Initiation au logiciel R destinée aux élèves-ingénieurs de l'ESSAI

Ines Abdeljaoued Tej et Vincent Richard

12 mars 2018

Le logiciel R est un logiciel de statistique libre. Il est accessible au plus grand nombre. Il est constitué de librariries ou de packages permettant de réaliser des études statistiques. Il répond à un besoin des chercheurs et des ingénieurs des pays comme la Tunisie, où les moyens sont limités mais où la matière grise est à profusion.

Le logiciel R a son propre langage et il oblige a une période d'apprentissage pour se familiariser avec sa programmation. Il utilise un langage orienté objet très pratique dans la maniuplation et le traitement des données.

Une fois le logiciel `R` bien installé, et que les premiers pas sur ce logiciel connus, nous ferons en sorte d'**importer** vos données dans `R`. Ceci signifie que les données, obtenues depuis le web ou pas, soit sous forme d'un fichier que vous téléchargez [1]. Sans données, il n'est pas possible de parler de traitement de données !

Une fois les données entre vos mains, il va falloir les **ranger**, c'est-à-dire les ordonner sous formes de colonnes (où chaque colonne représente une variables) et de lignes (ou chaque ligne est un enregistrement ou une observation). Il est primordial que vos données soient sous la bonne forme. Ainsi vous pouvez vous concentrer sur la compréhension, l'analyse, la visualisation, la modélisation de vos données. ceci permet également d'avoir une juste communication à propos de ces données.

A partir de données bien structurées, la première étape consiste en général à **transformer** ces données. Par exemple, s'assurer que les variables d'intérêts soient réduites (tous les élèves d'une même classe, tous les produits d'une même ferme, etc). Ceci passe aussi par la création de nouvelles variables fonction de variables existantes (comme le calcul de la vitesse et du temps) et le calcul d'un ensemble de statistiques (comme les chiffres ou les moyennes). Ensemble, le rangement et la transformation sont appelés «luttes», parce que le fait d'obtenir vos données sous une forme naturelle est souvent un combat !
Avec des données rangées et comportant des variables judicieusement choisies, il reste le plus important : **visualiser** ces données et les **modéliser**. Ces deux opérations ont leur force et leur faiblesse. Il s'agira de passer de l'une à l'autre pour mieux comprendre les données.
Une bonne visualisation permet de montrer des choses parfois inattendues, à propos des données, ou balayer des suppositions et des questionnements à propos des données (ça appelle par exemple à la collecte de nouvelles données). La modélisation permet de compléter la visualisation par des outils mathématiques. Elle dépend de la question à poser et donne une réponse à cette question, sous un certain nombre d'hypothèses et de suppositions.

La dernière étape dans le travail d'un statisticien et d'un analyste est de **communiquer**. C'est une étape critique de tout projet.

L'objectif de ce cours est d'aider les élèves-ingénieurs de l'ESSAI à importer des donnés, à les structurer, à les transformer afin de les visualiser et de proposer des modèles. Le but aussi est d'aider à communiquer autour des résultats obtenus.

Le plan de ce cours suit donc cette logique d'exploration des données, avec la mise en accent d'un certain nombre de packages (ou librairies) de plus en plus utilisés. Il y a par exemple `Shiny`, `ggplot2`, `data.table`, `devtools`, `dplyr` etc.

1. Il s'agit de sauvegarder ces données sous forme d'un tableau ou *data frame* sur `R`.

Table des matières

Chapitre 1

Premiers pas sous R

De omnibus dubitandum ! Il faut douter de toute chose ! dixit René Descartes (1596-1650).

1.1 Installation

Les élèves-ingénieurs sont invités à télécharger le logiciel R depuis le site web `http://cran.r-project.org` ainsi que l'environnement de travail `Rstudio`. Il faut respecter le type de machine (32bit ou 64bit) ainsi que le système d'exploitation (linux, win ou macosx).

Il est également demandé aux élèves-ingénieurs d'ouvrir un compte sur `http://cocalc.com`. Un projet appelé ProgrammationMathématique sera envoyé aux élèves, contenant tout le matériel du module, afin de suivre le cours. L'environnement de travail utilisé est alors `Jupyter` qui comprends plusieurs noyaux dont `Python`.

Il est enfin possible de se contenter d'une utilisation de `R` par ligne de commande, sur un terminal ou une console.

1.2 Pourquoi R ?

Le logiciel `R` sous licence GNU est facile à installer. C'est parmi les logiciels les plus utilisés de la communauté statistique académique et dans les services de recherche et développement des entreprises industrielles.

Dans sa structure, `R` est un langage de programmation d'une syntaxe proche de celle du logiciel `C`. Il est capable de manipuler des objets complexes sous forme de scalaire, vecteur, liste, matrice, et aussi tableaux ou *data frame*. Il propose des librairies sur pratiquement toutes les méthodes statistiques de la littérature. En effet, toutes les recherches sont d'abord développées et diffusées à l'aide de ce logiciel par la communauté scientifique.

1.3 Types de variables

Plusieurs documents sont disponibles sur le net mais nous avons choisi de travailler ce cours d'initiation à R en nous inspirant, très largement, du document produit par M. Vincent Richard de l'Institut Pasteur de Madagascar. Le document produit par Dr. Richard est très pédagogique et s'adapte bien à cet enseignement sous forme de Travaux Pratiques.

1.3.1 Créer des objets sous R

```
In [1]: sum(5,6)
```

Les fonctions étant nommées par des lettres (exemple `sum()`), il est recommandé pour les objets de type `données` que vous utilisiez une écriture commençant par une majuscule.
Il y a trois façons d'affecter une valeur à un objet R :

```
In [3]: C1 <- 5
        C2 = 6
        7 -> C3
        C1; C2; C3;

In [7]: # Ce symbole sert à mettre du commentaire
        C1

In [6]: print(C2); print(C3)

[1] 6
[1] 7
```

R peut être utilisé comme calculatrice : les opérateurs classiques sont l'addition +, la soustraction -, la multiplication *, la division / et la puissance ^.

```
In [11]: 5/3

In [14]: 320 + 120

In [13]: C1 + C2

In [15]: sum(C1,C2)
```

La manipulation des chaînes de caractères se fait avec " " ou ' ' :

```
In [11]: "ah !" -> A1
         A.1 <- "Hello Word"
         A.2 <- "Bonjour"
         A.1; A.2

In [23]: paste(A.1,"Utilisateurs de R à l ESSAI", sep="--")

In [28]: # La liste des objets crées dans la console est obtenue par :
         ls()

In [29]: rm(A1)

In [30]: ls()

In [31]: # Pour supprimer l'ensemble des objets de la console, utilisez :
         rm(list=ls())

In [32]: ls()
```

Pour retrouver la liste des commandes utilisées dans la console, il y a la commande `history()` ou `history(Inf)`. Elles vont ouvrir une fenêtre sur `Rstudio` dans laquelle vous verrez les dernières commandes utilisées (respectivement toutes les commandes utilisées). Vous pourrez ainsi sauvegarder vos commandes dans un fichier texte ou copier/coller dans un script de programme.

```
In [64]: #help(history)

In [65]: #?sum
```

La commande `help.start()`permet d'avoir accès directement à l'ensemble de l'aide au format html et ceci, à partir de votre navigateur.

1.3.2 Installer une extension (package)

Le logiciel `R` utilise un système d'extension, les packages ou librairies, pour répondre à des besoins d'analyse spécifiques. Le logiciel `R` est d'ailleurs lui-même une bibliothèque appelée **base** contenant un certain nombre d'extensions utiles à son fonctionnement de base.

```
In [43]: # Pour connaître la liste des extensions lancées dans votre session :
         search()
```

Pour installer d'autres packages, il faudra les télécharger sur le CRAN. Puis dans le menu `software` de la page d'accueil, vous devrez cliquer sur `packages`. La liste des extensions s'affiche et vous devrez choisir celle que vous souhaitez télécharger.

```
In [45]: # Pour appeler une extension dans la console, utiliser la fonction :
         library(nnet)
```

1.3.3 Répertoire de travail et Citation

```
In [2]: # w pour working et d pour directory
        getwd()
```

C'est le répertoire par défaut utilisé par R. Vous pouvez définir votre propre répertoire de travail afin de sauvegrader chacune de vos sessions de travail avec les fichiers de données que vous exploiterez avec R : `setwd(dirname(file.choose()))`. Notez que `file.choose()`permet de choisir un fichier de façon interactive d'où la nécessité d'avoir au moins un fichier dans le répertoire. La commande `dirname()` extrait uniquement le chemin du fichier.

```
In [7]: #Pour citer R :
        citation()
```

```
Out[7]:
        To cite R in publications use:

          R Core Team (2017). 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 = {2017},
            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.
```

```
In [6]: #Pour quitter R : q()
```

Lorsque vous quitterez R, une boîte de dialogue s'affichera pour vous demander si vous souhaitez sauvegarder votre espace de travail. Si oui, R créera un fichier avec une extension `.RData` contenant les objets crées lors de la session que vous venez de quitter. Cliquer sur ce fichier permet de l'ouvrir dans R. Un autre fichier est crée : `.Rhistory` pouvant être ouvert à l'aide de votre éditeur de texte. La commande `source()` permet de relancer rapidement votre script : `source(file.choose())`.

1.3.4 Exercices

Calcul de Taux de mortalité d'une population

1. Créer un objet R appelé 'Femme' contenant la valeur 257 836, correspondant à la population féminine d'une région donnée au cours de l'année.
2. Créer un objet R appelé 'Homme' contenant la valeur 247 643, correspondant à la population masculine d'une région donnée (à la même année).
3. Créer un objet R appelé 'Population' à partir des deux objets 'Femme' et 'Homme'. Afficher le total de la population de la région dans la feuille de travail.
4. Créer un objet R appelé 'Sexratio' qui sera le rapport 'Homme/Femme'. Afficher la valeur de cet objet dans la console.
5. Si le nombre de décès dans la région au cours de l'année chez les femmes est de 236 et celui des hommes de 328, à l'aide d'objets R vous donnerez le taux de mortalité pour 1000 personnes pour chacun des 2 sexes et pour l'ensemble de la population. Afficher les valeurs des objets dans la feuille de travail (ou la console).

```
In [4]: # Création de l'objet Femme
        Femme <- 257836

        # Création de l'objet Homme Homme <- 247643

        # Création de l'objet Population
        Population <- sum(Femme,Homme)
        # ou
        Population <- Femme+Homme

        # Afficher le contenu de l'objet Population
        print(Population)

        # Création de l'objet Sexratio
        Sexratio <- Homme/Femme

        # Affichage de l'objet Sexratio
        print(Sexratio)

        # Création de l'objet Dcfemme
        Dcfemme <- 236

        # Création de l'objet Dchomme
        Dchomme <- 328

        # Création de l'objet Dctotal pour la somme des décès
        Dctotal <- Dcfemme+Dchomme
        # ou
        DcTotal <- sum(Dcfemme,Dchomme)

        # Création de l'objet Mortfemme pour mortalité des femmes
        Mortfemme <- (Dcfemme/Femme)*1000
```

```
        # Création de l'objet Morthomme pour mortalité des hommes
        Morthomme <- (Dchomme/Homme)*1000

        # Création de l'objet Mortpop pour mortalité de la population
        Mortpop <- (Dctotal/Population)*1000

        # Affichage du contenu des objets sur la mortalité
        print(Mortfemme); print(Morthomme); print(Mortpop)
```

```
[1] 505479
[1] 0.9604671
[1] 0.9153105
[1] 1.324487
[1] 1.115773
```

Taux de naissance

1. Créer un objet R appelé 'Femme' contenant la valeur 135 429, correspondant à la population féminine d'une région donnée au cours de l'année.
2. Si le nombre de naissance dans la région au cours de l'année est de 378, à l'aide d'objets R vous donnerez le taux de naisance pour 1000 personnes. Afficher les valeurs des objets dans la feuille de travail (ou la console).

1.4 Différents types d'objets sous R

La nature des données sous R est soit 'null' (ou objet vide), soit 'logical' (ou booléen), soit 'numeric' (ou nombre réel comme 15, 3.76, pi, 1e-9), soit 'complex' (nombre complexe sous la forme 3i), soit 'character' (ou chaîne de caractères).

Il y a divers types d'objets sous R. Nous les détaillons dans les sections suivantes.

1.4.1 Un vecteur

le vecteur est un objet à une seule dimension contenant des informations de même nature, même mode (soit toutes numériques, soit toutes alphanumériques ou chaînes de caractères, soit logiques (T ou F), soit vides. C'est la fonction **c()** qui est utilisée pour définir un vecteur.

```
In [17]: # Vecteur
         Age <- c(2,4,5,10,8,7,12,11,3,5,6)
         #Age <- c(2,4,5,10,8,7,12,11,3,5,6, bycol=TRUE)
         mode(Age)
         #help(c)
         Age
```

1.4.2 Une matrice

Une matrie a deux dimensions contenant les informations de même nature, même mode (soit toutes numériques, soit toutes alphanumériques ou chaînes de caractères, soit logiques (T ou F), soit vides. On utilise la fonction **matrix()** avec un paramètre qui indique le nombre de colonnes :

```
In [16]: # Matrice
         Chiffre <- c(1:15)
         Matrice.1 <- matrix(Chiffre,ncol=3)
         Matrice.2 <- matrix(Chiffre,ncol=3,byrow=TRUE)
         print(Matrice.1); print(Matrice.2)
```

1.4.3 Une liste

C'est un objet qui permet de stocker des objets de différents modes (ou pas). Ils peuvent ne pas avoir la même longueur. Nous utilisons la fonction **list()** pour le créer.

```
In [17]: # Liste
         Maliste <- list(Age, Matrice.1, Matrice.2)
         print(Maliste)
```

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

[[2]]
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15

[[3]]
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
[5,]   13   14   15
```

1.4.4 Un tableau de données ou data frame

Un tableau de données comprend des colonnes de même longueur mais de modes qui peuvent être différents. les colonnes correspondent à des variables et les lignes à des enregistrements. Cet objet permet de manipuler les données collectées à partir de tableurs. La fonction utilisée est **data.frame()** sous R.

```
In [26]: # Tableau ou data frame
         Sexe <- rep(c("M","F"), c(16,14))
         print(Sexe)
```

```
 [1] "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "M" "F" "F" "F"
[20] "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F"
```

```
In [27]: Age <- rep(seq(10,14.5,by=0.5),3)
         Age
```

```
In [28]: Matable <- data.frame(Sexe, Age)
         #View(Matable)
         print(Matable)
```

```
   Sexe  Age
1     M 10.0
2     M 10.5
3     M 11.0
4     M 11.5
```

```
5     M 12.0
6     M 12.5
7     M 13.0
8     M 13.5
9     M 14.0
10    M 14.5
11    M 10.0
12    M 10.5
13    M 11.0
14    M 11.5
15    M 12.0
16    M 12.5
17    F 13.0
18    F 13.5
19    F 14.0
20    F 14.5
21    F 10.0
22    F 10.5
23    F 11.0
24    F 11.5
25    F 12.0
26    F 12.5
27    F 13.0
28    F 13.5
29    F 14.0
30    F 14.5
```

Pour voir la structure de la table avec les caractéristiques de chacune des colonnes la constituant, utiliser la commande **str()** :

```
In [24]: str(Matable)
```

```
'data.frame':        30 obs. of  2 variables:
 $ Sexe: Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ Age : num  10 10.5 11 11.5 12 12.5 13 13.5 14 14.5 ...
```

1.4.5 Exercices

Les opérateurs de comparaisons sont <, >, <=, >=, == et !=. Les opérateurs logiques sont &, | pour le ou, !pour le non. Les opérateurs d'extractions sont [] et $.

Extraction de données

1. Extraire les âges strictement inférieurs à 10 ans et supérieurs ou égaux à 5 ans ;
2. Extraire les âges qui sont différents de 5 ans ;
3. Extraire la quatrième valeur du vecteur Age.

