R Studio

Chapitre Introductif

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

April 18, 2022

"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison

Table de matières

Objectifs du cours

R est le langage de programmation statistique le plus utilisé. C'est le choix n°1 des data scientists et des analystes.

Durant ce cours, nous apprendrons les bases de R, nous verrons comment créer des programmes qui stockent et manipulent des données, ainsi qu'effectuer des tâches d'analyse de données en utilisant divers ensembles de données et visualiser les résultats à l'aide de graphiques et de diagrammes.

Attention !

Les compétences acquises pendant ce cours peuvent être appliquées dans n'importe quel domaine qui travaille avec des données, y compris la finance, la science des données, l'apprentissage automatique, et le nucléaire, etc.

Objectifs du chapitre

Dans ce chapitre, nous allons apprendre les types de variables, à rédiger un commentaire, à réaliser les opérations mathématiques de base.

À la fin de ce chapitre, vous serez en mesure :

- de saisir et d'afficher le contenu d'une variable à l'écran.
- de rédiger un commentaire
- de distinguer les différents types de variables
- de réaliser les opérations mathématiques de base

Prérequis du chapitre

Attention !

Pas de prérequis.

Premier programme

Commençons par écrire un programme simple qui affiche du texte :

print("R est facile !") $\Longrightarrow$ [1] "R est facile !"

La fonction **print()** permet **d'éditer du texte**.
Elle est suivie de parenthèses, qui incluent le texte que l'on souhaite éditer, entre guillemets.

Attention !

Notez que la sortie comprend également un numéro avant : c'est le numéro de ligne de la sortie.

Exercice d'application 1

Premier programme

Utilisez les mots (print, Salut, Bonjour, input, read) pour créer un programme qui affiche "Salut".

..... (" ..... ")

Commentaires

Les **commentaires** sont utilisés pour **expliquer votre code**. Ils sont ignorés lors de l'exécution de votre programme.
Vous pouvez **créer des commentaires** dans R en utilisant **#**.

```
# outputs "Hello, World!"
print("Hello, World!")
```

Tout ce qui vient après le symbole # sur cette ligne est ignoré.

Attention !

Les commentaires sont utiles, car ils aident à lire et à comprendre des segments de code plus importants, et expliquent ce que fait le code.

Exercice d'application 2

Commentaires

Remplissez les espaces vides pour créer un programme qui affiche "Apprendre à coder" et inclut un commentaire avant l'instruction d'affichage.

.....	mon premier programme
.....	("Apprendre à coder")

Variables 1

En général, chaque programme R traite des données. **Les variables vous permettent de stocker et de manipuler des données**. Les **variables** ont un **nom** et une **valeur**.

Par exemple, créons une variable nommée x et affectons la valeur 42 à cette variable :

$x = 55$

Notez que nous avons utilisé **l'opérateur d'affectation** = pour attribuer une valeur à la variable.

Variables 2

Maintenant, nous pouvons utiliser print pour afficher la valeur stockée dans x.

print(x) $\Longrightarrow$ [1] 55

Attention !

Les noms des variables doivent commencer par une lettre ou un point et peuvent inclure des lettres, des chiffres et un point ou des caractères de soulignement.

Exercice d'application 3

Variables

Remplissez les espaces vides (avec Noemi, print, var, nom, #, =) pour créer une variable "nom", lui attribuer la valeur "Noemi" et l'afficher.

nom [.....] " [.....] "
print([.....])

L'opérateur <- 1

Une **façon plus préférée d'attribuer des valeurs** aux variables dans R est d'utiliser **l'opérateur ¡-** vers la gauche :

x <- 55 print(x) $\Longrightarrow$ [1] 55

L'opérateur <- 2

Nous pouvons avoir plusieurs variables dans notre programme avec différentes valeurs, et leur attribuer de nouvelles valeurs au cours de notre programme :

```
tva <- 0.2
nom <- "Eric"
message <- "Bonjour Abidjan !"
tva <- 0.055
print(tva) ⟹ [1] 0.055
print(nom) ⟹ [1] "Eric"
```

Attention !

R est sensible à la casse, donc, par exemple, Nom et nom sont deux variables différentes.

Exercice d'application 4

L'opérateur <-

Quelle est la sortie de ce code ?

```
x <- 7
y <- 5
x <- y
print(x)
```

1. 7
2. 5
3. 2
4. 12

Les types de données 1

Les variables peuvent stocker différents types de données, comme des **entiers (integers)**, des **décimales (decimals)**, du **texte (text)**.
Dans R, vous n'avez pas besoin de spécifier le type d'une variable. Au lieu de cela, R obtient automatiquement le type à partir de la valeur à laquelle elle est affectée.

```
# numeric
var1 <- 3.14
#integer
var2 <- 88L
# text
var3 <- "hello"
```

print(var1) $\Longrightarrow$ [1] 3.14
print(var2) $\Longrightarrow$ [1] 88
print(var3) $\Longrightarrow$ [1] "hello"

Les types de données 2

Notez que pour les nombres entiers, nous devons faire précéder la valeur de la **lettre L**. Cela **oblige R à stocker la valeur comme un nombre entier**.

Vous pouvez également attribuer des nombres sans la lettre L, ce qui les stockera comme des numériques.

Attention !

L'utilisation de la notation L garantit que R utilise la valeur comme un nombre entier, qui prend moins de place en mémoire que les valeurs numériques, car les valeurs numériques peuvent également avoir des points décimaux.

Exercice d'application 5

Les types de données

Laquelle des valeurs suivantes sera stockée sous forme de nombre entier (integer) ?

1. 144
2. "9"
3. 1.70
4. 43L

Strings 1

Dans R, le **texte** est stocké sous forme de **string** (chaîne). Nous avons vu des **strings (chaînes de caractères)** dans nos leçons précédentes : elles sont **entourées** soit de **guillemets simples**, soit de **guillemets doubles** :

x <- "Bonjour" $\Longrightarrow$ [1] "Bonjour"
y <- 'Maman' $\Longrightarrow$ [1] "Maman"

Les guillemets que vous utilisez ne font aucune différence. Les deux créent une chaîne de caractères. Veillez simplement à ouvrir et fermer le texte en utilisant le même guillemet simple ou double.

Strings 2

Si vous devez **utiliser un guillemet dans le string**, vous pouvez l'échapper en utilisant un **backslash** (une barre oblique inverse) :

message <- "This is called \" escaping\"."
print(message) $\Longrightarrow$ [1] "This is called \"escaping\"."

Notez que lors de l'impression de la valeur, les backslashs seront également affichés. Vous pouvez utiliser **la fonction cat** au lieu de print **pour afficher la valeur sans les backslashs**.

message <- "This is called ëscaping:"
cat(message) $\Longrightarrow$ This is called "escaping".

Strings 3

Attention !

Par rapport à print, cat n'affiche pas les numéros de ligne de la sortie entre crochets.

Exercice d'application 6

Strings

Remplissez les espaces vides (text, <-, \, ', cat, ") pour créer une chaîne de caractères valide et l'afficher.

text [.....] ' welcome [.....]
print([.....])

Arithmétiques de base 1

R supporte les opérations arithmétiques de base.
Vous pouvez les utiliser pour des variables ou des valeurs.

```
x <- 11
y <- 4
# addition
print(x+y) ⟹ [1] 15
# soustraction
print(x-y) ⟹ [1] 7
# multiplication
print(x*y) ⟹ [1] 44
```

Arithmétiques de base 2

```
# division
print(x/y) ⟹ [1] 2.75
# puissance
print(x ∧ y) # or x**y ⟹ [1] 14641
# modulus (reste de la division)
print(x%%y) ⟹ [1] 3
# integer division
print(x%/%y) ⟹ [1] 2
```

Attention !

Notez, que R supporte deux types de division : la division et la division entière. La première version produit une décimale, tandis que la seconde produit un nombre entier.

Exercice d'application 7

Arithmétiques de base

Quelle est la sortie de ce code ?

```
a <- 7
b <- 2
print(a%%b)
```

1. 2
2. 3
3. 4

Fonctions mathématiques 1

R dispose également des fonctions permettant d'effectuer des tâches mathématiques.
Par exemple, **les fonctions min** et **max** peuvent être utilisées pour trouver le **minimum et le maximum** d'un ensemble donné de nombres :

```
a <- 23
b <- 5
# minimum
print(min(a, b)) ⟹ [1] 5
# maximum
print(max(a, b)) ⟹ [1] 23
```

Fonctions mathématiques 2

Vous pouvez également utiliser plus de deux nombres avec les fonctions min et max - il suffit de les séparer par des virgules.

De même, R dispose d'une **fonction sqrt** intégrée, qui est utilisée pour trouver **la racine carrée** d'un nombre donné :
print(sqrt(49)) $\Longrightarrow$ [1] 7

Attention !

N'oubliez pas que vous devez utiliser des parenthèses pour entourer les nombres dans les fonctions. Nous en apprendrons davantage sur les fonctions dans le prochain chapitre.

Exercice d'application 8

Fonctions mathématiques

Remplissez les espaces vides pour trouver et affichez le maximum des 3 variables données.

```
x <- 4
y <- 13
z <- 11
res <- [.....] (x [.....] y, [.....] )
print(res)
```

Les variables booléennes 1

Le **booléen** est un **autre type de données dans R**. Il peut avoir **l'une des deux valeurs** suivantes : (TRUE ou FALSE, 1 ou 0).

Les booléens sont créés lorsque nous comparons des valeurs.

```
x <- 23
print(x > 44)
```

Dans le code ci-dessus, nous avons utilisé l'opérateur greater than $>$ pour comparer x à la valeur 44.

Attention !

Le résultat de la comparaison est un booléen dont la valeur est FAUX, car x n'est pas supérieur à 20.

Exercice d'application 9

Les variables booléennes

Sélectionnez les 2 valeurs possibles d'un booléen.

1. FALSE
2. yes
3. TRUE
4. no
5. "un"

Les opérateurs relationnels 1

R prend en charge les **opérateurs relationnels** suivants, utilisés pour les **comparaisons** :

$>$ supérieur à
$<$ inférieur à
$<=$ inférieur ou égal à
$>=$ supérieur ou égal à
$==$ égal
$!=$ non égal (not equal)

Les opérateurs relationnels 2

```
x <- 42
print(x >= 8) ⟹ [1] TRUE
print(x < 24) ⟹ [1] FALSE
print(x == 42) ⟹ [1] TRUE
print(x != 42) ⟹ [1] FALSE
```

Attention !

Notez que vous devez utiliser **deux signes égal (==)** pour vérifier **l'égalité**, car un **seul signe égal (=)** est **l'opérateur d'affectation**.

Exercice d'application 10

Les opérateurs relationnels

Quel est le résultat de ce code ?

```
num <- 15
val <- num-6
print((num%/%val)>=2)
```

1. FALSE
2. 2
3. 1
4. TRUE

Input 1

R vous permet de prendre les entrées (user input) de l'utilisateur et de les stocker dans une variable.

La fonction readLines est utilisée **pour lire chaque ligne donnée en entrée séparément, ce qui en fait un moyen pratique de lire des entrées multiples**.

Input 2

input <- readLines('stdin')
Maintenant, **input est une variable** qui **contient chaque ligne de l'entrée donnée séparément**.
Notez le **paramètre 'stdin'** - il est nécessaire **pour lire l'entrée standard**.

Afin **d'accéder aux entrées**, nous devons **fournir le numéro de la ligne à laquelle nous voulons accéder en utilisant des crochets** :
print(input[1]) $\Longrightarrow$ Cela affichera la première entrée

Input 3

Attention !

La variable est en fait un **vecteur (vector)**. Nous en apprendrons davantage sur les vecteurs dans les prochains chapitres. Pour l'instant, retenez simplement la syntaxe d'accès à ses éléments.

N'exécutez pas readLines('stdin') dans votre R Studio, sinon vous aurez du mal à la stopper.

Exercice d'application 11

Input

Remplissez les blancs (avec read, stdin, 1, 2, line, 3, print) pour saisir un input et sortir la deuxième ligne.

x ¡- readLines(' ..... ')
..... (x[.....])

Inputs 4

Les entrées (inputs) sont des strings (chaînes de caractères), par défaut. Ainsi, afin d'effectuer des opérations numériques avec une entrée, nous devons **la convertir en un nombre** :

```
input <- readLines('stdin')
x <- input[1]
x <- as.integer(x)
print(x*2)
```

Maintenant, x est un entier et nous pouvons l'utiliser comme un nombre.

Attention !

Comme R lit chaque ligne comme input, n'oubliez pas de séparer les entrées ou saisies (inputs) multiples en utilisant une nouvelle ligne.

