subject : Handling imbalance Datasets in machine learning

Writer : Amir Mohammad Nouri

linkdin profile :www.linkedin.com/in/amir-mohammad-nouri-228123218 (http://www.linkedin.com/in/amir-mohammad-nouri-228123218)

link to download dataframe : https://www.kaggle.com/datasets/blastchar/telco-customer-churn (https://www.kaggle.com/datasets/blastchar/telco-customer-churn)

Dear my friend "BEST WAY TO LEARN HOW TO USE IMPORTANT LIBRARIES AND METHODS IS TO START CODING NOT JUST WATCHNG:)"

In [1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
df = pd.read_csv(r'E:\py_importants\Deep_Learning\Code_basics\WA_Fn-UseC_-Telco-C
df.head()
```

Out[2]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Ir
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

In [3]:
```
df.shape
```

Out[3]:
```
(7043, 21)
```

In [4]:
```
df.head()
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Ir
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	

5 rows × 21 columns

In [5]:
```
df.tail()
```

Out[5]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines
7038	6840-RESVB	Male	0	Yes	Yes	24	Yes	Yes
7039	2234-XADUH	Female	0	Yes	Yes	72	Yes	Yes
7040	4801-JZAZL	Female	0	Yes	Yes	11	No	No phone service
7041	8361-LTMKD	Male	1	Yes	No	4	Yes	Yes
7042	3186-AJIEK	Male	0	No	No	66	Yes	No

5 rows × 21 columns

drop customer ID column

In [6]:
```
df = df.drop('customerID',axis='columns')
```

In [7]:
```
df.dtypes
```

Out[7]:
```
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges        object
Churn               object
dtype: object
```

as we can see above monthlyCharges are float but total charges is object so we should convert it as float too

In [8]:
```
df['TotalCharges'].values
```

Out[8]:
```
array(['29.85', '1889.5', '108.15', ..., '346.45', '306.6', '6844.5'],
      dtype=object)
```

you can see that total charges are shown as string type

In [249]:
```
pd.to_numeric(df['TotalCharges'])
```

...

so one data has space it occurs an error we can ignore that error by this code below :

In [10]:
```
pd.to_numeric(df['TotalCharges'],errors= 'coerce').isnull()
```

Out[10]:
```
0       False
1       False
2       False
3       False
4       False
        ...
7038    False
7039    False
7040    False
7041    False
7042    False
Name: TotalCharges, Length: 7043, dtype: bool
```

now we should find totalcharges that are blank :

In [11]:
```
df[pd.to_numeric(df['TotalCharges'],errors= 'coerce').isnull()]
```

Out[11]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServ
488	Female	0	Yes	Yes	0	No	No phone service	D
753	Male	0	No	Yes	0	Yes	No	
936	Female	0	Yes	Yes	0	Yes	No	D
1082	Male	0	Yes	Yes	0	Yes	Yes	
1340	Female	0	Yes	Yes	0	No	No phone service	D
3331	Male	0	Yes	Yes	0	Yes	No	
3826	Male	0	Yes	Yes	0	Yes	Yes	
4380	Female	0	Yes	Yes	0	Yes	No	
5218	Male	0	Yes	Yes	0	Yes	No	
6670	Female	0	Yes	Yes	0	Yes	Yes	D
6754	Male	0	No	Yes	0	Yes	Yes	D

as we can see above these are column that have blank total charges row

In [12]:
```
df.iloc[488]
```

Out[12]:
```
gender                                  Female
SeniorCitizen                                0
Partner                                    Yes
Dependents                                 Yes
tenure                                       0
PhoneService                                No
MultipleLines                 No phone service
InternetService                             DSL
OnlineSecurity                             Yes
OnlineBackup                                No
DeviceProtection                           Yes
TechSupport                                Yes
StreamingTV                                Yes
StreamingMovies                             No
Contract                              Two year
PaperlessBilling                           Yes
PaymentMethod       Bank transfer (automatic)
MonthlyCharges                           52.55
TotalCharges
Churn                                       No
Name: 488, dtype: object
```

you can see in data 488 total charges is blank

now with code below we can drop those data with blank total charges column

In [13]:
```
df1 = df[df['TotalCharges']!=' ']
```

In [14]:
```
df1.shape
```

Out[14]:
```
(7032, 20)
```

In [15]:
```
df1.TotalCharges = pd.to_numeric(df1.TotalCharges)
```

...

