Project Description (House Loan Data Analysis):

“””

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.

Domain: Finance

Analysis to be done: Perform data preprocessing and build a deep learning prediction model.

Steps to be done:

- Load the dataset that is given to you
- Check for null values in the dataset
- Print percentage of default to payer of the dataset for the TARGET column
- Balance the dataset if the data is imbalanced
- Plot the balanced data or imbalanced data
- Encode the columns that is required for the model
- Calculate Sensitivity as a metrice
- Calculate area under receiver operating characteristics curve

“””

Source Code:

```
#importing Libraries

import pandas as pd

import numpy as np

import warnings

warnings.filterwarnings('ignore')


from sklearn.preprocessing    import LabelEncoder
```

```
from sklearn.impute          import SimpleImputer

from sklearn.preprocessing    import StandardScaler

from sklearn.model_selection  import train_test_split

from imblearn.over_sampling   import SMOTE

from sklearn.metrics import accuracy_score

import matplotlib.pyplot as plt

import matplotlib

import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.optimizers import adam_v2

import tensorflow as tf


file_path=input("enter path for the loan data file to load:")

house_loan_path=file_path.replace("\\",'/')

house_loan=pd.read_csv(house_loan_path)

print("-------------------------------------------------------")

print("checking for null values")

print(house_loan.isnull().sum())

print("----------------------------------------------------------")

print("percentage of default to payer:")

defaulters=(house_loan.TARGET==1).sum()

payers=(house_loan.TARGET==0).sum()
```

```
print((defaulters/payers)*100)



house_loan= house_loan.drop(['SK_ID_CURR'],axis=1)

house_loan = house_loan[pd.notnull(house_loan['EMERGENCYSTATE_MODE'])]

house_loan = house_loan.loc[house_loan['CODE_GENDER'] != 'XNA']

house_loan['NAME_TYPE_SUITE'] =
house_loan['NAME_TYPE_SUITE'].replace(np.nan,'Other_C')

house_loan['NAME_FAMILY_STATUS'] =
house_loan['NAME_FAMILY_STATUS'].replace('Unknown', 'Married')

house_loan['OCCUPATION_TYPE'] = house_loan['OCCUPATION_TYPE'].replace(np.nan,'Others')

house_loan['WALLSMATERIAL_MODE'] =
house_loan['WALLSMATERIAL_MODE'].replace(np.nan,'Others')

house_loan['HOUSETYPE_MODE'] =
house_loan['HOUSETYPE_MODE'].replace(np.nan,'Unkown')

house_loan['FONDKAPREMONT_MODE'] =
house_loan['FONDKAPREMONT_MODE'].replace(np.nan,'not available')

house_loan = house_loan[pd.notnull(house_loan['AMT_REQ_CREDIT_BUREAU_YEAR'])]

labels = house_loan.describe(include=['object']).columns.values



print("----------------------------------------------------------------")

print("encoding the data")

print("-----------------------------------------------------------------")

le = LabelEncoder()

for lab in labels:

   le.fit(house_loan[lab].values)
```

```
    house_loan[lab] = le.transform(house_loan[lab])
house_loan.info()

null_column = house_loan.columns[house_loan.isnull().any()]
for col in null_column:
   if(house_loan[col].isnull().sum()/house_loan.shape[0]*100 > 39):
      house_loan=house_loan.drop([col],axis=1)

house_loan = house_loan[pd.notnull(house_loan['AMT_ANNUITY'])]



imp1 = SimpleImputer(missing_values= np.nan, strategy='mean')
imp2 = SimpleImputer(missing_values= np.nan, strategy='median')

house_loan[['AMT_GOODS_PRICE','EXT_SOURCE_2',
    'EXT_SOURCE_3','APARTMENTS_AVG',
    'BASEMENTAREA_AVG','YEARS_BEGINEXPLUATATION_AVG',
    'YEARS_BUILD_AVG','ELEVATORS_AVG',
    'ENTRANCES_AVG','FLOORSMAX_AVG',
    'LANDAREA_AVG','LIVINGAREA_AVG',
    'NONLIVINGAREA_AVG','APARTMENTS_MODE',
    'BASEMENTAREA_MODE','YEARS_BEGINEXPLUATATION_MODE',
    'YEARS_BUILD_MODE','ELEVATORS_MODE','ENTRANCES_MODE',
```

```
    'FLOORSMAX_MODE','LANDAREA_MODE','LIVINGAREA_MODE',

    'NONLIVINGAREA_MODE','APARTMENTS_MEDI',

    'BASEMENTAREA_MEDI','BASEMENTAREA_MEDI',

    'YEARS_BEGINEXPLUATATION_MEDI','YEARS_BUILD_MEDI',

    'ELEVATORS_MEDI','ENTRANCES_MEDI','FLOORSMAX_MEDI',

    'LANDAREA_MEDI','LIVINGAREA_MEDI',

    'NONLIVINGAREA_MEDI','TOTALAREA_MODE',]]            =
