

Une très courte introduction aux statistiques avec R

Vincent Jalby, Université de Limoges

Janvier 2025

Table des matières

1 Introduction	1
2 Utilisation de RStudio	2
3 Packages	2
4 Bases du langage R	3
5 Importation des données	3
5.1 Cas des variables qualitatives	3
5.2 Valeurs manquantes	4
5.3 Vérification	4
5.4 Enregistrement des données	4
6 Traitements univariés	5
6.1 Variables quantitatives	5
6.2 Variables qualitatives	7
6.3 Question à réponses multiples (Multiple Response Variable)	8
6.4 Variables de Likert	9
7 Traitements bivariés	9
7.1 Variables quantitatives	9
7.2 Variables qualitatives	11
7.3 Variable quantitative / Variable qualitative	13
8 Intervalles de confiance	15
9 Tests	16
9.1 Tests sur un échantillon unique	16
9.2 Tests sur deux échantillons indépendants	17
9.3 Tests sur deux échantillons appariés	18
9.4 Tests de normalité	19
10 Analyse de la variance	20
11 Régression linéaire	21
12 Manipulation des données	22
13 Couleurs!	25
14 Probabilités et échantillons	26
15 Calcul matriciel	27

1 Introduction

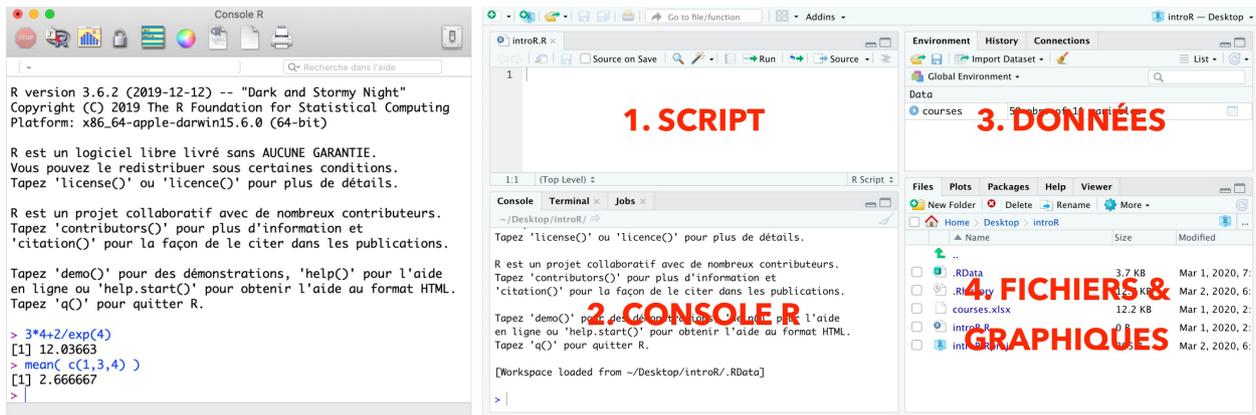
R est un logiciel libre de statistiques mais aussi un langage de programmation plus particulièrement adapté au traitement statistique des données. Il est possible de le télécharger gratuitement sur <https://cran.r-project.org>. Le logiciel **RStudio**, en complément de R, offre un environnement de travail grandement amélioré par

rapport à la console par défaut de R. Son téléchargement, gratuit, est vivement recommandé. Il est disponible sur le site <https://posit.co>.

Le langage de programmation **R** est assez sophistiqué et son temps d'apprentissage peut être relativement long. Il existe d'excellents livres et tutoriels en ligne sur ce sujet. Le but de ce document n'est pas d'en créer un nouveau, mais de vous apprendre rapidement à utiliser R pour effectuer des traitements statistiques basiques (niveau L). En complément de cette introduction, un document RStudio est disponible à l'adresse <https://vincent.jalby.org/resources/commun/introR.zip>.

2 Utilisation de RStudio

L'utilisation de **R** se fait en ligne de commande au travers d'une console : chaque instruction tapée et validée produit un résultat (écran ci-dessous à gauche). Mais cette interface n'offre que peu de possibilités de sauvegarde et de gestion des fichiers et graphiques.



Le logiciel **RStudio** facilite son utilisation, en particulier la gestion des projets et l'import des données. Pour RStudio, un projet (un document) correspond à un dossier (directory). Pour créer un projet dans **RStudio**, on utilise la commande **File > New Project...** puis **New Directory > New Project**; on précise alors le nom du projet (Directory Name) et l'endroit où on souhaite le créer (**Browse...**).

Afin de conserver un historique des instructions utilisées, il est préférable de créer un script via la commande **File > New File > R Script** (qu'il faudra ensuite enregistrer dans le dossier du projet).

La fenêtre principale de **RStudio** ressemble alors à la fenêtre ci-dessus à droite. A bas à gauche, la **console R** permet de taper des instructions et affiche les résultats de ces instructions. Attention, son contenu disparaît à la fin de la session (lorsqu'on quitte **RStudio**). Au dessus, la fenêtre du script (créé précédemment et enregistré sous le nom *intro.R*). Les instructions tapées dans cette zone sont modifiables et seront enregistrées. Pour exécuter une instruction, il faut valider la ligne au clavier en tapant <Control> + <Retour>.

Le résultat s'affiche toujours au niveau de la console (et les graphiques dans l'onglet Plots dans la zone en bas à droite).

Les résultats affichés ne sont pas enregistrés à la fin de la session. Pour les conserver, il faut les copier et les coller dans un traitement de texte en utilisant la police Courier ou Consolas pour un affichage correct.

3 Packages

La version de base de **R** contient déjà un grand nombre de fonctions statistiques, mais la puissance du logiciel réside dans de nombreuses extensions (appelées *packages*) permettant d'étendre ses fonctionnalités. Ces packages, écrits pour la plupart par des experts, doivent être téléchargés puis activés avant d'être utilisables. Certains sont installés, voire activés par défaut, par exemple les packages *utils* et *stats*.

Dans la suite, nous utiliserons à plusieurs reprises les packages *readxl*, *DescTools*, *psych* et *ggstats*. Pour les installer (les télécharger), utiliser l'instruction suivante :

```
install.packages(c('readxl', 'DescTools', 'psych', 'ggstats'))
```

Pour les *activer*, on utilise le code suivant :

```
library(readxl); library(DescTools); library(psych); library(ggstats)
options(scipen=999)
```

L'installation n'est à faire qu'une seule fois. Par contre, il faudra activer les packages à chaque session.

4 Bases du langage R

Les instructions et noms d'objet (variables, etc) sont sensibles à la casse. Donc, Mean et mean ne sont pas équivalents. Pour exécuter plusieurs instructions sur une même ligne de commande, on les sépare par des points-virgules (;). Pour affecter une valeur à une variable, on utilise l'opérateur (->) plutôt que (=).

```
123.45 -> x ; y <- 67.89
```

Un vecteur dans R est une liste d'objets identiques (numérique, caractère). L'instruction c() permet de définir un vecteur.

```
z <- c(12,3.45,-2.3) ; c('lundi','mardi','mercredi')->jours
```

Pour obtenir une liste des entiers entre deux valeurs, on utilise l'opérateur : :

```
3:12
```

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

Les valeurs booléennes vrai / faux sont notées indifféremment TRUE / FALSE ou T / F.

5 Importation des données

L'importation de fichier de données CSV, Excel, SPSS, SAS ou STATA se fait facilement à l'aide de l'item correspondant dans le menu **File > Import Dataset** de **RStudio**.

num (double)	age (double)	sexe (character)	sitfam (double)	frequence (double)	supermarche (double)	hypermarche (double)	proximite (double)	specialis (double)
1	25	H	1	1	1	0	1	0
2	29	F	1	2	1	1	0	1
3	35	F	3	3	0	1	0	0
4	19	F	2	2	1	0	0	1
5	23	H	5	3	NA	NA	NA	NA

Pour CSV et Excel, il faut que le fichier à importer soit composé uniquement des données, les variables en colonnes (les observations en lignes). La première ligne doit contenir les noms des variables (privilégier des noms très courts sans accent, sans espace). Le nom indiqué dans le champ Name (basé sur le nom du fichier importé) sera le nom du dataframe contenant les données.

Il est aussi possible d'importer le fichier à l'aide du code suivant (utilisant le package *readxl*) :

```
courses <- read_excel("courses.xlsx") # Placer courses.xlsx dans le dossier du projet
```

Après importation, on accèdera aux variables avec la syntaxe `nom_data_frame$nom_variable`.

5.1 Cas des variables qualitatives

Si l'importation de variables quantitatives (numériques) ne nécessite aucune précaution, il n'en est pas de même des variables qualitatives : Si la variable est alphanumérique (par exemple la variable sexe, codée 'H' et 'F'), il suffit de la redéfinir comme factor (=variable qualitative) :

```
courses$sexe <- factor(courses$sexe)
```

Si la variable a été codée numériquement (par exemple la variable lieu, codée de 1 à 4), il faut la redéfinir, en précisant les valeurs (levels) et les étiquettes (labels) associées.

```
courses$lieu <- factor(courses$lieu, level=c(1,2,3,4),
                      labels=c('Hyper', 'Super', 'Proximité', 'Spécialisé'))
```

Pour les variables qualitatives ordinales, il est préférable d'utiliser la fonction `ordered` au lieu de `factor`.

```
courses$frequence <- ordered(courses$frequence, level=c(1,2,3,4),
                             labels=c('Mensuel', 'Bimensuel', 'Hebdomadaire', 'Plus souvent'))
```

Attention! Lors des transformations ci-dessus, la variable originale est *écrasée* par sa redéfinition. En cas d'erreur, il faudra réimporter les données pour récupérer la variable d'origine.

5.2 Valeurs manquantes

Lors de l'import des données, les *cases vides* sont considérées comme des valeurs manquantes et remplacées par NA (Not Available).

Pour les variables qualitatives, dont les valeurs manquantes seraient codées par une valeur spécifique (par exemple 9), il suffit de ne pas indiquer cette valeur dans la liste des `levels` lors de l'utilisation de la fonction `factor`. Elle sera alors automatiquement considérée comme valeur manquante.

