Université Hassan 1er

Ecole Nationale des Sciences Appliquées de Berrechid

Rapport des TPs de Data Mining

Réalisé par : KNINECH Fatima Ezzahra

Supervisé par: Pr. HRIMECH Hamid

Filière : ISIBD – S9

Année universitaire : 2025/2026

TP1 : Implémentation des arbres de décision avec Scikit-Learn

Dans le cadre de ce travail pratique, nous appliquons les arbres de décision à différents jeux de données afin de mettre en pratique les notions théoriques vues en cours. Les différentes étapes du TP incluent le chargement des données, leur prétraitement, l'entraînement du modèle ainsi que l'analyse des résultats obtenus. Cette démarche permet de mieux comprendre le comportement des arbres de décision et d'évaluer leur efficacité dans des situations concrètes.

Installation des librairies nécessaires :

```
pip install --upgrade scikit-learn
```

Python

```
Requirement already satisfied: scikit-learn in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (1.7.2)
Requirement already satisfied: numpy>=1.22.0 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn) (2.
Requirement already satisfied: scipy>=1.8.0 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn) (1.1
Requirement already satisfied: joblib>=1.2.0 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from scikit-learn) (1.
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from scikit-lea
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Vérification :

```
import sklearn
print(sklearn.__version__)
```

```
1.7.2
```

```
pip install pandas
```

Python

```
Requirement already satisfied: pandas in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (2.3.3)
Requirement already satisfied: numpy>=1.23.2 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from pandas) (2.3.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\oure.ma\appdata\roaming\python\python311\site-packages (from pandas) (2.9.0.post
Requirement already satisfied: pytz>=2020.1 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\oure.ma\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.8.2->pandas) (
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Vérification :

```
import pandas
print(pandas.__version__)
```

```
2.3.3
```

```
pip install xlrd
```

```
Requirement already satisfied: xlrd in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (2.0.2)
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 25.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Construction d'un arbre de décision (variables quantitatives)

Dans un premier temps, nous traitons un problème de classification dans lequel l'ensemble des variables explicatives est de nature quantitative. Afin de mieux comprendre le principe de fonctionnement de l'algorithme des arbres de décision, nous nous appuyons sur un jeu de données simple et largement utilisé.

Présentation du jeu de données Breast Cancer Wisconsin

Le jeu de données *Breast Cancer Wisconsin* regroupe des informations relatives à des cellules mammaires. L'objectif de cette étude est de prédire la variable cible « classe », qui indique si une tumeur est bénigne ou maligne, à partir de neuf variables numériques décrivant différentes caractéristiques cellulaires.

Importation et exploration des données

Le jeu de données contient 699 observations et 10 variables au total. Après son importation, une phase d'exploration est réalisée afin d'examiner la structure des données. Les premières lignes du dataset sont présentées afin d'obtenir une vision globale des attributs disponibles.

```
import sklearn
import pandas
import os
df = pandas.read_excel("breast.xlsx", sheet_name=0)
print(df.shape)
```

```
(699, 10)
```

```
print(df.head())
```

```
   clump  ucellsize  ucellshape  mgadhesion  sepics  bnuclei  bchromatin  \
0      5          1           1           1       2        1           3
1      5          4           4           5       7       10           3
2      3          1           1           1       2        2           3
3      6          8           8           1       3        4           3
4      4          1           1           3       2        1           3

   normnucl  mitoses  classe
0         1        1  benign
1         2        1  benign
2         1        1  benign
3         7        1  benign
4         1        1  benign
```

Types des variables

```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   clump        699 non-null    int64 
 1   ucellsize    699 non-null    int64 
 2   ucellshape   699 non-null    int64 
 3   mgadhesion   699 non-null    int64 
 4   sepics       699 non-null    int64 
 5   bnuclei      699 non-null    int64 
 6   bchromatin   699 non-null    int64 
 7   normnucl     699 non-null    int64 
 8   mitoses      699 non-null    int64 
 9   classe       699 non-null    object
dtypes: int64(9), object(1)
memory usage: 54.7+ KB
None
```

Distribution des classes

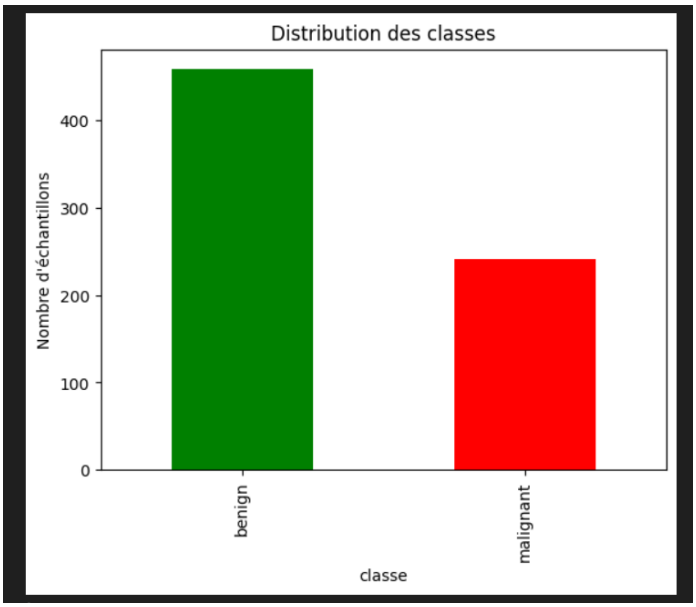
```
print(df.classe.value_counts())
```

```
classe
benign       458
malignant    241
Name: count, dtype: int64
```

```
print(df.classe.value_counts(normalize=True))
```

```
classe
benign       0.655222
malignant    0.344778
Name: proportion, dtype: float64
```

```
import matplotlib.pyplot as plt

df['classe'].value_counts().plot(kind='bar', color=['green','red'])
plt.title("Distribution des classes")
plt.ylabel("Nombre d'échantillons")
plt.show()
```



Partition apprentissage / test

```
from sklearn.model_selection import train_test_split

dfTrain, dfTest = train_test_split(df,test_size=300,random_state=1,stratify=df.classe)
print(dfTrain.shape)
print(dfTest.shape)
```

```
(399, 10)
(300, 10)
```

```
print("Distribution dans le train :")
print(dfTrain.classe.value_counts(normalize=True))

# Vérifier la distribution des classes dans test
print("\nDistribution dans le test :")
print(dfTest.classe.value_counts(normalize=True))
```

```
Distribution dans le train :
classe
benign       0.654135
malignant    0.345865
Name: proportion, dtype: float64

Distribution dans le test :
classe
benign       0.656667
malignant    0.343333
Name: proportion, dtype: float64
```

Instanciation de l'arbre de décision :

```
from sklearn.tree import DecisionTreeClassifier
arbreFirst = DecisionTreeClassifier(min_samples_split=30,min_samples_leaf=10)
```

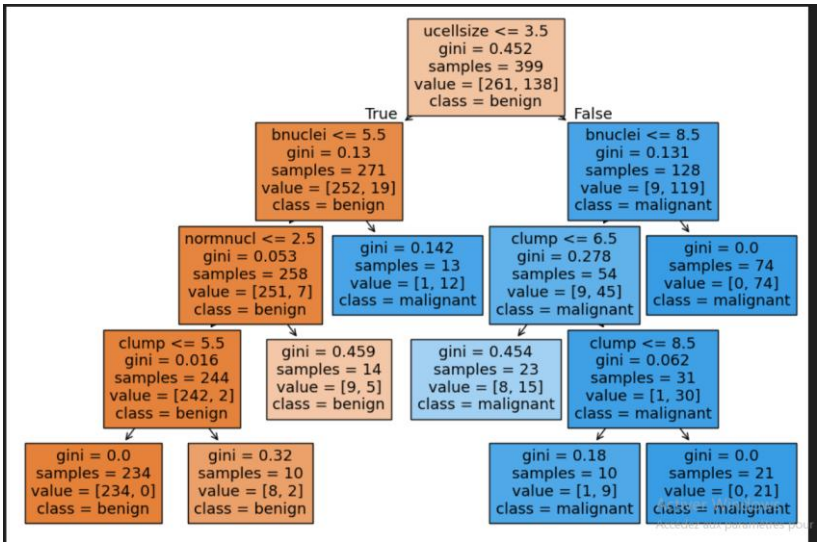
Apprentissage du modèle :

```
import numpy as np
arbreFirst.fit(np.nan_to_num(dfTrain.iloc[:,:-1]), y=dfTrain.classe)
```

DecisionTreeClassifier

▶ Parameters

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12,8))

plot_tree(
    arbreFirst,
    feature_names=list(df.columns[:-1]),
    class_names=arbreFirst.classes_,
    filled=True
)

plt.show()
```



Importance des variables :

```
import pandas as pd


impVarFirst = {
    "Variable": df.columns[:-1],
    "Importance": arbreFirst.feature_importances_
}

impVarDF = pd.DataFrame(impVarFirst).sort_values(by="Importance", ascending=False)

print(impVarDF)
```

```
     Variable  Importance
1   ucellsize    0.819165
5     bnuclei    0.137752
0       clump    0.022524
7    normnucl    0.020559
2  ucellshape    0.000000
4      sepics    0.000000
3  mgadhesion    0.000000
6  bchromatin    0.000000
8     mitoses    0.000000
```

Evaluation en test :

```
import numpy as np

predFirst = arbreFirst.predict(X=dfTest.iloc[:, :-1])
print(np.unique(predFirst, return_counts=True))
```

```
(array(['benign', 'malignant'], dtype=object), array([199, 101]))
```

Matrice de confusion :

```
from sklearn import metrics

print(metrics.confusion_matrix(dfTest.classe, predFirst))
```

```
[[191   6]
 [  8  95]]
```

Indicateurs de performances :

```
print(metrics.accuracy_score(dfTest.classe, predFirst))
```

```
0.9533333333333334
```

```
print(1.0 - metrics.accuracy_score(dfTest.classe, predFirst))
```

```
0.046666666666666634
```

```
print(metrics.recall_score(dfTest.classe, predFirst, pos_label='malignant'))
```

```
0.9223300970873787
```

```
from sklearn import metrics

print(metrics.precision_score(dfTest.classe, predFirst, pos_label='malignant'))
```

```
0.9405940594059405
```

```
from sklearn import metrics

print(metrics.f1_score(dfTest.classe, predFirst, pos_label='malignant'))
```

```
0.9313725490196079
```

Réduction de la taille de l'arbre

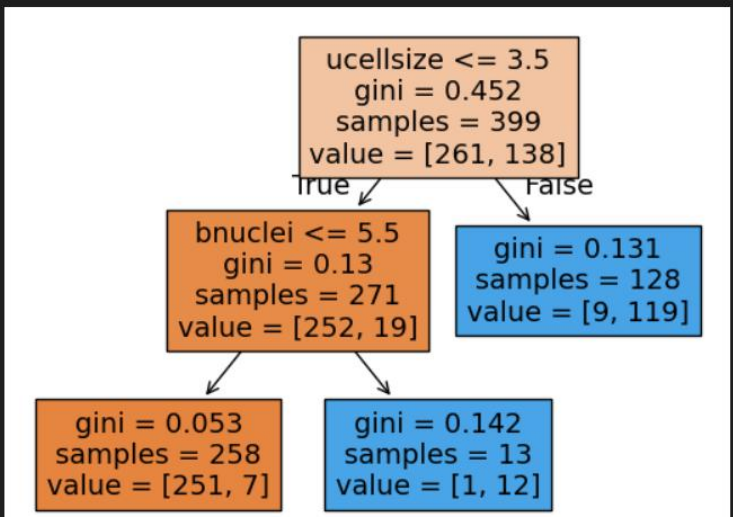
```
from sklearn.tree import DecisionTreeClassifier

arbreSecond = DecisionTreeClassifier(
    min_samples_split=30,
    min_samples_leaf=10,
    max_leaf_nodes=3
)
```

```
arbreSecond.fit(X=dfTrain.iloc[:, :-1], y=dfTrain.classe)
```

DecisionTreeClassifier

▶ Parameters

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plot_tree(arbreSecond, feature_names=list(df.columns[:-1]), filled=True)
plt.show()
```



L'arbre est fortement simplifié, tout en maintenant ses qualités prédictives :

```
predSecond = arbreSecond.predict(X=dfTest.iloc[:, :-1])
```

```
from sklearn import metrics

print(metrics.confusion_matrix(dfTest.classe, predSecond))
```

```
[[191   6]
 [  8  95]]
```

```
from sklearn import metrics

print(metrics.classification_report(dfTest.classe, predSecond))
```

```
              precision    recall  f1-score   support

      benign       0.96      0.97      0.96       197
   malignant       0.94      0.92      0.93       103

    accuracy                           0.95       300
   macro avg       0.95      0.95      0.95       300
weighted avg       0.95      0.95      0.95       300
```

B: Cas des prédictives exclusivement qualitatives :

Importation des données

