Probabilité et statistique I

Introduction au logiciel R

Année académique
2019–2020

Section
Bachelier en Sciences Informatiques
Titulaire
Gentiane Haesbroeck
Assistante
Sophie Klenkenberg

Ces notes sont destinées aux étudiants de premier bloc de bachelier en sciences informatiques. Elles constituent une initiation au logiciel statistique R et à sa manipulation. Le premier chapitre se rapporte à la programmation (notions de base) dans `R`. Les autres chapitres ont quant à eux un caractère plus statistique.
Ces notes ont été rédigées par M. Ernst et adaptées par S. Klenkenberg.

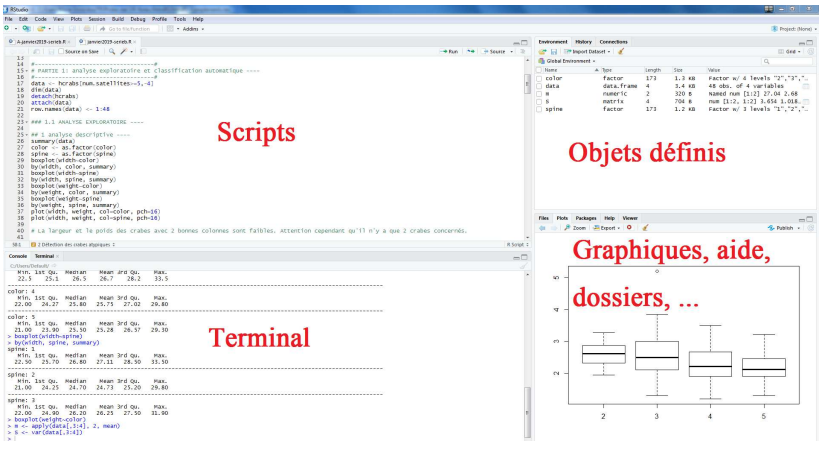
Janvier 2020

Notes préliminaires

Le logiciel `R` est un programme entièrement gratuit et librement diffusé ; il peut être installé sans aucune licence sur votre ordinateur, qu'il fonctionne sous Windows, sous Linux ou sous Mac. La description des commandes proposée dans ce manuel convient donc à tout utilisateur.

Le site web de référence sur le logiciel `R` est `www.r-project.org`. Ce site contient toutes sortes d'informations utiles liées à `R` dont des liens vers des postes de téléchargement de la dernière version de `R`, des notes d'introduction à `R` dans différentes langues, une FAQ destinées aux utilisateurs assidus de `R`, et bien d'autres choses encore...

Cependant, la plateforme par défaut du logiciel `R` est assez rudimentaire. Il existe une interface plus agréable, à savoir *R Studio*. Cette interface graphique est composée de 4 zones : la zone des scripts, la console, la liste des objets définis et une zone reprenant les aides, graphiques et explorateur de fichiers. De plus, l'éditeur des scripts est plus convivial que sur la plateforme par défaut puisqu'il utilise des codes couleurs spécifiques et une complétion semi-automatique. Ce logiciel possède également une version gratuite disponible sur le site `https://www.rstudio.com`. Un aperçu de cette plateforme est disponible ci-dessous.



Notez qu'une version de `R` doit être installée préalablement sur votre ordinateur pour pouvoir utiliser cette plateforme. Un lien vers la dernière version de `R` est également disponible depuis la page `https://cran.rstudio.com/`.

Démarrage de la session R

Lors du démarrage de R, plusieurs lignes de commentaires apparaissent. Celles-ci incluent entre autres la version du logiciel actuellement implémentée. Apparaît ensuite le symbole > qui annonce l'attente d'une commande à exécuter. Ce symbole ne doit toutefois pas être tapé dans R lorsque vous introduisez une commande ; il s'affiche automatiquement. Les résultats fournis par R, quant à eux, ne sont pas précédés du symbole >.

Dans ce texte, toute commande à introduire dans R sera précédée de > afin de garder une certaine continuité. Ainsi, en écrivant

```
> x <- c(1,2,3,4,5)
```

nous invitions le lecteur à introduire simplement `x <- c(1,2,3,4,5)`.

Scripts et environnements de travail

Afin de conserver une trace de votre travail et de faciliter votre travail dans la console, il est vivement conseillé de travailler dans un fichier texte, appelé script, depuis lequel l'exécution de commandes se fait à l'aide du raccourci CTRL+Enter ou via le bouton "*Run*" sur R Studio. Ce script peut aussi être sauvegardé (extension `.R`). Celui-ci permettra de recréer tous les objets définis lors d'une session précédente qui sont supprimés à chaque fermeture du logiciel.

Si vous souhaitez conserver vos objets pour une utilisation ultérieure, il est également possible de les sauvegarder. Pour réaliser une telle sauvegarde, il suffit de cliquer sur la petite disquette au dessus de la zone "*Environment*" de R Studio et de sélectionner "*Save Workspace as...*". Tous les objets de votre session seront dès lors sauvegardés dans un fichier de sauvegarde R d'extension .RData. Une fois ce fichier sauvé, pour le ré-ouvrir : cliquer sur l'icône en forme de dossier dans la fenêtre "*Environment*". Il ne vous reste plus qu'à sélectionner votre fichier de sauvegarde dans le menu qui apparaît à l'écran.

Notez que R vous propose automatiquement de sauvegarder vos objets lorsque vous quittez le logiciel.

Aide durant la session R

Il existe plusieurs façons d'obtenir de l'aide dans R. Pour avoir des renseignements généraux, un manuel en pdf (plus complet que le présent manuel qui se borne à présenter les outils utilisés dans le cadre du cours) est disponible dans l'onglet d'aide. Pour obtenir des informations complémentaires sur une commande précise de R, la fonction d'aide

```
> help(...)
```

doit être entrée dans le logiciel où les ... doivent être remplacés par le nom **exact** de la fonction. Les deux types d'aide sont souvent complémentaires étant donné qu'il faut connaitre le nom exact de la fonction `R` avant de pouvoir obtenir des informations précises à son sujet. En outre, le logiciel `R` est beaucoup utilisé. Il y a donc beaucoup d'aide disponible sur internet, n'importe quel moteur de recherche permettra d'obtenir des réponses à la plupart de vos questions.

Table des matières

Index des commandes

abline(a,b)	ajout d'une droite de pente b et d'ordonnée à l'origine a sur un graphique existant
abline(v=x)	ajout d'une droite verticale en l'abscisse x
abline(h=x)	ajout d'une droite horizontale en l'ordonnée x
abs(x)	valeur absolue de `x`
addmargins(tab)	ajout des distributions marginales au tableau de contingence `tab`
attach(data)	attachement des variables du fichier de données
barplot	tracé d'un diagramme en barres
boxplot()	tracé d'une boîte à moustaches
c(...)	définition d'un vecteur
cbind(...)	construction d'une matrice à partir de vecteurs colonnes
colors()	liste des couleurs disponibles
cos(v)	cosinus du vecteur `v`, composante à composante
cor(x,y)	corrélation entre les vecteurs `x` et `y`
cov(x,y)	covariance entre les vecteurs `x` et `y`
cumsum(x)	sommes cumulées des éléments de `x`
det(M)	déterminant de la matrice `M`
detach(data)	fin de travail avec l'ensemble de données `data`
dim(M)	dimension de la matrice `M`
ecdf(x)	renvoi de la fonction de répartition empirique du vecteur `x` (facile à représenter graphiquement via `plot(ecdf(x))`
eigen(M)	renvoi des vecteurs propres et valeurs propres de la matrice `M`
exp(v)	exponentielle du vecteur `v`, composante à composante
help(...)	accès à l'aide concernant une commande de R
hist()	tracé d'un histogramme ou affichage de la répartition en classes
identify()	identification d'une observation représentée sur un graphique
IQR(x)	écart-interquartile des composantes du vecteur `x`

legend()	ajout d'une légende à un graphique existant
length(v)	taille du vecteur `v`
levels(x)	modalités d'une variable qualitative `x`
lines()	superposition d'un graphique sur un autre existant
lm(y $\sim$ x)	calcul de la droite de régression de `y` en `x`
log(v)	logarithme népérien du vecteur `v`, composante à composante
log10(v)	logarithme en base 10 du vecteur `v`, composante à composante
matrix(v,n,p)	création d'une matrice à n lignes et p colonnes avec le vecteur `v`
max(v)	renvoi de la borne supérieure des composantes du vecteur `v`
mean(x)	moyenne des composantes du vecteur `x`
median(x)	médiane des composantes du vecteur `x`
min(v)	renvoie la borne inférieure des composantes du vecteur `v`
nlevels(x)	nombre de modalités d'une variable qualitative `x`
pairs(M)	création d'un tableau de graphiques avec les colonnes de la matrice `M`
par()	modification des paramètres d'affichage des graphiques
mfrow=c(.,.)	disposition des graphiques ligne par ligne
pie()	tracé d'un graphique en secteurs
plot()	réalisation d'un graphique
axes=F	ne pas tracer les axes
col="..."	couleurs du graphique (nombre ou nom de couleur)
lty=...	type de trait (nombre entre 0 et 6)
main="..."	titre principal
pch=...	type de point du graphique (nombre entre 0 et 25)
sub="..."	sous-titre
type="b"	les lignes ne touchent pas les points
type="h"	représentation d'un diagramme en bâtons
type="l"	uniquement tracer les lignes
type="o"	les points sont joints par des lignes
type="s"	représentation d'une fonction en escalier
xlab="..."	label de l'axe des x
xlim=c(.,.)	bornes de l'axe des x
ylab="..."	label de l'axe des y
ylim=c(.,.)	bornes de l'axe des y
points(.,.)	ajout de points sur le graphique
prod(v)	produit des composantes du vecteur `v`
prop.table(tab)	transformation du tableau de contingence des effectifs `tab` en le tableau de contingence des fréquences
quantile(x, α)	quantile α du vecteur `x`

range(v)	bornes inférieure et supérieure des composantes du vecteur `v`
rbind()	construction d'une matrice à partir de vecteurs lignes
read.table("...")	importation d'un fichier dans R dont ... est le chemin d'accès
rep(x,n)	répétition de l'objet `x n` fois
rm()	suppression d'un ou plusieurs objets
sd(x)	écart-type de `x`
seq()	génération d'une séquence régulière de nombres
setwd(...)	définition du nouveau répertoire courant dont ... est le chemin d'accès
sin(v)	sinus du vecteur `v`, composante à composante
sort(v)	tri des composantes du vecteur `v` par ordre croissant
solve(A)	recherche de l'inverse de la matrice `A`
sqrt(v)	racine carrée du vecteur `v`, composante à composante
stem(x)	diagramme en tiges et feuilles de `x`
sum(v)	somme des composantes du vecteur `v`
summary()	résumé des données qualitatives (via les effectifs) ou quantitatives (via le résumé à cinq valeurs)
t(A)	transposée de la matrice `A`
table(x)	effectifs calculés sur le vecteur `x`
table(x,y)	tableau de contingence des effectifs des vecteur `x` et `y`
tan(v)	tangente du vecteur `v`, composante à composante
unique(v)	recherche des valeurs différentes dans les composantes du vecteur `v`
var(x)	variance de `x`
View(data)	visualisation des données dans un tableau
A %*% B	produit matriciel de A et B

Chapitre 1

Notions de bases pour le logiciel R

1.1 Mathématiques de base

R peut être utilisé comme une calculatrice. Les symboles +, -, * et / désignent les quatre opérations arithmétiques. Ainsi, calculer $\frac{3\times4-1}{2}$ s'effectue par

```
> (3*4-1)/2
[1] 5.5
```

Remarquez que la réponse fournie par R (5.5) est précédée de [1]. Nous donnerons dans la suite quelques détails sur cette notation.

La mise à l'exposant s'effectue à l'aide du symbole ^. Par exemple,

```
> 3^2
[1] 9
```

Diverses commandes permettent d'obtenir les fonctions trigonométriques usuelles : sin, cos, tan, ... De même, R dispose de fonctions classiques : log, exp, sqrt (pour la racine carrée d'un nombre ou d'une expression), log10 (logarithme en base 10), etc. A cet effet, nous invitons le lecteur à consulter l'index des commandes en début d'ouvrage.

1.2 Manipulation de vecteurs

La structure la plus simple de R est un *vecteur*, i.e. une seule entité composée d'une suite ordonnée de nombres. Décidons de créer un vecteur x égal à $\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4.5 \end{pmatrix}$. Cela se traduit en R par

```
> x <- c(1,2,3,4.5)
```

Dans la suite, par souci de facilité, nous représenterons les vecteurs par des vecteurs lignes.

La fonction `c()` utilisée ici pour définir le vecteur `x` peut s'appliquer à des vecteurs de taille quelconque. Le lecteur initié à un langage de programmation verra l'analogie de la fonction `<-` avec l'affectation d'une valeur à une variable dont l'identificateur est `x`. Notons que les caractères `<-` peuvent être remplacés par le caractère = . Pour plus de clarté cependant, nous n'utiliserons pas cette notation.

