Aide à l'utilisation de **R** pour mettre en oeuvre les méthodes statistiques de base

Marie Laure Delignette-Muller

27 août 2020

Table des matières

1 Introduction

1.1 Qu'est-ce que R ?

R est un langage de programmation dédié à l'analyse statistique de données, qui peut être chargé librement et installé très facilement par exemple à partir de l'adresse internet http://cran.univ-lyon1.fr/ ou via une autre adresse trouvée en tapant "R CRAN" dans un moteur de recherche. **R** est un outil très performant avec un domaine d'application très large, mais d'un abord initial peu convivial pour certains. Ce document a pour but de vous permettre de l'utiliser de façon la plus simple possible (sans trop d'investissement de votre part) pour analyser vos données dans les cas les plus classiques, et notamment pour réaliser les tests paramétriques et non paramétriques classiques. Pour ceux qui voudraient en savoir un peu plus et mieux maîtriser le langage R, un manuel introductif est fourni par les auteurs du logiciels à l'adresse http://www.r-project.org/ dans la rubrique " Documentation " à la sous rubrique " Manuals ", et divers manuels sont aussi proposés par d'autres auteurs dont certains en français dans la même rubrique, à la sous rubrique " Contributed ". Le manuel rédigé par Emmanuel Paradis et intitulé "R pour les débutants " est particulièrement utile notamment pour la partie création et personnalisation de graphiques que je n'aborde quasiment pas dans ce document. D'autre part, lorsque l'on utilise cet outil pour des analyses statistiques que l'on souhaite publier dans une revue scientifique, la référence à citer est celle que l'on peut obtenir dans **R** en tapant `citation()` et qui dépendra de la version que vous avez téléchargée :

```
citation()

##
## To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
```

1.2 Rédaction de scripts et interface Rstudio

Il n'est pas conseillé de taper le code **R** directement dans la fenêtre de commande (sauf pour des codes que l'on ne veut pas garder comme les appels à l'aide d'une fonction donnée : `? nomdelafonction`) mais il est recommandé d'écrire le code (ou script) dans un fichier texte que l'on sauvegardera afin d'en garder une trace et de pouvoir lancer d'un coup toutes les lignes de code. Pour cela on peut utiliser le menu de **R** qui permet de créer, sauver et lancer tout ou partie d'un script. Il est recommandé d'insérer dans le script des lignes de commentaires, dont R ne tiendra pas compte. Une ligne de commentaire doit débuter par le signe #.

Lorsque l'on tape directement les commandes dans la fenêtre de commande, la touche "flèche montante" du clavier est bien utile car elle permet de rappeler les lignes de commandes qui ont été tapées précédemment.

L'utilisation de l'application Rstudio permet une gestion agréable de l'ensemble des fenêtres utiles lors de la programmation avec R et la coloration syntaxique du script qui aide à repérer d'éventuelles erreurs de frappe notamment. Je vous la recommande. Elle peut être téléchargée gratuitement à l'adresse suivante : http://www.rstudio.com/products/rstudio/download/. Elle doit être installée une fois que l'application **R** est elle-même installée. Elle est disponible pour MAC, PC et linux, et rend l'utilisation de **R** plus uniforme d'un système d'exploitation à l'autre. Lorsqu'on lance **Rstudio**, on voit apparaître

- en haut à gauche une fenêtre dans laquelle on écrit son script (succession de lignes de code),
- en bas à droite la console de **R** dans laquelle s'affiche les lignes de code du script lancées à l'aide du bouton "Run" et les sorties **R** correspondantes,
- en haut à droite les objets qui sont stockés dans la mémoire de travail de **R**

— et en bas à droite les pages d'aide, les graphes et les fichiers du dossier de travail (qu'on appelle aussi répertoire courant).

Pour que **Rstudio** se lance automatiquement lorsque l'on clique sur un fichier script, il faut pendre soin d'ajouter l'extension ".R" au nom du fihier script lorsqu'on l'enregistre dans **Rstudio** ou dans **R**.

1.3 Spécification du dossier de travail ou répertoire courant

Pour chaque série d'exercices, ou chaque projet, je vous conseille vivement de créer un dossier de travail dans lequel vous mettrez notamment tous vos jeux de données. Au début de chaque session de travail avec **R**, vous devrez alors spécifier le nom de ce dossier de travail ou répertoire courant.

Si vous utilisez **R** sans passer par **Rstudio**, vous irez dans les menus "File" (ou "Fichier" dans la version française) puis "Change dir" (ou "changer le répertoire courant") pour indiquer ce dossier de travail. Vous pouvez aussi taper directement le chemin d'accès à ce dossier en utilisant la fonction `setwd`. Dans **Rstudio**, vous irez dans les menus "Session" puis "Set Working directory".

1.4 Quelques notions de base à propos du langage R

R est un langage " orienté-objet ", ce qui signifie que les variables, fonctions, données, etc..., sont stockés dans la mémoire de l'ordinateur sous forme d'objets auxquels on a accès par leur nom. Un objet peut être créé simplement à partir du signe d'affectation `<-`. La commande suivante crée un objet de nom toto et lui affecte la valeur 12.

```
toto <- 12
```

Pour voir le contenu d'un objet, il suffit de taper son nom.

```
toto

## [1] 12
```

Pour créer un vecteur contenant plusieurs valeurs, il faut indiquer ces valeurs de la façon suivante : `c(val1,val2,...,valn)`. La commande suivante crée par exemple un vecteur de nom titi contenant 4 valeurs.

```
titi <- c(10.3,15.1,20.6,40.5)
```

Si l'on tape le nom de cet objet, on voit apparaître ces 4 valeurs. Le [1] indique le numéro de ligne. Si l'objet était une matrice, R la décrirait ligne par ligne en incrémentant ce numéro de ligne.

```
titi

## [1] 10.3 15.1 20.6 40.5
```

Pour accéder à la ième valeur d'un vecteur `V`, on tape `V[i]`. Par exemple on peut créer un nouvel objet de nom tata et lui affecter la 2ème valeur du vecteur titi en tapant :

```
tata <- titi[2]
```

Et si on tape ensuite `tata`, on voit le contenu de tata :

```
tata

## [1] 15.1
```

Pour créer une matrice, on peut utiliser la fonction `matrix`. La commande suivante crée par exemple une matrice à 2 colonnes de nom tutu contenant les 4 valeurs spécifiées dans le premier argument, la matrice étant remplie par ligne à partir de ces valeurs, l'argument `byrow` étant fixé à `TRUE`.

```
tutu <- matrix(c(10.3,15.1,20.6,40.5),ncol=2,byrow=TRUE)
```

Il suffit de taper le nom de l'objet pour voir s'afficher la matrice créée.

```
tutu

##      [,1] [,2]
## [1,] 10.3 15.1
## [2,] 20.6 40.5
```

On accède à la valeur stockée dans la ligne i et la colonne j de la matrice `M` en tapant `M[i,j]`, comme ci-dessous :

```
tutu[2,1]

## [1] 20.6
```

On peut aussi affecter le résultat d'une fonction à un objet an tapant `nomobjet<-nomfonction(parametres)`. La ligne suivante affecte à l'objet u un vecteur de 5 valeurs tirées au hazard dans une loi uniforme définie entre 10 et 12.

```
u <- runif(5,min=10,max=12)
```

Regardons le résultat :

```
u

## [1] 11.6 11.0 11.0 10.8 10.9
```

Pour calculer des paramètres statistiques sur un vecteur `V`, on peut donc lui appliquer directement les fonctions **R** correspondantes : `mean` pour la moyenne, `var` (resp. `sd`) pour la variance (resp. l'écart type) (par défaut variance et écart type corrigés), `median` pour la médiane, `min` pour le minimum, `max` pour le maximum, `quantile` pour calculer un quantile (voir paragraphe 2.1.2 pour les détails). La commande ci-dessous permet par exemple d'obtenir la moyenne du vecteur `titi`.

```
mean(titi)

## [1] 21.6
```

Enfin, **R** permet de faire de façon automatique du calcul vectoriel, dès lors qu'on lui demande un calcul sur des vecteurs de même taille. Par exemple le code suivant donne automatiquement un vecteur dont chaque composante est calculée par la formule $a_i + b_i^2$:

```
a <- c(1,2,3,4)
b <- c(1,1,10,10)
a + b^2

## [1]   2   3 103 104
```

Attention à un petit détail qui a son importance : dans le langage **R**, les minuscules et les majuscules sont interprétées différemment. Si vous taper à ce stade `mean(Titi)`, **R** vous dira que l'objet `Titi` est introuvable.

1.5 Comment importer des données pour les analyser avec R

Dans **R**, on travaille généralement avec des jeux de données codés sous la forme d'objet de type `data.frame`, c'est-à-dire une liste de vecteurs correspondant aux différentes variables étudiées, chacun étant formé soit d'un ensemble de nombres (vecteur numérique) soit d'un ensemble de mots (vecteur de chaînes de caractères). Par défaut les vecteurs numériques sont considérés comme des variables quantitatives et les vecteurs de chaînes de caractères comme des variables qualitatives ou facteurs. Si une variable qualitative est codée avec des nombres, il faudra appliquer au vecteur correspondant la fonction `as.factor` pour qu'elle soit bien considérée comme un facteur.

Un des moyens les plus commodes pour importer des données dans **R**, consiste à entrer le jeu de données dans Microsoft Excel puis à le sauver au format texte (dans Microsoft Excel aller dans "Fichier", "Enregistrer sous" et **choisir dans "Type de Fichier" le format "Texte(séparateur :tabulation)(*.txt)"** et donner un nom du type "nomfichier.txt" au fichier). Le jeu de données importé doit donc avoir le format classique en statistique : **une colonne par variable, et une ligne par individu, animal ou plus largement unité d'observation, avec en première ligne du fichier de données l'intitulé de chaque vecteur colonne, c'est-à-dire le nom de chaque variable**. Il est important de ne pas mettre de caractères spéciaux dans ces intitulés, ni de blanc, ni d'accents. Lorsqu'il y a des trous dans le fichier de données (données manquantes) il est impératif de les coder par les deux caractères `NA`.

Avant chaque analyse, il est nécessaire de charger les données dans R à partir du fichier "nomfichier.txt" pour les stocker dans un objet de type `data.frame` que l'on nommera comme on le souhaite (`dtartre` dans l'exemple suivant) (mieux vaut éviter les caractères spéciaux et accents dans les noms que l'on donne aux objets). Pour cela, on peut taper et entrer le code suivant si l'on veut par exemple charger les données du fichier `tartre.txt` :

```
dtartre <- read.table("tartre.txt", header=TRUE,dec=".", stringsAsFactors = TRUE)
```

*Si **R** vous dit qu'il ne trouve pas le fichier, il se peut que vous ayez oublié de spécifier le nom du dossier de travail, cf. chapitre 1.3.*

Décrivons les différents arguments utilisés dans la fonction `read.table()` :

— L'argument `header` est fixé à `TRUE` si les intitulés sont présents dans le fichier de données (en première ligne) et à `FALSE` s'ils sont absents.
— L'argument `dec` correspond au séparateur des décimales. Dans la configuration française standard de nos ordinateurs, ce séparateur est une virgule, mais dans la configuration anglo-saxonne standard ce séparateur est un point. Si l'on oublie de spécifier cet argument dans la fonction `read.table` il sera fixé par défaut au point et cela posera problème pour lire des fichiers obtenus à partir d'Excel sauf si l'ordinateur a été configuré avec des options régionales anglo-saxonnes où si les virgules ont été changées en points dans les fichiers issus d'Excel. Dans tous les exemples suivants, pour plus de simpicité, nous ne spécifierons pas cet argument, en supposant que les fichiers de données qui vous sont fournis comportent des points en séparateur de décimale.
— L'argument `stringsAsFactors` est fixé à `TRUE` si l'on veut qu'au moment de l'import des données, toute colonne non numérique (donc comportant des caractères ou chaîne de caractères), soit considérée comme une variable qualitative, de classe `factor`. Ceci est bien pratique et dans les anciennes versions de **R** (précédant la version 4) cet argument était fixé par défaut à `TRUE` et il n'était pas nécessaire de le spécifier pour bénéficier de cette option. C'est dorénavant nécessaire.

Si vous ne disposez pas de Microsoft Excel sur votre ordinateur, tout n'est pas perdu. Vous pouvez utiliser un logiciel équivalent du domaine publique, mais le format de fichier importé sera différent et la fonction d'import aussi. Avec les tableurs des suites Open Office ou Libre Office, il est possible d'exporter un table de données au format CSV ("nomdefichier.csv"). Au moment de l'export, il est alors **TRES IMPORTANT de choisir le séparateur de champs** (je vous conseille d'indiquer l'espace) et d'utiliser ce même même séparateur de champ lors de l'import du fichier dans **R** à l'aide de la fonction `read.csv`

```
nomdelobjetcree <- read.csv("nomdufichier.csv", header = TRUE, sep = " ")
```

1.6 Comment accéder dans R au jeu de données importé

Pour visualiser complètement le jeu de donnée chargé il suffit de taper le nom de l'objet créé par les fonctions `read.table` ou `read.csv`, par exemple ici :

```
dtartre
```

```
##    index traitement
## 1   0.49     temoin
## 2   1.05     temoin
## 3   0.79     temoin
## 4   1.35     temoin
## 5   0.55     temoin
## 6   1.36     temoin
## 7   1.55     temoin
## 8   1.66     temoin
## 9   1.00     temoin
## 10  0.34 supplement
## 11  0.76 supplement
## 12  0.45 supplement
## 13  0.69 supplement
## 14  0.87 supplement
## 15  0.94 supplement
## 16  0.22 supplement
## 17  1.07 supplement
## 18  1.38 supplement
```

Si le fichier de données est de grande taille on peut n'afficher que les premières lignes, on peut utiliser la fonction `head` :

```
head(dtartre)
```

```
##   index traitement
## 1  0.49     temoin
## 2  1.05     temoin
## 3  0.79     temoin
## 4  1.35     temoin
## 5  0.55     temoin
## 6  1.36     temoin
```

Enfin la fonction `str` peut être utilisée, notamment sur les très gros jeux de données (avec de nombreuses variables), pour obtenir une vision globale de la structure du jeu de données (variables numériques, variables qualitatives ou facteurs avec leurs modalités), et la fonction `summary` pour obtenir un résumé statistique de chaque variable.

```
str(dtartre)

## 'data.frame': 18 obs. of  2 variables:
##  $ index     : num  0.49 1.05 0.79 1.35 0.55 1.36 1.55 1.66 1 0.34 ...
##  $ traitement: Factor w/ 2 levels "supplement","temoin": 2 2 2 2 2 2 2 2 2 1 ...

summary(dtartre)

##      index            traitement
##  Min.   :0.220   supplement:9
##  1st Qu.:0.585   temoin    :9
##  Median :0.905
##  Mean   :0.918
##  3rd Qu.:1.280
##  Max.   :1.660
```

Pour accéder séparément à chaque variable du jeu de données, on peut spécifier le nom du jeu de données suivi du signe `$` puis du nom de la colonne correspondante. Par exemple pour accéder à la variable "index" on tape :

```
dtartre$index

##  [1] 0.49 1.05 0.79 1.35 0.55 1.36 1.55 1.66 1.00 0.34 0.76 0.45 0.69 0.87 0.94
## [16] 0.22 1.07 1.38
```

Il est aussi possible d'accéder aux différentes variables du jeu de données en les appelant directement par leur nom de colonne en "attachant" au préalable le jeu de données à l'aide de la fonction `attach` :

```
attach(dtartre)
```

Il n'est alors plus la peine de préciser le nom du jeu de données pour accéder à chaque variable. On accède par exemple à la variable "index" simplement en tapant :

```
index

##  [1] 0.49 1.05 0.79 1.35 0.55 1.36 1.55 1.66 1.00 0.34 0.76 0.45 0.69 0.87 0.94
## [16] 0.22 1.07 1.38
```

Néanmoins cette fonction est à utiliser avec prudence notamment lorsque l'on analyse plusieurs jeux de données durant la même session **R**, pour éviter les conflits entre deux jeux de données attachés ayant des colonnes de même nom.

1.7 Manipulation et transformation de données

1.7.1 Sélection de lignes dans un jeu de données

Si l'on veut créer un sous-jeu de données ne comprenant qu'une partie du jeu de données initial, on peut utiliser la fonction `subset` en indiquant en premier argument le jeu de données complet, et en deuxième argument la variable logique utilisée pour sélectionner les lignes que l'on veut garder : ainsi l'exemple suivant permet de créer un nouveau jeu de données qui garde uniquement les lignes correspondant au groupe "supplement".

```
dtartre.suppl<-subset(dtartre,traitement=="supplement")
dtartre.suppl

##    index traitement
## 10  0.34 supplement
## 11  0.76 supplement
## 12  0.45 supplement
## 13  0.69 supplement
## 14  0.87 supplement
## 15  0.94 supplement
## 16  0.22 supplement
## 17  1.07 supplement
## 18  1.38 supplement
```

Les principaux opérateurs logiques dans **R** sont `==` pour "égal à", `!=` pour "différent de", `|` pour le "ou" logique et `&` pour le "et" logique.

