Predict survival of patients with heart failure

Hamdi Imen Zarrami Hana
MP1 IDIRIA

About Dataset

Cet ensemble de données contient les dossiers médicaux de 299 patients souffrant d'insuffisance cardiaque, collectés au cours de leur période de suivi, où chaque profil de patient présente 13 caractéristiques cliniques.

Caractéristiques de l'ensemble de données : multivarié

Domaine : Santé et médecine
Type de caractéristique : entier, réel
Cas : 299
Caractéristiques : 12

Dataset Information

L'apprentissage automatique peut prédire la survie des patients souffrant d'insuffisance cardiaque à partir de la créatinine sérique et de la fraction d'éjection uniquement

Voici la liste organisée des caractéristiques cliniques :

1. Âge
   - **Explication :** Âge du patient
   - **Mesure** : Années
2. Anémie
   - **Explication :** Diminution des globules rouges ou de l'hémoglobine
   - **Mesure** : booléenne
3. Créatinine phosphokinase (CPK)
   - **Explication :** Niveau de l'enzyme CPK dans le sang
   - ***Mesure :*** mcg/L
4. Diabète
   - ***Explication :*** Si le patient est diabétique
   - ***Mesure :*** booléenne
5. La fraction d'éjection
   - ***Explication :*** Pourcentage de sang sortant du cœur à chaque contraction
   - ***Mesure :*** Pourcentage
6. Hypertension artérielle
   - ***Explication :*** Si le patient souffre d'hypertension
   - ***Mesure :*** booléenne
7. Plaquettes
   - ***Explication :*** Plaquettes dans le sang

- ***Mesure :*** kiloplaquettes/mL

8. Sexe
   - ***Explication :*** Femme ou homme
   - ***Mesure :*** binaire
9. Créatinine sérique
   - ***Explication :*** Niveau de créatinine sérique dans le sang
   - ***Mesure :*** mg/dL
10. sodium sérique
    - ***Explication :*** Niveau de sodium sérique dans le sang
    - ***Mesure :*** mEq/L
11. Fumeur
    - ***Explication :*** Si le patient fume ou non
    - ***Mesure :*** booléenne
12. Temps
    - ***Explication :*** Période de suivi
    - ***Mesure : j***ours
13. [Cible] Événement de mort
    - ***Explication :*** Si le patient est décédé pendant la période de suivi
    - ***Mesure :*** booléenne

Téléchargement et Affichage des Premières Lignes d'un Fichier CSV depuis Google Drive en Utilisant Pandas :

```
import pandas as pd

# Lien direct vers dataset dans drive
url = 'https://drive.google.com/uc?id=1qv9rpuygu1LpRmxI29tHVdJUH64-ggci'
# Télécharger et lire le fichier CSV
df = pd.read_csv(url)

# Afficher les premières lignes
print(df.head())
```

Ce code permet de lire un fichier CSV directement depuis un lien Google Drive et d'afficher les premières lignes du dataset.

Génération de Statistiques Descriptives pour un DataFrame en Utilisant Pandas :

```
df.describe()
```

La méthode df.describe() en pandas génère des statistiques descriptives pour les colonnes numériques d'un DataFrame. Voici une explication détaillée et un exemple de ce que produit **df.describe()**.

Fonctionnalité de df.describe()

Cette méthode calcule les statistiques suivantes pour chaque colonne numérique du DataFrame :
Count : Le nombre de valeurs non nulles.
Mean : La moyenne des valeurs.
Std : L'écart-type des valeurs.
Mir : La valeur minimale.
25% : Le premier quartile (Q1), c'est-à-dire la valeur à laquelle 25% des données sont inférieures.
50 : La médiane (Q2), c'est-à-dire la valeur à laquelle 50% des données sont inférieures.
75 : Le troisième quartile (Q3), c'est-à-dire la valeur à laquelle 75% des données sont inférieures.
Max : La valeur maximale.

Calcul du Nombre de Valeurs Manquantes dans Chaque Colonne d'un DataFrame en Utilisant Pandas :

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

La méthode **df.isnull().sum(** en pandas est utilisée pour identifier le nombre de valeurs manquantes (nulles) dans chaque colonne d'un DataFrame. Voici comment elle fonctionne et un exemple d'utilisation.

