R : Introduction à R — Analyse R | Cours 2

Ményssa Cherifa-Luron

2023-09-15

Contents

Introduction

Les structures de données en R servent à organiser et stocker les données de manière à faciliter leur manipulation, analyse et visualisation. En fonction de la nature et des exigences des données, différentes structures sont utilisées pour optimiser les performances, la facilité d'utilisation et l'efficacité du traitement des données.

Chaque structure a ses caractéristiques et est choisie en fonction des besoins spécifiques de l'analyse ou de la manipulation de données en question. En somme, ces structures sont essentielles pour une gestion efficace et une analyse rigoureuse des données dans R.

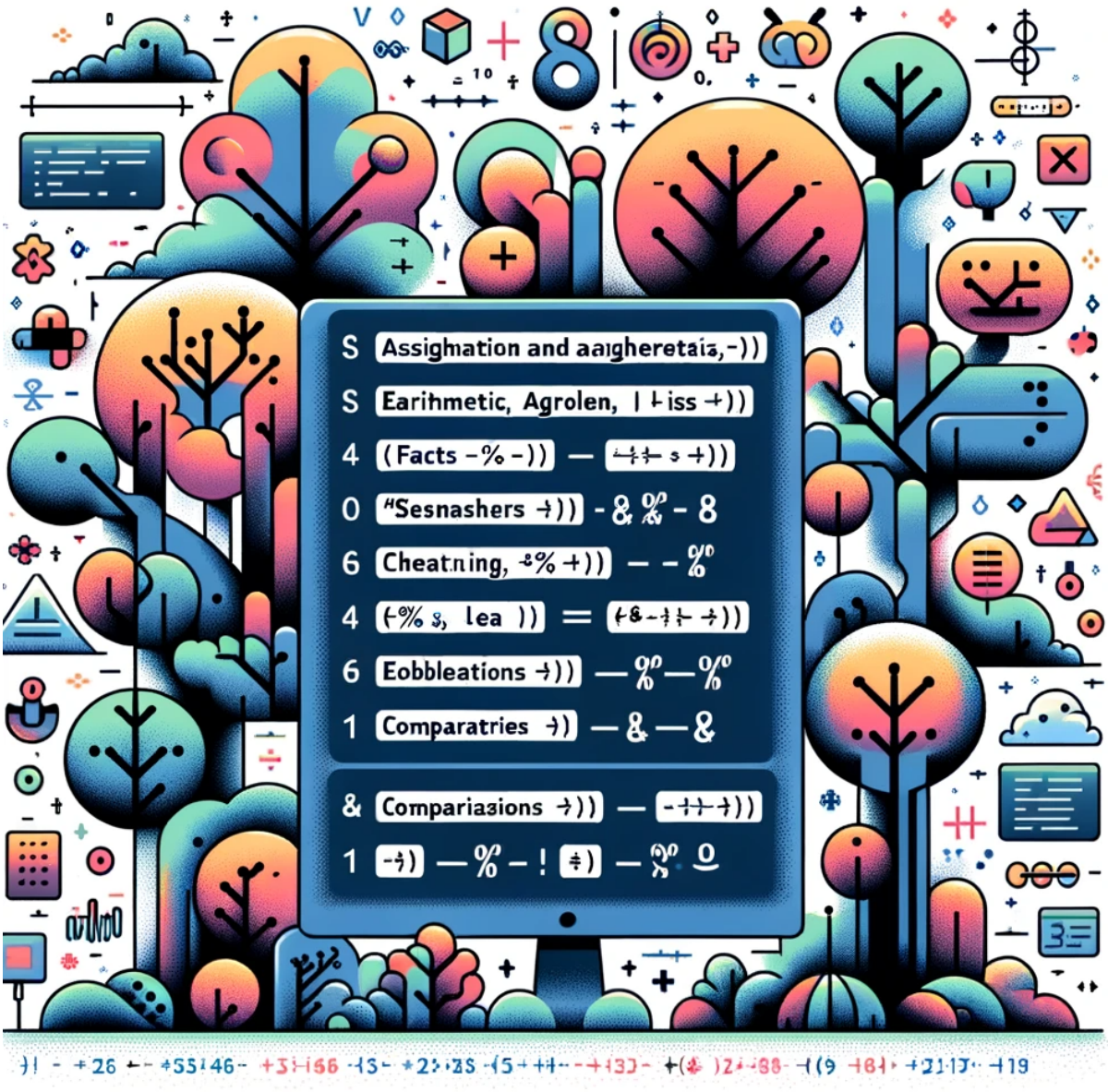

Figure 1: Illustration du langage R par DALL-E

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

Assignation et Opérateurs Arithmétiques, Booléens et Logiques

Opérateurs du Langage R

Tableau des principaux opérateurs en R :