```
In [28]: Age <- c(2,4,5,10,8,7,12,11,3,5,6)
         print(Age)
         Age[Age>=5 & Age<10]
```

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

```
In [26]: Age[Age!=5]

In [27]: Age[4]
```

Les attributs spéciaux

Certains attributs réalisent des fonctions et d'autres ne font rien. Nous pensons en particulier à **NA** pour les valeurs manquantes (Not Available) ; **TRUE** ou **T** qui est la valeur logique pour Vrai ; **FALSE** ou **F** qui est la valeur logique pour False ; **NULL** pour créer un objet vide ; **Inf** ou **-Inf** pour l'infini ou $-\infty$; **letters** pour obtenir les 26 lettres de l'alphabet en minsucule ; **pi** qui vaut $\pi = 3.141593$. Il y a la fonction **ifelse()** ou **if(){}else{}** qui permettent d'affecter une valeur à un objet en fonction de la valeur d'un autre objet. La fonction **for(){}** pour les répétitions en boucle ; **while(){}** ou **repeat()**.

Que représentent les instructions suivantes :

```
In [30]: # Les attributs spéciaux
         Parcelle <- letters[1:10]; print(Parcelle)

 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"


In [32]: Rayon <- c(5,6,7,8,12,4,3,10,8,9)
         Aire <- pi*Rayon^2; print(Aire)

 [1]  78.53982 113.09734 153.93804 201.06193 452.38934  50.26548  28.27433
 [8] 314.15927 201.06193 254.46900


In [33]: Aire[Parcelle=="e"]

In [34]: Limite.1 <- ifelse(Aire<100,0,1)
         Limite.1

         Limite.2 <- NULL
         for(i in 1:10){
             if(Aire[i]<100){Limite.2[i] <- 0}else{Limite.2[i] <-1}}
         Limite.2
```

1.5 Manipulation des données

Les données qualitatives peuvent être soit binaires (`TRUE` ou `FALSE`), nominales ("Cheval", "Vache", "Mouton") ou ordinales ("CP", "CE1", "CE2", "CM1", "CM2"). Les variables ordinales ont un classement dont l'ordre est fait sur chacun des niveaux qui la composent (contrairement aux variables nominales).

```
In [37]: # Exemple d'estimation d'une douleur lors d'une étude :
         Niveau <- rep(c("Supportable","Faible","Intense","Forte","Absente"), c(5,16,18,7,22))

In [38]: summary(Niveau)

Out[38]:    Length     Class      Mode
                68 character character

In [39]: print(Niveau)
```

```
 [1] "Supportable" "Supportable" "Supportable" "Supportable" "Supportable"
 [6] "Faible"      "Faible"      "Faible"      "Faible"      "Faible"
[11] "Faible"      "Faible"      "Faible"      "Faible"      "Faible"
[16] "Faible"      "Faible"      "Faible"      "Faible"      "Faible"
[21] "Faible"      "Intense"     "Intense"     "Intense"     "Intense"
[26] "Intense"     "Intense"     "Intense"     "Intense"     "Intense"
[31] "Intense"     "Intense"     "Intense"     "Intense"     "Intense"
[36] "Intense"     "Intense"     "Intense"     "Intense"     "Forte"
[41] "Forte"       "Forte"       "Forte"       "Forte"       "Forte"
[46] "Forte"       "Absente"     "Absente"     "Absente"     "Absente"
[51] "Absente"     "Absente"     "Absente"     "Absente"     "Absente"
[56] "Absente"     "Absente"     "Absente"     "Absente"     "Absente"
[61] "Absente"     "Absente"     "Absente"     "Absente"     "Absente"
[66] "Absente"     "Absente"     "Absente"
```

La fonction `ordered` permet de présenter vos résultats par ordre croissant d'importance :

```
In [42]: Niveau.1 <- ordered(rep(c("Supportable","Faible","Intense", "Forte","Absente"),c(5,16,18,7,22)),
    levels=c("Absente", "Faible","Supportable","Forte","Intense"))
         summary(Niveau.1)
```

La commande **factor** permet la factorisation de variables nominales (comme "Masculin", "Féminin"). Les données de format `character` sont factorisées automatiquement mais pas les données numériques. Pour qu'une variable quantitative soit considérée comme une classe, nous utilisons la commande `factor()`.
En effet, les données quantitatives sont soit **continues** : mesures avec une unité de mesure (par exemple l'âge) ; **discrètes** : nombre d'enfants, code postal, etc ; **temporelles** : dates.

```
In [47]: # Si nous codons l'estimation de la douleur en chiffres :
         Niveau.A <- rep(c(3,2,5,4,1),c(5,16,18,7,22))
         print(Niveau.A)
         summary(Niveau.A)
```

```
 [1] 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[39] 5 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
Out[47]:    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
            1.00    1.00    2.00    2.75    5.00    5.00
```

```
In [51]: Niveau.B <- factor(rep(c(3,2,5,4,1),c(5,16,18,7,22)))
         print(Niveau.B)
         summary(Niveau.B)
         table(Niveau.B)
```

```
 [1] 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[39] 5 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Levels: 1 2 3 4 5
```

```
Out[51]: Niveau.B
          1  2  3  4  5
         22 16  5  7 18
```

```
In [52]: Niveau.C <- ordered(rep(c(3,2,5,4,1),c(5,16,18,7,22)), levels=c(5,4,3,2,1))
         summary(Niveau.C)
```

1.5.1 Manipulation des dates

```
In [21]: # Convertir les données en date reconnue par R :
         Dcalend.1 <- c("17/09/92","27/02/92","14/01/92","28/02/92")
         Dcalend.1

In [22]: Dcalend.11 <- as.Date(Dcalend.1,"%d/%m/%y")
         Dcalend.11

In [23]: Dcalend.2 <- c("17/09/1992","27/02/1992","14/01/1992","28/02/2002")
         Dcalend.2

In [24]: Dcalend.21 <- as.Date(Dcalend.2,"%d/%m/%Y")
         Dcalend.21

In [25]: # Extraire le numéro de semaine d'une date :
         as.character(Dcalend.21,"%U")

In [26]: # Extraire l'année d'une date :
         as.character(Dcalend.21,"%Y"); substr(Dcalend.21,1,4)

In [27]: # Pour l'année à deux chiffres :
         as.character(Dcalend.21,"%y"); substr(Dcalend.21,3,4)

In [28]: # Le mois d'une date :
         as.character(Dcalend.21,"%m"); substr(Dcalend.21,6,7)

In [29]: # Le jour d'une date :
         as.character(Dcalend.21,"%d"); substr(Dcalend.21,9,10)

In [30]: # Le jour dans l'année :
         as.character(Dcalend.21,"%j")

In [35]: # La différence entre deux dates
         Dcalend.3 <- c("23/02/1993","17/07/1994","24/10/1995","06/03/1992")

In [36]: Dcalend.31 <- as.Date(Dcalend.3,"%j/%m/%Y")
         Dcalend.31

In [39]: Temps.d <- difftime(Dcalend.31, Dcalend.21,units="days")
         Temps.d

Out[39]: Time differences in days
         [1]   159   871  1379 -3646

In [40]: Temps.w <- difftime(Dcalend.31, Dcalend.21,units="weeks")
         Temps.w

Out[40]: Time differences in weeks
         [1]   22.71429  124.42857  197.00000 -520.85714

In [41]: Temps.m <- difftime(Dcalend.31, Dcalend.21,units="min")
         Temps.m

Out[41]: Time differences in mins
         [1]   228960  1254240  1985760 -5250240

In [46]: Temps <- difftime(Dcalend.31, Dcalend.21,units="auto")
         Temps <- as.numeric(Temps)
         Temps
```

Le paramètre **units** de la fonction **difftime** peut aussi prendre les valeurs **weeks**, **mins**, **secs**, **hours** ou **auto**. Notez que l'objet **Temps** n'est pas considéré comme une variable numérique. S'il faut l'utiliser pour des calculs, il faudra d'abord transformer **Temps** en variable numérique avec la fonction **as.numeric()** : `Temps <- as.numeric(Temps)`.

1.5.2 Créer une table de données

```
In [47]: set.seed(3)
```

La fonction `rbinom()` permet de générer de façon aléatoire des données selon une distribution binomiale. La fonction `set.seed()` permet de fixer le tirage réalisé par la fonction `rbinom()` et de s'assurer que chacun d'entre-vous aura un objet R qui aura le même contenu.

```
In [51]: Sexe <- rbinom(31, size=1, prob=.48)
         Sexe1 <- ifelse(Sexe==0,"Femme","Homme")
         Sexe1

In [53]: Tranchage <- rep(c("0:4","5:9",">10"),c(8,9,14))
         Tranchage

In [55]: Poids <- seq(10,25,by=0.5)
         Poids

In [58]: Matable <- data.frame(Sexe,Sexe1,Poids,Tranchage)

         # Les 5 premières lignes :
         Matable[1:5,]

In [60]: # La distribution de l'ensemble des donénes selon leur nature :
         summary(Matable)
```

```
Out[60]:       Sexe          Sexe1        Poids       Tranchage
          Min.   :0.0000   Femme:17   Min.   :10.00   0:4: 8
          1st Qu.:0.0000   Homme:14   1st Qu.:13.75   5:9: 9
          Median :0.0000              Median :17.50   >10:14
          Mean   :0.4516              Mean   :17.50
          3rd Qu.:1.0000              3rd Qu.:21.25
          Max.   :1.0000              Max.   :25.00
```

La fonction pour générer des données selon une loi de distribution normale :

```
In [61]: rnorm(10); rnorm(10, mean=100)
```

La fonction pour générer des données selon une loi de distribution uniforme :

```
In [62]: runif(10); runif(10, min=20, max=21)
```

1.5.3 Exporter des données

Le premier paramètre de la fonction `write.csv2()` correspond au nom de l'objet que l'on souhaite exporter. Le second paramètre est le nom que l'on souhaite donner au fichier de sortie. le paramètre `row.names=T` permet de prendre en compte l'identifiant des lignes attribués automatiquement par R.
Notez que le fichier se crée dans le même répertoire de travail.

```
In [63]: write.csv2(Matable, file="Matable.csv", row.names=F)
```

1.5.4 Importer des données

Nous utilisons, en général, la fonction `read.table()` pour importer nos données : `Matable <- read.table(file.choose(), header=T, sep=";", dec=",")`

```
In [70]: library(foreign)
         # read.dta(file.choose())
```

```
In [72]: #library(xlsReadWrite)
         # read.xls(file.choose(), clNames=TRUE, sheet=1, from=1, rowNames=T)

In [84]: print(getwd())

         Matable1 <- read.table("/home/user/.../Matable.csv",header=T,sep=";",dec=",")

[1] "/home/user/..."
```

Pour connaître le format du fichier, utiliser `file.show()`.

```
In [85]: dim(Matable1)
```

Extraction de données

1. Créer un objet appelé `Poids` qui sera une séquence de nombres de 20 à 45 avec un pas de 0.5 ;
2. Afficher le contenu de l'objet `Poids` ;
3. Extraire la 16ème valeur de l'objet `Poids` ;
4. Extraire les 1ère, 16ème et 31ème valeurs de l'objet `Poids` ;
5. Extraire les poids dont la valeur est supérieure à 38 ;
6. Extraire les poids dont les valeurs sont à la fois supérieures à 25 et inférieures à 37 ;
7. Créer un objet R appelé `Classpoids` qui prendra la valeur 1, si le poids est inférieur à 25, la valeur 2 si le poids est compris entre 25 et 30 et 3 si le poids est supérieur à 30. Quel est l'effectif de chacun des groupes ?

```
Out[77]: Classpoids
          1  2  3
         10 11 30
```

Manipulation de champs "date"

1. Créer deux objets R de type vecteur nommés respectivement Datdeb et Datfin. Le premier contiendra les chaînes de caractères suivantes : 10/01/95, 02/05/95, 22/12/95, 05/09/96 ; le second les valeurs suivantes : 25/02/1995, 15/06/1996, 20/05/1998, 14/04/1999 ;
2. Convertir ces vecteurs au format date ;
3. Créer un objet appelé Duree qui contiendra la durée entre Datdeb et Datfin et l'afficher sur la console ;
4. Créer un objet An qui ne contiendra que les années de l'objet Datdeb, un objet Mois qui ne contiendra que les mois et un objet Jour qui ne contiendra que les jours de l'objet Datdeb. Afficher sur la console chacun de ces objets.

Chapitre 2

Explorer les tables de données avec R

Cette leçon consiste à montrer aux élèves-ingénieurs de l'ESSAI comment explorer et nettoyer les tables de données avec R. L'objectif de ces travaux pratiques est de connaître le contenu d'une table de données, de pouvoir connaître puis modifier les noms des variables, de créer de nouvelles variables, d'identifier des données aberrantes et enfin de fusionner des tables de données. Cette leçon est largement inspirée du polycope de cours de Vincent Richard.

2.1 S'assurer de la complétude de l'importation

Commençons tout d'abord par récupérer une table de données (celle générée lors du dernier TP). Nous allons montrer comment connaître le contenu d'une table de données, connaître et modifier les noms de variables et créer de nouvelles variables. Il s'agira aussi d'identifier les données aberrantes et de fusionner des tables de données.
Il est également important de connaître les différents types de variables, de connaître les paramètres de dispersion des variables qualitatives et des variables quantitatives.

```
In [11]: getwd()
```

```
In [12]: Matable <- read.table("/home/user/ProgrammationMathematique/InitiationR/Matable.csv",
header=T,sep=";",dec=",")
```

```
In [13]: dim(Matable)
```

2.1.1 Pour connaître le nombre de lignes/colonnes

```
In [14]: print("nombre de lignes"); dim(Matable)[1]
         print("nombre de colonnes"); dim(Matable)[2] ; length(Matable)
```

```
[1] "nombre de lignes"


[1] "nombre de colonnes"
```

La fonction **length()** permet également de donner uniquement le nombre de colonnes.

2.1.2 Retrouver le nom des variables

Une fois l'importation effectuée, et afin de voir comment sont nommées les variables (pour les utiliser sans faire d'erreurs dans leur retranscription), il y a la fonction `names()` :

```
In [15]: # Retrouver le nom des variables :
         names(Matable)
```

```
In [16]: # La structure de la base de données
         str(Matable)
```

```
'data.frame':        31 obs. of  4 variables:
 $ Sexe     : int  0 1 0 0 1 1 0 0 1 1 ...
 $ Sexe1    : Factor w/ 2 levels "Femme","Homme": 1 2 1 1 2 2 1 1 2 2 ...
 $ Poids    : num  10 10.5 11 11.5 12 12.5 13 13.5 14 14.5 ...
 $ Tranchage: Factor w/ 3 levels "0:4","5:9",">10": 1 1 1 1 1 1 1 1 2 2 ...
```

2.1.3 Afficher les données

Pour voir si l'importation des données s'est bien passée, on peut afficher ces données. Soit en tapant le nom de l'objet sur la console, soit en utilisant la commande `head()`. Elle n'affiche par défaut que les six premières lignes (il est possible de lui demander d'afficher plus ou moins, en indiquant le nombre de lignes à afficher).

```
In [17]: head(Matable)
```

```
In [18]: head(Matable, 2)
```

Pour connaître l'identifiant des observations ou modifier ce dernier, la fonction **row.names()** est bien utile : attribuer des identifiants aux observations de l'objet Matable :

```
In [19]: # La fonction paste() permet de concaténer des chaînes de caractères :
         Ident <- paste("VR",c(1:31))
         row.names(Matable) <- Ident
         head(Matable)
```

Pour effacer l'identifiant des lignes, on attribue alors le paramètre NULL en utilisant la fonction suivante :

```
In [20]: row.names(Matable) <- NULL
         head(Matable)
```

La fonction **View(Matable)** permet d'afficher les données dans une nouvelle fenêtre. C'est une fonction qui commence par une majuscule.

2.1.4 Appeler une variable d'une table de données

Les opérateurs **$** ou **[]** permettent d'extraire de l'information (la fonction **attach(Matable)** permettra plus tard d'appeler directement les variables de Matable sans passer par $) :

```
In [21]: Matable$Sexe1
```

```
In [22]: Matable[3,]
```

```
In [23]: Matable[,3]
```

Pour modifier le nom d'une variable, utiliser la fonction **names()** :

```
In [24]: names(Matable)[Matable$Sexe] <- "CODSEX"
         names(Matable)[3] <- "POIDS"
```

```
In [25]: Matable[3,]
```

Pour modifier plusieurs noms en même temps :

```
In [26]: names(Matable)[c(2,4)] <- c("SEXE","GROUPAGE")
         Matable[3,]
```

A quoi servent les fonctions suivantes ?

```
In [27]: Nomvar <- names(Matable) # pour sauvegarder.
         names(Matable) <- NULL # pour effacer.
         #View(Matable) # pour voir les modifications de l'objet.
        names(Matable) <- Nomvar # pour réattribuer les noms.
         #View(Matable) # pour voir les modifications de l'objet.
```

Si l'on souhaite afficher les données collectées pour un enregistrement, (i.e. le contenu d'une ligne) ou encore les données d'une variable (i.e. d'une colonne), on utilise l'opérateur d'extraction [] :

```
In [28]: Matable[3,4]

In [29]: Matable[20:25,1:3]

In [30]: Matable[c(20,25),c(1,3)]

In [31]: # Autre façon d'extraire toutes les variables sauf la première
         Matable2 <- Matable[1:20,-1]

In [32]: # Cette commande extrait dans un objet R les enregistrements de 1 à 20pour les variables 1, 3 et 4.
         Matable3 <- Matable[1:20,c(1,3,4)]

In [33]: # Cette commande extrait dans un objet R tous les enregistrements sauf le 20ème,et toutes les variab
         Matable4 <- Matable[-20,c(-1,-3)]
```

Vous pouvez vérifier les résultats des extractions avec la fonction `View()` ou comparer le nombre de lignes et de colonnes de chacun des nouveaux objets R avec la fonction `dim()`.
Pour réaliser une extraction en fonction d'un critère, il existe une fonction `subset()`.

```
In [34]: # Cette fonction extrait des données pour les sujets de sexe masculin codé 1 :
         Matable5 <- subset(Matable, Matable$CODSEX==1)

In [35]: # Cette commande extrait les données pour les femmes :
         Matable6 <- subset(Matable, Matable$SEXE=="Femme")

In [36]: # On peut également faire une extraction selon un critère avec `[]` :
         Matable7 <- Matable[Matable$POIDS<20,]

In [37]: # Cette commande crée une variable binaire codée (0 ou 1) qui prend la valeur 1 si le poids est supé
         Matable$POIDS20 <- ifelse(Matable$POIDS >= 20,1,0)

In [38]: # La fonction `transform()` permet d'ajouter des variables du tableau :
         Matable <- transform(Matable,POIDSENG=POIDS*1000)

In [39]: Matable <- transform(Matable,POIDS15=ifelse(POIDS<15,1,0))
         #cette commande ajoute une variable `POIDS15` qui prend la valeur 1 si le poids est inférieur à 15 e

In [40]: # La fonction `cut()`permet d'effectuer des regroupements par classe de variables quantitatives.
         Matable$GROUPOIDS <- cut(Matable$POIDS,breaks=c(9,15,20,25))
```

2.1.5 Factoriser les variables qualitatives ou discrètes :

Pour que des variables qualitatives ou discrètes soient reconnues comme telles par le logiciel R, il est nécessaire de les factoriser avec la fonction `factor()`. Pour cela, la fonction `str()` permet d'identifier les variables à factoriser.

```
In [35]: str(Matable)
```

```
'data.frame':        31 obs. of  8 variables:
 $ CODSEX   : int  1 1 0 0 0 0 1 1 1 0 ...
 $ SEXE     : Factor w/ 2 levels "Femme","Homme": 2 2 1 1 1 1 2 2 2 1 ...
 $ POIDS    : num  10 10.5 11 11.5 12 12.5 13 13.5 14 14.5 ...
 $ GROUPAGE : Factor w/ 3 levels "0:4","5:9",">10": 1 1 1 1 1 1 1 1 2 2 ...
 $ POIDS20  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ POIDSENG : num  10000 10500 11000 11500 12000 12500 13000 13500 14000 14500 ...
 $ POIDS15  : num  1 1 1 1 1 1 1 1 1 1 ...
 $ GROUPOIDS: Factor w/ 3 levels "(9,15]","(15,20]",..: 1 1 1 1 1 1 1 1 1 1 ...
```

1ère étape : définir un objet R contenant les numéros des variables.

```
In [41]: Indic <- c(1,5,7)
```

2ème étape : créer la boucle avec l'opérateur `for()` :

```
In [42]: for(i in 1:3) {Matable[,Indic[i]] <- factor(Matable[,Indic[i]])}

In [43]: str(Matable)
```

```
'data.frame':        31 obs. of  8 variables:
 $ CODSEX   : Factor w/ 2 levels "0","1": 1 2 1 1 2 2 1 1 2 2 ...
 $ SEXE     : Factor w/ 2 levels "Femme","Homme": 1 2 1 1 2 2 1 1 2 2 ...
 $ POIDS    : num  10 10.5 11 11.5 12 12.5 13 13.5 14 14.5 ...
 $ GROUPAGE : Factor w/ 3 levels "0:4","5:9",">10": 1 1 1 1 1 1 1 1 2 2 ...
 $ POIDS20  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
 $ POIDSENG : num  10000 10500 11000 11500 12000 12500 13000 13500 14000 14500 ...
 $ POIDS15  : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
 $ GROUPOIDS: Factor w/ 3 levels "(9,15]","(15,20]",..: 1 1 1 1 1 1 1 1 1 1 ...
```

2.2 Valeurs aberrantes et manquantes

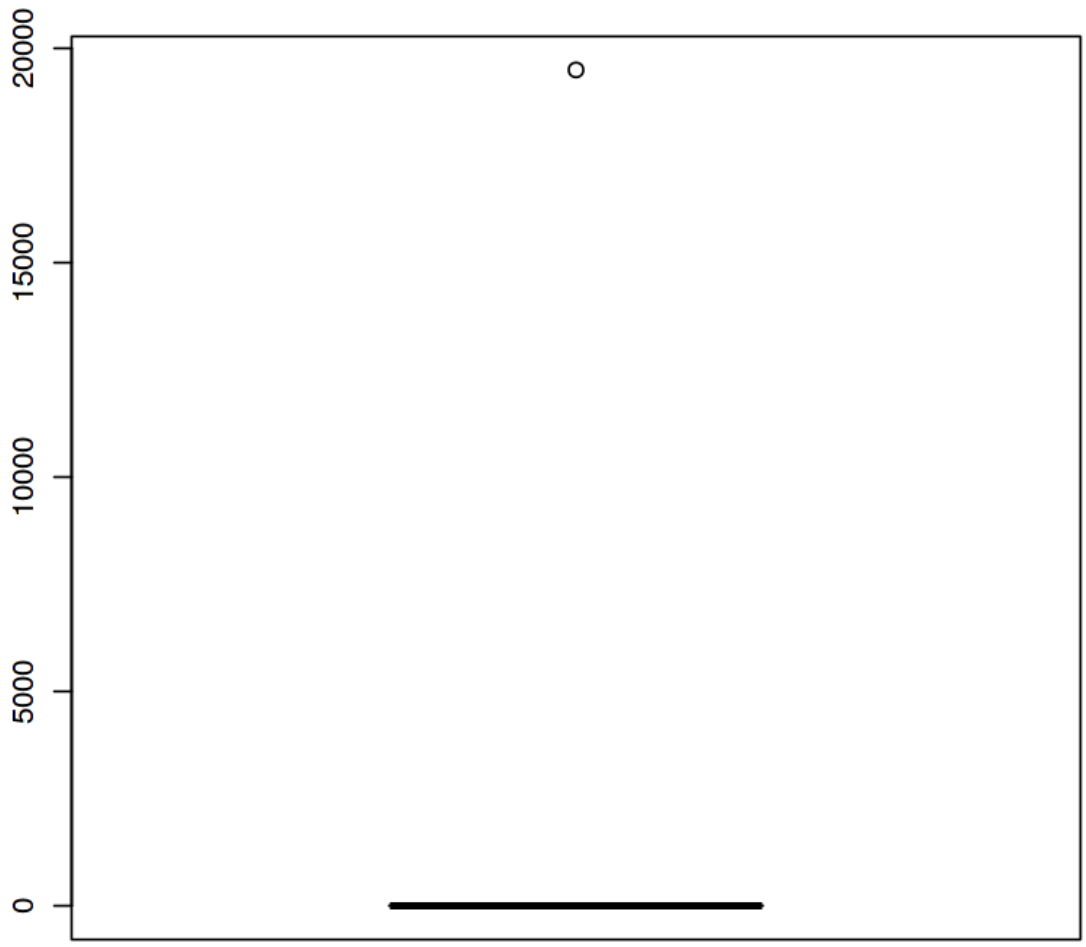
La commande **fix()** permet de corriger des données. la fonction **edit()** permet d'ouvrir les données dans une nouvelle fenêtre et de modifier les données.

2.2.1 Recherche de valeurs aberrantes :

Ici, on utilise la fonction `which()` pour identifier les coordonnées des points aberrants :

```
In [44]: # Créer une valeur aberrante :
         Matable[20,3] <- Matable[20,6]
         Result <- boxplot(Matable$POIDS)
```

Out[44]:




```
In [45]: # Les différents composants de `Result` :
         attributes(Result)
```

```
In [46]: which(Matable==20, arr.ind=T)
```

```
In [59]: #which(Matable > 24000, arr.ind=T)
```

2.2.2 Rechercher les valeurs manquantes :

```
In [48]: which(is.na(Matable), arr.ind=TRUE)
```

Corriger les données comme dans un tableur :

```
In [58]: #fix(Matable)

In [0]: Matable <- edit(Matable)
```

2.3 Lier deux tables de données entre elles :

```
In [50]: Matable10 <- rbind(Matable5, Matable6)

In [51]: Fievre <- rbinom(31, size=1, prob=0.25)
         Matable <- cbind(Matable, Fievre)

In [52]: names(Matable)[10] <- "FIEVRE"


        Error in names(Matable)[10] <- "FIEVRE": 'names' attribute [10] must be the same length as the vector
    Traceback:



In [53]: Toux <- rbinom(31, size=1, prob=0.28)
         Matable11 <- data.frame(Toux)
         row.names(Matable11) <- Ident[31:1]

In [54]: Matable <- merge(Matable, Matable11, by="row.names")

In [55]: names(Matable)[11] <- "TOUX"

In [57]: #View(Matable)
```

2.4 Exercice

Soit la matrice à coefficients réels suivante : $\begin{matrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 2 \\ 0 & 2 & 1 & 2 \\ 2 & 0 & 0 & 3 \\ 1 & 4 & 1 & 1 \\ 2 & 2 & 0 & 2 \end{matrix}$.

1. Créer l'objet Mat.1 correspondant à cette matrice ;
2. Créer l'objet Matable de type table de données à partir de l'objet Mat.1 ;
3. Renommer les colonnes avec les valeurs suivantes : SEXE, AGEJOUR, INFECTION, FRATRIE.
4. Renommer les lignes avec des valeurs allant de EPI1 à EPI6
5. Extraire la ligne correspondant à l'individu EPI4 ;
6. Afficher le contenu de la colonne nommée FRATRIE ;
7. Ajouter une colonne DECES ayant une valeur 0,0,1,1,0,0 ;
8. Ajouter une colonne SEXE2 qui prendra la valeur M si SEXE=1 et la valeur F si SEXE=2 (la valeur NP si SEXE=0).
9. Ajouter une colonne POIDS qui est égale à 2950+(AGE). Ajouter une colonne POIDSKG qui transforme la variable POIDS qui est exprimée en gramme.
10. Afficher le contenu de Matable dans une fenêtre différente de la console.

Chapitre 3

Statistiques Descriptives avec R

3.1 Généralités sur les types de variables

L'objectif de ce TP est de connaître les différents types de variables, ainsi que les paramètres de dispersion des variables qualitatives et des variables quantitatives. Il s'agira également de réaliser un tableau de fréquence. `Chapitre 4, Page 60 du poly de V. Richard`.

La collecte d'information nous amène à utiliser différents types de données classées en deux grands types :

Quantitatives	Qualitatives
Continues	Ordinales
Discrètes	Nominales
Temporelles	Binaires

1. Les variables **quantitatives continues** sont exprimées avec une unité de mesure (Poids exprimé en kg ou en g, Taille exprimée en m ou en cm, etc). La transformation d'une variable quantitative continue en classe s'appelle la **discrétisation** ou le **groupement par classe**.

2. Les variables **quantitatives discrètes** ne prennent que quelques valeurs entières dénombrables i.e. un nombre fini de valeurs. Par exemple, la Parité, le nombre d'enfants, l'Activité de soin, le nombre de consultations, etc.

3. Les variables **temporelles** sont des variables quantitatives qui sont exprimées en unité de mesure du temps. Par exemple, l'Age de grossesse (exprimé en semaines), Date de naissance (exprimé en jj/mm/aaaa), la Durée de la maladie, exprimée en jours, etc.

4. Les variables **qualitatives ordinales** sont des variables qui s'expriment en classes qui peuvent être ordonnées selon une échelle de valeurs. Par exemple, le Niveau des études (primaire, secondaire, universitaire), l'Intensité d'une douleur (faible, moyenne, intense), etc. Attention, ce ne sont pas des variables quantitatives discrètes (il ne faut pas les manipuler de façon arithmétique, mais utiliser la commande `factor()`).

5. Les variables **qualitatives nominales** n'ont pas besoin de hiérarchiser les classes, c'est-à-dire que leur ordre n'a pas d'importance. Par exemple, le groupe sanguin (A, B, AB, O), le statut marital (veuf, marié, divorcé, célibataire, en couple ou it's complicated), le type de toiture (paille, tôles, tuiles, etc).

6. Les variables **binaires** ou **dichotomiques** sont des variables nominales qui ne peuvent prendre que deux valeurs. Par exemple, le Sexe[1] (masculin ou féminin), Présence ou absence d'un signe clinique, d'une maladie, etc.

1. Cette variable a tendance à être remplacée par le Genre qui prend plus de deux valeurs ; voir les associations Mawjoudin et Shams qui militent en Tunisie pour les droits LGBT.

