

## **Licence 2 MIAGE**

### **ANALYSE DE DONNEES AVEC R**

# Chapitre 1: Introduction au langage R

## 1.1 Présentation du langage R

Le logiciel R est un logiciel de statistique créé par Ross Ihaka et Robert Gentleman au milieu des années 1990. Il est à la fois un langage informatique et un environnement de travail : les commandes sont exécutées grâce à des instructions codées dans un langage relativement simple, les résultats sont affichés sous forme de texte et les graphiques sont visualisés directement dans une fenêtre qui leur est propre. Ce logiciel sert à manipuler des données, à tracer des graphiques et à faire des analyses statistiques sur ces données. R est un logiciel multi-plates-formes: il fonctionne sous UNIX (et Linux), Microsoft Windows et Mac OS. R est un logiciel gratuit.

## 1.2 Installation du langage R

Ces premières instructions sont communes aux trois systèmes d'exploitation.

- Rendez-vous sur le site <http://www.r-project.org/>.
- Puis, à gauche sur la page d'accueil, vous trouverez un menu **Download, Packages**. Dans ce menu, cliquez sur **CRAN**.
- Choisissez un site miroir proche de chez vous.
- Un encadré blanc intitulé **Download and Install R** doit apparaître sur votre écran.

### 1.2.1 Installation du langage R sous Microsoft Windows

Pour installer R sous Windows, on peut

- Cliquez sur **Windows** puis sur **base**.
- Un encadré grisé doit apparaître dans lequel, à la première ligne, est inscrit **Download R 3.x.y for Windows (R 3.x.y** étant la dernière version de R disponible). Cliquez dessus pour télécharger le fichier.
- Exécutez le fichier que vous venez de télécharger en choisissant une **installation par défaut**.

### 1.2.2 Installation du langage R sous MacOS X

Sous **MacOS X**

- Cliquez sur **Download R for (Mac) OS X**.
- Une liste de fichiers à télécharger, intitulée **Files**: apparaît. Le premier élément de la liste est **R-3.x.y.pkg**. Cliquez dessus pour télécharger le fichier.
- Pour installer R, double-cliquez sur l'icône du package d'installation **R-3.x.y.pkg**.

### 1.2.3 Installation du langage R sous Linux

Sous **Linux**

- Cliquez sur **Download R for Linux** puis sur le nom de la distribution Linux installée sur votre ordinateur.
- Suivez les instructions détaillées d'installation sur le site.

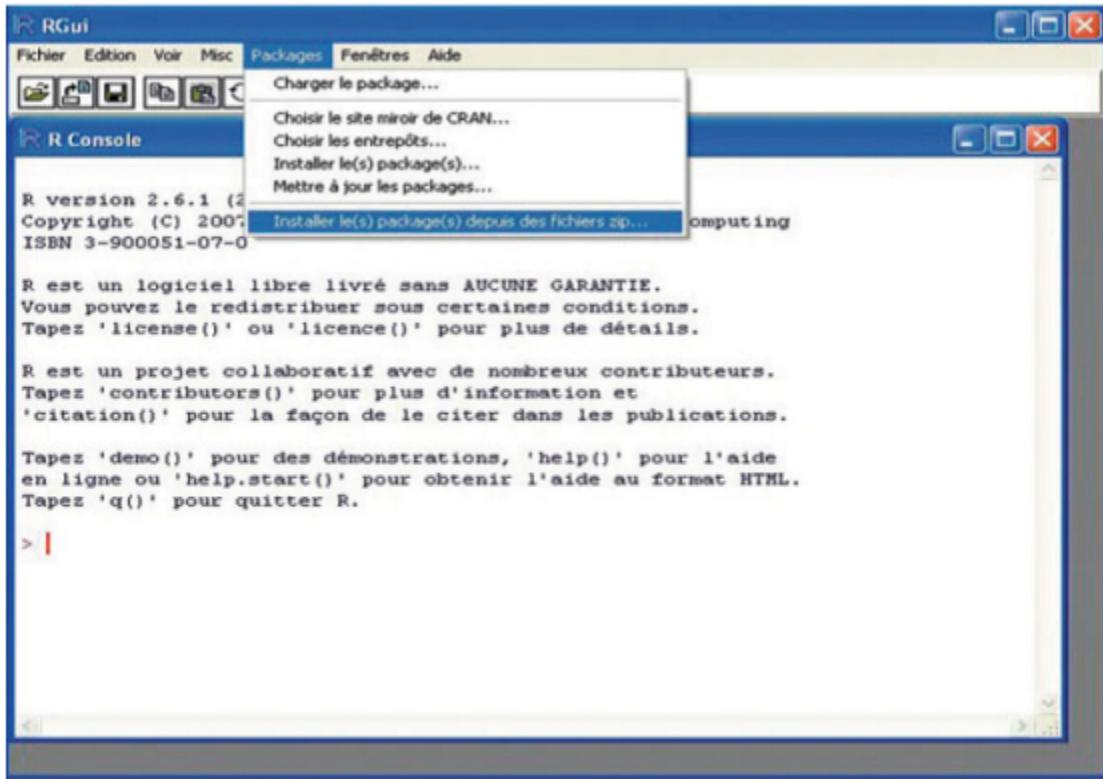
## 1.3 Installation des packages du langage R

Un package est une compilation d'outils. Certains sont déjà présents dans l'installation de base de R. En effet, lors de l'installation de R, un dossier **library** s'est créé par défaut. Il comprend les packages de base de R. Mais d'autres packages qui vous seront utiles pour réaliser vos analyses statistiques seront à télécharger puis à installer.

### 1.3.1 Installation à partir d'un fichier situé sur le disque

Pour installer, par exemple, le package **statmod.zip** situé sur votre disque,

- commencez par lancer le logiciel R en double-cliquant sur son icône;
- ensuite, allez dans le menu **Packages**, puis dans le sous-menu **Installer le(s) package(s) depuis des fichiers zip...** comme indiqué ci-dessous:



Sélectionnez alors le fichier **statmod.zip** situé sur le disque, puis cliquez sur **Ouvrir**.

### 1.3.2 Installation directement depuis l'internet

Pour installer, par exemple, les packages **Rcmdr**,

- commencez par lancer le logiciel R en double-cliquant sur son icône située sur le bureau;
- ensuite, allez dans le menu **Packages**, puis dans le sous-menu **Installer le(s) package(s)...**;
- sélectionnez un miroir (CRAN mirror) proche de votre situation géographique et cliquez sur **OK**;
- enfin, sélectionnez le package **Rcmdr**. Puis cliquez sur **OK**.

### 1.3.3 Installation depuis la ligne de commande

On peut se passer des menus de l'interface graphique de R. C'est par exemple utile sous Unix/Linux où le logiciel R ne possède pas d'interface graphique. Pour cela, tapez directement dans la console de R les commandes suivantes :

- pour des packages dont les fichiers d'extension .zip sont situés sur votre disque dur :  
`install.packages(choose.files(),repos = NULL)` ;
- pour un package (par exemple Rcmdr) dont le fichier est sur le site internet CRAN :  
`install.packages("Rcmdr")`.

## 1.4 Chargement des packages installés

Une fois que les packages souhaités ont été installés sur le disque dur de votre ordinateur, il faut les charger dans la mémoire de R pour pouvoir les utiliser.

Charger un package (en mémoire) signifie qu'il est temporairement mis à disposition de l'utilisateur dans R. Mais si l'on ferme puis rouvre R, ce package ne sera plus disponible depuis R. Il faudra donc le charger de nouveau.

Pour charger, par exemple, le package Rcmdr, on peut

- soit taper **require("Rcmdr")** dans la console de R,
- soit aller dans le menu **Packages** puis dans le sous-menu **Charger le package...** et sélectionner le package **Rcmdr** puis cliquer sur **OK**.

## 1.5 Prise en main du langage R

### 1.5.1 Démarrer R

Pour démarrer R, vous pouvez par exemple lancer le langage R en double-cliquant sur l'icône R qui se trouve par exemple sur votre bureau. La fenêtre R console s'ouvre. Elle vous affiche un texte contenant des informations sur le logiciel R (notamment la version de R que vous utilisez, quelques commandes pratiques de R, ... ).

Puis sous ce texte, le symbole rouge **>**, appelé prompt, apparaît également. Il signifie que R est prêt à travailler. C'est à la suite de **>** que vous taperez les lignes de commande de R. Une fois la commande tapée, vous devez toujours la valider en appuyant sur la touche **Entrée** du clavier.

### 1.5.2 Ecrire des scripts avec un éditeur de texte et les compiler sous R

Il est souvent plus pratique de composer le code R dans une fenêtre spécifique de l'interface graphique : la fenêtre de script.

Les entrées **Nouveau script** ou **Ouvrir un script** du menu **fichier** permettent de créer un nouveau script de commandes R ou d'accéder à un ancien script sauvegardé lors d'une session précédente d'utilisation du logiciel.

Pour exécuter des instructions à partir de la fenêtre de script il suffit de procéder par **copier-coller** ou de se servir du raccourci clavier **ctrl+R**.

### 1.5.3 Quitter R et sauvegarder sous R

Pour quitter R, vous utilisez la commande suivante : **q()**.

R vous propose alors de sauvegarder une image de la session de travail. Il suffit alors de cliquer sur **oui**. Cela permet que les commandes exécutées pendant la session et les objets enregistrés en mémoire soient conservés et soient donc <> et << réutilisables>> dans une future séance de travail. Cette image est automatiquement chargée en mémoire au prochain lancement de R, tel qu'indiqué à la fin du message d'accueil : **[Sauvegarde de la session précédente restaurée]**.

Si vous quittez R en choisissant la sauvegarde de l'espace de travail, deux fichiers sont créés : le fichier **.Rdata** qui contient des informations sur les variables utilisées et le fichier **.Rhistory** contient l'ensemble des commandes utilisées.

### 1.5.4 Consulter l'aide de R

R contient une aide en ligne très complète et très bien structurée sur toutes les fonctions que vous pouvez utiliser ainsi que sur les différents symboles du langage. Cette aide est accessible au moyen de plusieurs commandes dont la principale est **help()**.

### - La commande **help()**

La commande **help()** possède un alias qui est le point d'interrogation : **?**.

Si vous voulez obtenir l'aide sur la fonction **mean()** qui permet de calculer une moyenne, on peut taper l'instruction **help(mean)** ou l'instruction **?mean**.

### - Quelques commandes complémentaires

En plus de la commande principale **help()**, il existe des fonctions complémentaires qui peuvent se révéler utiles lorsque l'on cherche de l'aide sur une commande donnée. On peut citer les commandes:

**help.start()**: cette fonction ouvre un navigateur avec plusieurs liens vers des manuels au format HTML, de l'aide sur toutes les fonctions présentes dans tous les packages de R.

**help.search()** ou **??()**: cette fonction est à utiliser lorsque l'on ne connaît pas le nom d'une commande. Elle renvoie une liste des différentes fonctions (et des packages dans lesquels elles se trouvent) ayant un rapport avec votre requête.

**apropos()**: cette instruction renvoie une liste de toutes les fonctions dont le nom contient le paramètre d'appel. Ainsi, **apropos("mean")** renvoie toutes les fonctions dont le nom contient le mot **mean**.

**library(help=stats)**: cette commande permet de lister toutes les fonctions présentes dans la librairie **stats**.

**find("t.test")**: cette commande permet de savoir à quelle librairie appartient la fonction **t.test()**.

## 1.6 Interfaces graphiques de R

Il existe plusieurs interfaces graphiques qui ont vocation à faciliter l'utilisation du langage R. En voici deux parmi celles-ci qui méritent une attention particulière.

### 1.6.1 RStudio

RStudio est un outil puissant pour écrire facilement des scripts, des fonctions voire des bibliothèques R. C'est un programme à installer séparément de R et disponible pour les trois environnements Windows, Mac OSX et Linux. Il est téléchargeable sur le site <http://www.rstudio.com>. Il présente dans un même environnement le script, la console R, la liste ainsi que le contenu des objets présents dans la mémoire de R ainsi que les pages d'aide consultées ou les graphiques produits. Il permet également de transformer automatiquement un script R en un fichier au format html où sont intercalées les commandes R avec les résultats de celles-ci.

### 1.6.2 Rcmdr (RCommander)

C'est une bibliothèque pour le langage R, disponible pour les trois environnements Windows, Mac OSX et Linux. C'est avant tout une interface graphique pour un grand nombre de fonctions usuellement utilisées en statistique. Son utilisation rend la pratique de R proche de celles d'autres logiciels de statistique << à menus et boîtes de dialogue >> comme par exemple SPSS, Minitab ou Statistica. Elle permet en outre d'importer facilement des fichiers au format .csv, SPSS, SAS ou Minitab. Cette bibliothèque permet, elle aussi comme RStudio, de transformer automatiquement un script R en un fichier au format html où sont intercalées les commandes R avec les résultats de celles-ci.

# Chapitre 2: Les bases du langage R

## 2.1 Les variables

### 2.1.1 Affectation

Normalement, une expression est immédiatement évaluée et le résultat est affiché à l'écran :

```
4 + 7
```

```
## [1] 11  
cos(pi/4)  
  
## [1] 0.7071068
```

Le résultat d'une évaluation n'est pas enregistré, et est de fait perdu une fois affiché. Cependant, dans la plupart des cas, il est utile de conserver la sortie dans un objet(variable): cette opération s'appelle aussi une affectation du résultat dans une variable. Pour réaliser cette opération, on utilise la flèche d'affectation <-.

La flèche <- s'obtient en tapant le signe inférieur (<) suivi du signe moins (-). Une affectation évalue ainsi une expression, mais n'affiche pas le résultat qui est en fait stocké dans un objet. Pour afficher ce résultat, il vous suffira de taper le nom de cet objet suivi de la touche Entrée ou utiliser la fonction **print()**.

```
a<-11  
a
```

```
## [1] 11  
d<-45  
print(d)
```

```
## [1] 45
```

On peut affecter le résultat d'un calcul dans un objet et simultanément afficher ce résultat, il suffit de placer l'affectation entre parenthèses pour ainsi créer une nouvelle expression.

```
(b <- 12-7)
```

```
## [1] 5
```

#### Remarque

La création d'un objet peut se faire par affectation avec un des trois opérateurs "<-", ">-", "=" en lui donnant un nom.

Il est également possible de récupérer des données existantes de R grâce à la fonction **data**:

- **data()**: permet d'obtenir la liste des données existantes de R;
- **data(co2)**: permet de charger le jeu de données **co2**, qui est désormais exploitable.

### 2.1.2 Conventions de nommage des variables

Tout nom de variable ne peut être constitué que de caractères alphanumériques (les lettres minuscules a à z et majuscules A à Z, les chiffres 0 à 9), le point(.) ainsi que le caractère de soulignement{\_\_}.

Le R est sensible à la casse (R fait la distinction entre minuscules et majuscules); par exemple abc, Abc et ABC sont trois objets distincts.

Un nom de variable ne peut pas contenir des espaces ou commencer par un chiffre, sauf s'il est encadré de guillemets ("").

Certains mots sont réservés et il est interdit de les utiliser comme nom d'objet: break, else, for, function, if, in, next, repeat, return, while, TRUE, FALSE, Inf, NA, NaN, NULL, etc.

### 2.1.3 Modification et suppression d'objets

L'assignation du contenu d'un objet à un autre objet permet d'obtenir deux objets distincts. Ainsi, si un des deux objets est modifié, cela n'affecte pas l'autre.

```
a <- 12-7  
b <- a  
a <- 12+7  
  
a  
  
## [1] 19  
b  
  
## [1] 5
```

Par défaut **R** conserve en mémoire tous les objets créés dans la session que l'on peut connaître grâce aux fonctions **objects()** ou **ls()**. Il est recommandé de supprimer régulièrement des objets. Pour supprimer un objet, il faut utiliser la fonction **rm()**.

```
rm(a)
```

```
a  
Error: objet 'a' not found
```

## 2.2 Les types de données dans R

Le type ou le mode d'un objet est la nature des éléments qui le composent.

Pour obtenir le type d'un objet, on utilise la fonction **mode()** ou encore la fonction **typeof()**.  
On distingue quatre principaux types de données: numeric, complex, logical, character.

### 2.2.1 Type numérique (numeric)

Il y a deux types numériques : les entiers (**integer**) et les réels (**double**).

```
a<- 2; b<- 3.5  
  
mode(a)  
  
## [1] "numeric"  
mode(b)  
  
## [1] "numeric"  
typeof(a)  
  
## [1] "double"  
typeof(b)  
  
## [1] "double"  
c <- as.integer(a)  
typeof(c)  
  
## [1] "integer"
```

Les variables *a* et *b* sont du type “**double**” et la variable *c* est du type “**integer**”(elle a la même valeur que la variable *a* excepté qu’elle a été forcée à être du type “integer” grâce à la fonction **as.integer()**). L’intérêt du type “**integer**” est que le stockage d’entiers nécessite moins de place mémoire que le stockage de réels.

### 2.2.2 Type complexe (complex)

Les nombres complexes sont caractérisés par leur partie réelle, que l’on peut obtenir à l’aide de la fonction **Re()** et par leur partie imaginaire, que l’on obtient grâce à la fonction **Im()**. On crée un nombre complexe à l’aide de la lettre **i**.

