R : Introduction à R — Analyse R | Cours 1

Ményssa Cherifa-Luron

2023-09-15

Contents

Introduction

Créé en 1993, R est un langage de programmation open source spécialement conçu pour l'analyse statistique et la visualisation des données. Facile à prendre en main et très efficace, il est un outil essentiel pour tous les acteurs.trices de la data science.

Le cours vise à vous fournir une solide base de programmation en R, en mettant l'accent sur la compréhension des concepts fondamentaux, la pratique régulière et l'évaluation continue de vos compétences. Les démonstrations, les exercices et les QCM sont conçus pour vous aider à développer votre maîtrise de R au fil du cours.

Organisation

Contenu du Cours : - Démonstrations - Exercices

Ressources

- Rbloggers : Blog Populaire sur le langage R
- Datacamp : Plateforme d'apprentissage de la data science interactive
- Big Book of R : Edition qui rassemble des livres open-source sur le langage R
- Introduction accélérée au langage R pour la data science : L'essentiel et plus de tout ce que nous verrons

Les bases de la programmation R

Qu'est-ce que R ?

R est un langage de programmation et un environnement logiciel libre dédié à l'analyse statistique et à la représentation graphique. Il est largement utilisé par les statisticiens, data scientist et les chercheurs pour diverses analyses et projets de recherche.

L'une des principales raisons pour lesquelles R est si populaire est sa flexibilité et sa capacité à s'intégrer avec d'autres langages de programmation comme Python, Java, et C++. De plus, R dispose d'une vaste bibliothèque de packages, ce qui le rend extrêmement extensible.

Concepts de base de l'analyse des données.

L'analyse des données est le processus d'inspection, de nettoyage, de transformation et de modélisation des données dans le but de découvrir des informations utiles et de soutenir la prise de décision.

Avec R, vous pouvez effectuer toutes ces étapes de manière efficace et précise. Dans ce cours, vous retrouverez les concepts de base de l'analyse des données avec R : types de données, manipulation des données, visualisation des données et l'interprétation des résultats d'analyse.

Installation de R et RStudio sur différentes plateformes.

Installation de R : Rendez-vous sur le site officiel de R et téléchargez la version appropriée pour votre système d'exploitation (Windows, Mac, Linux). Suivez les instructions d'installation.

Installation de RStudio : Après avoir installé R, allez sur le site officiel de RStudio et téléchargez la version gratuite de RStudio Desktop. Installez-le comme n'importe quel autre logiciel.

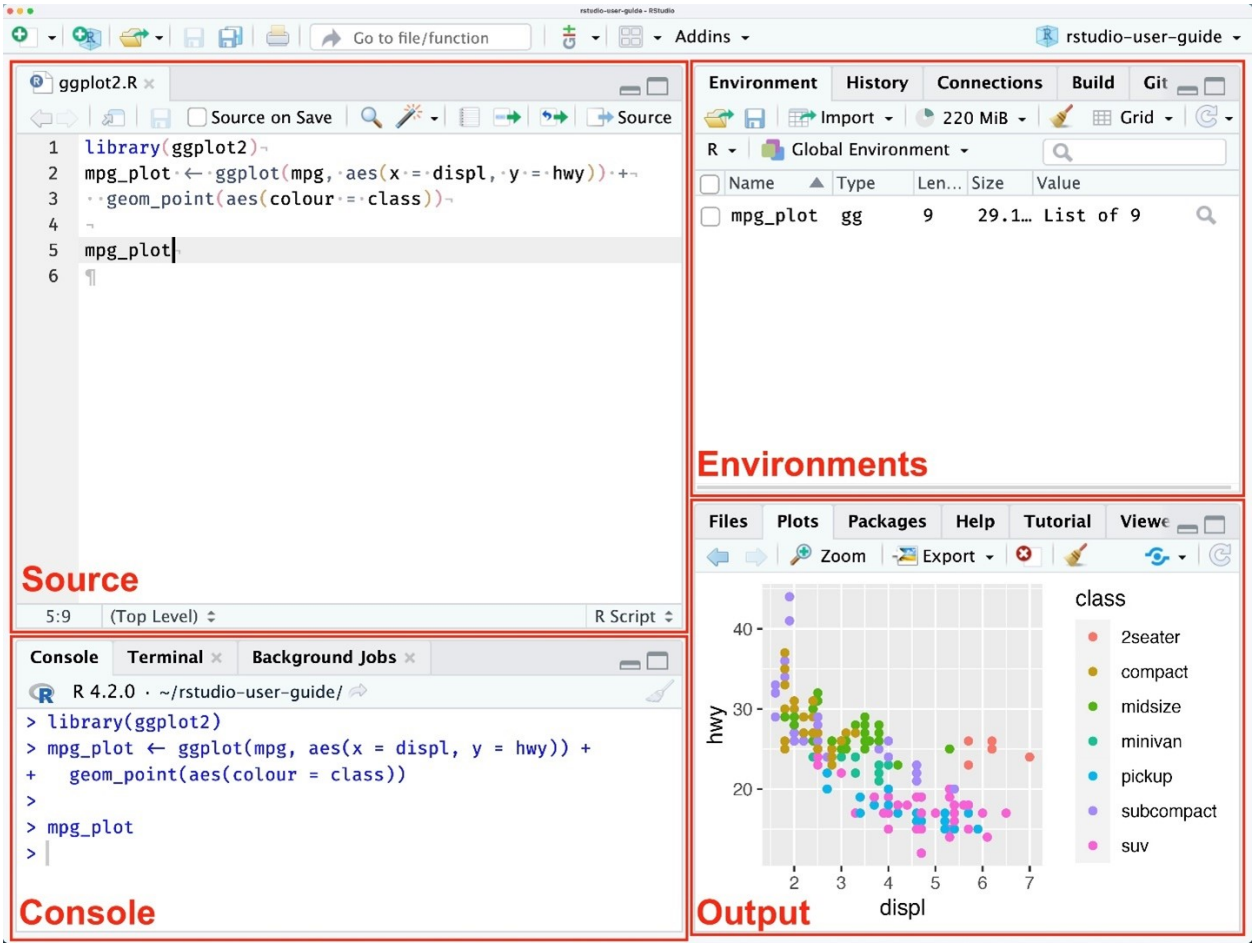

Figure 1: Interface de RStudio

Lancement de RStudio : Une fois RStudio installé, ouvrez-le. Vous verrez quatre panneaux principaux :

- **la 1) console**
- **2) l'éditeur de script (source),**
- **3) l'environnement et l'historique, et**

- **les 4) fichiers/plots/packages/aide (output).**

RStudio est une interface de développement intégrée (IDE) pour R.

Elle offre une interface utilisateur conviviale pour R, ce qui facilite l'écriture de code, l'exécution de scripts et la visualisation des résultats.

! Toujours prendre la bonne habitude de définir un projet dans RStudio

Premiers pas avec R

Pour commencer avec R et RStudio Dans la console, vous pouvez taper des commandes R et voir les résultats immédiatement. Essayez de taper 2 + 2 et appuyez sur Entrée. Vous devriez voir le résultat 4.

```
2 + 2
```

```
## [1] 4
```

Commentaires

Un commentaire en R est un morceau de texte dans votre script R qui n'est pas exécuté comme une partie du code. Les commentaires sont essentiels pour documenter ce que fait votre code, clarifier des parties complexes, ou laisser des notes pour vous-même ou pour d'autres personnes qui pourraient lire ou utiliser votre code plus tard.

En R, un commentaire est créé en utilisant le symbole #.

```
# Ceci est un commentaire simple

x <- 5  # Ceci est un commentaire suivant une instruction
```

Installer et importer un package

Les packages sont des collections de fonctions R et de données. Pour installer un package, utilisez la commande

```
# install.packages("nom_du_package")
```

Par exemple, pour installer le package ggplot2, tapez install.packages("ggplot2"). Pour utiliser le package, tapez library(ggplot2).

```
# library(ggplot2)
```

Demander de l'aide

Si vous êtes perdu ou si vous avez besoin d'informations sur une fonction, tapez ?nom_de_la_fonction. Par exemple, ?mean vous communiquera des informations sur la fonction mean.

```
?mean
```

```
## starting httpd help server ... done
```

Figure 2: Structure basique pour les projets. Source: https://martinctc.github.io/.

```
# help(mean)
```

Écrire et exécuter son premier script R.

Écrire et exécuter un script R est une étape fondamentale pour se familiariser avec le langage R. Un script R est simplement un fichier contenant une séquence de commandes que R peut exécuter. Voici comment vous pouvez écrire et exécuter votre premier script R.

