>	un vecteur dont le <i>i</i> ème élément est la somme de <code>x[1]</code> _a <code>x[i]</code>
<code>cumprod(x)</code>	idem pour le produit
<code>cummin(x)</code>	idem pour le minimum
<code>cummax(x)</code>	idem pour le maximum
<code>match(x, y)</code>	retourne un vecteur de même longueur que <code>x</code> contenant les éléments de <code>x</code> qui sont dans <code>y</code> (NA sinon)
<code>which(x == a)</code>	retourne un vecteur des indices de <code>x</code> pour lesquels l'opération de comparaison est vraie (TRUE), dans cet exemple les valeurs de <i>i</i> telles que <code>x[i] == a</code> (l'argument de cette fonction doit être une variable de mode logique)
<code>choose(n, k)</code>	calcule les combinaisons de <code>k</code> événements parmi <code>n</code> répétitions = $n! / [(n - k)!k!]$
<code>na.omit(x)</code>	supprime les observations avec données manquantes (NA) (supprime la ligne correspondante si <code>x</code> est une matrice ou un tableau de données)
<code>na.fail(x)</code>	retourne un message d'erreur si <code>x</code> contient au moins un NA
<code>unique(x)</code>	si <code>x</code> est un vecteur ou un tableau de données, retourne un objet similaire mais avec les éléments dupliqués supprimés
<code>table(x)</code>	retourne un tableau des effectifs des différentes valeurs de <code>x</code> (typiquement pour des entiers ou des facteurs)
<code>table(x, y)</code>	tableau de contingence de <code>x</code> et <code>y</code>
<code>subset(x, ...)</code>	retourne une sélection de <code>x</code> en fonction de critères (..., typiquement des comparaisons : <code>x\$V1 < 10</code>) ; si <code>x</code> est un tableau de données, l'option <code>select</code> permet de préciser les variables à sélectionner (ou à éliminer à l'aide du signe moins)
<code>sample(x, size)</code>	rééchantillonne aléatoirement et sans remise <code>size</code> éléments dans le vecteur <code>x</code> , pour rééchantillonner avec remise on ajoute l'option <code>replace = TRUE</code>

Exemples : on a importé dans R la table SAS `paysniv3` à partir des commandes suivantes (cf. R01 sur l'import de tables SAS) :

```
sashome <- "C:/Program Files/SAS/SAS 9.1"
paysniv3=read.ssd(file.path(sashome,"fic_R"),"paysniv3", sascmd = file.path (sashome, "sas.exe"))
```

Ce fichier comporte, pour 173 pays, des données démographiques, une variable de superficie, le PNB et des variables explicites sur les ressources du pays.

Pour afficher le contenu

Calcul de la corrélation entre la population totale en 1987 (POP87) et la projection de population en 2000 :

```
cor(paysniv3$POP87,paysniv3$POP00)
[1] 0.997568
```

Calcul de la corrélation entre les variables NAT (taux de mortalité), MORT (taux de mortalité) et ACCR (taux d'accroissement naturel) :

- on crée d'abord une matrice X qui contient toutes les variables numériques de paysniv3 (colonnes 2 à 22) :

```
X=as.matrix(paysniv3[2:22])
```

- on crée ensuite une matrice Y qui contient les colonnes 2 (variable NAT), 3 (variable MORT) et 4 (variable ACCR) de la matrice X :

```
Y=cbind(X[,2],X[,3],X[,4])
```

On peut afficher directement cor(Y)

```
      [,1]      [,2]      [,3]
[1,] 1.0000000 0.6003699 0.9092156
[2,] 0.6003699 1.0000000 0.2148088
[3,] 0.9092156 0.2148088 1.0000000
```

ou bien affecter le résultat à un objet Z=cor(Y).

4.3. Opérations sur les matrices et les vecteurs

R offre des facilités pour le calcul et la manipulation de matrices. Le paragraphe suivant illustre des situations souvent rencontrées.

Soient les matrices mat1 et mat2 :

```
mat1 = matrix(1 : 4, nrow = 2, ncol = 2)
mat1
```

```
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
mat2= matrix(5 : 8, nr = 2, nc = 2 )
mat2
```

```
      [,1] [,2]
[1,]    5    7
[2,]    6    8
```

La fonction rbind() empile les matrices

```
rbind(mat1, mat2)
```

```
      [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    7
[4,]    6    8
```

La fonction cbind() juxtapose des matrices en conservant les colonnes

```
cbind(mat1, mat2)
```

```
      [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

L'opérateur pour le produit terme à terme de matrices est « * » tandis que l'opérateur pour le produit de deux matrices est « %*% ». Les opérateurs pour l'addition et la soustraction de matrices terme à terme sont respectivement « + » et « - ».

```

rbind(mat1, mat2) %*% cbind(mat1, mat2)
      [,1] [,2] [,3] [,4]
[1,]   7  15  23  31
[2,]  10  22  34  46
[3,]  19  43  67  91
[4,]  22  50  78 106

```

```

cbind(mat1, mat2) %*% rbind(mat1, mat2)
      [,1] [,2]
[1,]   74 106
[2,]   88 128

```

La transposition d'une matrice se fait avec la fonction `t()`.

