```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression

# Load the dataset
# Replace 'path_to_csv' with the actual file path
data = pd.read_csv('creditcard.csv')
print(data)
```

```
             Time         V1         V2        V3        V4        V5        V6        V7  ...       V23       V24       V25       V26       V27       V28  Amount  Class
0             0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  ... -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62      0
1             0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  ...  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69      0
2             1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  ...  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66      0
3             1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  ... -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50      0
4             2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941  ... -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99      0
...           ...        ...        ...       ...       ...       ...       ...       ...  ...       ...       ...       ...       ...       ...       ...     ...    ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  ...  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77      0
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  ...  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79      0
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  ... -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88      0
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708 -0.686180  ... -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00      0
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  1.577006  ...  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00      0

[284807 rows x 31 columns]
```

```
# Check for missing values and basic dataset info
print(f"Dataset shape: {data.shape}")
```

```
[284807 rows x 31 columns]
Dataset shape: (284807, 31)
```

```
print(f"Missing values:\n{data.isnull().sum()}")
```

```
Missing values:
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Separate features and target

X = data.drop(columns=['Class']) # Replace 'Class' with your target column name if different

print(X)

y = data['Class']

print(y)

```
            Time         V1         V2        V3        V4        V5        V6        V7  ...       V22       V23       V24       V25       V26       V27       V28  Amount
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  ...  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  ... -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  ...  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  ...  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941  ...  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
...          ...        ...        ...       ...       ...       ...       ...       ...  ...       ...       ...       ...       ...       ...       ...       ...     ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  ...  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  ...  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  ...  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708 -0.686180  ...  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  1.577006  ...  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

[284807 rows x 30 columns]
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

Normalize the features

scaler = StandardScaler()

print(scaler)

X_scaled = scaler.fit_transform(X)

print(X_scaled)

```
            Time         V1         V2        V3        V4        V5        V6        V7  ...       V22       V23       V24       V25       V26       V27       V28  Amount
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  ...  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  ... -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  ...  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  ...  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941  ...  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
...          ...        ...        ...       ...       ...       ...       ...       ...  ...       ...       ...       ...       ...       ...       ...       ...     ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  ...  0.111864  1.014480 -0.509348  1.436807  0.250034  0.943651  0.823731    0.77
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  ...  0.924384  0.012463 -1.016226 -0.606624 -0.395255  0.068472 -0.053527   24.79
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  ...  0.578229 -0.037501  0.640134  0.265745 -0.087371  0.004455 -0.026561   67.88
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708 -0.686180  ...  0.800049 -0.163298  0.123205 -0.569159  0.546668  0.108821  0.104533   10.00
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  1.577006  ...  0.643078  0.376777  0.008797 -0.473649 -0.818267 -0.002415  0.013649  217.00

[284807 rows x 30 columns]
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42, stratify=y)

print(X_train,)

```
StandardScaler()
[[-1.99658302 -0.69424232 -0.04407492 ...  0.33089162 -0.06378115
   0.24496426]
 [-1.99658302  0.60849633  0.16117592 ... -0.02225568  0.04460752
  -0.34247454]
 [-1.99656197 -0.69350046 -0.81157783 ... -0.13713686 -0.18102083
   1.16068593]
 ...
 [ 1.6419735   0.98002374 -0.18243372 ...  0.01103672 -0.0804672
  -0.0818393 ]
 [ 1.6419735  -0.12275539  0.32125034 ...  0.26960398  0.31668678
  -0.31324853]
 [ 1.64205773 -0.27233093 -0.11489898 ... -0.00598394  0.04134999
   0.51435531]]
[[ 1.41309493  0.99390119 -0.45571564 ...  0.19235561 -0.09769602
  -0.3239634 ]
 [ 0.62464415  1.03903462 -0.02960101 ...  0.00624303 -0.20904363
  -0.34127512]
 [-1.12918609 -0.50641931  0.36528224 ...  0.09426583  0.56149664
   0.3468355 ]
 ...
 [-1.25321726  0.54616878  0.04366533 ...  0.07683742  0.07539328
  -0.17163632]
 [-1.48249593  0.65373478  0.18202927 ... -0.14824131 -0.0178351
  -0.3496711 ]
 [-1.3891251  -0.30536718  0.46935012 ...  0.9379349   0.62519565
  -0.32528277]]
```

Handle class imbalance using SMOTE

smote = SMOTE(random_state=42)

print(smote)

X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print(X_train_resampled,y_train_resampled)

```
1         0
2         0
3         0
4         0
         ..
454897    1
454898    1
454899    1
454900    1
454901    1
Name: Class, Length: 454902, dtype: int64
```

Choose a classification algorithm

Use Logistic Regression for faster training

You can switch to RandomForestClassifier if needed

classifier = LogisticRegression(max_iter=500, random_state=42)

print(classifier)

```
LogisticRegression(max_iter=500, random_state=42)
```

Train the model

print("Training the model...")

print(classifier.fit(X_train_resampled, y_train_resampled))

```
Training the model...
LogisticRegression(max_iter=500, random_state=42)
```

Make predictions on the test set

y_pred = classifier.predict(X_test)

print(y_pred)

```
[0 0 0 ... 0 0 0]
```

Evaluate the model's performance

print("\nConfusion Matrix:")

print(confusion_matrix(y_test, y_pred))

```
Confusion Matrix:
[[55399  1465]
 [    8    90]]
```

print("\nClassification Report:")

print(classification_report(y_test, y_pred))

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

           0       1.00      0.97      0.99     56864
           1       0.06      0.92      0.11        98

    accuracy                           0.97     56962
   macro avg       0.53      0.95      0.55     56962
weighted avg       1.00      0.97      0.99     56962
```

Calculate precision, recall, and F1-score

from sklearn.metrics import precision_score, recall_score, f1_score

precision = precision_score(y_test, y_pred)

recall = recall_score(y_test, y_pred)

f1 = f1_score(y_test, y_pred)

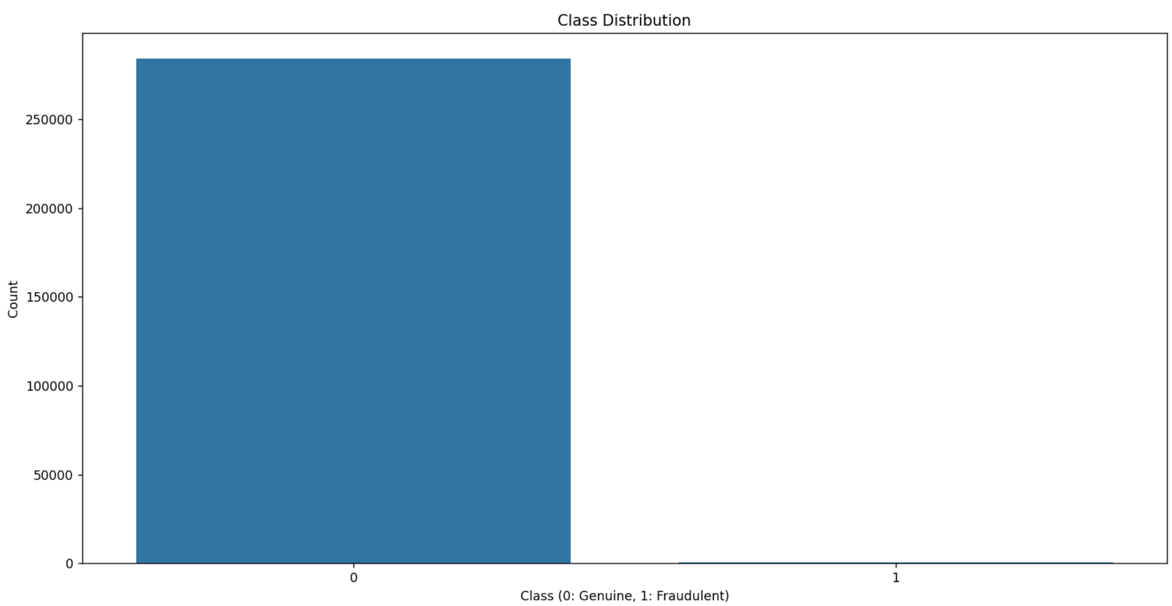
print(f"\nPrecision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

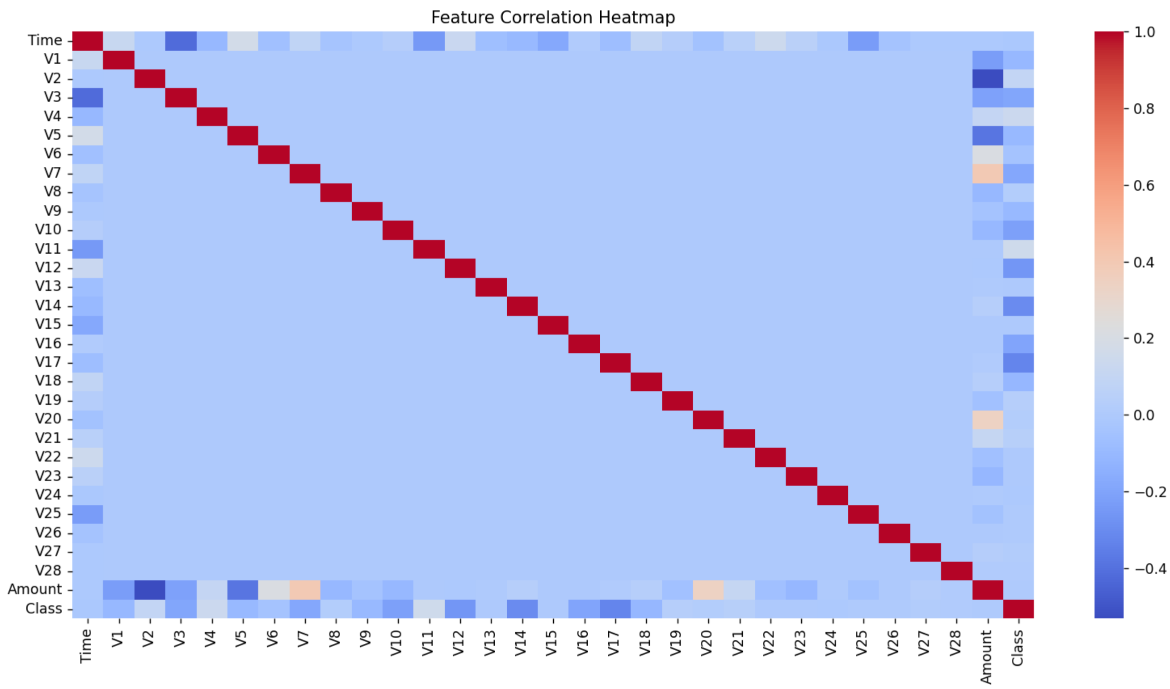
print(f"F1-Score: {f1:.2f}")

```
Precision: 0.06
Recall: 0.92
F1-Score: 0.11
```

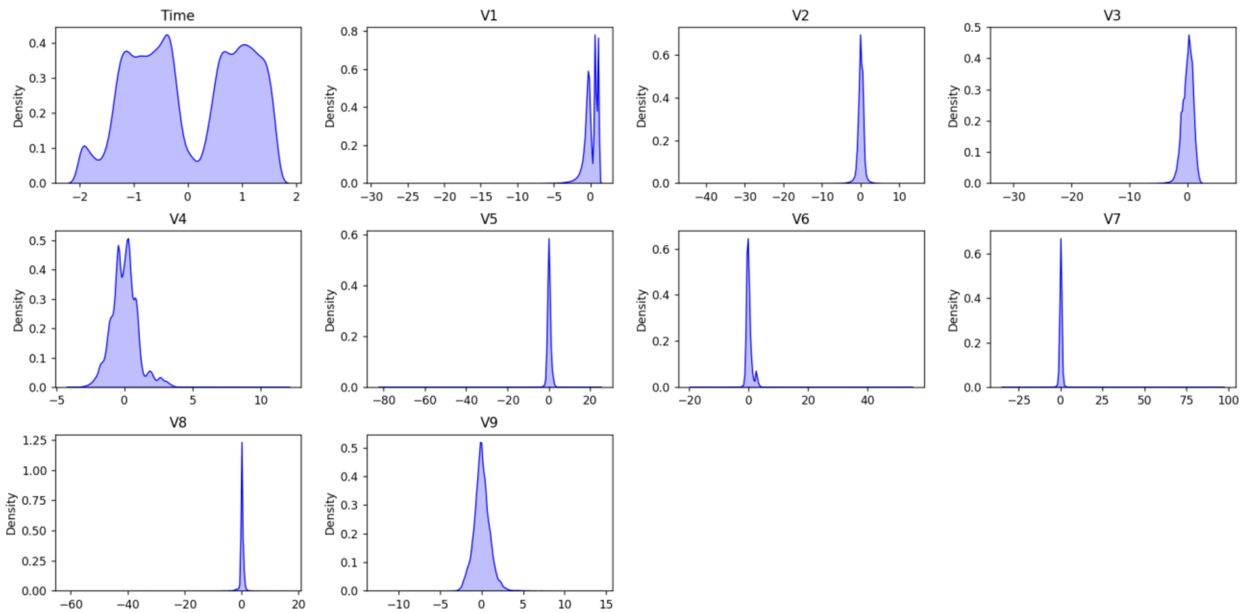
```
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import roc_curve, auc, precision_recall_curve


# Plot 1: Class Distribution5

plt.figure(figsize=(6, 4))

sns.countplot(x=y)

plt.title("Class Distribution")

plt.xlabel("Class (0: Genuine, 1: Fraudulent)")

plt.ylabel("Count")

plt.show()
```



```
# Plot 2: Correlation Heatmap

plt.figure(figsize=(12, 10))

correlation_matrix = data.corr()

sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False)

plt.title("Feature Correlation Heatmap")

plt.show()
```




```
# Plot 3: Feature Distribution (After Scaling)

plt.figure(figsize=(14, 8))

for i, col in enumerate(X.columns[:10], 1):  # Adjust for number of features to visualize

    plt.subplot(3, 4, i)

    sns.kdeplot(X_scaled[:, i - 1], color='blue', fill=True)

    plt.title(col)

plt.tight_layout()

plt.show()
```



Plot 4: ROC Curve

y_proba = classifier.predict_proba(X_test)[:, 1] # Probability scores for the positive class

print(y_proba)

fpr, tpr, _ = roc_curve(y_test, y_proba)

print(fpr, tpr)

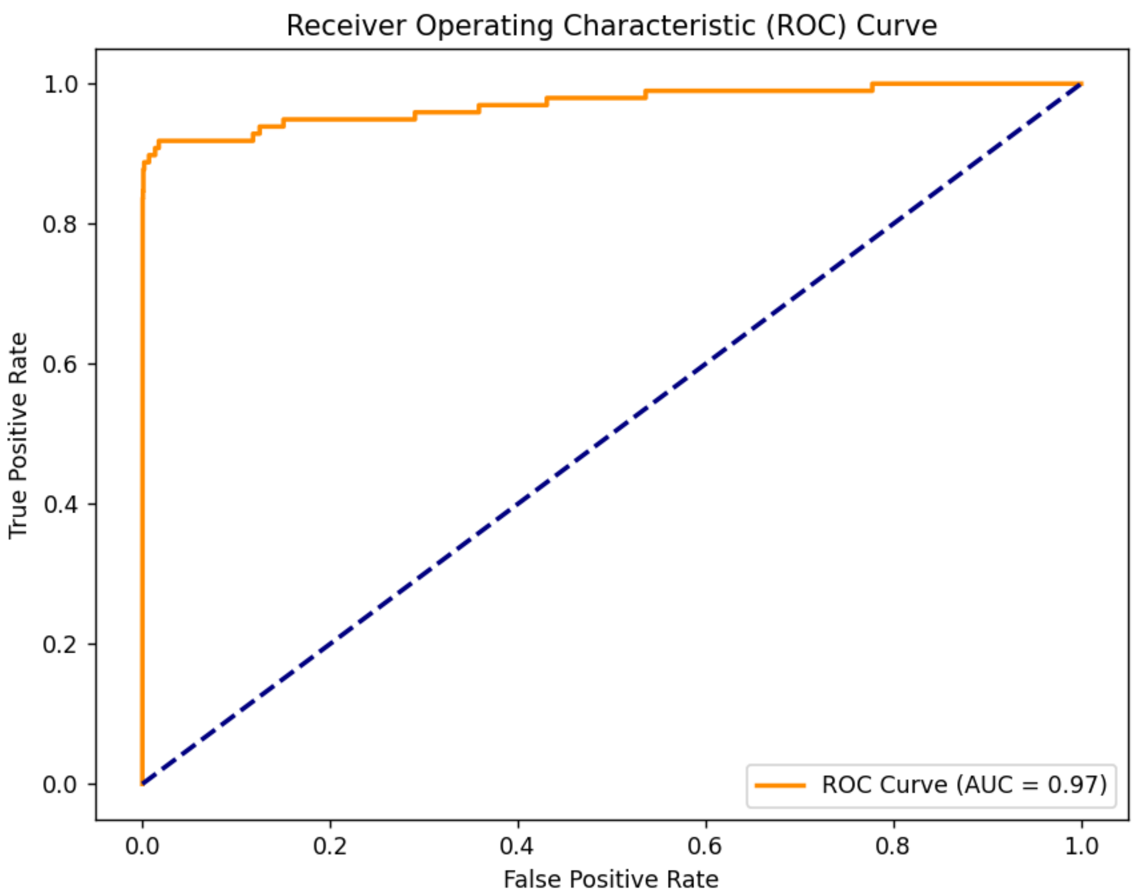
roc_auc = auc(fpr, tpr)

print(roc_auc)

```
 1.01726630e-03 5.69133528e-02]
[0.00000000e+00 1.93444007e-04 1.93444007e-04 1.93444007e-04
 2.11029826e-04 2.11029826e-04 2.28615644e-04 2.28615644e-04
 2.46201463e-04 2.46201463e-04 2.81373101e-04 2.81373101e-04
 3.86888014e-04 3.86888014e-04 6.85846933e-04 6.85846933e-04
 1.07273495e-03 1.07273495e-03 1.09032077e-03 1.09032077e-03
 1.95202589e-03 1.95202589e-03 7.08708497e-03 7.08708497e-03
 8.30050647e-03 8.33567811e-03 1.36465954e-02 1.36465954e-02
 1.75682330e-02 1.75682330e-02 2.87879854e-02 2.88231570e-02
 3.01420934e-02 3.01772651e-02 4.34018008e-02 4.34545582e-02
 5.11571469e-02 5.12274902e-02 5.44281092e-02 5.44632808e-02
 7.77469049e-02 7.77820765e-02 9.38027575e-02 9.38379291e-02
 1.12074423e-01 1.12109595e-01 1.15503658e-01 1.15538829e-01
 1.17350169e-01 1.17385340e-01 1.17912915e-01 1.17912915e-01
 1.25000000e-01 1.25000000e-01 1.30680219e-01 1.30715391e-01
 1.50358751e-01 1.50358751e-01 1.56443444e-01 1.56478616e-01
 1.66590461e-01 1.66625633e-01 1.70371412e-01 1.70406584e-01
 2.26118458e-01 2.26153630e-01 2.35896173e-01 2.35931345e-01
 2.55152645e-01 2.55187817e-01 2.60252532e-01 2.60287704e-01
 2.88425014e-01 2.88460186e-01 2.88495357e-01 2.88530529e-01
 2.90377040e-01 2.90377040e-01 3.36399128e-01 3.36434299e-01
 3.58557259e-01 3.58557259e-01 3.84971159e-01 3.85006331e-01
 3.89174170e-01 3.89226927e-01 4.15816685e-01 4.15851857e-01
 4.19052476e-01 4.19087648e-01 4.20951745e-01 4.20986916e-01
 4.30870146e-01 4.30870146e-01 5.36086100e-01 5.36086100e-01
 5.67037141e-01 5.67072313e-01 6.10579629e-01 6.10614800e-01
 6.47632949e-01 6.47668120e-01 6.51765616e-01 6.51800788e-01
```

```
  0.98979592 0.98979592 0.98979592 1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.         1.         1.         1.         1.         1.
  1.        ]
 0.970987885165321
```

```
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(loc="lower right")

plt.show()
```



Plot 5: Precision-Recall Curve

```
precision, recall, _ = precision_recall_curve(y_test, y_proba)
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.show()
```