In [16]:
```
df1.TotalCharges.dtype\
```

Out[16]:
```
dtype('float64')
```

In [17]:
```
df1[df1.Churn=='No']
```

Out[17]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServ
0	Female	0	Yes	No	1	No	No phone service	D
1	Male	0	No	No	34	Yes	No	D
3	Male	0	No	No	45	No	No phone service	D
6	Male	0	No	Yes	22	Yes	Yes	Fiber o
7	Female	0	No	No	10	No	No phone service	D
...	...	...	...	...	...	...	...	
7037	Female	0	No	No	72	Yes	No	
7038	Male	0	Yes	Yes	24	Yes	Yes	D
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber o
7040	Female	0	Yes	Yes	11	No	No phone service	D
7042	Male	0	No	No	66	Yes	No	Fiber o

5163 rows × 20 columns

In [18]:
```
tenure_churn_no = df1[df1.Churn=='No'].tenure
tenure_churn_yes = df1[df1.Churn=='Yes'].tenure
```

In [19]:
```
tenure_churn_no
```

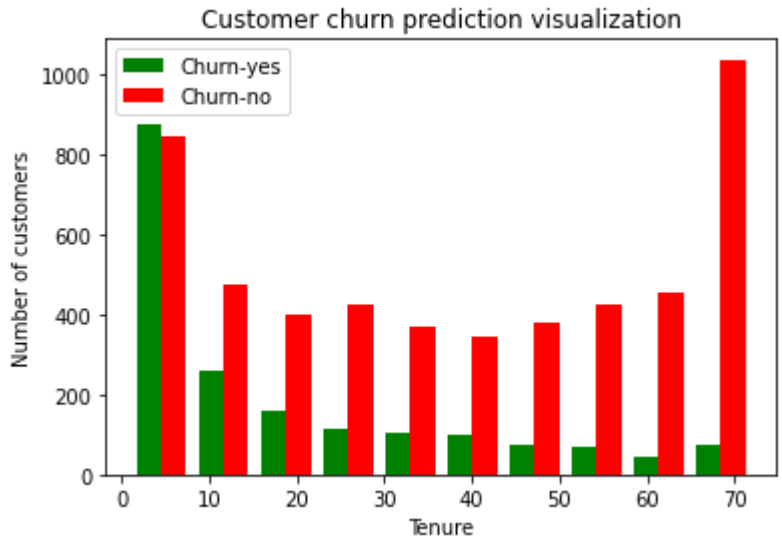
Out[19]:
```
0        1
1       34
3       45
6       22
7       10
       ..
7037    72
7038    24
7039    72
7040    11
7042    66
Name: tenure, Length: 5163, dtype: int64
```

In [20]:
```
tenure_churn_yes
```

Out[20]:
```
2        2
4        2
5        8
8       28
13      49
        ..
7021    12
7026     9
7032     1
7034    67
7041     4
Name: tenure, Length: 1869, dtype: int64
```

In [21]:
```
plt.xlabel('Tenure')
plt.ylabel('Number of customers')
plt.title('Customer churn prediction visualization')
plt.hist([tenure_churn_yes,tenure_churn_no] , color = ['green','red'],label=['Chu
plt.legend()
```

Out[21]:
```
<matplotlib.legend.Legend at 0x22ceb548>
```



find column parameters:

In [22]:
```
for column in df :
    print(column)
```

```
gender
SeniorCitizen
Partner
Dependents
tenure
PhoneService
MultipleLines
InternetService
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport
StreamingTV
StreamingMovies
Contract
PaperlessBilling
PaymentMethod
MonthlyCharges
TotalCharges
Churn
```

