Introduction

Date - 17/04/2021
Author - Divya Iyer
Churn Modelling dataset is about some customers who are withdrawing their account from the bank due to some loss and other issues. It is crutial for banks to understand nature of their customer to come up with significant strategies to retain them. With the help of existing data we will try to analyse customer behaviour which drives them to exit from the bank.

Our objective is to analyse customer actions and create a model to predict will they exit with utmost precision and accuracy.

Using Deep Learning Neural Network Algorithm to work with this dataset.

#Importing required library and dataset

Importing mathematical, visualising, metrics calculation, scaling, modelling libraries for analysing and modelling our dataset

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import classification_report,accuracy_score
from sklearn.model_selection import train_test_split

from scipy.stats import skew
import tensorflow as tf
```

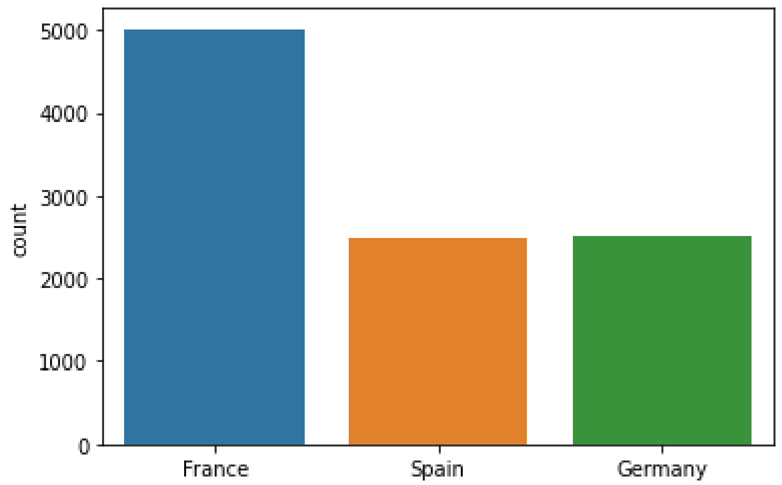
Reading dataset using pandas library

In [2]:
```
df = pd.read_csv("Churn_Modelling.csv")
```

#Data Visualization

In [3]:
```
sns.countplot(df.Geography.values)
```

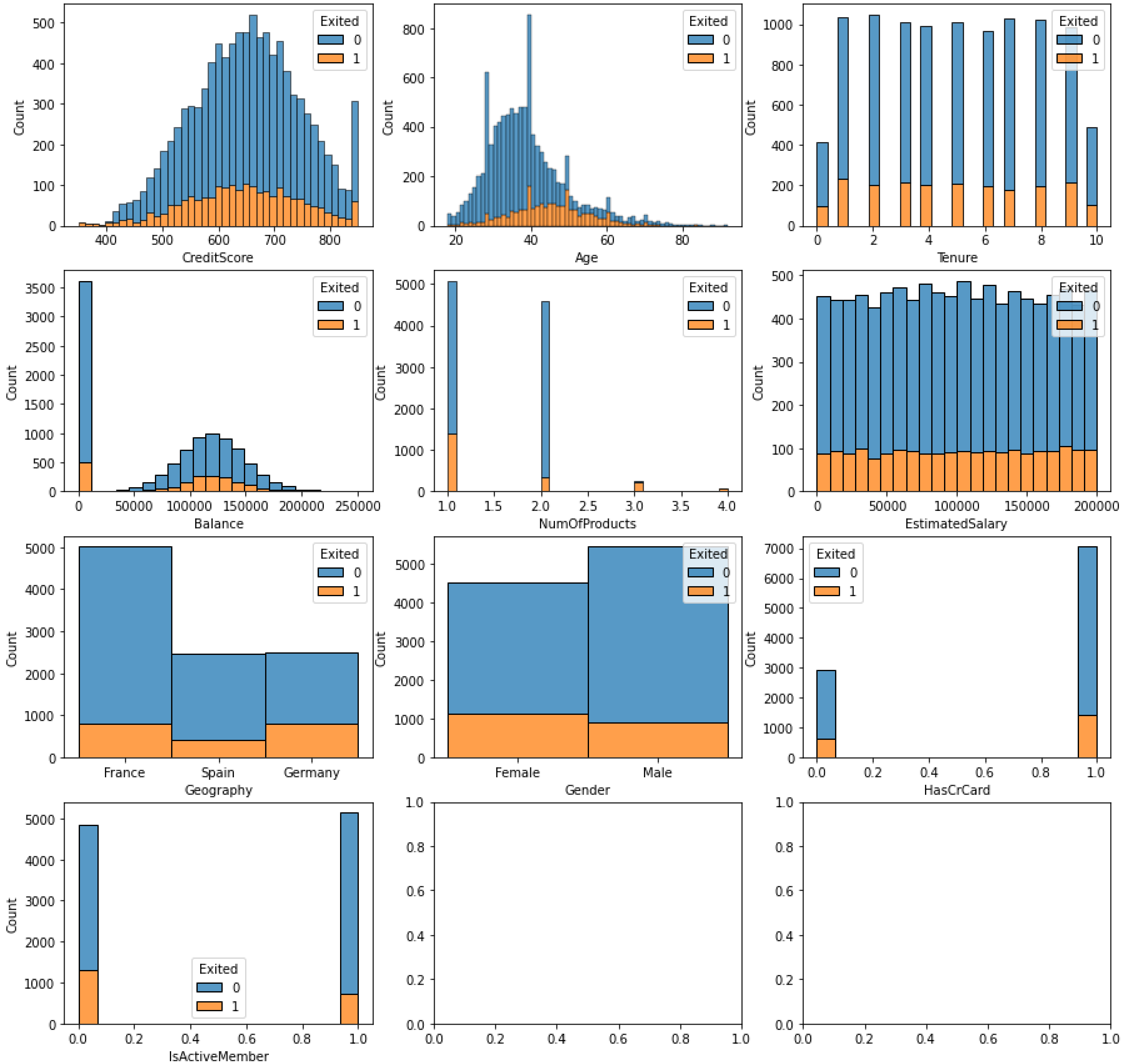
Out[3]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f598e7dd2d0>
```



Most of the bank customers are located in France.

In [4]:
```
fig, axes = plt.subplots(4, 3, figsize=(15,15))
sns.histplot(ax=axes[0, 0], data=df, x="CreditScore", hue="Exited", multiple="stack")
sns.histplot(ax=axes[0, 1], data=df, x='Age', hue="Exited", multiple="stack")
sns.histplot(ax=axes[0, 2], data=df, x='Tenure', hue="Exited", multiple="stack")
sns.histplot(ax=axes[1, 0], data=df, x='Balance', hue="Exited", multiple="stack")
sns.histplot(ax=axes[1, 1], data=df, x='NumOfProducts', hue="Exited", multiple="stack")
sns.histplot(ax=axes[1, 2], data=df, x='EstimatedSalary', hue="Exited", multiple="stack")
sns.histplot(ax=axes[2, 0], data=df, x='Geography', hue="Exited", multiple="stack")
sns.histplot(ax=axes[2, 1], data=df, x='Gender', hue="Exited", multiple="stack")
sns.histplot(ax=axes[2, 2], data=df, x='HasCrCard', hue="Exited", multiple="stack")
sns.histplot(ax=axes[3, 0], data=df, x='IsActiveMember', hue="Exited", multiple="stack")
```

Out[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5989eb41d0>
```



As observed, customers from the following groups are more likely to exit:

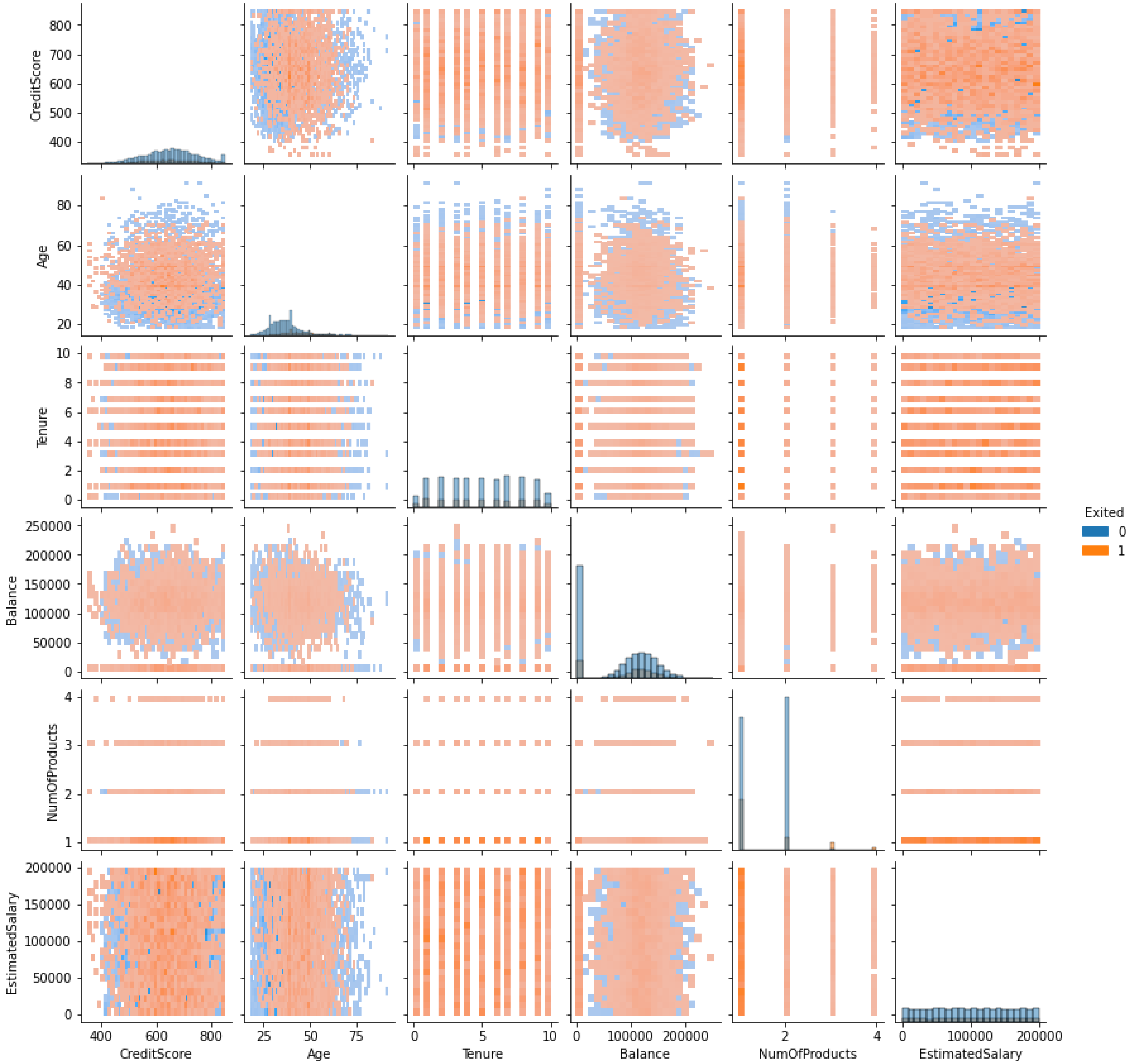
- Over the age of 40.
- From Germany.
- Female.

Customers from the following groups are less likely to exit:

- Having 2 products.
- Active members.

Creating a pair plot to visualize correlation between features

In [5]:
```python
cols = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary', 'Exited']
sns.pairplot(df[cols], hue='Exited', kind='hist', height=2)
plt.show()
```



#Analysis & EDA

Check for missing values

In [6]:
```python
df.shape
```

Out[6]: (10000, 14)

In [7]:
```
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
```

We can observe, all the rows has 10000 Non-Null entries. Hence there are no missing/NaN values in above dataset. We will check junk values in Geography and Gender column as they are of type object.

In [8]:
```
df.head()
```

Out[8]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Checking unique value for both Geography and Gender to understand if they have any junk values

In [9]:
```
df['Geography'].value_counts()
```

Out[9]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

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

Out[10]:
```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