1.7.2 Création d'une variable qualitative à partir d'une variable quantitative

Rappelons tout d'abord qu'une variable qualitative qui aurait été codée numériquement lors de la saisie des données doit être transformée en facteur, à l'aide de la fonction `factor`, si l'on veut que **R** la considère bien comme qualitative.

```
varqualmalcodee <- c(1,0,0,1,1,1,0)
str(varqualmalcodee)

##  num [1:7] 1 0 0 1 1 1 0

varqual <- factor(varqualmalcodee)
str(varqual)

##  Factor w/ 2 levels "0","1": 2 1 1 2 2 2 1
```

On peut aussi facilement créer des vecteurs logiques (à deux modalités : `TRUE` ou `FALSE`) en utilisant des opérateurs logiques vus au paragraphe précédent et le calcul vectoriel automatique de **R**. Par exemple la ligne de code suivante crée un vecteur logique dont la valeur sera égale à `TRUE` uniquement si l'index de tartre se situe entre 0.5 et 0.8. Ce genre de manipulation peut être utile par exemple lorsqu'on l'on veut définir une variable qualitative (dépassement ou non d'un seuil par exemple) à partir d'une variable quantitative.

```
veclogique <- (dtartre$index > 0.5) & (dtartre$index < 0.8)
dtartre$index

##  [1] 0.49 1.05 0.79 1.35 0.55 1.36 1.55 1.66 1.00 0.34 0.76 0.45 0.69 0.87 0.94
## [16] 0.22 1.07 1.38

veclogique

##  [1] FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
## [13]  TRUE FALSE FALSE FALSE FALSE FALSE
```

Plus généralement, on peut utiliser la fonction `cut` qui permet de coder l'appartenance d'une variable quantitative avec des intervalles définis par son argument `breaks`. Cette fonction renvoie un facteur (variable qualitative).

```
(classe.index <- cut(dtartre$index, breaks = c(0,0.5,0.8,2)))

##  [1] (0,0.5]   (0.8,2]   (0.5,0.8] (0.8,2]   (0.5,0.8] (0.8,2]   (0.8,2]
##  [8] (0.8,2]   (0.8,2]   (0,0.5]   (0.5,0.8] (0,0.5]   (0.5,0.8] (0.8,2]
## [15] (0.8,2]   (0,0.5]   (0.8,2]   (0.8,2]
## Levels: (0,0.5] (0.5,0.8] (0.8,2]
```

1.7.3 Changement de l'ordre ou des noms des modalités d'un facteur

Par défaut **R** classe les modalités d'un facteur par ordre alphabétique. Si l'on veut changer cet ordre, on peut utiliser une ligne de code de ce type :

```
dtartre$traitement

##  [1] temoin     temoin     temoin     temoin     temoin     temoin
##  [7] temoin     temoin     temoin     supplement supplement supplement
## [13] supplement supplement supplement supplement supplement supplement
## Levels: supplement temoin

dtartre$traitement <- factor(dtartre$traitement,levels = c("temoin", "supplement"))
dtartre$traitement

##  [1] temoin     temoin     temoin     temoin     temoin     temoin
##  [7] temoin     temoin     temoin     supplement supplement supplement
## [13] supplement supplement supplement supplement supplement supplement
## Levels: temoin supplement
```

On peut aussi changer les noms des modalités de la façon suivante :

```r
levels(dtartre$traitement) <- c("contrôle", "supplément")
dtartre$traitement
```

```
##  [1] contrôle   contrôle   contrôle   contrôle   contrôle   contrôle
##  [7] contrôle   contrôle   contrôle   supplément supplément supplément
## [13] supplément supplément supplément supplément supplément supplément
## Levels: contrôle supplément
```

1.7.4 Transformation d'une variable (ex. : log, racine carrée)

Si l'on veut travailler sur la transformation d'une variable, par exemple sur le logarithme de cette variable, il suffira d'appliquer la fonction `log` à cette variable. Par exemple la ligne de code ci-dessous permet d'afficher le logarithme népérien de la variable "index" :

```r
log(dtartre$index)
```

```
##  [1] -0.7133  0.0488 -0.2357  0.3001 -0.5978  0.3075  0.4383  0.5068  0.0000
## [10] -1.0788 -0.2744 -0.7985 -0.3711 -0.1393 -0.0619 -1.5141  0.0677  0.3221
```

Attention, dans **R** le logarithme népérien est donné par la fonction `log` et le logarithme décimal par la fonction `log10`. Il est utile aussi de connaître la fonction qui calcule la racine carrée : `sqrt`.

1.7.5 Création d'une nouvelle variable au sein du jeu de données

Pour créer une nouvelle variable intégrée au jeu de données il suffit de définir cette nouvelle variable en mettant devant son nom l'affectation au jeu de données sous la forme classique comme ci-dessous pour intégrer l'index en log directement dans le jeu de données `dtartre` :

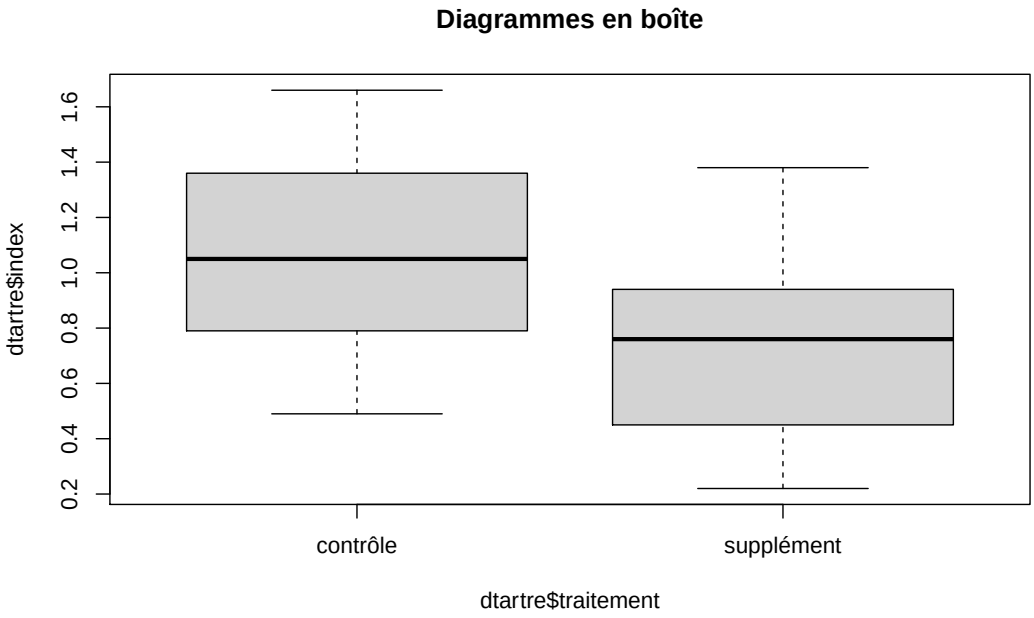
```r
dtartre$indexenlog <- log(dtartre$index)
# on peut voir le résultat en regardant à nouveau la structure de dtartre
str(dtartre)
```

```
## 'data.frame': 18 obs. of  3 variables:
##  $ index     : num  0.49 1.05 0.79 1.35 0.55 1.36 1.55 1.66 1 0.34 ...
##  $ traitement: Factor w/ 2 levels "contrôle","supplément": 1 1 1 1 1 1 1 1 1 2 ...
##  $ indexenlog: num  -0.7133 0.0488 -0.2357 0.3001 -0.5978 ...
```

1.8 Gestion des graphes

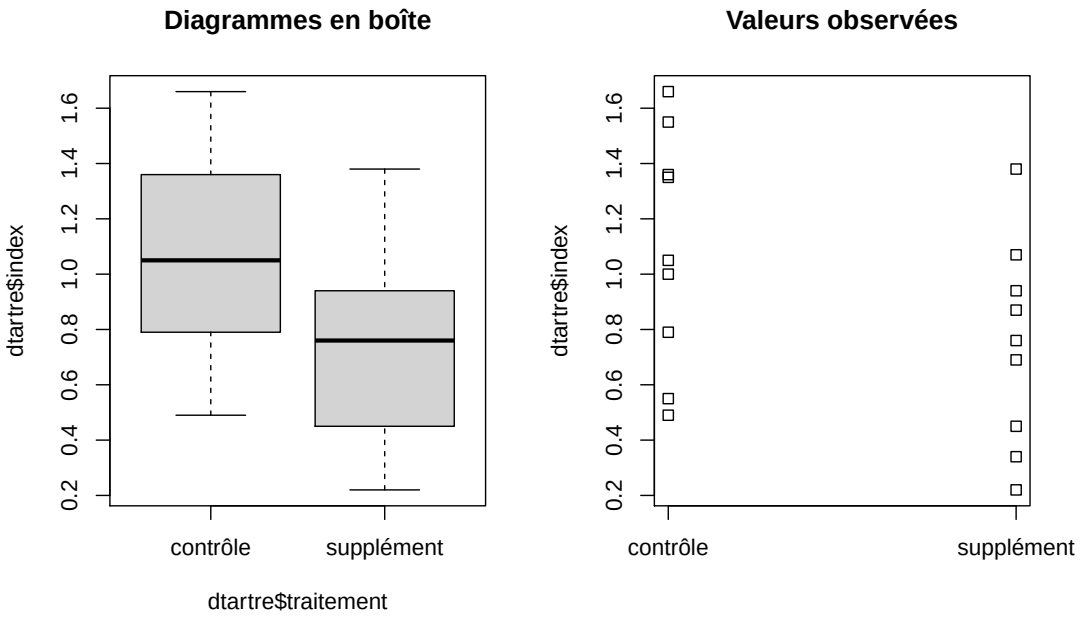
Pour ouvrir une nouvelle fenêtre graphique on tape généralement la commande `x11()`. Sous Windows on peut aussi utiliser la commande `windows()` et sous Mac OS il faut en principe utiliser la commande `quartz()`. Lorsque l'on travaille avec **Rstudio**, on n'a généralement pas besoin dl'ouvrir des fenêtres graphiques : celles-ci sont ouvertes à chaque fois que cela est nécessaire, stockées et accessibles avec les flèches horizontales au-dessus du bloc de gestion des graphes (en bas à droite). Essayez :

```r
boxplot(dtartre$index ~ dtartre$traitement, main = "Diagrammes en boîte")
```



Une fenêtre graphique peut éventuellement être partitionnée pour y créer plusieurs graphes. La commande `par(mfrow = c(n1,n2))` permet par exemple de partitionner la fenêtre en n1*n2 graphes, avec un découpage en n1 graphes sur la verticale et en n2 graphes sur l'horizontale. Voici un petit exemple de création d'une figure avec deux types de représentation des données du jeu de données `dtartre`, à gauche sous forme de deux diagrammes en boîte (un pour chaque groupe) et à droite en représentant directement toutes les valeurs observées pour les deux groupes.

```
par(mfrow = c(1,2))
boxplot(dtartre$index ~ dtartre$traitement,main = "Diagrammes en boîte")
stripchart(dtartre$index ~ dtartre$traitement, vertical = TRUE,
    main = "Valeurs observées")
```




```
par(mfrow = c(1,1)) # pour revenir ensuite sur une fenêtre classique non partagée
```

Une fois le ou les graphes réalisé(s), la fenêtre graphique peut être sauvée sous différents formats. La sauvegarde sous format " EPS " est de bonne qualité et souvent recommandée pour la publication d'articles. Pour sauver la fenêtre dans ce format et stocker le fichier correspondant sous le nom " nomfic.eps ", il suffit de taper la commande `dev.copy2eps(file="nomfic.eps")` suivie de `dev.off()`. On peut aussi sauver la fenêtre en format " JPEG " en tapant `dev.copy(device=jpeg,file="nomfic.jpg")` suivie de `dev.off()`. D'autres formats de sauvegarde sont disponibles. On peut consulter l'aide de la fonction `dev.copy()` pour les connaître.

Si l'on travaille avec **Rstudio**, on peut utiliser le menu "Export" du bloc de gestion des graphes pour exporter chaque fenêtre graphique souhaitée au format voulu.

1.9 Et si l'on ne peut pas installer R sur son ordinateur ?

Pour ceux qui ne voudraient pas installer **R** sur leur ordinateur, il y a moyen de l'utiliser à distance. C'est valable uniquement si on ne veut réaliser qu'un petit nombre d'analyses. On peut en effet exécuter du code **R** à distance (via une interface de type **RWeb**), par exemple à partir de l'adresse http://pbil.univ-lyon1.fr/Rweb/. La démarche à suivre est ensuite expliquée sur les sites. Il est nécessaire d'indiquer en fin de page du site (à " Select a local file to submit "), l'emplacement du fichier comportant les données, et formaté comme indiqué dans le paragraphe précédent. Il faut ensuite taper toutes les lignes de code à exécuter dans la première fenêtre puis les soumettre en cliquant sur le bouton " Submit ". Après quelques secondes ou minutes, une fenêtre apparaît avec tous les résultats. En ce qui concerne le chargement des données, il n'est pas nécessaire d'utiliser la fonction `read.table` ni la fonction `attach`, car cela est fait automatiquement, et les données sont stockées dans l'objet `X`. Plus précisément, le code réalisé automatiquement par défaut avant celui saisi par l'utilisateur est : `X <- read.table(" adresse du fichier indiqué par l'utilisateur ", header = TRUE, stringsAsFactors = TRUE)` suivi de `attach(X)`.

1.10 Présentation du document

Dans la suite de ce document, vous trouverez quelques exemples d'utilisation des tests classiques. Ils sont ordonnées en fonction du type de jeu de données à analyser. Comme vous avez dû vous en rendre compte en testant les exemples précédents, le signe > en début de ligne est le curseur qui apparaît automatiquement dans la fenêtre de R et qui n'est donc pas à saisir au clavier. Tout ce qui figure après le signe # constitue des commentaires explicatifs, non interprétés par **R**, et donc n'est pas à taper non plus. Le code peut avoir l'air un peu compliqué au début, mais vous verrez que pour faire des analyses répétitives, c'est beaucoup plus performant que d'utiliser des menus avec boutons. Dans ces exemples, une seule façon de procéder pour effectuer un test donné est décrite alors que plusieurs sont souvent possibles, notamment en ce qui concerne la présentation des données en entrée. Si vous voulez en savoir plus sur une fonction, utilisez l'aide en ligne : elle est disponible pour chaque fonction de **R**. Elle est accessible en tapant `help(nomdelafonction)` ou de façon équivalente `?nomdelafonction`.

ATTENTION, dans ces exemples, pour chaque jeu de données, divers tests sont souvent présentés (paramétrique et non paramétrique notamment) sans que le choix du bon test (le plus puissant parmi ceux dont les conditions d'utilisation sont respectées par les données) ne soit effectué. Ce choix reste à faire mais ce n'est pas le propos de ce guide qui a uniquement pour but de vous donner accès au code à saisir pour effectuer chacun des tests qui peuvent être utiles. CE MANUEL PRESENTE DONC DES ANALYSES ADAPTEES AU JEUX DE DONNEES ET D'AUTRES NON ADAPTEES !