5.3 Vérification

L'instruction `str()` permet de vérifier rapidement si l'importation s'est correctement effectuée.

```
str(courses)

## tibble [50 x 15] (S3: tbl_df/tbl/data.frame)
## $ num      : num [1:50] 1 2 3 4 5 6 7 8 9 10 ...
## $ age      : num [1:50] 25 29 35 19 23 45 33 22 21 38 ...
## $ sexe     : Factor w/ 2 levels "F","H": 2 1 1 1 2 NA 2 2 1 NA ...
## $ sitfam   : num [1:50] 1 1 3 2 5 4 3 9 3 2 ...
## $ frequence : Ord.factor w/ 4 levels "Mensuel"<"Bimensuel"<...: 1 2 3 2 2 3 3 2 1 1 ...
## $ lieu     : Factor w/ 4 levels "Hyper","Super",...: 1 2 1 2 4 1 3 1 4 1 ...
## $ supermarche: num [1:50] 1 1 0 1 NA 0 0 0 NA 0 ...
## $ hypermarche: num [1:50] 1 1 1 0 NA 1 0 1 NA 1 ...
## $ proximite : num [1:50] 1 0 0 0 NA 0 1 0 NA 1 ...
## $ specialise : num [1:50] 0 1 0 1 NA 0 1 0 NA 0 ...
## $ revenu    : num [1:50] 2560 1330 1850 NA 1200 3200 2210 2500 1340 2100 ...
## $ budget    : num [1:50] 430 146 260 120 168 520 223 300 174 230 ...
## $ LKprix    : num [1:50] 1 2 3 4 5 1 3 1 5 2 ...
## $ LKproximite: num [1:50] 1 5 1 5 4 2 3 1 5 1 ...
## $ LKvariete : num [1:50] 3 4 5 4 5 4 3 1 2 3 ...
```

La fonction `head()` permet d'afficher les premières observations.

```
head(courses)

## # A tibble: 6 x 15
##   num age sexe sitfam frequence lieu supermarche hypermarche proximite
##   <dbl> <dbl> <fct> <dbl> <ord> <fct> <dbl> <dbl> <dbl>
## 1 1 25 H 1 Mensuel Hyper 1 1 1
## 2 2 29 F 1 Bimensuel Super 1 1 0
## 3 3 35 F 3 Hebdomadaire Hyper 0 1 0
## 4 4 19 F 2 Bimensuel Super 1 0 0
## 5 5 23 H 5 Bimensuel Speci~ NA NA NA
## 6 6 45 <NA> 4 Hebdomadaire Hyper 0 1 0
## # i 6 more variables: specialise <dbl>, revenu <dbl>, budget <dbl>,
## # LKprix <dbl>, LKproximite <dbl>, LKvariete <dbl>
```

5.4 Enregistrement des données

A la fin de la session (lorsqu'on quitte l'application), les données (apparaissant dans l'onglet Environnement > data), c'est-à-dire, tout ce qui a été enregistré avec un nom avec l'instruction `->`, sont enregistrées dans le fichier `.RData`. Elles seront à nouveau disponibles lorsqu'on réouvrira le projet.

Il est aussi possible d'enregistrer un dataframe séparément dans un fichier pour pouvoir l'utiliser par ailleurs (dans un autre projet par exemple). Pour cela, on utilise la commande `save()` :

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

Pour charger ces données (dans un autre projet), on utilise la commande `load()` :

```
load("courses.RData")
```

Il faut bien-sûr placer le fichier dans le dossier du projet.

6 Traitements univariés

6.1 Variables quantitatives

Les fonctions `mean()`, `sd()`, `var()`, `min()`, `max()`, `median()`... permettent de calculer les principaux résumés statistiques d'une variable quantitative.

```
mean(courses$age);median(courses$age)
```

```
## [1] 31.98
## [1] 32.5
```

Lorsque la variable contient des valeurs manquantes (*NA*), il faut rajouter l'option `na.rm=TRUE`.

```
mean(courses$revenu);mean(courses$revenu, na.rm=TRUE)
```

```
## [1] NA
## [1] 1934.468
```

La fonction `summary()` permet aussi d'obtenir plus rapidement quelques résumés statistiques.

```
summary(courses$revenu)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##      1100   1475   1900   1934   2250   3200     3
```

Mais la fonction `describe()` du package *psych* est plus complète. L'option `fast=TRUE` n'affiche que les principaux résumés.

```
describe(courses$revenu, fast=TRUE)
```

```
##      vars n      mean      sd median min max range skew kurtosis  se
## X1     1 47 1934.47 540.28  1900 1100 3200 2100 0.39  -0.53 78.81
```

Pour l'appliquer à plusieurs variables simultanément, il faut les regrouper dans un dataframe ou utiliser la notation *tableau*.

```
describe( data.frame(courses$age, courses$revenu, courses$budget), fast=TRUE, skew=FALSE )
```

```
##              vars n      mean      sd median min max range  se
## courses.age      1 50  31.98   9.19  32.5  19  48   29 1.30
## courses.revenu   2 47 1934.47 540.28 1900.0 1100 3200 2100 78.81
## courses.budget   3 49  247.80  93.72  230.0  120  550  430 13.39
```

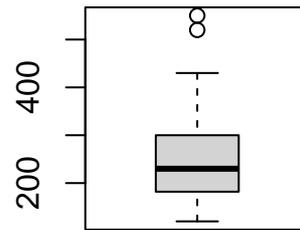
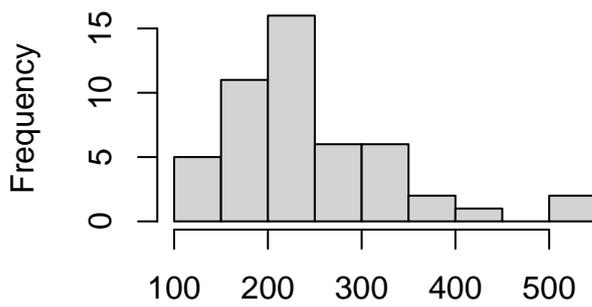
```
describe( courses[, c('age', 'revenu', 'budget')], fast=TRUE, skew=FALSE )
```

```
##              vars n      mean      sd median min max range  se
## age            1 50  31.98   9.19  32.5  19  48   29 1.30
## revenu         2 47 1934.47 540.28 1900.0 1100 3200 2100 78.81
## budget         3 49  247.80  93.72  230.0  120  550  430 13.39
```

Les fonctions `hist()` et `boxplot()` permettent obtenir un histogramme et une boîte à moustaches respectivement.

```
hist(courses$budget);boxplot(courses$budget)
```

Histogram of courses\$budget



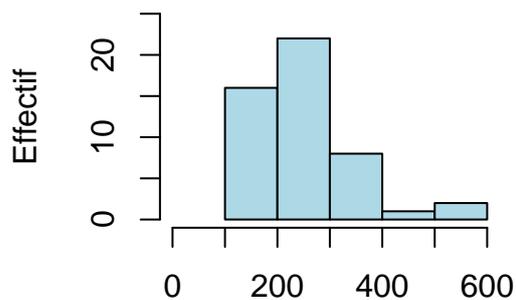
`courses$budget`

Diverses options permettent d'améliorer l'apparence des graphiques.

```
hist(courses$budget, col='lightblue', xlim=c(0,600), ylim=c(0,25),
      xlab='Budget', ylab='Effectif', main='Histogramme', breaks=6)
boxplot(courses$budget, col='orange', xlab='Budget', horizontal=TRUE,
```

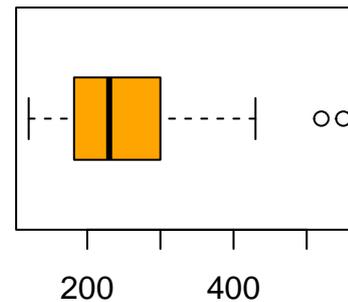
`main=`

Histogramme



`Budget`

Boite à moustaches

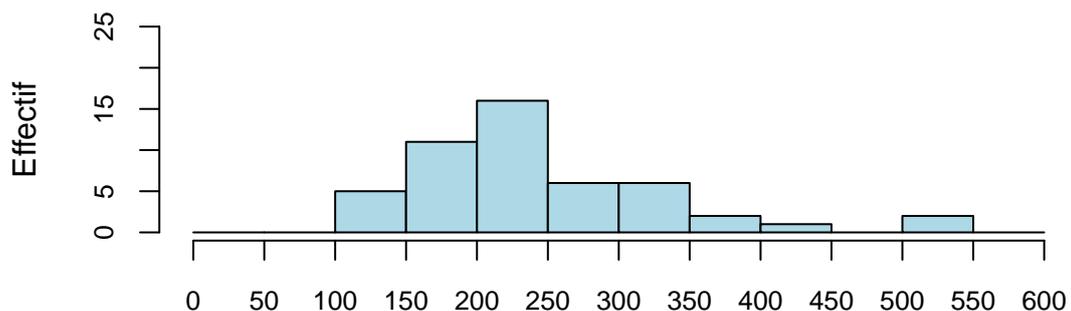


`Budget`

Pour obtenir des classes de largeurs prédéfinies (et égales), on utilise l'option `breaks`, en indiquant les bornes des classes, via l'instruction `seq(a,b,c)` générant la liste $a, a + c, a + 2c, \dots, b$. L'instruction `axis` permet de personnaliser l'axe horizontal (1) en ajoutant des marques (nombres) tous les 50.

```
hist(courses$budget, col='lightblue', xlim=c(0,600), ylim=c(0,25),
      xlab='Budget', ylab='Effectif',main='Histogramme',
      breaks=seq(0,600,50), cex.axis=0.8, axes=FALSE) # sans les axes
axis(1, at=seq(0,600, 50), cex.axis=0.8) # Axe horizontal
axis(2, cex.axis=0.8) # Axe vertical
```

Histogramme



`Budget`

6.2 Variables qualitatives

La fonction `table()` permet d'obtenir les effectifs des modalités. En rajoutant `prop.table()`, on obtient les fréquences (pourcentages).

```
table(courses$lieu)

##
##      Hyper      Super Proximité Spécialisé
##       20       21       5         2

prop.table(table(courses$lieu))

##
##      Hyper      Super Proximité Spécialisé
## 0.41666667 0.43750000 0.10416667 0.04166667
```

Mais la fonction `Freq()` du package *DescTools* est bien plus complète.

```
Freq(courses$lieu)

##      level freq perc cumfreq cumperc
## 1      Hyper  20 41.7%    20   41.7%
## 2      Super  21 43.8%    41   85.4%
## 3 Proximité   5 10.4%    46   95.8%
## 4 Spécialisé  2  4.2%    48  100.0%
```

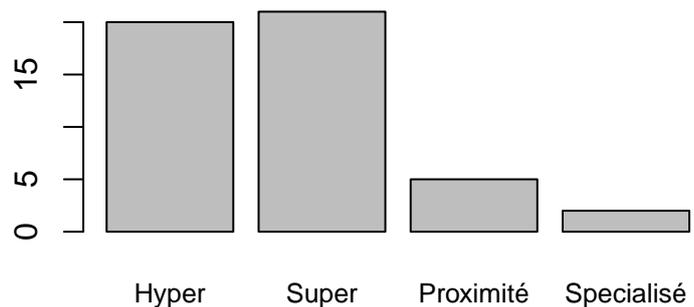
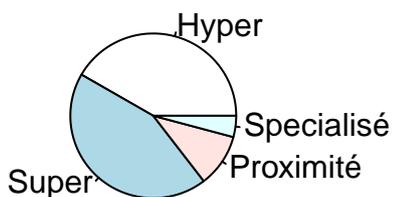
La variable `lieu` étant nominale, il est préférable de trier le tableau par effectifs décroissants.

```
Freq(courses$lieu, ord='desc')

##      level freq perc cumfreq cumperc
## 1      Super  21 43.8%    21   43.8%
## 2      Hyper  20 41.7%    41   85.4%
## 3 Proximité   5 10.4%    46   95.8%
## 4 Spécialisé  2  4.2%    48  100.0%
```

