

Akaichi Mohamed Aziz - 3 LNSI 3


Projet Machine Learning: *Heart Disease Anomaly Detection*

✓ 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 sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_score
```

✓ 2- Monter Google Drive


```
from google.colab import drive
drive.mount('/content/drive')
```

 Mounted at /content/drive

✓ 3- Chargement et Aperçu des Données

```
data_path = '/content/drive/MyDrive/Colab Notebooks/heart.csv'
data = pd.read_csv(data_path)
```

```
print("Aperçu des données :")
print(data.head())
print(data.info())
```

 Aperçu des données :

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	\
0	52	1	0	125	212	0	1	168	0	1.0	2	
1	53	1	0	140	203	1	0	155	1	3.1	0	
2	70	1	0	145	174	0	1	125	1	2.6	0	
3	61	1	0	148	203	0	1	161	0	0.0	2	
4	62	0	0	138	294	1	1	106	0	1.9	1	

	ca	thal	target
0	2	3	0
1	0	3	0
2	0	3	0
3	1	3	0
4	3	2	0

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1025 entries, 0 to 1024

Data columns (total 14 columns):

#	Column	Non-Null Count	Dtype
0	age	1025 non-null	int64
1	sex	1025 non-null	int64
2	cp	1025 non-null	int64
3	trestbps	1025 non-null	int64
4	chol	1025 non-null	int64
5	fbs	1025 non-null	int64
6	restecg	1025 non-null	int64
7	thalach	1025 non-null	int64
8	exang	1025 non-null	int64
9	oldpeak	1025 non-null	float64
10	slope	1025 non-null	int64
11	ca	1025 non-null	int64
12	thal	1025 non-null	int64
13	target	1025 non-null	int64

dtypes: float64(1), int64(13)

memory usage: 112.2 KB

None

✓ 4- Prétraitement des Données

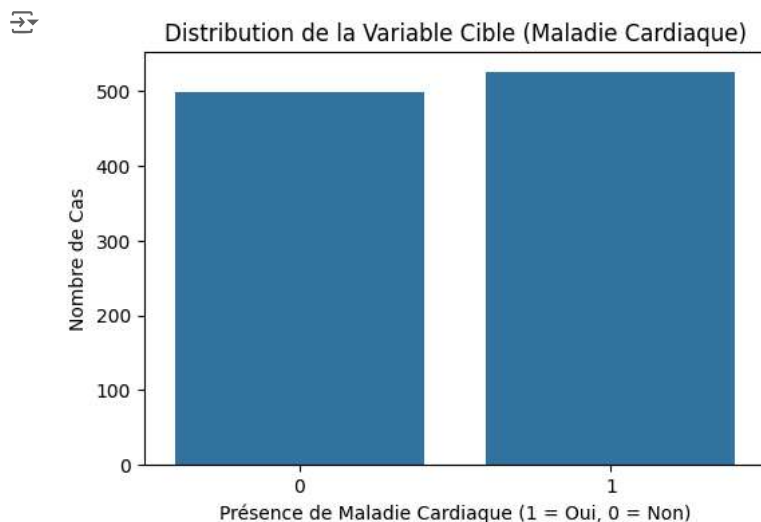
```
# Remplacer les valeurs manquantes
data.fillna(0, inplace=True)

# Séparer les caractéristiques et la cible(target)
features = data.drop(columns=['target'])
target = data['target']

# Normaliser les caractéristiques
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)
scaled_data = pd.DataFrame(scaled_features, columns=features.columns)
scaled_data['target'] = target
```

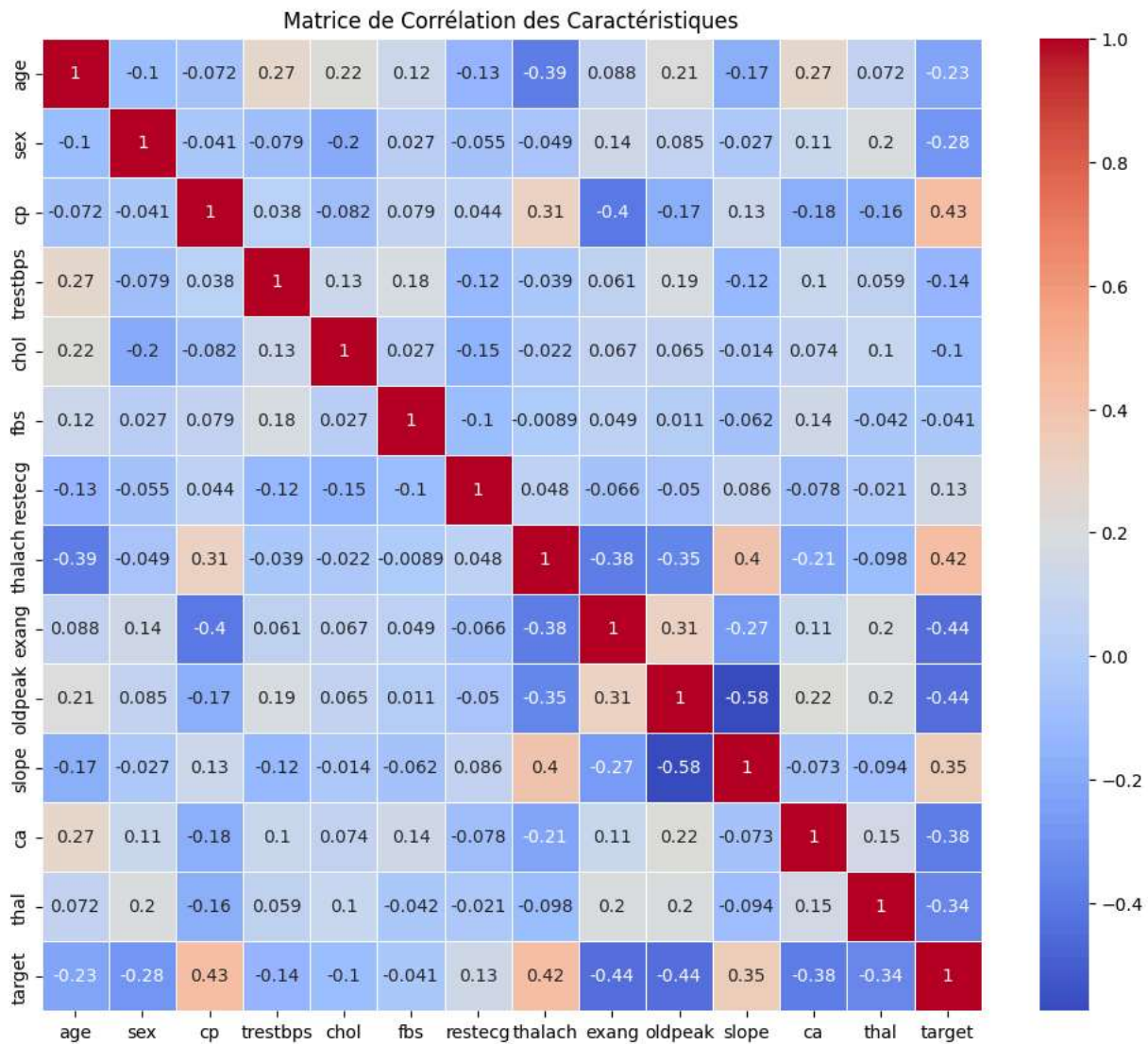
✓ 5- Visualisation des Données

```
# Visualisation de la distribution de la cible
plt.figure(figsize=(6, 4))
sns.countplot(x='target', data=data)
plt.title("Distribution de la Variable Cible (Maladie Cardiaque)")
plt.xlabel("Présence de Maladie Cardiaque (1 = Oui, 0 = Non)")
plt.ylabel("Nombre de Cas")
plt.show()
```



```
# Visualisation de la matrice de corrélation
plt.figure(figsize=(12, 10))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', linewidths=0.5)
plt.title("Matrice de Corrélation des Caractéristiques")
plt.show()
```

{}



6- Division des Données en Ensemble d'Entraînement et de Test

```
# Diviser les données en ensemble d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(
    scaled_data.drop(columns=['target']),
    scaled_data['target'],
    test_size=0.35, # 35% pour l'ensemble de test
    random_state=42
)
```

7- Entraînement et Évaluation des Modèles

```
# Définir les modèles
models = {
    'Régression Logistique': LogisticRegression(),
    'Random Forest': RandomForestClassifier(),
    'KNN': KNeighborsClassifier(),
    'Machine à Vecteurs de Support': SVC(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'Naive Bayes': GaussianNB(),
    'XGBoost': XGBClassifier()
}

# Initialiser le dictionnaire de performances
model_performance = {}

# Entraîner et évaluer les modèles
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
# Calculer les métriques de performance
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

model_performance[model_name] = {
    'Exactitude': accuracy,
    'Précision': precision,
    'Rappel': recall,
    'Score F1': f1
}

# Afficher le rapport de classification
print(f"\nRapport de Classification pour {model_name}:\n", classification_report(y_test, y_pred))
```



```
Rapport de Classification pour Régression Logistique:
      precision    recall  f1-score   support

     0       0.83       0.77       0.80        180
     1       0.78       0.84       0.81        179

 accuracy          0.80          0.80          0.80        359
 macro avg          0.80          0.80          0.80        359
weighted avg          0.80          0.80          0.80        359
```

```
Rapport de Classification pour Forêt Aléatoire:
      precision    recall  f1-score   support

     0       0.98       1.00       0.99        180
     1       1.00       0.98       0.99        179

 accuracy          0.99          0.99          0.99        359
 macro avg          0.99          0.99          0.99        359
weighted avg          0.99          0.99          0.99        359
```

```
Rapport de Classification pour K-Plus Proches Voisins:
      precision    recall  f1-score   support

     0       0.87       0.83       0.85        180
     1       0.83       0.87       0.85        179

 accuracy          0.85          0.85          0.85        359
 macro avg          0.85          0.85          0.85        359
weighted avg          0.85          0.85          0.85        359
```

```
Rapport de Classification pour Machine à Vecteurs de Support:
      precision    recall  f1-score   support

     0       0.93       0.88       0.90        180
     1       0.88       0.93       0.91        179

 accuracy          0.91          0.91          0.91        359
 macro avg          0.91          0.91          0.91        359
weighted avg          0.91          0.91          0.91        359
```

```
Rapport de Classification pour Gradient Boosting:
      precision    recall  f1-score   support

     0       0.96       0.97       0.96        180
     1       0.97       0.96       0.96        179

 accuracy          0.96          0.96          0.96        359
 macro avg          0.96          0.96          0.96        359
weighted avg          0.96          0.96          0.96        359
```

```
Rapport de Classification pour Naive Bayes:
      precision    recall  f1-score   support
```

✓ 8- Comparaison des Performances des Modèles

```
# Convertir les données de performance en DataFrame
performance_df = pd.DataFrame(model_performance).T
print("\nComparaison des Performances des Modèles :")
print(performance_df)
```



```
Comparaison des Performances des Modèles :
      Exactitude  Précision  Rappel  Score F1
```

Régression Logistique	0.802228	0.781250	0.837989	0.808625
Forêt Aléatoire	0.991643	1.000000	0.983240	0.991549
K-Plus Proches Voisins	0.849582	0.834225	0.871508	0.852459
Machine à Vecteurs de Support	0.905292	0.883598	0.932961	0.907609
Gradient Boosting	0.963788	0.966292	0.960894	0.963585
Naive Bayes	0.805014	0.773869	0.860335	0.814815
XGBoost	0.991643	1.000000	0.983240	0.991549

```
# Tracer les performances des modèles
performance_df.plot(kind='bar', figsize=(12, 6))
plt.title("Comparaison des Performances des Modèles")
plt.ylabel("Score")
plt.xlabel("Modèles")
plt.legend(loc="lower right")
plt.show()
```