Si un vecteur a été défini, on peut afficher son contenu en tapant simplement son nom. Par exemple, si on tape

```
> x
```

on obtient :

```
[1] 1.0 2.0 3.0 4.5
```

En tapant `x`, l'utilisateur demande à l'ordinateur d'afficher le contenu du vecteur référencé `x`.

Attention : `R` *fait la distinction entre les majuscules et les minuscules. Ainsi, à ce stade, le vecteur X n'existe pas ! De même, les vecteurs* `PremierVecteur` *et* `premiervecTeur`*, par exemple, seront différents.*

Ayant à notre disposition le vecteur `x`, nous pouvons calculer des expressions le faisant intervenir. Par exemple, si nous entrons comme commande,

```
> 1/x
```

les opérations étant effectuées composante à composante, le résultat suivant apparaît :

```
[1] 1.0000000 0.5000000 0.3333333 0.2222222
```

Notez que le résultat est uniquement affiché à l'écran. Le vecteur `x` n'a pas été modifié, il est toujours égal à (1,2,3,4.5). Si l'on veut conserver le résultat `1/x`, on doit créer un nouveau vecteur `y`, dont les composantes sont les inverses des composantes de `x`, par

```
> y <- 1/x
```

On aurait pu également remplacer le vecteur `x` par `1/x`. Pour ce faire, on aurait écrit

```
> x <- 1/x
```

Dans ce cas, les anciennes valeurs (1, 2, 3, 4.5) contenues dans `x` sont définitivement perdues et remplacées par les nouvelles valeurs (1, 0.5, 0.33333, 0.22222).

Les vecteurs peuvent être utilisés dans des expressions arithmétiques. Dans ce cas, les opérations sont effectuées composante à composante : par exemple, on peut taper

```
> v <- 2*x + y + 1
```

On peut appliquer les opérateurs arithmétiques élémentaires ainsi que les fonctions traditionnelles à des vecteurs. Comme cela a déjà été vu, elles s'appliquent alors composante à composante. Par exemple, voici une session possible :

```
> w <- c(5, 4, 2)
> w <- 2*w + 1
> w
[1] 11 9 5
> log(w)
[1] 2.397895 2.197225 1.609438
> w
[1] 11 9 5
```

Détaillons les opérations effectuées. La première ligne crée un vecteur à 3 composantes nommé `w`. A la deuxième ligne, ce vecteur est remplacé par un nouveau vecteur où chaque composante est égale à deux fois sa valeur précédente plus un. On demande ensuite d'afficher le vecteur `w`, puis d'afficher le logarithme népérien des composantes de `w`. Insistons encore une fois sur le fait que la commande `log(w)` n'a en aucun cas modifié le vecteur `w`.

La fonction `min` (resp. `max`) fournit la plus petite (resp. grande) valeur d'un vecteur. Si nous continuons l'exemple,

```
> min(w)
[1] 5
```

Remarquons une fois encore que la valeur affichée n'est en aucun cas stockée. Si nous voulons la conserver, nous devons écrire

```
> m <- min(w)
```

La fonction `range` quant à elle, fournit la plus petite et la plus grande valeur d'un vecteur. Puisque cette fonction a comme résultat deux nombres, ces deux nombres sont stockés dans un vecteur de taille 2. Ainsi,

```
> r <- range(w)
> r
[1] 5 11
```

Nous pouvons calculer la somme et le produit des composantes d'un vecteur, à l'aide des commandes `sum` et `prod`. Ainsi, en utilisant le vecteur `w`,

```
> prod(w)
[1] 495
```

et

```
> sum(w)
[1] 25
```

D'autres fonctions peuvent parfois s'avérer utiles. En voici quelques unes :

- `length` rend le nombre de composantes du vecteur
- `sort` ordonne par ordre croissant les valeurs du vecteur (l'ajout de l'argument `decreasing=TRUE` permet de trier les valeurs de manière décroissante)
- `unique` rend un vecteur contenant uniquement les composantes deux à deux différentes du vecteur source
- ... (voir l'index des commandes page **??**)

Ainsi,

```
> x <- c(1,1,5,6,44,5,4,1)
> unique(x)
[1] 1 5 6 44 4
> sort(x)
[1] 1 1 1 4 5 5 6 44
```

On peut utiliser des noms beaucoup plus explicites que `w`, `x` ou `z` pour définir les vecteurs. Rien ne vous empêche de nommer un vecteur par `LePetitVecteur22`. Rappelez-vous seulement que `R` distingue les majuscules des minuscules. De plus, il se peut, après un certain temps de travail, que l'on ait oublié certains vecteurs, ou encore que certains soient devenus inutiles. Dans ce cas, il est possible d'observer la liste des objets définis dans la fenêtre `Environment` (fenêtre supérieure droite) de R Studio.

Si un élément `v` est devenu inutile, on peut le supprimer au moyen de la commande `rm(v)` (qui est une abréviation de *remove*) ou encore directement en le sélectionnant dans la fenêtre `Environment` de R Studio.

Finalement, il est possible de créer un vecteur par juxtaposition d'autres vecteurs, en utilisant simplement la commande `c()`. Celle-ci agit par simple concaténation sur ses arguments. Par exemple,

```
> i <- c(1,2,3)
> j <- c(4,5)
> k <- c(6,7,8,9,10)
> x <- c(i,j,k)
> x
[1] 1 2 3 4 5 6 7 8 9 10
```

1.3 Génération d'une suite régulière

Si nous voulons créer un vecteur comprenant les entiers de 1 à 10, nous pouvons taper

```
> x <- c(1,2,3,4,5,6,7,8,9,10)
```

Un moyen plus rapide est

```
> x <- 1:10
```

On peut ainsi générer des suites d'entiers, par ordre croissant `1:10` ou par ordre décroissant `10:1`. Si nous voulons créer une suite de nombres réels dont la différence entre les nombres est constante deux à deux on peut utiliser la commande `seq`. Par exemple, pour créer la suite -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, il suffit de taper

```
> x <- seq(from=-1.5, to=2, by=0.5)
```

ou encore, si on donne la longueur de la suite,

```
> x <- seq(length=8, from=-1.5, by=0.5)
```

fournit le même résultat.

On a vu que l'on pouvait créer un vecteur par juxtaposition d'autres vecteurs. Si l'on désire juxtaposer plusieurs fois le même vecteur, la commande `rep` est utile. Ainsi,

```
> w <- rep(x, times=5)
```

permet de créer un vecteur `w` composé du vecteur `x` juxtaposé (répété) 5 fois. Notons que l'argument `times=` est facultatif ; il est donc équivalent d'écrire

```
> w <- rep(x,5)
```

Voici maintenant l'explication du symbole `[1]` se trouvant devant les résultats fournis par `R`. Il indique en fait l'indice de la première composante de la ligne correspondante. Comme jusqu'à présent, les résultats tenaient en une seule ligne, cet indice est toujours resté le même. Cette fois-ci, si l'on souhaite afficher le vecteur `w`, plusieurs lignes de résultats vont s'afficher :

```
> w
[1] -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5
[16] 2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0
[31] 1.5 2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0
```

Ainsi, on sait (sans devoir compter) que la première composante de la seconde ligne est en fait la $16^{\text{ème}}$ composante du vecteur `w`.

1.4 Composantes d'un vecteur

Dans R, on accède aux composantes d'un vecteur au moyen de crochets. Définissons tout d'abord un vecteur w :

```
> w <- 1:10
```

Actuellement, la cinquième composante de w contient la valeur 5. Nous pouvons nous en assurer en tapant

```
> w[5]
[1] 5
```

On place entre crochets le numéro de la composante à laquelle on veut accéder. Dès lors, on peut modifier cette composante, par exemple avec

```
> w[5] <- 33
```

et on obtient comme vecteur w

```
[1] 1 2 3 4 33 6 7 8 9 10
```

Le travail avec les composantes d'un vecteur suit les mêmes règles que le travail avec les vecteurs eux-mêmes. On peut voir une composante d'un vecteur comme un vecteur de longueur 1. Ainsi, on peut construire un nouveau vecteur comprenant certaines des composantes du vecteur w de la manière suivante :

```
> f <- c(w[1],w[3],w[7])
```

L'argument placé entre crochets peut très bien être un vecteur. Dès lors, l'instruction précédente est équivalente [1] à

```
> f <- w[c(1,3,7)]
```

Par conséquent, pour sélectionner des composantes successives d'un vecteur, on peut procéder en générant une suite régulière. Par exemple,

```
> w[3:5]
[1] 3 4 33
```

Il est également possible de sélectionner certaines composantes d'un vecteur suivant une condition bien précise. Par exemple, on obtient les composantes de w strictement supérieures à 9 en utilisant

```
> w[w>9]
[1] 33 10
```

Si par contre, on ne veut pas sélectionner une composante particulière (ou un ensemble de composantes), mais plutôt l'éviter, on utilise le symbole de la soustraction :

```
> w[-5]
[1] 1 2 3 4 6 7 8 9 10
```

1. Précision : il y a une différence dans le type d'objet défini mais cette différence ne perturbe en rien le traitement des données tel qu'enseigné dans ce cours.

1.5 Matrices

1.5.1 Définition d'une matrice

La structure de matrice est utilisée comme structure pour gérer des tableaux de nombres à double entrée. Les matrices sont définies grâce aux vecteurs. Pour comprendre ce qu'il se passe, définissons un nouveau vecteur

```
> vect <- 1:9
```

A partir de ce vecteur, on peut construire une matrice 3×3 en indiquant à `R` les nombres de lignes et de colonnes que la matrice doit comporter. On procède avec la commande `dim` :

```
> dim(vect) <- c(3,3)
```

Une matrice n'est donc rien d'autre qu'un vecteur muni d'un attribut caractérisant sa dimension. Les 3 premiers éléments de `vect` constituent la première colonne de la matrice, et ainsi de suite. En effet,

```
> vect
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

On peut donc définir une matrice à partir de ses colonnes. Si nous voulons construire une matrice à deux lignes et trois colonnes, dont les colonnes sont des vecteurs (à deux composantes) notés `c1`, `c2` et `c3`, on peut procéder de la manière suivante :

```
> c1 <- c(1,0)
> c2 <- c(2,3)
> c3 <- c(4,1)
> A <- c(c1,c2,c3)
> dim(A) <- c(2,3)
> A
     [,1] [,2] [,3]
[1,]    1    2    4
[2,]    0    3    1
```

Une introduction plus rapide mais moins intuitive de la matrice `A` aurait été

```
> A <- c(1,0,2,3,4,1)
> dim(A) <- c(2,3)
```

La méthode énoncée ci-dessus permet donc de créer une matrice comportant n lignes et p colonnes, et nécessite un vecteur de longueur $n \times p$. Une autre fonction pour créer une matrice (toujours à partir d'un vecteur) est la fonction `matrix`. La syntaxe est la suivante :

```
matrix(vecteur, nombre de lignes, nombre de colonnes)
```

Ainsi, on a

```
> v <- 1:9
> M <- matrix(v,3,3)
> M
      [,1]  [,2]  [,3]
 [1,]    1     4     7
 [2,]    2     5     8
 [3,]    3     6     9
```

On a donc créé, colonne par colonne, une matrice `M` à partir du vecteur `v`. Une option intéressante est la possibilité de construire la matrice ligne par ligne. Pour ce faire, il faut indiquer l'option `byrow=TRUE`. Ainsi,

```
> v <- 1:9
> M <- matrix(v,3,3, byrow=TRUE)
> M
      [,1]  [,2]  [,3]
 [1,]    1     2     3
 [2,]    4     5     6
 [3,]    7     8     9
```

Avec les deux exemples ci-dessus, on voit bien la différence dans la construction des matrices à partir d'un même vecteur `v`. On remarquera aussi qu'une matrice est la transposée de l'autre. Par défaut, la construction de matrices s'effectue colonne par colonne.

Deux autres fonctions permettent encore de créer une matrice à partir de vecteurs-lignes ou de vecteurs-colonnes, `rbind` et `cbind` (les initiales `r` et `c` correspondent respectivement à *row* et *column*). Par exemple,

```
> v1 <- c(1,2,3)
> v2 <- c(4,5,6)
> v3 <- c(7,8,9)
> M <- cbind(v1,v2,v3)
> M
      [,1]  [,2]  [,3]
 [1,]    1     4     7
 [2,]    2     5     8
 [3,]    3     6     9
```

Avec `rbind`, on obtient

```
> N <- rbind(v1,v2,v3)
> N
      [,1]  [,2]  [,3]
 [1,]    1     2     3
 [2,]    4     5     6
 [3,]    7     8     9
```

1.5.2 Composantes d'une matrice

Nous savons déjà comment accéder aux composantes d'un vecteur. `R` dispose de méthodes analogues pour les matrices. Pour accéder à l'élément situé à l'intersection entre la ligne i et la colonne j d'une matrice `A`, on tape `A[i,j]`. Ainsi, avec l'exemple précédent,