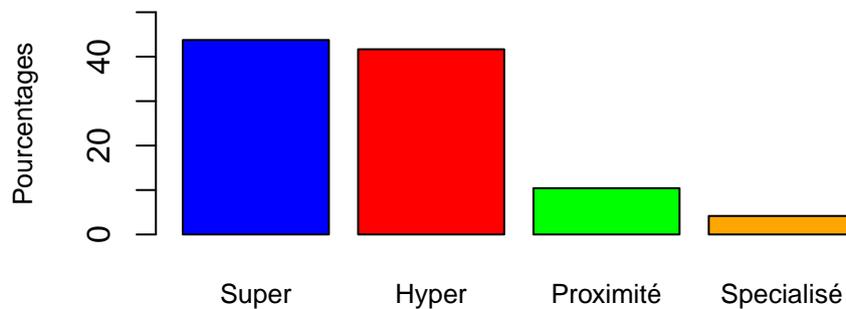
Les fonctions `pie()` et `barplot()`, couplées à la fonction `table()`, permettent d'obtenir un diagramme en secteurs ou en bâtons.

```
pie(table(courses$lieu))
barplot(table(courses$lieu))
```



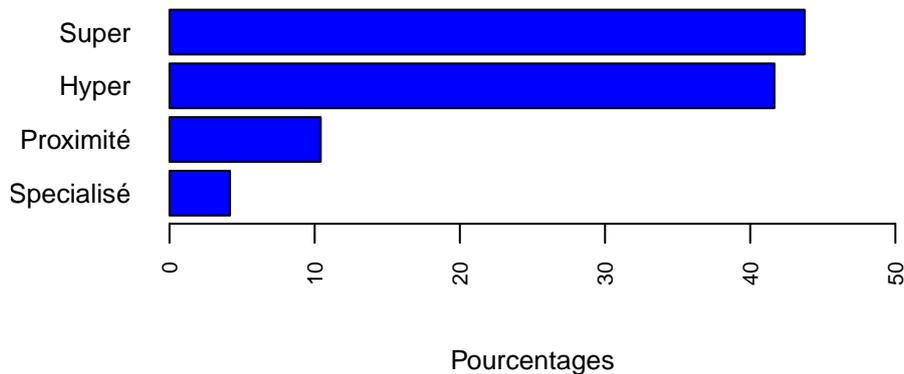
Pour afficher les fréquences (pourcentages) à la place des effectifs, il faut utiliser en plus la fonction `prop.table()` et la fonction `sort()` si on souhaite trier les bâtons par ordre décroissant.

```
barplot(
  sort(prop.table(table(courses$lieu))*100, decreasing=TRUE),
  ylab='Pourcentages', ylim=c(0,50), col=c('blue','red','green','orange'),
  cex.lab=0.8, cex.names=0.8
)
```



Finalement, on obtient un graphique horizontal avec l'option `horiz=TRUE` (et `las=2` pour avoir des intitulés horizontaux).

```
barplot(
  sort( prop.table( table(courses$lieu) ) * 100, decreasing=FALSE),
  xlab='Pourcentages', xlim=c(0,50),
  col='blue', cex.names=0.8, cex.lab=0.8, cex.axis=0.7, horiz=TRUE, las=2
)
```



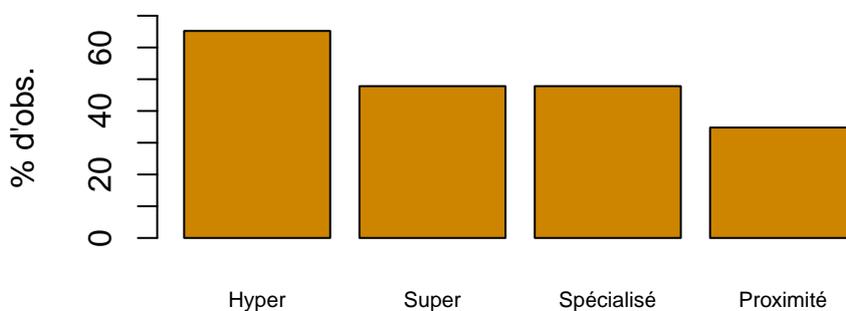
6.3 Question à réponses multiples (Multiples Response Variable)

Les questions à réponses multiples (cases à cocher) correspondent à autant de variables dichotomiques (1 / 0, coché / pas coché) que de réponses possibles.

R ne possède pas de fonction (ni de package) permettant d'étudier simplement les variables dichotomiques issues de ces questions. La fonction `multipleResponse` (qui se trouve dans le script `MultipleResponse.R`, chargé par l'instruction `source()`) permet de faire cette étude, et représentation graphique, facilement.

```
source('MultipleResponse.R')
multipleResponse(courses, c('supermarche', 'hypermarche', 'proximite', 'specialise'),
  order='decreasing', plot='percentage', ylim=c(0,70), main="Lieux d'achats", ylab="% d'obs.",
  names=c('Super', 'Hyper', 'Proximité', 'Spécialisé'), col='orange3', cex.name=0.7)
```

Lieux d'achats



```
##
## Multiple Response Variable (Number of cases: 46)
##      Option Frequency Perc. of responses Perc. of cases
## 2      Hyper          30          33.33333          65.21739
## 1      Super          22          24.44444          47.82609
```

## 4 Spécialisé	22	24.44444	47.82609
## 3 Proximité	16	17.77778	34.78261
## 5 Total	90	100.00000	195.65217

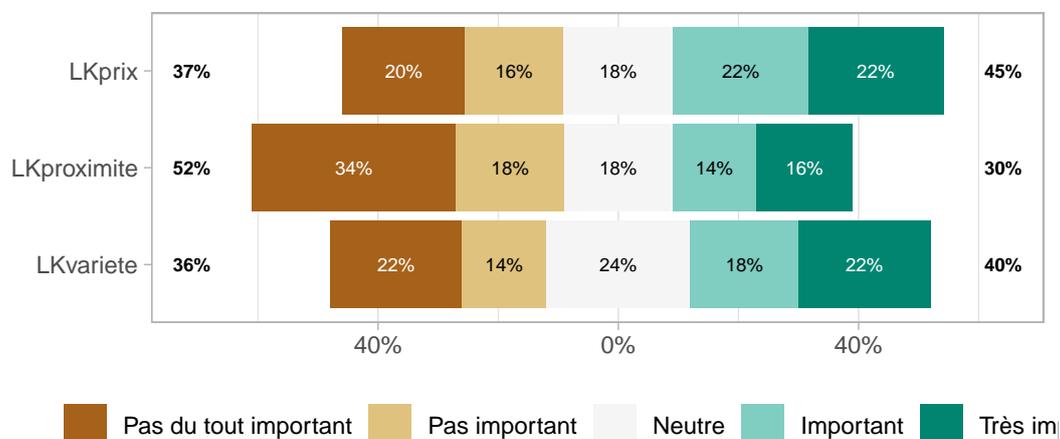
6.4 Variables de Likert

Le package ggstats permet de représenter très facilement des variables utilisant des échelles de Likert. Tout d'abord, il est préférable de transformer ces variables en facteurs.

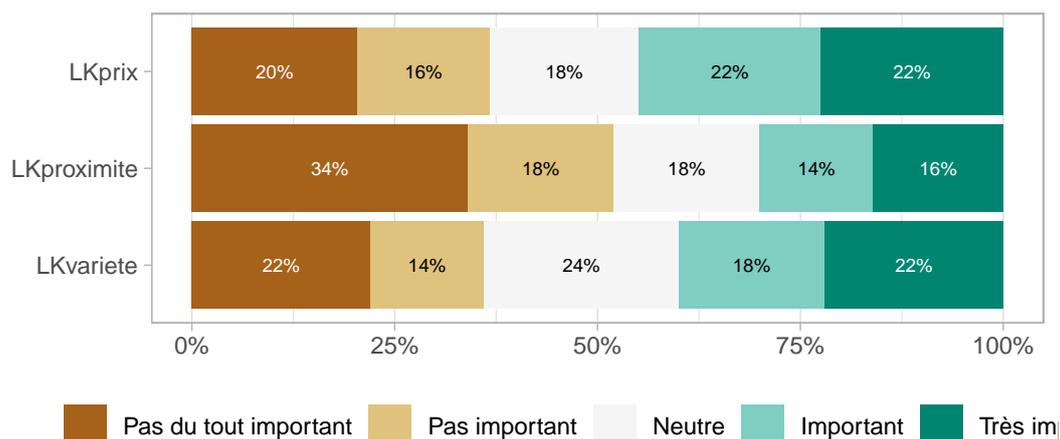
```
courses$LKprix <- ordered(courses$LKprix, level=c(1,2,3,4,5),
  labels=c('Pas du tout important', 'Pas important', 'Neutre', 'Important', 'Très important'))
courses$LKproximite <- ordered(courses$LKproximite, level=c(1,2,3,4,5),
  labels=c('Pas du tout important', 'Pas important', 'Neutre', 'Important', 'Très important'))
courses$LKvariete <- ordered(courses$LKvariete, level=c(1,2,3,4,5),
  labels=c('Pas du tout important', 'Pas important', 'Neutre', 'Important', 'Très important'))
```

On utilise alors une des deux fonctions `gglkert()` ou `gglkert_stacked()`.

```
gglkert(courses, include = c('LKprix', 'LKproximite', 'LKvariete'), labels_size=2.5)
```



```
gglkert_stacked(courses, include = c('LKprix', 'LKproximite', 'LKvariete'), labels_size=2.5)
```



Le package ggstats utilise `ggplot2`. Son utilisation est très différente des graphiques **R**. Se référer à ma *très courte introduction à GGPlot2* pour plus de détails.