Les fonctions **Mod()** et **Arg()** permettent d’obtenir respectivement le module et l’argument du nombre complexe.

```
z<- 3+4i  
Re(z)  
## [1] 3  
Mod(z)  
## [1] 5  
Im(z)  
## [1] 4  
Arg(z)  
## [1] 0.9272952  
mode(z)  
## [1] "complex"
```

### 2.2.3 Type booléen ou logique (logical)

Les données de type logique peuvent prendre deux valeurs : **TRUE** ou **FALSE**. Elles répondent à une condition logique.

```
a<- 2; b<- 3.5; c <- TRUE  
a==b  #test d'égalité  
## [1] FALSE  
b>a  
## [1] TRUE  
is.logical(c)  
## [1] TRUE  
is.integer(b)  
## [1] FALSE
```

## 2.2.4 Données manquantes, indéterminées et infinies

Supposons que l'on ait demandé à 10 personnes choisies au hasard dans la rue leur âge. Voici la série de mesures recueillies, arrondies à l'entier le plus proche, ainsi que le sexe de la personne.

<b>age</b>	18	27	34	18	24		30	28	19	19
<b>sex</b>	F	F	M	F	M	M	M	F	M	F

La sixième personne interrogée n'a pas donné son âge, on considère qu'il s'agit d'une valeur manquante représentée sous R par le symbole NA (not available). Les données manquantes NA sont considérées comme étant de type logical par R.

On peut créer une variable appelée **age** à laquelle on associe la liste des nombres présentés dans le tableau précédent.

```
age <- c(18, 27, 34, 18, 24, NA, 30, 28, 19, 19)
```

R propose plusieurs fonctions permettant de manipuler les données manquantes. La fonction **is.na()** permet de vérifier la présence de données manquantes tandis que la fonction **which(is.na())** permet d'obtenir le numéro de l'observation manquante dans la variable.

```
is.na(age)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE FALSE FALSE  
which(is.na(age))
```

```
## [1] 6
```

Par ailleurs, R dispose d'objets spéciaux pour représenter les valeurs mathématiques spéciales  $+\infty$  ou  $-\infty$  ainsi que les formes indéterminées du type  $\frac{0}{0}$  ou  $\infty - \infty$ .

*Inf* représente  $+\infty$ ;

*-Inf* représente  $-\infty$ ;

*NaN* (Not a Number) représente une forme indéterminée.

Ces valeurs sont testées avec les fonctions **is.infinite()**, **is.finite()** et **is.nan()**.

## 2.2.5 Type chaîne de caractères (character)

Toute information mise entre guillemets (simple ' ou double "") correspond à une chaîne de caractères.

```
a <- "Hello world!"  
a
```

```
## [1] "Hello world!"
```

```
typeof(a)
```

```
## [1] "character"
```

## 2.3 Les structures de données

Il existe de nombreuses structures servant à organiser les différents types de données dans R. Les principales structures de données sont: les vecteurs (vector), les matrices (matrix), les tableaux (arrays), les listes (list), les tableaux individus  $\times$  variables (data.frame), les facteurs (factor) et les variables ordinaires (ordered), les

dates.

Pour connaître la structure d'un objet, il est possible d'utiliser la fonction **class()**.

### 2.3.1 Les vecteurs (vector)

Les vecteurs constituent la structure de données la plus simple en R. Elle représente une suite de données de même type. Les formes les plus simples sont numeric, character et logical (TRUE ou FALSE).

La création d'un vecteur peut se faire à l'aide de la fonction de concaténation **c()**, la fonction **seq()**, par la fonction **rep()**, par l'opérateur séquence : ou encore par la fonction **sample**.

```
nomca<-c('toto', "tata", 'tonton')
nomca

## [1] "toto"   "tata"   "tonton"
class(nomca)

## [1] "character"
nomlo<-c(TRUE, FALSE)
nomlo

## [1] TRUE FALSE
class(nomlo)

## [1] "logical"
d<-c(3, 1, 7)
d

## [1] 3 1 7
class(d)

## [1] "numeric"
seq(from=0, to=1, by=0.1)

## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
vec <- 2:36
vec

## [1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
## [24] 25 26 27 28 29 30 31 32 33 34 35 36
rep(2,5)

## [1] 2 2 2 2 2
rep(c(1,2),each=4)

## [1] 1 1 1 1 2 2 2 2
sample(1:100, 5) # tirer 5 nombres entre 1 et 100 de manière aléatoire

## [1] 39 23 28 56 38
```

Il est possible d'attribuer un nom aux éléments d'un vecteur, soit lors de la création, soit à posteriori, en utilisant la fonction **names()**.

```

a <- c(nom = "Piketty", prenom = "Thomas", annee_naissance = "1971")
a

## nom prenom annee_naissance
## "Piketty" "Thomas" "1971"

vec <- c(1, 3, 6, 2, 7, 4, 8, 1, 0)
names(vec) <- letters[1:9] #les 9 premières lettres de l'alphabet
vec

## a b c d e f g h i
## 1 3 6 2 7 4 8 1 0

```

La fonction `is.vector()` permet de tester si l'objet passé en paramètre est un vecteur. Elle retourne TRUE si c'est un vecteur.

```
is.vector(vec)
```

```
## [1] TRUE
```

La fonction `length()` donne le nombre d'éléments qui composent le vecteur.

```
length(vec)
```

```
## [1] 9
```

Notez que lors de la création d'un vecteur, il est possible de mélanger des données de plusieurs types différents. R convertit les données dans le type le plus général.

```
c(5,TRUE,10)
```

```
## [1] 5 1 10
```

On peut effectuer les opérations arithmétiques sur les vecteurs : addition, soustraction, multiplication, division,etc. On peut aussi appliquer une fonction à un vecteur.

```
x<-c(1, 3, 7, 10, -1)
y<-c(2, 4, 3, 1, 6)
```

```
x+y      #addition
```

```
## [1] 3 7 10 11 5
```

```
x-y      #soustraction
```

```
## [1] -1 -1  4  9 -7
```

```
x*y      #multiplication
```

```
## [1] 2 12 21 10 -6
```

```
x/y      #division
```

```
## [1] 0.5000000 0.7500000 2.3333333 10.0000000 -0.1666667
```

```
x^y      #puissance
```

```
## [1] 1 81 343 10 1
```

```
sqrt(y)  #racine carrée de y
```

```
## [1] 1.414214 2.000000 1.732051 1.000000 2.449490
```

La fonction `sort()` permet de ranger les éléments d'un vecteur par valeurs croissantes ou décroissantes.

```

z<-c(12, 5, -4, 1, 0, 7, 41)
v<-c("bleu", "rouge", "vert", "noir")
sort(z) #par valeurs croissantes

## [1] -4  0  1  5  7 12 41

sort(v)

## [1] "bleu"  "noir"  "rouge" "vert"
sort(z, decreasing = TRUE) #par valeurs décroissantes

## [1] 41 12  7  5  1  0 -4
sort(v, decreasing = TRUE)

## [1] "vert"  "rouge" "noir"  "bleu"

```

On peut accéder aux éléments d'un vecteur en utilisant l'indexation. Cette indexation peut être de type numérique ou logique. Les crochets sont utilisés pour l'indexation.

```

jour<-c("dim", "lun", "mar", "mer", "jeu", "ven", "sam")
jour[2] #accès au deuxième élément du vecteur

## [1] "lun"

jour[c(1, 5)] #les premier et cinquième éléments du vecteur

## [1] "dim" "jeu"

jour[c(TRUE, FALSE, FALSE, TRUE, FALSE, FALSE)] # les premier et cinquième éléments du vecteur

## [1] "dim" "jeu"

jour[-3] # tous les éléments du vecteur sauf le troisième

## [1] "dim" "lun" "mer" "jeu" "ven" "sam"

jour[-c(1,5)] # tous les jours sauf le premier et le cinquième

## [1] "lun" "mar" "mer" "ven" "sam"

jour[-(1:3)] # tous les sauf les trois premiers

## [1] "mer" "jeu" "ven" "sam"

```

On peut aussi accéder à certains éléments d'un vecteur en se basant sur leur valeur ou sur d'autres éléments provenant d'autres objets R.

```

x <- c(2, 4, 5, 1, 7, 6)
y<-c(12, 30, 25, -74, 2)

x[x>3]

## [1] 4 5 7 6
x[(x>2)&(x<7)]

## [1] 4 5 6
x[y>20]

## [1] 4 5

```

Si on souhaite obtenir la position de certains éléments du vecteur, on peut utiliser la fonction **which()**. Les fonctions **which.min()** et **which.max()** retournent respectivement la position du (premier) minimum et du premier maximum d'un vecteur numérique ou logique.

```
x <- c(2, 4, 5, 1, 7, 6, 1, 7, 3)
which(x<5)
```

```
## [1] 1 2 4 7 9
```

```
which.min(x)
```

```
## [1] 4
```

```
which.max(x)
```

```
## [1] 5
```

Il est également possible de remplacer certains éléments d'un vecteur. Pour cela, on utilise la flèche d'affectation.

```
x <- c(2, 4, 5, 1, 7, 6)
x[2]<-12
x
```

```
## [1] 2 12 5 1 7 6
```

```
x[x %in% c(6,7)]<-9 #remplacer les nombres 6 et 7 par 9
x
```

```
## [1] 2 12 5 1 9 9
```

```
x[which(x==1)]<-10
x
```

```
## [1] 2 12 5 10 9 9
```

```
x[which(x<6)]<-15
x
```

```
## [1] 15 12 15 10 9 9
```

### 2.3.2 Les matrices (matrix) et les tableaux (arrays)

Les matrices et les tableaux généralisent la notion de vecteur puisqu'elles représentent des suites à double indice pour les matrices et à multiples indices pour les tableaux(array). Ici aussi les éléments doivent avoir le même type, sinon des conversions implicites seront effectuées.

On utilise la fonction **matrix()** pour créer une matrice. Les matrices sont remplies par colonne. Cependant, il est possible de changer l'ordre de remplissage des matrices (remplissage par ligne) en donnant la valeur **TRUE** à l'argument **byrow** de la fonction **matrix()**.

```
(Y <- matrix(1:12, nrow=4, ncol=3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
(X <- matrix(1:12, nrow=4, ncol=3, byrow=TRUE))
```

```
##      [,1] [,2] [,3]
```

```

## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
class(X)

```

```
## [1] "matrix"
```

Il est possible d'attribuer des noms aux lignes grâce à la commande **rownames** et aux colonnes grâce à la commande **colnames**.

```
(mtc <- matrix(3:14, nrow=4, ncol=3))

##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
## [3,]    5    9   13
## [4,]    6   10   14

rownames(mtc)<-c("ligne1", "ligne2", "ligne3", "ligne4")
colnames(mtc)<-c("colonne1", "colonne2", "colonne3")
mtc
```

```

##      colonne1 colonne2 colonne3
## ligne1        3        7       11
## ligne2        4        8       12
## ligne3        5        9       13
## ligne4        6       10       14
```

On peut concaténer des matrices avec les fonctions **cbind** (concaténation par colonne) et **rbind** (concaténation par ligne).

```
z<-cbind(X, Y)
z

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    2    3    1    5    9
## [2,]    4    5    6    2    6   10
## [3,]    7    8    9    3    7   11
## [4,]   10   11   12    4    8   12
```

```
H<-rbind(X, Y)
H

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
## [5,]    1    5    9
## [6,]    2    6   10
## [7,]    3    7   11
## [8,]    4    8   12
```

On utilise la fonction **array()** pour créer un tableau. Par exemple, le code ci-après permet de créer un tableau de dimension  $3 \times 4 \times 2$ .

```
array(1:24, dim=c(3, 4, 2))
```

```
## , , 1
```

```

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

```

Les fonctions **nrow()** et **ncol()** donnent le nombre de lignes (rows) et de colonnes (columns) d'une matrice. La fonction **dim()** donne un vecteur contenant les dimensions de la matrice ou du tableau donné en paramètre.

```
ncol(X)
```

```
## [1] 3
```

```
nrow(X)
```

```
## [1] 4
```

```
dim(X)
```

```
## [1] 4 3
```

### Calcul avec les matrices

Comme pour les vecteurs, on peut effectuer des calculs sur les matrices : addition, soustraction, multiplication élément par élément, produit matriciel,inversion matricielle, transposition, déterminant, etc.

```
A<- matrix(c(1, -2, 9, 4, 2, 7, -2, -5, 1), ncol=3)
B<- matrix(2:10, nrow=3, ncol=3)
```

```
A+B      #addition
```

```
## [,1] [,2] [,3]
## [1,]    3    9    6
## [2,]    1    8    4
## [3,]   13   14   11
```

```
A-B      #soustraction
```

```
## [,1] [,2] [,3]
## [1,]   -1   -1  -10
## [2,]   -5   -4  -14
## [3,]    5    0   -9
```

```
A*B      #produit élément par élément
```

```
## [,1] [,2] [,3]
## [1,]    2   20  -16
## [2,]   -6   12  -45
## [3,]   36   49   10
```

```
A%*%B  #produit matriciel
```

```
## [,1] [,2] [,3]
## [1,]    6   15   24
## [2,]  -18  -33  -48
```

```

## [3,] 43 94 145
det(A) #déterminant d'une matrice

## [1] -71
solve(A) #inverse de la matrice A

## [,1] [,2] [,3]
## [1,] -0.5211268 0.2535211 0.2253521
## [2,] 0.6056338 -0.2676056 -0.1267606
## [3,] 0.4507042 -0.4084507 -0.1408451

t(A) #transposée de A

## [,1] [,2] [,3]
## [1,] 1 -2 9
## [2,] 4 2 7
## [3,] -2 -5 1

eigen(B) #diagonalisation de la matrice B: valeurs propres et vecteurs propres

## eigen() decomposition
## $values
## [1] 1.894987e+01 -9.498744e-01 -9.051422e-17
##
## $vectors
## [,1] [,2] [,3]
## [1,] -0.4819390 -0.8680145 0.4082483
## [2,] -0.5725865 -0.2087420 -0.8164966
## [3,] -0.6632341 0.4505304 0.4082483

```

Il est également possible de résoudre des systèmes d'équations linéaires grâce à la fonction **solve()**. On souhaite, par exemple, résoudre le système suivant:  $\begin{cases} 23x + 31y = 1; \\ 34x + 46y = 2. \end{cases}$

On a:

```

D<-matrix(c(23, 34, 31, 46), ncol=2)
V<-c(1, 2)
solve(D, V)

```

```
## [1] -4 3
```

La solution du système est  $x=-4$  et  $y=3$ .

On peut appliquer une fonction aux lignes (**MARGIN=1**) ou aux colonnes d'une matrice (**MARGIN=2**) grâce à la fonction **apply**.

```

E<- matrix(c(1, -2, 9, 4, 2, 7, -2, -5, 1), ncol=3)
E

```

```

## [,1] [,2] [,3]
## [1,] 1 4 -2
## [2,] -2 2 -5
## [3,] 9 7 1

apply(E, MARGIN=2, sum) # sommes par colonnes

```

```
## [1] 8 13 -6
```

### Sélection d'éléments d'une matrice

Pour les matrices ou les tableaux de données, l'extraction par indice se fait dans le même esprit, mais il faut

indiquer un vecteur d'indices (i) pour les lignes et un pour les colonnes (j), de la manière suivante : A[i, j], avec A la matrice ou le tableau de données. En omettant le vecteur d'indices pour les lignes ou les colonnes, R retourne toutes les lignes ou les colonnes respectivement. Enfin, en ajoutant le symbole “moins” (-) devant le vecteur d'indices de lignes ou de colonnes, on demande à R de ne pas retourner les éléments dont les indices sont mentionnés.

```
A<- matrix(c(1, -2, 9, 4, 2, 7, -2, -5, 1), ncol=3)
A

##      [,1] [,2] [,3]
## [1,]    1    4   -2
## [2,]   -2    2   -5
## [3,]    9    7    1
A[1, 2] #élément de la première ligne et de la deuxième colonne
## [1] 4
A[1, ] #éléments de la première ligne
## [1] 1 4 -2
A[, 3] #éléments de la troisième colonne
## [1] -2 -5  1
A[1:2, ] # les deux premières lignes
##      [,1] [,2] [,3]
## [1,]    1    4   -2
## [2,]   -2    2   -5
A[1:2, -1] # les deux premières lignes sans la première colonne
##      [,1] [,2]
## [1,]    4   -2
## [2,]    2   -5
A[A>2] # les éléments de la matrice A supérieurs à 2
## [1] 9 4 7
```

Comme dans le cas des vecteurs, on peut remplacer certains éléments d'une matrice.

```
A<- matrix(c(1, -2, 9, 4, 2, 7, -2, -5, 1), ncol=3)
A

##      [,1] [,2] [,3]
## [1,]    1    4   -2
## [2,]   -2    2   -5
## [3,]    9    7    1
A[2, 1]<-3
A

##      [,1] [,2] [,3]
## [1,]    1    4   -2
## [2,]    3    2   -5
## [3,]    9    7    1
A[A>2]<-4
A
```

```

##      [,1] [,2] [,3]
## [1,]    1    4   -2
## [2,]    4    2   -5
## [3,]    4    4    1

```

### 2.3.3 Les listes (list)

Les listes dans R sont les structures les plus souples et à la fois les plus riches. Contrairement aux structures précédentes, les listes permettent de regrouper dans une même structure des données de types différents sans pour autant les altérer. De façon générale, chaque élément d'une liste peut ainsi être un vecteur, une matrice, un tableau ou même une liste. Il est possible de nommer les éléments de la liste, ce qui permet une meilleure lecture et un accès aux données parfois plus pratique.

Pour créer des listes, on utilise la fonction `list()`.

```
x<- list(size = c(1, 5, 2), user = "Joe", new = TRUE)
```

```
class(x)
```

```
## [1] "list"
```

```
length(x)
```

```
## [1] 3
```

Dans l'exemple ci-dessus, le premier élément de la liste (size) est de mode “**numeric**”, le second (user) est de mode “**character**” et le troisième (new) est de mode “**logical**”.