```
> N[2,3]
[1] 6
```

Avec une matrice, on peut aussi accéder à une ligne tout entière ou à une colonne tout entière. Par exemple, `N[,3]` renvoie un vecteur contenant la troisième colonne de la matrice `N` et `N[2,]` renvoie un vecteur contenant la deuxième ligne de `N`. Remarquez qu'à l'affichage d'une matrice, `R` utilise cette numérotation des lignes et des colonnes. Ainsi,

```
> N[,3]
[1] 3 6 9
> N[2,]
[1] 4 5 6
```

Puisque les matrices ne sont en fait que des vecteurs d'un genre particulier, on peut leur appliquer toutes les opérations arithmétiques et les fonctions définies composante à composante pour les vecteurs. Par exemple, en reprenant la matrice `A` définie précédemment,

```
> A <- 2*A
> A
      [,1]  [,2]  [,3]
 [1,]    2     4     8
 [2,]    0     6     2
```

Autre exemple : le logarithme népérien de chacune des composantes de `M` peut être obtenu par

```
> log(N)
          [,1]      [,2]     [,3]
 [1,] 0.000000 0.6931472 1.098612
 [2,] 1.386294 1.6094379 1.791759
 [3,] 1.945910 2.0794415 2.197225
```

1.5.3 Calcul matriciel

Nous présentons à présent différents éléments de calcul matriciel. Tout d'abord, la transposée d'une matrice est obtenue au moyen de la fonction `t`. Si nous reprenons l'exemple précédent,

```
> A
      [,1]  [,2]  [,3]
 [1,]    2     4     8
 [2,]    0     6     2
> t(A)
      [,1]  [,2]
 [1,]    2     0
 [2,]    4     6
 [3,]    8     2
```

Il est bien évident que `A[,1]` est égal à `t(A)[1,]`.

Si `A` et `B` sont deux matrices carrées de dimension k, `A % * % B` est le produit matriciel habituel (ligne par colonne) des matrices `A` et `B` :

$$\texttt{A \% * \% B [i,j]} = \sum_{n=1}^{k} a_{in} b_{nj}$$

Le produit matriciel `A % * % B` est également défini pour une matrice `A` de dimension $n \times p$ et une matrice `B` de dimension $p \times q$. Si les dimensions ne correspondent pas, un message d'erreur apparaîtra.

Attention : ne pas confondre le produit matriciel `A % * % B` avec `A * B`, qui est le produit élément à élément :

$$\texttt{A * B [i,j]} = a_{ij} b_{ij}.$$

On remarque la différence entre `A % * % B` et `A * B` sur l'exemple suivant :

```
> v <- 1:9
> A <- matrix(v,3,3)
> v <- c(2,0,0,0,3,0,0,0,4)
> B <- matrix(v,3,3)
> A
      [,1]  [,2]  [,3]
 [1,]    1     4     7
 [2,]    2     5     8
 [3,]    3     6     9
> B
```

```
      [,1]  [,2]  [,3]
 [1,]    2     0     0
 [2,]    0     3     0
 [3,]    0     0     4
> A % * % B
      [,1]  [,2]  [,3]
 [1,]    2    12    28
 [2,]    4    15    32
 [3,]    6    18    38
> A*B
      [,1]  [,2]  [,3]
 [1,]    2     0     0
 [2,]    0    15     0
 [3,]    0     0    36
```

Pour calculer l'inverse d'une matrice carrée, on utilise la fonction `solve`. Par exemple,

```
> c1 <- c(2,3)
> c2 <- c(1,1)
> M <- c(c1,c2)
> dim(M) <- c(2,2)
> M
      [,1]  [,2]
 [1,]    2     1
 [2,]    3     1
> solve(M)
      [,1]  [,2]
 [1,]   -1     1
 [2,]    3    -2
```

Si la matrice n'est pas inversible, `R` renvoie un message d'erreur.

1.6 Remarque sur le caractère vectoriel de `R`

Il est important d'insister sur le caractère vectoriel de `R`, déjà rencontré lors des opérations sur les vecteurs (Section **??**). Nous avons vu que, comme dans le calcul vectoriel, le fait de multiplier un vecteur par un nombre a pour effet de multiplier toutes les composantes. Contrairement aux règles de calcul vectoriel, cela est également valable pour l'addition :

```
> w<-c(5,4,2)
> w<-2*w+1
> w
[1] 11 9 5
```

Cette particularité du logiciel permet de diminuer le nombre de commandes nécessaires pour l'addition de deux objets de type différent. Cela reste également valable avec des vecteurs et des matrices :

```
> v1 <- c(1,2,3)
> v2 <- c(4,5,6)
> v3 <- c(7,8,9)
> M <- cbind(v1,v2,v3)
> M
v1 v2 v3
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
> x<-c(1,2,3)
> M+x
v1 v2 v3
[1,] 2 5 8
[2,] 4 7 10
[3,] 6 9 12
```

Ainsi, `R` a ajouté le vecteur colonne `x` à toutes les colonnes de la matrice `M`. Et l'on peut même additionner un nombre et une matrice :

```
> M+1
v1 v2 v3
[1,] 2 5 8
[2,] 3 6 9
[3,] 4 7 10
```

ce qui a le même effet que d'ajouter aux colonnes de la matrice le vecteur dont toutes les composantes valent 1 :

```
> y<-c(1,1,1)
> M+y
v1 v2 v3
[1,] 2 5 8
[2,] 3 6 9
[3,] 4 7 10
```

Il reste à ne pas oublier que ces règles de calcul sont propres à `R` et sont bien sûr fausses en algèbre vectorielle.

1.7 Les booléens et les tests

Un booléen est une variable ne pouvant prendre que deux valeurs possibles, dans notre cas TRUE (vrai - 1) et FALSE (faux - 0). `R` permet de réaliser des tests en utilisant des booléens. En voici un exemple.

```
> a <- c(1,5,2)
> b <- c(2,1,4)
> a<b
[1] TRUE FALSE TRUE
> vect <- a>1
> vect
[1] FALSE TRUE TRUE
```

Quelques explications s'imposent. Les tests sont réalisés composante à composante ; dès lors, `a<b` renvoie un vecteur dont la i-ème composante a pour valeur TRUE si la i-ème composante de `a` est strictement inférieure à la i-ème composante de `b`. La suite de l'exemple nous montre que les résultats d'un test peuvent être stockés dans un vecteur. Ce vecteur ne contient pas des nombres, mais uniquement les valeurs TRUE/FALSE observées.

Voici un récapitulatif des tests disponibles les plus courants (nous supposons ici que `a` et `b` sont des nombres).

	est vrai si et seulement si
`a<b`	`a` est strictement inférieur à `b`
`a<=b`	`a` est inférieur ou égal à `b`
`a>b`	`a` est strictement supérieur à `b`
`a>=b`	`a` est supérieur ou égal à `b`
`a==b`	`a` est égal à `b`
`a!=b`	`a` est différent de `b`

On peut également réaliser des opérations (composante à composante) entre deux vecteurs de booléens : la conjonction, la disjonction et la négation. Si `a` et `b` sont les composantes i de deux vecteurs de booléens, alors

	est vrai si	est faux si
`a & b`	`a` et `b` sont vrais	au moins un des deux est faux
`a|b`	au moins un des deux est vrai	`a` et `b` sont faux
`!a`	`a` est faux	`a` est vrai

Il est également possible de travailler sur un vecteur booléen afin d'obtenir une seule information. Par exemple, il est possible de compter le nombre de valeurs TRUE dans un vecteur booléen en effectuant simplement la commande `sum(vecteur)` car les

valeurs TRUE peuvent être vues comme des 1 et les valeurs FALSE correspondent à des 0. D'autres exemples sont les fonctions `any` et `all` qui permettent d'indiquer si "au moins une" ou "toutes les" valeurs du vecteurs sont TRUE. Par ailleurs, la fonction `which` permet d'obtenir les indices d'un vecteur qui vérifient une certaine condition. Par exemple, si l'on souhaite savoir quels éléments de `a` sont strictement supérieurs à 1, il suffit d'utiliser la commande

```
> which(a > 1)
[1] 2 3
```

pour vérifier que les $2^{\text{ème}}$ et $3^{\text{ème}}$ éléments vérifient cette condition. Dans le même ordre d'idée, les fonctions `which.min` et `which.max` rendent les indices du minimum et du maximum de `a`[2].

1.8 Importation de données

Habituellement, les données exploitées ne sont pas encodées manuellement mais sont disponibles dans un fichier Excel ou texte. Il est donc intéressant de pouvoir importer de tels fichiers de données dans le logiciel `R`.

Avant de démarrer l'importation d'un fichier en tant que telle, il est préférable de préciser au logiciel `R` le dossier dans lequel le fichier à importer se situe en "Changeant le répertoire courant" (ou "*Set as working directory*") via la fenêtre "*Files*" de R Studio. Plus précisément, il est possible de cliquer sur les trois points de suspension afin de modifier le dossier affiché. Lorsque celui-ci est visible dans la fenêtre, sélectionner "*More*" puis "*Set as working directory*". Il est également possible de le faire directement via la commande

```
> setwd("C:/Users/user/My Documents/Dossier de travail")
```

La démarche d'importation de fichier est expliquée ici pour des fichiers de données dont les lignes correspondent aux observations et les colonnes aux variables. La commande générique pour importer un tel fichier est

```
> data <- read.table("data1.txt")
```

Si la première ligne du fichier contient les noms des variables, l'argument `header=TRUE` doit être ajouté.

Par défaut, l'importation de fichier suppose que les différentes colonnes sont séparées par un ou plusieurs espaces. Si celles-ci sont séparées par un autre symbole (par exemple un point virgule habituellement utilisé pour les fichiers `.csv`), il faut ajouter l'option `sep=";"` dans la commande d'importation.

2. Si plusieurs éléments de `a` sont égaux au minimum (resp. maximum), la fonction `which.min` (resp. `which.max`) renvoie l'indice du premier élément vérifiant cette condition.

Une autre option qui peut-être intéressante est `row.names=`i qui permet de préciser que les noms des observations (i.e. des lignes) sont repris dans la $i$ème colonne. Si cette option est utilisée, les noms des observations peuvent par la suite être récupérés via la commande `row.names(data)`. Par exemple,

```
> row.names(data)[23]
```

rend le nom de la $23^{\text{ème}}$ observation.

Notez également que les données manquantes peuvent être traitées par le logiciel `R` qui les code par le symbole `NA` (pour *Not Available*). Si les données manquantes ont un format particulier dans le fichier de base, il est possible de le définir lors de l'importation des données. Le lecteur est invité à consulter l'aide de la commande `read.table` pour plus de précisions.

Une fois le fichier importé, l'objet `data` est disponible. La visualisation à l'aide de la commande `View(data)` permet de s'assurer que l'importation s'est bien déroulée.

Afin de travailler avec les différentes variables de cet ensemble, il existe plusieurs possibilités :

1. Attacher le fichier de données à l'aide de la commande

   ```
   > attach(data)
   ```

   Cela permet de travailler avec les variables en les appelant simplement par leur nom mentionné en titre de colonne. Lorsque le travail est terminé, on détache les données via

   ```
   > detach(data)
   ```

2. Si les noms des variables sont `var1`, `var2`, ..., on peut utiliser le symbole `$` comme suit

   ```
   > data$var1
   ```

   pour obtenir la première variable.

3. On peut également procéder comme si `data` était une matrice dont les colonnes sont les variables. Ainsi,

   ```
   > data[,1]
   ```

   permet également d'obtenir la première variable.

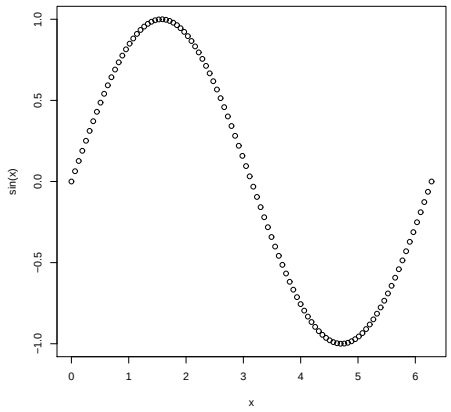
Enfin, notons qu'il existe des fonctions permettant d'importer des fichiers ayant un autre format que les formats `.txt` ou `.csv` (eg des fichiers `.xlsx` ou `.xls`). Cependant, dans le cadre de ce cours, il est conseillé de convertir au préalable tout fichier en un fichier `.txt` ou `.csv` afin d'utiliser la fonction `read.table` pour l'importer dans `R`.

1.9 Graphiques

La fonction `plot` permet de réaliser des graphiques de base. Il faut spécifier au logiciel le ou les vecteurs à représenter. Comme illustration, il est possible de représenter la fonction $\sin(x)$ pour $x \in [0, 2\pi]$. Pour commencer, il faut générer une suite de points situés dans cet intervalle. Ce vecteur sera représenté sur l'axe des abscisses et les ordonnées sont obtenues en évaluant le sinus de chacune de ces valeurs. Plus précisément, les commandes