7 Traitements bivariés

7.1 Variables quantitatives

La covariance et la corrélation entre deux variables s'obtiennent avec les fonctions `cov()` et `cor()`. En cas de valeurs manquantes, il est nécessaire de préciser l'option `use='complete.obs'` ou `use='pairwise.complete.obs'`.

```
cov(courses$revenu,courses$budget,use="complete.obs")
```

```
## [1] 43654.95
```

```
cor(courses$revenu,courses$budget,use="complete.obs")
```

```
## [1] 0.8621749
```

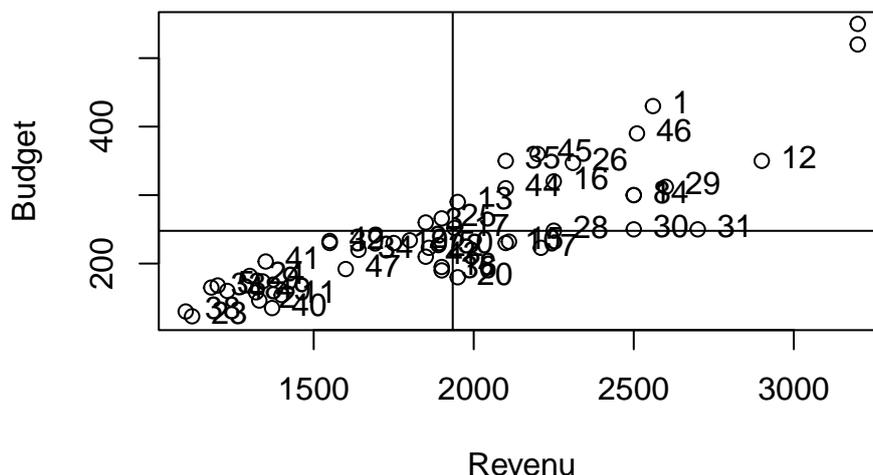
On obtient de même la matrice des corrélations entre deux (ou plus) variables.

```
cor(courses[,c('age','revenu','budget')],use="pairwise.complete.obs")
```

```
##           age    revenu    budget
## age      1.0000000 0.2796028 0.2824027
## revenu  0.2796028 1.0000000 0.8621749
## budget  0.2824027 0.8621749 1.0000000
```

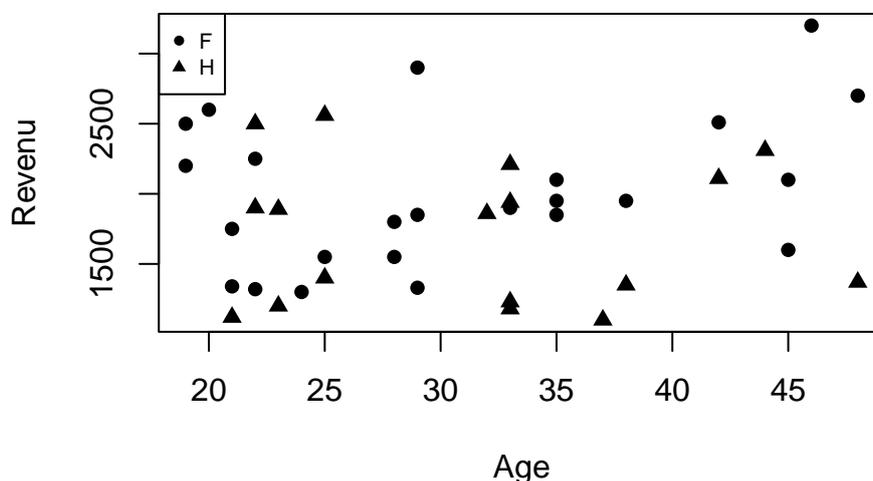
La représentation graphique du nuage de points se fait à l'aide de la fonction `plot()`. La fonction `text()` permet de rajouter au graphique précédent des étiquettes à chaque point. La fonction `abline()` permet de rajouter des lignes de référence horizontales ou verticales

```
plot(courses$revenu,courses$budget, xlab='Revenu',ylab='Budget')
text(courses$revenu,courses$budget,courses$num,pos=4)
abline(v=mean(courses$revenu, na.rm=TRUE) )
abline(h=mean(courses$budget, na.rm=TRUE) )
```



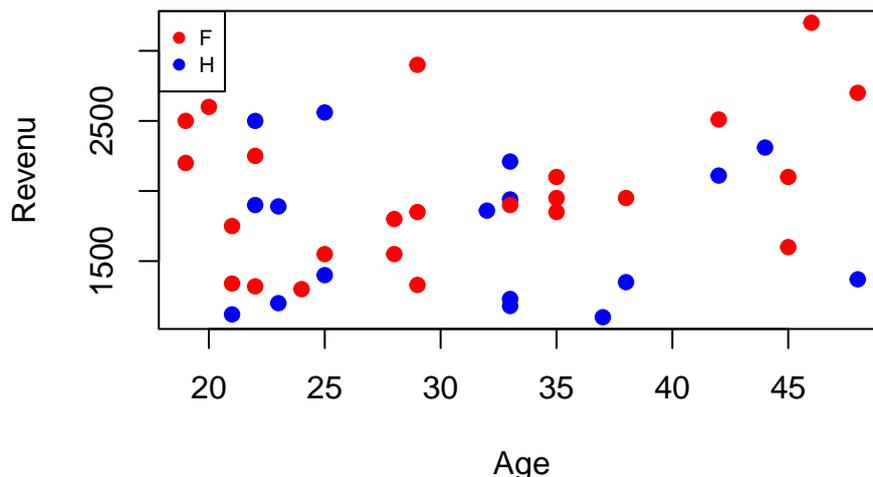
Le symbole utilisé pour représenter les points est modifiable avec les options `pch=0` à `pch=25`. Il est même possible d'utiliser un symbole différent selon les valeurs d'une variable qualitative (factor). La fonction `legend()` permet alors de préciser la légende.

```
plot(courses$age,courses$revenu,xlab='Age',ylab='Revenu', pch = c(16,17)[courses$sexe])
legend('topleft', legend = levels(courses$sexe), pch = c(16,17), cex=0.7)
```



Suivant le même principe, on peut aussi utiliser des couleurs différentes.

```
plot(courses$age,courses$revenu,xlab='Age',ylab='Revenu', pch= 19,
      col = c('red','blue')[courses$sexe] )
legend('topleft', legend = levels(courses$sexe), cex=0.7, pch=19, col=c('red','blue') )
```



7.2 Variables qualitatives

Les fonctions `table()` et `prop.table()` permettent d'obtenir le tri croisé (ou tableau de contingence) entre deux variables qualitatives.

```
table(courses$lieu,courses$frequence)
```

```
##
##           Mensuel Bimensuel Hebdomadaire Plus souvent
## Hyper           7           6             5             2
## Super           5           6             7             3
## Proximité       0           0             2             3
## Spécialisé      1           1             0             0
```

```
round( prop.table(table(courses$lieu,courses$frequence))*100, 2 )
```

```
##
##           Mensuel Bimensuel Hebdomadaire Plus souvent
## Hyper      14.58      12.50       10.42         4.17
## Super      10.42      12.50       14.58         6.25
## Proximité   0.00       0.00         4.17         6.25
## Spécialisé  2.08       2.08         0.00         0.00
```

Pour calculer les pourcentages en ligne ou colonne, il faut rajouter l'option `margin=1` ou `margin=2` respectivement.

```
round( prop.table(table(courses$lieu,courses$frequence), margin=1)*100, 2 )
```

```
##
##           Mensuel Bimensuel Hebdomadaire Plus souvent
## Hyper      35.00      30.00       25.00        10.00
## Super      23.81      28.57       33.33        14.29
## Proximité   0.00       0.00        40.00        60.00
## Spécialisé  50.00      50.00         0.00         0.00
```

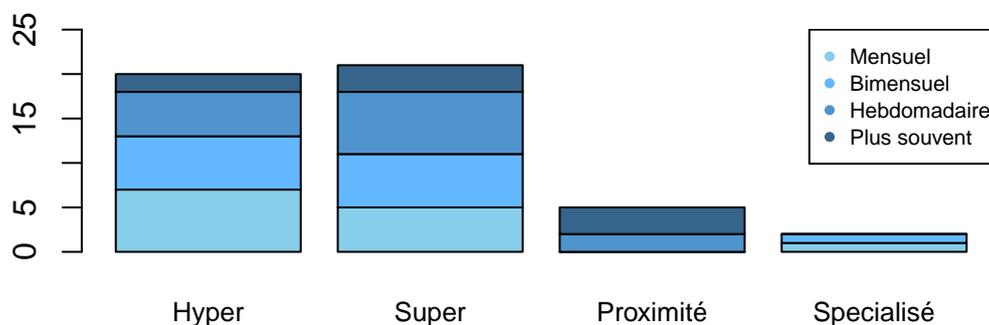
La fonction `PercTable()` du package *DescTools* offre plus d'options et permet d'obtenir facilement les marges de la table. L'option `freq=F` supprime l'affichage des effectifs (pour n'afficher que les pourcentages). L'option `rfrq="010"` (resp. `rfrq="001"`) affiche les pourcentages en ligne (resp. en colonne). `margins=2` (resp. `margins=1`) affiche la ligne (resp. colonne) somme.

```
PercTable(courses$lieu,courses$frequence, freq=F, rfrq="010", margins=2, digits=2)
```

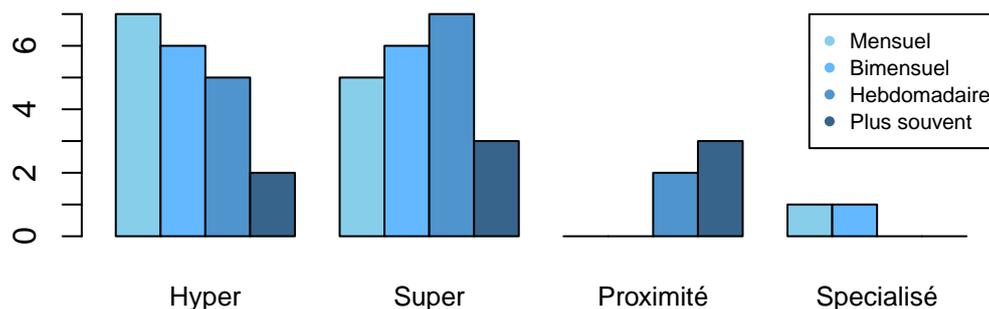
```
##
##           Mensuel  Bimensuel  Hebdomadaire  Plus souvent
##
## Hyper           35.00%    30.00%           25.00%           10.00%
## Super           23.81%    28.57%           33.33%           14.29%
## Proximité       0.00%     0.00%           40.00%           60.00%
## Spécialisé      50.00%    50.00%           0.00%            0.00%
## Sum             27.08%    27.08%           29.17%           16.67%
```

Les graphiques sont obtenus avec la fonction `barplot()` combinée avec `table()`.

```
barplot(
  table(courses$frequence,courses$lieu),
  col=c('skyblue','steelblue1','steelblue3','steelblue4'),
  ylim=c(0,25), cex.names=0.8
)
legend("topright",legend = levels(courses$frequence),
  col=c('skyblue','steelblue1','steelblue3','steelblue4'), pch = 16, cex=0.7)
```