Créer un nouveau script

Dans RStudio, allez dans File > New File > R Script. Cela ouvrira un nouvel onglet d'éditeur où vous pouvez commencer à écrire votre script.

Écrire le script Dans l'éditeur, commencez à taper vos commandes R. Par exemple, vous pouvez créer un vecteur, calculer sa moyenne, ou tracer un graphique.

Exécuter le script Pour exécuter votre script, vous pouvez soit sélectionner les lignes que vous souhaitez exécuter et cliquer sur le bouton Run, soit sauvegarder votre script et le source dans la console R en utilisant la commande source("chemin_vers_votre_script.R").

Sauvegarder le script Il est toujours recommandé de sauvegarder votre script pour une utilisation future. Pour ce faire, allez dans File > Save ou appuyez sur Ctrl + S.

Démonstration

> Travailler par projets dans RStudio. Création du premier script dans le projet enseignement R Tour d'horizon de RStudio : commandes d'exécution, cheatsheets, les packages… Concept de base : • Création, modification, suppression d'un objet et structure dans R • Opérations arithmétiques • Unité élémentaire : le vecteur

Gestion de l'environnement R

Cette partie du script R couvre plusieurs opérations de base liées à la gestion de l'environnement de travail, l'installation et le chargement de packages, et l'utilisation de jeux de données intégrés à R. Voici une explication détaillée pour chaque ligne de code :

Répertoire de Travail et Changement de Répertoire

1. **`getwd()` :** fonction qui retourne le chemin du répertoire de travail actuel. Le répertoire de travail est le dossier dans lequel R cherche les fichiers à lire et dans lequel il enregistre les fichiers.
2. **`setwd(dir = "chemin_vers_nouveau_repertoire")` :** utilisée pour changer le répertoire de travail courant de R. Vous remplacez `"chemin_vers_nouveau_repertoire"` par le chemin du dossier où vous souhaitez travailler.

Gestion des Packages

3. **`install.packages("ggplot2")` :** permet l'installation du package `ggplot2` depuis le CRAN (Comprehensive R Archive Network). `ggplot2` est largement utilisé pour la visualisation de données.
4. **`library(dplyr)` :** charge le package `dplyr` dans la session R actuelle. `dplyr` est un package pour la manipulation de données.

Utilisation de Jeux de Données

5. **`data()` :** fonction qui affiche tous les jeux de données disponibles par défaut dans R. Ces jeux de données sont souvent utilisés pour l'enseignement et la pratique.
6. **`?mtcars` :** le `?` devant un nom d'objet est une manière rapide d'accéder à l'aide ou à la documentation de cet objet. Ici, `?mtcars` ouvrira l'aide pour le jeu de données `mtcars`, fournissant des informations sur sa structure et son contenu.
7. **`data("mtcars")` :** permet de charger le jeu de données `mtcars` dans l'environnement de travail R. `mtcars` est un jeu de données classique contenant des informations sur différents modèles de voitures.

```r
# Répertoire de travail et changer de répertoire

# Afficher le répertoire courant : getwd()

# Changer de répertoire courant
# setwd(dir = "chemin_vers_nouveau_repertoire")

# Installation d'un package
# install.packages("ggplot2")

# Chargement d'un package
# library(dplyr)

# Afficher tous les jeux de données par défaut dans R : data()
data()

# Afficher la description d'un jeu de données
?mtcars

# Charger un jeu de données
data("mtcars")
```

Création et Affichage de Variables

Les variables sont des conteneurs permettant de stocker des valeurs de données, telles que des nombres ou des chaînes de caractères.

1. **Création de la Variable `x` :**
   - `x <- 2` crée une variable nommée `x` et lui affecte la valeur `2`. Le symbole `<-` est l'opérateur d'affectation en R, utilisé pour assigner des valeurs aux variables.
2. **Affichage de la Variable `x` :**
   - `print(x)` affiche la valeur de la variable `x`. Ici, elle affichera `2`.
3. **Affichage du Type de la Variable `x` :**
   - `typeof(x)` renvoie le type de données stockées dans `x`. Pour `x <- 2`, le type sera `double` (nombre à virgule flottante).
4. **Création d'une Chaîne de Caractères :**
   - `chaine <- "Hello, World!"` crée une variable `chaine` contenant la chaîne de caractères `"Hello, World!"`.