Pour accéder au premier élément de la liste  $x$ , on peut utiliser les instructions suivantes:  $x['size']$  ou  $x[[1]]$  ou  $x\$size$ .

```
x['size']
```

```
## $size
```

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

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

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

```
x$new
```

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

On peut remplacer, ajouter ou supprimer certains éléments d'une liste grâce à la flèche d'affectation ou concaténer des listes avec la fonction `c()`. On ne peut ajouter ou supprimer que des éléments situés à la fin de la liste.

```
y<- list(size = c(1, 5, 2), user =c("Joe", "Karl") , new = TRUE)
```

```
z<- list(longueur = c(4, 6, 2), enfant = "Alain", statut = c(TRUE, FALSE))
```

```
y$user[1]<-"Jack"
```

```
y
```

```
## $size
```

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

```
##
```

```
## $user
```

```
## [1] "Jack" "Karl"
```

```
##
```

```
## $new
```

```
## [1] TRUE
```

```

y[[4]]<-c(7, 5)
y

## $size
## [1] 1 5 2
##
## $user
## [1] "Jack" "Karl"
##
## $new
## [1] TRUE
##
## [[4]]
## [1] 7 5

y[[4]]<-NULL
y

## $size
## [1] 1 5 2
##
## $user
## [1] "Jack" "Karl"
##
## $new
## [1] TRUE
concat<-c(z,y)
concat

## $longueur
## [1] 4 6 2
##
## $enfant
## [1] "Alain"
##
## $statut
## [1] TRUE FALSE
##
## $size
## [1] 1 5 2
##
## $user
## [1] "Jack" "Karl"
##
## $new
## [1] TRUE

```

### 2.3.4 Les data frames(data.frame)

Un data frame est une liste composée d'un ou plusieurs vecteurs de même longueur. Le mode des vecteurs n'est pas nécessairement identique à celui des autres. Il est généralement représenté sous la forme d'un tableau individus  $\times$  variables dont les lignes correspondent aux individus et les colonnes aux variables (ou caractères) mesurées sur ces derniers. Chaque colonne représente une variable particulière dont tous les éléments sont du même type.

La fonction **data.frame()** permet de créer un data frame tandis que la fonction **as.data.frame()** permet quant à elle de convertir un objet d'un autre type en data frame.

Comme pour les matrices, on peut obtenir les dimensions d'un data frame avec la fonction **dim()**, le nombre de lignes avec **nrow()** et le nombre de colonnes avec **ncol()**.

```
IMC<-data.frame(Sexe = c("H", "F", "H", "F", "H", "F"),
                  taille = c(1.83, 1.76, 1.82, 1.60, 1.90, 1.66),
                  poids = c(67, 58, 66, 48, 75, 55),
                  row.names = c("Rémy", "Lol", "Pierre", "Domi", "Ben", "Cécile"))
IMC
```

```
##           Sexe taille poids
## Rémy      H    1.83   67
## Lol       F    1.76   58
## Pierre    H    1.82   66
## Domi     F    1.60   48
## Ben      H    1.90   75
## Cécile   F    1.66   55
```

```
class(IMC)
```

```
## [1] "data.frame"
```

```
ncol(IMC)
```

```
## [1] 3
```

```
nrow(IMC)
```

```
## [1] 6
```

```
dim(IMC)
```

```
## [1] 6 3
```

La fonction **str()** permet d'afficher la structure de chacune des colonnes d'un data.frame, la fonction **dimnames()** donne les noms des lignes et des colonnes (sous la forme d'une liste), la fonction **rownames()** donne les noms des lignes tandis que les fonctions **names()** et **colnames()** permettent d'obtenir le nom des colonnes.

```
str(IMC)
```

```
## 'data.frame': 6 obs. of 3 variables:
```

```
## $ Sexe : Factor w/ 2 levels "F","H": 2 1 2 1 2 1
```

```
## $ taille: num 1.83 1.76 1.82 1.6 1.9 1.66
```

```
## $ poids : num 67 58 66 48 75 55
```

```
names(IMC)
```

```
## [1] "Sexe"    "taille"   "poids"
```

```
rownames(IMC)
```

```
## [1] "Rémy"    "Lol"     "Pierre"   "Domi"    "Ben"     "Cécile"
```

```
dimnames(IMC)
```

```
## [[1]]
## [1] "Rémy"    "Lol"     "Pierre"   "Domi"    "Ben"     "Cécile"
```

```
##
```

```
## [[2]]
## [1] "Sexe"    "taille"   "poids"
```

On peut ajouter ou supprimer des colonnes.

```
IMC["profession"]<-c("ouvrier", "cadre", "artisan", "etudiant", "sans emploi", "commerçant")
IMC

##      Sexe taille poids  profession
## Rémy     H   1.83    67    ouvrier
## Lol      F   1.76    58     cadre
## Pierre   H   1.82    66     artisan
## Domi    F   1.60    48    etudiant
## Ben      H   1.90    75 sans emploi
## Cécile   F   1.66    55  commerçant

IMC["profession"]<-NULL
IMC
```

```
##      Sexe taille poids
## Rémy     H   1.83    67
## Lol      F   1.76    58
## Pierre   H   1.82    66
## Domi    F   1.60    48
## Ben      H   1.90    75
## Cécile   F   1.66    55
```

On peut extraire certaines colonnes et certaines lignes en utilisant les crochets.

```
dftaille<-IMC[["taille"]]#extraction de la colonne "taille"
dftaille

##      taille
## Rémy     1.83
## Lol      1.76
## Pierre   1.82
## Domi    1.60
## Ben      1.90
## Cécile   1.66

dftaillepoids<-IMC[c("taille", "poids")]#extraction des colonnes "taille" et "poids"
dftaillepoids
```

```
##      taille poids
## Rémy     1.83    67
## Lol      1.76    58
## Pierre   1.82    66
## Domi    1.60    48
## Ben      1.90    75
## Cécile   1.66    55
```

```
dfIMC12<-IMC[1:2, ] #extraction des deux premières lignes
dfIMC12
```

```
##      Sexe taille poids
## Rémy     H   1.83    67
## Lol      F   1.76    58
```

On peut aussi sélectionner une partie des observations suivant un ou plusieurs critères et placer le résultat dans un nouveau data frame.

```
IMCfemme<-subset(IMC, Sexe=="F") #data frame contenant uniquement les femmes
IMCfemme
```

```

##      Sexe taille poids
## Lol      F    1.76    58
## Domi     F    1.60    48
## Cécile   F    1.66    55

```

### 2.3.5 Les facteurs (factor) et les variables ordinales (ordered)

Les facteurs offrent un moyen alternatif au stockage des chaînes de caractères. R permet d'organiser les chaînes de caractères de façon plus astucieuse au moyen de la fonction **factor()**.

La commande **factor()** permet de faire connaître à R la nature qualitative de cette variable, et d'associer à chaque modalité ou niveau du facteur (levels) des étiquettes textuelles plus informatives. Par défaut, la commande **factor()** ne fait qu'ajouter des niveaux à une variable, les niveaux retenus correspondant aux valeurs uniques, triées par ordre lexicographique, présentes dans la variable.

La fonction **levels()** retourne les niveaux du facteur tandis que la commande **nlevels()** donne le nombre de niveaux du facteur.

```

pays <- factor(c("France", "France", "Chine", "Espagne", "Chine"))
pays

```

```

## [1] France France Chine Espagne Chine
## Levels: Chine Espagne France

```

```

class(pays)

```

```

## [1] "factor"

```

```

levels(pays)

```

```

## [1] "Chine"   "Espagne" "France"

```

```

nlevels(pays)

```

```

## [1] 3

```

Les facteurs peuvent être un moyen efficace de stockage des chaînes de caractères lorsqu'il y a répétition des éléments du vecteur contenant les chaînes de caractères. Ceci est possible car les niveaux d'un facteur sont codés, en interne, par des nombres entiers.

```

as.integer(pays)

```

```

## [1] 3 3 1 2 1

```

Lorsque les variables catégorielles sont ordonnées, on utilise la fonction **ordered()**. Le paramètre **levels** de la fonction **ordered()** permet de spécifier l'ordre des modalités de la variable.

```

z <- ordered(c("Petit", "Grand", "Moyen", "Grand", "Moyen", "Petit", "Petit"),
levels=c("Petit", "Moyen", "Grand"))

```

```

class(z)

```

```

## [1] "ordered" "factor"

```

### 2.3.6 Les dates

Pour pouvoir effectuer des opérations sur les dates, il est indispensable de les transformer en objet date grâce à la fonction **as.Date()**. La fonction **as.Date()** convertit une chaîne de caractères en date.

```

dat <- c("27/05/98", "18/06/99", "14/07/19", "01/09/19")
dat<- as.Date(dat, "%d/ %m/ %y", tz = "UTC")

```

```
## Warning in strptime(x, format, tz = "GMT"): unable to identify current timezone 'T':  
## please set environment variable 'TZ'  
dat
```

```
## [1] "1998-05-27" "1999-06-18" "2019-07-14" "2019-09-01"  
class(dat)
```

```
## [1] "Date"
```

Dans R, pour afficher la date courante, on peut utiliser les fonctions suivantes: **Sys.time()**, **date()**, **Sys.Date()**.

```
Sys.time()
```

```
## [1] "2020-07-13 12:33:57 UTC"  
date()  
## [1] "Mon Jul 13 12:33:57 2020"  
Sys.Date()
```

```
## [1] "2020-07-13"
```

Le package **lubridate** propose également des fonctions pour gérer les dates. Il est nécessaire de charger le package en mémoire. Les fonctions **ymd()**, **dmy()** et **ydm()** permettent de convertir les dates au format date.

Les fonctions **year()**, **month(x)** et **day(x)** permettent d'extraire respectivement l'année, le mois et le jour.

```
library(lubridate)
```

```
##  
## Attaching package: 'lubridate'  
## The following object is masked from 'package:base':  
##  
##     date  
x <- dmy("16 août 2019")  
x
```

```
## [1] "2019-08-16"
```

```
year(x)
```

```
## [1] 2019
```

```
month(x)
```

```
## [1] 8
```

```
month(x, label = TRUE)
```

```
## [1] août
```

```
## 12 Levels: janv\.\. < févr\.\. < mars < avr\.\. < mai < juin < ... < déc\.\.
```

```
day(x)
```

```
## [1] 16
```

```
yday(x)
```

```
## [1] 228
```

```
wday(x)

## [1] 6

wday(x, label = TRUE)

## [1] ven\\.
## 7 Levels: dim\\\. < lun\\\. < mar\\\. < mer\\\. < jeu\\\. < ... < sam\\..
```

## 2.4 Importation et exportation de données

Pour pouvoir analyser des données, il faut pourvoir les charger en mémoire. Il existe des fonctions qui permettent de les importer, depuis divers formats. Il est également possible d'enregistrer les données dans plusieurs formats. Il convient de préciser qu'un environnement de travail (working directory) est assigné à R. C'est dans ce répertoire, que l'on peut connaître grâce à l'instruction `getwd()` et modifier à l'aide de `setwd()`, que R ira chercher les fichiers à importer et enregistrer les données lors de l'exportation. Si on souhaite lire des fichiers qui sont hors de ce répertoire, il est nécessaire d'indiquer à R leur chemin.

R peut lire des données stockées dans des fichiers texte (ASCII), les données exportées depuis un tableau de type Excel au format **csv**. Des packages spécialisés permettent d'importer des sources de données enregistrées à partir d'autres logiciels statistiques (SPSS, Stata, SAS,...).

### 2.4.1 Données au format csv

Un fichier csv est un fichier texte dans lequel les valeurs des colonnes sont séparées par une virgule. Les fichiers au format **csv** peuvent être importées sous R à l'aide de la commande `read.csv()`, si le séparateur de champ est une virgule, ou `read.csv2()`, si le séparateur de champ est un point-virgule. On donne ci-dessous le contenu d'un fichier nommé **smp3** et enrégistré au format **csv**(seules les trois premières lignes du fichier sont affichées). Ici, le séparateur de champ est une virgule (dans ce cas, le séparateur décimal est obligatoirement un point).

```
"age","sexe","opinion"
18,"F","Pas du tout d'accord"
27,"F","Sans Opinion"
```

Pour importer ce fichier, on utilisera une instruction du type :

```
smp <- read.csv("D:/smp3.csv")
```

Pour écrire un fichier au format **csv**, on peut utiliser la fonction `write.csv()`. L'instruction ci-dessous

```
write.csv(smp, file="data.csv", row.names=FALSE)
```

permet d'écrire un data frame au format csv où **smp** est le nom du data frame, **file** est le chemin ou l'emplacement du fichier à écrire et **row.names** est une valeur logique qui permet d'indiquer si les noms des lignes doivent être écrits.

### 2.4.2 Données au format texte

R propose deux principales fonctions pour importer les données au format texte: `read.table()` et `scan()`. La fonction `read.table()` est très pratique lorsque les données sont déjà organisées dans un tableau dans le fichier à importer. Elle crée alors un data frame contenant les données. Par exemple l'instruction ci-dessous

permet de stocker les données du fichier **causals.txt** dans un data frame nommé **df** où certaines des options par défaut ont été renseignées explicitement:

```
df <- read.table("D:/causals.txt", header = TRUE, sep = "", dec = ".")
```

La fonction **read.table()** comprend de nombreux paramètres dont les plus utilisés sont décrits dans le tableau suivant:

Paramètre	Description
file	Le nom du fichier, doit être une chaîne de caractères. Il peut être précédé du chemin relatif ou absolu. Attention (utile pour les utilisateurs de Windows) le caractère “\” est proscrit, et doit être remplacé par “/” ou bien “\\”. À noter qu'il est possible de saisir une adresse web (URL) en guise de chaîne de caractère.
header	Valeur logique ( <b>header=FALSE</b> par défaut) indiquant si la première ligne contient les noms de variables.
sep	Le séparateur de champ dans le fichier (chaîne vide par défaut, ce qui est au final traduit par une espace comme séparation). Par exemple, <b>sep=";"</b> si les champs sont séparés par un point-virgule, ou encore <b>sep="\t"</b> s'ils sont séparés par une tabulation.
dec	Le caractère employé pour les décimales (par défaut, <b>dec=".,"</b> ).
row.names	Un vecteur contenant le nom des lignes (de type caractère), ou bien le numéro ou le nom d'une variable du fichier. En omettant ce paramètre, les lignes sont numérotées.
na.strings	Une chaîne de caractère (ou un vecteur de chaînes de caractères) indiquant la valeur des données manquantes (par défaut, <b>na.strings="NA"</b> ). Ces données manquantes seront converties en NA.
colClasses	Un vecteur de caractères indiquant les modes des colonnes.

La fonction **scan()** est beaucoup plus souple que **read.table()**. Son emploi est requis dès que les données ne sont pas organisées sous la forme d'un tableau. La nature des variables peut être spécifiée en renseignant le paramètre **what**. On retrouve la plupart des paramètres de la fonction **read.table()**.

Pour enregistrer des données au format texte depuis un data.frame, un vecteur ou une matrice, la fonction **write.table()** peut être utilisée. Par exemple, si le data.frame se nomme **smp2.csv**, l'instruction ressemble à :

```
write.table("smp2.csv", file = "nom_fichier.txt", sep = ";")
```

### 2.4.3 Données enregistrées à partir d'autres logiciels statistiques

Le package **foreign** (installé de base sous Windows, mais non chargé automatiquement en mémoire) contient des commandes permettant d'importer des sources de données enregistrées à partir d'autres logiciels statistiques. En particulier, les données au format SPSS, Stata, SAS et Minitab peuvent être chargées sous R respectivement

à l'aide des commandes **read.spss()**, **read.dta()**, **read.xport()** et **read.mtp()**.

Le package **R.matlab** permet d'importer des données enrégistrées à partir du logiciel Matlab à l'aide de la commande **readMat()**.

Les fichiers Excel au format **xls** ou **xlsx** peuvent être importés dans R grâce à la fonction **read.xls()** du package **gdata** et à la fonction **read.xlsx()** du package **xlsx**.

# Chapitre 3: Statistique descriptive avec R

## 3.1 Tableaux de données

On considère le fichier **cstat** contenu dans le répertoire de travail. C'est un data frame comportant 799 lignes (individus) et 26 colonnes (variables).

```
cstat<-read.csv("D:/c2stat.csv")
```

La structure générale des colonnes peut être obtenue grâce à la fonction **str()**. On peut également afficher les premières lignes (6 par défaut) du fichier chargé pour vérifier l'intégrité des données grâce à la fonction **head()**.

```
str(cstat)
```

```
## 'data.frame':    799 obs. of  26 variables:  
## $ age         : int  31 49 50 47 23 34 24 22 52 45 ...  
## $ prof        : Factor w/ 8 levels "agriculteur",...: 3 NA 7 6 8 6 3 2 6 6 ...  
## $ duree       : int  4 NA 5 NA 4 NA NA 5 4 NA ...  
## $ discip      : int  0 0 0 0 1 0 0 0 1 0 ...  
## $ n.enfant    : int  2 7 2 0 1 3 5 2 1 2 ...  
## $ n.fratrie   : int  4 3 2 6 6 2 3 9 12 5 ...  
## $ ecole       : int  1 2 2 1 1 2 1 2 1 2 ...  
## $ separation  : int  0 1 0 1 1 0 1 0 1 0 ...  
## $ juge.enfant : int  0 0 0 0 NA 0 1 0 1 0 ...  
## $ place       : int  0 0 0 1 1 0 1 0 0 0 ...  
## $ abus        : int  0 0 0 0 0 0 0 0 1 1 ...  
## $ grav.cons   : int  1 2 2 1 2 1 5 1 5 5 ...  
## $ dep.cons    : int  0 0 0 0 1 0 1 0 1 0 ...  
## $ ago.cons    : int  1 0 0 0 0 0 0 0 0 0 ...  
## $ ptsd.cons   : int  0 0 0 0 0 0 0 0 0 0 ...  
## $ alc.cons    : int  0 0 0 0 0 0 0 0 1 1 ...  
## $ subst.cons  : int  0 0 0 0 0 0 1 0 1 0 ...  
## $ scz.cons    : int  0 0 0 0 0 0 0 0 0 0 ...  
## $ char        : int  1 1 1 1 1 1 1 1 4 1 ...  
## $ rs          : int  2 2 2 2 2 1 3 2 3 2 ...  
## $ ed          : int  1 2 3 2 2 2 3 2 3 2 ...  
## $ dr          : int  1 1 2 2 2 1 2 2 1 2 ...  
## $ suicide.s   : int  0 0 0 1 0 0 3 0 4 0 ...  
## $ suicide.hr  : int  0 0 0 0 0 0 1 0 1 0 ...  
## $ suicide.past: int  0 0 0 0 1 0 1 0 1 0 ...  
## $ dur.interv  : int  NA 70 NA 105 NA NA 105 84 78 60 ...
```

```
head(cstat)
```

```
##   age           prof duree discip n.enfant n.fratrie ecole separation  
## 1  31           autre   4     0     2      4     1      0  
## 2  49           <NA>  NA     0     7      3     2      1  
## 3  50  prof.intermediaire   5     0     2      2     2      0  
## 4  47           ouvrier  NA     0     0      6     1      1  
## 5  23           sans emploi   4     1     1      6     1      1  
## 6  34           ouvrier  NA     0     3      2     2      0  
##   juge.enfant place abus grav.cons dep.cons ago.cons ptsd.cons alc.cons  
## 1           0     0     0      1      0      1      0      0  
## 2           0     0     0      2      0      0      0      0  
## 3           0     0     0      2      0      0      0      0
```

```

## 4      0   1   0     1   0   0   0   0
## 5      NA  1   0     2   1   0   0   0
## 6      0   0   0     1   0   0   0   0
##   subst.cons scz.cons char rs ed dr suicide.s suicide.hr suicide.past
## 1      0       0    1   2   1   1      0      0      0
## 2      0       0    1   2   2   1      0      0      0
## 3      0       0    1   2   3   2      0      0      0
## 4      0       0    1   2   2   2      1      0      0
## 5      0       0    1   2   2   2      0      0      1
## 6      0       0    1   1   2   1      0      0      0
##   dur.interv
## 1      NA
## 2      70
## 3      NA
## 4      105
## 5      NA
## 6      NA

```

On peut accéder aux variables **prof** et **n.enfant** grâce aux instructions **cstat\$prof** et **cstat\$n.enfant**. On peut aussi créer deux nouvelles variables nommées **prof** et **n.enfant** grâce à l'affectation:

```
prof<-cstat$prof; n.enfant<-cstat$n.enfant
```

### 3.1.1 Tableaux des effectifs et des fréquences

Pour construire des tableaux d'effectifs, on peut utiliser les fonctions suivantes: **table()** ou **ftable()**.

```
table(prof)
```

```

## prof
##      agriculteur          artisan          autre
##                 6              90              31
##      cadre            employe        ouvrier
##                 24             135             227
## prof.intermediaire      sans emploi
##                 58             222

```

```
ftable(n.enfant)
```

```

## n.enfant  0   1   2   3   4   5   6   7   8   9   10  11  13
## 
##      214 220 125 101  55  31   7   7   7   2   2   1   1

```

A partir du tableau des effectifs, on peut construire le tableau des fréquences grâce à la fonction **prop.table()** en sauvegardant le tableau des effectifs dans une variable.

```
tdeffprof<-table(prof); tdeffenfant<-table(n.enfant)
```

```
prop.table(tdeffprof)
```

```

## prof
##      agriculteur          artisan          autre
##      0.007566204      0.113493064      0.039092055
##      cadre            employe        ouvrier
##      0.030264817      0.170239596      0.286254729
## prof.intermediaire      sans emploi
##      0.073139975      0.279949559

```

```

prop.table(tdeffenfant)

## n.enfant
##      0       1       2       3       4       5
## 0.276843467 0.284605433 0.161707633 0.130659767 0.071151358 0.040103493
##      6       7       8       9      10      11
## 0.009055627 0.009055627 0.009055627 0.002587322 0.002587322 0.001293661
##      13
## 0.001293661

```

Notons également qu'il est possible de limiter l'affichage des décimales à l'aide de la fonction **round()**, en précisant le nombre de décimales à afficher en deuxième argument.

```
round(prop.table(tdeffprof), 3)
```

```

## prof
## agriculteur      artisan      autre
##      0.008      0.113      0.039
## cadre           employe      ouvrier
##      0.030      0.170      0.286
## prof.intermediaire sans emploi
##      0.073      0.280

```

```
round(prop.table(tdeffenfant), 3)
```

```

## n.enfant
##      0       1       2       3       4       5       6       7       8       9       10      11
## 0.277 0.285 0.162 0.131 0.071 0.040 0.009 0.009 0.009 0.003 0.003 0.001
##      13
## 0.001

```

Il est parfois intéressant de représenter un tableau de données individuelles (ou tableau de données brutes), récoltées sur une ou plusieurs variables quantitatives, sous une forme plus condensée. On utilise pour cela un tableau de données regroupées en classes. Par exemple, avec la variable **age**, il serait tout à fait possible de créer une variable **age.cat** à 4 classes d'effectifs à peu près équilibrés à l'aide de la commande **cut()**. Avec cette commande, il est important de préciser l'option **include.lowest = TRUE** pour ne pas oublier d'inclure l'observation dont l'âge vaut l'âge minimal, puisque par défaut les intervalles ont des bornes ouvertes (c'est-à-dire n'incluant pas) à gauche.

```
age.cat <- cut(cstat$age, breaks = c(19, 25, 35, 45, 83),
include.lowest = TRUE)
```

On peut déterminer le tableau des effectifs:

```
tdeffagecat<-table(age.cat)
tdeffagecat
```

```

## age.cat
## [19,25] (25,35] (35,45] (45,83]
##      136      223      207      231

```

On peut également déterminer le tableau des fréquences:

```
prop.table(tdeffagecat)
```

```

## age.cat
## [19,25] (25,35] (35,45] (45,83]
## 0.1706399 0.2797992 0.2597240 0.2898369

```

### Remarque

Par défaut, R regroupe les données dans des intervalles de la forme: ]a,b]. Si on souhaite avoir des intervalles de la forme: [a,b[, il faut ajouter le paramètre **right=FALSE**.

```
age.catbis <- cut(cstat$age, breaks = c(19, 25, 35, 45, 83), right=FALSE,
include.lowest = TRUE)
tdeffagecatbis<-table(age.catbis)
tdeffagecatbis

## age.catbis
## [19,25) [25,35) [35,45) [45,83]
##     114      229      206      248
```

### 3.1.2 Tableaux croisant deux variables

Lorsque l'on dispose du tableau des données individuelles des deux variables, on peut utiliser la fonction **table()** pour obtenir le tableau de contingence observé du couple.

Considérons le data frame **IMC2** ci-dessous:

```
IMC2<-data.frame(sexe = c("H","F","H","F","H","F", "H", "F"),
taille = c(1.83, 1.76, 1.83, 1.60, 1.90, 1.66, 1.76, 1.83),
poids = c(66, 58, 66, 48, 75, 55,66, 48),
row.names = c("Rémy", "Lol", "Pierre", "Domi", "Ben", "Cécile", "Paul", "Dan"))
IMC2

##       sexe taille poids
## Rémy     H   1.83   66
## Lol      F   1.76   58
## Pierre   H   1.83   66
## Domi    F   1.60   48
## Ben      H   1.90   75
## Cécile   F   1.66   55
## Paul     H   1.76   66
## Dan      F   1.83   48
```

### Tableaux de contingence

Le tableau de contingence des variables **sexe** et **poids** (encore appelé tri croisé en effectifs) est donné par:

```
sexe<-IMC2$sexe
poids<-IMC2$poids

tablecont<-table(sexe, poids)
tablecont

##       poids
## sexe 48 55 58 66 75
##   F   2   1   1   0   0
##   H   0   0   0   3   1
```

On peut ajouter les marges à ce tableau en utilisant la fonction **addmargins()**.

```
table.cont.marg <- addmargins(tablecont,FUN=sum,quiet=TRUE)
table.cont.marg

##       poids
## sexe 48 55 58 66 75 sum
##   F   2   1   1   0   0   4
##   H   0   0   0   3   1   4
```

```
##   sum  2  1  1  3  1  8
```

Dans le tableau précédent, on peut remplacer **sum** par **Total** en définissant la fonction Total de la manière suivante:

```
Total<-sum
```

Ensuite, on remplace **sum** par **Total** dans la fonction **addmargins()**.

```
table.cont.marg1 <- addmargins(tablecont,FUN=Total,quiet=TRUE)
table.cont.marg1
```

```
##      poids
## sexe    48 55 58 66 75 Total
##   F      2  1  1  0  0     4
##   H      0  0  0  3  1     4
##   Total  2  1  1  3  1     8
```

### Distribution conjointe

Le tableau de la distribution conjointe (encore appelé tri croisé en fréquences relatives) du couple (sexe, poids) s'obtient à partir du tableau de contingence **tablecont** précédent.

```
tableconfreq<-prop.table(tablecont)
tableconfreq
```

```
##      poids
## sexe    48    55    58    66    75
##   F  0.250 0.125 0.125 0.000 0.000
##   H  0.000 0.000 0.000 0.375 0.125
```

Si l'on veut aussi obtenir les marges, on peut utiliser l'instruction suivante :

```
table.cont.marg.freq <- addmargins(tableconfreq,FUN=Total,quiet=TRUE)
table.cont.marg.freq
```

```
##      poids
## sexe    48    55    58    66    75 Total
##   F  0.250 0.125 0.125 0.000 0.000 0.500
##   H  0.000 0.000 0.000 0.375 0.125 0.500
##   Total 0.250 0.125 0.125 0.375 0.125 1.000
```

### Distributions marginales

L'obtention des marges d'une table de distribution ou d'une table de contingence s'obtient au moyen de la fonction **margin.table()**.

```
margin.table(tableconfreq,1)
```

```
## sexe
##   F   H
## 0.5 0.5
```

```
margin.table(tableconfreq,2)
```

```
## poids
##    48    55    58    66    75
## 0.250 0.125 0.125 0.375 0.125
```

### Distributions conditionnelles

Les tableaux des distributions conditionnelles s'obtiennent au moyen de la fonction **prop.table()** à partir de la table de contingence.

La distribution conditionnelle de la variable **poids** sachant les valeurs de **sexe** est donnée par:

```
prop.table(tablecont, 1) # fréquences lignes

##      poids
## sexe   48   55   58   66   75
##   F  0.50 0.25 0.25 0.00 0.00
##   H  0.00 0.00 0.00 0.75 0.25
```

La distribution conditionnelle de la variable **sex**e sachant les valeurs de **poids** est donnée par:

```
prop.table(tablecont, 2) #fréquences colonnes
```

```
##      poids
## sexe 48 55 58 66 75
##   F  1  1  1  0  0
##   H  0  0  0  1  1
```

## 3.2 Caractéristiques d'une série statistique

On considère à nouveau le data frame **cstat** comportant 799 lignes (individus) et 26 colonnes (variables).

### 3.2.1 Caractéristiques de position

#### Le mode

Pour obtenir le mode d'une variable, on commence par construire le tableau des effectifs puis on utilise les fonctions **names** et **which** pour extraire la modalité (ou les modalités) ayant le plus grand effectif.

Pour la variable **age**, on a:

```
teffage<-table(cstat$age)
modeage<-names(teffage)[which(teffage==max(teffage))]
modeage
```

```
## [1] "24" "26"
```

On peut aussi utiliser l'instruction:

```
teffage<-table(cstat$age)
modeage<-names(teffage)[max(teffage)==teffage]
modeage
```

```
## [1] "24" "26"
```

Pour la variable **prof**, on a:

```
teffprof<-table(cstat$prof)
modeprof<-names(teffprof)[which(teffprof==max(teffprof))]
modeprof
```

```
## [1] "ouvrier"
```

#### La médiane

Pour calculer la médiane d'une variable quantitative, on utilise la fonction **median()**.  
Lorsque la variable contient une valeur manquante, le résultat affiché est **NA**.

```
median(cstat$age)
```

```
## [1] NA
```

Pour contourner ce problème, on peut retirer toutes les valeurs manquantes grâce à la fonction **na.omit()** avant de calculer la médiane.

```
age.NA<-na.omit(cstat$age)  
median(age.NA)
```

```
## [1] 37
```

On peut aussi calculer directement la médiane en utilisant le paramètre **na.rm=TRUE** de la fonction **median()** de la manière suivante:

```
median(cstat$age, na.rm=TRUE)
```

```
## [1] 37
```

### La moyenne

Pour calculer la moyenne d'une variable quantitative, on utilise la fonction **mean()**.

```
mean(cstat$age, na.rm=TRUE)
```

```
## [1] 38.89962
```

### Les fractiles

Pour obtenir les différents fractiles, on utilise la fonction **quantile()**.

Pour les trois **quartiles Q1, Q2 et Q3**, on a:

```
quantile(cstat$age, probs=c(0.25,0.5,0.75), na.rm=TRUE)
```

```
## 25% 50% 75%  
## 28 37 48
```

On peut aussi calculer un quartile particulier(exemple Q3) de la manière suivante

```
quantile(cstat$age, probs=0.75, na.rm=TRUE)
```

```
## 75%  
## 48
```

Pour les **déciles**, on a:

```
quantile(cstat$age, probs=1:10/10, na.rm=TRUE)
```

```
## 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%  
## 23 26 30 33 37 41 45 50 57 83
```

On peut aussi calculer seulement le premier décile et le neuvième décile:

```
quantile(cstat$age, probs=c(0.1,0.9), na.rm=TRUE)
```

```
## 10% 90%  
## 23 57
```

Il est également possible d'utiliser la commande **summary()** pour obtenir la plupart de ces indicateurs concernant la distribution d'une variable numérique, ou un tableau d'effectifs dans le cas d'une variable qualitative et le nombre de valeurs manquantes de la variable.

```
summary(cstat$age)
```

```
##   Min. 1st Qu. Median   Mean 3rd Qu.   Max.   NA's  
## 19.0   28.0   37.0   38.9   48.0   83.0       2
```

```
summary(cstat$prof)
```

```

##      agriculteur          artisan          autre
##      6                  90                 31
##      cadre            employe        ouvrier
##      24                  135                227
## prof.intermediaire    sans emploi     NA's
##                      58                  222                 6

```

### Remarque

Plus généralement, la fonction **summary()** permet d'obtenir un résumé détaillé de la distribution de chaque variable d'un data frame.

```
summary(cstat)
```

```

##      age           prof       duree      discip
## Min.   :19.0   ouvrier    :227   Min.   :1.000   Min.   :0.000
## 1st Qu.:28.0   sans emploi:222   1st Qu.:4.000   1st Qu.:0.000
## Median :37.0   employe     :135   Median :5.000   Median :0.000
## Mean   :38.9   artisan     : 90   Mean   :4.302   Mean   :0.232
## 3rd Qu.:48.0   prof.intermediaire: 58   3rd Qu.:5.000   3rd Qu.:0.000
## Max.   :83.0   (Other)     : 61   Max.   :5.000   Max.   :1.000
## NA's    :2     NA's       : 6   NA's   :223   NA's   :6
##      n.enfant      n.fratrie      ecole      separation
## Min.   : 0.000   Min.   : 0.000   Min.   :1.000   Min.   :0.0000
## 1st Qu.: 0.000   1st Qu.: 2.000   1st Qu.:1.000   1st Qu.:0.0000
## Median : 1.000   Median : 3.000   Median :2.000   Median :0.0000
## Mean   : 1.755   Mean   : 4.287   Mean   :1.866   Mean   :0.4226
## 3rd Qu.: 3.000   3rd Qu.: 6.000   3rd Qu.:2.000   3rd Qu.:1.0000
## Max.   :13.000   Max.   :21.000   Max.   :5.000   Max.   :1.0000
## NA's    :26      NA's       : 5   NA's   :11
##      juge.enfant      place      abus      grav.cons
## Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :1.000
## 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:2.000
## Median :0.0000   Median :0.0000   Median :0.0000   Median :4.000
## Mean   :0.2771   Mean   :0.2285   Mean   :0.2778   Mean   :3.643
## 3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:5.000
## Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :7.000
## NA's    :5       NA's       : 7   NA's   :4
##      dep.cons      ago.cons      ptsd.cons      alc.cons
## Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
## 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000
## Median :0.0000   Median :0.0000   Median :0.0000   Median :0.0000
## Mean   :0.3967   Mean   :0.1665   Mean   :0.2165   Mean   :0.1865
## 3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.0000
## Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##
##      subst.cons      scz.cons      char      rs
## Min.   :0.0000   Min.   :0.0000   Min.   :1.000   Min.   :1.000
## 1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:1.000   1st Qu.:1.000
## Median :0.0000   Median :0.0000   Median :1.000   Median :2.000
## Mean   :0.2653   Mean   :0.0826   Mean   :1.512   Mean   :2.057
## 3rd Qu.:1.0000   3rd Qu.:0.0000   3rd Qu.:2.000   3rd Qu.:3.000
## Max.   :1.0000   Max.   :1.0000   Max.   :4.000   Max.   :3.000
##
##      ed          dr      suicide.s      suicide.hr
## Min.   :1.000   Min.   :1.000   Min.   :0.0000   Min.   :0.0000