```
import sklearn
import pandas
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import numpy as np
#importer les données import pandas
dfVote = pandas.read_excel("house.xlsx",sheet_name = 0)
print(dfVote.shape)
print(dfVote.info())
```

```
(435, 7)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 7 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   adoption_of_the_budget_re  435 non-null    object
 1   physician_fee_freeze       435 non-null    object
 2   mx_missile                 435 non-null    object
 3   superfund_right_to_sue     435 non-null    object
 4   crime                      435 non-null    object
 5   duty_free_exports          435 non-null    object
 6   groupe                     435 non-null    object
dtypes: object(7)
memory usage: 23.9+ KB
None
```

```
print(dfVote.head())
```

```
  adoption_of_the_budget_re physician_fee_freeze mx_missile  \
0                         n                    y          n
1                         n                    y          n
2                         y                    ?          n
3                         y                    n          n
4                         y                    n          n

  superfund_right_to_sue crime duty_free_exports      groupe
0                      y     y                 n  republican
1                      y     y                 n  republican
2                      y     y                 n    democrat
3                      y     n                 n    democrat
4                      y     y                 y    democrat
```

Construction de l’arbre après recodage

```
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np

dfVote.replace('?', np.nan, inplace=True)

for col in dfVote.columns[:-1]:
    dfVote[col] = LabelEncoder().fit_transform(dfVote[col].astype(str))
arbreVote = DecisionTreeClassifier(max_depth=3)



arbreVote.fit(X=dfVote.iloc[:,:-1], y=dfVote.groupe)
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
import pandas as pd
import numpy as np

dfVote.replace('?', np.nan, inplace=True)

for col in dfVote.columns[:-1]:
    dfVote[col] = LabelEncoder().fit_transform(dfVote[col].astype(str))
arbreVote = DecisionTreeClassifier(max_depth=3)
```

```
arbreVote.fit(X=dfVote.iloc[:,:-1], y=dfVote.groupe)
```

▾ DecisionTreeClassifier ❶ ❷

▶ Parameters

```
print(dfVote.columns)
```

```
Index(['adoption_of_the_budget_re', 'physician_fee_freeze', 'mx_missile',
       'superfund_right_to_sue', 'crime', 'duty_free_exports', 'groupe'],
      dtype='object')
```

```
import pandas as pd


dfVoteBit = pd.get_dummies(dfVote[dfVote.columns[:-1]])

print(dfVoteBit.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 6 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   adoption_of_the_budget_re  435 non-null    int64
 1   physician_fee_freeze       435 non-null    int64
 2   mx_missile                 435 non-null    int64
 3   superfund_right_to_sue     435 non-null    int64
 4   crime                      435 non-null    int64
 5   duty_free_exports          435 non-null    int64
dtypes: int64(6)
memory usage: 20.5 KB
None
```

Codage disjonctif complet des prédictives :

```
print(dfVote['crime'].value_counts(normalize=True))
```

```
crime
2    0.570115
0    0.390805
1    0.039080
Name: proportion, dtype: float64
```

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(['y','n','?'])
print(list(le.classes_))
```

```
[np.str_('?'), np.str_('n'), np.str_('y')]
```

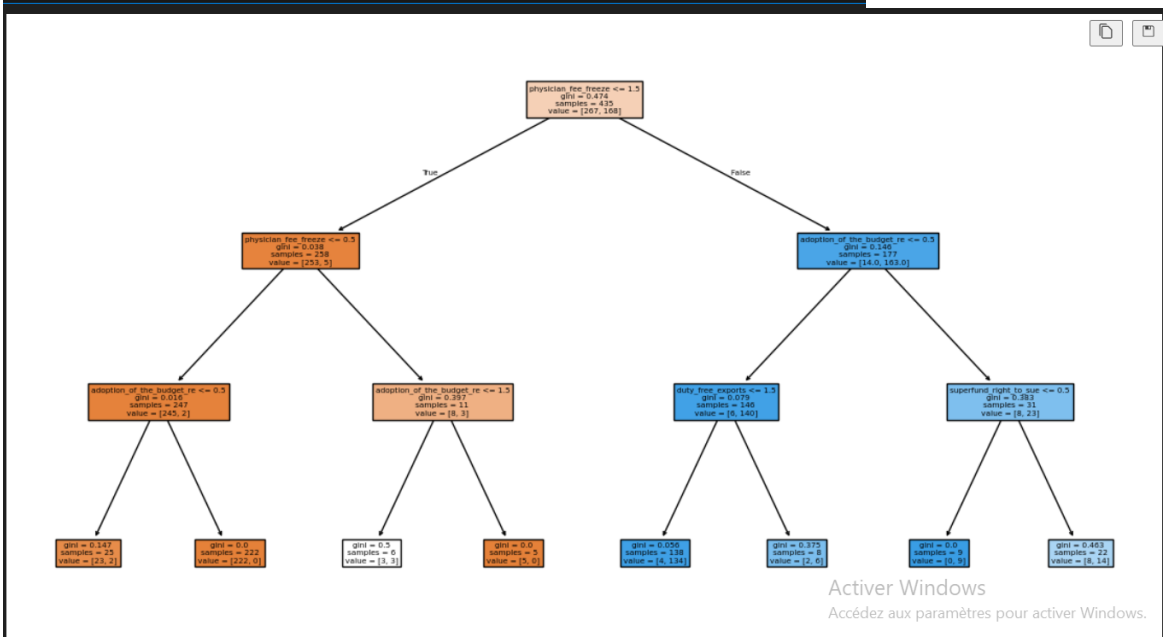
```
print(dfVoteBit.apply(np.mean, axis=0))
```

```
adoption_of_the_budget_re    1.188506
physician_fee_freeze         0.839080
mx_missile                   1.002299
superfund_right_to_sue       1.018391
crime                        1.179310
duty_free_exports            0.864368
dtype: float64
```

```
dfVoteBit['groupe']=dfVote.groupe
print(dfVoteBit.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435 entries, 0 to 434
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   adoption_of_the_budget_re  435 non-null    int64
 1   physician_fee_freeze       435 non-null    int64
 2   mx_missile                 435 non-null    int64
 3   superfund_right_to_sue     435 non-null    int64
 4   crime                      435 non-null    int64
 5   duty_free_exports          435 non-null    int64
 6   groupe                     435 non-null    object
dtypes: int64(6), object(1)
memory usage: 23.9+ KB
None
```

```python
from sklearn.tree import DecisionTreeClassifier

arbreVote = DecisionTreeClassifier(max_depth = 3)
arbreVote.fit(X = dfVoteBit.iloc[:,:-1], y = dfVoteBit.groupe)
plt.figure(figsize=(14,7.5))
plot_tree(arbreVote,feature_names = list(dfVoteBit.columns[:-1]),filled=True)
plt.show()
```



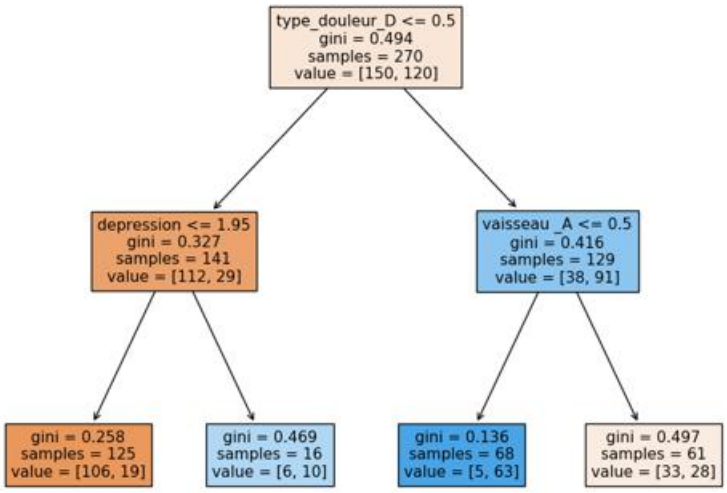
C: Cas des prédictives mixtes

Importation des données :

```python
import sklearn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree

print("Version scikit-learn :", sklearn.__version__)
dfHeart = pd.read_csv("heart.csv")
print("Shape du dataset :", dfHeart.shape)
print(dfHeart.info())
print("Colonnes du dataset :", dfHeart.columns.tolist())

col_cible = 'target'

lstQuali = dfHeart.select_dtypes(include=['object', 'category']).columns.tolist()
print("Colonnes qualitatives :", lstQuali)

if lstQuali:
    dfQualiEncoded = pd.get_dummies(dfHeart[lstQuali], drop_first=True)
else:
    dfQualiEncoded = pd.DataFrame()
print(dfQualiEncoded.info())

lstQuanti = dfHeart.select_dtypes(exclude=['object', 'category']).columns.tolist()
lstQuanti.remove(col_cible)
print("Colonnes quantitatives :", lstQuanti)

dfNew = pd.concat([dfQualiEncoded, dfHeart[lstQuanti]], axis=1)

dfNew['coeur'] = dfHeart[col_cible]
print(dfNew.info())
```

```python
arbreHeart = DecisionTreeClassifier(max_depth=2)
arbreHeart.fit(X=dfNew.iloc[:, :-1], y=dfNew['coeur'])


plt.figure(figsize=(12,8))
plot_tree(arbreHeart, feature_names=list(dfNew.columns[:-1]), filled=True)
plt.show()
```

```
Version scikit-learn : 1.7.2
Shape du dataset : (69, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69 entries, 0 to 68
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   survival  69 non-null     float64
 1   censors   69 non-null     float64
 2   age       69 non-null     float64
dtypes: float64(3)
memory usage: 1.7 KB
None
Colonnes du dataset : ['survival', 'censors', 'age']
Colonnes qualitatives : []
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 0 entries
Empty DataFrame
None
```



Tp 2: Recommandation personnalisée : système simplifié

Objectif du TP

Ce travail pratique a pour objectif de concevoir un système de recommandation personnalisé simple, fondé sur le calcul de similarité entre utilisateurs. Pour ce faire, nous utilisons le langage Python ainsi que les bibliothèques **pandas**, **numpy** et la mesure de similarité cosinus fournie par sklearn.metrics.pairwise.

Données utilisées

Les données sont fournies sous la forme d'une matrice utilisateur–produit stockée dans un fichier CSV intitulé *recommandation.csv*. Les colonnes représentent les produits, les lignes correspondent aux utilisateurs, et les valeurs associées indiquent les notes attribuées, comprises entre 0 et 5.

Étapes du TP

Partie 1---Chargement des données

1. Charger le fichier recommandation.csv avec pandas

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

df = pd.read_csv("recommandation.csv", index_col=0)
```

2. Afficher les premières lignes de la matrice

```
df.head()
```
✓ 0.0s

	Produit_1	Produit_2	Produit_3	Produit_4	Produit_5	Produit_6
Utilisateur_1	0	4	0	4	0	1
Utilisateur_2	0	2	2	0	3	2
Utilisateur_3	5	4	1	3	0	0
Utilisateur_4	0	0	0	0	3	0
Utilisateur_5	0	4	0	0	0	2

Aperçu général des données :

```
print("Dimensions de la matrice :", df.shape)
print("\nProduits :")
print(df.columns.tolist())
print("\nUtilisateurs :")
print(df.index.tolist())
```
✓ 0.0s Python

```
Dimensions de la matrice : (10, 6)

Produits :
['Produit_1', 'Produit_2', 'Produit_3', 'Produit_4', 'Produit_5', 'Produit_6']

Utilisateurs :
['Utilisateur_1', 'Utilisateur_2', 'Utilisateur_3', 'Utilisateur_4', 'Utilisateur_5', 'Utilisateur_6', 'Utilisateur_7', 'Utilisateur_8', 'Utilisat
```

Vérification des valeurs manquantes :

```
df.isnull().sum()
```
✓ 0.0s

```
Produit_1    0
Produit_2    0
Produit_3    0
Produit_4    0
Produit_5    0
Produit_6    0
dtype: int64
```

Partie 2 — Calcul de la similarité

1. Appliquer cosine_similarity entre les utilisateurs

```python
similarite_utilisateurs = cosine_similarity(df)

sim_user_df = pd.DataFrame(
    similarite_utilisateurs,
    index=df.index,
    columns=df.index
)

sim_user_df
```

✓ 0.0s Python

	Utilisateur_1	Utilisateur_2	Utilisateur_3	Utilisateur_4	Utilisateur_5	Utilisateur_6	Utilisateur_7	Utilisateur_8	Utilisateur_9	Utilisateur_10
Utilisateur_1	1.000000	0.379869	0.682521	0.000000	0.700649	0.372839	0.064651	0.165145	0.839782	0.938035
Utilisateur_2	0.379869	1.000000	0.305566	0.654654	0.585540	0.687322	0.162088	0.621059	0.592157	0.419961
Utilisateur_3	0.682521	0.305566	1.000000	0.000000	0.500979	0.335196	0.650064	0.000000	0.633300	0.835400
Utilisateur_4	0.000000	0.654654	0.000000	1.000000	0.000000	0.629941	0.000000	0.316228	0.301511	0.000000
Utilisateur_5	0.700649	0.585540	0.500979	0.000000	1.000000	0.281718	0.166091	0.424264	0.809040	0.688530
Utilisateur_6	0.372839	0.687322	0.335196	0.629941	0.281718	1.000000	0.467908	0.796819	0.303895	0.266711
Utilisateur_7	0.064651	0.162088	0.650064	0.000000	0.166091	0.467908	1.000000	0.352332	0.000000	0.178685
Utilisateur_8	0.165145	0.621059	0.000000	0.316228	0.424264	0.796819	0.352332	1.000000	0.095346	0.000000
Utilisateur_9	0.839782	0.592157	0.633300	0.301511	0.809040	0.303895	0.000000	0.095346	1.000000	0.870388
Utilisateur_10	0.938035	0.419961	0.835400	0.000000	0.688530	0.266711	0.178685	0.000000	0.870388	1.000000

2. Créer une matrice utilisateur-utilisateur (symétrique)

```python
sim_user_df.round(3)
```

✓ 0.0s Python

	Utilisateur_1	Utilisateur_2	Utilisateur_3	Utilisateur_4	Utilisateur_5	Utilisateur_6	Utilisateur_7	Utilisateur_8	Utilisateur_9	Utilisateur_10
Utilisateur_1	1.000	0.380	0.683	0.000	0.701	0.373	0.065	0.165	0.840	0.938
Utilisateur_2	0.380	1.000	0.306	0.655	0.586	0.687	0.162	0.621	0.592	0.420
Utilisateur_3	0.683	0.306	1.000	0.000	0.501	0.335	0.650	0.000	0.633	0.835
Utilisateur_4	0.000	0.655	0.000	1.000	0.000	0.630	0.000	0.316	0.302	0.000
Utilisateur_5	0.701	0.586	0.501	0.000	1.000	0.282	0.166	0.424	0.809	0.689
Utilisateur_6	0.373	0.687	0.335	0.630	0.282	1.000	0.468	0.797	0.304	0.267
Utilisateur_7	0.065	0.162	0.650	0.000	0.166	0.468	1.000	0.352	0.000	0.179
Utilisateur_8	0.165	0.621	0.000	0.316	0.424	0.797	0.352	1.000	0.095	0.000
Utilisateur_9	0.840	0.592	0.633	0.302	0.809	0.304	0.000	0.095	1.000	0.870
Utilisateur_10	0.938	0.420	0.835	0.000	0.689	0.267	0.179	0.000	0.870	1.000

3. Identifier les utilisateurs les plus similaires à un utilisateur donné

```python
def utilisateurs_similaires(utilisateur, n=3):
    return sim_user_df[utilisateur].sort_values(ascending=False)[1:n+1]

utilisateurs_similaires("Utilisateur_1")
```

✓ 0.0s

```
Utilisateur_10    0.938035
Utilisateur_9     0.839782
Utilisateur_5     0.700649
Name: Utilisateur_1, dtype: float64
```

Partie 3 — Génération de recommandations

1. Identifier les produits non notés par l'utilisateur

```python
def produits_non_notes(utilisateur):
    return df.columns[df.loc[utilisateur] == 0]

produits_non_notes("Utilisateur_1")
```

✓ 0.0s

```
Index(['Produit_1', 'Produit_3', 'Produit_5'], dtype='object')
```

2. Pondérer les notes des utilisateurs similaires pour recommander

```python
def recommander_produits(utilisateur, top_n=3):
    similaires = utilisateurs_similaires(utilisateur)
    produits_a_recommander = produits_non_notes(utilisateur)

    scores = {}

    for produit in produits_a_recommander:
        score = 0
        for autre_user, sim in similaires.items():
            score += sim * df.loc[autre_user, produit]
        scores[produit] = score

    return sorted(scores.items(), key=lambda x: x[1], reverse=True)[:top_n]
```

✓ 0.0s

Partie 4 — Analyse et interprétation

Quel produit recommander à l'utilisateur 1 ? Pourquoi ?

```python
recommander_produits("Utilisateur_1")
```

✓ 0.0s

```
[('Produit_1', np.float64(0.9380352942419313)),
 ('Produit_3', np.float64(0.9380352942419313)),
 ('Produit_5', np.float64(0.8397822097303648))]
```

```python
produit, score = recommander_produits("Utilisateur_1")[0]
print(f"Produit recommandé à Utilisateur1 : {produit}")
print(f"Score de recommandation : {score:.3f}")
```

✓ 0.0s

```
Produit recommandé à Utilisateur1 : Produit_1
Score de recommandation : 0.938
```

— L'utilisateur 2 a-t-il un profil similaire à un autre ?

```python
utilisateurs_similaires("Utilisateur_2")
```

✓ 0.0s

```
Utilisateur_6    0.687322
Utilisateur_4    0.654654
Utilisateur_8    0.621059
Name: Utilisateur_2, dtype: float64
```

— Quelle serait la prochaine action commerciale à proposer ?

Analyse commerciale :

- Proposer une promotion ou une recommandation ciblée sur le produit recommandé

- Offrir un bundle avec des produits similaires

- Envoyer une notification personnalisée à l'utilisateur

3. Bonus

— Appliquer le même principe pour les colonnes (produit-produit)

```
similarite_produits = cosine_similarity(df.T)

sim_produit_df = pd.DataFrame(
    similarite_produits,
    index=df.columns,
    columns=df.columns
)

sim_produit_df.round(3)
```

✓ 0.0s

	Produit_1	Produit_2	Produit_3	Produit_4	Produit_5	Produit_6
Produit_1	1.000	0.369	0.330	0.488	0.201	0.393
Produit_2	0.369	1.000	0.558	0.739	0.153	0.266
Produit_3	0.330	0.558	1.000	0.369	0.365	0.238
Produit_4	0.488	0.739	0.369	1.000	0.360	0.418
Produit_5	0.201	0.153	0.365	0.360	1.000	0.739
Produit_6	0.393	0.266	0.238	0.418	0.739	1.000

```
def produits_similaires(produit, n=3):
    return sim_produit_df[produit].sort_values(ascending=False)[1:n+1]

produits_similaires(df.columns[0])
```

✓ 0.0s

```
Produit_4    0.487869
Produit_6    0.393369
Produit_2    0.368795
Name: Produit_1, dtype: float64
```

— Ajouter un seuil de similarité minimale

```
def recommander_avec_seuil(utilisateur, seuil=0.3):
    similaires = sim_user_df[utilisateur]
    similaires = similaires[similaires > seuil]
    similaires = similaires.drop(utilisateur)

    produits = produits_non_notes(utilisateur)
    scores = {}

    for produit in produits:
        score = sum(similaires * df.loc[similaires.index, produit])
        scores[produit] = score

    return sorted(scores.items(), key=lambda x: x[1], reverse=True)
```

✓ 0.0s

— Créer une fonction Python recommander(utilisateur)

```
def recommander(utilisateur):
    return recommander_produits(utilisateur)

recommander("Utilisateur_1")
```

✓ 0.0s

```
[('Produit_1', np.float64(0.9380352942419313)),
 ('Produit_3', np.float64(0.9380352942419313)),
 ('Produit_5', np.float64(0.8397822097303648))]
```

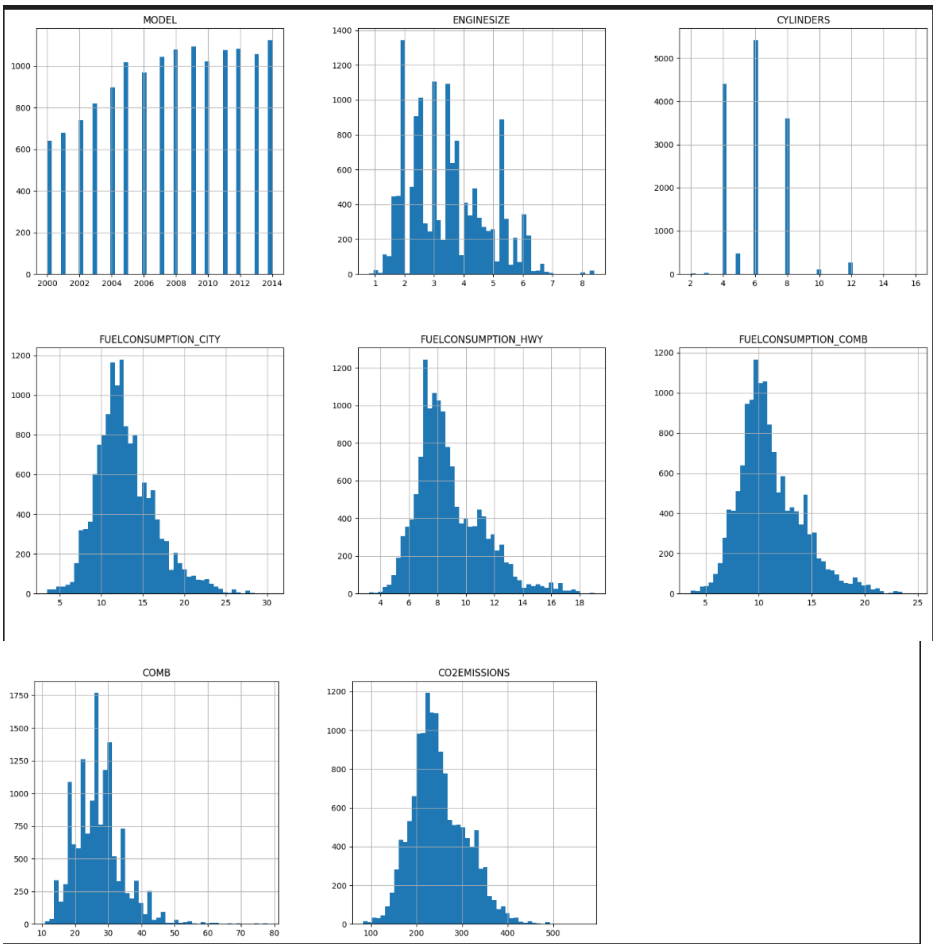
TP 3 : La régression

Présentation des données

```
import matplotlib.pyplot as plt
import pandas as pd
import pylab as pl
import numpy as np
import os
import missingno as msno
from sklearn import metrics
os. chdir ("C:/Users/oure.ma/Documents/TP MACHINE LEARNING/TP5")

df = pd. read_excel ("Fuel_Consumption.xlsx", sheet_name = 0)

print (df. head () )

print (df. describe () )
print (df. shape)
df. hist (bins = 50 , figsize = (20,20) )
plt. show ()
```

Le résultat :

```
   MODEL   MAKE  MODEL.1 VEHICLE CLASS  ENGINESIZE  CYLINDERS TRANSMISSION  \
0   2000  ACURA    1.6EL       COMPACT         1.6          4           A4
1   2000  ACURA    1.6EL       COMPACT         1.6          4           M5
2   2000  ACURA    3.2TL      MID-SIZE         3.2          6           AS5
3   2000  ACURA    3.5RL      MID-SIZE         3.5          6           A4
4   2000  ACURA  INTEGRA    SUBCOMPACT         1.8          4           A4

  FUEL TYPE  FUELCONSUMPTION_CITY  FUELCONSUMPTION_HWY  FUELCONSUMPTION_COMB  \
0         X                   9.2                  6.7                   8.1
1         X                   8.5                  6.5                   7.6
2         Z                  12.2                  7.4                  10.0
3         Z                  13.4                  9.2                  11.5
4         X                  10.0                  7.0                   8.6

   COMB  CO2EMISSIONS
0    35         186.0
1    37         175.0
2    28         230.0
3    25         264.0
4    33         198.0
             MODEL    ENGINESIZE     CYLINDERS  FUELCONSUMPTION_CITY  \
count  14337.000000  14301.000000  14337.000000          14325.000000
mean    2007.629420      3.476414      5.991281             12.929340
std        4.149667      1.309143      1.768949              3.533551
min     2000.000000      0.800000      2.000000              3.500000
...
50%             8.300000             10.500000     27.000000    239.000000
75%            10.200000             12.800000     31.000000    285.000000
max            19.000000             24.800000     78.000000    570.000000
(14337, 13)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



On a installé la bibliothèque missingno :

```
pip install missingno
```

Python

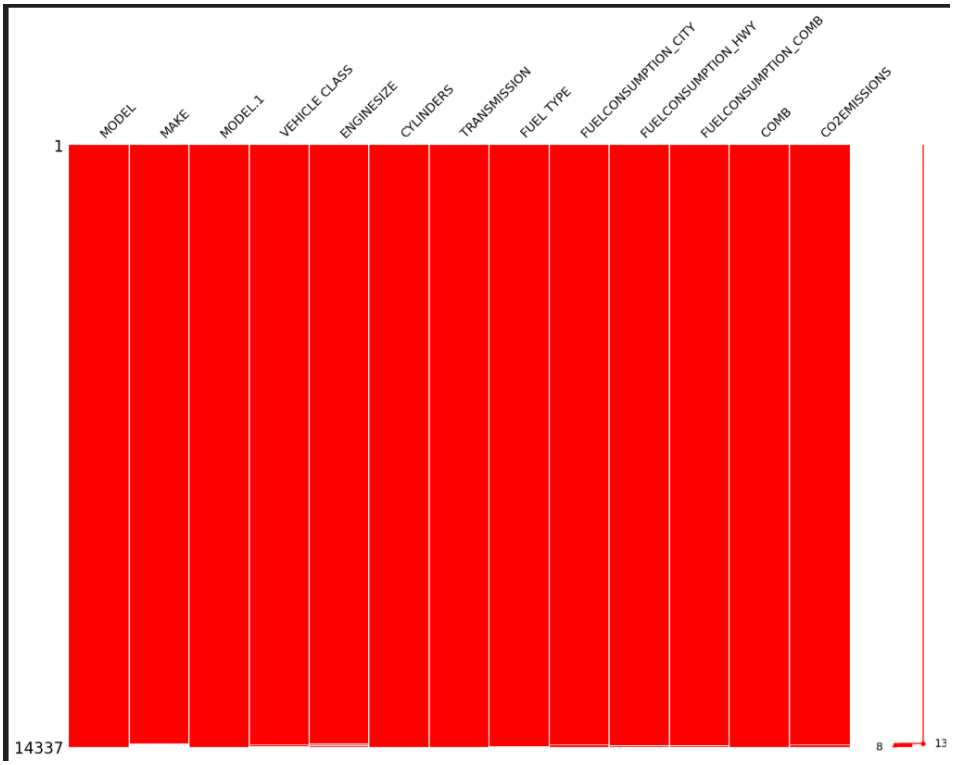
```
Collecting missingno
  Obtaining dependency information for missingno from https://files.pythonhosted.org/packages/87/22/cd5cf999af21c2f97486622c551ac3d07361ced8125121
  Downloading missingno-0.5.2-py3-none-any.whl.metadata (639 bytes)
Requirement already satisfied: numpy in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from missingno) (2.3.3)
Requirement already satisfied: matplotlib in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from missingno) (3.10.7)
Requirement already satisfied: scipy in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from missingno) (1.16.2)
Collecting seaborn (from missingno)
  Obtaining dependency information for seaborn from https://files.pythonhosted.org/packages/83/11/00d3c3dfc25ad54e731d91449895a79e4bf2384dc3ac0188
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from matplotlib->mi
Requirement already satisfied: cycler>=0.10 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from matplotlib->missin
Requirement already satisfied: fonttools>=4.22.0 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from matplotlib->m
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from matplotlib->m
Requirement already satisfied: packaging>=20.0 in c:\users\oure.ma\appdata\roaming\python\python311\site-packages (from matplotlib->missingno) (25
Requirement already satisfied: pillow>=8 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from matplotlib->missingno
Requirement already satisfied: pyparsing>=3 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from matplotlib->missin
Requirement already satisfied: python-dateutil>=2.7 in c:\users\oure.ma\appdata\roaming\python\python311\site-packages (from matplotlib->missingno
Requirement already satisfied: pandas>=1.2 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from seaborn->missingno)
Requirement already satisfied: pytz>=2020.1 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from pandas>=1.2->seabo
Requirement already satisfied: tzdata>=2022.7 in c:\users\oure.ma\appdata\local\programs\python\python311\lib\site-packages (from pandas>=1.2->sea
Requirement already satisfied: six>=1.5 in c:\users\oure.ma\appdata\roaming\python\python311\site-packages (from python-dateutil>=2.7->matplotlib-
Downloading missingno-0.5.2-py3-none-any.whl (8.7 kB)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn, missingno
Successfully installed missingno-0.5.2 seaborn-0.13.2
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip is available: 23.2.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

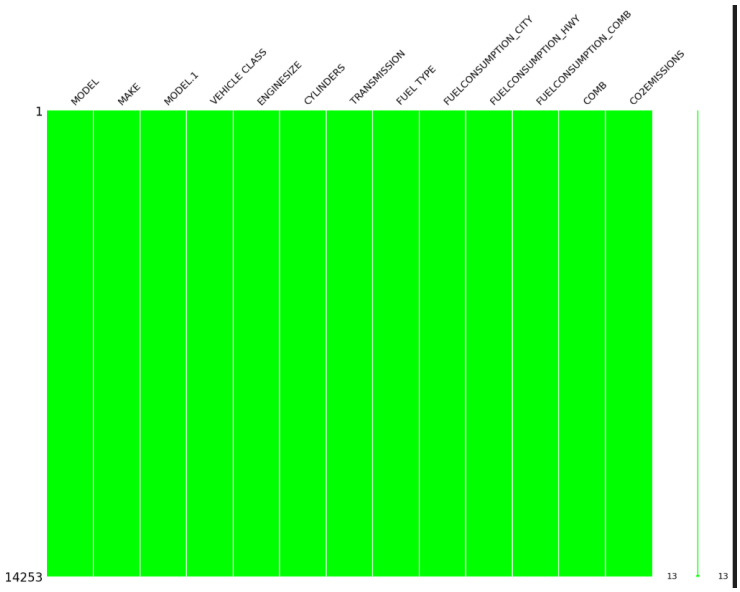
Activer Windows
Accédez aux paramètres pour activer Windows.

Des petites transformations :

```
print(df.isnull ().sum ())
msno.matrix (df=df, figsize= (20,14), color= (1,0,0) )
df. dropna (axis=0, subset= [ 'MAKE'], inplace=True)
print (df.isnull ().sum () )
msno.matrix (df=df, figsize= (20,14), color= (0,1, 0) )
```

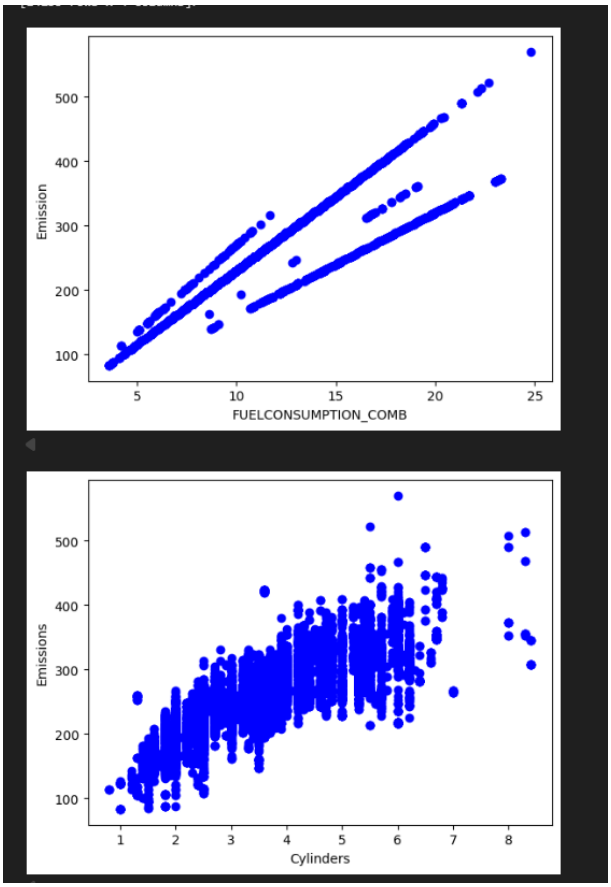
```
MODEL                    0
MAKE                    84
MODEL.1                  0
VEHICLE CLASS           48
ENGINESIZE              36
CYLINDERS                0
TRANSMISSION             0
FUEL TYPE               24
FUELCONSUMPTION_CITY    12
FUELCONSUMPTION_HWY     48
FUELCONSUMPTION_COMB    24
COMB                     0
CO2EMISSIONS            24
dtype: int64
MODEL                   0
MAKE                    0
MODEL.1                 0
VEHICLE CLASS           0
ENGINESIZE              0
CYLINDERS               0
TRANSMISSION            0
FUEL TYPE               0
FUELCONSUMPTION_CITY    0
FUELCONSUMPTION_HWY     0
FUELCONSUMPTION_COMB    0
COMB                    0
CO2EMISSIONS            0
dtype: int64

<Axes: >
```






```
#Créer une nouvelle Dataframe CDF ET VIa à partir, d'une trame de données extatante
cdf=df[['ENGINESIZE', 'CYLINDERS', 'FUELCONSUMPTION_COMB', 'CO2EMISSIONS']]
print (cdf.head () )
#Créer data frame VIZ avec des colonnes sélectionnées A partir du edf existanataframe
viz= cdf[['CYLINDERS', 'ENGINESIZE', 'CO2EMISSIONS', 'FUELCONSUMPTION_COMB' ]]
print (viz. hist)
plt.show ()
plt.scatter (cdf. FUELCONSUMPTION_COMB, cdf. CO2EMISSIONS, color='blue')
plt.xlabel("FUELCONSUMPTION_COMB")
plt.ylabel ("Emission")
plt.show ()
plt.scatter (cdf. ENGINESIZE, cdf. CO2EMISSIONS,color ='blue' )
plt. xlabel ("Cylinders")
plt.ylabel ("Emissions")
plt.show ()
```

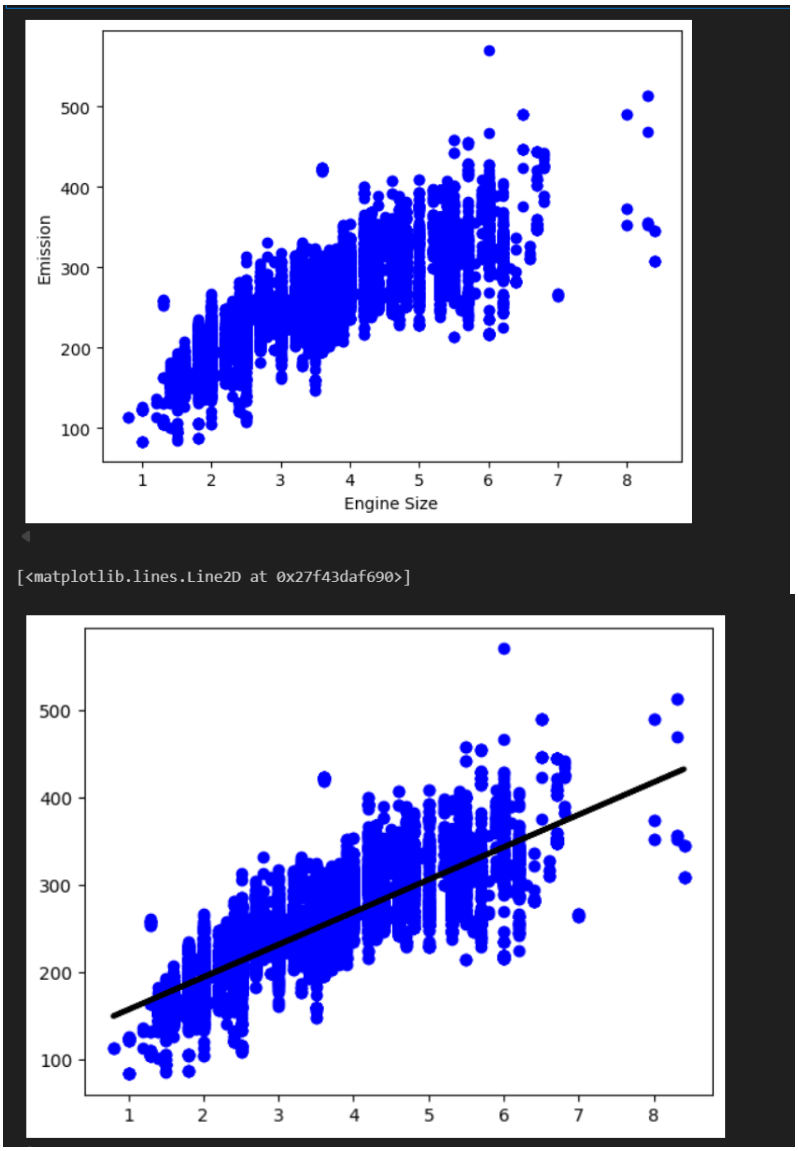
```
   ENGINESIZE  CYLINDERS  FUELCONSUMPTION_COMB  CO2EMISSIONS
0         1.6          4                   8.1         186.0
1         1.6          4                   7.6         175.0
2         3.2          6                  10.0         230.0
3         3.5          6                  11.5         264.0
4         1.8          4                   8.6         198.0
<bound method hist_frame of        CYLINDERS  ENGINESIZE  CO2EMISSIONS  FUELCONSUMPTION_COMB
0              4         1.6         186.0                   8.1
1              4         1.6         175.0                   7.6
2              6         3.2         230.0                  10.0
3              6         3.5         264.0                  11.5
4              4         1.8         198.0                   8.6
...          ...         ...           ...                   ...
14248          6         3.0         237.0                  10.3
14249          6         3.2         230.0                  10.0
14250          6         3.0         237.0                  10.3
14251          6         3.2         225.0                   9.8
14252          6         3.2         258.0                  11.2

[14253 rows x 4 columns]>
```



Création d'un ensemble de données pour les trains et les tests :

```
msk =np.random.rand(len(df)) < 0.8
train = cdf[msk]
test = cdf[~msk]
```

```
#Simple Regression Model
plt. scatter (train. ENGINESIZE, train. CO2EMISSIONS, color = 'blue')
plt. xlabel ("Engine Size")
plt. ylabel ("Emission")
plt. show ()
### Modélisation
#*utilisation du progiciel sklearn pour modéliser les données*
from sklearn import linear_model
regr = linear_model. LinearRegression ()
train_x = np. asanyarray (train[['ENGINESIZE' ]])
train_y = np. asanyarray (train[['CO2EMISSIONS' ]] )
regr. fit (train_x, train_y)
#affichage des resultats
train_Y_= regr.predict (train_x)
plt. scatter (train. ENGINESIZE, train. CO2EMISSIONS, color ='blue')
plt. plot (train_x, train_Y_, color='black', linewidth =3)
```




```
[<matplotlib.lines.Line2D at 0x27f43daf690>]
```

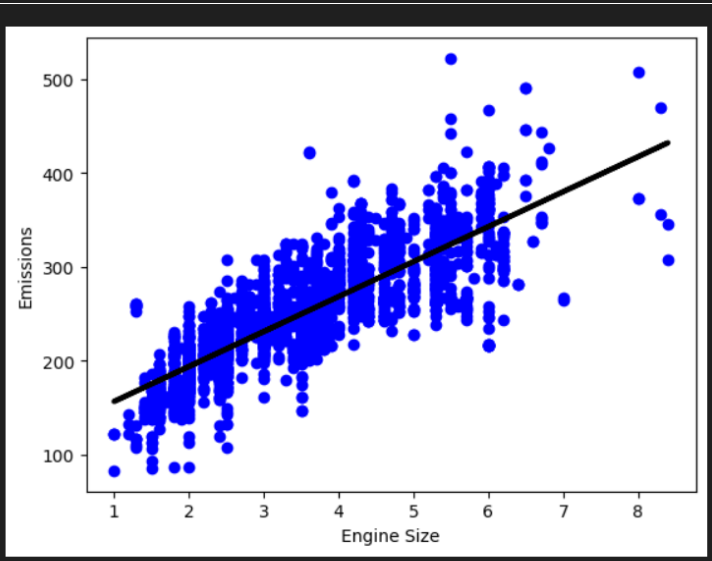
Evaluation Régression Linéaires (1) :

```
#Evaluation
#Evalues le modele avec les données du test*
test_x = np.asanyarray (test [['ENGINESIZE']])
test_y = np.asanyarray (test [['CO2EMISSIONS']])
test_Y_ = regr.predict (test_x)
print
(" Linear Regression Residual Sum of Squares : %.2f:" % np.mean ((test_Y_ - test_y)**2))
#Explained variance score: 1 est une prédiction parfaite
print ("Linear Regession Variance Score : %.2f" %metrics.explained_variance_score (test_y, test_Y_))
#Précision de la régression linéaire avec jeu de
accuracy_lf = metrics.r2_score (test_y,test_Y_)
print (' Linear Regression Accuracy: ' , accuracy_lf)
print('Linear Regression Coefficients: ' ,regr. coef_)
print('Linear Regression Intercept : ', regr. intercept_)
```

```
Linear Regession Variance Score : 0.68
 Linear Regression Accuracy:  0.6780081460745003
Linear Regression Coefficients:  [[37.2361211]]
Linear Regression Intercept :  [119.41286583]
```

```
y_Train=list(train_y)
y_Predict=train_Y_
error=0
for i in range (len(y_Train)):
    error+= (abs (y_Train [i]- y_Predict [i]) /y_Train [i])
train_error_lin=error/len (y_Train) *100
print ("Linear RegressionTrain error = "'()'. format (train_error_lin)+" percent ")
Predict=test_Y_
Y_test=list (test_y)
error=0
for i in range (len (test_y)):
    error+= (abs(y_Predict [i] -Y_test [i])/y_Predict[i])
test_error_lin=error/len (Y_test) *100
print("Linear RegressionTest error = "'{}'.format(test_error_lin) +"percent ")
## Plot Outputs,
plt. scatter (test_x, test_y,color ='blue')
plt. plot (test_x, test_Y_,color ='black', linewidth =3)
plt. xlabel ("Engine Size")
plt. ylabel ("Emissions")
plt. show ()
```

```
Linear RegressionTrain error = () percent
Linear RegressionTest error = [24.61179647]percent
```



Régression Polynomiale :

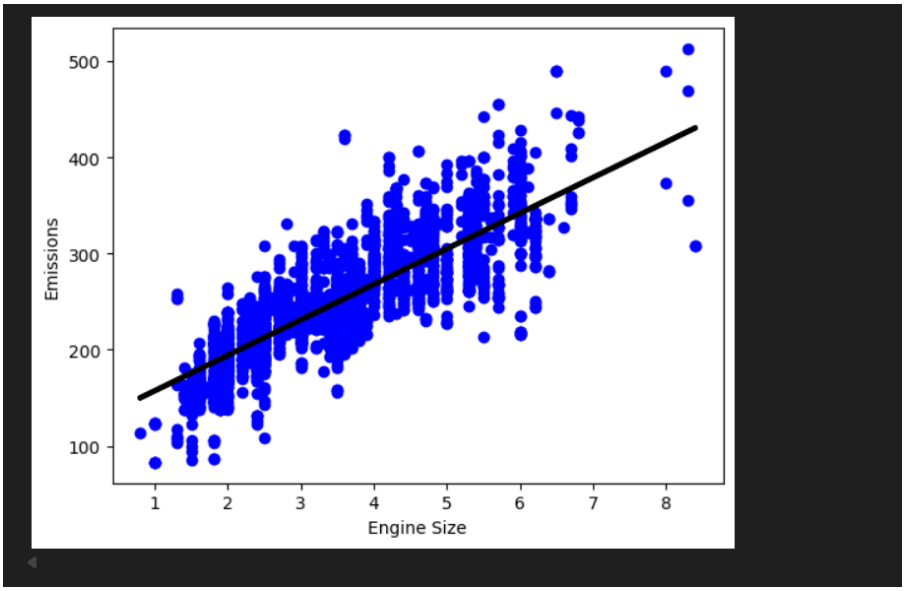
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly_reg = PolynomialFeatures(degree=2)

X_poly = poly_reg.fit_transform(train_x)

pol_reg = LinearRegression()
pol_reg.fit(X_poly, train_y)

train_y_pred = pol_reg.predict(X_poly)
```

```
from sklearn import metrics
import numpy as np

print("Polynomial Regression Residual Sum of Squares : %.2f"
      % np.mean((test_Y_ - test_y) ** 2))

# Explained variance score : 1 is perfect prediction
print("Polynomial Regression Variance Score : %.2f"
      % metrics.explained_variance_score(test_y, test_Y_))

accuracy_lf = metrics.r2_score(test_y, test_Y_)
print('Polynomial Regression Accuracy : ', accuracy_lf)

print('Polynomial Regression Coefficients :', pol_reg.coef_)
print('Polynomial Regression Intercept :', pol_reg.intercept_)
```

```
Polynomial Regression Residual Sum of Squares : 1086.34
Polynomial Regression Variance Score : 0.70
Polynomial Regression Accuracy :  0.6978218504651579
Polynomial Regression Coefficients : [[ 0.         60.91226275 -3.1364434 ]]
Polynomial Regression Intercept : [80.38357662]
```

```
y_Train=list(train_y)

y_predict=train_Y_

error=0
for i in range(len(y_Train)):
    error+=(abs(y_Train[i]-y_predict[i])/y_Train[i])
train_error_pol=error/len(y_Train)*100
print("Polynomial Regression Train error = ""()".format(train_error_pol)+" percent ")
y_Predict=test_Y_
Y_test=list(test_y)

error=0
for i in range(len(test_y)):
    error+=(abs(y_Predict[i]-Y_test[i])/y_Predict[i])
test_error_pol=error/len(Y_test)*100
print("Polynomial Regression Test error = ""()".format(test_error_pol)+" percent in Ridge Regression")

plt.scatter(test_x,test_y, color ='blue')
plt.plot(test_x, test_Y_ , color ='black', linewidth =3)
plt.xlabel("Engine Size")
plt.ylabel("Emissions")
plt.show()
```

```
Polynomial Regression Train error = () percent
Polynomial Regression Test error = () percent in Ridge Regression
```



Régression Ridge :

```
#Ridge Regression
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

model=linear_model.Ridge(alpha=0.01)
model.fit(train_x,train_y)
train_y =model.predict(train_x)
test_y =model.predict(test_x)
```

✓ 0.0s

```
print (" Ridge Regression Residual Sum of Squares : %.2f"
    % np.mean((test_Y_ - test_y)**2))
# Explained variance score : 1 is perfect prediction
print('Ridge Regression Variance Score : %.2f' % metrics.explained_variance_score(test_y,test_Y_))
#Linear Regression Accuracy with test set
accuracy_lf = metrics.r2_score(test_y, test_Y_)
print(' Ridge Regression Accuracy: ', accuracy_lf)

print ('Ridge Regression Coefficients :', model.coef_)
print ('Ridge RegressionIntercept : ', model.intercept_)
```

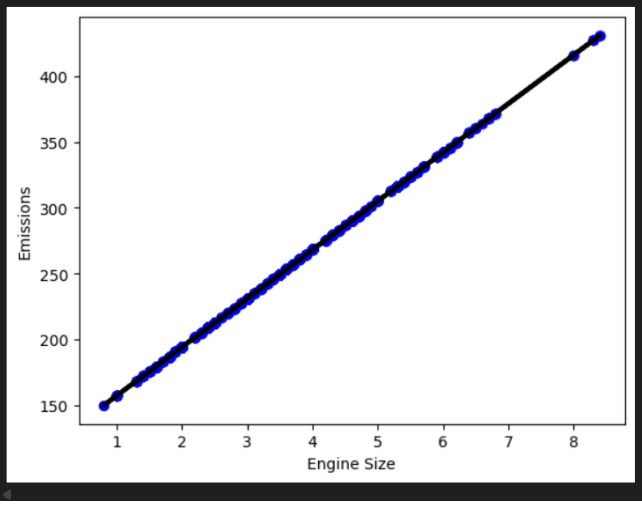
✓ 0.0s

```
 Ridge Regression Residual Sum of Squares : 4673.01
Ridge Regression Variance Score : 1.00
 Ridge Regression Accuracy:  0.9999999999997418
Ridge Regression Coefficients : [36.9349968]
Ridge RegressionIntercept :  [120.45669716]
```

```
y_Train=list(train_y)
y_predict=train_Y_

error=0
for i in range(len(y_Train)):
    error+=(abs(y_Train[i]-y_predict[i])/y_Train[i])
train_error_ridge=error/len(y_Train)*100
print("Ridge RegressionTrain error = "'()'.format(train_error_ridge)+" percent ")
y_Predict=test_Y_
Y_test=list(test_y)

error=0
for i in range(len(test_y)):
    error+=(abs(y_Predict[i]-Y_test[i])/y_Predict[i])
test_error_ridge=error/len(Y_test)*100
print("Ridge Regression Test error = "'()'.format(test_error_ridge)+" percent ")
plt.scatter(test_x,test_y, color ='blue')
plt.plot(test_x, test_y, color ='black', linewidth =3)
plt.xlabel("Engine Size")
plt.ylabel("Emissions")
plt.show()
```

✓ 0.1s

```
Ridge RegressionTrain error = () percent
Ridge Regression Test error = () percent
```



Régression Bayésienne :

Implémenter une « Régression de type Bayésienne»

• X=ENGINESIZE

• Y=CO2EMISSIONS

Affichage des résultats

Evaluer le modèle avec les données du test

```
reg = linear_model.BayesianRidge()
reg.fit(train_x,train_y)
y1_reg=reg.predict(train_x)
y1_reg=list(y1_reg)
y2_reg=reg.predict(test_x)
y2_reg=list(y2_reg)
```

✓ 0.0s

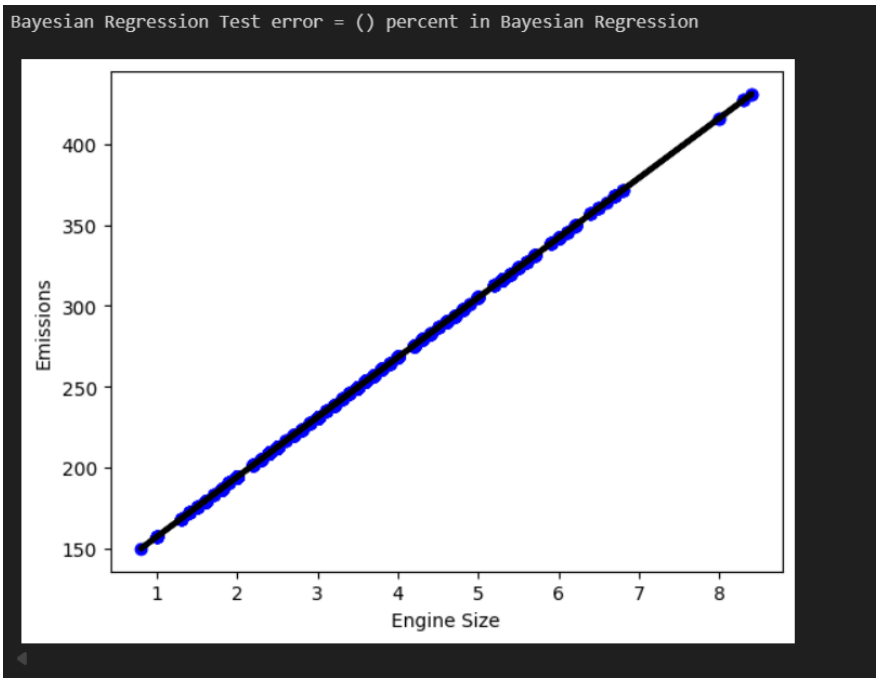
```
print (" Bayesian Regression Residual Sum of Squares : %.2f"
    % np.mean((y2_reg - test_y)**2))
#Explained_variance_score : 1 is perfect prediction
print("Bayesian Regression Variance Score : %.2f" % metrics.explained_variance_score(test_y,y2_reg))
#Linear Regression Accuracy with test set
accuracy_If = metrics.r2_score(test_y, y2_reg)
print(" Bayesian Regression Accuracy: ", accuracy_If)

print ("Bayesian Regression Coefficients :", reg.coef_)
print ("Bayesian RegressionIntercept : ", reg.intercept_)
```

✓ 0.0s

```
Bayesian Regression Residual Sum of Squares : 0.00
Bayesian Regression Variance Score : 1.00
 Bayesian Regression Accuracy:  1.0
Bayesian Regression Coefficients : [36.9349968]
Bayesian RegressionIntercept :  120.45669715543275
```

```
error=0
for i in range(len(test_y)):
    error+= (abs(y2_reg[i]-Y_test[i])/Y_test[i])
test_error_day=(error/len(Y_test))*100
print("Bayesian Regression Test error = "+'()'.format(test_error_day)+" percent"+" in Bayesian Regression")

plt.scatter(test_x,test_y, color ='blue')
plt.plot(test_x, y2_reg, color ='black', linewidth =3)
plt.xlabel("Engine Size")
plt.ylabel("Emissions")
plt.show()
```

✓ 0.1s



TP 4 : Régression Linéaire– Prédiction du Prix d'un Forfait Mobile

Objectif pédagogique :

Ce travail pratique a pour but d'implémenter un modèle de régression linéaire simple afin d'estimer le prix d'un forfait mobile en fonction de l'âge du client. Il permet d'introduire les concepts de l'apprentissage supervisé, d'étudier la relation entre deux variables quantitatives et d'évaluer les performances d'un modèle de régression.

Partie 1 : Importation et préparation des données

1. Créer un DataFrame contenant deux colonnes : age et prix_forfait

2. Charger les données depuis un fichier .csv ou générer les données manuellement

3. Afficher un aperçu des premières lignes du dataset

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import os

os. chdir ("C:/Users/oure.ma/Documents/TP MACHINE LEARNING/tp6")

df = pd.read_csv ("data_forfait_10000.txt")


# Aperçu des premières lignes
print(df.head())
```

```
   âge  prix_forfait
0   56         39.14
1   46         30.95
2   32         21.02
3   60         36.99
4   25         20.22
```

Partie 2 : Régression linéaire

1. Appliquer un modèle LinearRegression pour prédire le prix en fonction de l'âge

2. Découper les données en jeu d'apprentissage (train) et jeu de test

3. Afficher les coefficients appris : — Intercept (b0) — Slope (b1)

```python
# Variables explicatives et cible
X = df[['âge']]
y = df['prix_forfait']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = LinearRegression()
model.fit(X_train, y_train)

print(f"Intercept (b0) : {model.intercept_:.2f}")
print(f"Slope (b1) : {model.coef_[0]:.2f}")
```

```
[9]
...  Intercept (b0) : 7.96
     Slope (b1) : 0.50
```

3. Effectuer des prédictions sur les données test

Partie 3 : Évaluation et visualisation

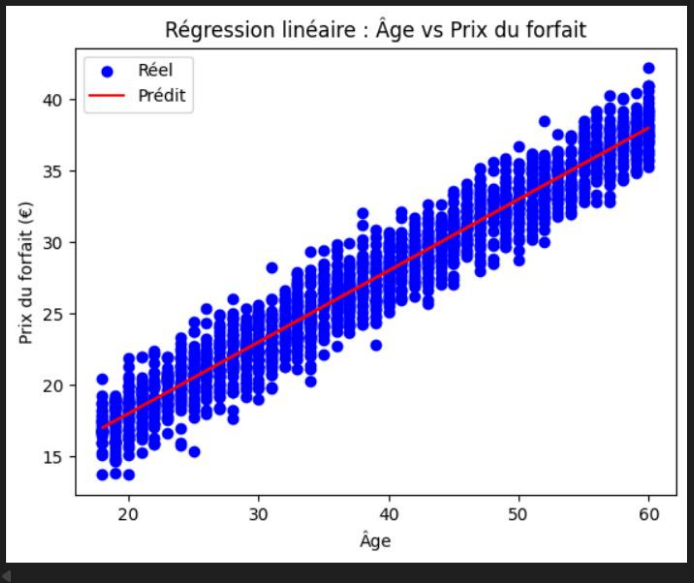
1. Calculer l'erreur quadratique moyenne (MSE et RMSE)

2. Afficher la courbe des valeurs réelles vs. valeurs prédites

3. Tracer la droite de régression sur le nuage de points

```python
# Prédictions
y_pred = model.predict(X_test)

# MSE et RMSE
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"MSE : {mse:.2f}")
print(f"RMSE : {rmse:.2f}")
print(f"R² : {r2:.2f}")

# Courbe des valeurs réelles vs prédites
plt.scatter(X_test, y_test, color='blue', label='Réel')
plt.plot(X_test, y_pred, color='red', label='Prédit')
plt.xlabel('Âge')
plt.ylabel('Prix du forfait (€)')
plt.title('Régression linéaire : Âge vs Prix du forfait')
plt.legend()
plt.show()
```

```
MSE : 2.32
RMSE : 1.52
R² : 0.94
```



Partie 4 : Analyse

- Interprétation de la pente du modèle

La pente du modèle de régression représente l'influence de l'âge sur le prix du forfait. Étant donné que le coefficient associé à l'âge est positif (b1 = 0,50), cela indique que le prix du forfait tend à augmenter lorsque l'âge du client augmente. Plus précisément, une augmentation d'une année de l'âge entraîne en moyenne une hausse de 0,50 € du prix du forfait. On observe donc une relation croissante entre ces deux variables.

- Effet de l'ajout de bruit dans le jeu de données

L'ajout de bruit consiste à introduire des variations aléatoires dans les valeurs des variables, telles que l'âge ou le prix du forfait. Cette perturbation a pour conséquence de dégrader la qualité des données. En pratique, les prédictions du modèle deviennent moins fiables, les

erreurs augmentent, ce qui se traduit par une hausse des indicateurs MSE et RMSE. Par ailleurs, la valeur du coefficient R^2 diminue, indiquant que le modèle explique moins bien la variabilité des données. Ainsi, la présence de bruit réduit la stabilité et la précision du modèle.

- Robustesse du modèle et interprétation des indicateurs

Avec les données actuelles, les résultats montrent de bonnes performances du modèle. La valeur du RMSE égale à 1,52 indique que l'erreur moyenne de prédiction reste relativement faible par rapport aux prix réels des forfaits. De plus, un R^2 de 0,9 signifie que le modèle parvient à expliquer environ 90 % de la variance observée, ce qui traduit une bonne capacité de généralisation. Le MSE, de valeur 2,32, confirme également que les écarts entre les valeurs réelles et prédites sont limités.

Cependant, la robustesse du modèle dépend fortement de la qualité des données. En présence de bruit important ou de valeurs aberrantes, les performances pourraient se dégrader de manière significative.

Conclusion générale

La régression linéaire met en évidence une relation positive entre l'âge du client et le prix du forfait mobile. Les indicateurs de performance obtenus, notamment un RMSE faible et un R^2 élevé, montrent que le modèle est efficace sur le jeu de données étudié. Néanmoins, sa précision reste sensible aux perturbations des données, ce qui souligne l'importance d'un prétraitement adéquat avant l'apprentissage.

TP 5 : Introduction à la régression logistique

Importation des packages :

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt. rc ("font", size=14)
from sklearn. linear_model import LogisticRegression
from imblearn.over_sampling import SMOTE
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
sns.set (style="white")
sns.set (style="whitegrid", color_codes=True)
```

Visualisation des données :

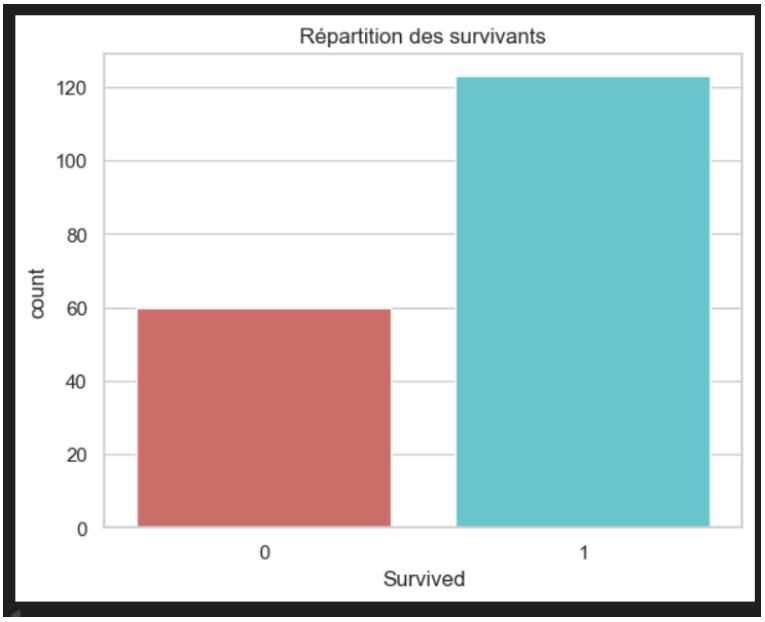
```python
#importation des données:
data = pd.read_csv ('train.csv', header=0)
data = data. dropna ()
print (data. shape)
print (list(data. columns))
print (data.head () )
```

```
(183, 12)
['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked']
    PassengerId  Survived  Pclass  \
1             2         1       1
3             4         1       1
6             7         0       1
10           11         1       3
11           12         1       1

                                                 Name     Sex   Age  SibSp  \
1   Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
3        Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
6                             McCarthy, Mr. Timothy J    male  54.0      0
10                   Sandstrom, Miss. Marguerite Rut  female   4.0      1
11                          Bonnell, Miss. Elizabeth  female  58.0      0

    Parch    Ticket     Fare Cabin Embarked
1       0  PC 17599  71.2833   C85        C
3       0    113803  53.1000  C123        S
6       0     17463  51.8625   E46        S
10      1   PP 9549  16.7000    G6        S
11      0    113783  26.5500  C103        S
```

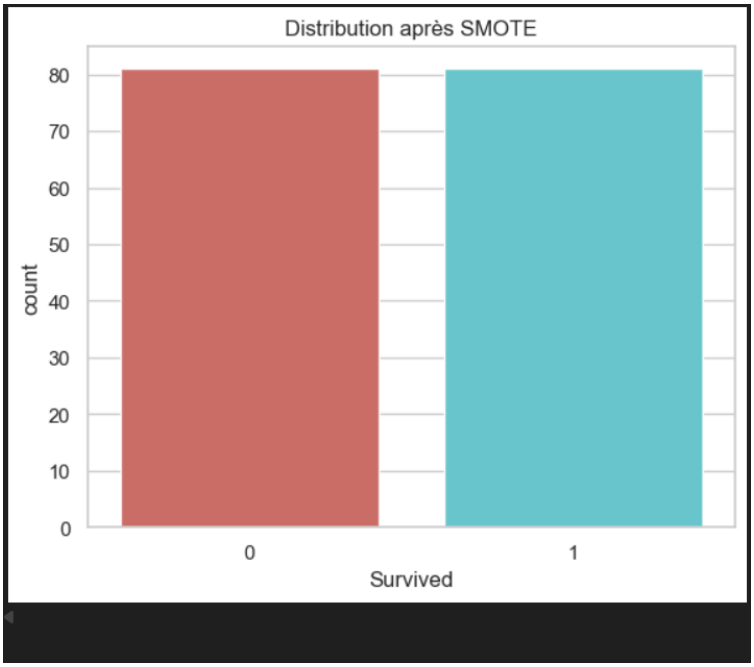
Activer Wind

Transformation :

```python
data['Age'].fillna(data['Age'].median(), inplace=True)

# Remplir les valeurs manquantes de 'Embarked' par le mode
data['Embarked'].fillna(data['Embarked'].mode()[0], inplace=True)

# Supprimer la colonne 'Cabin' (trop de valeurs manquantes)
data.drop(['Cabin', 'Name', 'Ticket', 'PassengerId'], axis=1, inplace=True)

# Encodage des variables catégorielles
cat_vars = ['Sex', 'Embarked', 'Pclass']
data = pd.get_dummies(data, columns=cat_vars, drop_first=True)

# Affichage de la distribution de la cible
sns.countplot(x='Survived', data=data, palette='hls')
plt.title("Répartition des survivants")
plt.show()
```



```python
#Équilibrage des classes avec SMOTE
X = data.drop('Survived', axis=1)
y = data['Survived']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

sm = SMOTE(random_state=0)
X_res, y_res = sm.fit_resample(X_train, y_train)

sns.countplot(x=y_res, palette='hls')
plt.title("Distribution après SMOTE")
plt.show()
```




```python
# Corrélation avec la variable cible
cor = pd.concat([X_res, y_res], axis=1).corr()
cor_target = abs(cor['Survived'])
relevant_features = cor_target[cor_target > 0.1]
print("Variables pertinentes:\n", relevant_features)
```

```
Variables pertinentes:
 Age           0.294847
SibSp          0.173137
Fare           0.194233
Sex_male       0.607527
Embarked_S     0.176826
Pclass_3       0.131812
Survived       1.000000
Name: Survived, dtype: float64
```

```python
#Réduction de dimension avec RFE
logreg = LogisticRegression(max_iter=1000)
rfe = RFE(logreg, n_features_to_select=5)
rfe.fit(X_res, y_res)

selected_cols = X.columns[rfe.support_]
print("Colonnes sélectionnées:", selected_cols)
```

```
Colonnes sélectionnées: Index(['SibSp', 'Sex_male', 'Embarked_Q', 'Embarked_S', 'Pclass_3'], dtype='object')
```

```python
#Régression logistique et évaluation
X_train_sel = X_res[selected_cols]
X_test_sel = X_test[selected_cols]

logreg.fit(X_train_sel, y_res)
y_pred = logreg.predict(X_test_sel)

print("Matrice de confusion:\n", confusion_matrix(y_test, y_pred))
print("\nRapport de classification:\n", classification_report(y_test, y_pred))
```

```
Matrice de confusion:
 [[11  2]
 [12 30]]

Rapport de classification:
              precision    recall  f1-score   support

           0       0.48      0.85      0.61        13
           1       0.94      0.71      0.81        42

    accuracy                           0.75        55
   macro avg       0.71      0.78      0.71        55
weighted avg       0.83      0.75      0.76        55
```

TP : La sélection de variables

Importation de exploration des données :

```python
import pandas as pd
dataframe = pd.read_excel("FINAL.xlsx",sheet_name=0)
print(dataframe.head())
print(dataframe.shape)
X = dataframe.drop("Results", axis=1)

Y = dataframe["Results"]
```

```
      Ks  Ks.Axis     Kf  Kf.Axis   AvgK   CYL    AA  Ecc.9.0mm.   ACCP  \
0  44.53       21  39.22      111  41.87  5.32  86.7        0.91  41.67
1  43.84       39  42.46      129  43.15  1.38  88.2        0.65  43.14
2  44.81       66  44.41      156  44.61  0.40  83.0        0.48  44.70
3  44.00       51  42.31      141  43.15  1.69  97.3        0.60  43.16
4  45.42       26  45.20      116  45.31  0.22  93.3        0.69  45.29

     Ks.1  ...  coma.5  coma.axis.5  SA.C40..5  S35.coma.like..5  \
0  49.61  ...   3.131           97     -0.722             3.350
1  48.84  ...   0.575           97      0.085             0.921
2  49.92  ...   0.177            9      0.268             0.263
3  49.01  ...   0.492          275     -0.281             3.396
4  50.60  ...   0.571           85      0.109             0.691

   S46.sph..like..5  HOAs.S3456..5  AA.5  ESI.Anterior.  ESI.Posterior.  \
0             1.053          3.512    99             45              27
1             0.290          0.966   100              0               0
2             0.640          0.692   100              0               0
3             1.419          3.680   100              0              29
4             0.181          0.714   100              0               7

   Results
0        1
1        2
2        2
...
4        2

[5 rows x 422 columns]
(3162, 422)
```

Caractéristiques constantes :

```
from sklearn.feature_selection import VarianceThreshold
def Constant_Features(numerical_x):
    vs_constant = VarianceThreshold(threshold=0)
    vs_constant.fit(numerical_x)
    constant_columns = [ column for column in numerical_x.columns  if column not in numerical_x.columns[vs_constant.get_support]]
    len(constant_columns)
    numerical_x=numerical_x.drop(labels=constant_columns, axis =1)
    return numerical_x
```

Fonctionnalités quasi-constantes :

```
import numpy as np
import pandas as pd

def quasi_constant_features(numerical_x, threshold=0.98):

    quasi_constant_features = []

    for feature in numerical_x.columns:

        predominant = (numerical_x[feature].value_counts() / np.float(len(numerical_x))).sort_values(ascending=False).values[0]


        if predominant >= threshold:
            quasi_constant_features.append(feature)

    numerical_x = numerical_x.drop(labels=quasi_constant_features, axis=1)
    return numerical_x
```

Fonctionnalités dupliquées :

```
def duplicate_features(numerical_x):

    train_features_T = numerical_x.T

    print(f"Number of duplicated features: {train_features_T.duplicated().sum()}")

    duplicated_columns = train_features_T[train_features_T.duplicated()].index.values

    print(f"Duplicated features to remove: {duplicated_columns}")

    numerical_x = numerical_x.drop(labels=duplicated_columns, axis=1)

    return numerical_x
```

Méthodes de filtrage par corrélation

```
import pandas as pd
import numpy as np
from sklearn.feature_selection import mutual_info_classif, chi2, f_classif, SelectKBest, SelectPercentile
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.metrics import roc_auc_score, mean_squared_error

def correlation_features(numerical_x, threshold):

    correlated_features = set()
    correlation_matrix = numerical_x.corr()

    for i in range(len(correlation_matrix.columns)):
        for j in range(i):
            if abs(correlation_matrix.iloc[i, j]) > threshold:
                colname = correlation_matrix.columns[i]
                correlated_features.add(colname)

    numerical_x = numerical_x.drop(labels=correlated_features, axis=1)
    return numerical_x
```

Information mutuelle :

```python
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import matplotlib.pyplot as plt

def Mutual_Information_Features(numerical_x, numerical_y, select_k):

    # Sélection de K meilleures features basées sur la Mutual Information
    selection = SelectKBest(mutual_info_classif, k=select_k).fit(numerical_x, numerical_y)

    # Affichage des scores pour chaque feature
    for i in range(len(selection.scores_)):
        print(f"Feature {numerical_x.columns[i]} : Score = {selection.scores_[i]}")

    # Visualisation des scores
    plt.bar(range(len(selection.scores_)), selection.scores_)
    plt.title("Mutual Information Scores")
    plt.xlabel("Features")
    plt.ylabel("Score")
    plt.show()

    # Garder seulement les colonnes sélectionnées
    numerical_x = numerical_x.loc[:, selection.get_support()]

    return numerical_x
```

Test univarié ANOVA :

```python
from sklearn.feature_selection import SelectKBest, f_classif
import matplotlib.pyplot as plt

def classif_Features(numerical_x, numerical_y, select_k):

    selection = SelectKBest(f_classif, k=select_k).fit(numerical_x, numerical_y)
    for i in range(len(selection.scores_)):
        print(f"Feature {numerical_x.columns[i]} : Score = {selection.scores_[i]}")

    plt.bar(range(len(selection.scores_)), selection.scores_)
    plt.title("ANOVA F-score for Each Feature")
    plt.xlabel("Features")
    plt.ylabel("Score")
    plt.show()

    numerical_x = numerical_x.loc[:, selection.get_support()]

    return numerical_x
```

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
import pandas as pd
import matplotlib.pyplot as plt

def ROC_AUC_Features(X_train, Y_train, X_test, Y_test, threshold):

    roc_auc = []

    for feature in X_train.columns:

        clf = DecisionTreeClassifier()
        clf.fit(X_train[[feature]], Y_train)

        y_scored = clf.predict_proba(X_test[[feature]])

        roc_auc.append(roc_auc_score(Y_test, y_scored[:, 1]))

    roc_values = pd.Series(roc_auc, index=X_train.columns)
    roc_values = roc_values.sort_values(ascending=False)

    roc_values.plot.bar(figsize=(10, 5))
    plt.title("ROC-AUC Score for Each Feature")
    plt.ylabel("ROC-AUC")
    plt.show()

    sel = roc_values[roc_values > threshold]

    numerical_x = X_train[sel.index]

    return numerical_x
```

Les méthodes Wrapper :

Sélection de fonction avant Forward Feature Selection :

```
#Forward Feature Selection
def SFS_Features(x_train, y_train, select_k,cv):
    # create the SequentialFeaturesSelector object, and configure the parameters.
    sfs = SequentialFeaturesSelector(RandomForestClassifier(),
    k_features=10,
    forward=True,
    floating=False,
    scoring='accuracy',
    cv=cv)
    # fit the object to the training data.
    sfs = sfs.fit(x_train, y_train)
    # print the selected_features
    selected_features = x_train.columns[list(sfs.k_feature_idx_)]
    # print the final prediction score.
    print(sfs.k_score_)
    return selected_features
```

Élimination des fonctionnalités vers l'arrière Backward Feature Elimination

```
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
def SBS_Features(x_train,y_train,select_k,cv_k):
    sbs = SequentialFeatureSelector(RandomForestClassifier(),
        k_features=select_k,
        forward=False,
        floating=False,
        scoring="accuracy"  ,
        cv = cv_k)
    sbs = sbs.fit(x_train,y_train)
    selected_features = x_train.columns[list(sbs.k_feature_idx_)]
    print(sbs.k_score_)
    return selected_features
```

Sélection complète des fonctionnalités Exhaustive Feature Selection :

```
from mlxtend.feature_selection import ExhaustiveFeatureSelector


# import the algorithm you want to evaluate on your features.
from sklearn.ensemble import RandomForestClassifier

#Exhaustive Feature Selection
def efs_Features(x_train, y_train, select_k_min, select_k_max, cv_k):
    # create the ExhaustiveFeatureSelector object.
    efs = ExhaustiveFeatureSelector(RandomForestClassifier(),
    min_features=select_k_min,
    max_features=select_k_max,
    scoring='roc_auc',
    cv=cv_k)
    # fit the object to the training data.
    efs = efs.fit(x_train, y_train)
    # print the selected features.
    selected_features = x_train.Columns[list(efs.k_feature_idx)]
    print(efs.k_score)
    return selected_features
✓ 0.0s
```

```python
#Embedded Methods
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel

def Embedded(x_train, y_train,n_estimators_k):
    # create the random forest with your hyperparameters.
    model = SelectFromModel(RandomForestClassifier(n_estimators = n_estimators_k),threshold=0.1)
    # fit the model to start training.
    model.fit(x_train, y_train)
    # get the importance of the resulting features.
    selected_feat=x_train.Columns([model.get_support()])
    for feature in zip(x_train.Columns, model.get_support(indices=True)):
     print(feature)
    print(len(selected_feat))
    # create a data frame for visualization.
    plt.show()
    return selected_feat
```

TP 6 : Les règles d'association

Installation :