Exercice d'application 12

Multiples Inputs

readLines() stocke l'entrée (la saisie ou (l'input) comme un vecteur de :

1. newlines
2. chaînes de caractères (strings)
3. entiers (integers)

Output 1

Comme nous l'avons vu dans les parties précédentes, nous pouvons **afficher** des valeurs en utilisant les fonctions **print** et **cat**.

x <- "Papa"
print(x) $\Longrightarrow$ [1] "Papa"
cat(x) $\Longrightarrow$ Papa

Output 2

Vous pouvez utiliser le symbole spécial \n pour ajouter de nouvelles lignes au texte.

x <- "hello\nthere!"
print(x) $\Longrightarrow$ [1] "hello\nthere!"
cat(x) $\Longrightarrow$
hello
there!

Vous pouvez avoir plusieurs symboles\n dans votre texte.

Attention !

Notez que la fonction **cat affiche le saut de ligne** dans la sortie (output), tandis que la fonction **print affiche le caractère \n sans le saut de ligne**.

Exercice d'application 9

Output

Lequel des symboles représente une nouvelle ligne ?

1. \"
2. \t
3. \r
4. \n

Quiz 1

1) Affectation et affichage

Complétez (avec =, var, out, prix, print, cat, 23) de manière à affecter 23 à la varaible prix et l'afficher.

.....	<-	.....
.....	(prix)	

Quiz 2

2) Output

Quel est l'output du programme ci-dessous ?

```
x <- 4
y <- 1
x <- x-y
print(x%%2)
```

.....

Quiz 3

3) Output

Quel est l'output du programme ci-dessous ?

```
a <- 8
b <- a/3
print(b < 2)
```

1. TRUE
2. 2
3. FALSE
4. 0

Quiz 4

4) Conversion et affichage

Complétez (avec in, for, readLines, as, integers, convert, input) pour prendre une entrée (input), la convertir en un nombre entier et afficher son double.

input <- [.....] ('stdin')
x <- [.....] . [.....] (input[1])
print(x*2)

Quiz 5

5) Output

Quel est l'output du programme ci-dessous ?

```
x <- 6
y <- x-2
z <- min(x, y)
print(sqrt(z))
```

.....

R Studio

Chapitre I : Programmer en R

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

April 19, 2022

"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons apprendre à manipuler les instructions conditionnelles, les boucles et les fonctions.

À la fin de ce chapitre, vous serez en mesure :

- de créer et de manipuler les intructions conditionnelles (if/else if/else, switch),
- de vous servir des boucles for, while, ainsi que des mots clés break et next,
- de créer et manipuler les fonctions avec ou sans return

Prérequis du chapitre

Attention !

Avoir suivi le chapitre précédent.

Les instructions conditionnelles 1

Dans de nombreuses situations, vous devez prendre une décision en fonction d'une condition. Pour cela, vous pouvez utiliser l'instruction **if**.

```
x <- 10
if(x < 38) {
      print("x est plus petit que 38")
}
```

$\Longrightarrow$ [1] "x est plus petit que 38"

Les instructions conditionnelles 2

Comme vous pouvez le voir, le mot clé **if** est suivi de la condition entre parenthèses et d'un bloc de code entre accolades, **qui est exécuté si la condition est VRAIE (TRUE)**.

Attention !

Si la condition de l'instruction if est FAUX (FALSE), le code entre accolades ne sera pas exécuté.

Exercice d'application 1

Les instructions conditionnelles

Remplissez les espaces vides pour créer une instruction if valide.

logged_in = TRUE
[.....] (logged_in == TRUE [.....] {
    print("Welcome")
[.....]

L'instruction Else 1

Si vous devez exécuter du code lorsque **la condition d'une instruction if est FAUX**, vous pouvez utiliser **une instruction else** :

```
x <- 32
if(x >= 56) {
      print("x is big")
} else {
      print("x is less than 56") ⟹ [1] "x is less than 56"
}
```

L'instruction Else 2

Si vous avez besoin de **plusieurs vérifications**, vous pouvez utiliser **plusieurs instructions else if**.
Par exemple, nous allons afficher la version anglaise du numéro donné :

```
num <- 3
if(num == 1) {
      print("One")
} else if(num == 2) {
      print("Two")
} else if (num == 3) {
      print("Three") ⟹ [1] "Three"
} else {
      print("Something else")
}
```

L'instruction Else 3

Attention !

Vous pouvez avoir autant de déclarations "else if" que vous le souhaitez.

Exercice d'application 2

L'instruction Else

Quel est le résultat de ce code ?

```
x <- 7
if(x == 0) {
    print("Zéro")
} else if(x %% 2 == 0) {
    print("Even")
} else {
    print("Impair")
}
```

1. Impair
2. Zéro
3. Pair

Les opérateurs logiques 1

Les opérateurs logiques vous permettent de combiner plusieurs conditions.

L'opérateur logique & **(AND)** permet de combiner deux conditions et **renvoie VRAI uniquement si les deux conditions sont VRAIES**.

```
x <- 6
y <- 2
if(x > y & x < 10) {
    print("Yes") ⟹ [1] "Yes"
}
```

Les opérateurs logiques 2

De même, l'opérateur logique | **(OR)** renvoie VRAI **si l'une de ses conditions est VRAIE** :

```
x <- 6
y <- 2
if(x > y | x > 43) {
      print("Yes") ⟹ [1] "Yes"
}
```

Les opérateurs logiques 3

L'opérateur logique **!** **(NOT)** renvoie **l'opposé de la condition donnée** :

x <- TRUE print(!x) $\Longrightarrow$ [1] FALSE

Attention !

Vous pouvez combiner plusieurs conditions à l'aide des opérateurs logiques et regrouper les conditions à l'aide de parenthèses, tout comme les opérations mathématiques.

Exercice d'application 3

Les opérateurs logiques

Quel est le résultat de cette condition ?

$(((15 > 4)\ \&\ (8 < 9))\ |\ (4 > 6))$

1. FALSE
2. TRUE

L'opérateur switch 1

Vous vous souvenez que nous avions écrit un code pour afficher la version anglaise d'un nombre donné ? Il utilisait plusieurs instructions else if pour vérifier le nombre.
R fournit une **instruction switch pour tester une expression par rapport à une liste de valeurs** et rend le **code beaucoup plus court**, comparé à l'utilisation des instructions else if.

```
num <- 3
result <- switch(
num,
"One",
"Two",
"Three",
"Four"
)
print(result) ⟹ [1] "Three"
```

L'opérateur switch 2

L'instruction switch prend son premier paramètre et renvoie la valeur dont l'index correspond à ce numéro.
Au lieu de l'index, vous pouvez également fournir les valeurs à comparer et les valeurs à (afficher) en cas de correspondance :

```
x <- "c"
result <- switch(
x,
"a" = "One",
"b" = "Two",
"c" = "Three",
"d" = "Four"
) print(result) ⟹ [1] "Three"
```

L'opérateur switch 3

Attention !

Vous pouvez avoir autant de cas que vous le souhaitez. N'oubliez pas de les séparer par des virgules.

Exercice d'application 4

L'opérateur switch

Quel est le résultat de ce code ?

```
x <- 2
choix <- switch(
x,
"Coffee",
"Tea",
"Water"
)
```

1. Tea
2. 2
3. Coffee
4. Water

Les boucles 1

Les boucles vous permettent **de répéter un bloc de code jusqu'à ce qu'une condition donnée soit VRAIE**.
La boucle while a la syntaxe suivante :

```
while (condition) {
code to run
}
```

Les boucles 2

Utilisons la boucle while pour afficher les chiffres de 1 à 9 :

```
i <- 1
while (i < 10) {
    print(i)
i <- i + 1
}
```

Le code ci-dessus vérifie si i est inférieur à 10, affiche sa valeur, puis l'incrémente de 1. Cela signifie que la boucle affichera les nombres 1 à 9 et s'arrêtera lorsque i atteindra la valeur 10.

Chaque fois que l'ordinateur parcourt une boucle, il s'agit d'une **itération**.

Les boucles 3

Attention !

Il est important de **modifier la valeur de la condition pendant les itérations** de la boucle while, car si vous ne le faites pas, vous obtiendrez **une boucle infinie**, car **la condition restera toujours VRAIE**.

Exercice d'application 5

Les boucles

Combien de nombres la boucle suivante affichera-t-elle ?

```
x <- 10
while(x > 0) {
      print(x)
      x <- x - 2
}
```

.....

La boucle for 1

Une autre boucle que R fournit est la boucle for.
Elle est utilisée pour itérer sur une séquence donnée.

```
for (x in 1:10) {
      print(x)
}
```

R nous permet de créer une séquence de nombres en utilisant deux points et en spécifiant les limites inférieure et supérieure. Dans le code ci-dessus, la séquence comprendra les nombres 1 à 10.

La boucle for 2

Au cours de chaque itération de la boucle for, la variable x prendra la valeur du nombre suivant dans la séquence, ainsi, la sortie résultante sera les nombres 1 à 10.

Attention !

La boucle for est également utilisée pour itérer sur des listes et des vecteurs. Nous les découvrirons dans les prochains chapitres.

Exercice d'application 6

La boucle for

Remplissez les espaces vides pour n'afficher que les nombres pairs dans la séquence de 1 à 100. Un nombre est pair si, lorsqu'il est divisé par 2, le reste est égal à 0.

```
[.....] (x in 1 [.....] 100) {
    [.....] (x%%2 == 0) {
    print( [.....] )
    }
}
```

L'instruction break 1

L'instruction **break** vous permet **d'arrêter une boucle**.

```
i <- 8
while(i > 0) {
    print(i)
    i <- i - 1
    if(i == 4) {
        break
    }
}
```

Le code ci-dessus arrêtera la boucle lorsque i atteindra la valeur 4.

L'instruction break 2

Cela peut être particulièrement utile lorsque vous devez prendre plusieurs entrées de l'utilisateur et vous arrêter au cas où une entrée spécifique est donnée.

Attention !

L'instruction break peut également être utilisée avec une boucle for.

Exercice d'application 7

L'instruction break

Combien de chiffres le code suivant affichera-t-il ?

```
i <- 1
while(i < 5) {
    print(i)
    i <- i + 1
    if(i == 3) {
        break
    }
}
```

.....

L'instruction next 1

L'instruction **next** vous permet **de sauter une itération et de poursuivre l'exécution de la boucle à l'itération suivante**.

Par exemple, disons que nous voulons afficher tous les chiffres de 1 à 15, sauf 13 :

```
for(x in 1:15) {
    if(x == 13) {
        next
    }
    print(x)
}
```

L'instruction next 2

Notez que nous vérifions la condition de l'instruction suivante avant d'afficher la valeur.

Attention !

Tout comme l'instruction break, l'instruction next peut être utilisée avec les boucles while et for.

Exercice d'application 8

L'instruction next

Remplissez les blancs (avec while, in |, break, next, else, &, as) pour afficher tous les chiffres de 1 à 50, sauf les chiffres 10 et 20.

```
for ( x [ ..... ] 1:50 ) {
      if( x == 10 [ ..... ] x == 20 ) {
      [ ..... ]
      }
      print(x)
}
```

Les fonctions 1

Une **fonction** est un **bloc de code** qui peut être **appelé en utilisant son nom**.
Une fonction peut également prendre des paramètres en entrée (input) et renvoyer des valeurs.

R possède de nombreuses fonctions intégrées. Nous avons déjà vu certaines d'entre elles.
Par exemple, **print("Hello")** appelle la f**onction print avec le paramètre "Hello"**.

Les fonctions 2

Les paramètres sont transmis aux fonctions entre parenthèses.
Les fonctions peuvent avoir plusieurs paramètres, séparés par des virgules. Par exemple, **la fonction max peut prendre plusieurs paramètres et renvoyer le plus grand** :
res <- max(9, 2, 131, 34
print(res) $\Longrightarrow$ [1] 88

Attention !

Les fonctions peuvent être appelées plusieurs fois dans votre code et prendre différentes valeurs de paramètres.

Exercice d'application 9

Les fonctions

Remplissez les espaces vides (avec (,), y, ,,) pour appeler la fonction min avec les paramètres "x" et "y".

min [.....] x [.....] [.....] [.....]

Les fonctions definies par l'utilisateur 1

En plus des fonctions intégrées, vous pouvez également définir vos propres fonctions et les utiliser dans votre code.
Pour cela, il faut utiliser le mot-clé **function** et l'associer à un nom. Par exemple :

```
pow <- function(x, y) {
    result <- x ^ y
    print(result)
}
```

La fonction est nommée **pow**, elle prend **deux paramètres**, appelés **x** et **y**, et **affiche la valeur de x élevée à la puissance de y**.

