Start coding or generate with AI.

```
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, LabelEncoder,OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.compose import ColumnTransformer
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc
from sklearn.decomposition import PCA
```

Load the dataset

```
# Load the Titanic dataset

df = pd.read_csv("/content/Titanic Dataset.csv")
```

Double-click (or enter) to edit

Preprocessing

```
df.drop(['name', 'ticket', 'cabin'], axis=1, inplace=True)
df['age'].fillna(df['age'].median(), inplace=True)
df['embarked'].fillna(df['embarked'].mode()[0], inplace=True)
df['fare'].fillna(df['fare'].median(), inplace=True)
```

```
# Ensure no missing values remain
print("Missing values in each column after filling:")
print(df.isna().sum())
```

```
# Define features and target
X = df.drop('survived', axis=1)
y = df['survived']
```

```
X = df.drop('survived', axis=1)
y = df['survived']
```

```
Missing values in each column after filling:
pclass          0
survived        0
sex             0
age             0
sibsp           0
parch           0
fare            0
embarked        0
boat          823
body         1188
home.dest     564
dtype: int64
```

```python
numerical_features = ['age', 'fare', 'sibsp', 'parch']
categorical_features = ['sex', 'embarked', 'pclass']

numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(drop='first')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)


X_preprocessed = preprocessor.fit_transform(X)


X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.3, random_state=42)
```

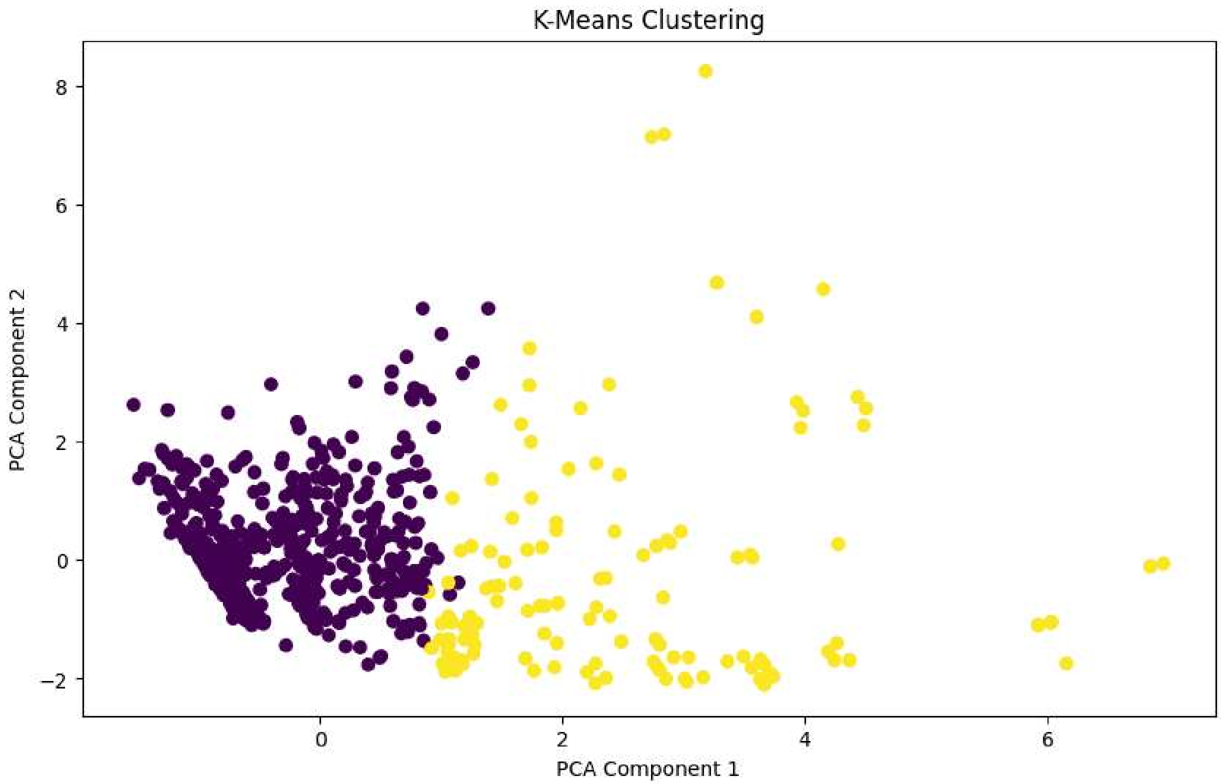
```python
# KNN
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
```

```
KNN Accuracy: 0.7735368956743003
```

```python
# K-Means
kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_train)
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X_train)
plt.figure(figsize=(10, 6))
plt.scatter(pca_result[:, 0], pca_result[:, 1], c=clusters)
plt.title("K-Means Clustering")
plt.xlabel("PCA Component 1")
```

```
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```

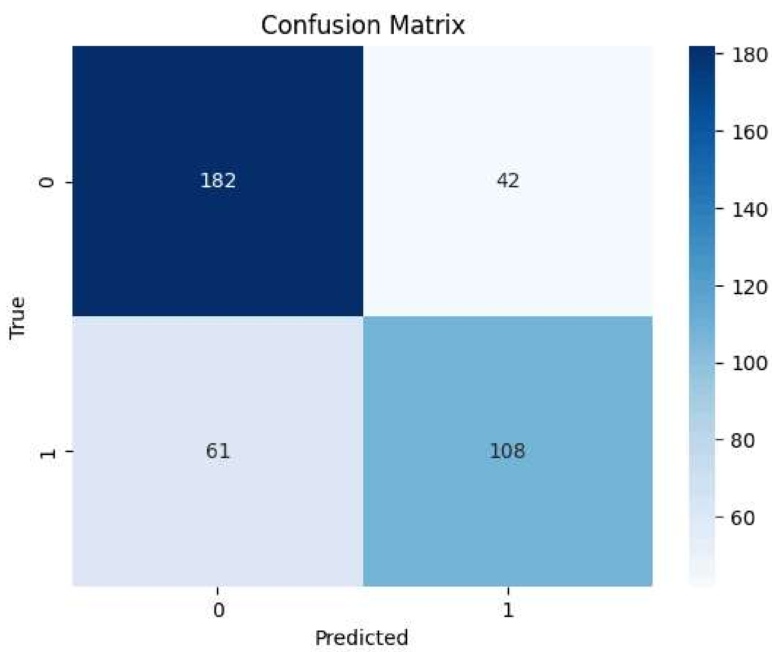
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the val
  warnings.warn(
```




```
# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
```

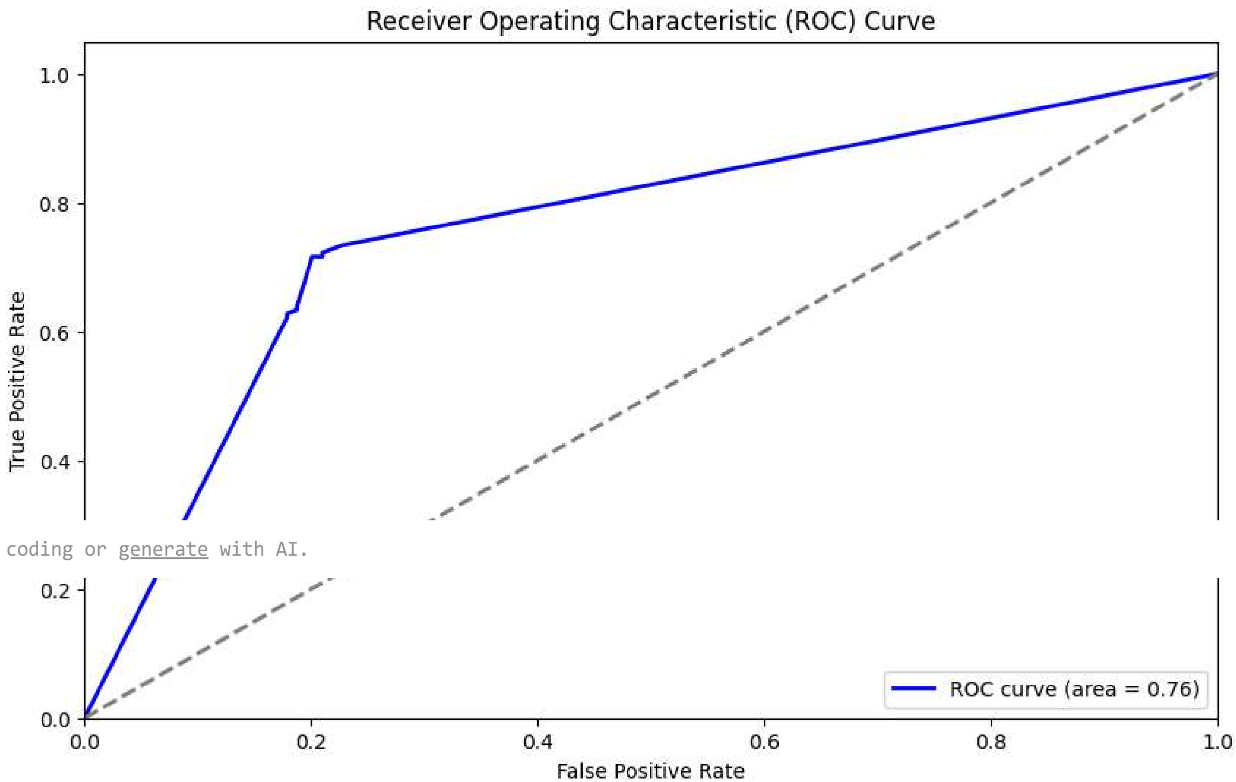
```
Decision Tree Accuracy: 0.7379134860050891
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```



```
# Classification Report
print("Classification Report:\n", classification_report(y_test, y_pred_dt))

# ROC Curve
y_pred_proba = dt.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='gray', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()



min_length = min(len(feature_names), len(dt.feature_importances_))
feature_names = feature_names[:min_length]
importances = dt.feature_importances_[:min_length]

feature_importances = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importances)
plt.title('Feature Importance')
plt.show()
```

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

           0       0.75      0.81      0.78       224
           1       0.72      0.64      0.68       169

    accuracy                           0.74       393
   macro avg       0.73      0.73      0.73       393
weighted avg       0.74      0.74      0.74       393
```



Start coding or generate with AI.