Import Libraries

In [8]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Loads the data into a Pandas DataFrame

In [9]:
```
df=pd.read_csv("Travel.csv")
```

Display the top 5 rows of Travel.csv dataset.

In [10]:
```
df.head(5)
```

Out[10]:

	CustomerID	ProdTaken	Age	TypeofContact	CityTier	DurationOfPitch	Occupation	G
0	200000	1	41.0	Self Enquiry	3	6.0	Salaried	F
1	200001	0	49.0	Company Invited	1	14.0	Salaried	
2	200002	1	37.0	Self Enquiry	1	8.0	Free Lancer	
3	200003	0	33.0	Company Invited	1	9.0	Salaried	F
4	200004	0	NaN	Self Enquiry	1	8.0	Small Business	

Check the shape of the DataFrame

In [11]:
```
df.shape
```

Out[11]:
```
(4888, 20)
```

Displays a concise summary of the DataFrame

In [12]:
```
print(df.info())
```

```
<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   ProdTaken                 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   NumberOfPersonVisiting    4888 non-null   int64  
 9   NumberOfFollowups         4843 non-null   float64
 10  ProductPitched            4888 non-null   object 
 11  PreferredPropertyStar     4862 non-null   float64
 12  MaritalStatus             4888 non-null   object 
 13  NumberOfTrips             4748 non-null   float64
 14  Passport                  4888 non-null   int64  
 15  PitchSatisfactionScore    4888 non-null   int64  
 16  OwnCar                    4888 non-null   int64  
 17  NumberOfChildrenVisiting  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
None
```

Number of Unique Values

In [13]:
```
print(df.nunique())
```

```
CustomerID                4888
ProdTaken                    2
Age                         44
TypeofContact                2
CityTier                     3
DurationOfPitch             34
Occupation                   4
Gender                       3
NumberOfPersonVisiting       5
NumberOfFollowups            6
ProductPitched               5
PreferredPropertyStar        3
MaritalStatus                4
NumberOfTrips               12
Passport                     2
PitchSatisfactionScore       5
OwnCar                       2
NumberOfChildrenVisiting     4
Designation                  5
MonthlyIncome             2475
dtype: int64
```

Deep Copy of Original DataFrame (Safe for Transformations)

In [14]:
```
df_copy = df.copy(deep=True)
```

Dropping Columns with High Cardinality

Removed columns with excessively high cardinality (i.e., too many unique values), which can increase model complexity and reduce performance in machine learning tasks.

In [15]:
```
df.drop(["CustomerID"], axis=1, inplace=True)
```

Check Duplicates

In [16]:
```
print(df.duplicated().sum())
```

```
141
```

Drop Duplicates

In [17]:
```
df.drop_duplicates(keep='last', inplace=True)
```

Count of missing (null/NaN) values in each column

In [18]:
```
print(df.isnull().sum())
```

```
ProdTaken                   0
Age                       216
TypeofContact              25
CityTier                    0
DurationOfPitch           246
Occupation                  0
Gender                      0
NumberOfPersonVisiting      0
NumberOfFollowups          44
ProductPitched              0
PreferredPropertyStar      26
MaritalStatus               0
NumberOfTrips             138
Passport                    0
PitchSatisfactionScore      0
OwnCar                      0
NumberOfChildrenVisiting   60
Designation                 0
MonthlyIncome             224
dtype: int64
```

Median for Numeric, Skewed Columns

In [19]:
```
for col in ['Age', 'DurationOfPitch', 'NumberOfFollowups', 'NumberOfTrips', 'Monthl
    df[col].fillna(df[col].median(), inplace=True)
```

