

# Opérateurs et fonctions dans R

par  
Odile Wolber

Les manipulations présentées dans ce polycopié peuvent être reproduites à partir de fichiers présents dans le document « R03 – données ». Une fois dézippés, placez les fichiers dans le répertoire "C:/Program Files/R/R-2.2.1", puis chargez les en mémoire par l'instruction `load("file.RData")`.

## 1. 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
```

## 2. 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 :

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

```
X=data.frame(Z)
identical(Z,X)
[1] FALSE
x=TRUE
y=lx
y
[1] FALSE
```

Dans l'exemple suivant, la comparaison s'effectue pour chacun des éléments pour z et t, et seulement sur le premier élément pour u et s :

```
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
```

### 3. Les fonctions simples

Nous distinguerons les fonctions statistiques des autres fonctions.

Pour les fonctions statistiques, il est nécessaire d'effectuer préalablement un traitement des données manquantes. Sinon si la variable traitée est manquante pour certaines observations, la valeur retournée par la fonction sera NA.

#### 3.1. Statistiques descriptives / Variables numériques

##### 3.1.1. Fonctions pour obtenir plusieurs statistiques univariées : *summary*, *statdes*

###### La fonction *summary*

Pour obtenir les statistiques descriptives usuelles, on peut commencer par utiliser la fonction `summary()`. On obtient la moyenne, la médiane, le 1<sup>er</sup> et le 3<sup>ème</sup> quartile, le minimum et le maximum.

**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.

On veut obtenir des statistiques descriptives basiques pour la variable `NAT` :

```
summary(paysniv3$NAT)
  Min. 1st Qu. Median Mean 3rd Qu. Max.
 10.00 20.00 33.00 32.02 44.00 53.00
```

On peut également stocker ces valeurs dans un objet R que l'on pourra ainsi sauvegarder et/ou exporter :

```
stat_des=summary(paysniv3$NAT)
On peut obtenir la description de l'objet stat_des avec la commande ls.str() :
ls.str(pattern="stat_des")
stat_des : Class 'table' Named num [1:6] 10 20 33 32 44 ...
```

On peut obtenir aussi ces statistiques descriptives sur l'ensemble des variables de la `data.frame` `paysniv3`, simplement avec la commande :

```
summary(paysniv3)
```

On obtient pour les variables numériques les précédentes statistiques, et pour les variables nominales les fréquences.

###### La fonction *stat.des*

La fonction `stat.des` est disponible dans le package `pastecs`. Pour y accéder, on charge donc préalablement ce package à l'aide de l'instruction `library(pastecs)`.

```
stat.desc(x, basic=TRUE, desc=TRUE, norm=FALSE, p=0.95)
```

`x` une `data.frame`

`basic` si `TRUE`, retourne les statistiques de base suivantes : nombre de valeurs (`nbr.val`), nombre d'observations égales à 0 (`nbr.null`), nombre de valeurs manquantes (`nbr.na`), minimum (`min`), maximum (`max`), différence entre maximum et minimum (`range`), somme des valeurs non manquantes (`sum`).

**desc** si TRUE, retourne les statistiques descriptives suivantes : médiane (median), moyenne (mean), écart-type de la moyenne (SE.mean), intervalle de confiance de la moyenne (CI.mean) pour le niveau p, variance (var), écart-type (std.dev), coefficient de variation (coef.var).

**norm** si TRUE, retourne les statistiques de normalité suivantes : coefficient de symétrie g1 (skewness), son critère de significativité (skew.2SE, i.e. g1/2.SEG1; si skew.2SE > 1, alors le coefficient de symétrie est significativement différent de zero), le coefficient de concentration g2 (kurtosis), son critère de significativité (kurt.2SE), la statistique d'un test de normalité de Shapiro-Wilk (normtest.W) et sa probabilité associée (normtest.p)

**p** niveau de probabilité pour calculer l'intervalle de confiance de la moyenne (CI.mean). Par défaut, p=0.95

**Exemples :**

*L'instruction suivante permet d'obtenir les précédentes statistiques descriptives sur la variable NAT de la data.frame paysniv3 :*

`stat.desc(paysniv3$NAT, basic=TRUE, desc=TRUE, norm=TRUE,p=0.95)`

nbr.val	nbr.null	nbr.na	min	max
1.730000e+02	0.000000e+00	0.000000e+00	1.000000e+01	5.300000e+01
range	sum	median	mean	SE.mean
4.300000e+01	5.540000e+03	3.300000e+01	3.202312e+01	9.763624e-01
CI.mean.0.95	var	std.dev	coef.var	skewness
1.927195e+00	1.649181e+02	1.284204e+01	4.010241e-01	-1.411638e-01
skew.2SE	kurtosis	kurt.2SE	normtest.W	normtest.p
-3.822605e-01	-1.308505e+00	-1.781465e+00	9.357610e-01	5.456007e-07

*L'instruction suivante permet d'obtenir ces statistiques pour plusieurs variables en même temps :*

`stat.desc(paysniv3[2:5], basic=TRUE, desc=TRUE, norm=TRUE,p=0.95)`

	POP87	NAT	MORT	ACCR
nbr.val	1.730000e+02	1.730000e+02	1.730000e+02	1.730000e+02
nbr.null	0.000000e+00	0.000000e+00	0.000000e+00	4.000000e+00
nbr.na	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
min	5.000000e-02	1.000000e+01	3.000000e+00	-2.000000e-01
max	1.062000e+03	5.300000e+01	2.900000e+01	4.000000e+00
range	1.061950e+03	4.300000e+01	2.600000e+01	4.200000e+00
sum	5.025250e+03	5.540000e+03	1.940000e+03	3.606000e+02
median	6.400000e+00	3.300000e+01	1.000000e+01	2.400000e+00
mean	2.904769e+01	3.202312e+01	1.121387e+01	2.084393e+00
SE.mean	8.043848e+00	9.763624e-01	4.106185e-01	7.981273e-02
CI.mean.0.95	1.587737e+01	1.927195e+00	8.105002e-01	1.575385e-01
var	1.119370e+04	1.649181e+02	2.916911e+01	1.102022e+00
std.dev	1.058003e+02	1.284204e+01	5.400843e+00	1.049773e+00
coef.var	3.642297e+00	4.010241e-01	4.816216e-01	5.036346e-01
skewness	7.772360e+00	-1.411638e-01	8.910332e-01	-5.104930e-01
skew.2SE	2.104694e+01	-3.822605e-01	2.412848e+00	-1.382375e+00
kurtosis	6.617603e+01	-1.308505e+00	1.385944e-01	-7.159104e-01
kurt.2SE	9.009536e+01	-1.781465e+00	1.886894e-01	-9.746763e-01
normtest.W	2.459745e-01	9.357610e-01	9.202851e-01	9.481337e-01
normtest.p	4.353533e-26	5.456007e-07	3.993889e-08	5.777623e-06

### 3.1.2. Fonctions pour obtenir une seule statistique univariée

On peut obtenir la moyenne, la médiane, le minimum, le maximum et les quantiles séparément à partir des fonctions :

<code>max(x)</code>	maximum des éléments de x
<code>min(x)</code>	minimum des éléments de x
<code>mean(x)</code>	moyenne des éléments de x
<code>median(x)</code>	médiane des éléments de x
<code>quantile(x)/fivenum(x)</code>	quantiles de x

**Exemple :**

```
quantile(paysniv3$NAT)
0% 25% 50% 75% 100%
10 20 33 44 53
```

```
fivenum(paysniv3$NAT)
[1] 10 20 33 44 53
```

Il existe de nombreuses autres fonctions pour obtenir des statistiques descriptives. 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>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>	min et max des éléments de x
<code>length(x)</code>	nombre d'éléments dans x
<code>sd(x)</code>	écart-type de x
<code>var(x)</code>	variance des éléments de x (calculée avec le dénominateur n - 1)
<code>IQR(x)</code>	écart inter-quartile

Ces fonctions retournent une valeur simple (donc un vecteur de longueur 1), sauf `range` qui retourne un vecteur de longueur 2.

**Exemples :**

*Ecart inter-quartile de NAT (3<sup>ème</sup> quartile=44, 1<sup>er</sup> quartile=20) :*

```
IQR(paysniv3$NAT)
[1] 24
```

*Différence entre le maximum et le minimum de NAT (maximum=53, minimum=10) :*

```
diff(range(paysniv3$NAT))
[1] 43
```

*Ecart-type :*

```
sd(paysniv3$NAT)
[1] 12.84204
```

*Affichage simultané de ces trois statistiques par la commande :*