2 Avec une série d'observations

2.1 Variable quantitative

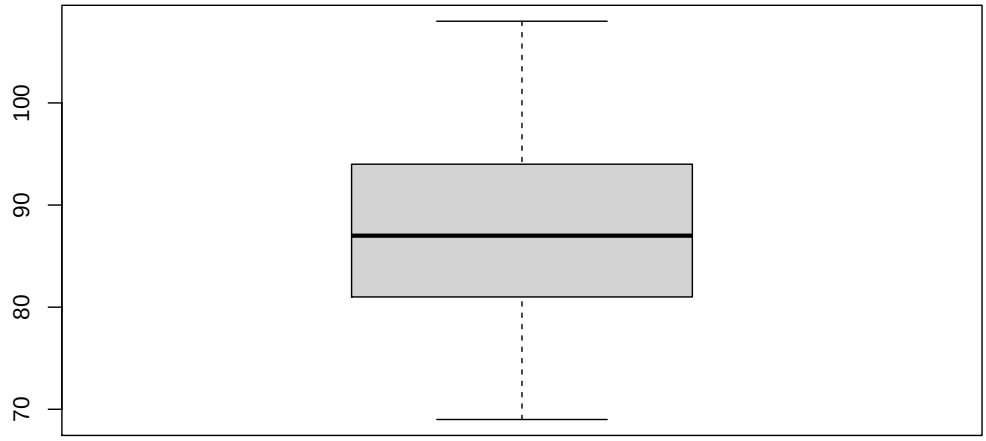
Il convient tout d'abord d'importer les données. Il s'agit ici d'une seule variable observée nommée `temps`.

```
d <- read.table("reflexe.txt", header=TRUE, stringsAsFactors = TRUE)
d

##    temps
## 1    108
## 2    100
## 3     78
## 4     81
## 5     69
## 6     87
## 7     88
## 8     81
## 9     87
## 10    94
```

2.1.1 Examen de la distribution

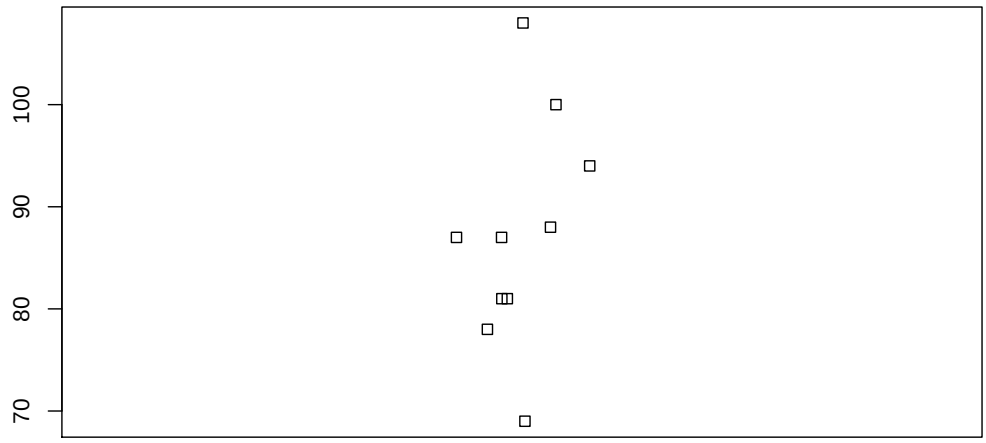
On représente classiquement ce type de données par un diagramme en boîte :

```
boxplot(d$temps)
```



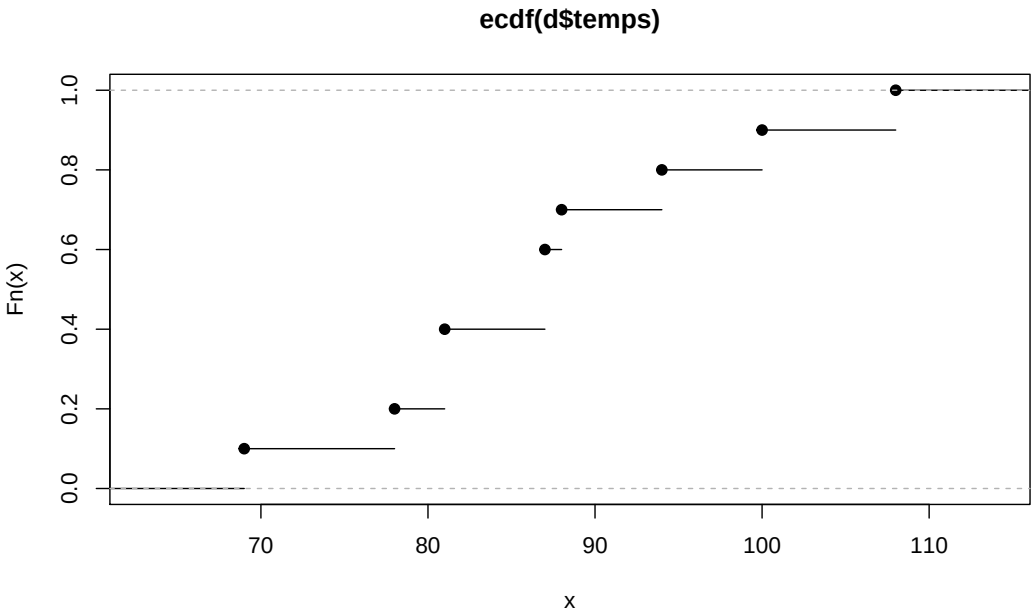
Lorsque l'effectif est très petit il est préférable de représenter directement tous les points observés à l'aide de la fonction `stripchart`. L'argument `method` de cette fonction peut être fixé à `"jitter"` ou `"stack"` pour visualiser les ex-aequeos.

```
stripchart(d$temps, vertical = TRUE, method = "jitter")
```



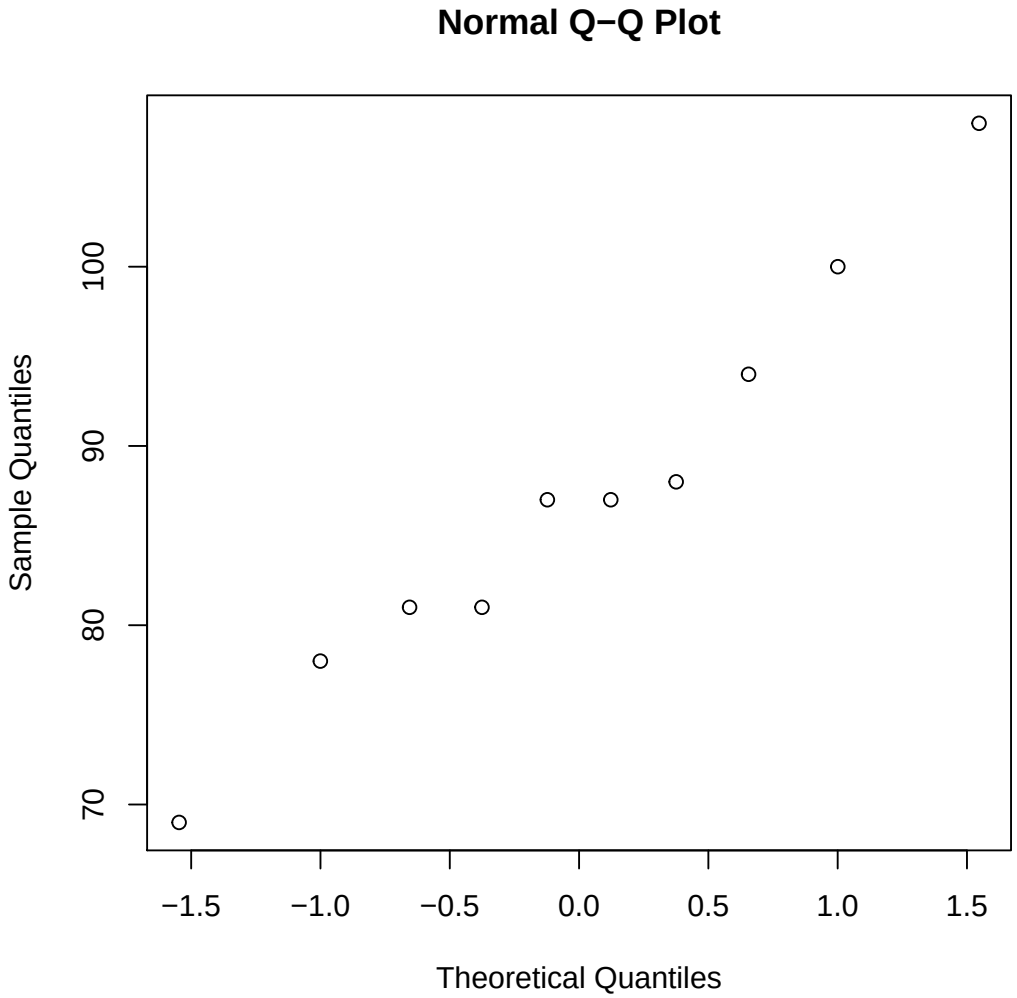
Lorsque l'effectif est très grand il est possible de représenter la distribution par un histogramme à l'aide de la fonction `hist`. Ici l'effectif est trop petit mais on peut par contre obtenir facilement la courbe de fréquence cumulée à l'aide de la commande suivante :

```
plot(ecdf(d$temps))
```



Pour juger de la normalité de la distribution, il est souvent utile de représenter les quantiles observés en fonction des quantiles de la loi normale :

```
qqnorm(d$temps)
```



Un test de normalité peut être utilisé en complément, **tout en gardant à l'esprit qu'un tel test manque de puissance pour les petits échantillons et qu'en aucun cas ce test ne permet pas de montrer la normalité d'une distribution**.

```r
shapiro.test(d$temps)

##
##  Shapiro-Wilk normality test
##
## data:  d$temps
## W = 1, p-value = 0.9
```

2.1.2 Description de la distribution (moyenne, écart type, quantiles)

On peut bien entendu facilement calculer les paramètres décrivant la distribution observée :
— la moyenne à l'aide de la fonction `mean`,

```r
mean(d$temps)
## [1] 87.3
```

— la variance estimée à l'aide de la fonction `var` et l'écart type estimé à l'aide de la fonction `sd`,

```r
var(d$temps)
## [1] 126
sd(d$temps)
## [1] 11.2
```

— la médiane à l'aide de la fonction `median` et plus généralement n'importe quel quantile à l'aide de la fonction `quantile`, en indiquant en argument les probabilités auxquelles on veut calculer ces quantiles dans le vecteur `probs`.

```r
median(d$temps)
## [1] 87
quantile(d$temps, probs = c(1/4,1/2,3/4))
##  25%  50%  75%
## 81.0 87.0 92.5
```

2.1.3 Test de conformité : comparaison de la moyenne observée à une moyenne théorique

Le test T de conformité à une moyenne théorique est réalisé de la façon suivante, en affectant l'argument `mu` de la fonction `t.test` à la moyenne théorique :

```r
t.test(d$temps, mu = 83.2)

##
##  One Sample t-test
##
## data:  d$temps
## t = 1, df = 9, p-value = 0.3
## alternative hypothesis: true mean is not equal to 83.2
## 95 percent confidence interval:
##  79.3 95.3
## sample estimates:
## mean of x
##      87.3
```

2.1.4 Intervalle de confiance autour d'une moyenne

Pour calculer un intervalle de confiance autour d'une moyenne observée sans réaliser de test, on utilise la même fonction que pour réaliser le test de conformité de Student, sans spécifier l'argument `mu` et on récupère uniquement la sortie de la fonction qui correspond à l'intervalle de confiance. On peut modifier le seuil de confiance à l'aide de l'argument `conf.level` par défaut fixé à 0.95.

```r
t.test(d$temps, conf.level = 0.95)$conf.int

## [1] 79.3 95.3
## attr(,"conf.level")
## [1] 0.95
```

Cette fonction permet occasionnellement de calculer des intervalles de confiance unilatéraux. L'argument `alternative` doit être fixé à

— `"less"` pour obtenir un intervalle de confiance du type $[-\infty,a]$
— `"greater"` pour obtenir un intervalle de confiance du type $[a,+\infty]$
— `"two.sided"` pour obtenir un intervalle de confiance du type [a,b] (option par défaut)

2.2 Variable qualitative

2.2.1 Calcul des effectifs observés dans chaque classe et examen de la distribution

Lorsque les données sont présentées sous forme brute, il est nécessaire au préalable d'utiliser la fonction `table` sur la variable qualitative afin d'obtenir de façon automatique les effectifs observés comme ci-dessous (exemple reportant le type de lien entre les animaux et leur maître)) :

```
d <- read.table("animfamibrut.txt", header=TRUE, stringsAsFactors = TRUE)
str(d) # pour voir ce qu'il y a dans le jeu de données

## 'data.frame': 1100 obs. of  2 variables:
##  $ espece: Factor w/ 2 levels "chat","chien": 2 2 2 2 2 2 2 2 2 2 ...
##  $ lien  : Factor w/ 4 levels "ami","enfant",..: 2 2 2 2 2 2 2 2 2 2 ...

t <- table(d$lien)
t

##
##     ami  enfant famille  invite
##     204     208     638      50
```

Si l'on travaille directement à partir des effectifs, par exemple tels que récupérés dans un article, on peut créer cette table manuellement :

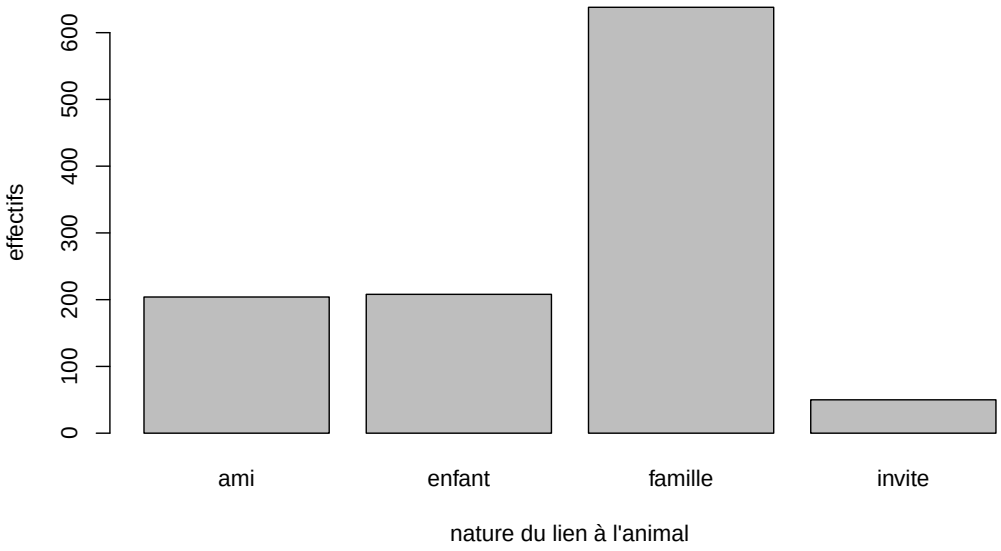
```
t <- as.table(c(204, 208, 638, 50))
row.names(t) <- c("ami", "enfant", "famille", "invite")
t

##     ami  enfant famille  invite
##     204     208     638      50
```

On peut si besoin calculer les fréquences observées dans chaque classe à partir de la table des effectifs :

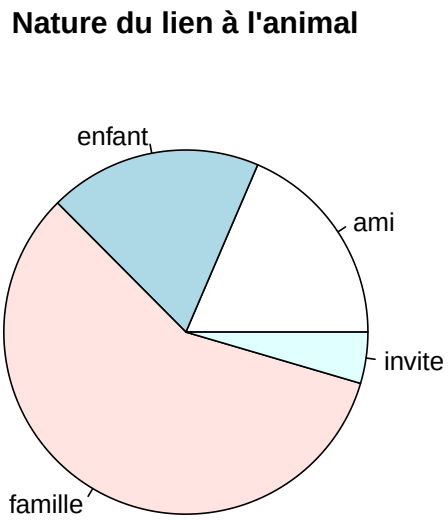
```
tf <- prop.table(t)
tf

##     ami  enfant famille  invite
##  0.1855  0.1891  0.5800  0.0455
```

On peut représenter les tables `t` ou `tf` en diagramme en bâton ou en diagramme en secteurs (moins recommandé).

```
barplot(t, xlab = "nature du lien à l'animal", ylab = "effectifs")
```




```
pie(t, main = "Nature du lien à l'animal")
```



2.2.2 Test du χ^2 d'ajustement

Pour réaliser le test du χ^2 d'ajustement, on doit coder les effectifs observés dans un vecteur ou une table que l'on nommera `nobs` comme ci-dessous (cf. section 2.2.1 pour les explications relatives à la création de tables) et les proportions théoriques dans chaque classe dans un vecteur que l'on nommera `ptheo` dans cet exemple) :

```
## Définition des effectifs observés selon 3 façons possibles
# 1/ nobs défini comme une table créée à partir des données brutes
d1 <- read.table("insemination.txt", header=TRUE, stringsAsFactors = TRUE)
nobs <- table(d1$resultat)
# 2/ ou nobs défini comme une table créée manuellement
nobs <- as.table(c(78, 275))
row.names(nobs) <- c("echec", "succes")
# 3/ ou nobs défini comme un simple vecteur
```

```r
nobs <- c(78, 275)

## Définition des proportions théoriques
ptheo <- c(0.25, 0.75)
```

Le test du χ^2 d'ajustement est ensuite réalisé en mettant comme premier argument de la fonction `chisq.test` les effectifs observés, et en affectant l'argument `p` aux proportions théoriques :

```r
khi2<-chisq.test(nobs, p = ptheo)
khi2

##
##  Chi-squared test for given probabilities
##
## data:  nobs
## X-squared = 2, df = 1, p-value = 0.2
```

Les effectifs théoriques calculés automatiquement à partir des proportions théoriques peuvent être affichés afin de vérifier les conditions d'utilisation du test du χ^2 (effectifs théoriques tous supérieurs à 5) :

```r
print(khi2$expected)

## [1]  88.2 264.8
```

2.2.3 Test exact (utilisant la loi binomiale) de comparaison d'une fréquence observée à une fréquence théorique

La fonction `binom.test` permet de réaliser ce test. Elle doit être paramétrée avec
- comme 1er argument le nombre de réalisation de l'évènement étudié,
- comme 2ème argument le nombre total de tirages,
- comme 3ème argument `p`, la fréquence théorique à laquelle on veut comparer l'observée,
- et comme 4ème argument le type d'hypothèse alternative testée (par défaut fixé à `"two.sided"`, c'est-à-dire bilatéral).