- `print(chaine)` affiche cette chaîne de caractères.
- `typeof(chaine)` indique que `chaine` est de type `character`.

```
# Création de la variable x : " <-"
x <- 2

# Afficher la variable x
print(x)
```

```
## [1] 2
```

```
# Afficher le type de la variable x
typeof(x)
```

```
## [1] "double"
```

```
# Creation d'une chaine de caractere
chaine <- "Hello, Word!"
print(chaine)
```

```
## [1] "Hello, Word!"
```

```
typeof(chaine)
```

```
## [1] "character"
```

Règles de Nommage des Variables

- **Sensibilité à la Casse :** R est sensible à la casse, ce qui signifie que `mavariable` et `MaVariable` seraient considérées comme deux variables distinctes.
- **Différents Styles de Nommage :**
  - `mavariable` : Tout en minuscules.
  - `ma.variable` : Séparation par un point.
  - `nom_variable` : Séparation par un trait de soulignement (underscore).
  - `nomVariable` : Style lowerCamelCase, avec la première lettre de chaque mot en majuscule, sauf la première lettre de la variable.
  - `NomVariable` : Style PascalCase, avec la première lettre de chaque mot en majuscule.
- **Exemples d'Affectation :**

```
mavariable <- "jour"
ma.variable <- "semaine"
mavariable123 <- "mois"
print(mavariable)
```

```
## [1] "jour"
```

```r
print(ma.variable)
```

```
## [1] "semaine"
```

```r
print(mavariable123)
```

```
## [1] "mois"
```

Affichage et Formatage de Variables

1. **`print()` et `sprintf()` :**
   - `print(x)` affiche la valeur de la variable `x`.
   - `sprintf("%d", x)` utilise un formatage de style C pour afficher `x`. `%d` est un spécificateur de format pour les nombres entiers.
   - `sprintf("%s", chaine)` utilise `%s` pour formater et afficher une chaîne de caractères.
   - `sprintf("J'ai %d bananes", x)` combine une chaîne de caractères avec la valeur de `x`, en remplaçant `%d` par la valeur de `x`.

```r
# Affichage de la variable x : print() / sprintf
print(x)
```

```
## [1] 2
```

```r
sprintf("%d",x)
```

```
## [1] "2"
```

```r
sprintf("%s", chaine)
```

```
## [1] "Hello, Word!"
```

```r
x <- 3
sprintf("J'ai %d bananes", x)
```

```
## [1] "J'ai 3 bananes"
```

Arrondi de Nombres

2. **Utilisation de `round()` :**
   - `mon_nombre <- 3.2` crée une variable avec une valeur à virgule flottante.
   - `round(mon_nombre, 0)` arrondit `mon_nombre` à l'entier le plus proche.

```r
# Valeur arrondie : round()
mon_nombre <-  3.2

# Arrondir à la valeur entière
round(mon_nombre, 0)
```

```
## [1] 3
```

Vérification et Conversion de Types

3. **Type et Conversion :**
   - `class(mon_nombre)` renvoie le type de la variable `mon_nombre`.
   - `is.numeric(mon_nombre)` vérifie si `mon_nombre` est de type numérique.
   - `is.character(mon_nombre)` vérifie si `mon_nombre` est une chaîne de caractères.
   - Conversion de type :
     - `age_character <- "16"` : une chaîne de caractères représentant un âge.
     - `is.character(age_character)` vérifie que c'est bien une chaîne.
     - `age_numeric <- as.numeric(as.character(age_character))` convertit la chaîne en nombre.
     - `is.numeric(age_numeric)` confirme que la conversion a réussi.

```
# Vérification du type de la variable : class, is.character, is.numeric ...
class(mon_nombre)
```

```
## [1] "numeric"
```

```
is.numeric(mon_nombre) # renvoie TRUE
```

```
## [1] TRUE
```

```
is.character(mon_nombre) # renvoie FALSE
```

```
## [1] FALSE
```

```
# Convertir des variables numériques <-  char | char -> numerique
age_character <- "16"
is.character(age_character)
```

```
## [1] TRUE
```

```
age_numeric <-  as.numeric(as.character(age_character))
is.numeric(age_numeric)
```

```
## [1] TRUE
```

Gestion de l'Environnement de Travail

4. **Suppression et Listing d'Objets :**
   - `rm(age)` supprime l'objet `age` de l'environnement.
   - `ls()` liste tous les objets présents dans l'environnement de travail.
   - `rm(list = ls())` supprime tous les objets de l'environnement, nettoyant ainsi l'espace de travail.

```
# Suppression d'un objet : rm
rm(age)
```

```
## Warning in rm(age): objet 'age' introuvable
```

```r
# Affiche les éléments dans l'enV
ls()
```

```
## [1] "age_character" "age_numeric"   "chaine"        "ma.variable"
## [5] "mavariable"    "mavariable123" "mon_nombre"    "mtcars"
## [9] "x"
```

```r
# Supprime tous les éléments dans l'enV
rm(list = ls())
```

Opérations Arithmétiques de Base

1. **Addition (+) :**
    - `x + y` calcule la somme de `x` et `y`.

```r
x <- 3
y <- 5
x + y
```

```
## [1] 8
```

2. **Soustraction (-) :**
    - `x - y` calcule la différence entre `x` et `y`.

```r
x - y
```

```
## [1] -2
```

3. **Multiplication (*) :**
    - `x * y` calcule le produit de `x` et `y`.

```r
x * y
```

```
## [1] 15
```

4. **Division (/) :**
    - `x / y` calcule le quotient de `x` divisé par `y`.

```r
x / y
```

```
## [1] 0.6
```

Opérations Avancées

5. **Puissance (^ ou **) :**
    - `x ^ 2` ou `x**2` calcule `x` à la puissance de 2.

```r
x ^ 2
```

```
## [1] 9
```

6. **Racine Carrée (`sqrt()`) :**
   - `sqrt(25)` calcule la racine carrée de 25, ce qui donne 5.

```r
sqrt(25)
```

```
## [1] 5
```

7. **Division Entière (`%/%`) :**
   - `x %/% y` donne le résultat de la division entière de `x` par `y`. C'est le quotient de la division sans la partie fractionnaire.

```r
x %/% y
```

```
## [1] 0
```

8. **Modulo (`%%`) :**
   - `x %% y` donne le reste de la division entière de `x` par `y`.

```r
x %% y
```

```
## [1] 3
```

Unité élémentaire : le vecteur

Création et Affichage de Vecteurs

1. **Création de Vecteurs :**
   - Les vecteurs sont créés avec la fonction `c()`. `mon_vecteur` est un vecteur de nombres entiers, et `vecteur_caractere` est un vecteur de chaînes de caractères.

```r
# Création d'un vecteur : int, char
mon_vecteur <- c(1, 9, 10, 35)
print(mon_vecteur)
```

```
## [1]  1  9 10 35
```

```r
vecteur_caractere <-  c("Hello", "Word", "!")
print(vecteur_caractere)
```

```
## [1] "Hello" "Word"  "!"
```

2. **Typage Automatique :**
   - Dans `vecteur_multiple`, le mélange de types (chaîne de caractères et nombre) conduit R à convertir automatiquement tous les éléments en chaînes de caractères pour maintenir l'homogénéité du vecteur.

```r
# Typage automatique de 2 en chaine de caractère
vecteur_multiple <-  c("Hello", 2)
print(vecteur_multiple)
```

```
## [1] "Hello" "2"
```

Indexation des Vecteurs

3. **Accéder aux Éléments d'un Vecteur :**
    - `vecteur_caractere[2]` accède au deuxième élément du vecteur.
    - `mon_vecteur[1:3]` extrait les trois premiers éléments de `mon_vecteur`.

```r
# Indicage
vecteur_caractere[2] # affiche l'élément 2 du vecteur
```

```
## [1] "Word"
```

```r
vecteur_caractere[3] # affiche l'élément 3 du vecteur
```

```
## [1] "!"
```

```r
mon_vecteur[1:3] # affiche les 3 premieres valeurs du vecteur
```

```
## [1]  1  9 10
```

Opérations sur les Vecteurs

4. **Calculs avec des Éléments de Vecteur :**
    - `addition` montre un exemple de calcul en additionnant des éléments spécifiques du vecteur.
    - `length`, `sum`, et `mean` sont des fonctions pour obtenir la longueur du vecteur, la somme et la moyenne de ses éléments, respectivement.

```r
mon_vecteur <- c(1, 9, 10, 35)
addition <- mon_vecteur[1] + mon_vecteur[4]
print(addition)
```

```
## [1] 36
```

```r
# Opérations d'un vecteur : Longueur, somme, moyenne, ...
length(mon_vecteur)
```

```
## [1] 4
```

```r
sum(mon_vecteur)
```

```
## [1] 55
```

```r
mean(mon_vecteur)
```

```
## [1] 13.75
```

Gestion des Valeurs Manquantes

5. **Valeurs Manquantes (`NA`) dans les Vecteurs :**
    - `sum(vecteur_valeur_manquante, na.rm = TRUE)` et `mean(vecteur_valeur_manquante, na.rm = TRUE)` montrent comment calculer la somme et la moyenne en ignorant les valeurs manquantes (`NA`).

```r
# Cas des Valeurs manquantes dans un vecteur
vecteur_valeur_manquante <-  c(1, NA, 10, 24)

sum(vecteur_valeur_manquante) # renvoie NA
```

```
## [1] NA
```

```r
sum(vecteur_valeur_manquante, na.rm = TRUE) # retire les valeurs manquantes
```

```
## [1] 35
```

```r
mean(vecteur_valeur_manquante) # renvoie NA
```

```
## [1] NA
```

```r
mean(vecteur_valeur_manquante, na.rm = TRUE) # retire les valeurs manquantes
```

```
## [1] 11.66667
```

Combinaison de Vecteurs

6. **Fusion de Vecteurs :**
    - `nouveau_vecteur` est créé en combinant `mon_vecteur1` et `mon_vecteur2` en un seul vecteur.

```r
# Combinaison de vecteur
mon_vecteur1 <- c(1, 9, 10, 35)
mon_vecteur2 <- c(27, 49)

nouveau_vecteur <- c(mon_vecteur1, mon_vecteur2)
print(nouveau_vecteur)
```

```
## [1]  1  9 10 35 27 49
```

Collage de Chaînes de Caractères

7. **Formation de Chaînes de Caractères :**
    - `paste` et `paste0` sont utilisés pour concaténer des chaînes de caractères. `paste` permet d'insérer un séparateur entre les chaînes, tandis que `paste0` les concatène directement.
    - `sprintf` est utilisé pour le formatage de style C des chaînes.

```
# Combinaison de vecteur et "collage": paste, paste0 et sprintf (%s, %f ou %g)
nom <- "Dupont"
prenom <- "Jean"
age <- 15

identite <- paste(prenom, nom, sep = " ")
print(identite)
```

```
## [1] "Jean Dupont"
```

Recyclage dans les Vecteurs

8. **Recyclage de Vecteurs :**
    - Lorsque des opérations sont effectuées sur des vecteurs de longueurs différentes (`x + y`), R recycle les éléments du vecteur le plus court pour correspondre à la longueur du plus long.

```
# Recyclage de identité avec ajout de l'age
identite <- paste(identite, age, sep = " ")

# Recyclage d'identité avec l'ajout de "ans"
identite <- paste0(identite, " ans")
print(identite)
```

```
## [1] "Jean Dupont 15 ans"
```

```
# Définition de l'identité avec paste0
identite <- paste0("Je m'appelle ", prenom, " ", nom, ",", " j'ai ", age, " ans")
print(identite)
```

```
## [1] "Je m'appelle Jean Dupont, j'ai 15 ans"
```

```
# Recyclage d'un vecteur
x <-  c(1,5)
y <- c(9, 76, 20, 30)
x + y
```

```
## [1] 10 81 21 35
```

Exercices

Pour commencer à pratiquer, suivez ces étapes :

1. **Accédez au Dépôt GitHub :** Visitez l'URL fournie : https://github.com/universdesdonnees/Introduction-a-R pour accéder au dépôt GitHub contenant les matériaux du cours.
2. **Trouvez le Fichier des Exercices :** Dans le dépôt, localisez le fichier nommé **exercices1.txt**. Ce fichier contient les premiers exercices que vous devez pratiquer.
3. **Lisez et Essayez de Résoudre les Exercices :** Ouvrez le fichier **exercices1.txt** et lisez attentivement les exercices. Essayez de les résoudre par vous-même dans votre environnement R (comme RStudio). Il est important de pratiquer par vous-même avant de regarder les solutions pour mieux apprendre.
4. **Consultez la Correction :** Une fois que vous avez tenté de résoudre les exercices, ou si vous rencontrez des difficultés, consultez le fichier **correction_exercices1.R** pour voir les solutions. Analysez les solutions pour comprendre les méthodes et logiques utilisées.