Module 2
Introduction au langage R

Sommaire

Dernière mise à jour le 19 septembre 2018

Introduction

L'apparition du domaine de la fouille des données dans les années 1990 a entraîné l'apparition de plusieurs langages de programmation/outils/librairies/modules pour l'analyse des données. Nous pouvons citer à titre d'exemple Clementine de SPSS, Enterprise Miner de SAS, Insightfull Miner de Splus, KXEN, SPAD, Statistica Data Miner, Statsoft, WEKA, R, Python et plusieurs autres.
Avec l'évolution du domaine de la science des données, les langages R et Python ont pris une place considérable au sein de la communauté de la science des données du fait de leur flexibilité et leur simplicité. Bien que R fut développé à l'origine pour le calcul statistique, il est largement utilisé à la fois par les programmeurs et par les non programmeurs. Contrairement à R, Python est destiné aux programmeurs, donc la connaissance de la programmation aiderait à bien avancer avec Python.
Vu la simplicité et la flexibilité du langage R, nous avons choisi de l'utiliser dans le cadre du cours. Tous les exemples et exercices présentés dans ce cours se basent sur le langage R. Dans ce qui suit, nous allons introduire les bases de R et quelques librairies et fonctions pertinentes pour l'analyse des données. Notez bien que l'objectif de ce module n'est pas d'introduire en détails le langage R, mais plutôt de présenter les concepts de base qui vous aideront 1) à suivre les modules suivants, et 2) à répondre à un problème de science des données dans toutes ses étapes, à savoir l'analyse des données, le traitement et la visualisation. Les possibilités offertes par R étant très vastes, il est utile dans un premier temps d'assimiler certaines notions et concepts afin d'évoluer plus aisément par la suite.

2.1 Langage R

R est un langage de programmation et un logiciel d'analyse statistique et graphique créé par Ross Ihaka et Robert Gentleman [1]. R est distribué librement sous les termes de

1. Ihaka R. and Gentleman R. 1996. R : a language for data analysis and graphics. Journal of Computational and Graphic al Statistics 5 : 299-314.

la GNU General Public Licence et est facile à installer à partir de la page du CRAN (Comprehensive R Archive Network)[2]. Dans le site du CRAN, on peut trouver toutes les informations et ressources nécessaires à l'utilisateur, à savoir des fichiers d'installation, des mises à jour, des librairies, FAQ, newsletter, documentation, etc.
R est le langage le plus utilisé au sein de la communauté statistique académique. Il est également en croissante utilisation chez les industriels particulièrement dans les services R&D des entreprises, et par les organismes gouvernementaux. Par exemple, le gouvernement anglais utilise R pour moderniser les rapports des statistiques officielles[3], et la cité de Chicago qui utilise R pour émettre des alertes sur la sécurité des plages[4]. R comporte de nombreuses fonctions pour l'analyse statistique et la visualisation. R offre également la possibilité d'exporter les graphiques générées sous différents formats à savoir jpg, png, pdf, ps, etc. dépendamment du système d'exploitation utilisé.

2.1.1 Comment R fonctionne ?

R est un langage interprété et non compilé, c'est-à-dire, les commandes tapés au clavier sont directement exécutées sans qu'il soit besoin de construire un programme complet comme c'est le cas avec d'autres langages de programmation (C, Pascal, Fortran, etc.)[5]. La syntaxe de R est très simple et intuitive. D'une manière générale, le nom d'une fonction, ou son abbréviation, reflète le sens et l'objectif de cette fonction. À titre d'exemple, la commande $lm(Y\ X)$ désigne une modèle de régression linéaire. De la même façon, les fonctions de lecture, read, et d'écriture, write, désignent respectivement la lecture et écriture des données. Toute fonction est appelée avec son nom suivi des parenthèses. Même si la fonction n'a pas d'argument, on spécifie quand même les deux parenthèses vides. Si les parenthèses ne sont pas fournies, R affiche une description détaillée du contenu des instructions de la fonction en question.

2. http ://cran.r-project.org/
3. http ://blog.revolutionanalytics.com/government/
4. http ://blog.revolutionanalytics.com/government/
5. *https* : *//cran.r − project.org/doc/contrib/Paradis − rdebuts*$_f$*r.pdf*

Quand on utilise R, les variables, les données, les fonctions, et les résultats sont stockés dans la mémoire de l'ordinateur sous forme d'objets qui ont chacun un nom. Nous pouvons donc agir sur ces objets avec des opérateurs (arithmétiques, logiques, etc.) ou avec des fonctions qui sont elles-mêmes des objets.
Dans, les sections suivantes, nous allons donner plus de détails sur les fonctions de R relatives au chargement, à l'exploration et à la visualisation des données. Rappelons que l'objectif de ce module est d'introduire les bases de R pour que vous puissiez progresser dans ce cours, et non pas de fournir une introduction exhaustive à R. Nous allons utiliser l'éditeur RStudio [6] dans tous les exemples présentés dans ce module.

2.1.2 Chargement des données dans R

Dans cette section, nous allons présenter les différentes fonctions de R qui permettent de charger des données externes dans l'environnement de R pour pouvoir les exploiter. Les fonctions de lecture des données externes sont multiples et sont utilisées en fonction du format (fichier) dans lequel ces données sont stockées (CSV, Excel, texte, dat, etc.).

2.1.3 Chargement d'un fichier CSV dans R

Les fichiers CSV représentent le format le plus utilisé pour stocker des données structurées en colonnes. Pour lire un fichier CSV dans R, nous allons procéder comme suit :

```
1 # Lecture d'un fichier CSV
  MyData <- read.csv("icecream.csv", header = T, sep = ',')
```

Dans le code ci-dessus, le contenu du fichier CSV est stocké dans la variable `MyData`. La fonction `read.csv` lit le fichier "icecream.csv" avec son entête. Nous avons également spécifié le caractère qui sépare les colonnes dans le fichier "icecream.csv", dans ce cas une virgule. Dans l'éditeur RStudio, l'exécution de ce bout de code donne le résultat présenté dans la Figure 2.1.

6. https ://www.rstudio.com/

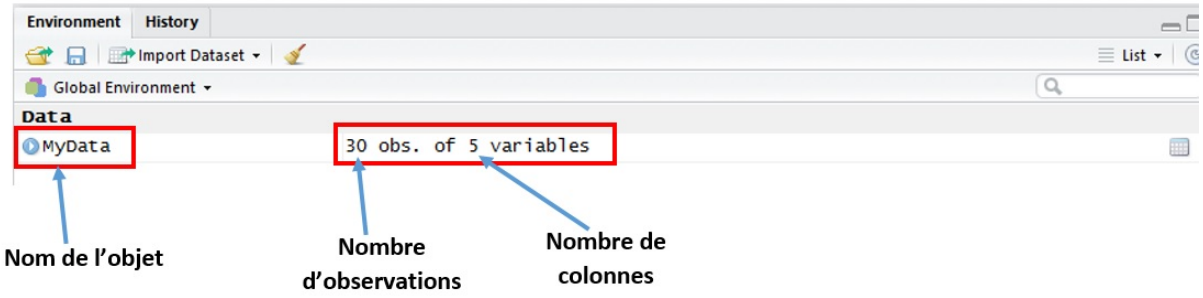

Figure 2.1: Lecture d'un fichier CSV avec la fonction read.csv.

Si le fichier CSV est volumineux. Il existe des fonctions plus efficaces que la fonction **`read.csv`**. *Par exemple, la fonction* **`fread`** *de la librairie* **`data.table`** *pourrait être utilisée dans ce contexte comme suit :*

```
# Lecture d'un fichier CSV volumineux
MyData <- fread("icecream.csv", header = T, sep = ',')
```

Il est toujours important d'importer l'entête du fichier en mettant l'attribut **`header`** *à* **`TRUE (T)`**. *Cela permet de manipuler les colonnes avec leurs noms et non pas des noms de variables générées automatiquement par R.*

2.1.4 Chargement d'un fichier Excel dans R

De la même façon qu'avec les fichiers CSV, les fichiers Excel sont importés dans R avec des fonctions comme `readWorksheetFromFile` de la librairie `XLConnect`, ou la fonction `read.xlsx` de la librairie `xlsx` pour les fichiers Excel avec une extension xlsx. Il existe

évidemment d'autres fonctions et librairies pour l'importation des fichiers Excel dans R comme la fonction `read_excel` de la librairie `readxl`, et la fonction `read.xls` de la librairie `gdata`. Nous n'allons pas passer en revue de toutes ces fonctions, mais l'idée est de présenter la façon dont les fichiers Excel sont importés.
Rappelons que certaines librairies ne sont pas installées par défaut dans R. Par conséquent, si la librairie n'est pas installée, vous devriez tout d'abord l'installer avec la commande `install.packages('nom de la librairie')`.

```
# Lecture d'un fichier excel avec la librairie gdata
library(gdata)
MyData <- read.xls("Iris.xls", sheet = 1, header = T)
```

Nous pouvons toujours vérifier dans l'environnement RStudio que le fichier a été chargé correctement comme le montre la Figure 2.2.

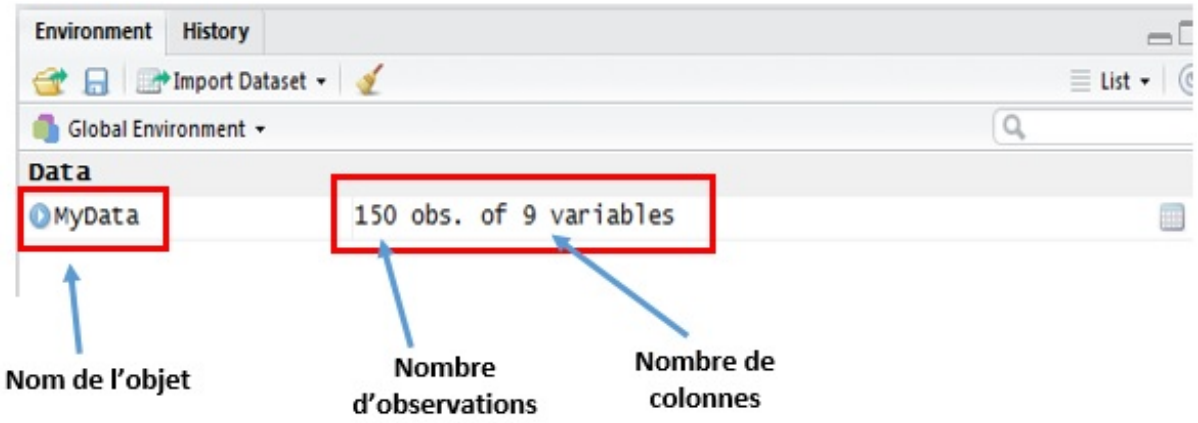

Figure 2.2: Lecture d'un fichier excel en R avec la fonction read.xls.

Voici un autre exemple de lecture d'un fichier excel avec la librairie `XLConnect` :

```r
# Lecture d'un fichier excel avec la librairie XLConnect
library(gdata)
MyData <- read.xls("Iris.xls", sheet = 1, header = T)
library(XLConnect)
wk <- loadWorkbook("Iris.xls")
MyData <- readWorksheet(wk, sheet = 1)
```

Dans les deux cas, nous avons le même nombre d'observations (150) et de colonnes (9).

2.1.5 Chargement d'un fichier texte dans R

Nous pouvons toujours avoir le cas où les données sont stockées dans un fichier texte. Dans ce cas, il existe des librairies dédiées à ce genre de situation. Nous pouvons citer, à titre d'exemple, la librairie `utils` avec les fonctions `data.table`, et `read.delim`. Dans l'exemple suivant, nous utilisons la fonction `read.table` pour charger le contenu d'un fichier texte.

```r
# Lecture d'un fichier texte avec la librairie utils
MyData <- read.table("Icecream.txt", header = T, sep = ',')
```

2.1.6 Chargement d'un fichier SPSS dans R

Les utilisateurs du logiciel SPSS peuvent également exporter leur données sous format SPSS et l'importer en suite dans R pour des manipulations avancées. La lecture d'un fichier SPSS dans R pourrait se faire avec la librairie `foreign` de la façon suivante :

```r
# Lecture d'un fichier SPSS avec la librairie foreign
library(foreign)
Mydata <- read.spss("Cancer.sav", to.data.frame= T)
```

Notez bien que toutes les données chargées, tel que nous l'avons vu précédemment, sont stockées dans des objets de type `Data Frame` *notés généralement* `dataframe` *ou simplement* `df`*. Les objets de type dataframe offrent beaucoup de flexibilité lors de la manipulation des données. Nous allons voir dans les sections suivantes, comment ces objets facilitent la manipulation des données. On peut s'informer de la nature de cet objet en utilisant la fonction* `class(nom_de_l'objet)`*. Nous pouvons également passer d'un type dataframe à un type de données matrix(matrice) en utilisant des fonctions comme par exemple* `as.matrix`*.*

2.2 Manipulation des données avec R

Dans la section précédente, nous avons présentés quelques fonctions pour la lecture des différents types de fichiers de données dans R. Dans cette section, nous allons présenter quelques concepts liés à la manipulation des données une fois importées dans R.

2.2.1 Selection d'un sous-ensemble de données

Il arrive souvent de manipuler des sous-ensembles de données au lieu de la totalité des données. Par exemple, manipuler un certain nombre de colonnes, de lignes ou des requêtes spécifiques. Pour cela, nous aurons besoin de méthodes simples et efficaces pour extraire ces sous-ensembles de données à partir de l'objet contenant la totalité des données. R offre plusieurs mécanismes pour extraire des sous-ensembles de données à partir d'un objet `dataframe`.
Par exemple, pour sélectionner les deux colonnes "income" et "price" à partir de l'objet `MyData` après la lecture du fichier csv, nous procéderons comme suit :

```r
# Sélectionner certaines colonnes par leur noms
MyData <- read.csv("icecream.csv", header = T, sep = ',')
MyData_income_price <- subset(MyData, select= c("income", "price"))
```

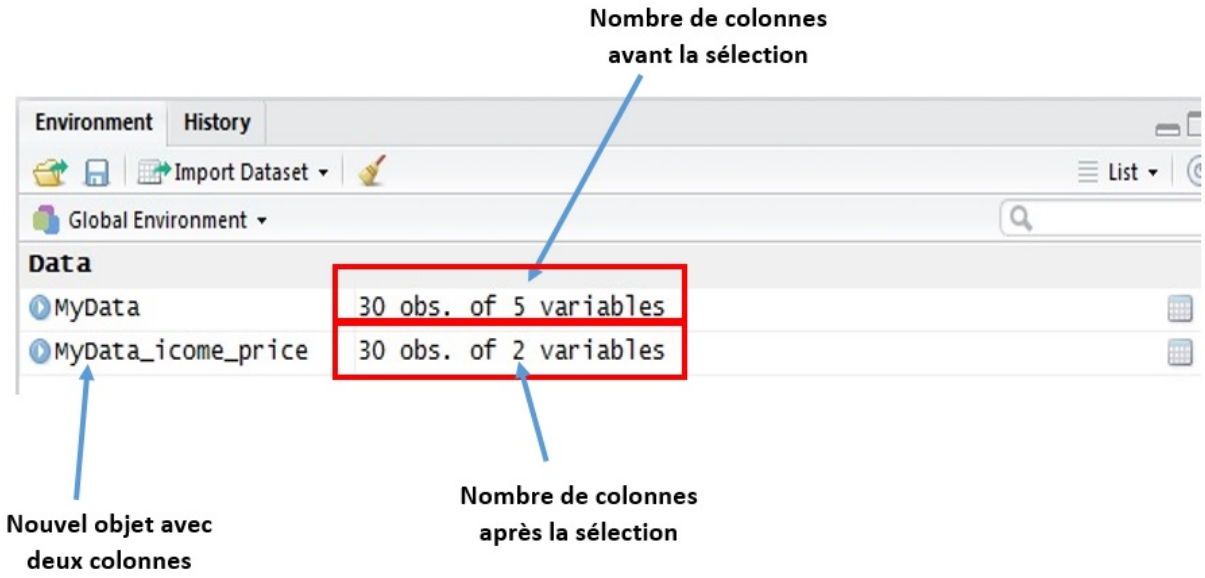

FIGURE 2.3: Sélection des colonnes par leurs noms.

Nous pouvons vérifier l'objet `MyData_income_price` en affichant son contenu :

```r
# Résultat de la requête de sélection des données avec la condition price > 0.270
> MyData_income_price
  income price
1     78 0.270
2     79 0.282
3     81 0.277
4     80 0.280
```



```
5       76 0.272
6       78 0.262
7       82 0.275
...
```

Nous pouvons également sélectionner les deux colonnes en spécifiant leurs indexes dans l'objet `MyData` comme suit :

```
# Sélectionner certaines colonnes par leur indexes
MyData <- read.csv("icecream.csv", header = T, sep = ',')
MyData_income_price <- subset(MyData, select= c(3,4))
```

Rappelons qu'il existe d'autres façons de sélectionner les colonnes comme l'expression `MyData[, c(3,4)]` qui donne le même résultat que les deux autres méthodes présentées ci-dessus.

2.2.2 Sélection conditionnelle

Nous pouvons avoir la situation où nous devons travailler avec des données qui satisfassent certaines conditions. Par exemple, des données dont la date est égale à une certaine date, ou des valeurs numériques qui sont non nulles, etc. Prenons l'exemple du fichier *icecream.csv*, nous pouvons sélectionner uniquement les données correspondant à un prix ($price > 0.270$). Pour cela, nous pouvons écrire :

```
# Sélectionner des données avec la condition price > 0.270
MyData <- read.csv("icecream.csv", header = T, sep = ",")
MyData_selection <- MyData[MyData$price > 0.270, ]
```

Puisque la sélection concerne les lignes, donc la condition doit se faire dans la partie gauche des crochets [], et la partie droite doit rester vide pour dire que nous aurons besoin de toutes les colonnes. Le résultat de cette requête est présenté ci-dessous :

```
# Résultat de la requête de sélection des données avec la condition price > 0.270
> MyData_selection
    X  cons income price temp
2   2 0.374     79 0.282   56
3   3 0.393     81 0.277   63
4   4 0.425     80 0.280   68
5   5 0.406     76 0.272   69
7   7 0.327     82 0.275   61
10 10 0.256     79 0.277   24
11 11 0.286     82 0.282   28
...
```

Il existe également des fonctions de filtrage comme la fonction `filter` de la librairie `dplyr` qui pourrait être utilisée comme suit :

```
# Sélectionner des données avec la fonction \textbf{filter}
library(dplyr)
MyData <- read.csv("icecream.csv", header = T, sep = ",")
MyData_selection_filter <- filter(MyData, MyData$price > 0.270)
```

Nous pouvons bien évidemment ajouter plusieurs conditions à la fois qui touchent plusieurs colonnes. Par exemple :

```
# Sélectionner des données avec la fonction \textbf{filter}
library(dplyr)
MyData <- read.csv("icecream.csv", header = T, sep = ",")
MyData_selection_filter_conditions <- filter(MyData, MyData$price > 0.270 & MyData$income > 80)
```

Ce qui donne le résultat suivant :

```
# Résultat de la requête de sélection des données avec la condition price > 0.270 et income > 80
> MyData_selection_filter_conditions
```

```
    X  cons income price temp
1   3 0.393     81 0.277   63
2   7 0.327     82 0.275   61
3  11 0.286     82 0.282   28
4  13 0.329     86 0.272   32
5  14 0.318     83 0.287   40
6  15 0.381     84 0.277   55
...
```

2.2.3 Sélection aléatoire

La sélection aléatoire consiste à choisir de façon aléatoire un sous-ensemble de données. La sélection aléatoire est largement utilisée dans l'entraînement des algorithmes d'apprentissage automatique. Il existe deux types de sélection aléatoire : avec remise et sans remise. Dans la première, les données sélectionnées sont remises pour pouvoir les réutiliser dans la prochaine sélection, tandis que dans la sélection sans remise, les données ne sont pas remises et donc elles ne seront pas disponibles pour la prochaine sélection. Nous pouvons utiliser la fonction `sample` pour sélectionner aléatoirement un sous-ensemble de données comme suit :

```
# Lecture des données
MyData <- read.csv("icecream.csv", header = T, sep = ",")
# Sélection aléatoire sans remise
Selection_sans_remise <- MyData[sample(1:nrow(MyData), 10,
   replace=FALSE),]

> Selection_sans_remise
    X  cons income price temp
5   5 0.406     76 0.272   69
27 27 0.376     94 0.265   41
13 13 0.329     86 0.272   32
16 16 0.381     82 0.287   63
6   6 0.344     78 0.262   65
```

```
18 18 0.443      78 0.277   72
28 28 0.416      96 0.265   52
21 21 0.319      85 0.292   44
29 29 0.437      91 0.268   64
7   7 0.327      82 0.275   61
```

Dans cet exemple, nous avons sélectionné un sous-ensemble de taille 10 à partir de l'objet `MyData`. Le paramètre `replace=FALSE` indique que la sélection est sans remise.

2.2.4 Gestion des données manquantes

Les valeurs manquantes représentent l'un des problèmes les plus fréquents dans la manipulation des données. Beaucoup de jeux de données disponibles sur le Web ont des valeurs manquantes qui requièrent d'être traitées ou filtrées. D'une manière générale, les valeurs manquantes sont présentes dans les données sous forme de valeurs nulles (`null`), des champs vides (`NA`), ou des symboles d'infini (`Inf`).
R offre beaucoup de flexibilité en ce qui a trait aux valeurs manquantes. Le cas le plus simple est d'omettre ces valeurs manquantes. Les fonctions `na.omit` et `na.exclude` peuvent être utilisées dans ce cas comme illustré dans l'exemple ci-dessous. Nous pouvons également vérifier l'existence des valeurs manquantes dans un objet avec la fonction `is.na`.

▶ **Exemple 1** Dans cet exemple, nous allons d'abord créer un objet de type `dataframe` qui contient trois lignes et deux colonnes. Supposons aussi que l'entrée de la troisième ligne et la deuxième colonne contient une valeur manquante.

```
# Créer un objet dataframe
df <- as.data.frame(matrix(c(1:5, NA), ncol = 2))
> df
  V1 V2
1  1  4
2  2  5
3  3 NA
```

```
# Vérifier l'existence des valeurs manquantes
> is.na(df)
        V1    V2
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,] FALSE  TRUE # cette entrée contient une valeur manquante

> na.omit(df)
  V1 V2
1  1  4
2  2  5

> na.exclude(df)
  V1 V2
1  1  4
2  2  5
```

Les deux fonctions `na.omit` et `na.exclude` retournent l'objet `dataframe` sans les lignes contenant des valeurs manquantes.

Ce que nous avons présenté, ce sont des méthodes qui permettent d'omettre les valeurs manquantes, mais dans beaucoup de cas, nous pouvons remplacer les valeurs manquantes par d'autres valeurs numériques par exemples des zéros, des moyennes par rapport à chaque colonne, etc. Par exemple, pour remplacer les valeurs manquantes par des zéros dans le `dataframe` de l'exemple précédent, nous pouvons écrire :

```
# Remplacer des valeurs manquantes par des zéros
> df[is.na(df)] <- 0
> df
  V1 V2
1  1  4
2  2  5
3  3  0 # NA remplacée par 0
```

2.2.5 Fusion des données

La fusion des données est un concept très fréquent dans la science des données. Nous pouvons fusionner plusieurs sources de données horizontalement ou verticalement selon le besoin. Par exemple, si nous avons deux dataframes, la fusion de ces dataframes horizontalement et verticalement peut se faire de la façon suivante :

```
# Fusionner deux dataframes
df1 <- as.data.frame(matrix(c(1:6), ncol = 2))
> df1
  V1 V2
1  1  4
2  2  5
3  3  6

df2 <- as.data.frame(matrix(c(7:15), ncol = 3))
df2
  V1 V2 V3
1  7 10 13
2  8 11 14
3  9 12 15
# Fusion horizontale si les deux dataframes ont le même nombre de lignes
df <- cbind(df1, df2)
> df
  V1 V2 V1 V2 V3
1  1  4  7 10 13
2  2  5  8 11 14
3  3  6  9 12 15

# Fusion verticale si les deux dataframes ont le même nombre de colonnes
df1 <- as.data.frame(matrix(c(1:6), ncol = 2))
> df1
  V1 V2
```

```
1  1  4
2  2  5
3  3  6

df2 <- as.data.frame(matrix(c(7:15), ncol = 2))
> df2
   V1 V2
1   7 12
2   8 13
3   9 14
4  10 15
5  11  7

df <- rbind(df1, df2)
> df
   V1 V2
1   1  4
2   2  5
3   3  6
4   7 12
5   8 13
6   9 14
7  10 15
8  11  7
```

2.2.6 Quelques statistiques de base

R offre plusieurs fonctions de base pour obtenir des statistiques descriptives comme la moyenne, la médiane, l'écart type, la variance, etc. Dans ce qui suit, nous allons présenter quelques fonctions qui permettent d'obtenir des statistiques de bases sur des données. Pour des raisons de simplicité, nous allons étudier ces fonctions pour un vecteur, et les fonctions sont aussi valables pour des matrices, `dataframes`, etc.

```
# Création d'un vecteur
> vecteur <- c(3.6, 5.9, 8.66, 15, 11.5, 12.9, 14, 6.2, 10, 7.8)
> vecteur
 [1]  3.60  5.90  8.66 15.00 11.50 12.90 14.00  6.20 10.00  7.80

# Moyenne
> mean(vecteur)
[1] 9.556

# Médiane
> median(vecteur)
[1] 9.33

# Écart type
> sd(vecteur)
[1] 3.779098

# Variance
> var(vecteur)
[1] 14.28158

# Racine carrée
> sqrt(vecteur)
 [1] 1.897367 2.428992 2.942788 3.872983 3.391165 3.591657 ↩
    3.741657 2.489980
 [9] 3.162278 2.792848

# Quartiles
> quantile(vecteur)
   0%   25%   50%   75%  100%
 3.60  6.60  9.33 12.55 15.00

# Minimum
> min(vecteur)
[1] 3.6
```

```
# Maximum
> max(vecteur)
[1] 15

# Résumé des statistiques descriptives
> summary(vecteur)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  3.600   6.600   9.330   9.556  12.550  15.000
```

Notez que les fonctions de statistique présentées ci-dessus peuvent avoir également des paramètres. Par exemple, la fonction `mean` peut avoir l'argument `na.rm = T` qui indique d'omettre les valeurs manquantes lors du calcul de la moyenne.

2.2.7 Visualisation des données

La visualisation des données est primordiale pour comprendre les données et leur variation. Il existe deux types de visualisation :

- Visualisation exploratoire qui nous aide à comprendre les données.
- Visualisation explicative qui nous aide à partager les résultats trouvés avec les autres.

Pour chaque type de visualisation, nous avons beaucoup de formes de visualisation, comme nous pouvons le voir par la suite dans le module de visualisation des données, à savoir les histogrammes, les nuages de points, les diagrammes de batons, etc. L'exemple suivant présentes des graphiques des deux types de visualisation.

► **Exemple 2** Visualisation exploratoire : Dans la base données "Icecream.csv", la fonction plot() permet de visualiser les relations entre les variables comme le montre la figure suivante.

```
# Lecture du fichier Icecream.csv
MyData <- read.csv("icecream.csv", header = T, sep = ",")
plot(MyData)
```

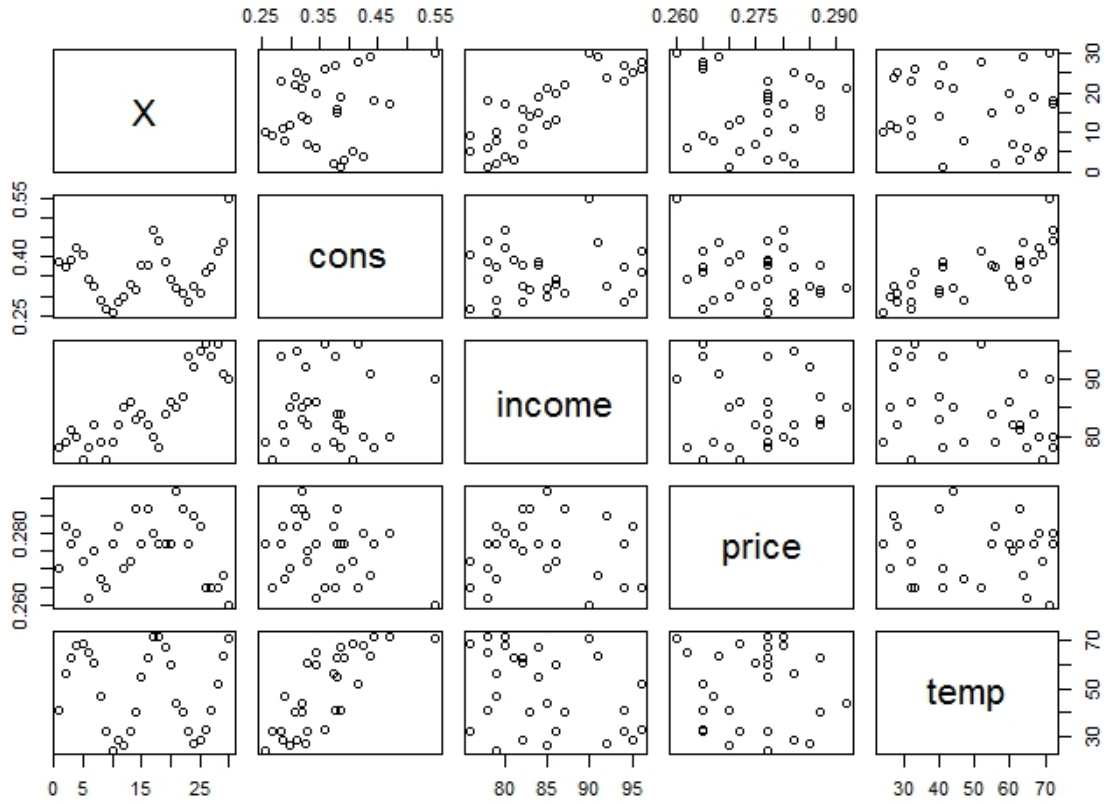

Figure 2.4: Visualisation exploratoire des données.

Ce graphique nous montre si des correlations existent entre les variables.

▶ **Exemple 3** Visualisation explicative : avec la même fonction `plot`, nous pouvons montrer aux autres l'existence des relations importantes entre quelques variables dans les données. Par exemple, avec les données du fichier "iris.csv", nous pouvons voir la correlation entre la catégorie des fleurs et la longueur comme suit :

```r
# Lecture du fichier iris.csv
MyData <- read.csv("iris.csv", header = T, sep = ",")
plot(plot(MyData$species, MyData$sepal_length))
```

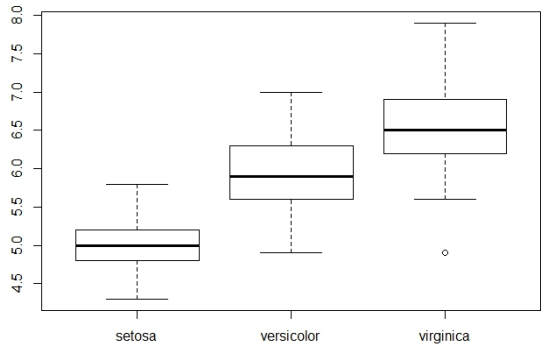

FIGURE 2.5: Visualisation explicative par un boxplot des données.

Nous pouvons également visualiser la relation entre les deux variables de longueur et largeur comme suit :

```r
plot(MyData$petal_length, MyData$petal_width)
```

Vous allez voir dans le module 4 du cours plusieurs formes de graphiques qui peuvent être utilisées pour la visualisation exploratoire et explicative.

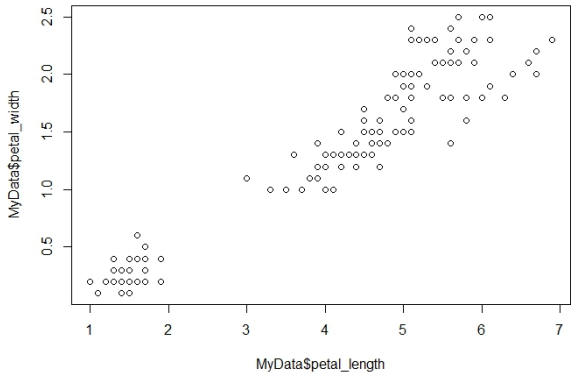

Figure 2.6: Visualisation explicative par un nuage de points des deux variables longueur et largeur.