Voici un exemple sur lequel on peut utiliser cette fonction : un traitement préventif d'une maladie a été testée sur une population animale dont on sait que sans traitement environ 4% des animaux sont touchés par cette maladie. Sur 100 animaux traités, 2 sont atteints par la maladie. Ce pourcentage observé est-il significativement différent du 4% observé habituellement sans traitement ? Pour résoudre cet exemple on utilisera le code suivant :

```r
binom.test(2, 100, p = 0.04, alternative = "two.sided",conf.level = 0.95)

##
##  Exact binomial test
##
## data:  2 and 100
## number of successes = 2, number of trials = 100, p-value = 0.4
## alternative hypothesis: true probability of success is not equal to 0.04
## 95 percent confidence interval:
##  0.00243 0.07038
## sample estimates:
## probability of success
##                   0.02
```

2.2.4 Intervalle de confiance autour d'une fréquence

Pour calculer un intervalle de confiance autour d'une fréquence, on utilise la même fonction `binom.test` que précédemment (cf. section 2.2.3 pour la définition des arguments), sans spécifier l'argument `p` si on n'a pas de valeur théorique à laquelle on veut comparer la fréquence observée. On récupère alors uniquement la sortie de la fonction qui correspond à l'intervalle de confiance (sortie `conf.int`). On peut modifier le seuil de confiance à l'aide de l'argument `conf.level` par défaut fixé à 0.95.

Cette fonction permet si nécessaire de calculer des intervalles de confiance unilatéraux. L'argument `alternative` doit être fixé à
- `"less"` pour obtenir un intervalle de confiance du type [0,a]
- `"greater"` pour obtenir un intervalle de confiance du type [a,1]

— `"two.sided"` pour obtenir un intervalle de confiance du type [a,b] (option par défaut)

Voici un exemple de calcul d'intervalle de confiance du type [0, a] : en comparant deux méthodes diagnostiques on observe 3 discordances sur 100. On veut calculer le seuil au dessous duquel on est sûr à 95% que se trouve la proportion théorique de discordances entre les deux méthodes (intervalle de confiance unilatéral [0; seuil]). Voici comment on répond à cette question avec **R** :

```
binom.test(3, 100, alternative = "less", conf.level = 0.95)$conf.int

## [1] 0.0000 0.0757
## attr(,"conf.level")
## [1] 0.95
```

Si l'on souhaite simplement obtenir l'intervalle de confiance bilatéral sur cette proportionde discordances on tapera :

```
binom.test(3, 100, alternative = "two.sided",conf.level = 0.95)$conf.int

## [1] 0.00623 0.08518
## attr(,"conf.level")
## [1] 0.95
```

Voici un exemple de calcul d'intervalle de confiance du type [a, 1] : On évalue la sensibilité d'un test diagnostique : sur 100 animaux malades, le test en détecte 96. On veut calculer le seuil au dessus duquel on est sûr à 95% que se trouve la sensibilité théorique du test diagnostique (intervalle de confiance unilatéral [seuil; 1]). Voici comment on répond à cette question avec **R** :

```
binom.test(96, 100, alternative = "greater",conf.level = 0.95)$conf.int

## [1] 0.911 1.000
## attr(,"conf.level")
## [1] 0.95
```

3 Avec deux séries indépendantes d'observations

3.1 Variable quantitative

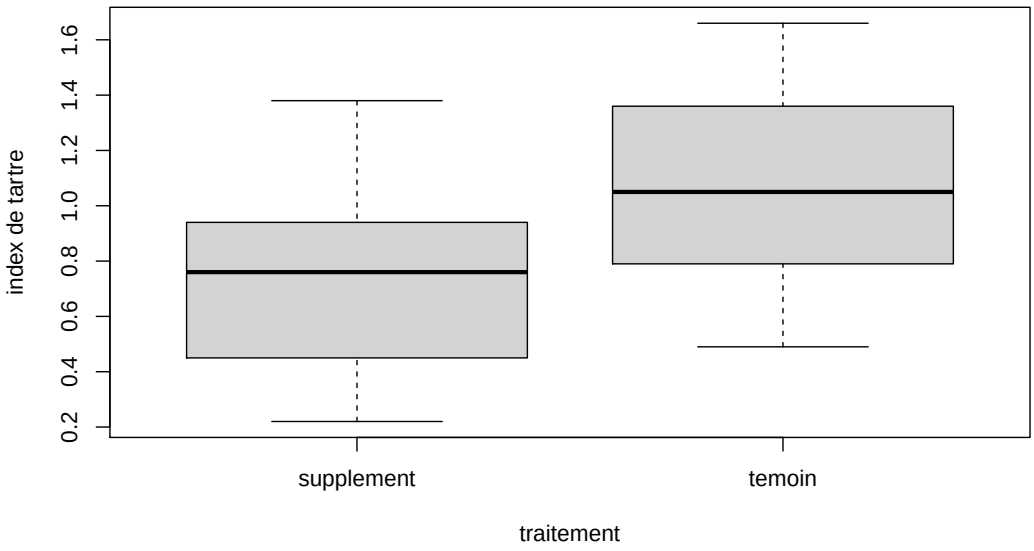
Les données peuvent être importées sous la forme d'un tableau contenant deux colonnes, une colonne avec les valeurs observées de la variable quantitative (nommée index dans cet exemple) et une colonne avec le facteur codant pour le groupe d'appartenance de chaque observation (nommé traitement dans cet exemple). L'ordre de présentation des colonnes et des lignes n'a pas d'importance

```
d2 <- read.table("tartre.txt", header = TRUE, stringsAsFactors = TRUE)
str(d2)

## 'data.frame': 18 obs. of  2 variables:
##  $ index     : num  0.49 1.05 0.79 1.35 0.55 1.36 1.55 1.66 1 0.34 ...
##  $ traitement: Factor w/ 2 levels "supplement","temoin": 2 2 2 2 2 2 2 2 2 1 ...
```

3.1.1 Examen des distributions

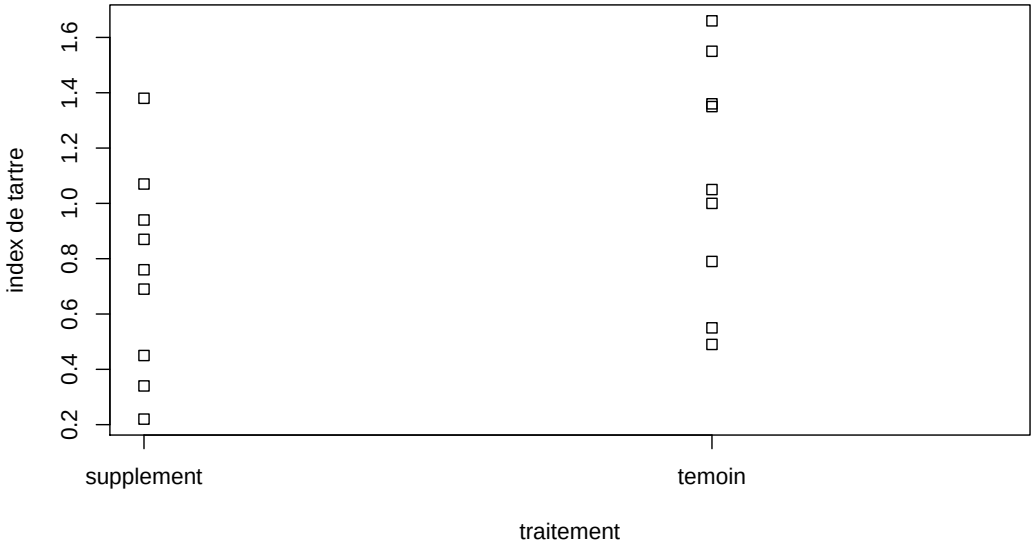
Les distributions sont le plus souvent représentées sous forme de deux diagrammes en boîte.

```
plot(d2$index ~ d2$traitement, xlab = "traitement", ylab = "index de tartre")
```



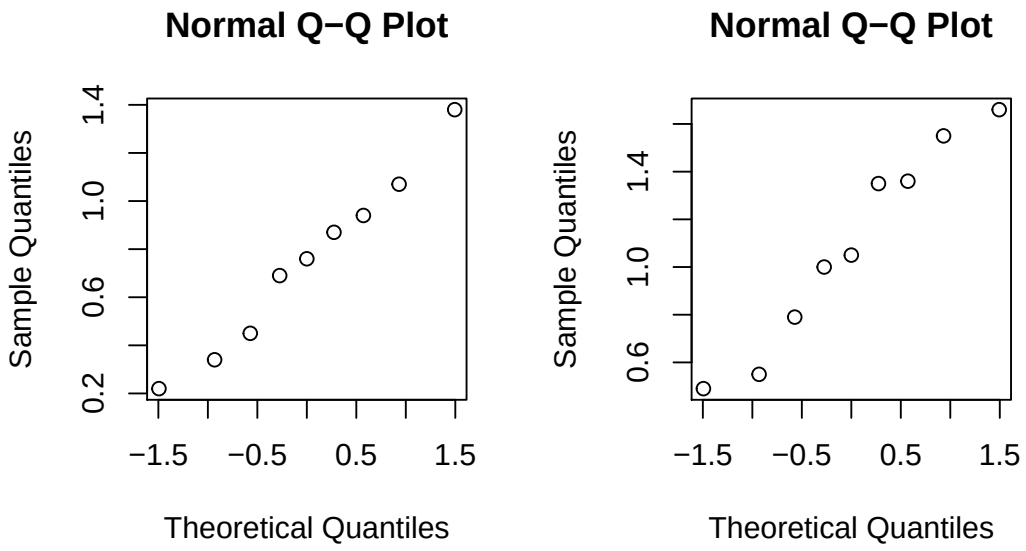
Lorsque les effectifs sont très petits il est préférable de représenter directement tous les points observés dans chaque groupe à l'aide de la fonction **stripchart**, éventuellement avec l'une des options `method = "jitter"` ou `method = "stack"` qui permettent de distinguer les points ex-aequos.

```
stripchart(d2$index ~ d2$traitement, vertical = TRUE, method = "stack",
    xlab = "traitement", ylab = "index de tartre")
```



Il est possible de tracer les graphes des quantiles associés aux quantiles de la loi normale pour chacun des deux groupes en utilisant la fonction **tapply** qui permet d'appliquer une fonction (ici **qqnorm**) à chaque sous-partie de la colonne spécifiée en premier argument (ici **index**), définie par la variable indiquant le groupe d'appartenance spécifiée en deuxième argument (ici **traitement**). <si vous utilisez comme ci-dessous le partage de fenêtre pour faire apparaître les deux figures sur la même fenêtre, pensez bien à revenir à la fenêtre classique ensuite avec la dernière commande. Ce partage n'est pas utile avec Rstudio vu qu'on peut naviguer entre les fenêtres graphiques successives créées.

```
par(mfrow = c(1, 2))
tapply(d2$index, d2$traitement, qqnorm)
```




```
par(mfrow = c(1, 1))
```

3.1.2 Tests de comparaison de 2 moyennes (ou tendances centrales)

Suivant la forme et la dispersion des distributions et la taille des effectifs, on pourra réaliser un des trois tests suivants de comparaison de deux moyennes ou tendances centrales :

— soit le **test T avec variances égales (Student)**

```
t.test(d2$index ~ d2$traitement, paired = FALSE, var.equal = TRUE)
##
##  Two Sample t-test
##
## data:  d2$index by d2$traitement
## t = -2, df = 16, p-value = 0.09
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.7389  0.0544
## sample estimates:
## mean in group supplement     mean in group temoin
##                    0.747                    1.089
```

— soit le **test T avec variances inégales (test de Welch dit aussi Student avec variances inégales)**

```
t.test(d2$index ~ d2$traitement, paired = FALSE, var.equal = FALSE)
##
##  Welch Two Sample t-test
##
## data:  d2$index by d2$traitement
## t = -2, df = 16, p-value = 0.09
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.739  0.055
## sample estimates:
## mean in group supplement     mean in group temoin
##                    0.747                    1.089
```

— soit le **test non paramétrique de la somme des rangs (Mann-Whitney-Wilcoxon)**

```
wilcox.test(d2$index ~ d2$traitement, paired = FALSE)
##
##  Wilcoxon rank sum exact test
##
## data:  d2$index by d2$traitement
## W = 22, p-value = 0.1
## alternative hypothesis: true location shift is not equal to 0
```

```
# comme on ne peut plus parler strictement de comparaison de moyennes
# dans ce cas il peut être intéressant de calculer les médianes par groupe
tapply(d2$index, d2$traitement, median)
## supplement     temoin
##       0.76       1.05
```

3.1.3 Intervalle de confiance autour de la différence entre deux moyennes

Lorsqu'un test paramétrique de comparaison de moyennes est réalisé avec la fonction `t.test` (cf. paragraphe précédent), il est accompagné du calcul de l'intervalle de confiance autour de la différence entre les deux moyennes comparées (sortie `conf.int`) affiché en dernier dans la sortie de la fonction.

Dans certains cas (par ex. la réalisation d'un test d'équivalence), on s'intéressera directement et uniquement à ce calcul d'intervalle de confiance, que l'on pourra d'ailleurs extraire du résultat de la fonction `t.test` de la façon suivante dans l'exemple d'une statistique de Student supposant les variances égales :

```
t.test(d2$index ~ d2$traitement, paired = FALSE, var.equal = TRUE)$conf.int

## [1] -0.7389  0.0544
## attr(,"conf.level")
## [1] 0.95
```

3.1.4 Tests de comparaison de 2 variances

Si les distributions peuvent être supposées normales, et que les variances semblent différentes, un test de comparaison de variances peut être réalisé de la façon suivante :

```
var.test(d2$index ~ d2$traitement)

##
##  F test to compare two variances
##
## data:  d2$index by d2$traitement
## F = 0.8, num df = 8, denom df = 8, p-value = 0.7
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.173 3.391
## sample estimates:
## ratio of variances
##              0.765
```

Rappelons que ce test ne présente pas beaucoup d'intérêt pour juger s'il est raisonnable de supposer les variances égales dans un test de comparaison de moyennes, notamment du fait de son manque de puissance pour de faibles effectifs. Dans ce but on se basera plus souvent sur une représenttion graphique des distributions. **Ce test est par contre utile pour démontrer l'existence d'un différence entre variances** lorsque celle-ci est d'intérêt biologique.

3.2 Variable qualitative

3.2.1 Résumé des données sous la forme d'une table de contingence

— **Lorsque l'on dispose des données brutes** (jeu de données comprenant les deux colonnes correspondant aux deux variables qualitatives observées) il est facile de créer la table de contingence associée en utilisant la fonction `table`. Voici un exemple sur un jeu de données correspondant à l'étude de la nature du lien entre les animaux et leur maître :

```
d4 <- read.table("animfamibrut.txt", header = TRUE, stringsAsFactors = TRUE)
str(d4)
## 'data.frame': 1100 obs. of  2 variables:
##  $ espece: Factor w/ 2 levels "chat","chien": 2 2 2 2 2 2 2 2 2 2 ...
##  $ lien  : Factor w/ 4 levels "ami","enfant",..: 2 2 2 2 2 2 2 2 2 2 ...
# Etape préliminaire de construction de la table de contingence
tlien <- table(d4$espece, d4$lien)
tlien
```

```
##
##         ami enfant famille invite
##   chat  128     86     342     44
##   chien  76    122     296      6
```

Il est parfois opportun dans un tel cas de réordonner les modalités des deux facteurs (variables qualitatives), dans l'ordre qui nous convient (ici en suivant un ordre décroissant pour l'attachement à l'animal) comme ci-dessous :

```
d4$lien <- factor(d4$lien, levels = c("enfant", "famille", "ami", "invite"))
tlien <- table(d4$espece, d4$lien)
tlien
##
##         enfant famille ami invite
##   chat      86     342 128     44
##   chien    122     296  76      6
```

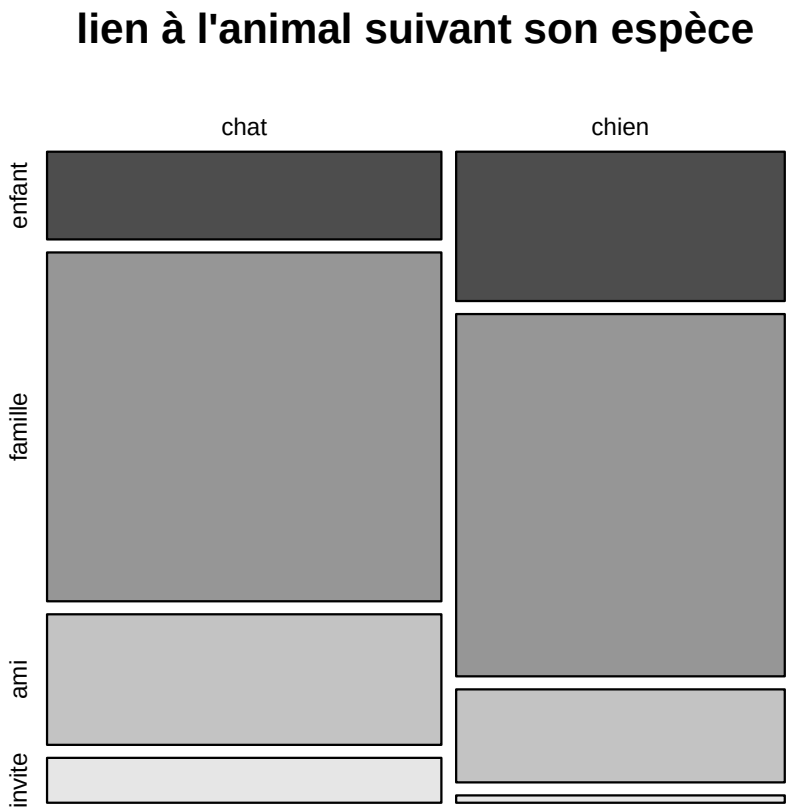
— **Si les données dont on dispose sont déjà sous forme d'une table de contingence**, il convient de saisir cette table de contingence dans **R** avec ses intitulés en utilisant les fonctions `matrix` puis `as.table`. Voici un exemple sur un jeu de données correspondant à l'étude de la nature du lien entre les animaux et leur maître :

```
m <- matrix(c(86, 342, 128, 44, 122, 296, 76, 6), nrow = 2, byrow = TRUE)
rownames(m) <- c("chat", "chien")
colnames(m) <- c("enfant","famille","ami","invite")
tlien <- as.table(m)
tlien
##       enfant famille ami invite
## chat      86     342 128     44
## chien    122     296  76      6
```

