

# Project to Submit – Project 1

## DESCRIPTION

### House Loan Data Analysis

Course-end Project 1

#### Description

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

You can download the datasets from here

- [https://www.dropbox.com/s/smt43gz12eijbo6/loan\\_data%20%281%29.csv?dl=0](https://www.dropbox.com/s/smt43gz12eijbo6/loan_data%20%281%29.csv?dl=0)

#### Write Up:

##### Analysis Tasks to be performed:

Perform data preprocessing and build a deep learning prediction model.

##### Steps to be done:

1. • Load the dataset that is given to you
2. • Check for null values in the dataset
3. • Print percentage of default to payer of the dataset for the TARGET column
4. • Balance the dataset if the data is imbalanced
5. • Plot the balanced data or imbalanced data
6. • Encode the columns that is required for the model
7. • Calculate Sensitivity as a metrice
8. • Calculate area under receiver operating characteristics curve

#### Screenshots

1. Check for null values in the dataset

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[5]: import cufflinks as cf
cf.go_offline()
import pickle
import gc

[6]: import lightgbm as lgb
warnings.filterwarnings('ignore')

Load the dataset that is given to you Loading the dataset

[7]: import csv
# csv file name
house_loan = pd.read_csv(r'D:\OneDrive\Knowledge Center\AI - ML\Masters in Artificial Engineering\Deep Learning with Keras and Tensorflow\loan_data.csv')

[8]: house_loan.describe()

[9]: house_loan.columns
```

The output of `house_loan.describe()` is as follows:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE	DAYS_BIRTH	DAYS_EMPLO
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072230e+05	307511.000000	307511.000000	307511.000000
mean	278160.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	0.020868	-16036.995067	63815.04
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	0.013831	4363.988632	141275.76
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	0.000290	-25229.000000	-17912.00
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.010006	-19682.000000	-2760.00
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.018850	-15750.000000	-1213.00
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.028663	-12413.000000	-289.00
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508	-7489.000000	365243.00

8 rows x 106 columns

## 2. Check for null values in the dataset

The screenshot shows a Jupyter Notebook interface with the following code and output:

```
[12]: null_values = house_loan.isnull()
null_count = house_loan.isnull().sum()

[13]: print(null_values, null_count)
```

The output of `print(null_values, null_count)` is as follows:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False	False	False	False	False	False	False
307510	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
307506	False	False	False	False	False	False	False	False	False
307507	False	False	False	False	False	False	False	False	False
307508	False	False	False	False	False	False	False	False	False
307509	False	False	False						

```
[14]: null_count
[14]: SK_ID_CURR          0
      TARGET            0
      NAME_CONTRACT_TYPE 0
      CODE_GENDER       0
      FLAG_OWH_CAR       0
      ...
      AHT_REQ_CREDIT_BUREAU_YEAR 41519
      Length: 122, dtype: int64

[15]: # Instead of dropping null values house_loan = house_loan.dropna() filling it up with mean
      # fill null values with mean
      # fill missing values in numeric columns with mean
      for col in house_loan.select_dtypes(include=['int64', 'float64']).columns:
          house_loan[col] = house_loan[col].fillna(house_loan[col].mean())

      # fill missing values in non-numeric columns with most frequent value
      for col in house_loan.select_dtypes(exclude=['int64', 'float64']).columns:
          house_loan[col] = house_loan[col].fillna(house_loan[col].mode()[0])

[16]: null_count = house_loan.isnull().sum()
      null_count

[16]: SK_ID_CURR          0
      TARGET            0
      NAME_CONTRACT_TYPE 0
      CODE_GENDER       0
      FLAG_OWH_CAR       0
      ...
      AHT_REQ_CREDIT_BUREAU_YEAR 41519
      Length: 122, dtype: int64
```

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

```
Print percentage of default to payer of the dataset for the TARGET column Assuming the defaulters as 1 and 0

[17]: defaulters=(house_loan.TARGET==1).sum()
      payers=(house_loan.TARGET==0).sum()
      print(defaulters, payers)
      print((defaulters/payers)*100)

      24825 282686
      8.781828608145662

[18]: #Percentage of defaulters
      print((defaulters*100/(defaulters+payers)))

      8.072881945686495

Basic cleaning of data. Removinf duplicate, filling up NaN or null and encoding for analysis

[19]: from sklearn.preprocessing import LabelEncoder

[20]: le = LabelEncoder()
      #select non-numeric columns
      non_numeric_columns = house_loan.select_dtypes(exclude=['int64', 'float64']).columns

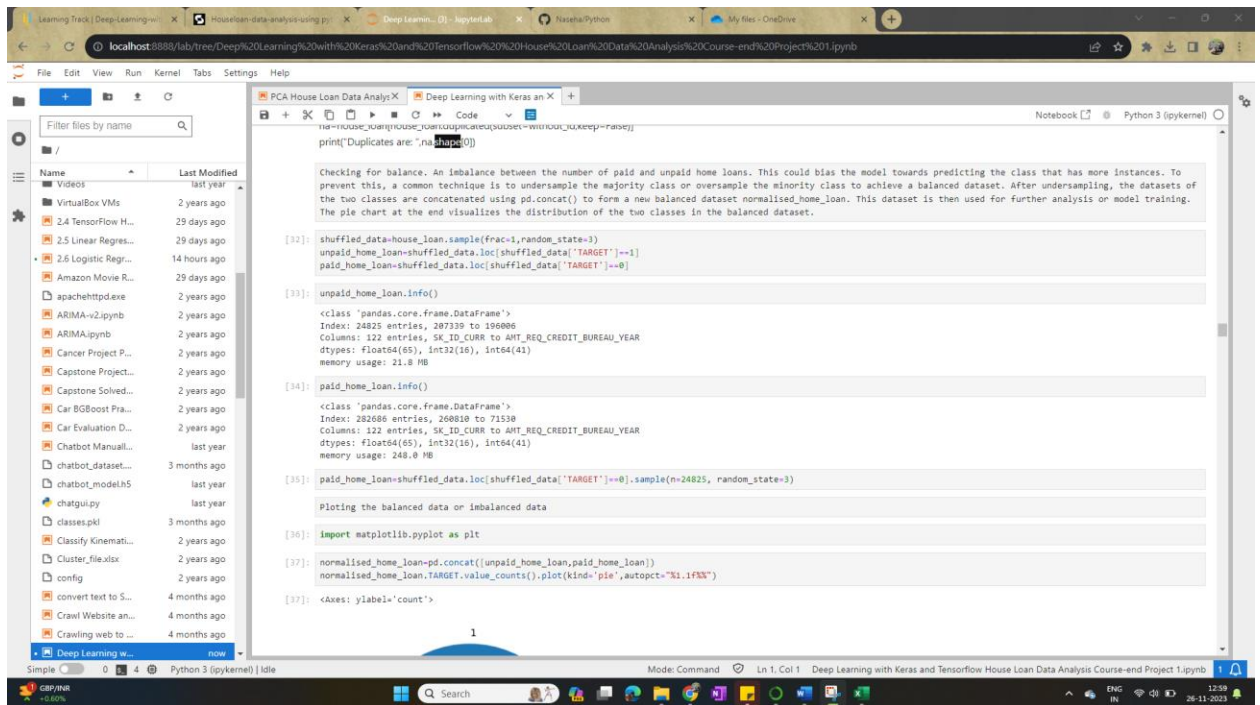
[21]: # create a dictionary to store the LabelEncoder objects for each column
      le_dict = {}

      for col in non_numeric_columns:
          le = LabelEncoder()
          house_loan[col] = le.fit_transform(house_loan[col])
          le_dict[col] = le

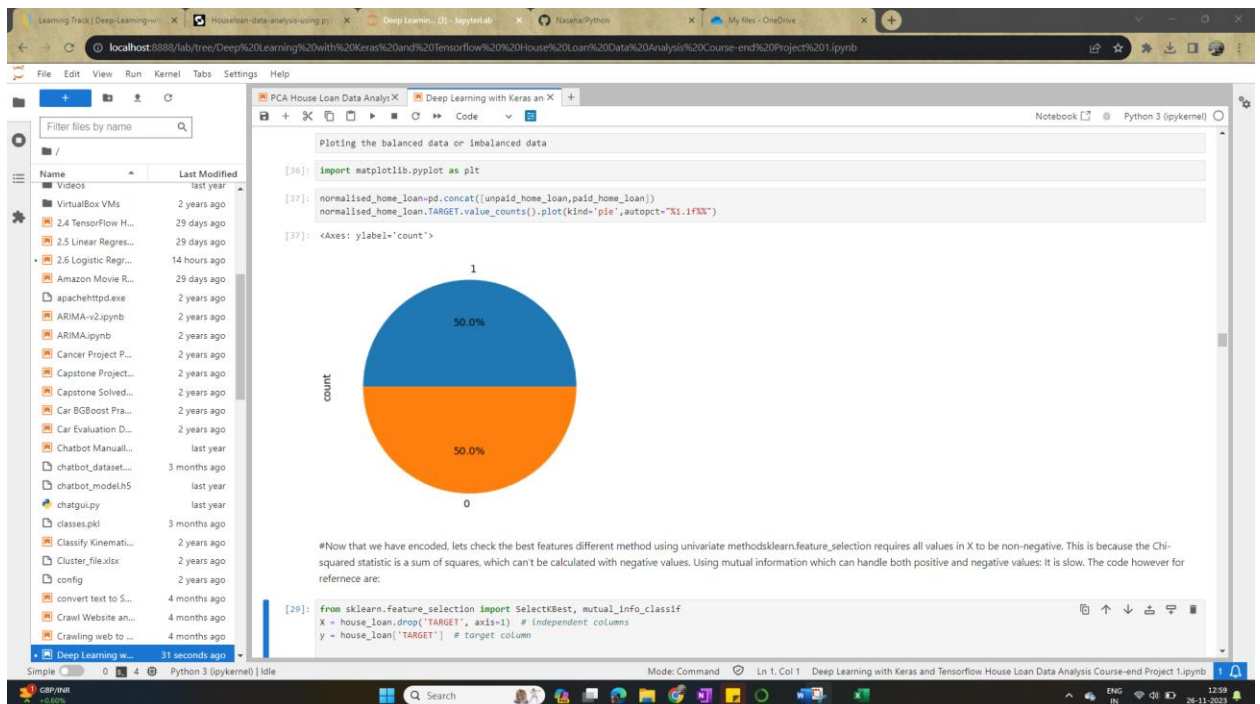
      # print the encoded DataFrame
      print(house_loan)

      # print the original values for each encoded number
      for col, le in le_dict.items():
          print(f'{col}:')
          for class_, label in enumerate(le.classes_):
              print(f'{class_}: {label}')
```

- Balance the dataset if the data is imbalanced



## 5. • Plot the balanced data or imbalanced data



## 6. • Encode the columns that is required for the model



Learning Track | Deep Learning with TensorFlow and Keras | PCA House Loan Data Analysis Course-end Project 1.ipynb

File Edit View Run Kernel Tabs Settings Help

Filter files by name

PCA House Loan Data Analysis Course-end Project 1.ipynb

Now that we have encoded, let's check the best features different method using univariate methods. `sklearn.feature_selection` requires all values in X to be non-negative. This is because the Chi-squared statistic is a sum of squares, which can't be calculated with negative values. Using mutual information which can handle both positive and negative values: it is slow. The code however for reference are:

```
[20]: from sklearn.feature_selection import SelectKBest, mutual_info_classif
X = house_loan.drop("TARGET", axis=1) # independent columns
y = house_loan["TARGET"] # target column

[30]: # apply SelectKBest class to extract top 10 best features
best_features = SelectKBest(score_func=mutual_info_classif, k=10)
fit = best_features.fit(X, y)

[31]: df_scores = pd.DataFrame(fit.scores_)
df_columns = pd.DataFrame(X.columns)
# concatenate two dataframes for better visualization
features_scores = pd.concat([df_columns, df_scores], axis=1)
features_scores.columns = ['Specs', 'Score'] # naming the dataframe columns

print(features_scores.nlargest(10, 'Score')) # print 10 best features
```

	Specs	Score
21	FLAG_MOBIL	0.000002
24	FLAG_CONTRACT_TYPE	0.099999
96	FLAG_DOCUMENT_3	0.056605
22	FLAG_EMP_PHONE	0.056169
4	FLAG_OWN_REALTY	0.055875
85	FONDKAPREHONT_MODE	0.053577
14	NAME_HOUSING_TYPE	0.052065
88	WALLS_MATERIAL_MODE	0.042359
61	COMMONAREA_MODE	0.040748
76	COMMONAREA_MEDT	0.040420

```
[38]: import tensorflow as tf

[39]: normalised_house_loan.info()
<class 'pandas.core.frame.DataFrame'>
Index: 49558 entries, 207339 to 139806
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int32(16), int64(41)
```

Simple 0 4 Python 3 (ipykernel) Idle

Mode: Command Ln 1, Col 1 Deep Learning with Keras and Tensorflow House Loan Data Analysis Course-end Project 1.ipynb

Learning Track | Deep Learning with TensorFlow and Keras | PCA House Loan Data Analysis Course-end Project 1.ipynb

File Edit View Run Kernel Tabs Settings Help

Filter files by name

PCA House Loan Data Analysis Course-end Project 1.ipynb

```
[40]: normalised_house_loan.head
```

```
[40]: <bound method NDFrame.head of ...>
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	\
207339	340318	1	0	0	0	
8756	110186	1	0	1	1	
230444	360811	1	0	0	0	
178329	306645	1	0	1	1	
55586	164487	1	0	1	0	
...	...	...	...	...	...	...
303856	452050	0	0	0	0	
140173	262532	0	0	0	0	
44575	151640	0	1	0	0	
106175	223189	0	0	0	0	
139806	262117	0	0	0	0	
...	...	...	...	...	...	...
207339	0	0	112500.0	405000.0		
8756	0	0	135000.0	544491.0		
230444	1	0	112500.0	221000.0		
178329	1	0	157500.0	595273.5		
55586	0	0	157500.0	521451.0		
...	...	...	...	...	...	...
303856	1	0	180000.0	314100.0		
140173	0	0	202500.0	490495.5		
44575	0	0	180000.0	315000.0		
106175	1	0	180000.0	207396.0		
139806	1	0	243000.0	67500.0		
...	...	...	...	...	...	...
207339	...	...	FLAG_DOCUMENT_18	FLAG_DOCUMENT_19	\	
8756	...	...	0	0		
230444	...	...	17563.5	0		
178329	...	...	17905.5	0		
55586	...	...	29083.5	0		
...	...	...	35406.0	0		
303856	...	...	...	...		
140173	...	...	17167.5	0		
44575	...	...	46701.0	0		
106175	...	...	15750.0	0		
139806	...	...	13183.0	0		
...	...	...	7267.5	0		
207339	...	...	FLAG_DOCUMENT_20	FLAG_DOCUMENT_21	AMT_REQ_CREDIT_BUREAU_HOUR	\
207339	0	0	0	0	0.000000	

Simple 0 4 Python 3 (ipykernel) Idle

Mode: Command Ln 1, Col 1 Deep Learning with Keras and Tensorflow House Loan Data Analysis Course-end Project 1.ipynb

Learning Track | Deep Learning with Keras | PCA House Loan Data Analysis | Deep Learning with Keras and TensorFlow House Loan Data Analysis Course-end Project 1.ipynb

File Edit View Run Kernel Tabs Settings Help

Filter files by name

Name Last Modified

- Videos
- VirtualBox VMs
- 2.4 TensorFlow H...
- 2.5 Linear Regres...
- 2.6 Logistic Regr...
- Amazon Movie R...
- apachettpd.exe
- ARIMA-v2.ipynb
- ARIMA.ipynb
- Cancer Project P...
- Capstone Project...
- Capstone Solved...
- Car BGBBoost Pra...
- Car Evaluation D...
- Chatbot Manual...
- chatbot\_dataset...
- chatbot\_model.h5
- chatgui.py
- classes.pkl
- Classify Kinemat...
- Cluster\_file.xlsx
- config
- convert text to S...
- Crawl Website an...
- Crawling web to ...
- Deep Learning w...

207339 0 0 0.000000  
8756 0 0 0.000000  
230344 0 0 0.000402  
178329 0 0 0.000402  
55586 0 0 0.000000  
...  
303856 0.000000 0.000000  
148173 0 0 0.000000  
44575 0 0 0.000000  
106175 0 0 0.000000  
139806 0 0 0.000000  
...  
AMT\_REQ\_CREDIT\_BUREAU\_DAY AMT\_REQ\_CREDIT\_BUREAU\_WEEK  
207339 0.000 0.000000  
8756 0.000 0.000000  
230344 0.007 0.034362  
178329 0.007 0.034362  
55586 0.000 0.000000  
...  
303856 0.000 0.000000  
148173 0.000 0.000000  
44575 0.000 0.000000  
106175 0.000 0.000000  
139806 0.000 0.000000  
...  
AMT\_REQ\_CREDIT\_BUREAU\_MON AMT\_REQ\_CREDIT\_BUREAU\_QRT  
207339 0.000000 0.000000  
8756 0.000000 0.000000  
230344 0.267395 0.265474  
178329 0.267395 0.265474  
55586 0.000000 0.000000  
...  
303856 0.000000 0.000000  
148173 0.000000 0.000000  
44575 0.000000 0.000000  
106175 0.000000 0.000000  
139806 0.000000 0.000000  
...  
AMT\_REQ\_CREDIT\_BUREAU\_YEAR  
207339 0.000000  
8756 0.000000  
230344 1.899974  
178329 1.899974

Simple 0 4 Python 3 (ipykernel) Idle

Mode: Command Ln 1, Col 1 Deep Learning with Keras and TensorFlow House Loan Data Analysis Course-end Project 1.ipynb

Learning Track | Deep Learning with Keras | PCA House Loan Data Analysis | Deep Learning with Keras and TensorFlow House Loan Data Analysis Course-end Project 1.ipynb

File Edit View Run Kernel Tabs Settings Help

Filter files by name

Name Last Modified

- Videos
- VirtualBox VMs
- 2.4 TensorFlow H...
- 2.5 Linear Regres...
- 2.6 Logistic Regr...
- Amazon Movie R...
- apachettpd.exe
- ARIMA-v2.ipynb
- ARIMA.ipynb
- Cancer Project P...
- Capstone Project...
- Capstone Solved...
- Car BGBBoost Pra...
- Car Evaluation D...
- Chatbot Manual...
- chatbot\_dataset...
- chatbot\_model.h5
- chatgui.py
- classes.pkl
- Classify Kinemat...
- Cluster\_file.xlsx
- config
- convert text to S...
- Crawl Website an...
- Crawling web to ...
- Deep Learning w...

1.999974 0.000000 0.000000  
...  
303856 6.000000  
148173 2.000000  
44575 1.000000  
106175 2.000000  
139806 3.000000  
...  
[49650 rows x 122 columns]  
[41]: normalised\_home\_loan.dropna(axis=0)  
normalised\_home\_loan.info()  
<class 'pandas.core.frame.DataFrame'  
Index: 49650 entries, 207339 to 139806  
Columns: 122 entries, SK\_ID\_CURR to AMT\_REQ\_CREDIT\_BUREAU\_YEAR  
dtypes: float64(65), int32(16), int64(41)  
memory usage: 43.6 MB  
[42]: normalised\_home\_loan.isnull().sum()  
[42]: SK\_ID\_CURR 0  
TARGET 0  
NAME\_CONTRACT\_TYPE 0  
CODE\_GENDER 0  
FLAG\_Own\_CAR 0  
...  
AMT\_REQ\_CREDIT\_BUREAU\_DAY 0  
AMT\_REQ\_CREDIT\_BUREAU\_WEEK 0  
AMT\_REQ\_CREDIT\_BUREAU\_MON 0  
AMT\_REQ\_CREDIT\_BUREAU\_QRT 0  
AMT\_REQ\_CREDIT\_BUREAU\_YEAR 0  
Length: 122, dtype: int64  
[43]: print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_DAY))  
print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_WEEK))  
print(pd.unique(normalised\_home\_loan.AMT\_REQ\_CREDIT\_BUREAU\_MON))

Simple 0 4 Python 3 (ipykernel) Idle

Mode: Command Ln 1, Col 1 Deep Learning with Keras and TensorFlow House Loan Data Analysis Course-end Project 1.ipynb

The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a notebook cell on the right. The notebook cell contains the following content:

```
[44]: normalised_house_loan.dropna(axis=0)
```

	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
207339	340318	1	0	0	0	0	0	112500.0	405000.0	21969.0	...	...
8756	110186	1	0	1	1	0	0	135000.0	544491.0	17563.5	...	...
230344	366811	1	0	0	0	1	0	112500.0	225000.0	17905.5	...	...
178329	306645	1	0	1	1	1	0	157500.0	595273.5	29083.5	...	...
55586	164407	1	0	1	0	0	0	157500.0	521451.0	35406.0	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...
303856	452050	0	0	0	0	1	0	180000.0	314100.0	17167.5	...	...
140173	262532	0	0	0	0	0	0	202500.0	490495.5	46701.0	...	...
44575	151640	0	1	0	0	0	0	180000.0	315000.0	15750.0	...	...
106175	223189	0	0	0	0	1	0	180000.0	207396.0	13383.0	...	...
139806	262117	0	0	0	0	1	0	243000.0	67500.0	7267.5	...	...

49650 rows x 122 columns

Column - 'SK\_ID\_CURR' - to remove. 'TARGET' - Tells if person has defaulted. We have balanced it. Adding the parameters we think is important. 'NAME\_CONTRACT\_TYPE' - It is Type of loan  
 'CODE\_GENDER' - Gender 'FLAG\_OWN\_CAR' - Needed 'CNT\_CHILDREN' - Count of children 'AMT\_INCOME\_TOTAL' - Better to group it

Now adding the important ones that we got earlier 21 FLAG\_MOBIL 0.079671 24 FLAG\_CONT\_MOBILE 0.058517 4 FLAG\_OWN\_REALTY 0.056045 22 FLAG\_EMP\_PHONE 0.055531 96  
 FLAG\_DOCUMENT\_3 0.055391 85 FONDAPKPREMOT\_MODE 0.054326 14 NAME\_HOUSING\_TYPE 0.051551 88 WALLSMATERIAL\_MODE 0.044663 61 COMMONAREA\_MODE 0.040486 47  
 COMMONAREA\_AVG 0.040430

Visual representation of the individual component

```
[45]: normalised_house_loan.NAME_CONTRACT_TYPE.value_counts().plot(kind='pie', autopct='%1.1f%%')
```