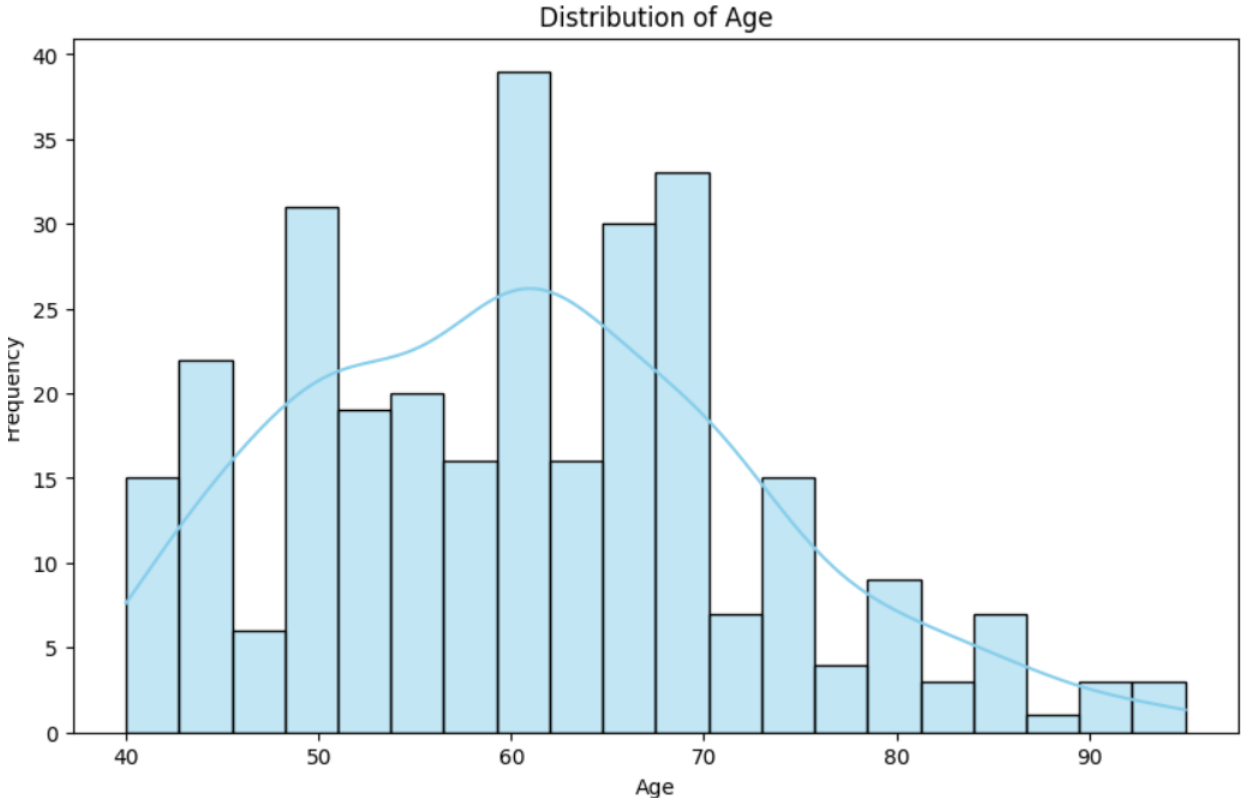
Fonctionnalité de df.isnull().sum()

df.isnull(: Cette méthode renvoie un DataFrame du même type que **df** avec des valeurs booléennes, où **Tru** indique une valeur manquante (nulle) et **Fals** indique une valeur présente.
df.isnull().sum(: Enchaîner **.sum()** à **df.isnull(** compte le nombre de **Tru** dans chaque colonne, ce qui donne le nombre de valeurs manquantes par colonne.

Visualisation de la Distribution de l'Âge avec un Histogramme et une Estimation de la Densité Kernel "EDK" en Utilisant Seaborn :

```
# Distribution of age
plt.figure(figsize=(10, 6))
sns.histplot(df['age'], bins=20, kde=True, color='skyblue')
plt.title('Distribution of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.show()
```

Ce code utilise **matplotlib** et **seaborn** pour visualiser la distribution de l'âge dans un DataFrame **df**.



Diagrammes en Barres des Caractéristiques Binaires en Utilisant Seaborn dans une Mise en Page 2x3 :

```
# Count plot of binary features
binary_features = ['anaemia', 'high_blood_pressure', 'diabetes', 'sex', 'smoking', 'DEATH_EVENT']
plt.figure(figsize=(12, 10))
for i, feature in enumerate(binary_features, 1):
    plt.subplot(2, 3, i)
    sns.countplot(data=df, x=feature, palette='Set2')
    plt.title(f'Count Plot of {feature}')
    plt.xlabel('')
    plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

Ce code crée des diagrammes en barres (count plots) pour des caractéristiques binaires dans un DataFrame **df** en utilisant **matplotli** et **seaborn**.

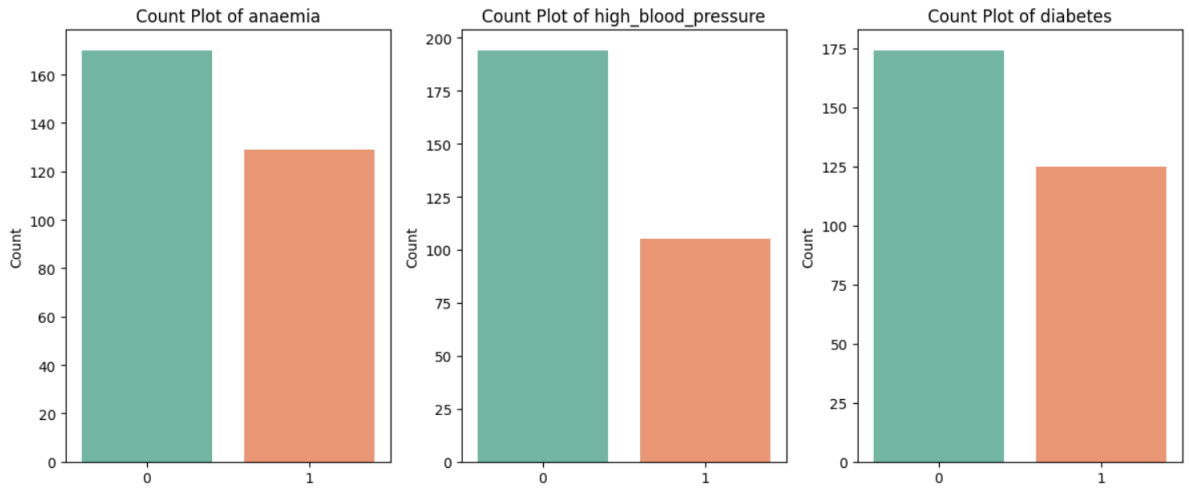
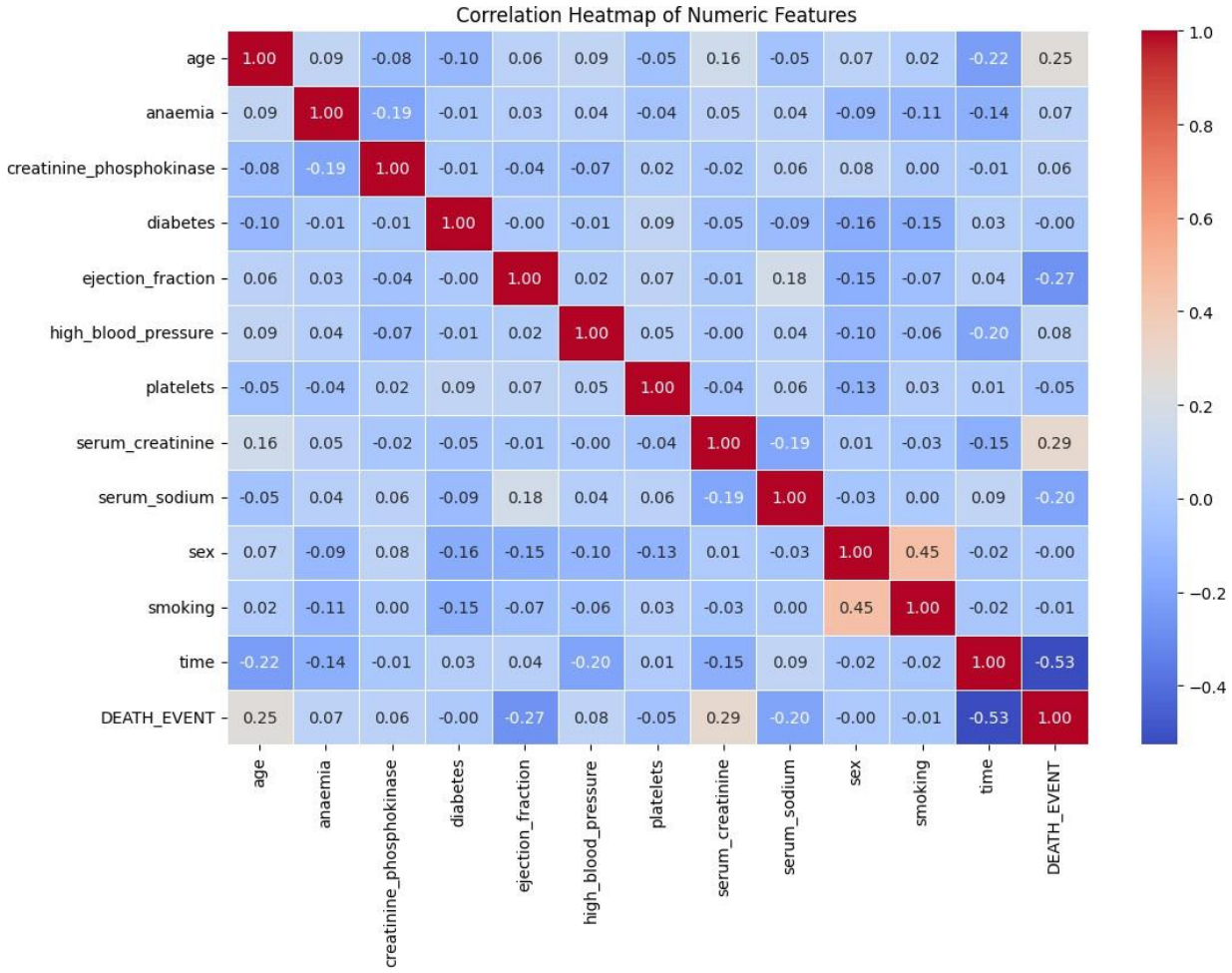


Diagramme de Chaleur de Corrélation des Caractéristiques Numériques avec Annotations en Utilisant Seaborn :

```
# Correlation heatmap of numeric features
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap of Numeric Features')
plt.show()
```

Ce code crée une heatmap (carte de chaleur) des corrélations entre les caractéristiques numériques d'un DataFrame **df** en utilisant **matplotlib** et **seaborn**.



Entraînement d'un Classificateur RandomForest sur un Jeu de Données Fractionné en Utilisant Scikit-Learn :

```
# Machine learning approach: Random Forest Classifier
# Splitting the data into features (X) and target variable (y)
X = df.drop(columns=['DEATH_EVENT'])
y = df['DEATH_EVENT']

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Model training
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

Ce code utilise un Random Forest Classifier pour prédire une variable cible `DEATH_EVENT` dans un DataFrame **df**.

Importation des bibliothèques nécessaires :

train_test_split : Fonction pour diviser le dataset en ensembles d'entraînement et de test.
RandomForestClassifie : Classe pour le modèle de classification par forêt aléatoire.

- **Séparer les données en caractéristiques (X) et variable cible (y) :**

  df.drop(columns=['DEATH_EVENT']) : Supprime la colonne **DEATH_EVE** du DataFrame **df** pour obtenir les caractéristiques ( ).
  df['DEATH_EVENT'] : Sélectionne la colonne **DEATH_EVE** comme variable cible (y).

- **Diviser les données en ensembles d'entraînement et de test :**

  train_test_split(X, y, test_size=0.2, : Divise les données en ensembles d'entraînement et de test avec 80% des données pour l'entraînement et 20% pour le test. Le **random_state=42** garantit que la division est reproductible.

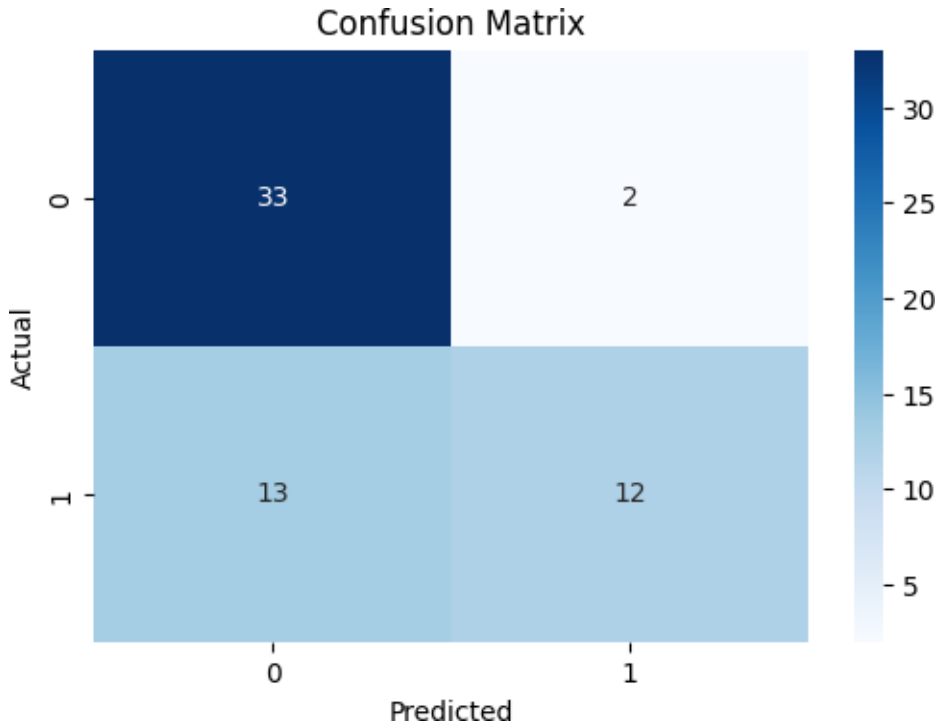
- **Entraînement du modèle :**

  RandomForestClassifier(random_state=42) : Crée une instance du classifieur par forêt aléatoire avec un **random_state** pour assurer la reproductibilité.
  rf_classifier.fit(X_train, y_train) : Entraîne le modèle sur les données d'entraînement.

Évaluation d'un Classificateur RandomForest : Précision, Rapport de Classification et Visualisation de la Matrice de Confusion :

```
# Model evaluation
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Classification Report:\n", classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

Pour évaluer le modèle de classification par forêt aléatoire, nous allons utiliser plusieurs mesures de performance : la précision (accuracy), le rapport de classification (classification report), et la matrice de confusion (confusion matrix).



SVM :

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
# Initialize SVM classifier
# Model training
svc_classifier = SVC(random_state=42)
svc_classifier.fit(X_train, y_train)  # Fit the model
```

Ce code montre l'utilisation de la bibliothèque `scikit-learn` pour entraîner un modèle de classification par machine à vecteurs de support (SVM).

- **Importation des bibliothèques nécessaires** : `SVC` pour le classificateur SVM et `accuracy_score` pour évaluer les performances du modèle.
- **Initialisation du classificateur SVM** : Création d'un objet SVM avec un état aléatoire fixé pour garantir la reproductibilité des résultats.
- **Entraînement du modèle** : Ajustement du classificateur SVM aux données d'entraînement `X_train` (caractéristiques) et `y_train` (étiquettes).

```
[ ]  # Predictions
     y_predSV = svc_classifier.predict(X_test)

     # Evaluating the model
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Model accuracy: {accuracy}")
```

Ce code montre comment faire des prédictions avec le modèle SVM entraîné et évaluer sa performance. Voici les étapes expliquées :

- **Prédictions** : Utilisation du classificateur SVM entraîné pour prédire les étiquettes des données de test `X_test`.
- **Évaluation du modèle** : Calcul de la précision du modèle en comparant les étiquettes prédites `y_predSV` avec les étiquettes réelles `y_test` à l'aide de la fonction `accuracy_score`.

