

# 1 # XG\_Boost

In [1]:

```

1 #importing all library
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 from sklearn.preprocessing import LabelEncoder
6 from sklearn.compose import ColumnTransformer
7 from sklearn.preprocessing import OneHotEncoder
8 from sklearn.metrics import confusion_matrix
9 from sklearn.metrics import accuracy_score
10 from sklearn.metrics import roc_auc_score, roc_curve
11

```

In [2]:

```

1 df = pd.read_csv("/Users/myyntiimac/Desktop/Churn_Modelling.csv")
2 df.head()

```

Out[2]:

Number	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsAct
1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	



In [3]:

```
1 df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   RowNumber              10000 non-null  int64  
1   CustomerId             10000 non-null  int64  
2   Surname                 10000 non-null  object  
3   CreditScore             10000 non-null  int64  
4   Geography               10000 non-null  object  
5   Gender                  10000 non-null  object  
6   Age                     10000 non-null  int64  
7   Tenure                  10000 non-null  int64  
8   Balance                 10000 non-null  float64 
9   NumOfProducts           10000 non-null  int64  
10  HasCrCard               10000 non-null  int64  
11  IsActiveMember          10000 non-null  int64  
12  EstimatedSalary         10000 non-null  float64 
13  Exited                  10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB

```

In [5]:

```

1 X = df.iloc[:, 3:-1].values
2

```

In [7]:

```
1 X
```

Out[7]:

```
array([[619, 'France', 'Female', ..., 1, 1, 101348.88],
       [608, 'Spain', 'Female', ..., 0, 1, 112542.58],
       [502, 'France', 'Female', ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 'Female', ..., 0, 1, 42085.58],
       [772, 'Germany', 'Male', ..., 1, 0, 92888.52],
       [792, 'France', 'Female', ..., 1, 0, 38190.78]], dtype=object)
```

In [9]:

```
1 y = df.iloc[:, -1].values
2 y
```

Out[9]:

```
array([1, 0, 1, ..., 1, 1, 0])
```

In [10]:

```
1 #converting gender column into numerical
2 le = LabelEncoder()
3 X[:, 2] = le.fit_transform(X[:, 2])
```

In [11]:

```
1 X
```

Out[11]:

```
array([[619, 'France', 0, ..., 1, 1, 101348.88],
       [608, 'Spain', 0, ..., 0, 1, 112542.58],
       [502, 'France', 0, ..., 1, 0, 113931.57],
       ...,
       [709, 'France', 0, ..., 0, 1, 42085.58],
       [772, 'Germany', 1, ..., 1, 0, 92888.52],
       [792, 'France', 0, ..., 1, 0, 38190.78]], dtype=object)
```



In [12]:

```
1 # Assuming 'X' is your input array and 'column_index' is the index of the column you want to one-hot
2 column_index = 1 # Example column index
3
4 # Convert the array to a DataFrame
5 df = pd.DataFrame(X)
6
7 # Perform one-hot encoding using get_dummies
8 encoded_df = pd.get_dummies(df, columns=[column_index], drop_first=True)
9
10 # Extract the values from the encoded DataFrame
11 X_encoded = encoded_df.values
```

In [13]:

```
1 X_encoded
```

Out[13]:

```
array([[619, 0, 42, ..., 101348.88, 0, 0],
       [608, 0, 41, ..., 112542.58, 0, 1],
       [502, 0, 42, ..., 113931.57, 0, 0],
       ...,
       [709, 0, 36, ..., 42085.58, 0, 0],
       [772, 1, 42, ..., 92888.52, 1, 0],
       [792, 0, 28, ..., 38190.78, 0, 0]], dtype=object)
```

In [14]:

```
1 #Split the dataset
2 from sklearn.model_selection import train_test_split
3 X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size = 0.2, random_state = 0
```

In [15]:

```
1 from xgboost import XGBClassifier
2 classifier = XGBClassifier()
```

In [16]:

```
1 classifier.fit(X_train, y_train)
```

Out[16]:

```
▼
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,
               gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
               interaction_constraints=None, learning_rate=None, max_bin=None,
               max_cat_threshold=None, max_cat_to_onehot=None,
               max_delta_step=None, max_depth=None, max_leaves=None,
               min_child_weight=None, missing=nan, monotone_constraints=None,
               n_estimators=100, n_jobs=None, num_parallel_tree=None,
```

In [17]:

```
1 y_pred = classifier.predict(X_test)
```



In [18]:

```
1 y_pred
```

Out[18]:

```
array([0, 0, 0, ..., 0, 0, 0])
```

In [19]:

```
1
2 cm = confusion_matrix(y_test, y_pred)
3 print(cm)
```

```
[[1491  104]
 [ 187  218]]
```

In [20]:

```
1 ac = accuracy_score(y_test, y_pred)
2 print(ac)
```

```
0.8545
```

In [21]:

```
1 bias = classifier.score(X_train, y_train)
2 bias
```

Out[21]:

```
0.95775
```

In [22]:

```
1 Variance = classifier.score(X_test,y_test)
2 Variance
```

Out[22]:

0.8545

In [23]:

```
1 #ROC AND AUC
2 from sklearn.metrics import roc_curve, roc_auc_score
```

In [24]:

```
1 fpr, tpr, thresholds = roc_curve(y_test, y_pred)
2 fpr, tpr, thresholds
```

Out[24]:

```
(array([0.          , 0.06520376, 1.          ]),
 array([0.          , 0.5382716, 1.          ]),
 array([2, 1, 0]))
```

In [25]:

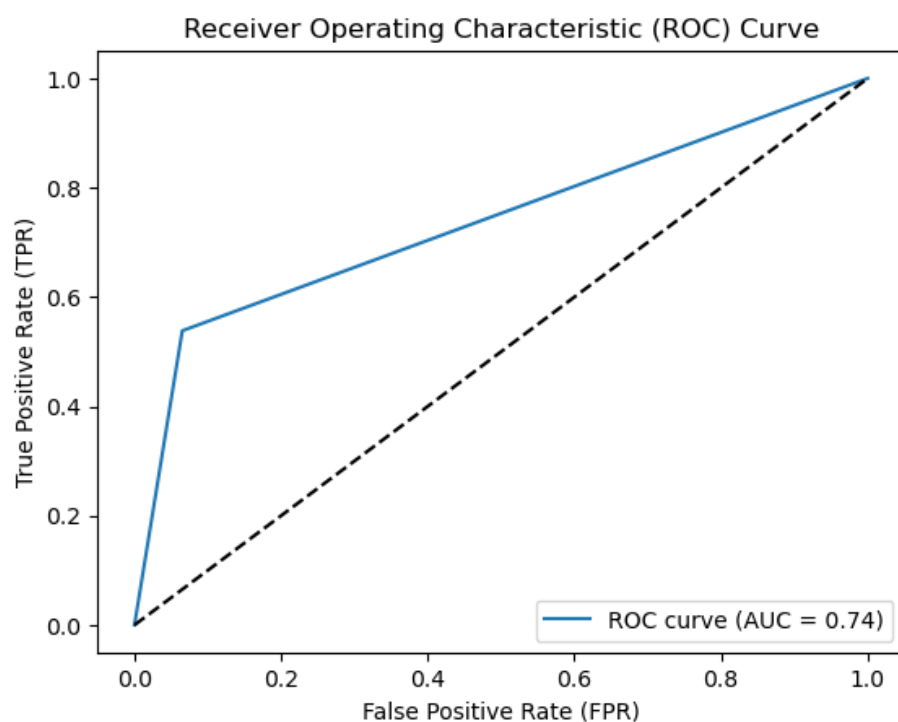
```
1 auc = roc_auc_score(y_test, y_pred)
2 auc
```

Out[25]:

0.7365339215913929

In [26]:

```
1 # Plotting the ROC curve
2 plt.plot(fpr, tpr, label='ROC curve (AUC = {:.2f})'.format(auc))
3 plt.plot([0, 1], [0, 1], 'k--') # Random guess line
4 plt.xlabel('False Positive Rate (FPR)')
5 plt.ylabel('True Positive Rate (TPR)')
6 plt.title('Receiver Operating Characteristic (ROC) Curve')
7 plt.legend(loc='lower right')
8 plt.show()
```



```
1 Insight: In the case of  $AUC = 0.75$ , the model demonstrates reasonable discriminative ability, but there is still room for improvement. It correctly ranks 75% of the positive samples higher than the negative samples, on average, across different classification thresholds. However, it might misclassify some instances, leading to false positives or false negatives.
```