## 7. • Calculate Sensitivity as a metric

The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a notebook cell on the right. The notebook cell contains the following code:

```
# Instantiate the model
model = LogisticRegression()

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

# Train the model
model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY))

# Evaluate the model
loss, accuracy = model.evaluate(testX, testY)
print(f"loss: {loss}, Accuracy: {accuracy}")
```

```
Epoch 1/10
1040/1040 [=====] - 8s 5ms/step - loss: 0.7824 - accuracy: 0.5809 - val_loss: 0.3779 - val_accuracy: 0.8236
Epoch 2/10
1040/1040 [=====] - 4s 4ms/step - loss: 0.2714 - accuracy: 0.8956 - val_loss: 0.1916 - val_accuracy: 0.9407
Epoch 3/10
1040/1040 [=====] - 4s 4ms/step - loss: 0.1478 - accuracy: 0.9698 - val_loss: 0.1110 - val_accuracy: 0.9857
Epoch 4/10
1040/1040 [=====] - 5s 4ms/step - loss: 0.0892 - accuracy: 0.9911 - val_loss: 0.0695 - val_accuracy: 0.9957
Epoch 5/10
1040/1040 [=====] - 4s 4ms/step - loss: 0.0577 - accuracy: 0.9970 - val_loss: 0.0462 - val_accuracy: 0.9982
Epoch 6/10
1040/1040 [=====] - 3s 3ms/step - loss: 0.0392 - accuracy: 0.9985 - val_loss: 0.0328 - val_accuracy: 0.9991
Epoch 7/10
1040/1040 [=====] - 3s 3ms/step - loss: 0.0277 - accuracy: 0.9990 - val_loss: 0.0229 - val_accuracy: 0.9996
Epoch 8/10
1040/1040 [=====] - 4s 4ms/step - loss: 0.0202 - accuracy: 0.9992 - val_loss: 0.0169 - val_accuracy: 0.9996
Epoch 9/10
1040/1040 [=====] - 4s 4ms/step - loss: 0.0151 - accuracy: 0.9993 - val_loss: 0.0127 - val_accuracy: 0.9996
Epoch 10/10
1040/1040 [=====] - 3s 3ms/step - loss: 0.0115 - accuracy: 0.9995 - val_loss: 0.0095 - val_accuracy: 0.9997
513/513 [=====] - 1s 2ms/step - loss: 0.0098 - accuracy: 0.9997
Loss: 0.009800774045288563, Accuracy: 0.99969482421875
```

```
[170]: # Predict the values
predicted_values = model.predict(testX)

# Convert the predicted values to a suitable format
predicted_values = tf.squeeze(predicted_values)
```

```
File Edit View Run Kernel Tabs Settings Help
PCA House Loan Data AnalysisX Deep Learning with Keras an...
Notebook Python 3 (ipykernel)

# Convert the predicted values to a suitable format
predicted_values = tf.squeeze(predicted_values)

# Calculate the difference
difference = tf.abs(predicted_values - testy)

# Create a DataFrame
df = pd.DataFrame({
    'Predicted Values': predicted_values.numpy(),
    'Actual Values': testy.numpy(),
    'Difference': difference.numpy()
})

# Print the DataFrame
print(df)

513/513 [.....] - 2s 3ms/step
Predicted Values Actual Values Difference
0 0.003062 0.0 0.003062
1 0.006138 1.0 0.003062
2 0.994333 1.0 0.005667
3 0.021449 0.0 0.021449
4 0.000256 0.0 0.000256
...
16380 0.999531 1.0 0.000469
16381 0.999976 1.0 0.000024
16382 0.015913 0.0 0.015913
16383 0.999802 1.0 0.000198
16384 0.000675 0.0 0.000675
[16385 rows x 3 columns]

[171]: import numpy as np
from sklearn.metrics import confusion_matrix, recall_score
# Convert the predicted values to binary
predicted_values_binary = np.round(predicted_values)
# Calculate the confusion matrix
cm = confusion_matrix(testy, predicted_values_binary)
print('Confusion Matrix: \n', cm)
# Calculate sensitivity
sensitivity = recall_score(testy, predicted_values_binary)
print('Sensitivity: \n', sensitivity)
Confusion Matrix:
[[8198  5]
 [ 0 8182]]
Sensitivity:
1.0

To Calculate Sensitivity, using confusion matrix. Formula of sensitivity
Sensitivity = True Positives / (True Positives + False Negatives)

[171]: # Calculate sensitivity
```

```
File Edit View Run Kernel Tabs Settings Help
PCA House Loan Data AnalysisX Deep Learning with Keras an...
Notebook Python 3 (ipykernel)

# Convert the predicted values to a suitable format
predicted_values = tf.squeeze(predicted_values)

# Calculate the difference
difference = tf.abs(predicted_values - testy)

# Create a DataFrame
df = pd.DataFrame({
    'Predicted Values': predicted_values.numpy(),
    'Actual Values': testy.numpy(),
    'Difference': difference.numpy()
})

# Print the DataFrame
print(df)

513/513 [.....] - 2s 3ms/step
Predicted Values Actual Values Difference
0 0.003062 0.0 0.003062
1 0.006138 1.0 0.003062
2 0.994333 1.0 0.005667
3 0.021449 0.0 0.021449
4 0.000256 0.0 0.000256
...
16380 0.999531 1.0 0.000469
16381 0.999976 1.0 0.000024
16382 0.015913 0.0 0.015913
16383 0.999802 1.0 0.000198
16384 0.000675 0.0 0.000675
[16385 rows x 3 columns]

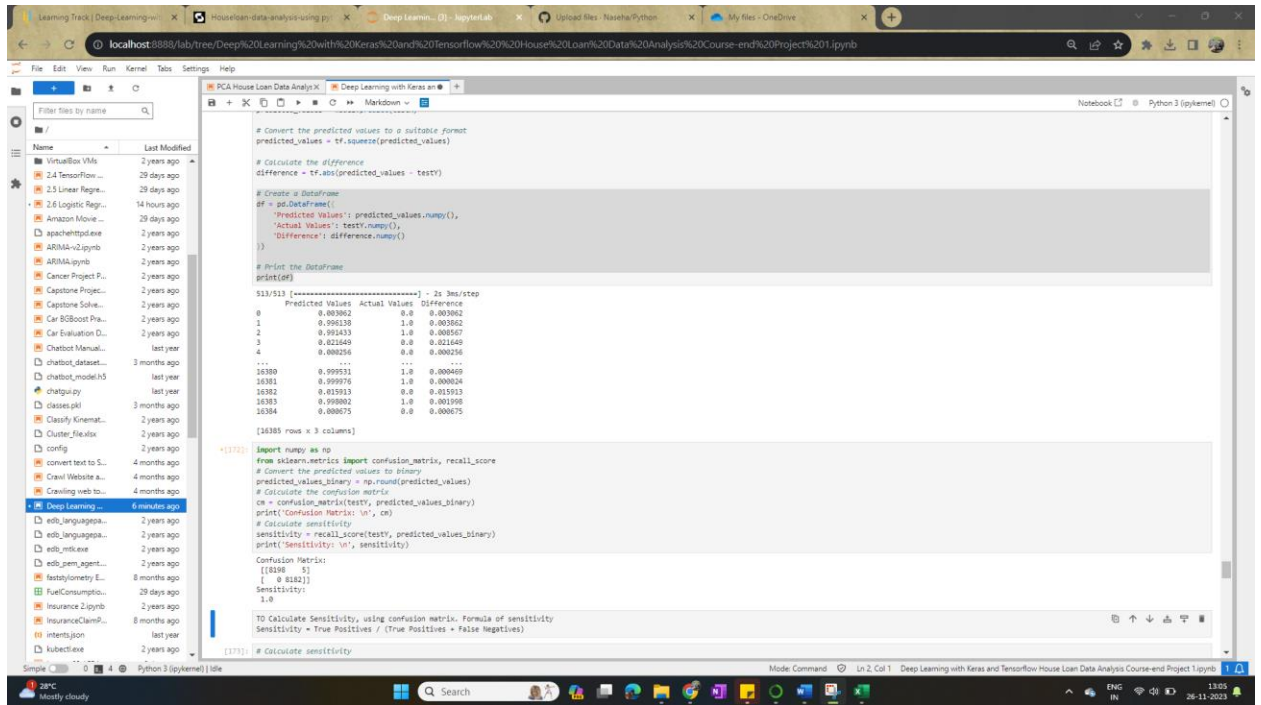
[172]: import numpy as np
from sklearn.metrics import confusion_matrix, recall_score
# Convert the predicted values to binary
predicted_values_binary = np.round(predicted_values)
# Calculate the confusion matrix
cm = confusion_matrix(testy, predicted_values_binary)
print('Confusion Matrix: \n', cm)
# Calculate sensitivity
sensitivity = recall_score(testy, predicted_values_binary)
print('Sensitivity: \n', sensitivity)
Confusion Matrix:
[[8198  5]
 [ 0 8182]]
Sensitivity:
1.0

To Calculate Sensitivity, using confusion matrix. Formula of sensitivity
Sensitivity = True Positives / (True Positives + False Negatives)

[171]: # Calculate sensitivity
```

