Deep learning I Project 1 I House Loan

For safe and secure lending experience, it's important to analyze the past data. In this project, you have to build a deep learning model to predict the chance of default for future loans using the historical data. As you will see, this dataset is highly imbalanced and includes a lot of features that make this problem more challenging.

Objective: Create a model that predicts whether or not an applicant will be able to repay a loan using historical data.

In [1]:

```
# import data and required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
hloan=pd.read_csv("F:\Bipasha\loan_data.csv")
pd.options.display.max_columns = None
hloan.head(5)
```

Out[2]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002	1	Cash loans	M	N	Y	0
1	100003	0	Cash loans	F	N	N	0
2	100004	0	Revolving loans	M	Y	Y	0
3	100006	0	Cash loans	F	N	Y	0
4	100007	0	Cash loans	M	N	Y	0

In [3]:

```
hloan.shape    # dimension check
```

Out[3]:

```
(307511, 122)
```

In [4]:

```
hloan.info()    # data type check
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
```

In [5]:

```
pd.options.display.max_rows = None
hloan.columns     # check all column names
```

Out[5]:

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

In [6]:

```
hloan.describe()    # summary of data
```

Out[6]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRI
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e-
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e-
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e-
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e-
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e-
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e-
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e-
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e-

In [7]:

```
hloan.describe(include='O')    # summary of object columns
```

Out[7]:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_TYPE_SUITE	NAME_INCOMI
count	307511	307511	307511	307511	306219	
unique	2	3	2	2	7	
top	Cash loans	F	N	Y	Unaccompanied	W
freq	278232	202448	202924	213312	248526	

In [8]:

```
hloan.set_index(keys=['SK_ID_CURR'],inplace=True)    # set column 'SK_ID_CURR' as index
```

Missing value handle

In [9]:

```
hloan.isna().sum().head(4)   # sum of null values
```

Out[9]:

```
TARGET                0
NAME_CONTRACT_TYPE    0
CODE_GENDER           0
FLAG_OWN_CAR          0
dtype: int64
```

In [10]:

```
(hloan.isna().sum()/hloan.shape[0])*100   # percentage of null values in each column
```