```
In [1]: Matable <- read.table("/home/user/.../Matable.csv",header=T,sep=";",dec=",")
```

3.2 Généralités sur les statistiques descriptives

Pour décrire les données, nous utilisons des indicateurs de **position** et des indicateurs de **dispersion**.

3.2.1 Description des variables quantitatives continues

Les paramètres de position sont la Médiane (`median()`, `summary()`), la Moyenne (`mean()`, `summary()`) et les Quartiles (`quantile()`, `summary()`).

Les paramètres de dispersion sont le Minimum et étendue (`max()`, `quantile()`, `summary()`), le Maximum et étendue (`min()`, `quantile()`, `summary()`), la Variance (`var()`), l'Ecart-type (`sd()`) et l'Intervalle de confiance (`t.test()`).

La médiane et les quartiles

La médiane est un paramètre de dispersion qui permet de résumer une dispersion [2] ; elle indique la valeur qui partage une série de données d'une variable quantitative en deux groupes d'effectifs égaux.

```
In [2]: median(Matable$Poids, na.rm=T)
```

Associés à la médiane, nous pouvons calculer les extrêmes, l'étendue et l'intervalle interquartile. Ce sont des paramètres de dispersion liés à la médiane.

Le minimum et le maximum sont les deux valeurs extrêmes de la distribution :

```
In [3]: min(Matable$Poids, na.rm=T)
```

```
In [4]: max(Matable$Poids, na.rm=T)
```

La différence entre les deux valeurs extrêmes est l'étendue. Attention au cas de valeurs aberrantes, dans ce cas, l'étendue donne une fausse image de la dispersion des valeurs d'une variable.

```
In [5]: Etendue <- max(Matable$Poids)-min(Matable$Poids)
```

Les quartiles sont trois paramètres de position qui partagent une série de données en quatre groupes d'effectifs égaux :

```
In [6]: quantile(Matable$Poids, na.rm=T)
```

La moyenne est la somme algébrique des valeurs observées divisée par le nombre de sujets. C'est un paramètre de tendance centrale qui sert à résumer une série de données d'une variable quantitative.

```
In [7]: mean(Matable$Poids, na.rm=T)
```

L'intervalle interquartile est la différence entre le premier et le troisième quartile :

```
In [8]: IntQuart <- quantile(Matable$Poids)[4]-quantile(Matable$Poids)[2]
        IntQuart
```

2. La médiane n'est pas utilisée pour les tests statistiques non-paramétriques.

La variance et l'écart-type

La variance est le meilleur indicateur de la dispersion autour de la moyenne. Elle utilise toutes les valeurs de la distribution et elle résume l'ensemble des écarts de chaque valeur d'une distribution par rapport à la moyenne :

$$\sigma^2 = \frac{\sum(x_i - \bar{x})^2}{N}$$

```
In [9]: var(Matable$Poids, na.rm=T)
```

Plus cette valeur est faible, plus la dispersion est minimale. Plus la valeur de la variance est élevée, plus la dispersion est importante.

La racine carré de la variance est appelé l'écart-type (il traduit le fait d'avoir utilisé le carré pour le calcul de la variance). L'écart-type mesure l'écart par rapport à la moyenne, qui varie dans le même sens de la variance et s'exprime dans la même unité que la moyenne.

```
In [10]: sd(Matable$Poids, na.rm=T)
```

L'intervalle de confiance de la moyenne mesure l'intervalle où se trouve la moyenne d'une mesure d'une population, à partir des données d'un échantillon avec un risque d'erreur de 5%.

```
In [11]: t.test(Matable$Poids, conf.level=0.95)$conf.int
```

Il y a donc cinq chances sur cent que la moyenne de la population se trouve à l'extérieur de cet intervalle.

Pour résumer, la fonction `summary()` permet d'obtenir les extrêmes, les quartiles, la médiane et la moyenne d'une variable quantitative continue. Cette fonction ne tient pas compte des valeurs manquantes (il n'est donc pas nécessaire d'utiliser le paramètre `na.rm=T` avec cette fonction).

```
In [12]: summary(Matable$Poids)
```

```
Out[12]:    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
          10.00   13.75   17.50   17.50   21.25   25.00
```

3.2.2 Description des variables qualitatives et quantitatives discrètes

Pour les données regroupées en classe, nous utilisons habituellement les tableaux de fréquence pour présenter les résultats. La construction de ces tableaux nécessite de connaître les effectifs dans chacune des classes de regroupement et également les pourcentages de ces effectifs rapportés à l'effectif global observé.

```
In [13]: head(Matable)
```

La description des données regroupées passe dans un premier temps par la description des effectifs par calsse : pour déterminer les Effectifs, on utilise la commande `table()`. Si on cherche la proportion de la population observée dans chacune des classes de la variable, nous cherchons la Fréquence relative avec la commande `prop.table()`. Les effectifs totaux peuvent être différents du nombre d'enregistrements du fait des valeurs manquantes : pour les Effectifs totaux nous utilisons la commande `margin.table()`. Le calcul des effectifs cumulés se fait avec la commande `cumsum()` appliquée au résultat de la fonction `table()`.

Si les variables sont reconnues par `R` comme des variables factorisées, nous pouvons utiliser la fonction `summary()`. Si les variables ne sont pas factorisées, nous utilisons la fonction `as.factor()` [3] avec `summary()` (ou `table()`) :

```
In [14]: summary(as.factor(Matable$Sexe))
```

3. La fonction `as.factor()` permet à `R` de factoriser une variable uniquement au moment de l'utilisation de cette commande. La variable n'est pas alors factorisée de façon définitive comme avec `factor()`.

```
In [15]: table(Matable$Tranchage)
```

```
Out[15]:
         0:4 5:9 >10
           8   9  14
```

```
In [16]: summary(Matable$Sexe1)
```

La fréquence relative s'exprime sous la forme d'un pourcentage avec un chiffre après la virgule. Nous pouvons utiliser la fonction `round()` et l'argument 1 associé (pour le nombre de chiffres après la virgule).

```
In [17]: Table1 <- table(Matable$Tranchage)
         prop.table(Table1)
```

```
Out[17]:
               0:4       5:9       >10
         0.2580645 0.2903226 0.4516129
```

```
In [19]: round(prop.table(Table1)*100, 1)
```

```
Out[19]:
          0:4  5:9  >10
         25.8 29.0 45.2
```

Les effectifs totaux sont calculés par :

```
In [21]: margin.table(Table1)
```

Le calcul des sommes cumulées doit suivre une certaine cohérence. Il est par exemple nécessaire, dans le cas de groupes d'âge, de classer les résultats dans un sens décroissant (car les personnes âgés sont passées par les tranches d'âge les plus jeunes). Nous utilisons alors la sélection avec l'opérateur d'extraction [] :

```
In [22]: Table1 <- table(Matable$Tranchage)
         cumsum(Table1[c(1,3,2)])
```

Ainsi, avec l'ensemble des commandes à notre disposition, nous pouvons construire un tableau des fréquences :

```
In [23]: Variable <- Matable$Tranchage
         Effectifs <- table(Variable)
         Proportion <- round(prop.table(Effectifs)*100, 1)
         E.cumul <- cumsum(Effectifs[c(1,3,2)])
         P.cumul <- cumsum(Proportion[c(1,3,2)])
```

```
In [24]: Age1 <- c(Effectifs[2], Proportion[2], E.cumul[3], P.cumul[3])
         Age2 <- c(Effectifs[3], Proportion[3], E.cumul[2], P.cumul[2])
         Age3 <- c(Effectifs[1], Proportion[1], E.cumul[1], P.cumul[1])
```

```
In [25]: Age1
```

```
In [26]: TableAge <- rbind(Age1, Age2, Age3)
         colnames(TableAge) <- c("Eff", "%", "Eff Cum", "(% cum)")
         rownames(TableAge) <- c("0-4 ans", "5-9 ans", ">=10ans")
         TableAge
```

L'intervalle de confiance d'une proportion (ou d'un pourcentage) sera calculé pour une variable qualitative binaire. Nous utilisons la fonction `prop.test()`. Il s'agit de l'intervalle de confiance de la première valeur de la variable. La fonction `prop.table()` permet de visualiser les pourcentages de chacune

```
In [27]: prop.test(table(Matable$Sexe))$conf.int
```

3.3 Exercices d'application

3.3.1 Variables quantitatives

1. Créer un objet appelé Mesure1 comprenant 30 mesures répondant à une loi de distribution uniforme avec un minimum à 20 et un maximum à 35.
2. Calculer la médiane, la moyenne, l'écart-type de cet objet
3. Calculer l'intervalle de confiance de la moyenne
4. Créer un objet appelé Mesure2 comprenant 100 mesures répondant à une loi de distribution uniforme avec un minimum à 20 et un maximum à 35 ;
5. Calculer la médiane, la moyenne, l'écart-type de cet objet
6. Calculer l'intervalle de confiance de la moyenne

```
In [28]: # Création de l'objet Mesure1
         Mesure1 <- runif(30,20,35)

In [29]: # Calculer la médiane et la moyenne
         summary(Mesure1)

Out[29]:    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
          20.19   26.58   29.46   28.99   32.22   34.38

In [30]: # Calculer l'écart-type
         sd(Mesure1)

In [31]: # Calculer l'intervalle de confiance de la moyenne
         t.test(Mesure1)$conf.int

In [32]: # Création de l'objet Mesure2
         Mesure2 <- runif(100,20,35)

In [33]: # Calculer la médiane et la moyenne
         summary(Mesure2)

Out[33]:    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
          20.16   22.84   26.93   27.01   30.65   34.98

In [34]: # Calculer l'écart-type
         sd(Mesure2)

In [35]: # Calculer l'intervalle de confiance de la moyenne
         t.test(Mesure2)$conf.int
```

3.3.2 Variables qualitatives

1. Créer un objet appelé Caracter1 qui suit une loi binomiale de taille 1 et de 0.15, comprenant 110 valeurs ;
2. Créer un objet appelé Caracter2 qui suit une loi binomiale de taille 2 et de 0.40, comprenant 100 valeurs ;
3. Calculer les effectifs et les pourcentages par classe de chacun des deux objets ;
4. Dresser les tableaux présentant les effectifs associés aux pourcentages pour chacun des deux objets.

```
In [36]: # Création de l'objet Caracter1
         Caracter1 <- rbinom(110, 1, 0.15)

In [37]: # Création de l'objet Caracter2
         Caracter2 <- rbinom(110, 2, 0.40)
```

```
In [38]: # Calcul des effectifs de Caracter1
         A1 <- table(Caracter1)
         A1

Out[38]: Caracter1
          0  1
         96 14

In [39]: # Calcul des pourcentages de Caracter1
         P1 <- round(prop.table(A1)*100, 0)
         P1

Out[39]: Caracter1
          0  1
         87 13

In [40]: # Table pour Caracter1
         Matable1 <- cbind(A1[1], P1[1], A1[2], P1[2], margin.table(A1), 100)
         colnames(Matable1) <- c("Eff1", "%", "Eff2", "%", "Total", "%")
         rownames(Matable1) <-""
         Matable1

In [41]: # Idem pour les effectifs de Caracter2
```

Chapitre 4

Représentations graphiques

Nous allons commencer par installer une collection de fonctions, données et documentations qui enrichissent les fonctions de base de R. Ce matériel est disponible et facilement téléchargeable via la commande `install.packages("tidyverse")` [1].

4.1 Présentation graphique des données

Nous nous concentrons sur les objectifs suivants : connaître les différents types de graphiques, ajouter des informations complémentaires sur un graphique, insérer plusieurs graphiques dans une même fenêtre puis sauvegarder les graphiques sous différents formats.

4.1.1 Diagramme en barre

Importer les packages utiles à la représentation graphique et autres bases de données utiles :

```
In [1]: library(tidyverse)
```

```
Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyr
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr
Conflicts with tidy packages ----------------------------------------------------
filter(): dplyr, stats
lag():    dplyr, stats
```

Création de la table de données :

```
In [2]: set.seed(3)
        Sexe <- rbinom(31, size=1, prob=.48)
        Sexe1 <- ifelse(Sexe==0,"Femme","Homme")
        Tranchage <- rep(c("0:4","5:9",">10"),c(8,9,14))
        Poids <- seq(10,25,by=0.5)
        Matable <- data.frame(Sexe,Sexe1,Poids,Tranchage)
        Matable$GROUPOIDS <- cut(Matable$Poids,breaks=c(9,15,20,25))
        head(Matable)
```

1. Utiliser aussi `tidyverse_update()` pour être sûr que c'est la dernière version utilisée.

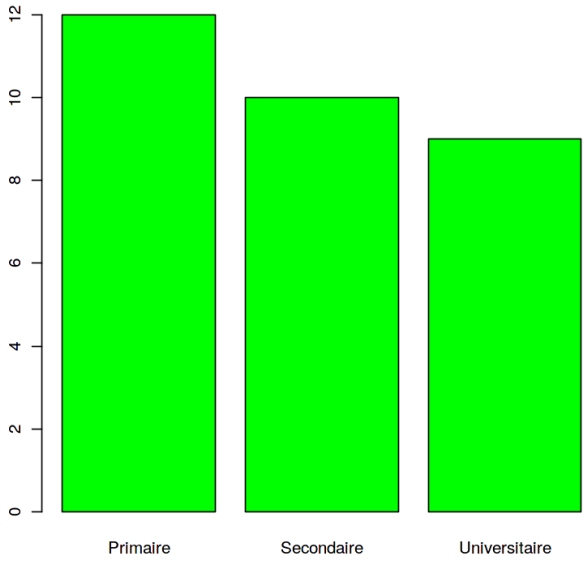
Créer une nouvelle variable Niveau scolaire de la mère, de type *factor* ou *facteur* :