```

t(rbind(mat1, mat2) %*% cbind(mat1, mat2))
      [,1] [,2] [,3] [,4]
[1,]   7  10  19  22
[2,]  15  22  43  50
[3,]  23  34  67  78
[4,]  31  46  91 106

```

La fonction `diag()` sert à extraire et - ou modifier la diagonale d'une matrice. Elle permet également de construire des matrices diagonales.

```

diag(mat1)
[1] 1 4

```

```

diag(rbind(mat1, mat2) %*% cbind(mat1, mat2))
[1] 7 22 67 106

```

```

diag(4)
      [,1] [,2] [,3] [,4]
[1,]   1   0   0   0
[2,]   0   1   0   0
[3,]   0   0   1   0
[4,]   0   0   0   1

```

```

v = c(1, 2, 3, 4)

```

```

diag(v)
      [,1] [,2] [,3] [,4]
[1,]   1   0   0   0
[2,]   0   2   0   0
[3,]   0   0   3   0
[4,]   0   0   0   4

```

On utilise `solve()` pour l'inversion de matrice, `qr()` pour la décomposition, `eigen()` pour le calcul des valeurs propres et `svd()` pour la décomposition en valeurs singulières.

```

a = solve(diag(v)); a

```

```

      [,1] [,2] [,3] [,4]
[1,]   1   0   0   0
[2,]   0 0.5   0   0
[3,]   0   0 0.33333 0
[4,]   0   0   0 0.25

```

```
eigen(diag(v))
$values
[1] 4 3 2 1
$vectors
      [,1] [,2] [,3] [,4]
[1,]  0   0   0   1
[2,]  0   0   1   0
[3,]  0   1   0   0
[4,]  1   0   0   0
```

5. Effacer les objets

La fonction `rm()` permet d'effacer des objets de la mémoire ; ces options sont proches de celles de `ls()`

```
ls()
[1] "dicton" "name" "w" "x" "y" "z"
```

Pour effacer l'objet nommé `x` de la mémoire, on tape la commande suivante :

```
rm(x);ls()
[1] "dicton" "name" "w" "y" "z"
```

Pour effacer tous les objets dont le nom commence par « `n` » :

```
> rm(list = ls(pat = "^n")); ls()
[1] "dicton" "w" "z" "y"
```

Pour effacer tous les objets de la mémoire :

```
> rm(list = ls()); ls()
character(0)
```

6. Sauvegarder les données

Pour enregistrer des objets, cette fois de n'importe quel type, on utilisera la commande

```
save(x, y, z, file="xyz.RData")
```

Pour faciliter l'échange de fichiers entre machines et systèmes d'exploitation, on peut utiliser l'option `ascii=TRUE`. Les données peuvent ultérieurement être chargées en mémoire avec `load("xyz.RData")`.

La fonction `save.image` est un raccourci pour `save(list=ls(all=TRUE),file=".RData")`.

Exemple :

Pour sauvegarder la matrice `Z` et la `data.frame` `paysniv3`, on utilise les commandes suivantes :

```
save(Z,file="Z.RData")
save(paysniv3,file="paysniv3.RData")
```

Les fichiers sont sauvegardés dans le répertoire `R-2.2.1` (à l'emplacement où on a installé le logiciel).

À la prochaine ouverture d'une session `R`, on peut récupérer les objets sauvegardés par la commande `load(file="objet.RData")`. Par exemple, la commande suivante permet de récupérer la `data.frame` `paysniv3` :

```
load(file="paysniv3.RData")
```

7. Exporter les données

La fonction `write.table` écrit dans un fichier un objet (tableau de données, vecteur, matrice, ..) :

```
write.table( x,  
            file = "",  
            append = FALSE,  
            quote = TRUE,  
            sep = " ",  
            eol = "\n",  
            na = "NA",  
            dec = ".",  
            row.names = TRUE,  
            col.names = TRUE,  
            qmethod = c("escape", "double"))
```

<code>x</code>	le nom de l'objet à écrire
<code>file</code>	le nom du fichier (par défaut l'objet est affiché à l'écran)
<code>append</code>	si TRUE ajoute les données sans effacer celles éventuellement existantes dans le fichier
<code>quote</code>	une variable logique ou un vecteur numérique : si TRUE les variables de mode caractère et les facteurs sont écrits entre "", sinon le vecteur indique les numéros des variables à écrire entre "" (dans les deux cas les noms des variables sont écrits entre "" mais pas si quote = FALSE)
<code>sep</code>	le séparateur de champ dans le fichier
<code>eol</code>	le caractère imprimé à la fin de chaque ligne ("\n" correspond à un retour chariot)
<code>na</code>	caractère utilisé pour les données manquantes
<code>dec</code>	caractère utilisé pour les décimales
<code>row.names</code>	variable logique indiquant si les noms des lignes doivent être écrits dans le fichier
<code>col.names</code>	variable logique indiquant si les noms des colonnes doivent être écrits dans le fichier
<code>qmethod</code>	spécifie, si quote=TRUE, comment sont traitées les guillemets doubles " incluses dans les variables de mode caractère : si "escape" (ou "e"), chaque " est remplacée par \", si "d" chaque " est remplacée par ""

Exemple :

La commande suivante permet d'exporter en fichier texte la `data.frame` `paysniv3` :

```
write.table(paysniv3,'d:/paysniv3.txt')
```