imp1.fit_transform(house_loan[['AMT_GOODS_PRICE','EXT_SOURCE_2',

                                'EXT_SOURCE_3','APARTMENTS_AVG',

'BASEMENTAREA_AVG','YEARS_BEGINEXPLUATATION_AVG',

                                'YEARS_BUILD_AVG','ELEVATORS_AVG',

                                'ENTRANCES_AVG','FLOORSMAX_AVG',

                                'LANDAREA_AVG','LIVINGAREA_AVG',

                                'NONLIVINGAREA_AVG','APARTMENTS_MODE',

'BASEMENTAREA_MODE','YEARS_BEGINEXPLUATATION_MODE',

'YEARS_BUILD_MODE','ELEVATORS_MODE','ENTRANCES_MODE',

'FLOORSMAX_MODE','LANDAREA_MODE','LIVINGAREA_MODE',

                                'NONLIVINGAREA_MODE','APARTMENTS_MEDI',

                                'BASEMENTAREA_MEDI','BASEMENTAREA_MEDI',

'YEARS_BEGINEXPLUATATION_MEDI','YEARS_BUILD_MEDI',
```

```
'ELEVATORS_MEDI','ENTRANCES_MEDI','FLOORSMAX_MEDI',

                                                    'LANDAREA_MEDI','LIVINGAREA_MEDI',

                                                    'NONLIVINGAREA_MEDI','TOTALAREA_MODE',]]  )




house_loan=house_loan.drop(['FLOORSMIN_AVG', 'FLOORSMIN_MODE',
'FLOORSMIN_MEDI'],axis=1)

house_loan[['CNT_FAM_MEMBERS','OBS_30_CNT_SOCIAL_CIRCLE',

   'DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE',

   'OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE']] =
imp2.fit_transform(house_loan[['CNT_FAM_MEMBERS','OBS_30_CNT_SOCIAL_CIRCLE',


'DEF_30_CNT_SOCIAL_CIRCLE','OBS_60_CNT_SOCIAL_CIRCLE',


'OBS_60_CNT_SOCIAL_CIRCLE','DEF_60_CNT_SOCIAL_CIRCLE']])




print("------------------------------------------------------------")

print("plot of balanced data")



null_columns=house_loan.columns[house_loan.isnull().any()]

var = house_loan.var()[house_loan.var()==0].index.values
```

```
house_loan=house_loan.drop(var,axis=1)


sc = StandardScaler()

house_loan[['AMT_INCOME_TOTAL','AMT_ANNUITY',

   'AMT_CREDIT','AMT_GOODS_PRICE',

   'DAYS_BIRTH','DAYS_EMPLOYED',

   'DAYS_REGISTRATION','DAYS_ID_PUBLISH',

   'DAYS_LAST_PHONE_CHANGE']]             =
sc.fit_transform(house_loan[['AMT_INCOME_TOTAL','AMT_ANNUITY',

                                             'AMT_CREDIT','AMT_GOODS_PRICE',

                                             'DAYS_BIRTH','DAYS_EMPLOYED',

                                             'DAYS_REGISTRATION','DAYS_ID_PUBLISH',

                                             'DAYS_LAST_PHONE_CHANGE']])


corr = house_loan.corr()

import seaborn as sns

sns.heatmap(corr, annot=False, cmap=plt.cm.Reds)

plt.show()


corr_matrix = house_loan.corr().abs()

upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape),k=1).astype(np.bool))
```

```
to_drop = [col for col in upper.columns if any(upper[col]>0.90)]


house_loan = house_loan.drop(house_loan[to_drop], axis=1)


corr = house_loan.corr()

sns.heatmap(corr, annot=False, cmap=plt.cm.Reds)

plt.show()


x = house_loan.drop(['TARGET'], axis=1)

y = house_loan.TARGET


x_train, x_test, y_train, y_test = train_test_split(x, y,test_size= 0.2, random_state= 10, stratify=y)

print(x_train.shape)

print(y_train.shape)

print()

print(y_train.value_counts())


smt = SMOTE(random_state= 10, n_jobs=-1,sampling_strategy='all' )

x_train, y_train = smt.fit_resample(x_train,y_train)


print("--------------------------------------------------------------------------------")

print("evaluating sensitivity and area under receiver operating characteristics curve using cnn model")
```

```
model = Sequential()

model.add(Dense(units= 53,activation = 'relu',input_dim=79)) # first hidden and first input layer

model.add(Dropout(0.2))

model.add(Dense(units= 53,activation = 'relu')) # second hidden layer

model.add(Dropout(0.2))

model.add(Dense(units= 1,activation = 'sigmoid')) # output layer



model.compile(optimizer='adam',loss='binary_crossentropy',metrics=[tf.keras.metrics.SpecificityAtSensitivity(0.5),tf.keras.metrics.AUC()])


model.fit(x_train,y_train,batch_size=10,epochs=20,validation_data=(x_test,y_test))


score = model.evaluate(x_test,y_test)


print('Test Sensitivity : ', score[1])

print('Test AUC : ', score[2])


print("---------------------------------------------------------------------------------------------")
```