```
> x <- seq(0, 2*pi, l=100)
> plot(x, sin(x))
```

permettent d'obtenir le graphique ci-dessous.



Par défaut, la fonction `plot` trace des points. L'argument `type="..."` permet de varier les graphiques :

- `type="o"` pour joindre les différents points du graphique ;
- `type="l"` pour ne tracer que des lignes (les points ne sont plus affichés) ;
- `type="b"` pour que les lignes joignant les points ne se touchent pas ;
- `type="h"` pour représenter des bâtons verticaux (diagramme en bâtons) ;
- `type="s"` pour tracer en escalier, où les deuxièmes composantes correspondent au haut des paliers ;
- ...

Initialement, les axes ont pour label le nom des vecteurs utilisés dans la fonction `plot`. Si l'on veut modifier l'intitulé des axes, il faut utiliser l'argument `xlab="..."` ou `ylab="..."`, respectivement pour l'axe des `x` ou des `y`. De plus, pour donner un titre au graphique, on utilise l'argument `main="..."`. Ce titre sera placé en haut de la figure.

Lors de chaque utilisation de la fonction `plot`, tout graphique contenu dans la fenêtre est remplacé par un nouveau graphique (dans R Studio, il est possible de consulter les graphiques précédents en cliquant sur les flèches au dessus du graphique affiché).

Il est néanmoins possible de superposer plusieurs graphiques sur un premier, déjà affiché dans la fenêtre. Pour cela, on utilise la commande `lines`. Celle-ci est similaire à la fonction `plot` pour les arguments classiques : `lines(x, y, type="o")`, par exemple, est autorisé. De même, on peut ne rajouter que des points sur un graphique existant, à l'aide de la commande `points`.

Lorsque `R` trace un graphique, il détermine automatiquement les limites des axes, afin d'afficher toutes les données. Si vous le souhaitez, vous pouvez modifier ces limites (par exemple pour ne pas afficher des valeurs extrêmes trop éloignées de la majorité des observations). Pour ce faire, il suffit d'utiliser les arguments `xlim=c(.,.)` et `ylim=c(.,.)` pour les axes des `x` et des `y` respectivement. Les composantes des vecteurs précisées dans ces commandes représentent les bornes inférieures et supérieures des échelles d'axes.

Signalons encore quelques commandes graphiques parfois très utiles pour des représentations multiples de graphiques. Tout d'abord, il est possible de modifier la couleur des différents graphiques à afficher. Pour cela, il suffit de taper l'argument `col="..."`, où `...` doit être remplacé par un chiffre entre 1 et 8 ou par le nom (en anglais) de la couleur choisie. Pour obtenir la liste des différentes couleurs disponibles (plus de 600 !), il suffit de taper la commande `colors()`. De plus, vous pouvez modifier le symbole représentant les points du graphique (et qui, par défaut, sont représentés par des cercles vides). Pour ce faire, utilisez l'argument `pch= ...` où `...` est ici remplacé par un entier de 1 à 25. Enfin, les graphiques tracés peuvent être représentés selon différents types de traits. L'argument `lty= ...` contrôle ce paramètre. Il suffit de remplacer `...` par un entier pour obtenir un trait différent. Ainsi, `lty=1` donnera un trait plein, `lty=2` un trait en tirets, `lty=3` un trait en points,...

Les différentes options mentionnées ici sont les plus utilisées. Il existe néanmoins une quantité d'arguments optionnels permettant de personnaliser encore plus le graphique. Nous laissons au lecteur intéressé le soin de consulter l'aide de `R` (ou d'autres références) sur le sujet.

En plus des options internes aux graphiques, il est possible de rajouter une légende sur un graphique grâce à la commande `legend()`. Ses arguments sont, de façon non-exhaustive, la position de la boite encadrant la légende (`x="bottom"`, `"topleft"`,...), le texte de la légende sous la forme d'un vecteur de textes (`legend=c("...", "...")`), un vecteur de couleurs ou encore un vecteur de traits.

Enfin, terminons cette section en mentionnant la commande `identify` qui permet de sélectionner un point dans un graphique afin de l'identifier. Ses arguments doivent être identiques à ceux du `plot` utilisé. Notez qu'il faut bien être attentif à "quitter le graphique interactif" (Esc ou bouton "Finish" sur R Studio) pour poursuivre l'utilisation du logiciel.

1.9.1 Exportation de graphiques

Les graphiques peuvent être copiés dans le presse-papier ou directement sauvegardés. Cette capacité est intéressante lors de la rédaction éventuelle d'un rapport. Voici comment procéder : dans la fenêtre "*Plot*" de R Studio (fenêtre inférieure droite), sélectionner "*Export*" puis le mode de sauvegarde choisi (format image avec l'extension de votre choix, format PDF ou presse-papier).

1.9.2 Travail avec plusieurs graphiques

Il peut être intéressant de placer plusieurs graphiques les uns à côté des autres dans une même image. Pour ce faire, on utilise la fonction `par` qui modifie les paramètres des fonctions de graphique. De nombreux paramètres peuvent être modifiés ; nous encourageons le lecteur à consulter l'aide pour avoir des renseignements sur les autres possibilités de la fonction `par`. Dans les exemples repris ci-après, nous utilisons la commande `par(mfrow=c(n,p))` pour stipuler à `R` de tracer les $n \times p$ graphiques dans un tableau de n lignes et p colonnes, en progressant ligne par ligne. Si on désire créer un tableau de graphiques en progressant colonne par colonne, on utilise alors l'instruction `par(mfcol=c(n,p))`.

1.10 Les librairies

La majorité des commandes présentées ici font partie du package de base, installé par défaut dans `R`. Néanmoins, un nombre croissant de personnes utilisent ce logiciel, et de ce fait de nouvelles fonctions, commandes et autres packages (ou librairies) sont créés par leurs utilisateurs. Généralement, ceux-ci sont disponibles directement par le logiciel `R`. Il est dès lors possible de télécharger ces nouveaux packages et de bénéficier de nouvelles fonctions. Pour ce faire, il suffit de télécharger les nouveaux packages via la commande `install.packages("nom")` ou via l'onglet "*Tools*" et puis "*Install packages*" de R Studio. Cette installation ne doit être réalisée qu'une seule fois. Ensuite, afin d'utiliser une nouvelle librairie, il faut entrer la commande `library("nom")`. Ainsi, les fonctions enregistrées dans le package sont accessibles à l'utilisateur.

Chapitre 2

Organisation et représentation des données

Rappelons que la façon de présenter les données diffère en fonction du type (qualitatif, quantitatif discret et quantitatif continu) de la variable. Les trois cas vont être considérés séparément dans trois sections différentes. Avant toute chose, il faut également s'assurer que le logiciel `R` identifie correctement les différents types de variables. Pour ce faire, la fonction `summary(x)` permettra de faire la distinction entre variable quantitative (un résumé quantitatif sera donné) et variable qualitative (un tableau des effectifs sera donné).

2.1 Variable qualitative

Si la variable X étudiée est qualitative, son analyse commence par la construction d'un tableau statistique mettant en évidence les effectifs et fréquences. Pour obtenir les effectifs relatifs à chacune des modalités, il suffit d'employer la commande `table`. Considérons par exemple la série suivante, enregistrant les parfums de glace choisis par 20 enfants. L'encodage dans `R` se fait par les commandes suivantes :

```
> parfum <- c("vanille", "vanille", "fraise", "vanille", "vanille",
"chocolat", "chocolat", "citron", "vanille", "moka", "vanille",
"vanille", "chocolat", "citron", "chocolat", "fraise", "citron",
"vanille", "moka", "chocolat")
> parfum <- factor(parfum)
```

La seconde commande permet de préciser à `R` que les éléments cités dans le vecteur sont des modalités ("factor") d'une variable qualitative.

L'ensemble des modalités s'obtient par la commande `unique` ou à l'aide de la commande `levels` qui permet d'obtenir les modalités triées par ordre alphabétique :

```
> levels(parfum)
```

qui donne

```
[1] "chocolat" "citron" "fraise" "moka" "vanille"
```

La commande `table` fournit les effectifs relatifs à chaque modalité présente dans l'échantillon :

```
> eff <- table(parfum)
```

Le résultat est le vecteur suivant :

```
 parfum
 chocolat  citron  fraise  moka  vanille
 5         3       2       2     8
```

Notez que les effectifs peuvent également être obtenus à partir de la commande

```
summary(parfum)
```

qui permet, en même temps, de vérifier que la variable est bien considérée comme une variable qualitative par le logiciel `R`.

Les fréquences peuvent être obtenues de deux façons : soit on divise les effectifs par l'effectif total via la commande

```
> freq <- eff/sum(eff)
```

soit on utilise la commande suivante

```
> freq <- prop.table(eff)
```

Dans les deux cas, le résultat est

```
 parfum
 chocolat  citron  fraise  moka  vanille
 0.25      0.15    0.10    0.10  0.40
```

Le tableau statistique traditionnel combinant ces informations peut être obtenu en juxtaposant les deux vecteurs :

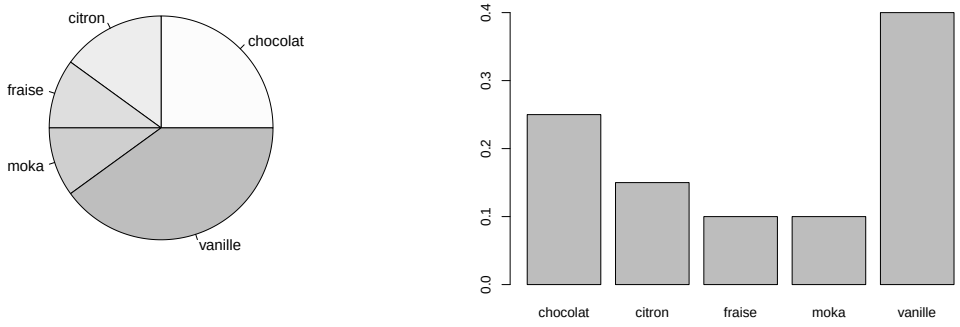
```
> rbind(eff, freq)
```

ce qui donne le résultat

```
       chocolat  citron  fraise  moka  vanille
 eff   5         3       2       2     8
 freq  0.25      0.15    0.1     0.1   0.4
```

Pour représenter la distribution des effectifs ou des pourcentages, un diagramme en secteurs et un diagramme en barres peuvent être représentés. Pour tracer un diagramme en secteurs, on utilise la fonction `pie`. On transmet à cette fonction un vecteur dont les composantes correspondent aux surfaces des différents secteurs. On peut également transmettre, à l'aide de l'argument `labels`, un vecteur contenant les noms des différentes sections. Enfin, l'argument `col` permet d'associer une couleur à chaque secteur. Considérons l'exemple suivant (représenté graphiquement ci-dessous à gauche de l'image) :

```
> parfum.nom <- levels(parfum)
> parfum.vente <- freq
> parfum.couleur <- c("gray99", "gray93", "gray87", "gray81", "gray75")
> pie(parfum.vente, labels=parfum.nom, col=parfum.couleur)
```



Une autre façon de représenter les variables qualitatives est d'utiliser la commande `barplot` pour obtenir un diagramme en barres. Sur le même exemple, la commande

```
> barplot(parfum.vente, names=parfum.nom)
```

permet d'obtenir le diagramme à droite ci-dessus. L'option `names=...` demande à `R` de placer les noms des modalités sous les barres correspondantes. L'option `col=...` existe également. Il est aussi possible de réaliser ces deux graphiques au départ des effectifs.

2.2 Variable quantitative discrète

L'étude d'une variable discrète commence également par la construction du tableau statistique. Considérons par exemple la série suivante, enregistrant les tailles de 15 ménages observés lors d'une étude particulière :

```
> ménage <- c(6,4,3,5,3,4,1,4,5,2,3,4,3,2,4)
```

Les effectifs et fréquences se calculent de la même façon que pour une variable qualitative, à l'aide de la commande `table` :

```
> eff <- table(ménage)
> freq <- eff/sum(eff)
```

Pour une variable quantitative discrète, il est également intéressant de calculer les effectifs et fréquences cumulés. Ceux-ci peuvent être obtenus à l'aide de la commande `cumsum` (pour *cumulative sum*). A l'aide des données encodées précédemment, on utilise le code suivant :

```
> effcum <- cumsum(eff)
```

et l'on obtient :

```
 1  2  3  4   5   6
 1  3  7  12  14  15
```

Pour obtenir les fréquences cumulées, il suffit de diviser les effectifs cumulés par la taille de l'échantillon (donnée par la fonction `length(ménage)`) ou d'effectuer la somme cumulée des fréquences :