8. • Calculate area under receiver operating characteristics curve





```
# Convert the predicted values to a suitable format
predicted_values = tf.squeeze(predicted_values)

# Calculate the difference
difference = tf.abs(predicted_values - testY)

# Create a DataFrame
df = pd.DataFrame({
    'Predicted Values': predicted_values.numpy(),
    'Actual Values': testY.numpy(),
    'Difference': difference.numpy()
})

# Print the DataFrame
print(df)

513/513 [=====] - 2s 36s/step
Predicted Values Actual Values Difference
0 0.003062 0.0 0.003062
1 0.006130 1.0 0.003062
2 0.991433 1.0 0.008567
3 0.021640 0.0 0.021640
4 0.000256 0.0 0.000256
...
16380 0.999551 1.0 0.000449
16381 0.999976 1.0 0.000024
16382 0.015913 0.0 0.015913
16383 0.999802 1.0 0.001198
16384 0.000075 0.0 0.000075
[16383 rows x 3 columns]

16383/16383 [=====] - 2s 36s/step
Import numpy as np
from sklearn.metrics import confusion_matrix, recall_score
predicted_values_binary = np.round(predicted_values)
# Calculate the confusion matrix
cm = confusion_matrix(testY, predicted_values_binary)
print('Confusion Matrix: \n', cm)
# Calculate sensitivity
sensitivity = recall_score(testY, predicted_values_binary)
print('Sensitivity: \n', sensitivity)

Confusion Matrix:
[[8148 5]
 [ 0 8132]]
Sensitivity:
1.0

To Calculate Sensitivity, using confusion matrix. Formula of sensitivity
Sensitivity = True Positives / (True Positives + False Negatives)

(177) # Calculate sensitivity
```

Embedded File:

Python File:



Deep Learning with  
Keras and Tensorflo

Code hosted on GitHub :

<https://github.com/Naseha/Python>

<https://github.com/Naseha/Python/blob/main/Deep%20Learning%20with%20Keras%20and%20Tensorflow%20House%20Loan%20Data%20Analysis%20Course-end%20Project%201.ipynb>

Downloaded pdf Attached:



Deep Learning with  
Keras and Tensorflo

<https://github.com/Naseha/Python/blob/main/Deep%20Learning%20with%20Keras%20and%20Tensorflow%20House%20Loan%20Data%20Analysis%20Course-end%20Project%201.pdf>

Codes:

Deep Learning with Keras and Tensorflow

House Loan Data Analysis Course-end Project 1

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.

In [ ]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

In [ ]:

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

In [ ]:

```
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import os
import warnings
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.datasets import make_blobs
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
```

In console type

Adding Path setx PATH "%PATH%;C:\path\to\Anaconda3\" conda update conda pip install plotly pip install -upgrade pip on terminal

conda install -c conda-forge cufflinks-py pip install lightgbm

In []:

```
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
import plotly.offline as py
import plotly.graph_objs as go
from plotly.offline import init_notebook_mode, iplot
from sklearn.model_selection import train_test_split
init_notebook_mode(connected=True)
```

In []:

```
import cufflinks as cf
cf.go_offline()
import pickle
import gc
```

In []:

```
import lightgbm as lgb
warnings.filterwarnings('ignore')
Load the dataset that is given to you Loading the dataset
```

In []:

```
import csv
# csv file name
house_loan = pd.read_csv(r'D:\OneDrive\Knowledge Center\AI - ML\Masters in Artificial Engineer\Deep
Learning with Keras and Tensorflow\loan_data.csv')
```

In []:

```
house_loan.describe()
```

In []:

```
house_loan.columns
```

In []:

```
print(house_loan.head())
```

In []:

```
house_loan.info()
Check for null values in the dataset
```

In []:

```
null_values = house_loan.isnull()
null_count = house_loan.isnull().sum()
```

In []:

```
print(null_values, null_count)
```

In []:

```
null_count
```

In []:

```
#Instead of dropping null values house_loan = house_loan.dropna() filling it up with mean
# fill null values with mean
# fill missing values in numeric columns with mean
for col in house_loan.select_dtypes(include=['int64', 'float64']).columns:
    house_loan[col] = house_loan[col].fillna(house_loan[col].mean())
```

```
# fill missing values in non-numeric columns with most frequent value
for col in house_loan.select_dtypes(exclude=['int64', 'float64']).columns:
    house_loan[col] = house_loan[col].fillna(house_loan[col].mode()[0])
```

In [ ]:

```
null_count = house_loan.isnull().sum()
null_count
```

Print percentage of default to payer of the dataset for the TARGET column Assuming the defaulters as 1 and 0

In [ ]:

```
defaulters=(house_loan.TARGET==1).sum()
payers=(house_loan.TARGET==0).sum()
print(defaulters, payers)
print(((defaulters/payers)*100))
```

In [ ]:

```
#Percentage of defaulters
print(((defaulters*100/(defaulters+payers))))
Basic cleaning of data. REMoving duplicate, filling up NaN or null and encoding for analysis
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
```

In [ ]:

```
le = LabelEncoder()
#select non-numeric columns
non_numeric_columns = house_loan.select_dtypes(exclude=['int64', 'float64']).columns
```