In [23]:
```
for column in df :
    print(df[column].unique())
```

```
['Female' 'Male']
[0 1]
['Yes' 'No']
['No' 'Yes']
[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
 39]
['No' 'Yes']
['No phone service' 'No' 'Yes']
['DSL' 'Fiber optic' 'No']
['No' 'Yes' 'No internet service']
['Yes' 'No' 'No internet service']
['No' 'Yes' 'No internet service']
['No' 'Yes' 'No internet service']
['No' 'Yes' 'No internet service']
['No' 'Yes' 'No internet service']
['Month-to-month' 'One year' 'Two year']
['Yes' 'No']
['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
[29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
['No' 'Yes']
```

In [24]:
```
for column in df :
    print(f'{column} :{df[column].unique()}')
```

```
gender :['Female' 'Male']
SeniorCitizen :[0 1]
Partner :['Yes' 'No']
Dependents :['No' 'Yes']
tenure :[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 2
7
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
 39]
PhoneService :['No' 'Yes']
MultipleLines :['No phone service' 'No' 'Yes']
InternetService :['DSL' 'Fiber optic' 'No']
OnlineSecurity :['No' 'Yes' 'No internet service']
OnlineBackup :['Yes' 'No' 'No internet service']
DeviceProtection :['No' 'Yes' 'No internet service']
TechSupport :['No' 'Yes' 'No internet service']
StreamingTV :['No' 'Yes' 'No internet service']
StreamingMovies :['No' 'Yes' 'No internet service']
Contract :['Month-to-month' 'One year' 'Two year']
PaperlessBilling :['Yes' 'No']
PaymentMethod :['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges :[29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges :['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
Churn :['No' 'Yes']
```

print those columns that are not numerical

In [25]:
```
for column in df :
    if df[column].dtype=='object':
        print(f'{column} :{df[column].unique()}')
```

```
gender :['Female' 'Male']
Partner :['Yes' 'No']
Dependents :['No' 'Yes']
PhoneService :['No' 'Yes']
MultipleLines :['No phone service' 'No' 'Yes']
InternetService :['DSL' 'Fiber optic' 'No']
OnlineSecurity :['No' 'Yes' 'No internet service']
OnlineBackup :['Yes' 'No' 'No internet service']
DeviceProtection :['No' 'Yes' 'No internet service']
TechSupport :['No' 'Yes' 'No internet service']
StreamingTV :['No' 'Yes' 'No internet service']
StreamingMovies :['No' 'Yes' 'No internet service']
Contract :['Month-to-month' 'One year' 'Two year']
PaperlessBilling :['Yes' 'No']
PaymentMethod :['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
TotalCharges :['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
Churn :['No' 'Yes']
```

make to work as a function

In [26]:
```
def print_unique_col_values(df):
    for column in df :
        if df[column].dtype=='object':
            print(f'{column} :{df[column].unique()}')
```

In [27]:
```
print_unique_col_values(df1)
```

```
gender :['Female' 'Male']
Partner :['Yes' 'No']
Dependents :['No' 'Yes']
PhoneService :['No' 'Yes']
MultipleLines :['No phone service' 'No' 'Yes']
InternetService :['DSL' 'Fiber optic' 'No']
OnlineSecurity :['No' 'Yes' 'No internet service']
OnlineBackup :['Yes' 'No' 'No internet service']
DeviceProtection :['No' 'Yes' 'No internet service']
TechSupport :['No' 'Yes' 'No internet service']
StreamingTV :['No' 'Yes' 'No internet service']
StreamingMovies :['No' 'Yes' 'No internet service']
Contract :['Month-to-month' 'One year' 'Two year']
PaperlessBilling :['Yes' 'No']
PaymentMethod :['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn :['No' 'Yes']
```

replace 'No internet service' and 'No phone service' in each column which has by 'no'

pandas library has function named replace :

In [28]:
```
df1.replace('No internet service','No',inplace=True)
```

...