```
> freqcum <- effcum / length(ménage)
```

ou

```
> freqcum <- cumsum(freq)
```

Les fréquences cumulées sont

```
     1          2          3          4          5          6
0.06666667 0.20000000 0.46666667 0.80000000 0.93333333 1.00000000
```

Enfin, il est plus commode de regrouper toutes ces informations au sein d'un seul tableau, avec en lignes les différentes modalités, et en colonnes les effectifs, effectifs cumulés, fréquences et fréquences cumulées. On peut obtenir un tel tableau grâce à la juxtaposition des vecteurs colonnes :

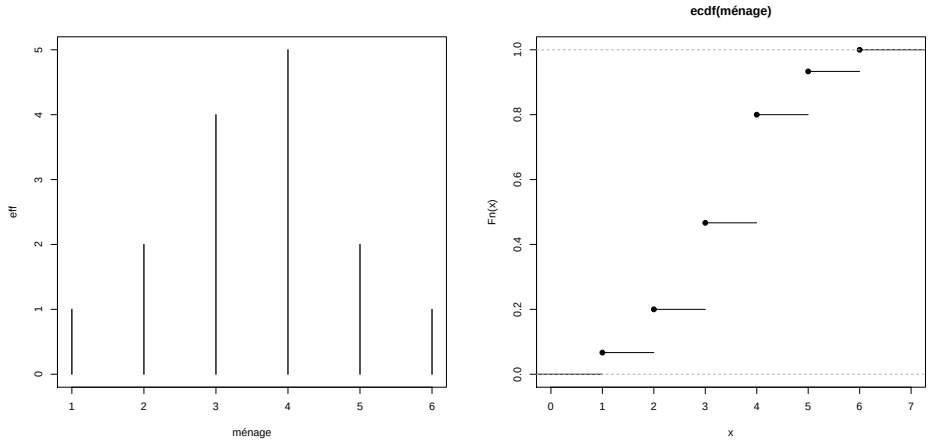
```
> cbind(eff, effcum, freq, freqcum)
```

Le résultat final (fourni par `R`) est le suivant

```
   eff  effcum     freq      freqcum
1   1      1    0.06666667  0.06666667
2   2      3    0.13333333  0.20000000
3   4      7    0.26666667  0.46666667
4   5     12    0.33333333  0.80000000
5   2     14    0.13333333  0.93333333
6   1     15    0.06666667  1.00000000
```

La représentation naturelle de la distribution des effectifs ou fréquences d'une variable discrète est le diagramme en bâtons. La fonction `plot` permet de réaliser des graphiques de base. Le diagramme en bâtons est obtenu en utilisant les effectifs ou les fréquences et l'argument `ecdf(...)` permet de tracer la fonction de répartition empirique (ou courbe cumulative des fréquences cumulées -*empiricial cumulative distribution function*) de la série. Nous donnons ci-dessous les commandes de `R` pour représenter les données créées auparavant.

```
> plot(eff)
> plot(ecdf(ménage))
```



2.3 Variable quantitative continue

Si la variable étudiée est continue, la commande `table` n'est pas adaptée car elle identifie chaque valeur distincte comme étant une valeur unique. Un tel tableau statistique n'est donc pas facile à lire, surtout si le nombre d'observations est élevé. Dans le cas d'une variable continue, un regroupement en classes est plus adéquat. Une solution consiste à utiliser la commande `hist` permettant de créer un histogramme (cette commande sera vue en détails plus tard dans cette section).

Supposons disposer des mesures du poids de 20 personnes (en kilos). Soit `poids` le vecteur contenant ces valeurs :

```
> poids <- c(59.4, 63.8, 67.6, 81.3, 77.3, 74.9, 56.7, 62.1, 73.7,
92.2, 69.5, 78.3, 84.3, 70.2, 65.8, 66.2, 65.3, 68.1, 72.1, 103.2)
```

Supposons à présent vouloir regrouper ces poids en cinq classes : $[55, 65]$, $]65, 75]$, $]75, 85]$, $]85, 95]$ et $]95, 105]$. La commande à utiliser est la suivante :

```
> eff <- hist(poids, breaks=c(55,65,75,85,95,105), plot=F)$counts
```

qui donne le résultat suivant

```
[1] 4 10 4 1 1
```

Quelques explications sur la commande `hist` et ses arguments :

1. En premier élément de la commande doit figurer le vecteur contenant les valeurs étudiées ; les autres arguments peuvent être ajoutés dans n'importe quel ordre.
2. `plot=F` signifie que `R` ne produira pas le graphique mais fournira les résultats relatifs à ce diagramme.
3. `breaks=c(...)` spécifie l'ensemble des valeurs limites des différentes classes. Ici, comme nous souhaitons quatre classes, nous avons dû intégrer les cinq valeurs formant les limites de ces classes.
4. Enfin, l'élément `$counts` se trouvant après la commande nous permet de ne sélectionner que les effectifs des classes dans l'ensemble des résultats produits par `R`.

Notons que si l'on souhaite considérer les classes avec les bornes inférieures comprises et les bornes supérieures exclues (i.e., en inversant les crochets), il suffit d'ajouter l'argument `right=FALSE` dans la commande `hist` (notons que dans ce cas, cette distinction n'est pas nécessaire vu qu'aucune valeur observée ne se trouve à la limite des classes).

A partir du vecteur `eff`, les fréquences, les effectifs cumulés et les fréquences cumulées s'obtiennent avec les mêmes manipulations que précédemment :

```
> freq <- eff/sum(eff)
> effcum <- cumsum(eff)
> freqcum <- cumsum(freq)
```

Il est ensuite aisé de représenter la table statistique classique à laquelle nous pouvons ajouter les bornes des classes :

```
> borne_Inf <- c(55,65,75,85,95)
> borne_Sup <- c(65,75,85,95,105)
> cbind(borne_Inf, borne_Sup, eff, effcum, freq, freqcum)
```

ce qui donne comme résultat final

borne_Inf	borne_Sup	eff	effcum	freq	freqcum
55	65	4	4	0.20	0.20
65	75	10	14	0.50	0.70
75	85	4	18	0.20	0.90
85	95	1	19	0.05	0.95
95	105	1	20	0.05	1.00

Pour représenter la distribution des effectifs ou fréquences d'une variable continue, le diagramme en "tiges et feuilles" (*Stem-and-leaf plot*) et l'histogramme peuvent être exploités. Commençons par le diagramme en "tiges et feuilles". L'obtention d'un tel diagramme s'effectue à l'aide de la commande `stem`.

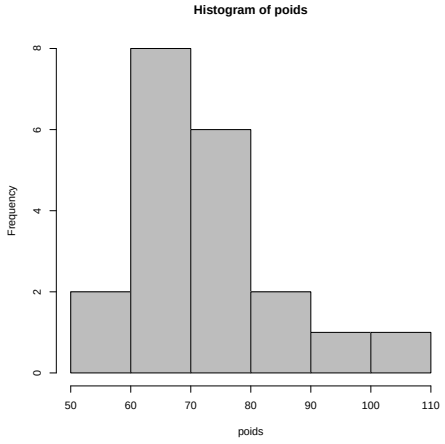
```
> stem(poids)
```

Le résultat fourni par R est le suivant

```
The decimal point is 1 digit(s) to the right of the |
5 | 79
6 | 2456688
7 | 0024578
8 | 14
9 | 2
10 | 3
```

Pour créer un histogramme à partir d'un vecteur de données, on utilise la fonction `hist`. R définira alors les classes de la forme $]a, b]$ et tracera l'histogramme des effectifs. On peut utiliser le paramètre `col="gray"` pour dessiner les rectangles en gris. Comme déjà énoncé, si l'on souhaite obtenir des classes de la forme $[a, b[$, il faut ajouter l'argument `right=FALSE`. On peut également définir le nombre approximatif de classes que R doit prendre en considération, par exemple `hist(poids, 5)`, ou en lui spécifiant les points séparant deux classes au moyen d'un vecteur, par exemple `hist(poids, breaks=c(55,65,75,85,95,105))`. Notez que dans ce cas, il faut préciser la borne inférieure de la première classe et la borne supérieure de la dernière classe. Reprenons l'ensemble de données `poids` ; voici un exemple dans lequel on a laissé à R le choix du nombre de classes.

```
> hist(poids, col="gray")
```

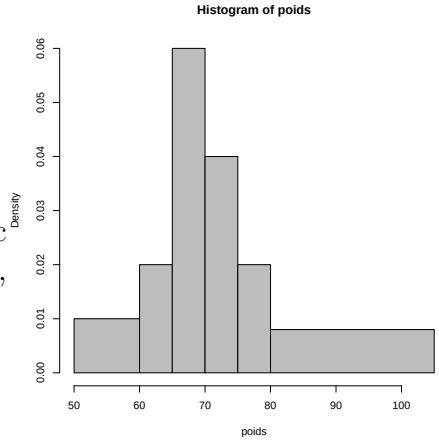


On remarque par exemple que la classe $[50, 60]$ contient 2 observations et que la classe $]60, 70]$ contient 8 observations (ce qui peut se voir directement depuis le vecteur `poids`). Plutôt que de tracer l'histogramme des effectifs, il est également possible de tracer l'histogramme d'aire unitaire en utilisant le paramètre `freq=FALSE` ou, de manière équivalente, `proba=TRUE`.

Si des classes non équidistantes sont utilisées, la commande `hist` réalisera par défaut un histogramme d'aire unitaire. Ainsi, avec la commande

```
> hist(poids, c(50,60,65,70,75,80,105),
col="gray", right=F)
```

on obtient un histogramme d'aire unitaire avec 6 classes déterminées par les bornes $[50, 60[$, $[60, 65[$, $[65, 70[$, $[70, 75[$, $[75, 80[$ et $[80, 105]$.



Remarque : Par défaut, `R` *utilise l'anglais et désigne l'effectif par "frequency" et la fréquence par "density". Ne vous trompez pas ! Vous pouvez néanmoins modifier ces appellations en utilisant l'argument* `ylab="..."` *dans la commande* `hist`.

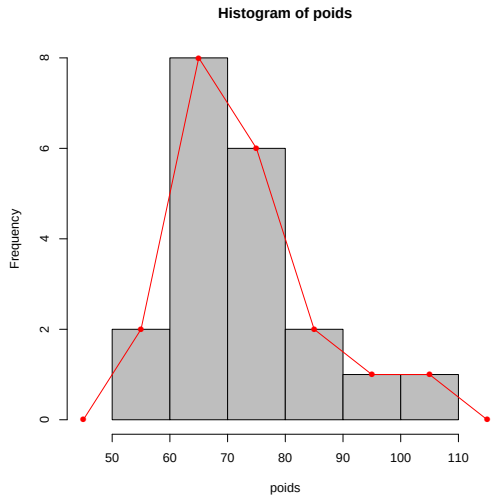
Le polygone des effectifs ou des fréquences peut être ajouté à l'histogramme. Pour ce faire, reprenons tout d'abord l'histogramme dont les classes ont toutes la même amplitude.

```
> h1 <- hist(poids, col="gray", xlim=c(45,115))
```

Dans ce cas, les sommets du polygone des effectifs ont pour abscisses les centres des classes, donnés par `h1$mids`, et pour ordonnées les effectifs des classes, donnés par `h1$counts`. Pour compléter le polygone, il faut ajouter les points de coordonnées $(0, 45)$ et $(0, 115)$. Ainsi, la commande

```
> lines(c(45,h1$mids,115), c(0,h1$counts,0), type="o", pch=16, col=2)
```

permet d'ajouter le polygone des effectifs à l'histogramme déjà représenté et d'obtenir le graphique ci-dessous.



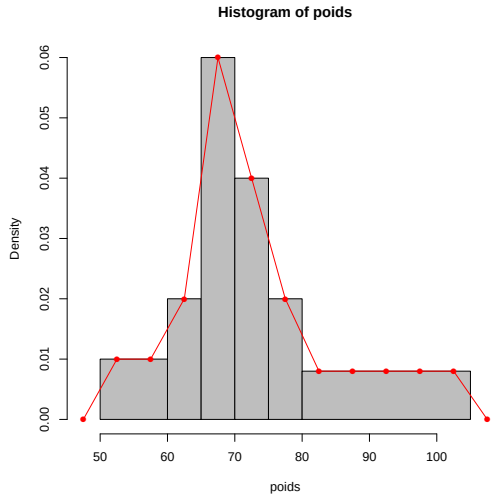
Reprenons à présent l'histogramme construit sur bases de classes ayant des amplitudes différentes.

```
> h2 <- hist(poids, c(50,60,65,70,75,80,105), col="gray", right=F,
xlim=c(47.5,107.5))
```

Il faut cette fois considérer les fréquences ajustées et non plus les effectifs. Les fréquences ajustées des différentes classes sont données par `h2$density`. Cependant, puisque les classes sont d'amplitudes différentes, relier les milieux des côtés supérieurs des rectangles de l'histogramme ne suffit pas. Il faut redéfinir les vecteurs des abscisses et ordonnées. Ainsi, les commandes

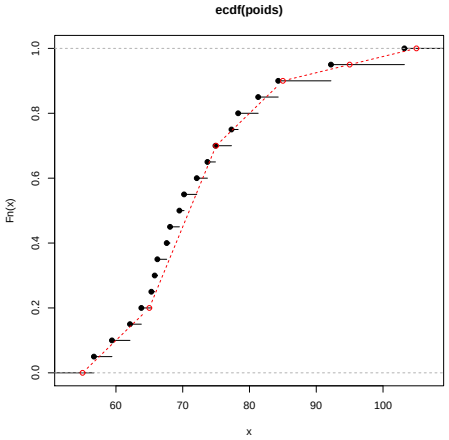
```
> x <- seq(47.5,107.5, by=5)
> y <- rep(c(0,h2$density,0), c(1,2,1,1,1,1,5,1))
> lines(x, y, type="o", pch=16, col=2)
```

permettent d'ajouter le polygone des effectifs à l'histogramme déjà représenté et d'obtenir le graphique ci-dessous.



La distribution des effectifs cumulés ou des fréquences cumulées peut être représentée de plusieurs façons. Si toutes les données sont disponibles, il est possible de représenter la fonction de répartition empirique à l'aide de la commande `ecdf` (comme dans le cas discret). Cependant, si nous ne disposons pas des données brutes mais uniquement d'une répartition en classes, l'ogive des effectifs cumulés ou des fréquences cumulées peut facilement être dessinée en exploitant la commande `plot(X,Y)` qui, en toute généralité, représente les points (x_i, y_i). Dès lors, en repartant de la définition du vecteur `eff` contenant les effectifs ou du vecteur `freq` contenant les fréquences et d'un vecteur `bornes` contenant les bornes des classes, les opérations suivantes permettent de construire l'ogive des effectifs cumulés ou celle des fréquences cumulées ajoutée au graphique ci-dessous (attention, vu que les vecteurs `eff` et `bornes` n'ont pas le même nombre de composantes, il faut ajouter un effectif nul en première position pour correspondre à la borne inférieure de la première classe) :

```
> plot(ecdf(poids))
> bornes <- c(55,65,75,85,95,105)
> lines(bornes, c(0,freqcum), type='b',
col='red', lty=2)
```



Chapitre 3

Réduction des données

3.1 Paramètres de position

Dans le cadre de ce cours, différents paramètres de position sont étudiés, à savoir la moyenne arithmétique (version classique, pondérée ou tronquée), la médiane (ainsi que les quantiles) et le mode. Les commandes relatives aux deux premiers paramètres sont assez simples et détaillées ci-dessous. Pour le dernier, il suffit d'étudier les tableaux statistiques ou les graphiques expliqués dans le chapitre précédent.

Pour commencer, la commande `mean` rend la moyenne des composantes d'un vecteur. Ainsi, en reprenant les données du chapitre précédent, on obtient

```
> mean(poids)
[1] 72.6
```

Cette commande permet également de calculer les moyennes tronquées à l'aide de l'option `trim=`α où α est le seuil de la moyenne tronquée (proportion d'observations qui sont tronquées de chaque côté de la distribution). Ainsi, si on retire 1%, à savoir 2 observations de chaque côté, la moyenne devient

```
> mean(poids, trim=0.1)
[1] 71.28125
```

Passons désormais à la médiane qui est donnée par la commande `median`. Sur l'exemple du vecteur des poids, on obtient

```
> median(poids)
[1] 69.85
```

La médiane peut également être calculée à partir de la commande `quantile` qui permet de calculer n'importe quel quantile empirique pour un vecteur donné. Il suffit de lui indiquer comme argument le vecteur utilisé et une ou plusieurs proportion(s) correspondant à la/les proportion(s) d'observations qui lui seront inférieures. Par exemple, si on souhaite obtenir les quartiles Q_1 et Q_3 ainsi que la médiane du vecteur `poids`, il suffit de considérer les proportions 0.25, 0.5 et 0.75.

```
> quantile(poids, c(0.25,0.5,0.75))
 25%     50%     75%
 65.675  69.850  77.550
```

Ainsi, nous obtenons $Q_1 = 65.675$, $\tilde{x} = 69.850$ et $Q_3 = 77.55$ pour le vecteur des poids. Notez que le minimum et le maximum d'un vecteur correspondent à des proportions 0 et 1. Ainsi, les deux commandes

```
> quantile(poids, c(0, 1))
 0%    100%
 56.7  103.2
```

ou

```
> range(poids)
[1] 56.7 103.2
```

permettent d'obtenir les valeurs minimale et maximale du vecteur donné.

Pour finir, notez que la commande `summary` appliquée à un vecteur issu d'une variable quantitative [1] permet d'avoir directement plusieurs informations sur sa distribution, à savoir le résumé à cinq valeurs (minimum, premier quartile, médiane, troisième quartile et maximum) ainsi que la moyenne.

```
> summary(poids)
 Min.   1st Qu.  Median  Mean   3rd Qu.  Max.
 56.70  65.67    69.85   72.60  77.55    103.20
```

3.2 Paramètres de dispersion et de forme

Différents paramètres peuvent être utilisés afin de décrire la dispersion de données quantitatives. Dans un premier temps, il est possible de caractériser des données en déterminant son étendue, à savoir la différence entre le maximum et le minimum. Celle-ci est facilement calculable. Par exemple, pour les données du poids, nous avons obtenu ci-dessus les valeurs minimale et maximale à l'aide de la commande `range`. Ainsi, l'étendue est $103.2 - 56.7 = 44.3$.

Un autre paramètre qui est également défini à partir de la série ordonnée est l'écart interquartile. Celui-ci peut être calculé manuellement à partir des quartiles ou via la commande `IQR` (*Interquartile range*). Par exemple, pour le poids, EIQ $= Q_3 - Q_1 = 77.55 - 65.675 = 11.875$ ou directement

```
> IQR(poids)
[1] 11.875
```

1. Comme mentionné au début de ce chapitre, la commande `summary` permet également de distinguer les variables quantitatives des variables qualitatives pour lesquelles les effectifs de chaque modalité sont donnés.

L'étendue et l'écart interquartile exploitent essentiellement les observations occupant certains rangs particuliers dans les séries ordonnées. D'autres paramètres de dispersion utilisent quant à eux toutes les observations. C'est notamment le cas du paramètre le plus courant, à savoir la variance. Il s'agit de la moyenne des carrés des écarts entre les observations et leur moyenne arithmétique. Plus précisément, `var` rend la variance des composantes d'un vecteur [2]. Au vu de sa définition, la variance s'exprime dans le carré de l'unité utilisée pour les valeurs observées. C'est pourquoi on lui préfère généralement l'écart-type qui est défini comme la racine carrée de la variance. Dans `R`, celui-ci est donné par la commande `sd` (*standard deviation*). Pour le vecteur poids, on obtient

```
> var(poids)
[1] 126.8568
> sd(poids)
[1] 11.26307
```

On peut facilement vérifier que l'écart-type est bien la racine carrée de la variance :

```
> sqrt(var(poids))
[1] 11.26307
```

De plus, il est parfois utile de vérifier que les fonctions implémentées dans `R` sont correctes. Ces vérifications peuvent être obtenues très rapidement en combinant la fonction `mean` et le caractère vectoriel des opérations dans le logiciel `R`. Par exemple, la commande

```
> mean( (poids-mean(poids))^2 )
[1] 120.514
```

permet de calculer la variance du vecteur poids telle que définie dans le cadre de ce cours. Si l'on décompose cette commande, `x-mean(x)` donne le vecteur des observations centrées en 0, `(x-mean(x))^2` donne le vecteur des carrés de ces observations centrées en 0 et `mean((x-mean(x))^2)` donne finalement la moyenne du vecteur des puissances carrées des observations centrées en 0.

De manière similaire, certains paramètres statistiques comme les différents moments (centrés ou non), le coefficient de variation,... peuvent être obtenus très rapidement en combinant les fonctions `mean`, `sd`, etc et le caractère vectoriel des opérations dans le logiciel `R`. Ainsi, les coefficients de dissymétrie et d'aplatissement peuvent être calculés "à la main". Cependant, la libraire `moments` permet d'obtenir directement le coefficient de dissymétrie de Fisher ainsi que le coefficient d'aplatissement de Pearson via les commandes suivantes.

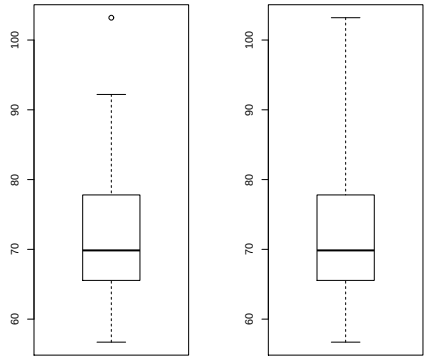
2. Attention : `R` calcule la variance en divisant la somme des écarts à la moyenne par $n-1$ au lieu de n. L'écart-type est donné par la racine carrée de ce paramètre.

```
> library(moments)
> skewness(poids)
[1] 1.10963
> kurtosis(poids)
[1] 4.030091
```

3.3 Boites à moustaches

Pour tracer une boite à moustaches, on utilise la commande `boxplot`. L'argument à fournir est un vecteur contenant toutes les observations. Nous représentons ci-dessous deux boites à moustaches, associées au vecteur `poids`.

```
> par(mfrow=c(1,2))
> boxplot(poids)
> boxplot(poids, range=0)
> par(mfrow=c(1,1))
```



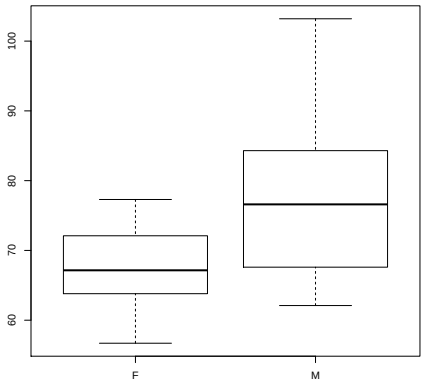
Voici quelques remarques sur la commande utilisée [3]. Par défaut, `R` crée la boite à moustaches ajustée : le graphique de gauche. Dès lors, les valeurs extrêmes ne sont plus prises en compte pour la représentation des moustaches, mais elles restent représentées sur le graphe par des cercles. Si vous souhaitez toutefois représenter la boite à moustaches classique, i.e., pour laquelle les extrémités des moustaches correspondent aux valeurs extrêmes, vous devez ajouter l'argument `range=0` dans la commande `boxplot`. Le graphique de droite représente cette boite, pour les mêmes données.

Il est également possible de représenter sur la même échelle la distribution d'une variable quantitative conditionnellement à chaque modalité d'une variable qualitative.

3. La première commande `par` sert à modifier les paramètres graphiques définis par défaut (voir Section **??**) et `mfrow=c(1,2)` à obtenir un tableau 1×2 de graphiques que l'on remplit par ligne. La dernière commande permet de revenir à l'option par défaut (tableau 1×1). Supprimer tous les graphiques permet également de revenir à l'option par défaut.

Par exemple, si nous connaissons le sexe des 20 personnes dont le poids est repris dans le vecteur `poids`, les commandes suivantes permettent d'obtenir les boites à moustaches des poids pour les femmes et pour les hommes séparément.

```
> sexe <- c('F', 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'F', 'M', 'M', 'M',
'M', 'F', 'F', 'F', 'M', 'F', 'F', 'M')
> sexe <- factor(sexe)
> boxplot(poids ~ sexe)
```



Si vous disposez de plusieurs vecteurs quantitatifs basés sur la même échelle, vous pouvez représenter des boites provenant de ceux-ci dans un même cadre. Celles-ci seront alors représentées avec la même échelle. Par exemple, pour inclure les boites relatives aux séries des poids des femmes et des hommes à partir de deux vecteurs distincts `poidsF` et `poidsM`, le même graphique que précédemment peut être obtenu via les commandes suivantes.