```
c(IQR(paysniv3$MORT),diff(range(paysniv3$NAT)),sd(paysniv3$NAT))
[1] 8.00000 43.00000 12.84204
```

*Stockage des valeurs dans un objet R que l'on pourra ainsi sauvegarder ou exporter :*

```
X=c(IQR(paysniv3$MORT),diff(range(paysniv3$NAT)),sd(paysniv3$NAT))
X
[1] 8.00000 43.00000 12.84204
```

### 3.1.3. Fonctions pour obtenir des statistiques univariées par groupe : *tapply*, *aggregate*

#### Fonction *tapply*

La fonction *tapply* permet d'appliquer une fonction à des sous-populations. On peut ainsi appliquer une des précédentes fonctions à plusieurs groupes d'observations définis à partir d'une variable de la *data.frame*.

`tapply(x,y,fonction)`

**x** variables sur lesquelles on effectue des statistiques  
**y** variable définissant les sous-groupes d'observations  
**fonction** fonction que l'on souhaite appliquer

#### Exemples :

La commande suivante permet d'obtenir les statistiques descriptives de la fonction *summary* sur la variable *NAT*, pour chacun des continents :

```
tapply(paysniv3$NAT,paysniv3$CONT,summary)
```

\$"1" (Afrique)

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
19.00	41.00	46.00	43.62	48.00	53.00

\$"2" (Asie)

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
12.00	29.75	34.00	34.41	44.25	53.00

\$"3" (Amérique)

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
15.00	21.00	27.00	27.28	33.00	43.00

\$"4" (Europe)

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
10.00	12.00	13.00	13.93	16.00	26.00

\$"5" (Océanie)

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
16.00	24.00	28.00	28.89	36.00	42.00

La commande suivante permet d'obtenir les quartiles de la variable *NAT*, pour chacun des continents :

```
tapply(paysniv3$NAT,paysniv3$CONT,quantile)
```

\$"1"

0%	25%	50%	75%	100%
19	41	46	48	53

\$"2"

0%	25%	50%	75%	100%
12.00	29.75	34.00	44.25	53.00

\$"3"

0%	25%	50%	75%	100%
15	21	27	33	43

\$"4"

0%	25%	50%	75%	100%
10	12	13	16	26

\$"5"

0%	25%	50%	75%	100%
16	24	28	36	42

## Fonction aggregate

La fonction aggregate permet d'effectuer des statistiques sur des sous-populations définies par le croisement de plusieurs variables.

`aggregate(x,list(y,z,...),FUN=)`

`x` variable sur laquelle on effectue des statistiques  
`list(y,z,...)` variables définissant les sous-populations  
`FUN=` fonction que l'on souhaite appliquer

### Exemples :

La commande suivante retourne la moyenne par continent (variable CONT) des taux de mortalité (variable NAT) :

`aggregate(paysniv3$NAT,list(paysniv3$CONT),FUN=mean)`

	Group.1	x
1	1	43.62264
2	2	34.40909
3	3	27.28205
4	4	13.92857
5	5	28.88889

*FUN* doit toujours renvoyer un scalaire. On ne peut pas effectuer plusieurs statistiques comme avec la fonction `tapply`. En revanche, on peut définir des sous-populations par le croisement de plusieurs variables alors qu'avec la fonction `tapply`, on ne peut utiliser qu'une seule variable pour définir les sous-populations.

Dans les deux exemples suivants, les moyennes des taux de natalité et du PNB sont calculées sur les sous-populations formées par le croisement des modalités de la variable CONT (continent) et SURFACE (superficie divisée en deux catégories : petits ou grands territoires). Avec l'option `list`, on peut utiliser autant de variables que l'on veut pour former les sous-populations :

`aggregate(paysniv3$NAT,list(paysniv3$CONT,paysniv3$SURFACE),FUN=mean)`

	Group.1	Group.2	x
1	1	1	43.18919
2	2	1	34.27778
3	3	1	27.13333
4	4	1	13.74074
5	5	1	30.50000
6	1	2	44.62500
7	2	2	35.00000
8	3	2	27.77778
9	4	2	19.00000
10	5	2	16.00000

`aggregate(paysniv3$PNB,list(paysniv3$CONT,paysniv3$SURFACE),FUN=mean)`

	Group.1	Group.2	x
1	1	1	NA
2	2	1	NA
3	3	1	NA
4	4	1	7500.556
5	5	1	3105.000
6	1	2	NA
7	2	2	1820.000
8	3	2	4642.222
9	4	2	7400.000
10	5	2	10840.000



## Fonction aggregate.table

La fonction `aggregate.table` est assez similaire à la fonction `aggregate`. Elle permet également de produire des statistiques descriptives sur des sous-populations définies par le croisement de deux variables. Les résultats obtenus sont présentés sous la forme d'une table de contingence.

La fonction `aggregate.table` fait partie de la librairie `gdata`. On ne peut donc l'utiliser qu'après avoir chargé `gdata` par l'instruction `library(gdata)`.

```
aggregate.table(x,y,z,FUN=,...)
```

x	variable d'analyse
y,z	variables définissant les sous-populations
FUN=	fonction statistiques appliquée à x
...	arguments optionnels pour 'FUN'

### Exemples :

*On reprend l'exemple du calcul de la moyenne des taux de natalité par continent \* taille du pays :*

```
aggregate.table ( paysniv3$NAT, paysniv3$CONTI, paysniv3$SURFACE,
FUN=mean)
      1      2
1 43.18919 44.62500
2 34.27778 35.00000
3 27.13333 27.77778
4 13.74074 19.00000
5 30.50000 16.00000
```

*On peut également affecter cette table de contingence à un vecteur, puis à une data.frame dont on renommara les colonnes :*

```
X=aggregate.table ( paysniv3$NAT, paysniv3$CONTI, paysniv3$SURFACE,
FUN=mean)
X
      1      2
1 43.18919 44.62500
2 34.27778 35.00000
3 27.13333 27.77778
4 13.74074 19.00000
5 30.50000 16.00000

Y=as.data.frame(X)
Y
      1      2
1 43.18919 44.62500
2 34.27778 35.00000
3 27.13333 27.77778
4 13.74074 19.00000
5 30.50000 16.00000
```

```
colnames(Y)=c("Petit pays","Grand pays")
rownames(Y)=c("Afrique","Asie","Amérique","Europe","Océanie")
```

	Petit pays	Grand pays
Afrique	43.18919	44.62500
Asie	34.27778	35.00000
Amérique	27.13333	27.77778
Europe	13.74074	19.00000
Océanie	30.50000	16.00000

### 3.1.4. Fonctions pour obtenir des statistiques bivariées

Certaines fonctions permettent d'examiner les relations entre deux variables numériques. Il s'agit notamment de la covariance, qui présente l'inconvénient d'être sensible aux unités de mesure des variables, et du coefficient de corrélation linéaire, qui mesure à la fois la force de l'association et le sens de la relation entre les deux variables.

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

**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, on utilise la commande :

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

La commande suivante permet d'obtenir le 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
```

Pour obtenir 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)`.

On peut aussi utiliser l'indexation pour calculer les corrélations entre un sous-groupe de variables d'une data.frame :

```
cor(voitures[2:7])
      Cylindrée Puissance Longueur  Largeur  Poids  Vitesse
Cylindrée 1.0000000 0.7966277 0.7014619 0.6297572 0.7889520 0.6649340
Puissance 0.7966277 1.0000000 0.6413624 0.5208320 0.7652930 0.8443795
Longueur 0.7014619 0.6413624 1.0000000 0.8492664 0.8680903 0.4759285
Largeur 0.6297572 0.5208320 0.8492664 1.0000000 0.7168739 0.4729453
Poids 0.7889520 0.7652930 0.8680903 0.7168739 1.0000000 0.4775956
Vitesse 0.6649340 0.8443795 0.4759285 0.4729453 0.4775956 1.0000000
```

### 3.2. Statistiques descriptives / Variables nominales : fonction table

La fonction table permet d'obtenir la répartition des observations entre les modalités d'une variable nominale. Les résultats sont fournis en valeurs absolues. On n'obtient pas les pourcentages directement, mais ils peuvent être facilement calculés dans une deuxième étape.

Les comptages peuvent être effectués sur plusieurs bases : soit l'ensemble des observations (que la variable soit renseignée ou non) ou en excluant les non-réponses.

On peut également effectuer des tris croisés pour examiner la mesure d'association entre deux variables nominales. La signification de l'association entre les deux variables peut être examinée à partir du test du Chi-deux.

```
table(..., exclude =, dnn = list.names(...))
```

**exclude** valeurs exclues de l'analyse (notamment NA)  
**dnn** noms données aux dimensions de la table obtenue

#### Exemples:

*Dans les trois exemples suivants, on calcule la répartition des observations de paysniv3 entre les continents, tout d'abord en prenant en compte tous les continents, puis en éliminant les observations relatives à l'Afrique, puis en nommant CONTINENT la dimension relative aux modalités des variables :*

```
table(paysniv3$CONTI)
 1  2  3  4  5
53 44 39 28  9
```

```
table(paysniv3$CONTI,exclude=1)
 2  3  4  5
44 39 28  9
```

```
table(paysniv3$CONTI,dnn="CONTINENT")
CONTINENT
 1  2  3  4  5
53 44 39 28  9
```

On utilise maintenant le fichier voiture, qui décrit 18 voitures de marque différentes à partir de variables numériques, cylindrée, puissance, longueur, largeur, poids, vitesse, prix et de deux variables nominales, origine et finition.

La commande suivante permet d'obtenir le tableau croisé des variables origine et finition :

```
table(voitures$Origine,voitures$Finition)
```

	B	M	TB
D	2	0	1
F	2	2	2
GB	1	0	0
I	2	0	2
J	0	2	1
U	0	1	0

On peut stocker le précédent résultat dans une data.frame :

```
X=table(voitures$Origine,voitures$Finition)
```

```
X
      B  M  TB
D     2  0   1
F     2  2   2
GB    1  0   0
I     2  0   2
J     0  2   1
U     0  1   0
```

```
Y=as.data.frame(X)
```

```
  Var1 Var2 Freq
1  D    B     2
2  F    B     2
3  GB   B     1
4  I    B     2
5  J    B     0
6  U    B     0
7  D    M     0
8  F    M     2
9  GB   M     0
10 I    M     0
11 J    M     2
12 U    M     1
13 D   TB     1
14 F   TB     2
15 GB  TB     0
16 I   TB     2
17 J   TB     1
18 U   TB     0
```

La commande suivante permet d'effectuer le test du Chi-deux sur l'association des variables Origine et Finition. Le test du Chi-deux s'effectue sur l'objet X :

```
chisq.test(X)
```

```
Pearson's Chi-squared test
```

```
data: X
```

```
X-squared = 10.0857, df = 10, p-value = 0.433
```

```
Warning message:
```

```
l'approximation du Chi-2 est peut-être incorrecte in: chisq.test(X)
```

Les effectifs de chaque catégorie obtenue par le croisement des variables Origine et Finition sont systématiquement inférieurs à 5. Or, le test du Chi-deux a pour limite d'être assez sensible à la taille de l'échantillon : pour que le test soit fiable, il faut que l'effectif attendu pour chaque catégorie soit supérieur à 5. On regroupe donc les modalités des

variables *finition* et *origine*, en créant deux nouvelles variables dans la *data.frame* *voitures*:

*Origin2* : D/F/I/J/Autres

*Finition2* : M / B-TB

```
voitures$Origin2=factor(voitures$Origine,levels=c("I", "D", "F", "J", "Autres"))
```

Les observations correspondant aux modalités GB et J sont codées manquantes <NA>.

On les recode « Autres » :

```
voitures$Origin2[voitures$Origine=="GB"]="Autres"
```

```
voitures$Origin2[voitures$Origine=="J"]="Autres"
```

```
voitures$Finition2=factor(voitures$Finition,levels=c("M", "B-TB"))
```

Les observations correspondant aux modalités B et TB sont codées manquantes <NA>.

On les recode « B-TB » :

```
voitures$Finition2[voitures$Finition=="B"]="B-TB"
```

```
voitures$Finition2[voitures$Finition=="TB"]="B-TB"
```

```
X=table(voitures$Origin2,voitures$Finition2,dnn=c("Origine","Finition"))
```

Origine	Finition	
	M	B-TB
I	0	4
D	0	3
F	2	4
J	2	1
Autres	1	1

```
Y=as.data.frame(X)
```

	Origine	Finition	Freq
1	I	M	0
2	D	M	0
3	F	M	2
4	J	M	2
5	Autres	M	1
6	I	B-TB	4
7	D	B-TB	3
8	F	B-TB	4
9	J	B-TB	1
10	Autres	B-TB	1

On peut éventuellement calculer des pourcentages :

```
Y$Percent=Y$Freq/18
```

	Origine	Finition	Freq	Percent
1	I	M	0	0.00000000
2	D	M	0	0.00000000
3	F	M	2	0.11111111
4	J	M	2	0.11111111
5	Autres	M	1	0.05555556
6	I	B-TB	4	0.22222222
7	D	B-TB	3	0.16666667
8	F	B-TB	4	0.22222222
9	J	B-TB	1	0.05555556
10	Autres	B-TB	1	0.05555556

```
chisq.test(X)
```

Pearson's Chi-squared test

data: X

X-squared = 5.5385, df = 4, p-value = 0.2364

Warning message:

l'approximation du Chi-2 est peut-être incorrecte in: `chisq.test(X)`

Le test du Chi-deux n'est toujours pas fiable car les effectifs sont encore inférieurs à 5 dans chaque catégorie. Il faudra encore regrouper les valeurs.

### 3.3. Autres fonctions

Les fonctions de R étant nombreuses, nous 'en citons que quelques-unes qui peuvent s'avérer utiles.

<code>round(x, n)</code>	arrondit les éléments de x à n chiffres après la virgule
<code>rev(x)</code>	inverse l'ordre des éléments de x
<code>sort(x)</code>	trie les éléments de x dans l'ordre ascendant ; pour trier dans l'ordre descendant : <code>rev(sort(x))</code>
<code>order(x)</code>	tri d'une data.frame
<code>rank(x)</code>	rangs des éléments de x
<code>log(x, base)</code>	calcule le logarithme à base <code>base</code> de <code>x</code>
<code>scale(x)</code>	si x 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 ième 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 ième 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 x contenant les éléments de x qui sont dans y (NA sinon)
<code>which(x == a)</code>	retourne un vecteur des indices de x pour lesquels l'opération de comparaison est vraie (TRUE), dans cet exemple les valeurs de 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 k évènements parmi n répétitions = $n! / [(n - k)!k!]$
<code>na.omit(x)</code>	supprime les observations avec données manquantes (NA) (supprime la ligne correspondante si x est une matrice ou un tableau de données)
<code>na.fail(x)</code>	retourne un message d'erreur si x contient au moins un NA
<code>unique(x)</code>	si x est un vecteur ou un tableau de données, retourne un objet similaire mais avec les éléments dupliqués supprimés
<code>subset(x, ...)</code>	retourne une sélection de x en fonction de critères (... , typiquement des comparaisons : <code>x\$V1 &lt; 10</code> ) ; si x 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 :

Trois fonctions peuvent s'avérer particulièrement utiles :

- `subset`, qui permet de sélectionner les observations vérifiant certains critères,
- `sort` et `order` qui permettent de trier des données.

On veut sélectionner les 8 premières variables de la data.frame voitures et les observations correspondant aux voitures fabriquées en France :

```
X=subset(voitures[1:8],voitures$Origine=="F")
```

```
X
```

	Nom	Cylindrée	Puissance	Longueur	Largeur	Poids	Vitesse	Origine
3	SIMCA_1307_GLS	1294	68	424	168	1050	152	F
4	CITROEN_GS_CLUB	1222	59	412	161	930	151	F
7	PEUGEOT_504	1796	79	449	169	1160	154	F
8	RENAULT_16_TL	1565	55	424	163	1010	140	F
9	RENAULT_30_TS	2664	128	452	173	1320	180	F
15	RANCHO	1442	80	431	166	1129	144	F

On sélectionne les pays africains dont le taux de natalité est inférieur à 43,6, et les 10 premières variables de la `data.frame` `paysniv3` :

```
Y=subset(paysniv3[1:10],(paysniv3$CONTI==1&paysniv3$NAT<43.6))
```

Avec `sort(x)`, on ne peut trier qu'une seule colonne :

```
sort(Y$NOM)
[1] ALGERIA CAMEROON CAPE VERDE CHAD DJIBOUTI
[6] EGYPT EQUATORIAL GUINEA GABON GHANA GUINEA-BISSAU
[11] LESOTHO LIBYA MAURITIUS MOROCCO REUNION
[16] SAO TOME AND PRINCIPE SEYCHELLES SOUTH AFRICA TUNISIA
```

Pour trier toute la `data.frame`, on utilisera la fonction `order`.

On peut trier la `data.frame` sur une seule variable, par exemple sur le `NOM` (1<sup>ère</sup> variable de la `data.frame` `Y`) :

```
Y[order(Y[,1]), ]
  NOM      POP87 NAT MORT ACCR DOUB POP00 POP20 MORTI FERTI
1  ALGERIA    23.5  42  10  3.2  22  33.7  49.4  81.0  6.4
41 CAMEROON   10.3  43  16  2.7  26  14.5  23.5 103.0  5.9
9  CAPE VERDE  0.3  35  8  2.6  26  0.5  0.7  76.5  5.1
...
6  TUNISIA    7.6  32  7  2.5  27  9.9  13.6  78.0  4.5
```

On peut aussi faire un tri sur plusieurs variables, par exemple, les 3<sup>ème</sup> et 4<sup>ème</sup> variables de la `data.frame` `Y` :

```
Y[order(Y[,3],Y[,4]),]
  NOM      POP87 NAT MORT ACCR DOUB POP00 POP20 MORTI FERTI
30 MAURITIUS  1.1  19  7  1.2  57  1.3  1.6  25.1  2.3
...
50 LESOTHO    1.6  41  15  2.6  27  2.3  3.6 106.0  5.8
14 GUINEA-BISSAU 0.9  41  21  2.0  35  1.2  1.8 138.0  5.4
1  ALGERIA    23.5  42  10  3.2  22  33.7  49.4  81.0  6.4
12 GHANA     13.9  42  14  2.8  25  20.5  33.9  94.0  5.8
41 CAMEROON   10.3  43  16  2.7  26  14.5  23.5 103.0  5.9
25 DJIBOUTI   0.3  43  18  2.5  28  0.4  0.7 132.0  6.5
43 CHAD       4.6  43  23  2.0  35  6.3  9.5 143.0  5.9
```

## Exercices corrigés

### 1. Traitement des fichiers victimisation rates 1.csv et victimisation 2.csv

- 1.1. Importez dans R les fichiers victimisation rates 1.csv (data.frame victim1) et victimisation 2.csv (data.frame victim2) et affichez leur contenu à l'écran.

```
victim1=read.csv('d:/R_données_import/Victimisation rates 1.csv',sep = ";", dec=",")  
(ou victim1=read.csv('d:/R_données_import/Victimisation rates 1.csv',sep = ";",  
dec=",",nrows=22) si dernière ligne NA)  
> victim1  
  
victim2=read.csv('d:/R_données_import/Victimisation rates 2.csv',sep = ";", dec=",")  
> victim2
```

- 1.2. Affichez à l'écran le descriptif de chacune des data.frames victim1 et victim2 obtenues.

```
ls.str(pattern="victim1")  
ls.str(pattern="victim2")
```

- 1.3. Triez les data.frames victim1 et victim2 pour qu'elles soient dans le même ordre sur la première colonne (nom des pays).

```
victim1[order(victim1[,1]), ]  
victim2[order(victim2[,1]), ]
```

- 1.4. Juxtaposez les data.frames victim1 et victim2 en une data.frame victim.

```
victim=cbind(victim1,victim2[2:5])
```

- 1.5. Créez une variable continent qui prendra les modalités « Europe », « Amérique du Nord », « Océanie », « Asie ».

```
victim$continent=factor("Europe",levels=c("Europe", "Asie", "Amérique du Nord",  
"Océanie"))  
  
victim$continent[victim$X=="Japan"]="Asie"  
victim$continent[victim$X=="Canada"]="Amérique du Nord"  
victim$continent[victim$X=="United States"]="Europe"  
victim$continent[victim$X=="United States"]="Amérique du Nord"  
victim$continent[victim$X=="New Zealand"]="Océanie"  
victim$continent[victim$X=="Australia"]="Océanie"
```

- 1.6. Affichez pour la variable Car\_vandalism les statistiques descriptives suivantes par continent : minimum, maximum, moyenne, médiane, 1<sup>er</sup> et 3<sup>ème</sup> quartile.

```
tapply(victim$Car.vandalism,victim$continent,summary)
```

- 1.7. Affichez pour les variables numériques les statistiques suivantes : nombre de valeurs, nombre d'observations égales à 0, nombre de valeurs manquantes, minimum, maximum, différence entre maximum et minimum, somme des valeurs non manquantes, médiane, moyenne, écart-type de la moyenne, intervalle de confiance de la moyenne pour le niveau p, variance, écart-type, coefficient de variation, coefficient de symétrie et son critère de significativité, coefficient de concentration g2 et son critère de significativité, statistique d'un test de normalité de Shapiro-Wilk et sa probabilité associée.

```
library(pastecs)  
stat.desc(victim[2:10,], basic=TRUE, desc=TRUE, norm=FALSE, p=0.95)  
  
Eventuellement, il faudra préalablement installer le package pastecs.
```

- 1.8. Calculez la matrice de corrélation des variables numériques de victim.

```
cor(victim[2:10])
```



1.9. Triez la table *victim* sur les variables *Car.theft* et *Motorcycle.theft*.

```
victim[order(victim[,3],victim[,5]),]
```

1.10. Sauvez la table *victim*

```
save(victim,file="victim.RData")
```

## 2. Traitement du fichier *chien.RData*

2.1. Placez le fichier *chien.RData* dans le répertoire "C:/Program Files/R/R-2.2.1", puis chargez le en mémoire. Affichez le contenu du fichier.

```
load("chien.RData")  
ls.str(pattern="chien")
```

2.2. Convertissez toutes les variables en facteurs et affichez de nouveau le contenu du fichier.

```
chien$Taille=factor(chien$Taille)  
chien$Poids=factor(chien$Poids);  
chien$Velocite=factor(chien$Velocite);  
chien$Intelligence=factor(chien$Intelligence)  
chien$Affection=factor(chien$Affection)  
chien$Agressivite=factor(chien$Agressivite)  
chien$Fonction=factor(chien$Fonction)  
  
ls.str(pattern="chien")
```

2.3. Affichez le tri à plat de la variable *Fonction* (fonction du chien : 1. chien de compagnie, 2. chien de chasse, 3. chien de garde).

```
chien$Fonction=factor(chien$Fonction, labels=c("compagnie","chasse","garde"))  
table(chien$Fonction,dnn="Fonction")
```

2.4. Affichez le tri à plat de la variable *Fonction* pour les chiens de chasse et de garde uniquement.

```
table(chien$Fonction,dnn="Fonction",exclude="garde")
```

2.5. Créez la *data.frame* *tableau1* correspondant au croisement des variables *Taille* et *Intelligence*. Ajoutez une colonne correspondant aux pourcentages.

```
X=table(chien$Taille,chien$Intelligence,dnn=list("Taille","Intelligence"))  
tableau1=data.frame(X)  
tableau1$Percent=tableau1$Freq/27*100
```

2.6. Effectuez le test du Chi-2 pour le croisement des variables *Taille* et *Intelligence*.

```
chisq.test(X)
```

Attention : le test ne peut pas se faire à partir d'une *data.frame*.

2.7. Sélectionnez dans une *data.frame* *chien\_chasse* les chiens de chasse de taille moyenne (*Taille=2*). On ne conservera pas les variables *Taille* et *Fonction*.

```
chien_chasse=subset(data.frame(chien[1],chien[3:7]),(chien$Taille=="2"&chien$Fonction=="chasse"))
```

2.8. Affichez le croisement des variables *Fonction*, *Affection* et *Agressivite*.

```
table(chien$Fonction,chien$Affection,chien$Agressivite,dnn=list("Fonction","Affection","Agressivite"))
```

3. *Traitement du fichier voitures.RData.*

3.1. *Chargez le fichier voitures.*

```
load("voitures.RData")
```

3.2. *Affichez le prix moyen en fonction de l'origine et de la finition.*

```
aggregate.table(voitures$Prix,voitures$Origine,voitures$Finition,mean)
```

4. *Créez la matrice mat suivante :*

```
mat
  [,1] [,2] [,3] [,4]
[1,]  9   7  24  15
[2,]  4  18  13  19
[3,]  8  23   2   3
[4,] 16  12   6   8
```

*Calculez la matrice diagonale des valeurs propres de mat et la transposée de la matrice des vecteurs propres de mat.*

```
x=c(9,4,8,16,7,18,23,12,24,13,2,6,15,19,3,8)
mat=matrix(x,4,4)

mat1=diag(eigen(mat)$values)
mat2=eigen(mat)$vectors
```