```
In [3]: Matable$NIVSCOLMERE <- factor(c(rep("Primaire",12), rep("Secondaire",10), rep("Universitaire",9)))
```

Ce type de graphique permet de réaliser une représentation sur une seule coordonnée des variables **qualitatives ordinales** ou **nominales**. Nous utiliserons la fonction `plot()` :

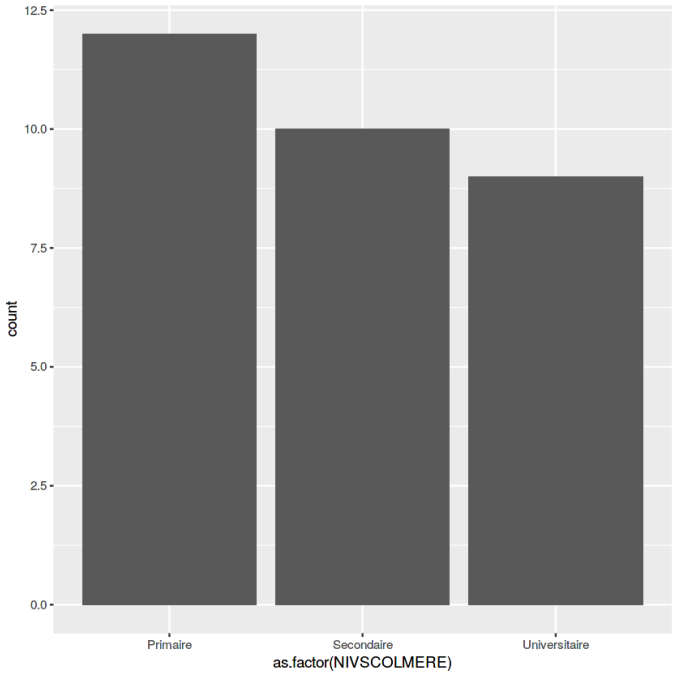
```
In [4]: plot(Matable$NIVSCOLMERE, col='green')
```

Out[4]:



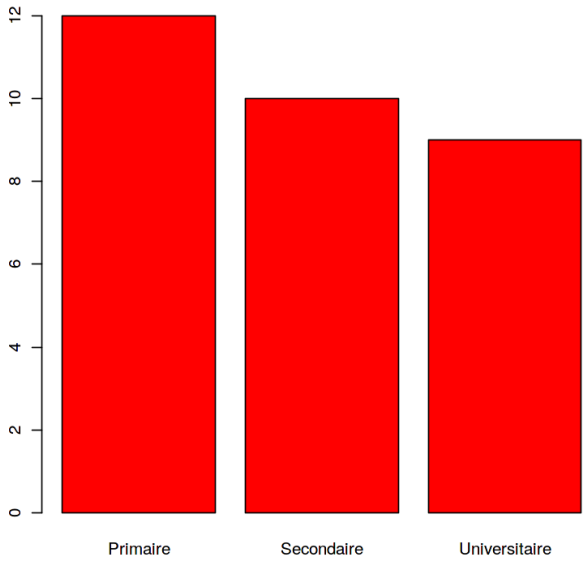

```
In [5]: ggplot(Matable, aes(x=as.factor(NIVSCOLMERE))) + geom_bar()
```

Out[5]:

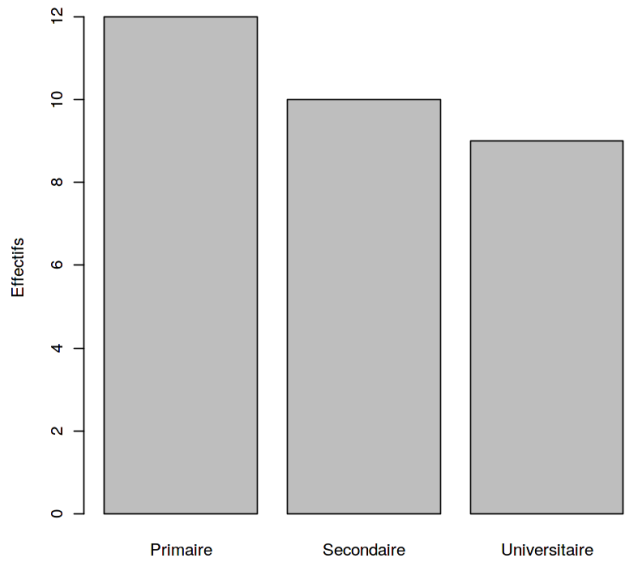


Il est possible d'ajouter des informations sur un graphique. Le paramètre *ylab* va permettre d'ajouter un titre sur l'axe des ordonnées ; le paramètre ylim permet de modifier les valeurs de l'axe des ordonnées. Le paramètre main permet d'insérer un titre au dessus du graphique[2].

```
In [6]: plot(Matable$NIVSCOLMERE, col='red')
```

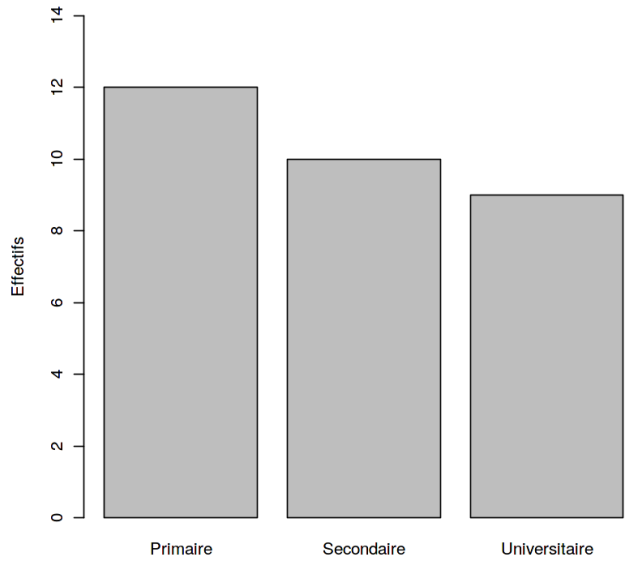



```
In [7]: plot(Matable$NIVSCOLMERE, ylab='Effectifs')
```

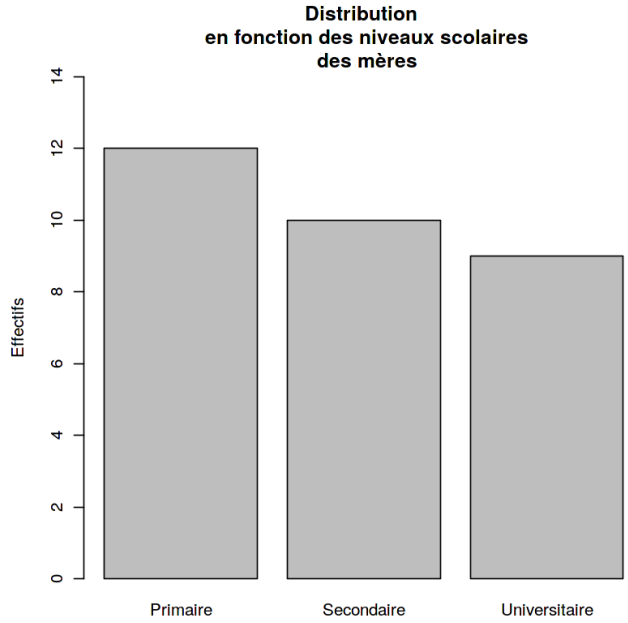


2. L'expression \n permet un retour à la ligne dans une chaîne de caractères.

```
In [8]: plot(Matable$NIVSCOLMERE, ylab='Effectifs', ylim=c(0,14))
```

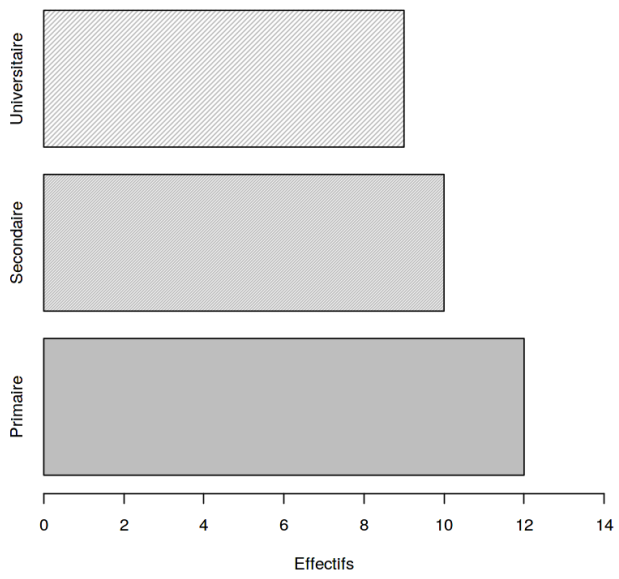



```
In [9]: plot(Matable$NIVSCOLMERE, ylab='Effectifs', ylim=c(0,14), main='Distribution \n en fonction ... des m
```



Le diagramme en barre horizontale répond à la présentation des variables de même type que le diagramme en barre. Les paramètres **horiz=T** et **density=c(NA,50,30)** ont permis pour le premier de présenter les barres à l'horizontale et pour le second de faire varier la densité de la couleur avec un format hachuré. Les commandes **ylim** et **ylab** ont été remplacés par **xlim** et **xlab**.

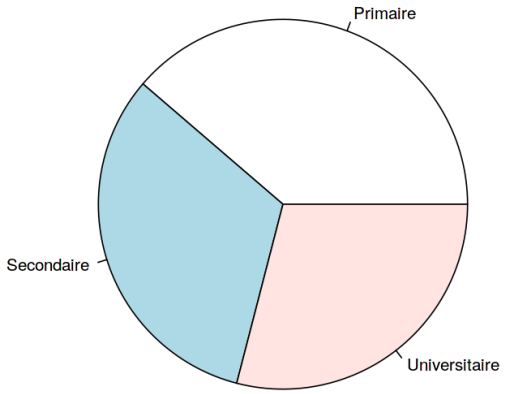
```
In [10]: plot(Matable$NIVSCOLMERE, density=c(NA,50,30), xlab='Effectifs', xlim=c(0,14), horiz=T)
```



4.1.2 Camembert

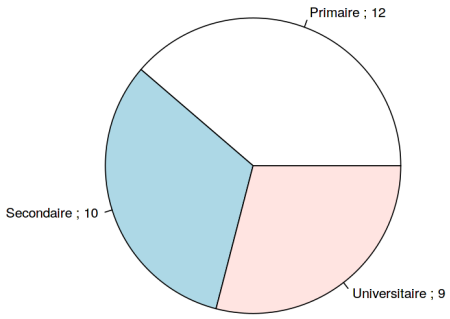
Ce graphique permet la présentation des variables de même type que les diagrammes en barre. Vous utiliserez la fonction **pie** (Ne s'applique pas sur des données brutes).

```
In [11]: pie(table(Matable$NIVSCOLMERE))
```



Il est possible d'ajouter sur ce graphique les effectifs de chacune des classes avec le paramètre **labels** :

```
In [12]: Nom <- levels(Matable$NIVSCOLMERE)
         Donnee <- table(Matable$NIVSCOLMERE)
         pie(Donnee, labels=c(paste(Nom[1],';',Donnee[1]), paste(Nom[2],';',Donnee[2]), paste(Nom[3],';',Donn
```

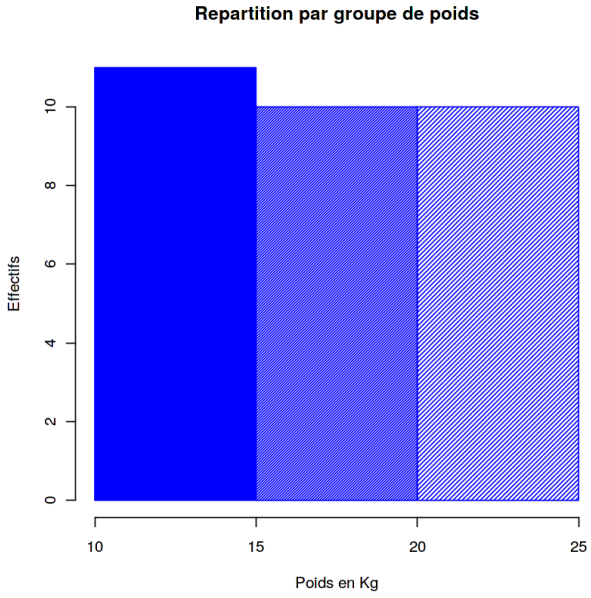


La fonction **paste()** est utilisée ici pour concaténer des chaînes de caractères alpha-numériques. Ell est urtilisée avec le paramètre **labels** pour ajouter une information sur les effectifs séparés du nom de la classe par un point-virgule.

4.1.3 Histogramme

Il est utilisé pour représenter des données quantitatives discrétisées (en classes). Vous utiliserez la fonction **hist()** pour le réaliser.

```
In [13]: Title <- 'Repartition par groupe de poids'
         hist(Matable$Poids, breaks=c(10,15,20,25), density=c(NA, 50, 30),
              col='blue', main=Title, xlab='Poids en Kg', ylab='Effectifs')
```

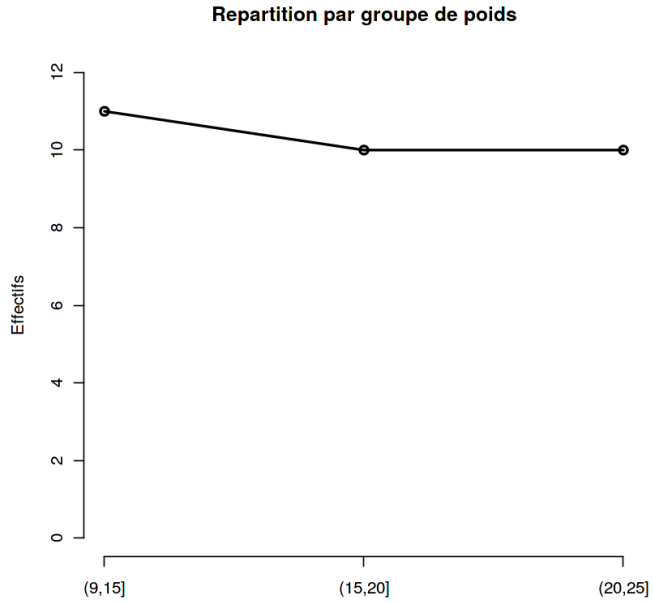


Le paramètre **breaks** permet de donner les limites de chaque classe. Pour cette commande, nous avons utiliser les données brutes de la variable Poids.

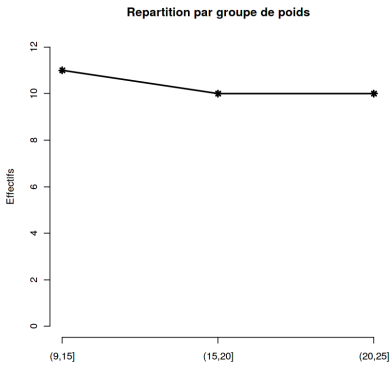
4.1.4 Polygone des fréquences

Il s'agit d'une courbe réunissant le sommet de chacune des classes d'une variable quantitative discrétisée. Vous utiliserez la fonction **plot()** avec des données agrégées (fonction **table()**) et l'argument **type** en fonction du type de courbe que vous souhaitez obtenir.

```
In [17]: Donnee <- table(Matable$GROUPOIDS)
         plot(Donnee, type='o', main=Title, ylab='Effectifs', ylim=c(0,12))
```



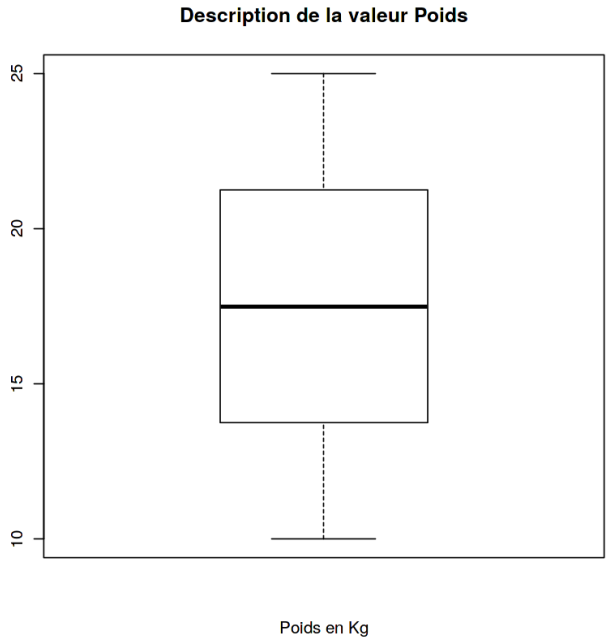

```
In [19]: plot(Donnee, type='o', main=Title, ylab='Effectifs', ylim=c(0,12), pch=8)
```



4.1.5 Boîte à moustache

La boîte à moustache permet de représenter sur un graphique les paramètres de position et de dispersion d'une variable quantitative continue : médiane, quartiles et étendue. Vous utiliserez la fonction **boxplot()** pour réaliser ce graphique.

```
In [20]: boxplot(Matable$Poids, xlab='Poids en Kg', main='Description de la valeur Poids')
```



La fonction **boxplot()** permet de retrouver les valeurs associées à chacun des paramètres de ce graphique. Les moustaches inférieures et supérieures sont calculées par le retrait à la valeur du 1er quartile ou l'ajout à la valeur du 3ème quartile de 1.5 fois l'intervalle interquartile.

Pour ajouter la valeur de la moyenne sur le graphique, il y a la commande **points()**.

```
In [23]: #points(x=1, y=mean(Matable$Poids), pch=4)
```

4.1.6 Afficher des graphiques

```
In [24]: par(mfrow=c(3,2))
         layout.show(6)
         Zone <- matrix(c(2,4,1,1,3,5), byrow=T, ncol=2)
         layout(Zone)
         layout.show(Zone)
```

4.1.7 Sauvegarder un graphique

```
In [25]: graphics.off()
         pdf(file='figure.pdf')
         Nom <- levels(Matable$NIVSCOLMERE)
         Donnee <- table(Matable$NIVSCOLMERE)
         pie(Donnee, labels=c(paste(Nom[1],';',Donnee[1]), paste(Nom[2],';',Donnee[2]), paste(Nom[3],';',Donn
         dev.off()
```

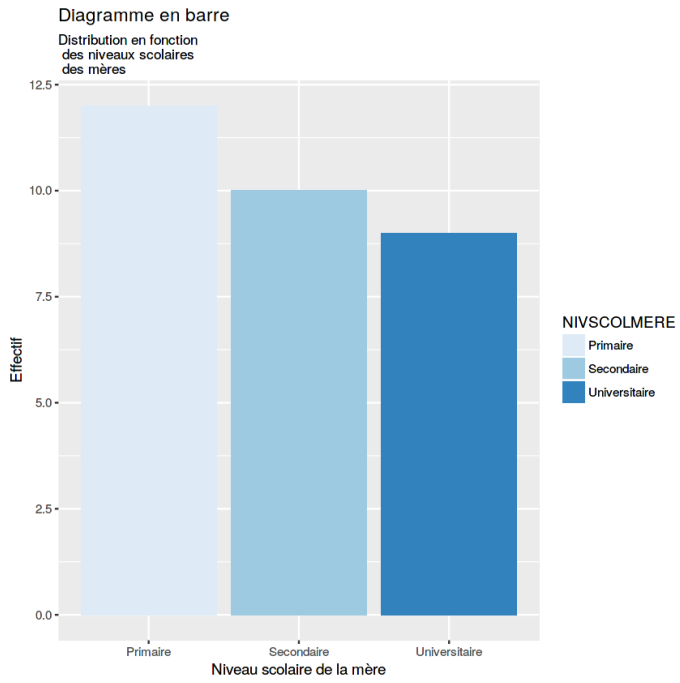
4.1.8 Exercice sur ggplot2

Reprendre les graphiques précédents et tout convertir sur ggplot2.

In [1]:
```
set.seed(3)
Sexe <- rbinom(31, size=1, prob=.48)
Sexe1 <- ifelse(Sexe==0,"Femme","Homme")
Tranchage <- rep(c("0:4","5:9",">10"),c(8,9,14))
Poids <- seq(10,25,by=0.5)
Matable <- data.frame(Sexe,Sexe1,Poids,Tranchage)
Matable$GROUPOIDS <- cut(Matable$Poids,breaks=c(9,15,20,25))

Matable$NIVSCOLMERE <- factor(c(rep("Primaire",12), rep("Secondaire",10), rep("Universitaire",9)))
#View(Matable)
library(tidyverse)
```

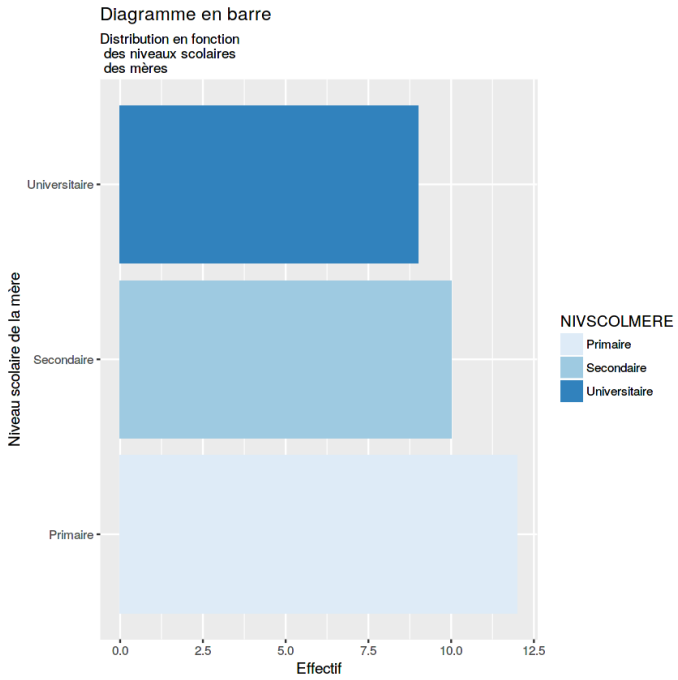
In [2]:
```
p <- ggplot(Matable, aes(NIVSCOLMERE))
p <- p + geom_bar(aes(fill=NIVSCOLMERE))
p <- p + scale_fill_brewer(palette="Blues")
p <- p + labs(x="Niveau scolaire de la mère", y="Effectif", title="Diagramme en barre",
     subtitle="Distribution en fonction \n des niveaux scolaires\n des mères")
p
```

Out[2]:



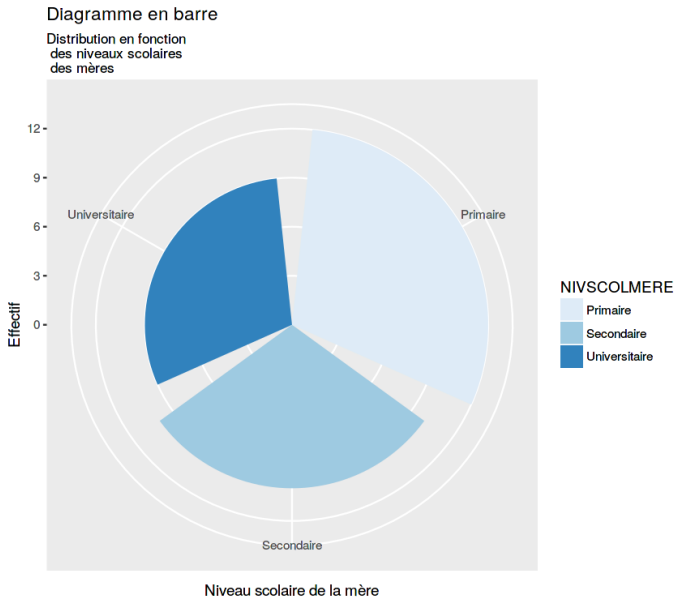
In [3]:
```
p + coord_flip()
```

Out[3]:



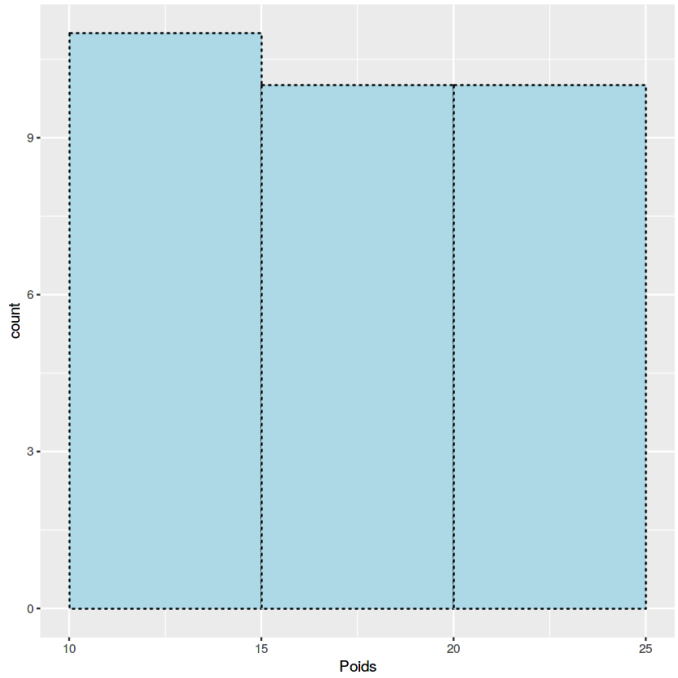
In [4]: p + coord_polar(theta="x", direction=1)

Out[4]:



In [5]: q <- ggplot(Matable, aes(Poids))
q + geom_histogram(breaks=c(10,15,20,25), color="black", fill="lightblue", linetype="dashed")

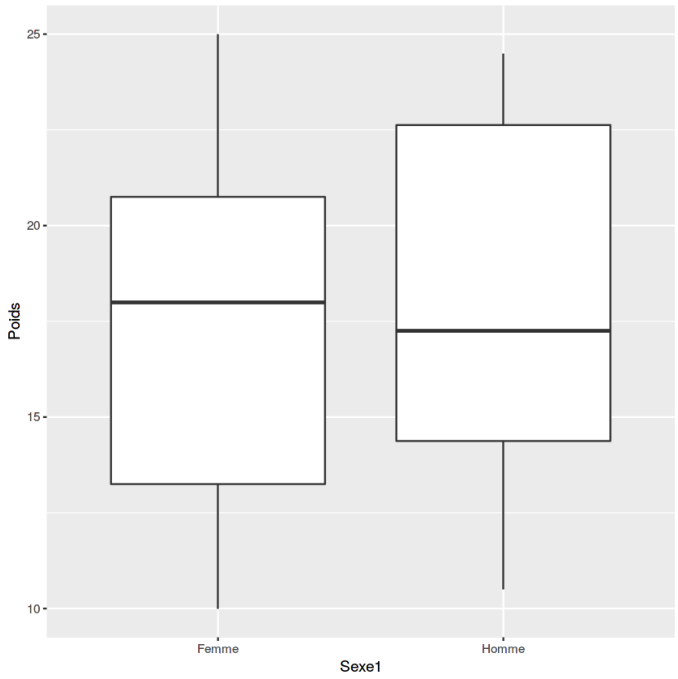
Out[5]:



In [6]:
```
#last_plot()
#ggsave("plot.png", width = 5, height = 5)
```

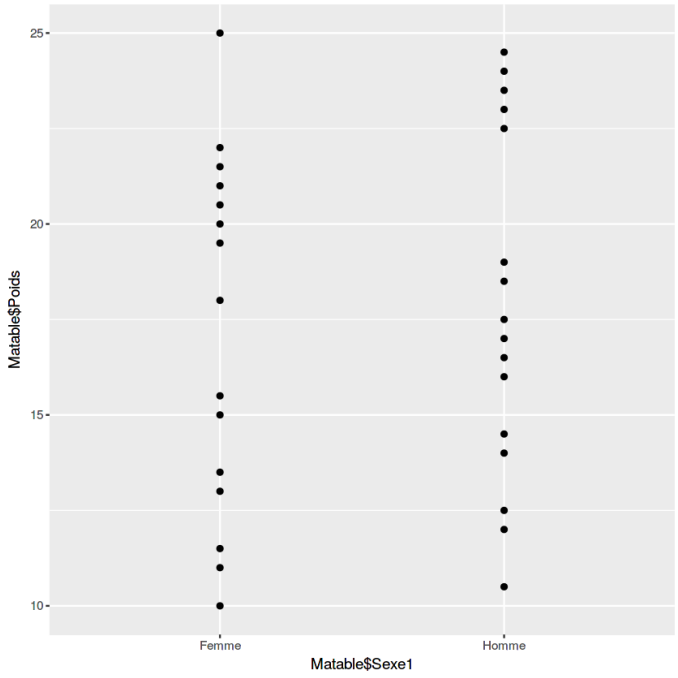
In [8]:
```
ggplot(Matable, aes(Sexe1, Poids)) + geom_boxplot()
```

Out[8]:



```
In [9]: qplot(Matable$Sexe1, Matable$Poids, geom="point")
```

Out[9]:



4.2 Analyses univariées

Dans cette partie, nous allons décrire une variable quantitative en fonction de classes de variables qualitatives. Puis, décrire une variable qualitative en fonction d'une variable qualitative ? Ensuite, nous allons réaliser des tableaux de contingences et enfin, réaliser des graphiques par classe.

4.2.1 Variables quantitatives en fonction de variables qualitatives

Après avoir décrit la distribution générale d'une variable, l'étape suivante vous conduit à l'analyse univariée pour décrire des variables en fonction de sous-groupes. Vous utiliserez les fonctions **by()**, **tapply()** ou **aggregate()** pour réaliser ces analyses des variables quantitatives en fonction des classes d'une variable qualitative.