```
> poidsF <- poids[sexe == "F"]
> poidsM <- poids[sexe == "M"]
> boxplot(poidsF, poidsM)
```

Plusieurs arguments optionnels peuvent s'avérer utiles. Tout d'abord, `horizontal= TRUE` présente la boite à moustaches horizontalement (par défaut, elle est dessinée verticalement). De plus, `plot=FALSE` vous permet d'obtenir un résumé de l'information nécessaire à la construction de la boite (l'effectif n, les quartiles, ...). Différentes options supplémentaires sont disponibles ; nous invitons le lecteur intéressé à consulter l'aide sur cette fonction.

3.4 Traitement des données manquantes

La présence de valeurs manquantes (pour rappel, dans `R`, les données manquantes sont codées `NA` pour *not available*) dans un ensemble de données peut, dans certains

cas, poser problème lors de l'exécution de certaines commandes. Nous allons décrire plusieurs situations ci-dessous.

Pour commencer la plupart des graphiques et certaines fonctions s'exécutent sans tenir compte des valeurs manquantes présentes (elles sont juste ignorées). Par exemple, si nous ajoutons une valeur manquante à notre vecteur des `poids`, celle-ci ne perturbe pas et n'apparait pas dans les résultats des différentes commandes citées, de manière non exhaustive, ci-dessous :

```
> x <- c(poids, NA)
> hist(x)
> plot(ecdf(x))
> table(x)
> boxplot(x)
```

Lorsque la fonction agit sur un réel (ou séparément sur chaque composante d'un vecteur, grâce au caractère vectoriel du langage `R`), une donnée manquante induit un résultat manquant mais n'affecte pas les résultats des autres observations. Par exemple, si on souhaite calculer le logarithme de chaque composante du vecteur `x` précédemment défini, tous les logarithmes sont donnés, sauf le dernier pour lequel la valeur est manquante `NA`.

```
> log(x)
[1] 4.084294 4.155753 4.213608 4.398146 4.347694 4.316154 4.037774
[8] 4.128746 4.300003 4.523960 4.241327 4.360548 4.434382 4.251348
[15] 4.186620 4.192680 4.178992 4.220977 4.278054 4.636669 NA
```

Un comportement similaire est observé avec les fonctions telles que `exp`, `sin`, `cos`, `tan`, `sqrt`, exposant (`a^b`), soustraction (`a-b`), addition (`a+b`) ou division (`a/b`) composantes à composantes,...

Par contre, pour certaines fonctions qui prennent comme argument un vecteur afin de rendre une unique valeur (i.e. des fonctions $f : \mathbb{R}^n \mapsto \mathbb{R}$), la présence d'une donnée manquante empêche la commande d'effectuer le calcul et donne comme résultat `NA`. C'est notamment le cas des fonctions `sum`, `prod`, `min`, `max`, `mean`, `median`, `var`, `sd`, `quantile`,... Cependant, ces fonctions ont à disposition l'option `na.rm=TRUE` qui permet de retirer les valeurs manquantes lors de l'évaluation de la fonction. Par exemple, si nous reprenons notre vecteur `x` défini à partir du vecteur `poids`, on obtient les résultats suivants.

```
> sum(x)
[1] NA
> sum(x, na.rm=TRUE)
[1] 1452
```

Cependant, cette option n'est pas disponible pour toutes les commandes [4]. Dans ce cas, il est important de traiter préalablement les données manquantes **avant** d'exécuter ces commandes. Il y a plusieurs manières de traiter cette absence d'information. Une première idée, la plus simple, est de retirer les données manquantes afin de procéder à l'analyse voulue. Cependant, d'autres méthodes peuvent être plus adaptées. Par exemple, estimer une valeur à imputer à la place d'une valeur manquante. Dans ces notes, nous nous contenterons d'exposer la méthode la plus simple qui consiste à supprimer les observations (i.e. les lignes) de la base de données qui ont au moins une valeur manquante. La commande `na.omit` permet de définir un sous-ensemble pour lequel toutes les lignes avec au moins une valeur manquante sont retirées.

Considérons par exemple un ensemble de données relatives à différentes voitures (fichier nommé `car2.txt`). Pour démarrer, il est conseillé d'exécuter la commande `summary` sur l'ensemble des données afin de vérifier le type des différentes variables ainsi que la présence de valeurs manquantes.

```
> data <- read.table("cars2.txt", header=TRUE)
> summary(data)
           Make  Fuel        Doors     Length         Width
    toyota : 32  diesel: 20  four:114  Min. :141.1    Min. :60.30
    nissan : 18  gas :185    two : 89  1st Qu.:166.3  1st Qu.:64.10
     mazda : 17              NA's: 2   Median :173.2  Median :65.50
     honda : 13                        Mean :174.0    Mean :65.91
 mitsubishi: 13                        3rd Qu.:183.1  3rd Qu.:66.90
    subaru : 12                        Max. :208.1    Max. :72.30
   (Other) :100
    Horsepower  Price
   Min. : 48.0  Min. : 5118
 1st Qu.: 70.0  1st Qu.: 7775
 Median : 95.0  Median :10295
   Mean :104.3  Mean :13207
 3rd Qu.:116.0  3rd Qu.:16500
   Max. :288.0  Max. :45400
       NA's :2  NA's :4
```

Au vu de la sortie de la commande `summary`, nous pouvons observer que nous sommes en présence de trois variables qualitatives (`Make`, `Fuel` et `Doors`) car les effectifs des différentes modalités sont donnés ainsi que quatre variables quantitatives (`Length, Width, Horsepower` et `Price`) pour lesquelles le résumé à cinq valeurs est donné. De plus, nous pouvons observer qu'il y a deux valeurs manquantes pour la variable `Doors`, deux pour la variable `Horsepower` et quatre pour `Price`.

4. En cas de doute lors de l'utilisation d'une fonction, nous invitons le lecteur à consulter l'aide de `R`.

Il est possible de supprimer les lignes où apparaissent ces données manquantes grâce à la commande `na.omit`.

```
> data2 <- na.omit(data)
> dim(data)
[1] 205 7
> dim(data2)
[1] 197 7
```

Au vu des dimensions des deux ensembles de données, il est clair que les huit données manquantes étaient présentes sur des lignes différentes car huit lignes sont supprimées à l'aide de la commande `na.omit`. Il est ensuite possible de travailler sur ce sous-ensemble `data2` qui n'a pas de données manquantes.

Chapitre 4

Série statistique bivariée

4.1 Représentation des données

Lorsque deux variables sont étudiées simultanément, la distribution conjointe des effectifs ou celle des fréquences est décrite au moyen du tableau de contingence. Imaginons qu'avec la taille des ménages, on ait également observé le nombre de chambres du logement :

```
> ménage <- c(6,4,3,5,3,4,1,4,5,2,3,4,3,2,4)
> chambre <- c(3,4,2,5,4,3,1,3,4,1,2,4,3,2,5)
```

Le tableau de contingence des effectifs s'obtient au moyen des commandes

```
> tab <- table(ménage, chambre)
> tab
         chambre
 ménage  1  2  3  4  5
    1    1  0  0  0  0
    2    1  1  0  0  0
    3    0  2  1  1  0
    4    0  0  2  2  1
    5    0  0  0  1  1
    6    0  0  1  0  0
```

Le fait de sauvegarder le tableau de contingence sous l'objet `tab` n'est nécessaire que si l'on souhaite obtenir dans un second temps le tableau de contingence des fréquences. Cela peut se faire via la commande

```
> prop.table(tab)
```

Ces différents tableaux ne sont rien d'autre que des matrices dont les dimensions correspondent aux nombres de valeurs distinctes que peuvent prendre les deux variables. Ainsi, `tab` est une matrice de 6 lignes et 5 colonnes à laquelle toutes les opérations vues précédemment pour les matrices peuvent s'appliquer (transposée, extraction d'une

ou plusieurs composantes, ...). Cela peut s'avérer utile pour obtenir les distributions conditionnelles.

Il est également possible d'ajouter à ces tableaux les distributions marginales :

```
> addmargins(tab)
```

Le résultat final est alors une matrice avec une ligne et une colonne supplémentaires.

```
         chambre
 ménage  1  2  3  4  5  Sum
    1    1  0  0  0  0   1
    2    1  1  0  0  0   2
    3    0  2  1  1  0   4
    4    0  0  2  2  1   5
    5    0  0  0  1  1   2
    6    0  0  1  0  0   1
  Sum    2  3  4  4  2  15
```

Considérons un nouvel exemple : reprenons les données du poids de 20 personnes (en kg). Supposons qu'on dispose également de leur taille en cm. Ces données peuvent être représentées par un graphique bivarié poids/taille. Voici comment procéder :

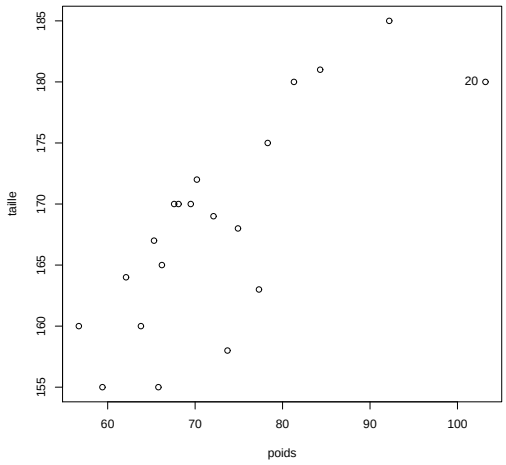
```
> poids <- c(59.4, 63.8, 67.6, 81.3, 77.3, 74.9, 56.7, 62.1, 73.7,
92.2, 69.5, 78.3, 84.3, 70.2, 65.8, 66.2, 65.3, 68.1, 72.1, 103.2)
> taille <- c(155, 160, 170, 180, 163, 168, 160, 164, 158, 185, 170,
175, 181, 172, 155, 165, 167, 170, 169, 180)
> plot(poids, taille)
```

Nous avons placé dans deux vecteurs, et dans le même ordre, le poids et la taille des différentes personnes. Ainsi, nous avons un premier individu mesurant 155 cm et pesant 59,4 kg. Il est donc important de conserver le même ordre à l'encodage pour les deux vecteurs. La dernière ligne de commande va tracer un graphique dont chaque point (p_i, t_i) représente un individu i de poids p_i et de taille t_i.

Si l'on souhaite connaitre l'indice d'une certaine observation, nous pouvons utiliser la commande `identify` afin d'obtenir cette information. Attention qu'il est important de quitter le graphique interactif après son utilisation afin de pouvoir continuer à utiliser le logiciel. Dans notre exemple, la commande

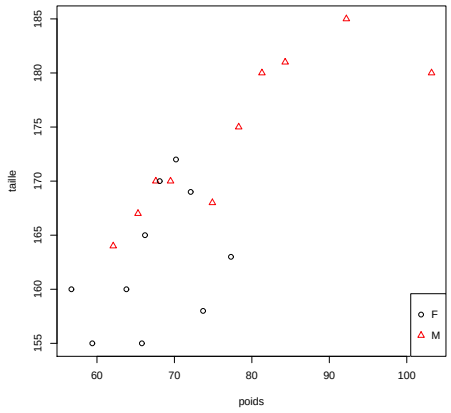
```
> identify(poids, taille)
[1] 20
```

permet par exemple d'aller détecter l'observation 20 qui s'éloigne du nuage de points. Cette commande indique le numéro de l'observation sur le graphique ainsi que dans le terminal.



Il est parfois intéressant de distinguer les individus en fonction d'une variable qualitative, par exemple le sexe. Dans ce cas, il suffit d'imposer des couleurs et/ou symboles différents pour chacune des modalités. En reprenant les sexes des 20 individus repris ci-dessous, les commandes suivantes permettent d'obtenir le graphique de droite.