Opérateur	Rôle
$	Extraction d'une liste ou d'une colonne
^	Puissance
-	Changement de signe
:	Génération de suites
%*%, %%, %/%	Produit matriciel, modulo, division entière
* /	Multiplication, division
+ -	Addition, soustraction
<, <=, ==, >=, >, !=	Comparaisons
!	Négation logique
&, &&	Opérateur ET logique
|, ||	Opérateur OU logique
->, -»	Assignation (droite)
<- , «-	Assignation (gauche)

Opérateurs d'Assignation (cf.cours1)

```
# Exemple d'assignation
x <- 10  # Assignation de 10 à x
```

Opérateurs Arithmétiques (cf.cours1)

```
# Exemple d'opération arithmétique
5 + 3 # Addition, renvoie 8
```

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

Opérateurs Booléens (de Comparaison)

Les opérateurs de comparaison en R sont utilisés pour comparer deux valeurs ou expressions et renvoient un résultat booléen (TRUE ou FALSE).

Ces opérateurs sont essentiels dans la programmation pour les tests de condition, les boucles, et le filtrage des données.

Les opérateurs de comparaison en R sont utilisés pour comparer deux valeurs ou expressions et renvoient un résultat booléen (TRUE ou FALSE). Ces opérateurs sont essentiels dans la programmation pour les tests de condition, les boucles, et le filtrage des données.

```
a <- 4
b <- 8
```

1. **Inférieur à (<) :**
    - `x < y` renvoie `TRUE` si `x` est inférieur à `y`.
    - Exemple : `3 < 5` renvoie `TRUE`.

```
# Inférieur
a < b
```

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

2. **Supérieur à (>) :**
    - `x > y` renvoie `TRUE` si `x` est supérieur à `y`.
    - Exemple : `5 > 3` renvoie `TRUE`.

```
# Supérieur
a > b
```

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

3. **Inférieur ou égal à (<=) :**
    - `x <= y` renvoie `TRUE` si `x` est inférieur ou égal à `y`.
    - Exemple : `3 <= 3` renvoie `TRUE`.

```
# Inférieur ou égal
a <= b
```

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

4. **Supérieur ou égal à (>=) :**
    - `x >= y` renvoie `TRUE` si `x` est supérieur ou égal à `y`.
    - Exemple : `5 >= 5` renvoie `TRUE`.

```
# Supérieur ou égal
a >= b
```

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

5. **Égal à (==) :**
    - `x == y` renvoie `TRUE` si `x` est égal à `y`.
    - Exemple : `3 == 3` renvoie `TRUE`.

```
# Egalité
a == b
```

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

6. **Différent de (!=) :**
    - `x != y` renvoie `TRUE` si `x` n'est pas égal à `y`.
    - Exemple : `3 != 5` renvoie `TRUE`.

```
# Non égalité
a != b
```

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

Opérateurs Logiques

Les opérateurs logiques sont utilisés pour combiner des expressions conditionnelles et renvoient des valeurs booléennes (TRUE ou FALSE).

Ils sont essentiels pour construire des conditions complexes dans les structures de contrôle de flux et pour filtrer les données.

1. **ET Logique (& et &&) :**
    - `x & y` : Renvoie `TRUE` si `x` ET `y` sont tous deux `TRUE`. Opère élément par élément dans le cas des vecteurs.
    - `x && y` : Similaire à `&`, mais il n'évalue que le premier élément de chaque vecteur. Utilisé principalement dans les structures de contrôle où une seule comparaison est nécessaire.

```
# Définition des variables
x <- TRUE
y <- FALSE

# Utilisation de & (ET logique)
resultat_et <- x & y  # Renvoie FALSE car y est FALSE
print(resultat_et)
```

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

```
# Utilisation de && (ET logique, mais ne vérifie que le premier élément)
resultat_et_court <- x && y  # Renvoie également FALSE
print(resultat_et_court)
```

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

2. **OU Logique (| et ||) :**

- `x | y` : Renvoie `TRUE` si `x` OU `y` est `TRUE`. Comme `&`, il opère élément par élément pour les vecteurs.
- `x || y` : Similaire à `|`, mais évalue uniquement le premier élément de chaque vecteur.

```
# Définition des variables
x <- TRUE
y <- FALSE

# Utilisation de | (OU logique)
resultat_ou <- x | y  # Renvoie TRUE car x est TRUE
print(resultat_ou)
```

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

```
# Utilisation de || (OU logique, mais ne vérifie que le premier élément)
resultat_ou_court <- x || y  # Renvoie également TRUE
print(resultat_ou_court)
```

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

3. **NON Logique (`!`) :**
   - `!x` : Renvoie `TRUE` si `x` est `FALSE` et vice versa. Utilisé pour inverser une condition.

```
# Définition de la variable
x <- TRUE

# Utilisation de ! (NON logique)
resultat_non <- !x  # Renvoie FALSE car x est TRUE
print(resultat_non)
```

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

Autres Opérateurs et Fonctions Utiles

1. `%in%` : Test d'Appartenance

`%in%` est utilisé pour tester si les éléments d'un vecteur se trouvent dans un autre ensemble de valeurs. Ici, il renvoie un vecteur booléen indiquant pour chaque élément de `x` s'il appartient à l'ensemble `c("D", "B")`.

```
x <- c("A", "B", "C", "D")
x %in% c("D", "B") # Renvoie un vecteur de valeurs logiques indiquant si chaque élément de x est dans c
```

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

2. `:` : Générateur de Séquences `:` est un opérateur simple pour générer des séquences de nombres entiers. Dans cet exemple, `1:5` génère une séquence de 1 à 5.

```r
variable <- 1:5 # Crée une séquence de nombres de 1 à 5
```

3. `[]` : Extraction d'Éléments

`[]` est utilisé pour extraire un ou plusieurs éléments d'une structure de données comme un vecteur, une matrice, ou un data frame. `variable[1:3]` renvoie les trois premiers éléments de `variable`.

```r
variable[1:3] # Extrait les trois premiers éléments de la séquence
```

```
## [1] 1 2 3
```

4. `[[]]` : Extraction d'Éléments d'une Liste ou d'un Data Frame

`[[...]]` est utilisé pour extraire un élément spécifique d'une liste ou d'un data frame. Par exemple, `maListe[[1]]` renvoie le premier élément de la liste `maListe`.

5. `$` : Extraction par Nom

`$` est utilisé pour extraire des éléments par leur nom d'une liste ou d'un data frame. Par exemple, `monDataFrame$colonne` renvoie la colonne nommée `colonne` de `monDataFrame`.

Les structures de données

En R, plusieurs types de structures de données sont utilisés pour stocker et manipuler les données. Chacune de ces structures a des caractéristiques et des usages spécifiques.

1. Vecteurs

- **Définition :** Les vecteurs sont des séquences d'éléments du même type (numérique, caractère, logique, etc.).
- **Création :** Utilisation de `c()` (combine) pour créer des vecteurs. Exemple : `vecteur <- c(1, 2, 3)`.
- **Usage :** Adaptés pour des opérations sur des séries d'éléments homogènes.

Rappels sur les vecteurs

```r
# Création d'un vecteur numérique
vecteur_num <- c(1, 2, 3, 4, 5)
print(vecteur_num)
```

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

```r
# Création d'un vecteur de caractères
vecteur_char <- c("un", "deux", "trois", "quatre", "cinq")
print(vecteur_char)
```

```
## [1] "un"     "deux"   "trois"  "quatre" "cinq"
```

```r
# Création d'un vecteur logique
vecteur_logique <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
print(vecteur_logique)
```

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

```r
# Accéder à un élément d'un vecteur
print(vecteur_num[3])  # Affiche la troisième valeur, i.e. 3
```

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

```r
# Longueur d'un vecteur
longueur <- length(vecteur_num)
print(paste("La longueur du vecteur est:", longueur))
```

```
## [1] "La longueur du vecteur est: 5"
```

```r
# Séquence de nombres
seq_num <- seq(1, 10, by = 2)  # Crée une séquence de 1 à 10 avec un pas de 2
print(seq_num)
```

```
## [1] 1 3 5 7 9
```

```r
# Opérations arithmétiques avec des vecteurs
vecteur_a <- c(1, 2, 3)
vecteur_b <- c(4, 5, 6)

somme <- vecteur_a + vecteur_b
print(somme)
```

```
## [1] 5 7 9
```

```r
# Combinaison de vecteurs
vecteur_combine <- c(vecteur_a, vecteur_b)
print(vecteur_combine)
```

```
## [1] 1 2 3 4 5 6
```

```r
# Utilisation de fonctions avec des vecteurs
moyenne <- mean(vecteur_num)
print(paste("La moyenne du vecteur est:", moyenne))
```

```
## [1] "La moyenne du vecteur est: 3"
```

```r
# Trouver le minimum et le maximum
print(paste("Minimum:", min(vecteur_num)))
```

```
## [1] "Minimum: 1"
```

```r
print(paste("Maximum:", max(vecteur_num)))
```

```
## [1] "Maximum: 5"
```

```r
# Trouver la somme et le produit
print(paste("Somme:", sum(vecteur_num)))
```

```
## [1] "Somme: 15"
```

```r
print(paste("Produit:", prod(vecteur_num)))
```

```
## [1] "Produit: 120"
```

Nouvelles fonctions

```r
print(vecteur_num)
```

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

```r
# Inverser un vecteur
inverse_vecteur_num <- rev(vecteur_num)
print(inverse_vecteur_num)
```

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

```r
# Modifier une valeur d'un vecteur
inverse_vecteur_num[2] <- 90
print(inverse_vecteur_num)
```

```
## [1]  5 90  3  2  1
```

```r
# Modifier plusieurs valeurs d'un vecteur
inverse_vecteur_num[c(3,4)] <- c(20, 200)
inverse_vecteur_num[2:4] <- c(78, 90, 30)
print(inverse_vecteur_num)
```

```
## [1]  5 78 90 30  1
```

```r
# Répétition d'un vecteur
x <- c("a", "b", "c")
x_repete <- rep(x, 10)
print(x_repete)
```

```
##  [1] "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a"
## [20] "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c"
```

```r
# Nommer les éléments d'un vecteur
identite <-  c(names = "roger", prenom = "toto")
print(identite)
```

```
##   names  prenom
## "roger"  "toto"
```

```r
print(identite["names"])
```

```
##   names
## "roger"
```

```r
# ajouter un element
vec <- c(1,2,90, 200)
print(vec)
```

```
## [1]   1   2  90 200
```

```r
vec <-  c(vec, 100)
print(vec)
```

```
## [1]   1   2  90 200 100
```

```r
print(vecteur_num)
```

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

```r
# Filtrage des éléments d'un vecteur
vecteur_num_filtre <- vecteur_num[vecteur_num >= 3]
print(vecteur_num_filtre)
```

```
## [1] 3 4 5
```

```r
# Filtrer les éléments supérieurs à la moyenne : mean(vecteur)
vecteur_num_filtre <- vecteur_num[vecteur_num >= mean(vecteur_num)]
print(vecteur_num_filtre)
```

```
## [1] 3 4 5
```

```r
# Utilisation de la fonction which() pour obtenir les indices de position
vecteur_num2 <-  seq(1,100, by = 7)
which(vecteur_num2 <= 50)
```

```
## [1] 1 2 3 4 5 6 7 8
```

```r
vecteur_num2[which(vecteur_num2 <= 50)]
```

```
## [1]  1  8 15 22 29 36 43 50
```

```r
# Trier un vecteur :  order() et sort() asc and decreasing
order(inverse_vecteur_num)
```