Both of the columns do not have junk values.

Dropping column RowNumber, CustomerId, Surname as they have huge unique values and are less relevant to our analysis.

In [11]:
```
df.drop(['CustomerId','RowNumber','Surname'],axis=1,inplace=True)
```

In [12]:
```
df.head()
```

Out[12]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

Column Gender and Geography has categorical value. Splitting the dataset according to their datatype for convertion.

In [13]:
```
df_cat = df.select_dtypes(object)
df_num = df.select_dtypes(["float64", 'int64'])
```

In [14]:
```
df_cat.head()
```

Out[14]:

	Geography	Gender
0	France	Female
1	Spain	Female
2	France	Female
3	France	Female
4	Spain	Female

Label Encoding our categorical columns into numerical value for analysis

In [15]:
```
for col in df_cat:
  le =LabelEncoder()
  df_cat[col] = le.fit_transform(df_cat[col])
print(df_cat.head())
```

```
   Geography  Gender
0          0       0
1          2       0
2          0       0
3          0       0
4          2       0
```

In [16]:
```
df_cat.head()
```

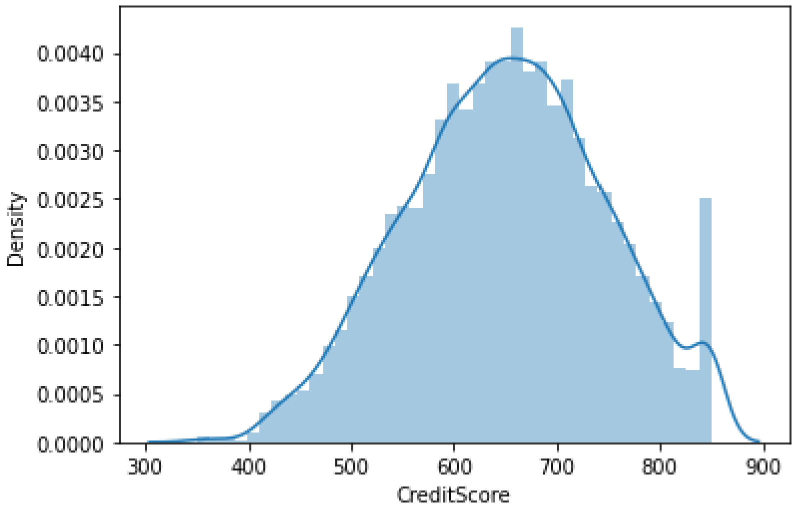
Out[16]:

	Geography	Gender
0	0	0
1	2	0
2	0	0
3	0	0
4	2	0

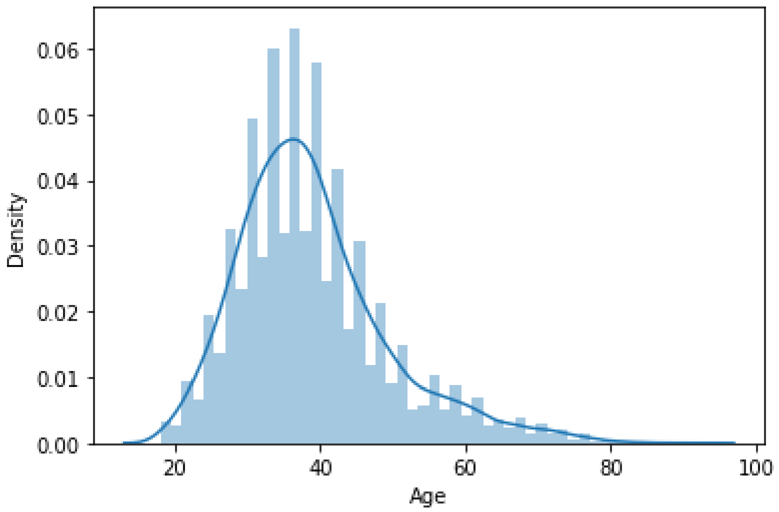
Analysing skewness of numerical dataset.

In [17]:
```
for col in df_num:
  try:
    print(col, "=", skew(df_num[col]))
    sns.distplot(df_num[col])
    plt.show()
  except:
    pass
  finally:
    print("*********************************")

#skew value is < -0.5 or >0.5
# High Skewness - Age, NUmof Products
# slight skewness - Hascrcard
```

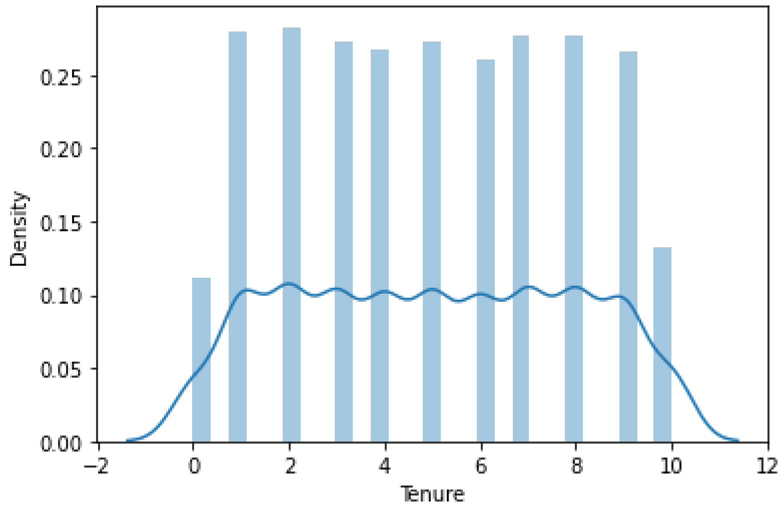
```
CreditScore = -0.07159586676212397
```



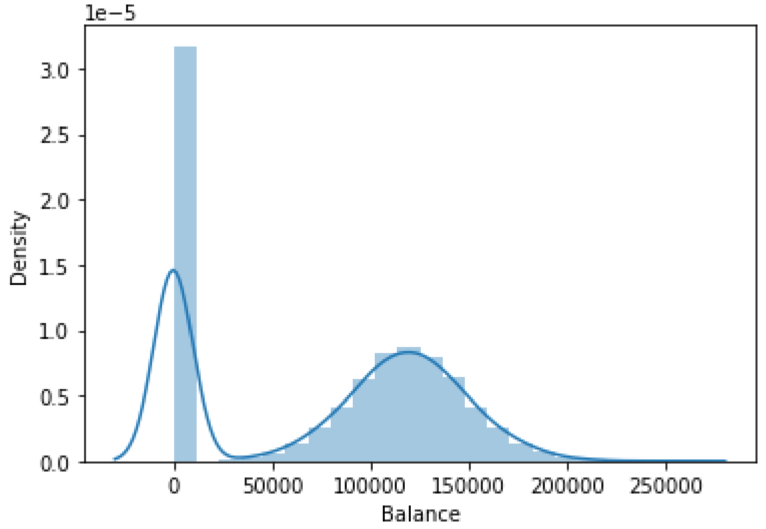

```
*********************************
Age = 1.0111685586628079
```



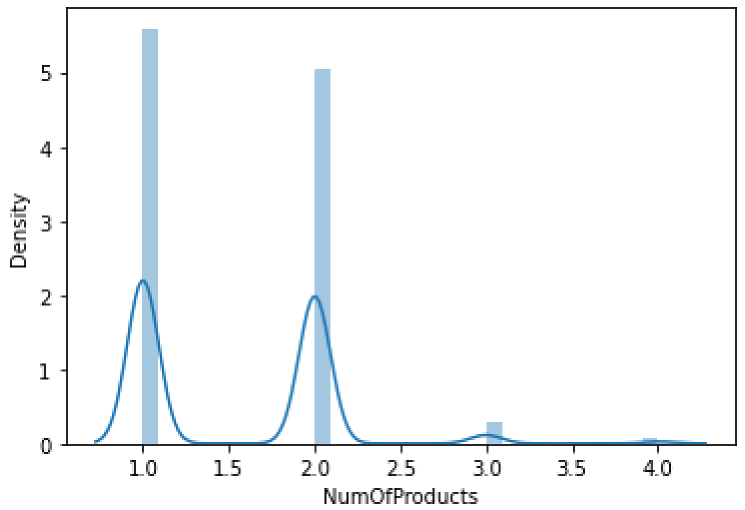

```
*********************************
Tenure = 0.010989809189781041
```



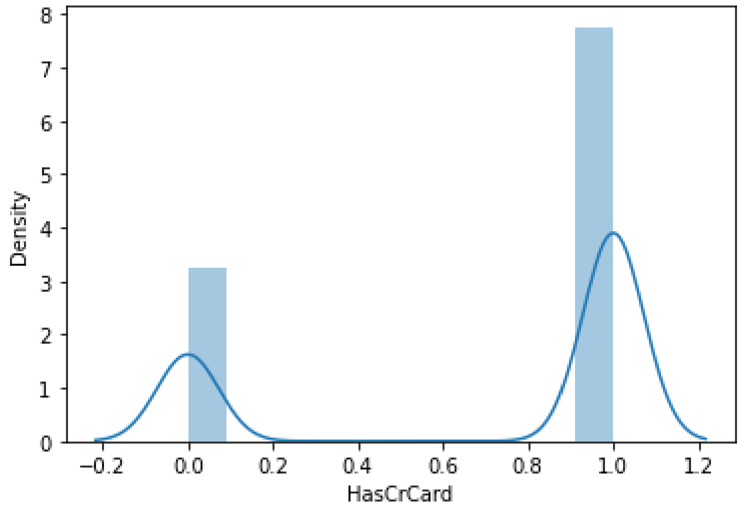

```
*******************************
Balance = -0.14108754375291138
```



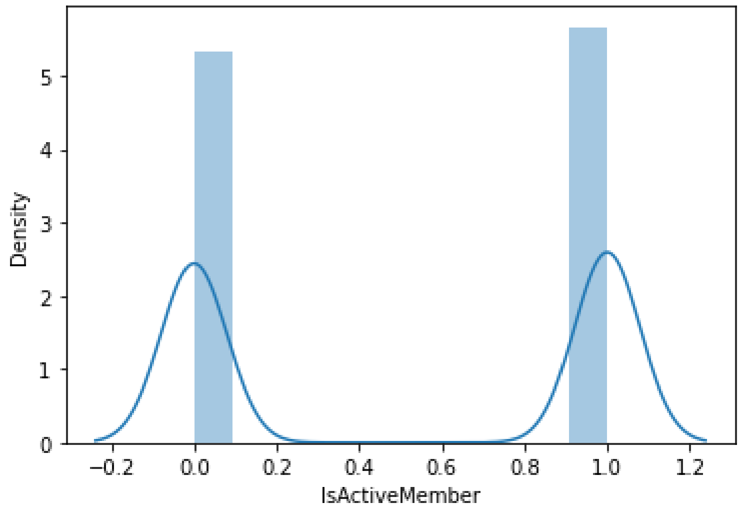

```
*******************************
NumOfProducts = 0.745456048438949
```



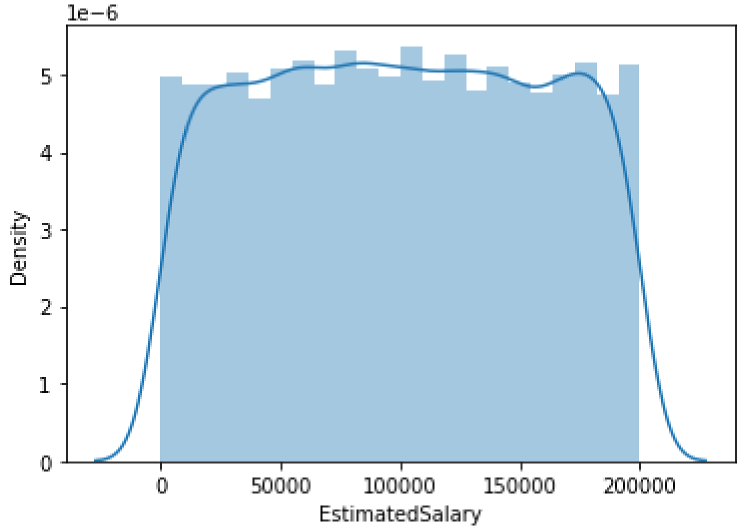

```
*******************************
HasCrCard = -0.9016763178640548
```



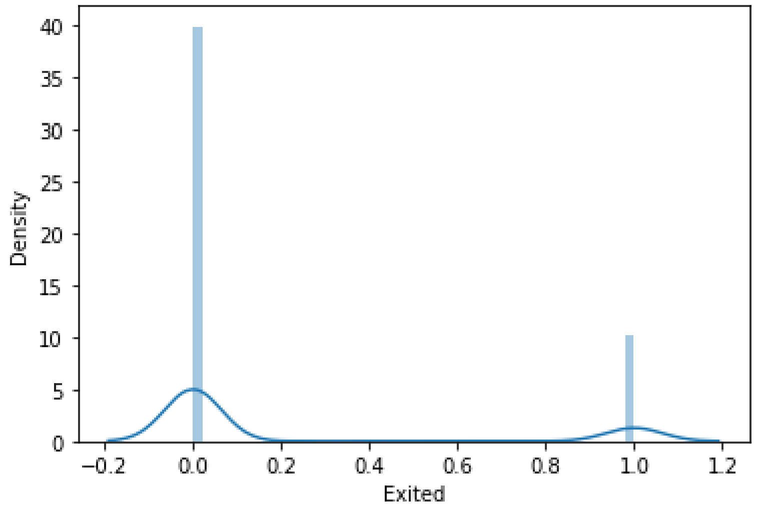
****
IsActiveMember = -0.06042756246298516



****
EstimatedSalary = 0.0020850448448748848

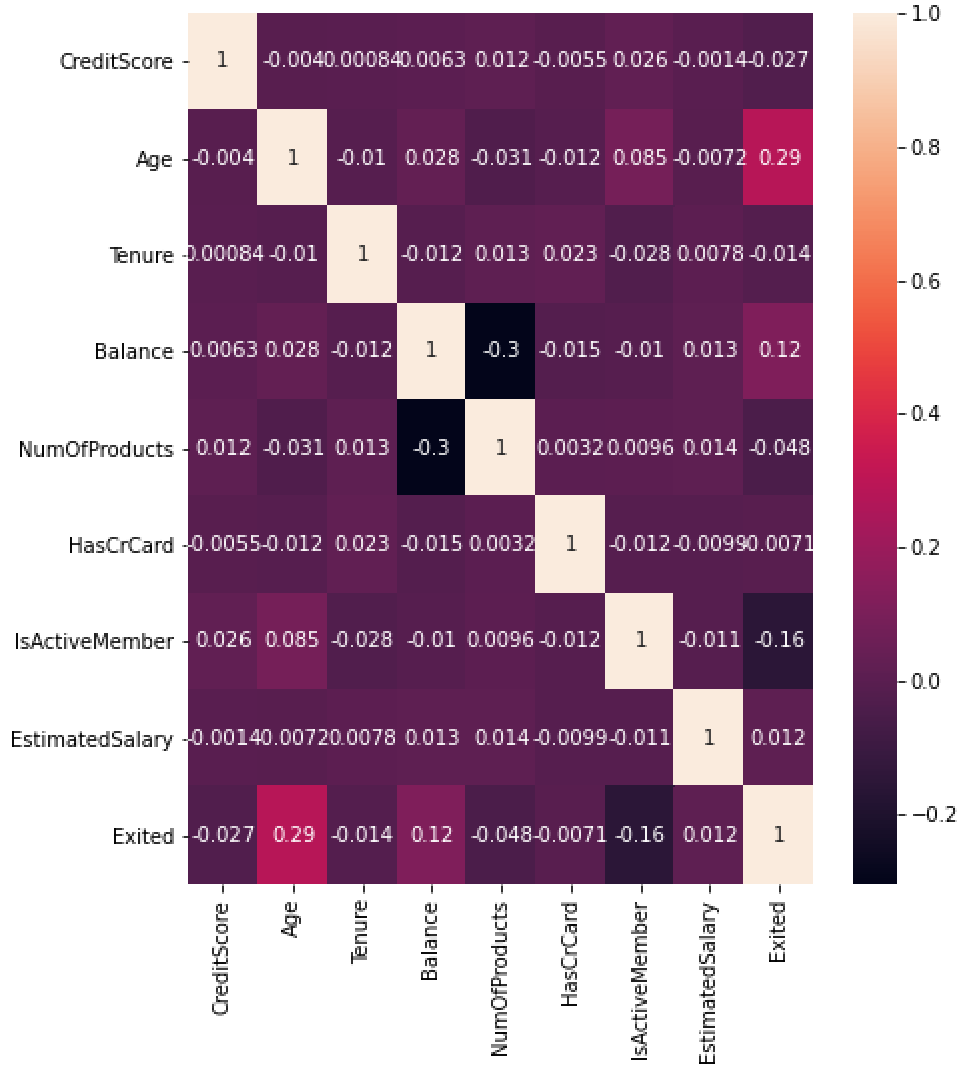


****
Exited = 1.4713899141398699



****

In [18]:
```
plt.figure(figsize=(7,8))
sns.heatmap(df_num.corr(), annot=True)
plt.show()
```



Skewness out of range are as follows:

- Age = 1.01 - ***High***
- NumofProducts = 0.74 - ***High***

Both columns do not share high correlation with target feature, handling their skewness below.

Checking for minimum and maximum value as no skewness has to be performed if the column has negative values(handling skewness for negative values leads to NaN)

In [19]:
```
print("Min: ",min(df['Age']))
print("Max: ",max(df['Age']))
```

```
Min:  18
Max:  92
```

In [20]:
```
skewed_data = np.sqrt(df_num['Age'])
skew(skewed_data)
```

Out[20]:
```
0.5933159623197802
```

In [21]:
```
df_num['Age'] = np.sqrt(df_num['Age'])
```

In [22]:
```
print("Min: ",min(df['NumOfProducts']))
print("Max: ",max(df['NumOfProducts']))
```

```
Min:  1
Max:  4
```

In [23]:
```
skewed_data = np.sqrt(df_num['NumOfProducts'])
skew(skewed_data)
```

Out[23]:
```
0.4204651463627845
```

In [24]:
```
df_num['NumOfProducts'] = np.sqrt(df_num['NumOfProducts'])
```

In [25]:
```
df_num.head()
```

Out[25]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	6.480741	2	0.00	1.000000	1	1	101348.88	1
1	608	6.403124	1	83807.86	1.000000	0	1	112542.58	0
2	502	6.480741	8	159660.80	1.732051	1	0	113931.57	1
3	699	6.244998	1	0.00	1.414214	0	0	93826.63	0
4	850	6.557439	2	125510.82	1.000000	1	1	79084.10	0

Skewness has been reduced for both columns.

Scaling Dataset

Excluding column Exited as it is the target column and HasCrCard is a categorical column by nature

In [26]:
```
for col in df_num.drop(['Exited','HasCrCard'],axis=1):
  ss = StandardScaler()
  df_num[col] = ss.fit_transform(df_num[[col]])
df_num.head()
```

Out[26]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	-0.326221	0.363279	-1.041760	-1.225848	-0.940717	1	0.970243	0.021886	1
1	-0.440036	0.267739	-1.387538	0.117350	-0.940717	0	0.970243	0.216534	0
2	-1.536794	0.363279	1.032908	1.333053	2.253455	1	-1.030670	0.240687	1
3	0.501521	0.073098	-1.387538	-1.225848	0.866630	0	-1.030670	-0.108918	0
4	2.063884	0.457688	-1.041760	0.785728	-0.940717	1	0.970243	-0.365276	0

Concatenating EDA Processed data set for further modelling

In [27]:
```
new_df = pd.concat([df_num, df_cat], axis=1)
```

In [28]:
```
new_df.head()
```

Out[28]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited	Geography	Gender
0	-0.326221	0.363279	-1.041760	-1.225848	-0.940717	1	0.970243	0.021886	1	0	0
1	-0.440036	0.267739	-1.387538	0.117350	-0.940717	0	0.970243	0.216534	0	2	0
2	-1.536794	0.363279	1.032908	1.333053	2.253455	1	-1.030670	0.240687	1	0	0
3	0.501521	0.073098	-1.387538	-1.225848	0.866630	0	-1.030670	-0.108918	0	0	0
4	2.063884	0.457688	-1.041760	0.785728	-0.940717	1	0.970243	-0.365276	0	2	0

#Modeling the dataset

As modelling the dataset and calculating accuracy of the same will be called multiple times through out notebook. Hence, creating a function to call as and when requried to reduce code duplication.

In [29]:
```
def Neural_Network(X,y,X_train,X_test,y_train,y_test):
  model = tf.keras.Sequential([
    tf.keras.layers.Dense(6, activation="relu", input_shape=(X.shape[1],)),
    tf.keras.layers.Dense(6, activation="relu"),
    tf.keras.layers.Dense(1, activation="sigmoid")
    ])

  model.compile(optimizer="adam", loss="binary_crossentropy",metrics = ['accuracy'])
  trained_model = model.fit(X_train, y_train, epochs=50,batch_size=10)

  print("******* PLOT **********")
  plt.plot(trained_model.history["loss"])

  y_pred = model.predict(X_test)
  y_pred = np.where(y_pred >= 0.5,1,0)

  print("******* CLASSIFICATION REPORT **********")
  print(classification_report(y_test,y_pred))

  print("Accuracy Score: ", accuracy_score(y_test, y_pred))

  return y_pred
```

In [30]:
```
new_df['Exited'].value_counts()
```

Out[30]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

As dataset are highly imbalanced modelling dataset with all possiblities to check accuracy of prediction.

Modelling the dataset without balancing.

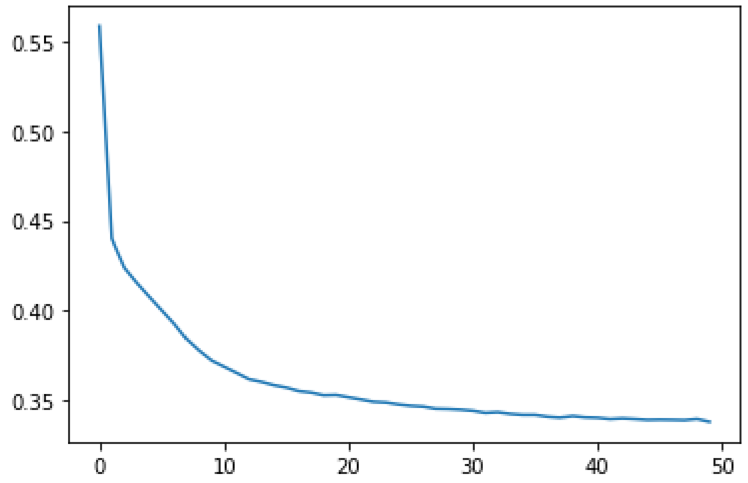
In [31]:
```
X = new_df.drop('Exited',axis='columns')
y = new_df['Exited']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=15,stratify=y)
```

In [32]:
```
Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
700/700 [==============================] - 1s 1ms/step - loss: 0.6564 - accuracy: 0.5858
Epoch 2/50
700/700 [==============================] - 1s 993us/step - loss: 0.4402 - accuracy: 0.8016
Epoch 3/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4397 - accuracy: 0.8042
Epoch 4/50
700/700 [==============================] - 1s 1ms/step - loss: 0.4355 - accuracy: 0.8099
Epoch 5/50
700/700 [==============================] - 1s 988us/step - loss: 0.4116 - accuracy: 0.8275
Epoch 6/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3883 - accuracy: 0.8350
Epoch 7/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3956 - accuracy: 0.8304
Epoch 8/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3927 - accuracy: 0.8356
Epoch 9/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3752 - accuracy: 0.8456
Epoch 10/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3737 - accuracy: 0.8472
Epoch 11/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3776 - accuracy: 0.8437
Epoch 12/50
700/700 [==============================] - 1s 1000us/step - loss: 0.3753 - accuracy: 0.8435
Epoch 13/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3531 - accuracy: 0.8552
Epoch 14/50
700/700 [==============================] - 1s 999us/step - loss: 0.3563 - accuracy: 0.8519
Epoch 15/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3543 - accuracy: 0.8523
Epoch 16/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3503 - accuracy: 0.8565
Epoch 17/50
700/700 [==============================] - 1s 995us/step - loss: 0.3643 - accuracy: 0.8500
Epoch 18/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3614 - accuracy: 0.8502
Epoch 19/50
700/700 [==============================] - 1s 983us/step - loss: 0.3521 - accuracy: 0.8518
Epoch 20/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3545 - accuracy: 0.8506
Epoch 21/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3588 - accuracy: 0.8458
Epoch 22/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3524 - accuracy: 0.8507
Epoch 23/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3639 - accuracy: 0.8471
Epoch 24/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3410 - accuracy: 0.8597
Epoch 25/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3451 - accuracy: 0.8584
Epoch 26/50
700/700 [==============================] - 1s 997us/step - loss: 0.3453 - accuracy: 0.8562
Epoch 27/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3471 - accuracy: 0.8530
Epoch 28/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3414 - accuracy: 0.8549
Epoch 29/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3441 - accuracy: 0.8579
Epoch 30/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3553 - accuracy: 0.8519
Epoch 31/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3412 - accuracy: 0.8541
Epoch 32/50
```

```
700/700 [==============================] - 1s 1ms/step - loss: 0.3485 - accuracy: 0.8544
Epoch 33/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3376 - accuracy: 0.8616
Epoch 34/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3376 - accuracy: 0.8566
Epoch 35/50
700/700 [==============================] - 1s 989us/step - loss: 0.3404 - accuracy: 0.8555
Epoch 36/50
700/700 [==============================] - 1s 984us/step - loss: 0.3377 - accuracy: 0.8579
Epoch 37/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3330 - accuracy: 0.8660
Epoch 38/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3367 - accuracy: 0.8515
Epoch 39/50
700/700 [==============================] - 1s 987us/step - loss: 0.3352 - accuracy: 0.8634
Epoch 40/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3372 - accuracy: 0.8607
Epoch 41/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3316 - accuracy: 0.8637
Epoch 42/50
700/700 [==============================] - 1s 997us/step - loss: 0.3446 - accuracy: 0.8595
Epoch 43/50
700/700 [==============================] - 1s 986us/step - loss: 0.3378 - accuracy: 0.8608
Epoch 44/50
700/700 [==============================] - 1s 993us/step - loss: 0.3445 - accuracy: 0.8548
Epoch 45/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3478 - accuracy: 0.8567
Epoch 46/50
700/700 [==============================] - 1s 993us/step - loss: 0.3374 - accuracy: 0.8599
Epoch 47/50
700/700 [==============================] - 1s 995us/step - loss: 0.3263 - accuracy: 0.8635
Epoch 48/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3335 - accuracy: 0.8613
Epoch 49/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3332 - accuracy: 0.8608
Epoch 50/50
700/700 [==============================] - 1s 1ms/step - loss: 0.3398 - accuracy: 0.8594
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.88      0.96      0.92      2389
           1       0.75      0.47      0.58       611

    accuracy                           0.86      3000
   macro avg       0.81      0.71      0.75      3000
weighted avg       0.85      0.86      0.85      3000

Accuracy Score:  0.86
```

Out[32]:
```
array([[0],
       [0],
       [1],
       ...,
       [0],
       [0],
       [0]])
```



Modelling the data as is without balancing does give us a good accuracy of 86%. But if we observe classification report, there is a high difference between f1 score for both the classes. Also it has a high recall score.

