



FACULTE DES SCIENCES BEN M'SICK
UNIVERSITÉ HASSAN II DE CASABLANCA

HASSAN II UNIVERSITY
FACULTY OF SCIENCES BEN M'SICK

Langage R et la théorie de la statistique

Authors:
El Mehdi BELLFKIH

23 Mars 2023

Table des matières

1	L’installation R et RStudio	2
2	l’initiation	3
2.1	L’interface de RStudio	3
2.2	L’initiation	4
2.3	Les packages	4
2.4	Les types et les structure de données	5
2.4.1	Vecteurs	7
2.4.2	Matrices	9
2.4.3	Tableaux	11
2.4.4	Cadres de données	12
2.4.5	Listes	14
3	La programmation dans R	16
3.1	Les conditions et boucles	16
3.1.1	Conditions	16
3.1.2	La boucle For	17
3.1.3	La boucle While	17
3.2	Les fonctions	18
4	R markdown	21
5	La statistique descriptive	25
5.1	Mesures	25
5.2	Visualisation des données	27
5.2.1	plot()	27
5.2.2	ggplot2	35
6	La statistique inférentielle	41
6.1	échantillonnage	41
6.2	Test d’hypothèse	44
6.3	tests statistiques	49
6.3.1	Deux variables qualitatives	49
6.3.2	Une variable quantitative et une variable qualitative	50
6.3.3	Deux variables quantitatives	52
7	Les modèles de régressions	54
7.1	La régression linéaire simple	54

7.1.1	Modélisation mathématique	54
7.1.2	Le modèle	54
7.2	la régression linéaire multiple	58
7.2.1	Modélisation mathématique	58
7.2.2	Le modèle	59
7.3	La régression des moindres carrés partiels	60

Introduction

La plupart des langages de programmation sont créés avec des objectifs spécifiques en tête. Ces objectifs aident à définir les caractéristiques du langage et déterminent ce que vous pouvez faire avec le langage. Il n'existe pas vraiment de moyen de créer un langage de programmation qui tout faire, car les gens ont des objectifs et des besoins concurrents lorsqu'ils créent des applications. En ce qui concerne langage R, l'objectif principal était de créer un langage de programmation qui rendrait les programmeurs efficaces.

Demander des informations à l'ordinateur est une chose, mais lui fournir des instructions en est une autre. La programmation est simplement un type de communication similaire à d'autres d'autres formes de communication que vous avez déjà avec votre ordinateur. Parler à une machine peut sembler étrange au premier abord, mais c'est nécessaire car un ordinateur ne peut pas lire dans votre esprit. ordinateur ne peut pas lire dans vos pensées pas encore. Même si l'ordinateur lisait dans vos pensées, il serait toujours en train de communiquer avec vous. Rien ne peut se produire sans un échange d'informations entre la machine et vous.

Lorsque vous écrivez un script, vous écrivez une procédure qui définit une série d'étapes que l'ordinateur doit exécuter pour accomplir la tâche que vous avez en tête. Si vous omettez une étape, les résultats ne seront pas ceux que vous attendiez. L'ordinateur ne saura pas ce que vous voulez dire ou que vous vouliez qu'il exécute certaines tâches automatiquement. La seule chose que l'ordinateur sait, c'est que vous lui avez fourni une procédure spécifique et qu'il l'a exécutée. que vous lui avez fourni une procédure spécifique et qu'il doit l'exécuter.

Les spécialistes des données ont souvent du mal à choisir entre R et Python, car les deux langages sont adaptés à l'analyse statistique et au type de graphiques dont les spécialistes des données ont besoin pour comprendre les modèles de données. Les deux langages sont également open source et prennent en charge un large éventail de plateformes. Cependant, langage R est un peu plus spécialisé que Python dans la prtie concernant la théorie de la statistique et tend à s'adresser au secteur universitaire.

Chapitre 1

L'installation R et RStudio

Pour écrire une application à l'aide du langage de programmation R, vous avez besoin des applications requises pour le faire. Ces applications vous aident à travailler avec R en créant du code R, en fournissant des informations d'aide lorsque vous en avez besoin et en vous permettant d'exécuter le code que vous avez écrit. Vous devez disposer d'une version de R qui fonctionne sur votre plate-forme particulière. R est un environnement logiciel libre pour le calcul statistique et les graphiques. Il se compile et fonctionne sur une grande variété de plateformes UNIX, Windows et MacOS.

Pour obtenir la bonne version pour votre plateforme, vous devez vous rendre à l'adresse suivante <https://www.r-project.org/> et choisissez votre miroir CRAN préféré <https://cran.r-project.org/mirrors.html> L'installation est très facile, il suffit de suivre les instructions.

R n'est pas un programme que vous pouvez ouvrir et commencer à utiliser. Il s'agit plutôt d'un langage informatique, comme Python, C++ ou d'autres langages. Vous utilisez R en écrivant des commandes dans le langage R et en demandant à votre ordinateur de les interpréter. Aujourd'hui, presque tout le monde utilise R avec une application appelée RStudio, qui sera utilisée dans le reste de ce document. RStudio est un environnement de développement intégré pour R, un langage de programmation pour le calcul statistique et les graphiques. RStudio est une application qui vous aide à écrire en R. Elle rend l'utilisation de R beaucoup plus facile. De plus, l'interface de RStudio est la même pour Windows, Mac OS et Linux. Vous pouvez télécharger RStudio gratuitement à partir de <https://posit.co/downloads/>. Une fois que vous avez installé, vous pouvez l'ouvrir comme n'importe quel autre programme sur votre ordinateur en cliquant sur une icône sur votre bureau.

Chapitre 2

l'initiation

2.1 L'interface de RStudio

Une fois vous ouvrez RStudio, une fenêtre apparaît avec trois volets, comme dans la Figure 2.1. Le plus grand volet est une fenêtre de console. C'est là que vous exécuterez votre code R et que vous verrez les résultats. La fenêtre de la console est exactement ce que vous verriez si vous exécutez R à partir d'une console UNIX ou des interfaces graphiques Windows ou Mac. Tout le reste est propre à RStudio. Les autres volets cachent un éditeur de texte, une fenêtre graphique, un débogueur, un gestionnaire de fichiers, et bien plus encore.

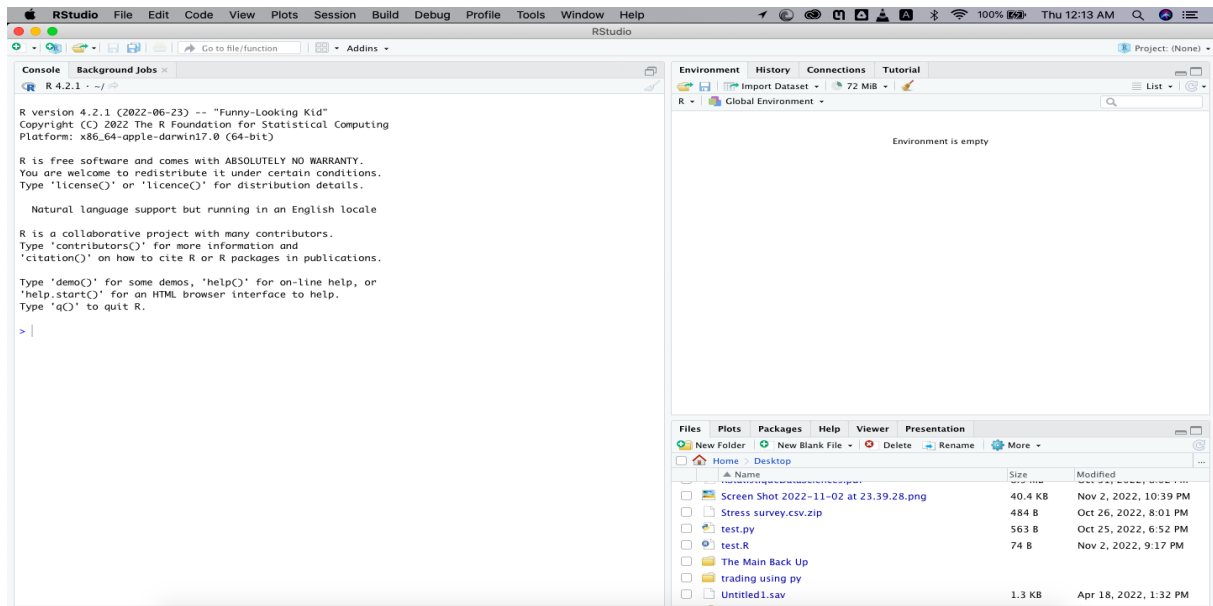


FIGURE 2.1 – L'IDE RStudio pour R

De maintenant que vous avez R et RStudio sur votre ordinateur, vous pouvez commencer à utiliser R en ouvrant le programme RStudio.

2.2 L'initiation

Une commande est simplement une étape dans une procédure. Lorsque vous travaillez avec R, une commande, telle que `print()`, est simplement la même chose ; une étape dans une procédure. Pour dire à l'ordinateur ce qu'il doit faire, comme dans l'exemple suivant pour imprimer "Hello Ben M'sick".

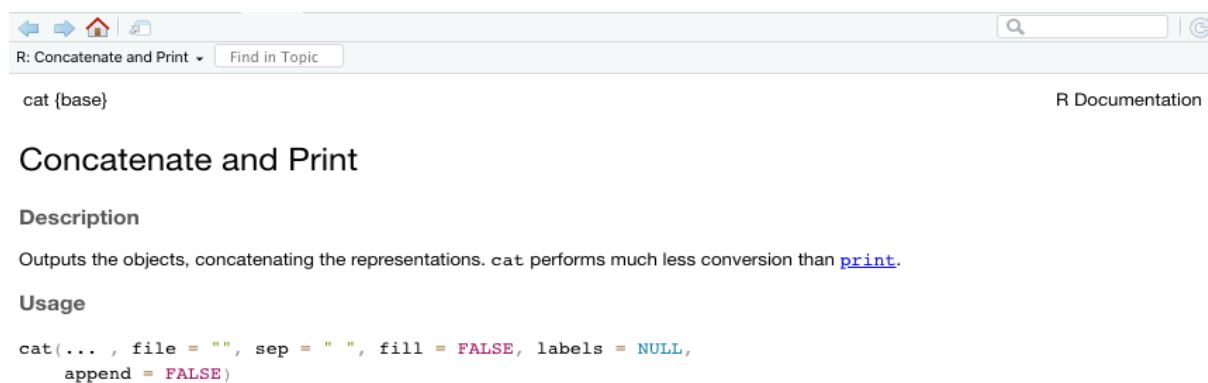
```
1 print("Hello Ben M'sick")
```

Output:

```
[1] "Hello Ben M'sick"
```

Lorsque vous tapez une commande dans la console et que vous appuyez sur Entrée, votre ordinateur exécute la commande et vous montre les résultats. Ensuite, RStudio affiche un résultat de votre commande, Vous remarquerez qu'un [1] apparaît à côté de votre résultat. R vous indique simplement que cette ligne commence par la première valeur de votre résultat. C'est généralement une bonne idée d'essayer de vous aider vous-même. R comprend de nombreuses fonctions permettant d'accéder à la documentation et de rechercher de l'aide. La fonction `help()` et l'opérateur `?` de R permettent d'accéder aux pages de documentation des fonctions, des ensembles de données et d'autres objets de R, tant pour les packages de la distribution standard de R que pour les packages contribués.

```
1 help("cat") #ou tapez help(cat), ?cat ?"cat"
```



2.3 Les packages

Un package, est simplement un ensemble de programmes qui complète et permet d'augmenter les fonctionnalités de R . Un package est généralement dévolu à une méthode particulière ou à un domaine d'application spécifique. L'installation est à faire une seule fois. Les packages sont disponibles sur le réseau du CRAN [http ://cran.r-project.org/](http://cran.r-project.org/) Pour installer un package et aussi faire mise a jour on utilise la fonction suivante

```
1 > install.packages("PackageName")
2 > update.packages() #Mettre à jour tous les packages disponibles
```

```
3 > update.packages(ask = FALSE) #Mettre à jour sans intervention de l'utilisateur.
```

Les fonctions du `NomPackage` sont alors disponibles pour être utilisées. Pour cela y a trois modes

```
1 > library(PackageName)
2 > require(PackageName)
3 > PackageName::FonctionName #Appel directement la fonction, sans charger PackageName
```

La différence entre `library` et `require` est que `library` retourne une erreur et `require` retourne un avertissement au cas où un paquet n'est pas encore installé.

2.4 Les types et les structure de données

R utilise des fonctions ou des opérateurs qui agissent sur des objets (vecteurs, matrices, listes, etc.). R est un langage orienté objet, alors, un objet en R est tout ce qui peut être affecté à une variable (constantes, structures de données, fonctions, graphiques)

- Objets de données : utilisés pour stocker des valeurs numériques réelles ou complexes, des valeurs logiques ou des caractères. Ces objets sont toujours des vecteurs.
- Objets du langage : fonctions, expressions.

Des objets peuvent être créés en affectant avec l'un des trois opérateurs "`<-`", "`>`", "`=`" en nommant cet objet. Le symbole "`#`" indique que le reste de la ligne ne sera pas interprété par R (cela nous permet de ajouter des commentaires dans notre programme). L'affectation créera l'objet s'il n'existe pas. Sinon, l'affectation écrase la valeur précédente sans avertissement.

```
1 var <- pi #var reçoit pi, On peut utiliser = ou bien pi -> var
2 print(var) # Affichage de var, comme on peut taper seulement var
```

Output:

```
[1] 3.141593
```

Par défaut, R conserve tous les objets créés dans une session en mémoire. Utilisez les fonctions `objects()` ou `ls()` pour accéder aux objets de la session, pour en supprimer ces objet.

```
1 > rm(var) #Suppression l'objet var
2 > rm(var1,var2,...,varn) #Suppression les objets var1, var2,..., et varn
3 > rm(list=ls()) # suppression tous les objets
```

On peut connaître les modes de ces objets par la commande `typeof()`. On peut aussi facilement vérifier le mode particulier d'un objet, Le résultat est un booléen. Et même de convertir un objet `var` d'un mode à un autre.

```

1 > mode(var) #Donne le mode null, logical, numeric, complex ou character.
2 > is.null(var)
3 > is.logical(var)
4 > is.numeric(var)
5 > is.complex(var)
6 > is.character(var)
7 > as.logical(var)
8 > as.numeric(var)
9 > as.complex(var)
10 > as.character(var)

```

mais R reconnaît également un autre types : brut. Il est peu probable que vous les utilisiez un jour pour analyser des données.

```

1 > raw(5)
2 [1] 00 00 00 00 00
3 > mode(raw(5))
4 [1] "raw"

```

Dans R on a cinq majeurs types de structures de données comme dans la Figure 2.2 :

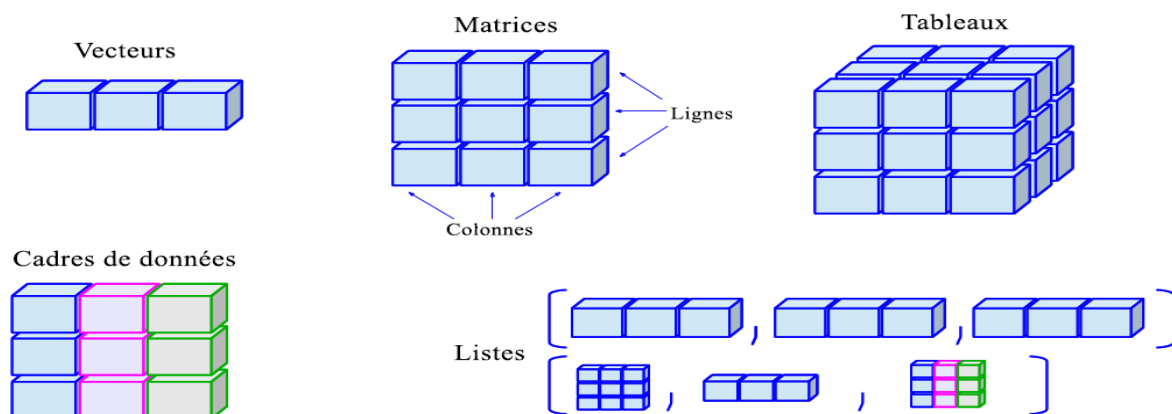


FIGURE 2.2 – Types de structures de données

- Vecteurs : tableaux unidimensionnels utilisés pour stocker une collection de données de même mode.
- Matrices : tableaux à deux dimensions permettant de stocker des collections de données de même mode. On y accède par deux indices entiers.
- Tableaux : similaires aux matrices mais ils peuvent être multidimensionnels (plus de deux dimensions).
- Cadres de données : généralisation des matrices où différentes colonnes peuvent stocker des données de mode différent.
- Listes : collection ordonnée d'objets, dont les éléments peuvent être de différents types.

2.4.1 Vecteurs

Il existe plusieurs façons d'assigner des valeurs à une variable :

```
1 > vec <- 20.7 #Assigne une valeur à un vecteur à un seul élément
2 > print(vec)
3 [1] 20.7
4 > vec <- c(20, 7, 33) #Assigne 3 valeurs à un vecteur par la commande de concaténation c()
5 [1] 20 7 33
6 > vec * 2 #Evalue la valeur carrée de chaque élément du vecteur
7 [1] 400 49 1089
8 > sqrt(vec) #Evalue la racine carrée de chaque élément du vecteur
9 [1] 4.472136 2.645751 5.744563
10 > vec - 2 #soustrayez 2 de chaque élément du vecteur
11 [1] 18 5 31
```

Aussi, il y a la fonction `scan` qui nous permet de créer un vecteurs où ses éléments seront collectés.

```
1 > vec <- scan(n=3)
2 1: 20
3 2: 7
4 3: 33
5 Read 3 items
6 > vec
7 [1] 20 7 33
```

On peut générer des vecteurs par des séquences et des répétitions en utilisant les commandes `seq()`, `rep()`

```
1 > seqvec1 <- 7:20 #Générer une séquence de 7 à 20
2 > seqvec1
3 [1] 7 8 9 10 11 12 13 14 15 16 17 18 19 20
4 > seqvec2 <- 20:7 #Générer une séquence de 20 à 7
5 > seqvec2
6 [1] 20 19 18 17 16 15 14 13 12 11 10 9 8 7
7 > seqvec3 <- seq(from=7, to=20, by=2) #Génère la séquence de 7 à 20 en utilisant l'étape 2
8 > seqvec3
9 [1] 7 9 11 13 15 17 19
10 > seqvec4 <- seq(length=7, from=7, by=2) #Génère une séquence de longueur fixe
11 > seqvec4
12 [1] 7 9 11 13 15 17 19
13 > length(seqvec2);length(seqvec3) #Savoir la longueur de chaque vecteur
14 [1] 14
15 [1] 7
16 > length(seqvec1)==length(seqvec2) #Comparer si les vecteurs de même longueur
17 [1] TRUE
18 > length(seqvec2)==length(seqvec3)
```

```

19 [1] FALSE
20 > vec <- 1:5; seqvec5 <- rep(vec, times=3); seqvec6 <- rep(vec, each=3) #
21 > vec;seqvec5;seqvec6
22 [1] 1 2 3 4 5
23 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
24 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5

```

Il faut faire attention quand on combine les vecteurs de tailles différentes, ils peuvent être combinés, dans la mesure où la longueur du vecteur le plus long est un multiple de la longueur du vecteur le plus court sinon un avertissement est émis.

```

1 > vec1 <- 1:5
2 > vec2 <- 11:20
3 > vec3 <- 6:9
4 > vec1+vec2 #Les dimensions propres
5 [1] 12 14 16 18 20 17 19 21 23 25
6 > vec1+vec3 #Les dimensions incorrectes
7 [1] 7 9 11 13 11
8 Warning message:
9 In vec1 + vec3 :
10 longer object length is not a multiple of shorter object length

```

On peut créer un vecteur logique et un vecteur de caractères par plusieurs Façons

```

1 > vec1 <- 1:9
2 > vec2 <- vec1 > sqrt(11) #Assigne des valeurs à partir d'une inéquation
3 > vec2
4 [1] FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE TRUE
5 > vec1[vec2] #Affiche les valeurs qui remplissent la condition
6 [1] 4 5 6 7 8 9
7 > vec3 <- c("B","E","L")
8 > vec3
9 [1] "B" "E" "L"
10 > paste(vec3,"txt",sep = ".") #Concaténer vec3 et "text" par "."
11 [1] "B.txt" "E.txt" "L.txt"
12 > paste(vec3,collapse = "")
13 [1] "BEL"

```

L'argument `collapse` rassemble tous les éléments dans un vecteur de longueur 1.

Les facteurs sont des vecteurs permettant la manipulation de données qualitatives. La longueur est donnée par la fonction `length`, le mode par `mode` et les modalités du facteur par `levels`.

```

1 > vec1 <- factor(c("White","White","Black","White","Black","White","Black","White"))
2 > vec1
3 [1] White White Black White Black White Black White
4 Levels: Black White
5 > length(vec1); levels(vec1) #La longueur et les niveaux d'un vecteur

```

```

6 [1] 8
7 [1] "Black" "White"
8 > vec2 <- factor(c("White","White","Black","White","Black","White","Black","White"),
9 +               labels =c("111111","000000")) #nommer les niveaux lors de la construction
10 > vec2
11 [1] 000000 000000 111111 000000 111111 000000 111111 000000
12 Levels: 111111 000000
13 > table(vec1)
14 vec1
15 Black White
16      3      5
17 > table(vec2)
18 vec2
19 111111 000000
20      3      5

```

2.4.2 Matrices

Les matrices sont des objets atomiques. Autrement dit, le même mode ou type pour toutes les valeurs. Chaque valeur de la matrice peut être identifiée par son numéro de ligne et son numéro de colonne. Deux attributs uniques de l'objet R sont la longueur, qui correspond ici au nombre total d'éléments de la matrice, et le mode, qui correspond au mode des éléments de cette matrice. Les matrices ont également un attribut de dimension `dim` qui renvoie le nombre de lignes et de colonnes.

```

1 > mat1 <- matrix(1:12, nrow=3, ncol=4) #Matrice de 3 lignes et 4 coloones
2 > mat1
3      [,1] [,2] [,3] [,4]
4 [1,]    2    5    8   11
5 [2,]    3    6    9   12
6 [3,]    4    7   10   13
7 > length(mat1);dim(mat1) #renvoie la longueur et les dimensions de la matrice
8 [1] 12
9 [1] 3 4 #(lignes,colonnes)
10 > mat2 <- matrix(2:13,nrow =3) #R ranger la matrice par colonne par default
11 > mat2
12      [,1] [,2] [,3] [,4]
13 [1,]    2    5    8   11
14 [2,]    3    6    9   12
15 [3,]    4    7   10   13
16 > mat3 <- matrix(2:13,nrow =3,byrow = TRUE) #Ranger la matrice par ligne
17 > mat3
18      [,1] [,2] [,3] [,4]
19 [1,]    2    3    4    5
20 [2,]    6    7    8    9
21 [3,]   10   11   12   13
22 > mat4 <- seq(7,20,by=3)
23 > mat4

```

```

24 [1] 7 10 13 16 19
25 > as.matrix(mat4)
26      [,1]
27 [1,]    7
28 [2,]   10
29 [3,]   13
30 [4,]   16
31 [5,]   19

```

La position d'un élément dans la matrice `mat` est généralement indiquée par son numéro de ligne et de colonne. Par conséquent, l'élément (i,j) de la matrice est sélectionné par `mat[i,j]`. On sélectionne une ou plusieurs lignes et/ou une ou plusieurs colonnes.

```

1 > mat <- matrix(2:7,nrow=2,ncol=3);
2 > mat
3      [,1] [,2] [,3]
4 [1,]    2    4    6
5 [2,]    3    5    7
6 > mat[,2] #Selectionner 2ème colonne
7 [1] 4 5
8 > mat[2,] #Selectionner 1ière ligne
9 [1] 3 5 7
10 > mat[1,2:3] #Selectionner les valeurs de toutes lignes et 2ème et 3ème colonnes
11      [,1] [,2]
12 [1,]    4    6
13 [2,]    5    7
14 > mat[mat>4] #Selectionner les valeurs de mat qui vérifient la conditions
15 [1] 5 6 7

```

On présente les fonctions les plus utilisées dans les calculs matriciels.

```

1 > mat1=matrix(2:5,nrow =2);mat2=matrix(3:6,nrow = 2)
2 > mat1+mat2 #La somme de deux matrices
3      [,1] [,2]
4 [1,]    5    9
5 [2,]    7   11
6 > mat1*mat2 #Produit élément par élément
7      [,1] [,2]
8 [1,]    6   20
9 [2,]   12   30
10 > mat1%%mat2 #Le produit de deux matrices
11      [,1] [,2]
12 [1,]   22   34
13 [2,]   29   45
14 > sqrt(mat1) #La racine carrée élément par élément
15      [,1] [,2]
16 [1,] 1.414214 2.000000
17 [2,] 1.732051 2.236068

```

```

18 > det(mat1) #le déterminant de la matrice mat
19 [1] -2

```

Le tableau suivant donne les principales fonctions utiles en algèbre linéaire

Fonction	Description
mat1%*%mat2	produit de matrices
t(mat)	transposition d'une matrice
diag(5)	matrice identité d'ordre 5
diag(vec)	matrice diagonale avec les valeurs du vec dans la diagonale
crossprod(mat1,mat1)	produit croisé (t(mat1)%*%mat2)
det(mat)	déterminant de la matrice X
svd(mat)	décomposition en valeurs singulières
eigen(mat)	diagonalisation d'une matrice
solve(mat)	inversion de matrice
solve(A,b)	résolution de système linéaire
chol(mat)	décomposition de Cholesky
qr(mat)	décomposition QR

TABLE 2.1 – Les principales fonctions utiles en algèbre linéaire.

Exemple : Résoudre le système d'équations

$$\begin{cases} 20x + 9y = 33 \\ 7x + 22y = 19 \end{cases}$$

Revient

$$\underbrace{\begin{pmatrix} 20 & 9 \\ 7 & 22 \end{pmatrix}}_A \underbrace{\begin{pmatrix} x \\ y \end{pmatrix}}_X = \underbrace{\begin{pmatrix} 33 \\ 19 \end{pmatrix}}_b$$

```

1 > A <- matrix(c(20, 7, 9, 22),2); b <- c(33,19)
2 > solve(A,b)
3 [1] 1.4721485 0.3952255

```

Donc la solution du système d'équations est (1.4721485, 0.3952255)

2.4.3 Tableaux

la fonction `array` crée un tableau n-dimensionnel. Vous pouvez utiliser tableau pour trier les valeurs dans un cube de trois dimensions ou un hypercube en n dimension. tableau n'est pas aussi personnalisable que la matrice et fait essentiellement la même chose que la configuration de l'attribut `dim`.

```

1 > arr1 <- array(c(2:5, 7:10, 11:14), dim = c(2, 2, 3))
2 > arr1

```

```

3  , , 1
4
5      [,1] [,2]
6  [1,]    2    4
7  [2,]    3    5
8
9  , , 2
10
11     [,1] [,2]
12 [1,]    7    9
13 [2,]    8   10
14
15 , , 3
16
17     [,1] [,2]
18 [1,]   11   13
19 [2,]   12   14
20 > dim(arr1); length(arr1)
21 [1] 2 2 3
22 [1] 12
23 > arr2 <- array(2:31,dim=c(3,5,2))
24 > arr2
25 , , 1
26
27     [,1] [,2] [,3] [,4] [,5]
28 [1,]    2    5    8   11   14
29 [2,]    3    6    9   12   15
30 [3,]    4    7   10   13   16
31
32 , , 2
33
34     [,1] [,2] [,3] [,4] [,5]
35 [1,]   17   20   23   26   29
36 [2,]   18   21   24   27   30
37 [3,]   19   22   25   28   31

```

Sélection d'éléments ou d'une partie d'un tableau et la matrice sont la même

2.4.4 Cadres de données

Les cadres de données sont la version bidimensionnelle d'une liste. Ils sont de loin la structure de stockage la plus utile pour l'analyse des données, et ils constituent un moyen idéal de stocker des données complètes. Vous pouvez considérer un cadre de données comme l'équivalent pour R d'une feuille de calcul Excel, car il stocke les données dans un format similaire. Les cadres de données regroupent les vecteurs dans un tableau bidimensionnel. Chaque vecteur devient une colonne dans le tableau. Par conséquent, chaque colonne d'un cadre de données peut contenir un type de données différent, mais dans une colonne, chaque cellule doit être du même type.

La création d'un cadre de données à la main demande beaucoup de frappe, mais vous

pouvez le faire avec la fonction `data.frame`. Donnez à `data.frame` un nombre quelconque de vecteurs, chacun séparé par une virgule. Chaque vecteur doit être égal à un nom qui le décrit. `data.frame` transformera chaque vecteur en une colonne du nouveau cadre de données

```
1 > df <- data.frame(score=c(20,7,33), type=c("B", "E", "L"))
2 > df
3   score type
4 1    20    B
5 2     7    E
6 3    33    L
```

En fait, chaque cadre de données est une liste de classe `data.frame`. Vous pouvez voir quels types d'objets sont regroupés par data frame avec la fonction `str`

```
1 > typeof(df); class(df) #
2 [1] "list"
3 [1] "data.frame"
4 > str(df)
5 'data.frame':      3 obs. of  2 variables:
6 $ score: num  20 7 33
7 $ type : chr  "B" "E" "L"
```

On a aussi la fonction `read.table` qui permet nous d'importer un cadre de données ; `as.data.frame` pour la conversion explicite.

```
1 > vec <- c(20,33,7,19); fac <- factor(c("B", "E", "L", "E")); boo <- c(T,F,T,F)
2 > df <- as.data.frame(cbind(vec,fac,boo)) #Fonction cbind pour fusionner les colonnes.
3 > df
4   vec fac boo
5 1  20  1  1
6 2  33  2  0
7 3   7  3  1
8 4  19  2  0
```

Il exist aussi `rbind` une fonction pour combiner des vecteurs, des matrices et/ou des cadres de données par lignes

```
1 > vec1 <- 2:3; vec2 <- 0:7
2 > df1 <- cbind.data.frame(vec1,vec2)
3 > df1
4   vec1 vec2
5 1     2    0
6 2     3    1
7 3     2    2
8 4     3    3
9 5     2    4
```

```

10 6    3    5
11 7    2    6
12 8    3    7

```

2.4.5 Listes

Une Liste est un ensemble ordonné d'objets qui n'ont pas toujours même mode ou même longueur, elle est un objet hétérogène. Les objets d'une liste peuvent avoir un nom. Elle a trois attributs `length`, `mode` et `names`. Les listes sont des objets importants et très utilisée car toutes les fonctions qui retournent plusieurs objets le font sous la forme d'une liste. Pour créer une liste on utilise la fonction `list`.

```

1  > vec <- c(20,7,33,19)
2  > fac <- factor(c("B","E", "L"))
3  > mat <- matrix(2:13,nrow = 3)
4  > df <- data.frame(Score=c(7,20,33),Type=c("e","l","b"))
5  > lis <- list(vec,fac,mat,df)
6  > lis
7  [[1]]
8  [1] 20  7 33 19
9
10 [[2]]
11 [1] B E L
12 Levels: B E L
13
14 [[3]]
15      [,1] [,2] [,3] [,4]
16 [1,]    2    5    8   11
17 [2,]    3    6    9   12
18 [3,]    4    7   10   13
19
20 [[4]]
21      Score Type
22 1       7    e
23 2      20    l
24 3      33    b
25 > length(lis);mode(lis);names(lis)
26 [1] 4
27 [1] "list"
28 NULL #C'est normal, on a pas assigné les noms des composantes
29
30 > names(lis) <- c("Vecteur", "facteur", "matrice", "dfscores")
31 > names(lis)
32 [1] "Vecteur" "facteur" "matrice" "dfscores"

```

Pour extraire une composante de la liste, se fait par la position ou bien par le nom de la composante

```
1 > lis[[2]]
2 [1] B E L
3 Levels: B E L
4 > lis[["facteur"]]
5 [1] B E L
6 Levels: B E L
7 > length(unlist(lis)) #unlist créer un seul vecteur contenant tous les éléments de la liste
8 [1] 25
```

il est possible de faire appeler la composante "facteur" par "> lis\$facteur", comme on peut extraire plusieurs composantes par exemple le "facteur" et "matrice" en utilisant la commande "lis[c(2,3)]"

Chapitre 3

La programmation dans R

Après avoir appris les bases, la programmation en R est la prochaine grande étape pour atteindre le nirvana du codage. Dans R, cette liberté s'acquiert en étant capable de manier et de manipuler le code pour qu'il fasse exactement ce que vous voulez, plutôt que de vous contorsionner, de vous plier et de vous restreindre à ce que d'autres ont construit. Il est bon de se rappeler que de nombreux paquets sont écrits dans le but de résoudre un problème que l'auteur avait. Votre problème peut être similaire, mais pas exactement le même. Dans ce cas, vous pouvez déformer, plier et restreindre vos données pour qu'elles fonctionnent avec le packaging, ou vous pouvez construire vos propres fonctions pour résoudre vos problèmes spécifiques.

3.1 Les conditions et boucles

3.1.1 Conditions

La prise de décision est une partie importante de la programmation. Ceci peut être réalisé en utilisant l'instruction conditionnelle `if` et `else`.

```
1  if (conditions){  
2    statement1  
3  } else{  
4    statement2  
5  }
```

Exemple : tester la parité d'un nombre

```
1  var1 <- 7  
2  var2 <- 20  
3  var3 <- 33  
4  if(var1 > var2 && var2 > var3)  
5  {  
6    print("var1 > var2 > var3")  
7  } else if(var1 < var2 && var2 > var3)  
8  {
```

```

9     print("var1 < var2 > var2")
10 } else if(var1 < var2 && var2 < var3)
11 {
12     print("var1 < var2 < var3")
13 }

```

Output:

```
[1] "var1 < var2 < var3"
```

Les boucles sont utilisées pour gérer les instructions répétitives. Dans R, il est plus pratique d'utiliser la vectorisation, mais les boucles sont assez bonnes pour commencer.

3.1.2 La boucle For

La boucle for permet d'itérer sur un vecteur où le nombre d'itération est connu d'avance, sa syntaxe :

```

1 for (var in vector){
2     statements
3 }

```

Exemple : création d'un vecteur de 7 éléments contenant les 7 puissances de 2.

```

1 vector <- 0 #Initialisation
2 for (var in 1:7){
3     vector[var]=2^var
4 }
5 vector #Affichage

```

Output:

```
[1] 2 4 8 16 32 64 128
```

3.1.3 La boucle While

la boucle while est utilisée quand le nombre d'itérations n'est pas connu d'avance. Elle permet de répéter une instruction tant qu'une condition est satisfaite, sa syntaxe :

```

1 While (conditions){
2     statements
3 }

```

```

1 var<-1
2 while (var^3<=126){

```

```
3  print(var)
4  var=var+1
5  }
```

Exemple : pour afficher les nombres où ses cubes sont inférieurs à 126

Output: 1

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

3.2 Les fonctions

Une fonction est un objet R. Un grand nombre de fonctions sont prédéfinies dans R, par exemple `max` calcule le maximum, `mean` calcule la moyenne, etc. Il est également possible de créer vos propres fonctions. Une fonction en général admet des arguments d'entrée et renvoie un résultat comme sortie. Les arguments de la fonction sont obligatoires ou optionnels. Dans le second cas, ils ont des valeurs par défaut.

Exemple : Prenons l'exemple de la fonction `rnorm` qui génère des nombres aléatoires selon la loi normale. Cette fonction admet trois arguments nombres de valeurs `n`, la moyenne `mean` et l'écart type `sd`, Ces deux derniers sont fixés par défaut à 0 et 1. Autre exemple, la fonction `rpois` qui génère des entiers selon la loi de Poisson, elle admet deux arguments nombres de valeurs `n` et la moyenne (= la variance) λ

```
1 > rnorm(7,2,3) #Génère 7 valeurs selon la loi normale N(2,3)
2 [1] 4.6200110 -1.7232773 -2.8782355 -2.1396351 -0.2170305 2.7954631 6.1950883
3 > set.seed(207) #Objets aléatoires qui peuvent être reproduits
4 > rnorm(7,0,1) #Génère 7 valeurs selon la loi normale N(0,1)
5 [1] -0.3140113 0.2608893 -1.5620907 -0.8896555 -0.2996734 -2.0207122 -0.6152073
6 > set.seed(207)
7 > rnorm(7) #Génère 7 valeurs selon la loi normale N(0,1)
8 [1] -0.3140113 0.2608893 -1.5620907 -0.8896555 -0.2996734 -2.0207122 -0.6152073
9 > rpois(7,5) #Génère 7 entiers selon la loi de Poisson rpois(n, lambda)
10 [1] 4 8 2 3 2 5 2
```

En R, une fonction est créée à l'aide du mot-clé `function`. La syntaxe de base d'une définition de fonction est la suivante

```
1 FuntionName = function(arg1,arg2,...,agrn){
2  statements
3 }
```

Le mode est la valeur la plus fréquente d'une série de données. R ne propose pas de formule directe pour calculer le mode et pour cela on va créer une fonction "findmode" qui prend un vecteur comme argument d'entrée. Pour cela on utilise R script qui est simplement un fichier texte contenant un ensemble de commandes et de commentaires (File > New file > R Script). Le script peut être enregistré et utilisé ultérieurement pour réexécuter les commandes enregistrées. Le script peut également être édité afin que vous puissiez exécuter une version modifiée des commandes.

```

1 findmode <- function(vec) {
2   uniqvec <- unique(vec)
3   maxfeq <- uniqvec[which.max(tabulate(match(vec, uniqvec)))]
4   paste("Le mode de la serie est : ", maxfeq)
5 }

```

Après l'exécution du script, on peut appeler la fonction findmode afin de déterminer le mode d'une série.

```

1 > vec <- c(20,7,20,20,7,7,20,7,20,33,20,7,20,20,20,20,33)
2 > findmode(vec)
3 [1] "Le mode de est : 20"

```

Il existe des fonctions R spéciales qui peuvent être utilisées pour répéter des instructions dans la ligne de commande et faciliter le processus de programmation :

- lapply : évalue une fonction pour chaque élément d'une liste
- sapply : évalue une fonction pour chaque élément d'une liste en simplifiant le résultat
- appliquer : appliquer une fonction sur les marges d'un tableau (généralement pour appliquer une fonction aux lignes/colonnes d'une matrice)
- tapply : Appliquer une fonction sur des sous-ensembles d'un vecteur (par exemple défini avec un facteur)
- mapply : version multivariée de lapply

```

1 > vec <- runif(19, min=7, max=33) # Générer 19 valeurs entre 7 et 33
2 > fac <- factor(c("B", "B", "B", "E", "E", "B", "E", "B", "B", "E", "L", "B", "E", "L", "E", "B",
3   + "E", "L", "L", "B"))
4 > ran1 <- cut(vec, breaks = 3, include.lowest = TRUE) # Ranger les valeurs de vec en trois intervalles
5 table(ran1) #Fréquence de chaque intervalle
6 #cut(vec,breaks=c(borne1,borne2,borne3,borne3)) ou bien par cut(vec,quantile(vec,(0:3)/3))
7   #+   en intervalles de même probabilité égale
8 ran1
9 [9.63,17.1] (17.1,24.6] (24.6,32]
10      8      5      6

```

```

1 > lis <- list("B"=runif(6, min=7, max=33),E=runif(6, min=7, max=33),L=runif(6, min=7, max=33))
2 > lapply(lis, mean); sapply(lis, mean) #calcule la moyenne des valeurs pour chaque modalité
3 \ $B
4 [1] 21.61995

```

```

5
6  \ $E
7  [1] 20.18823
8
9  \ $L
10 [1] 23.99149
11
12      B      E      L
13 21.61995 20.18823 23.99149
14 > vec <- runif(18, min=7, max=33)
15 > fac <- factor(c("B","B","B","E", "E","B", "B","B","E", "L","B","E", "L","E","B","E", "L", "L"))
16 > tapply(vec, fac, mean) #Calcule la moyenne des valeurs pour chaque niveau
17      B      E      L
18 19.67144 18.99024 24.25161

```

Chapitre 4

R markdown

R markdown est un langage simple et facile à utiliser pour combiner votre code R, les résultats de votre analyse de données (y compris les graphiques et les tableaux) et les commentaires écrits en un seul document bien formaté et reproductible (comme un rapport, une publication, un chapitre de thèse, etc...) Techniquement, R markdown est une variante d'un autre langage appelé Markdown et tous deux sont un type de langage de "balisage". Un langage de balisage fournit simplement un moyen de créer un fichier de texte brut facile à lire qui peut incorporer du texte formaté, des images, des en-têtes et des liens vers d'autres documents.

Vos données sont importées dans RStudio (ou R) comme précédemment, mais cette fois, tout le code R que vous avez utilisé pour analyser vos données, produire vos graphiques et votre texte écrit (Introduction, Matériaux et méthodes, Discussion, etc.) est contenu dans un seul document R markdown qui est ensuite utilisé (avec vos données) pour créer automatiquement votre document final. C'est exactement ce que R markdown vous permet de faire.

Pour utiliser R markdown, vous devez d'abord installer le **rmarkdown** dans RStudio. Pour la première fois, vous devez créer votre premier document R markdown. Dans RStudio, cliquez sur le menu **File -> New File -> R Markdown** Dans la fenêtre qui s'ouvre, donnez un " Titre " au document, saisissez les informations relatives à l'" Auteur " (votre nom) et sélectionnez un type comme sortie Figure 4.1. Nous pourrions modifier tout cela plus tard, alors ne vous en préoccupez pas pour l'instant.

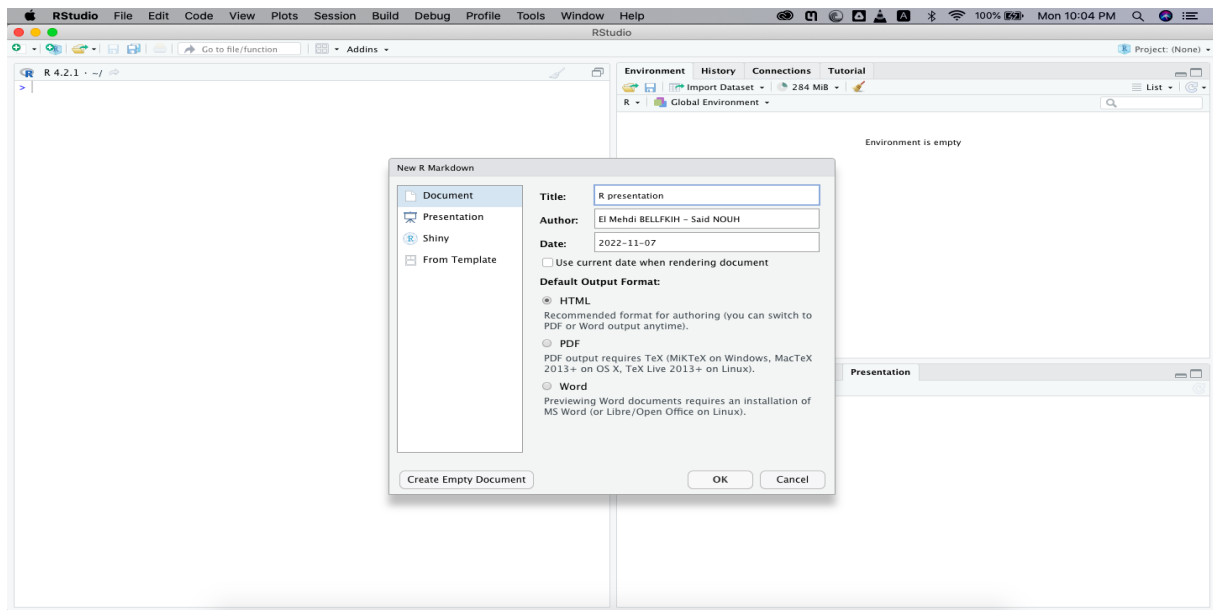


FIGURE 4.1 – Nouveau R markdown

Vous remarquerez que lorsque votre nouveau document R markdown est créé (vous devez enregistrer le document R markdown), il inclut un exemple de code R markdown. Remarquez que l'extension de votre nouveau fichier R markdown est ".Rmd". Normalement, vous devriez simplement mettre en surbrillance et supprimer tout ce qui se trouve dans le document, à l'exception des informations situées en haut entre les délimiteurs `---`, puis commencer à écrire votre propre code à l'aide de la commande.

```

1  ```{r}
2  your code
3  ```

```

puis l'exécuté en cliquant sur icon à droite ou bien CTRL + L.

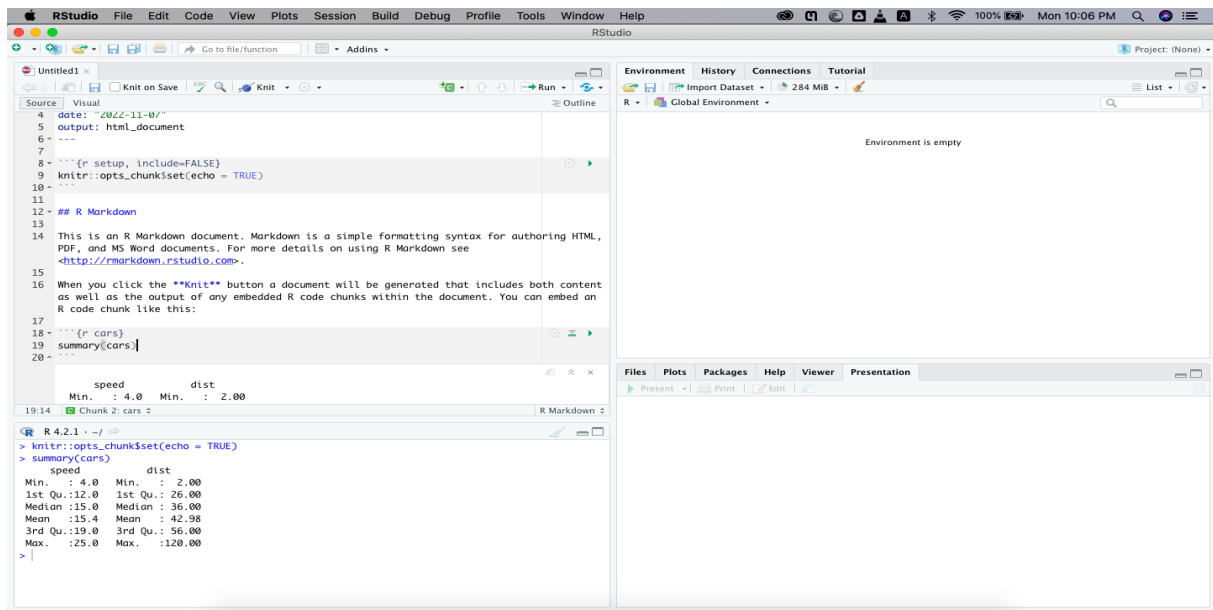


FIGURE 4.2 – Document R markdown

Pour convertir votre fichier ".Rmd" en document PDF, cliquez sur le petit triangle noir à côté de l'icône "Knit" en haut de la fenêtre source et sélectionnez "Knit" vers PDF (voir Figure 4.3).

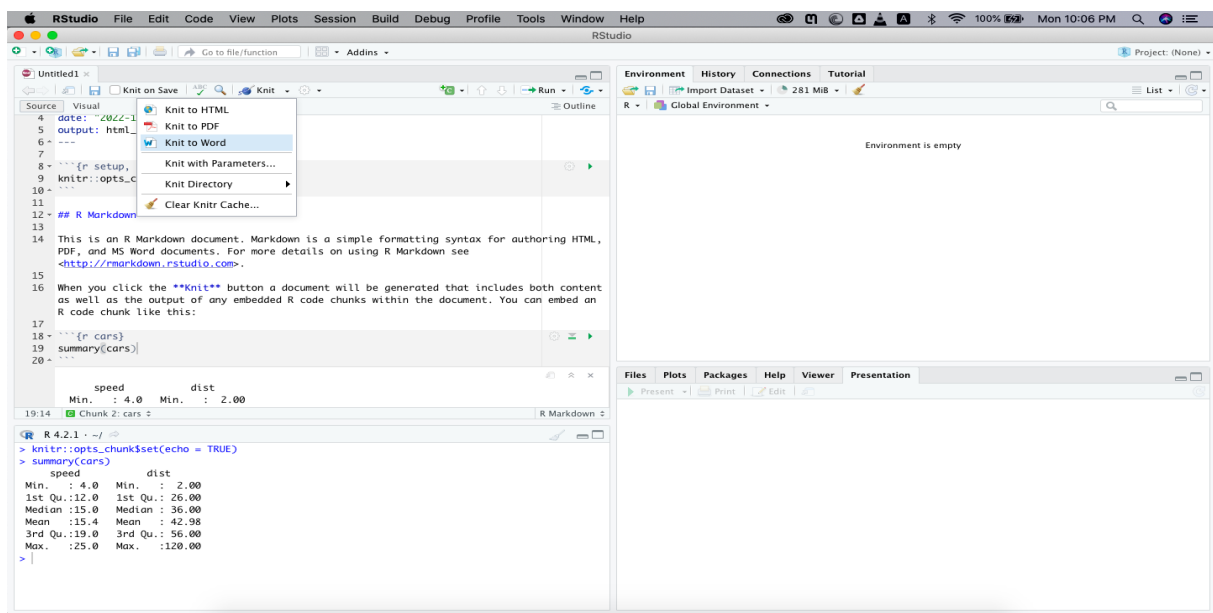


FIGURE 4.3 – la conversion de R markdown

RStudio affichera également un aperçu du fichier rendu dans une nouvelle fenêtre pour que vous puissiez le vérifier Figure 4.4.

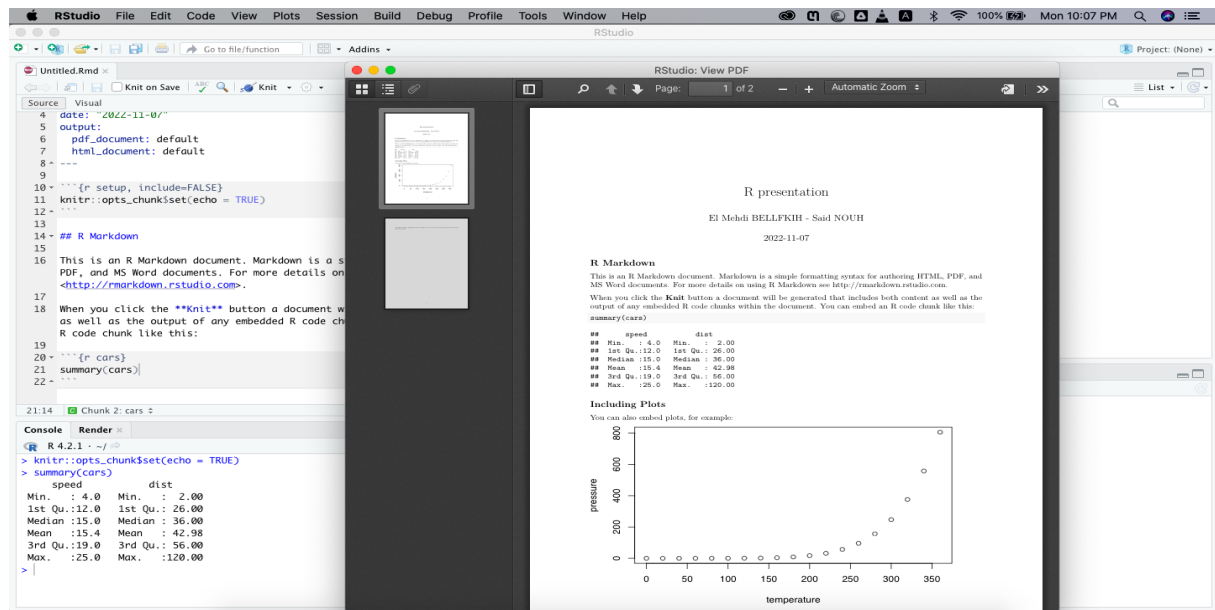


FIGURE 4.4 – Le report sour format PDF

Chapitre 5

La statistique descriptive

5.1 Mesures

L'objet des statistiques est de collecter des observations liées à des objets qui présentent un certain attribut et de convertir ces observations en nombres qui permettent d'avoir des renseignements sur cette propriété. Le but de la statistique descriptive est de structurer et de représenter l'information contenue dans les données. La population est l'ensemble des sujets observés. Le caractère est la propriété étudiée sur ces sujets. Afin de bien présenter cette partie on va utiliser database **iris** (mesures multiples dans les problèmes taxonomiques, contient trois espèces de plantes (setosa, virginica, versicolor))

```
1 > library(datasets) #Charger le package datasets
2 > head(iris) # Fonction head() Renvoyer la première ou la dernière partie d'un objet
3   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
4 1         5.1         3.5         1.4         0.2   setosa
5 2         4.9         3.0         1.4         0.2   setosa
6 3         4.7         3.2         1.3         0.2   setosa
7 4         4.6         3.1         1.5         0.2   setosa
8 5         5.0         3.6         1.4         0.2   setosa
9 6         5.4         3.9         1.7         0.4   setosa
```

La fonction `summary()` permet d'avoir la description statistique d'une variable ou d'une table de donnée.

```
1 > summary(iris)
2   Sepal.Length   Sepal.Width   Petal.Length   Petal.Width   Species
3   Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa   :50
4   1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
5   Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
6   Mean    :5.843   Mean    :3.057   Mean    :3.758   Mean    :1.199
7   3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
8   Max.    :7.900   Max.    :4.400   Max.    :6.900   Max.    :2.500
9 > summary(iris$Sepal.Length) #description statistique pour une seule variable
10   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
11 4.300 5.100 5.800 5.843 6.400 7.900
```

On peut utiliser les fonctions spéciales de R (voir [3.2](#))

```
1 > sapply(iris[1:4], mean) Calcule la moyenne pour les 4 premiers colonnes
2 Sepal.Length Sepal.Width Petal.Length Petal.Width
3 5.843333 3.057333 3.758000 1.199333
4 > sapply(iris[1:4], sd) #Calcule l'ecart type pour premiers colonnes 4 colonnes
5 Sepal.Length Sepal.Width Petal.Length Petal.Width
6 0.8280661 0.4358663 1.7652982 0.7622377
7 > sapply(iris[1:2], quantile) #Calcul du quantile
8 Sepal.Length Sepal.Width
9 0% 4.3 2.0
10 25% 5.1 2.8
11 50% 5.8 3.0
12 75% 6.4 3.3
13 100% 7.9 4.4
14 > sapply(iris[1:2], range) #Calcule de l'etendue
15 Sepal.Length Sepal.Width
16 [1,] 4.3 2.0
17 [2,] 7.9 4.4
18 > table(cut(iris$Sepal.Length, breaks = 3, include.lowest = TRUE))
19 #Range la variable Sepal.Length en trois intervalles
20 [4.3,5.5] (5.5,6.7] (6.7,7.9]
21 59 71 20
```

On est obligé toujours de vérifier si les variables suivent la loi normale, pour cela, il existe une fonction [shapiro.test](#)

```
1 > shapiro.test(iris$Sepal.Length)
2
3 Shapiro-Wilk normality test
4
5 data: iris$Sepal.Length
6 W = 0.97609, p-value = 0.01018
7 > library(moments)
8 > skewness(iris$Sepal.Length)
9 [1] 0.3117531
10 > kurtosis(rnorm(100,0,1))
11 [1] 2.965781
```

le test renvoie une p-value ($0.01018 < 5\%$) significative. la variable ne suit pas une loi normale. La fonction [skewness](#) qui appartient au package [moments](#), la valeur de l'asymétrie est positive ce qui confirme le résultat du [shapiro.test](#), donc cette valeur indique que la distribution (Sepal.Length) est asymétrique à droite, donc on a $\text{mode} < \text{mean} < \text{mean}$. D'autre part on a utilisé la fonction [kurtosis](#) pour avoir une information sur l'aplatissement de la distribution (100 valeurs $\sim \mathcal{N}(0, 1)$), la valeur positive de la fonction [kurtosis](#) indique que nous avons une distribution à longue queue. Il existe de nombreuses statis-

tiques sommaires qui nous donne toute l'information d'une manière simple et puissante, prenons par exemple la fonction `describe()` dans le package `psych` (package qui fournit outils d'analyse des données)

```
1 > library(psych)
2 > describe(iris)
```

	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
Sepal.Length	1	150	5.84	0.83	5.80	5.81	1.04	4.3	7.9	3.6	0.31	-0.61	0.07
Sepal.Width	2	150	3.06	0.44	3.00	3.04	0.44	2.0	4.4	2.4	0.31	0.14	0.04
Petal.Length	3	150	3.76	1.77	4.35	3.76	1.85	1.0	6.9	5.9	-0.27	-1.42	0.14
Petal.Width	4	150	1.20	0.76	1.30	1.18	1.04	0.1	2.5	2.4	-0.10	-1.36	0.06
Species*	5	150	1.00	0.00	1.00	1.00	0.00	1.0	1.0	0.0	NaN	NaN	0.00

5.2 Visualisation des données

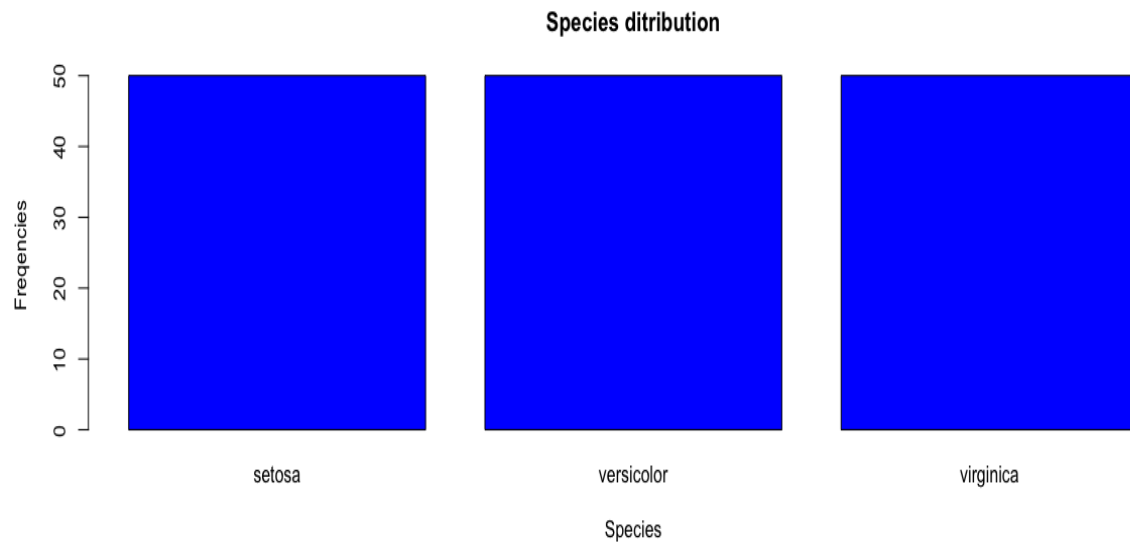
La fonction de visualisation la plus utilisée en programmation R est la fonction `plot()`. Il s'agit d'une fonction générique, ce qui signifie qu'elle possède de nombreuses méthodes qui sont appelées en fonction du type d'objet passé à `plot()`. Dans le cas le plus simple, nous pouvons passer un vecteur et nous obtiendrons un diagramme de dispersion de la magnitude par rapport à l'indice. Mais généralement, nous passons deux vecteurs et un nuage de points de ces points est tracé. Mais il existe aussi `ggplot2` un package dédié à la visualisation de données. Il peut améliorer considérablement la qualité et l'esthétique de vos graphiques, et vous rendra beaucoup plus efficace dans leur création. il permet de construire presque n'importe quel type de graphique.

5.2.1 `plot()`

On import le cadre de données de l'iris en utilisant le package `datasets`, on essaye de faire la visualisation de quatre cas par l'appelle du syntaxe `plot(x,y)` ou `plot(y x)`.

```
1 > library(datasets)
2 > plot(iris$Species,main="plot",xlab="Species",ylab="Frequencies",col="#ff0000")
```

la représentation d'une variable qualitative (Species) par `plot()` nous donne digramme en barre

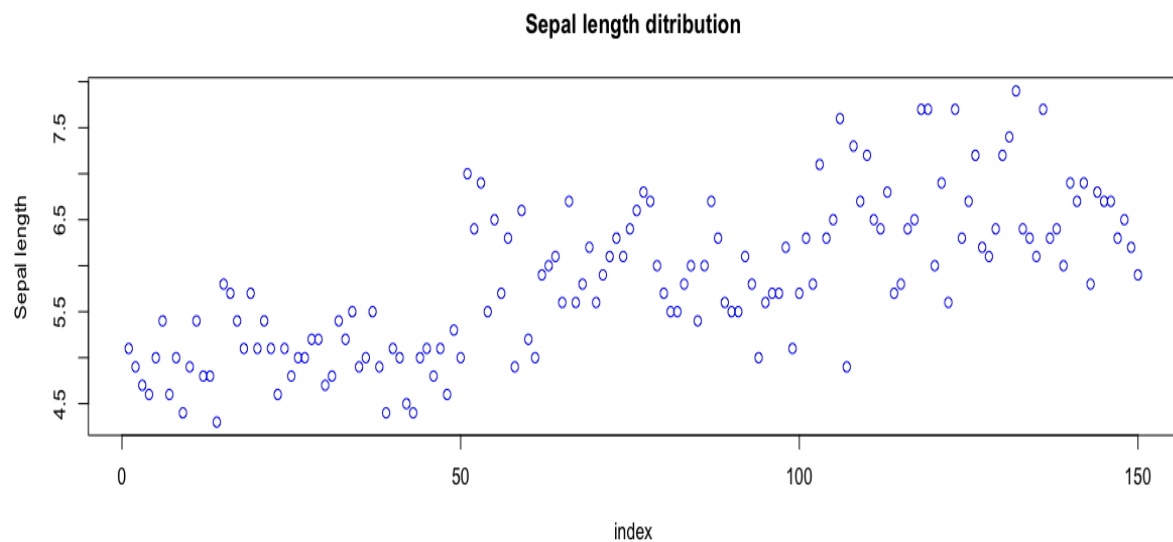


```

1 plot(iris\$Sepal.Length,main="Sepal length ditribution",xlab="index",
2 +   ylab="Sepal length",col="#0000ff")

```

la représentation d'une variable quantitative (Sepal length) par la fonction `plot()` nous nuage de points

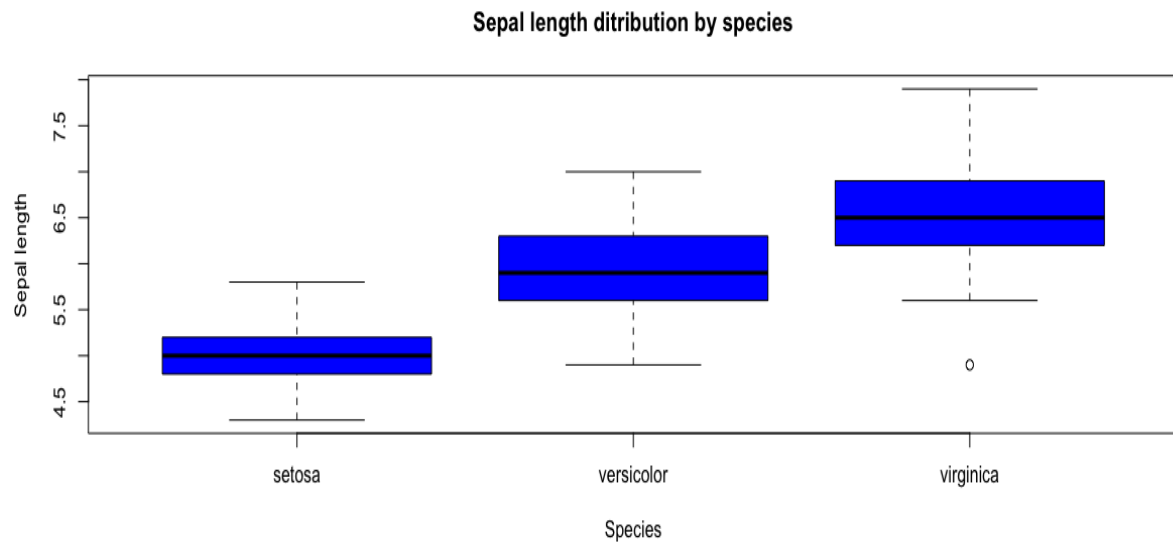


```

1 > plot(iris\$Species,iris\$Sepal.Length,main="Sepal length ditribution by species",
2 +   xlab="Species",ylab="Sepal length",col="#0000ff")

```

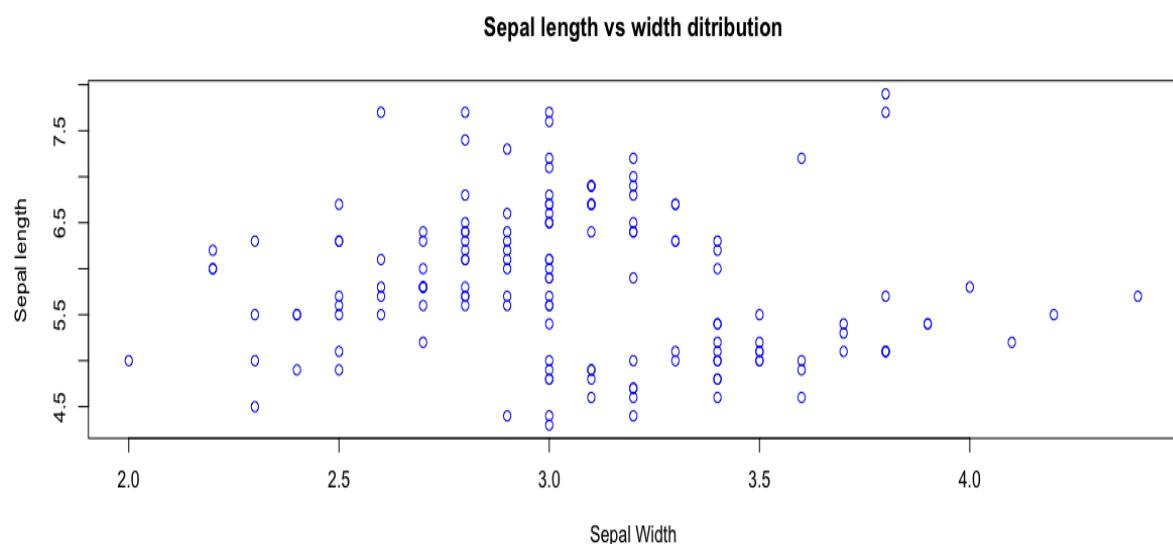
la représentation d'une variable quantitative (Sepal length) en fonction d'une variable qualitative (Species) par la fonction `plot()` nous donne des boîtes à moustaches pour chaque modalité



la représentation d'une variable quantitative (Sepal length) en fonction d'une variable qualitative (Species) par la fonction `plot()` nous donne nuage de points

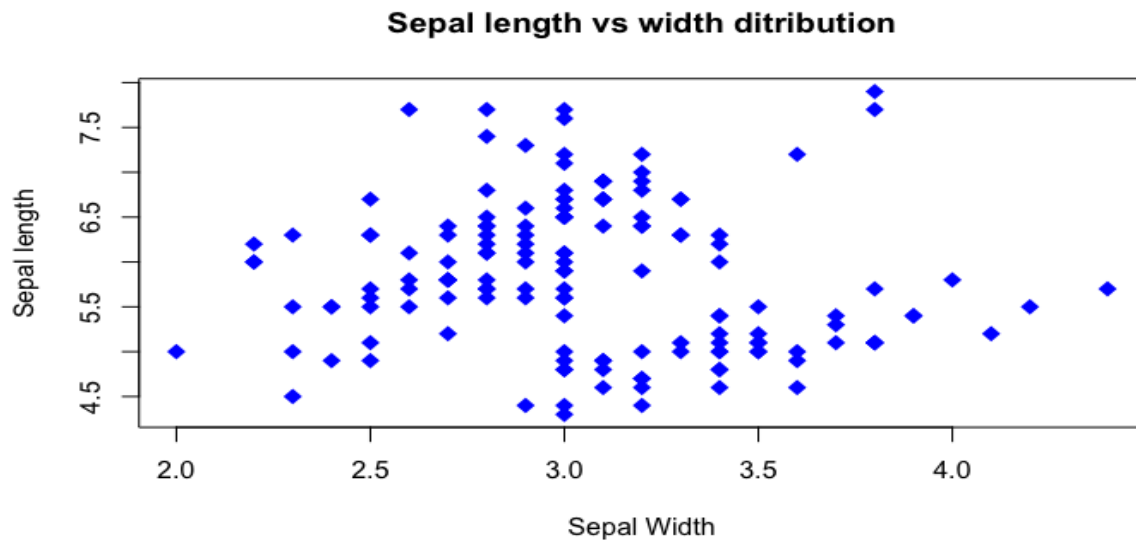
```
1 > plot(iris$Sepal.Width,iris$Sepal.Length,main="Sepal length vs width ditribution",
2 +      xlab="Sepal Width",ylab="Sepal length",col="#0000ff")
```

la représentation d'une variable quantitative (Sepal length) en fonction d'une variable qualitative (Species) par la fonction `plot()` nous donne des boîtes à moustaches pour chaque modalité



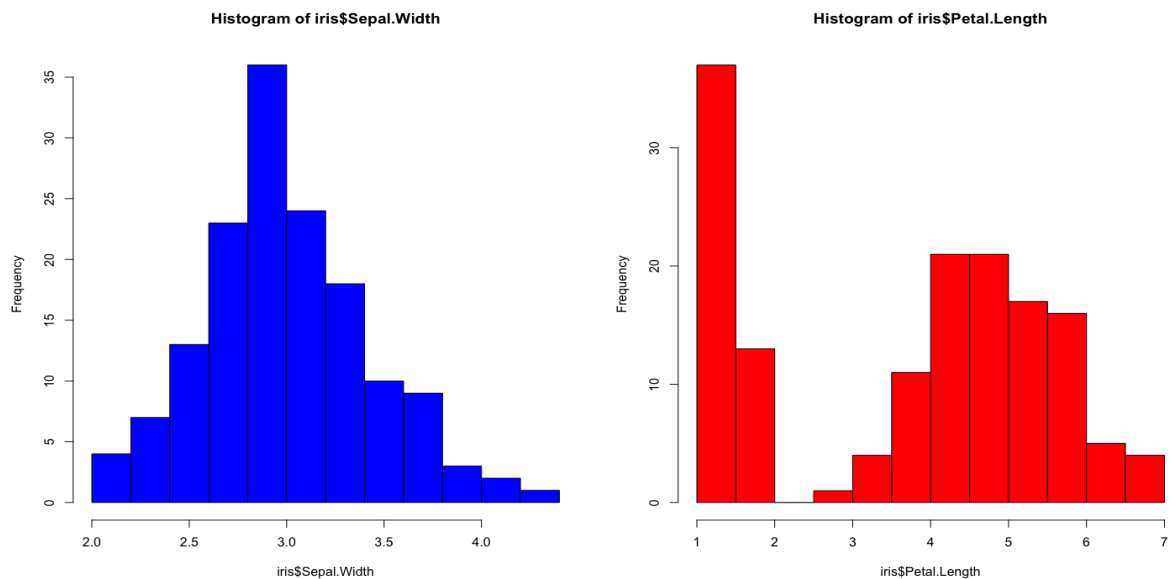
Il semble que la fonction `plot()` détecte automatiquement le type de variable, et les trace en fonction de leur type. On peut aussi personnaliser la visualisation en ajoutant quelques changements

```
1 > plot(iris$Sepal.Width,iris$Sepal.Length,  
2 +     cex=1.5,  
3 +     pch=18,  
4 +     main="Sepal length vs width ditribution",  
5 +     xlab="Sepal Width",  
6 +     ylab="Sepal length",  
7 +     col="#0000ff")
```



On peut présélectionner le type de graphique pour un certain type de données comme les données quantitative pour avoir un idée sur la distribution ou bien les aberrante...

```
1 > par(mfrow=c(1,2))  
2 > hist(iris$Sepal.Width,col="blue")  
3 > hist(iris$Petal.Length,col="red")
```

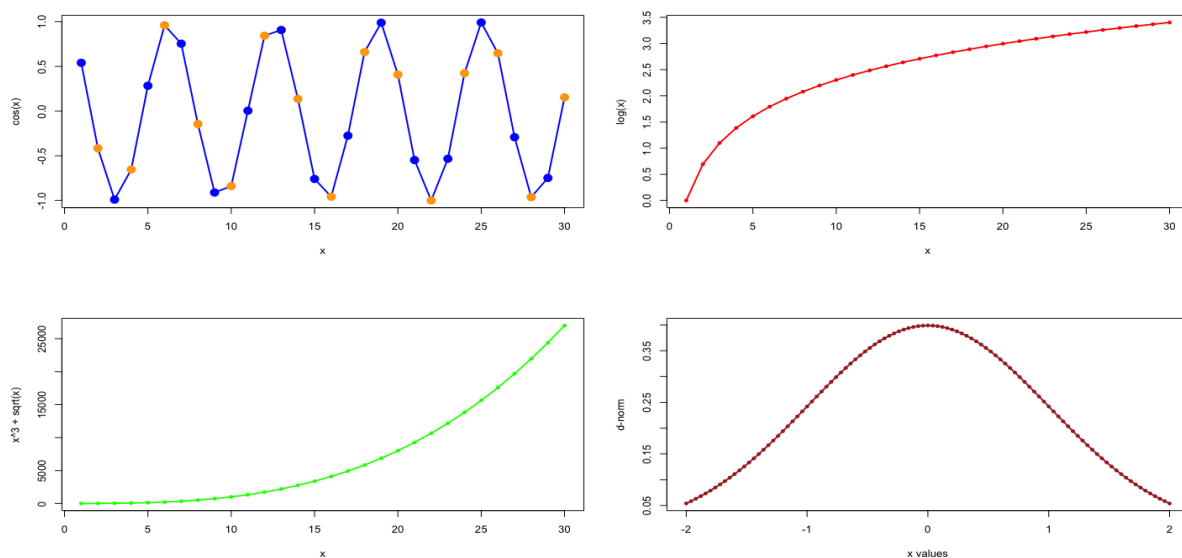


On a la possibilité de mettre plusieurs graphics dans une seule par par la commande `par(mfrow=c(l,c))`, le vecteur `c(l,c)` (nombre de lignes et nombre de colonnes) divise la fenêtre graphique en `l` lignes et `c` colonnes.

```

1 > par(mfrow=c(2,2))
2 > plot(x,cos(x),col=c("blue", "orange"),type="o",pch=19,lwd=2,cex=1.5)
3 > plot(x,log(x),col="red",type="o",pch=19,lwd=2,cex=0.5)
4 > plot(x,x^3+sqrt(x),col="green",type="o",pch=19,lwd=2,cex=0.5)
5 > plot(dnorm,-2,2,col="brown",type="o",pch=19,lwd=2,cex=0.5,xlab="x values", ylab="d-norm")

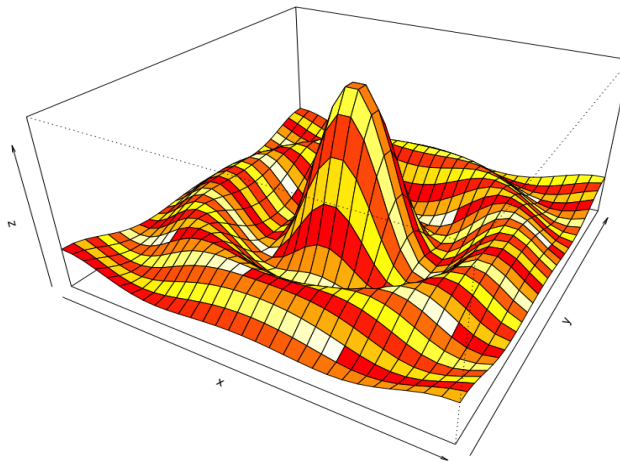
```



Les fonctions classiques de représentation 3D sont les fonctions persp, prenons un cette

exemple suivante $f(x, y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}}$

```
1 > f <- function(x,y){sin(sqrt(x^2+y^2))/sqrt(x^2+y^2)}  
2 > y <- x <- seq(-15,15,length=30)  
3 > persp(x,y,z,theta=30,phi=30,expand=0.5,col=heat.colors(50)) #theta et phi pour la position
```



Un tableau Table 5.1 de certains fonctions avec ses paramètres qui peuvent être utiliser dans la visualisation des données.

Fonction	Description
<code>barplot(vec)</code>	trace un diagramme en barres des valeurs de <code>vec</code>
<code>boxplot(vec)</code>	trace le graphe en boîte à moustaches de <code>vec</code>
<code>contour(vec₁,vec₂,vec₃)</code>	trace des courbes de niveau, voir aussi la fonction <code>filled.contour(vec₁,vec₂,vec₃)</code>
<code>coplot(vec₁,vec₂ val)</code>	trace le graphe bivarié de <code>vec₁</code> et <code>vec₂</code> pour chaque valeur de <code>val</code>
<code>filled.contour(vec₁,vec₂,vec₃)</code>	trace des courbes de niveau mais les aires entre les contours sont colorées, voir aussi <code>image(vec₁,vec₂,vec₃)</code>
<code>hist(vec,prob = TRUE)</code>	trace un histogramme de <code>vec</code>
<code>image(vec₁,vec₂,vec₃)</code>	trace des rectangles aux coordonnées <code>vec₁</code> , <code>vec₂</code> colorés selon <code>vec₃</code> , voir aussi <code>contour(vec₁,vec₂,vec₃)</code>
<code>pairs(vec)</code>	si <code>vec</code> est une matrice ou un data-frame, trace tous les graphes bivariés entre les colonnes de <code>x</code>
<code>persp(vec₁,vec₂,vec₃)</code>	trace une surface de réponse en 3D, voir <code>demo(persp)</code>
<code>pie(vec)</code>	trace un graphe en camembert
<code>plot(objet)</code>	trace le graphique correspondant à la classe de objet
<code>plot(vec₁,vec₂)</code>	trace le nuage de points de coordonnées <code>vec₁</code> et <code>vec₂</code>
<code>qqnorm(vec)</code>	trace les quantiles de <code>vec</code> en fonction de ceux attendus d'une loi normale
<code>qqplot(vec₁,vec₂)</code>	trace les quantiles de <code>vec₂</code> en fonction de ceux de <code>vec₁</code>
<code>spineplot(val₁,val₂)</code>	trace le diagramme en bandes correspondant à <code>val₁</code> et <code>val₂</code>
<code>stripplot(vec)</code>	trace le graphe des valeurs de <code>vec</code> sur une ligne
<code>sunflowerplot(vec₁,vec₂)</code>	idem mais les points superposés sont dessinés sous forme de fleurs dont le nombre de pétales correspond au nombre de points
<code>symbols(vec₁,vec₂,...)</code>	dessine aux coordonnées <code>vec₁</code> et <code>vec₂</code> des symboles (étoiles,cercles, boxplots, etc.)

TABLE 5.1 – Liste de fonctions graphiques

Un tableau Table 5.2 de certains paramètres qui peuvent être utiliser par les fonctions graphiques.

Fonction	Description
adj	contrôle la justification du texte par rapport au bord gauche du texte : 0 à gauche, 0.5 centré, 1 à droite ; $c(vec - 1, vec_2)$ justifie le texte horizontalement et verticalement
asp	précise le ratio entre $vec - 2$ et vec_1 : asp=1 pour un graphe orthonormé axes par défaut TRUE, les axes et le cadre sont tracés
bg	spécifie la couleur de l'arrière-plan : 1, 2, ou une couleur
bty	contrôle le tracé du cadre, valeurs permises : "o", "l", "7", "c", "u" ou "]"
bty="n"	supprime le cadre
cex	contrôle la taille des caractères et des symboles par rapport à la valeur par défaut qui vaut 1
cex.axis	contrôle la taille des caractères pour l'échelle des axes
cex.lab	contrôle la taille des caractères pour les libellés des axes
cex.main	contrôle la taille des caractères du titre
cex.sub	contrôle la taille des caractères du sous-titre
col	précise la couleur du graphe, valeurs possibles 1, 2, ou une couleur.
col.axis	précise la couleur des axes
col.main	précise la couleur du titre
font	contrôle le style du texte
font.axis	contrôle le style pour l'échelle des axes
font.lab	contrôle le style pour les libellés des axes
font.main	contrôle le style du titre
font.sub	contrôle le style du sous-titre
las	contrôle la disposition des annotations sur les axes (0 : parallèles aux axes, 1 : horizontales, : perpendiculaires aux axes, 3 : verticales)
lty	contrôle le type de ligne tracée, (1 : continue, 2 : tirets, 3 : points, 4 : points et tirets alternés, 5 : tirets longs, 6 : tirets courts et longs alternés)
lwd	contrôle l'épaisseur des traits
main	précise le titre du graphe, par exemple main="titre"
mfrow	vecteur $c(l, c)$ qui divise la fenêtre graphique en l lignes et c colonnes, les graphes sont ensuite dessinés en ligne
offset	précise le décalage du texte par rapport au point
pch	entier (entre 0 et 25) qui contrôle le type de symbole
pos	précise la position du texte, valeurs permises entier qui contrôle la taille en points du texte et des symboles
ps	précise le sous-titre du graphe, par exemple sub="sstitre"
sub	précise la longueur des graduations sur les axes
tck, tcl	précise le type de graphe dessiné, valeurs permises "n", "p", "l", "b", "h", "s", "S"
xlim, ylim	précise les limites des axes, par exemple xlim=c(0,10)
xlab, ylab	précise les annotations des axes

TABLE 5.2 – Liste des paramètres

5.2.2 ggplot2

La première étape pour produire un graphique avec `ggplot()` est d'installer et de charger le package

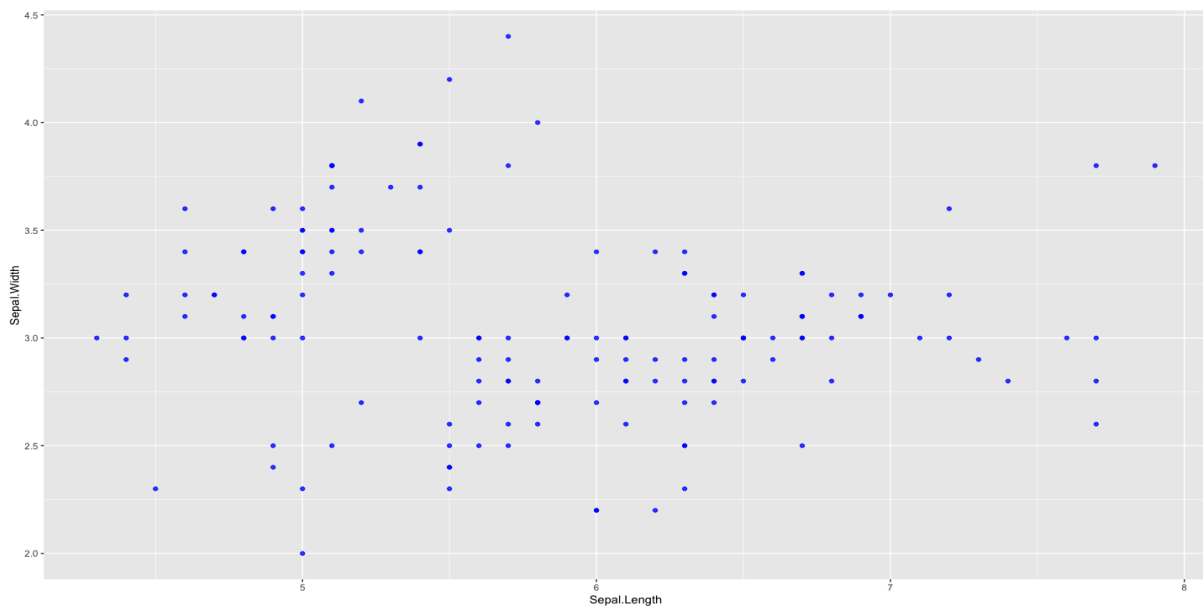
```
1 install.packages("ggplot2")
2 library(ggplot2)
```

Pour construire un `ggplot()`, nous utiliserons le modèle de base suivant qui peut être utilisé pour différents types de graphiques, par la syntaxe suivant

```
1 ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) + <GEOM_FUNCTION>()
```

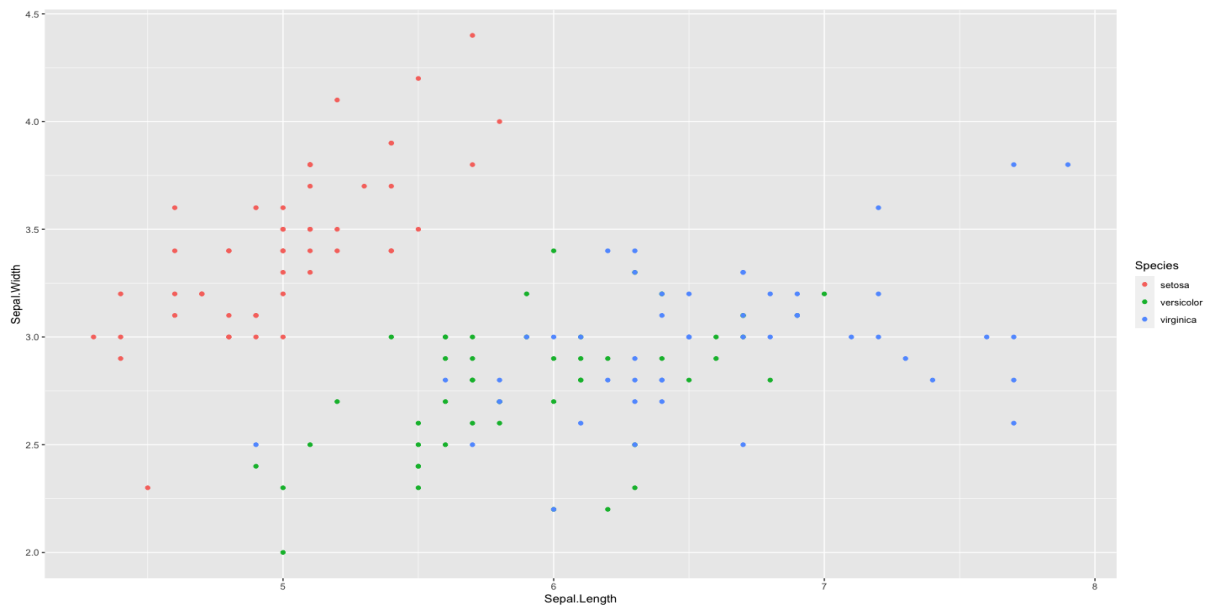
Bien entendu, nous utiliserons les mêmes données d' `iris` et aussi de `mtcars` qui se trouvent dans le paquet `datasets`. La création de graphiques avec `ggplot2` est généralement un processus itératif. Nous commençons par définir le cadre de données que nous allons utiliser, nous disposons les axes et nous choisissons un géomètre, `Sepal.Width` en fonction de `Sepal.Length`

```
1 > ggplot(data = iris, mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
2 +   geom_point(alpha = 2, color = "blue")
```



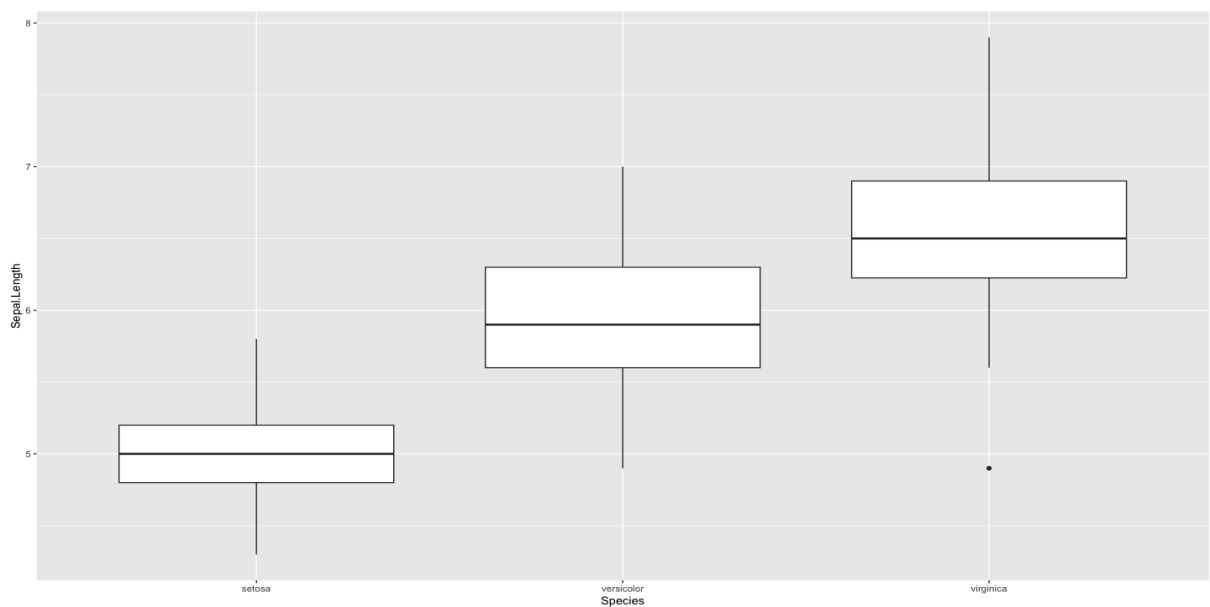
Même graphique mais en ajoutant les couleurs selon la variable `Species`

```
1 > ggplot(data = iris, mapping = aes(x = Sepal.Length, y = Sepal.Width)) +
2 +   geom_point(alpha = 2, aes(color = Species))
```

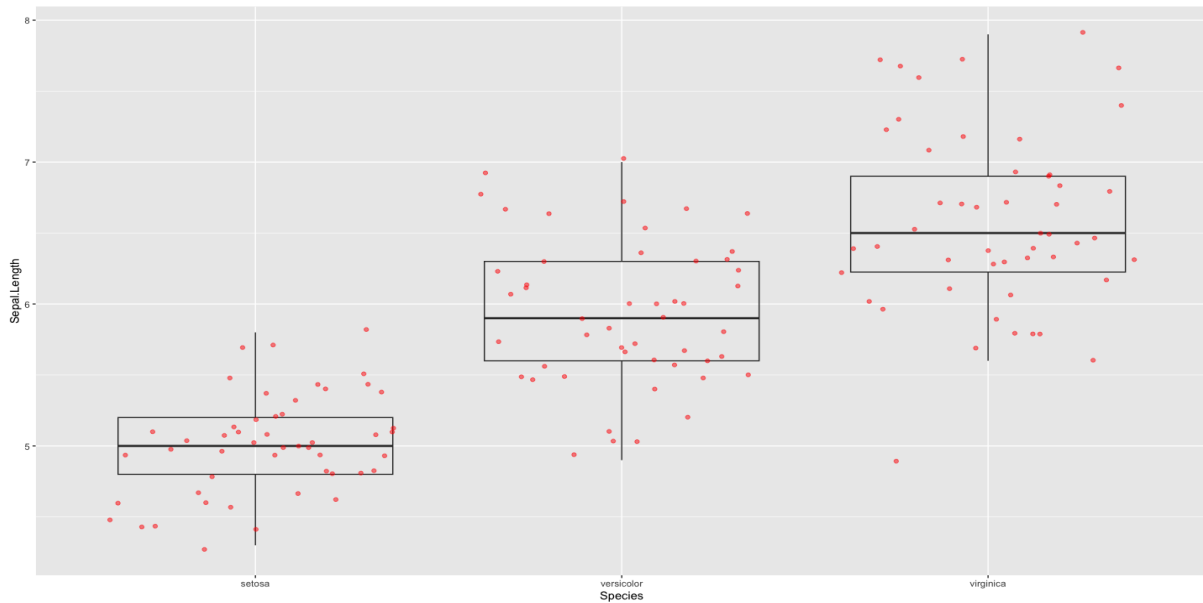


Nous pouvons utiliser des boxplots pour visualiser la distribution du Sepal.Length au sein de chaque type de Species,

```
1 > ggplot(data = iris, mapping = aes(x = Species, y = Sepal.Length)) +
2   +   geom_boxplot()
```

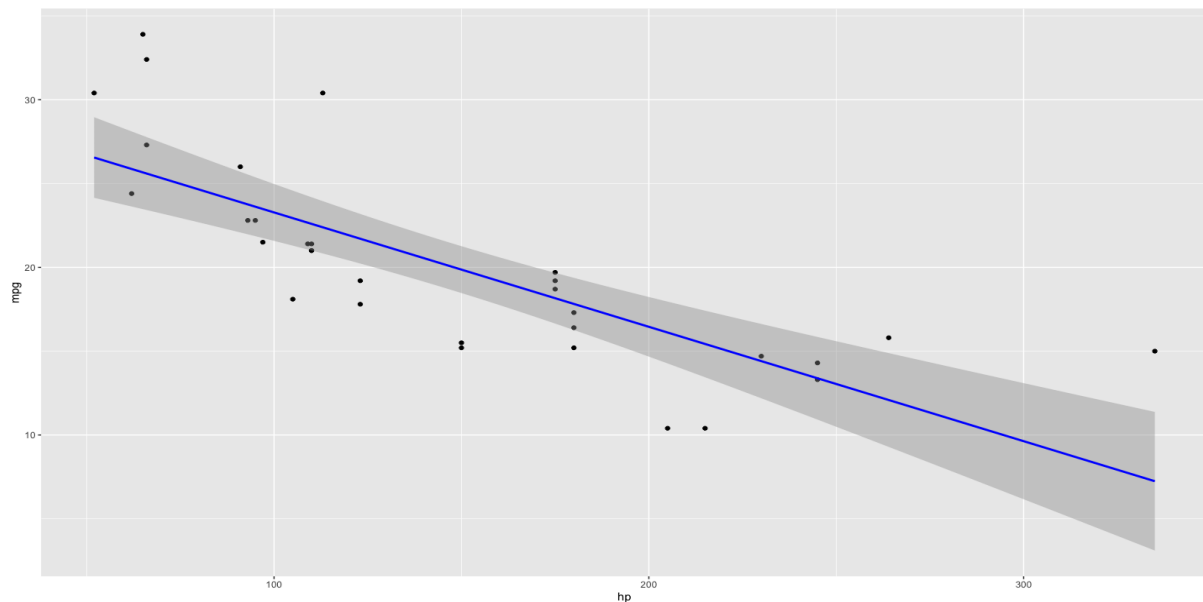


```
1 > ggplot(data = iris, mapping = aes(x = Species, y = Sepal.Length)) +
2   +   geom_boxplot(alpha = 0) +
3   +   geom_jitter(alpha = 0.5, color = "red")
```



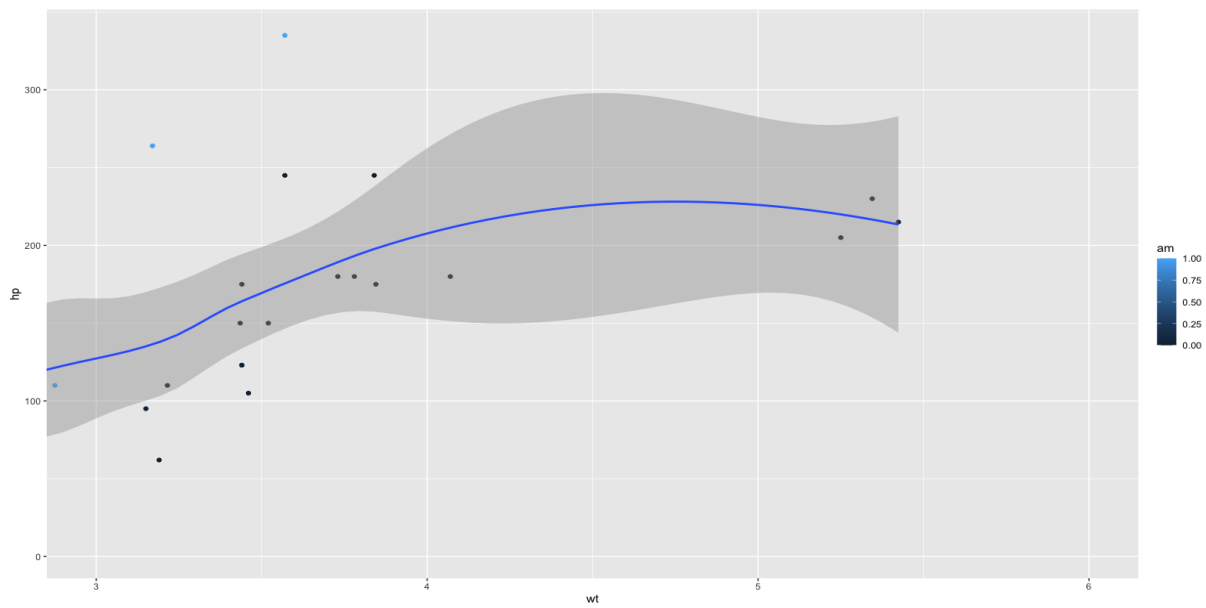
[H] Maintenantt avec le cadre de données `mtcars` (pour savoir [?mtcars](#)), dans cette couche, nous transformons nos données (`mpg` :Miles/(US) gallon en fonction `hp` : Gross horsepower) en utilisant le binning, le lissage, la description, l'intermédiaire.

```
1 > ggplot(data = mtcars, aes(x = hp, y = mpg)) + geom_point() + stat_smooth(method = lm, col = "blue")
```

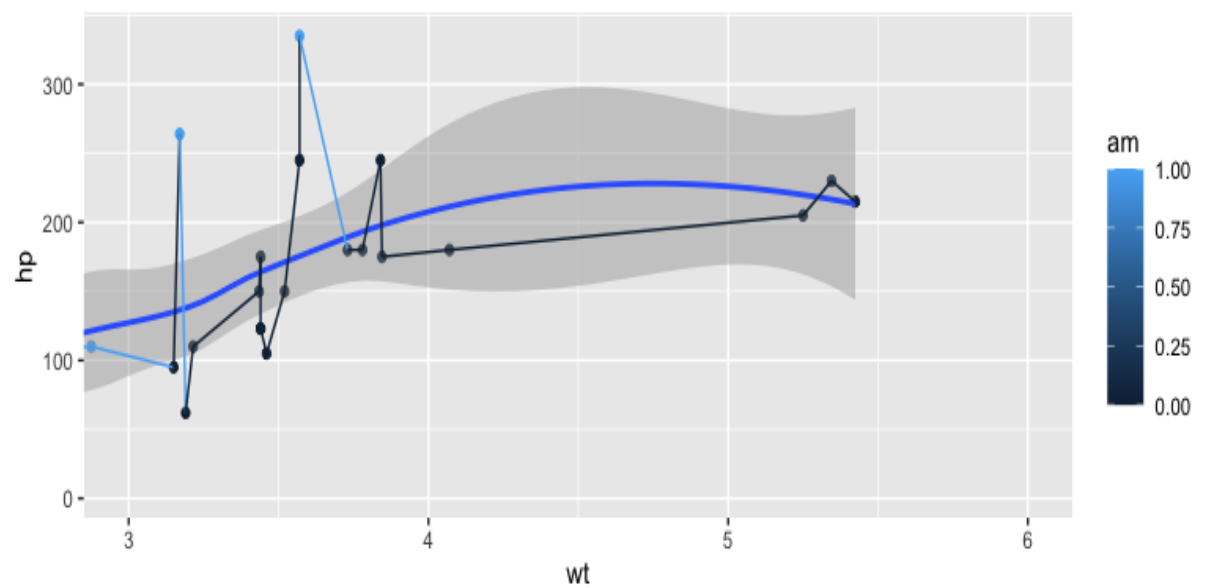


En ajoutant la fonction `coord_cartesian()` on contrôle les limites des axes x et y et vous permettent d'effectuer un zoom avant ou arrière sur votre tracé.

```
1 > ggplot(data = mtcars, aes(x = wt, y = hp, col = am)) + geom_point() + geom_smooth() +
2 +   coord_cartesian(xlim = c(3, 6))
```



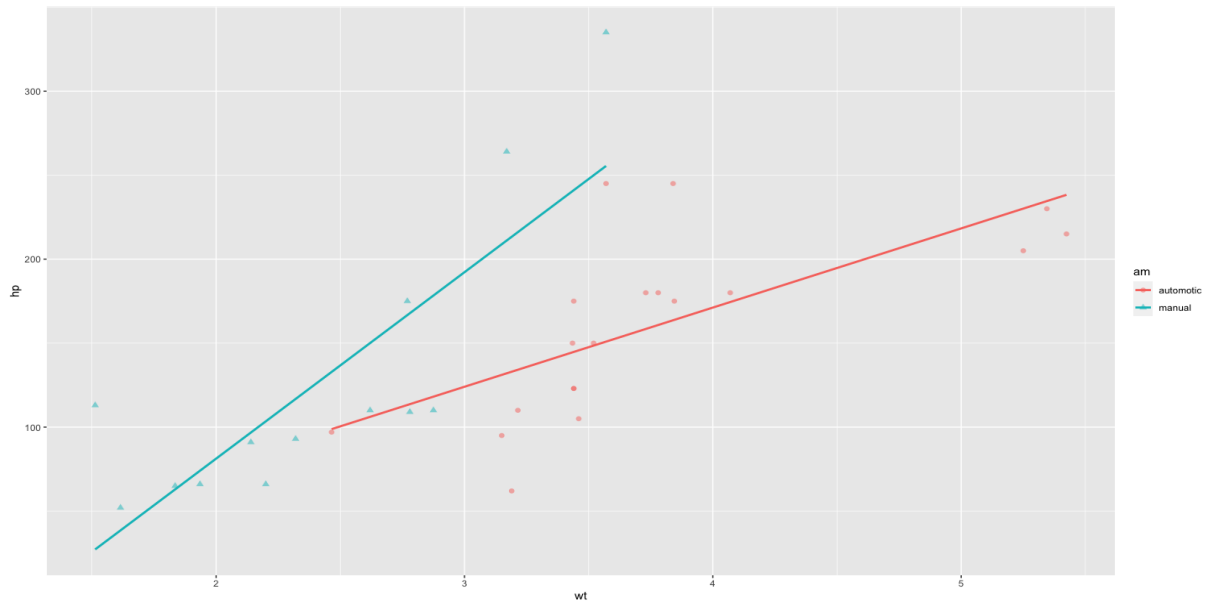
relient les points par des ligne en utilisant la fonction `geom_point()`



```

1 > ggplot(aes(x = wt, y = hp, colour = am), data = df) +
2   +   geom_point(aes(shape = am),size = 2, alpha=0.5) +
3   +   geom_smooth(method = "lm", se = FALSE)

```



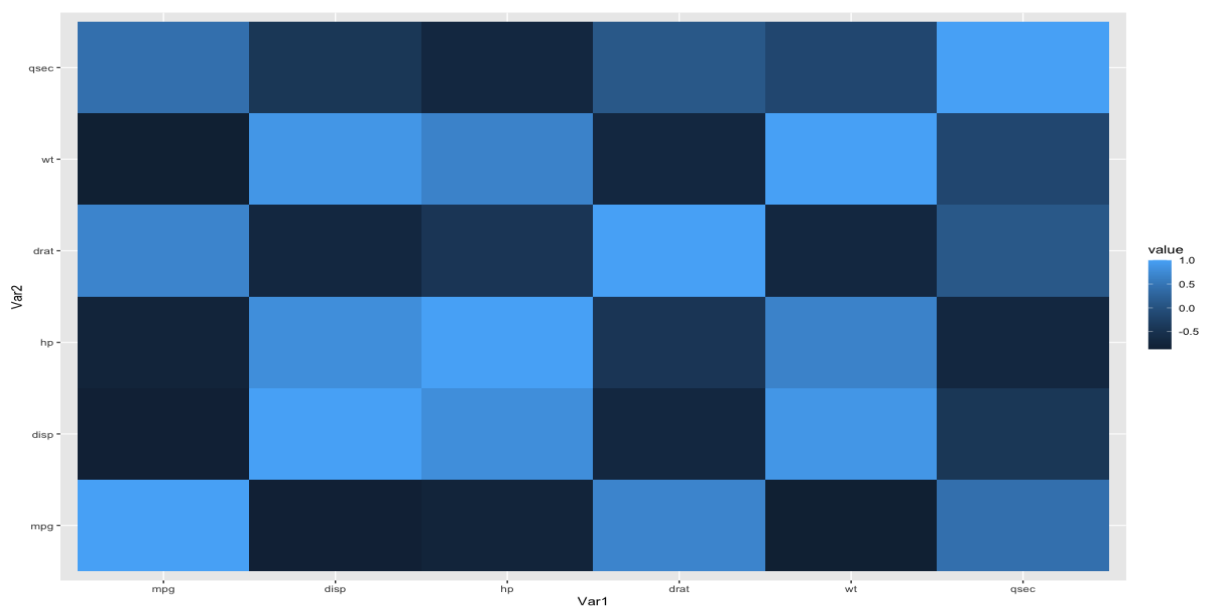
En ajoutant la fonction avec `facet_grid(vs ~ cyl)`, elle vas nous donnée une matrice de panneaux définie par des variables de facettes en ligne (Number of cylinders) et en colonne (Engine (0 = V-shaped, 1 = straight))

Nous visualisons une matrice de corrélation sur l'ensemble de données mtcars à l'aide du package `ggplot` et `reshape2` en utilisant la fonction `melt`

```

1 > df1 <- mtcars[, c(1,3,4,5,6,7)]
2 > cormat <- round(cor(df1),2)
3 > library(reshape2)
4 > melted_cormat <- melt(cormat)
5 > ggplot(data = melted_cormat, aes(x=Var1, y=Var2, fill=value)) + geom_tile()

```



Les tables 5.3, 5.4 présentent les fonctions peuvent être utilisées pour créer une variété de représentations graphiques.

Fonction Geom	Description
geom_point()	Nuage de points
geom_line()	Ligne (ordonnées selon <i>vec</i>)
geom_abline()	Droite
geom_path()	Ligne (ordre original)
geom_text()	Texte
geom_rect()	Rectangle
geom_polygon()	Polygone
geom_segment()	Segment
geom_bar()	Diagramme en barres
geom_histogram()	Histogramme
geom_boxplot()	Boîtes à moustaches
geom_density()	Densité
geom_contour()	Lignes de contour
geom_smooth()	Lissage

TABLE 5.3 – Liste des fonctions geom

Fonction Stat	Description
stat_identity()	Aucune transformation
stat_bin()	Comptage
stat_density()	Densité
stat_smooth()	Lissage
stat_boxplot()	Boxplot

TABLE 5.4 – Liste des fonctions statistics

Chapitre 6

La statistique inférentielle

6.1 échantillonnage

nous allons explorer les statistiques inférentielles. Nous commencerons avec la distribution d'échantillonnage, et continuerons avec le théorème de la limite centrale, l'intervalle de confiance et le test d'hypothèse.

Nous allons utiliser un cadre de données appelé **Ames**. Il s'agit de données immobilières de la ville d'Ames, Iowa, USA. Les détails de chaque transaction immobilière à Ames sont enregistrés par le bureau de l'assesseur de la ville.

```
1 > library(dplyr) #Package pour la manipulation des données
2 > df <- read.csv("~/Desktop/data.csv")
3 > glimpse(df)
4 ## Rows: 2,930
5 ## Columns: 82
6 ## $ Order      <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ~
7 ## $ PID        <int> 526301100, 526350040, 526351010, 526353~
8 ## ...
9 ## ...
```

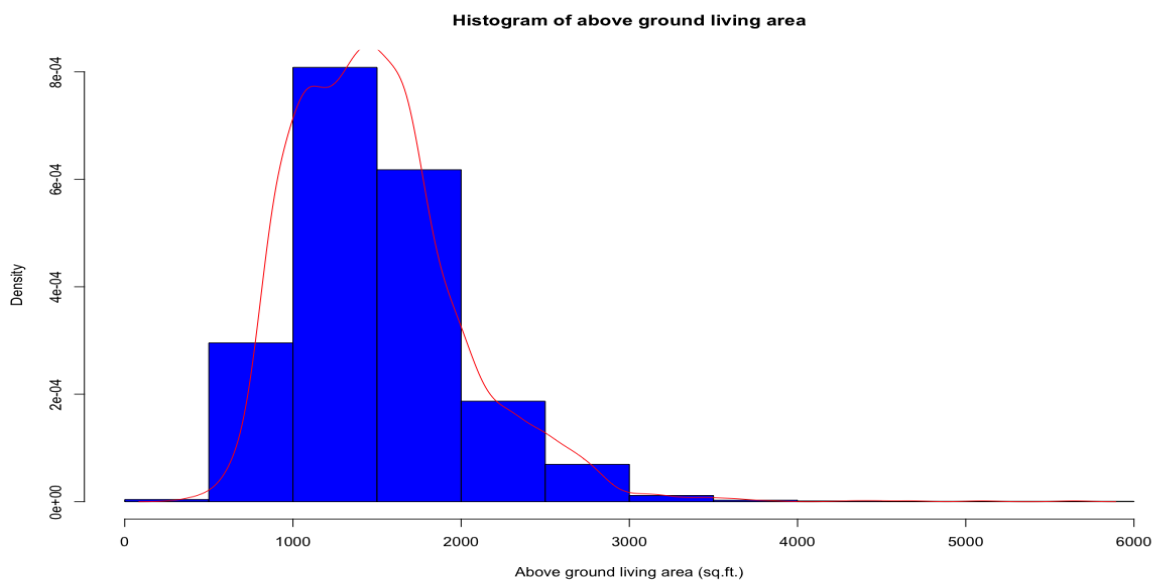
Nous limiterons notre attention à seulement deux des variables pour faire une analyse très approfondie : la superficie de la maison en pieds carrés (Gr.Liv.Area) et le prix de vente (SalePrice) en dollars.

```
1 > area <- df$Gr.Liv.Area
2 > price <- df$SalePrice
3 > area.sd <- sqrt(sum((area - mean(area))^2)/(2930)) #Écart type de la population
4 [1] 505.4226
5 > summary(area)
6   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
7   334    1126    1442    1500    1743    5642
```

La distribution de la variable area a une moyenne de 1499,69 pieds carrés, une médiane de 1442 pieds carrés et un écart type de population de 505,42 pieds carrés, comme on

peut le voir. L'ensemble de données contient des observations dont la taille minimale est de 334 pieds carrés et la taille maximale de 5642 pieds carrés.

```
1 > hist(area,  
2 +     main = "Histogram of above ground living area",  
3 +     xlab = "Above ground living area (sq.ft.)",  
4 +     col="blue",  
5 +     prob=TRUE)  
6 > lines(density(area),col="red")
```



L'histogramme de la variable (area) montre une asymétrie positive (droite) des données de la population.

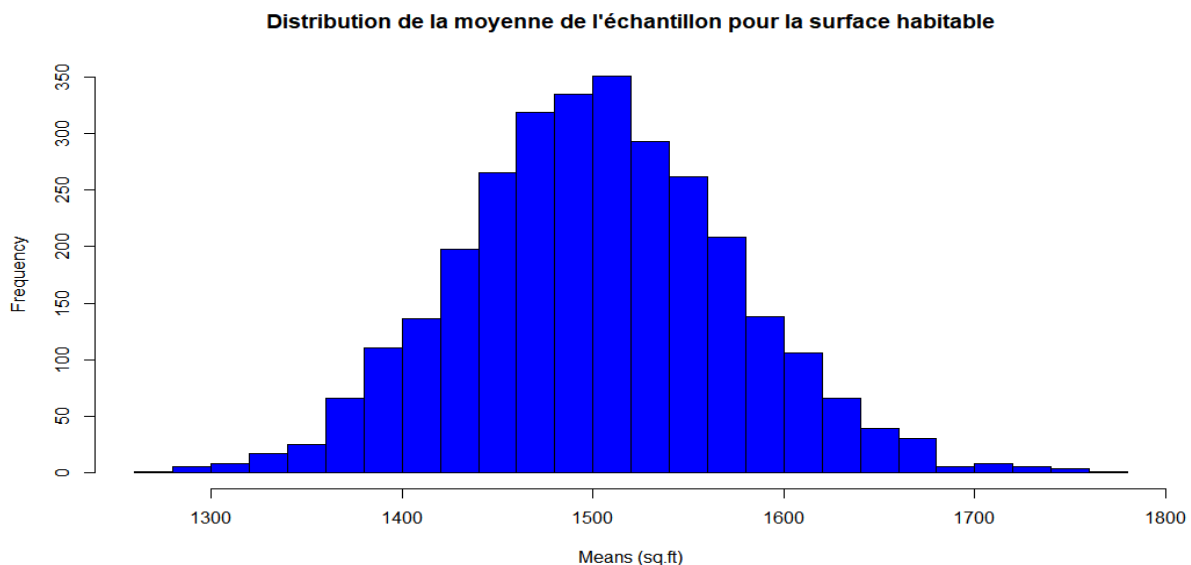
Il est souvent très coûteux ou difficile de collecter des données sur une population entière. Par conséquent, nous utilisons fréquemment un échantillon de la population pour comprendre les caractéristiques de cette dernière.

```
1 > sam <- sample(area, 50) #échantillon aléatoire de 50 observations d'area  
2 > mean(sam)  
3 [1] 1546.96
```

Nous obtenons une moyenne d'échantillon différente chaque fois que nous prenons un nouveau échantillon aléatoire. Il est utile de connaître le degré de variabilité à prévoir lorsque l'on mesure la moyenne de la population de cette manière. Nous pouvons comprendre cette variabilité en examinant la distribution d'échantillonnage, qui est la distribution des moyennes d'échantillon. Comme nous avons accès à la population dans ce laboratoire, nous pouvons effectuer les processus susmentionnés de manière répétée afin de construire la distribution d'échantillonnage de la moyenne de l'échantillon. La population sera divisée en 3000 échantillons, chacun de taille 50, et nous prendrons la moyenne de chaque

échantillon et stockerons les résultats dans un vecteur appelé `sam50`. L'histogramme de cette distribution d'échantillonnage sera ensuite tracé.

```
1 > sam50 <- rep(NA, 3000)
2 > for(i in 1:3000){samp <- sample(area, 50)
3 + sam50[i] <- mean(samp)}
4 > hist(sam50, breaks = 25,
5 +     main = "Distribution de la moyenne de l'échantillon pour la surface habitable",
6 +     xlab = "Means (sq.ft)",
7 +     col="blue")
```



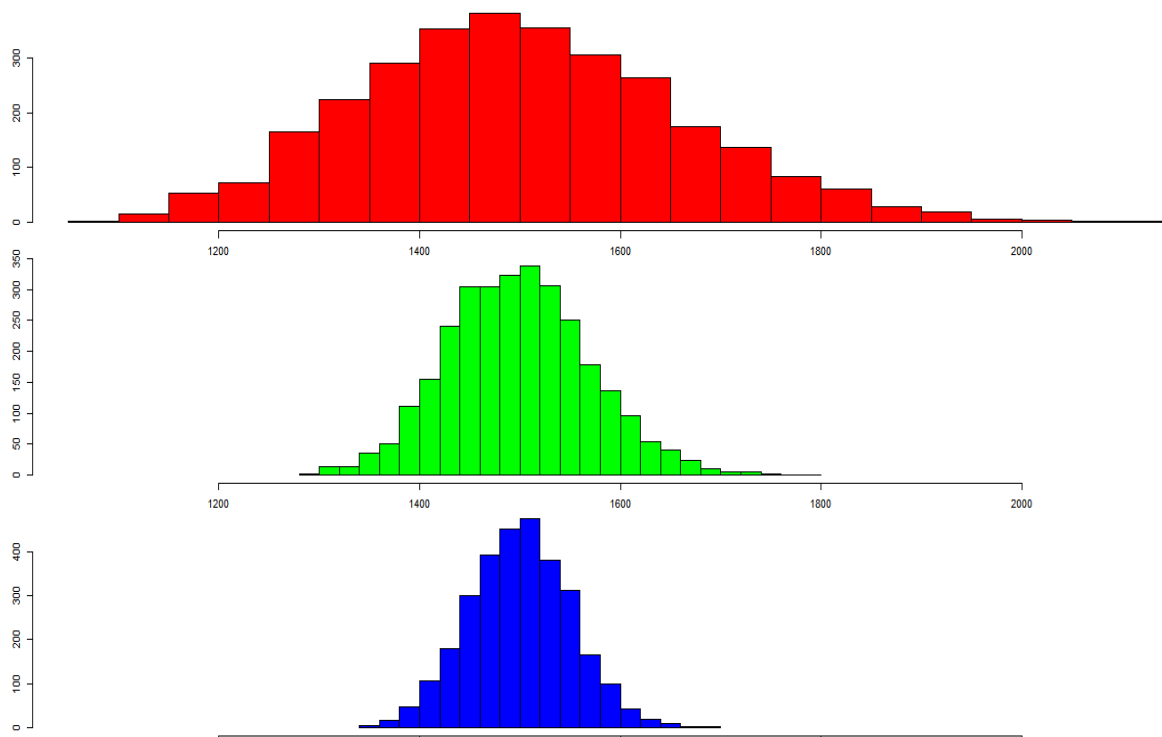
La distribution d'échantillonnage que nous avons calculée nous permet de déduire beaucoup de choses sur l'estimation de la surface habitable typique des logements d'Ames. La distribution d'échantillonnage est centrée sur la surface habitable moyenne réelle de la population, car la moyenne de l'échantillon est un estimateur sans biais, et l'étendue de la distribution montre la variabilité introduite par l'échantillonnage de seulement 50 ventes de logements. Construisons deux autres distributions d'échantillonnage, l'une basée sur un échantillon de 10 personnes et l'autre sur un échantillon de 100 personnes provenant d'une population de 3000 personnes, afin de mieux comprendre l'impact de la taille de l'échantillon sur notre distribution. Traçons les trois distributions l'une au-dessus (en utilisant la commande `par(mfrow = c(3, 1))`), en fixant le nombre de bars `breaks` en 21 bar, et aussi en utilisant même l'étendue de l'axe des x par `xlim`) de l'autre pour illustrer comment le changement de taille d'échantillon affecte la distribution d'échantillonnage.

```
1 > area <- df$Gr.Liv.Area
2 > samp10 <- rep(NA, 3000); samp50 <- rep(NA, 3000); samp100 <- rep(NA, 3000)
3 > for(i in 1:3000){
4 +   samp <- sample(area, 10)
5 +   samp10[i] <- mean(samp)
```

```

6 + samp <- sample(area, 50)
7 + samp50[i] <- mean(samp)
8 + samp <- sample(area, 100)
9 + samp100[i] <- mean(samp)}
10 > par(mfrow = c(3, 1))
11 > xlimits <- range(samp10)
12 > hist(samp10, breaks = 21, xlim = xlimits, col="red")
13 > hist(samp50, breaks = 21, xlim = xlimits, col="green")
14 > hist(samp100, breaks = 21, xlim = xlimits, col="blue")

```



Le centre n'est pas affecté par la taille de l'échantillon. La moyenne des moyennes de l'échantillon est toujours approximativement la même que la moyenne de la population. L'écart est plus faible pour les échantillons plus grands, donc l'écart type des moyennes de l'échantillon diminue lorsque la taille de l'échantillon augmente.

6.2 Test d'hypothèse

Nous utiliserons un vaste ensemble de données contenant des informations sur les naissances enregistrées dans l'État de Caroline du Nord (2004), aux États-Unis.

```

1 df <- read.csv("~/Desktop/data.csv")
2 > glimpse(df)
3 Rows: 1,000
4 Columns: 13
5 \ $ fage      <int> NA, NA, 19, 21, NA, NA, 18, 17, NA, 20, 30, NA, NA, ~

```

```

6 \ $ mage <int> 13, 14, 15, 15, 15, 15, 15, 15, 16, 16, 16, 16, 16,~
7 \ $ mature <chr> "younger mom", "younger mom", "younger mom", "young~
8 \ $ weeks <int> 39, 42, 37, 41, 39, 38, 37, 35, 38, 37, 45, 42, 40,~
9 \ $ premie <chr> "full term", "full term", "full term", "full term",~
10 \ $ visits <int> 10, 15, 11, 6, 9, 19, 12, 5, 9, 13, 9, 8, 4, 12, 15~
11 \ $ marital <chr> "married", "married", "married", "married", "marrie~
12 \ $ gained <int> 38, 20, 38, 34, 27, 22, 76, 15, NA, 52, 28, 34, 12,~
13 \ $ weight <dbl> 7.63, 7.88, 6.63, 8.00, 6.38, 5.38, 8.44, 4.69, 8.8~
14 \ $ lowbirthweight <chr> "not low", "not low", "not low", "not low", "not lo~
15 \ $ gender <chr> "male", "male", "female", "male", "female", "male",~
16 \ $ habit <chr> "nonsmoker", "nonsmoker", "nonsmoker", "nonsmoker",~
17 \ $ whitemom <chr> "not white", "not white", "white", "white", "not wh~

```

En utilisant la fonction `summary()` Nous essayons de déterminer quelles variables sont catégoriques et quelles variables sont d'échelle, s'il y a des valeurs aberrantes pour les variables d'échelle, ainsi que les enregistrements NA pour chaque variable.

```

1 > summary(df)
2      fage      mage      mature      weeks
3  Min.   :14.00  Min.   :13    Length:1000    Min.   :20.00
4  1st Qu.:25.00  1st Qu.:22    Class :character  1st Qu.:37.00
5  Median :30.00  Median :27    Mode  :character  Median :39.00
6  Mean   :30.26  Mean   :27                      Mean   :38.33
7  3rd Qu.:35.00  3rd Qu.:32                      3rd Qu.:40.00
8  Max.   :55.00  Max.   :50                      Max.   :45.00
9  NA's   :171                      NA's   :2
10     premie      visits      marital      gained
11 Length:1000    Min.   : 0.0    Length:1000    Min.   : 0.00
12 Class :character  1st Qu.:10.0    Class :character  1st Qu.:20.00
13 Mode  :character  Median :12.0    Mode  :character  Median :30.00
14                      Mean   :12.1                      Mean   :30.33
15                      3rd Qu.:15.0                      3rd Qu.:38.00
16                      Max.   :30.0                      Max.   :85.00
17                      NA's   :9                        NA's   :27
18     weight  lowbirthweight      gender      habit
19  Min.   : 1.000 Length:1000    Length:1000    Length:1000
20  1st Qu.: 6.380 Class :character  Class :character  Class :character
21  Median : 7.310 Mode  :character  Mode  :character  Mode  :character
22  Mean   : 7.101
23  3rd Qu.: 8.060
24  Max.   :11.750
25
26     whitemom
27 Length:1000
28 Class :character
29 Mode  :character

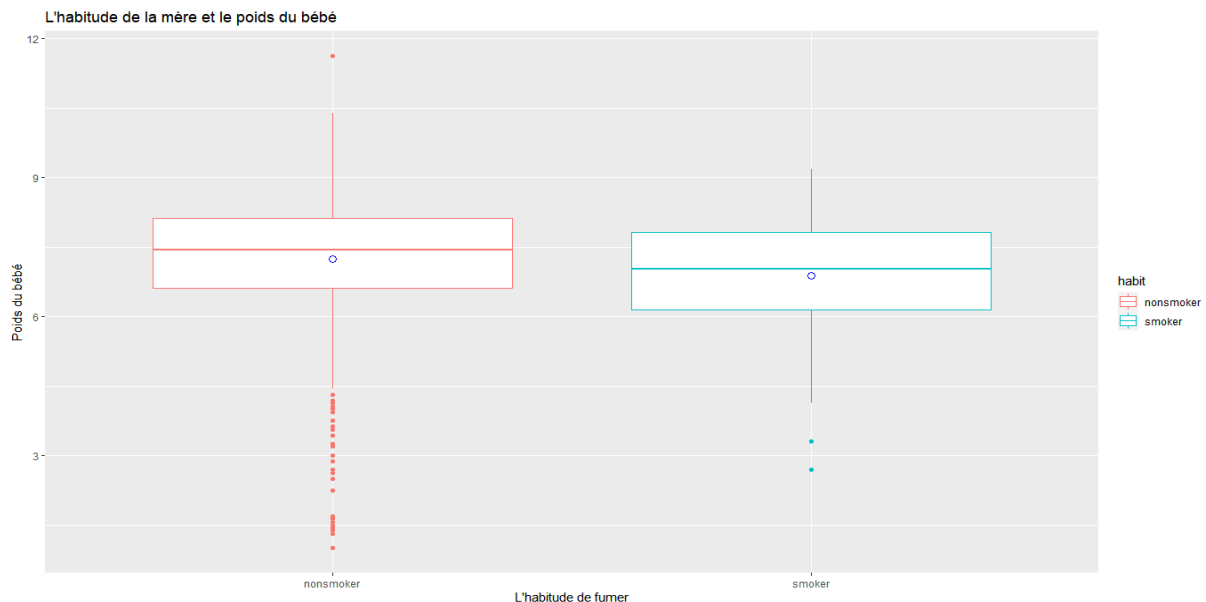
```

Par exemple, nous considérons la relation possible entre l'habitude de fumer d'une mère et le poids de son bébé pour les fumeuses et les non-fumeuses. En général, les observations

avec des valeurs manquantes doivent être examinées, éventuellement éliminées, ou traitées (nous allons juste l'ignorer). Les données doivent être tracées dans un premier temps car cela nous permet de voir facilement les tendances, la distribution, et développer des questions de recherche.

Faisons donc un boxplot côte à côte de la variable habit et de la variable weight en utilisant la fonction ggplot.

```
1 > ggplot(data = na.omit(df), aes(x = habit,  
2 + y = weight, colour = habit)) + geom_boxplot() +  
3 + xlab("L'habitude de fumer") + ylab("Poids du bébé") +  
4 + ggtitle("L'habitude de la mère et le poids du bébé") +  
5 + stat_summary(fun = mean, colour = "blue", geom = "point", #for means comparing  
6 + shape = 1, size = 3)
```



Les diagrammes en boîte montrent comment les médianes des deux distributions se comparent, mais en ajoutant les moyennes sur le diagramme en boîte à l'aide de la fonction `stat_summary()`, nous pouvons également comparer les moyennes des distributions. Bien qu'il y ait quelques différences de moyennes, Mais la question qui se pose ; cette variation est-elle statistiquement significative ? pour la répondre, Nous allons effectuer un test d'hypothèse.

Avant de procéder à un test d'hypothèse, il est important de vérifier si les conditions préalables à l'inférence sont remplies. Pour valider les conditions et vérifier les valeurs aberrantes, vous devez obtenir la taille des échantillons. La taille du groupe peut être calculée à l'aide de la formule `by(ncweight, nchabit, length)`.

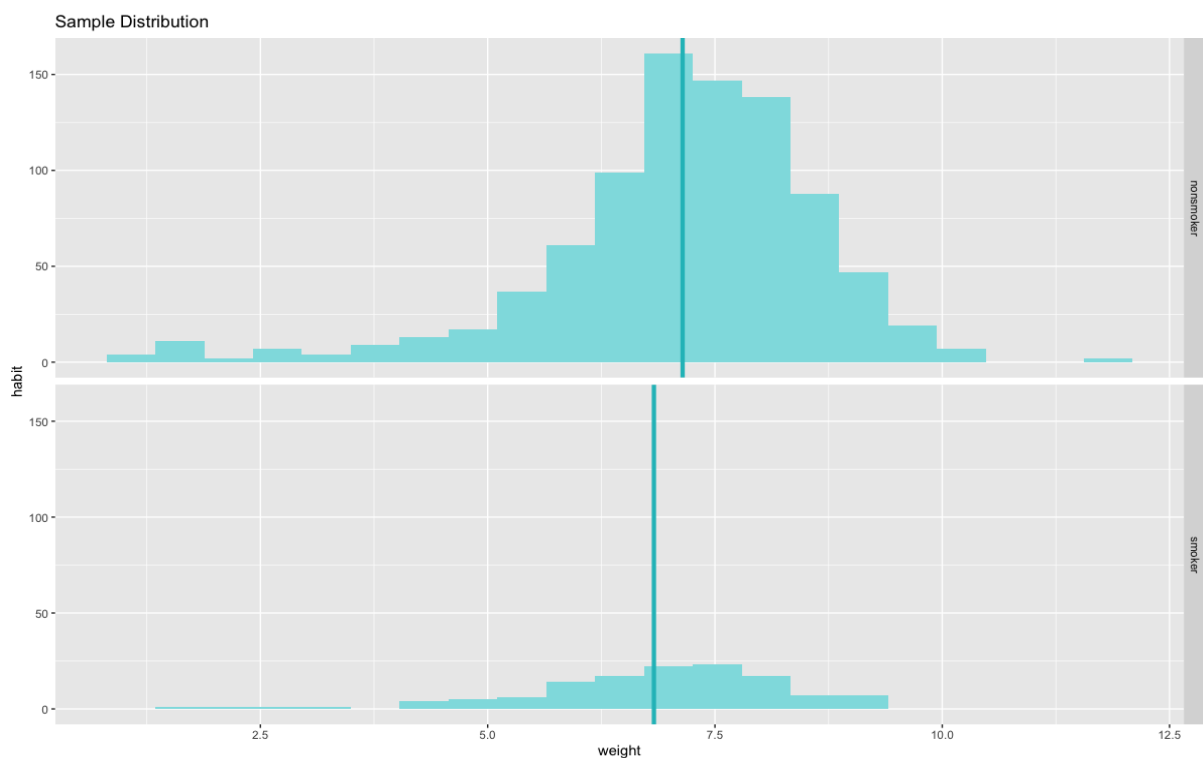
L'étape suivante consiste à développer les hypothèses pour tester si les poids moyens à la naissance des enfants nés de fumeurs et de non-fumeurs diffèrent. Nous utiliserons une fonction appelée `inference()` pour effectuer les tests. Nous pouvons effectuer des tests

d'hypothèses et créer des intervalles de confiance en utilisant cette méthode. Afin d'utiliser la fonction `inference()`, il faut installer le package `statsr` par la fameuse commande `install.packages()`

```

1 > install.packages("statsr")
2 > inference(y = weight, x = habit, data = df, # le poids est DV et l'habitude est IDV
3 +         statistic = c("mean"), #paramètre à estimer est la moyenne
4 +         type = c("ci"), # Intervalle de confiance
5 +         null = 0, #l'hypothèse nulle: les deux moyennes sont égales.
6 +         alternative = c("twosided"), #direction de l'hypothèse alternative
7 +         method = c("theoretical"), #Théorique parce que nous avons les données
8 +         conf_level = 0.95,
9 +         order = c("smoker", "nonsmoker"))
10 Response variable: numerical, Explanatory variable: categorical (2 levels)
11 n_smoker = 126, y_bar_smoker = 6.8287, s_smoker = 1.3862
12 n_nonsmoker = 873, y_bar_nonsmoker = 7.1443, s_nonsmoker = 1.5187
13 95% CI (smoker - nonsmoker): (-0.5803 , -0.0508)

```



Maintenant, omettons les valeurs manquantes de nos calculs, et executons les même lignes précédentes

```

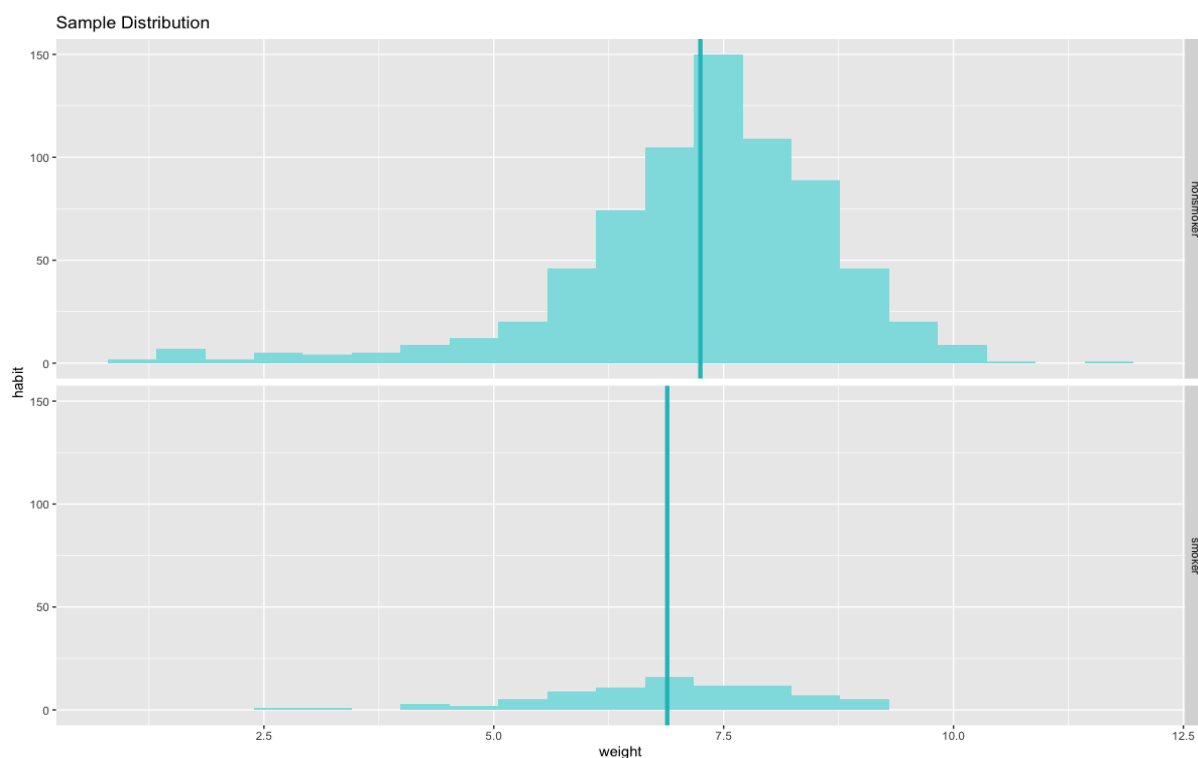
1 > dfomit = na.omit(df)
2 > inference(y = weight, x = habit, data = df, # le poids est DV et l'habitude est IDV
3 +         statistic = c("mean"), #paramètre à estimer est la moyenne
4 +         type = c("ci"), # Intervalle de confiance

```

```

5 +         null = 0, #l'hypothèse nulle: les deux moyennes sont egaux.
6 +         alternative = c("twosided"), #direction de l'hypothèse alternative
7 +         method = c("theoretical"), #Théorique parce que nous avons les données
8 +         conf_level = 0.95,
9 +         order = c("smoker", "nonsmoker"))
10 Response variable: numerical, Explanatory variable: categorical (2 levels)
11 n_smoker = 84, y_bar_smoker = 6.8864, s_smoker = 1.3064
12 n_nonsmoker = 716, y_bar_nonsmoker = 7.2468, s_nonsmoker = 1.4521
13 95% CI (smoker - nonsmoker): (-0.6637 , -0.057)

```



Dans les deux cas (avec ou sans valeurs manquantes), nous pouvons voir sur ces deux graphiques qu'il y a quelques variations dans la façon de mesurer la tendance centrale et l'écart. L'intervalle de confiance pour la différence entre les moyennes des fumeurs et des non-fumeurs sont également fournis dans le résultat. Nous pouvons affirmer que la différence est statistiquement significative et différente de 0 car l'intervalle de confiance exclut la valeur nulle (H_0 : la différence entre moyens est 0). Pour le déterminer, nous pouvons exécuter un test d'hypothèse dans cet exemple, on utilise un [t-test](#), pour cela, il suffit de spécifier le type "ht" dans la paramètre [type](#) comme suivante :

```

1 > dfomit = na.omit(df)
2 > inference(y = weight, x = habit, data = df, # le poids est DV et l'habitude est IDV
3 +         statistic = c("mean"), #paramètre à estimer est la moyenne
4 +         type = c("ci"), # Intervalle de confiance
5 +         null = 0, #l'hypothèse nulle: les deux moyennes sont égales.
6 +         alternative = c("twosided"), #direction de l'hypothèse alternative
7 +         method = c("theoretical"), #Théorique parce que nous avons les données

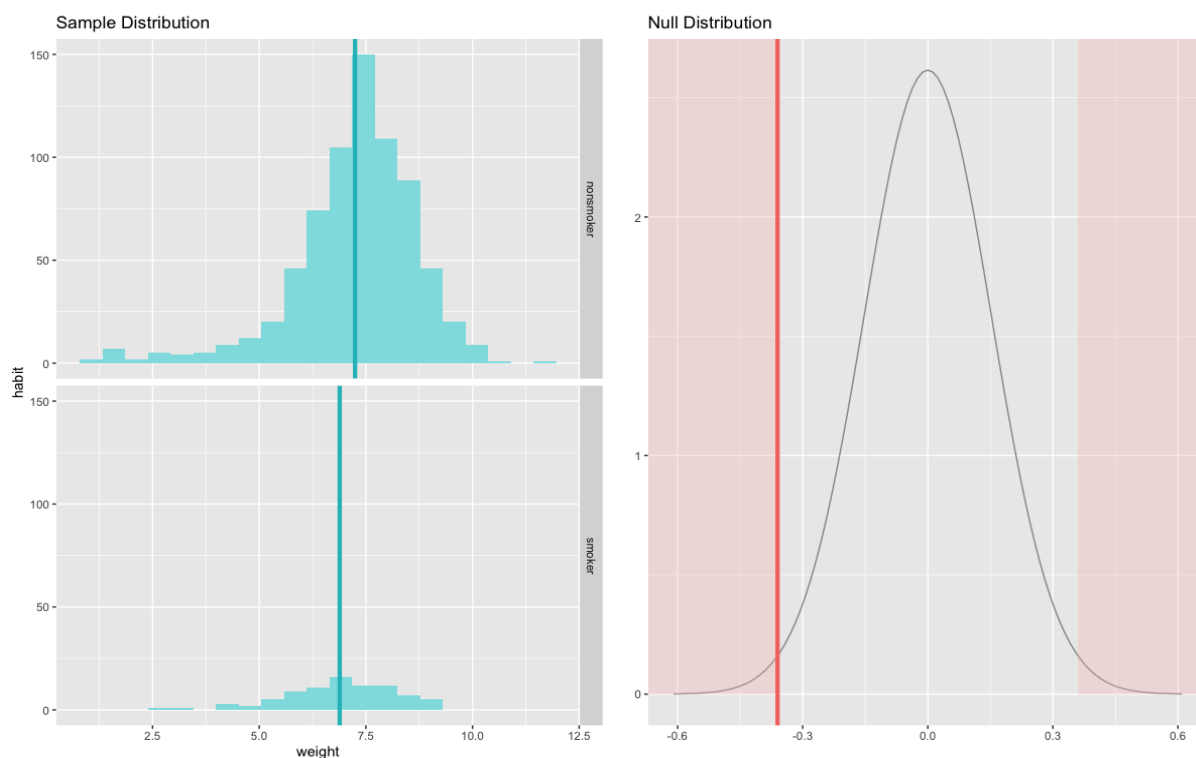
```

```

8 +         conf_level = 0.95,
9 +         order = c("smoker", "nonsmoker"))
10 Response variable: numerical
11 Explanatory variable: categorical (2 levels)
12 n_smoker = 84, y_bar_smoker = 6.8864, s_smoker = 1.3064
13 n_nonsmoker = 716, y_bar_nonsmoker = 7.2468, s_nonsmoker = 1.4521
14 H0: mu_smoker = mu_nonsmoker
15 HA: mu_smoker != mu_nonsmoker
16 t = -2.3625, df = 83
17 p_value = 0.0205

```

on passe directement vers La valeur $t = -2,36$ et la valeur $p\text{-value} = 0.0205$ inférieure à $\alpha = 0,05$ donc le test est significatif (nous rejetons l'hypothèse nulle), nous pouvons donc conclure au même résultat concernant la signification de la différence entre les moyennes.



6.3 tests statistiques

Nous rappelons encore une fois que le caractère qualitatif ou quantitatif des deux variables détermine le type d'analyse, du test, et de visualisation. Prenons le cadre de données (sans valeurs manquantes NA) utilisé dans la partie 6.2 .

6.3.1 Deux variables qualitatives

Un tableau croisé est créé lorsque deux variables qualitatives (Situation et l'habitude de fumer) doivent être comparées. Un tri à plat est réalisé à l'aide de la fonction [table](#).

```

1 > pairchi2 <- table(dfomit$marital,dfomit$habit)
2 > pairchi2

```

```

3
4           nonsmoker smoker
5 married           193     29
6 not married        523     55

```

Afin de compléter le tableau croisé par un test d'indépendance χ^2 . L'indépendance de la variable "Situation" et "l'habitude de fumer" du tableau peut être testée avec ce test et finalement rejetée. Pour effectuer un test de ce type, nous appliquons la fonction `chisq.test` à la table `pairchi2`.

```

1 > chisq.test(pairchi2)
2
3      Pearson's Chi-squared test with Yates' continuity correction
4
5 data:  pairchi2
6 X-squared = 1.787, df = 1, p-value = 0.1813

```

Dans ce cas, la valeur p-value égale à $0.1813 > 0.05$ (seuil de décision), nous on peut facilement retenir l'hypothèse nulle, c'est à dire les deux variables "Situation" et "l'habitude de fumer" sont indépendantes.

6.3.2 Une variable quantitative et une variable qualitative

On a vu de la partie 6.2 comment on effectue un test de t-test en utilisant la fonction `inference()`. Prenons une autre fonction `t.test`. Ainsi, en utilisant la procédure suivante, nous pouvons tester l'idée que les âges moyens sont égaux pour les variables "fage" et "mature"

```

1 > t.test(dfomit$fage ~ dfomit$mature)
2
3      Welch Two Sample t-test
4
5 data:  dfomit$fage by dfomit$mature
6 t = 18.681, df = 184.16, p-value < 2.2e-16
7 alternative hypothesis: true difference in means between group mature mom and group younger mom is not equal to 0
8 95 percent confidence interval:
9   8.440964 10.434428
10 sample estimates:
11 mean in group mature mom mean in group younger mom
12   38.28814                28.85044

```

Nous pouvons rejeter l'hypothèse nulle selon laquelle les moyennes des deux groupes sont égales, car le résultat du t-test est significatif et a une valeur p-value très faible ($p\text{-value} < 2.2e-16 \ll 0.05$). Le test fournit même un intervalle de confiance de 95% pour la valeur de la différence entre les deux moyennes ($CI=[8.440964,10.434428]$). Remarque : nous avons

effectué le test en supposant que la normalité de la distribution est vérifiée pour les deux groupes (fage).

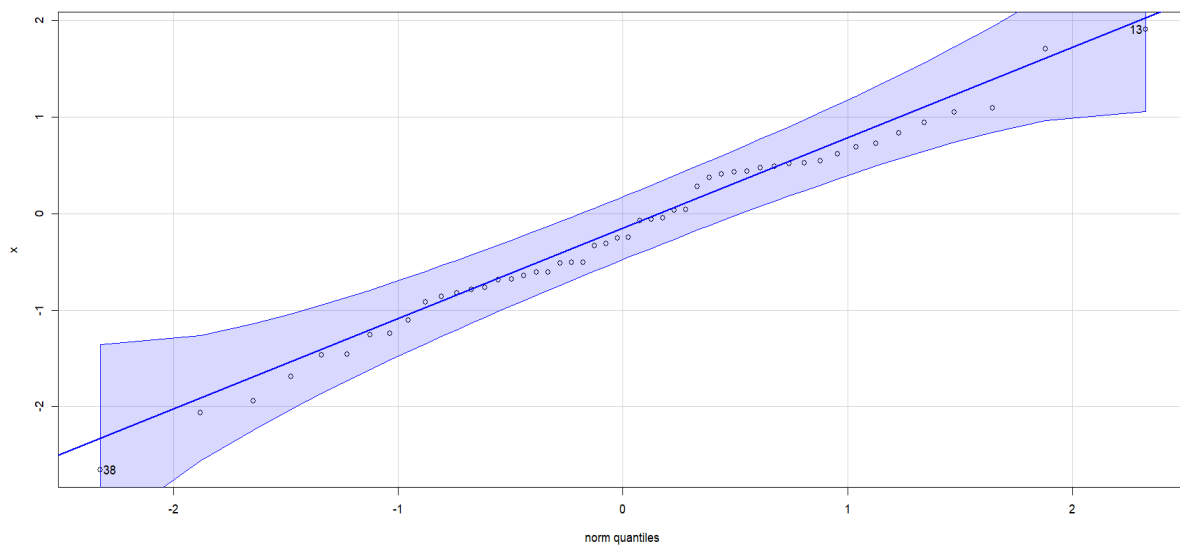
Supposons que nous ayons le cas, en testant l'hypothèse d'indépendance entre deux groupes (moyenne des visites selon l'habitude du fumeur). le test Shapiro-Wilk teste la normalité d'une distribution par la fonction `shapiro.test()`

```
1 > Smoker <- dfomit[dfomit$habit=="smoker",]
2 > Nonsmoker <- dfomit[dfomit$habit=="nonsmoker",]
3 > shapiro.test(Smoker$visits)
4
5         Shapiro-Wilk normality test
6
7 data:  Smoker$visits
8 W = 0.9269, p-value = 0.0001406
9
10 > shapiro.test(Nonsmoker$visits)
11
12        Shapiro-Wilk normality test
13
14 data:  Nonsmoker$visits
15 W = 0.95226, p-value = 1.828e-14
```

Le test est significatif dans les deux cas ($p\text{-value} = 0.0001406$, $p\text{-value} = 1.828e-14$) et réfute la normalité supposée des deux distributions.

On peut effectivement tester la normalité d'une distribution graphiquement par la fonction `qqPlot()` du package `car`,

```
1 > library("car")
2 > x <- rnorm(50,0,1)
3 > qqPlot(x)
```



La fonction `Trace` les résidus studentized d'un modèle linéaire, par rapport aux quantiles théoriques d'une distribution de comparaison. En cas des données sont normales, les points doivent former une ligne droite. Comme tous les points tombent approximativement le long de cette ligne de référence, nous pouvons supposer la normalité de cette série

On peut tester l'hypothèse d'égalité des visites moyennes selon l'habitude de fumer en utilisant un test non paramétrique appelé `wilcox.test()`

```

1 > wilcox.test(dfomit$visits~dfomit$habit)
2
3      Wilcoxon rank sum test with continuity correction
4
5 data:  dfomit$visits by dfomit$habit
6 W = 31526, p-value = 0.4655
7 alternative hypothesis: true location shift is not equal to 0

```

Puisque le test nous donne la valeur de p-value 0.4655 plus grand que 0.05, le test n'est pas significatif. On peut retenir l'hypothèse d'indépendance et considérer que les distributions des visites dans les deux groupes sont égales.

6.3.3 Deux variables quantitatives

Le lien linéaire entre deux variables quantitatives est mesuré par la corrélation. Sa valeur varie de -1 à 1. Il y a un lien linéaire négatif parfait si la corrélation est égale à -1. Il s'agit d'un lien linéaire positif parfait si la valeur est de 1. Il n'y a pas de relation linéaire entre les variables si elle est de 0. Afin de mesurer le degré d'association de deux variables quantitatives, on fait l'appel de la fonction `cor()`

```

1 > cor(dfomit$fage,dfomit$weight)
2 [1] 0.07333513

```

le coefficient de corrélation de pearson égale à 0.07333513 très petit, Il y a pas donc un lien linéaire entre les deux variables "fage" et "weight".

Chapitre 7

Les modèles de régressions

La régression comprend plusieurs méthodes d'analyse statistique qui vous permettent d'approximer une variable en fonction d'autres variables qui lui sont corrélées. Les problèmes de régression se distinguent des problèmes de classification (En apprentissage automatique). Par conséquent, nous considérons le problème de la prédiction des variables quantitatives comme un problème de régression et le problème de la prédiction des variables qualitatives comme un problème de classification. Certaines méthodes comme La régression logistique est à la fois une méthode de régression au sens de prédire la probabilité d'appartenir à chaque classe et aussi une méthode de classification.

7.1 La régression linéaire simple

La régression linéaire simple est une méthode statistique pour trouver une relation linéaire entre la variable explicative X et la variable expliquée Y . Ce modèle consiste à traiter Y comme une fonction affine de X . Alors, le but de la régression linéaire est de trouver une ligne qui correspond au nuage de points de Y en fonction de X .

7.1.1 Modélisation mathématique

L'ajustement affine de Y par X nous informe que Y peut s'écrire comme équation d'une droite :

$$Y = \alpha X + \beta + \epsilon \quad (7.1)$$

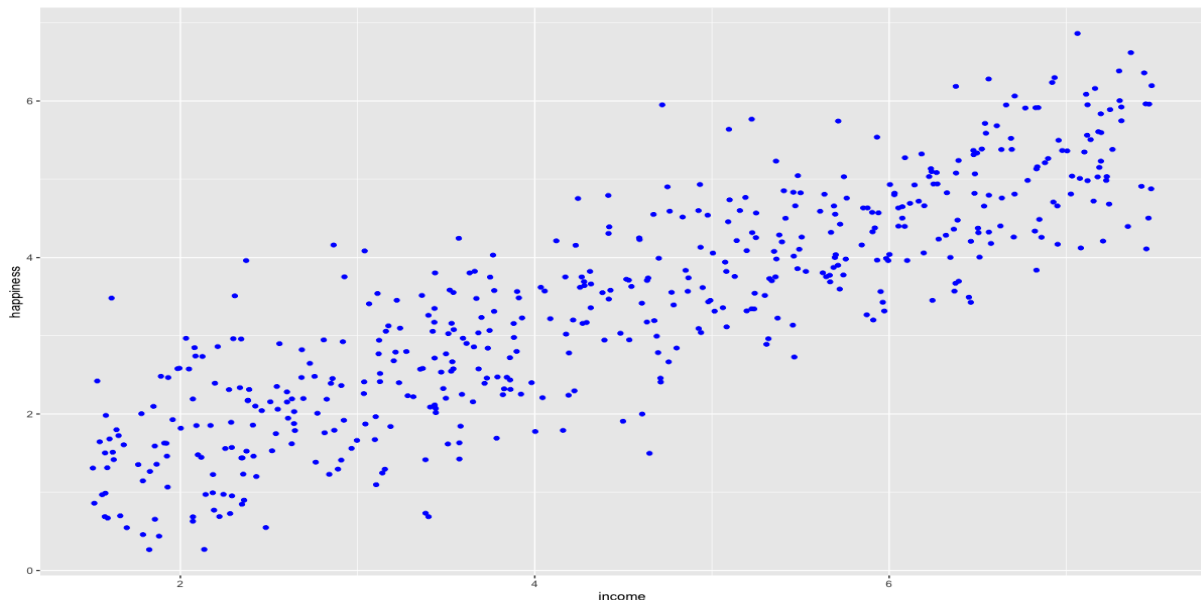
- Y : variable à expliquer (variable dépendante ou variable endogène)
- X : variable explicative (variable indépendante ou variable exogène)
- α : l'ordonnée à l'origine
- β : la pente de la droite
- ϵ : l'erreur $\sim \mathcal{N}(0, \sigma^2)$

7.1.2 Le modèle

Nous utiliserons les données de **income**, et nous allons prendre comme variable explicative X les revenus des individus "income" et le niveau de bonheur "happiness" comme variable à expliquer Y . La première étape c'est la représentation graphique du nuage de points.

En effet si le nuage ne s'apparente pas à une droite, la régression linéaire ne sera pas le meilleur modèle pour notre cas

```
1 > df <- read.csv("~/Desktop/income.csv")
2 > ggplot(data = df, aes(x = income, y = happiness)) + geom_point(col="blue")
```



Le graphique du nuage montre que la régression linéaire sera le meilleur modèle pour notre jeu de données. La régression linéaire simple tente de trouver la meilleure ligne pour prédire le niveau de bonheur sur la base du revenu des individus.

La fonction `lm()` peut être utilisée pour déterminer les coefficients du modèle linéaire (eq 7.1)

```
1 > SimRegModel <- lm(formula = happiness~income, data = df)
2 > summary(SimRegModel) #Le résumé statistique du modèle
3
4 Call:
5 lm(formula = happiness ~ income, data = df)
6
7 Residuals:
8      Min       1Q   Median       3Q      Max
9 -2.02479 -0.48526  0.04078  0.45898  2.37805
10
11 Coefficients:
12              Estimate Std. Error t value Pr(>|t|)
13 (Intercept)  0.20427    0.08884   2.299  0.0219 *
14 income       0.71383    0.01854  38.505 <2e-16 ***
15 ---
16 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
17
```

```

18 Residual standard error: 0.7181 on 496 degrees of freedom
19 Multiple R-squared:  0.7493,    Adjusted R-squared:  0.7488
20 F-statistic: 1483 on 1 and 496 DF,  p-value: < 2.2e-16

```

Nous devons confirmer que ce modèle est statistiquement significatif avant de l'utiliser pour prédire le "revenu", c'est-à-dire :

- Les variables dépendantes et indépendantes ont une relation statistiquement significative entre elles.
- Le modèle que nous avons créé correspond étroitement aux faits observés.
- Examiner l'erreur standard des résidus (RSE), la valeur R^2 et les F-statistiques pour déterminer dans quelle mesure le modèle s'adapte aux données.

dans le tableau des coefficients, les valeurs p-value pour l'ordonnée à l'origine et la variable prédicteur sont toutes deux hautement significatives ($p - value_{\beta} = 0.0219 < 0.05$, $p - value_{\alpha} = 2e-16 < 0.001 < 0.05$), nous pouvons donc rejeter l'hypothèse nulle et accepter l'hypothèse alternative, ce qui signifie qu'il existe une association significative entre la variable explicative et le variable à expliquer. Ainsi, notre modèle de régression simple peut être écrit comme une fonction du revenu

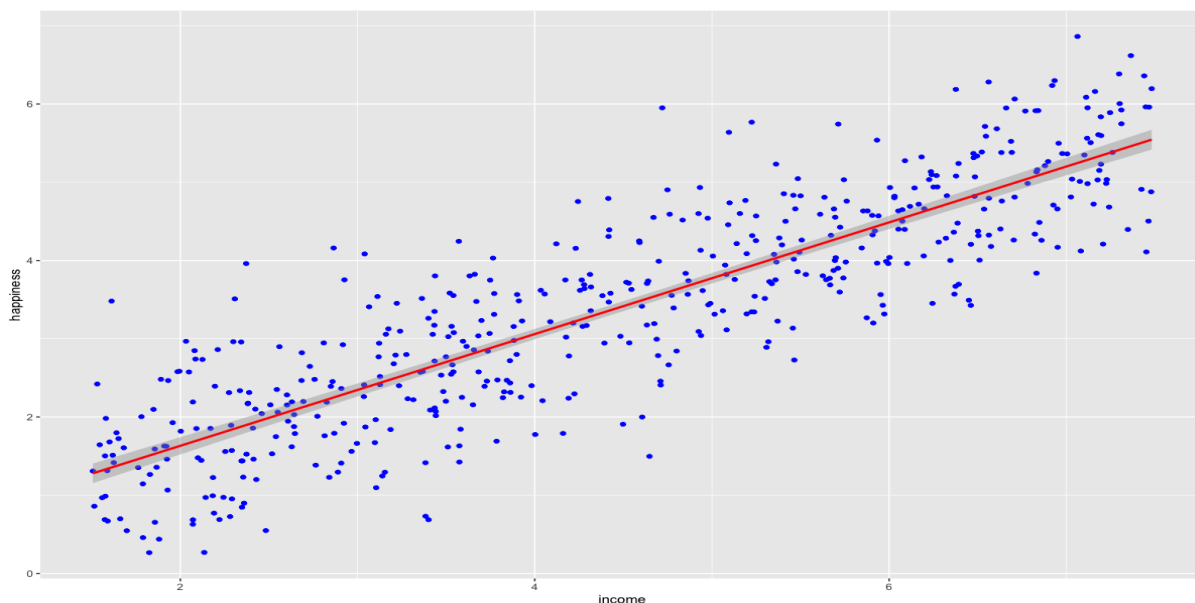
$$happiness = 0.71383 \times income + 0.20427 \quad (7.2)$$

Traçons la droite de régression avec la fonction `geom_smooth()`

```

1 > ggplot(data = df, aes(x = income, y = happiness)) +
2 +   geom_point(col="blue") +
3 +   geom_smooth(method = "lm", col="red")

```



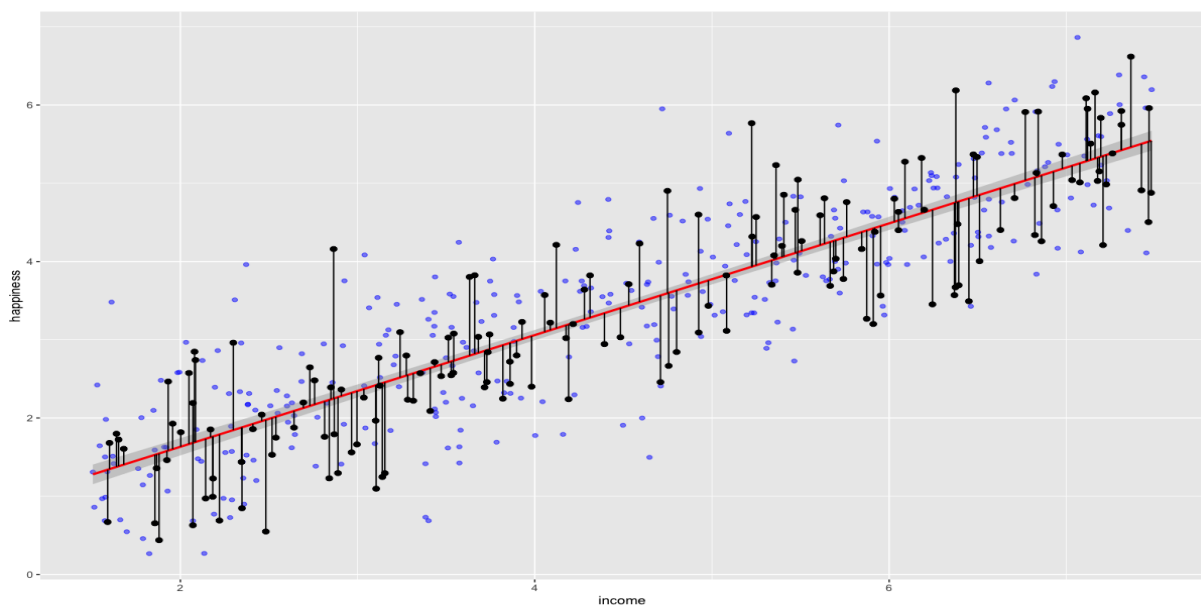
De plus on a Erreur standard résiduelle : l'estimation σ de ϵ qui est de 0.7181.

On visualise les résidus de 33% du cadre de donnée

```

1 > X <- df$income
2 > samp = sample(1:498)[1:164] #Indices de notre echantillon
3 > X_residuals = X[samp]
4 > y = df$happiness[samp]
5 > y_hat = SimRegModel$fitted.values[samp]
6 > ggplot() + geom_point(data = df, aes(x = income, y = happiness), alpha = 0.5, col = "blue") +
7 +   geom_smooth(data = df, aes(x = income, y = happiness), method = 'lm', col = "red") +
8 +   geom_point(aes(x = X_residuals, y = y), color = 'black', size = 2,) +
9 +   geom_segment(aes(x = X_residuals, y = y_hat, xend = X_residuals, yend = y), color = 'black')

```



Dans le tableau de précision des modèles, 0,7493 est le coefficient de détermination R^2 (est le carré du coefficient de corrélation de Pearson dans le cas de la régression simple). Dans ce cas, notre coefficient est relativement faible étant donné qu'un bon coefficient se situe autour de 0,85. La F-statistique donne la signification globale du modèle. Elle évalue si au moins une variable prédictive a un coefficient non nul. Elle devient plus importante lorsque l'on commence à utiliser des prédicteurs multiples, comme dans la régression linéaire multiple.

Nous utilisons la fonction `predict()` pour prévoir les variables que notre modèle devra expliquer.

```

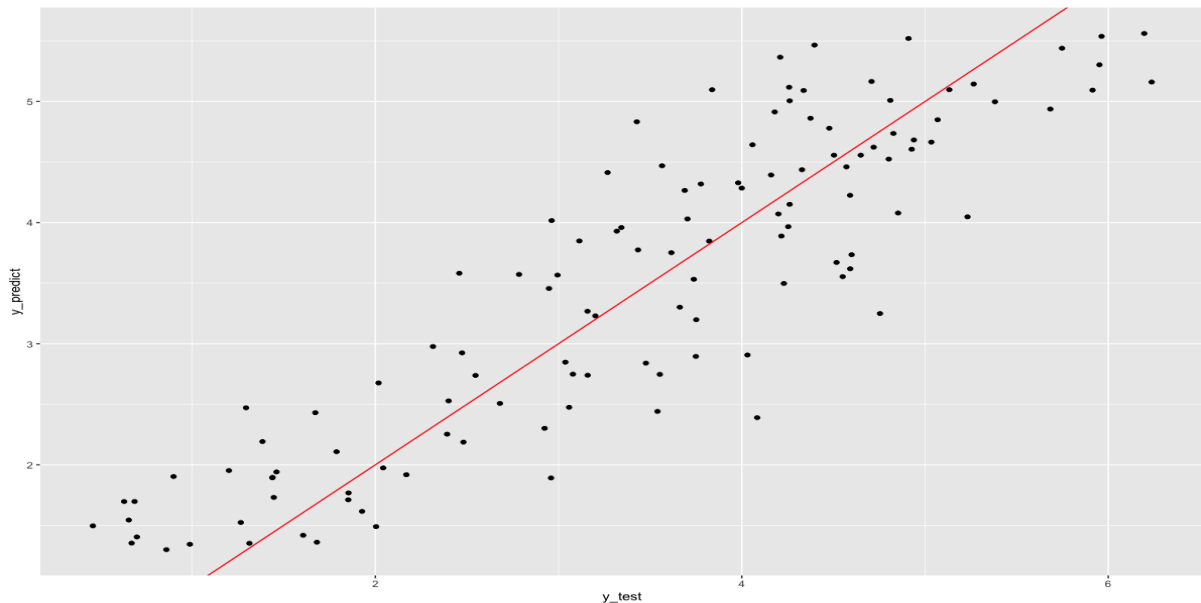
1 > set.seed(207)
2 > sampT <- sample(1:498)[1:120]
3 > df_train <- df[-sampT,]
4 > df_test <- df[sampT,]
5 > SimRegModelT <- lm(data = df_train, formula = happiness~income)
6 > y_predict <- predict(SimRegModelT, data.frame(income=df_test[,2]))
7 >- y_predict[1:10] #Afficher les 10 valeurs de y_predict
8           1           2           3           4           5           6           7           8           9          10

```

```
9 3.301276 1.697078 2.527711 3.229452 4.392723 5.165170 3.566386 1.941314 4.412933 2.839422
```

On peut visualiser la qualité de notre prédiction par le graphique du nuage

```
1 > y_test = df_test\shappiness
2 > ggplot() + geom_point(aes(x = y_test, y = y_predict)) +
3 +   geom_abline(slope = 1, color = 'red')
```



7.2 la régression linéaire multiple

La régression linéaire multiple est une méthode statistique pour trouver une relation linéaire entre plusieurs (nombre fini) variables X_i et la variable à expliquer Y . Ce modèle consiste à traiter Y comme une fonction de X . Le modèle de régression linéaire multiple est une généralisation du cas simple.

7.2.1 Modélisation mathématique

L'ajustement affine de Y par X nous informe que Y peut s'écrire sous forme matricielle :

$$Y = \alpha X + \beta + \epsilon \quad (7.3)$$
$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nk} \end{pmatrix} \quad \beta = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} \quad \epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

- Y : variable à expliquer (variable dépendante ou variable endogène)
- X : Matrice des variables explicatives (variable indépendante ou variable exogène)
- α : l'ordonnée à l'origine

- β_i : vecteur paramètres inconnus du modèle à estimer.
- ϵ_i : vecteur des erreurs ϵ_i associé à la i ème observation $\sim \mathcal{N}(0, \sigma^2)$

7.2.2 Le modèle

L'ensemble de données cardiaques utilisé dans cette partie a été collecté à partir d'une enquête portant sur 498 villes, où des informations ont été recueillies sur le pourcentage de résidents souffrant de maladies cardiaques (heart.disease), de fumeurs (smoking) et de cyclistes (biking) dans chaque ville. Nous recherchons des éléments qui pourraient affecter les maladies cardiaques (heart.disease).

Ainsi, le modèle de régression linéaire que nous voulons construire est une fonction comme $f(\text{heart.disease}) = (\text{biking}, \text{smoking})$.

```

1 > df <- read.csv("~/Desktop/heart.csv")
2 > summary(df)
3      biking      smoking      heart.disease
4  Min.   : 1.119   Min.   : 0.5259   Min.     : 0.5519
5  1st Qu.:20.205   1st Qu.: 8.2798   1st Qu.: 6.5137
6  Median :35.824   Median :15.8146   Median :10.3853
7  Mean   :37.788   Mean   :15.4350   Mean    :10.1745
8  3rd Qu.:57.853   3rd Qu.:22.5689   3rd Qu.:13.7240
9  Max.   :74.907   Max.   :29.9467   Max.    :20.4535

```

Nous utilisons le même processus que dans la section précédente et utilisons les mêmes fonctions.

```

1 > MulRegModel <- lm(formula = heart.disease~biking+smoking, data = df)
2 > summary(MulRegModel)
3
4 Call:
5 lm(formula = heart.disease ~ biking + smoking, data = df)
6
7 Residuals:
8      Min       1Q   Median       3Q      Max
9  -2.1789  -0.4463   0.0362   0.4422   1.9331
10
11 Coefficients:
12             Estimate Std. Error t value Pr(>|t|)
13 (Intercept)  14.984658   0.080137  186.99  <2e-16 ***
14 biking       -0.200133   0.001366 -146.53  <2e-16 ***
15 smoking       0.178334   0.003539   50.39  <2e-16 ***
16 ---
17 Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
18
19 Residual standard error: 0.654 on 495 degrees of freedom
20 Multiple R-squared:  0.9796,    Adjusted R-squared:  0.9795
21 F-statistic: 1.19e+04 on 2 and 495 DF,  p-value: < 2.2e-16

```

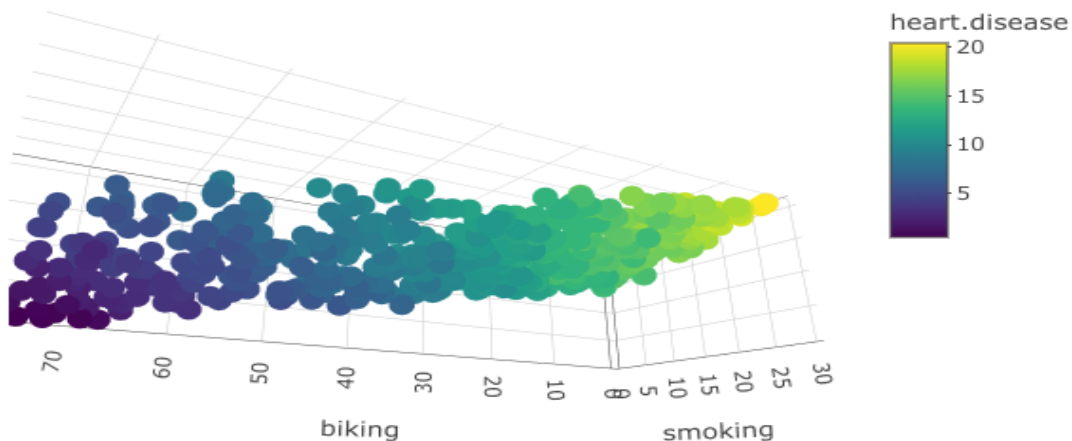
Nous rejetons l'hypothèse H_0 avec un niveau de confiance de 95% car nous pouvons voir que les valeurs, ($p - value_{intercept} = p - value_{biking} = p - value_{smoking} = 2e-16 < 0.05$) sont significativement inférieures à 5% pour toutes les variables, y compris l'intercept. Ce qui signifie qu'il existe une association significative entre la variable explicatives (biking et smoking) et le variable à expliquer (heart.disease) De plus, le coefficient d'ajustement R^2 est de 0,9796, ce qui donne à notre modèle un score de 97,96%, ce qui est un excellent résultat. Ainsi, notre modèle de régression multiple

$$heart.disease = -0.200133 \times biking + 0.178334 \times smoking + 14.984658 \quad (7.4)$$

Utilisons la fonction `plot_ly()` du package `plotly` pour représenter interactivement les trois variables en trois dimensions.

```

1 library(plotly)
2 plot_ly(df, x = ~biking, y = ~smoking, z = ~heart.disease, color = ~heart.disease)
3 \%>\% add_markers()
4 \%>\% layout(scene = list(xaxis = list(title = 'biking'),
5                             yaxis = list(title = 'smoking'),
6                             zaxis = list(title = 'heart.disease')))
```



7.3 La régression des moindres carrés partiels

La régression PLS (partial least square) est une méthode ancienne largement utilisée lors de l'analyse de données spectrales discrétisées et donc toujours de grande dimension. La régression PLS s'avère concrètement une méthode efficace qui justifie son emploi très répandu mais présente le défaut de ne pas se prêter à une analyse statistique traditionnelle qui exhiberait les lois de ses estimateurs. En pratique remplace souvent la régression linéaire ordinaire. Cette méthode s'applique quand le nombre de variables est plus important que le nombre d'observations, ou encore lorsqu'il existe une forte multicollinéarité

entre les variables explicatives. Elle s'applique également en présence de données manquantes. Elle est ainsi restée un marge des approches traditionnelles de la Statistique.

La régression PLS recherche de manière itérative une séquence de composants (ou variables sous-jacentes) orthogonaux les uns aux autres. Ces composants, appelées composantes PLS, sont choisies pour maximiser la covariance avec la variable explicative Y. Le choix du nombre de composants est important car il influence fortement la qualité de la prévision.

Le modèle de régression PLS est construit à partir de l'échantillon d'apprentissage puis les prédictions sont faites sur l'échantillon de validation. Nous construisons d'abord un modèle PLS à un composant, puis nous répétons ce processus pour un PLS à 2, 3, ..., k composants. Le nombre de composants retenus correspond au modèle résultant en une erreur de prédiction minimale. Cette procédure se fait dans le package `pls`. Pour construire le modèle prédictif, nous disposons d'une base de données de 177 types de sols (individus) avec à la fois une teneur en carbone organique (CO, variable à expliquer) et un spectre obtenu dans le visible et le proche infrarouge (400 nm-2500 nm), fournissant 2101 variables explicatives. Nous avons ensuite voulu prédire la teneur en carbone organique de trois nouveaux sols.

les données sont importées du site https://r-stat-sc-donnees.github.io/spe_bretagne.txt, calculons la moyenne et l'écart-type de chaque spectre et impliquons un chagement qui consiste à centrer réduire les données sauf colonnes "CO" correspond à Y.

```
1 > df <- read.table("https://r-stat-sc-donnees.github.io/spe_bretagne.txt", sep=";",
2 + header=TRUE, row.names=1)
3 > avr.raw <- apply(df[, -1], 1, mean)
4 > df[, -1] <- sweep(df[, -1], 1, avr.raw, FUN="-")
5 > sd.row <- apply(df[, -1], 1, sd)
6 > df[, -1] <- sweep(df[, -1], 1, sd.row, FUN="/")
```

La visualisation de la variable à expliquer Y par le graphique histogramme.

```
1 > hist(df[, "CO"], prob=TRUE, main="", xlab="Organic carbon content")
2 > lines(density(df[, "CO"]), col="red")
3 > rug(df[, "CO"], col="blue")
```

Dans la figure Figure 7.1 Toutes les observation sont moins de 5 sauf une qui est extrême (supérieure à 8)

```
1 > which(df[, 1]>8)
2 [1] 79
3 > max(df[-79, 1])
4 [1] 4.89
```

l'individu 79 prend une valeur très élevée (une valeur extrême qui doit être enlevé pour la construction du modèle), par contre les autres valeurs ne dépassent pas 4.89.

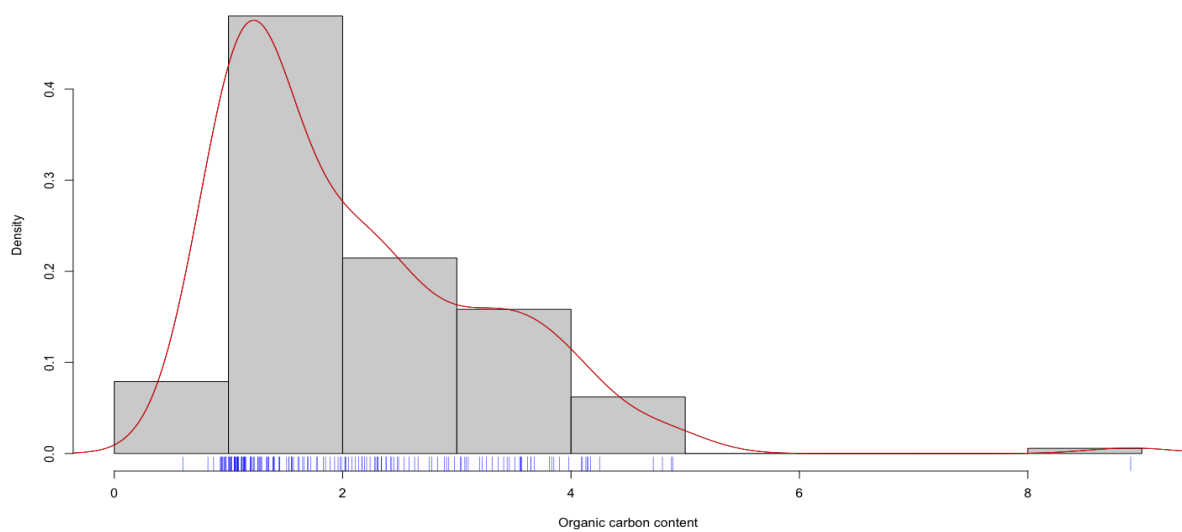


FIGURE 7.1 – Représentation de la teneur en carbone organique

```

1 > df <- df[-79,]
2 > dim(df)
3 [1] 176 2102

```

En représentant chaque spectre par un trait différent (trait plein, tiret, etc) et/ou par une couleur différente selon la valeur de la variable CO, on divisera la variable CO en 7 classes environ égales par fonction `cut`

```

1 > coul <- as.numeric(cut(df[,1], quantile(df[,1], prob=seq(0,1,by=1/7)), include.lowest = TRUE))
2 > palette(terrain.colors(7,alpha=0.5))
3 > matplot(x=400:2500, y=t(as.matrix(df[, -1])), type="l", lty=1,
4 +         col=coul, xlab="wave length", ylab="Reflectance")

```

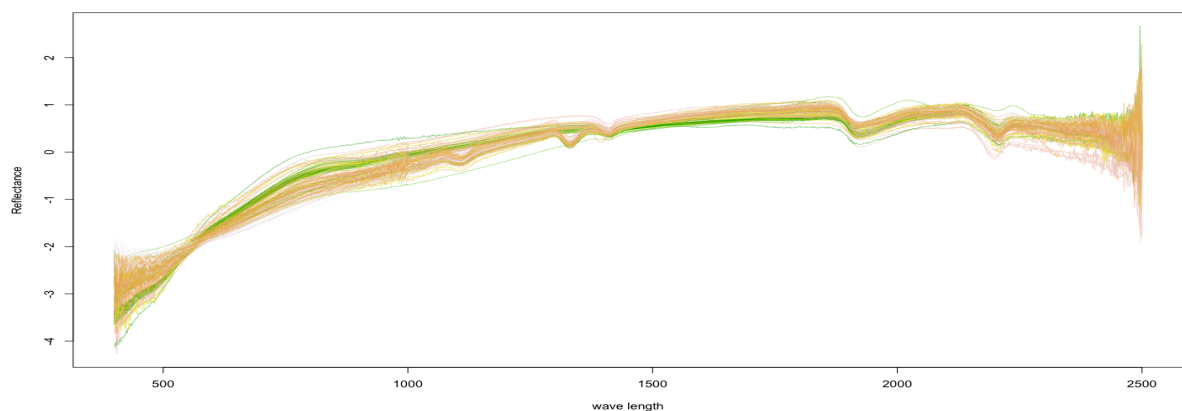


FIGURE 7.2 – Représentation des individus

des courbes assez proches supposent des valeurs proches pour la variable CO, en fait les courbes sont tracées avec le même couleur (voir Figure 7.2). Nous réduisons les variables explicatives grâce à l'argument `scale`, et nous fixer un nombre maximum de composantes PLS (notre cas 100).

```
1 library(pls)
2 > modele.pls <- plsr(CO~., ncomp=100, data=df, scale=TRUE, validation="CV")
```

La validation croisée établit le nombre de composantes PLS par défaut. Tout d'abord, calculons l'erreur d'ajustement et l'erreur de prédiction acquises à partir des composantes PLS 1,2, ... ,100. (voir Figure 7.3)

```
1 > msepcv.pls <- MSEP(modele.pls, estimate=c("train", "CV"))
2 > palette("default")
3 > plot(msepcv.pls, lty=1, type="l", legendpos="topright", main="")
```

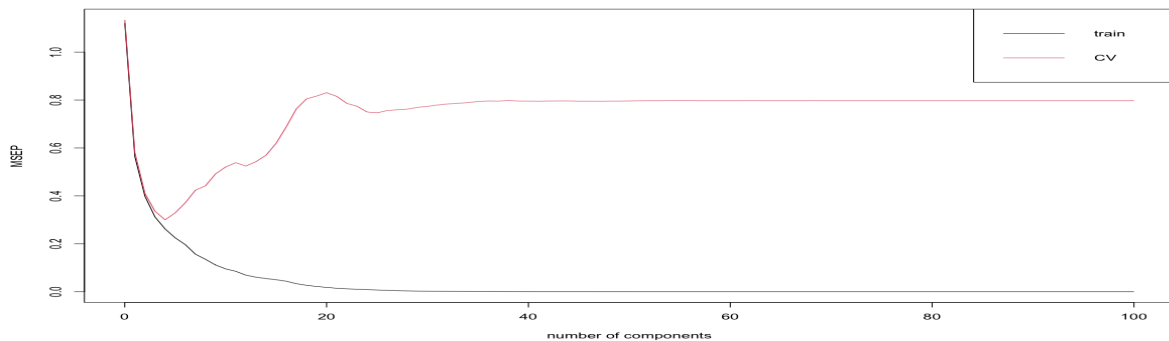


FIGURE 7.3 – Evolution des erreurs en fonction du nombre de composantes PLS.

L'erreur d'ajustement diminue avec le nombre de composants et une décroissance suivie d'une croissance pour l'erreur de prévision, le nombre optimal de composantes pour la prévision correspond à la valeur pour laquelle l'erreur de prévision est minimum.

```
1 > ncomp.pls <- which.min(msepcv.pls$val["CV",,]) - 1
2 > ncomp.pls
3 4 comps
4 4
```

On soustrait 1 car la première erreur est donnée pour 0 composante, et donc on a 4 composantes PLS. Et alors le modèle finale du PLS

```
4 > reg.pls <- plsr(CO~., ncomp=ncomp.pls, data=df, scale=TRUE)
```

La fonction `residuals` calcule les résidus du modèle à 4 composantes.

```
1 > res.pls <- residuals(reg.pls)
```

```

2 > plot(res.pls[, , ncomp.pls], pch=18, cex=.5, ylab="Résidus", main="", col="red")
3 > abline(h=c(-2,0,2), lty=c(2,1,2))

```

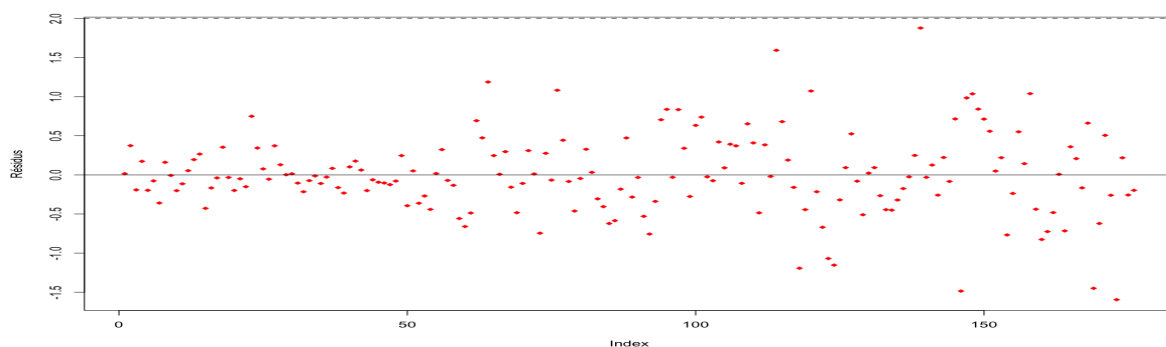


FIGURE 7.4 – Représentation des résidus du modèle à 4 composantes

Ça se voit clairement que la variance des résidus (plus grande pour les derniers individus, et plus faible pour les premiers) montre que l'échantillon n'est pas très homogène.

Pour prévoir une nouvelle valeur, et afin de comparer la teneur en CO prédite par la régression PLS avec les valeurs réelles observées, nous avons obtenu un ensemble de trois courbes complémentaires.

```

1 > dfNew <- read.table("https://r-stat-sc-donnees.github.io/spe_nouveau.txt", sep=";", +
2 +   header=TRUE, row.names=1)
3 > avr.raw <- apply(dfNew[, -1], 1, mean)
4 > dfNew[, -1] <- sweep(dfNew[, -1], 1, avr.raw, FUN="-")
5 > sd.row <- apply(dfNew[, -1], 1, sd)
6 > dfNew[, -1] <- sweep(dfNew[, -1], 1, sd.row, FUN="/")

```

La fonction `predict` est utilisée pour obtenir les valeurs prédites,

```

1 > pred <- predict(reg.pls, ncomp=ncomp.pls, newdata=dfNew[, -1])
2 > pred
3 , , 4 comps
4
5           CO
6 3236      0.4065104
7 rmqs_726  2.6930854
8 rmqs_549  2.6035296

```

On compare les valeurs prédites aux valeurs mesurées :

```

1 > dfNew[, 1]
2 [1] 1.08 1.60 1.85

```

Bibliographie

- [1] Philippe J. S. De BROUWER. *The Big R-Book : From Data Science to Learning Machines and Big Data*. en. Google-Books-ID : Usb9DwAAQBAJ. John Wiley & Sons, oct. 2020. ISBN : 978-1-119-63272-6.
- [2] François HUSSON, Pierre-André CORNILLON, Arnaud GUYADER, Nicolas JÉGOU, Julie JOSSE, Nicolas KLUTCHNIKOFF, Erwan Le PENNEC, Eric MATZNER-LOBER, Laurent ROUVIÈRE et Benoît THIEURMEL. *R pour la statistique et la science des données*. Pratique de la statistique. Rennes : Presses universitaires de Rennes, 2018. ISBN : 978-2-7535-7573-8.
- [3] Pierre-André CORNILLON et Eric MATZNER-LØBER. *Régression : Théorie et applications*. Statistique et probabilités appliquées. Springer, 2007, 317 p. URL : <https://hal.archives-ouvertes.fr/hal-00955892>.