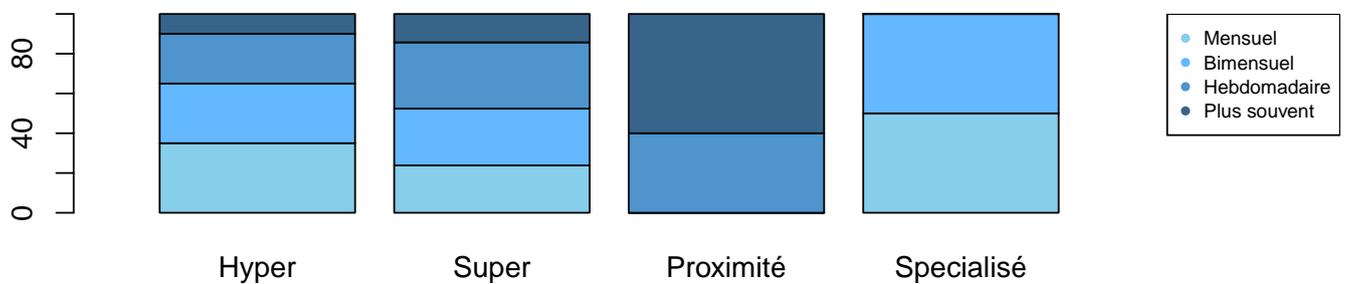


```
barplot(
  table(courses$frequence,courses$lieu),
  col=c('skyblue','steelblue1','steelblue3','steelblue4'),
  ylim=c(0,7), cex.names=0.8, beside=TRUE
)
legend("topright",legend = levels(courses$frequence),
  col=c('skyblue','steelblue1','steelblue3','steelblue4'), pch = 16, cex=0.7)
```



Finalement, un graphique empilé à 100 % s'obtient de la manière suivante :

```
barplot(
  prop.table(table(courses$frequence,courses$lieu),margin=2)*100,
  col=c('skyblue','steelblue1','steelblue3','steelblue4'),
  xlim=c(0,6))
legend("topright",legend = levels(courses$frequence),
  col=c('skyblue','steelblue1','steelblue3','steelblue4'), pch = 16, cex=0.7)
```



Les mesures (nominales) du lien entre les deux variables s'obtiennent à l'aide des fonctions de *DescTools* :

```
Phi(courses$lieu,courses$frequence)
```

```
## [1] 0.4939191
```

```
CramerV(courses$lieu,courses$frequence)
```

```
## [1] 0.2851643
```

Le test du Khi-Deux s'obtient aussi facilement.

```
chisq.test(courses$lieu,courses$frequence)
```

```
##
## Pearson's Chi-squared test
##
## data:  courses$lieu and courses$frequence
## X-squared = 11.71, df = 9, p-value = 0.2302
```

7.3 Variable quantitative / Variable qualitative

La fonction `tapply()` permet de calculer les résumés statistiques d'une variable quantitative pour chaque groupe défini par une variable qualitative.

```
tapply(courses$budget, courses$lieu, mean, na.rm=TRUE)
```

```
##      Hyper      Super  Proximité Spécialisé
##      304.6      218.8      183.6      171.0
```

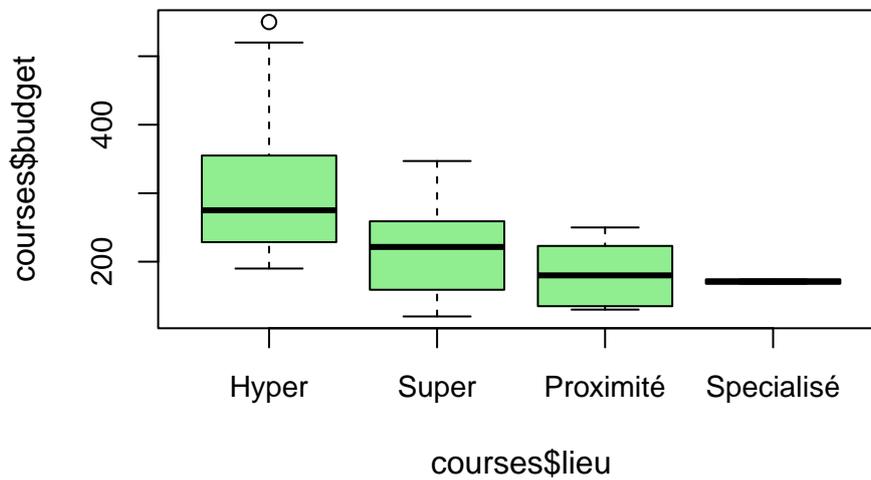
On peut aussi utiliser la fonction `describeBy()` du package *psych* :

```
describeBy(courses$budget, courses$lieu, mat=TRUE, fast=TRUE, skew=FALSE)
```

```
##      item      group1 vars  n mean      sd median min max range      se
## X11    1      Hyper    1 20 304.6 104.133214 275.0 190 550  360 23.28489
## X12    2      Super    1 20 218.8  67.295577 221.5 120 347  227 15.04775
## X13    3  Proximité    1  5 183.6  52.936755 180.0 130 250  120 23.67404
## X14    4 Spécialisé    1  2 171.0   4.242641 171.0 168 174    6  3.00000
```

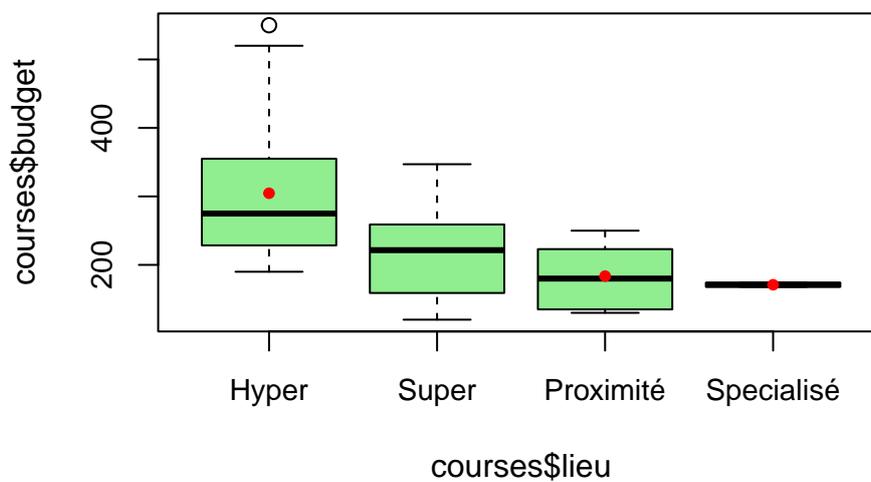
Une *formule* du type `<var_dep> ~ <var_ind>` permet d'obtenir les boîtes à moustaches pour chaque groupe.

```
boxplot(courses$budget ~ courses$lieu, col='lightgreen', cex.axis=0.9)
```



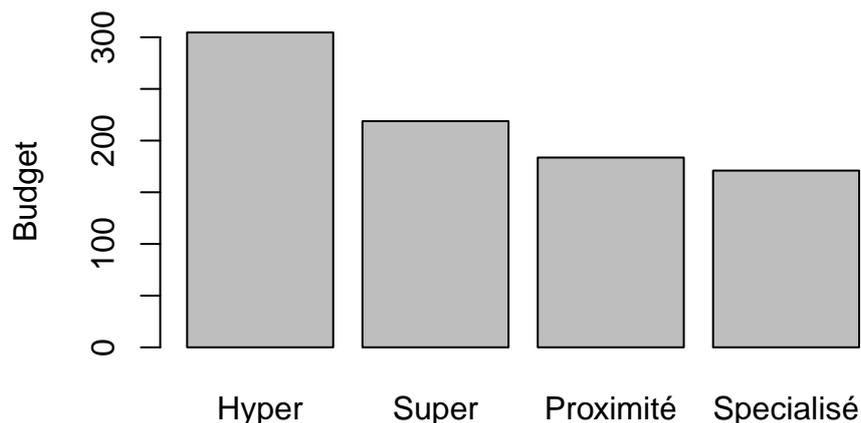
On peut aussi rajouter les moyennes par groupe au graphique précédent avec l'instruction `points()`.

```
boxplot(courses$budget ~ courses$lieu, col='lightgreen', cex.axis=0.9)
points( tapply(courses$budget, courses$lieu, mean), col="red", pch=20 )
```



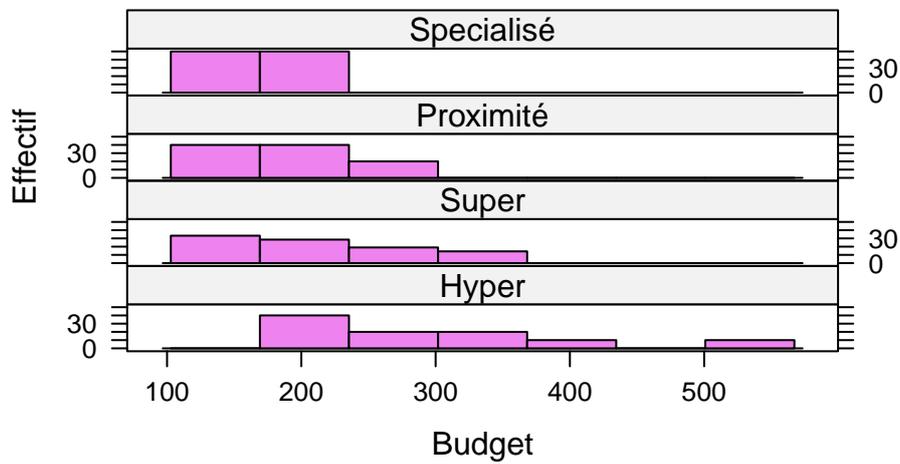
On peut aussi utiliser un graphique en barre, représentant les moyennes de chaque groupe.

```
barplot( tapply(courses$budget, courses$lieu, mean, na.rm=TRUE) , ylab="Budget")
```



Pour obtenir un histogramme pour chaque groupe, il est nécessaire d'utiliser le package `*lattice*` et d'utiliser une formule différente.

```
library(lattice)
histogram(~ courses$budget | courses$lieu, col='violet', xlab='Budget', ylab='Effectif',
          layout=c(1,5))
```



8 Intervalles de confiance

Il est possible d'obtenir les intervalles de confiance pour une moyenne (σ inconnu) en utilisant les fonctions de test de R.

```
t.test(courses$budget, conf.level = 0.90)$conf.int[1:2]
## [1] 225.3396 270.2523
```

Encore une fois, le package *DescTools* offre plus de possibilités.

```
MeanCI(courses$budget, conf.level = 0.90, na.rm=TRUE)
##      mean   lwr.ci   upr.ci
## 247.7959 225.3396 270.2523

MeanCI(courses$budget, conf.level = 0.90, sd=9, na.rm=TRUE) # sigma supposé connu, égal à 9
##      mean   lwr.ci   upr.ci
## 247.7959 245.6811 249.9107
```

De même pour l'intervalle de confiance d'une variance.

```
VarCI(courses$budget, conf.level = 0.95, na.rm=TRUE)
##      var   lwr.ci   upr.ci
## 8783.999 6108.609 13709.600
```

Pour obtenir l'intervalle de confiance d'une proportion, il est nécessaire de calculer *manuellement* le nombre de succès (k) et d'essais (n).

```
table(courses$supermarche)
##
##  0  1
## 24 22
```

On peut ensuite utiliser le test d'une proportion (Z) ou binomial (exact) selon la taille de l'échantillon.

```
prop.test(22,24+22, conf.level = 0.95)$conf.int[1:2]
## [1] 0.3313667 0.6287485

binom.test(22,24+22, conf.level = 0.95)$conf.int[1:2]
## [1] 0.3288787 0.6305435
```

Bien sûr, le package *DescTools* permet de calculer cet intervalle avec diverses méthodes.

```
BinomCI(24,24+22,conf.level=0.95, method="wilson")
```

```
##           est   lwr.ci   upr.ci  
## [1,] 0.5217391 0.3813741 0.6587531
```

```
BinomCI(24,24+22,conf.level=0.95, method="wald")
```

```
##           est   lwr.ci   upr.ci  
## [1,] 0.5217391 0.3773853 0.666093
```

9 Tests

L'ensemble des tests classiques sont disponibles dans R de base ou dans le package *DescTools*. La syntaxe et les sorties sont très similaires. L'hypothèse alternative est précisée avec les options `alternative='two.sided'` (test bilatéral), `alternative='less'` (test unilatéral à gauche) ou `alternative='greater'` (test unilatéral à droite).

9.1 Tests sur un échantillon unique

Les tests sur une moyenne (Test Z et test T) et sur une variance (test du Khi-Deux) s'obtiennent via les fonctions `ZTest()`, `t.test()` et `VarTest()`.

```
ZTest(courses$budget, mu=200, sd_pop=90, alternative = 'two.sided')
```

```
##  
## One Sample z-test  
##  
## data:  courses$budget  
## z = 3.7175, Std. Dev. Population = 90, p-value = 0.0002012  
## alternative hypothesis: true mean is not equal to 200  
## 95 percent confidence interval:  
## 222.5964 272.9955  
## sample estimates:  
## mean of x  
## 247.7959
```

```
t.test(courses$budget, mu=200, alternative = 'two.sided')
```

```
##  
## One Sample t-test  
##  
## data:  courses$budget  
## t = 3.5698, df = 48, p-value = 0.0008237  
## alternative hypothesis: true mean is not equal to 200  
## 95 percent confidence interval:  
## 220.8755 274.7163  
## sample estimates:  
## mean of x  
## 247.7959
```

```
VarTest(courses$budget, sigma.squared=90^2, alternative = 'greater')
```

```
##  
## One Sample Chi-Square test on variance  
##  
## data:  courses$budget  
## X-squared = 52.053, df = 48, p-value = 0.3191  
## alternative hypothesis: true variance is greater than 8100  
## 95 percent confidence interval:  
## 6469.648      Inf  
## sample estimates:
```

```
## variance of x
##      8783.999
```

Le test sur une proportion a deux variantes : exact (binomial) et approché (Test Z) – le premier étant à privilégier lorsque l'échantillon est petit. Comme pour l'intervalle de confiance (voir plus haut), il faut calculer séparément, le nombre de succès (k) et d'essais (n). Pour le test approché, R fait un test sur la statistiques Z^2 qui suit une loi $\chi^2(1)$.

```
prop.test(22,24+22, p=0.45, alternative = 'greater')
```

```
##
## 1-sample proportions test with continuity correction
##
## data: 22 out of 24 + 22, null probability 0.45
## X-squared = 0.056214, df = 1, p-value = 0.4063
## alternative hypothesis: true p is greater than 0.45
## 95 percent confidence interval:
##  0.3515955 1.0000000
## sample estimates:
##          p
## 0.4782609
```

```
binom.test(22,24+22, p=0.45, alternative = 'greater')
```

```
##
## Exact binomial test
##
## data: 22 and 24 + 22
## number of successes = 22, number of trials = 46, p-value = 0.4046
## alternative hypothesis: true probability of success is greater than 0.45
## 95 percent confidence interval:
##  0.3502462 1.0000000
## sample estimates:
## probability of success
##          0.4782609
```

9.2 Tests sur deux échantillons indépendants

Le test F de comparaison des variances s'obtient avec la fonction `var.test()`. L'hypothèse nulle est que le rapport des deux variances est égal à 1 (et donc l'égalité des variances!)

```
var.test(courses$revenu ~ courses$sexe, alternative='two.sided')
```

```
##
## F test to compare two variances
##
## data: courses$revenu by courses$sexe
## F = 1.0705, num df = 23, denom df = 16, p-value = 0.9061
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.4058792 2.6115561
## sample estimates:
## ratio of variances
##          1.070466
```

Si la normalité des échantillons n'est pas vérifiée (voir plus loin), il est préférable d'utiliser le test de Levene (*DescTools*).

```
LeveneTest(courses$revenu ~ courses$sexe, alternative='two.sided', center=mean)
```

```
## Levene's Test for Homogeneity of Variance (center = mean: "two.sided")
##      Df F value Pr(>F)
## group 1  0.1212 0.7296
##      39
```

Le test T de comparaison des moyennes s'obtient avec la fonction `t.test()` (déjà utilisée pour le test sur un échantillon). L'option `var.equal` permet de préciser si les variances sont égales (test T) ou différentes (test de Welch-Satterthwaite).

```
t.test(courses$revenu ~ courses$sexe, alternative='two.sided', var.equal=TRUE)

##
## Two Sample t-test
##
## data: courses$revenu by courses$sexe
## t = 1.7405, df = 39, p-value = 0.08966
## alternative hypothesis: true difference in means between group F and group H is not equal to 0
## 95 percent confidence interval:
## -46.17424 615.68405
## sample estimates:
## mean in group F mean in group H
##      2004.167      1719.412
```

Le test (Z) de comparaison de deux proportions utilise encore la fonction `prop.test()`. Comme pour le test pour un échantillon, il faut calculer le nombre de succès et d'essais pour chaque groupe et les indiquer dans des vecteurs.

```
table(courses$proximite,courses$sexe)

##
##      F H
## 0 22 5
## 1 2 10
```

```
prop.test(c(2,10),c(2+22,10+5))

##
## 2-sample test for equality of proportions with continuity correction
##
## data: c(2, 10) out of c(2 + 22, 10 + 5)
## X-squared = 12.134, df = 1, p-value = 0.0004951
## alternative hypothesis: two.sided
## 95 percent confidence interval:
## -0.9004399 -0.2662268
## sample estimates:
##      prop 1      prop 2
## 0.08333333 0.66666667
```

9.3 Tests sur deux échantillons appariés

On utilise encore la fonction `t.test()` en précisant l'option `paired=TRUE`. (L'exemple suivant compare le revenu moyen avec le budget moyen!)

```
t.test(courses$revenu, courses$budget, paired=TRUE, alternative='two.sided')

##
## Paired t-test
##
## data: courses$revenu and courses$budget
## t = 24.983, df = 46, p-value < 0.000000000000000022
## alternative hypothesis: true mean difference is not equal to 0
## 95 percent confidence interval:
## 1547.735 1818.989
## sample estimates:
## mean difference
##      1683.362
```

9.4 Tests de normalité

On peut tester la normalité d'un échantillon à l'aide du test de Shapiro-Wilk via la fonction `shapiro.test`.

```
shapiro.test(courses$budget)

##
## Shapiro-Wilk normality test
##
## data:  courses$budget
## W = 0.90237, p-value = 0.0006573
```

Le package *DescTools* propose d'autres tests classiques de normalité.

```
JarqueBeraTest(courses$budget, na.rm=TRUE)

##
## Robust Jarque Bera Test
##
## data:  structure(c(430, 146, 260, 120, 168, 520, 223, 300, 174, 230, 154, 350, 290, 300, 232, 320,
## X-squared = 39.026, df = 2, p-value = 0.000000003355

LillieTest(courses$budget)

##
## Lilliefors (Kolmogorov-Smirnov) normality test
##
## data:  courses$budget
## D = 0.15558, p-value = 0.00459

CramerVonMisesTest(courses$budget)

##
## Cramer-von Mises normality test
##
## data:  courses$budget
## W = 0.20524, p-value = 0.004353

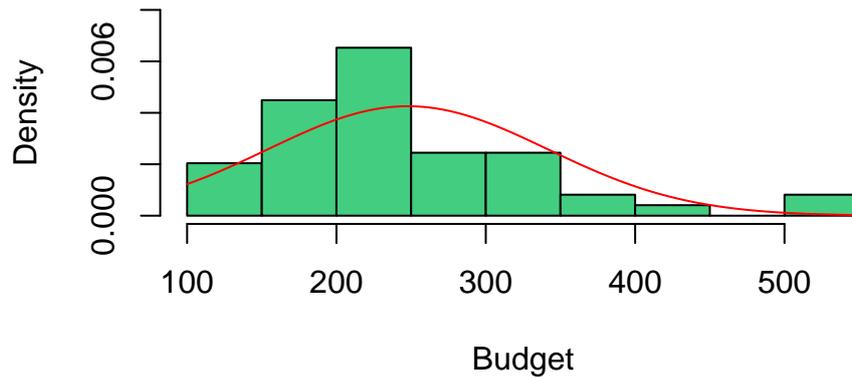
ShapiroFranciaTest(courses$budget)

##
## Shapiro-Francia normality test
##
## data:  courses$budget
## W = 0.90078, p-value = 0.001065
```

Il est aussi possible de tester *graphiquement* la normalité d'un échantillon via un histogramme ou un diagramme quantile-quantile.

