```python
#Importing all the libraries for further actions
# TAsk 1 : A data exploration and preprocessing
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
df = pd.read_csv('customer_data.csv')
df.head()
```

{"type":"dataframe","variable_name":"df"}

```python
# Taking overview
print(df.shape)
print(df.info())
print(df.describe())
```

```
(7043, 21)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7032 non-null   float64
 20  Churn             7043 non-null   object 
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
None
       SeniorCitizen       tenure  MonthlyCharges  TotalCharges
count    7043.000000  7043.000000     7043.000000   7032.000000
```

```
mean        0.162147    32.371149       64.761692   2283.300441
std         0.368612    24.559481       30.090047   2266.771362
min         0.000000     0.000000       18.250000     18.800000
25%         0.000000     9.000000       35.500000    401.450000
50%         0.000000    29.000000       70.350000   1397.475000
75%         0.000000    55.000000       89.850000   3794.737500
max         1.000000    72.000000      118.750000   8684.800000
```

```python
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],
errors='coerce')  # convert to numeric
print(df.isnull().sum())

# Drop or fill nulls
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)
```

```
gender                                     0
SeniorCitizen                              0
Partner                                    0
Dependents                                 0
tenure                                     0
PhoneService                               0
PaperlessBilling                           0
MonthlyCharges                             0
TotalCharges                               0
Churn                                      0
MultipleLines_No                           0
MultipleLines_No phone service             0
MultipleLines_Yes                          0
InternetService_DSL                        0
InternetService_Fiber optic                0
InternetService_No                         0
OnlineSecurity_No                          0
OnlineSecurity_No internet service         0
OnlineSecurity_Yes                         0
OnlineBackup_No                            0
OnlineBackup_No internet service           0
OnlineBackup_Yes                           0
DeviceProtection_No                        0
DeviceProtection_No internet service       0
DeviceProtection_Yes                       0
TechSupport_No                             0
TechSupport_No internet service            0
TechSupport_Yes                            0
StreamingTV_No                             0
StreamingTV_No internet service            0
StreamingTV_Yes                            0
StreamingMovies_No                         0
StreamingMovies_No internet service        0
StreamingMovies_Yes                        0
Contract_Month-to-month                    0
```

```
Contract_One year                          0
Contract_Two year                          0
PaymentMethod_Bank transfer (automatic)    0
PaymentMethod_Credit card (automatic)      0
PaymentMethod_Electronic check             0
PaymentMethod_Mailed check                 0
dtype: int64
```

```
binary_cols = ['gender', 'Partner', 'Dependents', 'PhoneService',
'PaperlessBilling', 'Churn']
le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])

scaler = StandardScaler()
df[['MonthlyCharges', 'TotalCharges', 'tenure']] =
scaler.fit_transform(df[['MonthlyCharges', 'TotalCharges', 'tenure']])

print(df.shape)
print(df.columns)
df.head()
```

```
(7043, 20)
Index(['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure',
       'PhoneService', 'MultipleLines', 'InternetService',
'OnlineSecurity',
       'OnlineBackup', 'DeviceProtection', 'TechSupport',
'StreamingTV',
       'StreamingMovies', 'Contract', 'PaperlessBilling',
'PaymentMethod',
       'MonthlyCharges', 'TotalCharges', 'Churn'],
      dtype='object')
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 7043,\n  \"fields\":
[\n    {\n      \"column\": \"gender\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"SeniorCitizen\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Partner\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          0,\n          1\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Dependents\",\n      \"properties\": {\n
```

```
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 1,\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          1,\n          0\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"tenure\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0000710000355943,\n        \"min\": -
1.318164947398796,\n        \"max\": 1.6137012404433893,\n
\"num_unique_values\": 73,\n        \"samples\": [\n          -
0.9924020376385531,\n          0.3106496014024181\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PhoneService\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MultipleLines\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No phone service\",\n          \"No\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"InternetService\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"DSL\",\
n          \"Fiber optic\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"OnlineSecurity\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"OnlineBackup\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"Yes\",\
n          \"No\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DeviceProtection\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TechSupport\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"StreamingTV\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"No\",\n          \"Yes\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"StreamingMovies\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
```

```
\"num_unique_values\": 3,\n        \"samples\": [\n          \"No\",\n
\"Yes\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Contract\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Month-to-month\",\n          \"One year\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"PaperlessBilling\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          0,\n
1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PaymentMethod\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          \"Mailed check\",\n          \"Credit card (automatic)\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"MonthlyCharges\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.0000710000355904,\n        \"min\": -
1.5458598200734601,\n        \"max\": 1.7943521502604476,\n
\"num_unique_values\": 1585,\n        \"samples\": [\n          -
0.5288400559717926,\n          -1.4860351280674797\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"TotalCharges\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.0000710000355915,\n        \"min\": -0.9991202865028981,\n
\"max\": 2.8267431919212935,\n        \"num_unique_values\": 6531,\n
\"samples\": [\n          1.02369576147677,\n          -
0.9984359928966718\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Churn\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"df"}
```


```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score
import xgboost as xgb

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

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

```
X_train.select_dtypes(include='object').columns
```

```
Index(['MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup',
       'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies',
       'Contract', 'PaymentMethod'],
      dtype='object')
```

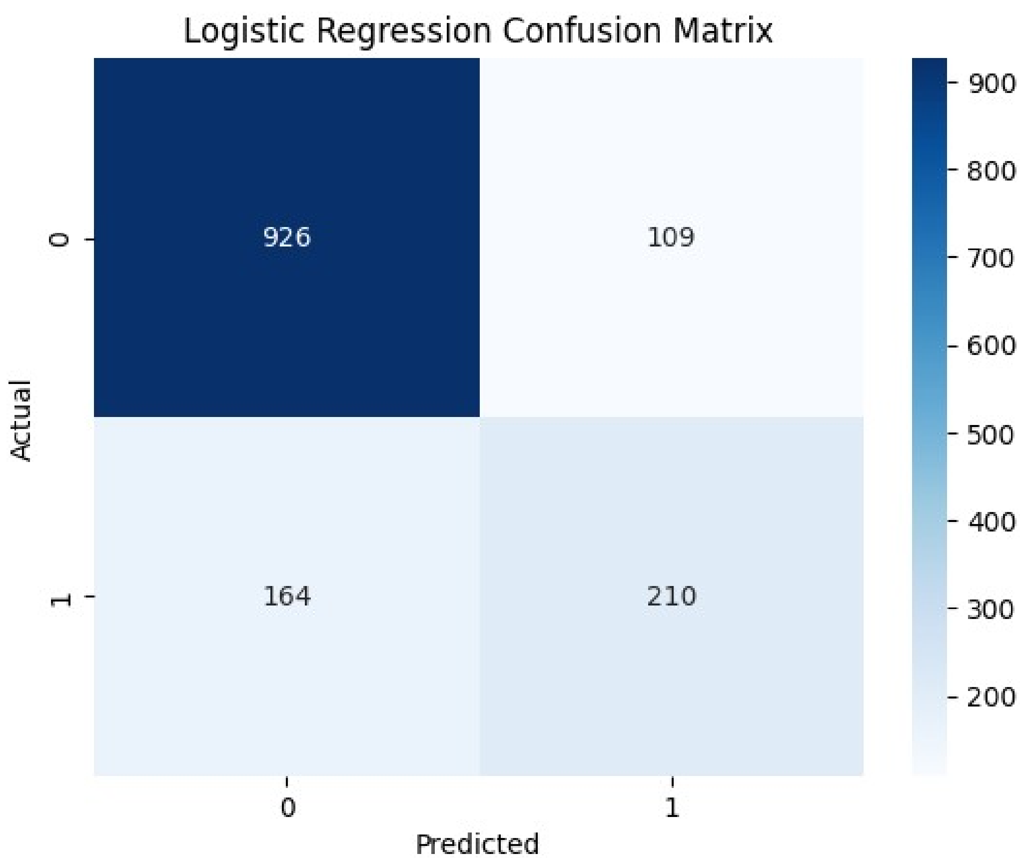
```
# Converting to numeric
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],
errors='coerce')
df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)

binary_cols = ['gender', 'Partner', 'Dependents', 'PhoneService',
'PaperlessBilling', 'Churn']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])


multi_class_cols = [
    'MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup',
    'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies',
    'Contract', 'PaymentMethod'
]
df = pd.get_dummies(df, columns=multi_class_cols)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df[['MonthlyCharges', 'TotalCharges', 'tenure']] =
scaler.fit_transform(df[['MonthlyCharges', 'TotalCharges', 'tenure']])
```