```

```

## 1st Qu.:1.000 1st Qu.:1.000 1st Qu.:0.0000 1st Qu.:0.0000
## Median :2.000 Median :2.000 Median :0.0000 Median :0.0000
## Mean   :1.866 Mean   :2.153 Mean   :0.7942 Mean   :0.2013
## 3rd Qu.:3.000 3rd Qu.:3.000 3rd Qu.:1.0000 3rd Qu.:0.0000
## Max.   :3.000 Max.   :3.000 Max.   :5.0000 Max.   :1.0000
## NA's    :107  NA's    :111  NA's    :41      NA's    :39
## suicide.past     dur.interv
## Min.   :0.0000  Min.   : 0.00
## 1st Qu.:0.0000 1st Qu.: 48.00
## Median :0.0000  Median : 60.00
## Mean   :0.2841  Mean   : 61.89
## 3rd Qu.:1.0000 3rd Qu.: 75.00
## Max.   :1.0000  Max.   :120.00
## NA's    :14     NA's    :50

```

### 3.2.2 Caractéristiques de dispersion

#### La variance

L'variance d'une variable quantitative s'obtient grâce à la fonction **var()**.

```
var(cstat$age, na.rm=TRUE)
```

```
## [1] 176.3844
```

#### Ecart-type

L'écart-type d'une variable quantitative s'obtient grâce à la fonction **sd()**.

```
sd(cstat$age, na.rm=TRUE)
```

```
## [1] 13.28098
```

#### Etendue

L'étendue d'une variable quantitative s'obtient grâce à la fonction **diff(range())**.

```
diff(range(cstat$age, na.rm=TRUE))
```

```
## [1] 64
```

On peut aussi faire la différence entre le maximum et le minimum de la variable.

```
max(cstat$age, na.rm=TRUE)-min(cstat$age, na.rm=TRUE)
```

```
## [1] 64
```

#### Ecart interquartile

L'écart interquartile d'une variable quantitative s'obtient grâce à la fonction **IQR()**.

```
IQR(cstat$age, na.rm=TRUE)
```

```
## [1] 20
```

#### Coefficient de variation

Le coefficient de variation s'obtient en divisant l'écart type par la moyenne.

```
coefvar<-sd(IMC$taille)/mean(IMC$taille)
coefvar
```

```
## [1] 0.06406254
```

#### Ecart moyen

L'écart moyen d'une variable quantitative x est donné par l'instruction **mean(abs(x-mean(x)))**.

```
ecmoy<-mean(abs(IMC$taille-mean(IMC$taille)))
ecmoy

## [1] 0.08833333
```

### 3.2.3 Caractéristiques de forme d'une distribution

Ces caractéristiques peuvent être calculés uniquement pour des variables quantitatives. Nous pouvons mentionner les coefficients d'asymétrie (skewness) obtenu par la fonction **skewness()** et d'aplatissement (kurtosis) obtenu par la fonction **kurtosis()** du package **moments**.

```
require(moments)

## Loading required package: moments
skewness(IMC$taille)

## [1] -0.3218226
require(moments)
kurtosis(IMC$taille)

## [1] 1.75981
```

### 3.2.4 Mesures de liaison entre deux variables

#### 3.2.4.1 Mesures de liaison entre deux variables quantitatives

Considérons les variables **taille** et **poids** du data frame **IMC**.

##### Covariance

La covariance entre deux variables quantitatives s'obtient à l'aide de la fonction **cov()**.

```
cov(IMC$taille, IMC$poids)

## [1] 1.071

Coefficient de correlation de Pearson

Le coefficient de correlation de Pearson entre deux variables quantitatives s'obtient à l'aide de la fonction cor().
```

**cor**(IMC\$taille, IMC\$poids)

```
## [1] 0.9793294
```

#### 3.2.4.2 Mesures de liaison entre une variable quantitative et une variable qualitative

Quand on parle de comparaison entre une variable quantitative et une variable qualitative, on veut en général savoir si la distribution des valeurs de la variable quantitative est la même selon les modalités de la variable qualitative.

Considérons les variables **sexé** et **poids** du data frame **IMC2**. On peut, par exemple, se demander si le poids moyen des femmes est différent de celui des hommes?

On peut calculer la moyenne des poids des deux groupes en utilisant la fonction **tapply**.

```
tapply(IMC2$poids, IMC2$sexé, mean)

##      F      H
## 52.25 68.25
```

### Le rapport de corrélation

Pour comparer une variable quantitative et une variable qualitative, on peut aussi calculer le rapport de corrélation.

Le rapport de corrélation  $\eta_{Y/X}^2$  indique dans quelle mesure les variations d'une variable quantitative  $Y$  sont expliquées par les modalités d'une variable qualitative  $X$  à  $p$  modalités.

Pour calculer le rapport de corrélation, on peut utiliser l'instruction suivante:

```
summary(lm(IMC2$poids~IMC2$sexe))$r.squared
```

```
## [1] 0.7882987
```

### 3.2.4.3 Mesures de liaison entre deux variables qualitatives

Pour comparer deux variables qualitatives, on utilise le tableau de contingence (voir la section 3.1.2). Pour mesurer la dépendance des deux variables qualitatives, on utilise le test du Khi-deux (voir chapitre 4).

## 3.3 Représentations graphiques d'une série statistique

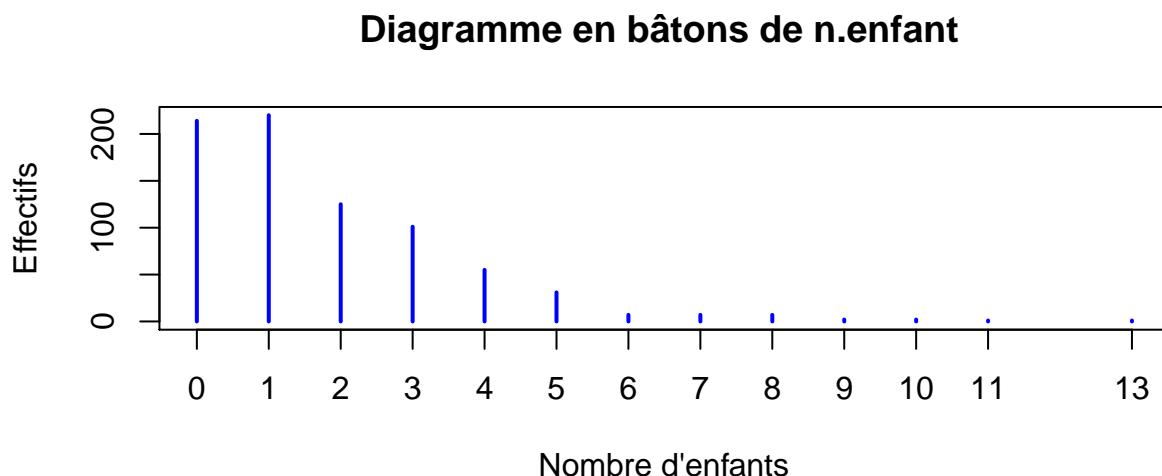
### 3.3.1 Graphiques pour les variables quantitatives

#### Diagramme en bâtons

Le diagramme en bâtons s'obtient grâce à la fonction `plot()`. On commence par construire le tableau des effectifs (ou des fréquences).

Le diagramme en bâtons de la variable `n.enfant` du data framme `cstat` est donné par:

```
teffenf<-table(cstat$n.enfant)
plot(teffenf, main="Diagramme en bâtons de n.enfant ", ylab="Effectifs", xlab="Nombre d'enfants",
     col="blue")
```



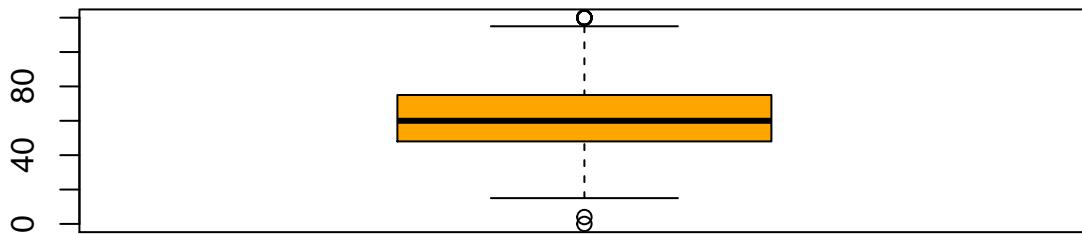
#### Boîte à moustaches

Pour tracer un diagramme en boîte à moustaches, il faut utiliser la fonction `boxplot()`.

Le diagramme en boîte à moustaches de la variable `dur.interv` est donné ci-dessous.

```
boxplot(cstat$dur.interv, main="boîte à moustaches de dur.interv", col="orange")
```

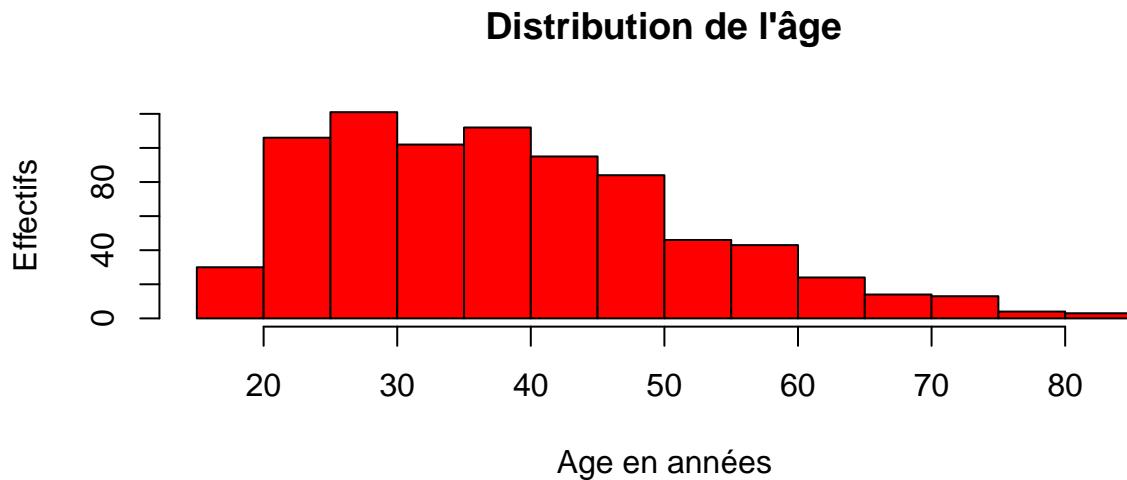
## boîte à moustaches de dur.interv



### Histogramme

L'histogramme s'obtient grâce à la fonction `hist()`. l'histogramme de la variable `age` avec des classes de même amplitude est donné par:

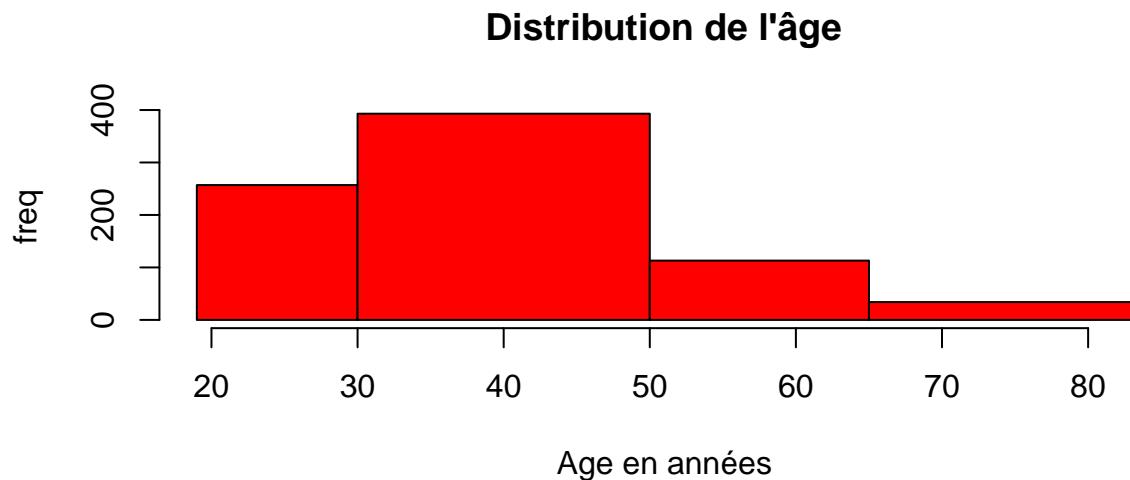
```
hist(cstat$age, main="Distribution de l'âge", ylab="Effectifs", xlab="Age en années", col="red")
```



On peut aussi représenter l'histogramme de la variable `age` avec des classes d'amplitudes différentes.

```
hist(cstat$age, freq=TRUE, main="Distribution de l'âge", xlab="Age en années", ylab="freq",  
breaks=c(19,30,50,65,83), col="red")
```

```
## Warning in plot.histogram(r, freq = freq1, col = col, border = border,  
## angle = angle, : the AREAS in the plot are wrong -- rather use 'freq =  
## FALSE'
```



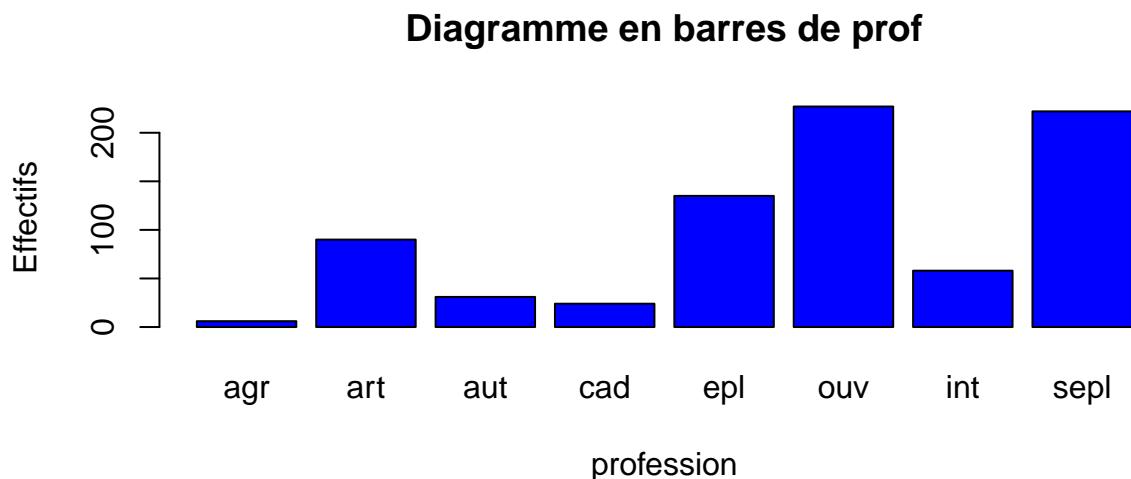
### 3.3.2 Graphiques pour les variables qualitatives

#### Diagramme en barres

Le diagramme en barres s'obtient grâce à la fonction **barplot()**. On commence par construire le tableau des effectifs (ou des fréquences).

Le diagramme en barres de la variable **prof** du data framme **cstat** est donné par:

```
teffprof<-table(cstat$prof)
profess<-c("agr", "art", "aut", "cad", "epl", "ouv", "int", "sepl")
barplot(teffprof, main="Diagramme en barres de prof", names.arg = profess, ylab="Effectifs",
        xlab="profession", col="blue")
```



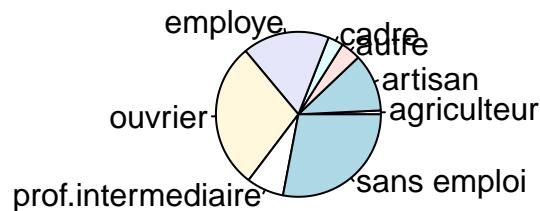
#### Diagramme circulaire

Le diagramme circulaire s'obtient grâce à la fonction **pie()** en utilisant le tableau des effectifs (ou des fréquences).

Le diagramme circulaire de la variable **prof** est donné par:

```
pie(teffprof, main="Diagramme circulaire de prof")
```

## Diagramme circulaire de prof



### 3.3.3 Graphiques pour le croisement de deux variables

#### 3.3.3.1 Graphiques pour le croisement de deux variables quantitatives

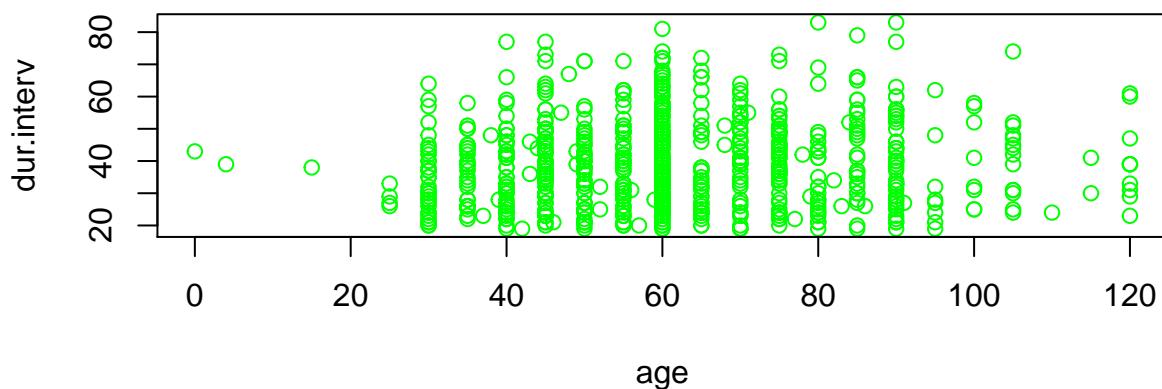
##### Nuage de points

Le nuage de points entre deux variables quantitatives est obtenu grâce à la fonction **plot()**.