In [29]:
```
df1.replace('No phone service','No',inplace=True)
#inplace=True exactly meaens df1 =df1.replace('No phone servicee','No')
```

In [30]:
```
print_unique_col_values(df1)
```

```
gender :['Female' 'Male']
Partner :['Yes' 'No']
Dependents :['No' 'Yes']
PhoneService :['No' 'Yes']
MultipleLines :['No' 'Yes']
InternetService :['DSL' 'Fiber optic' 'No']
OnlineSecurity :['No' 'Yes']
OnlineBackup :['Yes' 'No']
DeviceProtection :['No' 'Yes']
TechSupport :['No' 'Yes']
StreamingTV :['No' 'Yes']
StreamingMovies :['No' 'Yes']
Contract :['Month-to-month' 'One year' 'Two year']
PaperlessBilling :['Yes' 'No']
PaymentMethod :['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
Churn :['No' 'Yes']
```

define which columns have yes or no then replace yes with 1 and replace no with 0

In [31]:
```
yes_no_columns = df1[['Partner','Dependents','PhoneService',
                      'MultipleLines','OnlineSecurity','OnlineBackup',
                      'DeviceProtection','TechSupport','StreamingTV',
                      'StreamingMovies','PaperlessBilling','Churn']]

for col in yes_no_columns:
    df1.replace({'Yes':1 , 'No':0}, inplace=True)
```

In [32]:
```
for col in df1 :
    print(f'{col} :{df1[col].unique()}')
```

```
gender :['Female' 'Male']
SeniorCitizen :[0 1]
Partner :[1 0]
Dependents :[0 1]
tenure :[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 2
7
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService :[0 1]
MultipleLines :[0 1]
InternetService :['DSL' 'Fiber optic' 0]
OnlineSecurity :[0 1]
OnlineBackup :[1 0]
DeviceProtection :[0 1]
TechSupport :[0 1]
StreamingTV :[0 1]
StreamingMovies :[0 1]
Contract :['Month-to-month' 'One year' 'Two year']
PaperlessBilling :[1 0]
PaymentMethod :['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges :[29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges :[  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
Churn :[0 1]
```

coding Male and Female in Gender column

In [33]:
```
df1['gender'].replace({'Female':1 , 'Male':0}, inplace=True)
```

...

In [34]:
```
df1['gender'].unique()
```

Out[34]:
```
array([1, 0], dtype=int64)
```

In [35]:
```
for col in df1 :
    print(f'{col} :{df1[col].unique()}')
```

```
gender :[1 0]
SeniorCitizen :[0 1]
Partner :[1 0]
Dependents :[0 1]
tenure :[ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 2
7
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
PhoneService :[0 1]
MultipleLines :[0 1]
InternetService :['DSL' 'Fiber optic' 0]
OnlineSecurity :[0 1]
OnlineBackup :[1 0]
DeviceProtection :[0 1]
TechSupport :[0 1]
StreamingTV :[0 1]
StreamingMovies :[0 1]
Contract :['Month-to-month' 'One year' 'Two year']
PaperlessBilling :[1 0]
PaymentMethod :['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
MonthlyCharges :[29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
TotalCharges :[  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
Churn :[0 1]
```

For columns which have more than 2 categories we use ONE HOT ENCODING methode

In [36]:
```
df2 = pd.get_dummies(data=df1 , columns=['InternetService','Contract',
                                         'PaymentMethod',])
df2
```

Out[36]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSe
0	1	0	1	0	1	0	0	
1	0	0	0	0	34	1	0	
2	0	0	0	0	2	1	0	
3	0	0	0	0	45	0	0	
4	1	0	0	0	2	1	0	
...	...	...	...	...	...	...	...	
7038	0	0	1	1	24	1	1	
7039	1	0	1	1	72	1	1	
7040	1	0	1	1	11	0	0	
7041	0	1	1	0	4	1	1	
7042	0	0	0	0	66	1	0	