Mode for Categorical or Discrete

In [20]:
```
for col in ['TypeofContact','PreferredPropertyStar','NumberOfChildrenVisiting']:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

Check for Null Values After Filling Missing Data

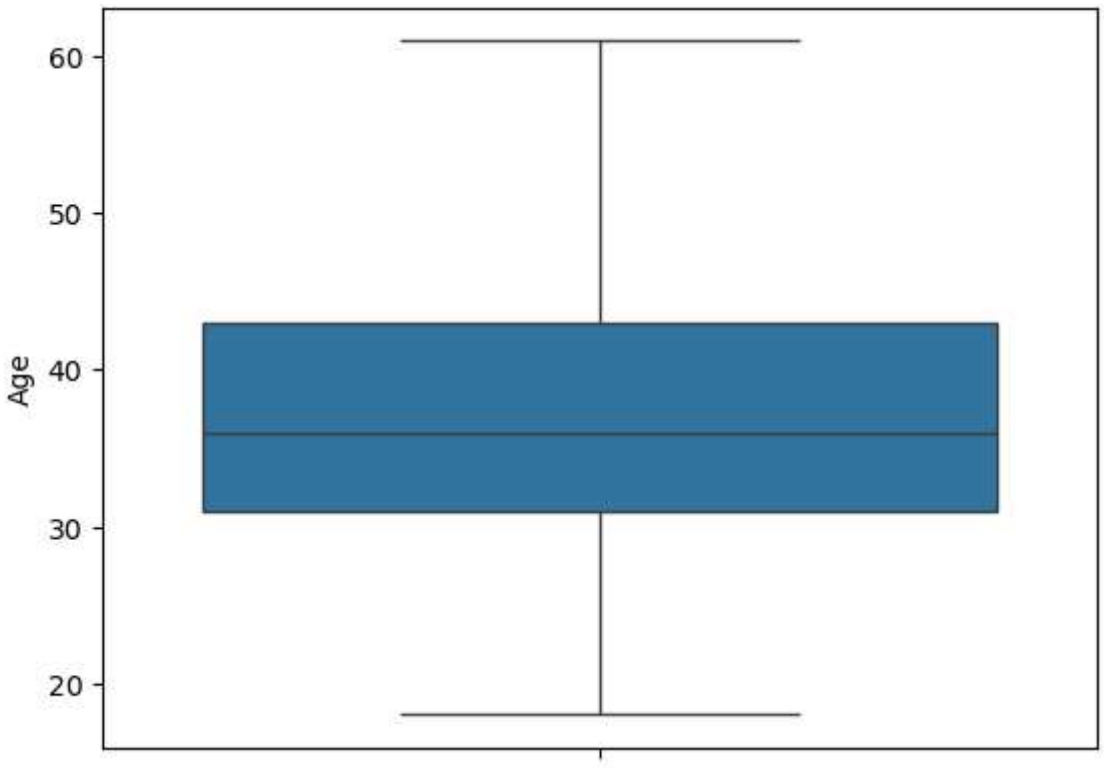
In [21]:
```
print(df.isnull().sum())
```

```
ProdTaken                 0
Age                       0
TypeofContact             0
CityTier                  0
DurationOfPitch           0
Occupation                0
Gender                    0
NumberOfPersonVisiting    0
NumberOfFollowups         0
ProductPitched            0
PreferredPropertyStar     0
MaritalStatus             0
NumberOfTrips             0
Passport                  0
PitchSatisfactionScore    0
OwnCar                    0
NumberOfChildrenVisiting  0
Designation               0
MonthlyIncome             0
dtype: int64
```

Outlier Detection in Age Column (Before Binning)

In [22]:
```
sns.boxplot(df["Age"])
```

Out[22]: <Axes: ylabel='Age'>



In [23]:
```
# Standard Age Bins
age_bins = [0, 18, 25, 35, 45, 60, 100]
```

```
age_labels = ['Teen', 'Youth', 'Young Adult', 'Adult', 'Middle Age', 'Senior']
df["AgeBin"] = pd.cut(df["Age"], bins=age_bins, labels=age_labels)
df['AgeBin'] = df['AgeBin'].astype(str)
```

Drop Age Column

In [24]:
```
df.drop(["Age"], axis=1, inplace=True)
```

Separate Categorical and Numerical Features

In [25]:
```
Categorical = df.select_dtypes(include='object')
Numerical = df.select_dtypes(exclude='object')
```

Check Unique Values in Each Categorical Column

In [26]:
```
for col in Categorical.columns :
    print(col)
    print(df[col].unique())
    print("="*100)
```