Out[10]:

```
TARGET                          0.000000
NAME_CONTRACT_TYPE              0.000000
CODE_GENDER                     0.000000
FLAG_OWN_CAR                    0.000000
FLAG_OWN_REALTY                 0.000000
CNT_CHILDREN                    0.000000
AMT_INCOME_TOTAL                0.000000
AMT_CREDIT                      0.000000
AMT_ANNUITY                     0.003902
AMT_GOODS_PRICE                 0.090403
NAME_TYPE_SUITE                 0.420148
NAME_INCOME_TYPE                0.000000
NAME_EDUCATION_TYPE             0.000000
NAME_FAMILY_STATUS              0.000000
NAME_HOUSING_TYPE               0.000000
REGION_POPULATION_RELATIVE      0.000000
DAYS_BIRTH                      0.000000
DAYS_EMPLOYED                   0.000000
DAYS_REGISTRATION               0.000000
DAYS_ID_PUBLISH                 0.000000
OWN_CAR_AGE                    65.990810
FLAG_MOBIL                      0.000000
FLAG_EMP_PHONE                  0.000000
FLAG_WORK_PHONE                 0.000000
FLAG_CONT_MOBILE                0.000000
FLAG_PHONE                      0.000000
FLAG_EMAIL                      0.000000
OCCUPATION_TYPE                31.345545
CNT_FAM_MEMBERS                 0.000650
REGION_RATING_CLIENT            0.000000
REGION_RATING_CLIENT_W_CITY     0.000000
WEEKDAY_APPR_PROCESS_START      0.000000
HOUR_APPR_PROCESS_START         0.000000
REG_REGION_NOT_LIVE_REGION      0.000000
REG_REGION_NOT_WORK_REGION      0.000000
LIVE_REGION_NOT_WORK_REGION     0.000000
REG_CITY_NOT_LIVE_CITY          0.000000
REG_CITY_NOT_WORK_CITY          0.000000
LIVE_CITY_NOT_WORK_CITY         0.000000
ORGANIZATION_TYPE               0.000000
EXT_SOURCE_1                   56.381073
EXT_SOURCE_2                    0.214626
EXT_SOURCE_3                   19.825307
APARTMENTS_AVG                 50.749729
BASEMENTAREA_AVG               58.515956
YEARS_BEGINEXPLUATATION_AVG    48.781019
YEARS_BUILD_AVG                66.497784
COMMONAREA_AVG                 69.872297
ELEVATORS_AVG                  53.295980
ENTRANCES_AVG                  50.348768
FLOORSMAX_AVG                  49.760822
FLOORSMIN_AVG                  67.848630
LANDAREA_AVG                   59.376738
LIVINGAPARTMENTS_AVG           68.354953
LIVINGAREA_AVG                 50.193326
NONLIVINGAPARTMENTS_AVG        69.432963
NONLIVINGAREA_AVG              55.179164
APARTMENTS_MODE                50.749729
BASEMENTAREA_MODE              58.515956
YEARS_BEGINEXPLUATATION_MODE   48.781019
YEARS_BUILD_MODE               66.497784
COMMONAREA_MODE                69.872297
ELEVATORS_MODE                 53.295980
ENTRANCES_MODE                 50.348768
FLOORSMAX_MODE                 49.760822
FLOORSMIN_MODE                 67.848630
```

```
LANDAREA_MODE                   59.376738
LIVINGAPARTMENTS_MODE           68.354953
LIVINGAREA_MODE                 50.193326
NONLIVINGAPARTMENTS_MODE        69.432963
NONLIVINGAREA_MODE              55.179164
APARTMENTS_MEDI                 50.749729
BASEMENTAREA_MEDI               58.515956
YEARS_BEGINEXPLUATATION_MEDI    48.781019
YEARS_BUILD_MEDI                66.497784
COMMONAREA_MEDI                 69.872297
ELEVATORS_MEDI                  53.295980
ENTRANCES_MEDI                  50.348768
FLOORSMAX_MEDI                  49.760822
FLOORSMIN_MEDI                  67.848630
LANDAREA_MEDI                   59.376738
LIVINGAPARTMENTS_MEDI           68.354953
LIVINGAREA_MEDI                 50.193326
NONLIVINGAPARTMENTS_MEDI        69.432963
NONLIVINGAREA_MEDI              55.179164
FONDKAPREMONT_MODE              68.386172
HOUSETYPE_MODE                  50.176091
TOTALAREA_MODE                  48.268517
WALLSMATERIAL_MODE              50.840783
EMERGENCYSTATE_MODE             47.398304
OBS_30_CNT_SOCIAL_CIRCLE         0.332021
DEF_30_CNT_SOCIAL_CIRCLE         0.332021
OBS_60_CNT_SOCIAL_CIRCLE         0.332021
DEF_60_CNT_SOCIAL_CIRCLE         0.332021
DAYS_LAST_PHONE_CHANGE           0.000325
FLAG_DOCUMENT_2                  0.000000
FLAG_DOCUMENT_3                  0.000000
FLAG_DOCUMENT_4                  0.000000
FLAG_DOCUMENT_5                  0.000000
FLAG_DOCUMENT_6                  0.000000
FLAG_DOCUMENT_7                  0.000000
FLAG_DOCUMENT_8                  0.000000
FLAG_DOCUMENT_9                  0.000000
FLAG_DOCUMENT_10                 0.000000
FLAG_DOCUMENT_11                 0.000000
FLAG_DOCUMENT_12                 0.000000
FLAG_DOCUMENT_13                 0.000000
FLAG_DOCUMENT_14                 0.000000
FLAG_DOCUMENT_15                 0.000000
FLAG_DOCUMENT_16                 0.000000
FLAG_DOCUMENT_17                 0.000000
FLAG_DOCUMENT_18                 0.000000
FLAG_DOCUMENT_19                 0.000000
FLAG_DOCUMENT_20                 0.000000
FLAG_DOCUMENT_21                 0.000000
AMT_REQ_CREDIT_BUREAU_HOUR      13.501631
AMT_REQ_CREDIT_BUREAU_DAY       13.501631
AMT_REQ_CREDIT_BUREAU_WEEK      13.501631
AMT_REQ_CREDIT_BUREAU_MON       13.501631
AMT_REQ_CREDIT_BUREAU_QRT       13.501631
AMT_REQ_CREDIT_BUREAU_YEAR      13.501631
dtype: float64
```

We impute null values with mean and mode. For numerical type columns we fill mean and object type we fill mode.