3.2.2 Représentation graphique de données à partir de la table de contingence

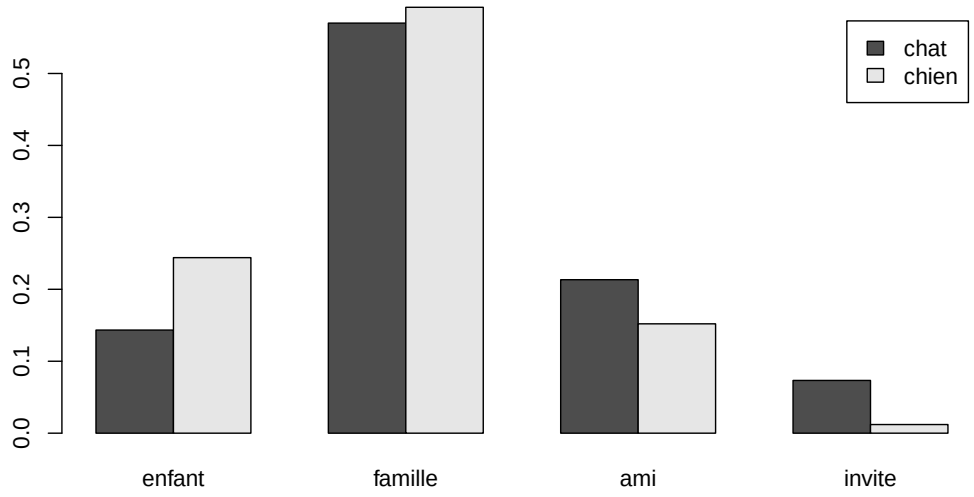
Une représentation en bandes d'une table de contingence peut être obtenue en utilisant simplement la fonction `plot` sur la table de contingence (voir section 3.2.1 pour le codage au préalable de la table de contingence).

```
plot(tlien, color = TRUE, main = "lien à l'animal suivant son espèce")
```



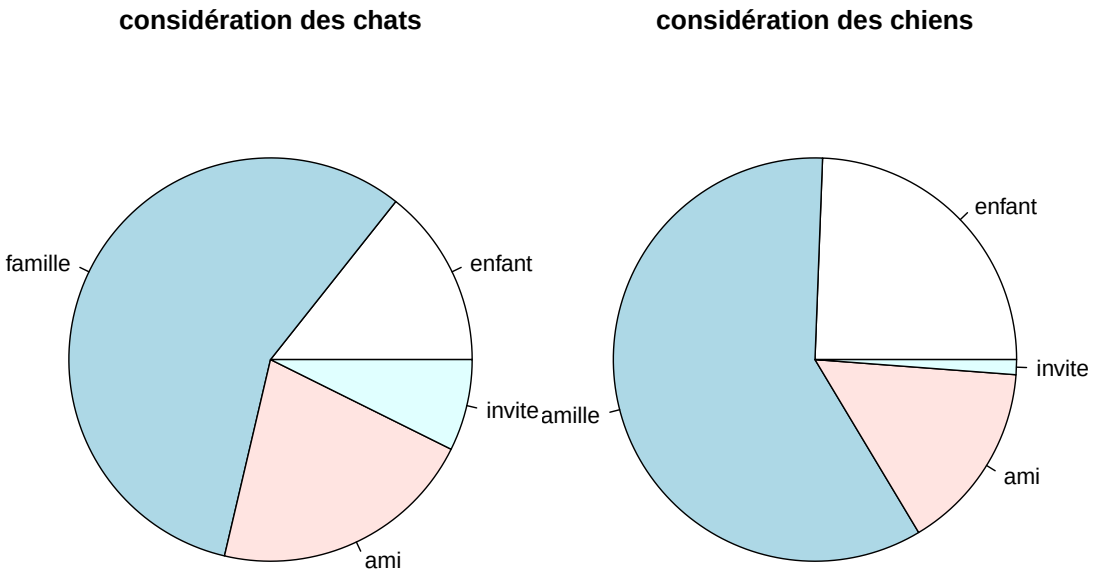
Il est aussi possible d'utiliser la fonction `prop.table` qui calcule automatiquement les fréquences sur chaque ligne avec l'argument `margin` fixé à 1 (avec l'argument `margin` fixé à 2 le calcul est fait pour chaque colonne) afin de représenter les deux diagrammes en bâtons côte à côte de la façon suivante à l'aide de la fonction `barplot` :

```
tlienf <- prop.table(tlien, margin = 1)
tlienf

##       enfant famille    ami invite
## chat  0.1433  0.5700 0.2133 0.0733
## chien 0.2440  0.5920 0.1520 0.0120

barplot(tlienf, beside = TRUE, legend.text = TRUE)
```



Les distributions peuvent aussi être représentées sous forme de diagrammes en secteurs comme ci-dessous :

```
# réduit les marges autours des graphes
par(mar = c(1, 1, 1, 1.5))
# partage de la fenêtre graphique en deux figures l'une à côté de l'autre
par(mfrow = c(1,2))
# réduit les marges autours des graphes
par(mar = c(0,0,1,0))
pie(tlien[1,], main = "considération des chats")
pie(tlien[2,], main = "considération des chiens")
```




```
par(mfrow = c(1,1))
```

3.2.3 Réalisation du test du χ^2 d'indépendance à partir de la table de contingence

On peut réaliser le test du χ^2 d'indépendance à partir de la table de contingence créée préalablement (voir section 3.2.1 pour le codage au préalable de la table de contingence).

```r
khi2 <- chisq.test(tlien)
khi2

##
##  Pearson's Chi-squared test
##
## data:  tlien
## X-squared = 43, df = 3, p-value = 3e-09
```

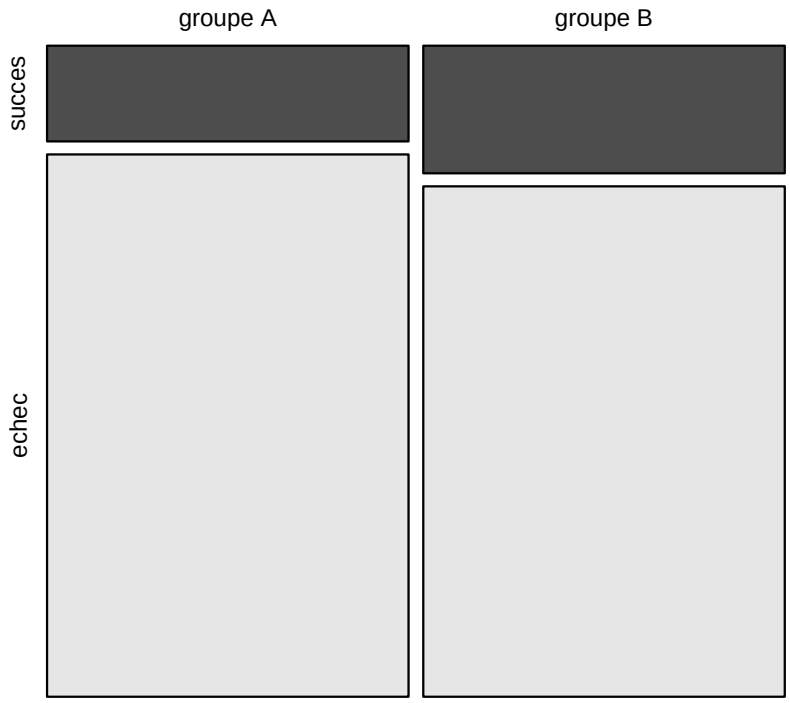
Le stockage du résultat de la fonction dans l'objet nommé ici `khi2` permet de récupérer ensuite quelques résultats non affichés par défaut, comme les effectifs théoriques (pour vérifier qu'ils sont tous supérieurs à 5) :

```r
print(khi2$expected)

##       enfant famille   ami invite
## chat   113.5     348 111.3   27.3
## chien   94.5     290  92.7   22.7
```

3.2.4 Cas particulier du test de comparaison de deux fréquences observées et intervalle de confiance autour de la différence entre deux fréquences

Dans le cas particulier de la comparaison de deux fréquences observées, on peut bien sûr utiliser le test du χ^2 comme décrit précédemment, mais on dispose aussi d'une fonction spécifique `prop.test` qui réalise le test du χ^2 d'indépendance et calcule l'intervalle de confiance autour de la différence entre les deux fréquences. Cette fonction peut être appelée avec comme argument la table de contingence créée préalablement soit à partir de données brutes soit directement à partir des données résumées (voir section 3.2.1 pour le codage au préalable de la table de contingence). Voici un exemple de codage correspondant à la comparaison de 2 fréquences de 15% et 20% observées chacune sur un échantillon de 200 animaux :

```r
# Dans cet exemple une nouvelle table de contingence est codée à partir des
# données résumées mais une table construite à partir des données brutes avec
# la fonction table() est bien sûr utilisable
t <- as.table(matrix(c(30, 170, 40, 160), nrow = 2, byrow = TRUE))
rownames(t) <- c("groupe A", "groupe B")
colnames(t) <- c("succes","echec")
plot(t, col = TRUE, main = "lien entre groupe et succès")
```




```
prop.test(t)
```

```
##
##  2-sample test for equality of proportions with continuity correction
##
## data:  t
## X-squared = 1, df = 1, p-value = 0.2
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  -0.1293  0.0293
## sample estimates:
## prop 1 prop 2
##   0.15   0.20
```

Attention, avant utilisation de l'intervalle de confiance sur la différence entre les 2 fréquences il est prudent de vérifier que les deux fréquences estimées, notées `prop 1` et `prop 2` dans la sortie de la fonction, correspondent bien aux deux fréquences dont on souhaite faire la différence. En effet, de façon peu intuitive, les fréquences sont calculées sur les lignes de la matrice donnée en argument de la fonction et non sur ses colonnes.

Enfin, toujours dans le cas de la comparaison de deux fréquences, un calcul exact de la p-value peut être réalisé à l'aide du test de Fisher, en appelant la fonction `fisher.test` avec comme argument la table de contingence. Ce test sera en particulier à utiliser lorsque les effectifs sont faibles. Voici son application sur les données précédentes :

```
fisher.test(t)
```

```
##
##  Fisher's Exact Test for Count Data
##
## data:  t
```

```
## p-value = 0.2
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.404 1.225
## sample estimates:
## odds ratio
##      0.707
```

4 Avec deux séries dépendantes (ou appariées) d'observations

4.1 Variable quantitative

Lorsque l'on souhaite comparer deux moyennes ou tendances centrales sur deux séries appariées, on peut importer les données sous la forme d'un tableau avec deux colonnes correspondant aux observations de la variable quantitative sur chacune des deux séries (colonnes nommées gestation1 et gestation2 dans cet exemple) afin de conserver l'information sur l'appariement des séries. On code ainsi les données de la même façon que s'il s'agissait de deux variables différentes (colonnes) mesurées sur les mêmes unités d'observation (lignes) :

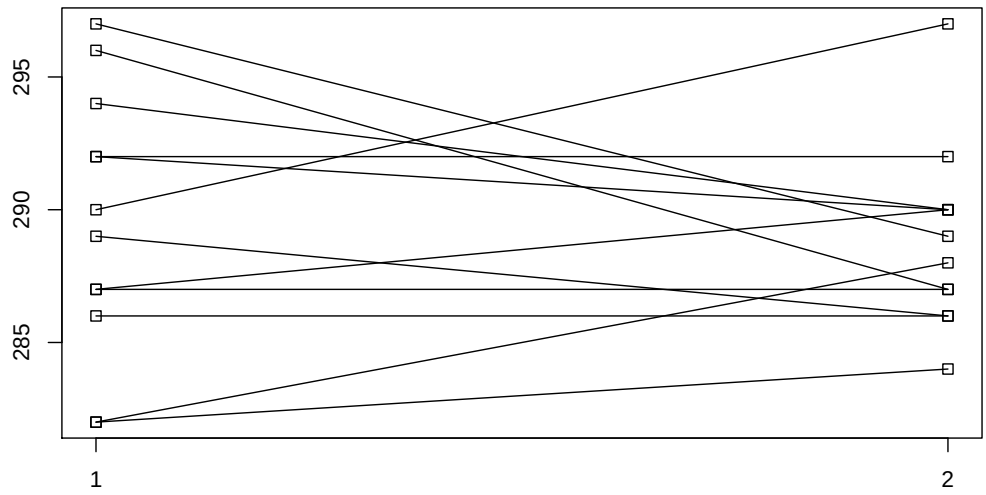
```
d5 <- read.table("gestation.txt",  header = TRUE, stringsAsFactors = TRUE)
str(d5)

## 'data.frame': 12 obs. of  2 variables:
##  $ gestation1: int  289 282 292 290 292 282 296 287 297 286 ...
##  $ gestation2: int  286 288 290 297 292 284 287 287 289 286 ...
```

ATTENTION, il est important de noter que le codage des données dans ce cas est différent de celui utilisé pour des séries indépendantes (avec des séries indépendantes dans un exemple de ce type, on aurait dans le jeu de données une colonne correspondant au numéro de gestation (gestation1 ou gestation2) et une colonne correspondant à la durée de gestation), codage qui ne permet pas de coder un appariement.

4.1.1 Visualisation des données appariées et examen de la distribution des différences

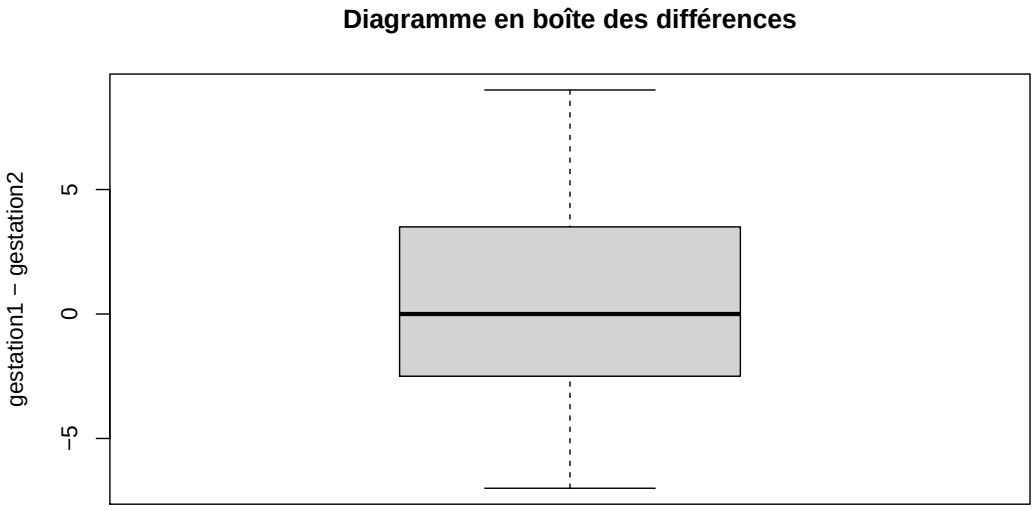
La figure ci-dessous, peu classique et pas très utile pour choisir le test à réaliser (paramétrique ou non paramétrique), permet de bien visualiser les résultats avec l'appariement entre les 2 séries d'observations.

```
stripchart(list(d5$gestation1, d5$gestation2), vertical = TRUE)
segments(1, d5$gestation1,  2, d5$gestation2)
```



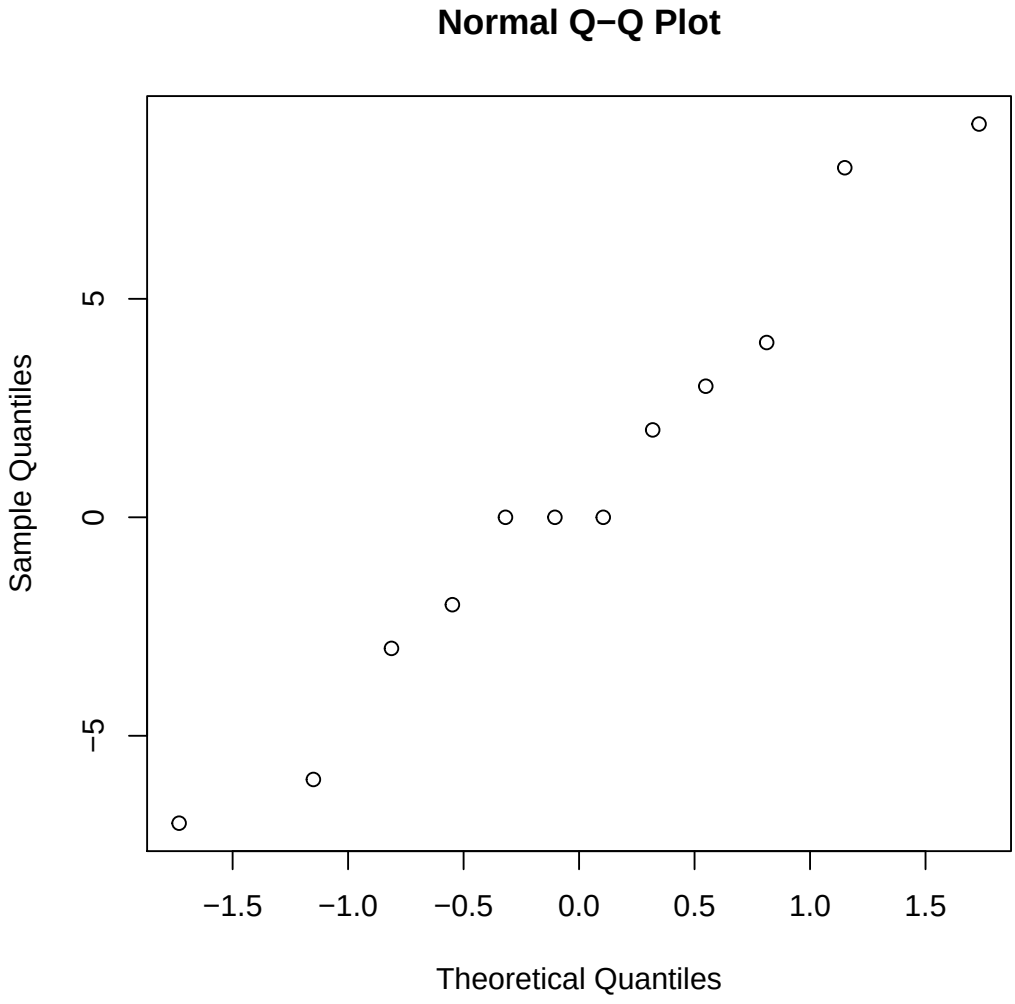
Plus utile pour le choix du test à réaliser, la distribution des différences entre les deux séries peut être représentée soit sous forme de diagramme en boîte comme ci-dessous, en appelant la fonction `boxplot` avec comme argument la différence entre les deux colonnes, soit à l'aide des fonctions `hist` dans le cas de grands effectifs ou de `stripchart` dans le cas de très petits effectifs.

```
boxplot(d5$gestation1 - d5$gestation2, main="Diagramme en boîte des différences",
        ylab = "gestation1 - gestation2")
```



Comme décrit précédemment dans le cas d'une série d'observations, la fonction `qqnorm` peut aussi être utilisée pour aider à juger de la normalité de la distribution des différences :

```
qqnorm(d5$gestation1 - d5$gestation2)
```



4.1.2 Test de comparaison de moyennes ou tendances centrales

Suivant que l'on pourra ou non supposer la distribution normale et suivant l'effectif, l'un des deux tests suivants pourra être réalisé :

— soit le **test T de comparaison de 2 moyennes sur séries appariées (Student des séries appariées)**

```
t.test(d5$gestation1, d5$gestation2, paired = TRUE)
##
##  Paired t-test
##
## data:  d5$gestation1 and d5$gestation2
## t = 0.5, df = 11, p-value = 0.6
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.46  3.80
## sample estimates:
## mean of the differences
##                   0.667
```

— soit le **test non paramétrique de Wilcoxon des rangs signés**

```
wilcox.test(d5$gestation1, d5$gestation2, paired = TRUE)
##
##  Wilcoxon signed rank test with continuity correction
##
## data:  d5$gestation1 and d5$gestation2
## V = 27, p-value = 0.6
## alternative hypothesis: true location shift is not equal to 0
```

4.1.3 Calcul de l'intervalle de confiance autour de la différence entre deux moyennes

Lorsque le test T de Student peut être utilisé, l'intervalle de confiance autour de la différence entre les deux moyennes est donné par la fonction `t.test` (sortie `conf.int`) et affiché en dernier dans la sortie de la fonction. Dans certains (ex. test d'équivalence) seul le calcul de l'intervalle de confiance intéressera l'utilisateur, et il aura peut-être intérêt, pour ne pas être parasité par le résultat du test de signification (p-value) à ne regarder que cet intervalle de confiance à l'aide da ligne de code suivante :

```
t.test(d5$gestation1, d5$gestation2, paired = TRUE)$conf.int
```

```
## [1] -2.46  3.80
## attr(,"conf.level")
## [1] 0.95
```

4.2 Variable qualitative à 2 modalités

4.2.1 création de la table de concordance

— **Si l'on dispose des données brutes**, la **table de concordance** peut être obtenue automatiquement à l'aide de la fonction `table` à partir des vecteurs correspondant aux deux séries appariées de la variable qualitative comme ci-dessous :

```
d8 <- read.table("testsouris.txt", header=TRUE, stringsAsFactors = TRUE)
str(d8)
## 'data.frame': 100 obs. of  2 variables:
##  $ test1: Factor w/ 2 levels "echec","succes": 2 2 2 2 2 2 2 2 2 2 ...
##  $ test2: Factor w/ 2 levels "echec","succes": 2 2 2 2 2 2 2 2 2 2 ...
tconcordance <- table(d8$test1, d8$test2)
tconcordance
##
##          echec succes
##   echec      6     18
##   succes     6     70
```

ATTENTION, il est important de noter que le codage des données dans ce cas est différent de celui utilisé pour des séries indépendantes : avec des séries indépendantes dans un exemple de ce type, on aurait dans le jeu de données une colonne correspondant au type de test (test1 ou test2) et une colonne correspondant au résultat du test, codage qui ne permet pas de coder un appariement.

— **Si l'on dispose directement de la table de concordance**, on peut la coder directement comme une table sous la forme suivante avec e, f, g, h (cf. illustration ci-dessous) les nombres de chaque combinaison possible de valeurs des deux séries (valeurs notées + ou - par exemple pour succès ou échec lors de la comparaison de deux tests diagnostiques), g+f correspondant au nombre total de discordances entre les deux séries :

	+	-
+	e	f
-	g	h

```
tconcordancebis <- as.table(matrix(c(70,6,18,6),nrow=2,byrow=TRUE))
rownames(tconcordancebis) <- c("TA+", "TA-")
colnames(tconcordancebis) <- c("TB+","TB-")
tconcordancebis
##     TB+ TB-
## TA+  70   6
## TA-  18   6
```

4.2.2 Réalisation du test du McNemar à partir de la table de concordance

Une fois la table de concordance obtenue, on réalise le test de McNemar de la façon suivante :

```
mcnemar.test(tconcordance)

##
##  McNemar's Chi-squared test with continuity correction
##
## data:  tconcordance
## McNemar's chi-squared = 5, df = 1, p-value = 0.02
```

5 Avec plusieurs séries indépendantes d'observations

5.1 Variable quantitative

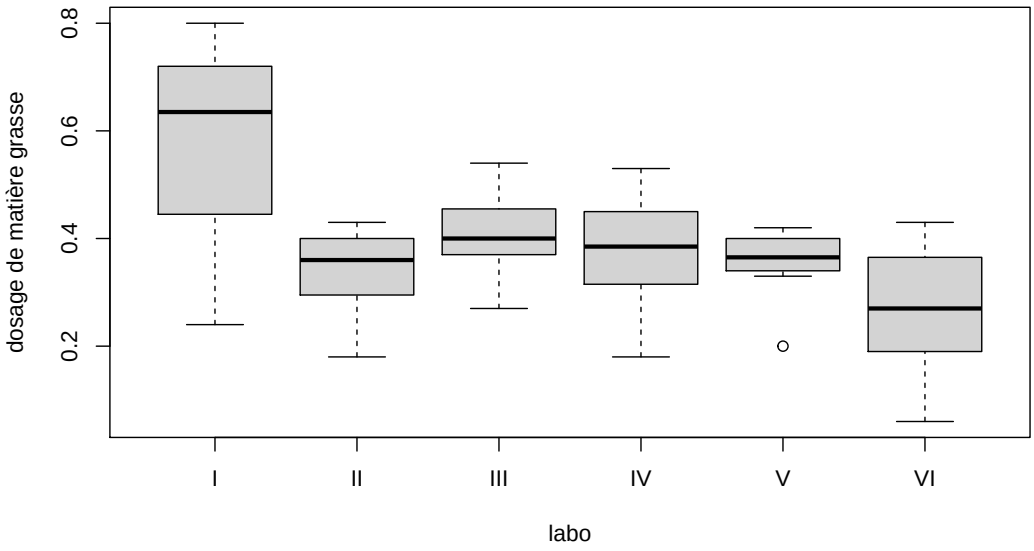
5.1.1 Test de comparaison globale de plusieurs moyennes ou tendances centrales

Comme dans le cas de la comparaison de deux moyennes observées sur des séries indépendantes, les données peuvent être importées sous la forme d'un tableau contenant deux colonnes, une colonne avec les valeurs observées de la variable quantitative (nommée "mgrasse" dans cet exemple) et une colonne avec l'indication du groupe d'appartenance de chaque observation (nommé "labo" dans cet exemple). L'ordre de présentation des colonnes et des lignes n'a pas d'importance.

```
d9 <- read.table("egg.txt", header = TRUE, stringsAsFactors = TRUE)
head(d9)

##   mgrasse labo
## 1    0.62    I
## 2    0.55    I
## 3    0.34    I
## 4    0.24    I
## 5    0.80    I
## 6    0.68    I
```

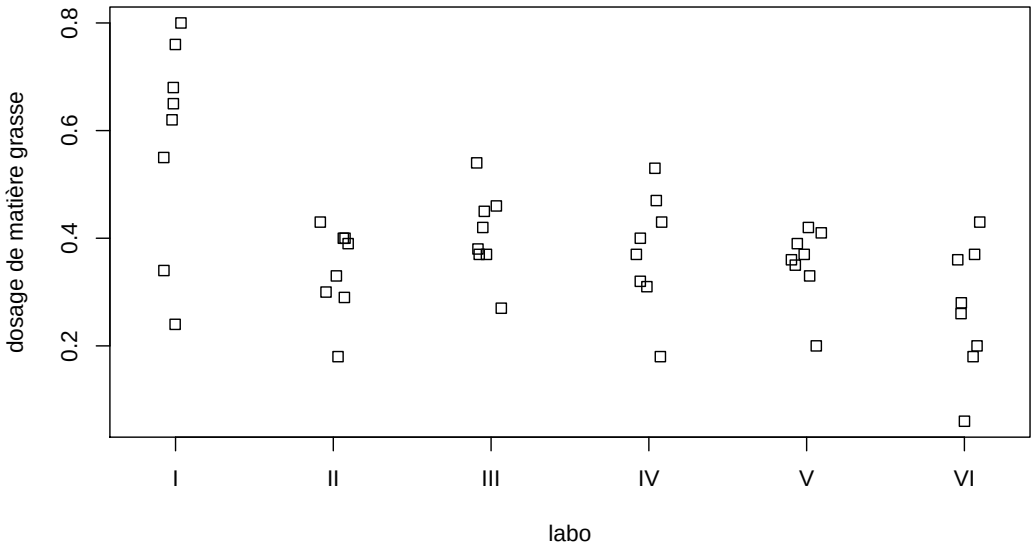
Les diagrammes en boîte des distributions dans chaque groupe peuvent être facilement tracés :

```
plot(d9$mgrasse ~ d9$labo, xlab = "labo", ylab = "dosage de matière grasse")
```



La fonction `stripchart` permettant de représenter tous les points observés par groupe est plus adaptée dans le cas de très faibles effectifs.

```
stripchart(d9$mgrasse ~ d9$labo, vertical = TRUE, method = "jitter",
           xlab = "labo", ylab ="dosage de matière grasse")
```



En fonction de l'exploration des distributions (normalité, effectifs, égalité des variances), l'un des trois tests suivants peut être réalisé :

— soit le **test de l'analyse de variance classique avec variances égales**

```
oneway.test(d9$mgrasse ~ d9$labo, var.equal = TRUE)
##
##  One-way analysis of means
##
## data:  d9$mgrasse and d9$labo
## F = 6, num df = 5, denom df = 42, p-value = 2e-04
# calcul complémentaire des moyennes par groupe pour la description des données
tapply(d9$mgrasse, d9$labo, mean)
##     I    II   III    IV     V    VI
## 0.580 0.340 0.408 0.376 0.354 0.268
```

— soit le **test de l'analyse de variance avec variances différentes (extension du test de Welch)** :

```r
oneway.test(d9$mgrasse ~ d9$labo,var.equal = FALSE)
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  d9$mgrasse and d9$labo
## F = 3, num df = 5, denom df = 19, p-value = 0.03
# calcul complémentaire des moyennes par groupe pour la description des données
tapply(d9$mgrasse, d9$labo, mean)
##     I    II   III    IV     V    VI
## 0.580 0.340 0.408 0.376 0.354 0.268
```

— soit le **test non paramétrique de comparaison de plusieurs tendances centrales de Kruskal-Wallis** :

```r
kruskal.test(d9$mgrasse ~ d9$labo)
##
##  Kruskal-Wallis rank sum test
##
## data:  d9$mgrasse by d9$labo
## Kruskal-Wallis chi-squared = 14, df = 5, p-value = 0.02
# calcul complémentaire des médianes par groupe pour la description des données
tapply(d9$mgrasse, d9$labo, median)
##     I    II   III    IV     V    VI
## 0.635 0.360 0.400 0.385 0.365 0.270
```

5.1.2 Test de comparaison deux à deux de plusieurs moyennes ou tendances centrales

Dans le cas de la mise en évidence d'une différence globale significative entre les moyennes, il est possible, si besoin, de comparer les moyennes deux à deux en corrigeant les valeurs de p-value par la méthode historique de Bonferroni ou par sa version améliorée dite de Bonferroni-Holm (choix par défaut dans R si on indique pas la méthode), ou par une autre méthode spécifiée dans l'argument `p.adjust.method` des fonctions ci-dessous, en cohérence avec le choix du test global réalisé auparavant (paramétrique ou non, avec variances égales ou non),

— ex. d'un **test de comparaison 2 à 2 paramétrique avec estimation d'une variance commune et la correction de Bonferroni** :

```r
pairwise.t.test(d9$mgrasse, d9$labo, pool.sd = TRUE, p.adjust.method = "bonferroni")
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  d9$mgrasse and d9$labo
##
##     I     II    III   IV    V
## II  0.003 -     -     -     -
## III 0.081 1.000 -     -     -
## IV  0.018 1.000 1.000 -     -
## V   0.006 1.000 1.000 1.000 -
## VI  6e-05 1.000 0.327 1.000 1.000
##
## P value adjustment method: bonferroni
```

— ex. d'un **test de comparaison 2 à 2 paramétrique avec estimation d'une variance commune et la correction de Bonferroni-Holm** :

```r
pairwise.t.test(d9$mgrasse, d9$labo, pool.sd = TRUE, p.adjust.method = "holm")
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  d9$mgrasse and d9$labo
##
##     I     II    III   IV    V
## II  0.003 -     -     -     -
## III 0.059 1.000 -     -     -
## IV  0.015 1.000 1.000 -     -
## V   0.005 1.000 1.000 1.000 -
## VI  6e-05 1.000 0.218 0.641 1.000
##
## P value adjustment method: holm
```

— ex. d'un **test de comparaison 2 à 2 paramétrique avec estimation d'une variance par groupe et la correction de Bonferroni-Holm** :

```
pairwise.t.test(d9$mgrasse, d9$labo, pool.sd = FALSE, p.adjust.method = "holm")
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  d9$mgrasse and d9$labo
##
##     I    II   III  IV   V
## II  0.15 -    -    -    -
## III 0.47 0.84 -    -    -
## IV  0.29 1.00 1.00 -    -
## V   0.18 1.00 1.00 1.00 -
## VI  0.04 1.00 0.21 0.70 0.84
##
## P value adjustment method: holm
```

— ex. d'un **test de comparaison 2 à 2 non paramétrique et la correction de Bonferroni-Holm** :

```
pairwise.wilcox.test(d9$mgrasse, d9$labo, p.adjust.method = "holm")
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  d9$mgrasse and d9$labo
##
##     I   II  III IV  V
## II  0.4 -   -   -   -
## III 0.8 1.0 -   -   -
## IV  0.5 1.0 1.0 -   -
## V   0.5 1.0 1.0 1.0 -
## VI  0.2 1.0 0.2 0.8 1.0
##
## P value adjustment method: holm
```

5.1.3 Test de comparaison globale de plusieurs variances

Dans le cas de distributions proches de lois normales, le test de Bartlett peut être réalisé à l'aide de la fonction `bartlett.test`, pour mettre en évidence une différence globale entre plusieurs variances. **Rappelons qu'il ne permet pas, par contre, de montrer l'égalité des variances** :

```
bartlett.test(d9$mgrasse ~ d9$labo)

##
##  Bartlett test of homogeneity of variances
##
## data:  d9$mgrasse by d9$labo
## Bartlett's K-squared = 11, df = 5, p-value = 0.06
```