```
TypeofContact
['Self Enquiry' 'Company Invited']
================================================================================
================
Occupation
['Salaried' 'Free Lancer' 'Small Business' 'Large Business']
================================================================================
================
Gender
['Female' 'Male' 'Fe Male']
================================================================================
================
ProductPitched
['Deluxe' 'Basic' 'Standard' 'Super Deluxe' 'King']
================================================================================
================
MaritalStatus
['Single' 'Divorced' 'Married' 'Unmarried']
================================================================================
================
Designation
['Manager' 'Executive' 'Senior Manager' 'AVP' 'VP']
================================================================================
================
AgeBin
['Adult' 'Middle Age' 'Young Adult' 'Youth' 'Teen' 'Senior']
================================================================================
================
```

To replace 'Fe male' with 'Female' in the Gender column

In [27]:
```
df["Gender"]=df["Gender"].replace('Fe Male', 'Female')
```

Feature Engineering: Visitors, Age Groups

In [28]:
```
# Total Visitors = Adults + Children
df["TotalVisitors"] = df["NumberOfPersonVisiting"] + df["NumberOfChildrenVisiting"]
```

Drop Columns NumberOfPersonVisiting, NumberOfChildrenVisiting

In [29]:
```
df.drop(columns=["NumberOfPersonVisiting", "NumberOfChildrenVisiting"], axis=1, inp
```

Using Lable Encoder on categorical variable

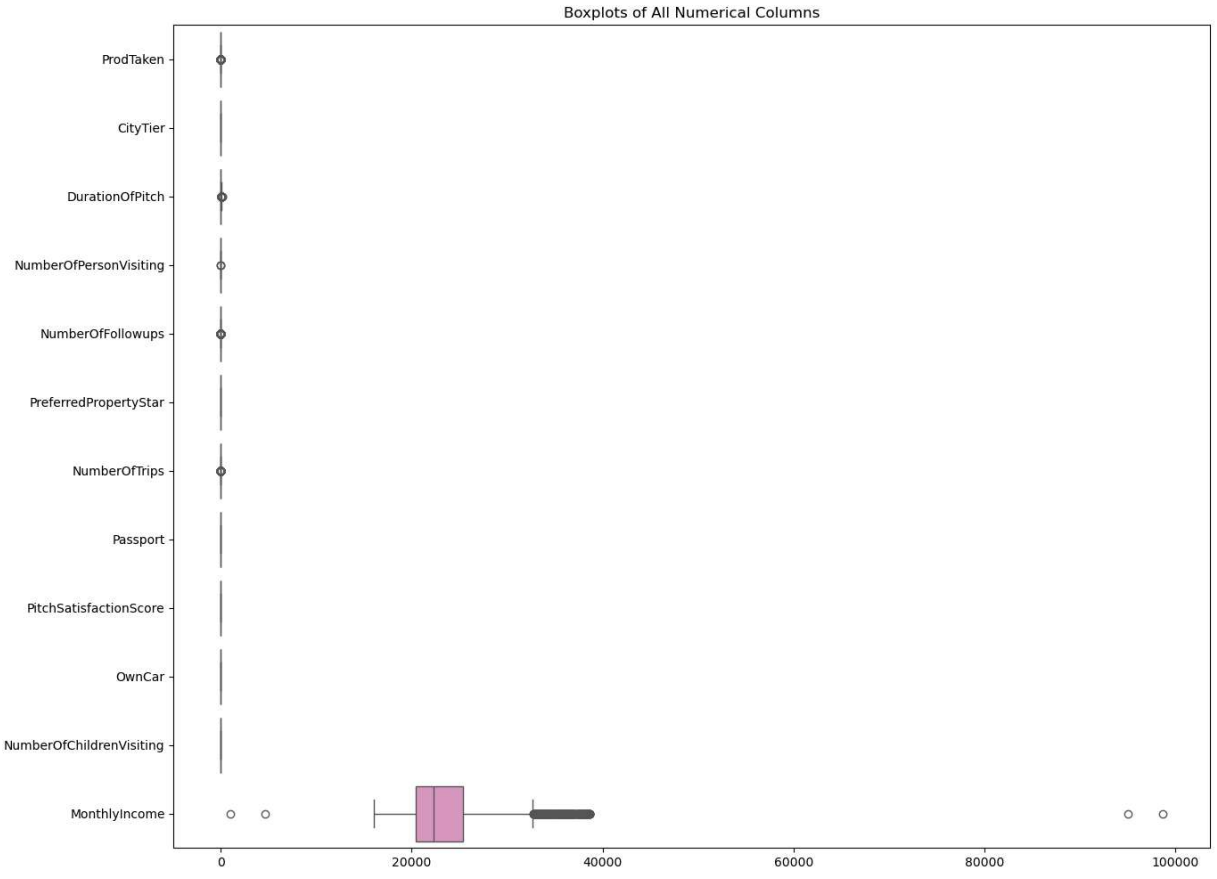
In [30]:
```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in Categorical :
    df[i]=le.fit_transform(df[i])
df.head()
```

Out[30]:

	ProdTaken	TypeofContact	CityTier	DurationOfPitch	Occupation	Gender	NumberOfFo
0	1	1	3	6.0	2	0	
1	0	0	1	14.0	2	1	
2	1	1	1	8.0	0	1	
3	0	0	1	9.0	2	0	
4	0	1	1	8.0	3	1	

Outlier Detecting

In [31]:
```
plt.figure(figsize=(15, 12))
sns.boxplot(data=Numerical, orient='h', palette='Set2')
plt.title("Boxplots of All Numerical Columns")
# plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



In [ ]:

Outlier Removal using the Z-score method

In [32]:
```
from scipy.stats import zscore
z=np.abs(zscore(df))
df1=df[(z<3).all(axis=1)]
```

In [33]:
```
print("Shape of the dateframe before removing outliers : ", df_copy.shape)
print("Shape of the dateframe after removing outliers : ", df1.shape)
print("Percentage of data loss post outlier removal : ", (df_copy.shape[0]-df1.shap
```

```
Shape of the dateframe before removing outliers :  (4888, 20)
Shape of the dateframe after removing outliers :  (4735, 18)
Percentage of data loss post outlier removal :  3.1301145662847794
```

Skewness Detecting

In [34]:
```
df1.skew()
```

Out[34]:
```
ProdTaken                 1.597909
TypeofContact            -0.933991
CityTier                  0.732430
DurationOfPitch           0.964975
Occupation               -0.427480
Gender                   -0.395823
NumberOfFollowups        -0.380593
ProductPitched            0.910108
PreferredPropertyStar     0.897199
MaritalStatus             0.493465
NumberOfTrips             0.905474
Passport                  0.932856
PitchSatisfactionScore   -0.103426
OwnCar                   -0.483821
Designation               0.417723
MonthlyIncome             1.092613
AgeBin                    0.191336
TotalVisitors             0.185750
dtype: float64
```

Spliting data in target and dependent feature

In [35]:
```
X=df1.drop(['ProdTaken'], axis=1)
y=df1['ProdTaken']
```

Applies Yeo-Johnson transformation by default to all columns in X

In [36]:
```
from sklearn.preprocessing import power_transform
df2=power_transform(X)
df3=pd.DataFrame(df2, columns=X.columns)
df3.skew()
```

Out[36]:
```
TypeofContact            -0.933991
CityTier                  0.656638
DurationOfPitch           0.047575
Occupation               -0.150521
Gender                   -0.395823
NumberOfFollowups         0.021746
ProductPitched            0.093064
PreferredPropertyStar     0.554077
MaritalStatus            -0.036343
NumberOfTrips             0.016091
Passport                  0.932856
PitchSatisfactionScore   -0.139414
OwnCar                   -0.483821
Designation              -0.015304
MonthlyIncome             0.047670
AgeBin                   -0.030907
TotalVisitors            -0.040198
dtype: float64
```

In [37]:
```
X=df3
y=df1['ProdTaken']
```

Feature Scaling

In [38]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_sc=sc.fit_transform(X)
```

Here's how to find the best random_state:

In [39]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
maxAccu=0
maxRs=0
for i in range(1, 250):
    X_train, X_test, y_train, y_test=train_test_split(X_sc, y, test_size=0.1, rando
    log_reg=LogisticRegression()
    log_reg.fit(X_train,y_train)
    y_pred=log_reg.predict(X_test)
```

```
    acc=accuracy_score(y_test, y_pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRs=i
print("Best Accuracy is : ", maxAccu, "on Random_state ", maxRs)
```

```
Best Accuracy is :  0.8860759493670886 on Random_state  122
```

Split scaled

In [40]:
```
from sklearn.linear_model import LogisticRegression
X_train, X_test, y_train, y_test=train_test_split(X_sc, y, test_size=0.9, random_st
print("X_train :", X_train.shape, "\ny_train :", y_train.shape)
print("X_test :", X_test.shape, "\ny_test :", y_test.shape)
```

```
X_train : (473, 17)
y_train : (473,)
X_test : (4262, 17)
y_test : (4262,)
```

