DONERS PREDICTION

In [1]:
```python
## Author : Saurabh Kumar
## date : 1st-oct
```

In [2]:
```python
pwd
```

Out[2]:
```
'E:\\DataScience\\MachineLearning\\Donors Prediction'
```

In [3]:
```python
path ='E:\\DataScience\\MachineLearning\\Donors Prediction'
```

In [4]:
```python
import os
os.listdir()
```

Out[4]:
```
['.ipynb_checkpoints',
 'DONERS PREDICTION.ipynb',
 'Donors Prediction.zip',
 'Predict_donor.csv',
 'Raw_Data_for_train_test.csv']
```

In [5]:
```python
import pandas as pd
df = pd.read_csv(path+'/Raw_Data_for_train_test.csv')
df.head()
```

Out[5]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	HOME_OW
0	0	NaN	5	101	87.0	0	?	?	.	
1	1	10.0	12	137	79.0	0	R	2	45	
2	0	NaN	37	113	75.0	0	S	1	11	
3	0	NaN	38	92	NaN	0	U	2	4	
4	0	NaN	41	101	74.0	0	R	2	49	

5 rows × 50 columns

In [6]:
```python
df.columns[df.isnull().any()]
```

Out[6]:
```
Index(['TARGET_D', 'DONOR_AGE', 'INCOME_GROUP', 'WEALTH_RATING',
       'MONTHS_SINCE_LAST_PROM_RESP'],
      dtype='object')
```

In [7]:
```python
# Fill numeric rows with the median
for label, content in df.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            # Fill missing numeric values with median since it's more robust than the mean
            df[label] = content.fillna(content.median())

df.columns[df.isnull().any()]
```

Out[7]:
```
Index([], dtype='object')
```

In [8]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19372 entries, 0 to 19371
Data columns (total 50 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   TARGET_B                     19372 non-null  int64  
 1   TARGET_D                     19372 non-null  float64
 2   CONTROL_NUMBER               19372 non-null  int64  
 3   MONTHS_SINCE_ORIGIN          19372 non-null  int64  
 4   DONOR_AGE                    19372 non-null  float64
 5   IN_HOUSE                     19372 non-null  int64  
 6   URBANICITY                   19372 non-null  object 
 7   SES                          19372 non-null  object 
 8   CLUSTER_CODE                 19372 non-null  object 
 9   HOME_OWNER                   19372 non-null  object 
 10  DONOR_GENDER                 19372 non-null  object 
 11  INCOME_GROUP                 19372 non-null  float64
 12  PUBLISHED_PHONE              19372 non-null  int64  
 13  OVERLAY_SOURCE               19372 non-null  object 
 14  MOR_HIT_RATE                 19372 non-null  int64  
 15  WEALTH_RATING                19372 non-null  float64
 16  MEDIAN_HOME_VALUE            19372 non-null  int64  
 17  MEDIAN_HOUSEHOLD_INCOME      19372 non-null  int64  
 18  PCT_OWNER_OCCUPIED           19372 non-null  int64  
 19  PER_CAPITA_INCOME            19372 non-null  int64  
 20  PCT_ATTRIBUTE1               19372 non-null  int64  
 21  PCT_ATTRIBUTE2               19372 non-null  int64  
 22  PCT_ATTRIBUTE3               19372 non-null  int64  
 23  PCT_ATTRIBUTE4               19372 non-null  int64  
 24  PEP_STAR                     19372 non-null  int64  
 25  RECENT_STAR_STATUS           19372 non-null  int64  
 26  RECENCY_STATUS_96NK          19372 non-null  object 
 27  FREQUENCY_STATUS_97NK        19372 non-null  int64  
 28  RECENT_RESPONSE_PROP         19372 non-null  float64
 29  RECENT_AVG_GIFT_AMT          19372 non-null  float64
 30  RECENT_CARD_RESPONSE_PROP    19372 non-null  float64
 31  RECENT_AVG_CARD_GIFT_AMT     19372 non-null  float64
 32  RECENT_RESPONSE_COUNT        19372 non-null  int64  
 33  RECENT_CARD_RESPONSE_COUNT   19372 non-null  int64  
 34  MONTHS_SINCE_LAST_PROM_RESP  19372 non-null  float64
 35  LIFETIME_CARD_PROM           19372 non-null  int64  
 36  LIFETIME_PROM                19372 non-null  int64  
 37  LIFETIME_GIFT_AMOUNT         19372 non-null  float64
 38  LIFETIME_GIFT_COUNT          19372 non-null  int64  
 39  LIFETIME_AVG_GIFT_AMT        19372 non-null  float64
 40  LIFETIME_GIFT_RANGE          19372 non-null  float64
 41  LIFETIME_MAX_GIFT_AMT        19372 non-null  float64
 42  LIFETIME_MIN_GIFT_AMT        19372 non-null  float64
 43  LAST_GIFT_AMT                19372 non-null  float64
 44  CARD_PROM_12                 19372 non-null  int64  
 45  NUMBER_PROM_12               19372 non-null  int64  
 46  MONTHS_SINCE_LAST_GIFT       19372 non-null  int64  
 47  MONTHS_SINCE_FIRST_GIFT      19372 non-null  int64  
 48  FILE_AVG_GIFT                19372 non-null  float64
 49  FILE_CARD_GIFT               19372 non-null  int64  
dtypes: float64(16), int64(27), object(7)
memory usage: 7.4+ MB
```

In [9]:
```python
# Turn categorical variables into numbers
for label, content in df.items():
    # Check columns which aren't numeric
    if not pd.api.types.is_numeric_dtype(content):
        # print the columns that are objectt type
        print(label)
        df[label] = pd.Categorical(content).codes+1
```

```
URBANICITY
SES
CLUSTER_CODE
HOME_OWNER
DONOR_GENDER
OVERLAY_SOURCE
RECENCY_STATUS_96NK
```

In [10]:
```python
# Cleaned data
df.head()
```

Out[10]:

	TARGET_B	TARGET_D	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	HOME_OW
0	0	13.0	5	101	87.0	0	1	5	1	
1	1	10.0	12	137	79.0	0	3	2	41	
2	0	13.0	37	113	75.0	0	4	1	4	
3	0	13.0	38	92	60.0	0	6	2	35	
4	0	13.0	41	101	74.0	0	3	2	45	

5 rows × 50 columns

In [11]:
```python
# There's no need of Target_D column. As we are taking TARGET_B as our target variable. So we can drop this
df = df.drop('TARGET_D', axis=1)
df.head()
```

Out[11]:

	TARGET_B	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	HOME_OWNER	DONC
0	0	5	101	87.0	0	1	5	1	1	
1	1	12	137	79.0	0	3	2	41	1	
2	0	37	113	75.0	0	4	1	4	1	
3	0	38	92	60.0	0	6	2	35	1	
4	0	41	101	74.0	0	3	2	45	2	

5 rows × 49 columns

In [12]:
```python
# input features
x = df.drop('TARGET_B', axis=1)

# Target variable
y = df['TARGET_B']

x.head()
```

Out[12]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	HOME_OWNER	DONOR_GENDER
0	5	101	87.0	0	1	5	1	1	3
1	12	137	79.0	0	3	2	41	1	3
2	37	113	75.0	0	4	1	4	1	2
3	38	92	60.0	0	6	2	35	1	2
4	41	101	74.0	0	3	2	45	2	2

5 rows × 48 columns

In [13]:
```python
y.head()
```

Out[13]:
```
0    0
1    1
2    0
3    0
4    0
Name: TARGET_B, dtype: int64
```

In [14]:
```python
# Import standard scaler
from sklearn.preprocessing import StandardScaler
ss = StandardScaler()

# apply scaler
x = ss.fit_transform(x)
x
```

Out[14]:
```
array([[-1.7292245 ,  0.66877603,  1.9225452 , ...,  0.59940354,
        -0.49710644,  0.37474069],
       [-1.72909912,  1.5414079 ,  1.36956114, ...,  1.39797251,
         0.21185264,  1.46005894],
       [-1.72865132,  0.95965332,  1.09306911, ...,  0.9454501 ,
         0.44286178,  2.32831353],
       ...,
       [ 1.70519674, -1.07648773,  0.05622399, ..., -1.2373051 ,
         0.24371597, -1.14470485],
       [ 1.70571618,  1.34748971,  1.30043813, ...,  1.58430527,
         0.622662  ,  1.24299529],
       [ 1.70578783, -1.07648773,  0.74745407, ..., -1.21068613,
         1.38169203, -0.9276412 ]])
```

Modelling We'll use following models and then evaluate them to find which model works well:

1.KNN 2.Random Forest 3.XGBoost Classifier

In [15]:
```python
##KNN
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2)


from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# define and configure the model
model = KNeighborsClassifier()

# fit the model
model.fit(xtrain, ytrain)

# evaluate the model
preds = model.predict(xtest)
accuracy_score(ytest, preds)
```

Out[15]:
```
0.7029677419354838
```

In [16]:
```python
## Random Forest
from sklearn.ensemble import RandomForestClassifier

# define and configure the model
model = RandomForestClassifier()

# fit the model
model.fit(xtrain, ytrain)

# evaluate the model
preds = model.predict(xtest)
accuracy_score(ytest, preds)
```

Out[16]:
```
0.7455483870967742
```

In [17]:
```python
## XGBOOST
from xgboost import XGBClassifier

# define and configure the model
model = XGBClassifier()

# fit the model
model.fit(xtrain, ytrain)

# evaluate the model
preds = model.predict(xtest)
accuracy_score(ytest, preds)
```

```
C:\Users\Saurabh Kumar\.conda\envs\tfod\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of labe
l encoder in XGBClassifier is deprecated and will be removed in a future release. To remove this warning, do th
e following: 1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your
labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[11:06:35] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

Out[17]:
```
0.7329032258064516
```

We can see Random forest perfomed best. So let's perform hyperperameter tuning for Random forest

In [18]:
```python
import numpy as np
from sklearn.model_selection import RandomizedSearchCV

# different randomforestregressor hyperperameters
rf_grid = {'n_estimators' : np.arange(10, 100, 10),
           'max_depth': [None, 3, 5, 10],
           'min_samples_split' : np.arange(2, 20, 2),
           'min_samples_leaf': np.arange(1, 20, 2),
            'max_features' : [0.5, 1, 'sqrt', 'auto']}

# instentiate randomizedsearchcv model
rs_model= RandomizedSearchCV(RandomForestClassifier(n_jobs = -1,
                                                    random_state=42),
                                                    param_distributions = rf_grid,
                                                    n_iter = 90,
                                                    cv=5,
                                                    verbose=True)


rs_model.fit(xtrain, ytrain)
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
```

Out[18]:
```
RandomizedSearchCV(cv=5,
                   estimator=RandomForestClassifier(n_jobs=-1, random_state=42),
                   n_iter=90,
                   param_distributions={'max_depth': [None, 3, 5, 10],
                                        'max_features': [0.5, 1, 'sqrt',
                                                         'auto'],
                                        'min_samples_leaf': array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19]),
                                        'min_samples_split': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18]),
                                        'n_estimators': array([10, 20, 30, 40, 50, 60, 70, 80, 90])},
                   verbose=True)
```

In [19]:
```python
rs_model.best_params_
```

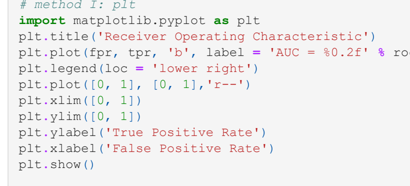
Out[19]:
```
{'n_estimators': 80,
 'min_samples_split': 12,
 'min_samples_leaf': 3,
 'max_features': 0.5,
 'max_depth': None}
```

We got the best parameters for our model. Now Let's create an ideal model that have these as it's parameters.

In [20]:
```python
ideal_model = RandomForestClassifier(n_estimators= 80,
                                     min_samples_split = 2,
                                     min_samples_leaf = 5,
                                     max_features = 'auto',
                                     max_depth = 10)

# fit the model
ideal_model.fit(xtrain, ytrain)

# evaluate the model
preds = ideal_model.predict(xtest)
accuracy_score(ytest, preds)
```

Out[20]:
```
0.7465806451612903
```

In [21]:
```python
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = ideal_model.predict_proba(xtest)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(ytest, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Now since we have a good model to predict. Let's Predict wheather a person donates or not for our Test data

In [22]:
```python
test_df = pd.read_csv(path+'/Predict_donor.csv')
test_df.head()
```

Out[22]:

	CONTROL_NUMBER	MONTHS_SINCE_ORIGIN	DONOR_AGE	IN_HOUSE	URBANICITY	SES	CLUSTER_CODE	HOME_OWNER	DONOR_GENDER
0	139	101	NaN	0	R	2	46	U	F
1	142	137	NaN	0	R	2	43	U	F
2	282	17	30.0	0	T	1	35	H	M
3	368	137	75.0	0	U	1	2	H	M
4	387	5	NaN	0	T	2	40	U	F

5 rows × 48 columns

In [23]:
```python
# Fill numeric rows with the median
for label, content in test_df.items():
    if pd.api.types.is_numeric_dtype(content):
        if pd.isnull(content).sum():
            # Fill missing numeric values with median since it's more robust than the mean
            test_df[label] = content.fillna(content.median())
```

In [24]:
```python
# Turn categorical variables into numbers
for label, content in test_df.items():
    # Check columns which aren't numeric
    if not pd.api.types.is_numeric_dtype(content):
        # print the columns that are object type
        print(label)
        test_df[label] = pd.Categorical(content).codes+1
```

```
URBANICITY
SES
CLUSTER_CODE
HOME_OWNER
DONOR_GENDER
OVERLAY_SOURCE
RECENCY_STATUS_96NK
```

In [25]:
```python
Target = ideal_model.predict(test_df)
Target
```

Out[25]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [26]:
```python
PREDICTED_df = pd.DataFrame()
PREDICTED_df['TARGET_B'] = Target
PREDICTED_df['CONTROL_NUMBER'] = test_df['CONTROL_NUMBER']
PREDICTED_df.head()
```

Out[26]:

	TARGET_B	CONTROL_NUMBER
0	0	139
1	0	142
2	0	282
3	0	368
4	0	387

In [ ]:

In [ ]: