




# TP1 : Introduction au langage R et Analyse univariée

**Durée : 6h**

*L'objectif de ce TP est de se familiariser avec le langage de programmation  et son environnement de travail RStudio. Cette prise en main sera l'occasion de revoir les outils élémentaires permettant de présenter de façon synthétique et claire une série de données brute, et d'en résumer les principales caractéristiques.*

## Introduction

R est un langage de programmation dédié à l'analyse de données, le datamining et les statistiques. Il vous permet d'organiser et de traiter rapidement et de manière flexible de gros volumes de données, et de représenter ces données à l'aide d'une grande variété de graphiques disponibles.

Développé à l'origine au début des années 1990 par Ross Ihaka et Robert Gentleman, ce langage est maintenant maintenu et mis à jour par une équipe de développeurs au sein du projet R. Cette structure assure des mises à jour fréquentes. Parmi ses caractéristiques intéressantes, on note les points suivants.

- Ce langage est basé sur la notion de vecteur, ce qui simplifie les calculs mathématiques et réduit considérablement l'utilisation de structures itératives (boucles for, while, etc.). De plus, aucune saisie ou déclaration de variables n'est requise. Cela permet d'écrire des programmes très courts (généralement seulement quelques lignes de code) et de réduire considérablement le temps de développement informatique.
- Il permet de créer facilement des graphiques personnalisables pour mieux visualiser les données et les résultats.
- Il est gratuit sous licence GPL, ce qui signifie que les sources sont disponibles et modifiables. Une communauté d'utilisateurs très active développe constamment de nouvelles fonctionnalités (bibliothèques appelées packages) pour le projet. Il est multi-plateforme, Windows, Mac OS ou Linux.

RStudio est un environnement de développement intégré (IDE) spécialement créé pour fonctionner avec R. Sa popularité a considérablement augmenté depuis 2014. Il vous permet de visualiser les fichiers de script, la ligne de code R, les fichiers d'aide, les graphiques, etc. chez un utilisateur interface conviviale. L'interface RStudio (figure 1) se compose de quatre fenêtres :

- Fenêtre d'édition (en haut à gauche): Dans cette fenêtre apparaissent les fichiers contenant les scripts R. En haut de cette fenêtre, des icônes sont utilisées pour enregistrer le fichier, exécuter un morceau de code sélectionné ou tout le code contenu dans le fichier. L'enregistrement du fichier avec l'extension .R permet une coloration syntaxique adaptée au langage R.
- Fenêtre de commande (en bas à gauche): cette fenêtre contient une console dans laquelle les lignes de code R sont entrées pour exécution.
- Fenêtre espace de travail / historique (en haut à droite): contient les objets en mémoire, qui peuvent être visualisés en cliquant sur leurs noms, ainsi que l'historique des commandes exécutées.

- Fenêtre pour les fichiers / graphiques / packages / aide (en bas à droite): l'explorateur vous permet de vous déplacer dans l'arborescence des répertoires, la fenêtre graphique contient les graphiques dessinés par R (il est possible de les exporter), la fenêtre du package affiche l'installation installée et les packages actuellement chargés et la fenêtre d'aide contient la documentation sur les fonctions et les packages.

