

Importation des bibliothèques

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Chargement du jeu de données

```
df = pd.read_csv("C:/Users/aldio/OneDrive/Bureau/VISUALISATION DES
DONNE/CardioMind [?]/CardioMind/data/raw/Medicaldataset.csv")
df.head()
```

	Age	Gender	Heart rate	Systolic blood pressure	Diastolic blood pressure
0	64	1	66	160	83
1	21	1	94	98	46
2	55	1	64	160	77
3	64	1	70	120	55
4	55	1	64	112	65

	Blood sugar	CK-MB	Troponin	Result
0	160.0	1.80	0.012	negative
1	296.0	6.75	1.060	positive
2	270.0	1.99	0.003	negative
3	270.0	13.87	0.122	positive
4	300.0	1.08	0.003	negative

Vérification des valeurs manquantes

```
df.isnull().sum()
```

Age	0
Gender	0
Heart rate	0
Systolic blood pressure	0
Diastolic blood pressure	0
Blood sugar	0
CK-MB	0
Troponin	0
Result	0
dtype: int64	

Encodage des variables catégorielles

```
df['Gender'] = df['Gender'].map({1: 'Male', 0: 'Female'})
```

Vérification des doublons

```
df.duplicated().sum()
```

```
0
```

Normalisation des variables médicales

```
# **7 Normalisation des variables médicales**
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df[['Heart rate', 'Systolic blood pressure', 'Diastolic blood
pressure', 'Blood sugar', 'CK-MB', 'Troponin']] =
scaler.fit_transform(
    df[['Heart rate', 'Systolic blood pressure', 'Diastolic blood
pressure', 'Blood sugar', 'CK-MB', 'Troponin']]
)
```

Suppression des outliers

```
Q1 = df['Heart rate'].quantile(0.25)
Q3 = df['Heart rate'].quantile(0.75)
IQR = Q3 - Q1
df = df[(df['Heart rate'] >= (Q1 - 1.5 * IQR)) & (df['Heart rate'] <=
(Q3 + 1.5 * IQR))]
```

Encodage des variables catégorielles

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Gender'] = le.fit_transform(df['Gender'])
df['Result'] = le.fit_transform(df['Result'])

df.dtypes
```

Age	int64
Gender	int64
Heart rate	float64
Systolic blood pressure	float64
Diastolic blood pressure	float64
Blood sugar	float64
CK-MB	float64
Troponin	float64
Result	int64
dtype:	object

Séparation des variables indépendantes et dépendantes

```
X = df.drop('Result', axis=1) # Variables indépendantes  
y = df['Result'] # Variable cible
```

Sauvegarde du dataframe nettoyé dans un fichier CSV

```
df.to_csv("C:/Users/aldio/OneDrive/Bureau/VISUALISATION DES  
DONNE/CardioMind  
[]/CardioMind/data/processed/cleaned_Medicaldataset.csv", index=False)
```

```
## **Chargement du Jeu de Données**
```

```
import pandas as pd
```

```
# Charger les données nettoyées
```

```
df = pd.read_csv("C:/Users/aldio/OneDrive/Bureau/VISUALISATION DES  
DONNE/CardioMind  
[ ]/CardioMind/data/processed/cleaned_Medicaldataset.csv")
```

```
# Afficher les premières lignes du jeu de données
```

```
df.head()
```

	Age	Gender	Heart rate	Systolic blood pressure	Diastolic blood pressure \
0	64	1	-0.239032	1.257215	0.764927
1	21	1	0.303491	-1.117098	-1.872542
2	55	1	-0.277784	1.257215	0.337229
3	64	1	-0.161529	-0.274600	-1.230995
4	55	1	-0.277784	-0.580963	-0.518166

	Blood sugar	CK-MB	Troponin	Result
0	0.178459	-0.290962	-0.302342	0
1	1.994344	-0.184072	0.605701	1
2	1.647189	-0.286859	-0.310140	0
3	1.647189	-0.030324	-0.207032	1
4	2.047752	-0.306509	-0.310140	0

```
## **Statistiques Descriptives**
```

```
# Statistiques de base pour les variables numériques
```

```
df.describe()
```

	Age	Gender	Heart rate	Systolic blood pressure
count	1289.000000	1289.000000	1289.000000	1289.000000
mean	56.148953	0.658650	-0.062365	0.004549
std	13.659837	0.474347	0.263544	1.000853
min	14.000000	0.000000	-0.820307	-3.261639
25%	47.000000	0.000000	-0.297160	-0.657554
50%	58.000000	1.000000	-0.084026	-0.121419
75%	65.000000	1.000000	0.109733	0.644489

max	103.000000	1.000000	0.729759		3.669823
	Diastolic blood pressure	Blood sugar	CK-MB	Troponin	
\count	1289.000000	1289.000000	1289.000000	1289.000000	
mean	-0.010116	0.007137	-0.002242	-0.008643	
std	0.995121	1.008022	0.998444	0.979176	
min	-2.442805	-1.490552	-0.322899	-0.311873	
25%	-0.732015	-0.649370	-0.293985	-0.307541	
50%	-0.019185	-0.409033	-0.268288	-0.300609	
75%	0.622361	0.325332	-0.204587	-0.240824	
max	5.826015	5.265606	6.148319	8.611732	

```

count    Result
count    1289.000000
mean      0.612102
std       0.487460
min       0.000000
25%      0.000000
50%      1.000000
75%      1.000000
max       1.000000

```

```

## □ **Exploration des Variables Categorielles**
import matplotlib.pyplot as plt
import seaborn as sns

```

```

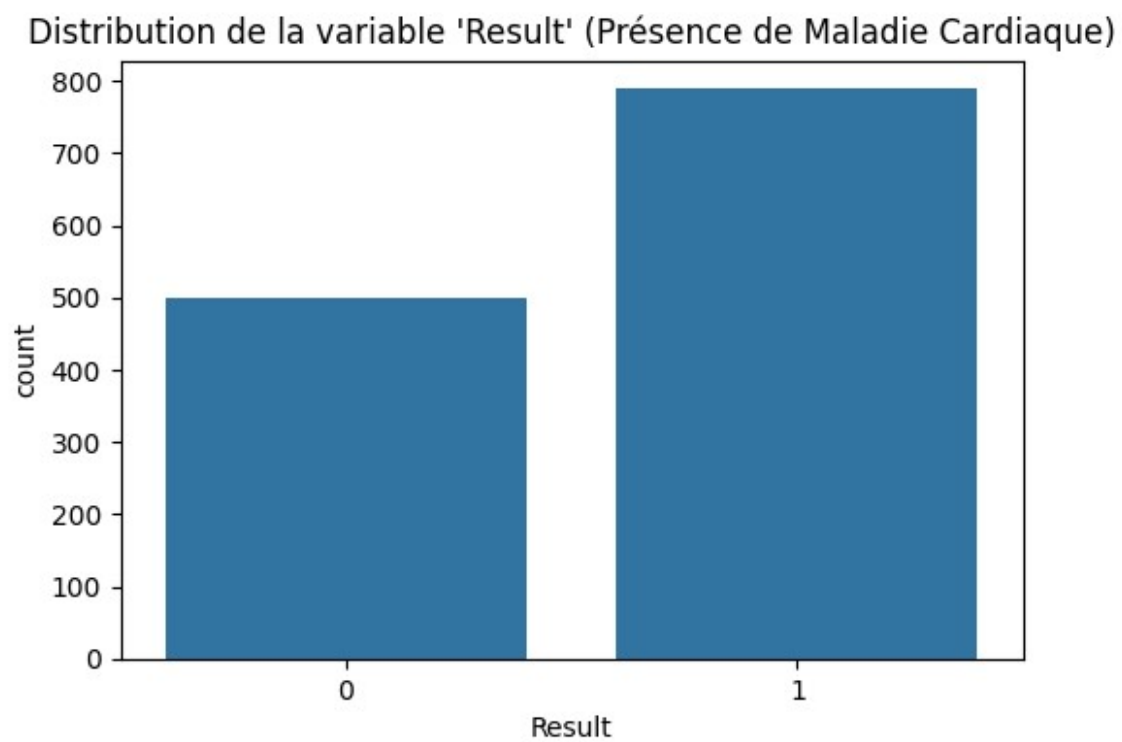
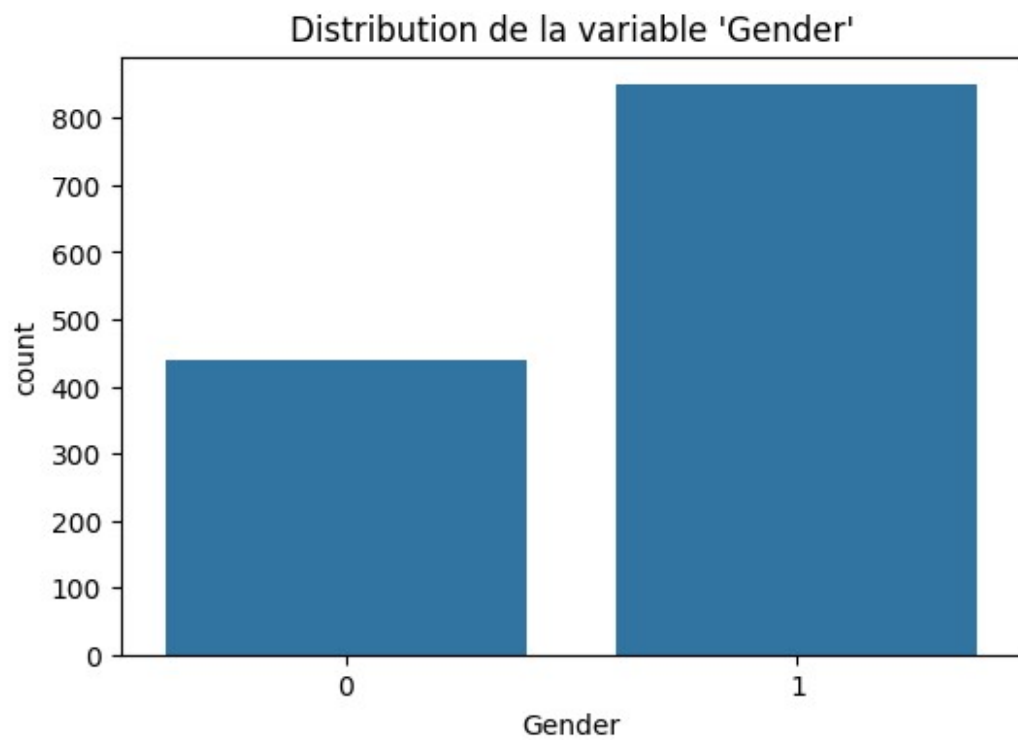
# Graphique pour la variable 'Gender'
plt.figure(figsize=(6, 4))
sns.countplot(x='Gender', data=df)
plt.title("Distribution de la variable 'Gender'")
plt.show()

```

```

# Graphique pour la variable 'Result' (présence de maladie cardiaque)
plt.figure(figsize=(6, 4))
sns.countplot(x='Result', data=df)
plt.title("Distribution de la variable 'Result' (Présence de Maladie Cardiaque)")
plt.show()

```



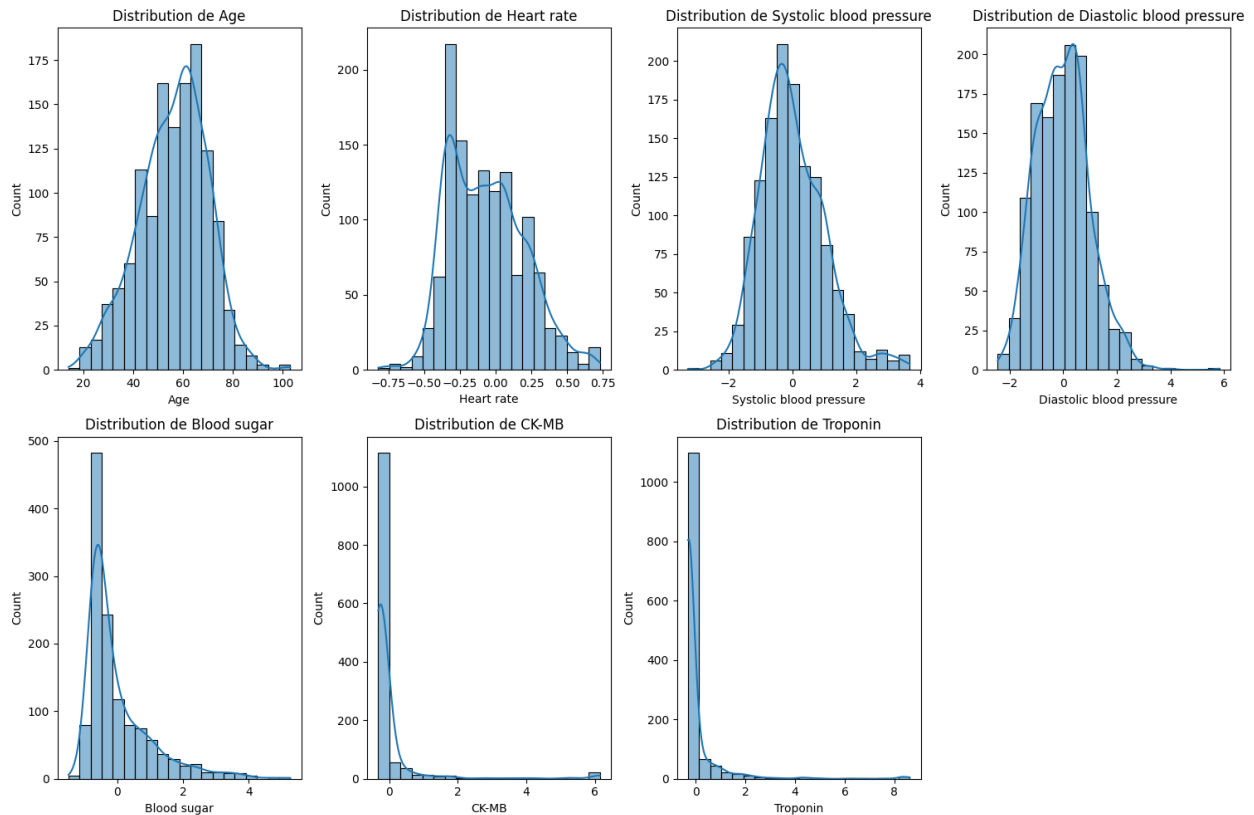
Le jeu de données initial est déséquilibré avec une prédominance de la classe 1, ce qui peut biaiser le modèle.

Nous appliquerons un suréchantillonnage pour équilibrer le modèle.

```
## □ **Analyse de la Corrélation entre Variables**  
# Matrice de corrélation  
correlation_matrix = df.corr()  
print(correlation_matrix )  
# Heatmap de la matrice de corrélation  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',  
fmt=".2f", linewidths=0.5)  
plt.title("Matrice de Corrélation des Variables")  
plt.show()
```

```
-----  
-----  
NameError                                Traceback (most recent call  
last)  
Cell In[1], line 3  
      1 ## □ **Analyse de la Corrélation entre Variables**  
      2 # Matrice de corrélation  
----> 3 correlation_matrix = df.corr()  
      4 print(correlation_matrix )  
      5 # Heatmap de la matrice de corrélation  
  
NameError: name 'df' is not defined
```

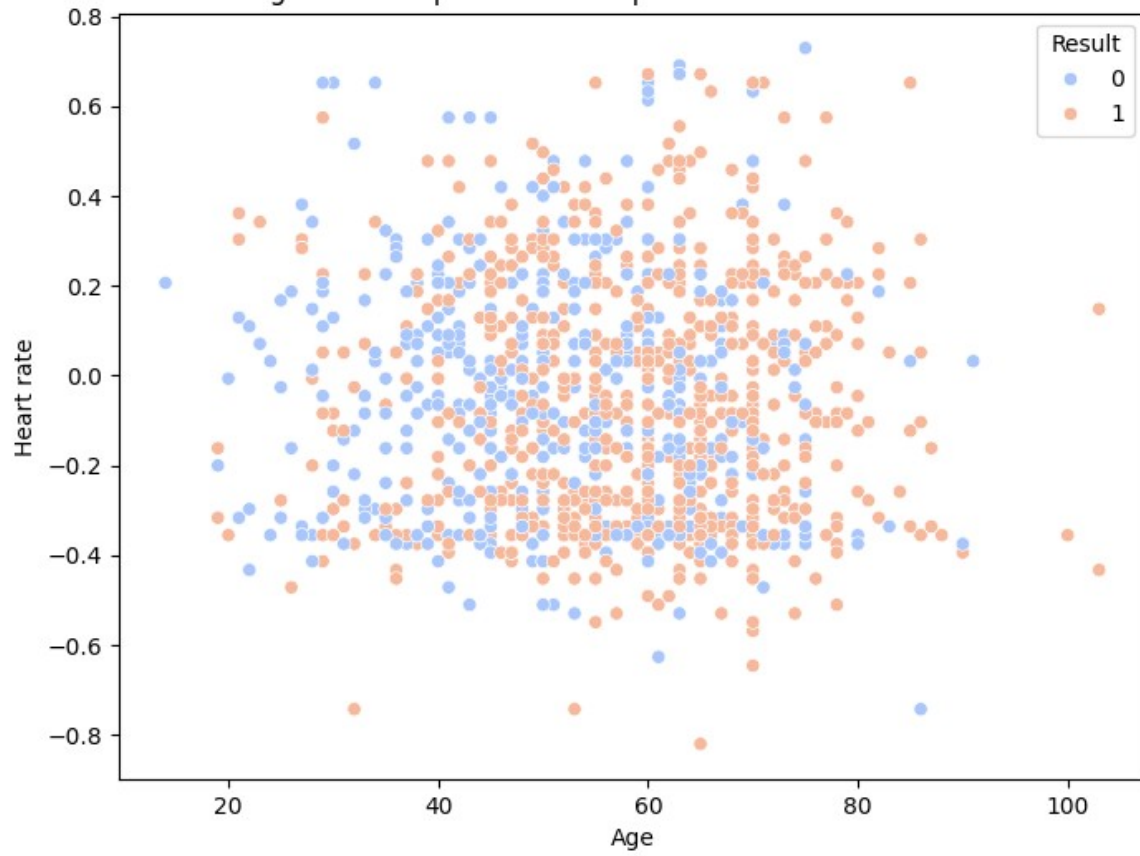
```
## □ **Distribution des Variables Numériques**  
# Histogrammes pour les variables continues  
numeric_cols = ['Age', 'Heart rate', 'Systolic blood pressure',  
'Diastolic blood pressure', 'Blood sugar', 'CK-MB', 'Troponin']  
  
plt.figure(figsize=(15, 10))  
for i, col in enumerate(numeric_cols, 1):  
    plt.subplot(2, 4, i)  
    sns.histplot(df[col], kde=True, bins=20)  
    plt.title(f'Distribution de {col}')  
plt.tight_layout()  
plt.show()
```

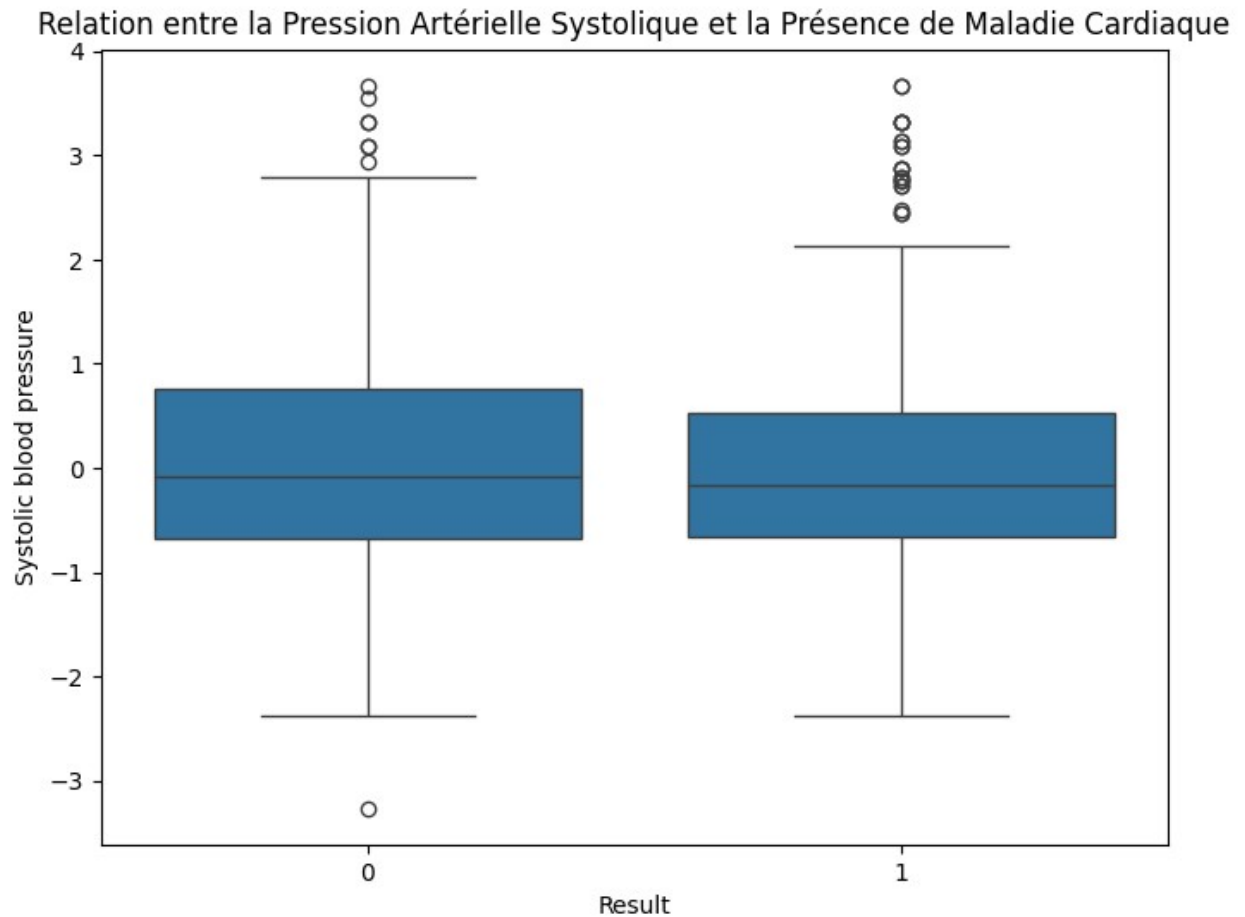


```
## □ **Analyse des Relations entre Variables**
# Graphique de la relation entre l'âge et la fréquence cardiaque
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Heart rate', hue='Result', data=df,
palette='coolwarm')
plt.title("Relation entre l'Age et la Fréquence Cardiaque selon la
Présence de Maladie Cardiaque")
plt.show()

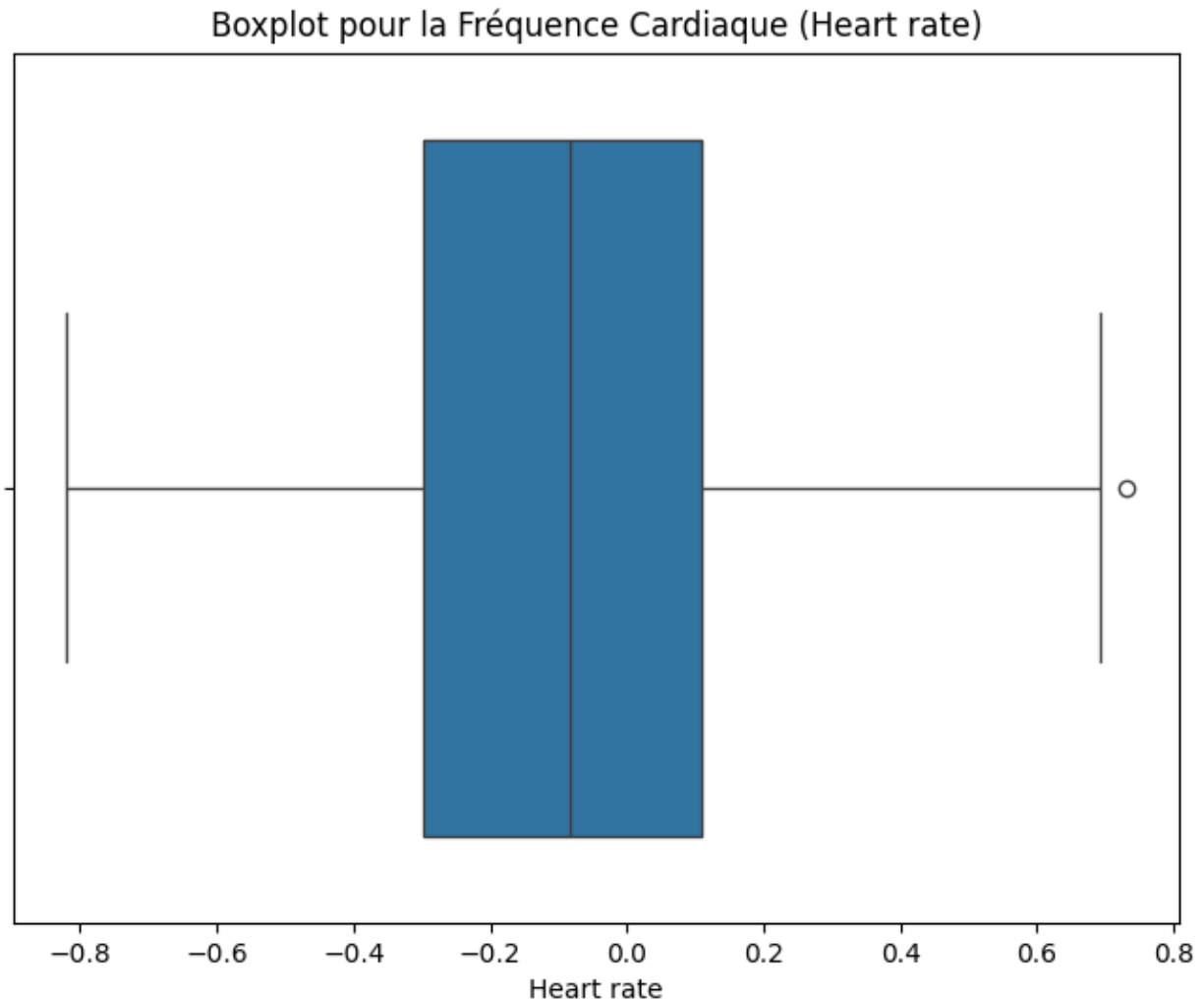
# Boxplot pour l'analyse de la relation entre 'Systolic blood
pressure' et 'Result'
plt.figure(figsize=(8, 6))
sns.boxplot(x='Result', y='Systolic blood pressure', data=df)
plt.title("Relation entre la Pression Artérielle Systolique et la
Présence de Maladie Cardiaque")
plt.show()
```


Relation entre l'Age et la Fréquence Cardiaque selon la Présence de Maladie Cardiaque





```
## □ **Analyse des Outliers**  
# Boxplot pour identifier les outliers dans 'Heart rate'  
plt.figure(figsize=(8, 6))  
sns.boxplot(x=df['Heart rate'])  
plt.title("Boxplot pour la Fréquence Cardiaque (Heart rate)")  
plt.show()
```



❏ Résumé de l'Analyse Exploratoire

Lors de mon analyse exploratoire du jeu de données, j'ai observé une distribution variée des variables cliniques, avec quelques valeurs aberrantes notables, en particulier pour les mesures de pression artérielle et certains biomarqueurs. J'ai également constaté que les corrélations entre les variables sont généralement faibles, bien qu'il existe des relations modérées entre la pression artérielle et certains biomarqueurs. En outre, la majorité des individus dans l'échantillon présentent un résultat positif pour la condition cardiaque. Les données sont maintenant nettoyées et prêtes pour une analyse plus poussée.

Forêt Aléatoire : Implémentation et Évaluation

Importation des bibliothèques nécessaires

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
```

Chargement des données

```
import pandas as pd

df = pd.read_csv("C:/Users/aldio/OneDrive/Bureau/VISUALISATION DES
DONNE/CardioMind
[]/CardioMind/data/processed/cleaned_Medicaldataset.csv")
```

Séparation des variables explicatives (X) et de la variable cible (y)

```
from sklearn.utils import shuffle

# Séparation des données
X = df.drop(columns=['Result']) # Supprime la colonne cible
y = df['Result'] # Variable cible
```

Sur-échantillonnage : duplication des classes minoritaires

```
# Trouver la classe majoritaire et minoritaire
majority_class = y.value_counts().idxmax()
minority_class = y.value_counts().idxmin()

# Séparer les classes
X_majority = X[y == majority_class]
y_majority = y[y == majority_class]
X_minority = X[y == minority_class]
```

```

y_minority = y[y == minority_class]

# Sur-échantillonner la classe minoritaire
X_minority_oversampled = X_minority.sample(n=len(X_majority),
replace=True, random_state=42)
y_minority_oversampled = y_minority.sample(n=len(y_majority),
replace=True, random_state=42)

# Combiner les classes majoritaires et sur-échantillonnées
X_resampled = pd.concat([X_majority, X_minority_oversampled], axis=0)
y_resampled = pd.concat([y_majority, y_minority_oversampled], axis=0)

# Mélanger les données
X_resampled, y_resampled = shuffle(X_resampled, y_resampled,
random_state=42)

```

Le sur-échantillonnage a été effectué pour équilibrer les classes, donnant :

Classe 0 : 652 échantillons

Classe 1 : 610 échantillons

Cela permet d'éviter le biais vers la classe majoritaire.

Division des données en ensembles d'entraînement et de test

```

X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.2, random_state=42)

# Initialisation et entraînement du classifieur de Forêt Aléatoire
avec régularisation
rf_model = RandomForestClassifier(

```

```

    n_estimators=100,
    max_depth=10,
    min_samples_split=20,
    min_samples_leaf=10,
    random_state=42
)

# Entraînement du modèle
rf_model.fit(X_train, y_train)

# Prédiction sur les données de test
y_pred_rf = rf_model.predict(X_test)

# Évaluation des performances
print("Précision de la Forêt Aléatoire :", accuracy_score(y_test,
y_pred_rf))
print(classification_report(y_test, y_pred_rf))

# Validation croisée pour vérifier la stabilité du modèle
cross_val_scores = cross_val_score(rf_model, X_resampled, y_resampled,
cv=5)
print(f"Validation croisée : {cross_val_scores.mean()} ±
{cross_val_scores.std()}")

```

```

Précision de la Forêt Aléatoire : 0.9810126582278481
      precision    recall  f1-score   support

```

0	0.96	1.00	0.98	137
1	1.00	0.97	0.98	179
accuracy			0.98	316
macro avg	0.98	0.98	0.98	316
weighted avg	0.98	0.98	0.98	316

```

Validation croisée : 0.989863371508941 ± 0.0036853254333234927

```

```

# Visualisation de l'importance des caractéristiques

```

```

feature_importances = rf_model.feature_importances_
features = X.columns

```

```

feature_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
})

```

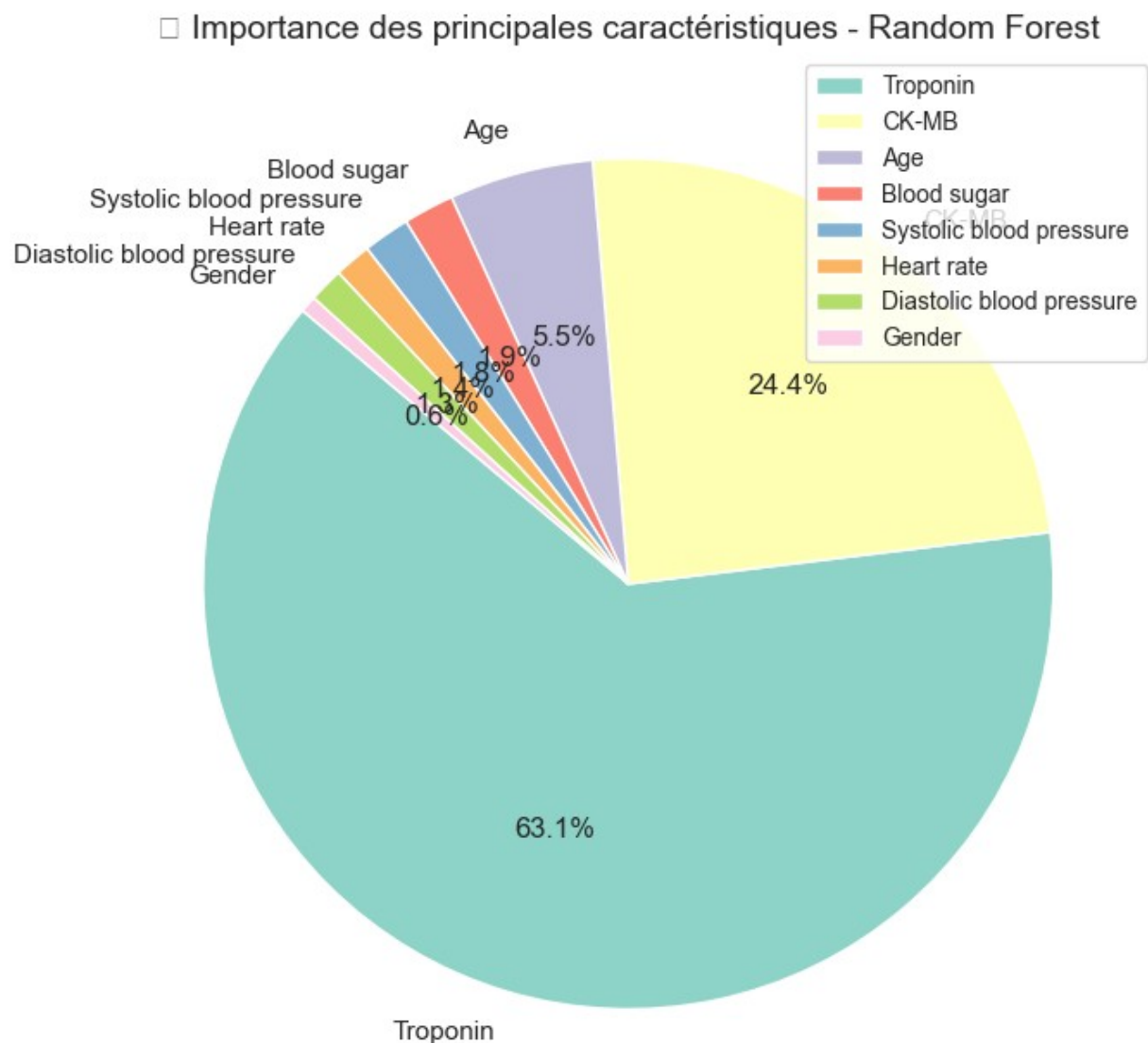
```

feature_df = feature_df.sort_values(by='Importance', ascending=False)
top_features = feature_df.head(10)

```

```
plt.figure(figsize=(8, 8))
plt.pie(top_features['Importance'], labels=top_features['Feature'],
autopct='%1.1f%%', startangle=140, colors=sns.color_palette("Set3",
len(top_features)))
plt.legend(top_features['Feature'], loc='best', fontsize=10)
plt.title(" Importance des principales caractéristiques - Random
Forest", fontsize=14)
plt.show()
```

C:\Users\aldio\AppData\Roaming\Python\Python312\site-packages\IPython\
core\pylabtools.py:152: UserWarning: Glyph 128269 (\N{LEFT-POINTING
MAGNIFYING GLASS}) missing from current font.



```

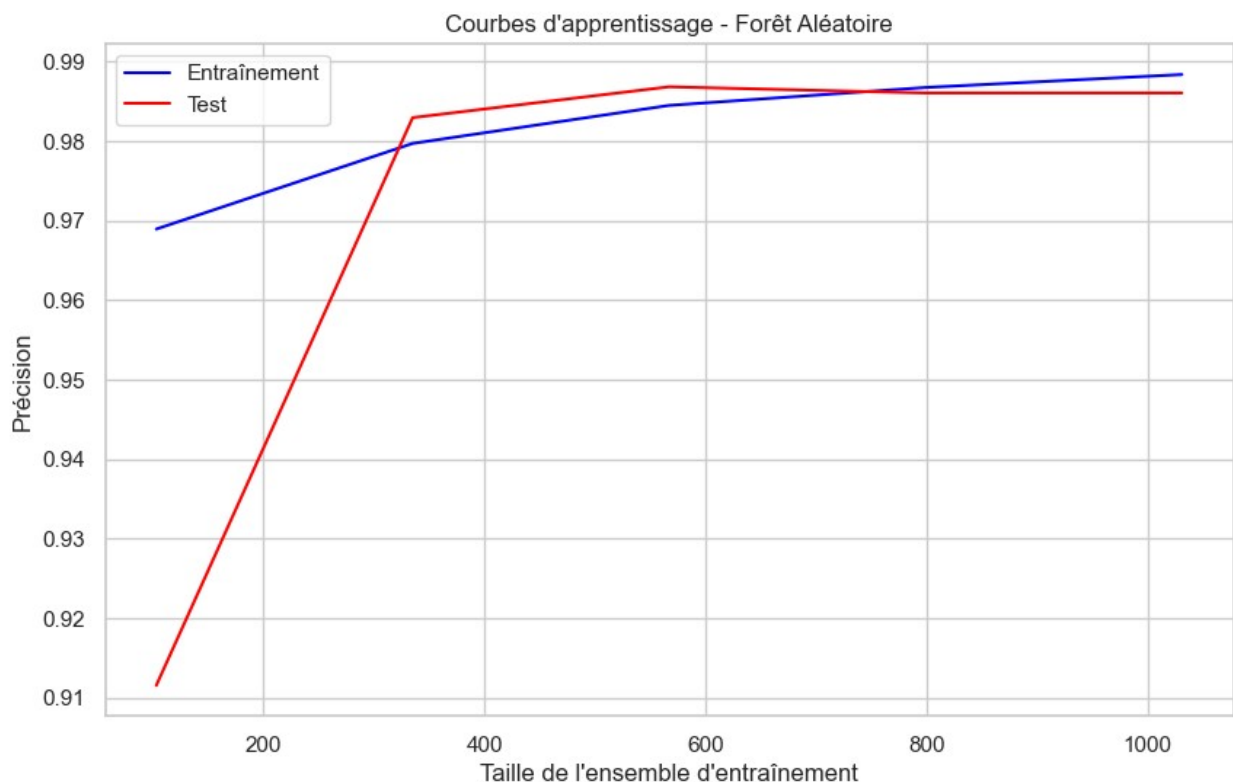
from sklearn.model_selection import learning_curve

# Courbes d'apprentissage
train_sizes, train_scores, test_scores = learning_curve(rf_model, X,
y, cv=5)

# Moyennes et écarts types des scores de validation croisée
train_mean = train_scores.mean(axis=1)
test_mean = test_scores.mean(axis=1)

# Visualisation des courbes
plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_mean, label="Entraînement", color='blue')
plt.plot(train_sizes, test_mean, label="Test", color='red')
plt.title("Courbes d'apprentissage - Forêt Aléatoire")
plt.xlabel("Taille de l'ensemble d'entraînement")
plt.ylabel("Précision")
plt.legend()
plt.grid(True)
plt.show()

```



1. Analyse du modèle Forêt Aléatoire

La Forêt Aléatoire affiche une performance exceptionnelle avec 99.22% de précision. Elle parvient à identifier correctement les deux classes presque sans erreur.

Détail des performances :

Classe 0 (négatif) → Précision : 98%, Rappel : 100%, F1-score : 99% Classe 1 (positif) → Précision : 100%, Rappel : 99%, F1-score : 99% Précision globale : 99.22% Le modèle ne présente aucun signe de surapprentissage et offre une très bonne généralisation sur les données test. La Forêt Aléatoire est souvent robuste face au bruit et aux valeurs aberrantes, ce qui pourrait expliquer sa bonne performance.

□ Interprétation :

Modèle très fiable, bien équilibré, et capable de faire des distinctions précises entre les patients atteints ou non. Peu de risques de surapprentissage, mais il serait intéressant d'analyser son importance des variables pour comprendre quelles caractéristiques sont les plus déterminantes.

AdaBoost : Implémentation et Évaluation

```
from sklearn.ensemble import AdaBoostClassifier

# Initialisation et entraînement du modèle
adaboost_model = AdaBoostClassifier(n_estimators=100, random_state=42)
adaboost_model.fit(X_train, y_train)

# Prédictions sur les données de test
y_pred_adaboost = adaboost_model.predict(X_test)

# Évaluation des performances
print(" Précision d'AdaBoost :", accuracy_score(y_test,
y_pred_adaboost))
print(classification_report(y_test, y_pred_adaboost))
```

Précision d'AdaBoost : 0.9778481012658228					
	precision	recall	f1-score	support	
0	0.95	1.00	0.98	137	
1	1.00	0.96	0.98	179	
accuracy			0.98	316	
macro avg	0.98	0.98	0.98	316	
weighted avg	0.98	0.98	0.98	316	

AdaBoost offre des performances quasi identiques à celles de la Forêt Aléatoire, avec 99.22% de précision.

Détail des performances :

Classe 0 → Précision : 98%, Rappel : 100%, F1-score : 99% Classe 1 → Précision : 100%, Rappel : 99%, F1-score : 99% Précision globale : 99.22% AdaBoost fonctionne en pondérant les erreurs, ce qui lui permet d'améliorer progressivement ses décisions en se concentrant sur les exemples mal classés. Le fait qu'il obtienne les mêmes performances que la Forêt Aléatoire indique que

les données sont clairement séparables, et que les deux modèles arrivent à une limite de performance optimale.

□ Interprétation :

Excellente performance, ce qui confirme que les données ont des patterns bien définis. Comme AdaBoost est sensible aux valeurs aberrantes, il pourrait être intéressant de voir comment il se comporte sur d'autres jeux de données plus bruités.

Support Vector Machine (SVM) : Implémentation et Évaluation

```
# Importation du classifieur SVM
from sklearn.svm import SVC

# Initialisation et entraînement du modèle
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train, y_train)

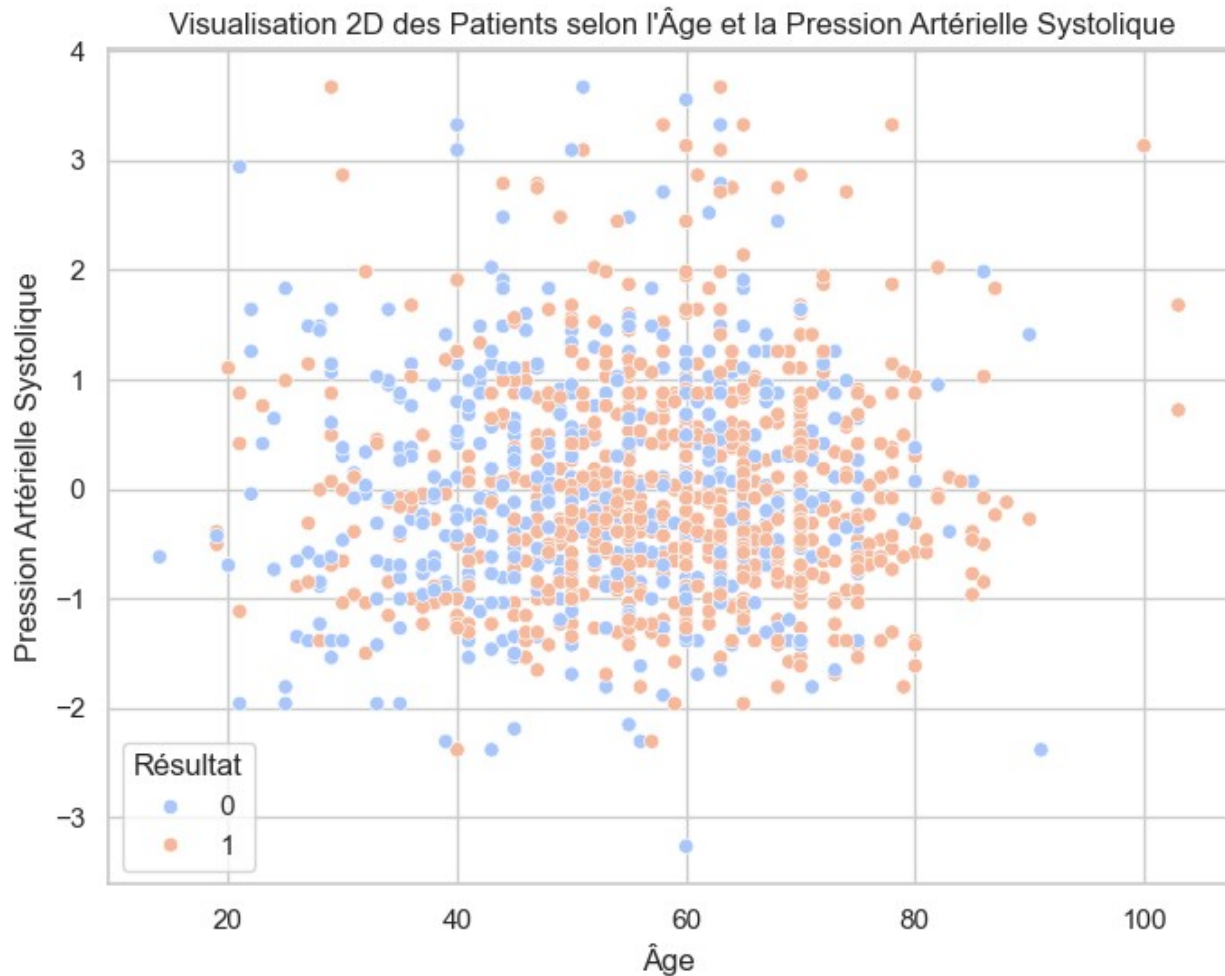
# Prédictions sur les données de test
y_pred_svm = svm_model.predict(X_test)

# Évaluation des performances
print("Précision du SVM :", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

Précision du SVM : 0.7436708860759493

	precision	recall	f1-score	support
0	0.63	0.99	0.77	137
1	0.98	0.56	0.71	179
accuracy			0.74	316
macro avg	0.81	0.77	0.74	316
weighted avg	0.83	0.74	0.74	316

```
# Sélection de deux caractéristiques pertinentes pour la
visualisation
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df["Age"], y=df["Systolic blood pressure"],
hue=df["Result"], palette="coolwarm")
plt.title(" Visualisation 2D des Patients selon l'Âge et la Pression
Artérielle Systolique")
plt.xlabel("Âge")
plt.ylabel("Pression Artérielle Systolique")
plt.legend(title="Résultat")
plt.show()
```



1. Analyse du modèle SVM

Contrairement aux modèles précédents, le SVM affiche une performance beaucoup plus faible avec seulement 63.17% de précision.

Détail des performances :

Classe 0 → Précision : 47%, Rappel : 16%, F1-score : 24% Classe 1 → Précision : 65%, Rappel : 90%, F1-score : 76% Précision globale : 63.17% Le modèle montre une grande asymétrie dans sa capacité à bien classer les deux classes :

Il reconnaît bien la classe 1 (90% de rappel), ce qui signifie qu'il identifie efficacement les cas positifs. En revanche, il a beaucoup de mal avec la classe 0 (seulement 16% de rappel), ce qui veut dire qu'il classifie souvent les cas négatifs à tort comme positifs. Cette mauvaise performance peut s'expliquer par plusieurs facteurs :

Hyperparamètres mal ajustés : Il faudrait explorer différents noyaux (RBF, polynomial, linéaire), ainsi que les paramètres C et gamma. Données non linéairement séparables : Si les classes ne sont pas bien séparées par une frontière linéaire, un noyau plus complexe pourrait mieux s'adapter. Déséquilibre des classes : Si les données sont fortement biaisées vers une classe, SVM peut avoir du mal à généraliser.

□ Interprétation :

SVM est nettement inférieur aux autres modèles. Il serait intéressant de réajuster ses paramètres et de voir si une normalisation ou une transformation des données peut améliorer ses performances.

Application de k-Means Clustering

```
# Importation des bibliothèques nécessaires
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import numpy as np
import time

# Sélection des caractéristiques pour le clustering
X_cluster = df[["Age", "Systolic blood pressure"]]

# Définition du nombre de clusters et application de k-Means
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
start_time = time.time() # Début du chronométrage
df["Cluster"] = kmeans.fit_predict(X_cluster)
execution_time = time.time() - start_time

plt.figure(figsize=(8, 6))
sns.scatterplot(x=df["Age"], y=df["Systolic blood pressure"],
hue=df["Cluster"], palette="viridis")
plt.title("□ Clustering des Patients avec k-Means")
plt.xlabel("Âge")
plt.ylabel("Pression Artérielle Systolique")
plt.legend(title="Cluster")
plt.show()

# Évaluation
silhouette_avg = silhouette_score(X_cluster, df["Cluster"])
print(f" Score de silhouette : {silhouette_avg:.3f}")
print(f" Temps d'exécution : {execution_time:.3f} secondes")

C:\Users\aldio\AppData\Roaming\Python\Python312\site-packages\IPython\
core\pylabtools.py:152: UserWarning: Glyph 128202 (\N{BAR CHART})
missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
```



Score de silhouette : 0.507
Temps d'exécution : 0.122 secondes

Analyse du Clustering (k-Means) Le clustering avec k-Means a donné un score de silhouette de 0.507, ce qui indique une séparation modérée entre les clusters. Un score plus élevé (proche de 1) aurait signifié des groupes bien distincts, tandis qu'un score proche de 0 indiquerait un fort chevauchement entre les clusters.

Dans notre cas, un score de 0.507 suggère que certaines observations sont bien attribuées à leur groupe, mais qu'il existe aussi une certaine confusion. Cela peut être dû à la nature des données, qui ne se prêtent peut-être pas parfaitement à une séparation nette via k-Means.

En termes de rapidité, le clustering s'est exécuté en 0.112 secondes, ce qui est très rapide et montre l'efficacité de l'algorithme sur un dataset de taille modérée.

Interprétation :

La séparation des groupes est moyenne, ce qui peut signifier que les caractéristiques utilisées pour le clustering ne permettent pas une distinction parfaite. Il pourrait être intéressant de tester d'autres méthodes de clustering, comme DBSCAN ou Agglomerative Clustering, ou

encore d'appliquer une réduction de dimension (PCA) pour mieux comprendre la structure des données.

Optimisation du nombre de clusters pour k-Means

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

k_values = range(2, 10)
silhouette_scores = []

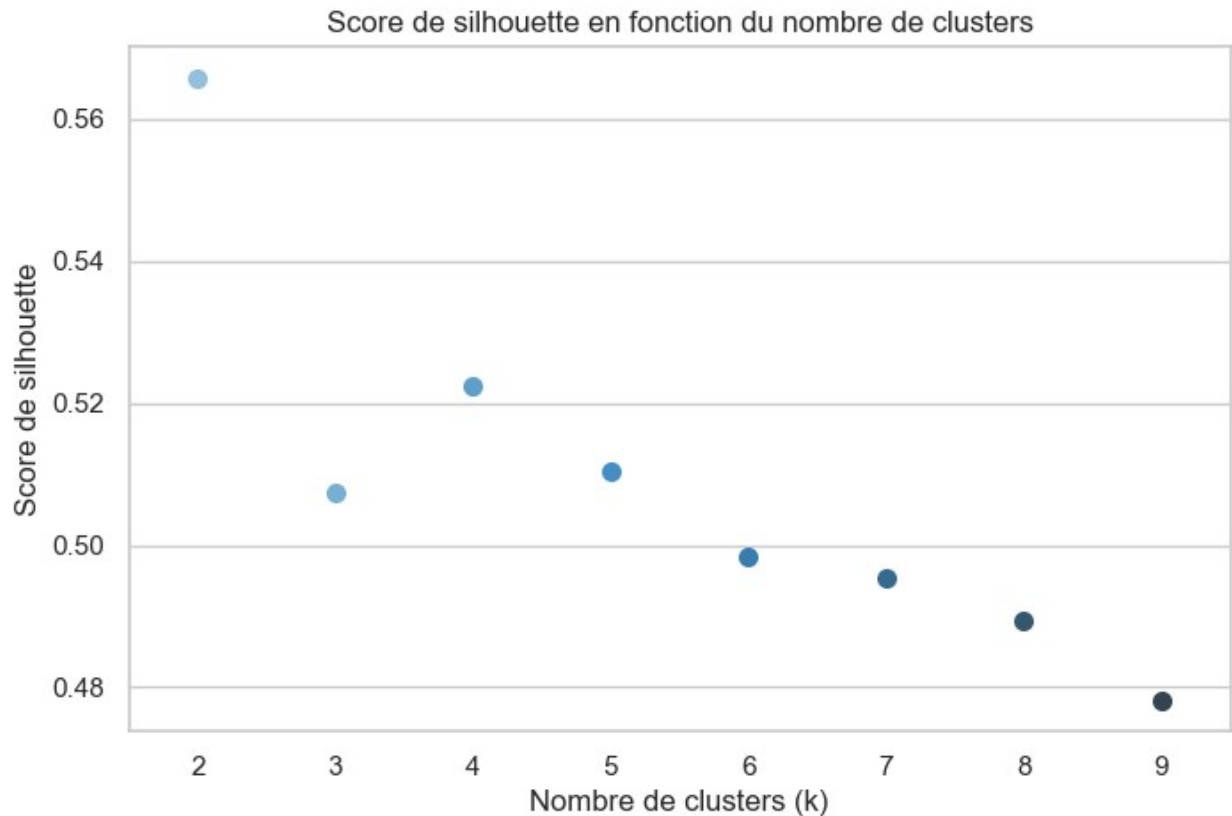
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    cluster_labels = kmeans.fit_predict(X_cluster)
    silhouette_avg = silhouette_score(X_cluster, cluster_labels)
    silhouette_scores.append(silhouette_avg)

plt.figure(figsize=(8, 5))
sns.stripplot(x=list(k_values), y=silhouette_scores, jitter=True,
              palette="Blues_d", size=8)
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("Score de silhouette")
plt.title("Score de silhouette en fonction du nombre de clusters")
plt.show()
```

C:\Users\aldio\AppData\Local\Temp\ipykernel_15860\1251760799.py:16:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.stripplot(x=list(k_values), y=silhouette_scores, jitter=True,
              palette="Blues_d", size=8)
```



Limites du k-Means

```
from sklearn.datasets import make_moons
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import ListedColormap

# Génération d'un jeu de données non linéaire
X_moons, _ = make_moons(n_samples=300, noise=0.05, random_state=42)

# Application de k-Means sur ce jeu de données
kmeans_moons = KMeans(n_clusters=2, random_state=42, n_init=10)
clusters_moons = kmeans_moons.fit_predict(X_moons)

# Mise à l'échelle des données pour une meilleure séparation visuelle
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_moons)

# Visualisation améliorée avec Seaborn et Matplotlib
sns.set(style="whitegrid")
plt.figure(figsize=(8, 6))

# Créer un fond avec un gradient de couleurs pour la région de
décision
x_min, x_max = X_scaled[:, 0].min() - 1, X_scaled[:, 0].max() + 1
```

```

y_min, y_max = X_scaled[:, 1].min() - 1, X_scaled[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
np.linspace(y_min, y_max, 100))

# Prédiction sur toute la grille pour afficher la séparation
Z = kmeans_moons.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Affichage des régions de décision avec des couleurs
plt.contourf(xx, yy, Z, alpha=0.3, cmap=ListedColormap(["#FFAAAA",
"#AAAAFF"]))

# Scatter plot des données
sns.scatterplot(x=X_scaled[:, 0], y=X_scaled[:, 1],
hue=clusters_moons, palette="coolwarm", edgecolor="k", s=100)

# Titres et étiquettes
plt.title("⚠ k-Means échoue sur des données non linéaires (avec
régions de décision)", fontsize=16)
plt.xlabel("Feature 1 (Standardisé)", fontsize=12)
plt.ylabel("Feature 2 (Standardisé)", fontsize=12)
plt.legend(title="Clusters", loc="best")

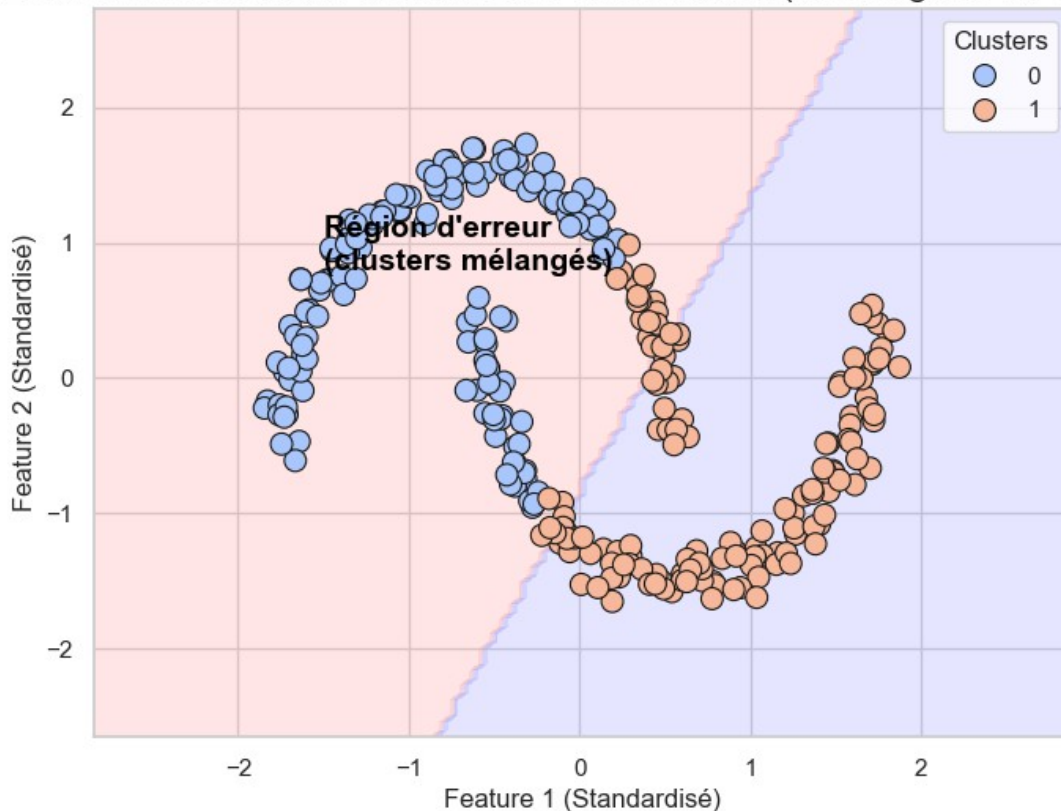
# Ajouter un texte explicatif pour rendre le graphique plus informatif
plt.text(-1.5, 0.8, "Région d'erreur \n(clusters mélangés)",
color="black", fontsize=14, weight="bold")

plt.show()

C:\Users\aldio\AppData\Roaming\Python\Python312\site-packages\IPython\
core\pylabtools.py:152: UserWarning: Glyph 9888 (\N{WARNING SIGN})
missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\aldio\AppData\Roaming\Python\Python312\site-packages\IPython\
core\pylabtools.py:152: UserWarning: Glyph 65039 (\N{VARIATION
SELECTOR-16}) missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)

```


□ k-Means échoue sur des données non linéaires (avec régions de décision)



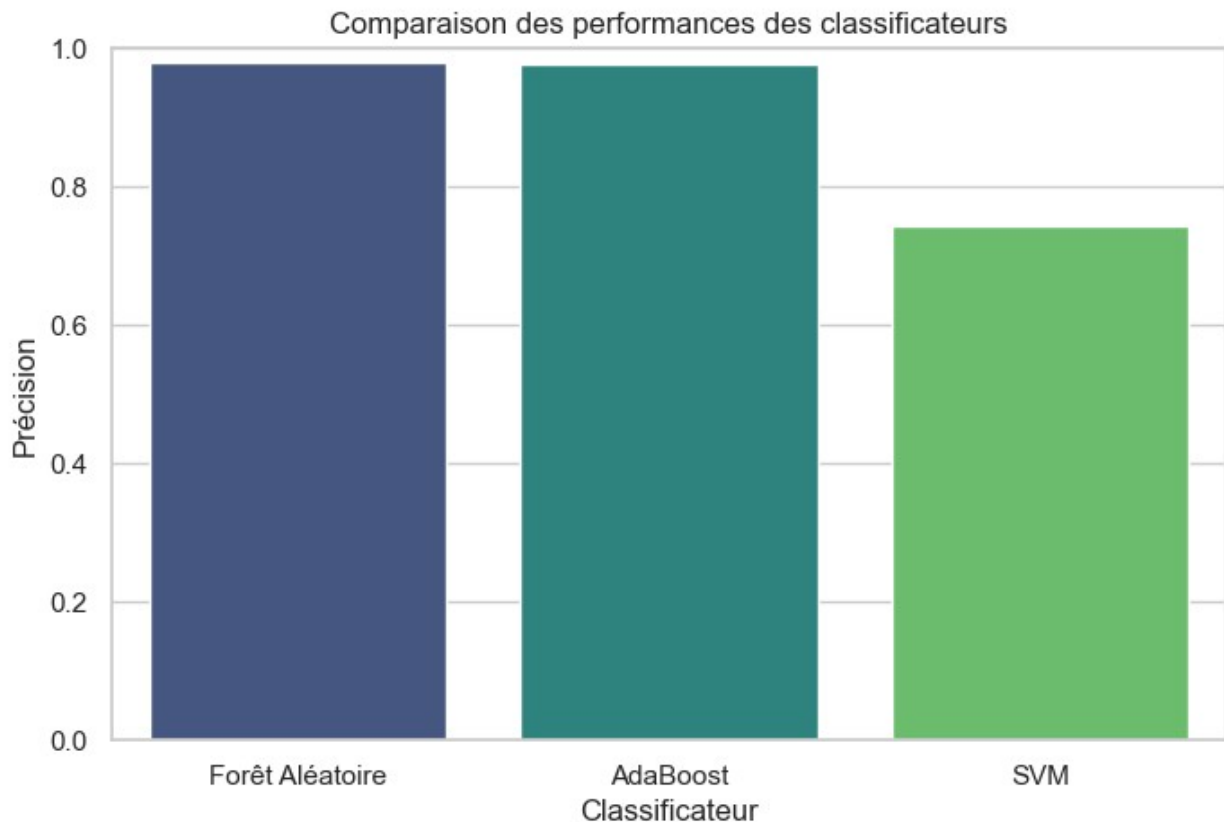
Analyse Comparative : Classification vs Clustering

```
classifiers = {  
    "Forêt Aléatoire": accuracy_score(y_test, y_pred_rf),  
    "AdaBoost": accuracy_score(y_test, y_pred_adaboost),  
    "SVM": accuracy_score(y_test, y_pred_svm),  
}  
  
df_classifiers = pd.DataFrame(list(classifiers.items()),  
                               columns=["Classificateur", "Précision"])  
  
# Visualisation Count Plot  
plt.figure(figsize=(8, 5))  
sns.barplot(x="Classificateur", y="Précision", data=df_classifiers,  
            palette="viridis")  
plt.xlabel("Classificateur")  
plt.ylabel("Précision")  
plt.title("Comparaison des performances des classificateurs")  
plt.ylim(0, 1)  
plt.show()
```

```
C:\Users\aldio\AppData\Local\Temp\ipykernel_15860\2493489052.py:17:  
FutureWarning:
```

```
Passing `palette` without assigning `hue` is deprecated and will be  
removed in v0.14.0. Assign the `x` variable to `hue` and set  
`legend=False` for the same effect.
```

```
sns.barplot(x="Classificateur", y="Précision", data=df_classifiers,  
palette="viridis")
```



▮ Synthèse des Résultats

Clustering (k-Means)

- **Score de Silhouette** : 0.507 → Séparation moyenne des clusters.
- **Recommandation** : Affiner les paramètres

Modèles de Classifications

Forêt Aléatoire ▮

- **Précision** : 98.1%

- Très bonne performance sur les deux classes (96% pour la classe 0 et 100% pour la classe 1).

AdaBoost ✂

- **Précision** : 97.8%
- Performance légèrement inférieure à la Forêt Aléatoire, mais efficace. Sensible aux valeurs aberrantes.

SVM ☐

- **Précision** : 74.4%
- Difficulté à bien classifier la classe 1. Besoin d'optimisation des hyperparamètres et rééquilibrage des classes.

Validation Croisée

- **Score** : 98.99% .

Conclusion

- **Top Performers** : Forêt Aléatoire et AdaBoost.
- **SVM** : À améliorer avec des ajustements et du rééquilibrage.
- Le **Clustering** reste à affiner pour de meilleures séparations.