Screenshot of the output:

Check for null values in the dataset:

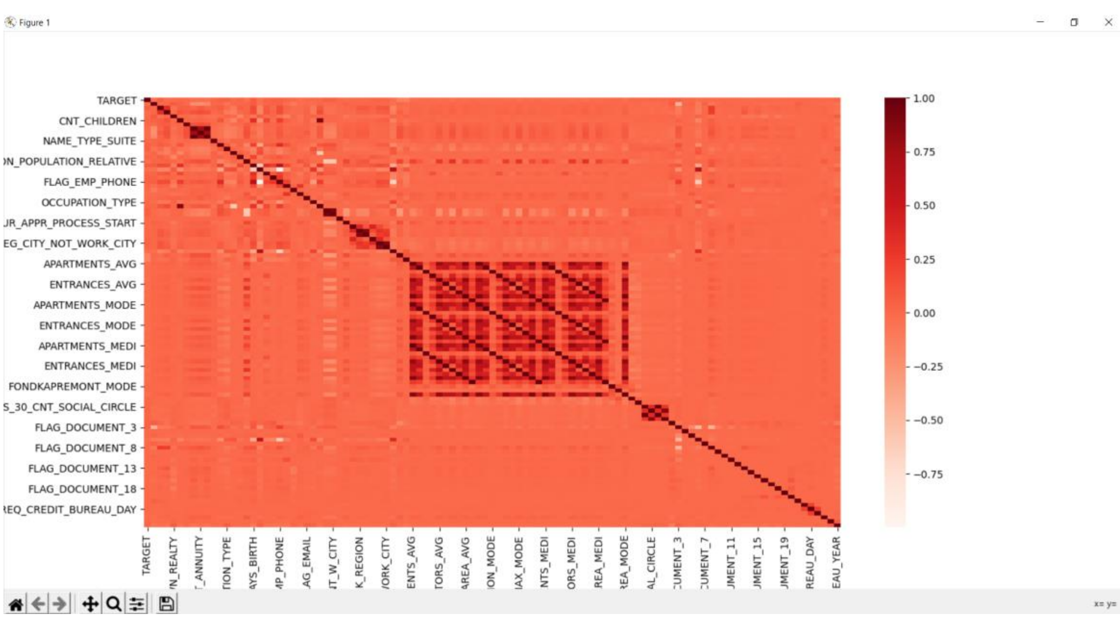
```
------------------------------------------------------------
checking for null values
SK_ID_CURR                        0
TARGET                            0
NAME_CONTRACT_TYPE                0
CODE_GENDER                       0
FLAG_OWN_CAR                      0
                                ...
AMT_REQ_CREDIT_BUREAU_DAY     41519
AMT_REQ_CREDIT_BUREAU_WEEK    41519
AMT_REQ_CREDIT_BUREAU_MON     41519
AMT_REQ_CREDIT_BUREAU_QRT     41519
AMT_REQ_CREDIT_BUREAU_YEAR    41519
Length: 122, dtype: int64
------------------------------------------------------------
```