In [ ]:

```
# create a dictionary to store the LabelEncoder objects for each column
le_dict = {}
```

```
for col in non_numeric_columns:
    le = LabelEncoder()
    house_loan[col] = le.fit_transform(house_loan[col])
    le_dict[col] = le
```

```
# print the encoded DataFrame
print(house_loan)
```

```
# print the original values for each encoded number
```

```
for col, le in le_dict.items():
    print(f"{col}:")
    for class_, label in enumerate(le.classes_):
        print(f"{class_}: {label}")
```

```
Anotehr method # create a label (category) encoder object fit and transform the non-numeric columns in the
DataFrame house_loan[non_numeric_columns] = house_loan[non_numeric_columns].apply(le.fit_transform)
print(house_loan)
```

Balance the dataset if the data is imbalanced The autogenerated column is SK\_ID\_CURR Checking and removing duplicate entries

In [ ]:

```
house_loan.TARGET.value_counts().plot(kind='pie',autopct='%1.1f%%')
```



```

without_id=[column for column in house_loan.columns if column!='SK_ID_CURR']
na=house_loan[house_loan.duplicated(subset=without_id,keep=False)]
print("Duplicates are: ",na.shape[0])
house_loan.TARGET.value_counts().plot(kind='pie',autopct='%1.1f%%') without_id=[column for column in
house_loan.columns if column!='SK_ID_CURR']
na=house_loan[house_loan.duplicated(subset=without_id,keep=False)]
print("Duplicates are: ",na.shape[0])

```

Checking for balance. An imbalance between the number of paid and unpaid home loans. This could bias the model towards predicting the class that has more instances. To prevent this, a common technique is to undersample the majority class or oversample the minority class to achieve a balanced dataset. After undersampling, the datasets of the two classes are concatenated using `pd.concat()` to form a new balanced dataset `normalised_home_loan`. This dataset is then used for further analysis or model training. The pie chart at the end visualizes the distribution of the two classes in the balanced dataset.

In [ ]:

```

shuffled_data=house_loan.sample(frac=1,random_state=3)
unpaid_home_loan=shuffled_data.loc[shuffled_data["TARGET"]==1]
paid_home_loan=shuffled_data.loc[shuffled_data["TARGET"]==0]

```

In [ ]:

```
unpaid_home_loan.info()
```

In [ ]:

```
paid_home_loan.info()
```

In [ ]:

```

paid_home_loan=shuffled_data.loc[shuffled_data["TARGET"]==0].sample(n=24825, random_state=3)
Plotting the balanced data or imbalanced data

```

In [ ]:

```
import matplotlib.pyplot as plt
```

In [ ]:

```

normalised_home_loan=pd.concat([unpaid_home_loan,paid_home_loan])
normalised_home_loan.TARGET.value_counts().plot(kind='pie',autopct="%1.1f%%")

```

## Code

Now that we have encoded, lets check the best features different method

using univariate methods `sklearn.feature_selection` requires all values in X to be non-negative. This is because the Chi-squared statistic is a sum of squares, which can't be calculated with negative values. Using mutual information which can handle both positive and negative values: It is slow. The code however for reference are:

In [ ]:

```

from sklearn.feature_selection import SelectKBest, mutual_info_classif
X = house_loan.drop("TARGET", axis=1) # independent columns
y = house_loan["TARGET"] # target column

```

In [ ]:

```

# apply SelectKBest class to extract top 10 best features
bestfeatures = SelectKBest(score_func=mutual_info_classif, k=10)
fit = bestfeatures.fit(X, y)

```

In [ ]:

```
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
# concatenate two dataframes for better visualization
featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Specs', 'Score'] # naming the dataframe columns
```

```
print(featureScores.nlargest(10, 'Score')) # print 10 best features
```

In [ ]:

```
import tensorflow as tf
```

In [ ]:

```
normalised_home_loan.info()
```

In [ ]:

```
normalised_home_loan.head
```

In [ ]:

```
normalised_home_loan.dropna(axis=0)
normalised_home_loan.info()
```

In [ ]:

```
normalised_home_loan.isnull().sum()
```

In [ ]:

```
print(pd.unique(normalised_home_loan.AMT_REQ_CREDIT_BUREAU_DAY))
print(pd.unique(normalised_home_loan.AMT_REQ_CREDIT_BUREAU_WEEK))
print(pd.unique(normalised_home_loan.AMT_REQ_CREDIT_BUREAU_MON))
print(pd.unique(normalised_home_loan.AMT_REQ_CREDIT_BUREAU_QRT))
print(pd.unique(normalised_home_loan.AMT_REQ_CREDIT_BUREAU_YEAR))
```

In [ ]:

```
normalised_home_loan.dropna(axis=0)
Column - 'SK_ID_CURR' - to remove, 'TARGET' - Tells if person has defaulted. We have balanced it. Adding the
parameters we think is important. 'NAME_CONTRACT_TYPE' - It is Type of loan 'CODE_GENDER' - Gender
'FLAG_OWN_CAR' - Needed 'CNT_CHILDREN' - Count of children 'AMT_INCOME_TOTAL' - Better to group it
```

```
Now adding the important ones that we got earlier 21 FLAG_MOBIL 0.079671 24 FLAG_CONT_MOBILE
0.058517 4 FLAG_OWN_REALTY 0.056045 22 FLAG_EMP_PHONE 0.055531 96 FLAG_DOCUMENT_3
0.055391 85 FONDKAPREMONT_MODE 0.054326 14 NAME_HOUSING_TYPE 0.051551 88
WALLSMATERIAL_MODE 0.044663 61 COMMONAREA_MODE 0.040486 47 COMMONAREA_AVG 0.040430
```

Visual representation of the individual component

In [ ]:

```
normalised_home_loan.NAME_CONTRACT_TYPE.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.CODE_GENDER.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.FLAG_OWN_CAR.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.CNT_CHILDREN.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.AMT_INCOME_TOTAL.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.AMT_INCOME_TOTAL.value_counts().nlargest(5).plot(kind='pie',  
autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.FLAG_MOBIL.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.FLAG_CONT_MOBILE.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.FLAG_OWN_REALTY.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.NAME_HOUSING_TYPE.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.WALLSMATERIAL_MODE.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.COMMONAREA_MODE.value_counts().plot(kind='pie', autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.COMMONAREA_MODE.value_counts().nlargest(5).plot(kind='pie',  
autopct="%1.1f%%")
```