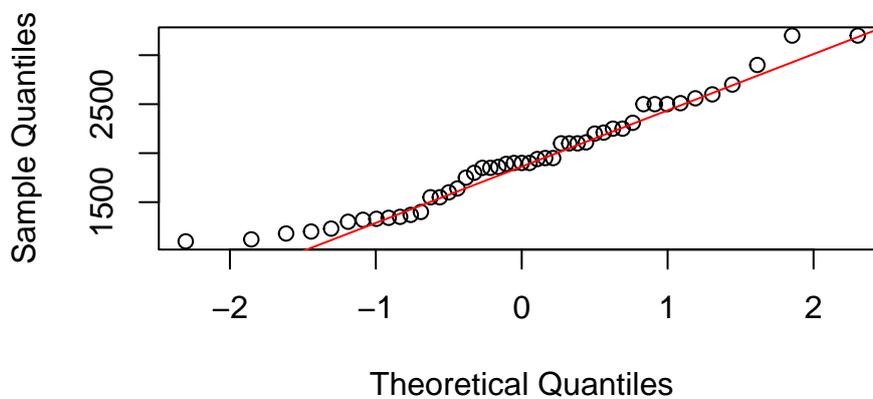
```
hist(courses$budget, freq = F, ylim = c(0, 0.008), col='seagreen3', xlab='Budget',
      main="Histogramme de budget")
curve(dnorm(x, mean(courses$budget,na.rm=TRUE), sd(courses$budget,na.rm=TRUE)),
      add=TRUE, col='red')
```

Histogramme de budget



```
qqnorm(courses$revenu)
qqline(courses$revenu,col='red')
```

Normal Q-Q Plot



10 Analyse de la variance

On obtient une analyse de la variance à un facteur (comparaison des moyennes d'une variable quantitative selon les modalités d'une variable qualitative) avec la fonction `avov`.

```
aovbudget <- aov(courses$budget ~ courses$lieu)
summary(aovbudget)

##           Df Sum Sq Mean Sq F value Pr(>F)
## courses$lieu 3 113609  37870  5.369 0.00314 **
## Residuals   43 303303   7054
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 3 observations deleted due to missingness
```

Il est alors possible d'effectuer des tests de comparaison « Post-Hoc » à l'aide de la fonction `TukeyHSD`.

```
TukeyHSD(aovbudget)

## Tukey multiple comparisons of means
## 95% family-wise confidence level
##
## Fit: aov(formula = courses$budget ~ courses$lieu)
##
## $`courses$lieu`
##           diff           lwr           upr      p adj
## Super-Hyper -85.8 -156.7756 -14.824356 0.0122238
## Proximité-Hyper -121.0 -233.2223 -8.777654 0.0301690
```

```
## Spécialisé-Hyper      -133.6 -300.0526  32.852639  0.1554060
## Proximité-Super      -35.2  -147.4223  77.022346  0.8359511
## Spécialisé-Super     -47.8  -214.2526 118.652639  0.8686383
## Spécialisé-Proximité -12.6  -200.3839 175.183903  0.9979202
```

Attention, l'analyse de la variance suppose l'égalité des variances entre les différents groupes qui peut être vérifiée à l'aide des fonctions `bartlett.test()` si les variables sont normales et `LeveneTest()` sinon.

```
bartlett.test(courses$budget ~ courses$lieu)

##
## Bartlett test of homogeneity of variances
##
## data:  courses$budget by courses$lieu
## Bartlett's K-squared = 8.6334, df = 3, p-value = 0.03458

LeveneTest(courses$budget ~ courses$lieu, center='mean',alternative='two.sided')

## Levene's Test for Homogeneity of Variance (center = "mean": "two.sided")
##      Df F value  Pr(>F)
## group 3  2.6347 0.06186 .
##      43
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

11 Régression linéaire

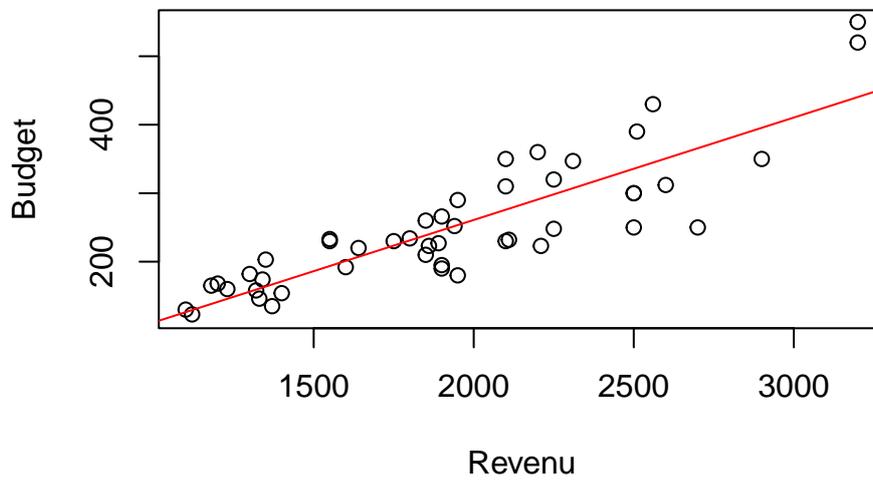
La fonction `lm()` permet d'obtenir la régression linéaire entre deux (ou plusieurs) variables.

```
lm( courses$budget ~ courses$revenu) -> reg
summary(reg)

##
## Call:
## lm(formula = courses$budget ~ courses$revenu)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -115.594  -35.683    3.004   30.437  109.630
##
## Coefficients:
##              Estimate Std. Error t value      Pr(>|t|)
## (Intercept)  -38.1985    26.2908  -1.453      0.153
## courses$revenu  0.1495     0.0131  11.416 0.000000000000000699 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 48 on 45 degrees of freedom
## (3 observations deleted due to missingness)
## Multiple R-squared:  0.7433, Adjusted R-squared:  0.7376
## F-statistic: 130.3 on 1 and 45 DF,  p-value: 0.0000000000000006991
```

On peut représenter les résultats avec le nuage de points suivant :

```
plot(courses$revenu, courses$budget, xlab='Revenu', ylab='Budget')
abline(reg, col='red')
```



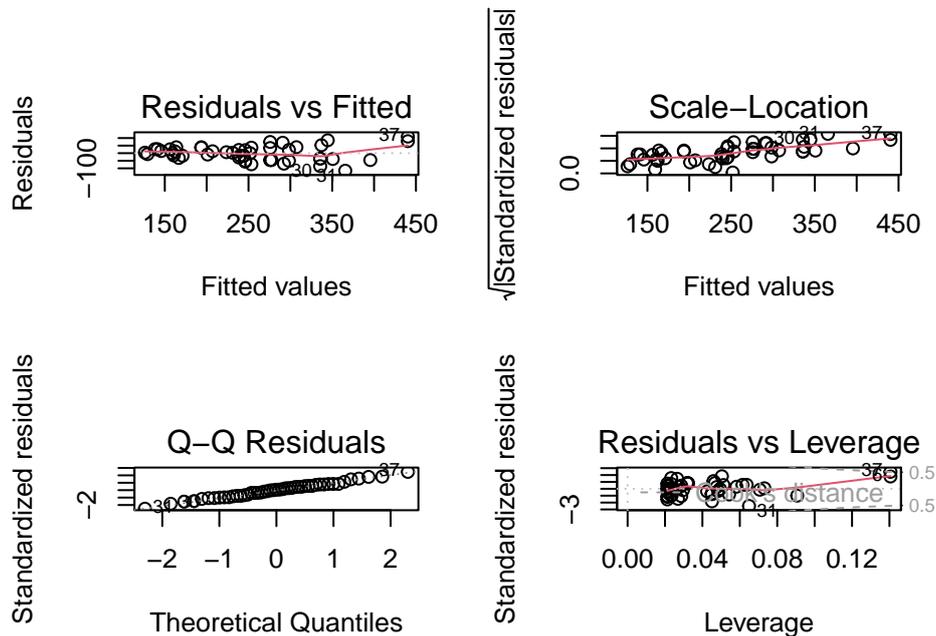
On vérifie la normalité des résidus avec un test de Shapiro.

```
shapiro.test(resid(reg))
```

```
##
## Shapiro-Wilk normality test
##
## data:  resid(reg)
## W = 0.99302, p-value = 0.9932
```

On obtient les représentations graphiques permettant de vérifier les hypothèses de la régression de la manière suivante :

```
layout(matrix(1:4, 2, 2))
plot(reg)
```



```
layout(matrix(1,1,1))
```

12 Manipulation des données

Outre la notation `data$variable` (ou directement `variable` si on a *attaché* le dataframe), il est aussi possible d'utiliser une notation « tableau » en utilisant le numéro ou le nom de la variable pour accéder aux variables.

```
courses[,2]
```

```
## # A tibble: 50 x 1
##   age
##   <dbl>
## 1    25
## 2    29
## 3    35
## 4    19
## 5    23
## 6    45
## 7    33
## 8    22
## 9    21
## 10   38
## # i 40 more rows
```

```
courses[, 'age']
```

```
## # A tibble: 50 x 1
##   age
##   <dbl>
## 1    25
## 2    29
## 3    35
## 4    19
## 5    23
## 6    45
## 7    33
## 8    22
## 9    21
## 10   38
## # i 40 more rows
```

Cela permet de sélectionner plusieurs variables avec une commande du type

```
courses[, c('age', 'revenu')]
```

```
## # A tibble: 50 x 2
##   age revenu
##   <dbl> <dbl>
## 1    25  2560
## 2    29  1330
## 3    35  1850
## 4    19    NA
## 5    23  1200
## 6    45  3200
## 7    33  2210
## 8    22  2500
## 9    21  1340
## 10   38  2100
## # i 40 more rows
```

De même, on peut sélectionner une ou plusieurs observations par son ou ses numéros

```
courses[7,]
```

```
## # A tibble: 1 x 15
##   num age sexe sitfam frequence lieu supermarche hypermarche proximite
##   <dbl> <dbl> <fct> <dbl> <ord> <fct> <dbl> <dbl> <dbl>
## 1    7  33 H          3 Hebdomadaire Proxi~ 0 0 1
## # i 6 more variables: specialise <dbl>, revenu <dbl>, budget <dbl>,
## # LKprix <ord>, LKproximite <ord>, LKvariete <ord>
```

```
courses[7:9,]
```

```
## # A tibble: 3 x 15
##   num age sexe sitfam frequence lieu supermarche hypermarche proximite
##   <dbl> <dbl> <fct> <dbl> <ord> <fct> <dbl> <dbl> <dbl>
## 1 7 33 H 3 Hebdomadaire Proxi~ 0 0 1
## 2 8 22 H 9 Bimensuel Hyper 0 1 0
## 3 9 21 F 3 Mensuel Speci~ NA NA NA
## # i 6 more variables: specialise <dbl>, revenu <dbl>, budget <dbl>,
## # LKprix <ord>, LKproximite <ord>, LKvariete <ord>
```

En combinant les deux, on obtient la valeur d'une variable pour une certaine observation.

```
courses[7, 'revenu']
```

```
## # A tibble: 1 x 1
##   revenu
##   <dbl>
## 1 2210
```

```
courses$revenu[7]
```

```
## [1] 2210
```

Il est souvent nécessaire de ne travailler que sur une partie des observations définie par un ou plusieurs critères. Les tests ==, <, >=, != permettent de sélectionner les observations dont une variable est égale, inférieure, supérieure ou égale, différente d'une valeur.

```
courses$sexe=='F'
```

```
## [1] FALSE TRUE TRUE TRUE FALSE NA FALSE FALSE TRUE NA FALSE TRUE
## [13] TRUE TRUE FALSE NA FALSE FALSE TRUE TRUE NA FALSE FALSE TRUE
## [25] NA FALSE TRUE TRUE TRUE NA TRUE TRUE FALSE NA TRUE TRUE
## [37] TRUE FALSE TRUE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE FALSE
## [49] TRUE FALSE
```

```
courses$revenu[courses$sexe=='F']
```

```
## [1] 1330 1850 NA NA 1340 NA 2900 1950 2500 NA 1750 1950 NA 1300 NA
## [16] 1800 2250 2600 NA 2700 NA NA 2100 1900 3200 1550 1550 1850 2100 2200
## [31] 2510 1600 1320
```

On peut combiner plusieurs conditions avec les conjonctions & (et) et | (ou).