In [11]:

```
col=hloan.columns
coln=[]
colo=[]
for i in col:
    if hloan[i].isna().sum()!=0:
        if hloan[i].dtype=='object':
            colo.append(i)
            hloan[i].fillna(hloan[i].mode()[0],inplace=True)
```

```
        else:
            coln.append(i)
            hloan[i].fillna(hloan[i].mean(),inplace=True)
```

In [12]:

```
hloan.head(2)
```

Out[12]:

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AI
SK_ID_CURR							
100002	1	Cash loans	M	N	Y	0	
100003	0	Cash loans	F	N	N	0	

In [13]:

```
hloan.isnull().any().sum()    # Check if there is any null value
```

Out[13]:

```
0
```

In [14]:

```
hloan['TARGET']=hloan['TARGET'].astype('category')   # make target column as category type
```

In [15]:

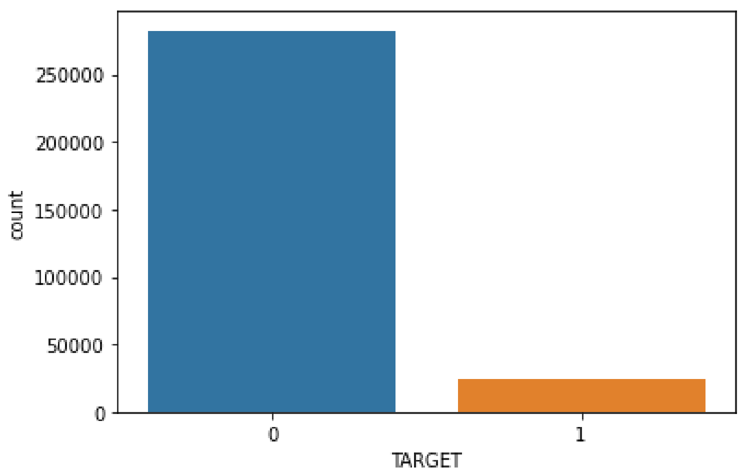
```
hloan['TARGET'].dtype
```

Out[15]:

```
CategoricalDtype(categories=[0, 1], ordered=False)
```

In [16]:

```
# Frequency plot of output column
sns.countplot(hloan['TARGET'])
plt.show()
```



In [17]:

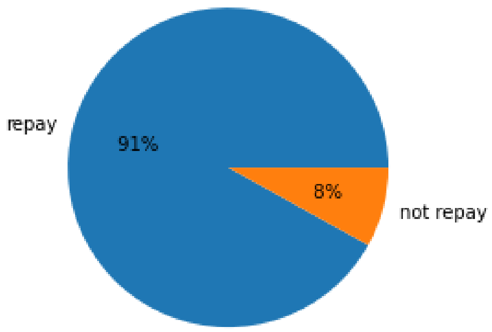
```
hloan['TARGET'].value_counts()
```

Out[17]:

```
0    282686
1     24825
Name: TARGET, dtype: int64
```

In [18]:

```
# Pie plot of target column
plt.pie(hloan['TARGET'].value_counts(),autopct="%3d%%",labels=['repay','not repay'])
plt.show()
```



Here 91% customers repaid house loan and 8% customers did not repay loan.

In [19]:

```
notrepay = hloan[hloan.TARGET == 1]
repay = hloan[hloan.TARGET == 0]
print(notrepay.shape)
print(repay.shape)
```

```
(24825, 121)
(282686, 121)
```

Encoding

In [20]:

```
# Import labelencoder to convert all object columns in labelwise category.
from sklearn.preprocessing import LabelEncoder
```

In [21]:

```
lb=LabelEncoder()
v=[]
for i in col:
    if hloan[i].dtype=='object':
        v.append(i)
        hloan[i]=lb.fit_transform(hloan[i])
```

In [22]:

```
hloan.head(5)
```

Out[22]:

	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AI
SK_ID_CURR							
100002	1	0	1	0	1	0	
100003	0	0	0	0	0	0	
100004	0	1	1	1	1	0	
100006	0	0	0	0	1	0	
100007	0	0	1	0	1	0	

Balance data

Undersampling data

In [23]:

```
# Resample data and make a balanced data. So we undersample data as we have enough rows.
from imblearn.under_sampling import RandomUnderSampler
```

In [24]:

```
# Seperate input and output variables.
X=hloan.drop(['TARGET'],axis=1)
y=hloan['TARGET']
```