In [ ]:

```
normalised_home_loan.COMMONAREA_AVG.value_counts().nlargest(5).plot(kind='pie', autopct="%1.1f%%")
```

The columns we have selected 'NAME\_CONTRACT\_TYPE' - It is Type of loan 'CODE\_GENDER' - Gender  
'FLAG\_OWN\_CAR' - 'CNT\_CHILDREN' - Count of children 'AMT\_INCOME\_TOTAL' - top categories

Now adding the important ones that we got earlier FLAG\_CONT\_MOBILE If they have mobile  
FLAG\_OWN\_REALTY - If they have real estate NAME\_HOUSING\_TYPE - type of house  
WALLSMATERIAL\_MODE - walls type COMMONAREA\_MODE - common area type COMMONAREA\_AVG - avg  
common area

In [ ]:

```
normalised_home_loan_features=['SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'CNT_CHILDREN',  
'AMT_INCOME_TOTAL', 'FLAG_CONT_MOBILE', 'FLAG_OWN_REALTY', 'NAME_HOUSING_TYPE',  
'WALLSMATERIAL_MODE', 'COMMONAREA_MODE', 'COMMONAREA_AVG']
```

In [ ]:

```
normalised_home_loan_features
```

To start calculating sensitivity Sensitivity is a measure of the proportion of actual positive cases that got  
predicted as positive (or, how many of the true positives were recalled). So, high Sensitivity means that the  
model predicted the positive cases very well

In [ ]:

```
from sklearn.model_selection import train_test_split  
X=normalised_home_loan[normalised_home_loan_features]
```

In [ ]:

X

setting up parameters blobs\_random\_seed = 42: This sets the seed for the random number generator to 42. centers = [(0,0), (5,5)] - defines centers. In this case, there will be two clusters: one centered at (0,0) and the other at (5,5). cluster\_std = 1: This sets the standard deviation of the clusters. A higher value will make the clusters more spread out. frac\_test\_split = 0.33: This is likely the fraction of the data that will be used for the test set in a train/test split. num\_features\_for\_samples = 2: This is the number of features for each sample in the synthetic dataset. In this case, each sample will have two features.num\_samples\_total = 49650: This is the total number of samples in the synthetic dataset. See from above.

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(inputs, targets, test_size=0.33, random_state=42)
```

In [ ]:

```
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

In [ ]:

*#Since the target variable is of categorical data (either default or not), using logistic regression*

In [ ]:

```
trainX = tf.constant(X_train, dtype='float32')
trainY = tf.constant(y_train, dtype='float32')
testX = tf.constant(X_test, dtype='float32')
testY = tf.constant(y_test, dtype='float32')
```

In [ ]:

*# Assuming your data has 'n' features*

```
n = trainX.shape[1]
```

*# Create a variable for weights*

```
weights = tf.Variable(tf.random.normal(shape=(n, 1), dtype='float32'))
```

*# Create a variable for biases*

```
bias = tf.Variable(tf.zeros(shape=(1,), dtype='float32'))
```

In [ ]:

```
print(n, weights, bias)
```

In [ ]:

```
import tensorflow as tf
```

*# Define the logistic regression model*

```
class LogisticRegression(tf.keras.Model):
```

```
    def __init__(self):
```

```
        super(LogisticRegression, self).__init__()
```

```
        self.dense = tf.keras.layers.Dense(1, activation='sigmoid')
```

```
    def call(self, inputs, training=None, mask=None):
```

```
        output = self.dense(inputs)
```

```
        return output
```

*# Instantiate the model*

```
model = LogisticRegression()
```

*# Compile the model*

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

*# Train the model*



```
model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY))
```

```
# Evaluate the model
```

```
loss, accuracy = model.evaluate(testX, testY)
print(f'Loss: {loss}, Accuracy: {accuracy}')
```

In []:

```
# Predict the values
```

```
predicted_values = model.predict(testX)
```

```
# Convert the predicted values to a suitable format
```

```
predicted_values = tf.squeeze(predicted_values)
```

```
# Calculate the difference
```

```
difference = tf.abs(predicted_values - testY)
```

```
# Create a DataFrame
```

```
df = pd.DataFrame({
    'Predicted Values': predicted_values.numpy(),
    'Actual Values': testY.numpy(),
    'Difference': difference.numpy()
})
```

```
# Print the DataFrame
```

```
print(df)
```

In []:

```
import numpy as np
```

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

```
# Convert the predicted values to binary
```

```
predicted_values_binary = np.round(predicted_values)
```

```
# Calculate the confusion matrix
```

```
cm = confusion_matrix(testY, predicted_values_binary)
```

```
print('Confusion Matrix: \n', cm)
```

```
# Calculate sensitivity
```

```
sensitivity = recall_score(testY, predicted_values_binary)
```

```
print('Sensitivity: \n', sensitivity)
```

TO Calculate Sensitivity, using confusion matrix. Formula of sensitivity  $\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$

In []:

```
# Calculate sensitivity
```

```
sensitivity = TP / (TP + FN)
```

```
print('Sensitivity: ', sensitivity)
```

```
print(predicted_labels_np)
```

```
print(y_test_np)
```

In []:

```
from sklearn.metrics import roc_auc_score
```

```
# Calculate the AUC-ROC
```

```
auc_roc = roc_auc_score(testY, predicted_values)
```

```
# Print the AUC-ROC
```

```
print('AUC-ROC: ', auc_roc)
```

In []:

```

from sklearn.metrics import roc_curve, auc
# Calculate the ROC curve
fpr, tpr, thresholds = roc_curve(testY, predicted_values)
# Calculate the AUC
roc_auc = auc(fpr, tpr)
# Plot the ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

```

In [ ]:

```

# Create a new figure
plt.figure()
# Plot a histogram of the actual values
plt.hist(testY, bins=30, alpha=0.5, label='Actual Values')

# Plot a histogram of the predicted values
plt.hist(predicted_values, bins=30, alpha=0.5, label='Predicted Values')

# Add labels and title
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Histogram of Actual vs Predicted Values')
plt.legend()

```

```

# Show the plot
plt.show()

```

In [ ]:

```

from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

# Calculate precision and recall
precision, recall, _ = precision_recall_curve(testY, predicted_values)

# Plot the precision-recall curve
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.show()

```

In [ ]: