R pour l'analyse de données

Prise en main

Ressources et cours consultés

- Udemy : *Programmer avec R en 5h en partant de zéro + Projet (2025) de Mikail Altundas*
- Udemy : *Programmer en R en 8h pour la Data Science de A à Z d'Amandine Velt*
- Wikipédia R
- Guide data.gouv R
- Bookdown — Introduction à R
- CRAN (site officiel)

11 septembre 2025

Table des matières

Introduction

L'**analyse de données** est l'ensemble des méthodes et techniques qui permettent de transformer des données brutes en informations compréhensibles et exploitables pour orienter la prise de décision.

Parmi les langages utilisés, **R** occupe une place centrale pour le traitement, l'analyse et la visualisation des données. Il est particulièrement apprécié pour ses capacités statistiques, sa richesse en bibliothèques spécialisées et ses outils graphiques puissants.

Ce support a pour objectif de présenter les bases du langage R, utiles pour manipuler des données, produire des analyses, et construire des visualisations claires et reproductibles.

1 Installation de R et RStudio

Pour utiliser le langage **R**, deux outils sont nécessaires :

— **R** : le moteur de calcul statistique.
— **RStudio** : une interface de développement (IDE) plus conviviale pour écrire et exécuter du code R.

1. Installer R

1. Aller sur le site officiel : `https://cran.r-project.org`
2. Choisir le système d'exploitation (*Windows*, *macOS* ou *Linux*).
3. Télécharger la dernière version stable de R.
4. Suivre les étapes d'installation par défaut.

2. Installer RStudio

1. Aller sur le site : `https://posit.co/download/rstudio-desktop`
2. Télécharger la version gratuite de **RStudio Desktop Open Source**.
3. Installer le logiciel comme n'importe quelle application.

3. Lancer RStudio

Une fois les deux outils installés, il suffit de lancer **RStudio**. Celui-ci utilise automatiquement l'interpréteur R installé en arrière-plan.

Remarque : Il est recommandé d'utiliser RStudio plutôt que l'interface de base de R, car il offre un environnement plus complet, avec éditeur de code, console, visualisation des données, historique des commandes, etc.

2 Présentation de l'interface RStudio

Une fois lancé, **RStudio** se présente sous la forme d'une interface divisée en quatre volets principaux :

- **Script (en haut à gauche)** : permet d'écrire et enregistrer du code R dans des fichiers.
- **Console (en bas à gauche)** : exécute les commandes R en temps réel.
- **Environnement / Historique (en haut à droite)** : affiche les objets en mémoire, les jeux de données, et l'historique des commandes.
- **Plots / Packages / Fichiers (en bas à droite)** : permet d'afficher les graphiques, gérer les packages, explorer les fichiers, etc.

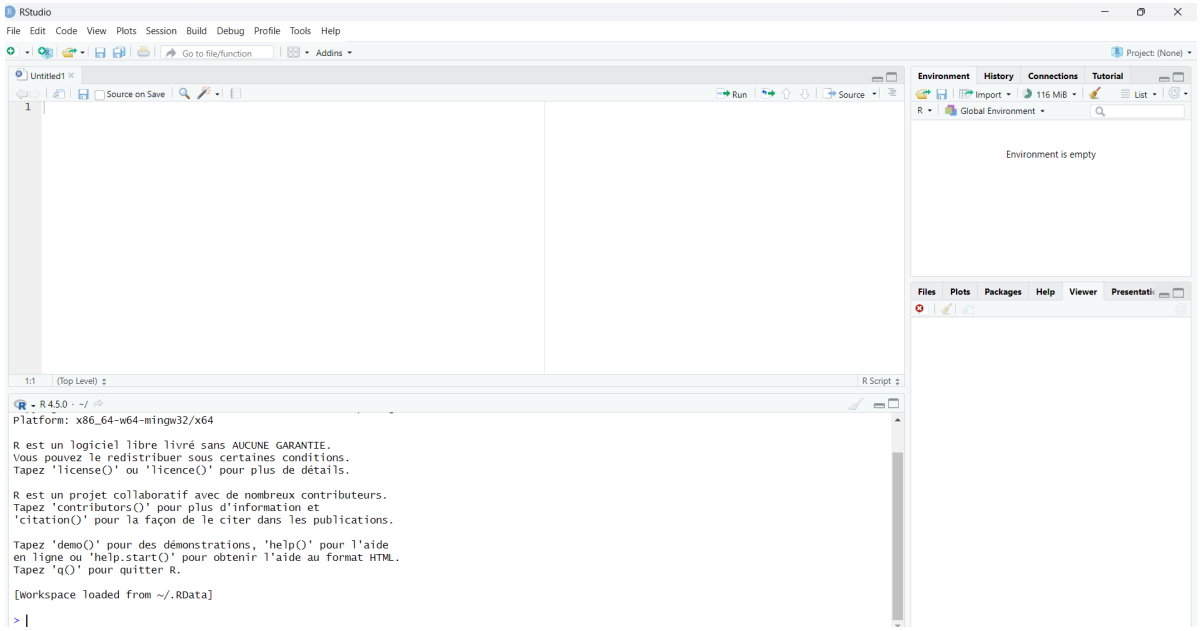


FIGURE 1 – Interface RStudio au démarrage

3 Créer et exécuter un script R dans RStudio

Dans RStudio, l'écriture du code se fait dans un fichier appelé **script R**, qui peut être sauvegardé et réutilisé.

1. Créer un nouveau fichier R

Pour créer un nouveau script R :

— Aller dans le menu `File` > `New File` > `R Script`.
— Un nouvel onglet s'ouvre dans la partie supérieure gauche de RStudio.
— Ce fichier peut être sauvegardé avec l'extension `.R`.

2. Types de fichiers courants

RStudio permet de créer différents types de fichiers selon les besoins :

— **R Script (.R)** : pour écrire du code R classique.
— **R Markdown (.Rmd)** : pour combiner code, texte et rendu dynamique.
— **Shiny App** : pour créer des applications interactives en R.
— **Quarto (.qmd)** : alternative moderne à R Markdown pour des rapports interactifs.

3. Exécuter du code R

Une fois un script ouvert, il est possible d'exécuter une ou plusieurs lignes de code de deux manières :

— **Ctrl** + **Entrée** (ou **Cmd** + **Entrée** sur Mac) : exécute la ligne sélectionnée.
— Cliquer sur le bouton `Run` en haut à droite de l'éditeur.

Le résultat de l'exécution s'affiche dans la console, en bas à gauche de l'interface.

4 Opérations Script.R

R peut être utilisé pour effectuer des opérations algébriques élémentaires. On peut également enregistrer des valeurs dans des variables et travailler ensuite avec celles-ci.

1. Calculs simples

Voici quelques exemples de calculs réalisés directement dans la console R :

```
3 + 2
3 - 2
3 * 2
```

```
3 / 2
3^2

# Plusieurs instructions sur une seule ligne
3 + 2 ; 3 - 2 ; 3 * 2 ; 3^2
```

2. Affectations et expressions plus complexes

On peut stocker des valeurs dans des variables (ou scalaires) avec le symbole = ou <-. Ensuite, ces variables peuvent être combinées dans des expressions :

```
x = 2
a = 2
b = 3
c = 4

x + c
b * x + c
a * x^2 + b * x + c
b / (x + c)^a
x ^ (a + b + c)
```

3. Opérations logiqueset de comparaisons

R permet également de manipuler des valeurs logiques :

```
TRUE
FALSE
Exemple
3 > 2      # TRUE
4 == 5     # FALSE
5 != 2     # TRUE

== : égalité
!= : différence
> : supérieur
< : inférieur
& : et logique
| : ou logique
```

5 Vecteurs

Les **vecteurs** sont les structures de base de R. Ce sont des séquences ordonnées d'éléments *de même type* : numériques, chaînes de caractères ou logiques.

5.1 Syntaxe

```
nom_vecteur <- c(valeur1, valeur2, ..., valeurN)
```

Exemple :

```r
vec1 = c(2.8, 2.4, 2.1, 3.6, 2.8)            # vecteur numérique
vec2 = c("rouge", "vert", "vert", "jaune")   # vecteur de chaînes de caractères
vec3 = c(TRUE, TRUE, FALSE, FALSE, FALSE)    # vecteur logique (booléen)
```

5.2 Opérations sur les vecteurs

5.2.1 Création par répétition

```r
rep(4, 3)                          # répète la valeur 4 trois fois -> 4 4 4
vec4 = rep(vec1, 2)                # répète tout le vecteur vec1 deux fois
vec5 = rep(vec1, c(2, 1, 3, 3, 2)) # répète chaque élément de vec1
```

5.2.2 Création par suite ou séquence

```r
vec6 = 1:10                 # crée une séquence d'entiers de 1 à 10
vec7 = seq(3, 5, by = 0.2)  # crée une séquence de 3 à 5 avec un pas de 0.2
```

5.2.3 Création élément par élément

```r
vec8 = numeric()      # crée un vecteur numérique vide
vec8[1] = 41.8        # affecte la valeur 41.8 à la 1ère position
vec8[2] = -0.3        # affecte la valeur -0.3 à la 2ème position
vec8[3] = 92          # affecte la valeur 92 à la 3ème position
vec8                  # affiche le vecteur rempli
```

5.2.4 Création avec nombres aléatoires

```
set.seed(123)                           # fixe le hasard
moyenne_de_la_classe <- sample(1:20, 20) # 20 nombres tirés entre 1 et 20
moyenne_de_la_classe                    # affiche le résultat
```

5.2.5 Opérations arithmétiques

```
vec1 = c(1, 2, 3)      # vecteur 1
vec2 = c(4, 5, 6)      # vecteur 2
vec1 + vec2            # addition élément par élément
vec1 - vec2            # soustraction élément par élément
vec1 * vec2            # multiplication élément par élément
vec2 / vec1            # division élément par élément
vec1 ^ 2               # puissance (carré) de chaque élément
```

5.2.6 Opérations logiques

```
v = c(4, 7, 9, 3, 6)  # vecteur numérique
v > 5                 # test supérieur à 5 (renvoie TRUE/FALSE)
v == 3                # test égal à 3
v != 7                # test différent de 7
```

5.2.7 Modification

```
v = c(10, 20, 30, 40, 50)   # vecteur initial
v[2] = 99                   # remplace la 2ème valeur par 99
v[c(1, 3)] = 0              # remplace les 1ère et 3ème valeurs par 0
v                           # affiche le vecteur modifié
```

5.2.8 Combinaison de vecteurs

```
a = c(1, 2, 3)              # vecteur a
b = c(4, 5)                 # vecteur b
c1 = c(a, b)                # concatène a et b
c2 = a + 10                 # ajoute 10 à chaque élément de a
c3 = a * b                  # multiplication élément par élément
```

6 Listes

Une **liste** en R est une structure de données flexible qui peut contenir des objets de types variés : vecteurs, matrices, fonctions, data frames ou même d'autres listes.

6.1 Syntaxe

```
nom_liste <- list(nom1 = objet1, nom2 = objet2, ...)
```

Exemple :

```r
ma_liste <- list(
  nom = "Alice",                    # élément texte (chaîne de caractères)
  notes = c(15, 18, 13),            # vecteur numérique
  moyenne = mean(c(15, 18, 13)),    # valeur calculée (moyenne des notes)
  validation = TRUE                 # valeur logique
)
ma_liste                            # affiche la liste complète
```

6.2 Opérations sur les listes

6.2.1 Accès aux éléments

```r
ma_liste$notes      # Accès par le nom
ma_liste[[2]]       # Accès par position
ma_liste[2]         # Renvoie une sous-liste
names(ma_liste)     # Noms des composantes
```

6.2.2 Modification des éléments

```r
ma_liste$remarque <- "Très bon travail"  # Ajout
ma_liste$validation <- FALSE             # Modification
ma_liste$nom <- NULL                     # Suppression
```

6.2.3 Listes imbriquées

```r
liste2 <- list(
  nom = "Projet",                 # élément texte
  contenu = list(                 # sous-liste
```

```
    data = c(1, 2, 3),              # vecteur numérique
    titre = "Analyse 2025"          # texte
  )
)
liste2$contenu$data                 # accès via le nom
liste2[[2]][["data"]]               # accès via les indices de liste
```

6.2.4 Fonctions utiles

```
length(ma_liste)      # Nombre d'éléments
str(ma_liste)         # Structure
names(ma_liste)       # Noms des composantes
is.list(ma_liste)     # Est-ce une liste ?
```

7 Matrices

Une **matrice** en R est une structure bidimensionnelle (lignes × colonnes) contenant uniquement des éléments du même type. Elle est utile pour les calculs statistiques, algébriques, ou pour représenter des tableaux numériques.

7.1 Syntaxe

`nom_matrice <- matrix(vecteur, nrow = x, ncol = y)`

Exemple :

```
mat1 <- matrix(1:10, ncol = 5)   # matrice avec 10 nombres sur 5 colonnes
mat1                             # affiche la matrice
```

7.2 Opérations sur les matrices

7.2.1 Accès aux éléments

```
mat1[ , 3]          # colonne 3
mat1[2, ]           # ligne 2
mat1[1, c(2, 4)]    # éléments aux colonnes 2 et 4 de la ligne 1
```

7.2.2 Assemblage de colonnes ou de lignes

```r
cbind(1:3, 4:6)            # concaténation colonne
rbind(c(1,2,3), c(4,5,6))  # concaténation ligne
```

7.2.3 Création d'une matrice diagonale

```r
diag(c(2, 1, 5))   # matrice diagonale (2,1,5 sur la diagonale)
```

7.2.4 Fonctions

```r
rowSums(mat)   # somme par ligne
colMeans(mat)  # moyenne par colonne
t(mat)         # transposée
dim(mat)       # dimensions (nb lignes, nb colonnes)
```

8 Data Frames

Un **data frame** est une table à deux dimensions, chaque colonne pouvant avoir un type différent (numérique, caractère, logique, etc.). C'est la structure la plus utilisée pour manipuler les jeux de données en R.

8.1 Syntaxe

`df <- data.frame(colonne1 = ..., colonne2 = ..., ...)`

Exemple :

```r
df <- data.frame(
  nom = c("Alice", "Bob", "Chloe"),    # colonne texte
  age = c(25, 30, 22),                 # colonne numérique
  etudiant = c(TRUE, FALSE, TRUE)      # colonne logique
)
```

8.2 Opérations sur les data frames

8.2.1 Accès aux éléments

```r
df$age              # accès à la colonne "age"
df[1, ]             # sélection de la 1ère ligne
df[, "etudiant"]    # accès à la colonne "etudiant"
```

8.2.2 Modification du contenu

```r
df$age[1] <- 26          # modifie la 1ère valeur de la colonne "age"
df$etudiant[3] <- FALSE  # change la 3ème valeur de la colonne "etudiant"
```

8.2.3 Ajout ou suppression de colonnes / lignes

```r
df$ville <- c("Paris", "Lyon", "Caen")        # ajoute une colonne "ville"
df[4, ] <- list("David", 28, TRUE, "Nice")    # ajoute une 4ème ligne complète
df$etudiant <- NULL                           # supprime la colonne "etudiant"
```

8.2.4 Nommer les lignes et colonnes

```r
colnames(df) <- c("Nom", "ge", "Étudiant", "Ville")  # renomme les colonnes
rownames(df) <- c("A", "B", "C", "D")                # renomme les lignes
```

9 Conditions, boucles et fonctions

Dans cette partie, nous allons découvrir les bases de la logique en programmation avec R : tester des conditions, répéter des actions, et créer des fonctions réutilisables.

Ce sont des outils essentiels pour rendre vos analyses automatiques, puissantes et intelligentes.

9.1 Structure conditionnelle : `if`

Les instructions conditionnelles permettent d'exécuter du code seulement si une condition est vraie.

Syntaxe :

```r
if (condition) {
  # instructions si vrai
} else {
```

```
  # instructions si faux
}
```

Exemples :

```
x <- 10
if (x > 5) {
  print("x est supérieur à 5")          # cas où x est > 5
} else {
  print("x est inférieur ou égal à 5")  # sinon
}
```

Avec else if :

```
x <- 7
if (x < 5) {
  print("Petit")    # si x est inférieur à 5
} else if (x == 5) {
  print("Moyen")    # si x est égal à 5
} else {
  print("Grand")    # sinon (x > 5)
}
```

9.2 La boucle `for`

La boucle `for` est utilisée pour répéter une instruction un certain nombre de fois, pour chaque élément d'une séquence.

Syntaxe :

```
for (variable in séquence) {
  # instructions à répéter
}
```

Exemples :

```
for (i in 1:5) {
  print(i^2)   # affiche le carré de i
}

for (jour in c("lundi", "mardi", "mercredi")) {
  print(paste("Aujourd'hui, c'est", jour))}  # affiche le jour
```

9.3 La boucle `while`

La boucle `while` répète une instruction tant qu'une condition reste vraie.

Syntaxe :

```
while (condition) {
  # instructions à répéter
}
```

Exemple :

```
x <- 1
while (x < 10) {
  print(x)     # affiche la valeur de x
  x <- x + 2   # incrémente x de 2
}
```

9.4 La boucle `repeat`

La boucle `repeat` répète un bloc d'instructions indéfiniment, jusqu'à rencontrer une instruction `break`.

Syntaxe :

```
repeat {
  # instructions
  if (condition) break
}
```

Exemple :

```
x <- 1
repeat {
  print(x)         # affiche la valeur de x
  x <- x + 1       # incrémente x
  if (x > 5) break # stoppe la boucle si x > 5
}
```

9.5 Créer ses propres fonctions

Une fonction est un bloc de code qui peut être réutilisé plusieurs fois. Elle prend des paramètres en entrée et retourne un résultat.

Syntaxe :

```
nom_fonction <- function(argument1, argument2, ...) {
  # instructions
  return(résultat)
}
```

Exemple simple :

```
somme_carre <- function(x) {
  return(x^2 + x)   # renvoie x**2 + x
}
somme_carre(3)      # appel de la fonction -> 12
```

10 Fonctions pour la manipulation de données en R

Ce chapitre regroupe les fonctions incontournables pour explorer, transformer, trier, résumer ou nettoyer des données en R et les fonctions applicables aux vecteurs.

10.1 1. Fonctions de base sur les objets R

```
#vecteurs
v = c(10, 20, 30, 40, 50)
sum(v)                          # la somme de toutes les valeurs
mean(v)                         # la moyenne des valeurs
min(v)                          # le minimum des valeurs
max(v)                          # le maximum des valeurs
length(v)                       # le nombre de valeur
sort(v)                         # trier par croissant
rev(v)                          # reverser la liste
range(v)                        # retourner le minimum et le maximum
unique(v)                       # les vaeurs uniques
duplicated(v)                   # TRUE pour les valeurs duppliquées
which(v > 30)                   # retourner les valeurs supérieurs à 30
any(v > 60)                     # est ce qu'il y'a au moins un vrai
all(v > 5)                      # est ce que c'est vrai pour toutes les valeurs
cumsum(v)                       # somme cumulée des valeurs
cumprod(v)                      # le produit cumulé des valeurs
diff(v)                         # la difference entre les valeurs
rep(v, times = 3)               # Répéter v trois fois
seq(1, 10, by = 2)              # Séquence
which.max(v)                    # Position du max
which(v > 5)                    # Positions vérifiant une condition
```

```r
gl(2, 5)                        # Générer un facteur
replicate(3, mean(rnorm(10))) # Répétitions
append(vec, 100, after = 2)   # Ajouter élément

# dataframe
head(df)                        # Premières lignes
tail(df)                        # Dernières lignes
dim(df)                         # Dimensions (lignes, colonnes)
nrow(df)                        # Nombre de lignes
ncol(df)                        # Nombre de colonnes
length(vec)                     # Longueur dun vecteur
names(df)                       # Noms des colonnes
colnames(df)                    # Colonnes
rownames(df)                    # Lignes
class(df)                       # Classe de lobjet
str(df)                         # Structure (type et aperçu)
summary(df)                     # Résumé statistique
```

10.2 2. Sélection et extraction

```r
df$colonne                  # Extraire une colonne
df[1, ]                     # Extraire le première ligne
df[, 2]                     # Extraire la deuxième colonne
df[1:3, c("Age", "Sexe")]  # Sous-table
subset(df, Age > 25)        # Filtrer les lignes
```

10.3 3. Fusion

```r
cbind(v1, v2)                # Coller des vecteurs en colonnes
rbind(v1, v2)                # Coller des vecteurs en lignes
merge(df1, df2, by = "ID")   # Jointure par colonne
```

10.4 4. Valeurs manquantes

```r
is.na(x)                      # vérifier, cellule par cellule, si la valeur NA
complete.cases(df)            # repérer les lignes complètes (sans NA)
anyNA(df)                     # vérifier existe au moins un NA
df = df[complete.cases(df),]  # garder uniquement les lignes complète
```

10.5 5. Application de fonctions (matrice, dataframe)

```r
apply(mat, 1, mean)          # Moyenne par ligne
apply(mat, 2, sd)            # Écart-type par colonne
lapply(df, class)            # Classe de chaque colonne
sapply(df, mean)             # Moyenne par colonne
vapply(df, is.numeric, TRUE) # type numerique?
tapply(df$Age, df$Sexe, mean)# Moyenne d'age par sexe

by(df, df$Sexe, summary)     # Résumé par groupe
```

10.6 6. Transformation et création

```r
transform(df, Age2 = Age + 10)    # Nouvelle colonne
within(df, { Note = Age * 2 })    # Modification interne
cut(df$Age, breaks = 3)           # Discrétiser
rank(c(10, 5, 3))                 # Rang
```

10.7 7. Réorganisation

```r
stack(df)                 # Transforme ton data frame en format long (values + ind)
unstack(stack(df))        # rend le même df de départ
split(df$Age, df$Sexe)    # Découper par groupe
```

10.8 8. Fonctions utiles de `dplyr`

```r
library(dplyr)

select(df, Age, Sexe)              # Sélection de colonnes
filter(df, Age > 20)               # Filtrage de lignes
mutate(df, Age2 = Age + 2)         # Nouvelle colonne
arrange(df, desc(Age))             # Tri décroissant
rename(df, NouveauNom = Age)       # Renommer une colonne
group_by(df, Sexe)                 # Grouper
distinct(df)                       # Supprimer doublons
count(df, Sexe)                    # Compter par groupe
relocate(df, Sexe, .before = Age) # Réorganiser
```

11 Exemple pratique

Voici un exemple de manipulation de données.

```r
# --- 0. Création dun data frame ---
employes <- data.frame(
  Nom         = c("Alice", "Bob", "Claire", "David", "Eva", "François"),
  Age         = c(25, 34, 28, NA, 45, 28),
  Sexe        = c("F", "H", "F", "H", "F", "H"),
  Salaire     = c(2000, 2500, 2200, 2700, NA, 2500),
  Département = c("RH", "IT", "IT", "Comptabilité", "RH", "IT")
)

# --- 1. Exploration de base ---
head(employes)                                  # 6 premières lignes
str(employes)                                   # structure du data frame
summary(employes)                               # résumé statistique
names(employes)                                 # noms des colonnes
nrow(employes)                                  # nombre de lignes
ncol(employes)                                  # nombre de colonnes

# --- 2. Sélection & extraction ---
employes$Nom                                    # extraire une colonne
employes[1:3, c("Nom", "Salaire")]              # lignes 1-3 et colonnes choisies
subset(employes, Age > 30)                      # employés > 30 ans
employes[is.na(employes$Age), ]                 # lignes avec Age manquant

# --- 3. Nettoyage ---
employes <- employes[complete.cases(employes), ]   # supprime les NA
unique(employes$Age)                               # âges distincts
duplicated(employes)                               # détecte les doublons

# --- 4. Fusion de data frames ---
infos_suppl <- data.frame(
  Nom        = c("Alice", "Bob", "Claire"),
  Anciennete = c(3, 5, 2)
)
employes <- merge(employes, infos_suppl, by = "Nom", all.x = TRUE) #fusion

# --- 5. Nouvelles colonnes ---
employes$Prime <- employes$Salaire * 0.1                  # prime = 10% du salaire
employes <- transform(employes, Bonus = Salaire * 0.05)    # autre méthode

# --- 6. Fonctions apply ---
apply(employes[, c("Salaire", "Prime", "Bonus")], 2,mean) # moyenne par colonne

# --- 7. Statistiques par groupes ---
```

```r
tapply(employes$Salaire, employes$Sexe, mean)           # moyenne par sexe
aggregate(Salaire ~ Département, data = employes, mean) # moyenne par département
by(employes$Salaire, employes$Département, summary)     # résumé par département

# --- 8. Tri des données ---
employes[order(employes$Age), ]                  # tri croissant par âge
employes[order(-employes$Salaire), ]             # tri décroissant par salaire

# --- 9. Manipulations avec dplyr ---
library(dplyr)
employes %>%
  select(Nom, Salaire) %>%                       # sélectionner colonnes
  filter(Salaire > 2300) %>%                     # filtrer
  mutate(Net = Salaire - 150) %>%                # nouvelle colonne Net
  arrange(desc(Net)) %>%                         # tri décroissant
  group_by(Sexe) %>%                             # regroupement par sexe
  summarise(Salaire_moyen = mean(Salaire), Max = max(Salaire)) # stats par sexe

# --- 10. Pipe avancé ---
employes %>%
  group_by(Département) %>%                      # regroupement par dép.
  filter(Salaire > 2000) %>%                     # filtrer
  summarise(Salaire_moyen = mean(Salaire))       # moyenne par groupe

# --- 11. Fonctions complémentaires ---
which(employes$Salaire == max(employes$Salaire) # indice du salaire max
rank(employes$Salaire)                          # rang des salaires
seq(18, 60, by = 6)                             # séquence
rep(employes$Nom[1], times = 3)                 # répète un nom
append(employes$Nom, "Zara", after = 2)         # insère "Zara"
```

12 Visualisation

R propose une grande variété de fonctions pour représenter visuellement les données. Ces fonctions peuvent être regroupées selon leur origine (base R ou packages) et leur usage (statistique, multivarié, interactif, etc.).

12.1 Graphiques statistiques classiques

- `plot()` : nuages de points, courbes...
- `hist()` : histogramme pour la distribution des valeurs.
- `boxplot()` : visualisation des quartiles et des valeurs extrêmes.

— `barplot()` : diagramme en barres.
— `pie()` : camembert.
— `dotchart()`, `stripchart()` : valeur individuelles en ligne ou en trait.
— `stem()` : graphique tige et feuille.
— `density()` : estimation de densité.
— `rug()` : ajoute des tics pour les observations.
— `pairs()` : nuages de points pour toutes les paires de variables.
— `coplot()` : nuages conditionnels.
— `mosaicplot()`, `assocplot()`, `fourfoldplot()` : pour les tableaux de contingence.

12.2 Graphiques multivariés et 3D

— `image()`, `heatmap()`, `filled.contour()` : représentation de matrices de valeurs.
— `contour()` : courbes de niveaux.
— `persp()` : représentation 3D.
— `symbols()` : graphiques à bulles, cercles proportionnels.

12.3 Visualisation avec `ggplot2`

— `ggplot()` : initialise le graphe.
— `aes()` : définit les esthétiques (x, y, couleur...).
— `geom_point()`, `geom_line()`, `geom_bar()`, `geom_col()`, `geom_boxplot()`, `geom_histogram()`, `geom_violin()` : types de graphes.
— `geom_density()`, `geom_rug()`, `geom_smooth()`, `geom_text()` : couches supplémentaires.
— `facet_wrap()`, `facet_grid()` : créer plusieurs graphes selon une variable.
— `labs()`, `theme()`, `coord_flip()`, `coord_polar()` : personnalisation.

12.4 Packages spécialisés pour la visualisation

— `lattice` : `xyplot()`, `bwplot()`, pour visualisation conditionnelle.
— `plotly`, `highcharter`, `echarts4r` : visualisations interactives.
— `corrplot`, `GGally`, `ggridges`, `ggraph` : pour corrélations, graphes, densités.
— `RColorBrewer`, `viridis` : palettes de couleurs.
— `patchwork`, `cowplot`, `gridExtra` : mise en page multiple de graphes `ggplot2`.

12.5 Personnalisation des axes

— **xaxt = "n"**, **yaxt = "n"** : Empêche l'affichage automatique des axes (X ou Y).

— **axis(side, at, labels)** : Ajoute un axe personnalisé.
  — `side` = 1 (bas), 2 (gauche), 3 (haut), 4 (droite)
  — `at` : vecteur des positions des graduations
  — `labels` : étiquettes à afficher
  — `tick` : afficher les ticks (`TRUE/FALSE`)
  — `col.axis`, `cex.axis`, `font.axis` : couleur, taille, police
— **las** : Orientation du texte des axes (`0` = parallèle, `1` = horizontal, `2` = perpendiculaire, `3` = vertical).
— **tck** : Longueur des ticks (`0` = invisibles, négatif = vers l'intérieur).
— **mgp = c(titre, label, ligne)** : Contrôle l'espacement entre titre, label et ligne des axes.

12.6 Grilles et repères

— **grid(nx, ny)** : Affiche une grille par défaut dans le fond du graphique.
  — `nx, ny` : nombre de lignes verticales et horizontales
  — `col`, `lty`, `lwd` : couleur, type, épaisseur de ligne
— **abline()** : Ajoute des lignes droites de repère :
  — `abline(h = valeur)` : ligne horizontale
  — `abline(v = valeur)` : ligne verticale
  — `abline(a, b)` : droite de régression $y = a + bx$
— **segments(x0, y0, x1, y1)** : Trace un segment.
— **arrows(x0, y0, x1, y1)** : Trace une flèche.

12.7 Légendes et textes

— **legend(x, y, legend)** : Ajoute une légende dans le graphique.
  — `x, y` : position ou mots-clés : `"topright"`, `"bottomleft"`, etc.
  — `legend` : vecteur des intitulés
  — `pch`, `lty`, `col`, `cex` : symboles et apparence
  — `bty = "n"` : supprime la boîte
— **text(x, y, labels)** : Affiche un texte à la position donnée.
— **mtext()** : Texte dans la marge.
— **title()** : Ajoute ou modifie le titre ou les axes.

12.8 Fenêtres graphiques multiples

— **par(mfrow = c(n, m))** : Affiche plusieurs graphiques dans une fenêtre, ligne par ligne.
— **par(mfcol = c(n, m))** : Affiche plusieurs graphiques colonne par colonne.

— **layout(mat)** : Organisation personnalisée des graphiques.
— **par(mar = c(b, l, t, r))** : Marges internes (bottom, left, top, right).
— **par(oma = c(b, l, t, r))** : Marges externes.
— **par(bg = "color")** : Couleur de fond de la fenêtre graphique.

13 Exemple pratique

13.1 La fonction `plot()`

La fonction `plot()` est la fonction graphique générique de R. Elle permet de tracer un nuage de points, une courbe ou un graphique personnalisé à partir de deux vecteurs ou d'un objet graphique.

Syntaxe

```
plot(x, y = NULL, type = "p", ...)
```

— `x, y` : vecteurs numériques ou objets graphiques.
— `type` : type de tracé :
  — `"p"` : points (valeur par défaut)
  — `"l"` : lignes
  — `"b"` : points + lignes
  — `"o"` : points + lignes superposés
  — `"h"` : segments verticaux (type histogramme)
  — `"s"`, `"S"` : marches (lignes en escalier)
  — `"n"` : ne trace rien (utile pour personnaliser à la main)
— `xlab`, `ylab` : noms des axes.
— `xlim`, `ylim` : limites des axes.
— `main`, `sub` : titre principal et sous-titre.
— `col`, `pch`, `lty`, `lwd` : couleur, symbole, type et épaisseur de ligne.
— `cex`, `cex.axis`, `cex.lab`, `cex.main` : tailles des éléments.
— `axes = FALSE`, `xaxt = "n"`, `yaxt = "n"` : suppression des axes.
— `asp` : rapport d'aspect (utile pour les cercles).
— `frame.plot = FALSE`, `bty = "n"` : désactivation de la boîte autour du graphe.

Des fonctions complémentaires permettent d'enrichir le graphique après un appel à `plot()` :
— `lines()`, `points()` : ajouter des courbes ou points supplémentaires.
— `abline()`, `segments()`, `arrows()` : ajouter des repères ou flèches.
— `text()`, `legend()`, `mtext()`, `title()` : annotation, titres et légendes.
— `grid()`, `box()`, `axis()` : personnalisation des axes et du fond.
— `identify()`, `locator()` : interaction avec la souris.

Exemple : Visualisation avec `plot()`

```r
# Génère une suite de 10 valeurs de 1 à 10
x <- 1:10

# Génère 10 valeurs aléatoires selon une loi normale
y <- rnorm(10)

# Simple tracé
plot(x, y)

# Tracé avec personnalisation avancée
plot(x, y,
     type = "b",
     col = "darkgreen",
     pch = 17,
     lwd = 2,
     main = "Évolution aléatoire",
     xlab = "Indice",
     ylab = "Valeur",
     xlim = c(0, 12),
     ylim = c(-3, 3),
     cex = 1.5,
     cex.main = 1.2,
     las = 1,
     bty = "l",
     xaxt = "n")

# Axe X personnalisé
axis(1,
     at = 1:10,
     labels = paste("V", 1:10),
     col.axis = "blue",
     cex.axis = 0.8)

# Grille et ligne horizontale
grid()
abline(h = 0, col = "gray", lty = 2)

# Légende
legend("topright",
       legend = "Série",
       col = "darkgreen",
       pch = 17,
       bty = "n")
```

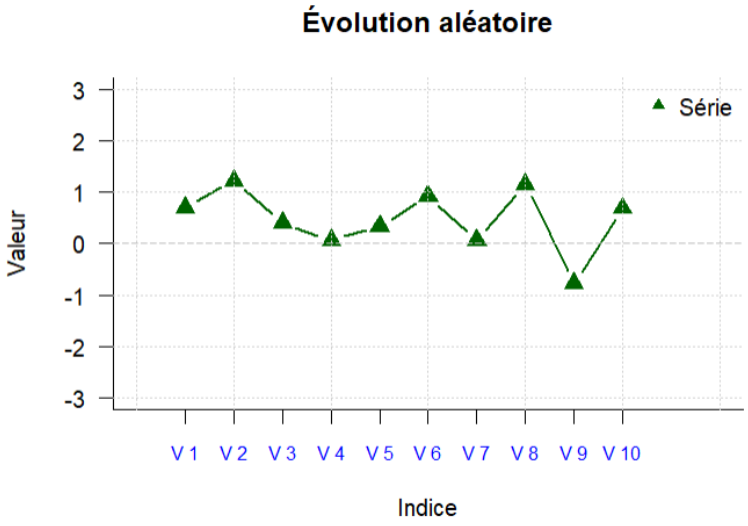


FIGURE 2 – Visualisation enrichie avec `plot()`

Ce graphique combine des points reliés, une personnalisation des axes, une grille, un repère horizontal, et une légende pour une visualisation claire et structurée.

13.2 La fonction `lines()`

La fonction `lines()` permet d'ajouter des lignes sur un graphique existant, souvent pour superposer plusieurs séries de données ou enrichir la courbe d'un tracé.

Syntaxe

```
lines(x, y, ...)
```

- `x`, `y` : vecteurs numériques représentant les coordonnées des points à relier.
- `col` : couleur de la ligne.
- `lty` : type de ligne (1 = continue, 2 = pointillée, etc.).
- `lwd` : épaisseur de la ligne.
- `type` : permet aussi d'ajouter des symboles avec `"b"`, `"o"`, etc.

Important : `lines()` ne crée pas de graphique seule — elle doit suivre un `plot()`.

Exemple complet

```
1 x <- 1:10
```

```r
y <- rnorm(10)

# Génère une série secondaire à superposer
y2 <- rnorm(10, mean = 0.5)

# Tracé du graphique principal
plot(x, y,
     type = "b",
     col = "darkgreen",
     pch = 17,
     lwd = 2,
     main = "Comparaison de deux séries",
     xlab = "Indice",
     ylab = "Valeur",
     xlim = c(0, 12),
     ylim = range(c(y, y2)),
     cex = 1.5,
     cex.main = 1.2,
     las = 1,
     bty = "l",
     xaxt = "n")

# Axe X personnalisé
axis(1, at = 1:10, labels = paste("V", 1:10), col.axis = "blue", cex.axis = 0.8)

# Ajoute une grille et un repère
grid()
abline(h = 0, col = "gray", lty = 2)

# Ajout de la deuxième série avec lines()
lines(x, y2,              # Coordonnées des points à relier
      type = "b",         # "b" = both  points + lignes
      col = "orange",     # Couleur orange pour différencier la série
      pch = 19,           # Forme du point : rond plein
      lty = 3,            # Type de ligne : pointillée
      lwd = 2)            # Épaisseur de la ligne

# Ajoute une légende pour les deux séries
legend("topright",
       legend = c("Série 1", "Série 2"),
       col = c("darkgreen", "orange"),
       pch = c(17, 19),
       lty = c(1, 3),
       bty = "n")
\end{
```

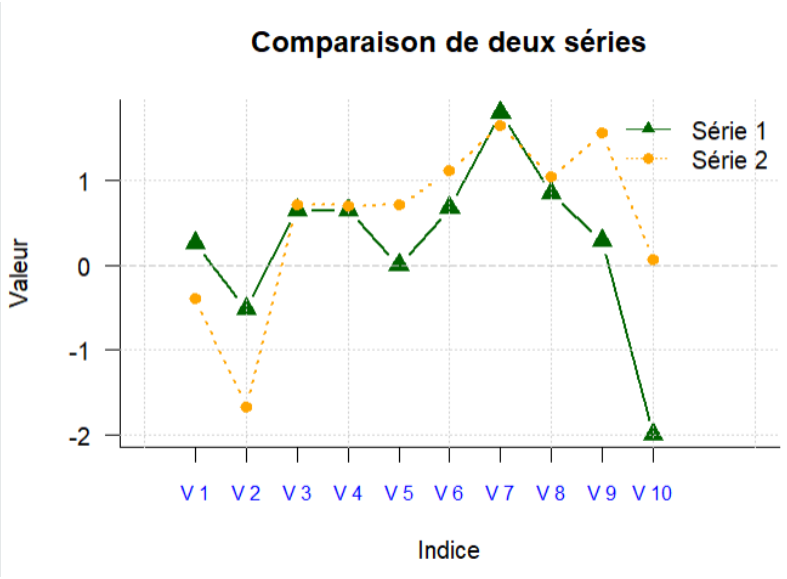


FIGURE 3 – Superposition de deux séries avec `plot()` et `lines()`

13.3 Ajout de repères et annotations graphiques

En plus de tracer des courbes, **R** permet d'ajouter manuellement des repères, des annotations, des flèches ou des points pour enrichir la lisibilité d'un graphique. Voici les principales fonctions utilisées à cet effet :

- `abline()` : ajoute une droite (horizontale, verticale ou oblique).
- `points()` : ajoute manuellement un ou plusieurs points.
- `segments()` : trace un segment entre deux points.
- `arrows()` : ajoute une flèche entre deux coordonnées (utile pour pointer ou annoter).

Exemple combiné

```
# Génération de données
x <- 1:10
y <- rnorm(10)

# Tracé principal avec courbe et points
plot(x, y,
     type = "b",              # both = points + lignes
     col = "blue",            # couleur bleue
     pch = 16,                # points ronds pleins
     main = "Exemples de repères et annotations",
     xlab = "Index",
     ylab = "Valeur")

```

```r
# Ajout d'une droite horizontale
abline(h = 0,
       col = "gray",          # couleur grise
       lty = 2)               # ligne pointillée

# Ajout dun point spécifique
points(5, 1.5,
       pch = 17,              # triangle plein
       col = "darkgreen",
       cex = 1.5)

# Ajout dun segment
segments(x0 = 2, y0 = -1, x1 = 8, y1 = 1.5,
         col = "orange",
         lwd = 2)

# Ajout d'une flèche
arrows(x0 = 5, y0 = -1, x1 = 5, y1 = 1.5,
       col = "brown",
       length = 0.1)

# Légende
legend("topright",
       legend = c("Série (plot)",
                  "Repère horizontal (abline)",
                  "Point manuel (points)",
                  "Segment (segments)",
                  "Flèche (arrows)"),
       col = c("blue", "gray", "darkgreen", "orange", "brown"),
       pch = c(16, NA, 17, NA, NA),
       lty = c(1, 2, NA, 1, 1),
       bty = "n")
```

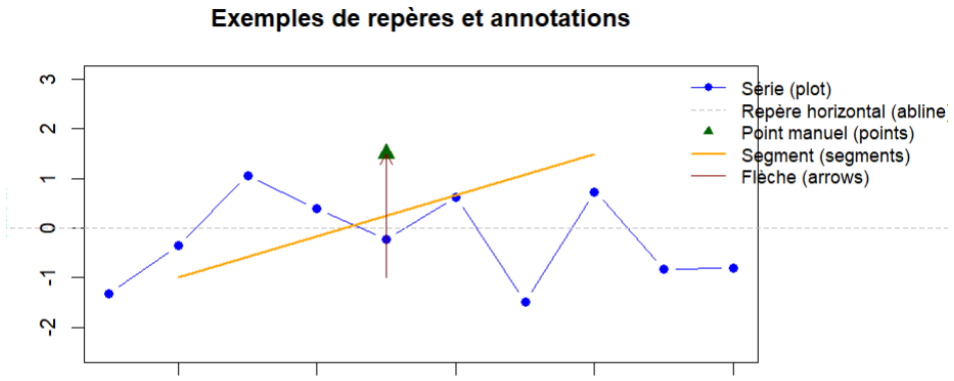


FIGURE 4 – Superposition d'éléments graphiques avec `abline()`, `points()`, `segments()` et `arrows()`

13.4 Histogramme avec `hist()`

L'histogramme permet de représenter la distribution d'une variable numérique continue en la divisant en intervalles (ou "classes") et en comptant le nombre d'observations dans chaque intervalle.

Syntaxe

```
hist(x,
     breaks = "Sturges",
     col = NULL,
     main = NULL,
     xlab = NULL,
     ylab = NULL,
     freq = TRUE,
     ...)
```

- `x` : vecteur numérique (variable à représenter)
- `breaks` : méthode ou nombre de classes (ex. `"Sturges"`, `10`, `seq(0, 100, 10)`)
- `col` : couleur des barres
- `main` : titre du graphique
- `xlab`, `ylab` : titres des axes
- `freq = TRUE` : affiche les effectifs (ou `FALSE` pour les densités)

Exemple : Visualisation avec `hist()`

```r
data(airquality)

# Histogramme simple
hist(airquality$Ozone)

# Histogramme personnalisé
hist(airquality$Ozone,
     breaks = 10,
     col = "skyblue",
     border = "white",
     main = "Distribution de l'Ozone",
     xlab = "Ozone (ppb)",
     ylab = "Fréquence",
     freq = T,
     xlim = c(0,150),
     ylim = c(0,40))
```

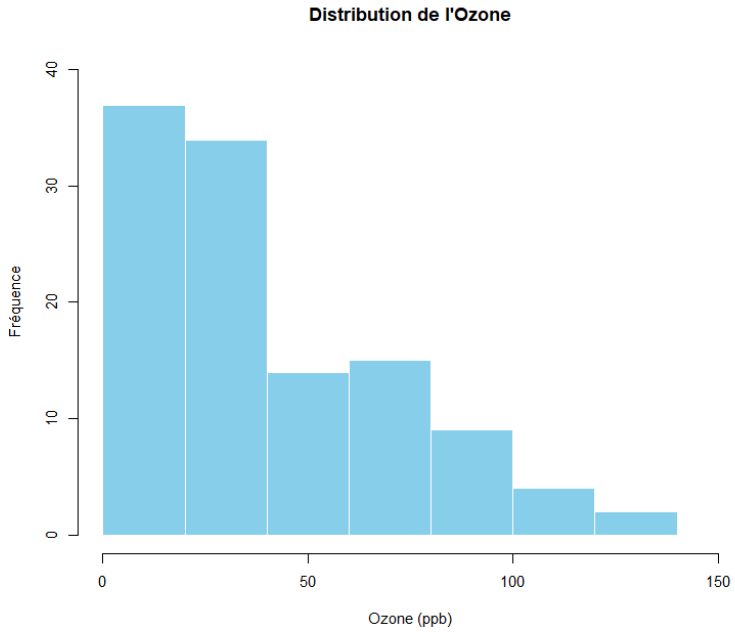


FIGURE 5 – Histogramme de la concentration d'ozone dans l'air

Cet histogramme met en évidence la distribution de la variable `Ozone`, avec un découpage en 10 classes et des barres colorées pour une meilleure lisibilité.

13.5 Diagramme en barres avec `barplot()`

Le diagramme en barres permet de représenter des **valeurs catégorielles** (ou des fréquences) sous forme de barres verticales ou horizontales. Il est souvent utilisé pour comparer les effectifs ou les moyennes entre plusieurs groupes.

Syntaxe

```
barplot(height,
        names.arg = NULL,
        horiz = FALSE,
        col = NULL,
        main = NULL,
        xlab = NULL,
        ylab = NULL,
        ...)
```

- `height` : vecteur numérique ou tableau (valeurs à représenter)
- `names.arg` : noms des catégories (axe X ou Y selon l'orientation)
- `horiz` : si `TRUE`, barres horizontales
- `col` : couleur des barres
- `main`, `xlab`, `ylab` : titres

Exemple : Visualisation avec `barplot()`

```r
genre <- c("Homme", "Femme", "Autre")
effectifs <- c(45, 52, 3)
barplot(effectifs,
        names.arg = genre,
        col = "steelblue",
        main = "Répartition par genre",
        ylab = "Effectifs",
        xlab = "Genre")
```

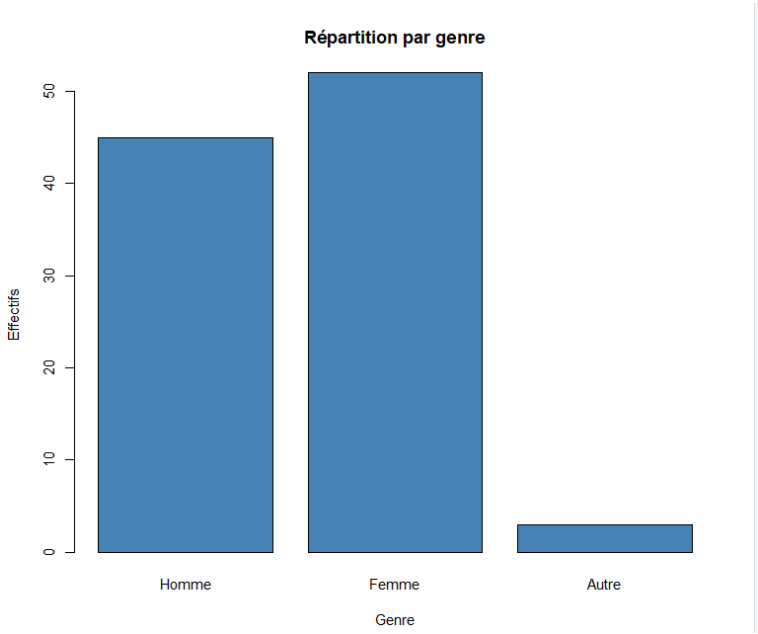


FIGURE 6 – Diagramme en barres représentant la répartition par genre

Ce graphique permet une comparaison rapide des effectifs pour chaque modalité de la variable `genre`. L'utilisation de couleurs rend la visualisation plus intuitive.

13.6 Diagramme en boîte avec `boxplot()`

Le **boxplot**, ou diagramme en boîte, est un graphique qui permet de résumer la distribution d'une variable quantitative : médiane, quartiles, minimum, maximum et valeurs aberrantes.

Syntaxe

```
boxplot(x,
        horizontal = FALSE,
        col = NULL,
        main = NULL,
        xlab = NULL,
        ylab = NULL,
        ...)
```

- `x` : vecteur numérique ou liste / formule (ex : `note ~ classe`)
- `horizontal` : si `TRUE`, boîte horizontale
- `col` : couleur
- `main`, `xlab`, `ylab` : titres

Exemple : Visualisation avec `boxplot()`

```r

note <- c(12, 15, 13, 18, 14, 9, 10, 16, 12, 20, 8, 17)
classe <- c(rep("A", 6), rep("B", 6))

# Boxplot comparatif entre deux classes
boxplot(note ~ classe,
        col = c("tomato", "skyblue"),
        main = "Distribution des notes par classe",
        xlab = "Classe",
```

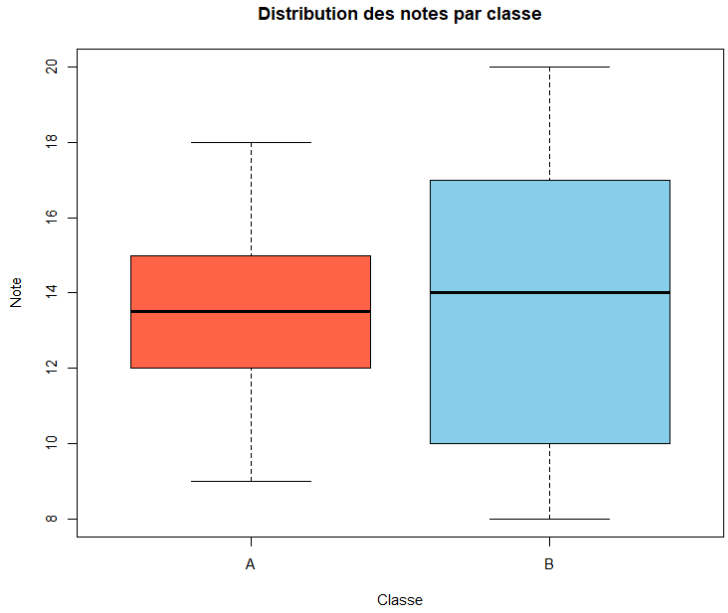


FIGURE 7 – Diagramme en boîte comparant la distribution des notes par classe

Ce graphique montre la répartition des notes dans chaque classe, en mettant en évidence la médiane, l'étendue interquartile, les extrêmes et les éventuels outliers.

13.7 Diagramme de dispersion avec `stripchart()`

La fonction `stripchart()` permet de visualiser la distribution d'une variable numérique à l'aide d'un nuage de points unidimensionnel. C'est une alternative compacte à l'histogramme ou au boxplot.

Syntaxe

```
stripchart(x, method = "overplot", vertical = FALSE, ...)
```

- `x` : vecteur numérique ou liste/groupes.
- `method` : façon de gérer les superpositions :
  - `"overplot"` (valeur par défaut) : les points peuvent se superposer.
  - `"jitter"` : ajout d'un bruit aléatoire pour éviter la superposition.
  - `"stack"` : empilement vertical ou horizontal.
- `vertical` : TRUE pour un affichage vertical, FALSE pour horizontal.
- `group.names` : noms des groupes (si liste ou facteur).

Exemple simple avec bruit (jitter)

```
    # Tracé horizontal avec jitter
stripchart(x,
           method = "jitter",     # évite les superpositions
           pch = 19,              # point plein
           col = "darkblue",     # couleur
           main = "Diagramme de dispersion",
           xlab = "Valeur")
```

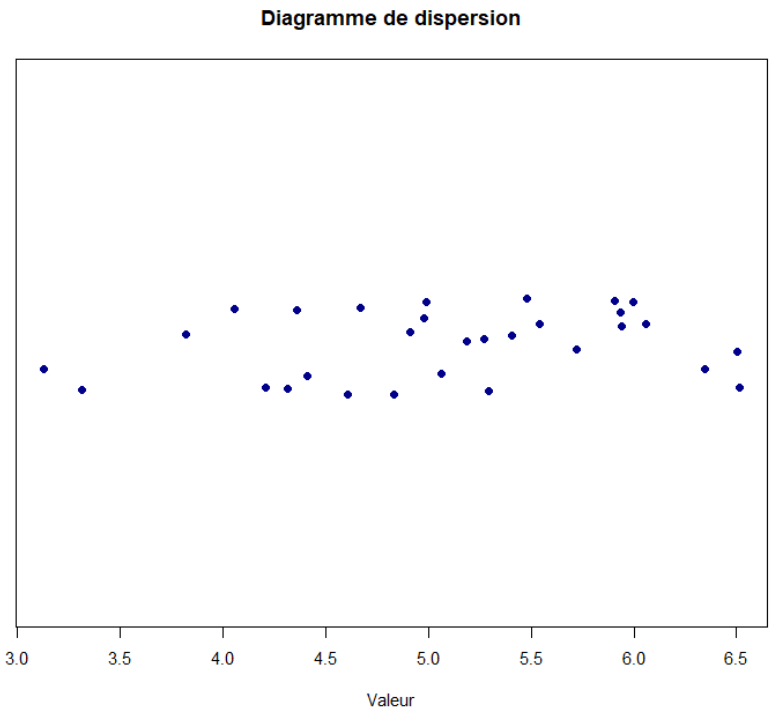


FIGURE 8 – Diagramme de dispersion horizontal avec `stripchart()` et bruit aléatoire

Ce type de graphique est très utile pour observer rapidement la densité locale et les valeurs aberrantes dans un petit échantillon.

13.8 Autres graphiques

`dotchart()`

```r
valeurs <- c(23, 17, 35, 29, 12)
noms <- c("Produit A", "Produit B", "Produit C", "Produit D", "Produit E")

# Dotchart simple
dotchart(valeurs,
         labels = noms,
         main = "Diagramme en points",
         xlab = "Quantité vendue",
         col = "darkblue",
         pch = 19)
```

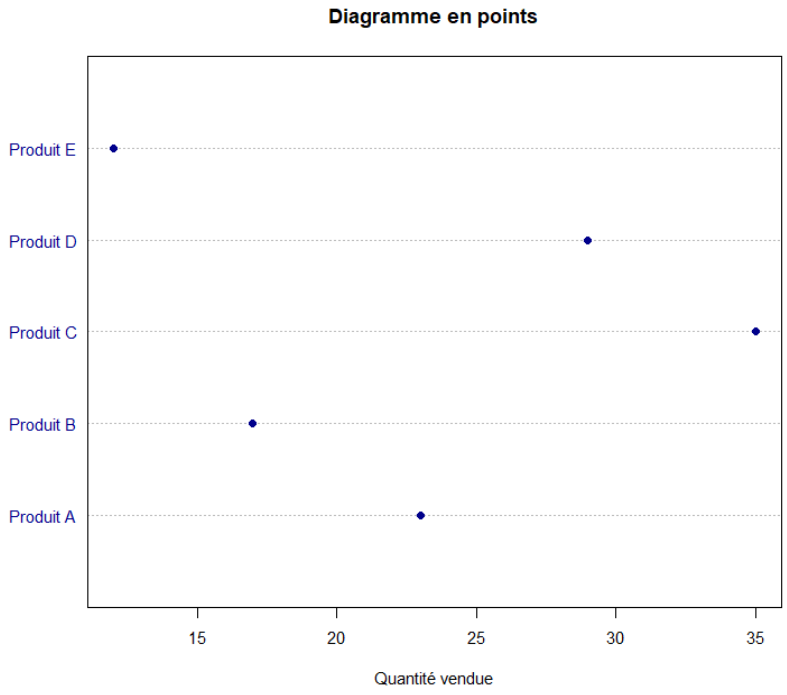


FIGURE 9 – Diagramme en points avec `dotchart()`

Matrice de scatterplots — `pairs()`

```r
# Exemple sur le jeu de données iris
data(iris)

# Affiche la matrice de scatterplots pour les variables numériques
pairs(iris[, 1:4],
      main = "Matrice de scatterplots - iris",
      pch = 21,
      bg = c("red", "green3", "blue")[iris$Species])
```

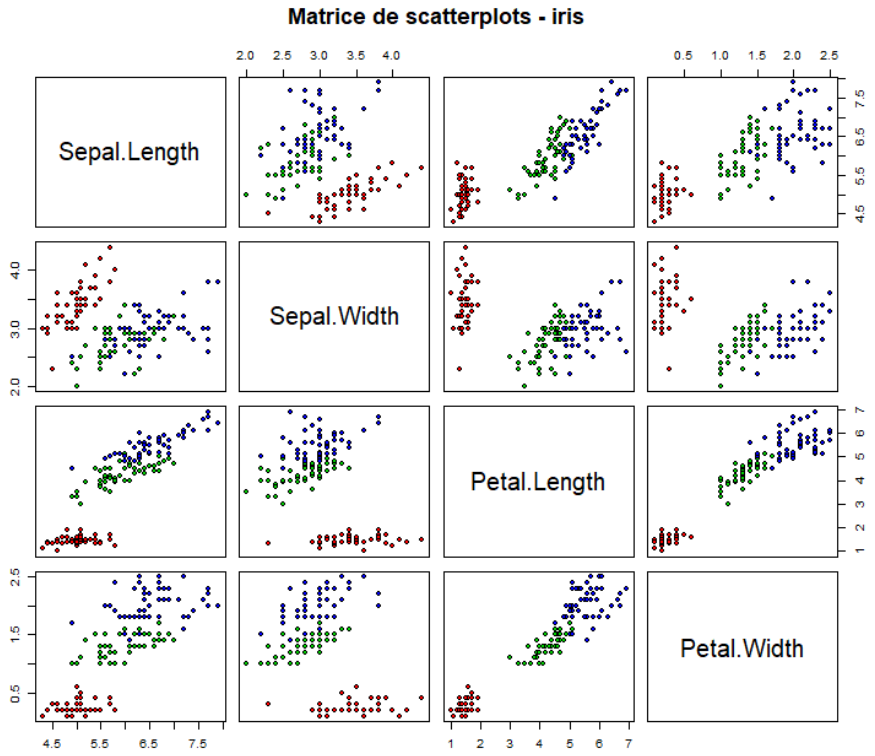


FIGURE 10 – Matrice de scatterplots avec `pairs()` sur le jeu iris

Tracés multiples — `matplot()`

```r
# Création de données
x <- 1:10
y <- cbind(sin(x), cos(x), sin(x / 2))

# Tracés multiples
matplot(x, y,
        type = "l",               # type ligne
        lty = 1,                  # ligne continue
        col = c("red", "blue", "darkgreen"),
        lwd = 2,
```

```
        main = "Tracés superposés",
        xlab = "Temps",
        ylab = "Valeur")

legend("topright",
       legend = c("sin(x)", "cos(x)", "sin(x/2)"),
       col = c("red", "blue", "darkgreen"),
       lty = 1,
       bty = "n")
```

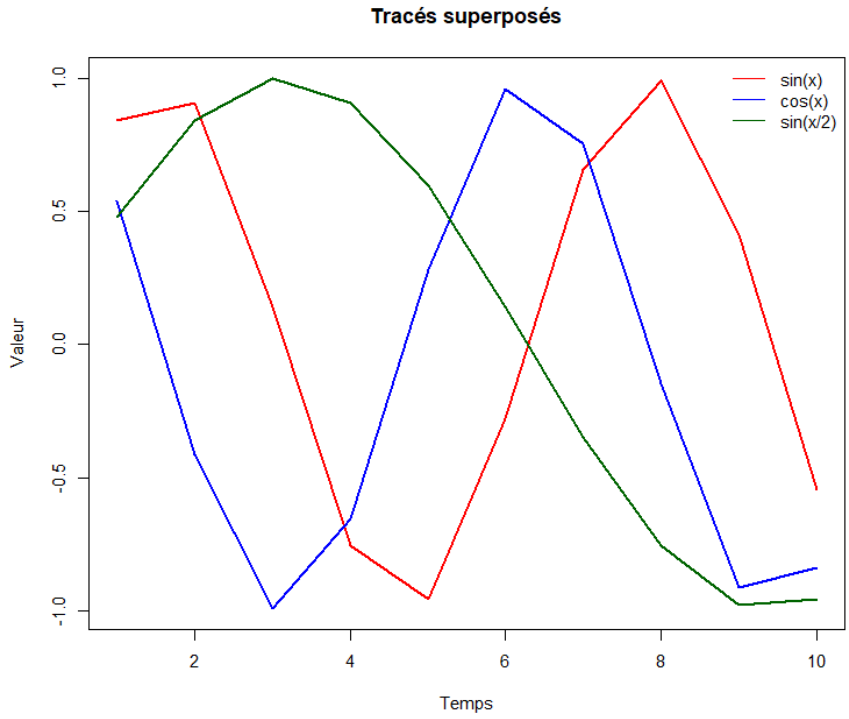


FIGURE 11 – Tracés multiples avec `matplot()`

13.9 Visualisation avancée en 2D et 3D

1. Affichage d'une image matricielle — `image()`

```
# Création d'une matrice de données
z <- outer(1:20, 1:20, function(x, y) sin(x / 3) + cos(y / 5))

# Affichage matriciel
image(z,
      main = "Visualisation matricielle",
      col = terrain.colors(20))
```

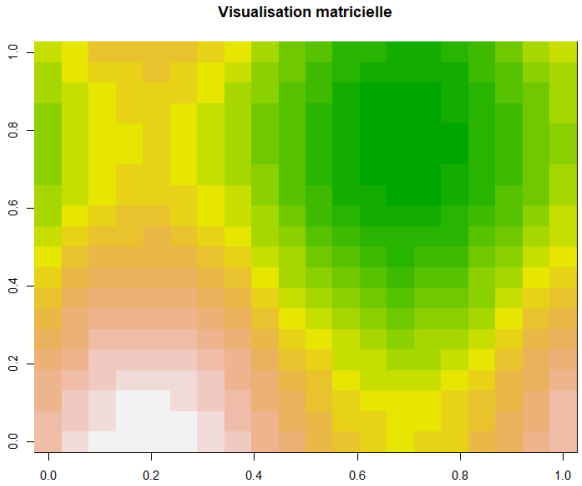


FIGURE 12 – Affichage matriciel avec `image()`

2. Courbes de niveau — `contour()`

```r
# Même matrice que précédemment
z <- outer(1:20, 1:20, function(x, y) sin(x / 3) + cos(y / 5))

# Courbes de niveau
contour(z,
        main = "Courbes de niveau",
        col = "blue")
```

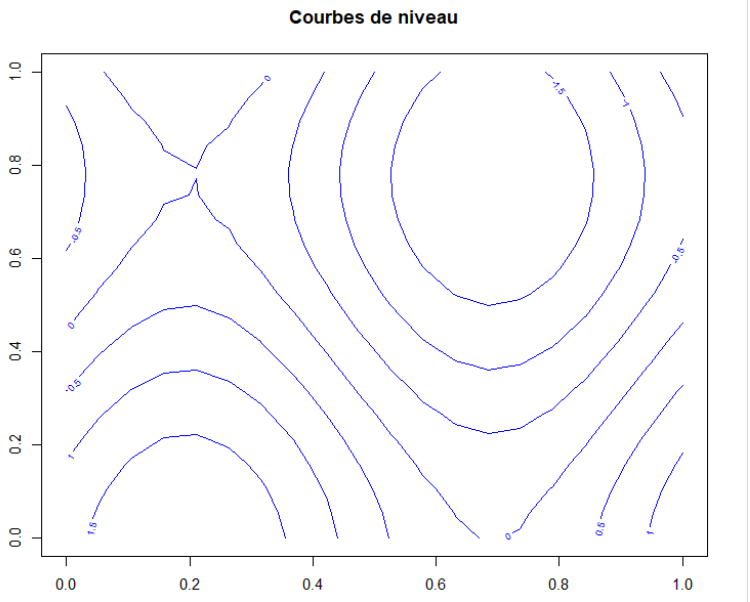


FIGURE 13 – Courbes de niveau avec `contour()`

3. Représentation 3D — `persp()`

```r
x <- seq(-10, 10, length = 30)
y <- seq(-10, 10, length = 30)
z <- outer(x, y, function(x, y) cos(sqrt(x^2 + y^2)))

# Affichage 3D
persp(x, y, z,
      theta = 30, phi = 30,
      col = "lightblue",
      border = NA,
      shade = 0.5,
      main = "Surface 3D")
```

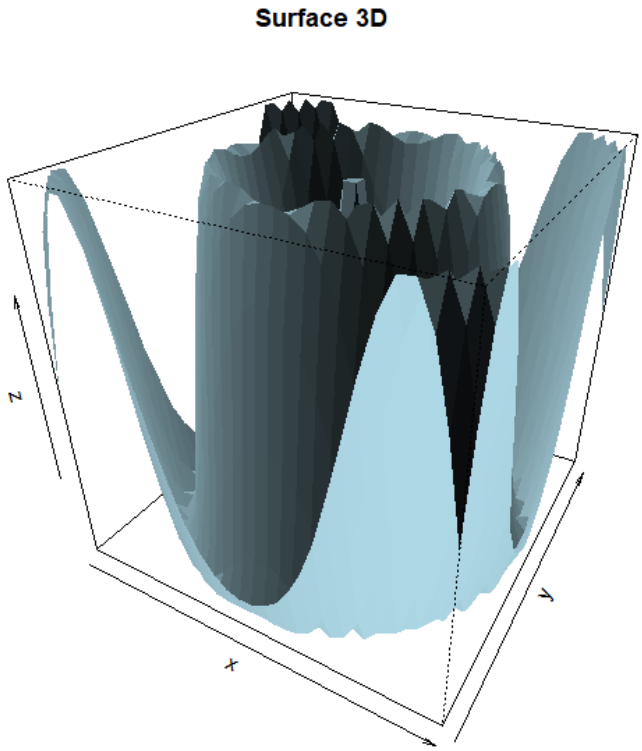


FIGURE 14 – Surface 3D avec `persp()`

14 Quelques lois usuelles :

Pour une loi L : `dL` (densité/proba), `pL` (répartition), `qL` (quantiles), `rL` (tirage).
Ex. : `dnorm(x,mean,sd)`, `pbinom(k,n,p)`, `rt(n,df)`.

14.1 Normale $\mathcal{N}(\mu, \sigma^2)$

Usage : somme de petits effets additifs, erreurs de mesure, phénomènes "centrés".
- Paramètres : moyenne μ, écart-type σ.
- $E[X] = \mu$, $\mathrm{Var}(X) = \sigma^2$.
- **Quand ?** Variable continue & symétrique ; théorème central limite ; données gaussiennes.
- R : `dnorm(x, mu, sd)`, `pnorm(x, mu, sd)`, `qnorm(p, mu, sd)`, `rnorm(n, mu, sd)`.

14.2 Student t_ν (ddl ν)

Usage : moyenne d'un petit échantillon quand σ inconnu, tests/IC avec n petit.
- Centre 0, queues plus épaisses que la normale (se rapproche de $\mathcal{N}(0,1)$ quand $\nu \to \infty$).
- **Quand ?** Statistique $T = \sqrt{n}\,\frac{\bar{X}-\mu_0}{S}$; modèles robustes aux valeurs extrêmes.
- R : `dt(x, df)`, `pt(x, df)`, `qt(p, df)`, `rt(n, df)`.

14.3 Khi-deux χ^2_ν

Usage : sommes de carrés normalisés, tests d'ajustement/indépendance.
- $\chi^2_\nu = \sum_{i=1}^{\nu} Z_i^2$ avec $Z_i \sim \mathcal{N}(0,1)$ iid.
- $E[X] = \nu$, $\mathrm{Var}(X) = 2\nu$.
- **Quand ?** Variance empirique $\sim \chi^2$; tableaux de contingence (test χ^2).
- R : `dchisq(x, df)`, `pchisq(x, df)`, `qchisq(p, df)`, `rchisq(n, df)`.

14.4 Binomiale $\mathcal{B}(n, p)$

Usage : n essais indépendants, succès/échec (Bernoulli).
- $k = 0, \ldots, n$, $\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.
- $E[X] = np$, $\mathrm{Var}(X) = np(1-p)$.
- **Quand ?** Comptage de succès (clics oui/non, défaut oui/non) avec n fixé.
- R : `dbinom(k, n, p)`, `pbinom(k, n, p)`, `qbinom(p, n, p)`, `rbinom(n, size, p)`.

14.5 Poisson $\mathcal{P}(\lambda)$

Usage : nombre d'événements rares sur un intervalle (temps/espace), indépendants.
- $k = 0, 1, \ldots$, $\Pr(X = k) = e^{-\lambda}\lambda^k/k!$.
- $E[X] = \lambda$, $\mathrm{Var}(X) = \lambda$.
- **Quand ?** Comptage sans n fixé (arrivées par minute, défauts par mètre).

— R : `dpois(k, lambda)`, `ppois(k, lambda)`, `qpois(p, lambda)`, `rpois(n, lambda)`.

14.6 Exponentielle $\mathrm{Exp}(\lambda)$

Usage : *temps d'attente* jusqu'au prochain événement de Poisson (mémoire nulle).
— $x \geq 0$, $f(x) = \lambda e^{-\lambda x}$; $E[X] = 1/\lambda$, $\mathrm{Var}(X) = 1/\lambda^2$.
— **Quand ?** Durées entre événements, fiabilité (sans vieillissement).
— R : `dexp(x, rate)`, `pexp(x, rate)`, `qexp(p, rate)`, `rexp(n, rate)`.

14.7 Uniforme $\mathcal{U}(a, b)$

Usage : ignorance totale dans un intervalle ; générateurs aléatoires de base.
— $E[X] = (a + b)/2$, $\mathrm{Var}(X) = (b - a)^2/12$.
— R : `dunif(x, min, max)`, `punif`, `qunif`, `runif`.

Choisir la loi ?

1. **Type de variable** : *discrète* (comptages) ou *continue* (mesures) ?
2. **Comptage** :
   — n **fixé**, succès/échec ⇒ **Binomiale** $\mathcal{B}(n, p)$.
   — n **non fixé**, événements rares dans un intervalle ⇒ **Poisson** $\mathcal{P}(\lambda)$.
3. **Durée jusqu'à un évènement** (processus de Poisson) ⇒ **Exponentielle**.
4. **Somme ou moyenne de mesures bruitées, symétriques ⇒ Normale**.
5. **Inférence sur la moyenne avec σ inconnu** :
   — n petit ⇒ **Student** t_{n-1} ;
   — n grand ⇒ **Normale** (approx.).
6. **Variance à partir de données normales** ⇒ χ^2.

— Densité en x_0 : `dnorm(x0, mean=mu, sd=sigma)`
— Probabilité $P(X \leq x_0)$: `pnorm(x0, mean=mu, sd=sigma)`
— Queue $P(X > x_0)$: `pnorm(x0, mean=mu, sd=sigma, lower.tail=FALSE)`
— Quantile q_p tel que $P(X \leq q_p) = p$: `qnorm(p, mean=mu, sd=sigma)`
— Simulation de n valeurs : `rnorm(n, mean=mu, sd=sigma)`

Erreurs fréquentes (à éviter)

— **Confondre variance et écart-type** dans R : `dnorm(..., sd = écart-type)`.

— Oublier **la correction de continuité** pour comparer une loi continue (normale) à une loi discrète (binomiale).
— Utiliser la **normale** quand np ou $n(1-p)$ sont trop petits (< 5) : préférer **Poisson** ou exact binomial.
— Croire qu'un **mélange de normales** est normal : *faux* (souvent bimodal).

15 Loi normale $\mathcal{N}(\mu, \sigma^2)$

15.1 Définition et paramètres

On dit que $X \sim \mathcal{N}(\mu, \sigma^2)$ si sa densité est

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\Big(- \frac{(x-\mu)^2}{2\sigma^2} \Big), \qquad x \in R.$$

Paramètres : μ (moyenne), $\sigma > 0$ (écart-type), variance $= \sigma^2$.

Faits clés.
— $Z = \dfrac{X-\mu}{\sigma} \sim \mathcal{N}(0,1)$ (standardisation).
— Environ 68% des valeurs dans $[\mu-\sigma, \mu+\sigma]$, 95% dans $[\mu-2\sigma, \mu+2\sigma]$, 99,7% dans $[\mu-3\sigma, \mu+3\sigma]$.
— **R et la normale** : `dnorm` (densité), `pnorm` (répartition), `qnorm` (quantiles), `rnorm` (simulation).

15.2 Quand l'utiliser ?

— Phénomènes continus, symétriques, bruit de mesure, somme de petits effets (TCL).
— Approximation de nombreuses statistiques (moyenne d'échantillon, grandes tailles).

15.3 Exemples :

Exemple 1 — tracer une densité. $X \sim \mathcal{N}(19, 3)$ (ici 3 est la *variance*, donc $sd = \sqrt{3}$).

```
mu <- 19; sd <- sqrt(3)
x  <- seq(mu-4*sd, mu+4*sd, length.out = 400)
plot(x, dnorm(x, mean=mu, sd=sd), type="l", lwd=2,
     main="Densité N(19, 3)", xlab="x", ylab="densité")
```

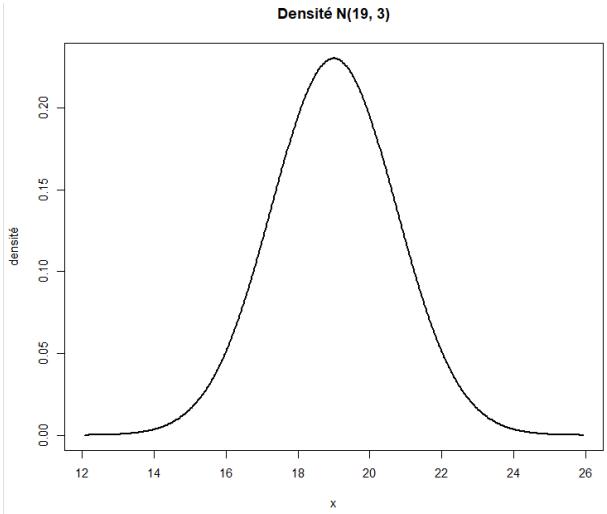


FIGURE 15 – Enter Caption

Exemple 2 — probas avec standardisation. $X \sim \mathcal{N}(15, 9)$, calculer $P(16 \leq X \leq 20)$.

```r
mu <- 15; sd <- 3
pnorm(20, mu, sd) - pnorm(16, mu, sd)      # direct
# ou en normalisant :
pnorm((20-mu)/sd) - pnorm((16-mu)/sd)      # Z ~ N(0,1)
```

Exemple 3 — quantiles (valeurs seuils). Trouver $q_{0.025}$ et $q_{0.975}$ de $N(0, 1)$.

```r
qnorm(0.025, 0, 1)    # ~ -1.96
qnorm(0.975, 0, 1)    # ~  +1.96
```

Pour $Y \sim \mathcal{N}(19, 3)$: $q_{0.975}(Y) = 19 + \sqrt{3}\, q_{0.975}(Z)$.

```r
mu <- 19; sd <- sqrt(3)
mu + sd * qnorm(0.975)   # vérifie la relation de changement d'échelle
```

Exemple 4 — simulation et histogramme avec densité.

```r
set.seed(123)
mu <- 15; sd <- sqrt(3)
x  <- rnorm(1000, mean=mu, sd=sd)
hist(x, freq=FALSE, col="gray90", border="white",
     main="Histogramme + densité théorique", xlab="x")
curve(dnorm(x, mean=mu, sd=sd), add=TRUE, lwd=2, col="red")
```

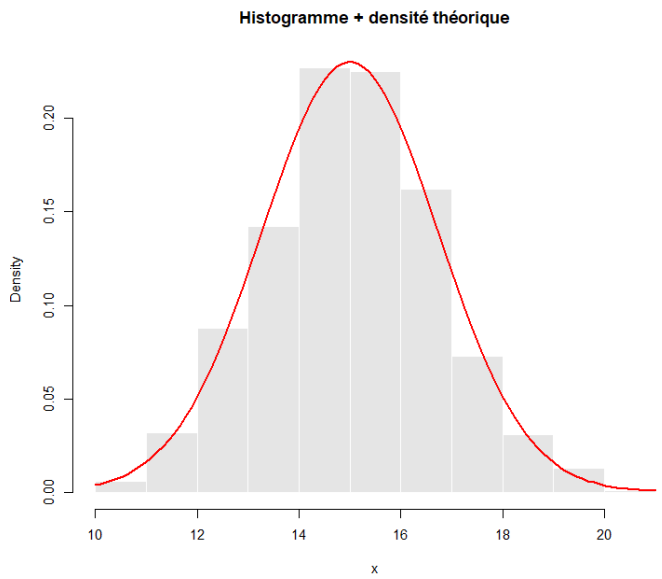


FIGURE 16 – Histogramme simulé (1000 tirages) et densité théorique de la loi $\mathcal{N}(15, 3)$.

Exemple 5 — tracer la fonction de répartition $F(x)$.

```
mu <- 0; sd <- 1
curve(pnorm(x, mu, sd), from=-4, to=4, lwd=2,
      main="Fonction de répartition N(0,1)", xlab="x", ylab="F(x)")
abline(h=c(0,1), v=0, lty=3)
```

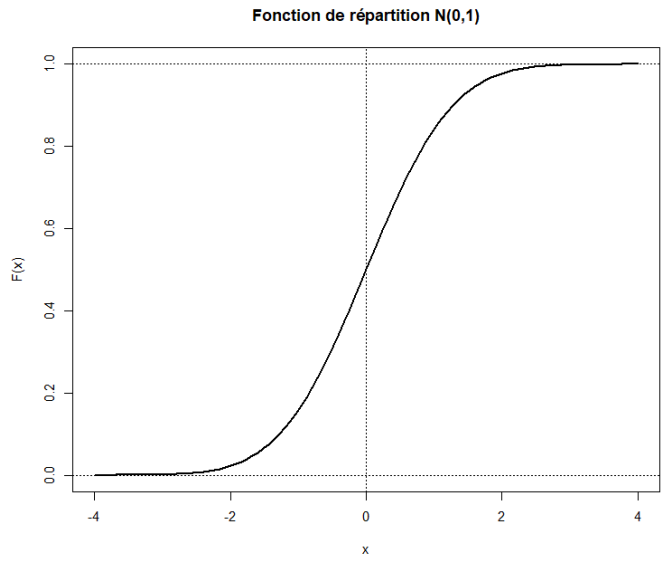


FIGURE 17 – Fonction de répartition de la loi normale standard $N(0, 1)$.

Exemple d'analyse avec le jeu de données `airquality`

Dans cette section, nous allons mettre en application l'ensemble des notions vues dans ce document (structures de données, manipulation, statistiques, visualisation, modélisation, etc.) en utilisant un jeu de données réel déjà inclus dans R : `airquality`.

1. Accéder aux jeux de données intégrés dans R

R met à disposition de nombreux jeux de données intégrés, très utiles pour l'apprentissage et l'expérimentation.

— Pour voir la liste de tous les jeux de données disponibles dans votre session (pratique si vous cherchez un dataset pour vos projets personnels) :

```
1 data()
```

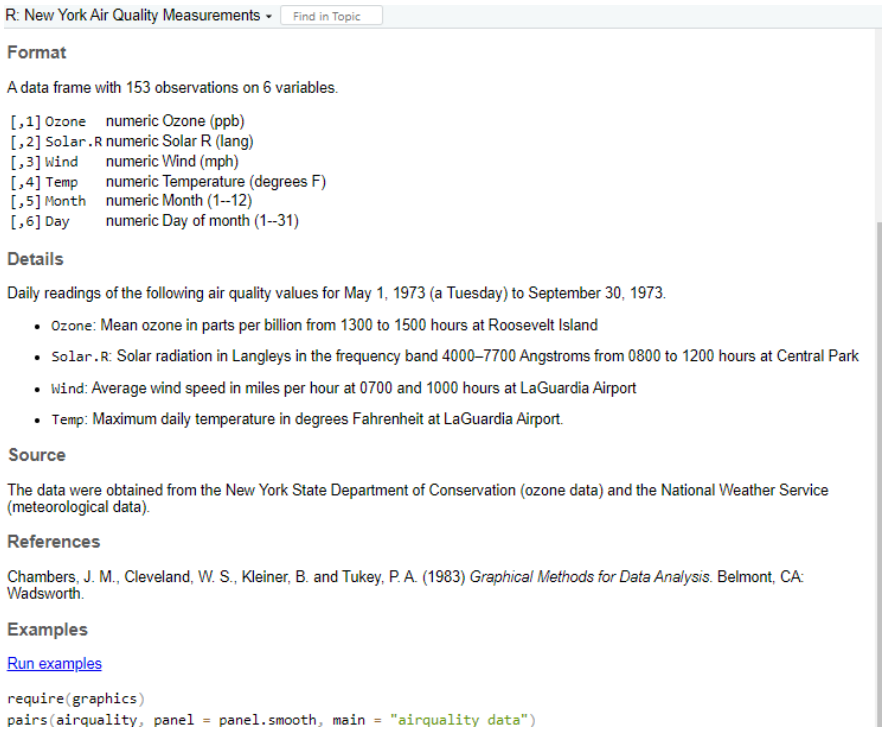
```
Data sets in package 'datasets':

AirPassengers                Monthly Airline Passenger Numbers 1949-1960
BJsales                      Sales Data with Leading Indicator
BJsales.lead (BJsales)       Sales Data with Leading Indicator
BOD                          Biochemical Oxygen Demand
CO2                          Carbon Dioxide Uptake in Grass Plants
ChickWeight                  Weight versus age of chicks on different diets
DNase                        Elisa assay of DNase
EuStockMarkets               Daily Closing Prices of Major European Stock Indices, 1991-1998
Formaldehyde                 Determination of Formaldehyde
HairEyeColor                 Hair and Eye Color of Statistics Students
Harman23.cor                 Harman Example 2.3
Harman74.cor                 Harman Example 7.4
Indometh                     Pharmacokinetics of Indomethacin
InsectSprays                 Effectiveness of Insect Sprays
JohnsonJohnson               Quarterly Earnings per Johnson & Johnson Share
LakeHuron                    Level of Lake Huron 1875-1972
LifeCycleSavings             Intercountry Life-Cycle Savings Data
Loblolly                     Growth of Loblolly Pine Trees
Nile                         Flow of the River Nile
Orange                       Growth of Orange Trees
OrchardSprays                Potency of Orchard Sprays
PlantGrowth                  Results from an Experiment on Plant Growth
Puromycin                    Reaction Velocity of an Enzymatic Reaction
Seatbelts                    Road Casualties in Great Britain 1969-84
Theoph                       Pharmacokinetics of Theophylline
Titanic                      Survival of passengers on the Titanic
ToothGrowth                  The Effect of Vitamin C on Tooth Growth in Guinea Pigs
UCBAdmissions                Student Admissions at UC Berkeley
UKDriverDeaths               Road Casualties in Great Britain 1969-84
UKgas                        UK Quarterly Gas Consumption
USAccDeaths                  Accidental Deaths in the US 1973-1978
USArrests                    Violent Crime Rates by US State
USJudgeRatings               Lawyers' Ratings of State Judges in the US Superior Court
USPersonalExpenditure        Personal Expenditure Data
UScitiesD                    Distances Between European Cities and Between US Cities
VADeaths                     Death Rates in Virginia (1940)
WWWusage                     Internet Usage per Minute
WorldPhones                  The World's Telephones
ability.cov                  Ability and Intelligence Tests
airmiles                     Passenger Miles on Commercial US Airlines, 1937-1960
airquality                   New York Air Quality Measurements
anscombe                     Anscombe's Quartet of 'Identical' Simple Linear Regressions
attenu                       The Joyner-Boore Attenuation Data
attitude                     The Chatterjee-Price Attitude Data
austres                      Quarterly Time Series of the Number of Australian Residents
```

FIGURE 18 – Extrait du résultat de la commande `data()`.

Chaque jeu de données intégré possède une page d'aide détaillée que l'on peut consulter. Exemple avec `airquality` :

```
?airquality
```



R: New York Air Quality Measurements — Find in Topic

Format

A data frame with 153 observations on 6 variables.

[,1]	Ozone	numeric Ozone (ppb)
[,2]	Solar.R	numeric Solar R (lang)
[,3]	Wind	numeric Wind (mph)
[,4]	Temp	numeric Temperature (degrees F)
[,5]	Month	numeric Month (1--12)
[,6]	Day	numeric Day of month (1--31)

Details

Daily readings of the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.

- `Ozone`: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
- `Solar.R`: Solar radiation in Langleys in the frequency band 4000–7700 Angstroms from 0800 to 1200 hours at Central Park
- `Wind`: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
- `Temp`: Maximum daily temperature in degrees Fahrenheit at LaGuardia Airport.

Source

The data were obtained from the New York State Department of Conservation (ozone data) and the National Weather Service (meteorological data).

References

Chambers, J. M., Cleveland, W. S., Kleiner, B. and Tukey, P. A. (1983) *Graphical Methods for Data Analysis*. Belmont, CA: Wadsworth.

Examples

Run examples

```
require(graphics)
pairs(airquality, panel = panel.smooth, main = "airquality data")
```

FIGURE 19 – Documentation du jeu de données `airquality`.

Téléchargement de jeux de données

Pour explorer d'autres jeux de données libres à télécharger :

- Rdatasets : collection de jeux de données R
- CRAN : site officiel de R
- data.gouv.fr : données publiques françaises

2. Chargement et description du jeu de données

Charger `airquality`, vérifier sa structure et comprendre les variables.

```
# 1) Charger le dataset
data("airquality")

# 2)Copie
data <- airquality

# 3) Dimensions, structure
```

```r
dim(data)          # nb lignes x nb colonnes
nrow(data)         # nb d'observations (lignes)
ncol(data)         # nb de variables (colonnes)
str(data)          # types des variables
head(data, 10)     # les 10 premières lignes
tail(data, 10)     # les 10 dernières lignes
summary(data)      # stats descriptives (montre aussi les NA)
names(data)        # noms des colonnes
```

```r
colnames(data)           # nom des colonnes = names(data)
rownames(data)           # noms des lignes

# 5) Types de données
sapply(data, class)      # classe de chaque variable
sapply(data, typeof)     # type de chaque variable

# 6) Valeurs manquantes
colSums(is.na(data))     # nb de NA par variable
anyNA(data)              # TRUE si des NA sont présents

# 7) Doublons
anyDuplicated(data)      # nb de lignes dupliquées
sum(duplicated(data))    # comptage exact

# 9) Statistiques élémentaires par variable numérique
sapply(data, min, na.rm=TRUE)   # minimum
sapply(data, max, na.rm=TRUE)   # maximum
sapply(data, mean, na.rm=TRUE)  # moyenne
sapply(data, median, na.rm=TRUE)# médiane
sapply(data, sd, na.rm=TRUE)    # écart-type
sapply(data, var, na.rm=TRUE)   # variance
sapply(data, IQR, na.rm=TRUE)   # intervalle interquartile
```

3 Exemple de visualisations sur le dataset

Explorer visuellement la distribution des variables et leurs relations afin de mieux comprendre la structure globale du jeu de données.

3.1 Histogrammes. Les histogrammes permettent d'observer la distribution d'une variable numérique.

```r
# Histogrammes (deux graphiques côte à côte)
par(mfrow = c(1,2))
hist(data$Ozone, main="Distribution de lOzone",
```

```
     xlab="Ozone (ppb)", col="lightblue")
hist(data$Temp, main="Distribution de la Température",
     xlab="Température (F)", col="lightcoral")
par(mfrow = c(1,1))  # revenir au mode 1 graphique
```

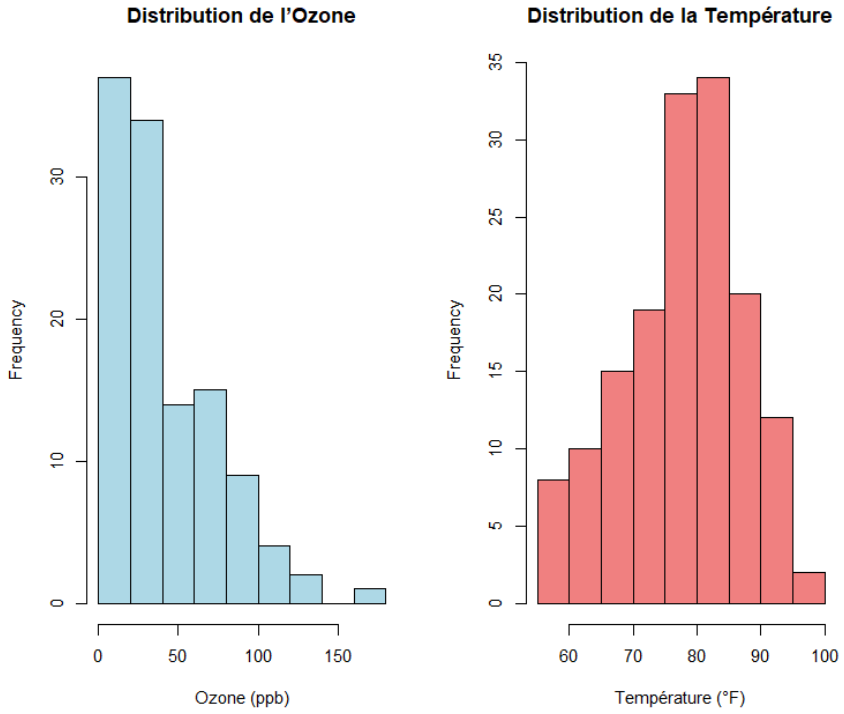


FIGURE 20 – Histogrammes de la distribution de l'ozone et de la température.

Remarque : La distribution de l'ozone est très asymétrique avec beaucoup de valeurs faibles et quelques pics élevés. La température, en revanche, est concentrée entre 70 et 90 °F, ce qui traduit une répartition plus régulière et saisonnière.

3.2 Boxplots. Les boxplots mettent en évidence la médiane, la dispersion et les valeurs extrêmes.

```
par(mfrow = c(1,2))
boxplot(data$Ozone, main="Boîte à moustaches  Ozone", col="lightgreen")
boxplot(data$Temp,  main="Boîte à moustaches  Température", col="orange")
par(mfrow = c(1,1))
```

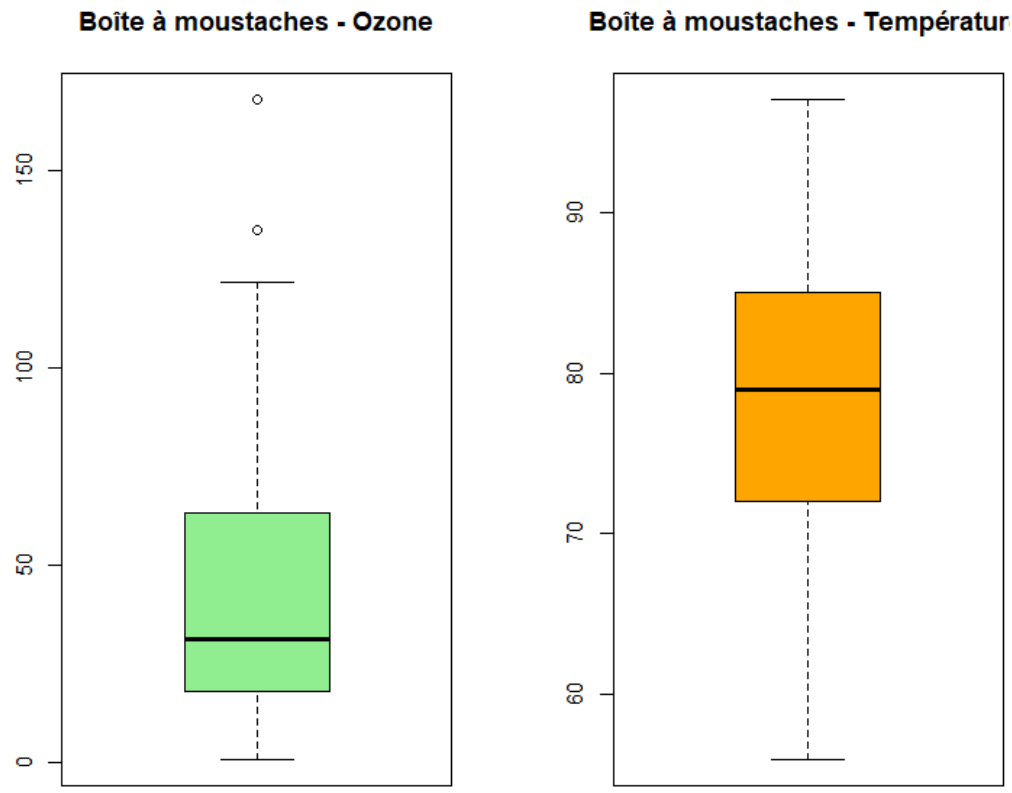


FIGURE 21 – Boxplots globaux de l'ozone et de la température.

L'ozone présente une forte variabilité avec des valeurs extrêmes au-dessus de 100 ppb. La température est plus homogène, centrée autour de 80 °F, avec une dispersion modérée et peu de valeurs aberrantes.

3.3 Boxplots par mois. Utile pour comparer la répartition de l'ozone selon les mois (mai à septembre).

```
# Boxplot Ozone par mois
boxplot(Ozone ~ Month, data = data,
        main = "Ozone par mois",
        xlab = "Mois", ylab = "Ozone (ppb)",
        col = "lightblue")
```

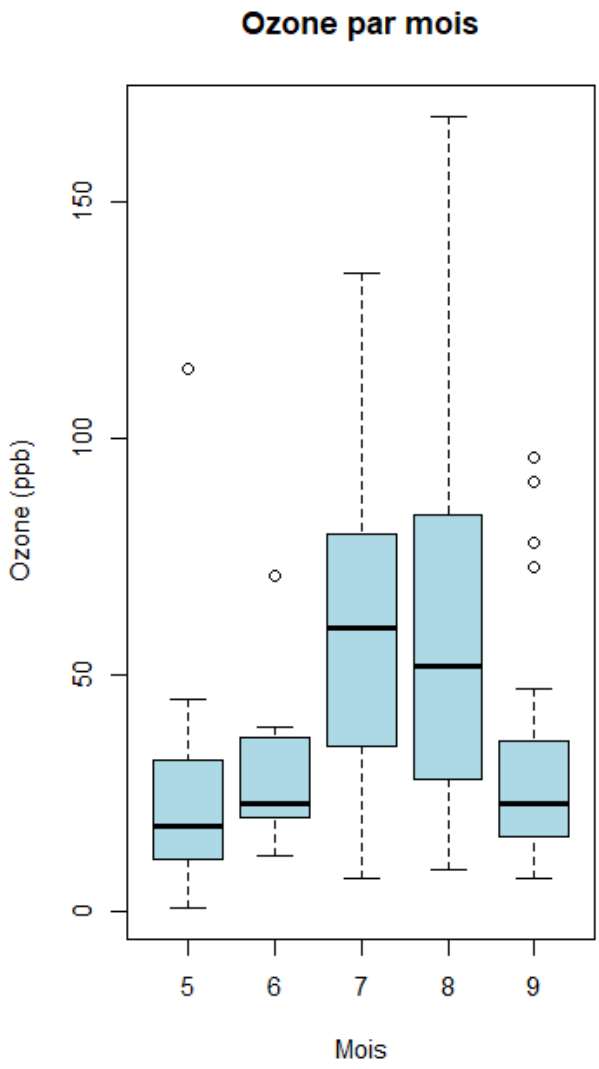


FIGURE 22 – Distribution de l'ozone par mois (mai–septembre).

Les niveaux d'ozone sont faibles au printemps (mai–juin) puis augmentent fortement en juillet et août, avant de redescendre en septembre. La variabilité est particulièrement marquée en été, avec de nombreux outliers.

3.4 Nuage de points. Permet d'examiner la relation entre la température et l'ozone.

```
# Scatterplot Ozone ~ Température
plot(data$Temp, data$Ozone,
     pch=19, col="darkgreen",
     main="Lien entre Ozone et Température",
     xlab="Température (F)", ylab="Ozone (ppb)")
```

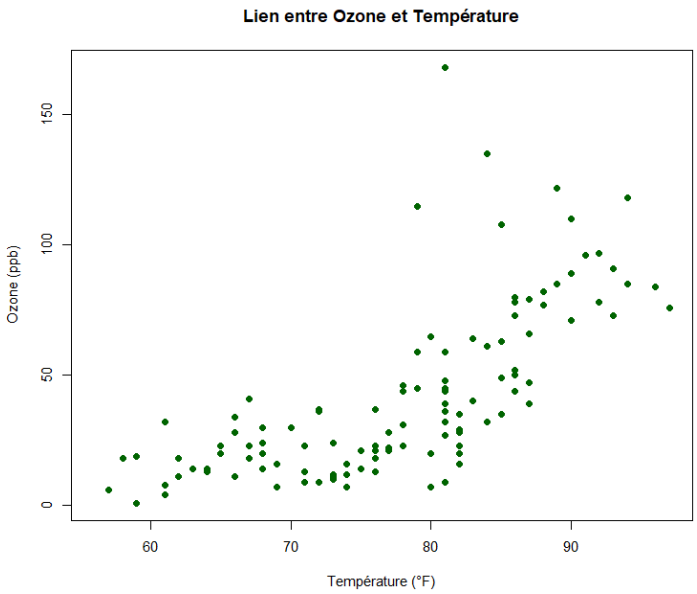


FIGURE 23 – Nuage de points entre la température et l'ozone.

On observe une corrélation positive entre la température et l'ozone : plus la température augmente, plus les niveaux d'ozone tendent à s'élever. Cependant, la dispersion reste importante pour les fortes chaleurs.

3.5 Matrice de corrélation visuelle. Visualise les relations entre les variables numériques principales.

```
# Matrice de nuages de points
pairs(data[, c("Ozone","Solar.R","Wind","Temp")],
      main = "Matrice de nuages de points  Variables principales")
```

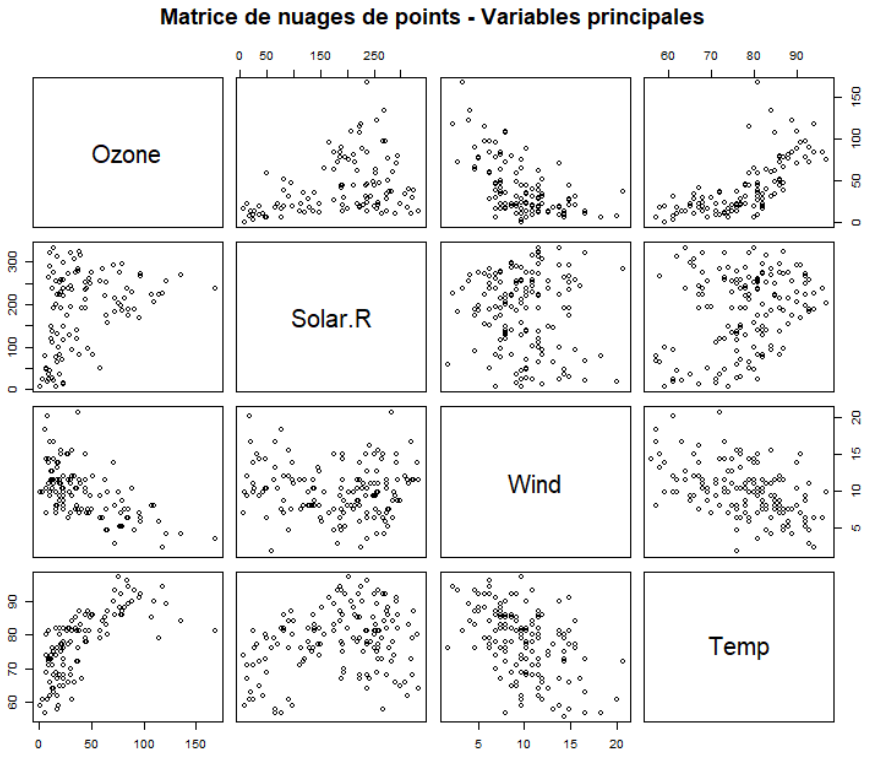


FIGURE 24 – Matrice de nuages de points pour Ozone, Solar.R, Wind et Temp.

La matrice confirme une corrélation positive entre l'ozone et la température, tandis que le vent est plutôt associé négativement avec l'ozone. La variable Solar.R montre une relation plus diffuse avec les autres indicateurs.

4. Exemple de nettoyage applicable au jeu de données

4.1 Valeurs manquantes

```r
# Créer un dataset avec uniquement les lignes complètes
data_cc <- na.omit(airquality)

# Vérifier les dimensions
dim(data_cc)
```

4.2 Imputation simple des NA

```r
# Imputation par la médiane de chaque variable
```

```r
data_imp <- airquality
for (col in c("Ozone", "Solar.R")) {
  data_imp[[col]][is.na(data_imp[[col]])] <- median(data_imp[[col]], na.rm = TRUE)
}

# Vérification des NA restants
colSums(is.na(data_imp))
```

4.3 Recodage des mois pour plus de lisibilité

```r
# Recoder la variable Mois en facteur avec labels
data_imp$Month <- factor(data_imp$Month,
                         levels = 5:9,
                         labels = c("Mai","Juin","Juillet","Août","Septembre"))

# Vérification du nouveau codage
table(data_imp$Month)
```

4.4 Renommage des colonnes

```r
names(data_imp) <- c("Ozone_ppb", "Rayonnement", "Vent",
                     "Temperature_F", "Mois", "Jour")
names(data_imp)
```

4.5 Conversion d'unités

```r
# Ajouter une colonne en C
data_imp$Temperature_C <- (data_imp$Temperature_F - 32) * 5/9
head(data_imp[, c("Temperature_F", "Temperature_C")], 5)
```

4.6 Détection et traitement des valeurs aberrantes

```r
# Identifier les valeurs extrêmes d'ozone (>150)
which(data_imp$Ozone_ppb > 150)
```

```r
# Supprimer les outliers d'ozone
data_no_outliers <- subset(data_imp, Ozone_ppb <= 150)
dim(data_no_outliers)
```

4.7 Normalisation d'une variable

```r
# Normalisation Min-Max de lozone (entre 0 et 1)
ozone_norm <- (data_imp$Ozone_ppb - min(data_imp$Ozone_ppb)) /
              (max(data_imp$Ozone_ppb) - min(data_imp$Ozone_ppb))
summary(ozone_norm)
```

```r
# Standardisation (moyenne 0, écart-type 1)
ozone_z <- scale(data_imp$Ozone_ppb)
summary(as.numeric(ozone_z))
```

5 Visualisation sur une varible

```r
# Comparer la distribution dOzone avant/après filtrage des outliers
op <- par(mfrow = c(1,2))  # sauvegarde de l'état graphique
boxplot(airquality$Ozone, main="Ozone brut", col="tomato")
boxplot(data_no_outliers$Ozone_ppb, main="Ozone nettoyé", col="lightgreen")
par(op)  # on rétablit l'état graphique initial
```

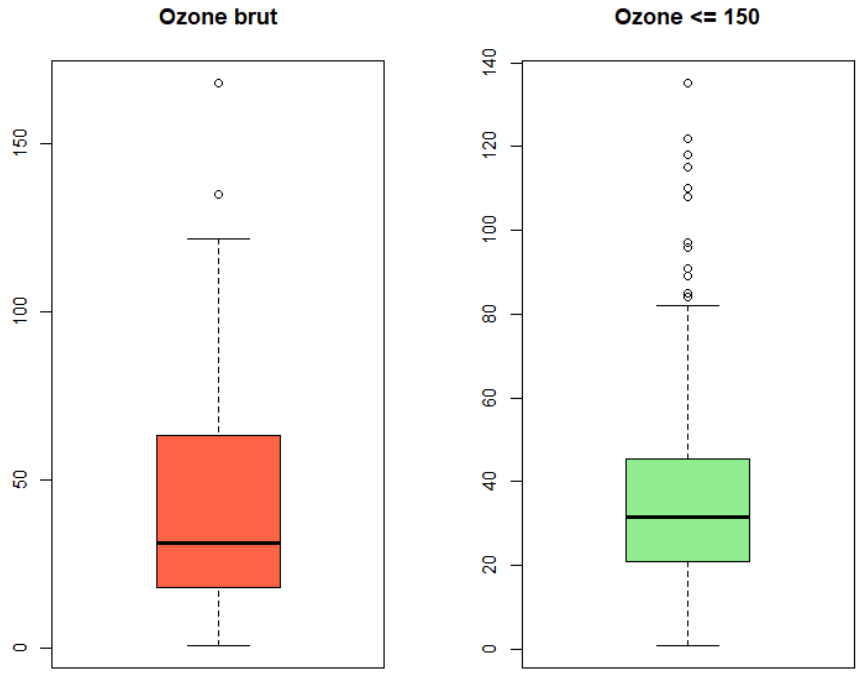


FIGURE 25 – Comparaison de la distribution de l'ozone avant et après filtrage des valeurs extrêmes.

6 Visualisation multiple

Le code suivant crée une figure en grille 2x3 avec six graphiques :

— Histogramme + densité pour l'Ozone, la Température, le Vent et le Rayonnement.
— Une comparaison de densités (Ozone vs Température).
— Un boxplot de l'Ozone par mois.

```r
# Fonction utilitaire : histogramme + densité
histo_dens <- function(x, main, xlab, col_hist, col_dens) {
  x <- x[!is.na(x)]
  if (length(x) < 2) { plot.new(); title(main); return(invisible()) }
  d  <- density(x)
  h0 <- hist(x, plot = FALSE, breaks = "FD")
  ylim_max <- max(max(h0$density, na.rm = TRUE), max(d$y, na.rm = TRUE)) * 1.08
  xlim_use <- range(c(range(h0$breaks, na.rm=TRUE), range(d$x, na.rm=TRUE)))
  hist(x, probability = TRUE, col = col_hist, border = "white",
       main = main, xlab = xlab, ylim = c(0, ylim_max), xlim = xlim_use)
  lines(d, col = col_dens, lwd = 2)
  rug(x)
}

op <- par(mfrow = c(2,3), mar = c(4,4,2,1), oma = c(0,0,1,0), xaxs = "i", yaxs = "i
    ")

# (1) Ozone
histo_dens(airquality$Ozone,   main = "Ozone",        xlab = "ppb",
           col_hist = "lightblue", col_dens = "red")

# (2) Température
histo_dens(airquality$Temp,    main = "Température",  xlab = "F",
           col_hist = "lightgreen", col_dens = "blue")

# (3) Vent
histo_dens(airquality$Wind,    main = "Vent",         xlab = "mph",
           col_hist = "lightyellow", col_dens = "darkgreen")

# (4) Rayonnement
histo_dens(airquality$Solar.R, main = "Rayonnement",  xlab = "lang",
           col_hist = "pink", col_dens = "purple")

# (5) Comparaison densités Ozone/Température
oz  <- na.omit(airquality$Ozone)
tm  <- na.omit(airquality$Temp)
d1  <- density(oz); d2 <- density(tm)
xlim_use <- range(c(d1$x, d2$x))
ylim_use <- c(0, max(d1$y, d2$y) * 1.08)
```

```r
plot(d1, col = "red", lwd = 2, main = "Comparaison des densités",
     xlab = "Valeur", xlim = xlim_use, ylim = ylim_use)
lines(d2, col = "blue", lwd = 2)
legend("topright", legend = c("Ozone", "Température"),
       col = c("red","blue"), lwd = 2, bty = "n")

# (6) Boxplot Ozone par mois
boxplot(Ozone ~ Month, data = airquality, col = "lightcyan",
        main = "Ozone par mois", xlab = "Mois (5=Mai ... 9=Sept.)", ylab = "ppb")
par(op)
```

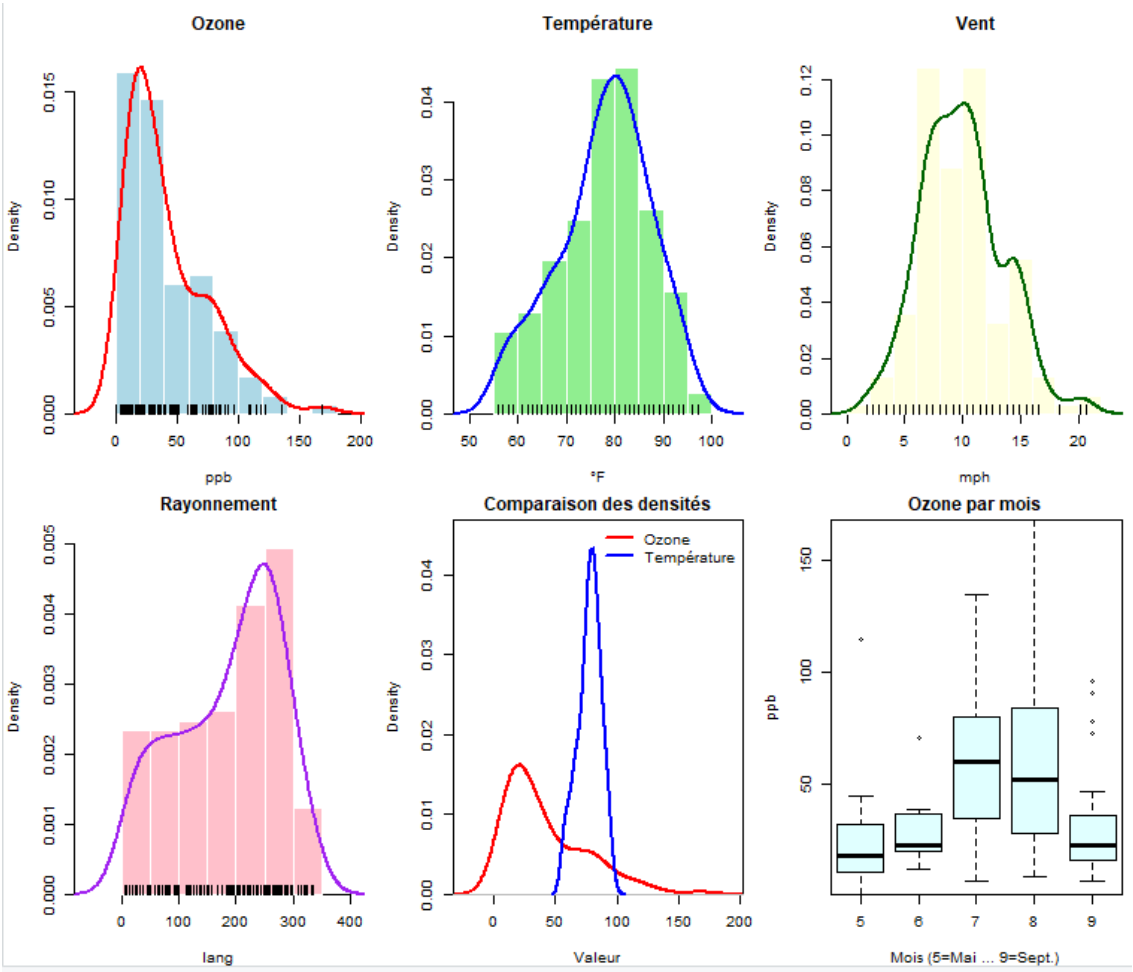


FIGURE 26 – Visualisation en grille (2x3) : distributions, densités et comparaison par mois.

Interprétation : Ces graphiques résument les distributions des variables principales du jeu de données `airquality`. Les histogrammes montrent que l'ozone est fortement concentré à de faibles valeurs, tandis que la température suit une distribution plus régulière centrée autour de 80°F. Le vent est globalement distribué entre 5 et 15 mph, et le rayonnement solaire présente une forte variabilité. La comparaison des densités met en évidence des profils très

différents pour l'ozone et la température. Enfin, le boxplot par mois révèle que l'ozone atteint ses valeurs les plus élevées en juillet et août, avec une forte variabilité et plusieurs valeurs extrêmes.

Après cette partie consacrée à l'exploration et à la visualisation des données, les sections suivantes aborderont des thématiques avancées : les *tests statistiques* pour valider des hypothèses, la *modélisation* afin de prédire et expliquer les phénomènes observés, et enfin l'outil **R Markdown**, qui permet de combiner code, résultats.