5.2 Variable qualitative

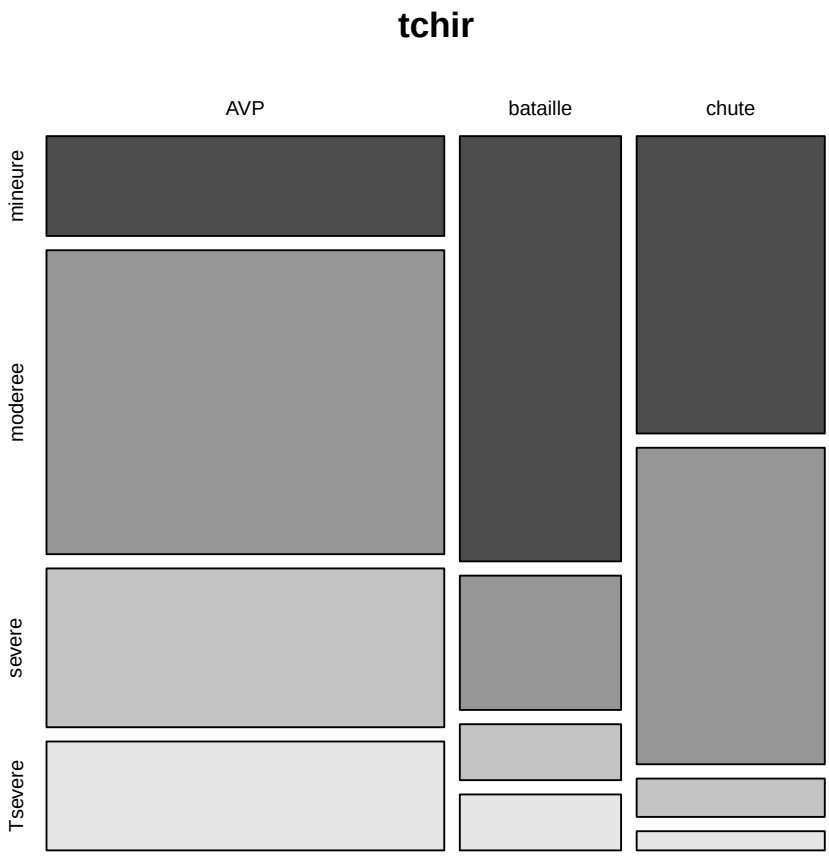
Le cas de la comparaison de plusieurs séries indépendantes pour une variable qualitative équivaut à la comparaison de plusieurs distributions ou plusieurs fréquences sur séries indépendantes et se traite par un test du χ^2 d'indépendance comme expliqué au chapitre 3.2 pour la comparaison de deux distributions ou fréquences. Voici un exemple sur un jeu de données correspondant à l'étude de la cause et de la gravité des traumatismes d'animaux admis au service de chirurgie sur le campus vétérinaire de Lyon.

```
d10 <- read.table('chirurgmodifie.txt', header = TRUE, stringsAsFactors = TRUE)
str(d10)

## 'data.frame': 278 obs. of  2 variables:
##  $ gravite: Factor w/ 4 levels "mineure","moderee",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ cause  : Factor w/ 3 levels "AVP","bataille",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```r
# Etape préliminaire de construction de la table de contingence
tchir <- table(d10$cause, d10$gravite)
tchir
```

```
##
##             mineure moderee severe Tsevere
##   AVP            22      67     35      24
##   bataille       38      12      5       5
##   chute          31      33      4       2
```

```r
# Visualisation des données
plot(tchir, col = TRUE)
```




```r
# test du khi2
khi2 <- chisq.test(tchir)
khi2
```

```
##
##  Pearson's Chi-squared test
##
## data:  tchir
## X-squared = 63, df = 6, p-value = 1e-11
```

6 Avec plusieurs séries dépendantes d'observations

6.1 Variable quantitative

ATTENTION, ce cas n'est pas traité dans le cours de base, car il nécessite l'abord de l'analyse de variance à plusieurs facteurs et des modèles mixtes qui est un sujet vaste et relativement délicat (abordé en optionnel de 5A).

6.2 Variable qualitative à 2 modalités

Les données doivent être importées sous la forme d'un tableau contenant trois colonnes, l'une codant le facteur étudié (celui qui différencie les séries, nommé site dans cet exemple), une autre codant la variable qualitative à deux modalités observée (nommé detection dans cet exemple), et enfin une dernière codant le facteur d'appariement des séries (facteur bloc, nommé carcasse dans cet exemple)

```
d11 <- read.table("carcasses.txt", header = TRUE, stringsAsFactors = TRUE)
head(d11)

##   site carcasse detection
## 1   BV        1         0
## 2   EC        1         0
## 3   GC        1         1
## 4   JC        1         0
## 5   JV        1         0
## 6   LC        1         0
```

Si l'on souhaite calculer au préalable les fréquences à comparer, d'une des deux modalités de la variable étudiée en fonction du facteur étudié (indépendamment du facteur bloc), on peut simplement calculer les moyennes de la variable qualitative à deux modalités (codée 0 ou 1) pour chaque série de la fonction suivante :

```
tapply(d11$detection, d11$site, mean)

##     BV     EC     GC     JC     JV     LC     PC     PV
## 0.0303 0.1515 0.1515 0.0606 0.0000 0.2424 0.2424 0.1212
```

Le test de Cochran-Mantel-Haenszel peut ensuite être réalisé à l'aide de la fonction `mantelhaen.test` en indiquant en premier argument la variable qualitative à deux modalités, en deuxième argument la variable codant la série d'appartenance, et en troisième argument le facteur d'appariement des séries :

```
mantelhaen.test(as.factor(d11$detection), d11$site,as.factor(d11$carcasse))

##
##  Cochran-Mantel-Haenszel test
##
## data:  as.factor(d11$detection) and d11$site and as.factor(d11$carcasse)
## Cochran-Mantel-Haenszel M^2 = 19, df = 7, p-value = 0.007

# rappelons que la fonction as.factor permet de transformer un variable
# codée numériquement (ici avec des 0 et des 1) en variable qualitative
```

7 Lorsque deux variables quantitatives sont observées sur les mêmes unités d'observation (corrélation ou régression linéaire)

7.0.1 Corrélation linéaire entre deux variables observées

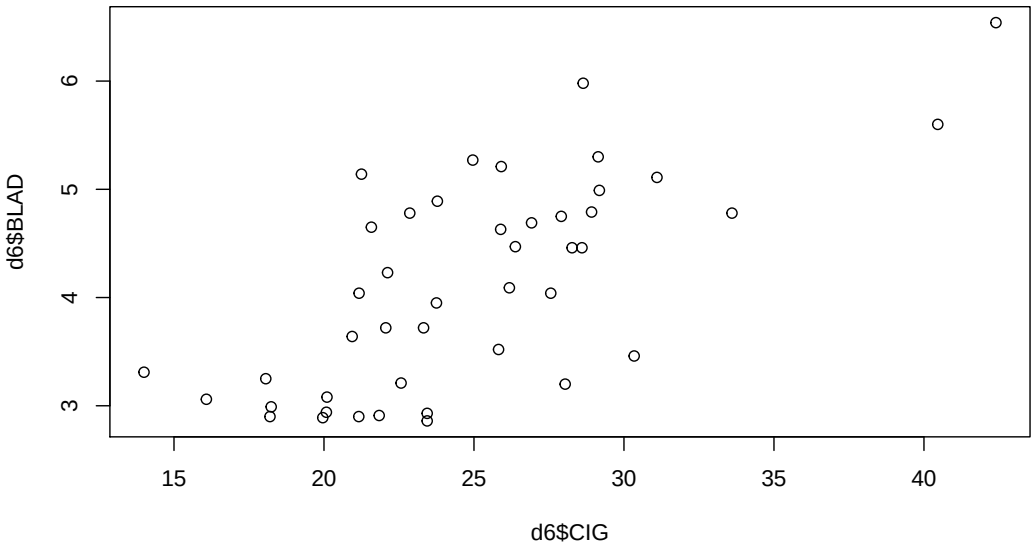
Les données seront importées sous la forme d'un tableau contenant les deux colonnes correspondant aux deux variables observées à corréler, dans cet exemple la consommation de cigarettes nommée "CIG" et la mortalité par cancer de la vessie nommée "BLAD".

```
d6 <- read.table("smoking.txt", header = TRUE, stringsAsFactors = TRUE)
str(d6)

## 'data.frame': 44 obs. of  6 variables:
##  $ STATE: Factor w/ 44 levels "AK","AL","AR",..: 2 4 3 5 6 8 7 9 10 11 ...
##  $ CIG  : num  18.2 25.8 18.2 28.6 31.1 ...
```

```
##  $ BLAD : num  2.9 3.52 2.99 4.46 5.11 4.78 5.6 4.46 3.08 4.75 ...
##  $ LUNG : num  17.1 19.8 16 22.1 22.8 ...
##  $ KID  : num  1.59 2.75 2.02 2.66 3.35 3.36 3.13 2.41 2.46 2.95 ...
##  $ LEUK : num  6.15 6.61 6.94 7.06 7.2 6.45 7.08 6.07 6.62 7.27 ...
```

Les données peuvent être représentées sous la forme d'un diagramme de dispersion (ou nuage de points) tout simplement avec la fonction `plot`.

```
plot(d6$CIG, d6$BLAD)
```



Suivant la forme plus ou moins elliptique du nuage de points, on pourra effectuer l'un des deux tests de corrélations suivants :

— soit le **test paramétrique du coefficient de corrélation linéaire de Pearson**

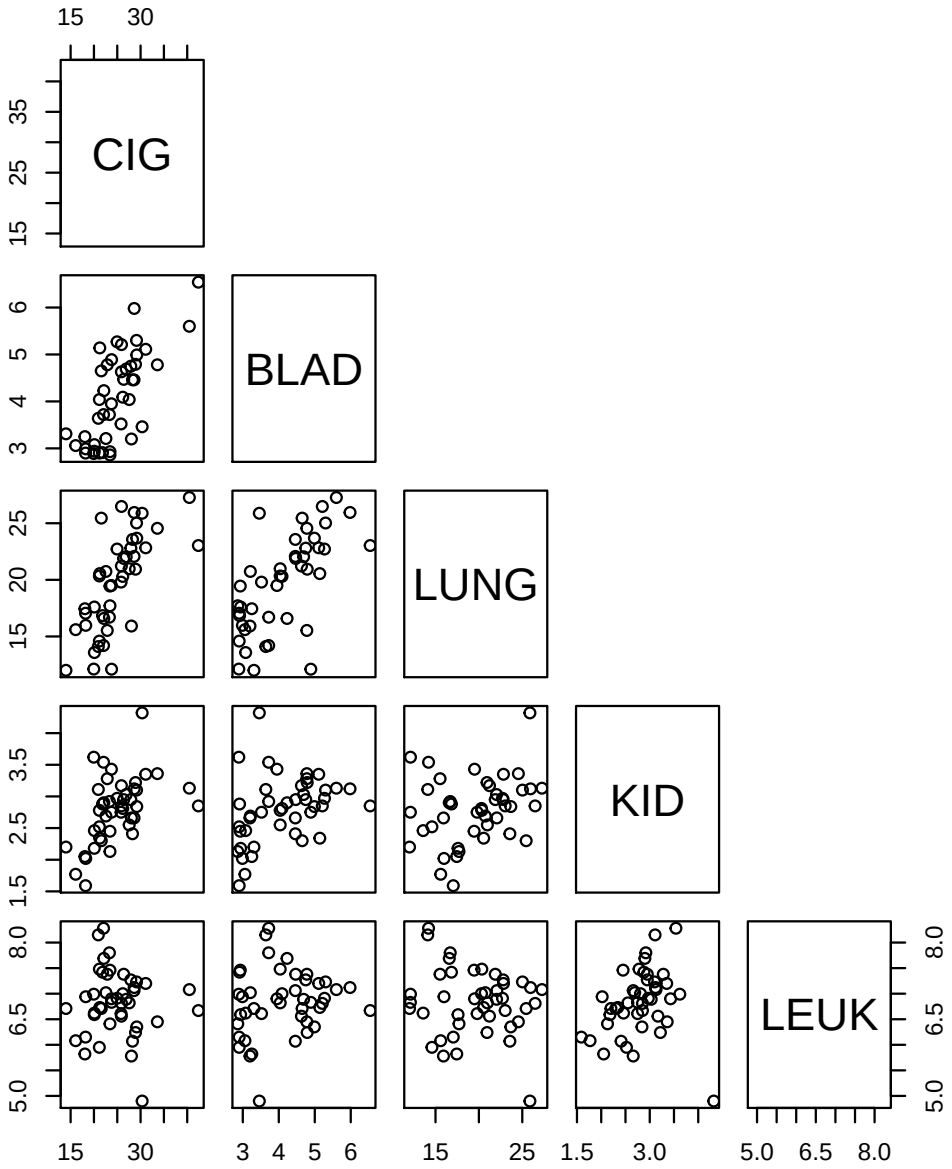
```
cor.test(d6$CIG, d6$BLAD, method = "pearson")
##
##  Pearson's product-moment correlation
##
## data:  d6$CIG and d6$BLAD
## t = 6, df = 42, p-value = 1e-07
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.514 0.828
## sample estimates:
##   cor
## 0.704
```

— soit le **test non paramétrique du coefficient de corrélation de rangs de Spearman**

```
cor.test(d6$CIG, d6$BLAD, method = "spearman")
##
##  Spearman's rank correlation rho
##
## data:  d6$CIG and d6$BLAD
## S = 4688, p-value = 7e-07
## alternative hypothesis: true rho is not equal to 0
## sample estimates:
##  rho
## 0.67
```

Il existe dans **R** une fonction souvent utile, qui permet de représenter en une seule fois tous les diagrammes de dispersion entre les variables quantitatives d'un jeu de données (dans l'exemple ci-dessous les colonnes 2 à 5 du jeu de données `d6`) prises 2 à 2.

```
pairs(d6[,2:6], upper.panel = NULL)
```



De la même façon, on peut aussi calculer en une seule fois tous les coefficients de corrélation (paramétriques ou non paramétriques suivant l'argument `method`) entre toutes les variables d'un jeu de données prises 2 à 2 :

```
cor(d6[,2:6], method = "spearman")
```

```
##          CIG  BLAD    LUNG   KID    LEUK
## CIG   1.0000 0.670  0.7502 0.513 -0.0245
## BLAD  0.6696 1.000  0.6606 0.440  0.1825
## LUNG  0.7502 0.661  1.0000 0.269 -0.0796
## KID   0.5134 0.440  0.2688 1.000  0.3802
## LEUK -0.0245 0.182 -0.0796 0.380  1.0000
```

7.0.2 Régression linéaire entre une variable observée et une variable contrôlée

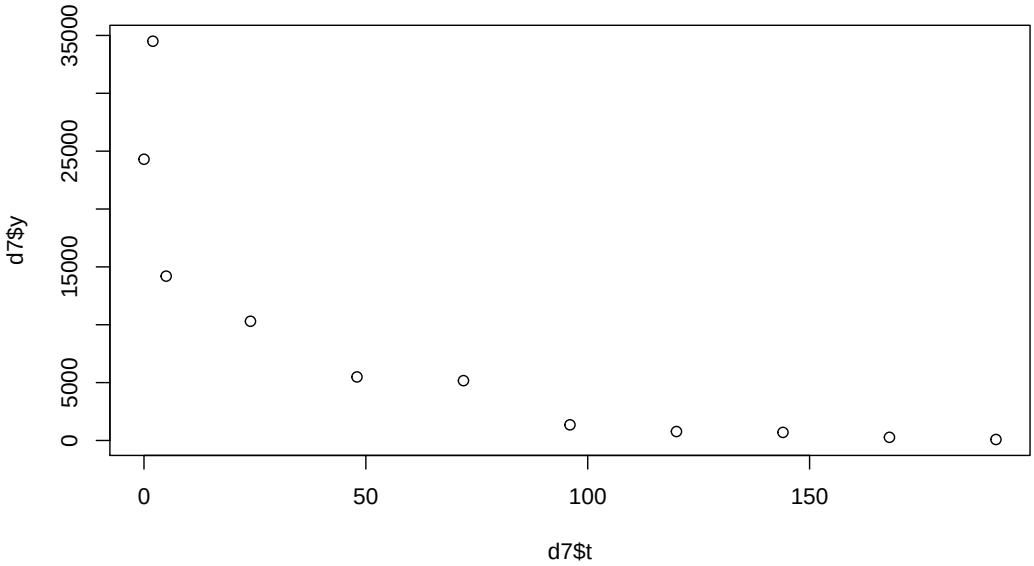
Les données seront importées sous la forme d'un tableau contenant les deux colonnes correspondant à la variable de contrôle ("t" dans cet exemple), et à la variable observée ("y" dans cet exemple).

```
d7<- read.table("survyaourtbrut.txt", header = TRUE, stringsAsFactors = TRUE)
str(d7)
```

```
## 'data.frame': 11 obs. of  2 variables:
##  $ t: int  0 2 5 24 48 72 96 120 144 168 ...
##  $ y: num  24300 34500 14200 10300 5490 5170 1350 773 699 273 ...
```

