```
# Importing necessary libraries
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import RocCurveDisplay
```

LOADING THE DATASET

```
# Load dataset
df = pd.read_csv("/content/UCI_Credit_Card.csv")
```

```
# To find the shape of the dataset
print(df.shape)
```

```
(30000, 25)
```

```
# To print the necessary data
print(df.head())
```

```
   ID  LIMIT_BAL  SEX  EDUCATION  MARRIAGE  AGE  PAY_0  PAY_2  PAY_3  PAY_4  \
0   1    20000.0    2          2         1   24      2      2     -1     -1
1   2   120000.0    2          2         2   26     -1      2      0      0
2   3    90000.0    2          2         2   34      0      0      0      0
3   4    50000.0    2          2         1   37      0      0      0      0
4   5    50000.0    1          2         1   57     -1      0     -1      0

   ...  BILL_AMT4  BILL_AMT5  BILL_AMT6  PAY_AMT1  PAY_AMT2  PAY_AMT3  \
0  ...        0.0        0.0        0.0       0.0     689.0       0.0
1  ...     3272.0     3455.0     3261.0       0.0    1000.0    1000.0
2  ...    14331.0    14948.0    15549.0    1518.0    1500.0    1000.0
3  ...    28314.0    28959.0    29547.0    2000.0    2019.0    1200.0
4  ...    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0
```

```
   PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
0       0.0       0.0       0.0                           1
1    1000.0       0.0    2000.0                           1
2    1000.0    1000.0    5000.0                           0
3    1100.0    1069.0    1000.0                           0
4    9000.0     689.0     679.0                           0

[5 rows x 25 columns]
```

```
# Describing the stats
print(df.describe())
```

```
                ID       LIMIT_BAL           SEX     EDUCATION      MARRIAGE  \
count  30000.000000    30000.000000  30000.000000  30000.000000  30000.000000
mean   15000.500000   167484.322667      1.603733      1.853133      1.551867
std     8660.398374   129747.661567      0.489129      0.790349      0.521970
min        1.000000    10000.000000      1.000000      0.000000      0.000000
25%     7500.750000    50000.000000      1.000000      1.000000      1.000000
50%    15000.500000   140000.000000      2.000000      2.000000      2.000000
75%    22500.250000   240000.000000      2.000000      2.000000      2.000000
max    30000.000000  1000000.000000      2.000000      6.000000      3.000000

                AGE         PAY_0         PAY_2         PAY_3         PAY_4  \
count  30000.000000  30000.000000  30000.000000  30000.000000  30000.000000
mean      35.485500     -0.016700     -0.133767     -0.166200     -0.220667
std        9.217904      1.123802      1.197186      1.196868      1.169139
min       21.000000     -2.000000     -2.000000     -2.000000     -2.000000
25%       28.000000     -1.000000     -1.000000     -1.000000     -1.000000
50%       34.000000      0.000000      0.000000      0.000000      0.000000
75%       41.000000      0.000000      0.000000      0.000000      0.000000
max       79.000000      8.000000      8.000000      8.000000      8.000000

       ...      BILL_AMT4      BILL_AMT5      BILL_AMT6       PAY_AMT1  \
count  ...   30000.000000   30000.000000   30000.000000   30000.000000
mean   ...   43262.948967   40311.400967   38871.760400    5663.580500
std    ...   64332.856134   60797.155770   59554.107537   16563.280354
min    ... -170000.000000  -81334.000000 -339603.000000       0.000000
25%    ...    2326.750000    1763.000000    1256.000000    1000.000000
50%    ...   19052.000000   18104.500000   17071.000000    2100.000000
75%    ...   54506.000000   50190.500000   49198.250000    5006.000000
max    ...  891586.000000  927171.000000  961664.000000  873552.000000

           PAY_AMT2      PAY_AMT3       PAY_AMT4       PAY_AMT5  \
count  3.000000e+04   30000.00000   30000.000000   30000.000000
```

```
mean   5.921163e+03    5225.68150    4826.076867    4799.387633
std    2.304087e+04   17606.96147   15666.159744   15278.305679
min    0.000000e+00       0.00000       0.000000       0.000000
25%    8.330000e+02     390.00000     296.000000     252.500000
50%    2.009000e+03    1800.00000    1500.000000    1500.000000
75%    5.000000e+03    4505.00000    4013.250000    4031.500000
max    1.684259e+06  896040.00000  621000.000000  426529.000000

            PAY_AMT6  default.payment.next.month
count   30000.000000                30000.000000
mean     5215.502567                    0.221200
std     17777.465775                    0.415062
min         0.000000                    0.000000
25%       117.750000                    0.000000
50%      1500.000000                    0.000000
75%      4000.000000                    0.000000
max    528666.000000                    1.000000

[8 rows x 25 columns]
```

```
# To see if there is missing value
print(df.isnull().sum())
```

```
ID                          0
LIMIT_BAL                   0
SEX                         0
EDUCATION                   0
MARRIAGE                    0
AGE                         0
PAY_0                       0
PAY_2                       0
PAY_3                       0
PAY_4                       0
PAY_5                       0
PAY_6                       0
BILL_AMT1                   0
BILL_AMT2                   0
BILL_AMT3                   0
BILL_AMT4                   0
BILL_AMT5                   0
BILL_AMT6                   0
PAY_AMT1                    0
PAY_AMT2                    0
PAY_AMT3                    0
PAY_AMT4                    0
```

```
PAY_AMT5                    0
PAY_AMT6                    0
default.payment.next.month  0
dtype: int64
```

PREPROCESSING

```
# Handling Missing values
df.fillna(df.median(), inplace=True)
```

```
# Encode Categorical Features (if any)
df = pd.get_dummies(df, drop_first=True)
```

```
# Separate Features & Target
X = df.drop("default.payment.next.month", axis=1)
y = df["default.payment.next.month"]
```

```
# Split - Training and Testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42, stratify=y
)
```

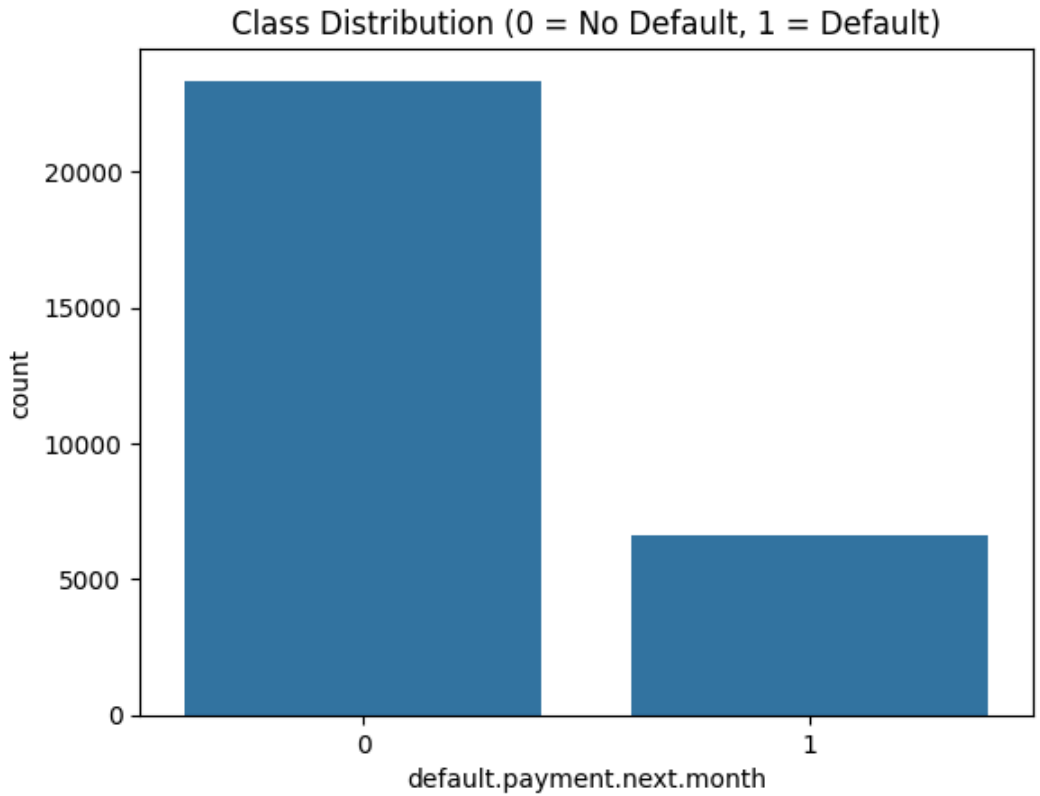
FEATURE SCALING

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

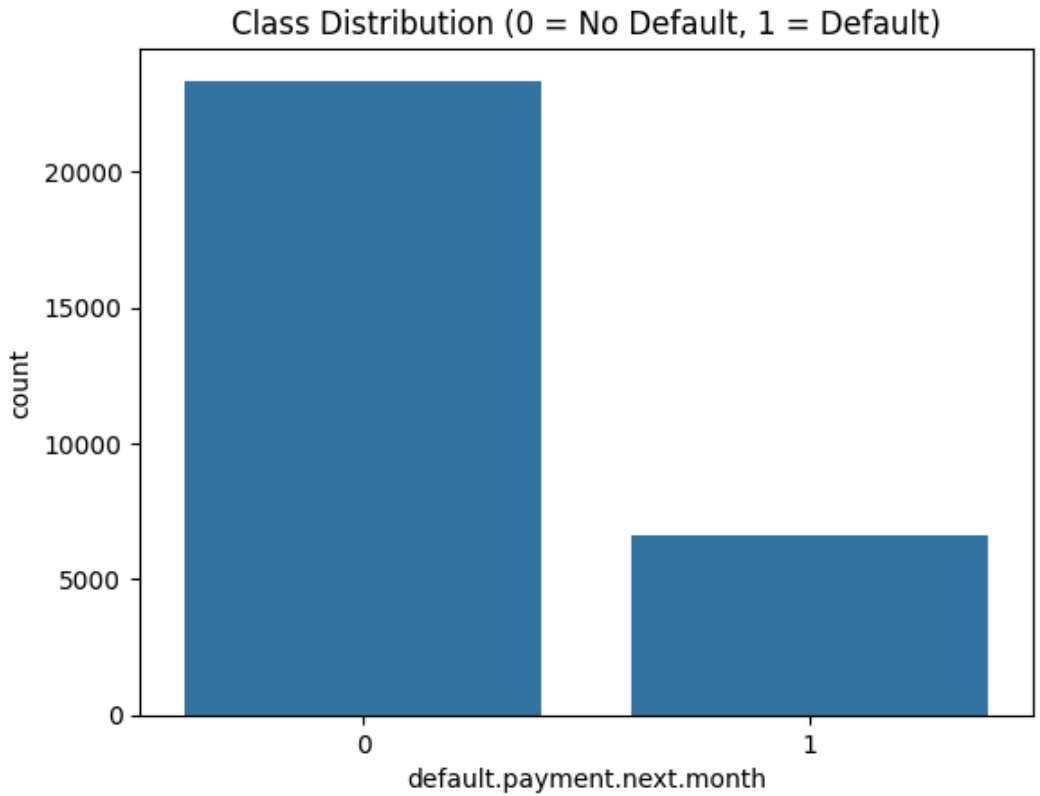
DATA VISUALIZATION

```
# CLASS DISTRIBUTION
plt.figure()
sns.countplot(x=y)
```

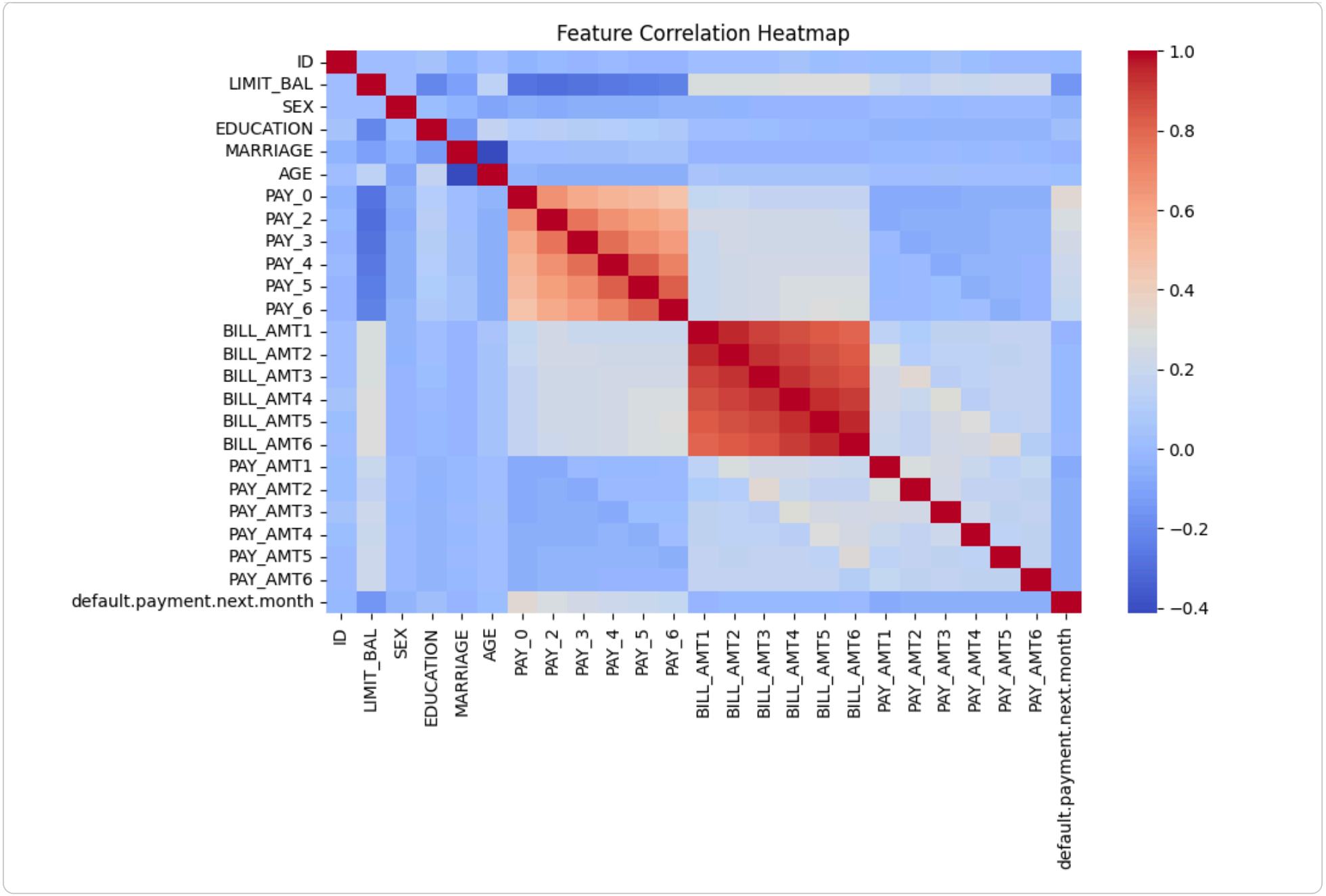
```
plt.title("Class Distribution (0 = No Default, 1 = Default)")
plt.show()
```




```
# FEATURE DISTRIBUTION
plt.figure()
sns.countplot(x=y)
plt.title("Class Distribution (0 = No Default, 1 = Default)")
plt.show()
```




```
#CORRELATION HEATMAPS
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), cmap="coolwarm")
plt.title("Feature Correlation Heatmap")
plt.show()
```



TRAINING THE MODEL

```
# Logistic Regression (baseline)
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train_scaled, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
# Random Forest
rf = RandomForestClassifier(n_estimators=200, random_state=42)
rf.fit(X_train, y_train)
```

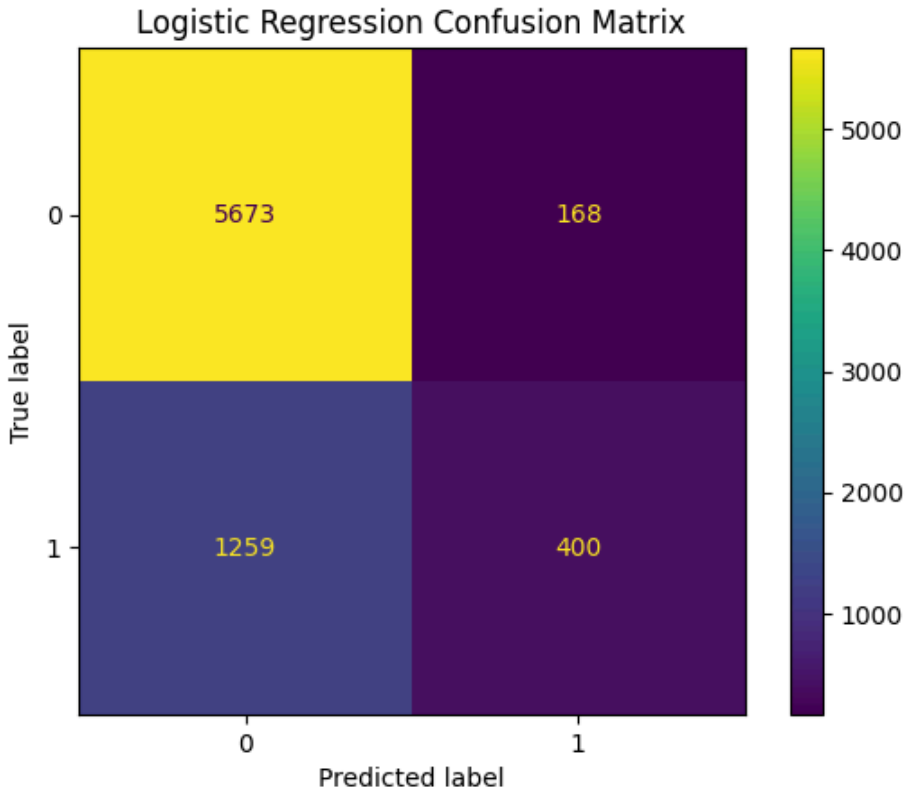
```
RandomForestClassifier
RandomForestClassifier(n_estimators=200, random_state=42)
```

```
# Model Evaluation
models = [("Logistic Regression", lr), ("Random Forest", rf)]

for name, model in models:
    y_pred = model.predict(X_test_scaled if name=="Logistic Regression" else X_test)
    print(f"=== {name} ===")
    print(classification_report(y_test, y_pred))
    print("ROC AUC:", roc_auc_score(y_test, model.predict_proba(X_test_scaled if name=="Logistic Regression" else X_test)[:,1]))
    print(confusion_matrix(y_test, y_pred))
```

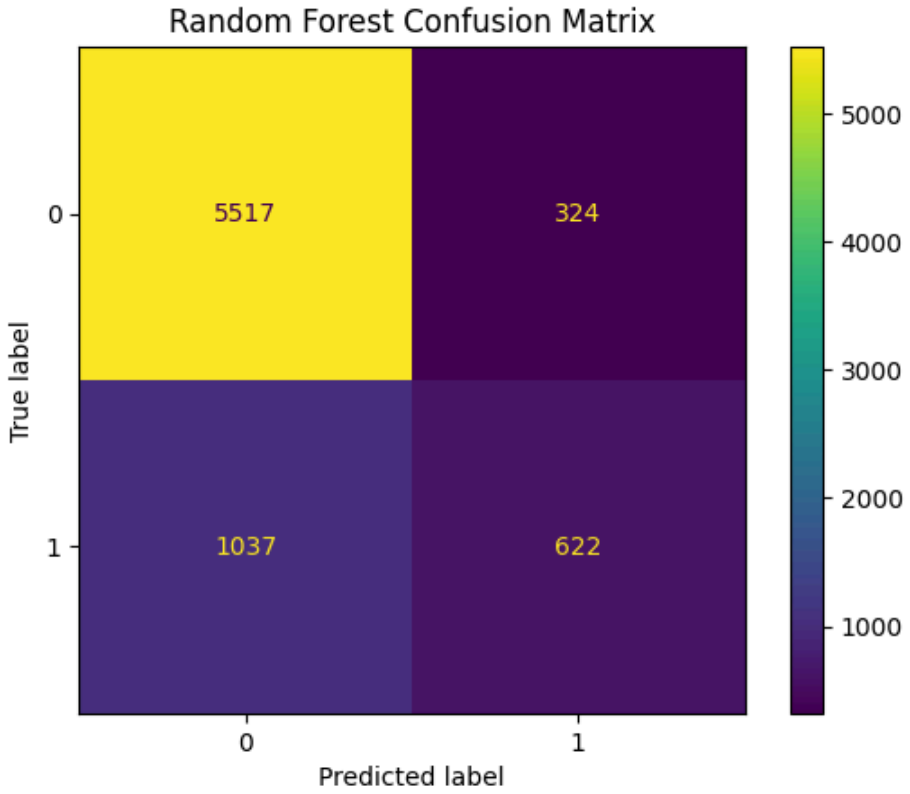
```
=== Logistic Regression ===
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      5841
           1       0.70      0.24      0.36      1659

    accuracy                           0.81      7500
   macro avg       0.76      0.61      0.62      7500
weighted avg       0.79      0.81      0.77      7500

ROC AUC: 0.7157150937455593
[[5673  168]
 [1259  400]]
=== Random Forest ===
              precision    recall  f1-score   support
```

```
              0       0.84      0.94      0.89      5841
              1       0.66      0.37      0.48      1659

       accuracy                           0.82      7500
      macro avg       0.75      0.66      0.68      7500
   weighted avg       0.80      0.82      0.80      7500

ROC AUC: 0.7597332423549973
[[5517  324]
 [1037  622]]
```

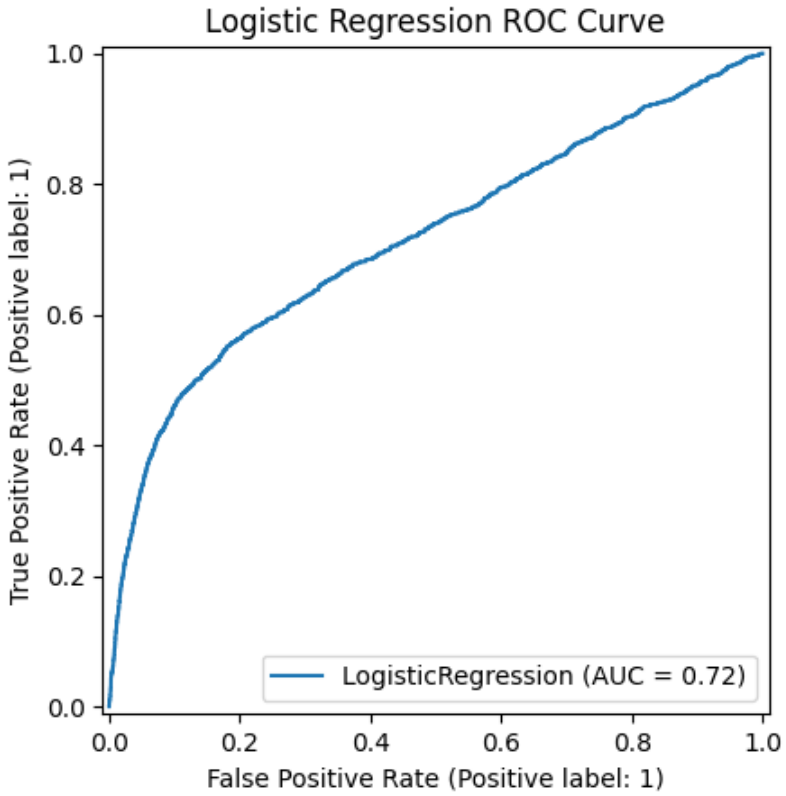
```
ConfusionMatrixDisplay.from_estimator(
    lr, X_test_scaled, y_test
)
plt.title("Logistic Regression Confusion Matrix")
plt.show()
```



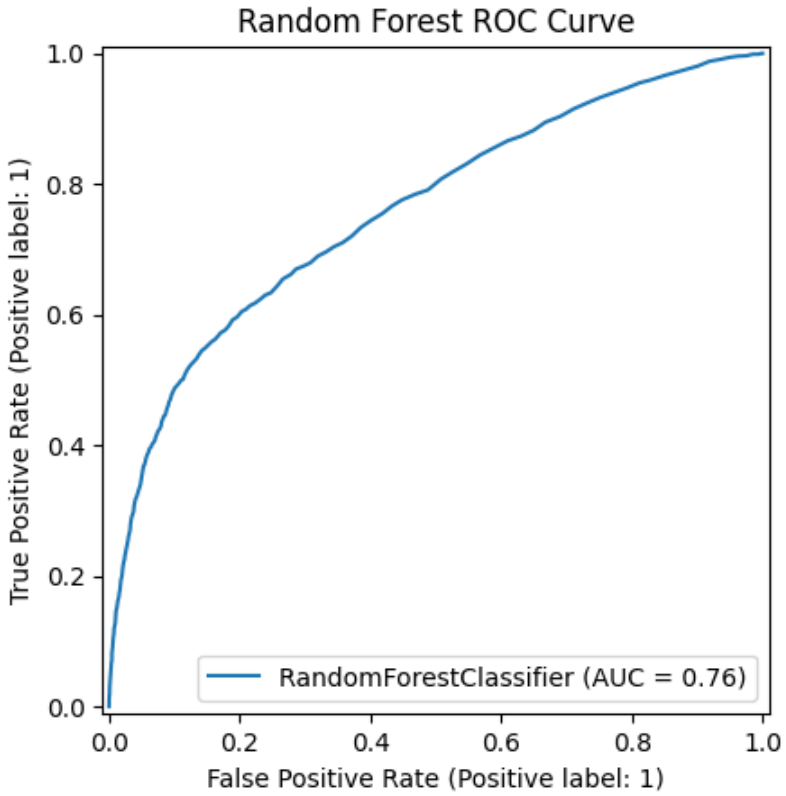
```
ConfusionMatrixDisplay.from_estimator(
    rf, X_test, y_test
)
plt.title("Random Forest Confusion Matrix")
plt.show()
```




```
RocCurveDisplay.from_estimator(
    lr, X_test_scaled, y_test
)
plt.title("Logistic Regression ROC Curve")
plt.show()
```




```
RocCurveDisplay.from_estimator(
    rf, X_test, y_test
)
plt.title("Random Forest ROC Curve")
plt.show()
```



SIMPLE PREDICTION

```
# Test Data
rf_predictions = rf.predict(X_test)

print(rf_predictions[:10])
```

```
[1 0 0 0 0 0 1 0 0 0]
```

```
# on probability
rf_probabilities = rf.predict_proba(X_test)

print(rf_probabilities[:5])
```

```
[[0.445 0.555]
 [0.56  0.44 ]
 [0.795 0.205]
 [0.59  0.41 ]
 [0.815 0.185]]
```

```
# Prediction for one new customer
new_customer = X_test.iloc[[0]]  # taking one real example

prediction = rf.predict(new_customer)
probability = rf.predict_proba(new_customer)
```