```python
import mlxtend
print(mlxtend.__version__)
```

```
0.23.4
```

Importation des données :

```python
#Chargement de dossier
import os
os.chdir("C:/Users/pc/Documents/Module Hrimech/TP3_7.Règles d'association_Python.0")

#Importation des donnees
import pandas
D = pandas.read_table("market_basket.txt", delimiter = "\t", header=0)

#10 premieres lignes
print(D.head())

#Verification des dimensions
print(D.shape)
```

```
   ID        Product
0   1        Peaches
1   2  Vegetable_Oil
2   2    Frozen_Corn
3   3          Plums
4   4    Pancake_Mix
(12935, 2)
```

Transformation en tableau binaire :

```
#Tableau croise 0/1
TC = pandas.crosstab(D.ID, D.Product)
print(TC.iloc[:20, :3])

#dimensions
print(TC.shape)
```

```
Product  100_Watt_Lightbulb  2pct_Milk  40_Watt_Lightbulb
ID
1                          0          0                  0
2                          0          0                  0
3                          0          0                  0
4                          0          0                  0
5                          0          0                  0
6                          0          0                  0
7                          0          1                  0
8                          0          0                  0
9                          0          0                  0
10                         0          0                  0
11                         0          0                  0
12                         0          0                  0
13                         0          0                  0
14                         0          0                  0
15                         0          0                  0
16                         0          0                  0
17                         0          0                  0
18                         0          0                  0
19                         0          0                  0
20                         0          0                  0
(1360, 303)
```

Liste des noms de produits :

```
print(TC.columns)
```

```
Index(['100_Watt_Lightbulb', '2pct_Milk', '40_Watt_Lightbulb',
       '60_Watt_Lightbulb', '75_Watt_Lightbulb', '98pct_Fat_Free_Hamburger',
       'AA_Cell_Batteries', 'Apple_Cinnamon_Waffles', 'Apple_Drink',
       'Apple_Fruit_Roll',
       ...
       'White_Bread', 'White_Wine', 'White_Zinfandel_Wine', 'Whole_Corn',
       'Whole_Green_Beans', 'Whole_Milk', 'Window_Cleaner', 'Wood_Polish',
       'flav_Fruit_Bars', 'flav_Ice'],
      dtype='object', name='Product', length=303)
```

Extraction des itemsets fréquents :

```
#Importation de la fct apriopri
from mlxtend.frequent_patterns import apriori

#itemsets frequents
freq_itemsets = apriori(TC, min_support = 0.025, max_len = 4, use_colnames = True)

#type -> pandas DataFrame
type(freq_itemsets)
```

```
c:\Users\pc\Documents\Module Hrimech\TP3_7.Règles d'association_Python.0\env\Lib\site-packages\m
  warnings.warn(

pandas.core.frame.DataFrame
```

Affichons les 15 premiers :

```python
#Liste des colonnes
print(freq_itemsets.columns)

#Nombre d'itemsets
print(freq_itemsets.shape)

#Affichage des 15 premiers itemsets
print(freq_itemsets.head(15))
```

```
Index(['support', 'itemsets'], dtype='object')
(603, 2)
     support                  itemsets
0   0.030147       (100_Watt_Lightbulb)
1   0.109559               (2pct_Milk)
2   0.037500        (60_Watt_Lightbulb)
3   0.031618        (75_Watt_Lightbulb)
4   0.093382  (98pct_Fat_Free_Hamburger)
5   0.031618        (AA_Cell_Batteries)
6   0.025735   (Apple_Cinnamon_Waffles)
7   0.026471             (Apple_Drink)
8   0.031618        (Apple_Fruit_Roll)
9   0.032353              (Apple_Jam)
10  0.033088            (Apple_Jelly)
11  0.032353            (Apple_Sauce)
12  0.053676                 (Apples)
13  0.066912                (Aspirin)
14  0.027941            (Avocado_Dip)
```

Filtrage des itemsets :

```python
#Fct de test d'inclusion
def is_inclus(x, items):
    return items.issubset(x)

#Recherche des index des items correspondant a une condition
import numpy
id = numpy.where(freq_itemsets.itemsets.apply(is_inclus, items= {'Aspirin'}))
print(id)
```

```
(array([ 13, 208, 249, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281,
       282, 283, 284, 285, 286, 509, 510, 511, 552, 553, 554, 555, 556]),)
```

```python
#Passer par une fct lambda si on est presse
numpy.where(freq_itemsets.itemsets.apply(lambda x, ensemble:ensemble.issubset(x), ensemble={'Aspirin'}))
```

```
(array([ 13, 208, 249, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281,
       282, 283, 284, 285, 286, 509, 510, 511, 552, 553, 554, 555, 556]),)
```

```python
#itemsets avec Aspirin
print(freq_itemsets[freq_itemsets['itemsets'].eq({'Aspirin'})])

#itemsets contenant Aspirin et Eggs
print(freq_itemsets[freq_itemsets['itemsets'].ge({'Aspirin', 'Eggs'})])

#itemsets contenant Aspirin et Eggs
print(freq_itemsets[freq_itemsets['itemsets'].ge({'Eggs' , 'Aspirin'})])
```

```
     support   itemsets
13  0.066912  (Aspirin)
      support                      itemsets
274  0.038235               (Aspirin, Eggs)
509  0.025735    (Aspirin, 2pct_Milk, Eggs)
552  0.025000  (Aspirin, Eggs, Potato_Chips)
553  0.029412   (Aspirin, Eggs, White_Bread)
      support                      itemsets
274  0.038235               (Aspirin, Eggs)
509  0.025735    (Aspirin, 2pct_Milk, Eggs)
552  0.025000  (Aspirin, Eggs, Potato_Chips)
553  0.029412   (Aspirin, Eggs, White_Bread)
```

Extraction (déduction) des règles d'association :

```python
#Fcy de calcul des regles
from mlxtend.frequent_patterns import association_rules

#Generation des regles a partir des itemsets frequents
regles = association_rules(freq_itemsets, metric="confidence", min_threshold=0.75)

#Type de l'objet renvoye
print(type(regles))

#Dimension
print(regles.shape)

#Liste des colonnes
print(regles.columns)

#5 premieres regles
print(regles.iloc[:5, :])
```

```
<class 'pandas.core.frame.DataFrame'>
(50, 14)
Index(['antecedents', 'consequents', 'antecedent support',
       'consequent support', 'support', 'confidence', 'lift',
       'representativity', 'leverage', 'conviction', 'zhangs_metric',
       'jaccard', 'certainty', 'kulczynski'],
      dtype='object')
                 antecedents    consequents  antecedent support  \
0       (Aspirin, 2pct_Milk)  (White_Bread)            0.034559
1       (2pct_Milk, Bananas)  (White_Bread)            0.031618
2     (White_Bread, Bananas)    (2pct_Milk)            0.032353
3        (Wheat_Bread, Cola)    (2pct_Milk)            0.032353
4  (2pct_Milk, Popcorn_Salt)         (Eggs)            0.033088

   consequent support   support  confidence      lift  representativity  \
0            0.119118  0.027206    0.787234  6.608878               1.0
1            0.119118  0.025735    0.813953  6.833190               1.0
2            0.109559  0.025735    0.795455  7.260525               1.0
3            0.109559  0.025735    0.795455  7.260525               1.0
4            0.122794  0.027206    0.822222  6.695941               1.0

   leverage  conviction  zhangs_metric   jaccard  certainty  kulczynski
0  0.023089    4.140147       0.879068  0.215116   0.758463    0.507815
1  0.021969    4.734743       0.881527  0.205882   0.788795    0.515001
2  0.022191    4.353268       0.891099  0.221519   0.770288    0.515177
3  0.022191    4.353268       0.891099  0.221519   0.770288    0.515177
4  0.023143    4.934283       0.879766  0.211429   0.797336    0.521890
```

Ce code sert à afficher uniquement les informations essentielles des règles d'association et à améliorer la lisibilité des résultats, puis à visualiser les premières règles obtenues.

```
#Regles en restrieignant l'affichage a qqs colonnes
myRegles = regles.loc[:, ['antecedents', 'consequents', 'lift']]
print(myRegles.shape)

#Pour afficher toutes les colonnes
pandas.set_option('display.max_columns', 5)
pandas.set_option('display.precision', 3)

#Affichage des 5 premieres regles
print(myRegles[:5])
```

```
(50, 3)
                 antecedents    consequents   lift
0       (Aspirin, 2pct_Milk)  (White_Bread)  6.609
1       (2pct_Milk, Bananas)  (White_Bread)  6.833
2     (White_Bread, Bananas)    (2pct_Milk)  7.261
3        (Wheat_Bread, Cola)    (2pct_Milk)  7.261
4  (2pct_Milk, Popcorn_Salt)         (Eggs)  6.696
```

Ce code permet de filtrer les règles d'association selon le critère du lift (≥ 7), d'identifier les règles les plus fortes en les triant par lift décroissant, puis d'analyser des règles spécifiques en fonction d'un conséquent donné (*2pct_Milk*) ou de la présence d'un item particulier (*Aspirin*) dans l'antécédent.

```
#affichage des regles avec un LIFT superieur ou egale à 7
print(myRegles[myRegles['lift'].ge(7.0)])
#trier les regles dans l'ordre du lift decroissants -10 meilleurs regles
print(myRegles.sort_values(by='lift',ascending=False)[:10])
#filtrer les regles menant à 2pct=_milk
print(myRegles[myRegles['consequents'].eq({'2pct_Milk'})])
#filtrer les regles contenant Aspirin dans l'antecedent
print(myRegles[myRegles['antecedents'].ge({'Aspirin'})])
```

```
                                antecedents                consequents   lift
2                    (White_Bread, Bananas)                 (2pct_Milk)  7.261
3                       (Wheat_Bread, Cola)                 (2pct_Milk)  7.261
8                     (Wheat_Bread, Onions)                 (2pct_Milk)  7.574
10                  (Potatoes, Wheat_Bread)                 (2pct_Milk)  7.053
13                (Toothpaste, Wheat_Bread)                 (2pct_Milk)  7.380
16            (White_Bread, Hamburger_Buns)  (98pct_Fat_Free_Hamburger)  8.202
17  (Wheat_Bread, 98pct_Fat_Free_Hamburger)                (White_Bread)  7.556
29              (Sweet_Relish, Hot_Dog_Buns)                   (Hot_Dogs)  9.031
35                 (Potatoes, Toilet_Paper)                (White_Bread)  7.319
37               (Toothpaste, Toilet_Paper)                (White_Bread)  7.346
41        (Eggs, White_Bread, Potato_Chips)                 (2pct_Milk)  7.143
44          (Toothpaste, Eggs, White_Bread)                 (2pct_Milk)  7.261
46    (Toothpaste, Potato_Chips, 2pct_Milk)                (White_Bread)  7.319
47  (Toothpaste, Potato_Chips, White_Bread)                 (2pct_Milk)  7.569
48     (Toothpaste, 2pct_Milk, White_Bread)               (Potato_Chips)  7.726
49   (Potato_Chips, 2pct_Milk, White_Bread)                 (Toothpaste)  9.514
                                antecedents                consequents   lift
49   (Potato_Chips, 2pct_Milk, White_Bread)                 (Toothpaste)  9.514
29              (Sweet_Relish, Hot_Dog_Buns)                   (Hot_Dogs)  9.031
16            (White_Bread, Hamburger_Buns)  (98pct_Fat_Free_Hamburger)  8.202
48     (Toothpaste, 2pct_Milk, White_Bread)               (Potato_Chips)  7.726
8                     (Wheat_Bread, Onions)                 (2pct_Milk)  7.574
47  (Toothpaste, Potato_Chips, White_Bread)                 (2pct_Milk)  7.569
17  (Wheat_Bread, 98pct_Fat_Free_Hamburger)                (White_Bread)  7.556
...
18          (Aspirin, Eggs)  (White_Bread)  6.458
19  (Aspirin, Potato_Chips)  (White_Bread)  6.339
20      (Potatoes, Aspirin)  (White_Bread)  6.962
21    (Toothpaste, Aspirin)  (White_Bread)  6.996
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

- **Conclusion**

Ces travaux pratiques ont permis de découvrir et de mettre en œuvre plusieurs techniques clés de la data science et du machine learning. Nous avons exploré les règles d'association, les arbres de décision, les systèmes de recommandation et la régression linéaire et logistique, en mettant l'accent sur l'interprétation des modèles et l'évaluation de leurs performances. La sélection de variables a montré l'importance de choisir des attributs pertinents pour améliorer la précision et la simplicité des modèles. Dans l'ensemble, ces TP ont renforcé notre compréhension théorique et pratique des méthodes d'analyse de données avec Python.