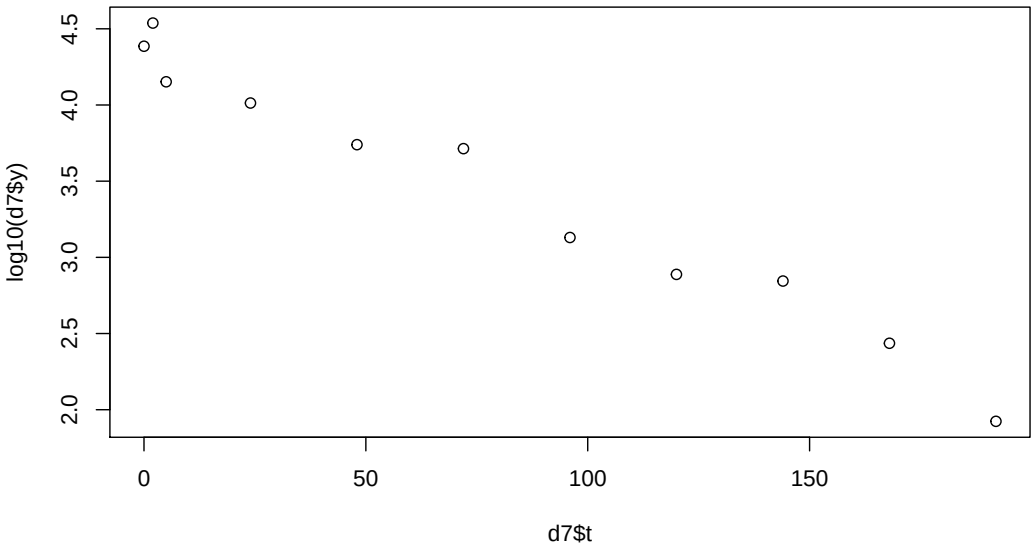
Les données pourront être représentées sous forme de diagramme de dispersion (ou nuage de points) avec la fonction `plot` en indiquant bien la variable contrôlée comme premier argument :

```
plot(d7$t, d7$y)
```



Dans cet exemple il conviendra de tracer les données avec la variable observée transformée en logarithme décimal :

```
plot(d7$t, log10(d7$y))
```



L'ajustement d'un modèle linéaire aux données par régression peut être réalisé à l'aide de la fonction `lm` (pour "linear model"). Il convient de lui mettre en argument la formule du modèle, composée du nom de la variable observée, suivi du signe $\sim$ et du nom de la variable de contrôle :

```
modele<- lm(log10(y) ~ t, data = d7)
summary(modele)
```

```
##
## Call:
## lm(formula = log10(y) ~ t, data = d7)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.1697 -0.0914 -0.0559  0.1236  0.1943
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.381893   0.067820    64.6  2.6e-13
## t           -0.011982   0.000656   -18.3  2.0e-08
##
## Residual standard error: 0.145 on 9 degrees of freedom
## Multiple R-squared:  0.974,Adjusted R-squared:  0.971
## F-statistic:  334 on 1 and 9 DF,  p-value: 2.02e-08
```

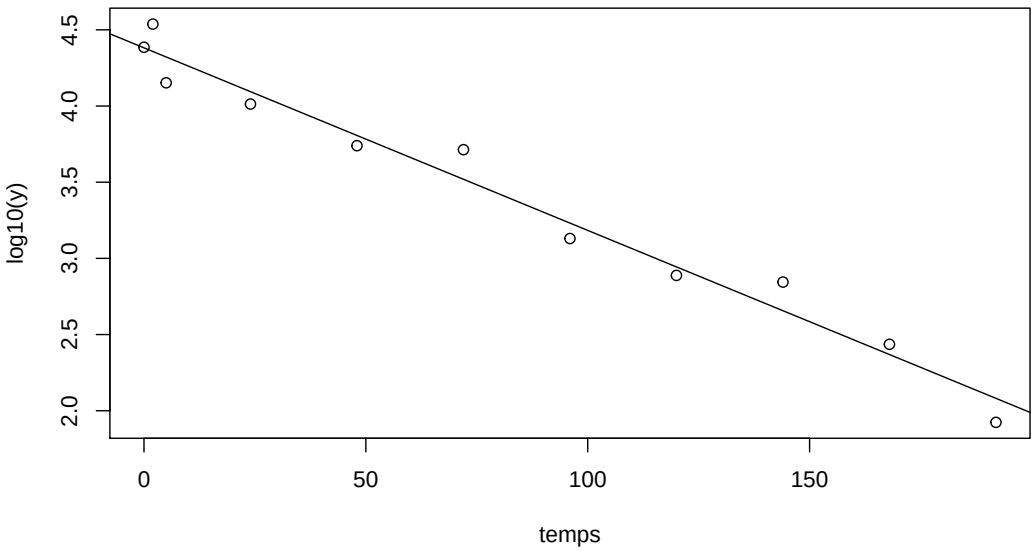
Les intervalles de confiance sur les paramètres du modèle peuvent être obtenus à l'aide de la fonction `confint` :

```
confint(modele)
```

```
##                2.5 %  97.5 %
## (Intercept)  4.2285  4.5353
## t           -0.0135 -0.0105
```

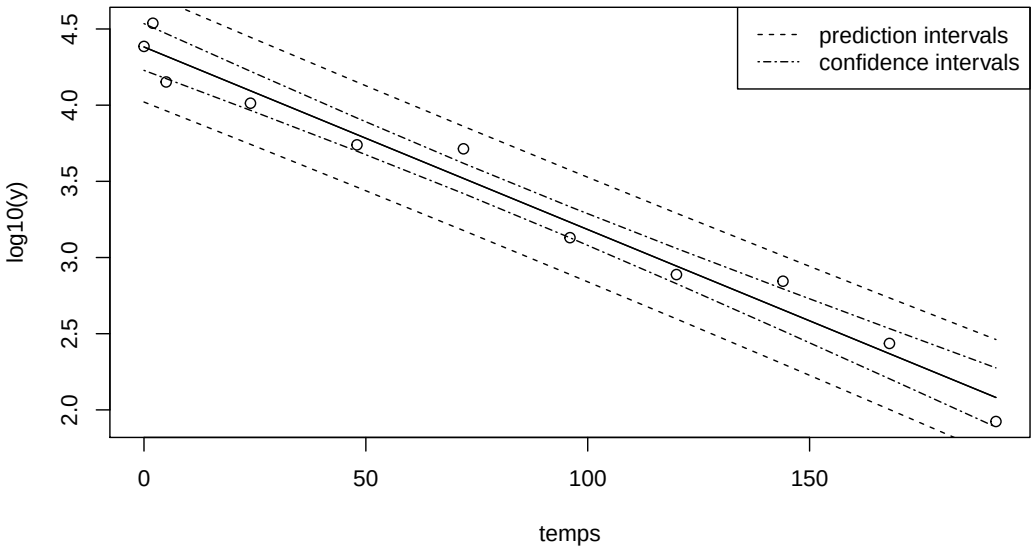
La droite ajustée peut être ajoutée au diagramme de dispersion réalisé préalablement à l'aide de la fonction `abline` :

```
plot(d7$t, log10(d7$y), xlab="temps", ylab="log10(y)") # xlab et ylab spécifient les légendes des axes
abline(modele)
```



On peut représenter les bandes de confiance associées à ces deux types d'intervalles en utilisant le code suivant.

```
ipred <- predict(modele, interval = "prediction")
iconf <- predict(modele, interval = "confidence")
plot(d7$t, log10(d7$y), xlab = "temps", ylab = "log10(y)")
matlines(d7$t, ipred, lty = c(1,2,2), col = "black")
matlines(d7$t, iconf, lty = c(1,6,6), col = "black")
legend("topright", legend = c("prediction intervals", "confidence intervals"), lty = c(2, 6))
```

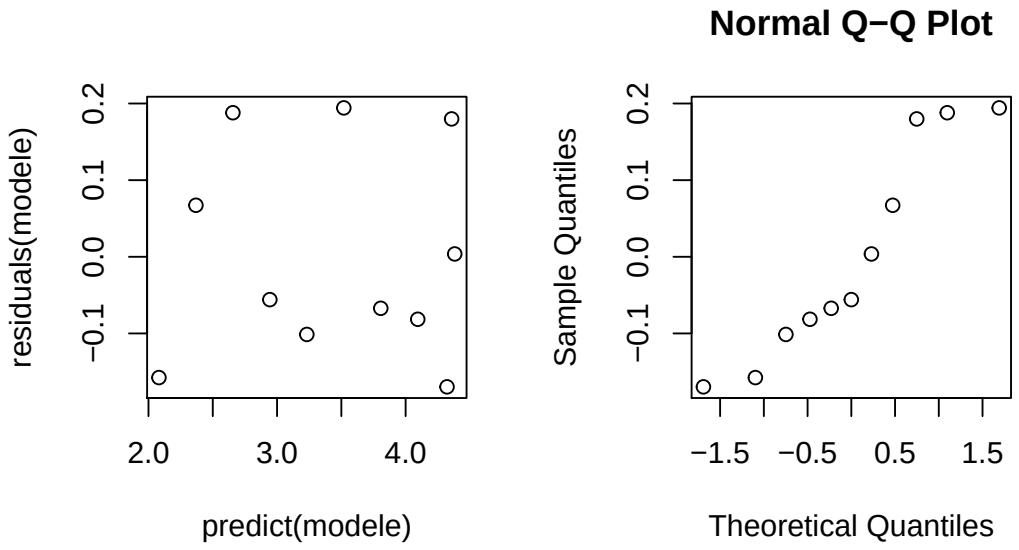


On peut obtenir les valeurs prédites pour de nouvelles valeurs de la variable explicative ainsi que les intervalles de confiance sur la valeur prédite (interval = "prediction") ou de confiance sur la moyenne prédite (interval = "confidence") ainsi :

```
# Prédiction de log10y pour t = 100 et 150
predict(modele, data.frame(t = c(100, 150)), interval = "prediction")

##    fit  lwr  upr
## 1 3.18 2.84 3.53
## 2 2.58 2.23 2.94
```

Enfin il est utile d'examiner les résidus (récupérés à l'aide de la fonction `residuals`) sous différentes formes :

```
par(mfrow = c(1, 2)) # partage la fenêtre graphique en 2 lignes
plot(predict(modele), residuals(modele))
qqnorm(residuals(modele))
```




```
par(mfrow = c(1,1))
```

8 Calcul de puissance ou détermination *a priori* du nombre d'observations nécessaires

8.1 Tests sur deux séries indépendantes d'observations

8.1.1 Variable quantitative

Prenons un exemple : on veut comparer la production laitière de vaches de race Holstein-Frisonne élevées dans deux régions différentes de l'Angleterre. La production moyenne attendue pour ces vaches en Angleterre est d'environ 6500 kg par lactation (période de 305 jours en moyenne). On veut savoir combien d'animaux il faudrait observer par échantillon (pris dans chaque région) pour avoir une probabilité de 85% de détecter avec un risque de 5% une différence moyenne de production de 250 kg. Des informations publiées précédemment indiquent un écart type de la production laitière de 1425 kg pour cette race bovine.

La fonction `power.t.test` permet de répondre à cette question en supposant qu'un test T avec variances égales sera applicable. Il faut affecter l'argument `type` de la fonction à `"two.sample"`, spécifier le type d'hypothèse alternative dans l'argument `alternative` (par défaut "two.sided" pour un spécifier un test bilatéral), et affecter les arguments numériques nécessaires, pour le calcul de l'argument affecté à `NULL`. `n` représente l'effectif de chaque groupe, `delta` la différence qu'on veut pouvoir détecter, `sd` l'écart type supposé commun dans les 2 groupes, `sig.level` le risque de première espèce et `power` la puissance du test.

Ainsi pour répondre à la question posée dans l'exemple il faudra saisir le code suivant :

```
power.t.test(n = NULL, delta = 250, sd = 1425, sig.level = 0.05, power = 0.85,
             type = "two.sample", alternative = "two.sided")

##
##      Two-sample t test power calculation
##
##               n = 584
##           delta = 250
##              sd = 1425
##       sig.level = 0.05
##           power = 0.85
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Si maintenant nous voulions connaître la puissance du test réalisé avec un effectif de 100 vaches par groupe, il faudrait saisir le code suivant :

```
power.t.test(n = 100, delta = 250, sd=1425, sig.level = 0.05, power = NULL,
             type = "two.sample", alternative = "two.sided")

##
##      Two-sample t test power calculation
##
##               n = 100
##           delta = 250
##              sd = 1425
##       sig.level = 0.05
##           power = 0.234
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Nous pourrions aussi calculer la différence que le test aurait une probabilité de 85% de détecter toujours avec un effectif de 100 vaches par groupe en saisissant :

```
power.t.test(n = 100, delta = NULL, sd = 1425, sig.level = 0.05, power = 0.85,
             type = "two.sample", alternative = "two.sided")

##
##      Two-sample t test power calculation
##
##               n = 100
##           delta = 607
```

```
##              sd = 1425
##       sig.level = 0.05
##           power = 0.85
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

8.1.2 Variable qualitative

Prenons un exemple : *Toxocara canis* est un parasite commun du chien qui peut provoquer une cécité chez les enfants contaminés. On veut savoir si la prévalence d'infestation par ce parasite est réellement plus importante chez les chiots (âgés de moins de 6 mois) que chez les chiens adultes (âgés de plus d'un an) en zone urbaine. On s'attend à une prévalence d'environ 30% chez les chiots et 5% chez les adultes. En se fixant un risque de première espèce de 5% et une puissance de 90%, combien faut-il que l'on observe d'animaux de chaque groupe ?

La fonction `power.prop.test` permet de répondre à cette question en supposant qu'un test de comparaison de deux fréquences (équivalent au test du χ^2 d'indépendance) sera applicable (à vérifier *a posteriori*). Il faut spécifier le type d'hypothèse alternative dans l'argument `alternative` (par défaut "two.sided" pour un spécifier un test bilatéral), et affecter les arguments numériques nécessaires, pour le calcul de l'argument affecté à `NULL`. `n` représente l'effectif de chaque groupe, `p1` la fréquence attendue dans le premier groupe, `p2` la fréquence attendue dans le deuxième groupe, `sig.level` le risque de première espèce et `power` la puissance du test.

Ainsi pour répondre à la question posée dans l'exemple il faudra saisir le code suivant :

```
power.prop.test(n = NULL, p1 = 0.3, p2 = 0.05, sig.level = 0.05,
                power = 0.9, alternative = "two.sided")
```

```
##
##      Two-sample comparison of proportions power calculation
##
##               n = 46.4
##              p1 = 0.3
##              p2 = 0.05
##       sig.level = 0.05
##           power = 0.9
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

8.2 Tests sur deux séries dépendantes (ou appariées) d'observations

8.2.1 Variable quantitative

Prenons un exemple : on veut comparer l'effet de deux exercices d'entraînement de chevaux sur tapis roulant sur la concentration plasmatique en lactate. Pour ceci on conduit un essai de type " cross-over " où chaque sujet est pris comme son propre témoin, c'est-à-dire que chaque cheval subit successivement et dans un ordre tiré au sort les deux exercices à comparer. Une différence de 1 mmol/l est considérée comme intéressante d'un point de vue biologique. Une étude pilote à permis d'estimer l'écart type des différences entre les concentrations plasmatiques en lactate entre les deux exercices à 1.7 mmol/l. Combien faudrait-il de chevaux pour avoir une probabilité de 90% de détecter avec un risque de 1% une différence de 1 mmol/l entre les deux exercices d'entraînement ?

La fonction `power.t.test` permet de répondre à ces questions en supposant qu'un test T de comparaison de séries appariées sera applicable. Il faut affecter l'argument `type` de la fonction à `"paired"`, spécifier le type d'hypothèse alternative dans l'argument `alternative` (par défaut "two.sided" pour un spécifier un test bilatéral), et affecter les arguments numériques nécessaires, pour le calcul de l'argument affecté à `NULL`. `n` représente le nombre de paires d'observations, `delta` la différence qu'on veut pouvoir détecter, `sd` l'écart type des différences, `sig.level` le risque de première espèce et `power` la puissance du test.

Pour répondre à cette question, il faut saisir le code suivant :

```
power.t.test(n = NULL, delta = 1, sd = 1.7, sig.level = 0.01, power = 0.90,
             type = "paired", alternative = "two.sided")
```

```
##
##      Paired t test power calculation
##
##               n = 46.4
```

```
##           delta = 1
##              sd = 1.7
##       sig.level = 0.01
##           power = 0.9
##     alternative = two.sided
##
## NOTE: n is number of *pairs*, sd is std.dev. of *differences* within pairs
```

9 Conclusion

Maintenant à vous de jouer et n'hésitez pas à utiliser l'aide en ligne de **R** afin de mieux vous approprier ce langage de programmation qui peut paraître un peu difficile à maîtriser au premier abord, mais qui s'avère très puissant dans le domaine de l'analyse de données.