Group 3: ML Assignment 5

Objective

To predict which customer is more likely to purchase the newly introduced telecom plan.

Data Dictionary

- **CustomerID:** Unique customer ID
- **PlanTaken:** Whether the customer has purchased the plan or not (0: No, 1: Yes)
- **Age:** Age of customer
- **TypeofContact:** How customer was contacted (Company Invited or Self Inquiry)
- **CityTier:** City tier depends on the development of a city, population, facilities, and living standards. The categories are ordered i.e. Tier 1 > Tier 2 > Tier 3
- **Occupation:** Occupation of customer
- **Gender:** Gender of customer
- **NumberOfPersons:** Total number of persons planning to take the plan with the customer (since these are Friends and Family plans)
- **PreferredServiceStar:** Preferred service rating by customer
- **MaritalStatus:** Marital status of customer
- **NumberOfUpgrades:** Average number of upgrades in a year by customer
- **iPhone:** The customer has an iphone or not (0: No, 1: Yes)
- **PhoneContract:** Whether the customers has a contracted phone or not (0: No, 1: Yes)
- **NumberOfChildren:** Total number of children planning to take the plan with the customer
- **Designation:** Designation of the customer in the current organization
- **MonthlyIncome:** Gross monthly income of the customer
- **PitchSatisfactionScore:** Sales pitch satisfaction score
- **PlanPitched:** Plan pitched by the salesperson
- **NumberOfFollowups:** Total number of follow-ups has been done by the salesperson after the sales pitch
- **DurationOfPitch:** Duration of the pitch by a salesperson to the customer

Tasks:

- EDA
- Data Cleaning
- Data Visualization
- Feature Engineering & Data Preprocessing
- Model Building & Evaluation
- Model Tuning 3 rounds

Note:

- Use Logistic Regression w/ Regularization and SVM techniques

- Compare the Logistic Regression technique you use for this problem with SVM techniques and interpret/explain the differences.
- Your models should be tested on at least 1000 randomly selected data points

Let's start coding!

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from matplotlib import pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, roc_curv
from imblearn.over_sampling import SMOTE
import warnings

warnings.filterwarnings('ignore')
```

Read the dataset

In [2]:

```
df_path = '~/code/loyalist-college/sem-1/ml-1/assignment-5/Telecom.xlsx'
df = pd.read_excel(df_path, sheet_name="Telecom")
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4888 entries, 0 to 4887
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   CustomerID              4888 non-null   int64  
 1   PlanTaken               4888 non-null   int64  
 2   Age                     4662 non-null   float64
 3   TypeofContact           4863 non-null   object 
 4   CityTier                4888 non-null   int64  
 5   DurationOfPitch         4637 non-null   float64
 6   Occupation              4888 non-null   object 
 7   Gender                  4888 non-null   object 
 8   NumberOfPersons         4888 non-null   int64  
 9   NumberOfFollowups       4843 non-null   float64
 10  PlanPitched             4888 non-null   object 
 11  PreferredServiceStar    4862 non-null   float64
 12  MaritalStatus           4888 non-null   object 
 13  NumberOfUpgrades        4748 non-null   float64
 14  iPhone                  4888 non-null   int64  
 15  PitchSatisfactionScore  4888 non-null   int64  
 16  PhoneContract           4888 non-null   int64  
 17  NumberOfChildren        4822 non-null   float64
 18  Designation             4888 non-null   object 
 19  MonthlyIncome           4655 non-null   float64
dtypes: float64(7), int64(7), object(6)
memory usage: 763.9+ KB
```

Out[2]:

	CustomerID	PlanTaken	Age	CityTier	DurationOfPitch	NumberOfPersons
count	4888.000000	4888.000000	4662.000000	4888.000000	4637.000000	4888.000000
mean	202443.500000	0.188216	37.622265	1.654255	15.490835	2.905074
std	1411.188388	0.390925	9.316387	0.916583	8.519643	0.724891
min	200000.000000	0.000000	18.000000	1.000000	5.000000	1.000000
25%	201221.750000	0.000000	31.000000	1.000000	9.000000	2.000000
50%	202443.500000	0.000000	36.000000	1.000000	13.000000	3.000000
75%	203665.250000	0.000000	44.000000	3.000000	20.000000	3.000000
max	204887.000000	1.000000	61.000000	3.000000	127.000000	5.000000

Feature Engineering

Remove duplicates

In [3]:

```
df
df.drop_duplicates()
```

Out[3]:

	CustomerID	PlanTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	Gende
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	Female
1	200001	0	49.0	Company Invited	1	14.0	Salaried	Male
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	Male
3	200003	0	33.0	Company Invited	1	9.0	Salaried	Female
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	Male
...	...	...	...	...	...	...	...	..
4883	204883	1	49.0	Self Enquiry	3	9.0	Small Business	Male
4884	204884	1	28.0	Company Invited	1	31.0	Salaried	Male
4885	204885	1	52.0	Self Enquiry	3	17.0	Salaried	Female
4886	204886	1	19.0	Self Enquiry	3	16.0	Small Business	Male
4887	204887	1	36.0	Self Enquiry	1	14.0	Salaried	Male

4888 rows × 20 columns

Drop Occupation - Free Lancer

In [4]:
```python
df = df.drop(df[df["Occupation"] == 'Free Lancer'].index)
```

Select features

- Select independent features as X
- Select dependent feature for y

In [5]:
```python
def select_features_for_model(df):
    X = df.drop("PlanTaken", axis=1)
    y = df["PlanTaken"]
    X.info()
    return X, y

X, y = select_features_for_model(df)
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4886 entries, 0 to 4887
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   CustomerID              4886 non-null   int64  
 1   Age                     4660 non-null   float64
 2   TypeofContact           4861 non-null   object 
 3   CityTier                4886 non-null   int64  
 4   DurationOfPitch         4635 non-null   float64
 5   Occupation              4886 non-null   object 
 6   Gender                  4886 non-null   object 
 7   NumberOfPersons         4886 non-null   int64  
 8   NumberOfFollowups       4841 non-null   float64
 9   PlanPitched             4886 non-null   object 
 10  PreferredServiceStar    4860 non-null   float64
 11  MaritalStatus           4886 non-null   object 
 12  NumberOfUpgrades        4746 non-null   float64
 13  iPhone                  4886 non-null   int64  
 14  PitchSatisfactionScore  4886 non-null   int64  
 15  PhoneContract           4886 non-null   int64  
 16  NumberOfChildren        4820 non-null   float64
 17  Designation             4886 non-null   object 
 18  MonthlyIncome           4653 non-null   float64
dtypes: float64(7), int64(6), object(6)
memory usage: 763.4+ KB
```

Split Train & Test

In [6]:
```python
def split_train_test(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
    return X_train, X_test, y_train, y_test

X_train, X_test, y_train, y_test = split_train_test(X, y)
```

Data Cleaning

Gender

In [7]:
```python
# Checking the unique values of gender in X_train
X_train['Gender'].unique()
```

Out[7]:
```
array(['Male', 'Female', 'Fe Male'], dtype=object)
```

In [8]:
```python
X_train['Gender'].value_counts()
```

Out[8]:
```
Gender
Male       2334
Female     1452
Fe Male     122
Name: count, dtype: int64
```

In [9]:
```python
# Correcting the Fe Male gender as Female
def clean_Gender(df):
    df['Gender'] = df['Gender'].map({'Male': 'Male', 'Female':'Female', 'Fe Male':
    return df

# Clean Train
X_train = clean_Gender(X_train)

# Clean Test
X_test = clean_Gender(X_test)
```

In [10]:
```python
X_train['Gender'].value_counts()
```

Out[10]:
```
Gender
Male      2334
Female    1574
Name: count, dtype: int64
```

MaritalStatus

In [11]:
```python
X_train['MaritalStatus'].value_counts()
```

Out[11]:
```
MaritalStatus
Married      1895
Divorced      751
Single        717
Unmarried     545
Name: count, dtype: int64
```

In [12]:
```python
# Merging Unmarried to Single
def clean_MaritalStatus(df):
    df['MaritalStatus'] = df['MaritalStatus'].map({'Married': 'Married',
                                                   'Divorced':'Divorced',
                                                   'Single': 'Single',
                                                   'Unmarried': 'Single'})

    return df

# Clean Train
X_train = clean_MaritalStatus(X_train)

# Clean Test
X_test = clean_MaritalStatus(X_test)
```

In [13]:
```python
X_train['MaritalStatus'].value_counts()
```

Out[13]:
```
MaritalStatus
Married     1895
Single      1262
Divorced     751
Name: count, dtype: int64
```

Impute missing values

In [14]:
```python
X_train.isnull().sum()
```

Out[14]:
```
CustomerID                 0
Age                      176
TypeofContact             19
CityTier                   0
DurationOfPitch          211
Occupation                 0
Gender                     0
NumberOfPersons            0
NumberOfFollowups         37
PlanPitched                0
PreferredServiceStar      20
MaritalStatus              0
NumberOfUpgrades         122
iPhone                     0
PitchSatisfactionScore     0
PhoneContract              0
NumberOfChildren          54
Designation                0
MonthlyIncome            182
dtype: int64
```

In [15]:
```python
def impute_features(df):
    numeric_cols = ['DurationOfPitch', 'MonthlyIncome', 'Age', 'NumberOfUpgrades',
    mean_values = df[numeric_cols].mean()

    df[numeric_cols] = df[numeric_cols].fillna(mean_values)

    categorical_cols = ['TypeofContact']
    df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().


    return df


X_train = impute_features(X_train)

X_train.isnull().sum()

X_test = impute_features(X_test)
```

Handling Outliers

- Income can be dependent on Occupation and Designation, therefore removing the outliers based on them

In [16]:
```python
def handle_outliers(df):

    numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns

    # Calculateing IQR
    Q1 = df[numeric_columns].quantile(0.25)
    Q3 = df[numeric_columns].quantile(0.75)
    IQR = Q3 - Q1

    # Outlier threshold - 1.5 times IQR
    threshold = 1.5

    # Finding the outliers using the threshold value
```

```
    outliers = np.logical_or(df[numeric_columns] < (Q1 - threshold * IQR), df[numer

    # Total outliers in each numerical columns
    outliers_count = outliers.sum(axis=0)

    columns_to_replace_with_median = ['DurationOfPitch', 'NumberOfUpgrades']
    df[columns_to_replace_with_median] = np.where(outliers[columns_to_replace_with_

    # Set income thresholds based on the 95th percentile of MonthlyIncome for each
    thresholds_by_designation = df.groupby('Designation')['MonthlyIncome'].quantile

    # Replace outliers based on IQR for each designation and occupation
    for (designation, occupation), group in df.groupby(['Designation', 'Occupation'
        # Use the threshold corresponding to the designation
        income_threshold = thresholds_by_designation.get(designation, 0)

        Q1 = group['MonthlyIncome'].quantile(0.25)
        Q3 = group['MonthlyIncome'].quantile(0.75)
        IQR = Q3 - Q1

        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Replace outliers with the threshold value
        df.loc[(df['Designation'] == designation) & (df['Occupation'] == occupation
               ((df['MonthlyIncome'] < lower_bound) | (df['MonthlyIncome'] > upper_
               'MonthlyIncome'] = income_threshold
    return df

X_train = handle_outliers(X_train)
```

Encoding

PlanPitched - Ordinal encoding

In [17]:
```
X_train['PlanPitched'].unique()
```

Out[17]:
```
array(['Basic', 'Deluxe', 'Super Deluxe', 'Standard', 'King'],
      dtype=object)
```

In [18]:
```
def encode_PlanPitched(df):
    # Define the custom order
    custom_order_PlanPitched = ['Basic', 'Standard', 'Deluxe', 'Super Deluxe', 'Kin

    # Initialize the OrdinalEncoder with the custom order
    encoder = OrdinalEncoder(categories=[custom_order_PlanPitched])

    # Fit and transform the labels
    df['PlanPitched'] = encoder.fit_transform(df['PlanPitched'].values.reshape(-1,1

    return df

# Encoding on Train
X_train = encode_PlanPitched(X_train)

# Encoding on Test
X_test = encode_PlanPitched(X_test)
```

Encode other categorical features

In [19]:
```
def encode_features(df):
    columns_to_encode = ['Gender', 'TypeofContact', 'Occupation', 'MaritalStatus',
    df = pd.get_dummies(df, columns=columns_to_encode)
    return df

# Encoding on Train
X_train = encode_features(X_train)

# Encoding on Test
X_test = encode_features(X_test)
```

Scale data

In [20]:
```
def scale_data(df):
    scaler = StandardScaler()
    df = scaler.fit_transform(df)
    return df

X_train = scale_data(X_train)
X_test = scale_data(X_test)
```

Helpers

Helper to display model metrics and AUC under ROC curve

In [21]:
```
def show_auc_under_roc(y_test, y_pred, y_prob):

    # Compute the false positive rate, true positive rate, and thresholds
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)

    accuracy = accuracy_score(y_test, y_pred)
    auc_score = auc(fpr, tpr)

    print("Accuracy: ", accuracy)
    print("Precision:", precision_score(y_test, y_pred, average="weighted"))
    print("Recall:", recall_score(y_test, y_pred, average="weighted"))

    plt.figure()
    plt.plot(fpr, tpr, label='ROC curve (AUC = %0.2f)' % auc_score)
    plt.plot([0, 1], [0, 1], 'k--')
    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')
    plt.legend(loc='lower right')
    plt.show()
```

Helper to display confusion matrix

In [22]:
```
def show_confusion_matrix(y_test, y_pred):
    cm = confusion_matrix(y_test, y_pred)

    # Create a heatmap of the confusion matrix
```

```
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")

    # Add labels, title, and ticks to the plot
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.title("Confusion Matrix for PlanTaken")
    plt.xticks(ticks=[0, 1])
    plt.yticks(ticks=[0, 1])

    # Show the plot
    plt.show()
```

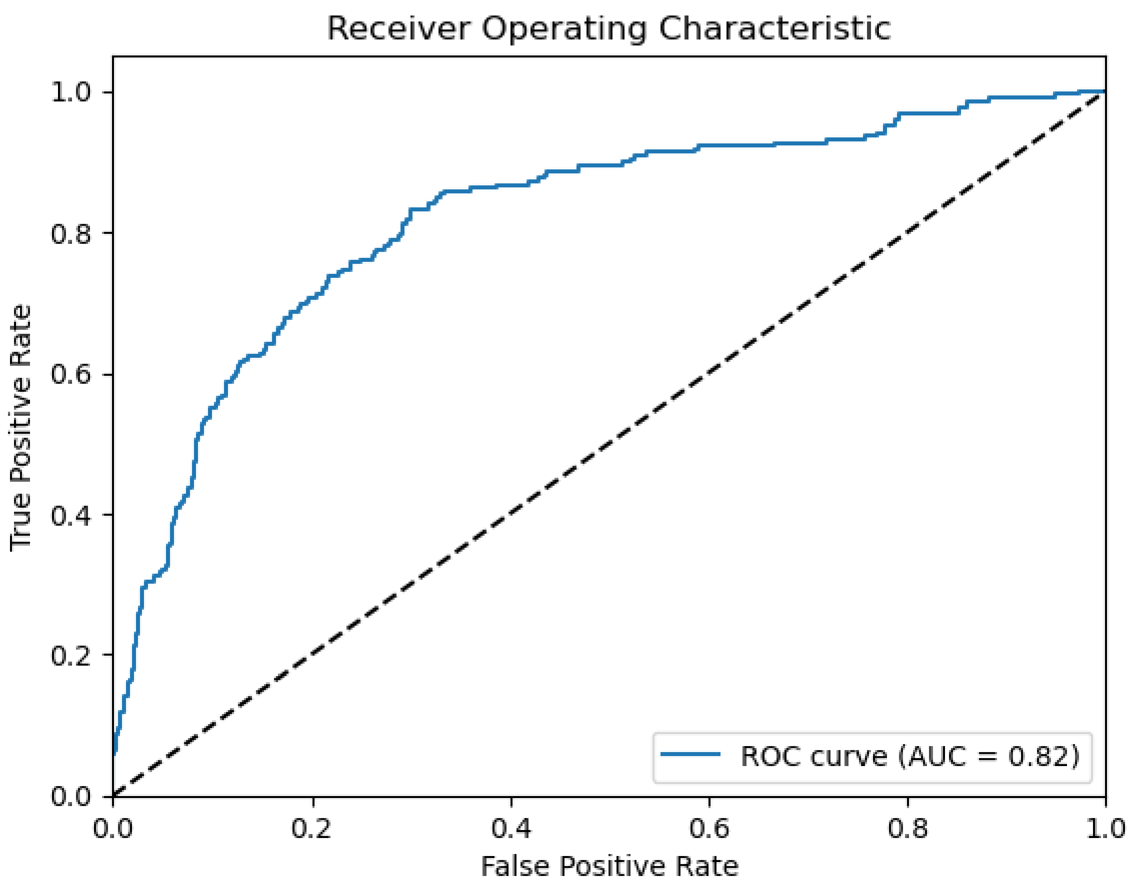
Logistic Regression

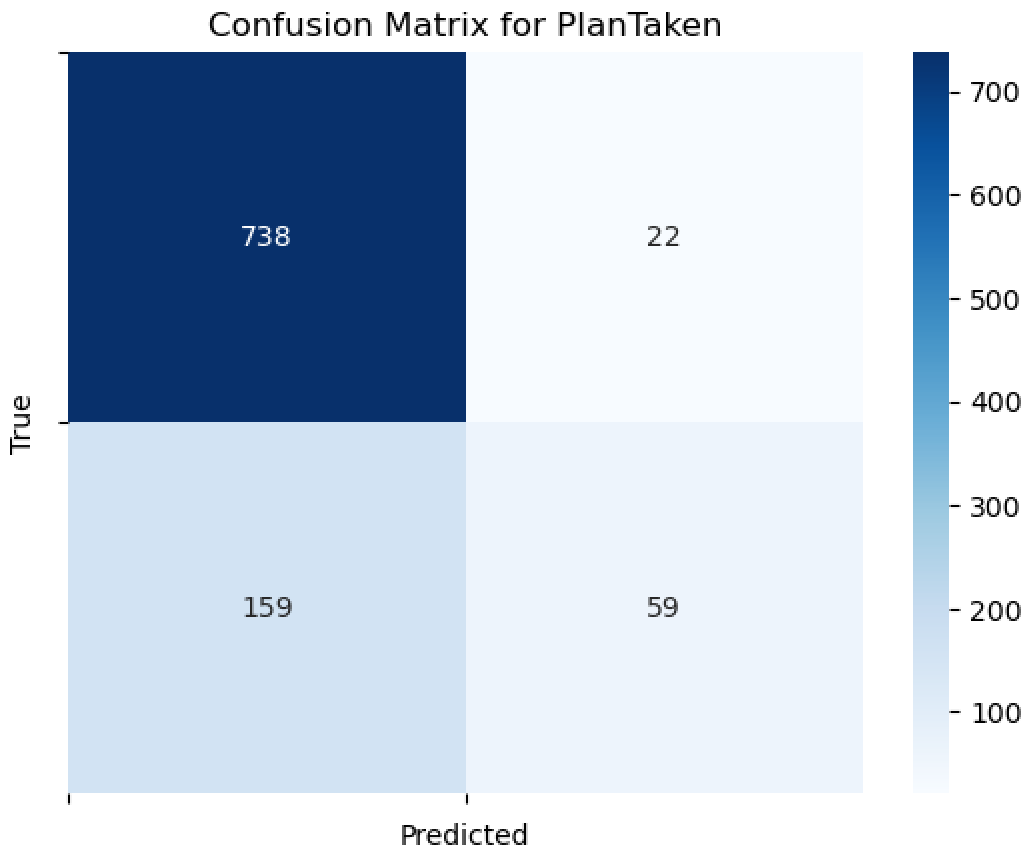
Round 1 - Logistic Regression

In [23]:

```
def build_logistic_regression_model(X_train, X_test, y_train, y_test):
    lr_model = LogisticRegression()
    lr_model.fit(X_train, y_train)

    y_pred = lr_model.predict(X_test)
    y_prob = lr_model.predict_proba(X_test)[:, 1]

    show_auc_under_roc(y_test, y_pred, y_prob)
    show_confusion_matrix(y_test, y_pred)


build_logistic_regression_model(X_train, X_test, y_train, y_test)
```

```
Accuracy:  0.8149284253578732
Precision: 0.8017120699317434
Recall: 0.8149284253578732
```



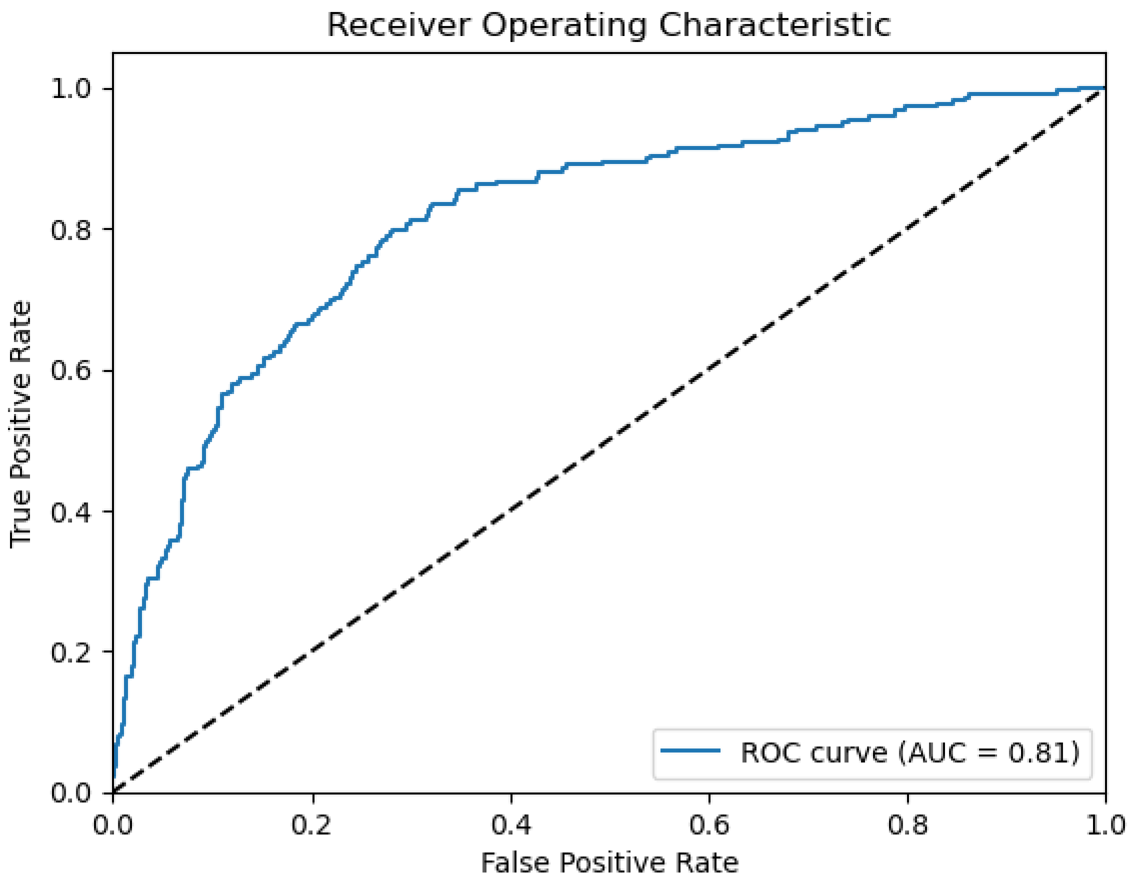


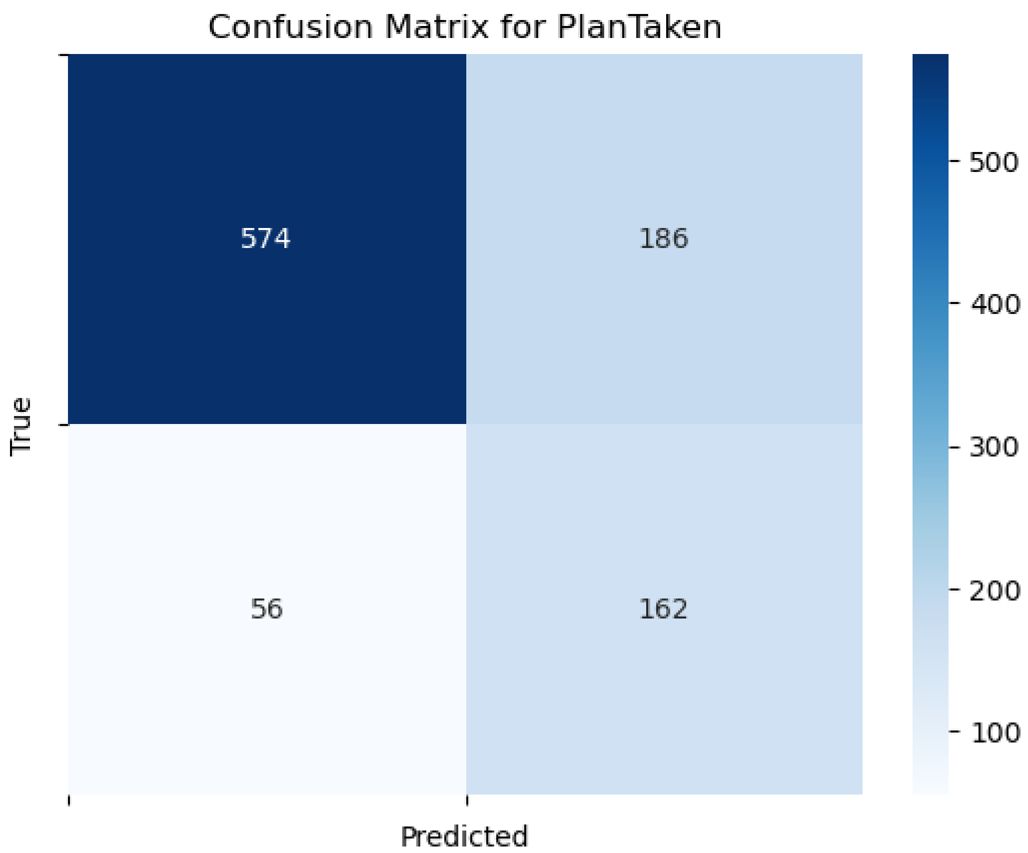
Round 2 - Logistic Regression with Regularization & SMOTE sampling

In [24]:

```
def build_logistic_regression_model_with_regularization(X_train, X_test, y_train, y
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    lr_model = LogisticRegression(penalty="l2", C=12)
    lr_model.fit(X_train_resampled, y_train_resampled)

    y_pred = lr_model.predict(X_test)
    y_prob = lr_model.predict_proba(X_test)[:, 1]

    show_auc_under_roc(y_test, y_pred, y_prob)
    show_confusion_matrix(y_test, y_pred)

build_logistic_regression_model_with_regularization(X_train, X_test, y_train, y_tes
```

```
Accuracy:  0.7525562372188139
Precision: 0.8117865062015687
Recall: 0.7525562372188139
```



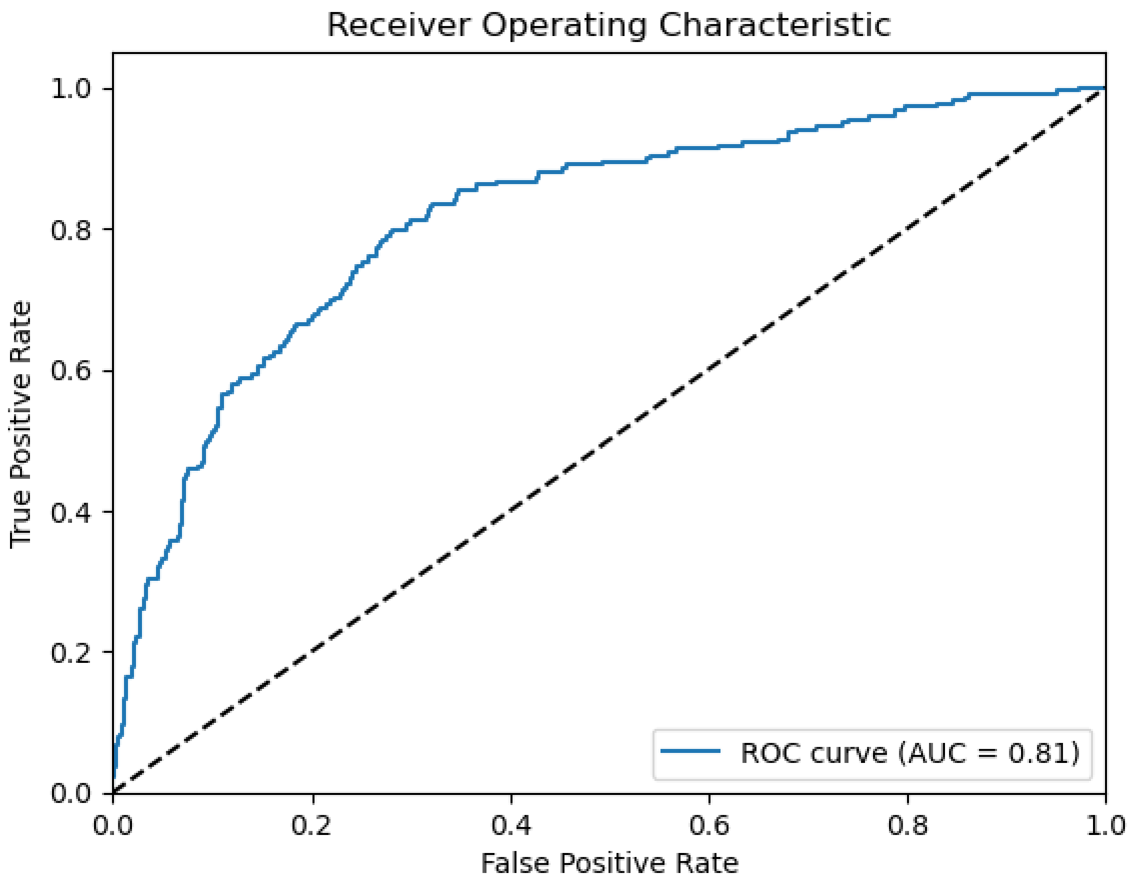


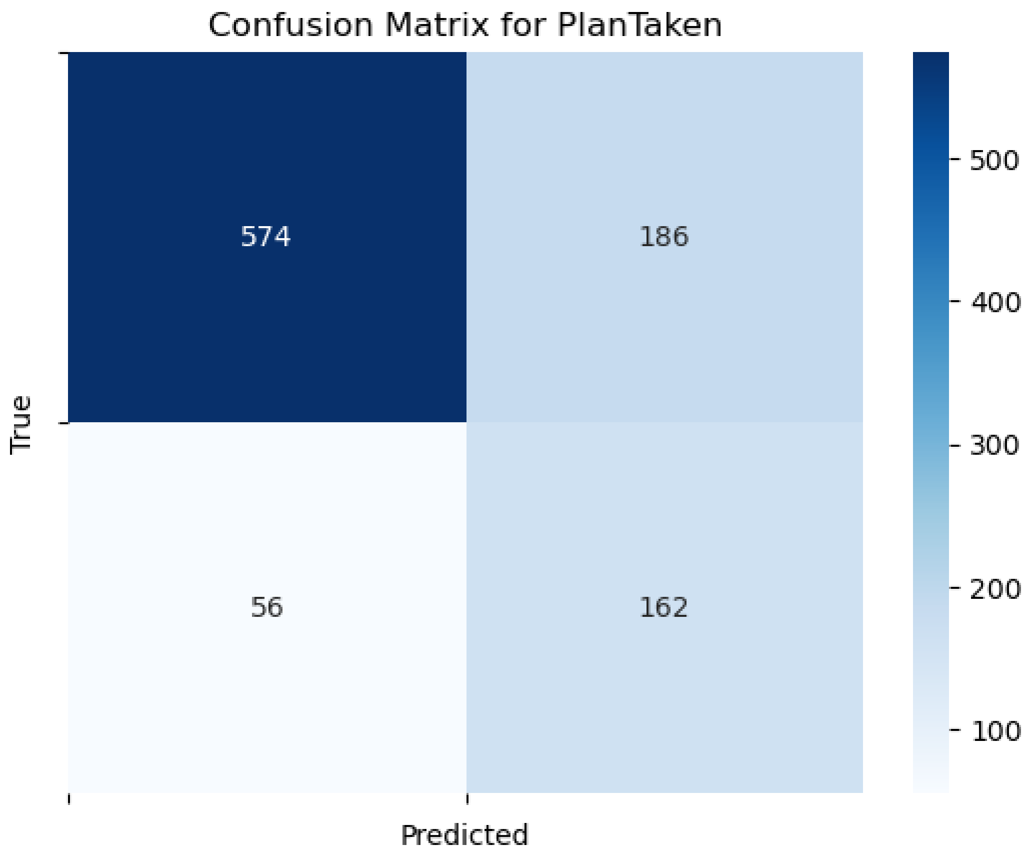
Round 3 - Logistic Regression with Regularization & hyperparameter tuning & solver & SMOTE sampling

In [25]:
```
def build_logistic_regression_model_with_regularization_and_tuning(X_train, X_test,
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

```
    lr_model = LogisticRegression(penalty="l2", C=13, solver="lbfgs")
    lr_model.fit(X_train_resampled, y_train_resampled)

    y_pred = lr_model.predict(X_test)
    y_prob = lr_model.predict_proba(X_test)[:, 1]

    show_auc_under_roc(y_test, y_pred, y_prob)
    show_confusion_matrix(y_test, y_pred)

build_logistic_regression_model_with_regularization_and_tuning(X_train, X_test, y_t
```

```
Accuracy:  0.7525562372188139
Precision: 0.8117865062015687
Recall: 0.7525562372188139
```



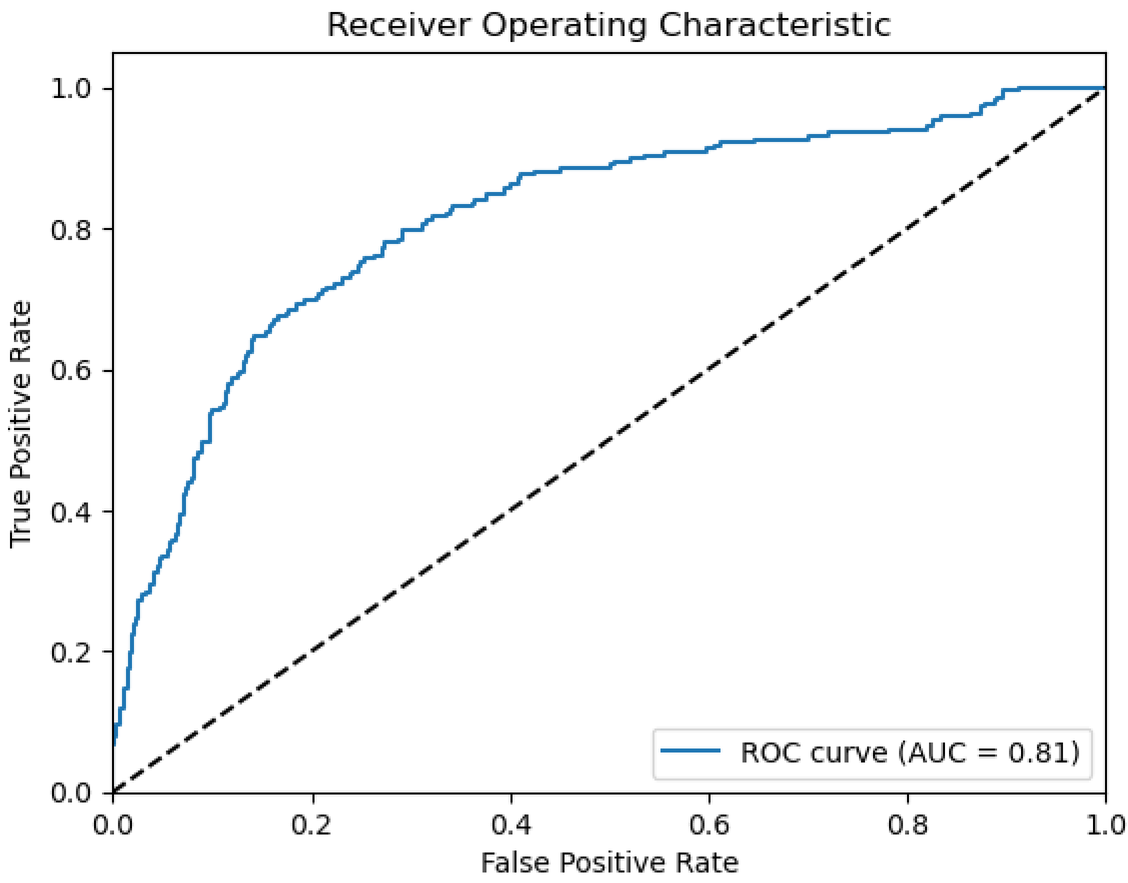


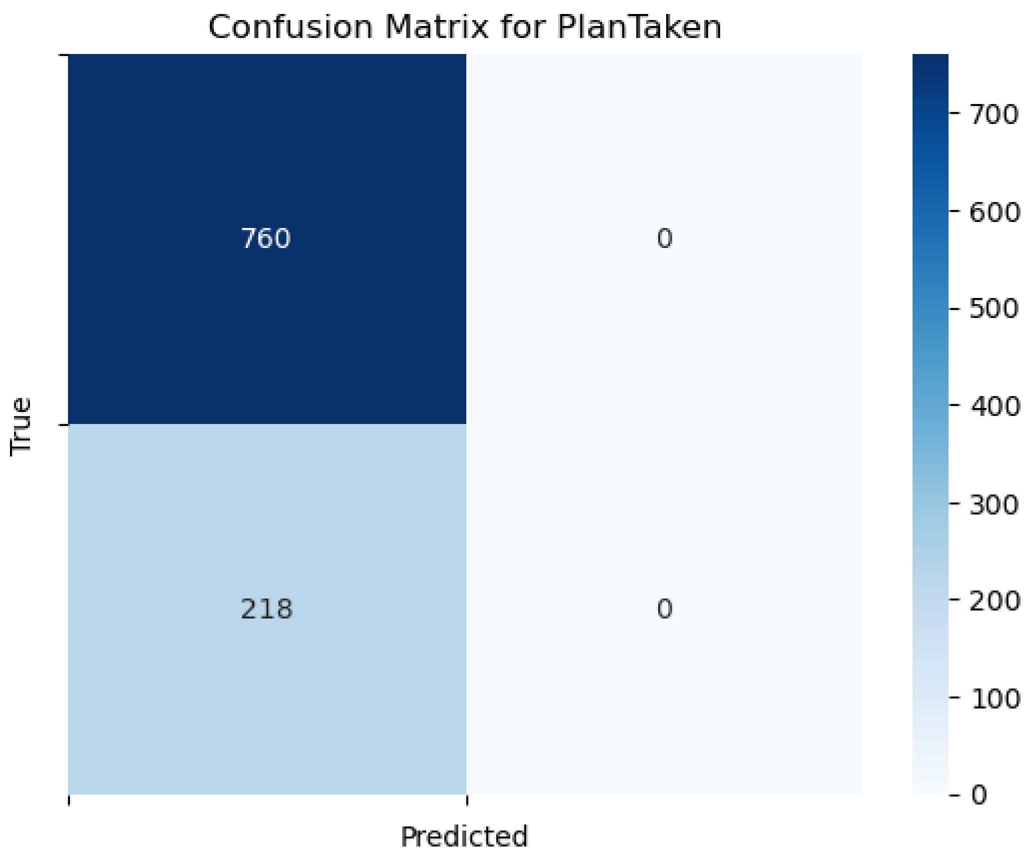
SVM

Round 1 - SVM

In [26]:
```
def build_svm_model(X_train, X_test, y_train, y_test):
    svm_model = SVC(kernel='linear', probability=True)
    svm_model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = svm_model.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)


    y_prob = svm_model.predict_proba(X_test)[:, 1]

    show_auc_under_roc(y_test, y_pred, y_prob)
    show_confusion_matrix(y_test, y_pred)


build_svm_model(X_train, X_test, y_train, y_test)
```

```
Accuracy:  0.7770961145194274
Precision: 0.603878371201191
Recall: 0.7770961145194274
```





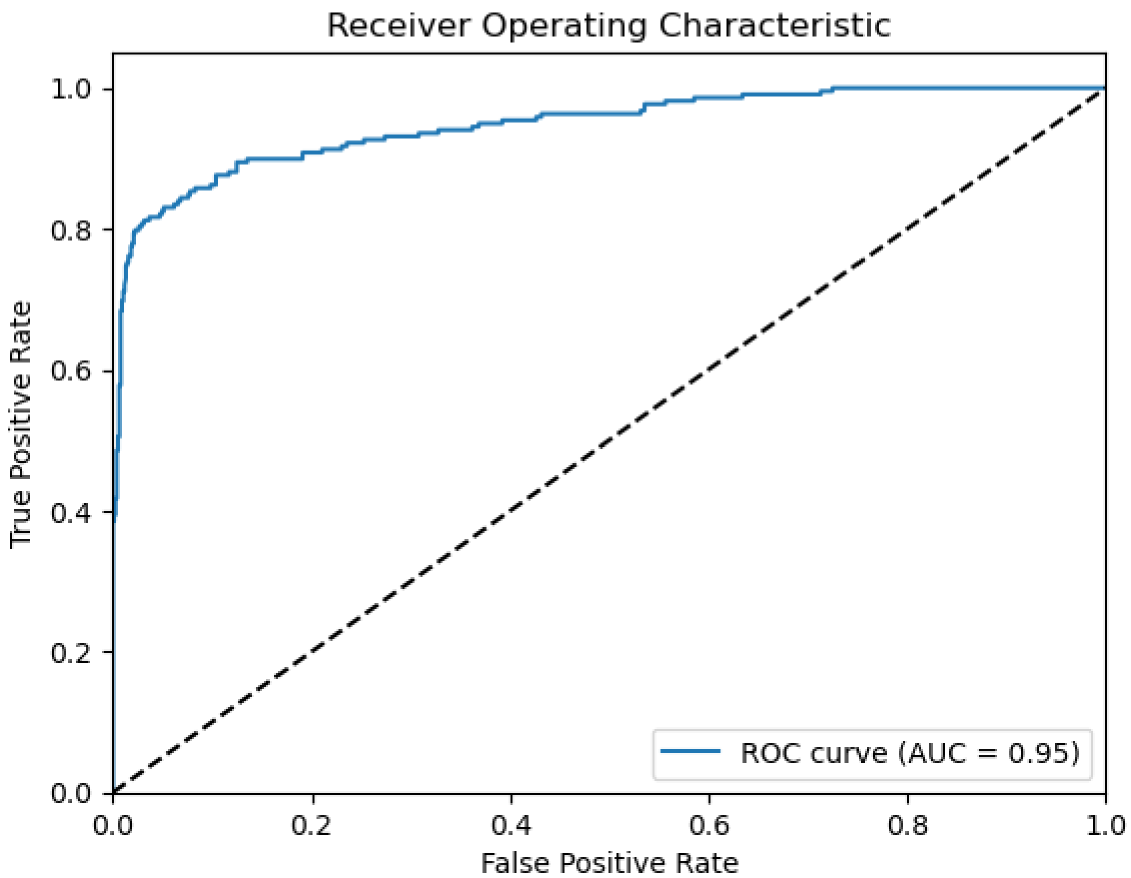
Conclusion - From the 0 values for false positive and true positive we see that the model is not learning using the "Linear" kernel

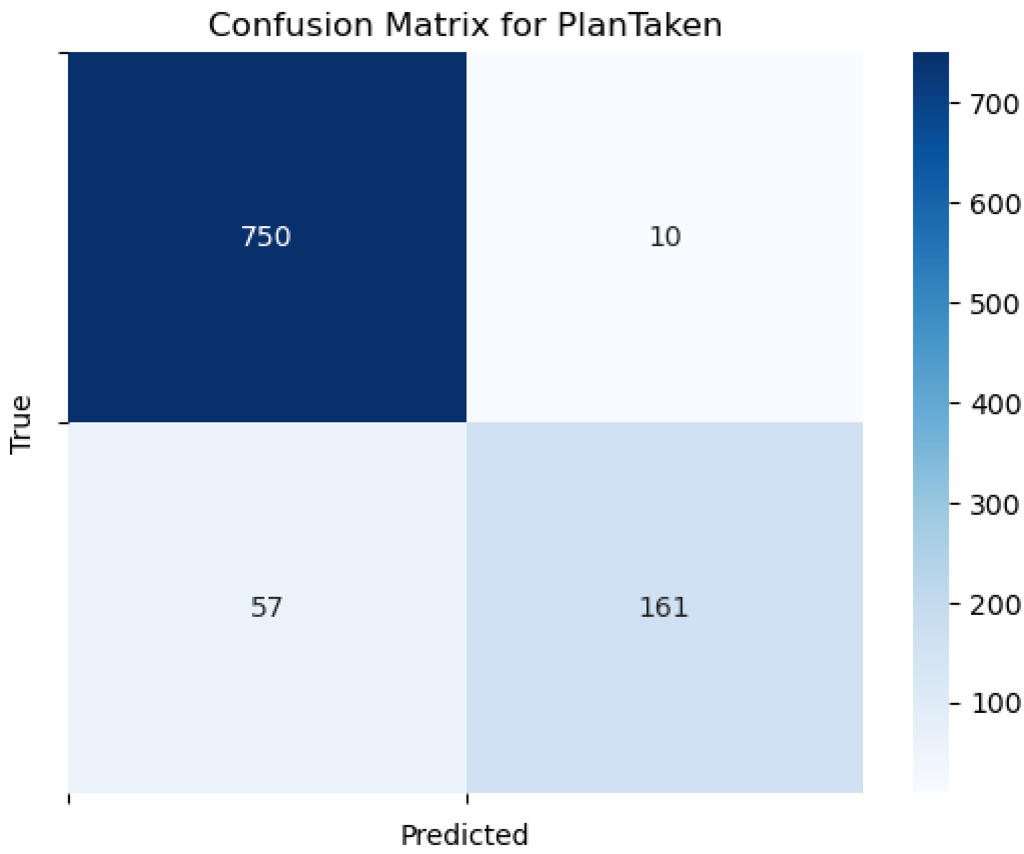
Round 2 - SVM with Regulariztion & `rbf` kernel

In [27]:

```python
def build_svm_model_with_regulariation_and_rbf(X_train, X_test, y_train, y_test):
    svm_model = SVC(kernel='rbf', C = 13.0, probability=True)
    svm_model.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred = svm_model.predict(X_test)
    y_prob = svm_model.predict_proba(X_test)[:, 1]

    show_auc_under_roc(y_test, y_pred, y_prob)
    show_confusion_matrix(y_test, y_pred)

build_svm_model_with_regulariation_and_rbf(X_train, X_test, y_train, y_test)
```

```
Accuracy:  0.9314928425357873
Precision: 0.9320768554095445
Recall: 0.9314928425357873
```



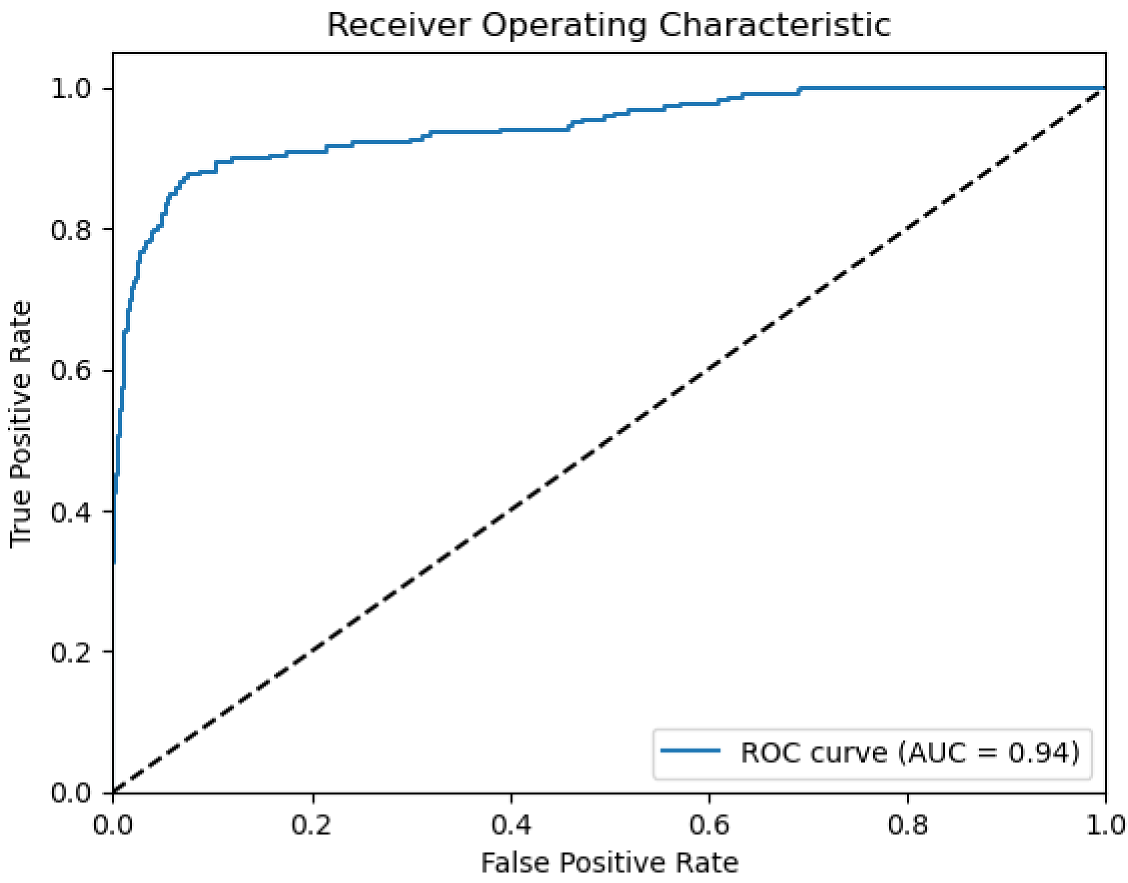


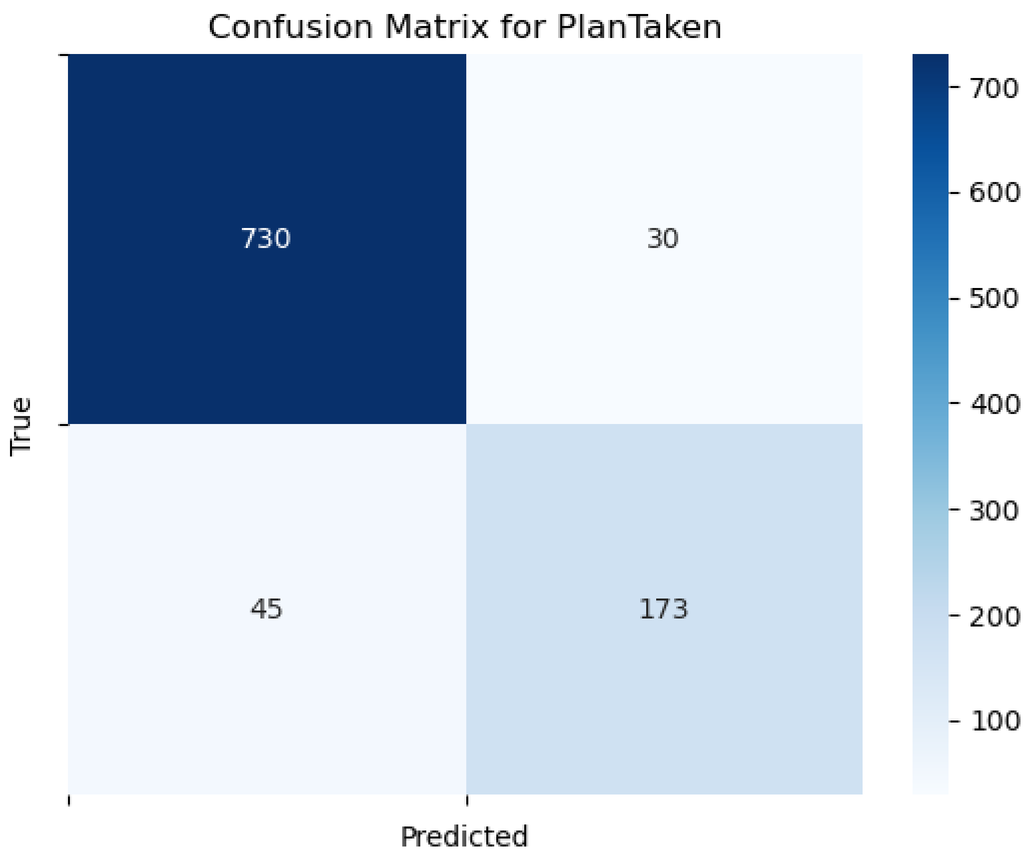
Round 3 - SVM with `rbf` kernel & Regularization & smote sampling

In [28]:

```
def build_svm_model_with_regulariation_and_rbf_and_smote(X_train, X_test, y_train,
    smote = SMOTE(random_state=42)
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

    svm_model = SVC(kernel='rbf', C=13.0, probability=True)
    svm_model.fit(X_train_resampled, y_train_resampled)

    # Make predictions on the test set
    y_pred = svm_model.predict(X_test)
    y_prob = svm_model.predict_proba(X_test)[:, 1]

    show_auc_under_roc(y_test, y_pred, y_prob)
    show_confusion_matrix(y_test, y_pred)

build_svm_model_with_regulariation_and_rbf_and_smote(X_train, X_test, y_train, y_te
```

```
Accuracy:  0.9233128834355828
Precision: 0.9219368292162193
Recall: 0.9233128834355828
```





In [29]:
```
X_train.shape
```

Out[29]:
```
(3908, 29)
```

Comparison between Logistic regression and Support vector machine.

- Time complexity

	Logistic_regression	SVM
Training time complexity	O(Iterations*m*n) M = size of training dataset n = number of feature Iterations = 3 m = 3908 n = 29 O(3*3908*29) = O(339996)	O(m^2*n) iteration = 3 m = 3908 n = 29 O(442901456)
Predictive time complexity	O(n) where n = number of feature O(29)	O(n) where n = number of feature O(29)

- If you compare Logistic regression and svm. Logistic regression takes less execution time than SVM.So, in case of execution time, logistic regression better than support vector machine .

- In case of multiple features classification support vector machine is better over logistic regression. SVM perform better in multiple dimensions (with respect to planes) over multiple features.

- Logistic regression accuracy = 0.75

- SVM accuracy = 0.92

- From above accuracy we can conclude two things SVM performed well with outliers and svm perform well with multiple features classification

- In case of Training time complexity(execution time) Logistic regression is better.

- In case of Predictive time complexity approx both perform same.