- **Affichage de la précision** : Impression de la précision du modèle.

```
[ ]  # Classification report
     print("\nClassification Report:")
     print(classification_report(y_test, y_predSV))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.58      1.00      0.74        35
           1       0.00      0.00      0.00        25

    accuracy                           0.58        60
   macro avg       0.29      0.50      0.37        60
weighted avg       0.34      0.58      0.43        60
```

Ce code affiche un rapport de classification pour évaluer plus en détail les performances du modèle SVM.

KNN :

```
from sklearn.neighbors import KNeighborsClassifier
# Créer un modèle KNN
knn_model = KNeighborsClassifier(n_neighbors=3)  # Vous pouvez ajuster le nombre de voisins (n_neighbors) en fonction de votre problème
```

Ce code montre comment créer un modèle de classification par les k-plus proches voisins (k-NN) en utilisant la bibliothèque `scikit-learn` Voici les étapes

- **Importation de la classe KNeighborsClassifier** :
- **Création du modèle k-NN** : `n_neighbors=3` signifie que le modèle prendra en compte les 3 plus proches voisins pour déterminer la classe d'un échantillon donné. (augmenter `n_neighbors` peut rendre le modèle plus robuste mais moins sensible aux petites variations dans les données, tandis que diminuer `n_neighbors` peut rendre le modèle plus sensible aux petites variations mais potentiellement plus bruité.)

```
# Entraîner le modèle sur l'ensemble d'entraînement
knn_model.fit(X_train, y_train)
```

Cette ligne de code ajuste le modèle k-NN pour qu'il apprenne à partir des données d'entraînement, ce qui lui permet ensuite de faire des prédictions sur de nouvelles données.

```
# Faire des prédictions sur l'ensemble de test
y_predKNN = knn_model.predict(X_test)
# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy: {accuracy}")
```

Ce code montre comment faire des prédictions avec le modèle k-NN entraîné et évaluer sa performance.

```
# Classification report
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

Ce code génère et affiche un rapport de classification pour évaluer en détail les performances du modèle k-NN. Voici l'explication des étapes :

- **Génération du rapport de classification** : `classification_report(y_test, y_predKNN)` crée un rapport détaillé des métriques de performance pour chaque classe, en comparant les étiquettes réelles `y_test` et les étiquettes prédites `y_predKNN`.
- **Affichage du rapport de classification** : suivi du contenu du rapport de classification.

Rapport de classification

Le rapport de classification fournit plusieurs métriques importantes pour évaluer les performances du modèle :

- **Precision** : La précision pour chaque classe, définie comme le ratio des vrais positifs sur le nombre total de prédictions positives.
- **Recall** : Le rappel pour chaque classe, défini comme le ratio des vrais positifs sur le nombre total de vrais échantillons de cette classe.
- **F1-score** : La moyenne harmonique de la précision et du rappel, donnant une mesure globale de la performance.
- **Support** : Le nombre d'échantillons réels pour chaque classe.

Choisir le meuilleur Algorithme :

Ce code effectue une validation croisée pour évaluer plusieurs modèles de classification, puis sélectionne le meilleur modèle basé sur la précision moyenne.

- **Importation et initialisation** : Importer la fonction de validation croisée et initialiser les modèles.
- **Validation croisée** : Exécuter une validation croisée pour chaque modèle et imprimer les résultats.
    - La validation croisée avec 5 divisions (5-fold cross-validation) est effectuée pour chaque modèle en utilisant la fonction `cross_val_score`.
    - Les scores de précision pour chaque division sont calculés.
    - Pour chaque modèle, la précision moyenne (`scores.mean()`) et l'écart-type (`scores.std()`) sont imprimés.

- **Sélection du meilleur modèle** : Identifier et imprimer le modèle ayant la meilleure précision moyenne.
    - Le modèle avec la précision moyenne la plus élevée est sélectionné en utilisant la fonction `max` et une lambda fonction pour calculer la précision moyenne des scores de validation croisée.

```python
from sklearn.model_selection import cross_val_score
# Initialize models
models = {
    'KNeighborsClassifier': KNeighborsClassifier(),
    'RandomForestClassifier': RandomForestClassifier(),
    'SVC': SVC()
}

# Perform cross-validation and evaluate each model
for model_name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5)  # 5-fold cross-validation
    print(f"***{model_name}:==> Mean Accuracy: {scores.mean()},==> Std: {scores.std()}")

# Choose the best performing model
best_model_name = max(models, key=lambda k: scores.mean())
print(f"Best Model: {best_model_name}")
```

```
***KNeighborsClassifier:==> Mean Accuracy: 0.618813559322034,==> Std: 0.04448470040020285
***RandomForestClassifier:==> Mean Accuracy: 0.7055932203389831,==> Std: 0.14145762215443816
***SVC:==> Mean Accuracy: 0.6789265536723164,==> Std: 0.006472811110088906
Best Model: KNeighborsClassifier
```

Choisir arbre de decision :

ce code permet d'extraire, d'exporter et de visualiser un arbre de décision entraîné à partir d'un modèle de forêt aléatoire dans un notebook IPython.

- **Importation des bibliothèques**: Les bibliothèques nécessaires sont importées, y compris `export_graphviz`, `pydotplus`, et `Image`.
- **Extraction de l'arbre de décision**: Un arbre de décision est extrait d'un modèle de forêt aléatoire (`best_rf`).
- **Exportation en format DOT**: L'arbre de décision est converti en format DOT à l'aide de `export_graphviz`.
- **Création de l'objet Graphviz**: Un objet Graphviz est créé à partir des données DOT.
- **Affichage de l'arbre de décision**: L'arbre de décision est affiché sous forme d'image PNG dans le notebook IPython à l'aide de `Image(graph.create_png())`.

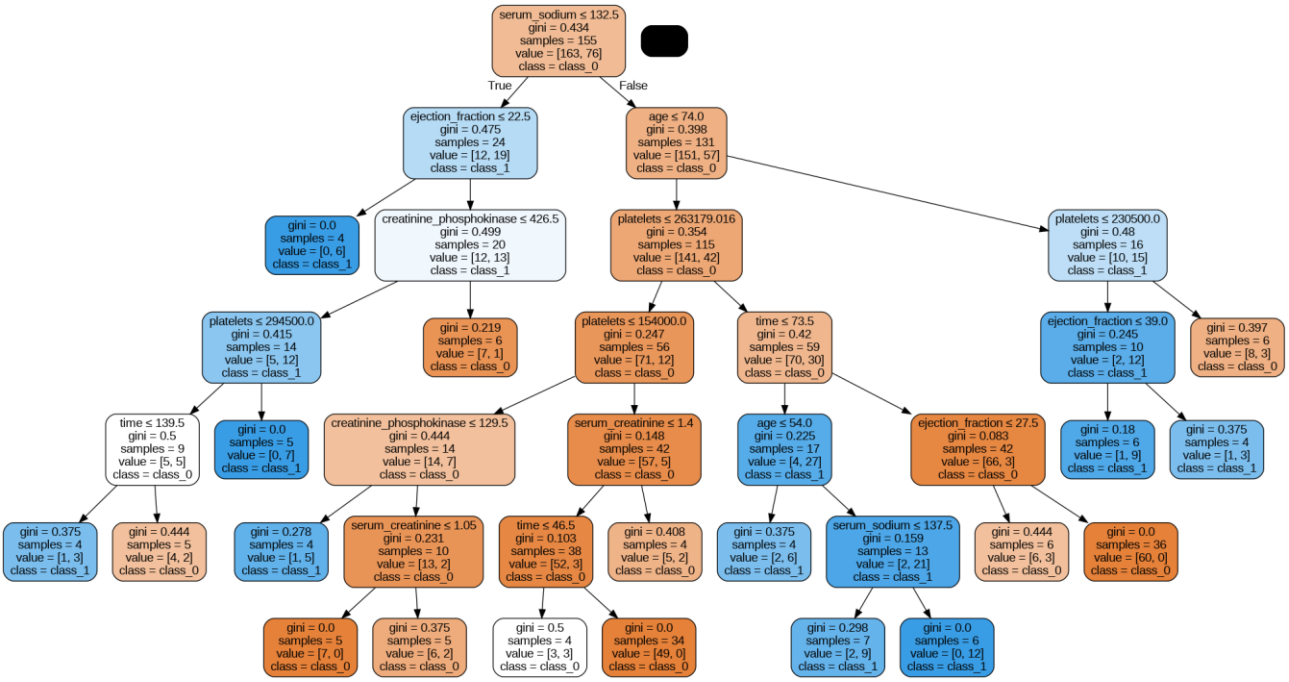
```python
from sklearn.tree import export_graphviz
import pydotplus
from IPython.display import Image

# Extract one of the decision trees (e.g., the first tree)
tree = best_rf.estimators_[0]

# Export the decision tree to a DOT file
dot_data = export_graphviz(tree, out_file=None,
                           feature_names=X_train.columns,  # Assuming X_train is a DataFrame
                           class_names=['class_0', 'class_1'],
                           filled=True, rounded=True,
                           special_characters=True)

# Create a Graphviz object from the DOT data
graph = pydotplus.graph_from_dot_data(dot_data)

# Display the decision tree using Graphviz
Image(graph.create_png())
```


serum_sodium ≤ 132.5
gini = 0.434
samples = 155
value = [163, 76]
class = class_0
True
False
ejection_fraction ≤ 22.5
gini = 0.475
samples = 24
value = [12, 19]
class = class_1
age ≤ 74.0
gini = 0.398
samples = 131
value = [151, 57]
class = class_0
gini = 0.0
samples = 4
value = [0, 6]
class = class_1
creatinine_phosphokinase ≤ 426.5
gini = 0.499
samples = 20
value = [12, 13]
class = class_1
platelets ≤ 263179.016
gini = 0.354
samples = 115
value = [141, 42]
class = class_0
platelets ≤ 230500.0
gini = 0.48
samples = 16
value = [10, 15]
class = class_1
platelets ≤ 294500.0
gini = 0.415
samples = 14
value = [5, 12]
class = class_1
gini = 0.219
samples = 6
value = [7, 1]
class = class_0
platelets ≤ 154000.0
gini = 0.247
samples = 56
value = [71, 12]
class = class_0
time ≤ 73.5
gini = 0.42
samples = 59
value = [70, 30]
class = class_0
ejection_fraction ≤ 39.0
gini = 0.245
samples = 10
value = [2, 12]
class = class_1
gini = 0.397
samples = 6
value = [8, 3]
class = class_0
time ≤ 139.5
gini = 0.5
samples = 9
value = [5, 5]
class = class_0
gini = 0.0
samples = 5
value = [0, 7]
class = class_1
creatinine_phosphokinase ≤ 129.5
gini = 0.444
samples = 14
value = [14, 7]
class = class_0
serum_creatinine ≤ 1.4
gini = 0.148
samples = 42
value = [57, 5]
class = class_0
age ≤ 54.0
gini = 0.225
samples = 17
value = [4, 27]
class = class_1
ejection_fraction ≤ 27.5
gini = 0.083
samples = 42
value = [66, 3]
class = class_0
gini = 0.18
samples = 6
value = [1, 9]
class = class_1
gini = 0.375
samples = 4
value = [1, 3]
class = class_1
gini = 0.375
samples = 4
value = [1, 3]
class = class_1
gini = 0.444
samples = 5
value = [4, 2]
class = class_0
gini = 0.278
samples = 4
value = [1, 5]
class = class_1
serum_creatinine ≤ 1.05
gini = 0.231
samples = 10
value = [13, 2]
class = class_0
time ≤ 46.5
gini = 0.103
samples = 38
value = [52, 3]
class = class_0
gini = 0.408
samples = 4
value = [5, 2]
class = class_0
gini = 0.375
samples = 4
value = [2, 6]
class = class_1
serum_sodium ≤ 137.5
gini = 0.159
samples = 13
value = [2, 21]
class = class_1
gini = 0.444
samples = 6
value = [6, 3]
class = class_0
gini = 0.0
samples = 36
value = [60, 0]
class = class_0
gini = 0.0
samples = 5
value = [7, 0]
class = class_0
gini = 0.375
samples = 5
value = [6, 2]
class = class_0
gini = 0.5
samples = 4
value = [3, 3]
class = class_0
gini = 0.0
samples = 34
value = [49, 0]
class = class_0
gini = 0.298
samples = 7
value = [2, 9]
class = class_1
gini = 0.0
samples = 6
value = [0, 12]
class = class_1