Logistic Regression Model

In [41]:
```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
LogReg=LogisticRegression()
LogReg.fit(X_train, y_train)
LogReg_pred=LogReg.predict(X_test)
# LogReg_acc=accuracy_score(y_test, LogReg_pred)
print(classification_report(LogReg_pred, y_test))
print(confusion_matrix(LogReg_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.87      0.91      3867
           1       0.34      0.68      0.45       395

    accuracy                           0.85      4262
   macro avg       0.65      0.77      0.68      4262
weighted avg       0.91      0.85      0.87      4262

[[3345  522]
 [ 128  267]]
```

Decision Tree Classifier Model

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred=dtc.predict(X_test)
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.86	0.84	0.85	3473
1	0.36	0.41	0.38	789
accuracy			0.76	4262
macro avg	0.61	0.62	0.62	4262
weighted avg	0.77	0.76	0.76	4262

Building a Classification Model using Random Forest

In [43]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(X_train, y_train)
```

Out[43]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [44]:
```
rfc.score(X_train, y_train)
```

Out[44]: 1.0

In [45]:
```
rfc.score(X_test, y_test)
```

Out[45]: 0.8470201783200375

In [46]:
```
y_pr=rfc.predict(X_test)
y_pr
```

Out[46]: array([0, 0, 0, ..., 0, 0, 0])

Define parameter grid for tuning

In [47]:
```
from sklearn.model_selection import GridSearchCV
param_grid= {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 9, 11, 13, 15,],
    'min_samples_split': [2, 5, 10, 15, 17],
    'min_samples_leaf' : [1, 2, 4, 7, 9, 11]
}
grid_search=GridSearchCV(dtc, param_grid, cv=5)
grid_search.fit(X_train, y_train)
print("Best Params : ", grid_search.best_params_)
```

```
Best Params :  {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 4, 'min_sam
ples_split': 2}
```

Decision Tree Classifier Model + Parameter

In [48]:
```
dtc1=DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=19,
dtc1.fit(X_train, y_train)
y_pred=dtc1.predict(X_test)
print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.98      0.83      0.90      4079
           1       0.14      0.58      0.22       183

    accuracy                           0.82      4262
   macro avg       0.56      0.71      0.56      4262
weighted avg       0.94      0.82      0.87      4262
```

Oversampleing using SMOT Techniques

In [49]:
```
df["ProdTaken"].value_counts()
```

Out[49]:
```
ProdTaken
0    3853
1     894
Name: count, dtype: int64
```

In [50]:
```
from imblearn.over_sampling import SMOTE
oversample=SMOTE()
X_sm, y_sm = oversample.fit_resample(X, y)
```

```
  File "C:\Users\rusti\anaconda3\Lib\site-packages\joblib\externals\loky\backend\con
text.py", line 257, in _count_physical_cores
    cpu_info = subprocess.run(
        "wmic CPU Get NumberOfCores /Format:csv".split(),
        capture_output=True,
        text=True,
    )
  File "C:\Users\rusti\anaconda3\Lib\subprocess.py", line 554, in run
    with Popen(*popenargs, **kwargs) as process:
         ~~~~~^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\rusti\anaconda3\Lib\subprocess.py", line 1039, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
    ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                        pass_fds, cwd, env,
                        ^^^^^^^^^^^^^^^^^^^
    ...<5 lines>...
                        gid, gids, uid, umask,
                        ^^^^^^^^^^^^^^^^^^^^^^
                        start_new_session, process_group)
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\rusti\anaconda3\Lib\subprocess.py", line 1554, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^
                             # no special security
                             ^^^^^^^^^^^^^^^^^^^^^
    ...<4 lines>...
                             cwd,
                             ^^^^
                             startupinfo)
                             ^^^^^^^^^^^^
```

Feature Scaling

In [51]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_sc=sc.fit_transform(X_sm)
```

Split scaled

In [52]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_sc, y_sm, test_size=0.88, ran
```

Logistic Regression Model + SMOT

In [53]:
```
LogReg=LogisticRegression()
LogReg.fit(X_train, y_train)
LogReg_pred=LogReg.predict(X_test)
print(classification_report(LogReg_pred, y_test))
print(confusion_matrix(LogReg_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.69      0.73      0.71      3189
           1       0.75      0.70      0.73      3579

    accuracy                           0.72      6768
   macro avg       0.72      0.72      0.72      6768
weighted avg       0.72      0.72      0.72      6768

[[2343  846]
 [1058 2521]]
```

Decision Tree Classifier Model Model + SMOT

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred=dtc.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.74      0.77      3401
           1       0.76      0.81      0.78      3367

    accuracy                           0.78      6768
   macro avg       0.78      0.78      0.78      6768
weighted avg       0.78      0.78      0.78      6768
```

