```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

```
  # Load your dataset
  # Replace 'your_dataset.csv' with your actual dataset file
  df = pd.read_csv('/content/KNN_Customer Churn_Dataset.csv')
```

```
# Drop unnecessary columns
df = df.drop(['customerID', 'testIndicator'], axis=1)
```

```
# Encode the target variable 'Churn' into numerical values
label_encoder = LabelEncoder()
df['Churn'] = label_encoder.fit_transform(df['Churn'])
```

```
# Identify and one-hot encode categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
df = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
```

```
# Separate features (X) and target variable (y)
X = df.drop('Churn', axis=1)
y = df['Churn']
```

```
 # Handle missing values in features
imputer = SimpleImputer(strategy='mean')  # You can change the strategy as needed
X = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

```
# Split the dataset 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)
```

```
# Initialize KNN classifier
knn_classifier = KNeighborsClassifier(n_neighbors=3)
```

```
# Train the model
knn_classifier.fit(X_train, y_train)
```

```
▾       KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

```
# Fit the KNeighborsClassifier model to the preprocessed data
knn_classifier = neighbors.KNeighborsClassifier(n_neighbors=5)
knn_classifier.fit(X, y)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
  # Make predictions on the test set
  y_pred = knn_classifier.predict(X_test)
```

```
  # Evaluate the model
  accuracy = accuracy_score(y_test, y_pred)
```

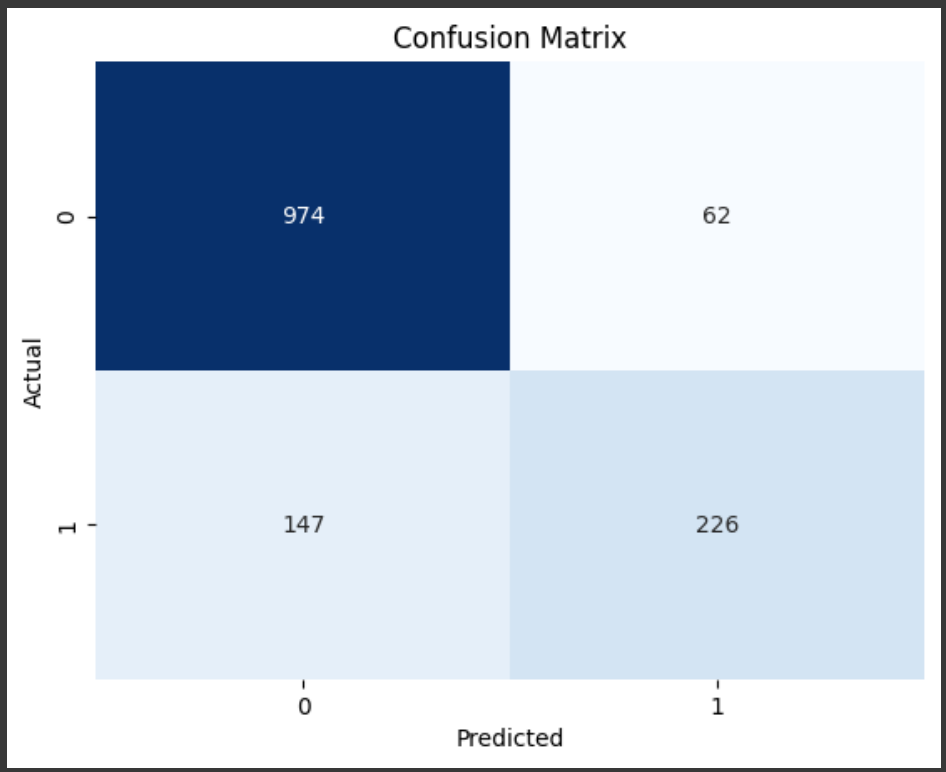
```
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

```
Accuracy: 0.85
```

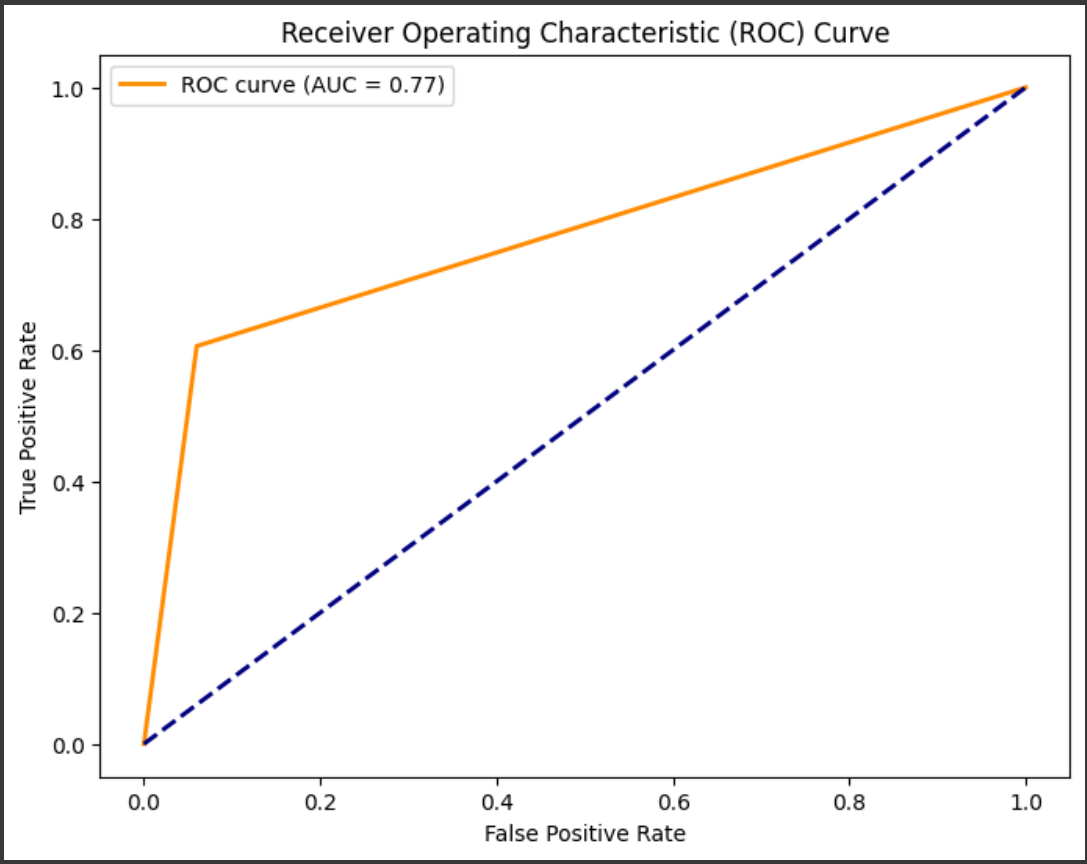
```
# Display classification report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.90      1036
           1       0.78      0.61      0.68       373

    accuracy                           0.85      1409
   macro avg       0.83      0.77      0.79      1409
weighted avg       0.85      0.85      0.85      1409
```

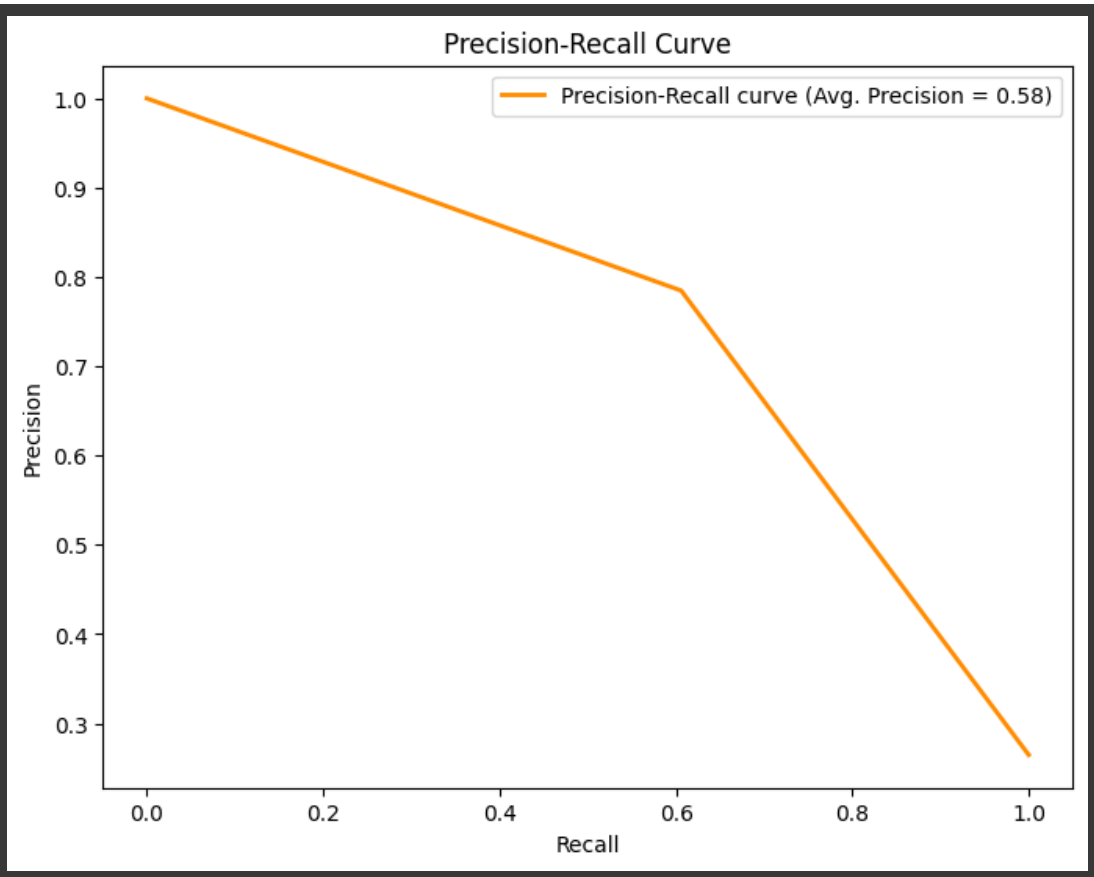
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



```python
from sklearn.metrics import roc_curve, auc

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```




```python
from sklearn.metrics import precision_recall_curve, average_precision_score

# Calculate Precision-Recall curve
precision, recall, _ = precision_recall_curve(y_test, y_pred)
average_precision = average_precision_score(y_test, y_pred)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='darkorange', lw=2, label=f'Precision-Recall curve (Avg. Precision =
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```


Precision-Recall Curve
Precision-Recall curve (Avg. Precision = 0.58)
Precision
Recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.0
0.2
0.4
0.6
0.8
1.0