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()
```

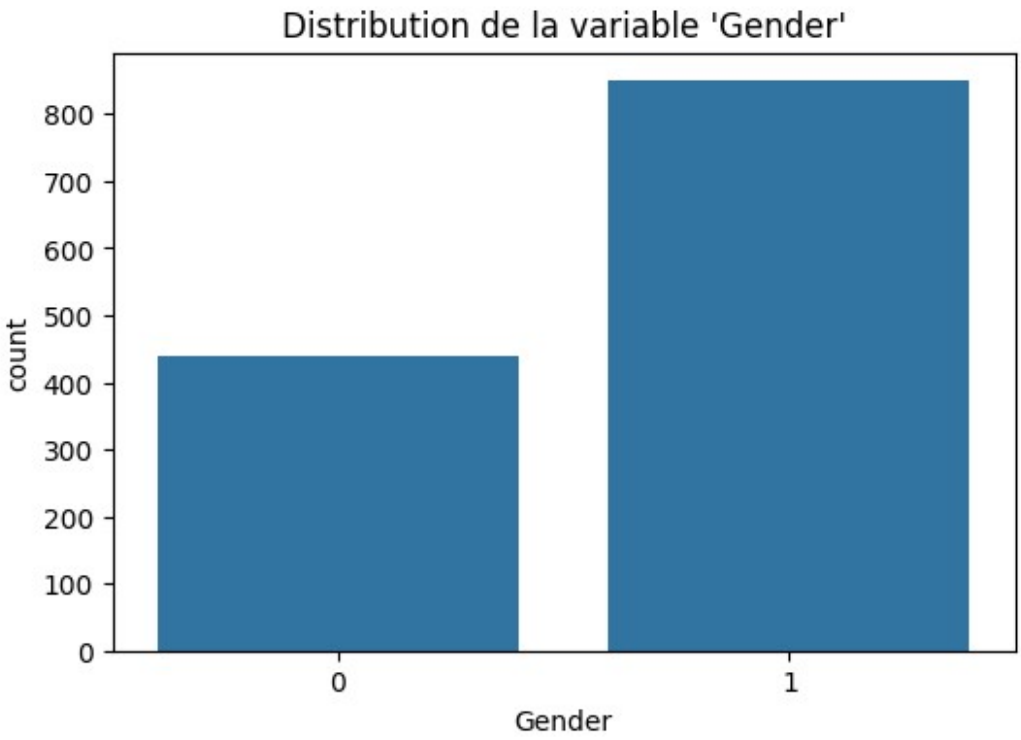
```
   Age  Gender  Heart rate  Systolic blood pressure  Diastolic blood
pressure  \
0   64       1   -0.239032                 1.257215
0.764927
1   21       1    0.303491                -1.117098                -
1.872542
2   55       1   -0.277784                 1.257215
0.337229
3   64       1   -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
```

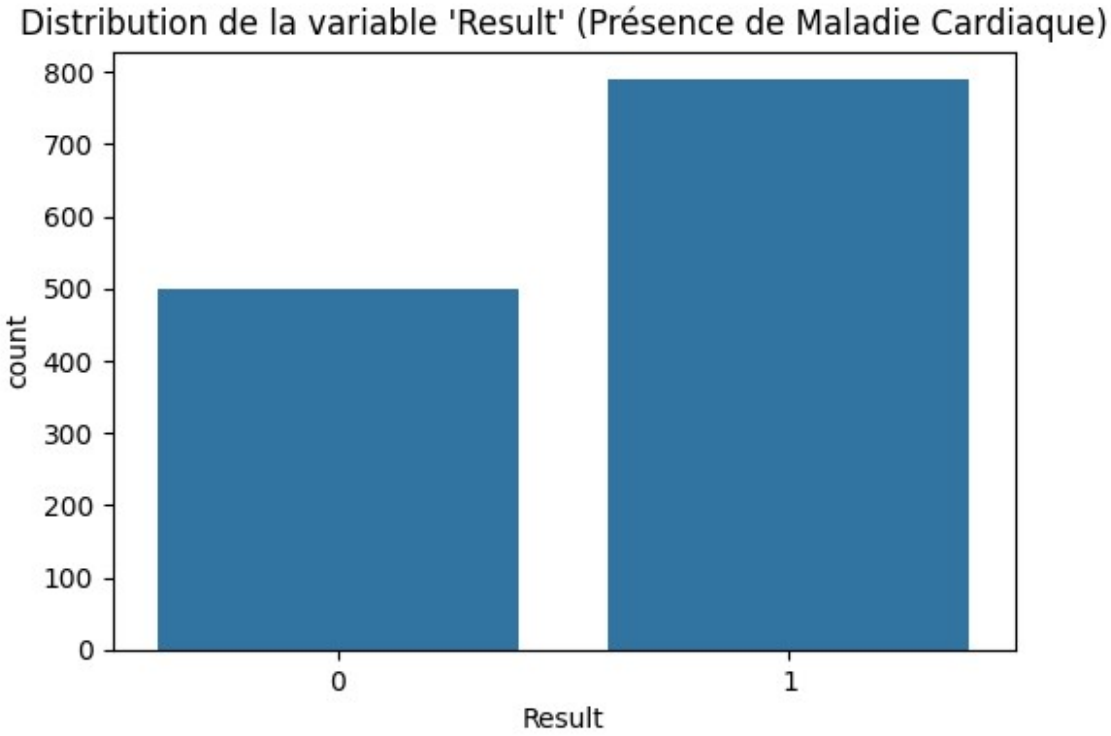
```
## 📊 **Statistiques Descriptives**
# Statistiques de base pour les variables numériques
df.describe()
```

```
                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


             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
```

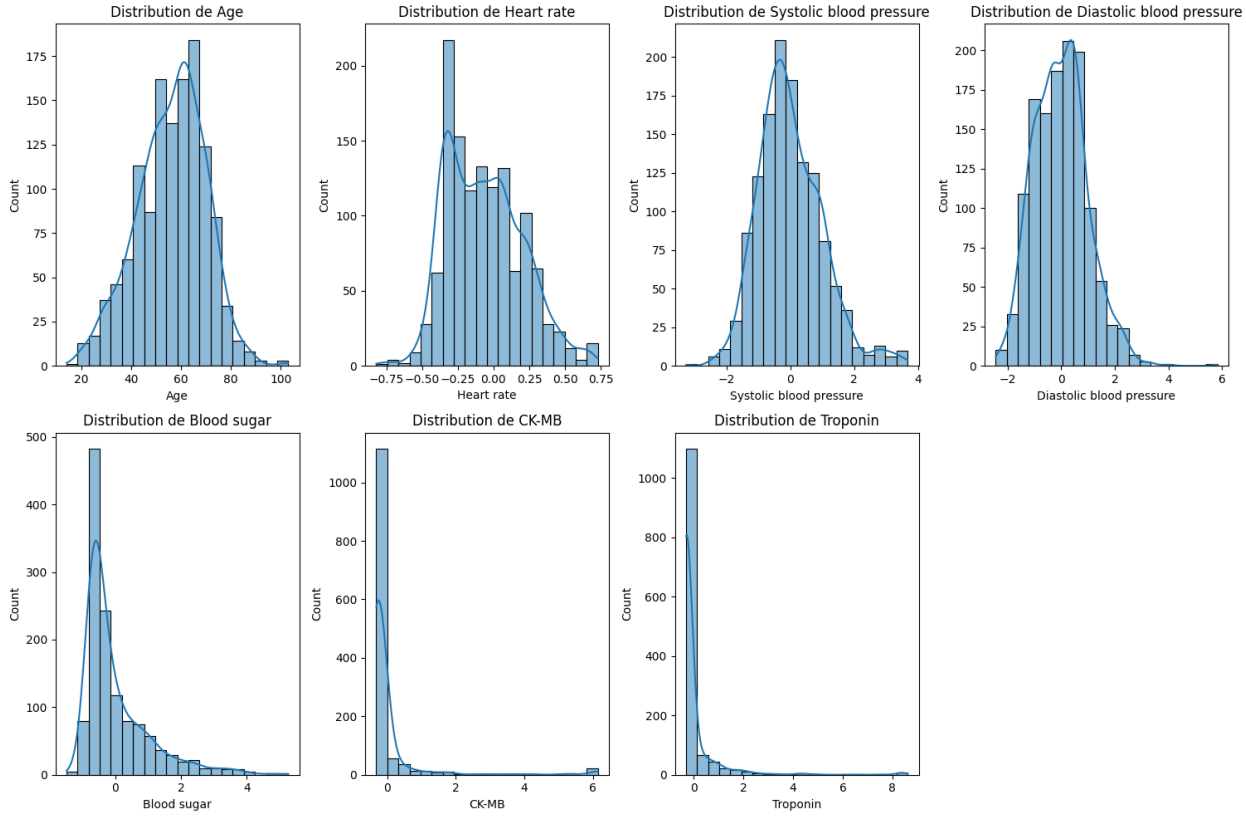
```
## □ **Exploration des Variables Categorielles**
import matplotlib.pyplot as plt
import seaborn as sns

# 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()
```



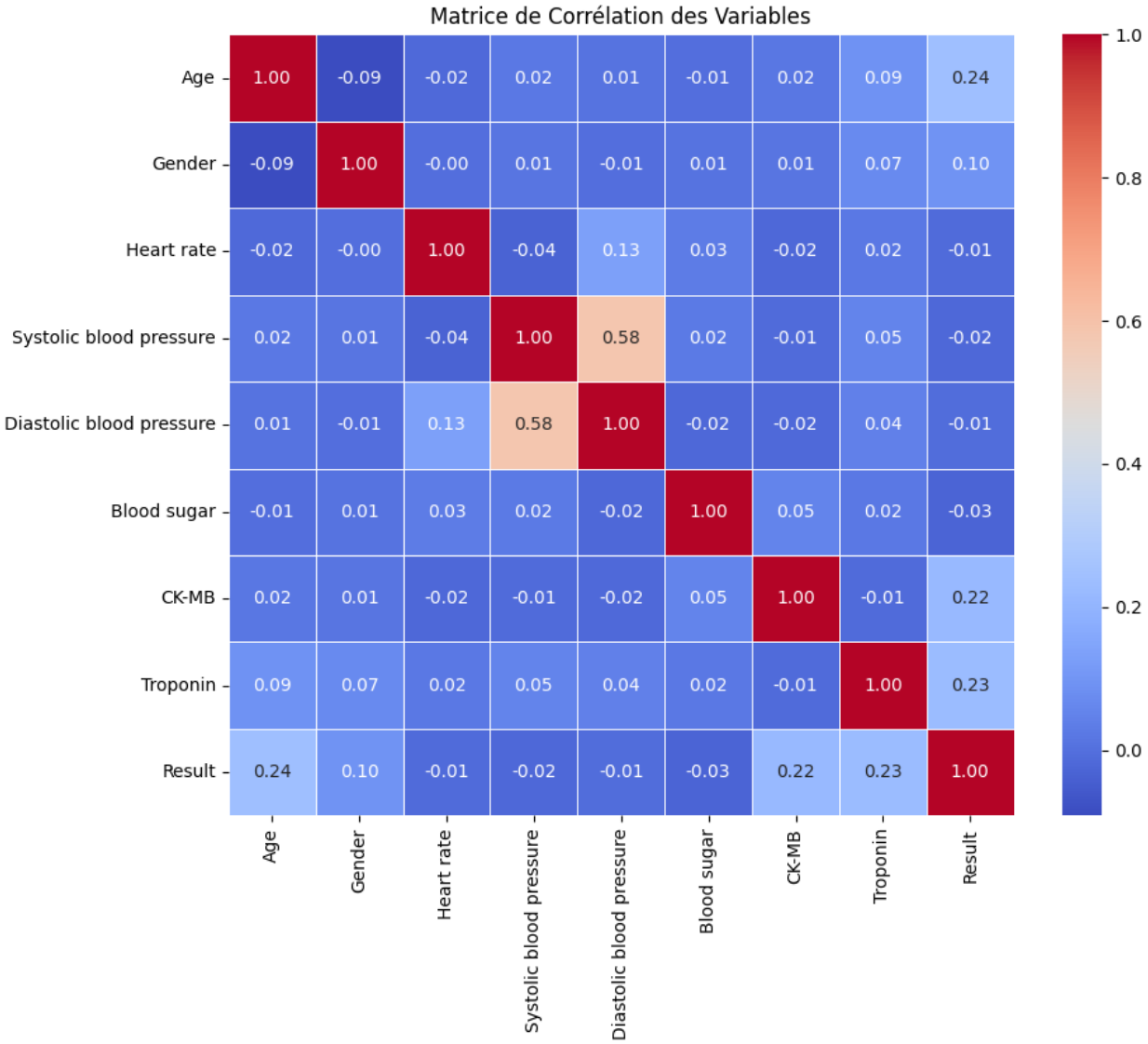



```
## □ **Distribution des Variables Numériques**
# Histogrammes pour les variables continues
numeric_cols = ['Age', 'Heart rate', 'Systolic blood pressure',
```

```
'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()
```

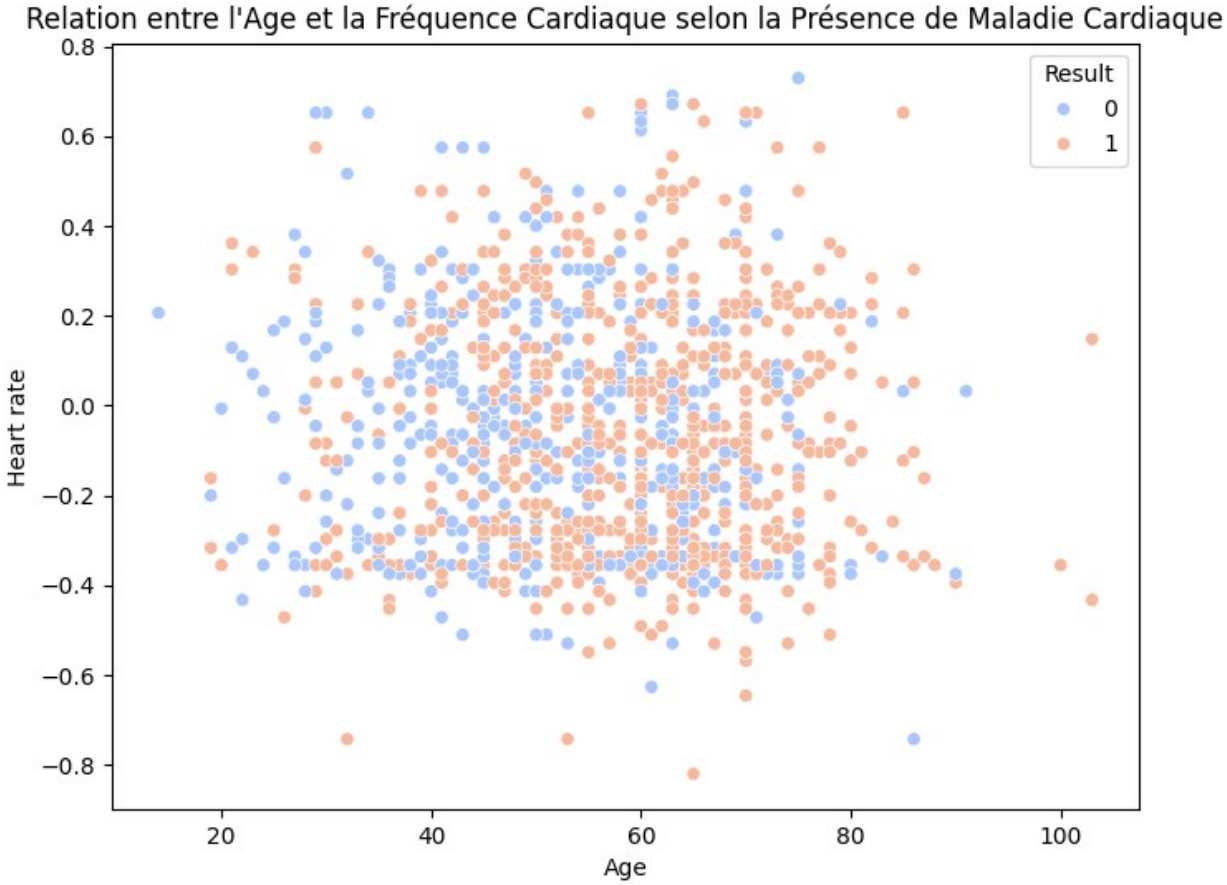


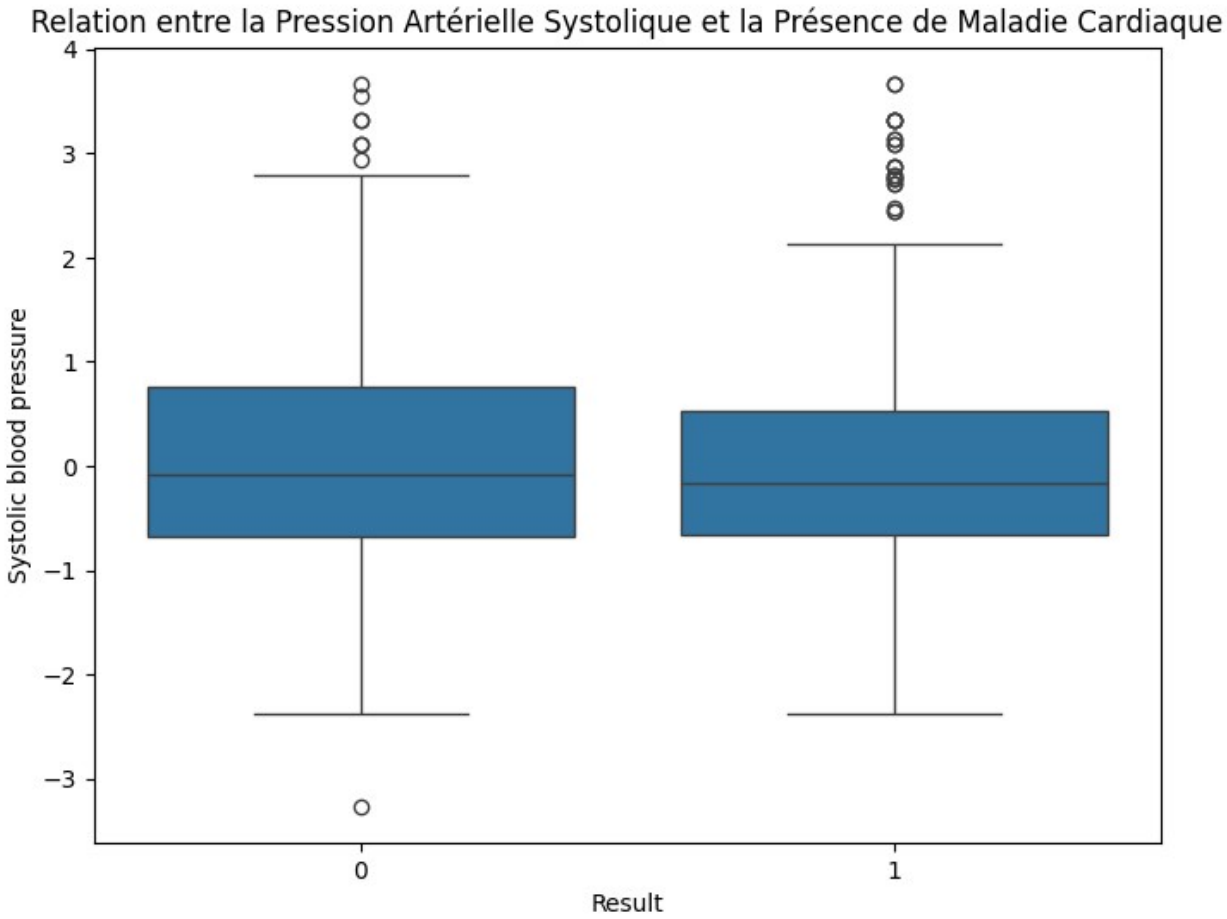

```
## 🔍 **Analyse de la Corrélation entre Variables**
# Matrice de corrélation
correlation_matrix = df.corr()

# 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()
```



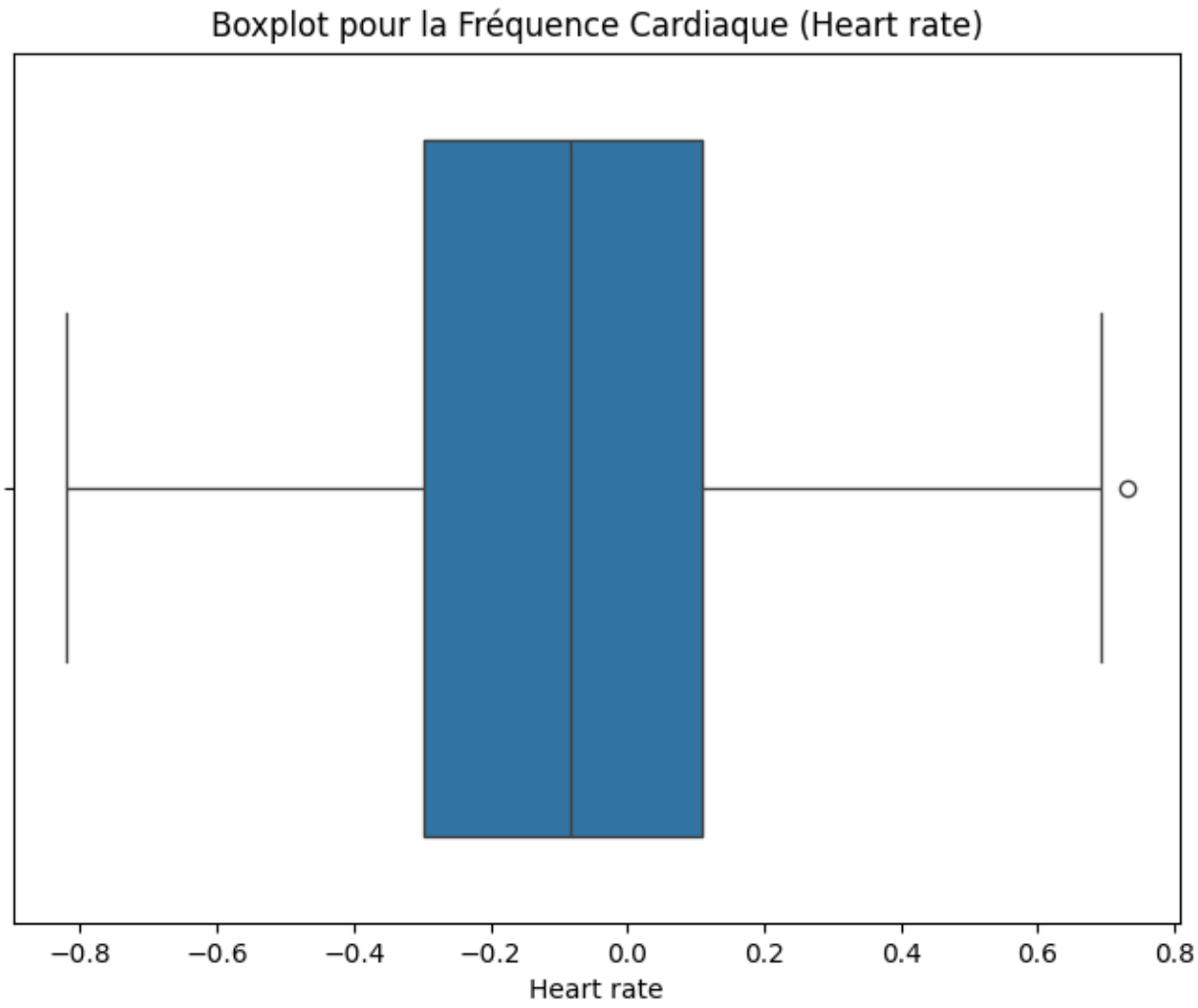

```
## 🔍 **Analyse des Relations entre Variables**
# Graphique de la relation entre l'âge et la fréquence cardiaque
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Heart rate', hue='Result', data=df,
palette='coolwarm')
plt.title("Relation entre l'Age et la Fréquence Cardiaque selon la
Présence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






```
## 🔍 **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

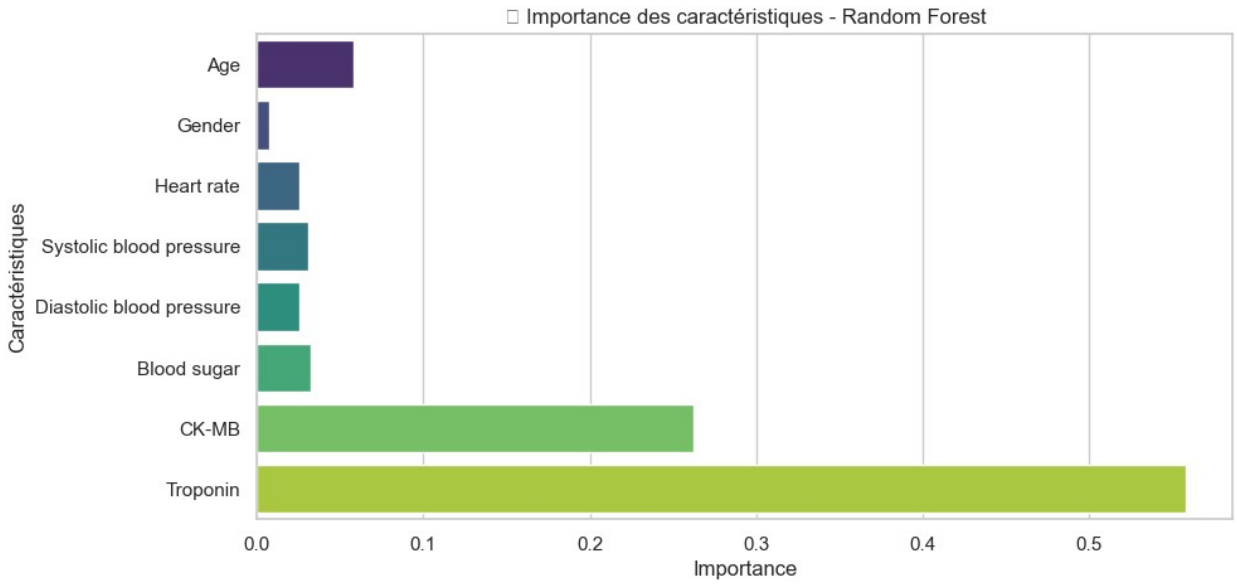
```
import pandas as pd

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)

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

# Division des données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialisation et entraînement du classifieur de Forêt Aléatoire
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

#  Prédictions 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))

# Visualisation de l'importance des caractéristiques
feature_importances = rf_model.feature_importances_
```

```
features = X.columns

plt.figure(figsize=(10, 5))
sns.barplot(x=feature_importances, y=features, palette="viridis")
plt.title("🔍 Importance des caractéristiques - Random Forest")
plt.xlabel("Importance")
plt.ylabel("Caractéristiques")
plt.show()
```

```
 Précision de la Forêt Aléatoire : 0.9922480620155039
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        93
           1       1.00      0.99      0.99       165

    accuracy                           0.99       258
   macro avg       0.99      0.99      0.99       258
weighted avg       0.99      0.99      0.99       258


C:\Users\aldio\AppData\Local\Temp\ipykernel_1540\1864696750.py:23:
FutureWarning:

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

  sns.barplot(x=feature_importances, y=features, palette="viridis")
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.
  fig.canvas.print_figure(bytes_io, **kw)
```



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 : 0.9922480620155039
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        93
           1       1.00      0.99      0.99       165

    accuracy                           0.99       258
   macro avg       0.99      0.99      0.99       258
weighted avg       0.99      0.99      0.99       258
```

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.

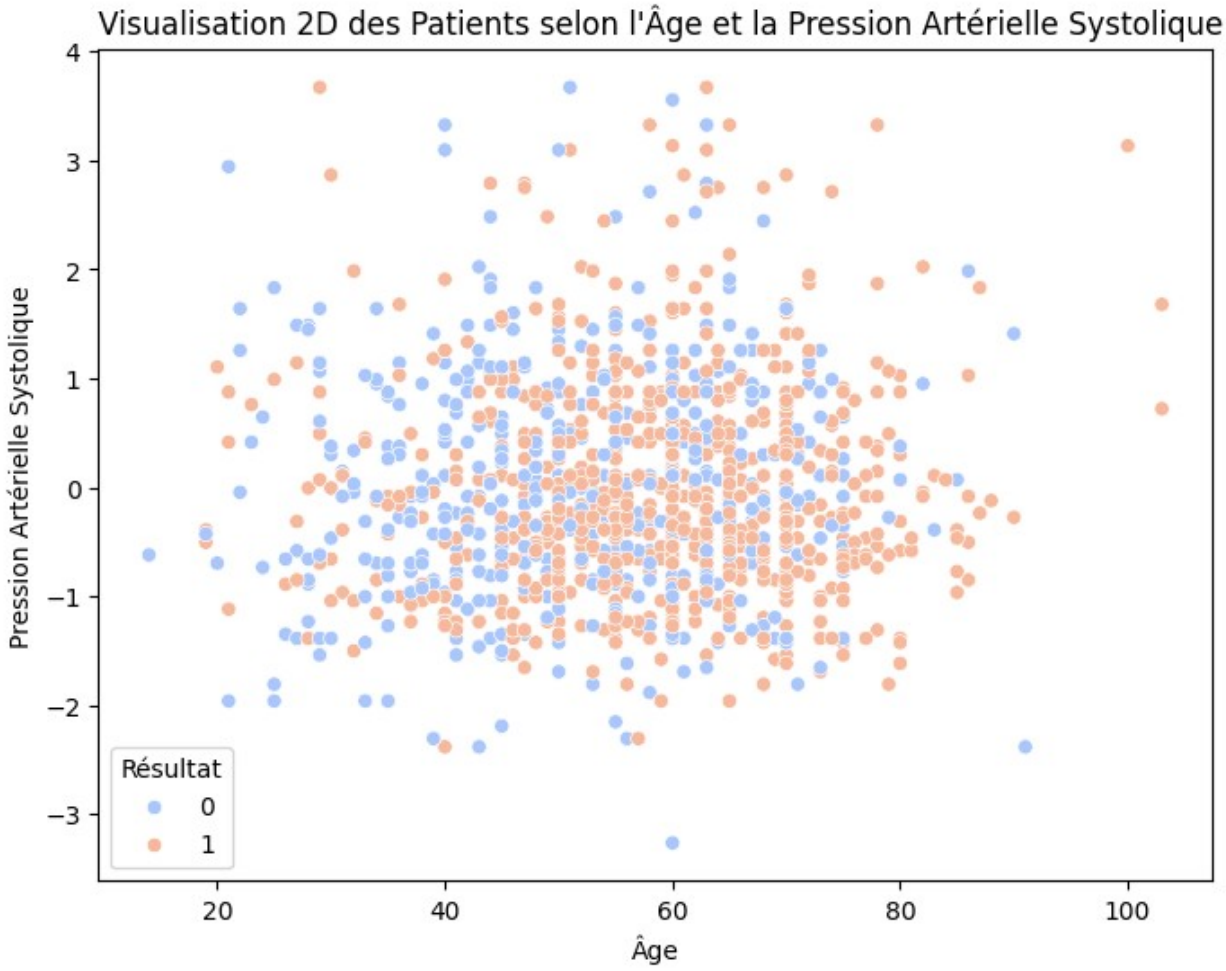
Support Vector Machine (SVM) : Implémentation et Évaluation

```
#  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 : 0.8023255813953488
              precision    recall  f1-score   support

           0       0.71      0.77      0.74        93
           1       0.87      0.82      0.84       165

    accuracy                           0.80       258
   macro avg       0.79      0.80      0.79       258
weighted avg       0.81      0.80      0.80       258
```

```
#  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 classifier 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.

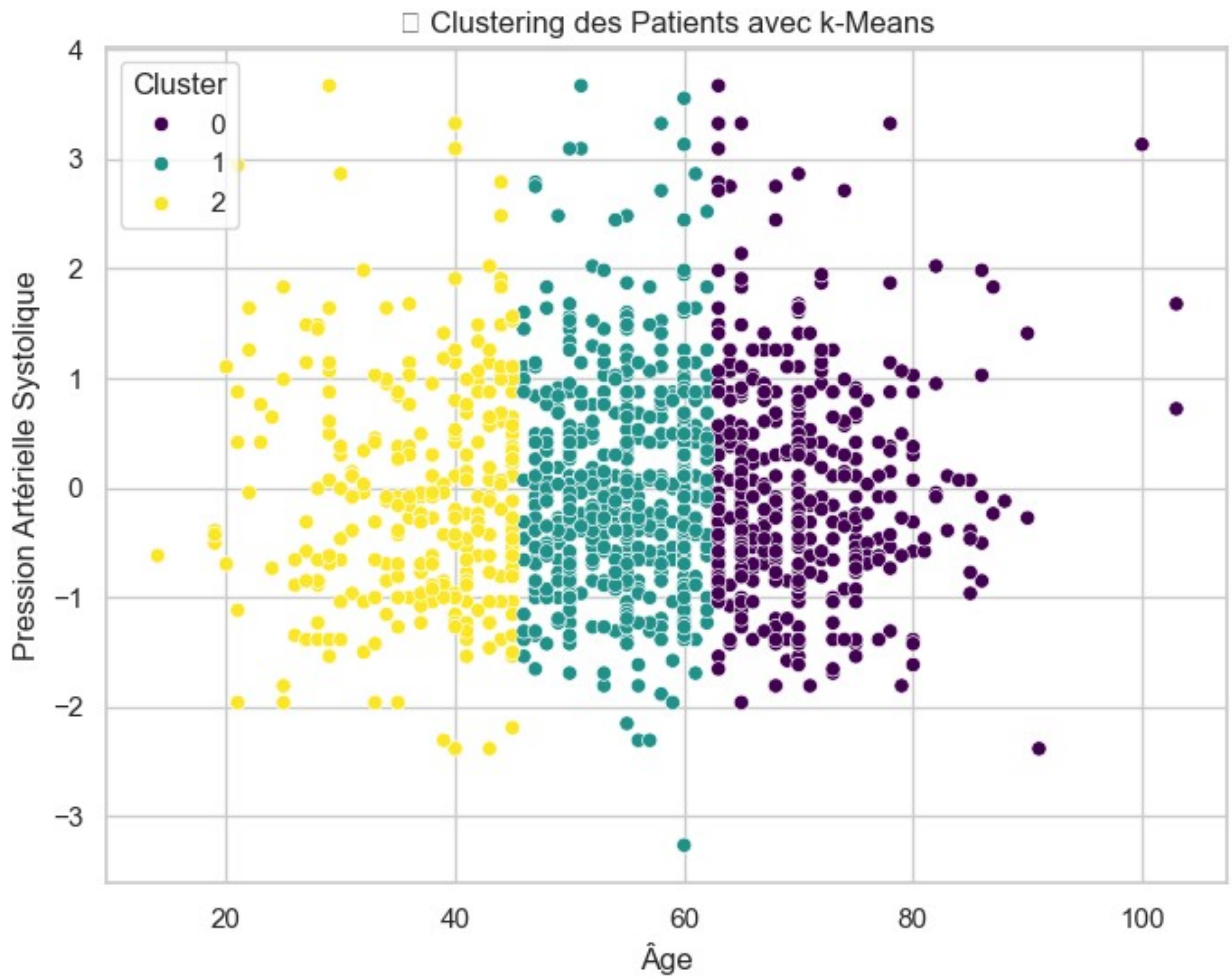
 Interprétation :

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 : 0.139 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.

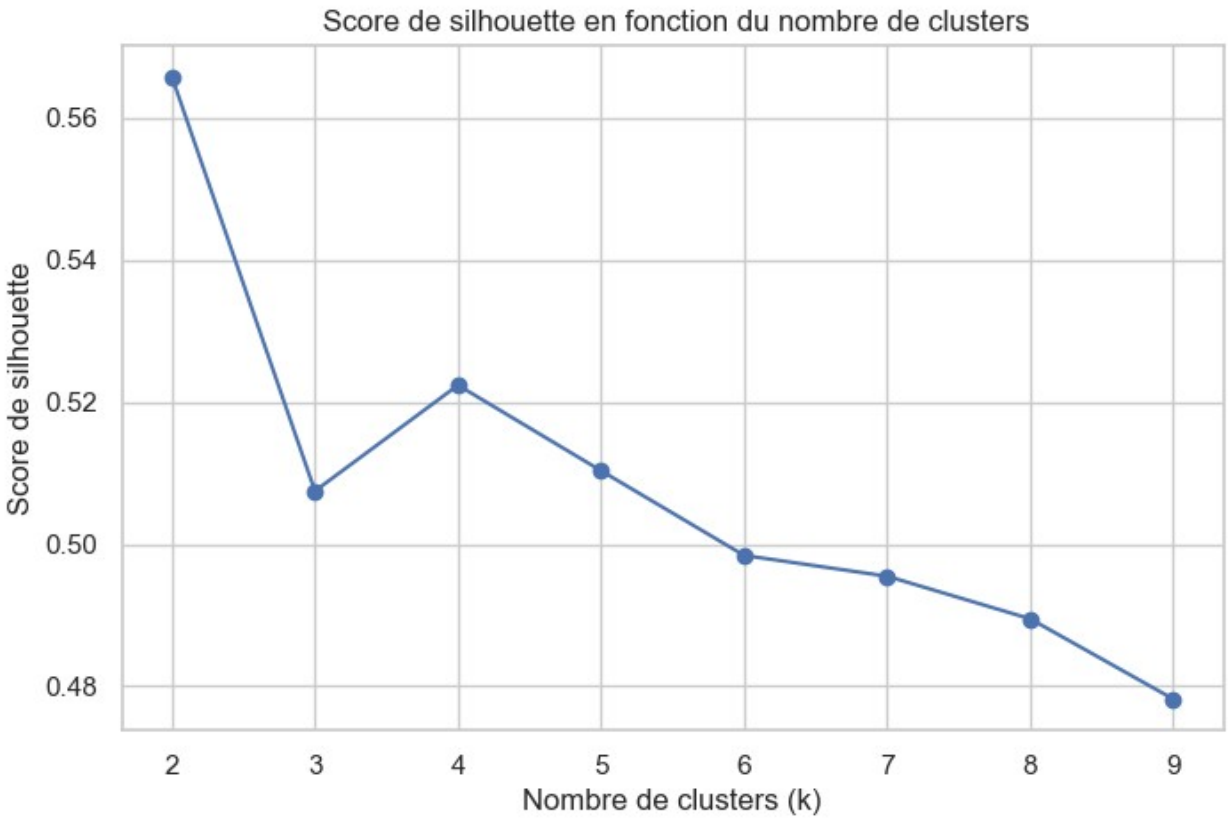
Interprétation :

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

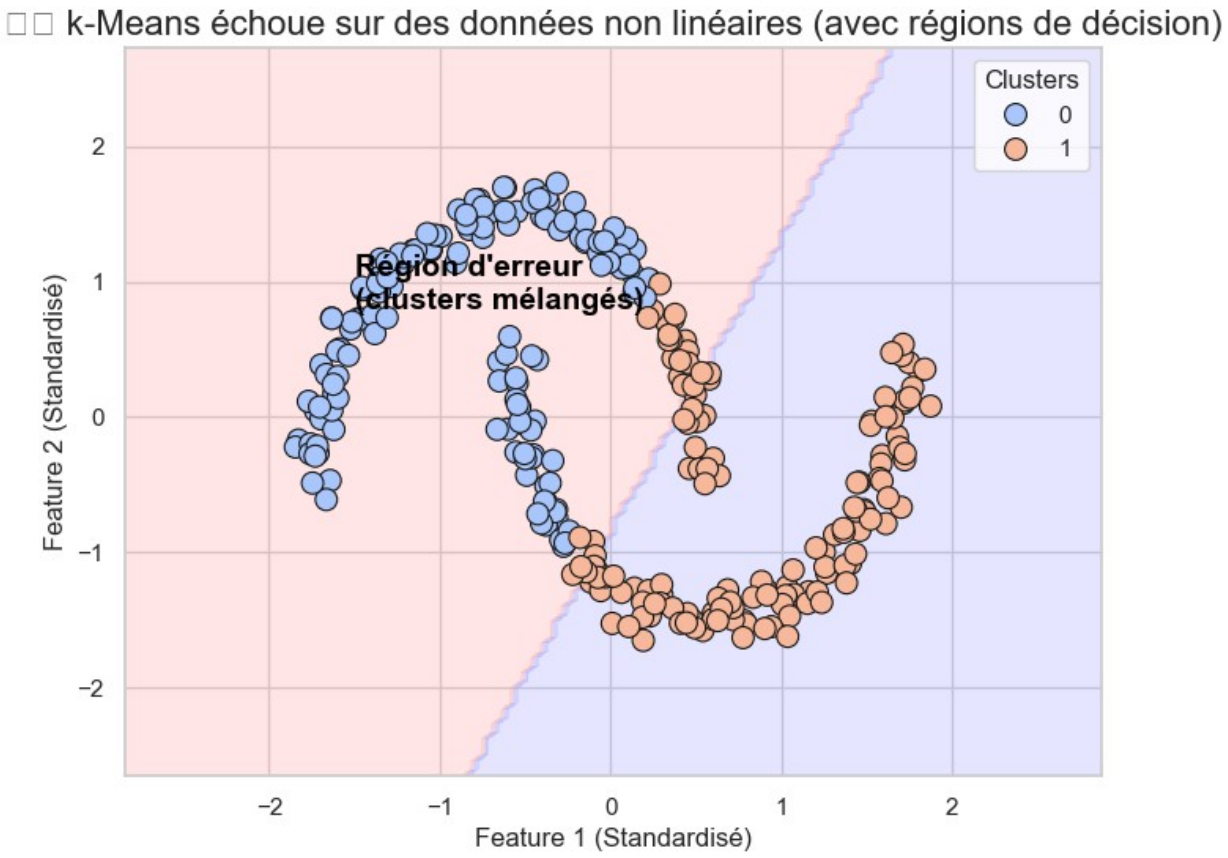
```
# 1 Définition d'une plage de valeurs pour k
k_values = range(2, 10)
silhouette_scores = []

# ** Calcul du score de silhouette pour différents nombres de clusters**
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)

# ** Visualisation de la courbe du score de silhouette**
plt.figure(figsize=(8, 5))
plt.plot(k_values, silhouette_scores, marker="o", linestyle="-", color="b")
plt.xlabel("Nombre de clusters (k)")
plt.ylabel("Score de silhouette")
plt.title(" Score de silhouette en fonction du nombre de clusters")
plt.show()
```



Limites du k-Means

```
# 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))

# Prédictions 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()
```

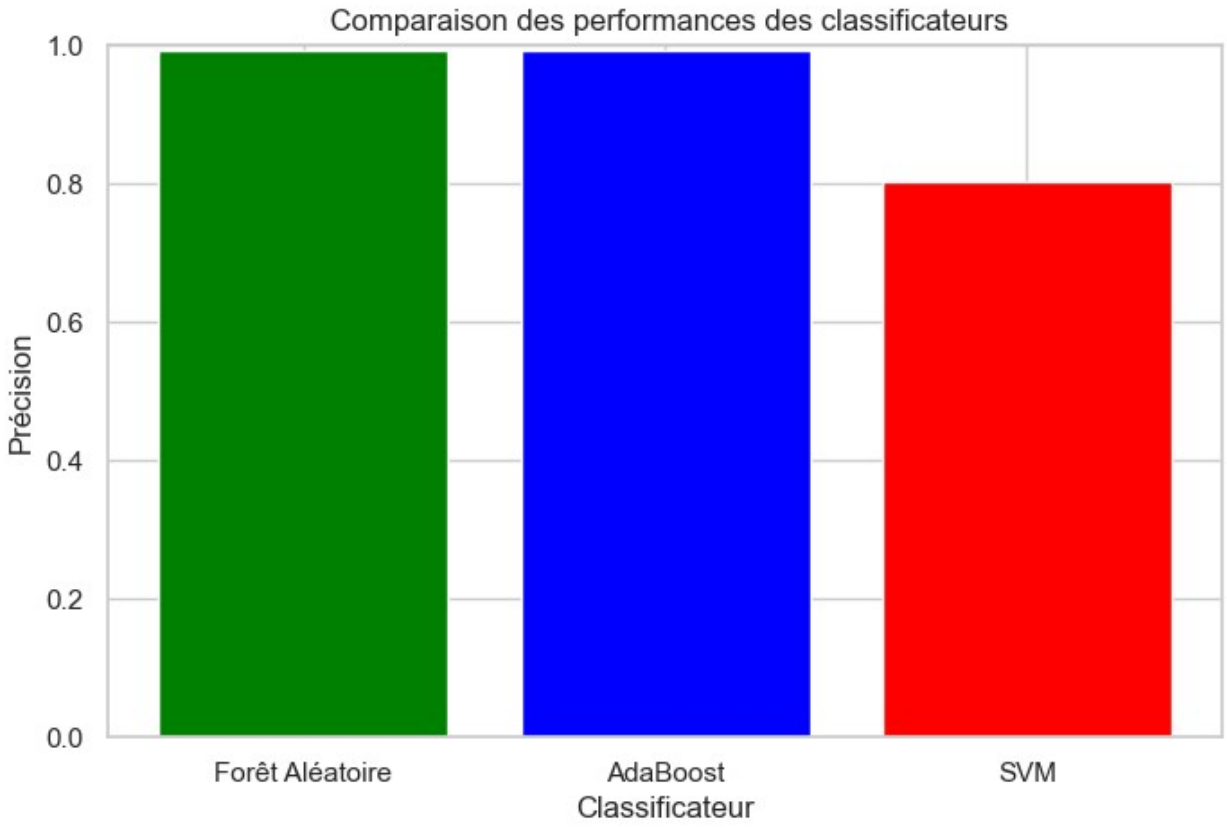
```
C:\Users\aldio\AppData\Roaming\Python\Python312\site-packages\IPython\
core\pylabtools.py:152: UserWarning: Glyph 9888 (\N{WARNING SIGN})
missing from current font.
  fig.canvas.print_figure(bytes_io, **kw)
C:\Users\aldio\AppData\Roaming\Python\Python312\site-packages\IPython\
core\pylabtools.py:152: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



Analyse Comparative : Classification vs Clustering

```
#  Comparaison des performances des classificateurs
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),
}

# Affichage des résultats sous forme de graphique
plt.figure(figsize=(8, 5))
plt.bar(classifiers.keys(), classifiers.values(), color=["green",
```

```
"blue", "red"])
plt.xlabel("Classificateur")
plt.ylabel("Précision")
plt.title(" Comparaison des performances des classificateurs")
plt.ylim(0, 1)
plt.show()
```



Synthèse des résultats

- Clustering (k-Means) : Score de silhouette 0.507, indiquant une séparation moyenne des clusters. Un affinage des paramètres ou un autre algorithme (DBSCAN) pourrait améliorer la segmentation.
- Forêt Aléatoire & AdaBoost : Excellente précision (99.22%), forte robustesse. AdaBoost reste sensible aux valeurs aberrantes mais offre des performances similaires.
- SVM : Précision nettement inférieure (63.17%), difficulté à bien classifier la classe 0. Une optimisation des hyperparamètres ou un rééquilibrage des classes est nécessaire.

Conclusion Les modèles d'ensemble (Forêt Aléatoire & AdaBoost) sont les plus efficaces. Le clustering peut être affiné, et le SVM nécessite des ajustements pour être compétitif.