In [25]:

```
rus=RandomUnderSampler(random_state=0)
X_res,y_res= rus.fit_resample(X, y)
```

In [26]:

```
# Shape of resampling data.
print(X_res.shape)
print(y_res.shape)
```

```
(49650, 120)
(49650,)
```

In [27]:

```
# Split data into training and testing part.
from sklearn.model_selection import train_test_split
```

In [28]:

```
X_train,X_test,y_train,y_test= train_test_split(X_res,y_res,test_size=0.2,random_state=1
2)
print(X_train.shape)
print(X_test.shape)
```

```
(39720, 120)
(9930, 120)
```

In [29]:

```
y_train.value_counts()
```

Out[29]:

```
0    19887
1    19833
Name: TARGET, dtype: int64
```

In [30]:

```
y_test.value_counts()
```

Out[30]:

```
1    4992
0    4938
Name: TARGET, dtype: int64
```

Scale data

In [31]:

```
# Scale data as range differ very much in each column.
from sklearn.preprocessing import StandardScaler
```

```
sc=StandardScaler()
X_train_scale=sc.fit_transform(X_train)
X_test_scale=sc.transform(X_test)
```

Model Building

In [32]:

```
# Import tensorflow and required libraries for model building. Build a simple deep learni
ng model.
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

In [33]:

```
model=Sequential()
```

In [34]:

```
model.add(Dense(units=128,activation='tanh',input_shape=(X_train_scale.shape[1],)))
model.add(Dropout(0.2))
```

In [35]:

```
model.add(Dense(units=64,activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(units=32,activation='tanh'))
model.add(Dropout(0.2))
model.add(Dense(units=16,activation='tanh'))
model.add(Dense(units=1,activation='sigmoid'))
```

In [36]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 128)               15488

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 32)                2080

 dropout_2 (Dropout)         (None, 32)                0

 dense_3 (Dense)             (None, 16)                528

 dense_4 (Dense)             (None, 1)                 17

=================================================================
Total params: 26,369
Trainable params: 26,369
Non-trainable params: 0
_________________________________________________________________
```

In [37]:

```
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accuracy'])
```