Progressing with balanced dataset ahead.

Undersampling exited class 0 to number of records of class 1, so that we have balanced number of records for both the classes.

In [33]:
```
new_df['Exited'].value_counts()
```

Out[33]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [34]:
```
#Creating two variables to store number of record of both the classes for future use
count_class_0, count_class_1 = new_df.Exited.value_counts()
```

In [35]:
```
df_class_0 = new_df[new_df['Exited']==0]
df_class_1 = new_df[new_df['Exited']==1]
```

In [36]:
```
#Creating a undersample of class 0 w.r.t number of record of class 1
df_class_0_under = df_class_0.sample(count_class_1)
```

In [37]:
```
df_class_0_under.shape
```

Out[37]:
```
(2037, 11)
```

In [38]:
```
#Concatenating records of both class 0 and class 1 to create dataset for modelling
df_undersampling = pd.concat([df_class_0_under,df_class_1],axis=0)
```

In [39]:
```
df_undersampling['Exited'].value_counts()
```

Out[39]:
```
1    2037
0    2037
Name: Exited, dtype: int64
```

We can observe we now have a dataset with 2037 records for both the classes, creating a balanced dataset.

In [40]:
```
X = df_undersampling.drop('Exited',axis='columns')
y = df_undersampling['Exited']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=15,stratify=y)
```

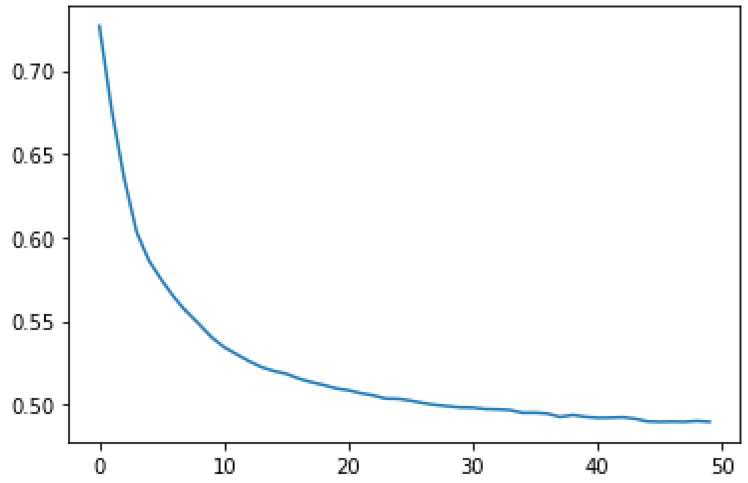
In [41]:
```
Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
286/286 [==============================] - 1s 1ms/step - loss: 0.7492 - accuracy: 0.4686
Epoch 2/50
286/286 [==============================] - 0s 1ms/step - loss: 0.6895 - accuracy: 0.5390
Epoch 3/50
286/286 [==============================] - 0s 1ms/step - loss: 0.6420 - accuracy: 0.6487
Epoch 4/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5945 - accuracy: 0.6820
Epoch 5/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5939 - accuracy: 0.6763
Epoch 6/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5819 - accuracy: 0.6923
Epoch 7/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5667 - accuracy: 0.7037
Epoch 8/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5420 - accuracy: 0.7303
Epoch 9/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5501 - accuracy: 0.7132
Epoch 10/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5516 - accuracy: 0.7164
Epoch 11/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5356 - accuracy: 0.7260
Epoch 12/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5184 - accuracy: 0.7422
Epoch 13/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5163 - accuracy: 0.7512
Epoch 14/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5101 - accuracy: 0.7544
Epoch 15/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5269 - accuracy: 0.7345
Epoch 16/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5201 - accuracy: 0.7449
Epoch 17/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5171 - accuracy: 0.7356
Epoch 18/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5218 - accuracy: 0.7305
Epoch 19/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5154 - accuracy: 0.7342
Epoch 20/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4990 - accuracy: 0.7504
Epoch 21/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5047 - accuracy: 0.7406
Epoch 22/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5042 - accuracy: 0.7411
Epoch 23/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5223 - accuracy: 0.7326
Epoch 24/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4983 - accuracy: 0.7586
Epoch 25/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4910 - accuracy: 0.7539
Epoch 26/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5028 - accuracy: 0.7418
Epoch 27/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4799 - accuracy: 0.7577
Epoch 28/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4945 - accuracy: 0.7515
Epoch 29/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4785 - accuracy: 0.7684
Epoch 30/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4978 - accuracy: 0.7494
Epoch 31/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5058 - accuracy: 0.7446
Epoch 32/50
```

```
286/286 [==============================] - 0s 1ms/step - loss: 0.5006 - accuracy: 0.7451
Epoch 33/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5173 - accuracy: 0.7460
Epoch 34/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5001 - accuracy: 0.7453
Epoch 35/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4860 - accuracy: 0.7586
Epoch 36/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5023 - accuracy: 0.7453
Epoch 37/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4932 - accuracy: 0.7430
Epoch 38/50
286/286 [==============================] - 0s 1ms/step - loss: 0.5070 - accuracy: 0.7328
Epoch 39/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4710 - accuracy: 0.7628
Epoch 40/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4945 - accuracy: 0.7495
Epoch 41/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4939 - accuracy: 0.7549
Epoch 42/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4931 - accuracy: 0.7532
Epoch 43/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4877 - accuracy: 0.7547
Epoch 44/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4996 - accuracy: 0.7411
Epoch 45/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4980 - accuracy: 0.7471
Epoch 46/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4818 - accuracy: 0.7558
Epoch 47/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4921 - accuracy: 0.7475
Epoch 48/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4885 - accuracy: 0.7539
Epoch 49/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4872 - accuracy: 0.7544
Epoch 50/50
286/286 [==============================] - 0s 1ms/step - loss: 0.4899 - accuracy: 0.7473
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.76      0.80      0.78       612
           1       0.78      0.74      0.76       611

    accuracy                           0.77      1223
   macro avg       0.77      0.77      0.77      1223
weighted avg       0.77      0.77      0.77      1223

Accuracy Score:  0.7702371218315618
```

Out[41]:

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



Oversampling exited class 1 to number of records of class 0, so that we have balanced number of records for both the classes.

In [42]:
```
#Creating a oversample of class 1 w.r.t number of record of class 0
df_class_1_over = df_class_1.sample(count_class_0,replace=True)
```

In [43]:
```
#Concatenating records of both class 0 and class 1 to create dataset for modelling
df_oversampling = pd.concat([df_class_0,df_class_1_over],axis=0)
```

In [44]:
```
df_oversampling['Exited'].value_counts()
```

Out[44]:
```
1    7963
0    7963
Name: Exited, dtype: int64
```

We can observe we now have a dataset with 7963 records for both the classes, creating a balanced dataset.

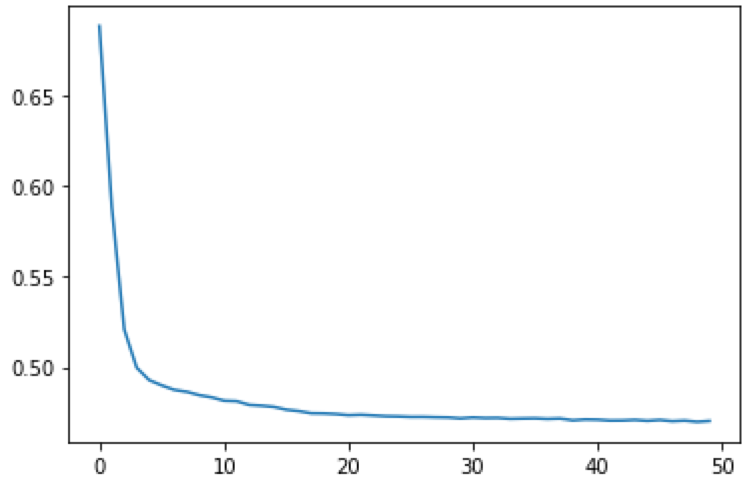
In [45]:
```
X = df_oversampling.drop('Exited',axis='columns')
y = df_oversampling['Exited']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=15,stratify=y)
```

In [46]:
```
Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
1115/1115 [==============================] - 2s 1ms/step - loss: 0.7329 - accuracy: 0.4860
Epoch 2/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.6157 - accuracy: 0.6908
Epoch 3/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.5271 - accuracy: 0.7394
Epoch 4/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.5035 - accuracy: 0.7448
Epoch 5/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4943 - accuracy: 0.7562
Epoch 6/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4945 - accuracy: 0.7486
Epoch 7/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4932 - accuracy: 0.7582
Epoch 8/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4840 - accuracy: 0.7601
Epoch 9/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4856 - accuracy: 0.7629
Epoch 10/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4770 - accuracy: 0.7650
Epoch 11/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4706 - accuracy: 0.7680
Epoch 12/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4850 - accuracy: 0.7586
Epoch 13/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4783 - accuracy: 0.7603
Epoch 14/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4813 - accuracy: 0.7556
Epoch 15/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4824 - accuracy: 0.7590
Epoch 16/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4776 - accuracy: 0.7607
Epoch 17/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4681 - accuracy: 0.7702
Epoch 18/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4763 - accuracy: 0.7591
Epoch 19/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4817 - accuracy: 0.7603
Epoch 20/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4851 - accuracy: 0.7556
Epoch 21/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4715 - accuracy: 0.7638
Epoch 22/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4744 - accuracy: 0.7602
Epoch 23/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4758 - accuracy: 0.7635
Epoch 24/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4707 - accuracy: 0.7651
Epoch 25/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4626 - accuracy: 0.7714
Epoch 26/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4740 - accuracy: 0.7672
Epoch 27/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4737 - accuracy: 0.7605
Epoch 28/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4672 - accuracy: 0.7680
Epoch 29/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4816 - accuracy: 0.7602
Epoch 30/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4763 - accuracy: 0.7560
Epoch 31/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4730 - accuracy: 0.7614
Epoch 32/50
```

```
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4713 - accuracy: 0.7635
Epoch 33/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4725 - accuracy: 0.7642
Epoch 34/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4676 - accuracy: 0.7670
Epoch 35/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4766 - accuracy: 0.7593
Epoch 36/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4718 - accuracy: 0.7666
Epoch 37/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4729 - accuracy: 0.7644
Epoch 38/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4708 - accuracy: 0.7656
Epoch 39/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4781 - accuracy: 0.7606
Epoch 40/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4642 - accuracy: 0.7678
Epoch 41/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4683 - accuracy: 0.7732
Epoch 42/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4668 - accuracy: 0.7676
Epoch 43/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4704 - accuracy: 0.7664
Epoch 44/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4694 - accuracy: 0.7668
Epoch 45/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4757 - accuracy: 0.7660
Epoch 46/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4730 - accuracy: 0.7684
Epoch 47/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4786 - accuracy: 0.7640
Epoch 48/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4644 - accuracy: 0.7715
Epoch 49/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4767 - accuracy: 0.7647
Epoch 50/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4597 - accuracy: 0.7734
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.76      0.76      0.76      2389
           1       0.76      0.76      0.76      2389

    accuracy                           0.76      4778
   macro avg       0.76      0.76      0.76      4778
weighted avg       0.76      0.76      0.76      4778

Accuracy Score:  0.7584763499372122
```

Out[46]:
```
array([[0],
       [0],
       [1],
       ...,
       [0],
       [0],
       [1]])
```



SMOTE Technique

Using SMOTE technique to balance the dataset for class with minor number of records, so that we have equal number of records for both the classes. This technique creates releavant data for class with minor number of records for balancing.

In [47]:
```
X = new_df.drop('Exited',axis='columns')
y = new_df['Exited']
```

In [48]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE(sampling_strategy='minority')
X_sm,y_sm=smote.fit_sample(X,y)
```

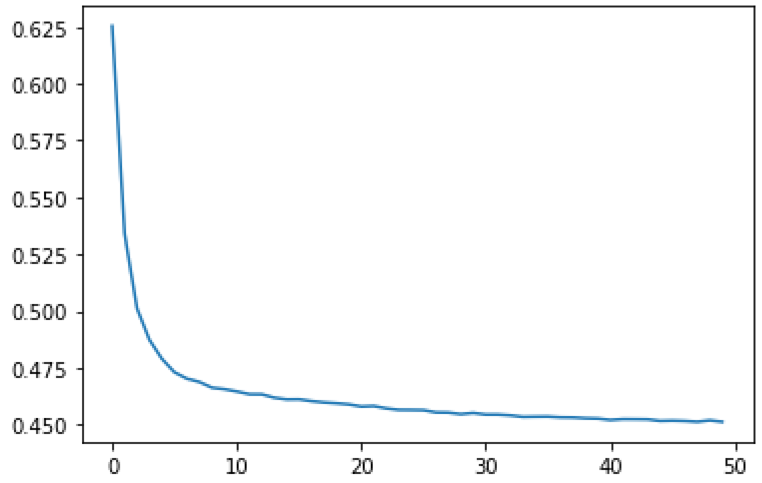
In [49]:
```
X_train,X_test,y_train,y_test = train_test_split(X_sm,y_sm,test_size=0.3,random_state=15,stratify=y_sm)
```

In [50]:
```
Neural_Network(X_sm,y_sm,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
1115/1115 [==============================] - 2s 1ms/step - loss: 0.6818 - accuracy: 0.5949
Epoch 2/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.5402 - accuracy: 0.7384
Epoch 3/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.5044 - accuracy: 0.7478
Epoch 4/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4883 - accuracy: 0.7603
Epoch 5/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4858 - accuracy: 0.7584
Epoch 6/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4657 - accuracy: 0.7748
Epoch 7/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4766 - accuracy: 0.7689
Epoch 8/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4687 - accuracy: 0.7704
Epoch 9/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4586 - accuracy: 0.7775
Epoch 10/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4643 - accuracy: 0.7786
Epoch 11/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4710 - accuracy: 0.7723
Epoch 12/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4591 - accuracy: 0.7781
Epoch 13/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4656 - accuracy: 0.7769
Epoch 14/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4683 - accuracy: 0.7716
Epoch 15/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4697 - accuracy: 0.7684
Epoch 16/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4509 - accuracy: 0.7847
Epoch 17/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4588 - accuracy: 0.7776
Epoch 18/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4558 - accuracy: 0.7801
Epoch 19/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4578 - accuracy: 0.7775
Epoch 20/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4561 - accuracy: 0.7803
Epoch 21/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4638 - accuracy: 0.7692
Epoch 22/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4511 - accuracy: 0.7848
Epoch 23/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4599 - accuracy: 0.7809
Epoch 24/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4569 - accuracy: 0.7758
Epoch 25/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4573 - accuracy: 0.7769
Epoch 26/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4614 - accuracy: 0.7757
Epoch 27/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4569 - accuracy: 0.7777
Epoch 28/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4517 - accuracy: 0.7824
Epoch 29/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4590 - accuracy: 0.7742
Epoch 30/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4594 - accuracy: 0.7737
Epoch 31/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4573 - accuracy: 0.7787
Epoch 32/50
```

```
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4559 - accuracy: 0.7757
Epoch 33/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4610 - accuracy: 0.7748
Epoch 34/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4559 - accuracy: 0.7711
Epoch 35/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4508 - accuracy: 0.7781
Epoch 36/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4583 - accuracy: 0.7750
Epoch 37/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4458 - accuracy: 0.7850
Epoch 38/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4443 - accuracy: 0.7854
Epoch 39/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4501 - accuracy: 0.7794
Epoch 40/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4448 - accuracy: 0.7816
Epoch 41/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4559 - accuracy: 0.7762
Epoch 42/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4559 - accuracy: 0.7794
Epoch 43/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4594 - accuracy: 0.7732
Epoch 44/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4535 - accuracy: 0.7756
Epoch 45/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4542 - accuracy: 0.7758
Epoch 46/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4543 - accuracy: 0.7837
Epoch 47/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4550 - accuracy: 0.7810
Epoch 48/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4450 - accuracy: 0.7844
Epoch 49/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4515 - accuracy: 0.7783
Epoch 50/50
1115/1115 [==============================] - 1s 1ms/step - loss: 0.4537 - accuracy: 0.7781
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.77      0.80      0.79      2389
           1       0.79      0.76      0.78      2389

    accuracy                           0.78      4778
   macro avg       0.78      0.78      0.78      4778
weighted avg       0.78      0.78      0.78      4778

Accuracy Score:  0.781707827542905
```

```
Out[50]: array([[0],
       [1],
       [1],
       ...,
       [0],
       [0],
       [1]])
```



Use of Ensemble with undersampling

As the column Exited class 0 has 4 times greater number of records as compared to class 1, diving the class 0 into equal number of 4 batches and concatenating the sub-batch with class 1 to create a balanced dataset.

In [51]:
```
X = new_df.drop('Exited',axis='columns')
y = new_df['Exited']

X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.3,random_state=15,stratify=y)
```

In [52]:
```
temp_df = X_train.copy()
temp_df['Exited'] = y_train

df3_class_0 = temp_df[temp_df['Exited']==0]
df3_class_1 = temp_df[temp_df['Exited']==1]
```

In [53]:
```
print(df3_class_0.shape," , ", df3_class_1.shape) #1394
```

```
(5574, 11)  ,  (1426, 11)
```

Function to create batches of dataset

In [54]:
```
def get_train_batch(df_majority,df_minority,start,end):
  df_train = pd.concat([df_majority[start:end],df_minority],axis=0)

  X_train = df_train.drop('Exited',axis='columns')
  y_train = df_train.Exited

  return X_train,y_train
```

Batch 1 of Ensemble Technique

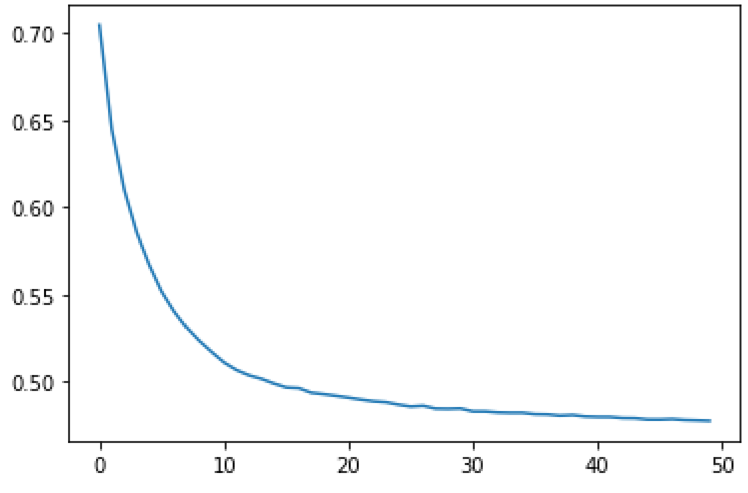
In [55]:
```
X_train, y_train = get_train_batch(df3_class_0,df3_class_1,0,1394)
```

In [56]:

```python
y_pred1 = Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
282/282 [==============================] - 1s 1ms/step - loss: 0.7414 - accuracy: 0.5143
Epoch 2/50
282/282 [==============================] - 0s 991us/step - loss: 0.6523 - accuracy: 0.6193
Epoch 3/50
282/282 [==============================] - 0s 1ms/step - loss: 0.6193 - accuracy: 0.6589
Epoch 4/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5890 - accuracy: 0.7006
Epoch 5/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5663 - accuracy: 0.7218
Epoch 6/50
282/282 [==============================] - 0s 986us/step - loss: 0.5472 - accuracy: 0.7383
Epoch 7/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5229 - accuracy: 0.7549
Epoch 8/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5335 - accuracy: 0.7383
Epoch 9/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5184 - accuracy: 0.7405
Epoch 10/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5168 - accuracy: 0.7403
Epoch 11/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5122 - accuracy: 0.7447
Epoch 12/50
282/282 [==============================] - 0s 984us/step - loss: 0.5039 - accuracy: 0.7442
Epoch 13/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4960 - accuracy: 0.7528
Epoch 14/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5063 - accuracy: 0.7520
Epoch 15/50
282/282 [==============================] - 0s 994us/step - loss: 0.5105 - accuracy: 0.7469
Epoch 16/50
282/282 [==============================] - 0s 985us/step - loss: 0.5094 - accuracy: 0.7427
Epoch 17/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5081 - accuracy: 0.7463
Epoch 18/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4888 - accuracy: 0.7669
Epoch 19/50
282/282 [==============================] - 0s 957us/step - loss: 0.4848 - accuracy: 0.7539
Epoch 20/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5053 - accuracy: 0.7466
Epoch 21/50
282/282 [==============================] - 0s 1000us/step - loss: 0.4942 - accuracy: 0.7538
Epoch 22/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4767 - accuracy: 0.7682
Epoch 23/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4748 - accuracy: 0.7680
Epoch 24/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4990 - accuracy: 0.7491
Epoch 25/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4875 - accuracy: 0.7568
Epoch 26/50
282/282 [==============================] - 0s 995us/step - loss: 0.4880 - accuracy: 0.7515
Epoch 27/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4752 - accuracy: 0.7654
Epoch 28/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4759 - accuracy: 0.7604
Epoch 29/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4779 - accuracy: 0.7659
Epoch 30/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4739 - accuracy: 0.7611
Epoch 31/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4874 - accuracy: 0.7594
Epoch 32/50
```

```
282/282 [==============================] - 0s 1ms/step - loss: 0.4882 - accuracy: 0.7507
Epoch 33/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4748 - accuracy: 0.7624
Epoch 34/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4909 - accuracy: 0.7588
Epoch 35/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4755 - accuracy: 0.7656
Epoch 36/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4708 - accuracy: 0.7681
Epoch 37/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4803 - accuracy: 0.7565
Epoch 38/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4803 - accuracy: 0.7571
Epoch 39/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4898 - accuracy: 0.7473
Epoch 40/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4695 - accuracy: 0.7680
Epoch 41/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4834 - accuracy: 0.7545
Epoch 42/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5003 - accuracy: 0.7428
Epoch 43/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4740 - accuracy: 0.7635
Epoch 44/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4729 - accuracy: 0.7750
Epoch 45/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4910 - accuracy: 0.7519
Epoch 46/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4796 - accuracy: 0.7613
Epoch 47/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4785 - accuracy: 0.7669
Epoch 48/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4666 - accuracy: 0.7742
Epoch 49/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4933 - accuracy: 0.7540
Epoch 50/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4772 - accuracy: 0.7679
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.93      0.73      0.82      2389
           1       0.42      0.77      0.55       611

    accuracy                           0.74      3000
   macro avg       0.67      0.75      0.68      3000
weighted avg       0.82      0.74      0.76      3000

Accuracy Score:  0.7403333333333333
```



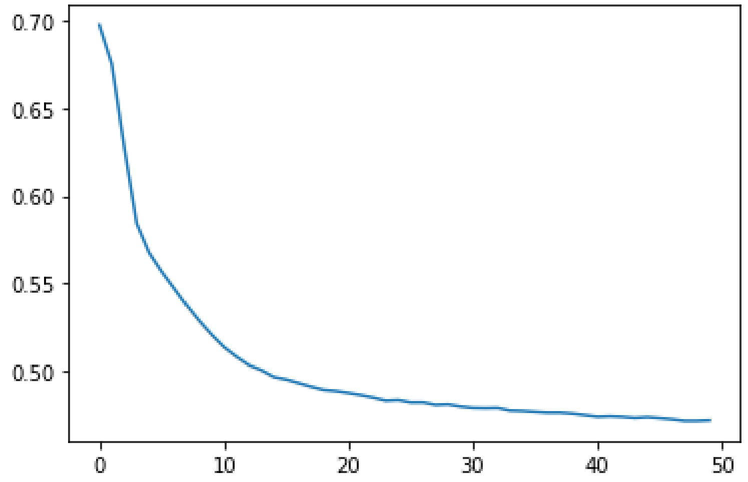
Batch 2 of Ensemble Technique

In [57]:
```
X_train, y_train = get_train_batch(df3_class_0,df3_class_1,1394,2788)
```

In [58]:
```
y_pred2 = Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
282/282 [==============================] - 1s 1ms/step - loss: 0.7059 - accuracy: 0.4510
Epoch 2/50
282/282 [==============================] - 0s 943us/step - loss: 0.6831 - accuracy: 0.5401
Epoch 3/50
282/282 [==============================] - 0s 1ms/step - loss: 0.6373 - accuracy: 0.6699
Epoch 4/50
282/282 [==============================] - 0s 984us/step - loss: 0.5820 - accuracy: 0.7083
Epoch 5/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5680 - accuracy: 0.7142
Epoch 6/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5595 - accuracy: 0.7133
Epoch 7/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5571 - accuracy: 0.7184
Epoch 8/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5433 - accuracy: 0.7280
Epoch 9/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5207 - accuracy: 0.7439
Epoch 10/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5272 - accuracy: 0.7289
Epoch 11/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5247 - accuracy: 0.7367
Epoch 12/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5168 - accuracy: 0.7508
Epoch 13/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5063 - accuracy: 0.7577
Epoch 14/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5042 - accuracy: 0.7559
Epoch 15/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4760 - accuracy: 0.7756
Epoch 16/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4803 - accuracy: 0.7745
Epoch 17/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4966 - accuracy: 0.7673
Epoch 18/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4799 - accuracy: 0.7706
Epoch 19/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4859 - accuracy: 0.7649
Epoch 20/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4930 - accuracy: 0.7659
Epoch 21/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4882 - accuracy: 0.7690
Epoch 22/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5039 - accuracy: 0.7501
Epoch 23/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4727 - accuracy: 0.7767
Epoch 24/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4902 - accuracy: 0.7681
Epoch 25/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5082 - accuracy: 0.7443
Epoch 26/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4908 - accuracy: 0.7644
Epoch 27/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4870 - accuracy: 0.7686
Epoch 28/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4630 - accuracy: 0.7780
Epoch 29/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4751 - accuracy: 0.7799
Epoch 30/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4824 - accuracy: 0.7729
Epoch 31/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4785 - accuracy: 0.7686
Epoch 32/50
```

```
282/282 [==============================] - 0s 1ms/step - loss: 0.4897 - accuracy: 0.7597
Epoch 33/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4760 - accuracy: 0.7761
Epoch 34/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4755 - accuracy: 0.7845
Epoch 35/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4747 - accuracy: 0.7743
Epoch 36/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4776 - accuracy: 0.7664
Epoch 37/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4654 - accuracy: 0.7841
Epoch 38/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4685 - accuracy: 0.7794
Epoch 39/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4785 - accuracy: 0.7700
Epoch 40/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4841 - accuracy: 0.7699
Epoch 41/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4742 - accuracy: 0.7745
Epoch 42/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4497 - accuracy: 0.7896
Epoch 43/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4712 - accuracy: 0.7648
Epoch 44/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4618 - accuracy: 0.7756
Epoch 45/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4797 - accuracy: 0.7735
Epoch 46/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4900 - accuracy: 0.7573
Epoch 47/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4740 - accuracy: 0.7709
Epoch 48/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4867 - accuracy: 0.7747
Epoch 49/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4795 - accuracy: 0.7874
Epoch 50/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4634 - accuracy: 0.7804
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.93      0.74      0.82      2389
           1       0.43      0.77      0.55       611

    accuracy                           0.75      3000
   macro avg       0.68      0.75      0.69      3000
weighted avg       0.83      0.75      0.77      3000

Accuracy Score:  0.748
```



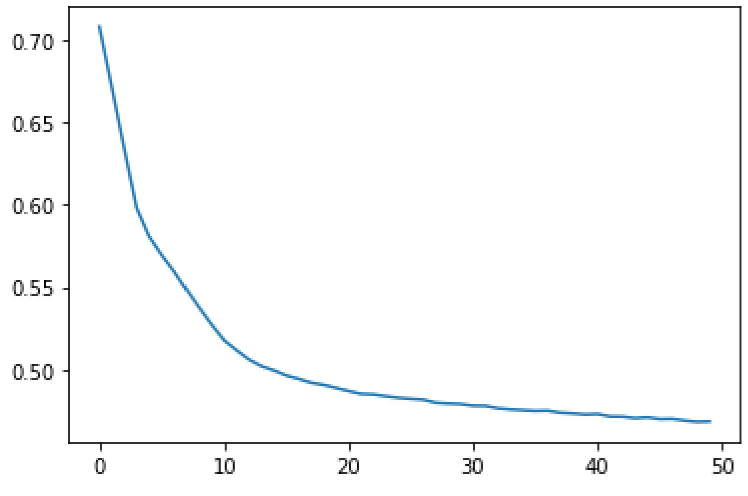
Batch 3 of Ensemble Technique

In [59]:
```
X_train, y_train = get_train_batch(df3_class_0,df3_class_1,2788,4182)
```

In [60]:
```
y_pred3 = Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
282/282 [==============================] - 1s 1ms/step - loss: 0.7292 - accuracy: 0.5000
Epoch 2/50
282/282 [==============================] - 0s 941us/step - loss: 0.6772 - accuracy: 0.5749
Epoch 3/50
282/282 [==============================] - 0s 991us/step - loss: 0.6448 - accuracy: 0.6322
Epoch 4/50
282/282 [==============================] - 0s 1ms/step - loss: 0.6030 - accuracy: 0.6886
Epoch 5/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5824 - accuracy: 0.7094
Epoch 6/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5756 - accuracy: 0.7072
Epoch 7/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5617 - accuracy: 0.7210
Epoch 8/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5604 - accuracy: 0.7307
Epoch 9/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5452 - accuracy: 0.7391
Epoch 10/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5099 - accuracy: 0.7593
Epoch 11/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5095 - accuracy: 0.7513
Epoch 12/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5133 - accuracy: 0.7498
Epoch 13/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5104 - accuracy: 0.7443
Epoch 14/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4976 - accuracy: 0.7569
Epoch 15/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5055 - accuracy: 0.7548
Epoch 16/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4982 - accuracy: 0.7568
Epoch 17/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4847 - accuracy: 0.7700
Epoch 18/50
282/282 [==============================] - 0s 994us/step - loss: 0.4926 - accuracy: 0.7550
Epoch 19/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4987 - accuracy: 0.7551
Epoch 20/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4832 - accuracy: 0.7521
Epoch 21/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4833 - accuracy: 0.7491
Epoch 22/50
282/282 [==============================] - 0s 1000us/step - loss: 0.5027 - accuracy: 0.7523
Epoch 23/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4890 - accuracy: 0.7578
Epoch 24/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4904 - accuracy: 0.7625
Epoch 25/50
282/282 [==============================] - 0s 999us/step - loss: 0.4718 - accuracy: 0.7771
Epoch 26/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4820 - accuracy: 0.7631
Epoch 27/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4854 - accuracy: 0.7650
Epoch 28/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4910 - accuracy: 0.7508
Epoch 29/50
282/282 [==============================] - 0s 975us/step - loss: 0.4815 - accuracy: 0.7513
Epoch 30/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4811 - accuracy: 0.7652
Epoch 31/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4763 - accuracy: 0.7645
Epoch 32/50
```

```
282/282 [==============================] - 0s 1ms/step - loss: 0.4695 - accuracy: 0.7841
Epoch 33/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4903 - accuracy: 0.7565
Epoch 34/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4755 - accuracy: 0.7589
Epoch 35/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4792 - accuracy: 0.7685
Epoch 36/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4617 - accuracy: 0.7735
Epoch 37/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4778 - accuracy: 0.7620
Epoch 38/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4568 - accuracy: 0.7804
Epoch 39/50
282/282 [==============================] - 0s 972us/step - loss: 0.4815 - accuracy: 0.7607
Epoch 40/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4750 - accuracy: 0.7625
Epoch 41/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4609 - accuracy: 0.7739
Epoch 42/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4764 - accuracy: 0.7639
Epoch 43/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4814 - accuracy: 0.7613
Epoch 44/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4787 - accuracy: 0.7697
Epoch 45/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4727 - accuracy: 0.7622
Epoch 46/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4657 - accuracy: 0.7753
Epoch 47/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4687 - accuracy: 0.7637
Epoch 48/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4830 - accuracy: 0.7586
Epoch 49/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4734 - accuracy: 0.7632
Epoch 50/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4637 - accuracy: 0.7718
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.92      0.79      0.85      2389
           1       0.47      0.72      0.57       611

    accuracy                           0.78      3000
   macro avg       0.69      0.75      0.71      3000
weighted avg       0.82      0.78      0.79      3000

Accuracy Score:  0.775
```

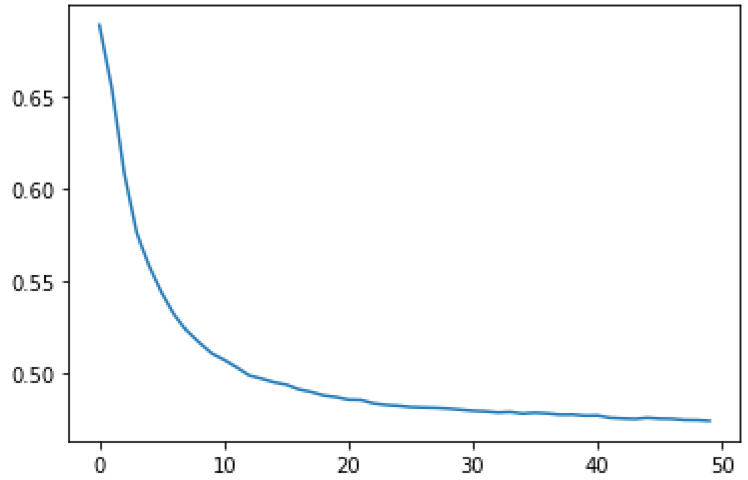


Batch 4 of Ensemble Technique

In [61]:
```
X_train, y_train = get_train_batch(df3_class_0,df3_class_1,4182,5575)
y_pred4 = Neural_Network(X,y,X_train,X_test,y_train,y_test)
```

```
Epoch 1/50
282/282 [==============================] - 1s 1ms/step - loss: 0.6958 - accuracy: 0.5132
Epoch 2/50
282/282 [==============================] - 0s 1ms/step - loss: 0.6662 - accuracy: 0.5986
Epoch 3/50
282/282 [==============================] - 0s 1ms/step - loss: 0.6121 - accuracy: 0.6759
Epoch 4/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5877 - accuracy: 0.6820
Epoch 5/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5538 - accuracy: 0.7023
Epoch 6/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5457 - accuracy: 0.7160
Epoch 7/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5308 - accuracy: 0.7351
Epoch 8/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5183 - accuracy: 0.7425
Epoch 9/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5230 - accuracy: 0.7397
Epoch 10/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5220 - accuracy: 0.7321
Epoch 11/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5266 - accuracy: 0.7323
Epoch 12/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5116 - accuracy: 0.7446
Epoch 13/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4999 - accuracy: 0.7472
Epoch 14/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5054 - accuracy: 0.7475
Epoch 15/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4976 - accuracy: 0.7412
Epoch 16/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4970 - accuracy: 0.7591
Epoch 17/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4955 - accuracy: 0.7512
Epoch 18/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5035 - accuracy: 0.7464
Epoch 19/50
282/282 [==============================] - 0s 1ms/step - loss: 0.5017 - accuracy: 0.7352
Epoch 20/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4849 - accuracy: 0.7569
Epoch 21/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4910 - accuracy: 0.7588
Epoch 22/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4807 - accuracy: 0.7664
Epoch 23/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4705 - accuracy: 0.7675
Epoch 24/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4843 - accuracy: 0.7568
Epoch 25/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4864 - accuracy: 0.7581
Epoch 26/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4767 - accuracy: 0.7641
Epoch 27/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4858 - accuracy: 0.7712
Epoch 28/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4918 - accuracy: 0.7512
Epoch 29/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4800 - accuracy: 0.7700
Epoch 30/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4759 - accuracy: 0.7633
Epoch 31/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4804 - accuracy: 0.7616
```

```
Epoch 32/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4759 - accuracy: 0.7617
Epoch 33/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4656 - accuracy: 0.7727
Epoch 34/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4802 - accuracy: 0.7573
Epoch 35/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4755 - accuracy: 0.7629
Epoch 36/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4791 - accuracy: 0.7587
Epoch 37/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4918 - accuracy: 0.7562
Epoch 38/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4862 - accuracy: 0.7653
Epoch 39/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4649 - accuracy: 0.7776
Epoch 40/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4579 - accuracy: 0.7742
Epoch 41/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4706 - accuracy: 0.7683
Epoch 42/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4651 - accuracy: 0.7865
Epoch 43/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4839 - accuracy: 0.7516
Epoch 44/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4750 - accuracy: 0.7684
Epoch 45/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4788 - accuracy: 0.7617
Epoch 46/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4619 - accuracy: 0.7707
Epoch 47/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4711 - accuracy: 0.7683
Epoch 48/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4676 - accuracy: 0.7631
Epoch 49/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4591 - accuracy: 0.7713
Epoch 50/50
282/282 [==============================] - 0s 1ms/step - loss: 0.4703 - accuracy: 0.7642
******* PLOT **********
******* CLASSIFICATION REPORT **********
              precision    recall  f1-score   support

           0       0.92      0.77      0.84      2389
           1       0.45      0.72      0.55       611

    accuracy                           0.76      3000
   macro avg       0.68      0.75      0.70      3000
weighted avg       0.82      0.76      0.78      3000

Accuracy Score:  0.7633333333333333
```



Taking average of all four batches to record output of majority of all the 4 balanced modelling performed.

In [62]:
```
y_pred_final = y_pred1.copy()

for i in range(len(y_pred1)):
  n_ones = y_pred1[i]+y_pred2[i]+y_pred3[i]+y_pred4[i]

  if n_ones>2:
    y_pred_final[i]=1
  else:
    y_pred_final[i]=0
```

In [63]:
```
print(classification_report(y_test,y_pred_final))
print("Accuracy Score: ", accuracy_score(y_test, y_pred_final))
```

```
              precision    recall  f1-score   support

           0       0.92      0.79      0.85      2389
           1       0.47      0.72      0.57       611

    accuracy                           0.78      3000
   macro avg       0.69      0.76      0.71      3000
weighted avg       0.83      0.78      0.79      3000

Accuracy Score:  0.775
```

```
#Observation
```

Analysing prediction all four balancing methods.

1. Modeling using unbalanced dataset - This method does give us a good accuracy but there are high false positives and false negatives. Also the difference between F1 scores is quite huge.
2. Modeling using undersampled/oversampled dataset - It gives us a little better outcome considering both accuracy as well as F1 scores.
3. Modeling using SMOTE dataset - This method does give us a good accuracy also the false positives and false negatives are quite low compared to above technique.
4. Modeling using ensembling with undersampling dataset - This method does give us a good accuracy also the false positives and false negatives are considerably low compared to all other technique. But the difference between F1 score of both the classes also cannot be ignored.

#***Conclusion***

Observing the accuracy of all the modelling SMOTE Technique for balancing helps us provide a better prediction with maximum accuracy of 80%, with a balanced F1 score of 80% for both the classes.

```
We chose a Deep Learning Neural Network with 2 hidden layers with 6 nodes each with relu as activation function and one output layer with sigmoid as activation function.
```