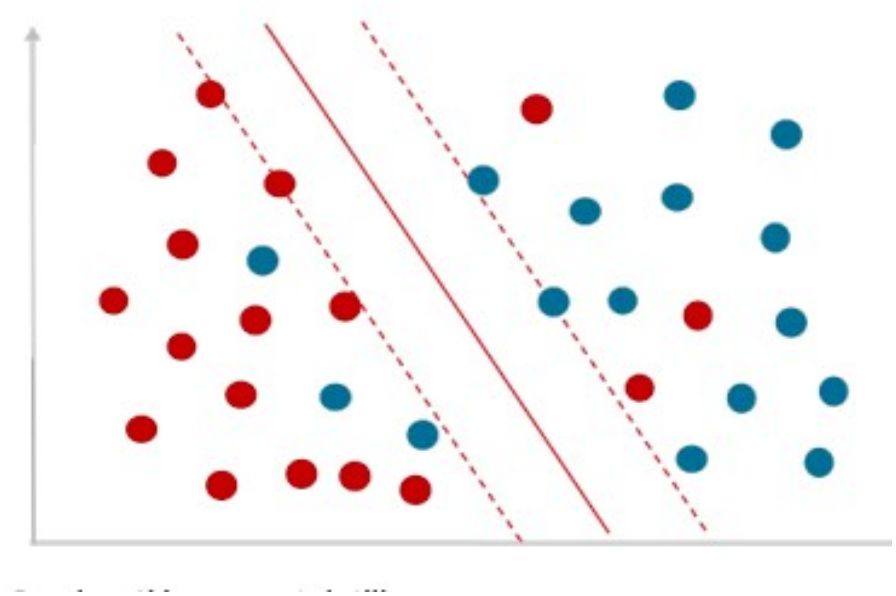
Université Abdelmalek Essaad Faculté ses Sciences et techniques de Tanger
Département Génie Informatique

Atelier 2
«Classification»



Réalisée par :
KAISSI Houda

Objective :
l'objective principal de cet atelier est de pratiquer les concepts de la classification, en traitant les
données d'une Data Sets, ainsi d'évaluer les algorithmes pour construire le modèle adéquat à notre
problématique.

Outils:
Python, Pandas, Sklearn, matplotlib

Partie 1 (Data Visualisation et Feature Selection et Normalisation):

1. En utilisant pandas essayer d'explorer les données du Data set.

```
import pandas as pd

#Le nom de dataset
link = "pima-indians-diabetes.csv"
# Les colones
Features = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

dataset = pd.read_csv(link, names=Features)

# Affichage de Dataset
print(" ----- Head -----")
print(dataset.head())
```

R ésultat :

```
 ----- Head -----
   preg  plas  pres  skin  test  mass   pedi  age  class
0     6   148    72    35     0  33.6  0.627   50      1
1     1    85    66    29     0  26.6  0.351   31      0
2     8   183    64     0     0  23.3  0.672   32      1
3     1    89    66    23    94  28.1  0.167   21      0
4     0   137    40    35   168  43.1  2.288   33      1
```

2. Afficher le résumer statistique du Data Sets avec une interprétation des résultats obtenues.

```python
#2
import pandas as pd

#Dataset
link = "pima-indians-diabetes.csv"
#Features
Features = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
#Lire dataset
dataset = pd.read_csv(link, names=Features)

 #Afficher Resumé statique
print(" La Résumé statistique   ")
print(dataset.describe())
```