Le nuage de points de la variable **dur.interv** en fonction de la variable **age** est représenté ci-dessous.

```
plot(cstat$dur.interv, cstat$age, main="Durée d'intervention en fonction de l'âge",
     xlab="age", ylab="dur.interv", col="green")
```

## Durée d'intervention en fonction de l'âge

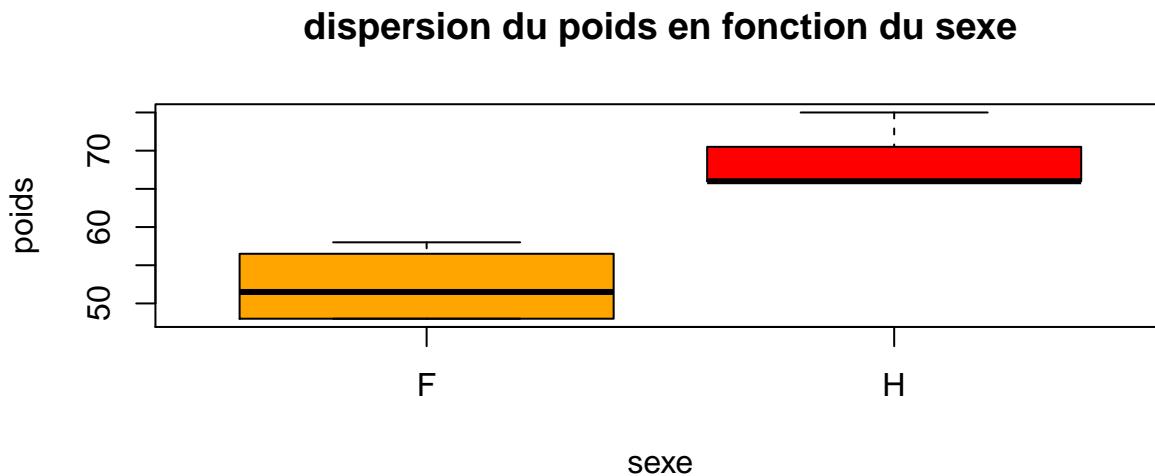


#### 3.3.3.2 Graphiques pour le croisement d'une variable quantitative et d'une variable qualitative

Pour comparer graphiquement une variable quantitative et une variable qualitative, on peut construire des diagrammes en boîte à moustaches de la variable quantitative en fonction de chaque modalité de la variable qualitative grâce à la fonction **boxplot()**.

Considérons les variables **sex**e et **poids** du data frame **IMC2**.

```
couleur<-c("orange", "red")
boxplot(IMC2$poids~IMC2$sex, col=couleur, main="dispersion du poids en fonction du sexe",
       xlab="sex", ylab="poids")
```



### 3.3.3.3 Graphiques pour le croisement de deux variables qualitatives

Considérons les variables **sex**e et **niveau** du data frame **IMC3** ci-dessous:

```
IMC3<-data.frame(sexe = c("H", "F", "H", "F", "H", "F", "H", "F"),
                   taille = c(1.83, 1.76, 1.83, 1.60, 1.90, 1.66, 1.76, 1.83),
                   poids = c(66, 58, 66, 48, 75, 55, 66, 48),
                   niveau= c("BAC", "Licence", "BAC", "Master", "Master", "Licence", "Licence", "BAC"),
                   row.names = c("Rémy", "Lol", "Pierre", "Domi", "Ben", "Cécile", "Paul", "Dan"))
IMC3
```

```
##          sexe taille poids niveau
## Rémy      H    1.83   66     BAC
## Lol        F    1.76   58 Licence
## Pierre    H    1.83   66     BAC
## Domi      F    1.60   48 Master
## Ben        H    1.90   75 Master
## Cécile    F    1.66   55 Licence
## Paul       H    1.76   66 Licence
## Dan        F    1.83   48     BAC
```

Pour comparer graphiquement deux variables qualitatives, on peut construire des diagrammes en barres des deux variables grâce à la fonction **barplot()** en utilisant le tableau de contingence des variables.

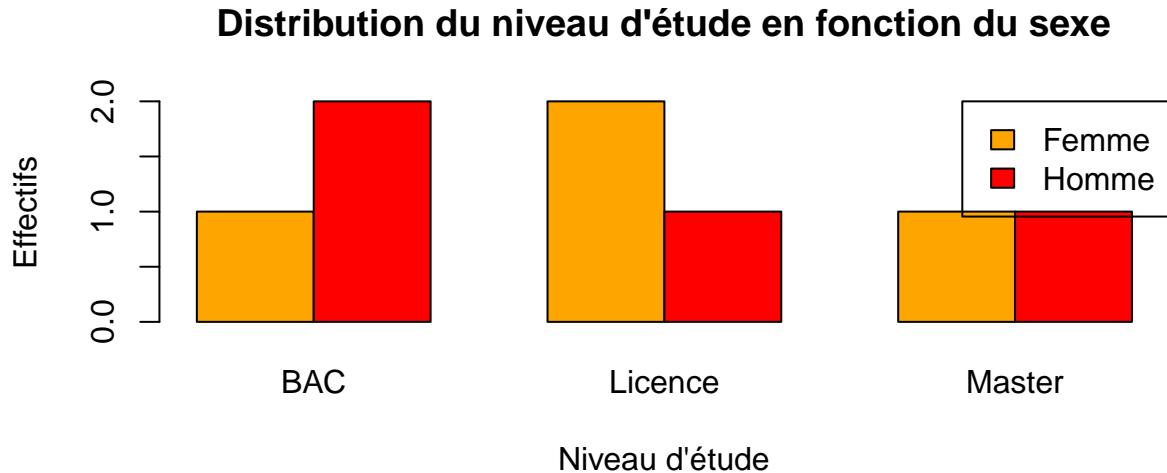
```
cont<-table(IMC3$sexe, IMC3$niveau)
cont
```

```
##
##      BAC Licence Master
```

```

##   F   1      2      1
##   H   2      1      1
colors<-c("orange", "red")
barplot(cont,beside=T,col=colors, main="Distribution du niveau d'étude en fonction du sexe",
xlab="Niveau d'étude",ylab="Effectifs")
legend("topright", xpd = TRUE, legend = c("Femme","Homme"),fill=colors)

```



L'option **beside=T** permet d'obtenir un diagramme en barre non-empilé.

## Chapitre 4: statistique inférentielle avec R

### 4.1 Lois de probabilité

Il existe dans R un large éventail de fonctions donnant directement accès aux caractéristiques de plusieurs lois de probabilité. Désignons par <<loi>>, la loi d'une certaine variable aléatoire X. Pour chaque loi, il existe quatre fonctions différentes :

1. `dloi(x,...)` calcule la fonction de densité de probabilité au point x (loi continue) ou la probabilité que la variable aléatoire X prenne la valeur x (loi discrète) ;
2. `ploi(x,...)` calcule la fonction de répartition au point x;
3. `qloi(a,...)` calcule le quantile d'ordre  $\alpha$ ;
4. `rloi(n,...)` retourne un échantillon de taille n de cette loi.

Les différentes lois de probabilité disponibles dans le système R de base, leur racine et le nom de leurs paramètres sont rassemblés dans le tableau ci-dessous.

Loi de probabilité	Racine dans R	Noms des paramètres
Bêta	<code>beta</code>	<code>shape1, shape2</code>
Binomiale	<code>binom</code>	<code>size, prob</code>
Binomiale négative	<code>nbinom</code>	<code>size, prob ou mu</code>
Cauchy	<code>cauchy</code>	<code>location, scale</code>
Exponentielle	<code>exp</code>	<code>rate</code>
F (Fisher)	<code>f</code>	<code>df1, df2</code>
Gamma	<code>gamma</code>	<code>shape, rate ou scale</code>
Géométrique	<code>geom</code>	<code>prob</code>
Hypergéométrique	<code>hyper</code>	<code>m, n, k</code>
Khi carré	<code>chisq</code>	<code>df</code>
Logistique	<code>logis</code>	<code>location, scale</code>
Log-normale	<code>lnorm</code>	<code>meanlog, sdlog</code>
Normale	<code>norm</code>	<code>mean, sd</code>
Poisson	<code>pois</code>	<code>lambda</code>
t (Student)	<code>t</code>	<code>df</code>
Uniforme	<code>unif</code>	<code>min, max</code>
Weibull	<code>weibull</code>	<code>shape, scale</code>
Wilcoxon	<code>wilcox</code>	<code>m, n</code>

Exemple d'une loi discrète: la loi de Poisson

L'instruction

```
dpois(18, 20)
```

```
## [1] 0.08439355
```

retourne la probabilité que la loi de Poisson de paramètre 20 prenne la valeur 18.

La commande

```
ppois(18, 20)
```

```
## [1] 0.3814219
```

retourne la fonction de répartition de la loi de Poisson de paramètre 20 au point 18 c'est-à-dire la probabilité que la variable aléatoire suivant la loi de Poisson de paramètre 20 prenne les valeurs inférieures ou égales à 18. Le quantile d'ordre 0.8 de loi de Poisson de paramètre 20 est donné par:

```
qpois(0.8, 20)
```

```
## [1] 24
```

Pour générer un échantillon de taille sept de la loi de Poisson de paramètre 20, on utilise l'instruction suivante:

```
rpois(7, 20)
```

```
## [1] 21 13 19 21 18 17 17
```

### Remarque

La fonction **sample** permet de simuler des nombres d'une distribution discrète quelconque.

Par exemple, pour simuler 20 nombres aléatoires compris entre 1 et 100 sans remise, utilisez la commande:

```
sample(1:100, 20)
```

```
## [1] 52 11 36 21 94 65 13 62 39 26 87 46 12 76 28 35 58 5 64 71
```

Il faut ajouter le paramètre **replace=TRUE** si on veut effectuer un tirage avec remise.

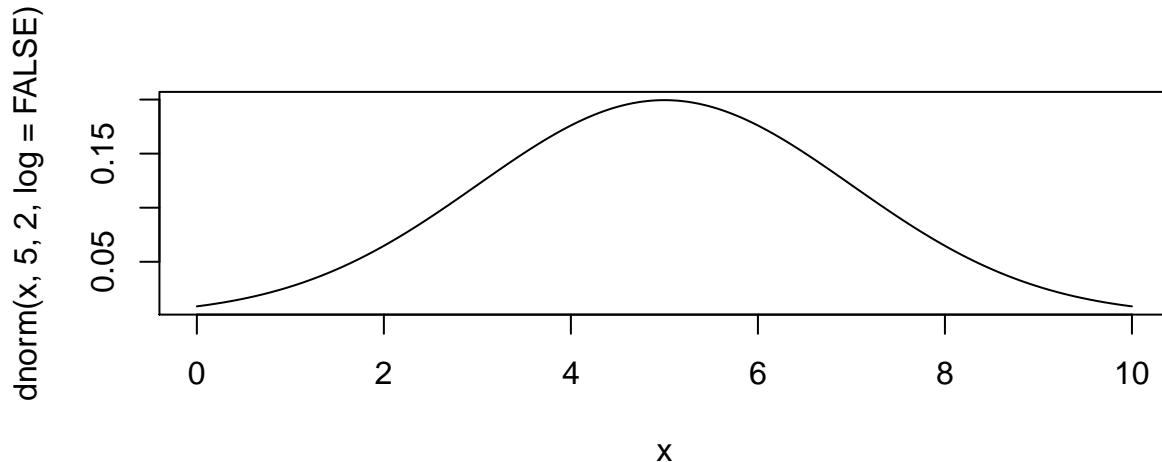
```
sample(1:100, 20, replace=TRUE)
```

```
## [1] 80 92 14 95 30 64 48 95 18 24 10 53 84 88 91 72 10 37 85 71
```

### Exemple d'une loi continue: la loi normale

L'instruction ci-dessous permet de représenter la fonction de densité de la loi normale de moyenne 5 et d'écart type 2 sur l'intervalle [0, 10].

```
x=seq(0, 10, 0.01)
plot(x, dnorm(x, 5, 2, log=FALSE), type="l")
```



Pour déterminer la valeur de la fonction de densité de la loi normale de moyenne 5 et d'écart type 2 au point 2.5, on utilise l'instruction suivante:

```
dnorm(2.5, mean=5, sd=2)
```

```
## [1] 0.09132454
```

### Remarque

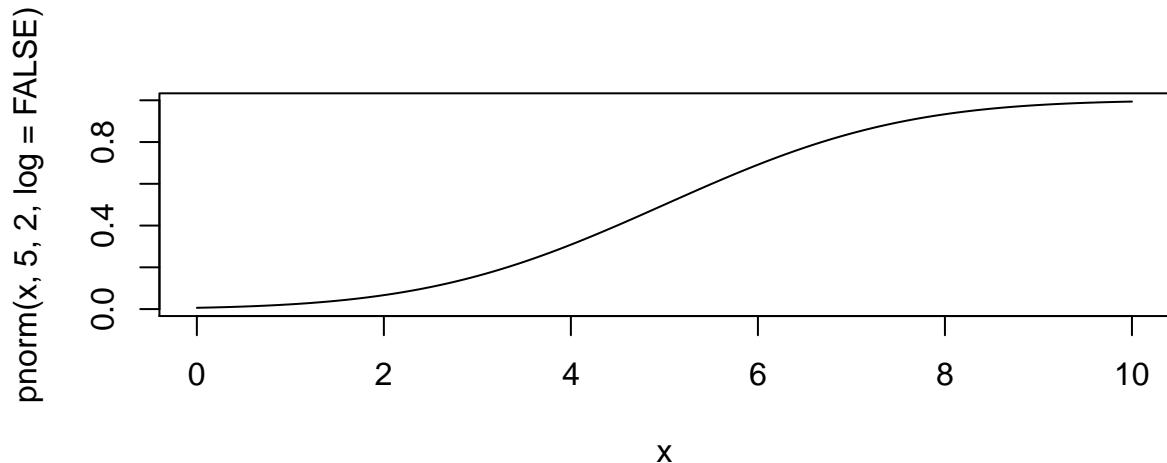
Pour la loi normale centrée réduite, il n'est pas nécessaire de préciser les paramètres(moyenne et écart type). Pour déterminer la valeur de la fonction de densité au point 2.5, on écrira simplement

```
dnorm(2.5)
```

```
## [1] 0.0175283
```

L'instruction ci-dessous permet de représenter la fonction de répartition de la loi normale de moyenne 5 et d'écart type 2 sur l'intervalle [0, 10].

```
x=seq(0, 10, 0.01)
plot(x, pnorm(x, 5, 2, log=FALSE), type="l")
```



L'instruction

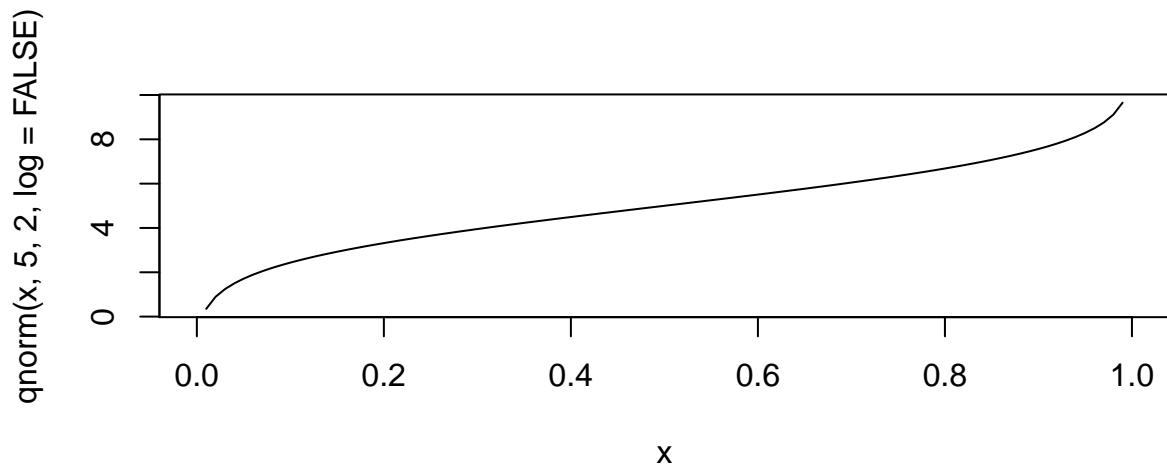
```
pnorm(2.5, 5, 2)
```

```
## [1] 0.1056498
```

retourne la fonction de répartition de la loi normale de moyenne 5 et d'écart type 2 au point 2.5 c'est-à-dire la probabilité que la variable aléatoire suivant la loi normale de moyenne 5 et d'écart type 2 prenne les valeurs inférieures ou égales à 2,5.

L'instruction ci-dessous permet de représenter la fonction quantile de la loi normale de moyenne 5 et d'écart type 2.

```
x=seq(0, 1, 0.01)
plot(x, qnorm(x, 5, 2, log=FALSE), type="l")
```



Le quantile d'ordre 0.95 de la loi normale de moyenne 5 et d'écart type 2 est donné par:

```
qnorm(0.95, mean=5, sd=2)
```

```
## [1] 8.289707
```

Pour générer un échantillon de taille 20 de la loi normale de moyenne 5 et d'écart type 2, on utilise l'instruction suivante:

```
rnorm(20, 5, 2)
```

```
## [1] 4.7673466 3.7917114 2.9390333 7.8265862 6.8090033 6.7060462 6.9158490
## [8] 4.3470101 4.1840481 3.6727096 1.2327889 7.4479645 0.5408598 4.2430616
## [15] 4.2965973 5.7405393 7.0892471 4.2878018 3.8149307 6.0671488
```

## 4.2 Estimation ponctuelle et par intervalle de confiance

### 4.2.1 Estimation ponctuelle de la moyenne et de la variance

Soit  $(X_1, \dots, X_n)$  un échantillon aléatoire de loi parente la loi de  $X$  de moyenne  $\mu$  et de variance  $\sigma^2$ .

#### 4.2.1.1 Estimation ponctuelle de la moyenne

Un estimateur ponctuelle de la moyenne  $\mu$  est donné par  $\hat{\mu} = \frac{1}{n} \sum_{i=1}^n X_i$ .  
 supposons que  $X$  suit la loi normale de moyenne  $\mu = 5$  (valeur supposée inconnue). Avec un échantillon de taille 10000, un estimateur de  $\mu$  est:

```
mu<-5
muchapeau<-mean(rnorm(10000,mean=mu))
muchapeau

## [1] 4.986779
```

#### 4.2.1.2 Estimation ponctuelle de la variance

Un estimateur ponctuelle de la variance  $\sigma^2$  est donné par  $\widehat{\sigma^2} = \frac{1}{n} \sum_{i=1}^n (X_i - \widehat{\mu})^2 = \frac{1}{n} \sum_{i=1}^n X_i^2 - (\frac{1}{n} \sum_{i=1}^n X_i)^2$ . supposons que X suit la loi normale de variance  $\sigma^2 = 3$  (valeur supposée inconnue). Avec un échantillon de taille 10000, un estimateur de  $\sigma^2$  est:

```
sigma2<-3
loiX<-rnorm(10000, sd=sqrt(sigma2))
sigma2chapeau<-mean(loiX^2)-(mean(loiX))^2
sigma2chapeau

## [1] 2.99573
```

### Remarque

L'estimateur précédent de la variance est biaisé. Un estimateur sans biais de la variance est l'estimateur corrigé  $\widehat{\sigma}_c^2 = \frac{1}{n-1} \sum_{i=1}^n (X_i - \widehat{\mu})^2$ .

## 4.2.2 Estimation par intervalle de confiance d'une moyenne, d'une variance et d'une proportion

### 4.2.2.1 Intervalle de confiance pour une moyenne

Un intervalle de confiance de niveau  $1 - \alpha$  de la moyenne  $\mu$  pour les grands échantillons ( $n > 30$ ) ou des petits échantillons avec hypothèse de normalité est donné par:

$I_c(\mu) = [\widehat{\mu} - t_{1-\alpha/2}^{n-1} \frac{\widehat{\sigma}}{\sqrt{n}}; \widehat{\mu} + t_{1-\alpha/2}^{n-1} \frac{\widehat{\sigma}}{\sqrt{n}}]$  où  $t_{1-\alpha/2}^{n-1}$  est le quantile d'ordre  $1 - \alpha/2$  de la loi de Student à  $n - 1$  degrés de liberté.

Pour obtenir cet intervalle de confiance, on utilise la fonction `t.test()`.

Dans le data frame `cstat`, un intervalle de confiance (de seuil  $\alpha = 0.05$ ) de l'âge moyen est:

```
t.test(cstat$age, conf.level=0.95)$conf.int

## [1] 37.97618 39.82307
## attr(),"conf.level")
## [1] 0.95
```

### 4.2.2.2 Intervalle de confiance pour une variance

Un intervalle de confiance de niveau  $1 - \alpha$  pour la variance  $\sigma^2$  d'une loi normale est donné par:  $I_c(\sigma^2) = [\frac{(n-1)\widehat{\sigma}^2}{q_{1-\alpha/2}^{n-1}}; \frac{(n-1)\widehat{\sigma}^2}{q_{\alpha/2}^{n-1}}]$  où  $q_{1-\alpha/2}^{n-1}$  est le quantile d'ordre  $1 - \alpha/2$  de la loi Khi-deux à  $n - 1$  degrés de liberté.

Pour obtenir cet intervalle de confiance, on utilise la commande `qchisq(alpha,n)` pour calculer  $q_\alpha^n$ .

Dans le data frame `cstat`, on peut déterminer la borne inférieure de l'intervalle de confiance de la variance (de seuil  $\alpha = 0.05$ ) de l'âge

```
age2<-na.omit(cstat$age)
n<-length(age2)
alpha<-0.05
(n-1)*var(age2)/qchisq(1-alpha/2,n-1)

## [1] 160.2618
```

et la borne supérieure de l'intervalle de confiance

```
(n-1)*var(age2)/qchisq(alpha/2,n-1)
```

```
## [1] 195.0816
```

Ainsi, l'intervalle de confiance de la variance est  $I_c(\sigma^2) = [160.2618; 195.0816]$ .

Dans le cas de grands échantillons sans hypothèse de normalité, on peut utiliser une approche asymptotique grâce au package **asympTest**.

```
require("asympTest")

## Loading required package: asympTest
asymp.test(age2, par="var", conf.level = 0.95)$conf

## [1] 159.2415 193.5272
## attr(),"conf.level")
## [1] 0.95
```

#### 4.2.2.3 Intervalle de confiance pour une proportion

Un intervalle de confiance asymptotique de niveau  $1 - \alpha$  pour la proportion inconnue  $p$  est donné par:  
 $Ic(p) = [\hat{p} - u_{1-\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}; \hat{p} + u_{1-\alpha/2} \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}]$  où  $u_{1-\alpha/2}$  est le quantile d'ordre  $1 - \alpha/2$  de la loi normale centrée réduite.

Pour obtenir cet intervalle, on peut utiliser la fonction **binom.approx()** du package **epitools**.

Dans le data frame **cstat**, un intervalle de confiance asymptotique (de seuil  $\alpha = 0.05$ ) de la proportion des **ouvriers** peut être obtenue de la manière suivante.

```
table(cstat$prof)

##
##      agriculteur      artisan      autre
##                 6             90            31
##      cadre      employe      ouvrier
##                 24            135           227
## prof.intermediaire      sans emploi
##                 58            222
require("epitools")

## Loading required package: epitools
binom.approx(227, 793)[c("lower", "upper")]

##
##      lower      upper
## 1 0.2547947 0.3177148
```

Dans le cas des petits échantillons, on peut déterminer de façon exacte l'intervalle de confiance grâce à la fonction **binom.test()** ou la fonction **binom.exact()** du package **epitools**.

```
binom.test(227, 793)$conf

## [1] 0.2550079 0.3191000
## attr(),"conf.level")
## [1] 0.95
```

### 4.3 Tests statistiques

#### 4.3.1 Test de normalité

On souhaite savoir si une variable aléatoire suit une loi normale. Pour cela, on utilise le test de Shapiro-Wilk. On teste l'hypothèse nulle  $H_0$ : “la variable suit une loi normale” contre l'hypothèse alternative  $H_1$ : “la variable ne suit pas une loi normale”. Pour effectuer ce test, on utilise la fonction **shapiro.test()**.

## Exemple

On mesure la taille du lobe frontal de 30 crabes. Les résultats sont contenus dans la variable **longueur** ci-dessous:

```
longueur<-c(12.6, 12.0, 20.9, 14.2, 16.2, 15.3, 10.4, 22.1, 19.8,
           15, 12.8, 20, 11.8, 20.6, 21.3, 11.7, 18, 9.1, 15,
           15.2, 15.1, 14.7, 13.3, 21.7, 15.4, 16.7, 15.6, 17.1,
           7.2, 12.6)
```

On peut maintenant effectuer le test grâce à la fonction **shapiro.test()**.

```
shapiro.test(longueur)
```

```
## 
## Shapiro-Wilk normality test
##
## data: longueur
## W = 0.96541, p-value = 0.4223
```

La p-value du test est égale à  $0.4223 > 0.05$ . On accepte l'hypothèse nulle (la taille du lobe frontal des crabes suit une loi normale) au risque 5% de se tromper.

## 4.3.2 Tests sur la variance

### 4.3.2.1 Comparaison de la variance théorique à une valeur de référence

Soit  $X$  une variable aléatoire suivant une loi normale de variance  $\sigma^2$ . On souhaite comparer cette variance à une valeur de référence  $\sigma_0^2$ . On considère les hypothèses

$H_0: \sigma^2 = \sigma_0^2$  et  $H_1: \sigma^2 \neq \sigma_0^2$  (ou  $H_0: \sigma^2 = \sigma_0^2$  et  $H_1: \sigma^2 < \sigma_0^2$  ou  $H_0: \sigma^2 = \sigma_0^2$  et  $H_1: \sigma^2 > \sigma_0^2$ ).

Sous  $H_0$  la statistique  $T = \frac{(n-1)\hat{\sigma}^2}{\sigma_0^2}$  suit la loi du Khi-deux avec  $n - 1$  degrés de liberté.

Pour effectuer un tel test, on peut utiliser la fonction **sigma2.test()** du package **sigma2tools**.

Sans hypothèse de normalité et pour de grands échantillons, vous pouvez utiliser la fonction **asymp.test(x,parameter="var",reference=10)**. Cette fonction se trouve dans le package **asympTest**.

#### Exemple

Une usine fabrique des boîtes de conserve de poids  $\mu$  avec une précision  $\sigma_0^2 = 10$ . On souhaite tester cette précision. Voici les poids d'une série de boîtes de conserve.

```
poids <- c(165.1,171.5,168.1,165.6,166.8,170,168.8,171.1,168.8,173.6,163.5,169.9,165.4,174.4,
          171.8,166,174.6,174.5,166.4,173.8,167.2,168.4,166.7,164.3,172.5,172.8,164.9,163.8,
          165.6,165.8,166.6,168.4,169.1,165.5)
```

```
poids
```

```
## [1] 165.1 171.5 168.1 165.6 166.8 170.0 168.8 171.1 168.8 173.6 163.5
## [12] 169.9 165.4 174.4 171.8 166.0 174.6 174.5 166.4 173.8 167.2 168.4
## [23] 166.7 164.3 172.5 172.8 164.9 163.8 165.6 165.8 166.6 168.4 169.1
## [34] 165.5
```

Les hypothèses de ce test sont  $H_0: \sigma^2 = 10$  et  $H_1: \sigma^2 \neq 10$ .

```
require(asympTest)
asymp.test(poids,parameter="var",reference=10)
```

```
## 
## One-sample asymptotic variance test
##
## data: poids
## statistic = 0.84126, p-value = 0.4002
```

```

## alternative hypothesis: true variance is not equal to 10
## 95 percent confidence interval:
##    7.900758 15.256479
## sample estimates:
## variance
## 11.57862

```

La p-value est de  $0.4002 > 0.05$ . Au seuil de 5%, on ne peut pas rejeter l'hypothèse  $H_0$ .

#### 4.3.2.2 Comparaison de deux variances

On considère deux variables quantitatives  $X_1$  et  $X_2$  (qui mesurent la même caractéristique, mais dans deux populations différentes). On suppose que  $X_1$  et  $X_2$  suivent des lois normales et que  $X_1$  a pour variance  $\sigma_1^2$  et  $X_2$  a pour variance  $\sigma_2^2$ . A partir des estimations calculées sur deux échantillons de tailles respectives  $n_1$  et  $n_2$  issus des deux populations, on veut comparer  $\sigma_1^2$  et  $\sigma_2^2$ .

Les hypothèses du test sont

$H_0: \sigma_1^2 = \sigma_2^2$  et  $H_1: \sigma_1^2 \neq \sigma_2^2$  (ou  $H_0: \sigma_1^2 = \sigma_2^2$  et  $H_1: \sigma_1^2 < \sigma_2^2$  ou  $H_0: \sigma_1^2 = \sigma_2^2$  et  $H_1: \sigma_1^2 > \sigma_2^2$ ).

Sous  $H_0$  la statistique  $T = \frac{\hat{\sigma}_1^2}{\hat{\sigma}_2^2}$  suit la loi de Fisher avec  $n_1 - 1$  et  $n_2 - 1$  degrés de liberté.

Pour effectuer un tel test, on peut utiliser la fonction `var.test()`.

#### Exemple

Dans le data frame `cstat`, on souhaite savoir s'il y a une différence significative de la variance de l'âge dans la population des **ouvriers** et dans la population des **sans emploi**.

```

cstatouv<-subset(cstat, prof=="ouvrier") #data frame contenant uniquement les ouvriers
ageouv<-cstatouv$age
cstatsemp<-subset(cstat, prof=="sans emploi") #data frame contenant uniquement les sans emploi
agesemp<-cstatsemp$age

var.test(ageouv, agesemp)

```

```

##
## F test to compare two variances
##
## data: ageouv and agesemp
## F = 0.68351, num df = 226, denom df = 221, p-value = 0.004583
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
## 0.5253975 0.8888570
## sample estimates:
## ratio of variances
## 0.6835102

```

La p-value est de  $0.004583 < 0.05$ . Au seuil de 5%, on accepte l'hypothèse  $H_1$ . On peut conclure qu'il y a une différence significative de la variance de l'âge des ouvriers et de la variance de l'âge des sans emploi, au risque 5% de se tromper.

#### 4.3.3 Tests de moyenne

##### 4.3.3.1 Comparaison de la moyenne théorique à une valeur de référence

Soit  $X$  une variable aléatoire de moyenne  $\mu$  et de variance  $\sigma^2$ . On souhaite comparer cette moyenne à une valeur de référence  $\mu_0$ . On considère les hypothèses

$H_0: \mu = \mu_0$  et  $H_1: \mu \neq \mu_0$  (ou  $H_0: \mu = \mu_0$  et  $H_1: \mu < \mu_0$  ou  $H_0: \mu = \mu_0$  et  $H_1: \mu > \mu_0$ ).

Sous  $H_0$  la statistique  $T = \sqrt{n}(\frac{\bar{X} - \mu_0}{\sigma})$  suit la loi de Student à  $n - 1$  degrés de liberté.

Ce test est applicable si on a la normalité des données ou si la taille de l'échantillon est grande (n > 30).

Pour effectuer un tel test, on peut utiliser la fonction `t.test()`.

#### Exemple

Une usine qui conditionne du café en paquets a mis à l'essai une machine réglée à 265 grammes. Voici les masses d'une série de paquets produits par la machine.

```
masse <- c(260, 262, 263, 264, 264, 265, 266, 266, 267, 268, 268, 268, 269, 269, 269, 269, 269, 269, 269, 270, 270, 271, 271, 271, 271, 271, 271, 272, 272, 273, 273, 274, 274, 276, 276, 277, 277, 279, 280)
masse

## [1] 260 262 263 264 264 265 266 266 267 268 268 268 269 269 269 269 269 269 270
## [18] 269 270 270 271 271 271 271 272 272 273 273 274 276 276 277 277 277
## [35] 279 280
```

On veut montrer que la machine est déréglée (masse moyenne différente de 265 grammes). Les hypothèses de ce test sont  $H_0: \mu = 265$  et  $H_1: \mu \neq 265$ .

```
t.test(masse, mu=265)
```

```
##
##  One Sample t-test
##
## data: masse
## t = 6.373, df = 35, p-value = 2.498e-07
## alternative hypothesis: true mean is not equal to 265
## 95 percent confidence interval:
## 268.3883 271.5561
## sample estimates:
## mean of x
## 269.9722
```

La p-value est inférieure à 0.05. Au seuil de 5%, on accepte l'hypothèse  $H_1$ . On peut conclure qu'il y a un déréglage de la machine, au risque 5% de se tromper.

#### 4.3.3.2 Comparaison de deux moyennes

On considère deux variables quantitatives  $X_1$  et  $X_2$  (qui mesurent la même caractéristique, mais dans deux populations différentes). On suppose que  $X_1$  et  $X_2$  suivent des lois normales et que  $X_1$  a pour variance  $\sigma_1^2$  et  $X_2$  a pour variance  $\sigma_2^2$ . A partir des estimations calculées sur deux échantillons de tailles respectives  $n_1$  et  $n_2$  issus des deux populations, on veut comparer  $\mu_1$  et  $\mu_2$ .

Les hypothèses du test sont

$H_0: \mu_1 = \mu_2$  et  $H_1: \mu_1 \neq \mu_2$  (ou  $H_0: \mu_1 = \mu_2$  et  $H_1: \mu_1 < \mu_2$  ou  $H_0: \mu_1 = \mu_2$  et  $H_1: \mu_1 > \mu_2$ ).

Avant de réaliser le test, il faut vérifier l'égalité des variances  $\sigma_1^2$  et  $\sigma_2^2$ . Ainsi, on spécifiera la valeur du paramètre `var.equal` dans la fonction `t.test()` en fonction du résultat du test d'égalité des variances.

#### Exemple

On reprend l'exemple des populations d'ouvriers et des sans emploi. On veut savoir s'il y a une différence significative entre les moyennes des âges des ouvriers et des sans emploi.

Le test des variances a conduit à une différence significative de la variance de l'âge des ouvriers et de la variance de l'âge des sans emploi.

On peut effectuer le test sur les moyennes.

```
t.test(ageouv, agesemp, var.equal=FALSE)
```

```
##
## Welch Two Sample t-test
##
```

```

## data: ageouv and agesemp
## t = 1.2239, df = 428.13, p-value = 0.2217
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -0.9090404 3.9091991
## sample estimates:
## mean of x mean of y
## 37.39648 35.89640

```

La p-value est de  $0.2217 > 0.05$ . Au seuil de 5%, on ne peut pas rejeter l'hypothèse  $H_0$ . On conclut qu'il n'y a pas de différence significative entre l'âge moyen des ouvriers et l'âge moyen des sans emploi, au risque 5% de se tromper.