In [38]:

```
result=model.fit(X_train_scale,y_train,epochs=100,verbose=1,batch_size=16)
```

```
Epoch 1/100
2483/2483 [==============================] - 9s 3ms/step - loss: 0.6214 - accuracy: 0.659
6
Epoch 2/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.6110 - accuracy: 0.671
7
Epoch 3/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.6076 - accuracy: 0.675
9
Epoch 4/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.6049 - accuracy: 0.676
6
Epoch 5/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.6018 - accuracy: 0.677
2
Epoch 6/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5995 - accuracy: 0.679
7
Epoch 7/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5968 - accuracy: 0.680
0
Epoch 8/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5952 - accuracy: 0.683
4
Epoch 9/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5915 - accuracy: 0.687
0
Epoch 10/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5902 - accuracy: 0.686
3
Epoch 11/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5873 - accuracy: 0.689
9
Epoch 12/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5849 - accuracy: 0.691
0
Epoch 13/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5813 - accuracy: 0.69
26
Epoch 14/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5799 - accuracy: 0.69
47
Epoch 15/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5780 - accuracy: 0.69
54
Epoch 16/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5766 - accuracy: 0.69
81
Epoch 17/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5740 - accuracy: 0.70
07
Epoch 18/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5704 - accuracy: 0.70
26
Epoch 19/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5689 - accuracy: 0.70
27
Epoch 20/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5670 - accuracy: 0.70
65
Epoch 21/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5653 - accuracy: 0.70
68
Epoch 22/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5639 - accuracy: 0.70
76
Epoch 23/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5624 - accuracy: 0.70
89
Epoch 24/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5582 - accuracy: 0.71
```

```
37
Epoch 25/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5574 - accuracy: 0.71
32
Epoch 26/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5568 - accuracy: 0.71
50
Epoch 27/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5553 - accuracy: 0.71
55
Epoch 28/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5535 - accuracy: 0.716
1
Epoch 29/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5512 - accuracy: 0.71
98
Epoch 30/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5485 - accuracy: 0.72
15
Epoch 31/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5479 - accuracy: 0.72
10
Epoch 32/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5472 - accuracy: 0.72
20
Epoch 33/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5454 - accuracy: 0.72
49
Epoch 34/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5450 - accuracy: 0.72
40
Epoch 35/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5426 - accuracy: 0.72
54
Epoch 36/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5428 - accuracy: 0.72
47
Epoch 37/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5405 - accuracy: 0.72
51
Epoch 38/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5371 - accuracy: 0.73
06
Epoch 39/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5358 - accuracy: 0.72
99
Epoch 40/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5368 - accuracy: 0.73
11
Epoch 41/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5385 - accuracy: 0.72
77
Epoch 42/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5358 - accuracy: 0.72
75
Epoch 43/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5332 - accuracy: 0.73
14
Epoch 44/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5324 - accuracy: 0.73
24
Epoch 45/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5313 - accuracy: 0.73
54
Epoch 46/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5311 - accuracy: 0.73
45
Epoch 47/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5305 - accuracy: 0.73
44
Epoch 48/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5258 - accuracy: 0.73
```

```
59
Epoch 49/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5286 - accuracy: 0.73
56
Epoch 50/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5275 - accuracy: 0.737
0
Epoch 51/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5258 - accuracy: 0.737
7
Epoch 52/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5224 - accuracy: 0.738
6
Epoch 53/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5231 - accuracy: 0.73
91
Epoch 54/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5236 - accuracy: 0.738
1
Epoch 55/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5210 - accuracy: 0.740
4
Epoch 56/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5203 - accuracy: 0.74
23
Epoch 57/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5221 - accuracy: 0.74
03
Epoch 58/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5217 - accuracy: 0.741
3
Epoch 59/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5183 - accuracy: 0.74
29
Epoch 60/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5208 - accuracy: 0.741
9
Epoch 61/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5186 - accuracy: 0.743
7
Epoch 62/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5149 - accuracy: 0.745
9
Epoch 63/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5200 - accuracy: 0.741
4
Epoch 64/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5163 - accuracy: 0.744
4
Epoch 65/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5165 - accuracy: 0.746
5
Epoch 66/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5154 - accuracy: 0.745
6
Epoch 67/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5157 - accuracy: 0.74
43
Epoch 68/100
2483/2483 [==============================] - 6s 3ms/step - loss: 0.5150 - accuracy: 0.745
3
Epoch 69/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5127 - accuracy: 0.747
4
Epoch 70/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5108 - accuracy: 0.74
93
Epoch 71/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5120 - accuracy: 0.74
77
Epoch 72/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5097 - accuracy: 0.74
```

```
90
Epoch 73/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5104 - accuracy: 0.749
0
Epoch 74/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5107 - accuracy: 0.748
1
Epoch 75/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5097 - accuracy: 0.749
5
Epoch 76/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5070 - accuracy: 0.75
31
Epoch 77/100
2483/2483 [==============================] - 7s 3ms/step - loss: 0.5078 - accuracy: 0.751
2
Epoch 78/100
2483/2483 [==============================] - 6s 3ms/step - loss: 0.5076 - accuracy: 0.751
4
Epoch 79/100
2483/2483 [==============================] - 9s 3ms/step - loss: 0.5096 - accuracy: 0.749
5
Epoch 80/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5094 - accuracy: 0.750
8
Epoch 81/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5076 - accuracy: 0.751
9
Epoch 82/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5061 - accuracy: 0.751
8
Epoch 83/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5051 - accuracy: 0.752
8
Epoch 84/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5077 - accuracy: 0.751
5
Epoch 85/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5028 - accuracy: 0.75
45
Epoch 86/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5055 - accuracy: 0.75
39
Epoch 87/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5040 - accuracy: 0.75
31
Epoch 88/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5031 - accuracy: 0.75
22
Epoch 89/100
2483/2483 [==============================] - 11s 4ms/step - loss: 0.5052 - accuracy: 0.75
36
Epoch 90/100
2483/2483 [==============================] - 10s 4ms/step - loss: 0.5030 - accuracy: 0.75
60
Epoch 91/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5009 - accuracy: 0.756
4
Epoch 92/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5017 - accuracy: 0.757
7
Epoch 93/100
2483/2483 [==============================] - 9s 4ms/step - loss: 0.5042 - accuracy: 0.755
0
Epoch 94/100
2483/2483 [==============================] - 7s 3ms/step - loss: 0.5008 - accuracy: 0.757
4
Epoch 95/100
2483/2483 [==============================] - 6s 3ms/step - loss: 0.5003 - accuracy: 0.757
3
Epoch 96/100
2483/2483 [==============================] - 7s 3ms/step - loss: 0.5017 - accuracy: 0.753
```

```
9
Epoch 97/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.5002 - accuracy: 0.757
7
Epoch 98/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.4969 - accuracy: 0.759
1
Epoch 99/100
2483/2483 [==============================] - 8s 3ms/step - loss: 0.4972 - accuracy: 0.757
0
Epoch 100/100
2483/2483 [==============================] - 9s 3ms/step - loss: 0.4990 - accuracy: 0.757
6
```

