Guide d'introduction à R et RStudio

Dans le cadre du cours

Méthodes statistiques avancées

HEC MONTRÉAL

Ce document a pour objectif de vous familiariser avec les bases de R et de RStudio, deux outils essentiels pour l'analyse de données, la modélisation statistique et la visualisation.

Table of contents

1. Introduction R et RStudio

R est un langage de programmation libre principalement utilisé pour l'analyse de données, les statistiques et la visualisation. Dans ce cours, on l'utilisera dans RStudio, un environnement de développement intégré conçu spécialement pour R.

RStudio est composé de 4 zones principales :

- **Script** : La zone destinée à l'écriture du code
- **Console** : La zone destinée à l'exécution du code
- **Environnement** : La zone pour visualiser les jeux de données et objets créés
- **Plots / Fichiers / Packages** : La zone pour visualiser certains graphiques, gérer les packages et les fichiers

2. Bases du langage R

2.1 Commentaires et langages

En R, les commentaires sont précédés par le symbole #. Tout ce qui est écrit après ce symbole sur une ligne ne sera pas exécuté. Il est important de commenter son code, car cela permet d'avoir un travail structuré et aide une personne externe à comprendre ce qui a été fait.

De plus, R est sensible à la casse, ce qui signifie qu'il fait la différence entre les majuscules et les minuscules. *Maman* et *maman* seront considérés comme deux objets différents.

```
#Bonjour
#Hello
```

2.2 Créer une variable

En R, on utilise = pour affecter une valeur à une variable ; <- fonctionne aussi.

Ici, on crée deux variables, x et y, puis une troisième z qui stocke leur somme. On affiche le résultat avec print().

Il est important de noter que le symbole égal peut aussi être utilisé, non pas pour affecter une valeur, mais pour comparer deux valeurs. Dans ce cas, on utilise ==, qui renvoie TRUE si les deux valeurs sont égales, et FALSE sinon.

```r
#Assignation
 x = 5
 y =3
 z =x + y
 print(z)
```

```
[1] 8
```

```r
#Comparaison
 x==y
```

```
[1] FALSE
```

2.3 Types de données

R manipule plusieurs types de données :

- **Numeric** : nombres entiers et décimaux (ex: 5,3.2)
- **Character** : chaînes de caractères entre guillemets (ex : “hello”)
- **Logical** : booléens TRUE ou FALSE

```r
class(5)
```

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

```r
class(3.2)
```

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

```r
class("bonjour")
```

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

```r
class(TRUE)
```

```
[1] "logical"
```

2.4 Créer un vecteur

Un vecteur est une suite de valeurs du même type. Ici, notes est un vecteur contenant trois valeurs numériques.

```r
notes =c(12, 15, 18)
print(notes)
```

```
[1] 12 15 18
```

2.5 Valeurs manquantes

En R, une valeur manquante est représentée par NA. Cela indique que la donnée est absente, inconnue ou non collectée.

```
# Cette opération renvoie NA car f est une valeur manquante
e=10
f=NA
g=12
```

2.6 Indexation

En R, l'indexation commence à 1. Cela signifie que le premier élément d'un vecteur est à la position [1], contrairement à d'autres langages comme Python, où elle commence à [0].

```
notes[1]

[1] 12

notes[0]

numeric(0)
```

3. Créer ou importer des données

Afin de faire des analyses en R, on a besoin de disposer de jeux de données. Deux solutions s'offrent à nous :

- **Créer manuellement** : un tableau de données (utile pour des tests simples)
- **Importer un fichier** : un jeu de données existant, externe à R

3.1 Créer un data frame

Les tableaux de données créés en R sont des dataframes. Ils disposent de colonnes présentant les variables et de lignes présentant les observations.

On crée un jeu de données avec 3 variables : nom, âge et note.

```
etudiants =data.frame(
  nom = c("Hélène", "Pierre ", "Myriam","Imran"),
  age = c(21, 34, 22,25),
  note = c(15, 12, 17,15)
)
```

3.2 Importer un fichier CSV

Importer un fichier externe de type CSV

- sep = “,” indique que les valeurs sont séparées par des virgules. Ça pourrait également être “;”
- header = TRUE signifie que la première ligne contient les noms de colonnes.

  Il suffit d’adapter le chemin du fichier à son emplacement sur l’ordinateur.

```
data =read.csv("chemin/vers/fichier.csv", sep = ",", header = TRUE)
```

4. Explorer les données

Différents outils permettent de faire une analyse exploratoire préliminaire du jeu de données :

- **head()** : affiche les 3 premières lignes du tableau
- **str()** : affiche la structure de l’objet ; pour chaque variable, on a son type et des exemples de valeurs
- **summary()** : affiche les statistiques descriptives basiques de chacune des variables

```
head(etudiants)
```

```
      nom age note
1  Hélène  21   15
2 Pierre   34   12
3  Myriam  22   17
4   Imran  25   15
```

```
str(etudiants)
```

```
'data.frame':   4 obs. of  3 variables:
 $ nom : chr  "Hélène" "Pierre " "Myriam" "Imran"
 $ age : num  21 34 22 25
 $ note: num  15 12 17 15
```

```
summary(etudiants)
```

```
     nom                 age             note
 Length:4           Min.   :21.00   Min.   :12.00
 Class :character   1st Qu.:21.75   1st Qu.:14.25
 Mode  :character   Median :23.50   Median :15.00
                    Mean   :25.50   Mean   :14.75
```

```
                  3rd Qu.:27.25   3rd Qu.:15.50
                  Max.   :34.00   Max.   :17.00
```

5. Manipuler les données

On peut également manipuler les données, en triant, filtrant ou sélectionnant seulement une partie des données.

```
# On affiche uniquement la variable note du jeu de données étudiants
etudiants$note
```

```
[1] 15 12 17 15
```

```
# On affiche uniquement la première ligne du jeu de données
etudiants[1, ]
```

```
     nom age note
1 Hélène  21   15
```

```
# On trie les observations par ordre croissant d'âge
etudiants[order(etudiants$age), ]
```

```
      nom age note
1  Hélène  21   15
3  Myriam  22   17
4   Imran  25   15
2 Pierre   34   12
```

```
# On trie les observations par ordre décroissant d'âge
etudiants[order(etudiants$age, decreasing = TRUE), ]
```

```
      nom age note
2 Pierre   34   12
4   Imran  25   15
3  Myriam  22   17
1  Hélène  21   15
```

```
# On affiche uniquement les étudiants ayant obtenu une note supérieure à 14
subset(etudiants, note > 14)
```

```
     nom age note
1 Hélène  21   15
3 Myriam  22   17
4  Imran  25   15
```

6. Créer et modifier des variables

On peut créer de nouvelles variables ou transformer des variables existantes.

- = : permet de créer une nouvelle variable qui correspond à la modification d'une variable existante par un calcul
- ifelse(condition, valeur_si_vrai, valeur_si_faux) : permet de créer une nouvelle variable selon des conditions
- as.character, as.factor, as.integer : permettent de transformer une variable en un type différent

```
# On crée une nouvelle variable appelée noteplus2 qui ajoute 2 points à la
note de chaque étudiant
etudiants$noteplus2 = etudiants$note + 2

# On crée une nouvelle variable selon une condition
etudiants$mention = ifelse(etudiants$note >= 15, "Bien", "Passable")

# On change le type de la variable age en caractère
etudiants$age = as.character(etudiants$age)
```

7. Variables catégorielles

Quand on travaille avec des données qualitatives (comme le niveau d'études, le sexe ou une catégorie), il faut les convertir en factor. En R, un factor est une variable catégorielle avec des niveaux bien définis.

Pour ce faire, on utilise factor() qui crée une nouvelle variable de type facteur. L'argument labels() permet d'ajouter des descriptions à chacun des niveaux.

```
# On crée une nouvelle variable niveau, qui donne le niveau d'étude de chaque
étudiant
etudiants$niveau = factor(
  c("Licence", "Master", "Licence", "Licence"),
  levels = c("Licence", "Master"),
  labels = c("Bac+3", "Bac+5")
)
```

8. Statistiques descriptives

Comme expliqué précédemment, afin d'afficher les statistiques descriptives d'un jeu de données, on utilise summary.

- mean() : permet d'obtenir la moyenne d'une variable donnée
- median() : permet d'obtenir la médiane d'une variable donnée
- sd() : permet d'obtenir l'écart type d'une variable donnée

```
summary(etudiants)
```

```
     nom                 age                 note          noteplus2    
 Length:4           Length:4           Min.   :12.00   Min.   :14.00  
 Class :character   Class :character   1st Qu.:14.25   1st Qu.:16.25  
 Mode  :character   Mode  :character   Median :15.00   Median :17.00  
                                       Mean   :14.75   Mean   :16.75  
                                       3rd Qu.:15.50   3rd Qu.:17.50  
                                       Max.   :17.00   Max.   :19.00  
   mention             niveau  
 Length:4           Bac+3:3  
 Class :character   Bac+5:1  
 Mode  :character            
```

```
mean(etudiants$note)
```

```
[1] 14.75
```

```
median(etudiants$note)
```

```
[1] 15
```

```
sd(etudiants$note)
```

```
[1] 2.061553
```

9. Les librairies

En R, une librairie est un module additionnel qui ajoute des fonctionnalités qui ne sont pas En R, une librairie est un module additionnel qui ajoute des fonctionnalités non incluses directement dans R.

Ces packages permettent de faire des analyses variées, précises et spécialisées. Chaque package est donc "spécialisé" dans une tâche précise.

On installe un package une seule fois avec install.packages(), puis on le charge à chaque session avec library().

10. Visualisation avec ggplot2

Afin de faire des visualisations sur R, l'option intégrée plot() existe, mais elle offre des visualisations simples. Pour obtenir des visualisations plus sophistiquées, on peut utiliser la librairie ggplot2.

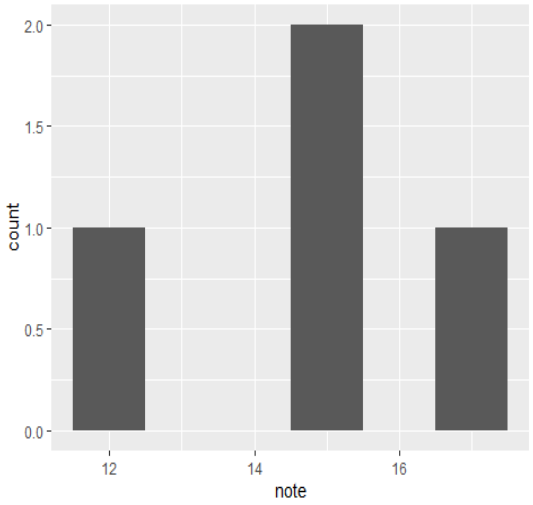
```
# On installe le package ggplot2 si ce n'est pas déjà fait
# install.packages("ggplot2")

# On charge la librairie ggplot2
library(ggplot2)
```

10.1 Histogramme

Utile pour observer la répartition d'une variable numérique.

Chaque barre représente le nombre d'étudiants ayant une note dans une certaine plage.