Les fonctions definies par l'utilisateur 2

Après avoir défini notre fonction, nous pouvons l'appeler dans notre code comme suit :

pow(2, 3) $\Longrightarrow$ [1] 8
pow(9, 2) $\Longrightarrow$ [1] 81

Attention !

Les fonctions peuvent prendre un nombre quelconque de paramètres. N'oubliez pas de les séparer par des virgules.

Exercice d'application 10

Les fonctions definies par l'utilisateur

Remplissez les espaces vides pour créer une fonction appelée "square", qui prend un paramètre et affiche son carré.

```
[.....] [.....] [.....] ( x ) {
    print(x ∧ 2)
}
```

Le paramètre par défaut 1

Lorsque vous appelez une fonction, vous devez fournir des valeurs pour tous ses paramètres.

La spécification des valeurs par défaut des paramètres vous permet d'appeler une fonction avec une partie seulement de ses paramètres, tandis que les autres utilisent les valeurs par défaut fournies.

```
pow <- function(x, y=2) {
    result <- x ^ y
    print(result)
}
```

Le paramètre par défaut 2

Maintenant, nous pouvons appeler la fonction en utilisant **un seul paramètre** :

pow(4) $\Longrightarrow$ [1] 16

Dans ce cas, la valeur **2** que nous avons spécifiée **sera utilisée pour le deuxième paramètre**.

Le paramètre par défaut 3

Attention !

Paramètres Vs arguments

Les termes **"paramètre"** et **"argument"** sont souvent utilisés pour désigner les **informations transmises à une fonction**.

Un paramètre est **la variable figurant entre les parenthèses dans la définition de la fonction**.
Un argument est **la valeur qui est envoyée à la fonction lorsqu'elle est appelée**.

Ainsi, dans notre cas, **x et y sont les paramètres**, tandis que **leurs valeurs que nous fournissons lors de l'appel de la fonction sont les arguments**.

Exercice d'application 11

Le paramètre par défaut

Quelle est l'output de ce code ?

```
test <- function(a, b=3) {
      result <- (a+b)*3
      print(result)
}
test(1)
```

.....

L'instruction return 1

Dans la plupart des cas, nous voulons que la valeur calculée par notre fonction soit affectée à une variable, au lieu de simplement l'afficher.

Dans ce cas, nous pouvons utiliser **la fonction return pour renvoyer une valeur de notre fonction**.

Par exemple, réécrivons **notre fonction pow** de l'exemple précédent pour retourner la valeur résultante :

```
pow <- function(x, y=2) {
    result <- x ^ y
    return (result)
}
```

L'instruction return 2

Maintenant, nous pouvons l'appeler et affecter la valeur à une variable :

a <- pow(5)
print(a) $\Longrightarrow$ [1] 25

Attention !

La plupart des fonctions R renvoient des valeurs. Par exemple, les fonctions min/max/sqrt et d'autres fonctions intégrées renvoient le résultat de l'opération correspondante.

Exercice d'application 12

L'instruction return

Remplissez les espaces vides pour créer une fonction appelée "add" qui renvoie la somme de ses deux paramètres.

add [.....] function (a, b) {
    [.....] (a [.....] b)
}

Quiz 1

1) If/else if/else

Remplissez les espaces vides (avec 0, 100, Oui, Non, &, else, |, if) pour obtenir "Oui" si la valeur "prix" est comprise entre 0 et 100, "Non" si elle est supérieure à 100 et "Invalide" si elle est inférieure à 0.

```
[.....] (price >= 0 [.....] price <= 100) {
    print("Oui")
} else if (price > 100) {
    print(" [.....] ")
} [.....] {
    print("Invalide")
}
```

Quiz 2

2) Boucles

Créez une boucle valide pour sortir les nombres de 8 à 2.

```
[.....] (x in 8 [.....] 2) {
    print( [.....] )
}
```

Quiz 3

3) square root function

Remplissez les espaces vides pour créer une fonction appelée "x", qui prend deux paramètres et renvoie la racine carrée de leur somme.

```
[.....] <- [.....] (a [.....] b) {
    return ( [.....] (a [.....] b))
}
```

Quiz 4

4) function définie

Quel est l'output du programme suivant ?

```
calc <- function(a, b) {
      res <- 0
      for(x in a:b) {
            res <- res + x
      }
return(res)
}
print(calc(4, 7))
```

.....

Quiz 5

5) fonction définie

Quel est l'output du programme suivant ?

```
sum <- function(x) {
      res <- 1
      for(val in 1:x) {
            if(val == 3) {
                  break
      }
      res <- res * val
      }
      return(res)
}
sum(5)
```

.....

R Studio

Chapitre II : Les Structures des données

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

April 20, 2022

"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons apprendre à manipuler les strings, les vecteurs (vectors), les listes (lists), les matrices et les dataframes.

À la fin de ce chapitre, vous serez en mesure :

- de créer et de manipuler les strings,
- de vous servir des vecteurs et des lists
- de créer et de manipuler les matrices et les dataframes

Prérequis du chapitre

Attention !

Avoir suivi le chapitre précédent.

Les strings 1

Lorsque nous travaillons avec des **ensembles de données**, nous devons utiliser des **structures de données (data structures)** pour stocker et manipuler les données.

R fournit un certain nombre de structures de données qui offrent des fonctions pour manipuler et gérer les données qu'elles stockent.

Commençons par l'une des structures de données les plus simples que nous avons déjà utilisées : **les chaînes de caractères (strings)**.

Nous verrons comment pouvons manipuler les chaînes de caractères. **La fonction paste vous permet de combiner plusieurs chaînes de caractères en une seule**.

Les strings 2

t1 <- "Salut"
t2 <- "Noemi"
t3 <- "!"
result <- paste(t1, t2, t3)
print(result) $\Longrightarrow$ [1] "Salut Noemi !"

La **fonction paste utilise un espace comme séparateur** et peut prendre un nombre quelconque de paramètres.

Les strings 3

Vous pouvez **définir votre propre séparateur** à l'aide du paramètre **sep=** :

result <- paste(t1, t2, t3, sep ="-")
print(result) $\Longrightarrow$ [1] "Salut-Noemi-!"

Attention !

Le processus de combinaison des chaînes de caractères est appelé concaténation.

Exercice d'application 1

Les strings

Remplissez les espaces vides pour créer une fonction appelée "concat" qui prend deux paramètres et renvoie leur concaténation, sans séparateur.

```
concat <- [.....] (x, y) {
      return( [.....] (x, y, [.....] ="") )
}
```

Les fonctions toupper() et tolower()

La fonction **toupper** permet de convertir une chaîne de caractères **en majuscules**. De même, la fonction **tolower** permet de convertir une chaîne de caractères **en minuscules** :

```
txt <- "salut"
txt <- toupper(txt)
print(txt) ⟹ [1] "SALUT"

txt <- tolower(txt)
print(txt) ⟹ [1] "salut"
```

La fonction nchar()

Pour trouver **le nombre de caractères** d'une chaîne de caractères, vous pouvez utiliser la **fonction nchar** :

txt <- "salut"
print(nchar(txt)) $\Longrightarrow$ [1] 5

La fonction substr()

Une autre fonction utile est **substr**, qui permet **d'obtenir une sous-chaîne à partir d'une chaîne de caractères donnée en utilisant un index de début et de fin** :

txt <- "bonjour"
print(substr(txt, 4, 7)) $\Longrightarrow$ [1] "jour"

Attention !

Exercice d'application 2

La fonction substr() et nchar()

Quel est l'output du programme ci-dessous ?

```
txt <- "some text"
x <- substr(txt, 2, 6)
print(nchar(x))
```

.....

Vecteurs 1

Un vecteur est une structure de données de base dans R. Il s'agit **d'une séquence d'éléments du même type**.

Pour **créer un vecteur**, nous devons utiliser **la fonction c() et séparer les éléments par des virgules**.

names <- c("David", "Richard", "Marcel")

Vecteurs 2

Pour **accéder à un élément du vecteur**, nous devons nous y référer en utilisant s**on indice entre crochets**.
Par exemple, affichons le 3ème élément :

names <- c("David", "Richard", "Marcel") print(names[2]) $\Longrightarrow$
[1] "Marcel"

Attention !

Contrairement à d'autres langages de programmation, les indices en R commencent à partir de 1, au lieu de 0.

Exercice d'application 3

Vecteurs

Remplissez les espaces vides pour créer un vecteur de nombres et affichez le deuxième élément.

n <- [.....] (4, 3, 6, 7, 2)
print(n[.....])

Les indices de caractères 1

Lors de la création d'un vecteur, nous pouvons également définir des indices de caractères pour les éléments, en plus des indices numériques.

ages <- c("Lorenz"=23, "Dav"=32, "Gilles"=33)

Les indices de caractères 2

Maintenant, nous pouvons accéder aux éléments par leur indice de caractère :
print(ages["Gilles"])
$\Longrightarrow$ Gilles
33
Cela donnera l'index du nom ainsi que la valeur correspondante.

Attention !

Si nous voulons afficher uniquement la valeur de cet indice, nous devons utiliser des doubles crochets : ages[["Gilles"]].

Exercice d'application 4

Les indices de caractères

Quel est l'output du programme ci-dessous ?

```
n <- c("a"=1, "b"=2, "c"=3)
x <- n[["a"]] + n[["c"]]
print(x)
```

.....

Negative index in vectors 1

On peut utiliser un indice négatif pour sauter un élément du vecteur.

names <- c("David", "Richard", "Marcel")
print(names[-3]) $\Longrightarrow$ [1] "David" "Richard"

Cela permet de sauter le troisième élément du vecteur.

Negative index in vectors 2

Nous pouvons également spécifier une plage pour ne sélectionner qu'un sous-ensemble des éléments.

names <- c("David", "Richard", "Marcel", "Anthony")
print(names[2:4]) $\Longrightarrow$ [1] "Richard" "Marcel" "Anthony"

Attention !

Ceci sélectionnera le 2ème au 4ème élément du vecteur.

Exercice d'application 5

Negative index in vectors

Quel est l'output du programme ci-dessous ?

```
n <- c(8, 4, 2, 3, 5)
x <- n[2:4]
x <- x[-1]
print(x[1])
```

1. 3
2. 2
3. 8
4. 4

Les fonctions liées aux vecteurs 1

La fonction length peut être utilisée pour trouver **le nombre d'éléments d'un vecteur** :

```
names <- c("David", "Richard", "Marcel", "Anthony")
print(length(names)) ⟹ [1] 4
```

La fonction sum renvoie **la somme de tous les éléments** :

```
x <- c(4, 8, 42, 1)
print(sum(x)) ⟹ [1] 55
```

Les fonctions liées aux vecteurs 2

La fonction **sort** peut être utilisée pour **trier les éléments d'un vecteur**.

x <- c(4, 8, 42)
print(sort(x)) $\Longrightarrow$ [1] 4 8 42

Cela permettra de trier le vecteur dans l'ordre croissant.

Attention !

Vous pouvez spécifier un paramètre supplémentaire pour trier par ordre décroissant :

sort(names, decreasing = TRUE)

Exercice d'application 6

Les fonctions liées aux vecteurs

Remplissez les espaces vides (avec sort, [["first"]], [1], TRUE, [0], vector, print, length) pour trier le vecteur dans l'ordre croissant et afficher le premier élément.

```
nums <- c(42, 105, 8, 3)
nums <- [.....] (nums)
[.....] (nums [.....] )
```

L'opérateur : (colons)

Nous avons déjà vu l'opérateur **: (colons)** lors de la définition d'une plage.
Cela crée en fait un vecteur de nombres consécutifs :

```
x <- 2:7
print(x) ⟹ [1] 2 3 4 5 6 7
```

La fonction seq()

Nous pouvons utiliser **la fonction seq()** pour **créer des séquences plus complexes qui suivent une règle donnée**.

x <- seq(2, 8, by=2)
print(x) $\Longrightarrow$ [1] 2 4 6 8

Attention !

Le paramètre by définit le pas à utiliser dans la séquence.

Exercice d'application 7

la fonction seq()

Remplissez les espaces vides pour créer une séquence de nombres pairs de 0 à 100, puis indiquez le nombre d'éléments du vecteur résultant.

x <- [.....] (0, [.....] , [.....] =2)
print([.....] (x))

Filtres 1

Nous pouvons également définir une condition permettant de filtrer les éléments.

x <- seq(1, 10, by=2)
print(x[x < 7]) $\Longrightarrow$ [1] 1 3 5

Cela ne sélectionnera que les éléments qui sont inférieurs à 7. Notez la syntaxe, nous utilisons le nom du vecteur dans la condition, qui représente chaque élément de la liste et le compare dans la condition.

Filtres 2

Si nous voulons modifier la valeur d'un élément, il suffit d'affecter l'indice correspondant à la nouvelle valeur :

```
x <- 1:5
x[2] <- 4
print(x) ⟹ [1] 1 4 3 4 5
```

Attention !

Comme vous pouvez le constater, les vecteurs possèdent un certain nombre de fonctions utiles qui leur permettent de manipuler les données qu'ils stockent.

Exercice d'application 8

Filtres

Quel est l'output du programme ci-dessous ?

```
x <- 5:9
x[3] <- 2
x <- x[x >=7]
print(length(x))
```

.....

Opérations arithmétiques sur les vecteurs 1

Deux vecteurs de même longueur peuvent être ajoutés, soustraits, multipliés ou divisés, ce qui donne un nouveau vecteur.

Voyons quelques exemples :

```
v1 <- c(2, 6, 1, 5)
v2 <- c(5, 3, 4, 8)
# addition
print(v1+v2) ⟹ [1] 7 9 5 13
# subtraction
print(v1-v2) ⟹ [1] -3 3 -3 -3
# multiplication
print(v1*v2) ⟹ [1] 10 18 4 40
# division
print(v1/v2) ⟹ [1] 0.400 2.000 0.250 0.625
```

Opérations arithmétiques sur les vecteurs 2

Attention !

Si nous appliquons des opérations arithmétiques à deux vecteurs de longueur inégale, les éléments du vecteur le plus court sont répétés pour compléter les opérations.

Exercice d'application 9

Opérations arithmétiques sur les vecteurs

Quelle est la sortie de ce code ?

```
a <- c(1, 2)
b <- c(3, 4)
x <- b/a
print(sum(x))
```

.....

Moyenne (mean)

La moyenne et la médiane sont des mesures de tendance centrale en statistiques.

R supporte les fonctions correspondantes pour les vecteurs afin de trouver ces valeurs. **La moyenne** de l'ensemble des données et peut être trouvée à l'aide de **la fonction mean()** :

v <- c(2, 6, 1, 5, 6)
print(mean(v)) $\Longrightarrow$ [1] 4

Médiane (median)

La médiane est le nombre moyen de l'ensemble des données ordonnées.
En R, vous pouvez utiliser **la fonction median** sur un vecteur :

```
v <- c(2, 6, 1, 5, 7)
print(median(v)) ⟹ [1] 5
```

Attention !

S'il y a un nombre impair d'éléments dans le vecteur, la fonction renvoie l'élément central. S'il y a un nombre pair d'éléments, la fonction renvoie la moyenne des deux médianes.

Exercice d'application 10

Médiane

Quelle est la valeur médiane de l'ensemble de données suivant ?

7, 5, 4, 3

1. 4.5
2. 3
3. 5
4. 4

Les lists 1

Les vecteurs ne peuvent contenir que des éléments du même type.
Les listes (lists) sont une autre structure de données dans R, qui sont **similaires aux vecteurs**, **mais peuvent contenir différents types de données**.

Une liste peut être créée à l'aide de la **fonction list()** :
x <- list("Enzo", "Guy", c(12, 1, 3), 33)

Comme vous pouvez le constater, **notre liste contient des chaînes de caractères, des nombres et un vecteur**.

Les lists 2

Vous pouvez accéder aux éléments de la liste comme à des vecteurs :

x <- list("Enzo", "Guy", c(12, 1, 3), 33)
print(x[[4]]) $\Longrightarrow$ [1] 33

Nous utilisons des doubles crochets pour afficher la valeur de l'élément, sans son index.

Attention !

Une liste peut également contenir une matrice ou une fonction comme éléments. Les listes sont utilisées dans R pour représenter des ensembles de données. Nous allons en apprendre davantage à ce sujet dans les prochains chapitres.

Exercice d'application 11

Les lists

Quel est le type du 2ème élément de cette liste ?

list("ABC", c(1, 2, 3), 42)

1. list
2. vector
3. numeric
4. string

L'opérateur $

Lors de l'utilisation d'éléments nommés, une alternative à [[, qui est souvent utilisée lors de l'accès au contenu d'une liste, est l'opérateur $:

x <- list("name"="Boris", "age"=35, "gender"=2,
"married"=TRUE)
print(x$name) $\Longrightarrow$ [1] "Boris"

Dans l'exemple ci-dessus, la liste x contient des données sur une personne. **Nous accédons à la valeur du nom en utilisant x$name**.
C'est **la même chose** que d'accéder à la valeur en utilisant **x[["name"]]**.

Ajout d'un nouvel élément à une liste

L'ajout d'un élément à une liste est facile. Il se fait à l'aide de l'index nommé :

```
x <- list("name"="Boris", "age"=35, "gender"=2,
"married"=TRUE)
x[["town"]] <- "Dakar"
print(x) ⟹ $name [1] "Boris"    $age [1] 35    $gender [1] 2
$married [1] TRUE    $town [1] "Dakar"
```

Nous avons ajouté un élément "town" à la liste, avec la valeur "Dakar".

Attention !

Notez que nous devons utiliser des doubles crochets pour assigner la valeur.

Exercice d'application 12

L'opérateur $

x$age est le même que :

1. age
2. x[["age"]]
3. x["age"]
4. x<-age

Les opérations sur les listes 1

Nous pouvons **fusionner deux listes** à l'aide de **la fonction c()** :

```
list1 <- list("A", "B", "C")
list2 <- list("D", "E")
x <- c(list1, list2)
print(x) ⟹ [[1]] [1] "A"      [[2]] [1] "B"      [[3]] [1] "C"
[[4]] [1] "D"      [[5]] [1] "E"
```

Les opérations sur les listes 2

Si nous voulons **appliquer des opérations vectorielles ou arithmétiques à nos éléments**, nous pouvons **convertir la liste en vecteur à l'aide de la fonction unlist()** :

x <- list(4, 2, 12)
y <- unlist(x)
print(sort(y)) $\Longrightarrow$ [1] 2 4 12
print(mean(y)) $\Longrightarrow$ [1] 6

Attention !

Le unlisting d'une liste contenant différents types de données les convertira au même type, car un vecteur ne peut contenir qu'un seul type d'éléments.

Exercice d'application 13

Les opérations sur les listes

Remplissez les espaces vides (avec c, mean, unlist, y, sum, length, x) pour convertir la liste en vecteur et indiquer le nombre d'éléments qu'elle contient.

```
x <- list("a", "b", "c")
y <- [.....] ( [.....] )
print( [.....] ( [.....] ) )
```

Les matrices 1

Une matrice est un ensemble de données bidimensionnelles comportant des lignes et des colonnes. Elle est similaire à un vecteur, mais possède une dimension supplémentaire.

Une **matrice peut être créée** avec **la fonction matrix()**, **en spécifiant les lignes et les colonnes à l'aide des paramètres nrow et ncol**.

```
x <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
print(x)
```

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Les matrices 2

Comme vous pouvez le voir, nous donnons un vecteur comme données et créons une matrice avec 2 lignes et 3 colonnes.

Par défaut, les données sont remplies par colonne.

Comme pour un vecteur, **tous les éléments doivent être du même type**.

Attention !

Vous pouvez ignorer l'un des paramètres, il sera automatiquement calculé à partir de la longueur des données.

Exercice d'application 14

Les matrices

Combien de lignes aura la matrice suivante ?

matrice(c(8, 0, 2, 5), ncol = 4)

1. 4
2. 2
3. 1

Accès aux éléments d'une matrice 1

Vous pouvez accéder aux éléments de la matrice en spécifiant la ligne et la colonne entre crochets, séparées par une virgule :
x <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
print(x[1, 3]) $\Longrightarrow$ [1] 5
Cela produira le troisième élément de la première ligne.

Vous pouvez accéder à une ligne entière en spécifiant son numéro et en sautant la colonne :
x <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
print(x[1,]) $\Longrightarrow$ [1] 1 3 5

Accès aux éléments d'une matrice 2

De même, vous pouvez accéder à une colonne entière :

x <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3) print(x[,1]) $\Longrightarrow$
[1] 1 2

Attention !

Vous pouvez attribuer des valeurs aux éléments de la matrice en utilisant leur indice, par exemple : x[2, 3] <- 8

Exercice d'application 15

Accès aux éléments d'une matrice

Remplissez les espaces vides pour accéder à l'élément de la matrice qui se trouve à la ligne 5 et à la colonne 3. La matrice est appelée "m".

m [..... ....]

Opérations matricielles 1

Une opération courante avec les matrices est leur transposition. La transposition d'une matrice est un opérateur qui renverse une matrice sur sa diagonale, c'est-à-dire qu'il intervertit les indices de ligne et de colonne de la matrice.

Nous pouvons **transposer** une matrice dans R avec la **fonction t()** :

Opérations matricielles 2

```
x <- matrix(c(1,2,3,4,5,6), nrow = 2, ncol = 3)
print(x)
```

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

```
print(t(x))
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Opérations matricielles 3

Attention !

Les matrices sont une structure de données très utilisée, car elles permettent de stocker et de manipuler des données ayant une structure en lignes et en colonnes.

Exercice d'application 16

Opérations matricielles

Quelle est la sortie de ce code ?

```
a <- matrix(c(8, 0, 2, 5), ncol = 2)
a <- t(a)
print(a[2, 1])
```

1. 0
2. 5
3. 8
4. 2

Les DataFrames 1

Nous avons vu comment stocker des données dans des vecteurs, des listes et des matrices.
Elles ont toutes des moyens similaires d'accéder à des éléments, de les modifier et d'en ajouter. La principale différence entre eux réside dans la structure et le type de données qu'ils peuvent stocker.

Les vecteurs stockent des **éléments de même type** en utilisant **une dimension**.
Les matrices sont comme les vecteurs et ont **2 dimensions : lignes et colonnes**.
Les listes sont similaires aux vecteurs et permettent de **stocker différents types d'éléments**.

Les DataFrames 2

Le plus souvent, nos données se présentent sous la forme d'un tableau et chaque colonne peut être de type différent.

Attention !

Que faire si nous voulons stocker différents types d'éléments en 2 dimensions ? C'est là qu'intervient la structure de données la plus importante de R : un **DataFrame**.

Exercice d'application 17

Les DataFrames

Laquelle des structures de données suivantes permet de stocker des éléments de différents types ?

1. matrix
2. vector
3. list

La fonction data.frame() 1

Un data frame est un tableau, où chaque colonne a un nom et peut contenir n'importe quel type de données.
Chaque colonne doit contenir le même nombre de données.

Nous pouvons **créer un dataf frame** à l'aide de la **fonction data.frame()**.

La fonction data.frame() 2

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
print(x)
```

```
  id  name age
1  1 James  42
2  2   Amy  18
```

Dans le code ci-dessus, nous avons créé un data frame avec 3 colonnes : id, nom et age. Chacune d'entre elles contient un vecteur de valeurs.

Attention !

Ainsi, le data frame résultant aura 3 colonnes et 2 lignes.

Exercice d'application 18

La fonction data.frame()

Remplissez les espaces vides pour créer un data frame valide.

x <- [.....] . [.....] (
"product"=c("A", "B"),
"price"=c(42, 55) [.....]

Accès aux éléments du data frame 1

Vous pouvez accéder aux éléments d'un data frame à l'aide de doubles crochets ou de l'opérateur $, en utilisant le nom de la colonne.

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
# second column
print(x[[2]]) ⟹ [1] "James" "Amy"
# the name column
print(x[["name"]]) ⟹ [1] "James" "Amy"
# the same as
print(x$name) ⟹ [1] "James" "Amy"
```

Les résultats de l'indexation ci-dessus sont des vecteurs.

Accès aux éléments du data frame 2

L'utilisation de crochets simples renvoie un data frame, composé de la colonne spécifiée :

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
print(x[2])
```

```
    name
1  James
2   Amy
```

Accès aux éléments du data frame 3

Nous pouvons également accéder aux éléments d'un data frame de manière similaire à une matrice, en spécifiant la ligne et la colonne :

x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
print(x[[2, 3]]) $\Longrightarrow$ [1] 18

Attention !

Nous pouvons également utiliser les noms des colonnes lors de l'indexation, par exemple : x[[2, "name"]] sélectionnera la colonne name de la 2ème ligne.

Exercice d'application 19

Accès aux éléments du data frame

Remplissez les cases vides pour obtenir le prix des Apples.

```
x <- data.frame( "product"=c("Banana", "Apple", "Orange"),
"price"=c(5.99, 2.49, 3.0))
print( [.....] $price[ [.....] ])
```

Opérations sur les dataframes 1

Vous pouvez ajouter une nouvelle colonne à un data frame en lui attribuant simplement un vecteur de valeurs :

Par exemple, ajoutons une colonne de pays à notre cadre de données :

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
x$country <- c("USA","Italy")
print(x)
```

	id	name	age	country
1	1	James	42	USA
2	2	Amy	18	Italy

Opérations sur les dataframes 2

Nous pouvons filtrer les lignes en fonction d'une condition, comme dans une matrice :

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
print(x[x$age¿21, ])
      id    name    age
  1    1    James    42
```

Cela sélectionne toutes les lignes dont la colonne âge est supérieure à 21. Notez la virgule après la condition, elle est nécessaire pour sélectionner toutes les colonnes des lignes résultantes.

Opérations sur les dataframes 3

Le filtrage peut également être effectué à l'aide de la fonction subset, qui prend le dataframe et la condition comme paramètres :

x <- data.frame("id" = 1:2, "name" = c("James", "Amy"), "age" = c(42,18)) print(subset(x, age >21))

	id	name	age
1	1	James	42

Vous pouvez avoir plusieurs conditions, combinées à l'aide des opérateurs logiques & et |. Notez que vous

Attention !

Notez que vous devez spécifier le nom de la colonne dans la condition, sans l'opérateur $.

Exercice d'application 20

Opérations sur les dataframes

Remplissez les espaces vides pour sélectionner uniquement les lignes du dataframe "x" dont le "price" est inférieur à 50 et l'"id" supérieur à 8.

[.....] (x, [.....] $<$ 50 [.....] id $>$ [.....])

Les statistiques de base 1

Comme les colonnes d'un dataframe sont des vecteurs, vous pouvez également utiliser la somme, la moyenne, la médiane et d'autres fonctions vectorielles sur les colonnes du dataframe.

Par exemple, calculons l'âge moyen dans nos données :

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
print(mean(x$age)) ⟹ [1] 30
```

Attention !

Consultez notre cours de Data Science pour en savoir plus sur les statistiques descriptives.

Les statistiques de base 2

Nous sélectionnons la colonne à l'aide de l'opérateur $, et la passons à la fonction correspondante.

Les dataframes R peuvent être examinés à l'aide de **la fonction summary**.
Elle produit les **statistiques récapitulatives pour chacune des colonnes** :

```
x <- data.frame("id" = 1:2, "name" = c("James", "Amy"),
"age" = c(42,18))
summary(x)
```

```
      id          name                age
Min.   :1.00   Length:2           Min.   :18
1st Qu.:1.25   Class :character   1st Qu.:24
Median :1.50   Mode  :character   Median :30
Mean   :1.50                      Mean   :30
3rd Qu.:1.75                      3rd Qu.:36
Max.   :2.00                      Max.   :42
```

Exercice d'application 21

Les statistiques de base

Remplissez les espaces vides (avec mean, sum, $score, results, count, x) pour obtenir la somme de la colonne "score" dans le data frame "résultats".

```
x <- [.....] ( [.....] [.....] )
print(x)
```

Les variables de type factors 1

Lorsqu'un dataframe comporte une **colonne de texte**, **R traite cette colonne comme une donnée catégorielle et crée des facteurs (factors) sur celle-ci**.

Les facteurs (factors) sont **des variables** dans R **qui prennent un nombre limité de valeurs différentes**, comme les noms de mois ou les jours de la semaine. C'est utile en analyse statistique, car cela permet de stocker et de traiter correctement ces données et de les utiliser dans différents modèles.

Par exemple, disons que notre data frame a besoin d'une colonne de sexe, qui peut prendre l'une des deux valeurs suivantes : "Male" ou "Female".

Les variables de type factors 2

Nous créons un facteur en utilisant la fonction factor, en lui passant un vecteur de valeurs dont nous avons besoin dans notre colonne :

```
gender <- factor(c("Male", "Female", "Male"))
```

Ces valeurs sont appelées **levels** (niveaux). Nous pouvons y **accéder** à l'aide de **la fonction levels ()** :

```
x <- data.frame("id" = 1:3, "name" = c("James", "Amy",
"Bob"), "age" = c(42,18, 33))
gender <- factor(c("Male", "Female", "Male"))
print(levels(gender)) ⟹ [1] "Female" "Male"
```

Les variables de type factors 3

Chaque valeur d'un facteur a son numéro correspondant.
Nous pouvons les **vérifier** à l'aide de **la fonction table()**:

```
x <- data.frame("id" = 1:3, "name" = c("James", "Amy",
"Bob"), "age" = c(42,18, 33))
gender <- factor(c("Male", "Female", "Male"))
print(table(gender))
```

```
gender
Female   Male
     1      2
```

Comme vous pouvez le voir, **Femme est 1**, **Homme est 2**, **parce que F vient avant M**.

Les variables de type factors 4

Maintenant, nous pouvons affecter notre facteur à une nouvelle colonne dans notre data frame :

```
x <- data.frame("id" = 1:3, "name" = c("James","Amy",
"Bob"), "age" = c(42,18, 33))
gender <- factor(c("Male", "Female", "Male"))
x$gender <- gender
print(x)
```

```
  id  name age gender
1  1 James  42   Male
2  2   Amy  18 Female
3  3   Bob  33   Male
```

Les variables de type factors 5

Attention !

Les facteurs représentent un moyen très efficace de stocker des valeurs textuelles, car chaque valeur textuelle unique n'est stockée qu'une seule fois, et les données elles-mêmes sont stockées sous forme de vecteur d'entiers.

Exercice d'application 22

Les variables de type factors

Créer un facteur pour le vecteur "mois" et afficher les valeurs et leurs nombres correspondants.

```
months <- c("Jan", "Feb", "Jan", "Apr", "Feb", "Jan")
x <- ..... (months)
print( ..... (x))
```

Quiz 1

1) min/max/sum

Quel est l'output du programme ci-dessous ?

```
x <- c(6, 1, 8, 3)
y <- min(x)
z <- max(x)
res <- sum(x)-z-y
print(res)
```

.....

Quiz 2

2) vecteurs

Remplissez les espaces vides pour créer un vecteur de nombres et produisez le troisième élément.

nums <- [.....] (4, 8, 1, 22)
print([.....] [3 [.....])

Quiz 3

3) Data structures

Laquelle des affirmations suivantes est vraie ?

1. Les listes peuvent contenir différents types de données
2. Les listes ne peuvent pas stocker de chaînes de caractères
3. Les vecteurs ne peuvent contenir que des nombres
4. Les vecteurs peuvent contenir différents types de données

Quiz 4

4) matrices

Quel est l'output du programme ci-dessous ?

```
x <- matrix(c(3, 1, 0, 8, 5, 3), nrow = 3)
print(x[3, 1])
```

1. 5
2. 0
3. 3
4. 8

Quiz 5

5) data frame

Quel est l'output du programme ci-dessous ?

```
x <- data.frame( "a" = c(7, 42, 3, 5), "b" = c("a","b","c","d") )
y <- subset(x, a < 10)
z <- max(y$a)
print(min(mean(y$a), z))
```

.....

Quiz 6

6) nouvelle colonne d'un data frame

Remplissez les espaces vides (avec data, new, <-, $brand, [1], cars, frame, ['second'], [2], $color) pour créer un cadre de données nommé "cars", ajoutez-y une nouvelle colonne "brand" et indiquez la couleur de la deuxième ligne.

cars <- [.....] . [.....] ("price" = c(4.99, 9.59, 11.3), "color" = c("red","blue","green"))
cars [.....] [.....] c("BMW", "Audi", "VW")
print(cars [.....] [.....])

R Studio

Chapitre III : Analyse des données

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

April 20, 2022

"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons apprendre comment importer les données, calculer les statistiques de base et réaliser les opérations mathématiques de base.

À la fin de ce chapitre, vous serez en mesure :

- d'importer les données dans R.
- de calculer les statistiques descriptives
- de réaliser les opérations mathématiques de base
- de manipuler les données (transformer, grouper, ajouter, supprimer, etc.)

Prérequis du chapitre

Attention !

Avoir suivi le chapitre précédent.

Importer les données

Les ensembles de données sont souvent présentés dans des tableaux au format CSV (comma-separated values).

R vous permet **d'importer des données depuis un fichier CSV** à l'aide de **la fonction read.csv** :

```
data <- read.csv("demo.csv")
```

Attention !

Maintenant, la variable data est un dataframe avec les données du fichier demo.csv.

Exercice d'application 1

Importer les données

Complétez (avec read, ,1, csv, x, 1, import, data, books.csv) pour importer les données du fichier "books.csv" et afficher la première ligne.

x <- [.....] . [.....] (" [.....] ")
print(x[[.....]])

Le jeu de données mtcars 1

Pour ce module, nous allons utiliser un ensemble de données intégré à R. Il s'agit de **"mtcars" (Motor Trend Car Road Tests)**.

Il s'agit de "mtcars" (Motor Trend Car Road Tests), qui est extrait du magazine américain Motor Trend de 1974 et comprend des données sur les voitures.

Vous pouvez y accéder par son nom :

```
print(mtcars)
```

Le jeu de données mtcars 2

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4
Chrysler Imperial	14.7	8	440.0	230	3.23	5.345	17.42	0	0	3	4
Fiat 128	32.4	4	78.7	66	4.08	2.200	19.47	1	1	4	1
Honda Civic	30.4	4	75.7	52	4.93	1.615	18.52	1	1	4	2
Toyota Corolla	33.9	4	71.1	65	4.22	1.835	19.90	1	1	4	1
Toyota Corona	21.5	4	120.1	97	3.70	2.465	20.01	1	0	3	1
Dodge Challenger	15.5	8	318.0	150	2.76	3.520	16.87	0	0	3	2
AMC Javelin	15.2	8	304.0	150	3.15	3.435	17.30	0	0	3	2
Camaro Z28	13.3	8	350.0	245	3.73	3.840	15.41	0	0	3	4
Pontiac Firebird	19.2	8	400.0	175	3.08	3.845	17.05	0	0	3	2
Fiat X1-9	27.3	4	79.0	66	4.08	1.935	18.90	1	1	4	1
Porsche 914-2	26.0	4	120.3	91	4.43	2.140	16.70	0	1	5	2
Lotus Europa	30.4	4	95.1	113	3.77	1.513	16.90	1	1	5	2
Ford Pantera L	15.8	8	351.0	264	4.22	3.170	14.50	0	1	5	4
Ferrari Dino	19.7	6	145.0	175	3.62	2.770	15.50	0	1	5	6
Maserati Bora	15.0	8	301.0	335	3.54	3.570	14.60	0	1	5	8
Volvo 142E	21.4	4	121.0	109	4.11	2.780	18.60	1	1	4	2

Le jeu de données mtcars 3

Il comporte 11 colonnes :

1. **mpg** : Miles/(US) gallon
2. **cyl** : Number of cylinders
3. **disp** : Displacement (cu.in.)
4. **hp** : Gross horsepower
5. **drat** : Rear axle ratio
6. **wt** : Weight (1000 lbs)
7. **qsec** : 1/4 mile time
8. **vs** : Engine (0 = V-shaped, 1 = straight)
9. **am** : Transmission (0 = automatic, 1 = manual)
10. **ear** : Number of forward gears
11. **carb** : Number of carburetors

Le jeu de données mtcars 4

Attention !

Par soucis de simplicité, nous utiliserons cet ensemble de données dans nos exemples. Cependant, vous pouvez utiliser n'importe quel autre ensemble de données pour effectuer une analyse de données à l'aide des techniques abordées dans ce chapitre.

Exercice d'application 2

Le jeu de données mtcars

Laquelle des propositions suivantes renvoie les 5 premières lignes de l'ensemble de données mtcars ?

1. mtcars[1, 5]
2. mtcars[5, 1]
3. mtcars[1:5,]
4. mtcars[, 1:5]

Summary statistics 1

Commençons par obtenir les statistiques descriptives (ou récapitulatives ou encore summary statistics) de notre ensemble de données :

summary(mtcars)

```
      mpg             cyl             disp             hp       
 Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
 1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
 Median :19.20   Median :6.000   Median :196.3   Median :123.0  
 Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
 3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
 Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
      drat             wt             qsec             vs        
 Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
 1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
 Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
 Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
 3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
 Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
       am              gear            carb      
 Min.   :0.0000   Min.   :3.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
 Median :0.0000   Median :4.000   Median :2.000  
 Mean   :0.4062   Mean   :3.688   Mean   :2.812  
 3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
 Max.   :1.0000   Max.   :5.000   Max.   :8.000  
```

Summary statistics 2

Comme nous pouvons le voir dans les résultats, les 11 colonnes ont toutes des valeurs numériques.

Attention !

Les statistiques récapitulatives montrent la distribution des valeurs dans les colonnes, y compris le min/max/moyen des valeurs.

Exercice d'application 3

Summary statistics

Vrai ou Faux : La fonction summary() inclut les valeurs médianes des colonnes de l'ensemble de données.

1. False
2. True

Variance and Standard Deviation 1

La variance est une mesure de la dispersion d'un ensemble de données.
L'écart-type est la mesure de la dispersion d'un ensemble de données par rapport à sa moyenne. **Plus la dispersion est importante, plus l'écart-type est grand**.

R supporte des fonctions permettant de trouver ces valeurs.
La **fonction var** renvoie la **variance d'un vecteur**, tandis que **sd** renvoie **l'écart type**.

Variance and Standard Deviation 2

Calculons ces valeurs pour notre colonne mpg :

var(mtcars$mpg) $\Longrightarrow$ [1] 36.3241
sd(mtcars$mpg) $\Longrightarrow$ [1] 6.026948

L'écart-type est la racine carrée de la variance.

Attention !

Vous pouvez en savoir plus sur la variance et l'écart-type dans notre programme de Data Science.

Exercice d'application 4

Variance and Standard Deviation

Remplissez les espaces vides (avec $age, people, 'age', sd, var, std) pour trouver l'écart type de la valeur de l'âge dans le dataframe "people".

..... (..... .....)

Les opérations basiques 1

Comme mtcars est un dataframe, nous pouvons facilement effectuer des opérations sur celui-ci. Par exemple, trouvons la somme des poids (sum of the weights) de toutes les voitures de l'ensemble de données :

sum(mtcars$wt) $\Longrightarrow$ [1] 102.952

Attention !

Vous pouvez utiliser toutes les opérations et fonctions que nous avons abordées dans les chapitres précédents.

Exercice d'application 5

Les opérations basiques

Remplissez les espaces vides pour trouver la valeur maximale de la colonne de height dans le data frame "team".

[.....] (team [.....] height)

Les opérations basiques 2

Rendons les choses un peu plus difficiles.
Nous devons trouver la ligne qui correspond à la voiture ayant la puissance maximale.

Pour ce faire, nous devons d'abord trouver la valeur de puissance maximale, puis filtrer l'ensemble de données en fonction de cette condition :
mtcars[mtcars$hp == max(mtcars$hp),]

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Maserati Bora	15	8	301	335	3.54	3.57	14.6	0	1	5	8

Les opérations basiques 3

Cela renverra la ligne qui correspond à la voiture ayant la puissance maximale (maximum horsepower).

Attention !

Notez la virgule après la condition, elle est nécessaire pour retourner toutes les colonnes, car notre ensemble de données est un tableau avec des lignes et des colonnes.

Exercice d'application 6

Autres opérations

Remplissez les espaces vides pour trouver les lignes de l'ensemble de données mtcars dont la valeur mpg est supérieure à 30.

mtcars[.....] $mpg [.....] 30 [.....]

Filter les données 1

Nous pouvons utiliser plusieurs conditions pour trouver les lignes dont nous avons besoin.

Par exemple, trouvons la voiture la plus rapide dotée d'une boîte de vitesses automatique (gearbox). La colonne **qsec** indique **le temps au 1/4 mile**, tandis que la colonne **am** indique **le type de transmission** : 0 = automatique, 1 = manuelle.

Filter les données 1

```
x <- mtcars[mtcars$am == 0, ]
x[x$qsec == min(x$qsec), ]
```

```
              mpg cyl disp  hp drat   wt  qsec vs am gear carb
Camaro Z28 13.3   8  350 245 3.73 3.84 15.41  0  0    3    4
```

Attention !

Nous filtrons d'abord les voitures à transmission automatique, puis nous obtenons la ligne qui correspond au qsec minimum.

Exercice d'application 7

Filter les données

Remplissez les espaces vides pour trouver la moyenne des "qsec" des voitures qui ont une "hp" supérieure à 100 et un "gear" inférieur à 5.

```
x <- mtcars[mtcars [.....] > 100 [.....] mtcars$gear < [.....] ,
'qsec']
print( [.....] (x))
```

Corrélation 1

Une **matrice de corrélation** est utilisée en analyse de données pour **trouver la dépendance entre plusieurs colonnes**.

Une valeur de corrélation proche de 1 indique que les colonnes comparées sont fortement corrélées, tandis qu'une valeur proche de 0 indique qu'elles sont moins corrélées.

Vous pouvez trouver la **matrice de corrélation** à l'aide de **la fonction cor** :

Corrélation 2

```
res <- cor(mtcars)
res <- round(res, 2)
print(res)
```

```
       mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
mpg   1.00 -0.85 -0.85 -0.78  0.68 -0.87  0.42  0.66  0.60  0.48 -0.55
cyl  -0.85  1.00  0.90  0.83 -0.70  0.78 -0.59 -0.81 -0.52 -0.49  0.53
disp -0.85  0.90  1.00  0.79 -0.71  0.89 -0.43 -0.71 -0.59 -0.56  0.39
hp   -0.78  0.83  0.79  1.00 -0.45  0.66 -0.71 -0.72 -0.24 -0.13  0.75
drat  0.68 -0.70 -0.71 -0.45  1.00 -0.71  0.09  0.44  0.71  0.70 -0.09
wt   -0.87  0.78  0.89  0.66 -0.71  1.00 -0.17 -0.55 -0.69 -0.58  0.43
qsec  0.42 -0.59 -0.43 -0.71  0.09 -0.17  1.00  0.74 -0.23 -0.21 -0.66
vs    0.66 -0.81 -0.71 -0.72  0.44 -0.55  0.74  1.00  0.17  0.21 -0.57
am    0.60 -0.52 -0.59 -0.24  0.71 -0.69 -0.23  0.17  1.00  0.79  0.06
gear  0.48 -0.49 -0.56 -0.13  0.70 -0.58 -0.21  0.21  0.79  1.00  0.27
carb -0.55  0.53  0.39  0.75 -0.09  0.43 -0.66 -0.57  0.06  0.27  1.00
```

Corrélation 3

Nous pouvons voir dans les résultats que, par exemple, la colonne des chevaux (**hp**) est **fortement corrélée** à celle des cylindres (**cyl**).

C'est logique, puisque la puissance dépend du nombre de cylindres.

Attention !

Nous avons utilisé la fonction round pour arrondir les résultats à 2 décimales pour une meilleure lisibilité.

Exercice d'application 8

Corrélation

Quelle serait la valeur de corrélation en comparant une colonne à elle-même ?

1. 1.0
2. 0.0
3. -1.0
4. 0.5

Grouper les données 1

R nous permet de **regrouper les données par une colonne et de calculer un agrégat pour les groupes**.

Par exemple, supposons que nous voulons trouver la puissance moyenne en chevaux (hp) de nos voitures, regroupées par type de transmission (am).
Nous pouvons le faire avec la **fonction by()**. Elle **applique une fonction à chaque groupe** :

Grouper les données 2

by(mtcars$hp, mtcars$am, mean)

```
mtcars$am: 0
[1] 160.2632
------------------------------------------------------------
mtcars$am: 1
[1] 126.8462
```

Grouper les données 3

Le premier paramètre est la colonne sur laquelle nous appliquons la fonction, le deuxième paramètre définit la colonne par laquelle regrouper les données, et le troisième paramètre est la fonction que nous voulons appliquer.

Attention !

Le code ci-dessus calculera la puissance moyenne pour les voitures à transmission automatique et manuelle.

Exercice d'application 9

Grouper les données

Remplissez les espaces vides (avec sum, mean, $wt, by, group, max, $vs) pour trouver la somme de la colonne "wt" pour toutes les voitures, regroupées par type de moteur (colonne "vs").

[.....] (mtcars [.....] , mtcars [.....] , [.....])

La fonction tapply() 1

Une autre façon de **regrouper et d'exécuter des fonctions agrégées** est la **fonction tapply** :

tapply(mtcars$hp, mtcars$am, mean)

```
       0        1
160.2632 126.8462
```

La fonction tapply() 2

Elle possède les mêmes paramètres que la fonction by().

La **principale différence** entre tapply et by est que **tapply renvoie une matrice**, tandis que **by renvoie un objet**, qui peut être converti en liste.
by() est en fait une enveloppe de tapply().
Choisissez la fonction de regroupement avec laquelle vous vous sentez le plus à l'aise.

Attention !

Comme nous l'avons vu dans ce module, R supporte des fonctions qui vous permettent de manipuler facilement les données stockées dans un data frame. L'utilisation de ces fonctions dépend de la tâche d'analyse de données que vous essayez d'effectuer.

Exercice d'application 10

La fonction tapply()

Remplissez les espaces vides pour trouver la voiture ayant la plus grande valeur de puissance (colonne hp) pour les voitures regroupées par nombre de cylindres (colonne cyl).

tapply(mtcars$ [.....] , mtcars$ [.....] , [.....])

Quiz 1

1) filtre et calcul

Remplissez les espaces vides (avec 'hp', mtcars, [0], $mtcars, $gear, all, ,) pour trouver la puissance en chevaux (hp) des voitures de l'ensemble de données mtcars qui ont plus de 4 vitesses.

[.....] [mtcars [.....] > 4, [.....]]

Quiz 2

2) la fonction by()

Remplissez les espaces vides (avec $cyl, mean, $mpg, mtcars, by, max, min) pour trouver le mpg maximum pour chaque cyl (cylindre).

by(mtcars [.....] , mtcars [.....] , [.....])

Quiz 3

3) Standard deviation

Quelle fonction est utilisée pour trouver l'écart-type ?

1. dev
2. sd
3. var
4. mean

Quiz 4

4) import, groupes et calculs

Remplissez les espaces vides pour importer les données du fichier "people.csv", sélectionnez les lignes correspondant aux personnes dont l'"age" est supérieur à 18 ans et affichez leur "height" moyenne.

```
data <- [.....].csv("people.csv")
data <- data[data$[.....]>18, ]
res <- [.....](data$[.....])
print([.....])
```

Quiz 5

5) Filtres

Remplissez les espaces vides pour trouver les voitures de l'ensemble de données mtcars dont la consommation est supérieure à 20 et dont le poids est le plus faible (wt).

```
x <- mtcars[ mtcars$ ..... > ..... , ]
print( ..... (x$ ..... ))
```

Classer les 10 chiffres 1

Attention !

Exercice d'application 9

Interprétation

Regardons l'enfant gauche du nœud ?

1. 144
2. 532
3. 170

R Studio

Chapitre IV : Visualization

David TCHOUTA

Académie Française du Numérique
www.frenchtechacademie.fr
Tél/Whatsapp : +33 (0)7 49 62 72 49

April 20, 2022

"Notre plus grande faiblesse réside dans l'abandon, la façon la plus sûre de réussir est d'essayer une autre fois." Thomas Edison

Table de matières

Objectifs du chapitre

Dans ce chapitre, nous allons apprendre à construire les diagrammes en courbes, en barres, les diagrammes circulaires, les boîtes de moustaches et les histogrammes.

À la fin de ce chapitre, vous serez en mesure :

- de construire un graphique,
- de construire un diagramme en courbe,
- de construire un diagramme en barres,
- de construire un diagramme circulaire,
- de construire une boîte de moustaches,
- de construire des histogrammes,

cyan

Prérequis du chapitre

Attention !

Avoir suivi le chapitre précédent.

Plotting 1

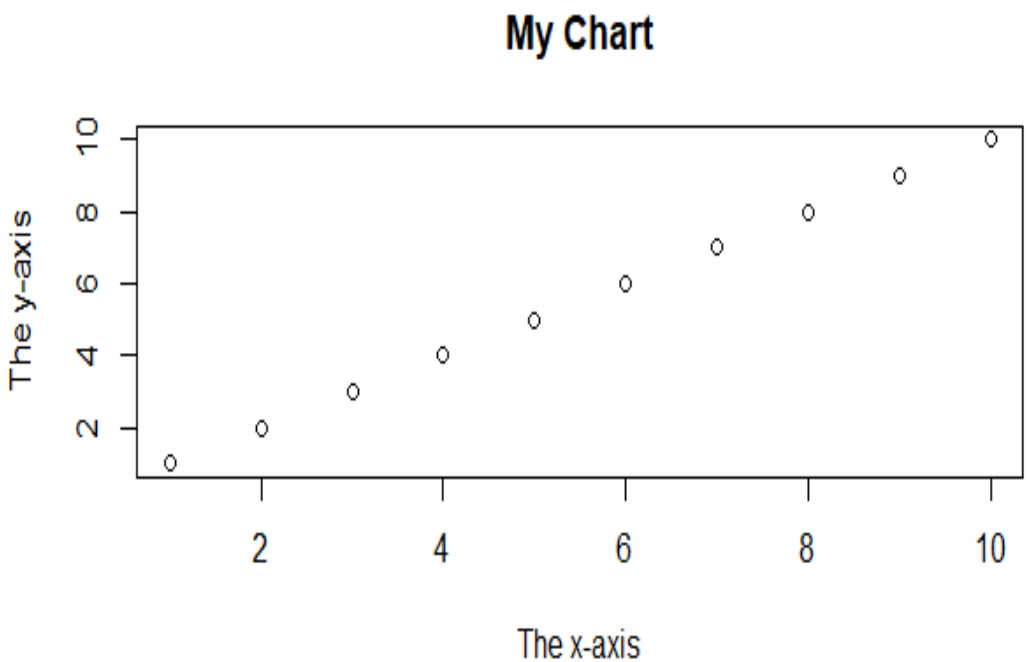
La visualisation des données vous permet de trouver des informations dans les données et de créer des rapports pour présenter les données et leurs caractéristiques.

R prend en charge des fonctions permettant de créer des diagrammes et des graphiques.

Commençons par créer un graphique simple de points de données pour comprendre la syntaxe :

Plotting 2

```
png(file = "chart.png")
plot(1:10, main="My Chart", xlab="The x-axis", ylab="The y-axis")
```



Plotting 3

La **fonction png** est utilisée pour **donner un nom à notre graphique**.

Ensuite, la **fonction plot()** est utilisée pour **créer le graphique**. Elle peut prendre une séquence de nombres comme paramètre et les utiliser comme points de données sur le graphique.

Le **paramètre main** est utilisé pour **donner un titre au graphique**. Les **paramètres xlab et ylab** permettent **d'étiqueter les axes x et y du graphique**.

Exercice d'application 1

Plotting

Remplissez les espaces vides pour créer un graphique valide des points 1 à 100.

[.....] (1 [.....] 100, main="My Chart", xlab="The x-axis",
[.....] ="The y-axis")

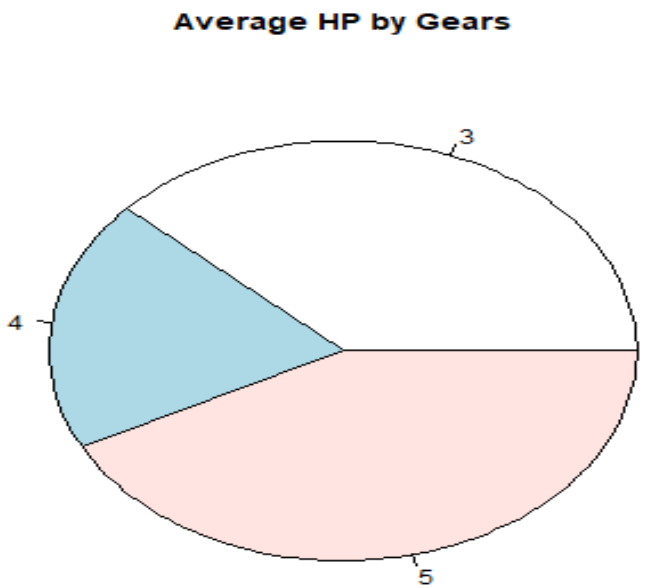
Plotting 4

Dans notre exemple précédent, nous avons utilisé une séquence de nombres pour les données de notre échantillon de graphique. Dans ce cas, les coordonnées x et y des points étaient extraites de la séquence, de sorte que le premier point avait les coordonnées (1, 1), le second (2, 2), etc.

Nous pouvons également fournir à la **fonction plot() les coordonnées x et y séparément**, sous forme de vecteurs.
Par exemple, **utilisons les colonnes wt et drat de l'ensemble de données mtcars comme coordonnées x et y** :

Plotting 5

```
x <- mtcars$wt
y <- mtcars$drat
png(file = "chart.png")
plot(x, y, xlab="weight", ylab="rear axle ratio")
```



Plotting 6

Ce graphique représente la corrélation entre le poids (weight) et le rapport de pont arrière (rear axle ratio) dans l'ensemble de données.

Attention !

Vous pouvez dessiner autant de points que vous le souhaitez.

Exercice d'application 2

Coordonnées y

Quelle est la coordonnée Y du deuxième point de ce graphique ?

```
a <- c(2, 8)
b <- c(5, 3)
plot(a, b)
```

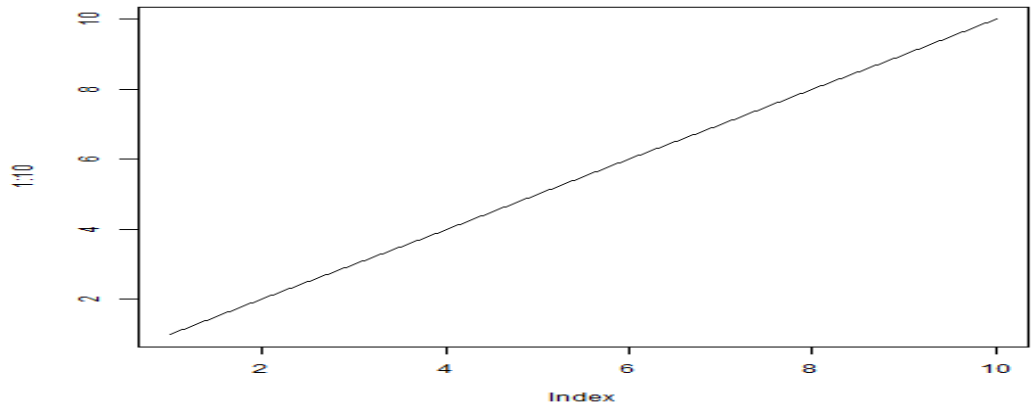
1. 5
2. 3
3. 8
4. 2

Les courbes 1

La **fonction plot** peut également être **utilisée pour dessiner des graphiques linéaires (courbes)**. Il suffit de spécifier le paramètre **type** et d'utiliser la valeur **"l"**.

```
png(file = "p2.png")
plot(1:10, type="l")
```

Les courbes 2



Attention !

Cette opération permet de tracer une ligne reliant tous les points du graphique.

Exercice d'application 3

Les courbes

Remplissez les espaces vides (avec line, l, type, chart, x, plot, data) pour créer un graphique linéaire basé sur les valeurs du vecteur "data".

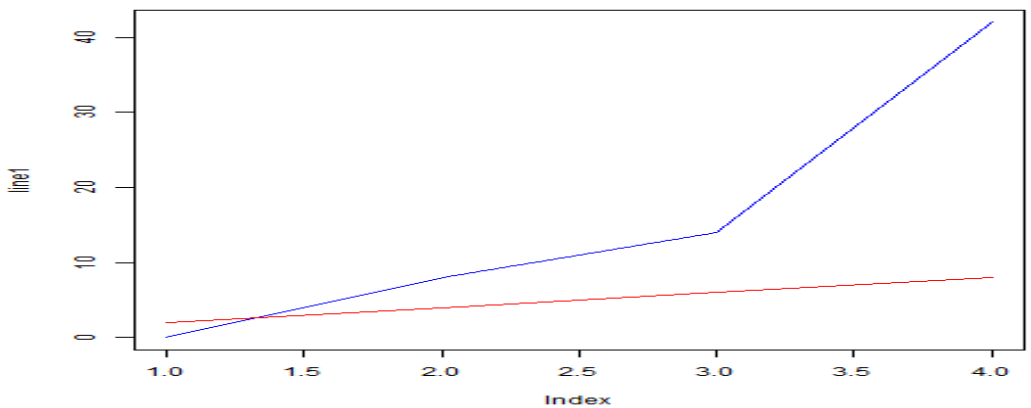
[.....] (data, [.....] = " [.....] ")

Courbes multiples 1

Nous pouvons dessiner **plusieurs courbes** sur un graphique à l'aide de la fonction **lines()** :

```
png(file = "p3.png")
line1 <- c(0, 8, 14, 42)
line2 <- c(2, 4, 6, 8)
plot(line1, type = "l", col = "blue")
lines(line2, type="l", col = "red")
```

Courbes multiples 2



Courbes multiples 3

Nous stockons chacune des coordonnées de la ligne dans des vecteurs. Ensuite, nous utilisons **la fonction plot()** pour tracer la première ligne. Pour les autres lignes, nous devons utiliser **la fonction lines()**, qui fonctionne de la même manière que la **fonction plot()**.

Attention !

Le paramètre col est utilisé pour spécifier la couleur des points de données et des lignes sur le graphique.

Exercice d'application 4

Courbes multiples

Quel symbole représente le graphique suivant ?

```
x1 <- c(1, 10)
y1 <- c(1, 10)
x2 <- c(1, 10)
y2 <- c(10, 1)
plot(x1, y1, type = "l")
lines(x2, y2, type="l")
```

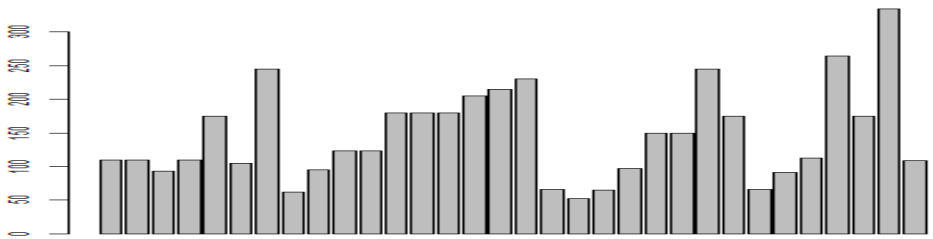
1. X
2. =
3. triangle
4. square

Bar chart 1

La fonction barplot peut être utilisée pour **tracer un diagramme à barres**.

Par exemple, créons un diagramme à barres pour la colonne hp de l'ensemble de données mtcars :

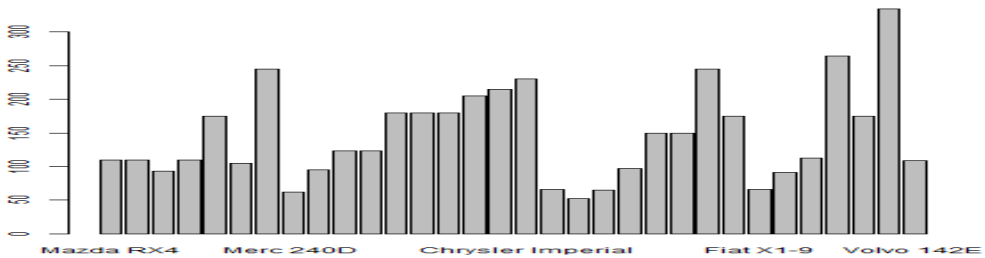
```
png(file = "p4.png")
barplot(mtcars$hp)
```



Bar chart 2

Afin de donner à chaque barre une étiquette correspondant aux noms des lignes de mtcars, nous pouvons définir le paramètre names.arg :

```
png(file = "p5.png")
barplot(mtcars$hp, names.arg = rownames(mtcars))
```



Attention !

rownames(mtcars) est utilisé pour sélectionner les noms de lignes de l'ensemble de données.

Exercice d'application 5

Bar chart

Quelle est l'étiquette de la barre ayant la valeur 42 sur le graphique suivant ?

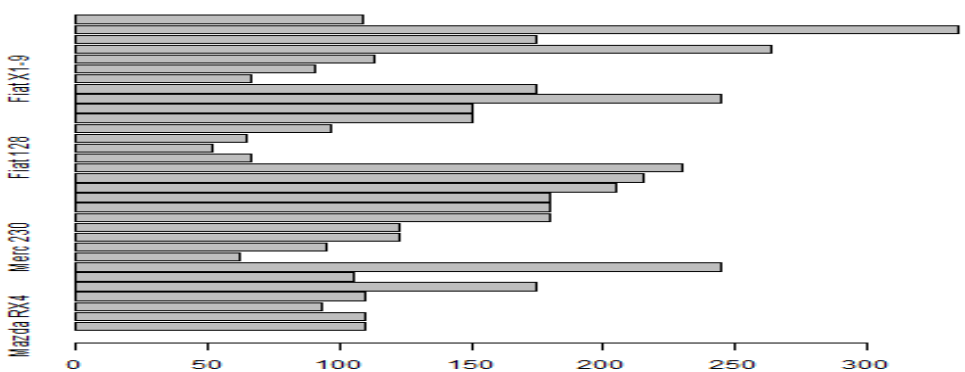
```
data <- c(10, 42, 8, 100)
x <- c("B", "Z", "Y", "A")
barplot(data, names.arg = x)
```

1. Z
2. A
3. B
4. Y

Bar chart 3

Les diagrammes à barres peuvent également être horizontaux.
Pour réaliser un diagramme à barres horizontal, il suffit de spécifier le paramètre horiz :

```
png(file = "p6.png")
barplot(mtcars$hp, horiz = TRUE, names.arg=rownames(mtcars))
```



Bar chart 4

Attention !

Comme pour les graphiques linéaires, le paramètre col peut être utilisé pour définir la couleur des barres :
barplot(mtcars$hp, col="blue")

Exercice d'application 6

horizontal bar chart

Remplissez les espaces vides (avec x, line, barplot, col, bar, data, horiz, y) pour réaliser un diagramme à barres horizontales avec les valeurs stockées dans le vecteur "data" et donnez-leur des étiquettes à partir du vecteur "x". Faites en sorte que les barres soient rouges.

[.....] (data, names.arg = [.....] , [.....] =TRUE, [.....] = "red")

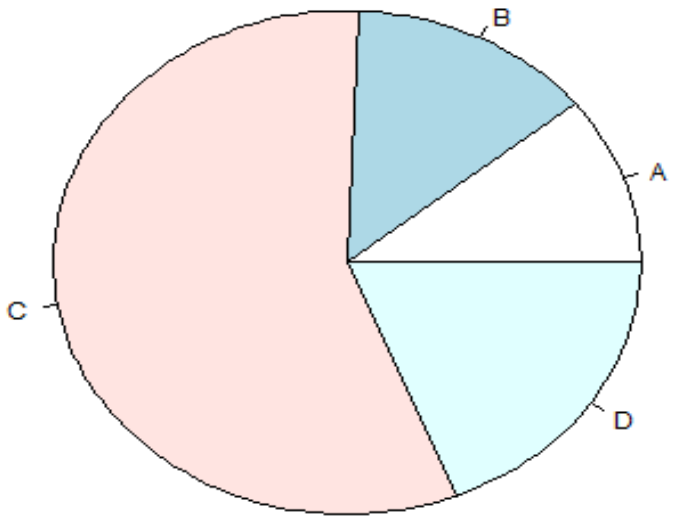
Pie chart 1

La fonction pie() est utilisée pour **créer un graphique circulaire**.

Elle prend comme paramètres un vecteur de valeurs pour les quartiers et des étiquettes pour chacun des quartiers.

Pie chart 2

```
png(file = "p7.png")
x <- c(8, 10, 42, 14)
y <- c("A", "B", "C", "D")
pie(x, label = y)
```



Pie chart 3

Attention !

Comme pour les autres fonctions graphiques, vous pouvez utiliser le paramètre main pour définir un titre de graphique.

Exercice d'application 7

Pie chart

Combien de sections le diagramme circulaire suivant aura-t-il ?

pie(1:8)

.....

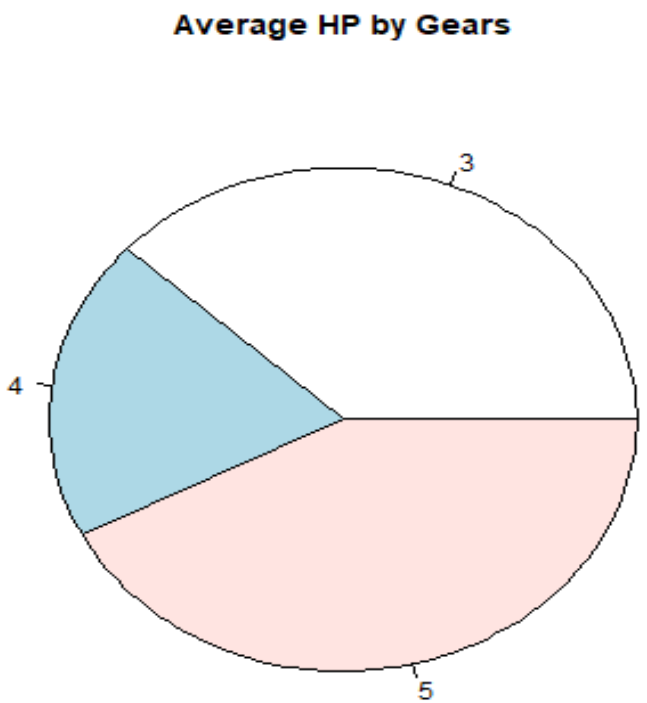
Pie chart 4

Utilisons l'ensemble de données mtcars pour construire un véritable graphique circulaire, montrant quelques informations sur les données.

Créons un graphique circulaire pour la puissance (hp) moyenne de chaque groupe de vitesse (gear).

Pie chart 5

```
x <- tapply(mtcars$hp, mtcars$gear, mean)
labels <- names(x)
png(file = "p8.png")
pie(x, label = labels, main="Average HP by Gears")
```



Pie chart 6

Nous utilisons la **fonction tapply** pour **regrouper les données par la colonne des vitesses** et **appliquons la fonction moyenne sur la colonne hp**.

Nous prenons **les étiquettes** à l'aide de la **fonction names()** et les utilisons pour notre graphique circulaire.

Attention !

Comme le résultat de tapply est un tableau, nous pouvons simplement le passer à la fonction pie() comme valeurs de parts ou de quartiers.

Exercice d'application 8

Autre pie chart

Que représentera le diagramme circulaire suivant ?

```
x <- tapply(mtcars$wt, mtcars$vs, length)
pie(x)
```

1. Nombre de voitures pour chaque groupe "vs"
2. Poids (wt) moyen de chaque groupe "vs"
3. Nombre total de voitures dans l'ensemble de la série
4. Nombre de véhicules pour chaque groupe "wt"

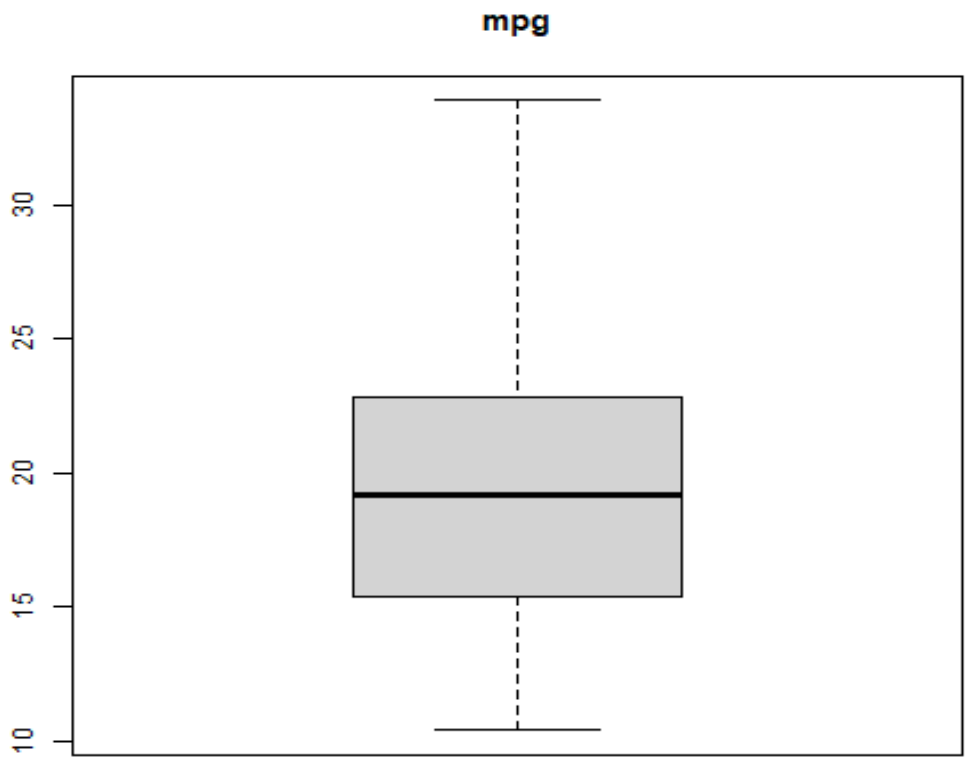
Boxplot (boîte à moustaches) 1

Un boxplot montre **la distribution des données**.
Il indique le minimum, le maximum, la médiane, le premier quartile et le troisième quartile de l'ensemble des données.

Dans R, vous pouvez **créer un boxplot** à l'aide de la **fonction boxplot()**.

Boxplot (boîte à moustaches) 2

```
png(file = "p9.png")
boxplot(mtcars$mpg)
```



Boxplot (boîte à moustaches) 3

Le code ci-dessus va dessiner un boxplot pour la colonne mpg.

Il affiche la moyenne des données, ainsi que les valeurs min/max.

Exercice d'application 8

Boxplot (boîte à moustaches)

La valeur moyenne est représentée dans le boxplot par une :

1. une ligne verticale en pointillés
2. une ligne supérieure
3. une boîte grise
4. une ligne horizontale noire

Histogramme 1

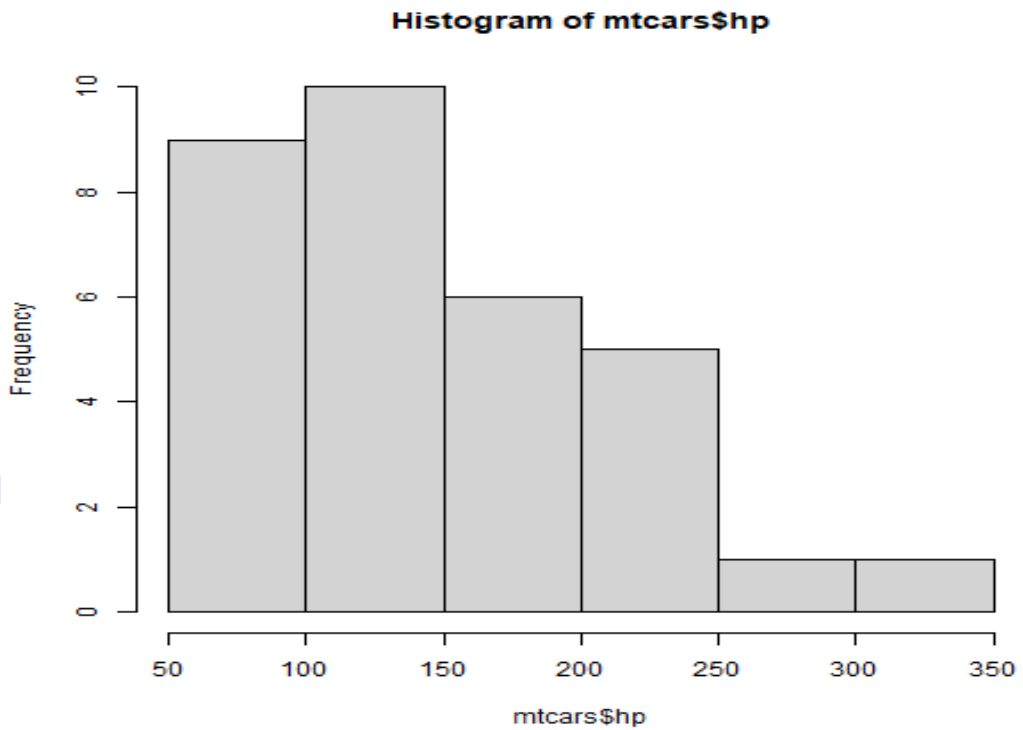
Un histogramme représente l**es fréquences des valeurs regroupées en plages**. Il est similaire à un diagramme à barres, mais il regroupe les valeurs dans des plages continues.

La fonction hist() est utilisée pour **créer un histogramme**.

Créons-en un pour la colonne hp de mtcars.

Histogramme 2

```
png(file = "p10.png")
hist(mtcars$hp)
```



Les fourchettes des buckets (plages) sont automatiquement définies en fonction des données.

Exercice d'application 9

Histogramme

Remplissez les espaces vides pour créer un histogramme pour la colonne "wt" de l'ensemble de données mtcars :

..... (..... $ .....)

Quiz 1

1) Combien de courbes le graphique suivant aura-t-il ?

```
a <- c(3, 2, 8, 9)
b <- c(7, 5)
c <- c(4, 8, 2, 10)
plot(a, type = "l")
lines(b, type="l")
lines(c, type="l")
```

.....

Quiz 2

2) Color

Quel paramètre est utilisé pour définir la couleur des points, des lignes et des barres d'un graphique ?

1. col
2. color
3. main
4. arg
5. names

Quiz 3

3) Pie chart

Remplissez les espaces vides (avec names, pie, label, y, barplot, x, plot) pour créer un graphique circulaire basé sur les données du vecteur "x", en utilisant le vecteur "y" pour les étiquettes.

[.....] ([.....] , [.....] = y)

Quiz 4

4) Boxplot

Vrai ou Faux : Un boxplot est visuellement similaire à un diagramme à barres.

1. Faux
2. Vrai

Quiz 5

5) colored a point

Remplissez les espaces vides pour dessiner un point vert dont les coordonnées sont x=3, y=8.

[.....] ([.....] , [.....] , [.....] =”green”)

Classer les 10 chiffres 1

Attention !

Exercice d'application 9

Interprétation

Regardons l'enfant gauche du nœud ?

1. 144
2. 532
3. 170