Print percentage of default to payer of the dataset for the TARGET column:

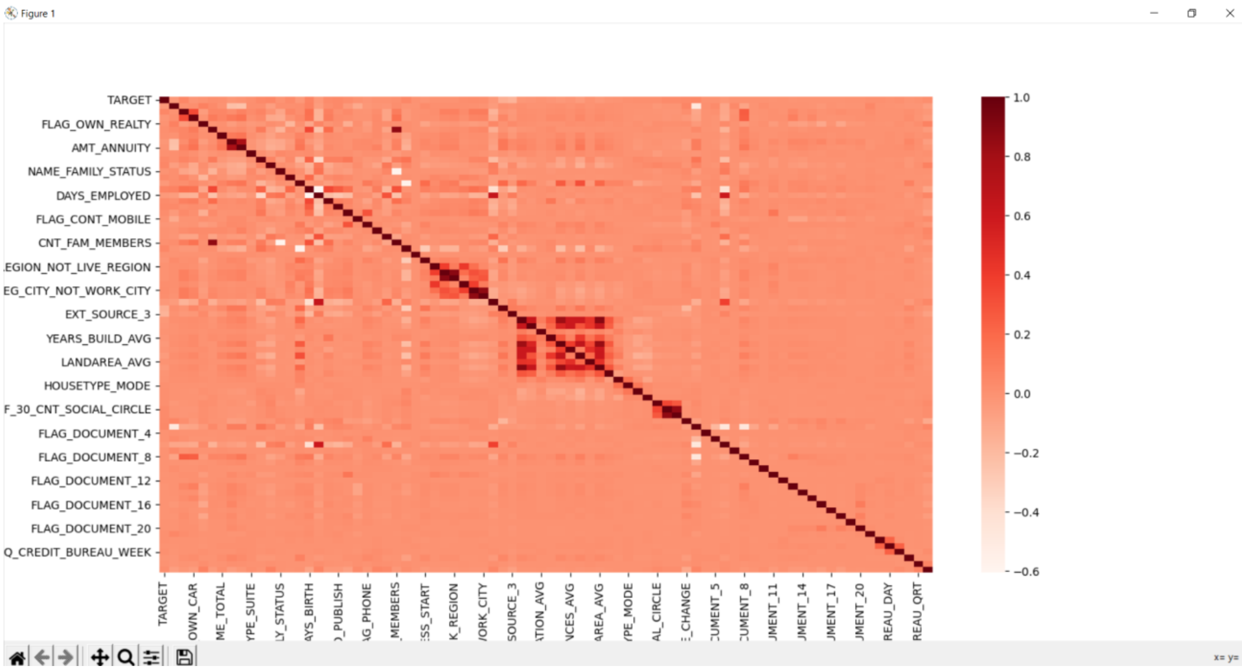
```
------------------------------------------------------------
percentage of default to payer:
8.781828601345662
------------------------------------------------------------
```

Plot the balanced data or imbalanced data:

Imbalanced data:



Balanced Data:



Calculate Sensitivity as a metrice and area under receiver operating characteristics curve:

```
884/884 [==============================] - 0s 548us/step - loss: 0.4291 - specificity_at_sensitivity: 0.6873 - auc: 0.6407
Test Sensitivity :  0.6872724294662476
Test AUC :  0.6407417058944702
```