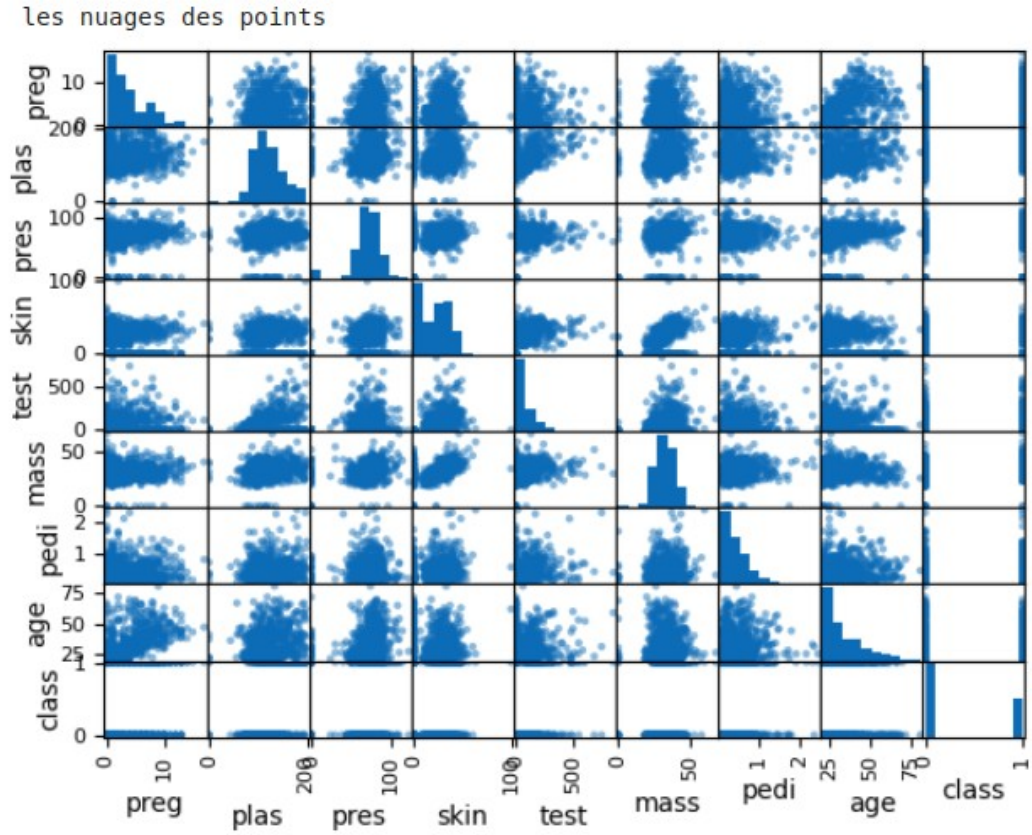
Resultat :

```
              preg        plas        pres        skin        test        mass  \
count  768.000000  768.000000  768.000000  768.000000  768.000000  768.000000
mean     3.845052  120.894531   69.105469   20.536458   79.799479   31.992578
std      3.369578   31.972618   19.355807   15.952218  115.244002    7.884160
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      1.000000   99.000000   62.000000    0.000000    0.000000   27.300000
50%      3.000000  117.000000   72.000000   23.000000   30.500000   32.000000
75%      6.000000  140.250000   80.000000   32.000000  127.250000   36.600000
max     17.000000  199.000000  122.000000   99.000000  846.000000   67.100000

              pedi         age       class
count  768.000000  768.000000  768.000000
mean     0.471876   33.240885    0.348958
std      0.331329   11.760232    0.476951
min      0.078000   21.000000    0.000000
25%      0.243750   24.000000    0.000000
50%      0.372500   29.000000    0.000000
75%      0.626250   41.000000    1.000000
max      2.420000   81.000000    1.000000
```

3. Afficher les nuages des points du data set selon les propriétés « Features » en utilisant matplotlib et pandas « scatter_matrix ».

```python
#3
import pandas as pd
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix

#Dataset
link = "pima-indians-diabetes.csv"
#Features
Features = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# Lire dataset
dataset = pd.read_csv(link, names=Features)
#créer une matrice de dispersion
print(" les nuages des points   ")
scatter_matrix(dataset)
plt.show()
```

Resultat



nuages de points

4. Appliquer les 4 méthodes de Features selection « Univariate Selection, PCA, Recursive Feature
Elimination et Feature Importance ».
modifier ds et capture decran