```
In [26]: by(Matable$Poids, Matable$Sexe, summary)
         tapply(Matable$Poids, list(Matable$Sexe), summary)
         aggregate(Matable$Poids, list(Matable$Sexe), summary)
         by(Matable$Poids, list(Matable$Sexe, Matable$GROUPOIDS), summary)
         aggregate(Matable$Poids, list(Matable$Sexe, Matable$GROUPOIDS), summary)
         tapply(Matable$Poids, list(Matable$Sexe, Matable$GROUPOIDS), summary)
```

```
Out[26]: Matable$Sexe: 0
            Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
           10.00   13.25   18.00   17.13   20.75   25.00
```

```
------------------------------------------------------------
Matable$Sexe: 1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.50   14.38   17.25   17.84   22.62   24.50
```

Out[26]:
```
$`0`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.00   13.25   18.00   17.13   20.75   25.00

$`1`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.50   14.38   17.25   17.84   22.62   24.50
```

Out[26]:
```
: 0
: (9,15]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.00   11.12   12.25   12.33   13.38   15.00
------------------------------------------------------------
: 1
: (9,15]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   10.5    12.0    12.5    12.7    14.0    14.5
------------------------------------------------------------
: 0
: (15,20]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  15.50   17.38   18.75   18.25   19.62   20.00
------------------------------------------------------------
: 1
: (15,20]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  16.00   16.62   17.25   17.42   18.25   19.00
------------------------------------------------------------
: 0
: (20,25]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   20.5    21.0    21.5    22.0    22.0    25.0
------------------------------------------------------------
: 1
: (20,25]
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   22.5    23.0    23.5    23.5    24.0    24.5
```

Nous utilisons les mêmes commandes pour effectuer des analyses descriptives à partir de plusieurs sous groupes décomposés de plusieurs variables qualitatives. Avec la commande **tapply()** vous ne pourrez pas utiliser la commande **summary** mais les autres commandes telles que **mean**, **median**, **var**, **sd**, **min**, **max**.

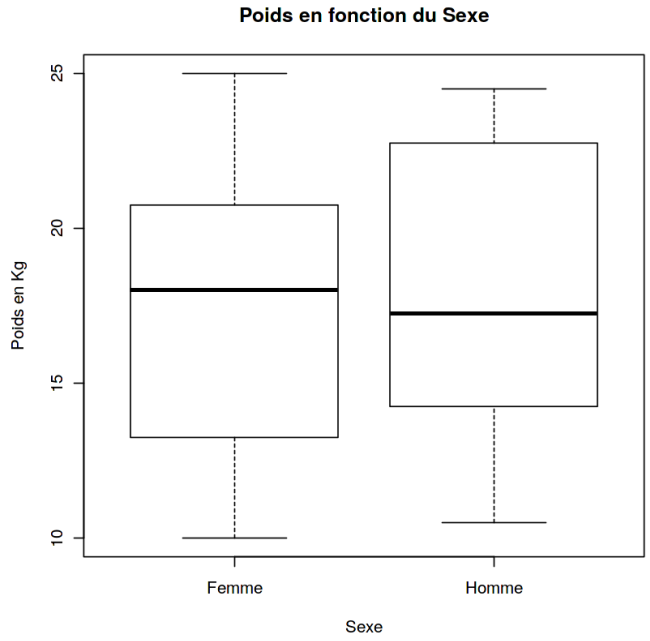
Notez qu'il faut ajouter **na.rm=TRUE** avec les fonctions autres que summary, si vous n'êtes pas sûr que votre fichier ne contient pas de donénes manquantes. Pour éviter toute surprise et erreur dans votre programmation, prenez l'habitude d'ajouter ce paramètre.

4.2.2 Graphiques en classe de variables quantitatives

Vous pouvez présenter les données sous forme de graphique en fonction des classes :

```
boxplot(Matable$Poids~Matable$Sexe1, xlab='Sexe', ylab='Poids en Kg', main='Poids en fonction du Sexe')
```

Out[27]:



Vous pouvez ajouter la valeur de la moyenne sur le graphique avec la fonction **points()**.

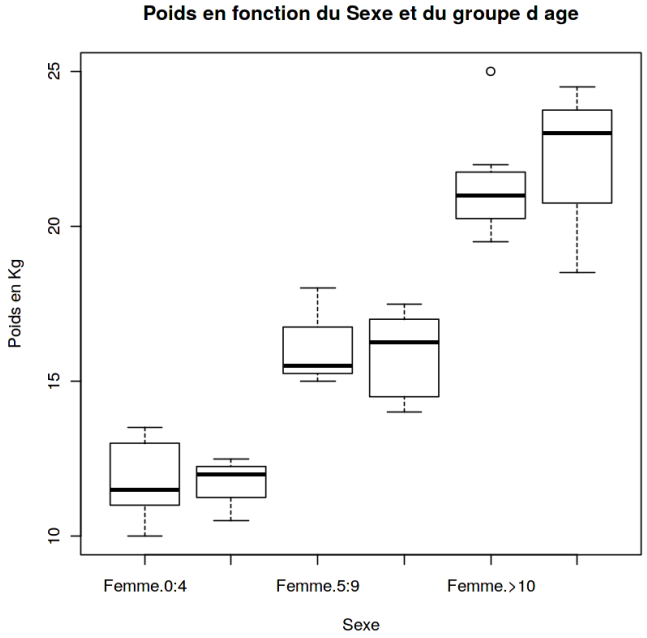
```
In [28]: #points(1, mean(Matable$Poids[Matable$Sexe1=="Femme"]), pch=4)
```

```
In [29]: #points(2, mean(Matable$Poids[Matable$Sexe1=="Homme"]), pch=4)
```

Mais aussi en fonction de plusieurs sous classes.

```
In [30]: boxplot(Matable$Poids~Matable$Sexe1+Matable$Tranchage, xlab='Sexe',
                ylab='Poids en Kg', main='Poids en fonction du Sexe et du groupe d age')
```

Out[30]:



4.2.3 Variables qualitatives en fonction d'une autre variable qualitatives

Comme pour les variables quantitatives, il est possible d'avoir une description d'une variable qualitative (effectifs et pourcentages) en fonction d'une variable qualitative. Vous utiliserez les fonctions **table()** et **prop.table()** pour décrire la distribution d'une variable qualitative.

```
In [31]: table(Matable$Sexe1, Matable$NIVSCOLMERE)
         Tab.1 <- table(Matable$Sexe1, Matable$NIVSCOLMERE)
         prop.table(Tab.1, 1)
         round(prop.table(Tab.1,1)*100, 1)
         round(prop.table(Tab.1,2)*100, 1)
```

```
Out[31]:
        Primaire Secondaire Universitaire
Femme          7          4             4
Homme          5          6             5
```

```
Out[31]:
        Primaire Secondaire Universitaire
Femme 0.4666667  0.2666667     0.2666667
Homme 0.3125000  0.3750000     0.3125000
```

```
Out[31]:
        Primaire Secondaire Universitaire
Femme       46.7       26.7          26.7
Homme       31.2       37.5          31.2
```

```
Out[31]:
        Primaire Secondaire Universitaire
Femme       58.3       40.0          44.4
Homme       41.7       60.0          55.6
```

Vous pouvez créer un tableau de contingence pour réunir l'ensemble de l'information concernant les effectifs et les pourcentages

```
In [32]: Var.1 <- Matable$Sexe1
         Var.2 <- Matable$NIVSCOLMERE
         a  <- table(Var.1, Var.2)
         b <- round(prop.table(a,1)*100, digit=1)
```

Nous utilisons des fonctions descriptives et nous construisons la table de contingence :

```
In [33]: Total.1 <- margin.table(a,2)

         Counting.1 <- rbind(a[1,],b[1,], a[2,],b[2,])
         dimnames(Counting.1) <- list(c('Male','%','Femelle','%'), c('Prim', 'Secon', 'Universit'))
         Counting <- rbind(Counting.1, Total.1)
         Total.1 <- margin.table(Counting.1, 1)
         Counting.1 <- cbind(Counting.1, Total=Total.1)
         Counting.1
```

Pour avoir les pourcentages à coté des effectifs, vous pourrez utiliser le programme suivant à insérer après l'objet **b** du programme précédent :

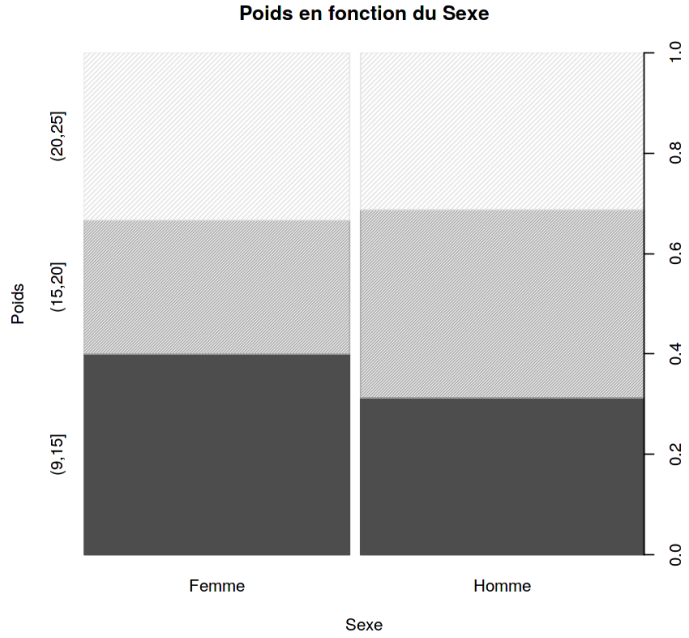
```
In [34]: Total.2 <- margin.table(a,1)
         Counting.2 <- cbind(a[,1],b[,1], a[,2],b[,2], a[,3],b[,3])
         dimnames(Counting.2) <- list(c('Male','Femelle'), c('Prim','%', 'Secon','%', 'Universit','%'))
         Counting.2 <- cbind(Counting.2, Total=Total.2)
         Total.2 <- margin.table(Counting.2, 2)
         Total.2[2] <- round((Total.2[1]/Total.2[7])*100, 1)
         Total.2[4] <- round((Total.2[3]/Total.2[7])*100, 1)
         Total.2[6] <- round((Total.2[5]/Total.2[7])*100, 1)
         Counting.2 <- rbind(Counting.2, Total=Total.2)
         Counting.2
```

4.2.4 Graphiques en classe de variables qualitatives

Vous utiliserez les fonctions **plot()** ou **barplot()** avec des arguments pour identifier la variable de factorisation.

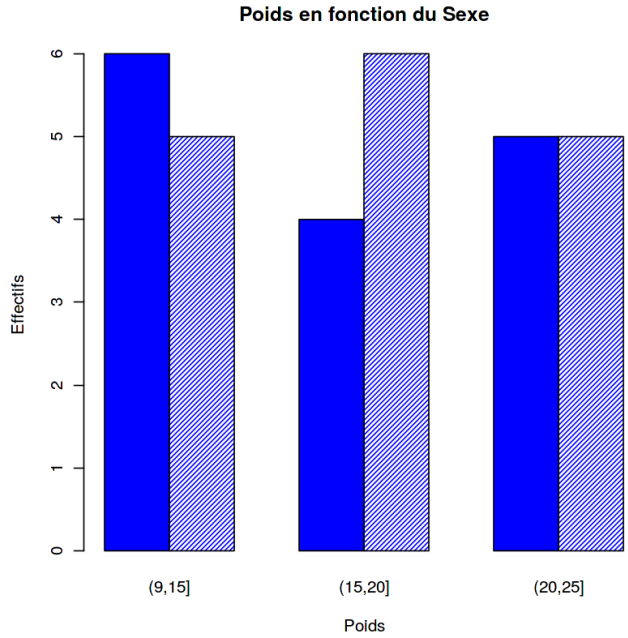
```
In [35]: plot(Matable$GROUPOIDS~Matable$Sexe1, density=c(NA,50,30), xlab='Sexe', ylab='Poids', main='Poids en
```

```
Out[35]:
```



In [36]:
```
barplot(table(Matable$Sexe1,Matable$GROUPOIDS), col='blue',
        density=c(NA,30), xlab='Poids', ylab='Effectifs', main='Poids en fonction du Sexe', beside=T
```

Out[36]:



In [38]:
```
#legend(7,6, legend=c("Feminin", "Masculin"), bty='n', fill='red', density=c(NA,30), cex=0.7)
```

4.2.5 Exercices sur ggplot2

ggplot2

Tout reprendre avec les commandes de ggplot2.

Tableau de contingence

Créer un vecteur appelé Sexe qui suit une loi binomiale de taille 1 et de 0.55, comprenant 120 valeurs. Créer un vecteur appelé NivScol qui suit une loi binomiale de taille 2 et de 0.40, comprenant 120 valeurs. Dresser le tableau de contingence de croisement du niveau scolaire en fonction du sexe.

Faire un graphique

Créer un vecteur appelé Poids contenant 120 valeurs qui suit une loi normale de moyenne 2.8kg et d'écart type 0.08. Créer un vecteur appelé Sexe qui suit une loi binomiale de taille 1 et de probabilité 0.55, comprenant 120 valeurs. Recodez la variable Sexe, qui prendra la valeur Masculin si la valeur est 0, Feminin si la valeur est 1. Donner la moyenne et la médiane du poids en fonction du sexe. Faire un graphique de la variable Poids en fonction du sexe. N'oubliez pas de donner un titre et des légendes au niveau des axes. Y insérer la valeur de la moyenne.

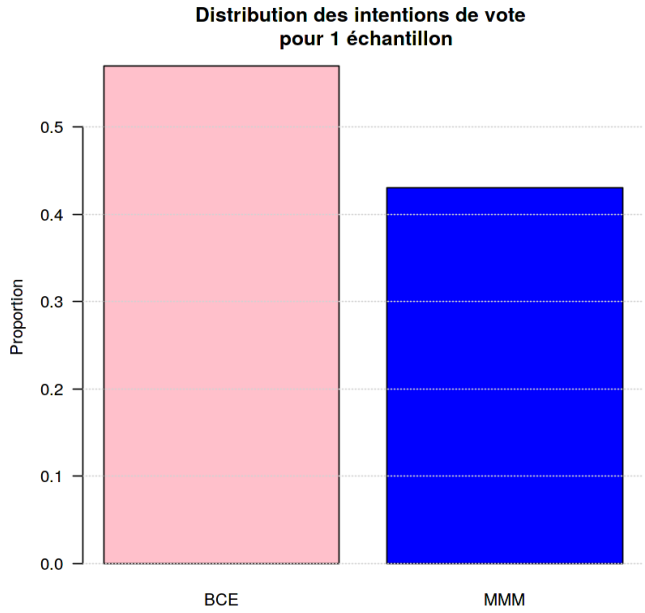
4.3 Application à la simulation des résultats de l'élection BCE-MMM

```
In [1]: #https://www.youtube.com/watch?v=XpQZqXhGj3Y

        a = 1:5
        #Fabriction de population
        prop_B = 0.5461
        prop_M = 0.4539

In [2]: taille_population = 1000000

        candidat = c("BCE", "MMM")
        proportion = c(prop_B, prop_M)
        population = rep(candidat, taille_population*proportion)
        table(population)

Out[2]: population
           BCE    MMM
        546100 453900

In [3]: n.ech = 1000
        n.pop = length(population)
        i = sample(1:n.pop, n.ech)
        ech = population[i]

        #La proportion dans la population
        (tablef = table(ech))

        #La proportion de l'échantillon
        (prop.ech = prop.table(tablef))

        #représenter graphiquement la distribution des intentions de votes.
        coul = c("pink", "blue")
        barplot(prop.ech, col = coul, las = 1, ylab = "Proportion", main = "Distribution des intentions de vo
        grid(nx=NA, ny=NULL)
```

Out[3]:
```
ech
BCE MMM
570 430
```

Out[3]:
```
 ech
  BCE  MMM
0.57 0.43
```

Out[3]:

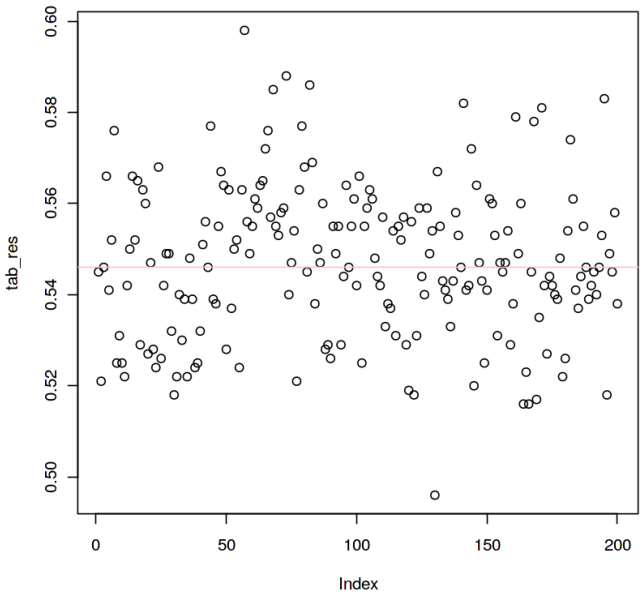


In [4]:
```
#Loi de Xhi2 pour voir si la proportion est en accord avec la population
chisq.test(tablef, p = proportion)
```

Out[4]:
```
        Chi-squared test for given probabilities

data:  tablef
X-squared = 2.3044, df = 1, p-value = 0.129
```

In [5]:
```
#Une boucle qui permet de faire plusieurs simulations
n_sim = 200
tab_res = rep(NULL, n_sim)
for (j in 1:n_sim) {
  i = sample(1:length(population), n.ech, replace=TRUE)
  ech = population[i]
  res = table(ech)
  tab_res[j] = res["BCE"]/n.ech
}

plot(tab_res)
abline(h=0.5461, col="pink")
```

Out[5]:

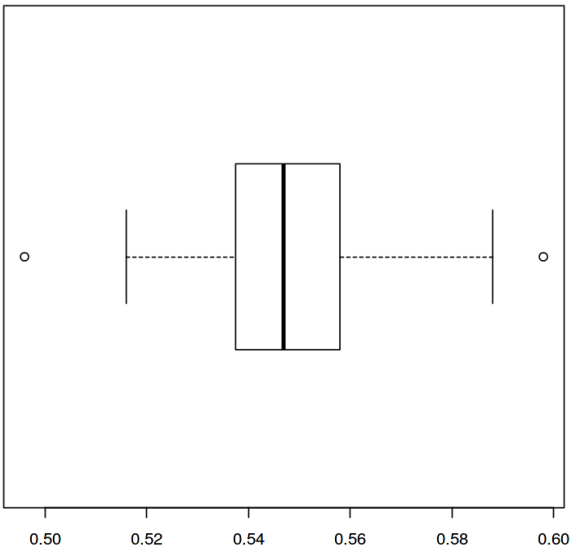


In [6]:
```
#On trace les intervalles de fluctuations. Il y a une dizaine de valeurs hors normes.
(n_interval = qbinom(c(0.025,0.975), size=n.ech, prob=0.5461))
(p_fluct = n_interval/n.ech)
abline(h=p_fluct, col="green")
ordre = sort(tab_res)
(val_pet = ordre[1:10])
(val_gran = tail(ordre, 10))
```

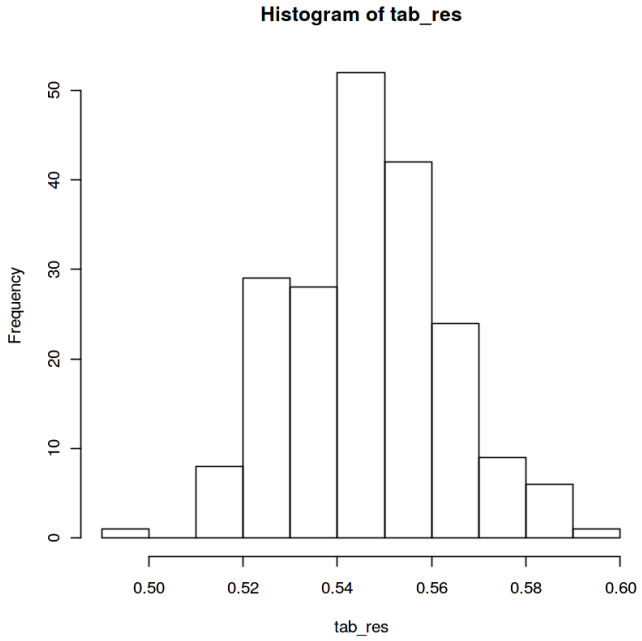
```
    Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...): plot.new has not been called yet
Traceback:


1. abline(h = p_fluct, col = "green")

2. int_abline(a = a, b = b, h = h, v = v, untf = untf, ...)
```

In [7]:
```
#Distribution des 200 simulations
boxplot(tab_res, horizontal = TRUE)
hist(tab_res)
```

Out[7]:



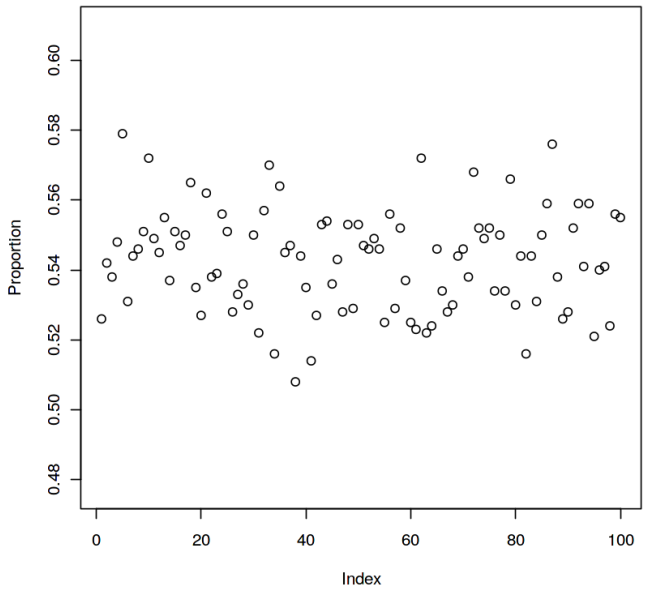
Out[7]:



In [8]:
```
#A présent, nous allons étudier le problème à l'envers.
#C'est à dire que nous allons regarder pour chaque valeur
#obtenue par simulation (par sondage) l'intervalle de
```

```
#confiance au niveau 95%

#D'abord avec la loi binomiale
n_sim = 100
tab_res = NULL
for (j in 1:n_sim) {
  i = sample(1:length(population), n.ech, replace=TRUE)
  ech = population[i]
  (res = table(ech))
  propor_B = res["BCE"]/n.ech
  interval = qbinom(c(0.025, 0.975),
                    size=n.ech,
                    prob = propor_B)/n.ech
  tab_res = rbind(tab_res, c(propor_B, interval))
}

y = range(tab_res)
plot(tab_res[,1], ylim=y, ylab="Proportion")
```

Out[8]:



In [9]:
```
title(paste("Estimation par intervalle (loi binomiale)(", n_sim, "simulations)", sep="" ))
segments(x0=1:n_sim, y0=tab_res[ ,2], x1=1:n_sim, y1=tab_res[, 3])
abline(h=prop_B, col="pink", lwd=2)
i = which(tab_res[,3] < prop_B)
length(i)
segments(x0=i, y0=tab_res[i, 2], x1=i, y1=tab_res[i, 3], col="red")
i = which(tab_res[,2] > prop_B)
length(i)
segments(x0=i, y0=tab_res[i, 2], x1=i, y1=tab_res[i, 3], col="red")
```

```
    Error in title(paste("Estimation par intervalle (loi binomiale)(", n_sim, : plot.new has not been cal
Traceback:


    1. title(paste("Estimation par intervalle (loi binomiale)(", n_sim,
 .     "simulations)", sep = ""))
```

Chapitre 5

Débuts sur Shiny

La librairie Shiny permet d'automatiser les résultats d'une étude donnée. Elle permet la création de pages web interactives incluant toutes les analyses avec R, tout en partageant l'application à des non-utilisateurs de R. Pour mieux appréhender ce package, nous proposons de démarrer avec la création d'une application shiny, partant de la table `Matable`.

5.1 Réaliser un projet R

La première étape consiste à collecter des données autour d'un thème choisi par l'élève-ingénieur (Agriculture, Tourisme, Industrie, Société, Politique, Culture, Musique, etc). De préférence, ce thème concerne la Tunisie et les données sont obtenues soit via le web, soit depuis les institutions et les organismes publics ou privés.

Une fois la base de données nettoyée et bien organisée, il s'agira de bien définir les différentes variables de cette base ainsi que des observations enregistrées. Il y aura des variables continues et d'autres discrètes et il est important de classer ses objets : variables quantitatives continues, discrètes ou temporelles et des variables qualitatives nominales, ordinales ou binaires.

La taille de la table de données (nombre d'enregistrements et nombre de variables) est une donnée initiale importante. A partir de là, et compte-tenu de la nature des variables, une étude descriptive est effectuée : des statistiques descriptives sur les diverses variables de la table permettent de donner une idée générale de la base.

Des représentations graphiques doivent être effectuées, afin de faciliter la visualisation des résultats concernant les analyses univariés des données (diagramme en barre, camembert, histogramme, polygone des fréquences, boxplot, etc).

Afin de rendre cette première approche visible, les élèves-ingénieurs sont invités à créer une application shiny.

Un poster est demandé par élève (travail individuel) où il y a de la visualisation de données, des collectes de très grosses bases de données. Par exemple, merger des bases qui proviennent de plusieurs sources (des bases sur l'économie et la politique fiscale, etc). Ses nouvelles bases seront disponibles sur le site de l'ESSAI. Il y a beaucoup de bases sur la Tunisie, et l'objectif est de les faires connaîtres.

5.2 Collecte des données sur le web

```
In [0]: install.packages("htmltab")
```

```
In [7]: library(htmltab)
        htmltab("https://fr.wikipedia.org/wiki/Tunisie", 3)
```

```
Neither <thead> nor <th> information found. Taking first table row for the header. If incorrect, specifiy hea
```

5.2.1 Récupérer des données à partir de pages web

```
In [10]: library(httr)
         library(XML)

         url <- "https://fr.wikipedia.org/wiki/Tunisie"
         wiki <- GET(url)
         wiki
```

```
Out[10]: Response [https://fr.wikipedia.org/wiki/Tunisie]
           Date: 2018-01-08 19:54
           Status: 200
           Content-Type: text/html; charset=UTF-8
           Size: 715 kB
         <!DOCTYPE html>
         <html class="client-nojs" lang="fr" dir="ltr">
         <head>
         <meta charset="UTF-8"/>
         <title>Tunisie - Wikipédia</title>
         <script>document.documentElement.className = document.documentElement.classNa...
         <script>(window.RLQ=window.RLQ||[]).push(function(){mw.config.set({"wgCanonic...
         mw.user.tokens.set({"editToken":"+\\","patrolToken":"+\\","watchToken":"+\\",...

         });mw.loader.load(["ext.cite.a11y","ext.kartographer.link","mw.MediaWikiPlaye...
         ...
```

```
In [11]: doc <- readHTMLTable(doc = content(wiki, "text"))
         doc[6]
```

```
In [0]: library(WDI)
        library(RJSONIO)
```

```
In [60]: data_index = rbind.data.frame(WDIsearch("research"), WDIsearch("tourism"), WDIsearch("cross-border")
         data_all = WDI(, start=1950, end=2017)
```

```
In [72]: library(reshape)
         i = which(data_all$year>=2016 & data_all$year<=2017)
         x = melt(data_all[i,], id.vars = colnames(data_all)[1:3])
         colnames(x)[2] = "indicator"
```

```
In [74]: #x
```

```
In [75]: library(DT)
         #datatable(x, options = list(pageLength = 5), rownames = FALSE)
```

5.2.2 Récupérer des données à partir d'un fichier pdf ?

Voir ce lien par exemple : http ://francoisguillem.fr/2011/04/extraire-des-donnees-dune-page-web-avec-r-1-les-tableaux/

5.2.3 Récupérer des données à partir de FB

```
In [83]: #install.packages("Rfacebook")
         #install.packages("httpuv")
         #https://developers.facebook.com/
         library(Rfacebook)
         library(httpuv)
```

5.3 Brève application shiny

Les commandes indispensables pour toute application shiny sont :

```
library(shiny)

ui <- fluidPage()

server <- function(input, output){}

shinyApp(ui = ui, server = server)
```

En exécutant ce code, on voit apparaître une page web vide. Ceci signifie que shiny fonctionne bien sur votre ordinateur.

```
library(shiny)

ui <- fluidPage("Bonjour l'ESSAI !")

server <- function(input, output){}

shinyApp(ui = ui, server = server)
```

```
library(shiny)
library(ggplot2)

ui <- fluidPage(
  titlePanel("Ma première application"),
  tabsetPanel(
    tabPanel("Le jeu de données Matable",
             titlePanel(""),
             sidebarLayout(
               sidebarPanel(
                 selectInput("id", "Variable", c("NIVSCOLMERE"))
               ),
               mainPanel(
                 plotOutput(outputId = "graphe")
               )
             )
)))


server <- function(input, output){
  output$graphe <- renderPlot({
    #plot(Matable$NIVSCOLMERE, col='red')
    p <- ggplot(Matable, aes(NIVSCOLMERE))
    p <- p + geom_bar(aes(fill=NIVSCOLMERE))
```

```
    p <- p + scale_fill_brewer(palette="Blues")
    p <- p + labs(x="Niveau scolaire de la mère", y="Effectif", title="Diagramme en barre",
                  subtitle="Distribution en fonction \n des niveaux scolaires\n des mères")})}


shinyApp(ui = ui, server = server)
```

5.4 Cloud de calculs

Lorsque la taille des données est de l'ordre de 20% de la mémoire RAM, le logiciel R fonctionne bien. Au delà, les données sont inutilisables, car la RAM est inadaptée. Les données sont considérées *massives* lorsqu'elles dépassent 50% de la mémoire RAM. Dans ce cas, il vaut mieux utiliser des machines de calculs dédiées à ces bigdata. Nous pensons particulièrement à RosettaHUB [1] qui met à la disposition de l'ESSAI un certain nombre d'outils dédiés au calcul parallèle.

1. La mémoire vive (RAM) est la mémoire informatique dans laquelle peuvent être stockées, puis effacées, les informations traitées par un appareil informatique. C'est une mémoire à accès direct, qui permet de fournir directement les données aux processeurs.
2. Un processeur est un calculateur : il reçoit des ordres du programme, les données sont tirées de la RAM et le microprocesseur exécute des calculs ou des instructions selon un algorithme. Les résultats sont envoyés à une mémoire.
3. Une horloge permet de synchroniser l'échange de données entre circuits. La fréquence de l'horloge a atteint 3 Ghz. Plus ce signal est élevé, plus l'exécution des instructions est rapide.
4. Un GPU est un processeur graphique qui a une structure parallèle, utile pour le calcul matriciel.

Les élèves-ingénieurs de l'ESSAI ont accès au cloud de caclul RosettaHUB. Ils peuvent créer leur propre machine de calculs et l'utiliser dans l'apprentissage et la manipulation de données. Le logiciel R ainsi que Python sont pré-installés sur les machines (avec l'environnement Jupyter et Rstudio).

1. https ://bit.ly/paris-dauphine