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

```r
sort(inverse_vecteur_num)
```

```
## [1]  1  5 30 78 90
```

```r
sort(inverse_vecteur_num, decreasing = TRUE)
```

```
## [1] 90 78 30  5  1
```

2. Facteurs

- **Définition :** Les facteurs sont utilisés pour stocker des données catégorielles.
- **Création :** Utilisation de `factor()` pour créer des facteurs. Exemple : `facteur <- factor(c("oui", "non", "oui"))`.
- **Usage :** Importants pour l'analyse statistique, en particulier pour la modélisation.

```r
# Création d'un facteur
genre_chaine <- factor(c("F", "M", "NB", "F", "M", "NB", "M"))

# Conversion du vecteur de chaînes en facteur
genre <- factor(genre_chaine)
```

- levels = c("1", "2", "3") définit l'ordre des niveaux dans le facteur. Les niveaux sont ordonnés selon l'ordre spécifié ici.
- labels = c("Pas d'accord", "Neutre", "D'accord") associe chaque niveau à une étiquette descriptive. Ainsi, au lieu de "1", "2", "3", les niveaux sont maintenant représentés par "Pas d'accord", "Neutre", et "D'accord".

```r
# Vecteur de données catégorielles
reponses <- c("1", "3", "2", "3", "1")

# Conversion en facteur avec labels et levels personnalisés
reponses_facteur <- factor(reponses,
                           levels = c("1", "2", "3"),
                           labels = c("Pas d'accord", "Neutre", "D'accord"))
```

3. Listes

- **Définition :** Les listes sont des collections d'éléments qui peuvent être de types différents.
- **Création :** Utilisation de `list()` pour créer des listes. Exemple : `liste <- list(nombre = 1, chaine = "R")`.
- **Usage :** Utiles pour stocker des données hétérogènes et complexes.

```r
# Création d'une liste
l <- list(1, "a", c(1, 2, 3))
print(l)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] 1 2 3
```

```r
# Les objets à mettre dans une liste
x <- c(45, 12, 56, 14, 16)
y <- c("Car", "Bike")
z <- matrix(1:12, ncol = 4)

# Création de la liste :list()
maliste <- list(x, y, z)
print(maliste)
```

```
## [[1]]
## [1] 45 12 56 14 16
##
## [[2]]
## [1] "Car"  "Bike"
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```r
# Nommer les éléments de la liste : names()
names(maliste) <- c("x", "y", "Z")
print(maliste)
```

```
## $x
## [1] 45 12 56 14 16
##
## $y
## [1] "Car"  "Bike"
##
## $Z
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
# Aperçu de la structure de notre liste : str()
str(maliste)
```

```
## List of 3
##  $ x: num [1:5] 45 12 56 14 16
##  $ y: chr [1:2] "Car" "Bike"
##  $ Z: int [1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
```

```
# Accéder aux éléments d'une liste : append()
a <- c(TRUE, FALSE, TRUE)
maliste <- append(maliste,a)
maliste <- list(maliste, a)
print(maliste)
```

```
## [[1]]
## [[1]]$x
## [1] 45 12 56 14 16
##
## [[1]]$y
## [1] "Car"  "Bike"
##
## [[1]]$Z
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## [[1]][[4]]
## [1] TRUE
##
## [[1]][[5]]
## [1] FALSE
##
## [[1]][[6]]
## [1] TRUE
##
##
## [[2]]
## [1]  TRUE FALSE  TRUE
```

```
# Modifier une liste "[[]]"
## modifier 1e élément
## ajouter un nouveau élément à la liste
## supprimer un élément de la liste
maliste <- list(x, y, z)
length(maliste)
```

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

```r
maliste[[2]] <- c("red", "blue", "marron")
maliste[[3]] <- NULL
print(maliste)
```

```
## [[1]]
## [1] 45 12 56 14 16
##
## [[2]]
## [1] "red"    "blue"   "marron"
```

```r
# Concater des listes : c()
maliste <- c(maliste, a)
print(maliste)
```

```
## [[1]]
## [1] 45 12 56 14 16
##
## [[2]]
## [1] "red"    "blue"   "marron"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] FALSE
##
## [[5]]
## [1] TRUE
```

4. Matrices

- **Définition :** Les matrices sont des collections d'éléments de même type organisés en un tableau à deux dimensions.
- **Création :** Utilisation de `matrix()` pour créer des matrices. Exemple : `matrice <- matrix(1:9, nrow = 3)`.
- **Usage :** Convient aux opérations mathématiques et statistiques impliquant des données en deux dimensions.

```r
# Création d'une matrice 3*3 : matrix()
mat <- matrix(1:9, ncol = 3, nrow = 3)
print(mat)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
# Remplissage par colonne : byrow
mat <- matrix(1:9, ncol = 3, nrow = 3, byrow = TRUE)
print(mat)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

```
# Nom lignes et colonnes : colnames(), rownames()
colnames(mat) <- c("Col1", "Col2", "Col3")
rownames(mat) <- c("Row1", "Row2", "Row3")
print(mat)
```

```
##      Col1 Col2 Col3
## Row1    1    2    3
## Row2    4    5    6
## Row3    7    8    9
```

```
# Combinaison de vecteur : cbind(), rbind()
x <- c(1,2,3)
y <- c(4, 5, 6)
z <- cbind(x,y)
print(z)
```

```
##      x y
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6
```

```
z <- rbind(x,y)
print(z)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y    4    5    6
```

```
# Accéder aux éléments d'une matrice "[]"
z[2,2]
```

```
## y
## 5
```

```
z[2,3]
```

```
## y
## 6
```

```
# Sélectioner toutes les lignes sauf la première : -1
a <- c(7,8,9)
z <- rbind(z,a)
z[-1,]
```

```
##   [,1] [,2] [,3]
## y    4    5    6
## a    7    8    9
```

```
z[,-1]
```

```
##   [,1] [,2]
## x    2    3
## y    5    6
## a    8    9
```

```
z <- rbind(x, a, y)
print(z)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## a    7    8    9
## y    4    5    6
```

```
# Modifier une élément
z[2,3] <- 10
print(z)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## a    7    8   10
## y    4    5    6
```

```
# Modifier un élément inf à 5
z[z < 5]
```

```
## [1] 1 4 2 3
```

```
# Opérations sur les matrices : t(), diag(), eigen(), solve()
x <- c(1,2,3)
y <- c(4, 5, 6)
z <- cbind(x,y)
print(z)
```

```
##      x y
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6
```

```
# Transposée de la matrice 'z'
t(z)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y    4    5    6
```

```r
# Diagnale de la matrice 'z'
z <- rbind(x, a, y)
diag(z) <- 0
print(z)
```

```
##   [,1] [,2] [,3]
## x    0    2    3
## a    7    0    9
## y    4    5    0
```

5. Data Frames

- **Définition :** Les data frames sont similaires aux matrices mais peuvent contenir différents types de données dans chaque colonne.
- **Création :** Utilisation de `data.frame()` pour créer des data frames. Exemple : `df <- data.frame(Nom = c("Alice", "Bob"), Age = c(23, 25))`.
- **Usage :** Très utilisés en analyse de données pour représenter et manipuler des ensembles de données de manière tabulaire.

```r
# Création d'un data frame
df <- data.frame(name=c("Alice", "Bob"), age=c(25, 30))
```

```r
# Creation d'un dataframe: data.frame()
df <-  data.frame(taille = c(167, 192, 173, 174, 172, 167, 171, 185, 163, 170),
                  poids = c(86, 74, 83, 50, 78, 66, 66, 51, 50, 55) ,
                  prog = c("Bac+2", "Bac", "Master", "Bac", "Bac", "DEA", "Doctorat", NA, "Certificat",
                  sexe = c("H", "H", "F", "H", "H", "H", "F", "H", "H", "H"))
print(df)
```

```
##    taille poids       prog sexe
## 1     167    86      Bac+2    H
## 2     192    74        Bac    H
## 3     173    83     Master    F
## 4     174    50        Bac    H
## 5     172    78        Bac    H
## 6     167    66        DEA    H
## 7     171    66   Doctorat    F
## 8     185    51       <NA>    H
## 9     163    50 Certificat    H
## 10    170    55        DES    H
```

Fonction usuelles sur un dataframe

Les fonctions mentionnées sont couramment utilisées pour obtenir des informations sur un data frame en R. Voici une explication de chacune d'entre elles : l()‘ renvoie par défaut les 6 dernières lignes.

1. **`length(df)` :** renvoie le nombre de colonnes dans le data frame `df`.

```r
length(df)
```

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

2. **`dim(df)` :** renvoie les dimensions du data frame sous forme de vecteur. Le premier élément est le nombre de lignes et le second le nombre de colonnes.

```r
dim(df)
```

```
## [1] 10  4
```

3. **`class(df)` :** renvoie la classe de l'objet. Pour un data frame, cela renverra `"data.frame"`.

```r
class(df)
```

```
## [1] "data.frame"
```

4. **`str(df)` :** donne la structure du data frame, y compris le type de chaque colonne, et affiche les premières entrées de chaque colonne. Utile pour un aperçu rapide de la structure des données.

```r
str(df)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ taille: num  167 192 173 174 172 167 171 185 163 170
##  $ poids : num  86 74 83 50 78 66 66 51 50 55
##  $ prog  : chr  "Bac+2" "Bac" "Master" "Bac" ...
##  $ sexe  : chr  "H" "H" "F" "H" ...
```

5. **`attributes(df)` :** renvoie une liste des attributs de l'objet `df`, tels que les noms, les rangs des lignes et des colonnes, et d'autres métadonnées.

```r
attributes(df)
```

```
## $names
## [1] "taille" "poids"  "prog"   "sexe"
##
## $class
## [1] "data.frame"
##
## $row.names
##  [1]  1  2  3  4  5  6  7  8  9 10
```

6. **`View(df)` :** ouvre le data frame `df` dans une fenêtre de visualisation dans RStudio (notez que cette fonction n'est utile que dans un environnement interactif comme RStudio).

```r
# View(df)
```

7. **`summary(df)` :** fournit un résumé statistique du data frame, y compris des statistiques telles que la moyenne, le minimum, le maximum, et la médiane pour les colonnes numériques, et la fréquence des niveaux pour les colonnes factorielles.

```r
summary(df)
```

```
##      taille          poids          prog               sexe
##  Min.   :163.0   Min.   :50.0   Length:10          Length:10
##  1st Qu.:167.8   1st Qu.:52.0   Class :character   Class :character
##  Median :171.5   Median :66.0   Mode  :character   Mode  :character
##  Mean   :173.4   Mean   :65.9
##  3rd Qu.:173.8   3rd Qu.:77.0
##  Max.   :192.0   Max.   :86.0
```

8. **`head(df, 6)` :** renvoie les 6 premières lignes du data frame. Si le second argument est omis, `head()` renvoie par défaut les 6 premières lignes.

```r
head(df, 6)
```

```
##   taille poids   prog sexe
## 1    167    86  Bac+2    H
## 2    192    74    Bac    H
## 3    173    83 Master    F
## 4    174    50    Bac    H
## 5    172    78    Bac    H
## 6    167    66    DEA    H
```

9. **`tail(df, 2)` :** renvoie les 2 dernières lignes du data frame. Si le second argument est omis, `tail` renvoie par défaut les 6 dernières lignes.

```r
tail(df, 2)
```

```
##    taille poids       prog sexe
## 9     163    50 Certificat    H
## 10    170    55        DES    H
```

```r
# Sélection par position ou nom
df[4, 2]  # par position
```

```
## [1] 50
```

```r
df[4, "poids"] # par position ou nom
```

```
## [1] 50
```

```r
df$poids[4] # sélection en utilisant l'opérateur '$'
```

```
## [1] 50
```

```r
# Ajouter des variables : cbind()
nom <- c("Benjamin", "Hugo", "Emma", "Alex", "Tom", "Axel", "Alice", "Martin", "Robin", "Enzo")
grade <- c("A", "A", "C", "B", "B", "B", "C", "A", "A", "A")

df <- data.frame(df,
                 names = nom, notes = grade)
print(df)
```

```
##    taille poids       prog sexe    names notes
## 1     167    86      Bac+2    H Benjamin     A
## 2     192    74        Bac    H     Hugo     A
## 3     173    83     Master    F     Emma     C
## 4     174    50        Bac    H     Alex     B
## 5     172    78        Bac    H      Tom     B
## 6     167    66        DEA    H     Axel     B
## 7     171    66   Doctorat    F    Alice     C
## 8     185    51       <NA>    H   Martin     A
## 9     163    50 Certificat    H    Robin     A
## 10    170    55        DES    H     Enzo     A
```