US :

(US) identifie les trois meilleures caractéristiques en utilisant le test du chi carré pour mesurer leur pertinence par rapport à la variable cible.

```python
#4
#US
import pandas as pd
import numpy as np
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

# --- Chemin d'accès au Dataset ---
url = "pima-indians-diabetes.csv"
# --- En-tête des colonnes ---
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# --- Lire le fichier ---
dataset = pd.read_csv(url, names=names)

array = dataset.values
X = array[:, 0:8]
Y = array[:, 8]

# Extraction des Features
us = SelectKBest(score_func=chi2, k=3) # On va choisir 3 des meilleurs caractéristiques
fit = us.fit(X, Y)

# --- summarize scores ---
np.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)

# --- summarize selected features ---
print(features[0:5, :])
```

Resultat :

```
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
[[148.   0.  50.]
 [ 85.   0.  31.]
 [183.   0.  32.]
 [ 89.  94.  21.]
 [137. 168.  33.]]
```

PCA :

(PCA) réduit la dimensionnalité des données en identifiant les trois composants principaux qui capturent la variance maximale, facilitant ainsi la visualisation des tendances dominantes.

```python
#PCA
import pandas as pd
from sklearn.decomposition import PCA

# --- Chemin d'accès au Dataset ---
url = "pima-indians-diabetes.csv"
# --- En-tête des colonnes ---
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# --- Lire le fichier ---
dataset = pd.read_csv(url, names=names)

array = dataset.values
X = array[:, 0:8]
Y = array[:, 8]

# --- Extraction des Features ---
pca = PCA(n_components=3) # On va choisir 3 composants principaux
fit = pca.fit(X)

# --- summarize components ---
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

```
Explained Variance: [0.889 0.062 0.026]
[[-2.022e-03  9.781e-02  1.609e-02  6.076e-02  9.931e-01  1.401e-02
   5.372e-04 -3.565e-03]
 [-2.265e-02 -9.722e-01 -1.419e-01  5.786e-02  9.463e-02 -4.697e-02
  -8.168e-04 -1.402e-01]
 [-2.246e-02  1.434e-01 -9.225e-01 -3.070e-01  2.098e-02 -1.324e-01
  -6.400e-04 -1.255e-01]]
```

RFE :

RFE) avec une régression logistique identifie 4 caractéristiques optimales en éliminant de manière récursive les moins importantes, indiquées par les valeurs de rang.

```
#RFE
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

# dataset
link = "pima-indians-diabetes.csv"
# Features
Features = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# Lire le fichier
dataset = pd.read_csv(link, names=Features)

array = dataset.values
X = array[:, 0:8]
Y = array[:, 8]

# Extraction des Features -
#Le modèle de régression logistique est convergé dans le nombre d'itérations par défaut.
model = LogisticRegression(max_iter=1000)
# Specifier nombre  de features pour selecter
rfe = RFE(model, n_features_to_select=4)

fit = rfe.fit(X, Y)
print("Nombre Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Rank: %s" % fit.ranking_)
```

```
Nombre Features: 4
Selected Features: [ True  True False False False  True  True False]
Feature Rank: [1 1 3 5 4 1 1 2]
```

FI :

(Feature Importance) avec un classificateur Extra Trees identifie l'importance relative des caractéristiques du dataset en fournissant les poids attribués à chaque attribut.

```
#FI
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier

# --- Chemin d'accès au Dataset ---
url = "pima-indians-diabetes.csv"
# --- En-tête des colonnes ---
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# --- Lire le fichier ---
dataset = pd.read_csv(url, names=names)

array = dataset.values
X = array[:, 0:8]
Y = array[:, 8]

# --- Extraction des Features ---
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

```
[0.11  0.238 0.098 0.08  0.074 0.142 0.119 0.14 ]
```

5. Normaliser les données des attributs qui nécessitent une normalisation.

```python
#5
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer

# --- Chemin d'accès au Dataset ---
url = "pima-indians-diabetes.csv"
# --- En-tête des colonnes ---
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
# --- Lire le fichier ---
dataset = pd.read_csv(url, names=names)
array = dataset.values
X = array[:, 0:8]
Y = array[:, 8]

# --- Scaling : MinMaxScaler ---
print(" ----- Scaling ----- ")
scaler = MinMaxScaler()
print(scaler.fit(dataset))
MinMaxScaler(copy=True, feature_range=(0, 1))
print(scaler.data_max_)
minMax = scaler.transform(dataset)
print(minMax)

# --- Scaling statistique : RobustScaler ---
print(" ----- Scaling statistique ----- ")
scaler = RobustScaler()
transformer = RobustScaler().fit(dataset)
print(transformer)
RobustScaler(copy=True, quantile_range=(25.0, 75.0), with_centering=True, with_scaling=True)
print(transformer.transform(dataset))

# --- Standariser : StandardScaler ---
print(" ----- Standariser ----- ")
scaler = StandardScaler()
print(scaler.fit(dataset))
print(scaler.mean_)
print(scaler.transform(dataset))

# --- Normaliser ---
print(" ----- Normaliser ----- ")
transformer = Normalizer().fit(dataset)
print(transformer)
Normalizer(copy=True, norm='l2')
print(transformer.transform(dataset))
```

Resultat :

```
 ----- Scaling -----
MinMaxScaler()
[ 17.   199.   122.    99.   846.    67.1    2.42  81.     1.  ]
[[0.353 0.744 0.59  ... 0.234 0.483 1.   ]
 [0.059 0.427 0.541 ... 0.117 0.167 0.   ]
 [0.471 0.92  0.525 ... 0.254 0.183 1.   ]
 ...
 [0.294 0.608 0.59  ... 0.071 0.15  0.   ]
 [0.059 0.633 0.492 ... 0.116 0.433 1.   ]
 [0.059 0.467 0.574 ... 0.101 0.033 0.   ]]
 ----- Scaling statistique -----
RobustScaler()
[[ 0.6    0.752  0.     ...  0.665  1.235  1.   ]
 [-0.4   -0.776 -0.333 ... -0.056  0.118  0.   ]
 [ 1.     1.6   -0.444 ...  0.783  0.176  1.   ]
 ...
 [ 0.4    0.097  0.     ... -0.333  0.059  0.   ]
 [-0.4    0.218 -0.667 ... -0.061  1.059  1.   ]
 [-0.4   -0.582 -0.111 ... -0.15  -0.353  0.   ]]
 ----- Standariser -----
StandardScaler()
[  3.845 120.895  69.105  20.536  79.799  31.993   0.472  33.241   0.349]
[[ 0.64   0.848  0.15  ...  0.468  1.426  1.366]
 [-0.845 -1.123 -0.161 ... -0.365 -0.191 -0.732]
 [ 1.234  1.944 -0.264 ...  0.604 -0.106  1.366]
 ...
 [ 0.343  0.003  0.15  ... -0.685 -0.276 -0.732]
 [-0.845  0.16  -0.471 ... -0.371  1.171  1.366]
 [-0.845 -0.873  0.046 ... -0.474 -0.871 -0.732]]
 ----- Normaliser -----
Normalizer()
[[0.034 0.828 0.403 ... 0.004 0.28  0.006]
 [0.008 0.716 0.556 ... 0.003 0.261 0.   ]
 [0.04  0.924 0.323 ... 0.003 0.162 0.005]
 ...
 [0.027 0.651 0.388 ... 0.001 0.161 0.   ]
 [0.007 0.838 0.399 ... 0.002 0.313 0.007]
 [0.008 0.736 0.554 ... 0.002 0.182 0.   ]]
```

Partie 2 (Classification choix de algorithme adéquat):

1. En utilisant l'API sklearn entraîner les modèles en utilisant ces algorithmes « KNN, Decision Tree, ANN,
Naive Bayes, SVM selon les kernels suivants : Linear, polynomial et guassain».

```
 from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import pickle

# Initialiser les modèles
models = {
    "KNN Linear": KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', n_jobs=
    "Decision Tree": DecisionTreeClassifier(),
    "ANN": MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000),
    "Naive Bayes": GaussianNB(),
    "SVM Linear": SVC(kernel='linear', probability=True),
    "SVM Polynomial": SVC(kernel='poly', degree=3, probability=True),
    "SVM Gaussian": SVC(kernel='rbf', probability=True),
}
```

Les etapes :

Le code effectue les étapes suivantes :

Importation des bibliothèques : Importe les bibliothèques pandas et scikit-learn.

Chargement du dataset : Charge le dataset "pima-indians-diabetes.csv" avec des colonnes spécifiées.

Séparation des features et de la cible : Crée les ensembles de features (X) et de la cible (y) à partir du dataset.

Division du dataset : Utilise la fonction train_test_split pour diviser le dataset en ensembles d'entraînement (X_train, y_train) et de test (X_test, y_test).

2. Sauvegarder les 5 modèles

```
#2
# Entraîner et sauvegarder les modèles
for name, model in models.items():
    model.fit(X_train, y_train)
    filename = f'model_{name.replace(" ", "_")}.sav'
    pickle.dump(model, open(filename, 'wb'))
```

3. Évaluer les modèles en utilisant ces métriques:
Classification Accuracy.
Logarithmic Loss.
Area Under ROC Curve.
Confusion Matrix.

Classification Report.

es modèles sont chargés, leurs performances sont évaluées sur un ensemble de test, et les résultats sont affichés pour chaque modèle. Cette approche permet une comparaison quantitative des performances des modèles

```
from sklearn.metrics import accuracy_score, log_loss, roc_auc_score, confusion_matrix, classification_report

# Charger les modèles
loaded_models = {name: pickle.load(open(f'model_{name.replace(" ", "_")}.sav', 'rb')) for name in models.keys()}

# Évaluer les modèles
evaluation_results = {}

for name, model in loaded_models.items():
    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]

    accuracy = accuracy_score(y_test, y_pred)
    log_loss_value = log_loss(y_test, y_prob)
    auc_value = roc_auc_score(y_test, y_prob)
    confusion = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred, target_names=["Non-Diabetic", "Diabetic"])

    evaluation_results[name] = {
        "Accuracy": accuracy,
        "Log Loss": log_loss_value,
        "AUC": auc_value,
        "Confusion Matrix": confusion,
        "Classification Report": report,
    }

# Afficher les résultats d'évaluation
for name, metrics in evaluation_results.items():
    print(f"--- {name} ---")
    print(f"Accuracy: {metrics['Accuracy']:.4f}")
    print(f"Log Loss: {metrics['Log Loss']:.4f}")
    print(f"AUC: {metrics['AUC']:.4f}")
    print("Confusion Matrix:\n", metrics['Confusion Matrix'])
    print("Classification Report:\n", metrics['Classification Report'])
    print("=" * 50)
```

Interpréter le résultat de l’évaluation.

-Les modèles d'ensemble tels que l'ANN et Naive Bayes semblent mieux performer que les modèles individuels.
-La perte logarithmique est relativement basse pour l'ANN, indiquant une meilleure calibration des probabilités prédites.
-La performance des SVM est assez bonne, mais le choix du noyau influence les résultats.
-L'évaluation complète permet de comparer et de choisir les modèles en fonction des métriques appropriées

Resultat :

```
--- KNN Linear ---
Accuracy: 0.6883
Log Loss: 2.4817
AUC: 0.7162
Confusion Matrix:
 [[114  37]
 [ 35  45]]
Classification Report:
              precision    recall  f1-score   support

Non-Diabetic       0.77      0.75      0.76       151
    Diabetic       0.55      0.56      0.56        80

    accuracy                           0.69       231
   macro avg       0.66      0.66      0.66       231
weighted avg       0.69      0.69      0.69       231

================================================
```

4. Comparer la performance des 8 algorithmes en utilisant la technique Spot-checking.

```python
#4
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

# Ajouter d'autres modèles pour le spot-checking
models.update({
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier(),
})

# Spot-checking
spot_checking_results = {}

for name, model in models.items():
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
    spot_checking_results[name] = scores

# Afficher les résultats du spot-checking
for name, scores in spot_checking_results.items():
    print(f"{name}: Mean Accuracy = {np.mean(scores):.4f}, Std Dev = {np.std(scores):.4f}")
```

Resultat :

```
KNN Linear: Mean Accuracy = 0.7281, Std Dev = 0.0406
Decision Tree: Mean Accuracy = 0.7095, Std Dev = 0.0414
ANN: Mean Accuracy = 0.6815, Std Dev = 0.0535
Naive Bayes: Mean Accuracy = 0.7542, Std Dev = 0.0299
SVM Linear: Mean Accuracy = 0.7802, Std Dev = 0.0339
SVM Polynomial: Mean Accuracy = 0.7560, Std Dev = 0.0196
SVM Gaussian: Mean Accuracy = 0.7579, Std Dev = 0.0178
Random Forest: Mean Accuracy = 0.7672, Std Dev = 0.0295
Gradient Boosting: Mean Accuracy = 0.7708, Std Dev = 0.0446
```

5. Charger les 5 modèles puis Prédire les données du data set de test, en utilisant les 8 modèles.

```python
#5
# Charger les 5 modèles
loaded_models = {name: pickle.load(open(f'{name.replace(" ", "_")}.sav', 'rb'))
                 for name in models.keys()}

# Prédire les données du data set de test
predictions = {}

for name, model in loaded_models.items():
    Y_pred = model.predict(X_test)
    predictions[name] = Y_pred

# Afficher les prédictions
for name, Y_pred in predictions.items():
    print(f"--- {name} ---")
    print("Predictions:\n", Y_pred)
    print("=" * 50)
```

6. Appliquer cette fois les trois techniques d'ensemble learning « bagging , stacking et boosting »

.Ces techniques d'ensemble visent à améliorer la stabilité et la précision des modèles en combinant plusieurs modèles de base. Chacune a ses propres stratégies pour tirer parti de la diversité des modèles de base, réduire la variance, et améliorer les performances globales du modèle d'apprentissage automatique.

```python
#6
from sklearn.ensemble import BaggingClassifier, StackingClassifier, AdaBoostClassifier

# Bagging
bagging_model = BaggingClassifier(base_estimator=DecisionTreeClassifier(), n_estimators=10, random_state=42)
bagging_model.fit(X_train, Y_train)

# Stacking
stacking_model = StackingClassifier(estimators=list(models.items()), final_estimator=LogisticRegression())
stacking_model.fit(X_train, Y_train)

# Boosting
# Boosting
boosting_model = AdaBoostClassifier(estimator=DecisionTreeClassifier(), n_estimators=50, random_state=42)

boosting_model.fit(X_train, Y_train)
```

7. Comparer les résultats obtenues des trois techniques avec les résultats de 8 algorithmes.

```python
#7
# Évaluer les modèles d'ensemble
bagging_accuracy = bagging_model.score(X_test, Y_test)
stacking_accuracy = stacking_model.score(X_test, Y_test)
boosting_accuracy = boosting_model.score(X_test, Y_test)

# Afficher les résultats
print(f"Bagging Accuracy: {bagging_accuracy:.4f}")
print(f"Stacking Accuracy: {stacking_accuracy:.4f}")
print(f"Boosting Accuracy: {boosting_accuracy:.4f}")
```

Resultat :

```
Bagging Accuracy: 0.7273
Stacking Accuracy: 0.7532
Boosting Accuracy: 0.7186
```

le modèle Stacking a la meilleure précision parmi les trois techniques d'ensemble learning évaluées