#### 4.3.4 Tests de proportion

##### 4.3.4.1 Comparaison d'une proportion théorique à une valeur de référence

Soit  $p$  la fréquence inconnue d'un caractère dans une population donnée. On observe des données de présence/absence de ce caractère sur les individus d'un échantillon de taille  $n$  de cette population. Les hypothèses du test sont:

$H_0: p = p_0$  et  $H_1: p \neq p_0$  (ou  $H_0: p = p_0$  et  $H_1: p < p_0$  ou  $H_0: p = p_0$  et  $H_1: p > p_0$ ).

Sous  $H_0$  la statistique  $T = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}$  suit la loi normale centrée réduite.

Avant de réaliser le test, il faut vérifier les conditions suivantes: l'échantillon doit être de grande taille,  $np_0 \geq 5$  et  $n(1 - p_0) \geq 5$ .

Pour effectuer le test, on peut utiliser la fonction **prop.test()**.

##### Exemple

Une étude visant à réduire la consommation d'alcool a été menée sur une population et les résultats sont représentés par la variable **alc.cons** du data frame **cstat**. On veut savoir si le taux de consommation d'alcool dans cette population est inférieur à 20%.

On commence par donner le tableau des effectifs de la variable **alc.cons**.

```
table(cstat$alc.cons)
```

```

##
##    0    1
## 650 149

```

On peut maintenant effectuer le test à l'aide de la fonction **prop.test()**.

```
prop.test(149, 799, 0.2, alternative="less", correc=FALSE)
```

```

##
## 1-sample proportions test without continuity correction
##
## data: 149 out of 799, null probability 0.2
## X-squared = 0.91239, df = 1, p-value = 0.1697
## alternative hypothesis: true p is less than 0.2
## 95 percent confidence interval:
## 0.0000000 0.2101926
## sample estimates:
##          p
## 0.1864831

```

La p-value est de  $0.1697 > 0.05$ . Au seuil de 5%, on ne peut pas rejeter l'hypothèse  $H_0$ . On ne peut pas affirmer que le taux de consommation d'alcool dans cette population est inférieur à 20%.

### Remarque

Dans le cas de petits échantillons, on peut utiliser la fonction **binom.test()**.

#### 4.3.4.2 Comparaison de deux proportions

Soit  $p_1$  (respectivement  $p_2$ ) la proportion inconnue d'individus présentant un certain caractère dans une population  $P_1$  (respectivement  $P_2$ ). On désire comparer  $p_1$  et  $p_2$ . Pour cela, on utilise les fréquences (notées  $\hat{p}_1$  et  $\hat{p}_2$ ) d'apparition de ce caractère dans deux échantillons représentatifs respectivement des deux populations de taille  $n_1$  et  $n_2$ .

Les hypothèses du test sont:

$H_0: p_1 = p_2$  et  $H_1: p_1 \neq p_2$  (ou  $H_0: \hat{p}_1 = \hat{p}_2$  et  $H_1: \hat{p}_1 < \hat{p}_2$  ou  $H_0: \hat{p}_1 = \hat{p}_2$  et  $H_1: \hat{p}_1 > \hat{p}_2$ ).

Sous  $H_0$  la statistique  $T = \frac{\hat{p}_1 - \hat{p}_2}{\sqrt{\frac{\hat{p}(1-\hat{p})}{n_1} + \frac{\hat{p}(1-\hat{p})}{n_2}}}$  suit la loi normale centrée réduite avec  $\hat{p} = \frac{n_1\hat{p}_1 + n_2\hat{p}_2}{n_1 + n_2}$ .

Avant de réaliser le test, il faut vérifier les conditions suivantes: l'échantillon doit être de grande taille,  $n_1\hat{p} \geq 5$ ,  $n_1(1 - \hat{p}) \geq 5$ ,  $n_2\hat{p} \geq 5$  et  $n_2(1 - \hat{p}) \geq 5$ .

Pour effectuer le test, on peut utiliser la fonction **prop.test()**.

### Exemple

Une machine 1 a produit  $n_1 = 96$  pièces dont 12 défectueuses et une machine 2 a produit  $n_2 = 55$  pièces dont 10 défectueuses. Les pourcentages de pièces défectueuses produites par ces machines sont respectivement  $\hat{p}_1 = \frac{12}{96} = 0.125$  et  $\hat{p}_2 = \frac{10}{55} = 0.182$ . Peut-on en conclure que la machine 1 est significativement plus performante que la machine 2 ?

```
prop.test(c(12, 10), c(96, 55), alternative="less", correct=F)
```

```
##  
## 2-sample test for equality of proportions without continuity  
## correction  
##  
## data: c(12, 10) out of c(96, 55)  
## X-squared = 0.90693, df = 1, p-value = 0.1705  
## alternative hypothesis: less  
## 95 percent confidence interval:  
## -1.00000000 0.04516349  
## sample estimates:  
## prop 1 prop 2  
## 0.1250000 0.1818182
```

La p-value est de  $0.1705 > 0.05$ . Au seuil de 5%, on ne peut pas rejeter l'hypothèse  $H_0$ . La différence entre ces deux proportions n'est pas significative. On peut donc affirmer que la machine 1 n'est pas plus performante que la machine 2, au risque 5% de se tromper.

#### 4.3.5 Test d'indépendance du Khi-deux

On souhaite savoir si deux variables qualitatives (ou rendues qualitatives par regroupement) sont indépendantes. On teste l'hypothèse nulle  $H_0$ : "les deux variables sont indépendantes" contre l'hypothèse alternative  $H_1$ : "les deux variables ne sont pas indépendantes".

### Exemple

Dans une ville, on a demandé à des enseignants combien de fois ont-ils utilisé l'informatique pour faire la classe durant la semaine qui s'est écoulée. Les réponses sont données dans le tableau ci-dessous:

	Autres	HG	Langues	Arts plastiques	Maths
plus d'une fois	35	10	22	15	10
une fois ou moins	6	10	20	19	19

Dans cet échantillon constitué de 173 enseignants, y a-t-il un lien entre la fréquence d'utilisation de l'informatique en classe et la discipline enseignée?

Commençons par construire un tableau de contingence.

```
tabinfo<-matrix(c(35, 6, 10, 10, 22, 20, 15, 19, 10, 19), ncol=5)
rownames(tabinfo)<-c("plus d'une fois", "une fois ou moins")
colnames(tabinfo)<-c("Autres", "HG", "Langues", "Arts plastiques", "Maths")
tabinfo
```

```
##          Autres HG Langues Arts plastiques Maths
## plus d'une fois      35 10       22           15     10
## une fois ou moins     6 10       20           19     19
```

Appliquons le test du Khi-deux au tableau de contingence grâce à la fonction **chisq.test()**.

```
chisq.test(tabinfo)
```

```
##
## Pearson's Chi-squared test
##
## data: tabinfo
## X-squared = 22.18, df = 4, p-value = 0.0001845
```

La p-value est inférieure à 0.05. Au seuil de 5%, on accepte l'hypothèse  $H_1$ . On peut conclure qu'il y a un lien entre la fréquence d'utilisation de l'informatique en classe et la discipline enseignée, au risque 5% de se tromper.

# Fiches de travaux pratiques

## FICHE de TP N°1

### Exercice 1

1. Créer le vecteur de coordonnées  $a=(2,5; 3; 1; 0)$  et le vecteur  $b=(-3; 4; 12; 2)$ .
2. Calculer la somme des vecteurs  $a$  et  $b$ .
3. Remplacer le deuxième élément du vecteur  $b$  par  $-5$ .
4. Créer les vecteurs  $x=(2; 2; 2; 2)$  à l'aide de la fonction **rep** et  $y=(-1; 0; 1; 2; 3)$  à l'aide de la fonction **seq**.
5. Ordonner le vecteur  $a$  avec la fonction **sort**.

Que fait la fonction **order**? Comment peut-on ordonner  $a$  en utilisant cette dernière fonction?

### Exercice 2

Soit  $x=(7; 9; 13; 8; 4; 2; 16; 1; 6; 19; 15; 12; 19; 14; 8; 2; 19; 11; 18; 7)$ . Ecrire une expression R permettant d'extraire les éléments suivants:

1. Le deuxième élément du vecteur.
2. Les cinq premiers éléments du vecteur.
3. Les éléments strictement supérieurs à 14.
4. Tous les éléments sauf les éléments en positions 6, 10 et 12.

### Exercice 3

1. A l'aide des fonctions **cbind** puis **matrix**, créer la matrice  $A = \begin{pmatrix} 2 & 23 & 8 \\ 10 & 6 & 90 \\ 4 & 7 & 12 \end{pmatrix}$
2. A l'aide du vecteur  $c=(1, -5, 4, 7, 15, 0, 2, 8, 9)$ , créer la matrice  $B = \begin{pmatrix} 1 & -5 & 4 \\ 7 & 15 & 0 \\ 2 & 8 & 12 \end{pmatrix}$
3. Calculer la somme et le produit matriciel des matrices  $A$  et  $B$ .
4. Calculer la moyenne et le produit des lignes et des colonnes de  $A$  à l'aide des commandes **apply**, **sum** et **prod**.
5. Calculer la somme des deux premières lignes de  $A$  puis la somme de la première et de la troisième colonne de  $A$ .
6. Calculer la trace et le déterminant de  $A$  puis l'inverse de  $A$  (si cela est possible).
7. Résoudre le système linéaire suivant:  $\begin{cases} 2x + 23y + 8z = 5; \\ 10x + 6y + 90z = 6; \\ 4x + 7y + 12z = 7. \end{cases}$

### Exercice 4

1. Construire une liste appelée LST dont les éléments sont nommés: **name**, **wife**, **number.children** et **ages** et prennent respectivement les valeurs: "Fred", Mary, 3 et **c(9, 7, 4)**.
2. Extraire de deux manières le premier élément de la liste.
3. Ajouter à la liste un élément nommé **profession** et prenant la valeur **worker**.

### Exercice 5

On considère dans cet exercice le jeu de données **iris** disponible sous R.

1. Vérifier que c'est un objet de type **data.frame**.
2. Donner les dimensions de ce **data frame**, les noms des variables ainsi que leurs types.
3. Créer un objet de type **data.frame**, nommé **iris2**, contenant uniquement les individus de l'espèce **setosa**.
4. Trier les individus de l'espèce **setosa** du **data frame** **iris2** en fonction de la longueur du sépale.

## FICHE de TP N°2

### Présentation des données

Des études observationnelles ont suggéré que la faible consommation alimentaire ou les faibles concentrations plasmatiques de rétinol, de bêta-carotène pourraient être associées à un risque accru de développer certains types de cancer. Cependant, relativement peu d'études ont étudié les déterminants des concentrations plasmatiques de ces micronutriments. Des chercheurs ont conçu une étude transversale portant sur 315 individus pour étudier la relation entre les caractéristiques personnelles et les facteurs alimentaires, et les concentrations plasmatiques de rétinol, de bêta-carotène.

Les données se présentent de la manière suivante:

- age : Age (en années)
- sexe : 1=Masculin, 2=Féminin
- tabac : 1=jamais, 2=autrefois, 3=actuellement
- bmi : poids/(taille<sup>2</sup>)
- vitamine : 1=oui, souvent, 2=oui, pas souvent, 3=non
- calories : Nombre de calories par jour
- graisses : Grammes de graisse consommés par jour
- fibres : Grammes de fibre consommés par jour
- alcool : Nombre de verres d'alcool consommés par semaine
- cholesterol : Cholesterol consommé (mg par jour)
- betadiet : beta-carotene consommé (mcg par jour) (pro-vitamine A)
- retdiet : retinol consommé (mcg par jour) (vitamine A)
- betaplasma : beta-carotene plasmatique (ng/ml) (pro-vitamine A)
- retplasma : Retinol plasmatique (ng/ml) (vitamine A).

Les exercices qui suivent sont basés sur ces données contenues dans le fichier **dataTPretinol.csv**.

### Exercice 1

1. Charger le fichier **dataTPretinol.csv** dans le repertoire de travail sous le nom **dataTPstat** à l'aide de la commande **read.csv2**.
2. Vérifier que le fichier est bien du type “data.frame”
3. Afficher le nombre de lignes et de colonnes du data frame.
4. Quelles sont les dimensions du data frame?
5. Afficher les noms des variables du data frame.
6. Afficher la structure générale des variables puis recoder les variables sexe, tabac et vitamine en utilisant la commande **factor**.
7. Vérifier le changement en affichant les premières lignes du fichier.
8. Afficher le résumé détaillé de la distribution de chaque variable du data frame. Le data frame contient-il des données manquantes?

### Exercice 2

Dans cet exercice, on s'intéresse aux variables **age** et **tabac**.

1. Déterminer le minimum, le maximum, la moyenne, la médiane, le premier et le troisième quartile de la variable **age** à l'aide d'une seule commande.
2. Déterminer l'étendue, la variance et l'écart type de la variable **age**.
3. Construire la boîte à moustaches de la variable **age**.
4. Regrouper les données de la variable **age** dans ces différentes classes: [0;30[, [30;40[, [40;50[, [50;70[, [70;90[. Enregistrer cette nouvelle variable sous le nom: **ageclasse**.
5. Donner le tableau des effectifs de la variable **ageclasse**.
6. Construire l'histogramme de la variable **ageclasse**.
7. Construire le tableau des effectifs puis des fréquences de la variable **tabac**.
8. Construire le diagramme en barres puis le diagramme circulaire de la variable **tabac**.

### Exercice 3

1. On s'intéresse aux variables **graisses** et **calories**.
  - a. Calculer le coefficient de corrélation entre les variables **graisses** et **calories**.
  - b. Construire le nuage de points entre les variables **graisses** et **calories**.
  - c. Tracer la droite de régression de  $Y = \text{graisses}$  en  $X = \text{calories}$ .
  - d. Afficher les coefficients de la droite de régression de  $Y$  en  $X$ .
2. On s'intéresse aux variables **cholesterol** et **sexe**.
  - a. Construire un data frame nommé **dfhom** contenant uniquement des hommes et un data frame nommé **dfem** contenant uniquement des femmes à partir du data frame **dataTPretinol.csv**
  - b. Calculer la moyenne et l'écart type de la variable **cholesterol** dans les data frames **dfhom** et **dfem** puis comparer les résultats.
  - c. Construire une boîte à moustaches de la variable **cholesterol** en fonction des modalités de la variable **sexe**.
  - d. La quantité de cholesterol consommée par les hommes est-elle différente de celle consommée par les femmes?

## FICHE de TP N°3

### Exercice 1

Dans un centre avicole, des études antérieures ont montré que la masse d'un oeuf choisi au hasard peut être considérée comme la réalisation d'une variable aléatoire normale  $X$ , de moyenne  $m$  et de variance  $\sigma^2$ . On admet que les masses des oeufs sont indépendantes les unes des autres. On prend un échantillon de  $n = 36$  oeufs que l'on pèse. Les mesures sont les suivantes: 50.34, 52.62, 53.79, 54.99, 55.82, 57.67, 51.41, 53.13, 53.89, 55.04, 55.91, 57.99, 51.51, 53.28, 54.63, 55.12, 55.95, 58.10, 52.07, 53.30, 54.76, 55.24, 57.05, 59.30, 52.22, 53.32, 54.78, 55.28, 57.18, 60.58, 52.38, 53.39, 54.93, 55.56, 57.31, 63.15.

1. Donner une estimation des paramètres  $m$  et  $\sigma$ .
2. Donner un intervalle de confiance au niveau 95% puis 98% de la masse moyenne d'un oeuf.
3. Donner un intervalle de confiance au niveau 95% de la variance de la masse d'un oeuf.
4. Tester si la moyenne de cette variable aléatoire est égale à 56.

### Exercice 2

Cet exercice est basé sur une étude portant sur la couleur des cheveux des filles et des garçons d'un district écossais. Les résultats sont donnés dans le tableau en fin d'exercice.

1. Nous souhaitons savoir si la couleur des cheveux est indépendante du sexe avec une erreur de première espèce égale à 5%.
  - a. Saisir les données dans une matrice en précisant le nom des lignes et des colonnes.
  - b. Tracer un diagramme en barres de la couleur des cheveux en fonction du sexe.
  - c. Calculer les fréquences lignes et les fréquences colonnes.
  - d. Réaliser le test d'indépendance entre la couleur des cheveux et le sexe. Conclure.
2. Nous souhaitons comparer les proportions de garçons pour différentes couleurs des cheveux.
  - a. Tester l'égalité des proportions des garçons pour différentes couleurs des cheveux.
  - b. Conclure.

	Blond	Roux	Châtain	Brun	Noir
Garçon	592	119	849	504	36
Fille	544	97	677	451	14

### Exercice 3

On reprend les données contenues dans le fichier **dataTPretinol.csv** du TP 2. On souhaite tester l'égalité de la moyenne des âges des femmes et des hommes avec une erreur de première espèce de 5%.

1. Importer les données.
2. Créer un data frame de la sous-population des femmes et un data frame de la sous-population des hommes.
3. Comparer graphiquement ces deux sous-populations en construisant une boîte à moustaches des âges en fonction du sexe.
4. Afficher le résumé détaillé de la variable **age** dans chaque sous-population.
5. Tester la normalité des données dans chaque sous-population.
6. Tester l'égalité des variances.
7. Tester l'égalité des moyennes. Conclure.