```r
# Creer la variable IMC <- poids / taille^2
df$IMC <- df$poids / df$taille**2
```

```r
# Afficher la distrbution de la variable 'sexe' : table()
table(df$sexe)
```

```
##
## F H
## 2 8
```

```r
# Ordonner les lignes : order() en fonction d'une variable
df_order <- df[order(df$taille),]
```

Famille de fonctions “apply”

La famille de fonctions “apply” en R est un ensemble puissant d’outils pour appliquer des fonctions de manière efficace et concise à des structures de données. Ces fonctions sont optimisées pour différentes structures et types de données, et elles facilitent l’écriture de code propre et lisible

La famille de fonctions “apply” en R est un ensemble puissant d’outils pour appliquer des fonctions de manière efficace et concise à des structures de données. Ces fonctions sont optimisées pour différentes structures et types de données, et elles facilitent l’écriture de code propre et lisible. Voici une explication de chaque fonction de la famille “apply” mentionnée dans votre script :

1. `apply`

- Utilisée principalement avec des matrices ou des data frames.
- `apply(df[,c("taille", "poids", "IMC")], 2, mean)` : Cette commande calcule la moyenne de chaque colonne (`2` indique que l’opération est effectuée colonne par colonne) dans les colonnes “taille”, “poids” et “IMC” du data frame `df`.
- `apply(df_poids, 1, mean)` : Calcule la moyenne de chaque ligne (indiqué par `1`) dans le data frame `df_poids`.

```r
# apply
apply(df[,c("taille", "poids", "IMC")], 2, mean)
```

```
##       taille        poids          IMC
## 1.734000e+02 6.590000e+01 2.205107e-03
```

```r
df_poids <- data.frame(poids1 = c(130, 170), poids2 = c(90, 50))
apply(df_poids, 1, mean)
```

```
## [1] 110 110
```

2. `lapply`

- Applique une fonction à chaque élément d'une liste et renvoie une liste.
- `lapply(maliste, mean)` : Calcule la moyenne de chaque élément (sous-liste) de la liste `maliste`.

```r
# lapply
maliste <- list(c(1:10), c(11:20), c(24:30))
lapply(maliste, mean)
```

```
## [[1]]
## [1] 5.5
##
## [[2]]
## [1] 15.5
##
## [[3]]
## [1] 27
```

3. `sapply`

- Semblable à `lapply`, mais simplifie le résultat en un vecteur ou un array, si possible.
- `sapply(maliste, mean)` : Comme `lapply`, mais renvoie un vecteur des moyennes au lieu d'une liste.

```r
# sapply : mean
sapply(maliste, mean)
```

```
## [1]  5.5 15.5 27.0
```

4. `mapply`

- Version multivariée de `sapply`.
- `mapply(median, df[,c("taille", "poids")])` : Applique la fonction `median` aux colonnes "taille" et "poids" du data frame `df`. Contrairement à `sapply` et `lapply`, `mapply` peut travailler avec plusieurs vecteurs ou listes simultanément.

```r
# mapply : median taille et poids
mapply(median, df[,c("taille", "poids")])
```

```
## taille  poids
##  171.5   66.0
```

5. `tapply`

- Utilisée pour appliquer une fonction à des sous-ensembles de données.
- `tapply(df$taille, df$sexe, mean)` : Calcule la moyenne de la variable "taille" séparément pour chaque groupe défini par la variable catégorielle "sexe" dans le data frame `df`.

```r
# tapply / by : taille et sexe
tapply(df$taille, df$sexe, mean)
```

```
##      F      H
## 172.00 173.75
```

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 **exercices2.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_exercices2.R** pour voir les solutions. Analysez les solutions pour comprendre les méthodes et logiques utilisées.