7032 rows × 27 columns

In [37]:
```
df2.dtypes
#all data should be number
```

Out[37]:
```
gender                                     int64
SeniorCitizen                              int64
Partner                                    int64
Dependents                                 int64
tenure                                     int64
PhoneService                               int64
MultipleLines                              int64
OnlineSecurity                             int64
OnlineBackup                               int64
DeviceProtection                           int64
TechSupport                                int64
StreamingTV                                int64
StreamingMovies                            int64
PaperlessBilling                           int64
MonthlyCharges                           float64
TotalCharges                             float64
Churn                                      int64
InternetService_0                          uint8
InternetService_DSL                        uint8
InternetService_Fiber optic                uint8
Contract_Month-to-month                    uint8
Contract_One year                          uint8
Contract_Two year                          uint8
PaymentMethod_Bank transfer (automatic)    uint8
PaymentMethod_Credit card (automatic)      uint8
PaymentMethod_Electronic check             uint8
PaymentMethod_Mailed check                 uint8
dtype: object
```

we should implement data scaling to have better accuracy

In [38]:
```
df2['tenure'].max()
```

Out[38]:
```
72
```

In [39]:
```
#df2['tenure']=df2['tenure']/72
```

In [40]:
```
df2.head()
```

Out[40]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSecurity
0	1	0	1	0	1	0	0	0
1	0	0	0	0	34	1	0	1
2	0	0	0	0	2	1	0	1
3	0	0	0	0	45	0	0	1
4	1	0	0	0	2	1	0	0

5 rows × 27 columns

also we can use minmaxscaler to do this :

In [41]:
```
cols_to_scale= ['tenure','MonthlyCharges','TotalCharges']

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df2[cols_to_scale] = scaler.fit_transform(df2[cols_to_scale])
```

In [42]:
```
df2[cols_to_scale].sample(3)
```

Out[42]:

	tenure	MonthlyCharges	TotalCharges
4846	0.084507	0.506468	0.054218
3666	0.690141	0.658209	0.472808
4161	0.014085	0.315423	0.010189

Here we go. our data frame is ready to use

data splitting

In [43]:
```
x = df2.drop('Churn',axis='columns')
y= df2['Churn']
```

In [44]:
```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(x,y,test_size=0.2,random_s
```

In [45]:
```
x_train.shape
```

Out[45]:
```
(5625, 26)
```

In [46]:
```
x_test.shape
```

Out[46]:
```
(1407, 26)
```

import tensorflow libraries

In [69]:
```
#imbalance dataset :
y_test.value_counts()
```

Out[69]:
```
0    1033
1     374
Name: Churn, dtype: int64
```

In [68]:
```
y_pred = ANN(x_train , y_train , x_test,y_test , 'binary_crossentropy',-1)
```

```
Epoch 89/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3587 - accur
acy: 0.8340
Epoch 90/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3590 - accur
acy: 0.8350
Epoch 91/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3586 - accur
acy: 0.8354
Epoch 92/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3589 - accur
acy: 0.8343
Epoch 93/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3579 - accur
acy: 0.8334
Epoch 94/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3569 - accur
acy: 0.8350
Epoch 95/100
176/176 [==============================] - 0s 2ms/step - loss: 0.3586 - accur
```

In [50]:
```
yp = model.predict(x_test)
yp[:5]
```

```
44/44 [==============================] - 0s 1ms/step
```

Out[50]:
```
array([[0.81748974],
       [0.0047618 ],
       [0.00101619],
       [0.0455246 ],
       [0.6271967 ]], dtype=float32)
```

convert 2dimensional array to 1dimensional array

In [51]:
```
y_pred = []
for element in yp:
    if element>0.5:
        y_pred.append(1)
    else:
        y_pred.append(0)
```

In [52]:
```
y_pred[:5]
```

Out[52]:
```
[1, 0, 0, 0, 1]
```

In [53]:
```
y_test[:5]
```

Out[53]:
```
3536    0
5804    0
3295    0
3541    0
490     0
Name: Churn, dtype: int64
```

print classification record

Handling imbalance dataset

method1) Undersampling

In [72]:
```
df2_class_0 = df2[df2['Churn']==0]
df2_class_1 = df2[df2['Churn']==1]
```

In [73]:
```
df2_class_0.shape
```

Out[73]:
```
(5163, 27)
```

In [75]:
```
df2_class_1.shape
```

Out[75]:
```
(1869, 27)
```

as you can see above we have 5163 data with churn = 0 /n while we have only 1869 data with churn =1

create data frame which both classes have equal number of data in it

In [80]:
```
count_class_0 , count_class_1 = df1.Churn.value_counts()
```

In [81]:
```
count_class_0,count_class_1
```

Out[81]: (5163, 1869)

In [84]:
```
df_class_0_new = df2_class_0.sample(count_class_1)
df_class_0_new.shape
```

Out[84]: (1869, 27)

Concat them together

In [88]:
```
df_test_balanced = pd.concat([df_class_0_new,df2_class_1],axis = 0)
df_test_balanced
```

Out[88]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSe
283	1	0	0	0	0.943662	1	1	
4076	0	0	1	0	0.971831	1	1	
107	1	0	0	0	0.436620	0	0	
1874	1	0	1	0	0.591549	1	1	
1175	1	0	0	0	0.169014	1	1	
...	...	...	...	...	...	...	...	
7021	0	0	0	0	0.154930	1	0	
7026	1	0	0	0	0.112676	1	0	
7032	0	1	0	0	0.000000	1	1	
7034	1	0	0	0	0.929577	1	1	
7041	0	1	1	0	0.042254	1	1	

3738 rows × 27 columns

In [89]:
```
df_test_balanced.shape
```

Out[89]: (3738, 27)

In [90]:
```
2*1869
```

Out[90]: 3738

In [162]:
```
df_test_balanced.Churn.value_counts()
```

Out[162]:
```
0    1869
1    1869
Name: Churn, dtype: int64
```

In [97]:
```
x2 = df_test_balanced.drop('Churn',axis = 'columns')
y2 = df_test_balanced.Churn
```

In [98]:
```
y2
```

Out[98]:
```
283     0
4076    0
107     0
1874    0
1175    0
       ..
7021    1
7026    1
7032    1
7034    1
7041    1
Name: Churn, Length: 3738, dtype: int64
```

In [101]:
```
from sklearn.model_selection import train_test_split

x2_train , x2_test , y2_train , y2_test = train_test_split(x2,y2,test_size=0.2 ,
```

In [102]:
```
y_pred2 = ANN(x2_train , y2_train , x2_test,y2_test , 'binary_crossentropy',-1)
```

```
94/94 [==============================] - 0s 2ms/step - loss: 0.3878 - accurac
y: 0.8207
Epoch 95/100
94/94 [==============================] - 0s 2ms/step - loss: 0.3873 - accurac
y: 0.8247
Epoch 96/100
94/94 [==============================] - 0s 2ms/step - loss: 0.3865 - accurac
y: 0.8244
Epoch 97/100
94/94 [==============================] - 0s 2ms/step - loss: 0.3864 - accurac
y: 0.8227
Epoch 98/100
94/94 [==============================] - 0s 2ms/step - loss: 0.3845 - accurac
y: 0.8254
Epoch 99/100
94/94 [==============================] - 0s 2ms/step - loss: 0.3862 - accurac
y: 0.8247
Epoch 100/100
94/94 [==============================] - 0s 2ms/step - loss: 0.3833 - accurac
y: 0.8241
```

you see that our precision , recall and f1 score is improved

method2) Oversampling

In [103]:
```
count_class_0 , count_class_1
```

Out[103]:
```
(5163, 1869)
```

count_class_1 is data frame which has less data so i should oversample it

In [106]:
```
df2_class_1.shape
```

Out[106]:
```
(1869, 27)
```

In [107]:
```
df2_class_1.sample(2000 , replace=True).shape
```

Out[107]:
```
(2000, 27)
```

df2_class_1 has only 1869 samples but when we want 2000 samples it will randomly select some samples and copy it to reach the size of 2000 sample

In [110]:
```
df2_class_1.sample(count_class_0 , replace=True).shape
```

Out[110]:
```
(5163, 27)
```

In [111]:
```
df_class_1_oversampling = df2_class_1.sample(count_class_0 , replace=True)
```

In [112]:
```
df_class_1_oversampling.shape
```

Out[112]:
```
(5163, 27)
```

In [115]:
```
df2_class_0.shape
```

Out[115]:
```
(5163, 27)
```

now lets concat them to have one dataframe

In [120]:
```
df_test_oversampling = pd.concat([df_class_1_oversampling,df2_class_0], axis = 0)
df_test_oversampling
```

Out[120]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	OnlineSe
3879	0	0	0	0	0.000000	1	0	
5947	1	0	1	1	0.450704	1	1	
6215	1	1	0	0	0.070423	1	1	
2282	0	1	1	1	0.915493	1	1	
809	0	0	0	0	0.000000	1	0	
...	...	...	...	...	...	...	...	
7037	1	0	0	0	1.000000	1	0	
7038	0	0	1	1	0.323944	1	1	
7039	1	0	1	1	1.000000	1	1	
7040	1	0	1	1	0.140845	0	0	
7042	0	0	0	0	0.915493	1	0	

10326 rows × 27 columns

In [121]:
```
df_test_oversampling.shape
```

Out[121]:
```
(10326, 27)
```

In [123]:
```
df_test_oversampling.Churn.value_counts()
```

Out[123]:
```
1    5163
0    5163
Name: Churn, dtype: int64
```

In [134]:
```
x3 = df_test_oversampling.drop('Churn',axis = 'columns')
y3 = df_test_oversampling.Churn
```

In [135]:
```
rom sklearn.model_selection import train_test_split

3_train , x3_test , y3_train , y3_test = train_test_split(x3,y3,test_size=0.2 , ra
```

In [136]:
```
y_pred3 = ANN(x3_train , y3_train , x3_test,y3_test , 'binary_crossentropy',-1)
```

```
259/259 [==============================] - 0s 2ms/step - loss: 0.3850 - accur
acy: 0.8214
Epoch 85/100
259/259 [==============================] - 0s 2ms/step - loss: 0.3848 - accur
acy: 0.8228
Epoch 86/100
259/259 [==============================] - 0s 2ms/step - loss: 0.3841 - accur
acy: 0.8225
Epoch 87/100
259/259 [==============================] - 0s 2ms/step - loss: 0.3830 - accur
acy: 0.8243
Epoch 88/100
259/259 [==============================] - 0s 2ms/step - loss: 0.3816 - accur
acy: 0.8269
Epoch 89/100
259/259 [==============================] - 0s 2ms/step - loss: 0.3820 - accur
acy: 0.8254
Epoch 90/100
259/259 [==============================] - 0s 2ms/step - loss: 0.3817 - accur
acy: 0.8240
```

method3) SMOTE

(synthetic minority oversampling technique)

In [130]:
```
x4 = df2.drop('Churn',axis = 'columns')
y4 = df2.Churn
```

In [146]:
```
y4.value_counts()
```

Out[146]:
```
0    5163
1    1869
Name: Churn, dtype: int64
```

In [152]:
```
#pip install imblearn   google search :(imbalanced learn)
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy = 'minority')
x_sm ,y_sm = smote.fit_resample(x4,y4)
```

In [153]:
```
y_sm.value_counts()
```

Out[153]:
```
0    5163
1    5163
Name: Churn, dtype: int64
```

In [155]:
```
from sklearn.model_selection import train_test_split

x4_train , x4_test , y4_train , y4_test = train_test_split(x_sm,y_sm,test_size=0.
```

In [157]:
```
y_pred4 = ANN(x4_train , y4_train , x4_test,y4_test , 'binary_crossentropy',-1)
```

```
acy: 0.8462
Epoch 85/100
259/259 [==============================] - 1s 2ms/step - loss: 0.3451 - accur
acy: 0.8460
Epoch 86/100
259/259 [==============================] - 1s 2ms/step - loss: 0.3438 - accur
acy: 0.8466
Epoch 87/100
259/259 [==============================] - 1s 2ms/step - loss: 0.3459 - accur
acy: 0.8462
Epoch 88/100
259/259 [==============================] - 1s 2ms/step - loss: 0.3456 - accur
acy: 0.8478
Epoch 89/100
259/259 [==============================] - 1s 2ms/step - loss: 0.3456 - accur
acy: 0.8456
Epoch 90/100
259/259 [==============================] - 1s 2ms/step - loss: 0.3415 - accur
acy: 0.8488
```

method4) Use of Ensemble with understanding

In [229]:
```
df2.Churn.value_counts()
```

Out[229]:
```
0    5163
1    1869
Name: Churn, dtype: int64
```

In [230]:
```
x = df2.drop('Churn',axis = 'columns')
y = df2.Churn
```

In [231]:
```
.model_selection import train_test_split

_test , y_train , y_test = train_test_split(x,y,test_size=0.2 , random_state = 2)
```

In [232]:
```
y5_train.value_counts()
```

Out[232]:
```
0    3304
Name: Churn, dtype: int64
```

In [233]:
```
4130/1495
```

Out[233]:
```
2.762541806020067
```

In [234]:
```
4130/3
```

Out[234]:
```
1376.6666666666667
```

#batch1 : 1495 / batch2 : 1495 / batch3 :1140

In [235]:
```
df3 = x_train.copy()
df3['Churn'] = y_train
```

In [236]:
```
df3_class0 = df3[df3.Churn==0]
df3_class1 = df3[df3.Churn==1]
```

In [237]:
```
df3_class0.shape , df3_class1.shape
```

Out[237]:
```
((4130, 27), (1495, 27))
```

So class0 is our major class

In [238]:
```
def get_train_batch(df_minority,df_majority,start , end):
    df_train = pd.concat([df_majority[start:end],df_minority] , axis=0)

    x_train = df_train.drop('Churn' , axis='columns')
    y_train = df_train.Churn

    return x_train , y_train
```

In [239]:
```
x_train , y_train = get_train_batch(df3_class0 ,df3_class1,0,1495 )
```

In [240]:
```
x_train.shape
```

Out[240]:
```
(5625, 26)
```

In [241]:
```
y_train.shape
```

Out[241]:
```
(5625,)
```

In [242]:
```
df3_class0.shape , df3_class1.shape
```

Out[242]:
```
((4130, 27), (1495, 27))
```

In [243]:
```
from sklearn.model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split(x_train,y_train, test_size
```

Print result for batch1

In [244]:
```
y_pred_1 = ANN(x_train , y_train , x_test,y_test , 'binary_crossentropy',-1)
```

```
141/141 [==============================] - 0s 2ms/step - loss: 0.3523 - accur
acy: 0.8318
Epoch 85/100
141/141 [==============================] - 0s 2ms/step - loss: 0.3529 - accur
acy: 0.8316
Epoch 86/100
141/141 [==============================] - 0s 2ms/step - loss: 0.3522 - accur
acy: 0.8327
Epoch 87/100
141/141 [==============================] - 0s 2ms/step - loss: 0.3515 - accur
acy: 0.8336
Epoch 88/100
141/141 [==============================] - 0s 2ms/step - loss: 0.3504 - accur
acy: 0.8344
Epoch 89/100
141/141 [==============================] - 0s 2ms/step - loss: 0.3503 - accur
acy: 0.8322
Epoch 90/100
141/141 [==============================] - 0s 2ms/step - loss: 0.3485 - accur
acy: 0.8362
```

Print result for batch2

In [245]:
```
x_train , y_train = get_train_batch(df3_class0 ,df3_class1,1495,2990 )
y_pred_2 = ANN(x_train , y_train , x_test, y_test , 'binary_crossentropy',-1)
```

. . .

Print result for batch3

In [246]:
```
x_train , y_train = get_train_batch(df3_class0 ,df3_class1,2990,4130 )
```

In [248]:
```
y_pred5 = ANN(x_train , y_train , x_test,y_test , 'binary_crossentropy',-1)
```

. . .