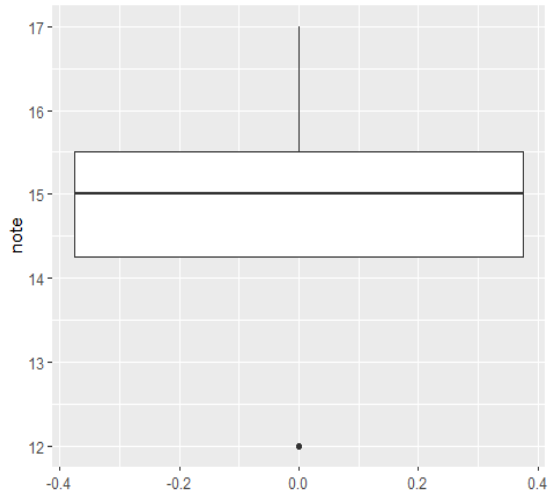
```
ggplot(etudiants, aes(x = note)) +
  geom_histogram(binwidth = 1)
```



10.2 Boxplot

Permet de visualiser la distribution d'une variable, les médianes, les valeurs extrêmes .

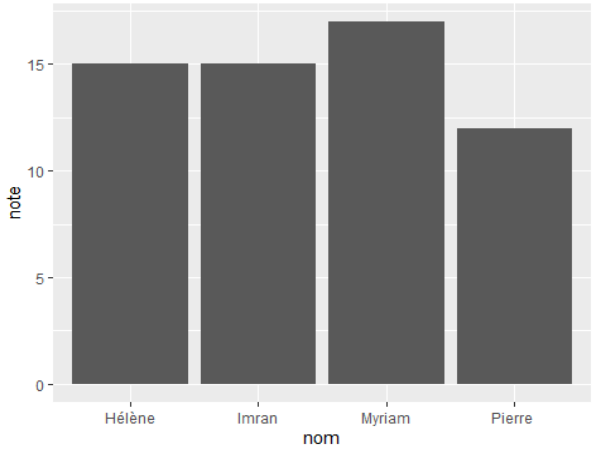
```
ggplot(etudiants, aes(y = note)) +
  geom_boxplot()
```



10.3 Barplot

Pour comparer des valeurs individuelles (par exemple, les notes par étudiant)

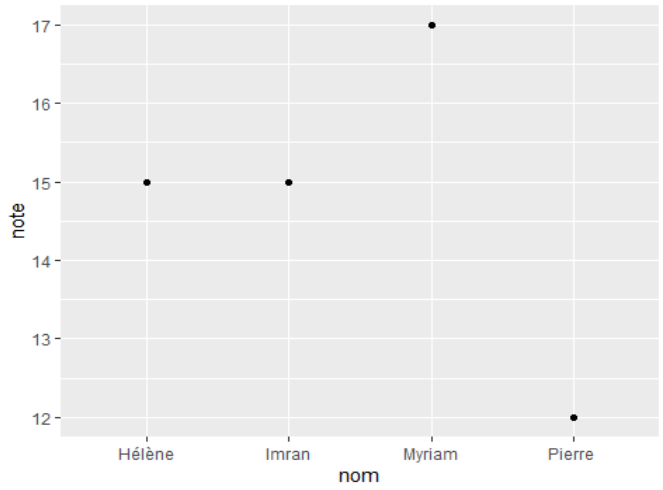
```
ggplot(etudiants, aes(x = nom, y = note)) +
  geom_bar(stat = "identity")
```



10.4 Scatterplot

Chaque point représente une observation, avec ses coordonnées correspondant à deux variables.

```
ggplot(etudiants, aes(x = nom, y = note)) +
  geom_point()
```



11. Manipulation avec dplyr

Dplyr est un package qui permet de manipuler plus facilement les jeu de donnee

- filter(note > 14) : garde seulement les lignes où la note est supérieure à 14.
- arrange(desc(note)) : trie les notes par ordre décroissant
- mutate(...) : crée une nouvelle variable appelée note_plus_1, où on ajoute 1 point à la note.

Le symbole %>% permet d'enchaîner les differentes opérations.

```
# On installe le package dplyr si ce n'est pas déjà fait
# install.packages("dplyr")

# On charge la librairie dplyr
library(dplyr)



etudiants %>%
  filter(note > 14) %>%
  arrange(desc(note)) %>%
  mutate(note_plus_1 = note + 1)
```

```
     nom age note noteplus2 mention niveau note_plus_1
1 Myriam  22   17        19    Bien  Bac+3          18
2 Hélène  21   15        17    Bien  Bac+3          16
3  Imran  25   15        17    Bien  Bac+3          16
```

12. Analyse de variance (ANOVA)

L'analyse de variance permet de comparer les moyennes d'un même indicateur (par exemple une note) entre plusieurs groupes (comme les niveaux d'étude). C'est utile quand on veut savoir si les différences entre groupes sont statistiquement significatives.

On utilise la fonction aov() pour réaliser le test ANOVA. summary() permet d'afficher les résultats, notamment la p-value.

```
anova_result =aov(note ~ niveau, data = etudiants)

summary(anova_result)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
niveau       1 10.083  10.083   7.563  0.111
Residuals    2  2.667   1.333
```

13. Régression linéaire

La régression linéaire sert à étudier la relation entre des variables numériques. On utilise lm() pour la réaliser. A ~ B signifie qu'on cherche à prédire A en fonction de B.

13.1 Régression linéaire simple

```
modele_simple =lm(note ~ age, data = etudiants)
summary(modele_simple)
```

```
Call:
lm(formula = note ~ age, data = etudiants)

Residuals:
ALL 4 residuals are 0: no residual degrees of freedom!

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.500e+01        NaN     NaN      NaN
age22        2.000e+00        NaN     NaN      NaN
age25       -1.088e-15        NaN     NaN      NaN
age34       -3.000e+00        NaN     NaN      NaN

Residual standard error: NaN on 0 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:    NaN
F-statistic:   NaN on 3 and 0 DF,  p-value: NA
```

13.2 Régression linéaire multiple

Régression linéaire multiple : si on veut un modèle qui prend en compte toutes les variables explicatives, on met un point "." après le ~.

```
modele_multiple =lm(note ~ age+niveau, data = etudiants)
summary(modele_multiple)
```

```
Call:
lm(formula = note ~ age + niveau, data = etudiants)

Residuals:
ALL 4 residuals are 0: no residual degrees of freedom!

Coefficients: (1 not defined because of singularities)
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.500e+01        NaN     NaN      NaN
age22        2.000e+00        NaN     NaN      NaN
age25       -1.088e-15        NaN     NaN      NaN
age34       -3.000e+00        NaN     NaN      NaN
niveauBac+5         NA         NA      NA       NA

Residual standard error: NaN on 0 degrees of freedom
Multiple R-squared:      1, Adjusted R-squared:    NaN
F-statistic:   NaN on 3 and 0 DF,  p-value: NA
```

Afin de réaliser un test ANOVA pour comparer les modèles, cette fois on utilise anova().

```
anova(modele_multiple,modele_simple)
```

```
Analysis of Variance Table

Model 1: note ~ age + niveau
Model 2: note ~ age
  Res.Df RSS Df Sum of Sq F Pr(>F)
1      0   0
2      0   0  0         0
```

14. Régression logistique

Quand la variable à prédire est binaire (par exemple : réussite ou échec), on utilise une régression logistique. Pour ce faire, on utilise la fonction glm().

glm permet de faire un modèle linéaire généralisé, qui comprend les modèles logistiques, de Poisson, etc.
Pour préciser qu'on fait un modèle logistique, on utilise l'option family = "binomial".

```r
# On crée une variable binaire "reussite" : elle prend la valeur 0 pour échec et 1 pour réussite
etudiants$reussite = ifelse(etudiants$note >= 12, 1, 0)

modele_logit =glm(reussite ~ age, data = etudiants, family = "binomial")
summary(modele_logit)


Call:
glm(formula = reussite ~ age, family = "binomial", data = etudiants)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) 2.357e+01  7.946e+04       0        1
age22       1.293e-09  1.124e+05       0        1
age25       1.293e-09  1.124e+05       0        1
age34       1.293e-09  1.124e+05       0        1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 0.000e+00  on 3  degrees of freedom
Residual deviance: 4.661e-10  on 0  degrees of freedom
AIC: 8

Number of Fisher Scoring iterations: 22
```

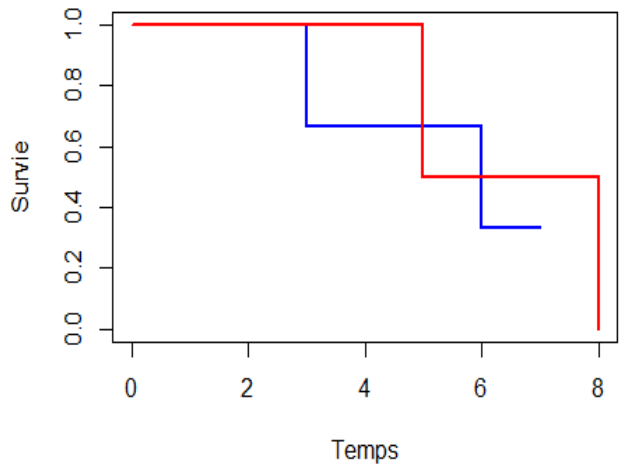
15. Analyse de survie

L'analyse de survie est utilisée pour étudier le temps avant un événement, par exemple le temps avant qu'un étudiant quitte un programme. Cette méthode prend en compte les cas où l'événement n'est pas encore survenu (appelé censure).

- Surv(temps, event) crée l'objet de survie : temps est la durée, event indique si l'événement a eu lieu (1) ou non (0).
- survfit() calcule la probabilité de survie à chaque instant.
- plot() trace la courbe de survie.

```r
# On installe le package survival si ce n'est pas déjà fait
# install.packages("survival")

# On charge la librairie survival
library(survival)

# On crée trois vecteurs pour simuler un jeu de données de survie

# Temps de survie
```

```r
temps = c(5, 8, 4, 6, 3, 7)

# Événement (1 = décès, 0 = censuré)
event = c(1, 1, 0, 1, 1, 0)

# Sexe
sexe = c("Homme", "Homme", "Homme", "Femme", "Femme", "Femme")

Objet_survie= Surv(temps, event)

fit =survfit(Objet_survie ~ sexe)

plot(fit, xlab = "Temps", ylab = "Survie", col = c("blue", "red"), lwd = 2)
```



Pour comparer deux courbes de survie, plusieurs tests existent, notamment le test du Log-Rank.

```r
test_logrank =survdiff(Objet_survie ~ sexe)
print(test_logrank)

Call:
survdiff(formula = Objet_survie ~ sexe)

             N Observed Expected (O-E)^2/E (O-E)^2/V
sexe=Femme 3        2     1.67    0.0667     0.154
sexe=Homme 3        2     2.33    0.0476     0.154

 Chisq= 0.2  on 1 degrees of freedom, p= 0.7
```

15.1 Modèle de Cox

Le modèle de Cox (est une méthode d'analyse de survie permettant d'estimer l'effet de covariables sur le risque d'occurrence d'un événement dans le temps.

```
# Ajuster un modèle de Cox avec la variable sexe
modele_cox =coxph(Objet_survie ~ sexe)

# Résumé du modèle
summary(modele_cox)
```

```
Call:
coxph(formula = Objet_survie ~ sexe)

  n= 6, number of events= 4

             coef exp(coef) se(coef)      z Pr(>|z|)
sexeHomme -0.4812    0.6180   1.2380 -0.389    0.697

          exp(coef) exp(-coef) lower .95 upper .95
sexeHomme     0.618      1.618    0.0546     6.995

Concordance= 0.6  (se = 0.181 )
Likelihood ratio test= 0.16  on 1 df,   p=0.7
Wald test            = 0.15  on 1 df,   p=0.7
Score (logrank) test = 0.15  on 1 df,   p=0.7
```

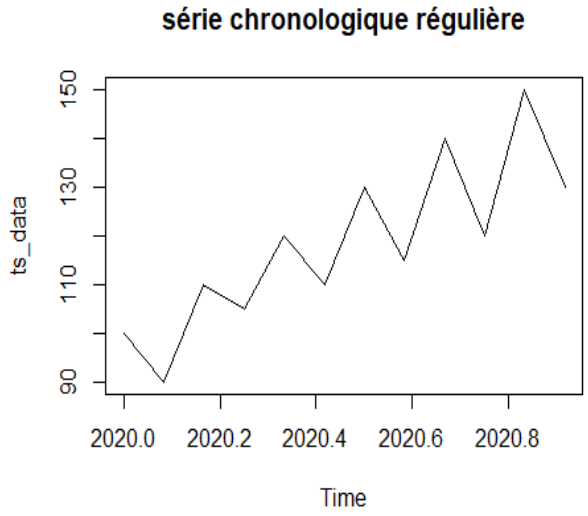
16. Séries chronologiques

Les séries chronologiques sont des données numériques observées dans le temps, à intervalles réguliers. Elles permettent d'évaluer l'évolution d'une variable dans le temps.

On peut utiliser ts, qui permet de créer et manipuler des séries simples à intervalles réguliers.

Le package zoo est, lui, utilisé pour manipuler des séries plus complexes avec des dates précises et des intervalles réguliers ou irréguliers.

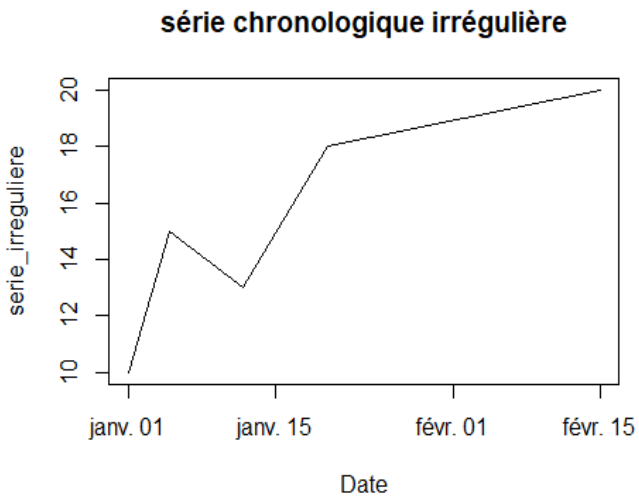
```
# On simule un jeu de données de série chronologique
ts_data = ts(c(100, 90, 110, 105, 120, 110, 130, 115, 140, 120, 150, 130),
start = c(2020), frequency = 12)


plot(ts_data, main = "série chronologique régulière")
```




```r
# On installe le package zoo si ce n'est pas déjà fait
# install.packages("zoo")

# On charge la librairie zoo
library(zoo)



# Quelques valeurs aléatoires
valeurs =c(10, 15, 13, 18, 20)

# Dates irrégulières
dates =as.Date(c("2023-01-01", "2023-01-05", "2023-01-12", "2023-01-20",
"2023-02-15"))

# Créer l'objet zoo
serie_irreguliere =zoo(valeurs, order.by = dates)

# Afficher la série
print(serie_irreguliere)
```

```
2023-01-01 2023-01-05 2023-01-12 2023-01-20 2023-02-15
        10         15         13         18         20
```

```r
# Tracer la série

plot(serie_irreguliere, xlab = "Date",main = "série chronologique
irrégulière")
```


série chronologique irrégulière
serie_irreguliere
10
12
14
16
18
20
janv. 01
janv. 15
févr. 01
févr. 15
Date