```
<ipython-input-24-e0aa3accb770>:3: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.
```

```
  df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)

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

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y, random_state=42)

from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(max_iter=1000)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)



# Machine Leraning model
# Import evaluation metric

from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report,
roc_auc_score

def evaluate_model(name, y_test, y_pred, y_proba=None):
    print(f"--- {name} ---")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))
    if y_proba is not None:
        print("ROC AUC Score:", roc_auc_score(y_test, y_proba))
    print("\nClassification Report:\n", classification_report(y_test,
y_pred))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

y_pred_lr = lr.predict(X_test)
y_proba_lr = lr.predict_proba(X_test)[:,1]
evaluate_model("Logistic Regression", y_test, y_pred_lr, y_proba_lr)
```

```
--- Logistic Regression ---
Accuracy: 0.8062455642299503
Precision: 0.658307210031348
Recall: 0.5614973262032086
F1 Score: 0.6060606060606061
ROC AUC Score: 0.8419153168513782
```

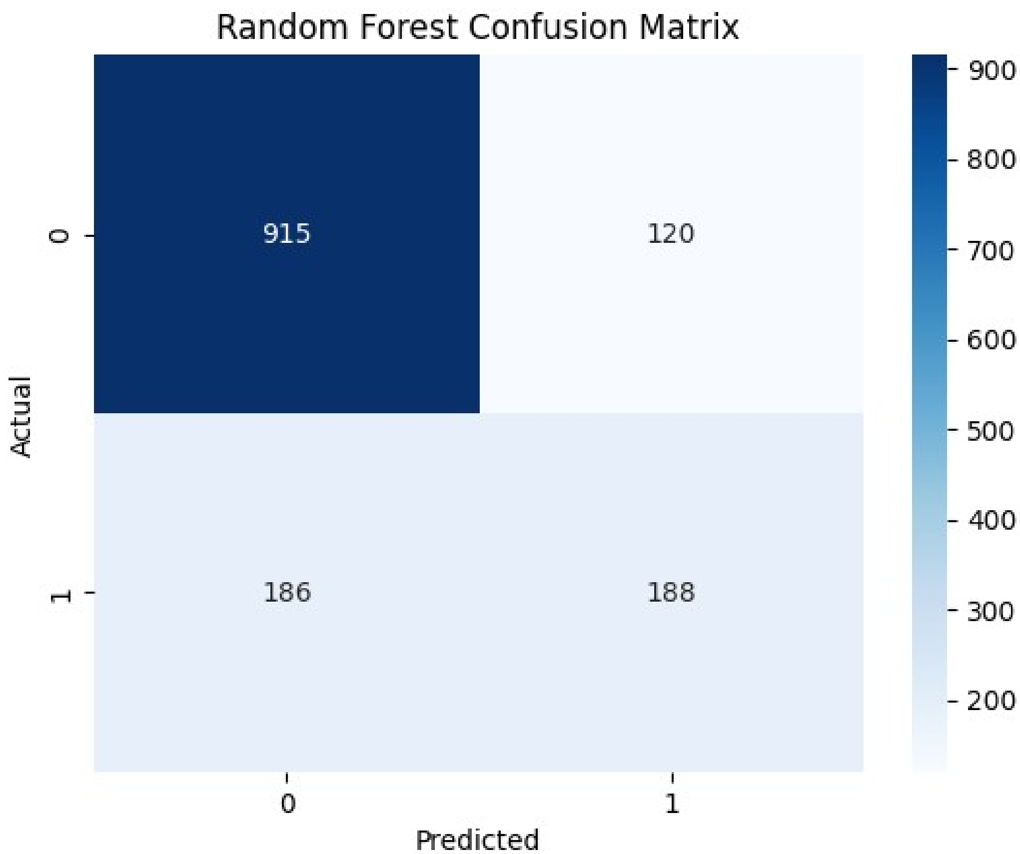
```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1035
           1       0.66      0.56      0.61       374

    accuracy                           0.81      1409
   macro avg       0.75      0.73      0.74      1409
weighted avg       0.80      0.81      0.80      1409
```




```python
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
y_proba_rf = rf.predict_proba(X_test)[:,1]
evaluate_model("Random Forest", y_test, y_pred_rf, y_proba_rf)
```

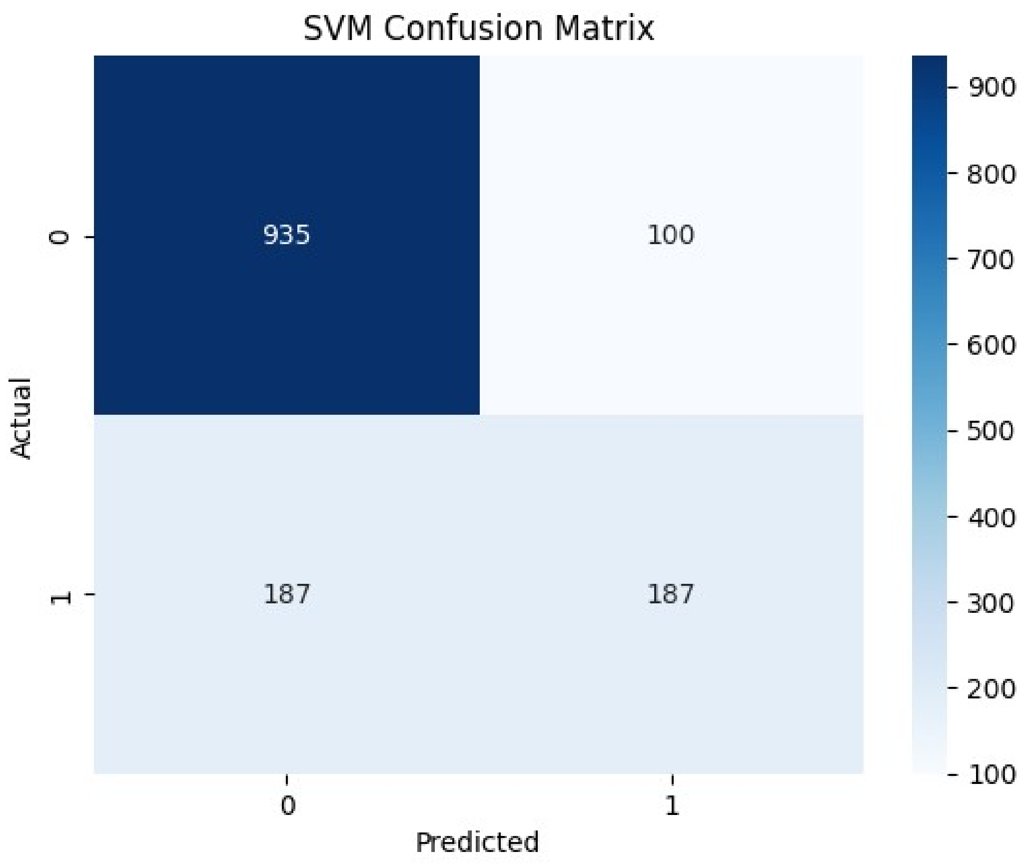
```
--- Random Forest ---
Accuracy: 0.7828246983676366
Precision: 0.6103896103896104
Recall: 0.5026737967914439
```

```
F1 Score: 0.5513196480938416
ROC AUC Score: 0.8180268154692707

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.88      0.86      1035
           1       0.61      0.50      0.55       374

    accuracy                           0.78      1409
   macro avg       0.72      0.69      0.70      1409
weighted avg       0.77      0.78      0.78      1409
```



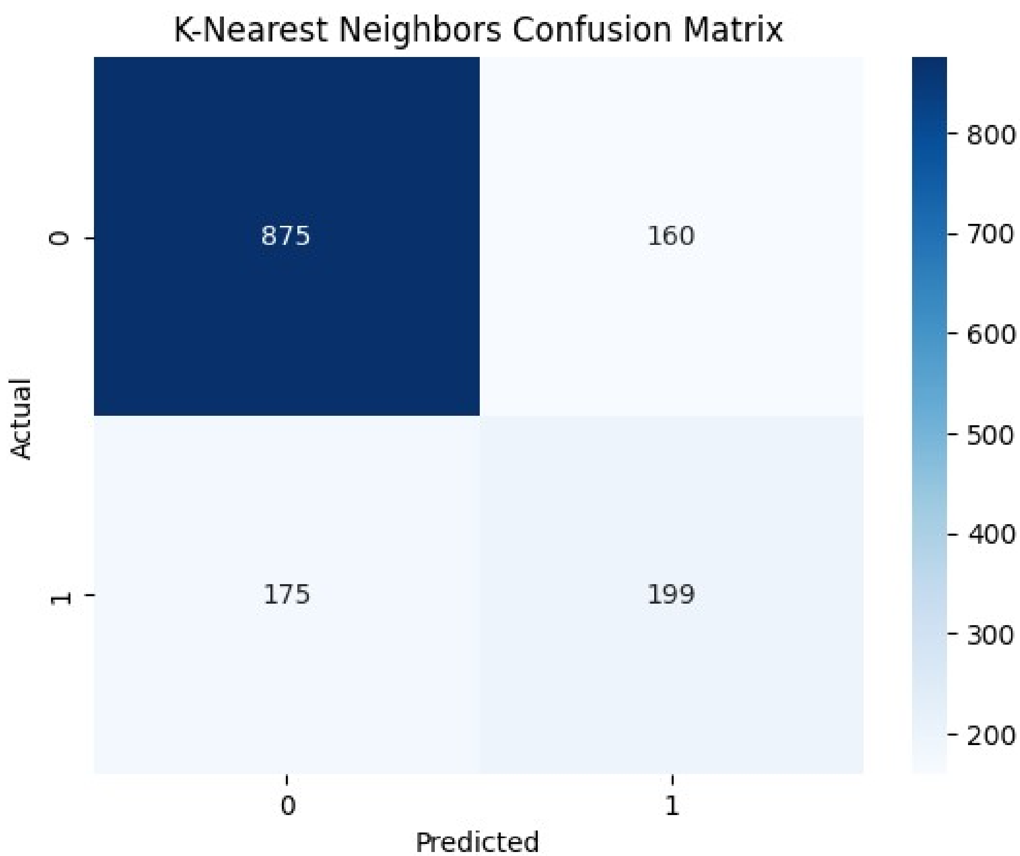

```
from sklearn.svm import SVC
svc = SVC(probability=True)
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)
y_proba_svc = svc.predict_proba(X_test)[:,1]
evaluate_model("SVM", y_test, y_pred_svc, y_proba_svc)

--- SVM ---
Accuracy: 0.7963094393186657
```

```
Precision: 0.6515679442508711
Recall: 0.5
F1 Score: 0.5658093797276853
ROC AUC Score: 0.7942752331499134

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.90      0.87      1035
           1       0.65      0.50      0.57       374

    accuracy                           0.80      1409
   macro avg       0.74      0.70      0.72      1409
weighted avg       0.79      0.80      0.79      1409
```




```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
y_proba_knn = knn.predict_proba(X_test)[:,1]
evaluate_model("K-Nearest Neighbors", y_test, y_pred_knn, y_proba_knn)
```

```
--- K-Nearest Neighbors ---
Accuracy: 0.7622427253371186
Precision: 0.5543175487465181
Recall: 0.5320855614973262
F1 Score: 0.5429740791268759
ROC AUC Score: 0.7766798418972332

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.85      0.84      1035
           1       0.55      0.53      0.54       374

    accuracy                           0.76      1409
   macro avg       0.69      0.69      0.69      1409
weighted avg       0.76      0.76      0.76      1409
```



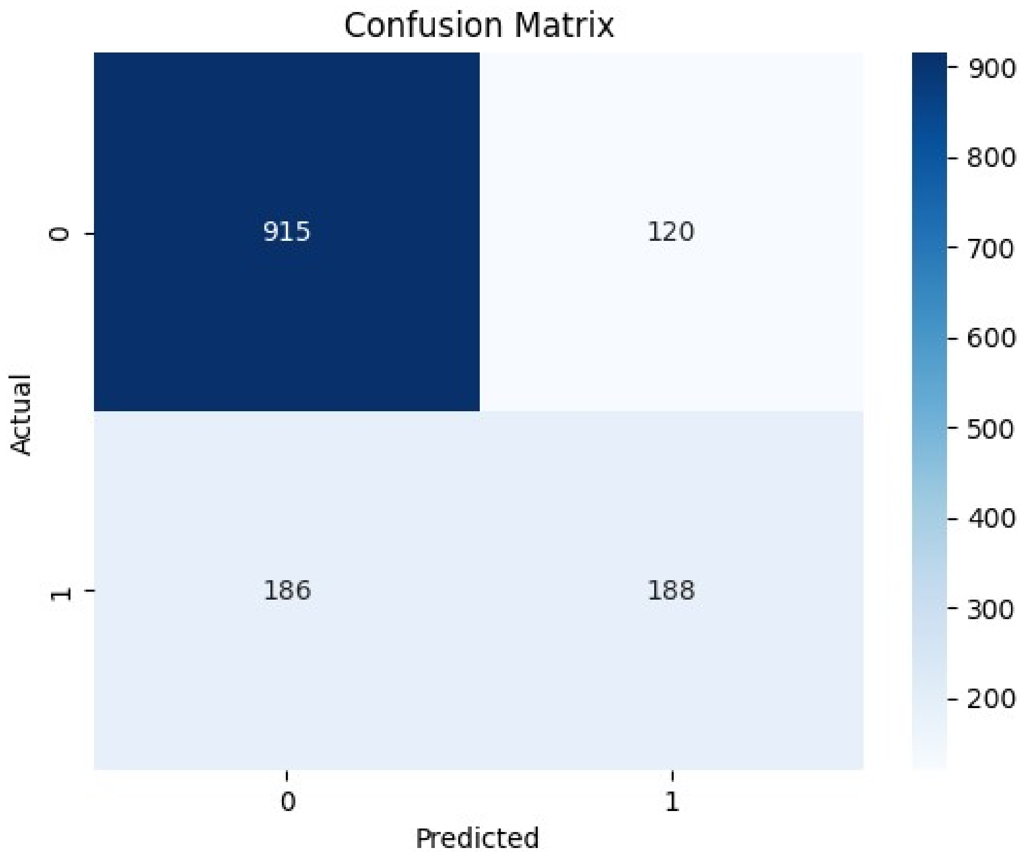

```
# Import libraries
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score,
```

```
recall_score, f1_score, classification_report, confusion_matrix

# Dataset load
df = pd.read_csv('customer_data.csv')

df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],
errors='coerce')

df['TotalCharges'].fillna(df['TotalCharges'].median(), inplace=True)

df.drop('customerID', axis=1, inplace=True)

binary_cols = ['gender', 'Partner', 'Dependents', 'PhoneService',
'PaperlessBilling', 'Churn']
le = LabelEncoder()
for col in binary_cols:
    df[col] = le.fit_transform(df[col])


df = pd.get_dummies(df, columns=[
    'MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup',
    'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies',
    'Contract', 'PaymentMethod'
])

scaler = StandardScaler()
df[['MonthlyCharges', 'TotalCharges', 'tenure']] =
scaler.fit_transform(df[['MonthlyCharges', 'TotalCharges', 'tenure']])

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

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


model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)


y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Accuracy: 0.7828246983676366
Precision: 0.6103896103896104
Recall: 0.5026737967914439
F1 Score: 0.5513196480938416
```

```python
print("\nClassification Report:\n", classification_report(y_test,
y_pred))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d',
cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

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

           0       0.83      0.88      0.86      1035
           1       0.61      0.50      0.55       374

    accuracy                           0.78      1409
   macro avg       0.72      0.69      0.70      1409
weighted avg       0.77      0.78      0.78      1409
```


Confusion Matrix
915
120
186
188
Actual
Predicted
0
1
900
800
700
600
500
400
300
200