```
courses$revenu[courses$sexe=='F' & (courses$revenu < 1500 | courses$revenu > 2000)]
```

```
## [1] 1330 NA NA 1340 NA 2900 2500 NA NA 1300 2250 2600 NA 2700 NA
## [16] 2100 3200 2100 2200 2510 1320
```

Attention, cela n'est possible que sur une variable particulière (et non un dataframe) et les conditions ne s'appliquent pas aux valeurs manquantes (NA). Pour supprimer ces deux limitations, il est préférable d'utiliser la fonction subset().

```
subset(courses, sexe=='F' & revenu > 2000) -> aSubset
head(aSubset)
```

```
## # A tibble: 6 x 15
##   num age sexe sitfam frequence lieu supermarche hypermarche proximite
##   <dbl> <dbl> <fct> <dbl> <ord> <fct> <dbl> <dbl> <dbl>
## 1 12 29 F 1 Hebdomadaire Hyper 0 1 0
## 2 14 19 F 2 Plus souvent Super 1 0 0
## 3 28 22 F 2 Hebdomadaire Super 1 1 0
## 4 29 20 F 2 Mensuel Super 1 0 0
## 5 31 48 F 4 Plus souvent Proxi~ 0 0 1
## 6 35 45 F 3 Mensuel Hyper 0 1 0
## # i 6 more variables: specialise <dbl>, revenu <dbl>, budget <dbl>,
## # LKprix <ord>, LKproximite <ord>, LKvariete <ord>
```

La fonction `cut()` permet de transformer une variable quantitative en une variable qualitative ordinale (mise en classes). On précise pour cela les bornes des intervalles (ouverts à gauche). L'option `dig.lab=5` évite l'utilisation de notation scientifique.

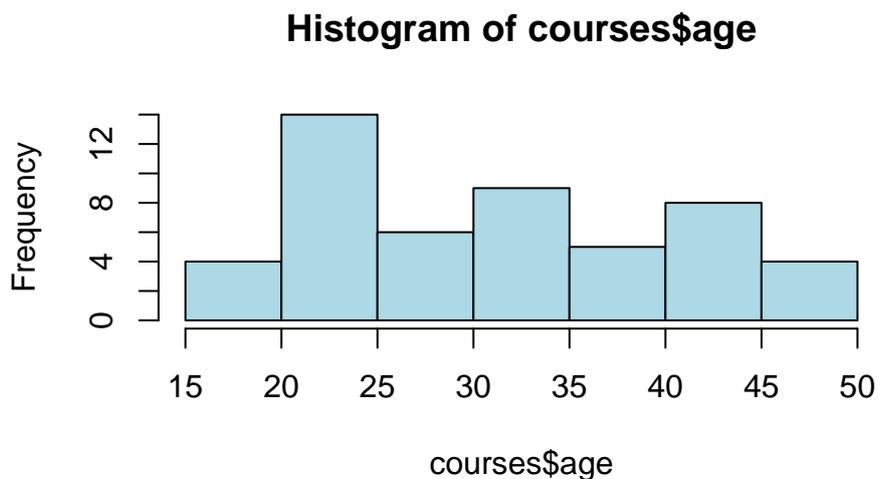
```
revenuclass <- cut(courses$revenu, c(0,1500,2000,2500,3000,3500), dig.lab = 5)
table(revenuclass)

## revenuclass
##      (0,1500] (1500,2000] (2000,2500] (2500,3000] (3000,3500]
##           12           16           12           5           2
```

13 Couleurs!

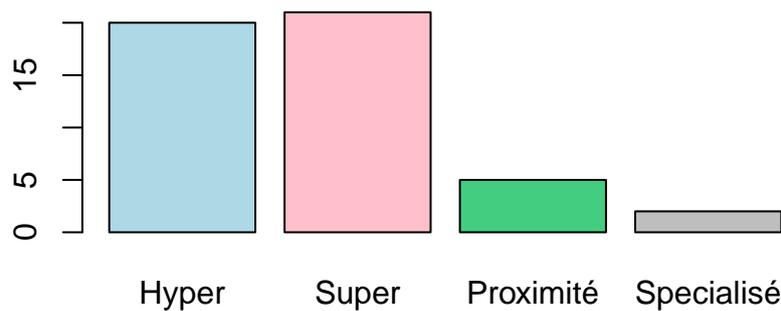
Dans la plupart des graphiques, il est possible d'indiquer la couleur du graphique via l'option `col` :

```
hist(courses$age, col='lightblue')
```



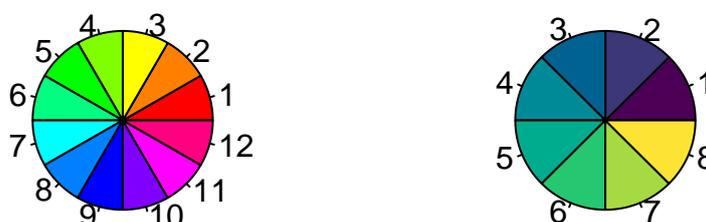
La liste des noms des 657 couleurs (de base) utilisables s'obtient avec la fonction `colors()`. Dans certains graphiques (`barplot`, `pie`), il est possible d'indiquer plusieurs couleurs (sous la forme d'un vecteur), une pour chaque bâton ou secteur :

```
barplot(table(courses$lieu), col=c('lightblue', 'pink', 'seagreen3', 'gray'))
```



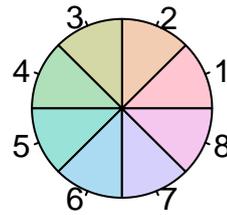
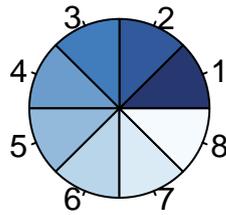
On peut aussi utiliser un *nuancier* prédéfini, via les fonctions `rainbow()` ou `hcl.colors()` en indiquant en argument le nombre de couleurs souhaitées.

```
pie(rep(1,12), col=rainbow(12))
pie(rep(1,8), col=hcl.colors(8))
```



Le nuancier `hcl.colors()` peut utiliser de nombreuses palettes de couleurs différentes via l'option `palette`. La fonction `hcl.pals()` donne la liste de toutes les palettes.

```
pie(rep(1,8),col=hcl.colors(8, palette="Blues"))
pie(rep(1,8),col=hcl.colors(8, palette="Pastel 1"))
```



14 Probabilités et échantillons

Il est bien-sûr possible de calculer les probabilités (fonctions de répartition) et quantiles des lois usuelles de la statistiques.

Les noms des fonctions à utiliser commencent par **p** pour les probabilités et par **q** pour les quantiles, suivi du nom de la loi parmi **norm** pour loi normale, **t** pour Student, **chisq** pour Khi-deux, **f** pour Fisher, **exp** pour exponentielle, **binom** pour binomiale, **hyper** pour hypergéométrique, **geom** pour géométrique, **pois** pour Poisson. Les paramètres dépendent bien-sûr des lois. Rechercher dans l'aide `pnom`, `qgeom`... pour obtenir la syntaxe exacte.

```
pnorm(1.96, mean=0, sd=1) # P(N(0,1) <= 1.96)
## [1] 0.9750021
qnorm(0.95, mean=0, sd=1) # z\_{0.95}
## [1] 1.644854
pt(2.1, df=50) # P(St(50) <= 2.1)
## [1] 0.9796023
qchisq(0.99, df=12) # quantile d'ordre 99% du khi-deux(12)
## [1] 26.21697
pf(9.3, df1=10, df2=15) # P(F(10,12) <= 9.3)
## [1] 0.9999104
pbinom(5, size=20, prob=0.25) # P(B(20,0.25) < 5)
## [1] 0.6171727
```

Il est aussi possible de générer un échantillon (aléatoire) de ces lois en utilisant le préfixe **r** (pour random). Par exemple, un 8-échantillon de la loi normale $\mathcal{N}(5, 3)$ est obtenu ainsi :

```
rnorm(8, mean=5, sd=3)
## [1] -0.170144 2.751677 3.616960 2.565120 2.699114 4.094478 2.520505
## [8] 1.849118
```

Pour obtenir le même échantillon à chaque utilisation, utiliser l'instruction `set.seed()`.

```
set.seed(12345678)
rnorm(5, mean=1, sd=2)
## [1] 3.2957822 4.2367987 -3.7581263 0.4866013 2.7847469
```

15 Calcul matriciel

Bien que cela ne soit pas son utilisation principale, il est aussi possible d'utiliser **R** pour effectuer du calcul matriciel.

On définit une matrice à l'aide de la fonction `matrix()`. Attention, les coefficients sont donnés sous forme de vecteur (en utilisant la fonction `c()`).

```
A <- matrix(c(1,2,3,4), ncol=2, nrow=2);A
##      [,1] [,2]
## [1,]   1   3
## [2,]   2   4

B <- matrix(c(1,2,3,4,5,6), ncol=3, byrow=TRUE);B
##      [,1] [,2] [,3]
## [1,]   1   2   3
## [2,]   4   5   6

C <- diag(c(-2,3));C
##      [,1] [,2]
## [1,]  -2   0
## [2,]   0   3
```

Les opérations standards se notent naturellement. On utilise la fonction `t()` pour calculer la transposée d'une matrice.

```
2*B
##      [,1] [,2] [,3]
## [1,]   2   4   6
## [2,]   8  10  12

A+C
##      [,1] [,2]
## [1,]  -1   3
## [2,]   2   7

t(B)
##      [,1] [,2]
## [1,]   1   4
## [2,]   2   5
## [3,]   3   6
```

La multiplication de deux matrices utilise la notation non standard `%*%`. L'inverse d'une matrice carrée se calcule avec la fonction `solve()`. La fonction `det()` calcule son déterminant.

```
A %*% B
##      [,1] [,2] [,3]
## [1,]  13  17  21
## [2,]  18  24  30

det(A)
## [1] -2

solve(A)
##      [,1] [,2]
## [1,]  -2  1.5
## [2,]   1 -0.5
```

Finalement, la fonction `eigen()` donne les valeurs propres et les vecteurs propres d'une matrice.

```
E <- matrix(c(2,1,1,2),ncol=2);E
##      [,1] [,2]
## [1,]    2    1
## [2,]    1    2
eigen(E)
## eigen() decomposition
## $values
## [1] 3 1
##
## $vectors
##      [,1] [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

Les valeurs propres sont donc 3 et 1. Les vecteurs propres sont normés (sont de norme 1). Habituellement (par le calcul à la main) on trouve plutôt :

```
eigen(E)$vectors * sqrt(2)
##      [,1] [,2]
## [1,]    1  -1
## [2,]    1    1
```