```
> sexe <- c('F', 'F', 'M', 'M', 'F', 'M', 'F', 'M', 'F', 'M', 'M', 'M',
'M', 'F', 'F', 'F', 'M', 'F', 'F', 'M')
> sexe <- factor(sexe)
> plot(poids, taille, col=as.integer(sexe), pch=as.integer(sexe))
> legend(x="bottomright", col=1:nlevels(sexe), pch=1:nlevels(sexe),
levels(sexe))
```

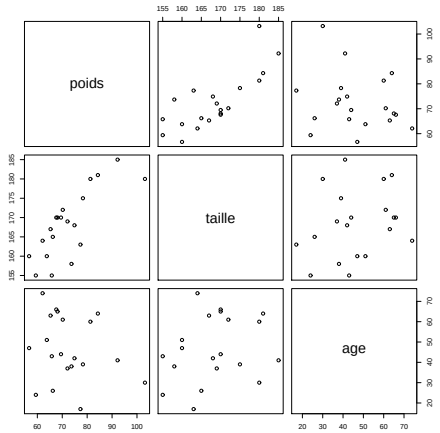


Si plus de deux variables quantitatives sont étudiées simultanément, on pourrait créer successivement des graphiques à l'aide de la commande `plot`. Toutefois, il est possible d'afficher toutes les combinaisons de deux variables en un seul graphique, appelé *scatter plot*. La commande à utiliser est `pairs`. Par exemple, si nous reprenons

nos données concernant 20 sujets, nous disposons déjà de leur taille (en centimètres, variable `taille`) et leur poids (en kilos, variable `poids`). Imaginons avoir également à disposition leur âge (en années, variable `age`). Les données sont reprises ci-dessous (sous forme de code `R`) ainsi que la commande pour le *scatter plot* :

```
> age <- c(24, 51, 66, 60, 17, 42, 47, 74, 38, 41, 44, 39, 64, 61, 43,
26, 63, 65, 37, 30)
> pairs(cbind(poids, taille, age))
```

Notez que l'argument de `pairs` doit être une matrice dont chaque colonne constitue une variable. Le résultat est présenté dans la Figure ci-dessous.



Par exemple, le graphique de la première ligne et de la deuxième colonne représente la variable `poids` en fonction de la `taille`, tandis que celui de la première colonne et deuxième ligne représente la `taille` en fonction du `poids`, et ainsi de suite. On peut voir qu'il y a un lien linéaire entre la taille et le poids : le nuage de points indique une tendance linéaire croissante. Par contre, il est évident que la variable `age` n'indique aucun lien particulier avec les deux autres variables considérées.

4.2 Corrélation et régression linéaire

Lorsqu'un nuage de points indique un lien linéaire entre deux variables, il est naturel de vouloir quantifier ce lien à l'aide d'une mesure comme la covariance ou la corrélation et d'essayer d'exprimer le lien sous forme d'une régression linéaire.

4.2.1 Covariance et corrélation

Pour calculer la corrélation entre deux variables, on utilise la commande `cor`. Ainsi,

```
> cor(poids, taille)
> cor(poids, age)
> cor(taille, age)
```

donnent respectivement 0.7804, -0.2051 et 0.2390. On constate donc une corrélation positive relativement forte entre la taille et le poids et des corrélations faibles entre l'âge et les deux autres variables, ce qui confirme l'observation visuelle précédente. Notons que, par définition de la corrélation, on peut aussi la calculer en utilisant le code suivant :

```
> cov(x,y) / (sd(x)*sd(y))
```

En effet, `cov` donne la covariance[1] entre deux variables, tandis que `sd` fournit l'écart-type de la variable considérée.

4.2.2 Régression linéaire

L'objectif de la régression linéaire est de déterminer la droite d'équation $y = a + b\,x$, où x et y sont respectivement les variables explicative et dépendante, b est la pente de la droite et a est l'ordonnée à l'origine (*intercept*).

Pour commencer, utilisons la méthode classique, à savoir la méthode des moindres carrés. Dans notre exemple, pour déterminer la droite de régression du `poids` par rapport à la `taille`, on utilise dans `R` le code suivant :

```
> lm(poids ~ taille)
```

(`lm` est l'abréviation de *linear model*). Le résultat fourni est une liste dont voici les deux premiers éléments :

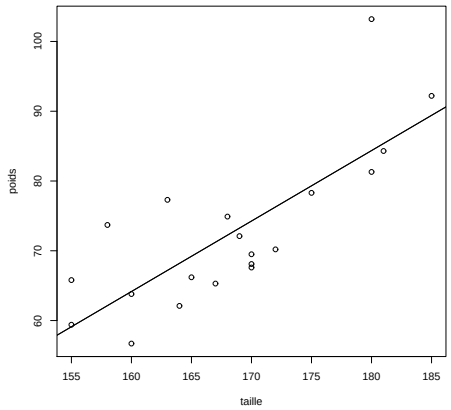
```
Call:
lm(formula = poids ~ taille)
Coefficients:
 (Intercept)  taille
 -97.45       1.01
```

On en déduit donc que $a = -97.45$ (ordonnée à l'origine) et $b = 1.01$ (pente). Notez que le paramètre b est référencé dans le code `R` par le nom de la variable `taille`, car il correspond au paramètre de cette variable. Nous obtenons donc bien une droite croissante ($b > 0$), comme prévu.

Il est possible de représenter la droite de régression superposée aux données. Pour ce faire, on utilise la commande `abline` comme suit (le résultat obtenu est représenté ci-dessous) :

1. Attention : Comme pour la variance, `R` calcule la covariance en divisant par $n - 1$ au lieu de n.

```
> plot(taille, poids)
> abline(-97.45, 1.01)
```



La commande `abline(u,v)` permet de superposer, sur un graphique existant, une droite dont l'ordonnée à l'origine vaut `u` et la pente est `v`. Il suffit donc de reprendre les valeurs obtenues grâce à la régression linéaire. Bien sûr, il est possible de modifier la couleur et le type de trait, entre autres, par l'ajout d'un argument (`col=...`, par exemple).

On peut cependant procéder un peu plus subtilement, en extrayant directement les valeurs des coefficients du modèle. Pour ce faire, on sauvegarde d'abord le modèle dans une variable, puis on extrait les coefficients qu'on place dans la commande `abline` :

```
> mod <- lm(poids ~ taille)
> abline(mod$coefficients)
```

Ces deux lignes effectuent le même travail que précédemment. L'utilisation de `$coefficients` permet d'extraire les paramètres du modèle (stocké dans `mod`).

On peut de plus extraire les valeurs prédites du modèle, i.e., les valeurs $\hat{y}_i = a + bx_i$, stockées dans la même liste sous le nom `fitted`. On utilise à cette fin la commande `...$fitted`. Par exemple, dans notre cas,

```
> mod$fitted
```

mène à

```
    1         2         3         4         5         6         7
59.11497  64.16555  74.26669  84.36783  67.19589  72.24646  64.16555
    8         9        10        11        12        13        14
68.20600  62.14532  89.41840  74.26669  79.31726  85.37795  76.28692
   15        16        17        18        19        20
59.11497  69.21612  71.23635  74.26669  73.25657  84.36783
```

De même, on obtient facilement les résidus des données $\delta_i = y_i - \hat{y}_i$, i.e., l'écart entre les valeurs observées et les valeurs prédites par le modèle ; on utilise pour cela la commande `...$residuals`. Ainsi,

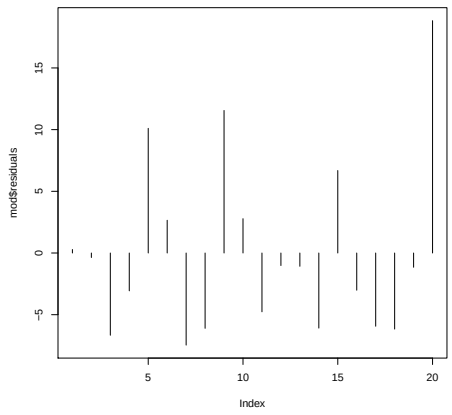
```
> mod$residuals
```

fournit le résultat

```
        1         2         3         4         5         6
0.2850266 -0.3655452 -6.6666887 -3.0678322 10.1041118 2.6535400
        7         8         9        10        11        12
-7.4655452 -6.1060026 11.5546835 2.7815961 -4.7666887 -1.0172604
       13        14        15        16        17        18
-1.0779465 -6.0869174 6.6850266 -3.0161169 -5.9363456 -6.1666887
       19        20
-1.1565743 18.8321678
```

que l'on peut représenter en utilisant un graphique indexé :

```
> plot(mod$residuals, type='h')
```



On constate directement que le modèle sous-estime le poids pour la cinquième observation car son résidu est positif tandis qu'il le surestime pour la septième personne car son résidu est négatif. De plus, le modèle semble le moins bien adapté pour l'observation 20 car son résidu est (en valeur absolue) le plus grand. Tout cela se confirme également visuellement à l'aide du graphique précédent ; il suffit de comparer la droite de régression avec les données réelles. Notez que si le nombre d'observations est relativement élevé, il sera difficile d'identifier exactement l'indice de chaque observation sur le graphique indexé des résidus. Dans ce cas, il serait intéressant d'utiliser la commande `identify` pour identifier les numéros des observations qui nous intéressent :

```
> identify(mod$residuals)
```

Enfin, le calcul du coefficient R^2 d'ajustement du modèle aux données est très facile à obtenir, puisqu'il correspond au carré du coefficient de corrélation entre les deux variables. Vu la section précédente, on trouve dans notre exemple un coefficient $R^2 = 0.7804^2 = 0.6090$. Cela signifie que 61% de la variabilité du `poids` est expliquée par la droite de régression par rapport à la variable `taille` ; cette dernière est dès lors un relativement bon prédicteur du poids. Notez que cette information est également disponible à l'aide de la commande

```
> summary(mod)
Call:
lm(formula = poids   taille)
Residuals:
 Min     1Q      Median  3Q     Max
 -7.465  -5.974  -1.117  2.686  18.832
Coefficients:
              Estimate  Std. Error  t value  Pr(>|t|)
 (Intercept)  -97.4528  32.1589     -3.030   0.00719   **
 taille       1.0101    0.1908      5.295    4.93e-05  ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 7.236 on 18 degrees of freedom
Multiple R-squared: 0.609, Adjusted R-squared: 0.5873
F-statistic: 28.03 on 1 and 18 DF, p-value: 4.929e-05
```

où, parmi d'autres choses, nous pouvons lire `Multiple R-squared: 0.609`.

Chapitre 5

Exemples de questions sur R

Cette liste de questions est indiquée à titre d'exemple. Il est évident que la matière du cours ne se limite pas à ces quelques questions.

Exemple 1

L'ensemble de données à considérer s'intitule `CommunesA.txt` et contient les informations suivantes obtenues sur 116 communes de Wallonie :
`Chomage` : le taux de chômage dans la commune (en %)
`Province` : la province de la commune (deux modalités : `Liège` et `Namur`)
`IndiceRichesse` : indicateur du niveau de richesse de la commune (val. de réf. : 100)
La colonne intitulée `Communes` reprend les noms des communes.

1. On s'intéresse dans un premier temps à la variable `Chomage`.
   (a) Représenter à l'aide d'un graphique adéquat la fonction de répartition empirique de cette série.
   (b) En faisant varier la répartition en classes, comparer la fonction de répartition empirique avec différentes ogives. Commenter.
   (c) A partir de la série brute, calculer la médiane, la moyenne ainsi que le coefficient de dissymétrie de Fisher de cette série. Décrire la forme de ces données
2. On souhaite comparer les taux de chômage dans les deux provinces. Pour obtenir ces deux séries à partir des données, définir dans `R`

   ```
   ChomageL <- Chomage[Province=="Liège"]
   ChomageN <- Chomage[Province=="Namur"]
   ```

   (a) Comparer les taux de chômage en représentant des boîtes à moustaches de la variable `Chomage` en fonction des modalités de la variable `Province` (les deux boîtes devant être représentées en une seule fois sur le même repère). Commenter ce graphique.
   (b) Pour la série de Liège, déterminer les valeurs pivots permettant de construire la boîte à moustaches et déterminer les noms de deux communes ayant permis de définir les valeurs adjacentes de gauche et de droite de cette boîte.

(c) Calculer la variance globale de la variable `Chomage`.

(d) Quelle part de la variance est dûe à la variabilité à l'intérieur des groupes (c'est-à-dire à l'hétérogénénité des communes entre elles) ?

3. A priori, il est naturel d'imaginer que l'indice de richesse d'une commune s'explique en partie par le taux de chômage dans la commune.

(a) A partir d'un outil adéquat, commenter l'adéquation de ce lien.

(b) Calculer la corrélation entre `IndiceRichesse` et `Chomage`. Commenter.

(c) Déterminer l'équation de la droite de régression de la variable `IndiceRichesse` par rapport à la variable `Chomage` lorsque ce paramètre est estimé par la technique des moindres carrés.

(d) Représenter le nuage de points de la variable `IndiceRichesse` en fonction de la variable `Chomage`, en distinguant les observations relatives aux différentes provinces, et y ajouter la droite de régression calculée.

(e) A partir du modèle linéaire ajusté ci-dessus, commenter l'adéquation du modèle pour la commune de Chaudfontaine.

(f) Représenter les résidus des observations en fonction de leur indice (observation n°1, 2, ...) et ajouter la droite $y = 0$ à ce graphique.

(g) Quelles sont les deux communes dont les résidus sont les plus grands en valeur absolue ? Commenter.

(h) On pourrait se demander si le lien entre les deux variables est le même dans chaque province. En définissant la nouvelle variable

```
IndiceRichesseL <- IndiceRichesse[Province=="Liège"]
```

déterminer l'équation de la droite de régression obtenue par la technique des moindres carrés lorsque seules les observations de la province de Liège sont considérées.

Quelle part de la variance de la variable `IndiceRichesse` est, dans ce cas, expliquée par la régression.

Références

Baclawski, K (2008) *Introduction to probability with R.* Boca Raton : Chapman & Hall, United States

Brostaux, Y (2002) *Introduction à l'environnement de programmation statistique R.* Faculté universitaire des Sciences Agronomiques de Gembloux, Belgique.

Magis, D. (2007) *Introduction au logiciel R - Première partie.* Département de Mathématique, Université de Liège, Belgique.

Paradis, E. (2002) *R pour les débutants.* Institut des Sciences de l'Evolution, Université Montpellier II, France.

Rigo, M. (1997) *Introduction à R.* Dossier n^o 9, G.E.M.M.E., Faculté d'Economie, de Gestion et de Sciences Sociales, Université de Liège, Belgique.