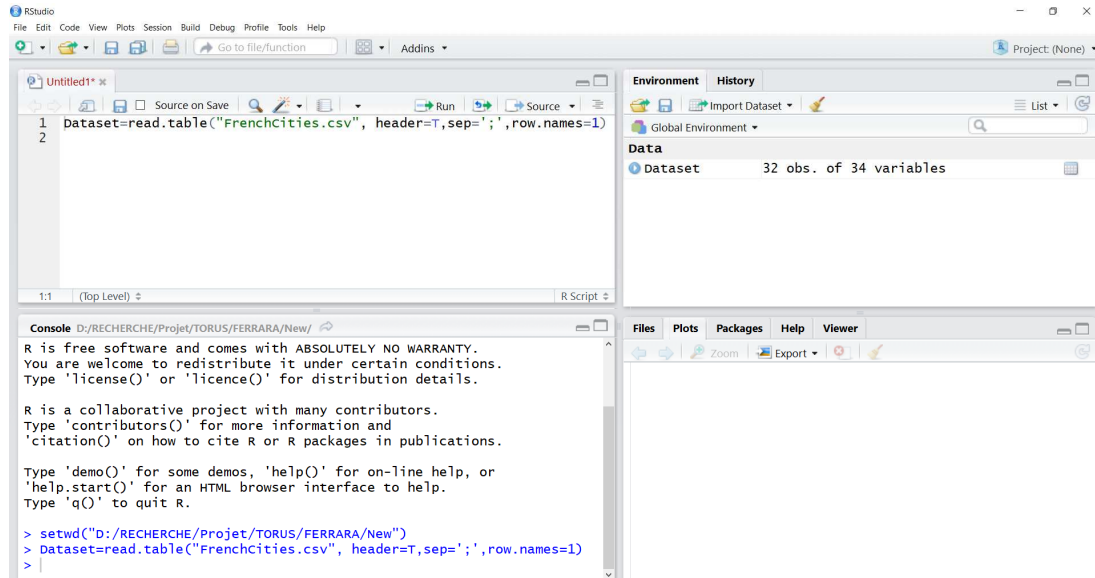


FIGURE 1. Interface RStudio

Sélectionnez un répertoire de travail (dans lequel vous allez mettre vos données) :

Session → Set Working Directory → Choose Directory ...

Ouvrez un nouveau script R :

File → New File → R Script

## Le jeu de données


Le jeu de données utilisé pour illustrer les chapitres suivants concerne les villes françaises (figure 2).

L'unité statistique est une ville. 32 villes ont été observées sur 34 variables :

```
CLIMAT : Kind of climate (Continental=1/Mediterranean=2/Oceanic=3/semi-
         oceanic=4)
NO2 : Nitrogen dioxide
DENSITY : People per km2
RAINFALL : Average annual rainfall (mm)
"MONTH"+r : Average monthly rainfall (12 variables)
DAY_RAINFALL : Average annual number of rainy days
"MONTH"+dr : Average monthly number of rainy days (12 variables)
TEMP : Average annual temperature (d° Celcius)
TEMP_RANGE : Temperature variation
SUNSHINE : Average of sunny days (hours per day)
LATITUDE : Latitude
LONGITUDE : Longitude
```



FIGURE. 2 French cities

Le jeu de données s'appelle "FrenchCities.csv". Afin de l'importer dans RStudio, on utilise l'instruction `read.table`. Ecrivez l'instruction suivante dans votre script R, sélectionnez la ligne et appuyez sur le bouton  Run pour l'exécuter.

```
Dataset=read.table("FrenchCities.csv", header=T, sep=';', row.names=1)
```


`header=T` précise que la première ligne du tableau contient le nom des variables (colonnes).

`row.names=1` indique que la première colonne contient le nom des observations (villes).

`sep=';'` indique que le séparateur de champ entre deux colonnes est le point-virgule.

L'objet créé par l'instruction `read.table` est un *dataframe* qu'on peut voir dans la fenêtre Global Environment (Figure 1). La flèche bleue à gauche donne la liste des variables ainsi que leur nature. On peut aussi utiliser l'instruction `attributes(Dataset)` pour avoir la liste des variables. Le petit carré à droit permet d'afficher le jeu de données dans une nouvelle fenêtre.

**!!! Toujours vérifier le nombre de variables et le nombre d'observations. Si ces nombres ne sont pas corrects, cela signifie que le jeu de données a été mal lu. On rappelle que le nom des observations n'est pas une variable.**

A la place de la ligne de code avec `read.table`, on peut aussi utiliser l'onglet  Import Dataset qui se trouve dans la fenêtre de l'environnement global.

### Comment sélectionner une variable (colonne)?

```
# Permet d'obtenir le nom des variables
names(Dataset)

# Sélection à l'aide de l'attribut $var
Dataset$NO2
```

```
# Sélection à l'aide du numéro de la colonne
Dataset[,2]

# Sélection à l'aide du nom de la colonne
Dataset[, "NO2"]
```

### Comment sélectionner une observation (ligne)?

```
# Permet d'obtenir le nom des observations
row.names(Dataset)

# Sélectionner les lignes 10, 11 et 12
Dataset[10:12,]

# Supprimer les lignes 1, 4 et 7
Dataset[-c(1,4,7),]^A

# Créer un sous-ensemble de lignes suivant une expression logique
subset(Dataset, NO2>50)

# Sélectionner une ligne à l'aide de son nom
subset(Dataset, row.names(Dataset)=="Paris")
```

La fonction `write.table` permet d'enregistrer un dataframe `x` dans un fichier

```
write.table(x, file='name.file', sep=';', row.names=T, col.names=T, ...)
```

## Les variables

L'instruction `summary` permet d'avoir un premier résumé numérique des variables.

```
summary(Dataset)
```

Les variables qualitatives sont appelées *factor*, et leurs modalités *levels*. Les variables quantitatives peuvent être du type *num* si elles sont continues ou *int* si elles sont discrètes. L'instruction `str` donne le type des variables.

```
str(Dataset)
```

La variable CLIMAT est une variable qualitative mais elle est codée avec des entiers :

```
CONT=1
MED=2
OCEAN=3
SEMI_OCEAN=4
```

Elle est donc reconnue comme une variable quantitative par RStudio. C'est pour cela que dans le résumé numérique sont calculés des indicateurs de position ou de dispersion (médiane, variance...) pour cette variable. Ce qui n'a pas de sens pour une variable qualitative. Il faut alors la transformer en type *factor*.

---

<sup>A</sup> L'instruction `c(v1,v2,...,vn)` crée un vecteur dont les composantes sont `v1, v2,...,vn`. Ecrire `c(1,2,3,4,5)` dans la console.

L'instruction `as.xx(var)` transforme `var` dans le type `xx`. `is.xx(var)` renvoie TRUE si `var` est du type `xx`. `levels(var)` donne les modalités d'une variables qualitative.

```
# Transforme la variable CLIMAT en factor
Dataset$CLIMAT=as.factor(Dataset$CLIMAT)
is.factor(Dataset$CLIMAT)

# Renomme ses modalités
levels(Dataset$CLIMAT) # Affiche les modalités actuelles
levels(Dataset$CLIMAT)=c("CONT", "MED", "OCEAN", "SEMI_OCEAN")
levels(Dataset$CLIMAT) # Affiche les nouvelles modalités
```

## Variables quantitatives

Résumé numérique :

```
mean(Dataset$NO2)      # valeur moyenne
sd(Dataset$NO2)        # Ecart-type (Standard Deviation)
median(Dataset$NO2)    # Médiane
quantile(Dataset$NO2)  # Quantiles
Q1 = quantile(x)[2]
Q3 = quantile(x)[4]
```

## Exercice

Refaire la même chose en ajoutant 10 à la série NO2. Conclusion.

Refaire la même chose en multipliant la série NO2 par 2. Conclusion.

Refaire la même chose en ajoutant la valeur supplémentaire 25 à la série NO2. Conclusion.

- Ajouter 10 augmente d'autant les caractéristiques de position mais ne change rien à ceux de dispersion
 

```
mean(Dataset$NO2+10)B
sd(Dataset$NO2+10)
median(Dataset$NO2+10)
quantile(Dataset$NO2+10)
```
- Multiplier par 2 la série multiplie toutes les caractéristiques par 2 sauf la variance qui est multipliée par 2<sup>2</sup>.
 

```
mean(Dataset$NO2*2)
sd(Dataset$NO2*2)
median(Dataset$NO2*2)
quantile(Dataset$NO2*2)
```
- Ajouter la valeur extrême 25 ne change quasiment rien aux quantiles mais augmente nettement la moyenne et la variance.
 

```
mean(c(Dataset$NO2, 25))C
sd(c(Dataset$NO2, 25))
median(c(Dataset$NO2, 25))
quantile(c(Dataset$NO2, 25))
```

<sup>B</sup> L'instruction `vecteur+10` ajoute 10 à chaque composante du vecteur. Idem pour la multiplication.

<sup>C</sup> L'instruction `c(vecteur,v)` ajoute la composante `v` à la fin du vecteur (`c` de concaténer). Ecrire dans la console : `vecteur=c(1,2,3,4,5)` puis `c(vecteur,6)`.

## Résumé graphique :

```
# Histogramme
hist (Dataset$TEMP,main='Histogram of TEMP',xlab='TEMP',col='blue')

# Quantile boxplot (boite de Tuckey)
boxplot (Dataset$TEMP,main='Boxplot of TEMP',ylab='TEMP',col='grey')
```

On constate sur la boîte de Tukey qu'il y a une observation atypique (outlier). Cette observation correspond à la « ville » *Error* dans le jeu de données. Il y a plusieurs méthodes permettant de supprimer *Error* du jeu de données. Nous allons utiliser le fait que *Error* est la seule ville ayant une température moyenne nulle. Il suffit donc de sélectionner toutes les villes ayant une température strictement positive.

```
Dataset0=Dataset # Dataset0 est le jeu de données avec Error
Dataset=subset (Dataset,TEMP>0)
```

**Exercice**

Recalculer les indicateurs numériques du nouveau jeu de données et comparer avec les anciens indicateurs.

Comparer les graphiques notamment les boîtes de Tukey

**Variables qualitatives**

## Table de contingence et graphique

```
tab=table (Dataset$CLIMAT)
tab
pie(tab)
barplot(tab) # Donne un diagramme en bâtons alors qu'il faut un
              # diagramme en barre
barplot(as.matrix(tab),legend=TRUE)
prop.table(tab)
addmargins(tab)
```

## Transformation d'une variable quantitative en variable qualitative

On va créer une nouvelle variable SUN prenant les modalités « TRES » si la variable SUNSHINE est supérieure à 2085, « PEU » si elle est inférieure à 1793 et « MODERE » sinon.

```
SUN=hist (Dataset$SUNSHINE,breaks=c(0,1793,2085,2917))$counts
barplot (SUN,names.arg=c("PEU","MODERE","TRES"))
```

**Exercice**

En transformant cette variable, on perd de l'information notamment concernant les indicateurs numériques. Proposer une méthode pour estimer la moyenne d'ensoleillement à partir de la variable SUN.

Pour chaque classe de l'histogramme, on prend une valeur représentative, par exemple le milieu. On multiplie cette valeur par l'effectif de la classe. On somme puis on divise par l'effectif total.

> SUN

[1] 8 16 8

$$\left(\frac{1793+0}{2} * 8 + \frac{2085+1793}{2} * 16 + \frac{2917+2085}{2} * 8\right) / 32 = 1818.875$$

La vraie moyenne est 2004.062.

### Exercice

Reprendre toutes les instructions précédentes avec le jeu de données WineQuality.csv. Ce jeu de données représente des vins rouges (codés 1) et des vins blanc (codés 0) portugais "Vinho Verde". Ils sont définis par des caractéristiques chimiques ainsi qu'un score de qualité entre 0 et 10.

- Combien y-a-t-il de variables ? d'observations ?
- Quelle est la nature des variables ? Faut-il faire des transformations ?
- Donner les résumés numériques et graphiques appropriés aux variables ?
- Y-a-il des valeurs aberrantes ?
- Créer une nouvelle variable, QualiSup, qui prend la valeur « SUP » si le score de qualité est supérieur à 8 et « INF » sinon.
- Donner le tableau de contingence de cette nouvelle variable.

```
Mydata=read.table("winequality.csv", sep=";", header=T)
QualiSup=Mydata$quality>8
QualiSup=Mydata$quality>8
QualiSup=as.factor(QualiSup)
levels(QualiSup)
[1] "FALSE" "TRUE"
levels(QualiSup)=c("SUP", "INF")
table(QualiSup)
      QualiSup
      SUP   INF
6492      5
```

### Exercice

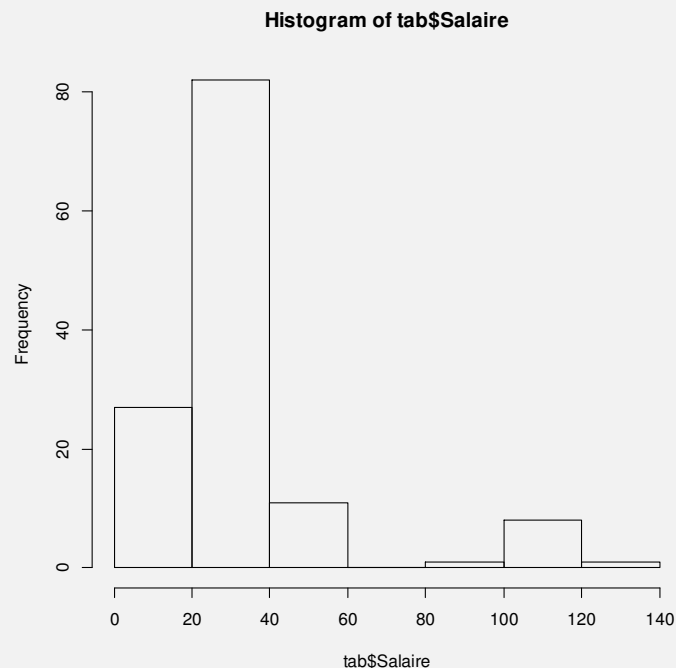
Le fichier SalariesData.csv présente les salariés d'une entreprise ayant 3 sites (A, B et C). On y indique leur sexe, leur salaire annuel (en milliers d'euros), leur catégorie (cadre supérieur, moyen ou ouvrier employé), leur âge et leur site.

- 1) Calculer le salaire moyen et le salaire médian. Que signifie la différence entre ces deux valeurs ?
- 2) Calculer le 1er et le 3eme quartile, ainsi que le 1er et le 9eme déciles de la série des salaires.
- 3) Lesquels faudrait-il mettre en avant lors d'une négociation salariale, et pourquoi ?
- 4) Etablir un tableau avec effectifs ou fréquences pour la série des catégories et celle des établissements. Faire une représentation graphique adaptée.
- 5) Représenter les variables discrètes salaire et âge ? Cette représentation vous semble-t-elle adaptée ?
- 6) Effectuer un regroupement par classe des salaires puis des âges (pas plus de 5 classes). Recalculer la moyenne et comparer avec la série d'origine.

```

tab <- read.table("SalariesData.csv",header=T,sep=";")
1)
mean(tab$Salaire)
      [1] 32.03846
median(tab$Salaire)
      [1] 23
hist(tab$Salaire)
La différence entre salaire moyen et médian est due aux très gros salaires des cadres supérieurs.
2)
quantile(tab$Salaire,prob=c(0.1,0.25,0.75,0.90))
      10%  25%  75%  90%
      18.0 21.0 26.0 55.1
3)
Les dirigeants mettraient en avant les quartiles qui masquent les écarts de salaires, alors que les salariés parleraient des déciles.
4)
pEtab=table(tab$Etablissement)/130*100
pEtab
      A      B      C
46.15385 30.76923 23.07692
barplot(as.matrix(pEtab),legend=T,col=heat.colors(3))
5)
barplot(tab$Salaire)
barplot(tab$Age)
Les représentations par un diagramme en bâtons ne sont pas adaptées car il y a trop de valeurs dans chaque série. Il faut faire un regroupement par classe.
6)
h <- hist(tab$Salaire, freq=TRUE,breaks=5)

```



```

seg <-h$breaks
seg

```



```

      [1] 0 20 40 60 80 100 120 140
eff <- h$counts
eff
      [1] 27 82 11 0 1 8 1
nbclasses <- length(eff)
nbclasses
      [1] 7
lclasses=max(tab$Salaire)/nbclasses
lclasses
      [1] 20
midclasses <- seq(lclasses/2
midclasses <- midclasses[1:nbclasses]
midclasses
      [1] 10 30 50 70 90 110 130
sum(midclasses*eff)/sum(eff)
      [1] 33.69231
mean(tab$Salaire)
      [1] 32.03846

```

La différence entre les moyennes est due à l'hypothèse de répartition uniforme dans chaque classe.

## Quelques éléments de programmation

R est un langage de programmation matricielle (comme Matlab ou Scilab). Cela signifie qu'il offre des fonctionnalités de calcul directement sur les vecteurs/matrices. Il est particulièrement performant quand ses fonctionnalités sont utilisées en remplacement d'une boucle.

### Création de vecteurs et matrices

L'instruction `vector(mode,length)` permet de créer un vecteur de longueur `length` dont les composantes sont du type `mode("logical", "numeric", "character"...)`. Cela remplace l'instruction `c(x1,x2,...,xn)` que nous avons vu précédemment.

L'instruction `matrix(data,nrow,ncol)` permet de créer une matrice à `nrow` lignes et `ncol` colonnes avec les éléments de `data`.

```

V=vector(mode="numeric",length=10)
V
M=matrix(0,nrow=2,ncol=5)
M
V=c(1,2,3,4,5,6,7,8,9,10)
M=matrix(V,nrow=2,ncol=5)      # remplit la matrice colonne par colonne
M
M=matrix(V,nrow=2,ncol=5,byrow=T) # remplit la matrice ligne par ligne
M
V=c(1,2,3)
M=matrix(V,nrow=2,ncol=5)
M

```

### Opérations sur les vecteurs et matrices

Lorsqu'on applique une opération sur un vecteur ou une matrice, elle s'opère composante par composante. Si on souhaite effectuer un produit matriciel il faut utiliser l'instruction `%*%` à la place de `*`. Attention à la compatibilité des dimensions.

```
V1=c(1,2,3,4,5,6,7,8,9,10)
V2=V1*2
V1+V2
M1=matrix(V1,nrow=2,ncol=5)
M2=matrix(V2,nrow=2,ncol=5)
M1*M2
M1%%M2
M1%*%t(M2) # t transpose la matrice

M=matrix(V,nrow=3,ncol=3)
diag(M) # diagonale d'une matrice
det(M) # déterminant d'une matrice
sum(M) # somme les éléments d'une matrice
exp(M) # applique la fonction exponentielle à tous les éléments de la matrice

cbind(M1,M2) # concaténation de matrices par les colonnes
rbind(M1,M2) # concaténation de matrices par les lignes

which(V2>=8) # donne les index des éléments d'un vecteur vérifiant la condition logique
which.min(V2) # donne l'index du minimum (idem pour maximum)
```

### Passage d'une matrice à un dataframe

On note la différence d'affichage entre une matrice et un dataframe dans la fenêtre d'environnement global de RStudio. Il s'agit de deux objets différents. Toutes les opérations possibles sur les matrices ne le sont pas forcément sur les dataframe et vice-versa. De façon générale pour transformer la nature d'un objet, on utilise l'instruction `as.nature-de-l'objet`, donc ici, `as.data.frame` ou `as.matrix`.

```
M=as.data.frame(M)
M
names(M)
names(M)=c("X1", "X2", "X3")
M
row.names(M)
```

### Boucles

Les instructions pour les différents types de boucles sont

```
if(cond) expr
if(cond) cons.expr else alt.expr

for(var in seq) expr
while(cond) expr
repeat expr
```

S'il y a un bloc d'instruction à exécuter, ce bloc doit être entre accolades.

```
V1=vector(mode="numeric",length=5)
V2=vector(mode="numeric",length=5)
for (i in 1:5)
{
  V1[i]=i
  V2[i]=2*i
}

# version simplifiée quand une seule instruction
for (i in 1 :5) V2[i]=2*i
```

Il est préférable d'éviter de faire une boucle si on peut la remplacer par une opération matricielle. Les instructions suivantes permettent de comparer le temps CPU du remplissage d'une matrice avec ou sans boucle.

```
# Avec boucle
A=matrix(0,2000,2000)
time1<-Sys.time()
for (i in 1:2000)
{
  for (j in 1:2000)
  {
    A[i,j]=runif(1)
  }
}
time2<-Sys.time()
difftime(time2, time1)

# Sans boucle
A=matrix(0,2000,2000)
time1<-Sys.time()
A=runif(2000*2000)
time2<-Sys.time()
difftime(time2, time1)
```

## Fonction

L'instruction `function` permet de créer une fonction

```
function(arguments)
{
  expr
  return(value)
}
```

```
demicercle=function(x,y)
{
  if (x>0 & y>0) z= x^2 + y^2
  else z=0
  return(z)
}
demicercle(-1,2)
demicercle(1,2)
```

```
# version simplifiée quand une seule instruction  
cercle=function(x, y) x^2 + y^2  
cercle(-1,2)
```