In [39]:

```
score=model.evaluate(X_test_scale,y_test,verbose=0)
score
```

Out[39]:

```
[0.6535560488700867, 0.6671701669692993]
```

In [40]:

```
# Model prediction
y_train_pred=model.predict(X_train_scale)
y_test_pred=model.predict(X_test_scale)
```

In [41]:

```
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [42]:

```
confusion_matrix(y_train_pred > 0.5,y_train)
```

Out[42]:

```
array([[15754,  3555],
       [ 4133, 16278]], dtype=int64)
```

In [43]:

```
accuracy_score(y_train_pred >0.5 ,y_train)
# accuracy is good for training data
```

Out[43]:

```
0.8064451158106747
```

In [44]:

```
confusion_matrix(y_test_pred >0.5 ,y_test)
```

Out[44]:

```
array([[3244, 1611],
       [1694, 3381]], dtype=int64)
```

In [45]:

```
accuracy_score(y_test_pred >0.5 ,y_test)
# Testing data accuracy is not so good means it is overfitted.
```

Out[45]:

```
0.6671701913393756
```

In [46]:

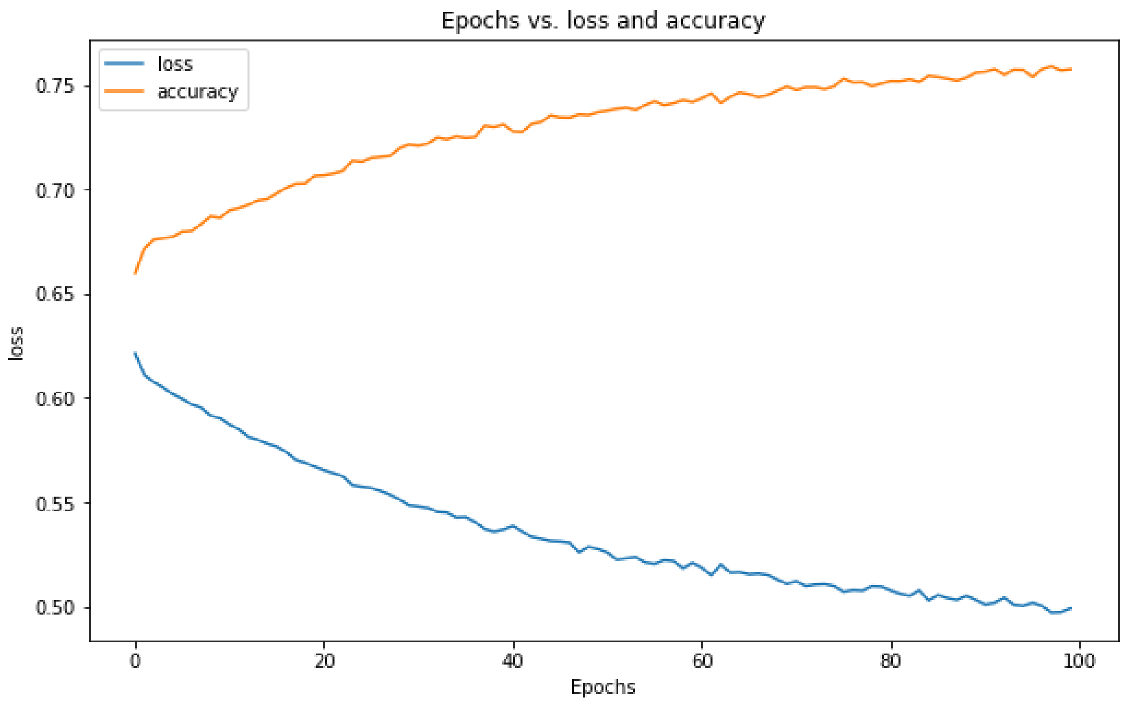
```
history=pd.DataFrame(result.history)
history.head(5)
```

Out[46]:

	loss	accuracy
0	0.621380	0.659643
1	0.611009	0.671727
2	0.607560	0.675856
3	0.604891	0.676586
4	0.601789	0.677216

In [47]:

```
# Epochs vs. loss and accuracy graph. Increase number of epochs accuracy increases and lo
ss decreases.
plt.figure(figsize=(10,6))
plt.plot(history.loss,label='loss')
plt.plot(history.accuracy,label='accuracy')
plt.title("Epochs vs. loss and accuracy")
plt.xlabel("Epochs")
plt.ylabel("loss")
plt.legend()
plt.show()
```



In [48]:

```
from sklearn.metrics import (precision_recall_curve,auc,roc_curve,recall_score,precision_
score,
classification_report,roc_auc_score)
```

In [49]:

```
mse = np.mean(np.power(X_test_scale - y_test_pred, 2), axis=1)
```

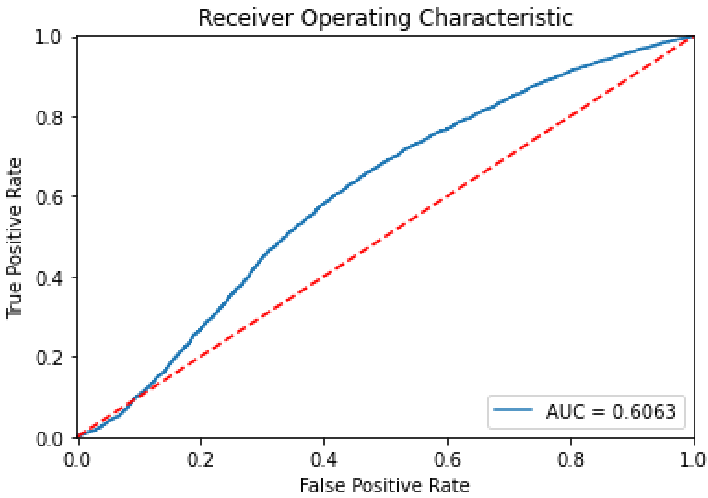
In [50]:

```
fpr, tpr, thresholds = roc_curve(y_test, mse)
roc_auc = auc(fpr, tpr)
```

In [51]:

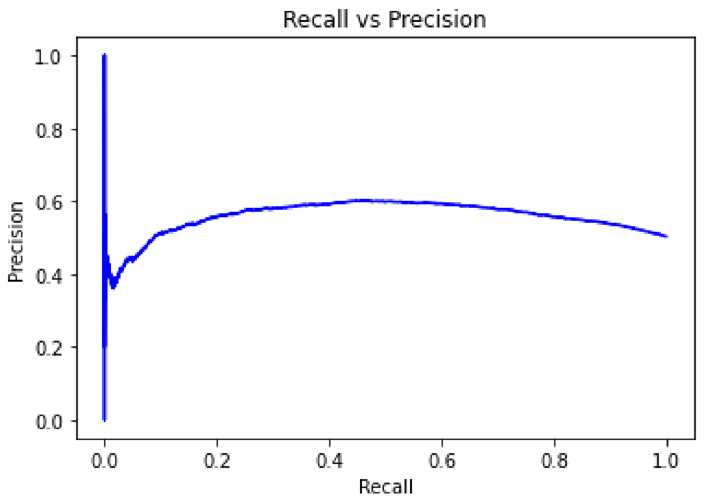
```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='AUC = %0.4f'% roc_auc)
plt.legend(loc='lower right')
```

```
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.001, 1])
plt.ylim([0, 1.001])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [52]:

```
precision, recall, th = precision_recall_curve(y_test, mse)
plt.plot(recall, precision, 'b', label='Precision-Recall curve')
plt.title('Recall vs Precision')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```



In [53]:

```
# Let's look at overall result of this model
print(classification_report(y_test, y_test_pred>0.5))
```

```
              precision    recall  f1-score   support

           0       0.67      0.66      0.66      4938
           1       0.67      0.68      0.67      4992

    accuracy                           0.67      9930
   macro avg       0.67      0.67      0.67      9930
weighted avg       0.67      0.67      0.67      9930
```

In [54]:

```
roc_auc_score(y_test, y_test_pred)
```

Out[54]:

0.7225745680735999

```
0.7223745680735999
```

ROC score is 0.72 which is good. So, this model can be used for future prediction.

In [ ]:

```
# End
```