Decision Tree Classifier Model + Parameter + SMOT

In [55]:
```
dtc1=DecisionTreeClassifier(criterion='entropy', max_depth=15, min_samples_leaf=2,
dtc1.fit(X_train, y_train)
y_pred=dtc1.predict(X_test)
print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.75      0.78      0.77      3287
           1       0.78      0.76      0.77      3481

    accuracy                           0.77      6768
   macro avg       0.77      0.77      0.77      6768
weighted avg       0.77      0.77      0.77      6768
```

Import Required Libraries

In [56]:
```
from sklearn.model_selection import RandomizedSearchCV
```

Define Hyperparameter Grid for Random Search

In [57]:
```
n_estimators=[int(x) for x in np.linspace(10, 500, 50)]
max_samples = [int(x) for x in np.linspace(100, 1000, 7)]
# Number of features to consider at every split
max_features = ['sqrt', 'log2']
max_depth = [int(x) for x in np.linspace(10, 500, 50)]
# Minimum number of samples required to split a node
min_samples_split = [int(x) for x in np.linspace(2, 20, 5)]
# Minimum number of samples required at each Leaf node
min_samples_leaf = [int(x) for x in np.linspace(1, 30, 10)]
# Create the random grid
random_grid = { "max_samples" : max_samples,
                "n_estimators" : n_estimators,
                "max_features" : max_features,
                "max_depth" : max_depth,
                "min_samples_split" : min_samples_split,
                "min_samples_leaf" : min_samples_leaf,
                'criterion' : ['entropy', 'gini']}
print(random_grid)
```

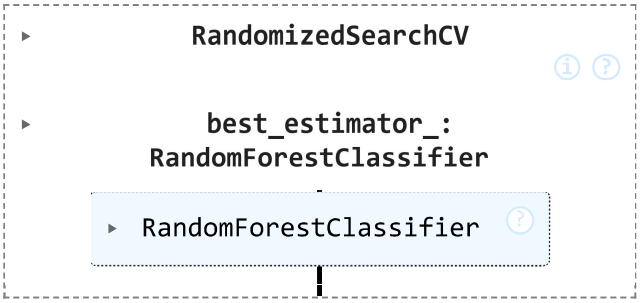
```
{'max_samples': [100, 250, 400, 550, 700, 850, 1000], 'n_estimators': [10, 20, 30, 4
0, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 2
20, 230, 240, 250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370, 380,
390, 400, 410, 420, 430, 440, 450, 460, 470, 480, 490, 500], 'max_features': ['sqr
t', 'log2'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 1
40, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 250, 260, 270, 280, 290, 300,
310, 320, 330, 340, 350, 360, 370, 380, 390, 400, 410, 420, 430, 440, 450, 460, 470,
480, 490, 500], 'min_samples_split': [2, 6, 11, 15, 20], 'min_samples_leaf': [1, 4,
7, 10, 13, 17, 20, 23, 26, 30], 'criterion': ['entropy', 'gini']}
```

Perform Hyperparameter Tuning using RandomizedSearchCV

In [58]:
```
rfc1=RandomForestClassifier()
rfc_rcv=RandomizedSearchCV(estimator=rfc1, param_distributions=random_grid,
                           n_iter=35, cv=5, verbose=2,
                           random_state=0, n_jobs=-1)
# fit the randomized model
rfc_rcv.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 35 candidates, totalling 175 fits
```

Out[58]:



Train Final Random Forest Model with Best Parameters

In [59]:
```
rfc_rcv.best_estimator_
```

Out[59]:
```
RandomForestClassifier(max_depth=390, max_features='log2', max_samples=700,
                       min_samples_split=15, n_estimators=120)
```

In [60]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(criterion='entropy', max_depth=260, max_samples=850,
                           min_samples_split=10, n_estimators=340)
rfc.fit(X_train, y_train)
```

Out[60]:
```
RandomForestClassifier(criterion='entropy', max_depth=260, max_samples=850,
                       min_samples_split=10, n_estimators=340)
```

Make Predictions and Evaluate the Model

In [61]:
```
y_pred=rfc.predict(X_test)
```

In [62]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.85	0.86	0.85	3401
1	0.86	0.84	0.85	3367
accuracy			0.85	6768
macro avg	0.85	0.85	0.85	6768
weighted avg	0.85	0.85	0.85	6768

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: