Machine Learning Project on Credit Card Applications

By

Yogini Sanmugarajah

WELCOME TO THE PRESENTATION ON CREDIT CARD APPROVAL PREDICTION

- **Dataset: Credit Card Approval Prediction**

- **Source:** https://www.kaggle.com/datasets/rikdifos/credit-card-approval-prediction?select=application_record.csv

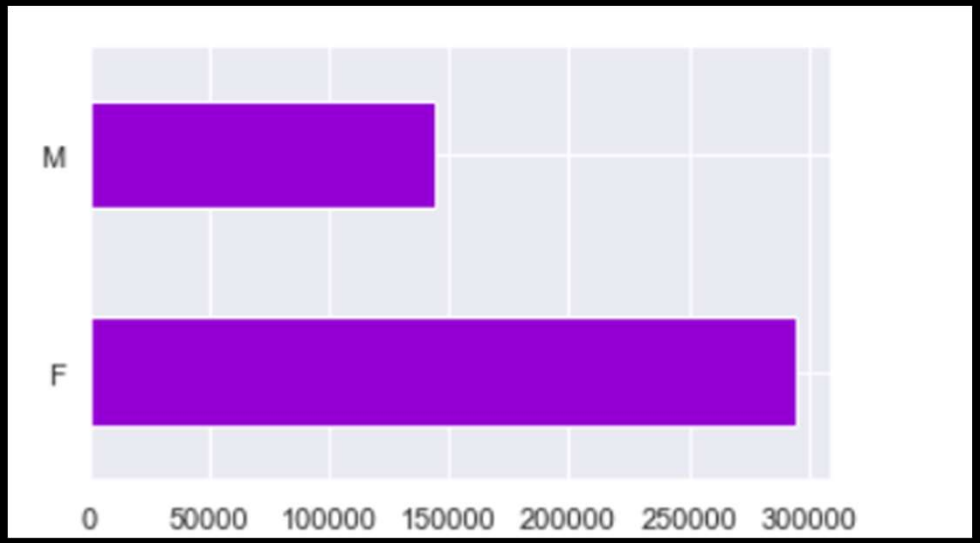
- There are two sets of data
  - Application record
  - Credit record

Problem Statement

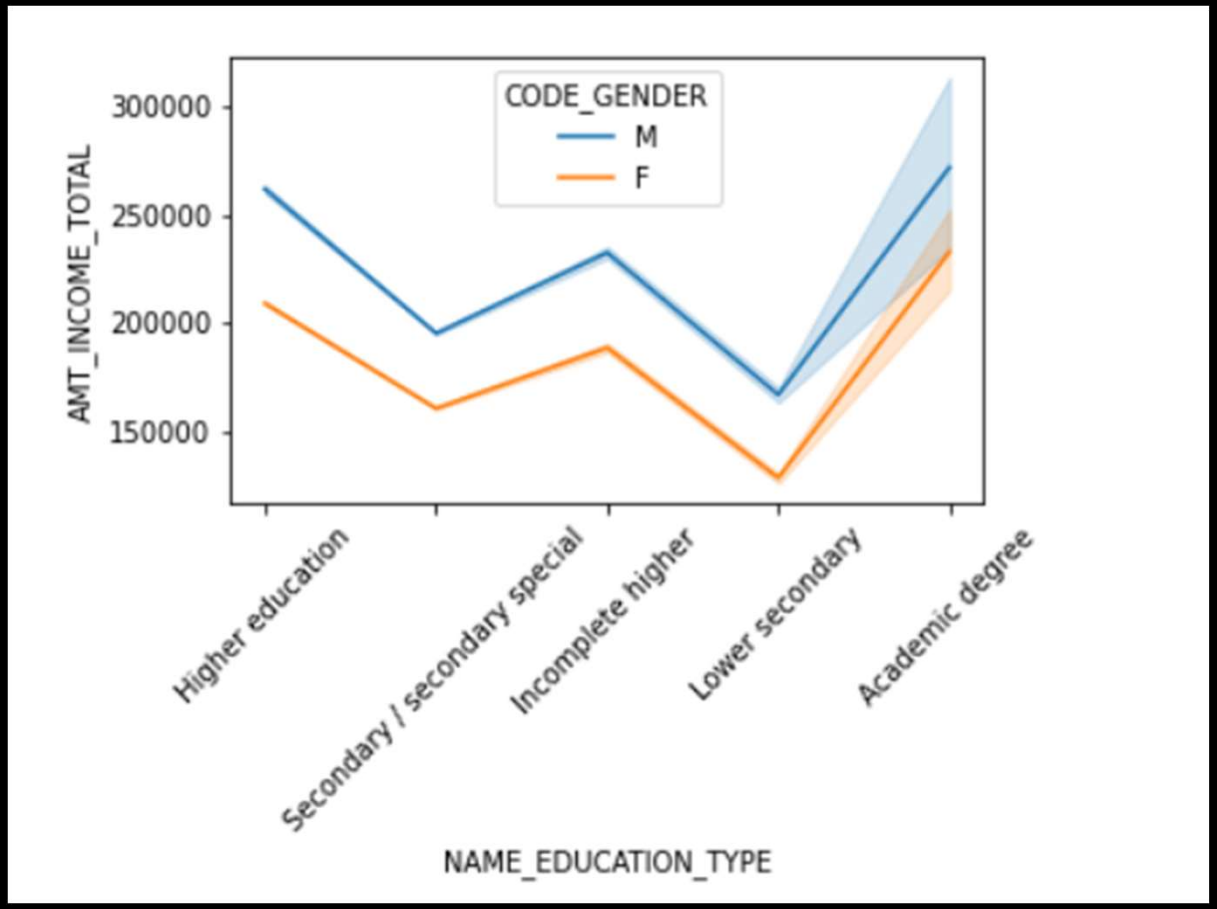
- How can we use the machine learning model to automate the credit card application rather than manual doing?

Overall Approach

- Different ways of data pre-processing, fitting different algorithms, and choosing the best four models to train the dataset.
- Forces on feature engineering, feature scaling, evaluation, and hyperparameter tuning



- Males are less than females in this dataset
- This is an imbalanced dataset

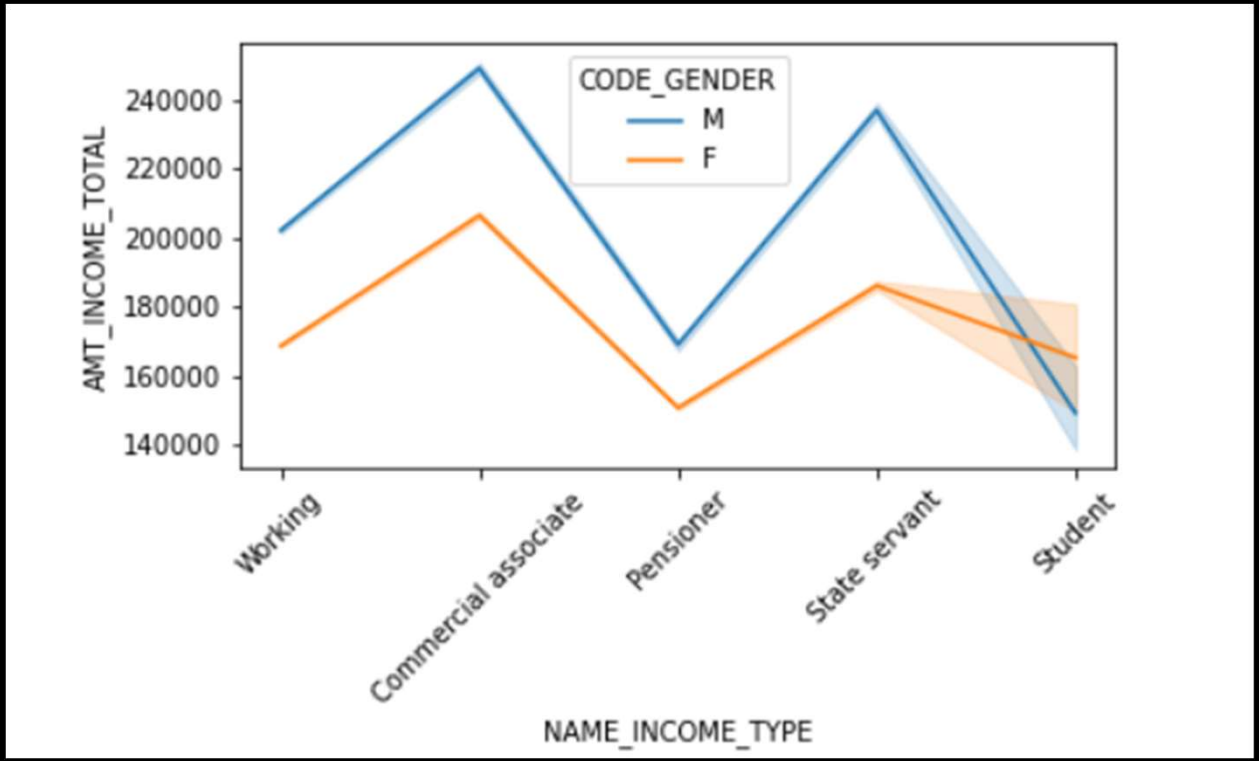


Education vs Income Observations

- Higher education and Academic degree holders earn more than the rest
- Males are earning more than females

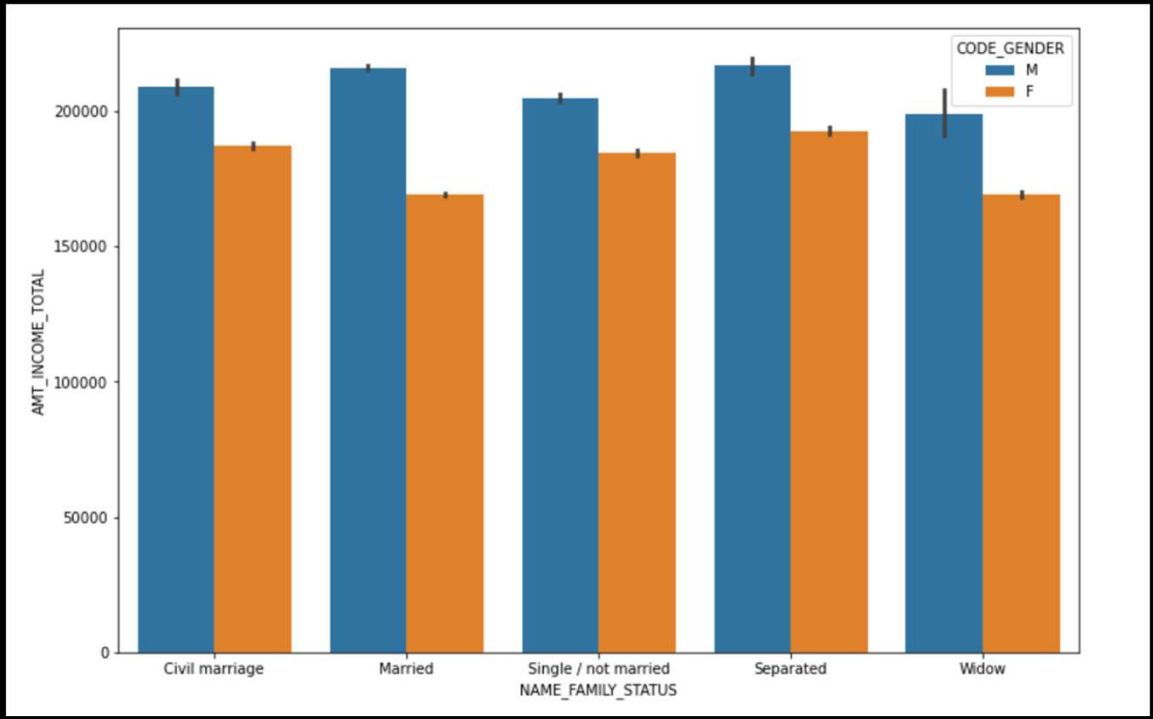
Income Type vs Income observations

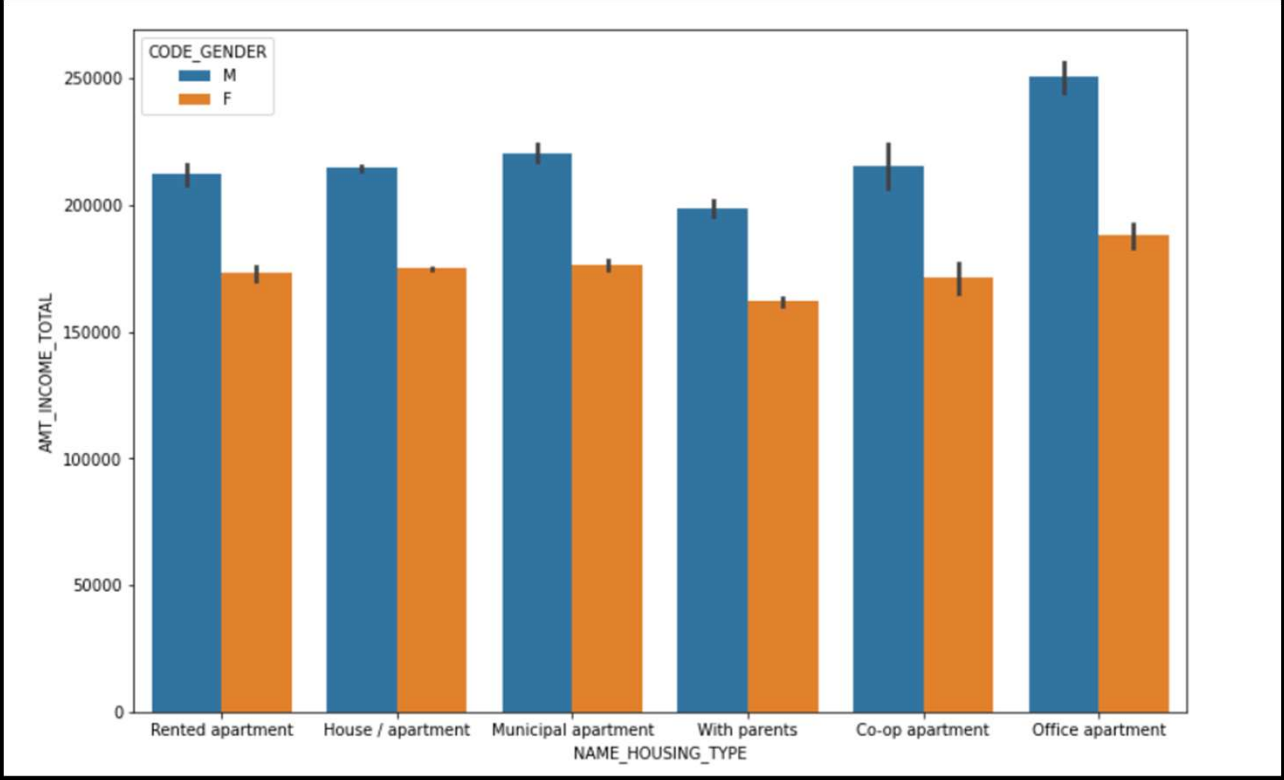
- Commercial associate income is higher than the rest
- Female income has a huge variance in State servant
- Female students are earning higher than male students. Is it due to not many male students working?



Marital Status vs Income

- Males' income did not fluctuate much, but married females' income is much less compared to other marital status





Housing Type vs Income

- Male staying in office apartment earn much higher than others

- Female and male with parent category income is lower compared to others

Data Preprocessing

Libraries used for project

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder,MinMaxScaler,StandardScaler
from scipy.stats.mstats import winsorize
from imblearn.over_sampling import RandomOverSampler,SMOTENC
from sklearn.model_selection import train_test_split,RandomizedSearchCV,GridSearchCV
from sklearn import metrics
from sklearn.ensemble import GradientBoostingClassifier,RandomForestClassifier,ExtraTreesClassifier,RandomForestRegressor
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest,chi2
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
```

Application dataset first 10 records

```
1 df_application.head(10)
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	N
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	
1	5008805	M	Y	Y	0	427500.0	Working	Higher education	
2	5008806	M	Y	Y	0	112500.0	Working	Secondary / secondary special	
3	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	
5	5008810	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	
6	5008811	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	
7	5008812	F	N	Y	0	283500.0	Pensioner	Higher education	
8	5008813	F	N	Y	0	283500.0	Pensioner	Higher education	
9	5008814	F	N	Y	0	283500.0	Pensioner	Higher education	

No of Rows & Columns

```
1 df_application.shape
```

```
(438557, 18)
```

```
1 df_application.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   ID                   438557 non-null  int64  
 1   CODE_GENDER          438557 non-null  object 
 2   FLAG_OWN_CAR         438557 non-null  object 
 3   FLAG_OWN_REALTY      438557 non-null  object 
 4   CNT_CHILDREN         438557 non-null  int64  
 5   AMT_INCOME_TOTAL     438557 non-null  float64
 6   NAME_INCOME_TYPE     438557 non-null  object 
 7   NAME_EDUCATION_TYPE  438557 non-null  object 
 8   NAME_FAMILY_STATUS   438557 non-null  object 
 9   NAME_HOUSING_TYPE    438557 non-null  object 
 10  DAYS_BIRTH           438557 non-null  int64  
 11  DAYS_EMPLOYED        438557 non-null  int64  
 12  FLAG_MOBIL           438557 non-null  int64  
 13  FLAG_WORK_PHONE      438557 non-null  int64  
 14  FLAG_PHONE           438557 non-null  int64  
 15  FLAG_EMAIL           438557 non-null  int64  
 16  OCCUPATION_TYPE      304354 non-null  object 
 17  CNT_FAM_MEMBERS      438557 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 60.2+ MB
```

- Column Information

```
1 #drop duplicates in dataset
2 df_application.drop_duplicates(subset = 'ID')
```

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYI
0	5008804	M	Y	Y	0	427500.0	Working	Higher educati
1	5008805	M	Y	Y	0	427500.0	Working	Higher educati
2	5008806	M	Y	Y	0	112500.0	Working	Secondary / seconda spec
3	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / seconda spec
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / seconda spec
...	...	...	...	...	...	...	...	
438552	6840104	M	N	Y	0	135000.0	Pensioner	Secondary / seconda spec
438553	6840222	F	N	N	0	103500.0	Working	Secondary / seconda spec
438554	6841878	F	N	N	0	54000.0	Commercial associate	Higher educati
438555	6842765	F	N	Y	0	72000.0	Pensioner	Secondary / seconda spec
438556	6842885	F	N	Y	0	121500.0	Working	Secondary / seconda spec

438510 rows × 18 columns

Handling Duplicates

```
1 df_application['ID'].duplicated().sum()
```

```
47
```

Duplicate rows dropped because the number of duplicate rows are very small compared to the total number of rows in the Application dataset

```
1  #checking for nul values
2  df_application.isna().sum()
```

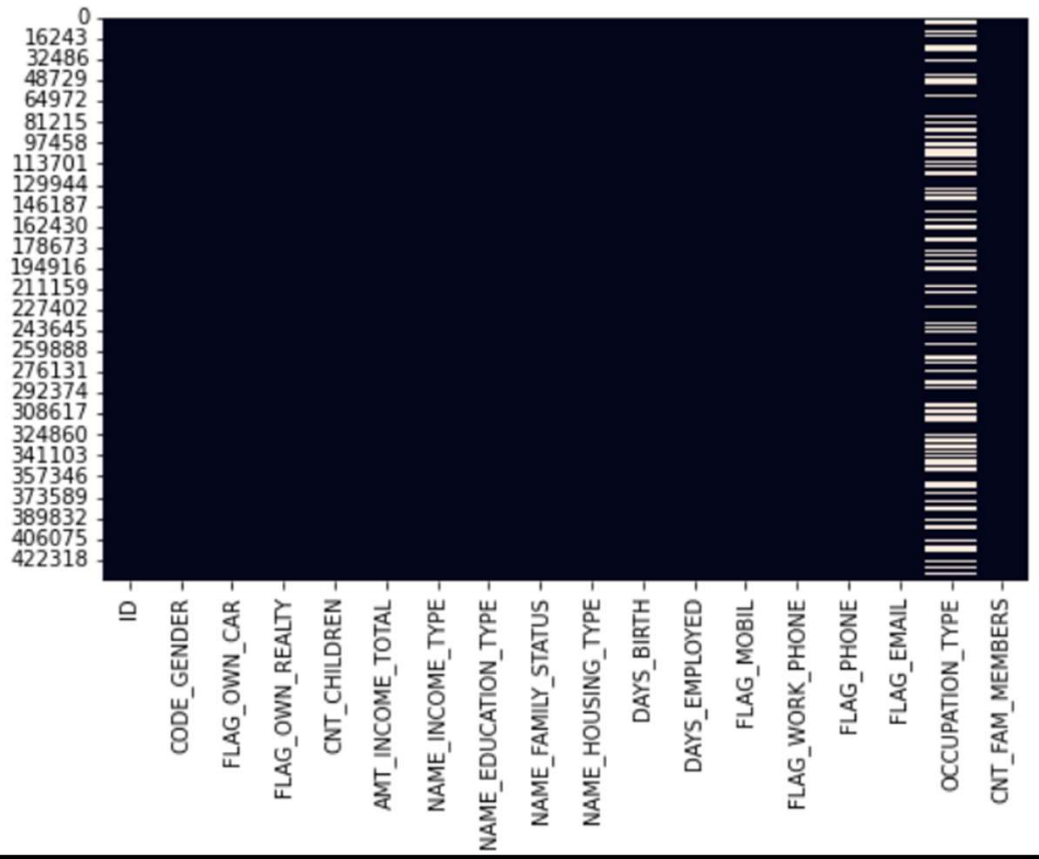
```
ID                          0
CODE_GENDER                 0
FLAG_OWN_CAR                0
FLAG_OWN_REALTY             0
CNT_CHILDREN                0
AMT_INCOME_TOTAL            0
NAME_INCOME_TYPE            0
NAME_EDUCATION_TYPE         0
NAME_FAMILY_STATUS          0
NAME_HOUSING_TYPE           0
DAYS_BIRTH                  0
DAYS_EMPLOYED               0
FLAG_MOBIL                  0
FLAG_WORK_PHONE             0
FLAG_PHONE                  0
FLAG_EMAIL                  0
OCCUPATION_TYPE        134203
CNT_FAM_MEMBERS             0
dtype: int64
```

- MISSING VALUES

- Missing Data Columns

```
1  #ploting the nul values
2  fig = plt.figure(figsize = (8,5))
3  sns.heatmap(df_application.isnull(),cbar=False)
```

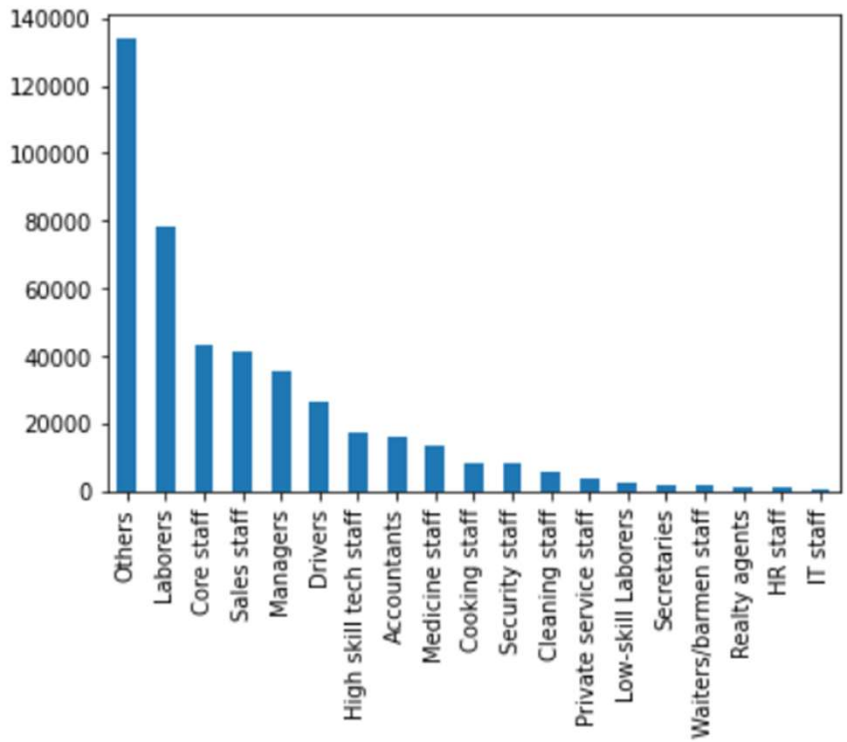
```
<AxesSubplot:>
```



```
#fill the missing value with value-"others"
df_application['OCCUPATION_TYPE'].fillna(value = 'Others', inplace = True)
```

```
#visualise column OCCUPATION_TYPE
df_application['OCCUPATION_TYPE'].value_counts().plot.bar(y='OCCUPATION_TYPE')
```

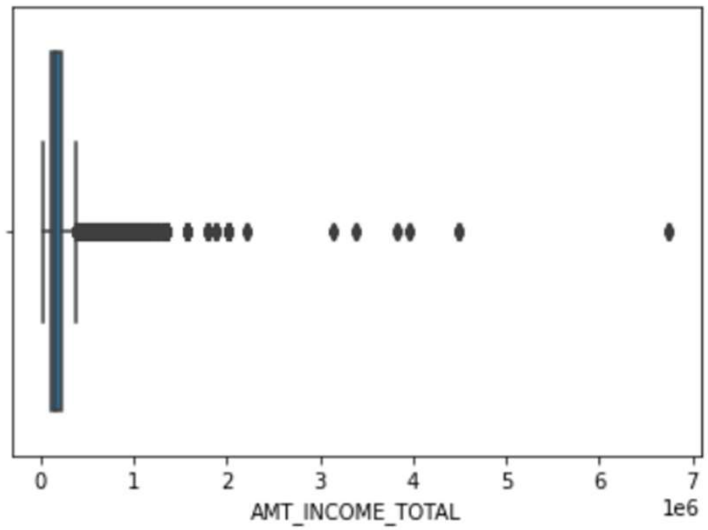
```
<AxesSubplot:>
```



```
1  #visualise column AMT_INCOME_TOTAL for outliers
2  sns.boxplot(df_application['AMT_INCOME_TOTAL'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\seabo
a keyword arg: x. From version 0.12, the only valid positio
explicit keyword will result in an error or misinterpretati
  warnings.warn(
```

```
<AxesSubplot:xlabel='AMT_INCOME_TOTAL'>
```



Handling Outliers

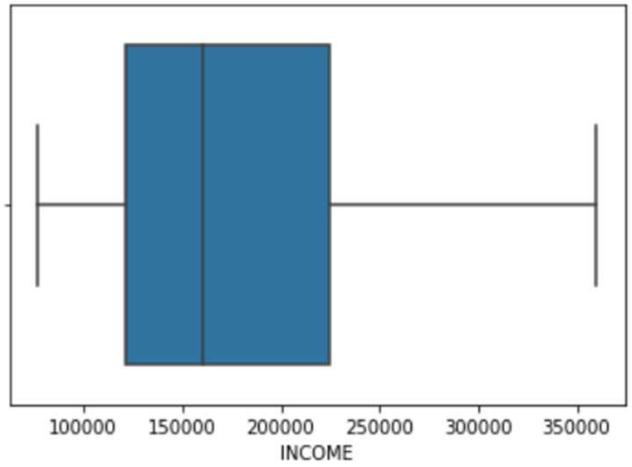
- Winsorization is used to eliminate the outliers.
- Winsorisation eliminates the outliers by bringing the value of the outlier to the mentioned quantile
- Outliers can mislead the training process and lead to poor result.

```
#use winsorize to eliminate the 5 percent of outliers & create new column INCOME
df_application['INCOME'] = winsorize((df_application['AMT_INCOME_TOTAL']),limits = [0.05,0.05],inplace = True)
```

```
#visualise the column INCOME after winsorization
sns.boxplot(df_application['INCOME'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

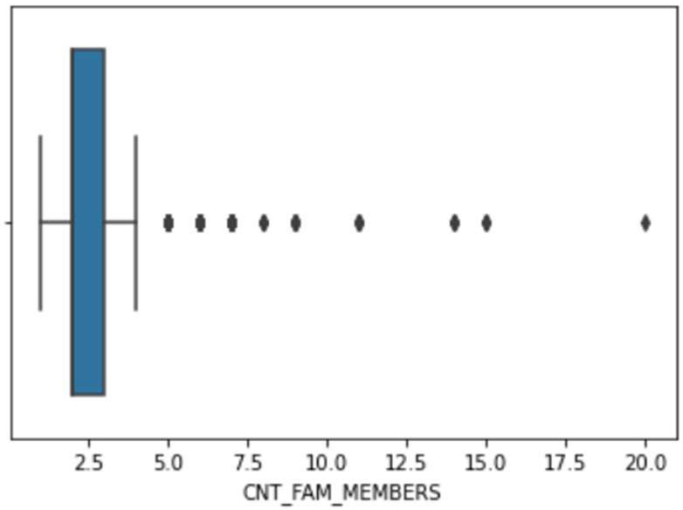
```
<AxesSubplot:xlabel='INCOME'>
```



```
1 #visualise column CNT_FAM_MEMBERS for outliers
2 sns.boxplot(df_application['CNT_FAM_MEMBERS'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\s
a keyword arg: x. From version 0.12, the only valid pos
explicit keyword will result in an error or misinterpre
  warnings.warn(
```

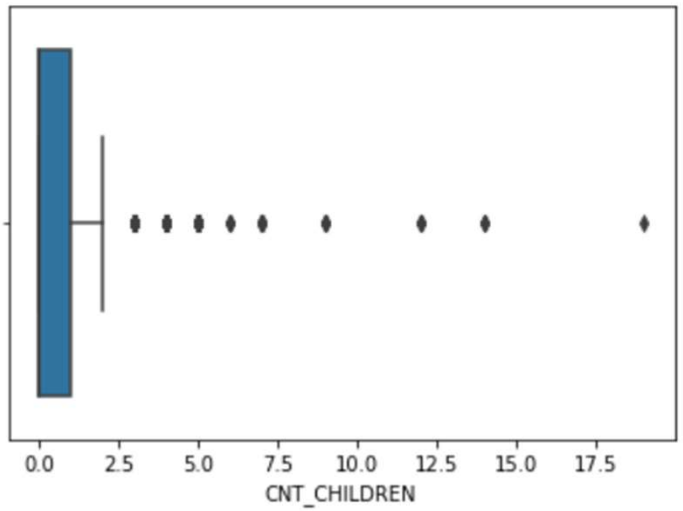
```
<AxesSubplot:xlabel='CNT_FAM_MEMBERS'>
```




```
1 #visualising CNT_CHILDREN for outliers
2 sns.boxplot(df_application['CNT_CHILDREN'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\s
a keyword arg: x. From version 0.12, the only valid pos
explicit keyword will result in an error or misinterpre
  warnings.warn(
```

```
<AxesSubplot:xlabel='CNT_CHILDREN'>
```

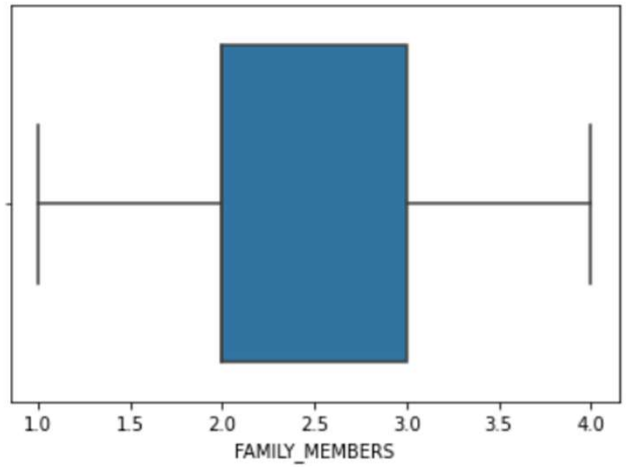


```
#use winsorize to eliminate the 5 percent of outliers & create new column FAMILY_MEMBERS
df_application['FAMILY_MEMBERS'] = winsorize((df_application['CNT_FAM_MEMBERS']),limits = [0.05,0.05],inplace = True)
```

```
#visualise the column FAMILY_MEMBERS after winsorization
sns.boxplot(df_application['FAMILY_MEMBERS'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='FAMILY_MEMBERS'>
```

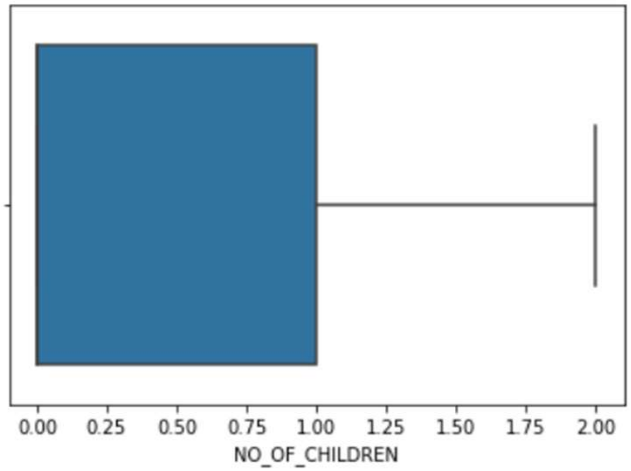


```
#use winsorize to eliminate the 5 percent of outliers & create new column NO_OF_CHILDREN
df_application['NO_OF_CHILDREN'] = winsorize((df_application['CNT_CHILDREN']),limits = [0.05,0.05],inplace = True)
```

```
sns.boxplot(df_application['NO_OF_CHILDREN'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='NO_OF_CHILDREN'>
```



Dropped columns - AMT_INCOME_TOTAL , CNT_FAM_MEMBERS , CNT_CHILDREN

Encoding Categorical Data Columns

```
df_application =pd.get_dummies(data = df_application, columns=['CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY','NAME_INCOME_T
df_application
```

NAME_EDUCATION_TYPE_Secondary / secondary special	NAME_FAMILY_STATUS_Married	NAME_FAMILY_STATUS_Separated	NAME_FAMILY_STATUS_Single / not married	NAME_FAMILY_STATUS_W
0	0	0	0	
0	0	0	0	
1	1	0	0	
1	0	0	1	
1	0	0	1	
...	...	...	...	
1	0	1	0	
1	0	0	1	
0	0	0	1	
1	1	0	0	
1	1	0	0	

- Encoding categorical columns with too many unique values can sparse the dataset. I have used dummy encoding and dropped the first column.

Feature Engineering

```
#create new column AGE columns from DAYS_BIRTH
df_application['AGE'] = -(df_application.DAYS_BIRTH/365).round(0)
df_application
```

ME_HOUSING_TYPE_House / apartment	NAME_HOUSING_TYPE_Municipal apartment	NAME_HOUSING_TYPE_Office apartment	NAME_HOUSING_TYPE_Rented apartment	NAME_HOUSING_TYPE_With parents	AGE
0	0	0	1	0	33.0
0	0	0	1	0	33.0
1	0	0	0	0	59.0
1	0	0	0	0	52.0
1	0	0	0	0	52.0
...	...	...	...	...	...
1	0	0	0	0	62.0
1	0	0	0	0	44.0
0	0	0	0	1	22.0
1	0	0	0	0	59.0

Driving new column AGE from DAYS_BIRTH column

```
#create new column EMPLOYMENT_YEARS columns from DAYS_EMPLOYED
df_application['EMPLOYMENT_YEARS'] = -(df_application.DAYS_EMPLOYED/365).round(0)
df_application
```

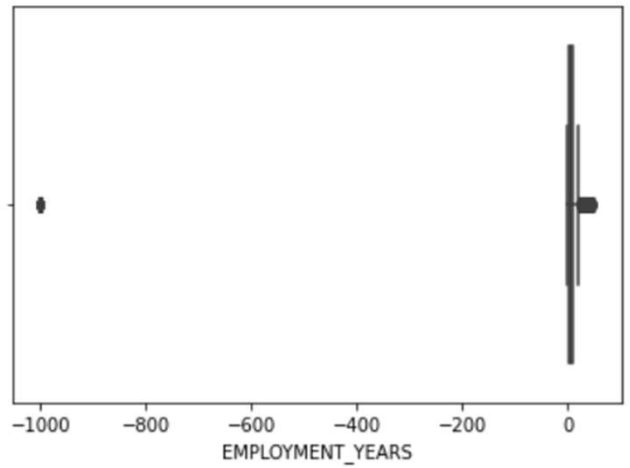
↓house tment	NAME_HOUSING_TYPE_Municipal apartment	NAME_HOUSING_TYPE_Office apartment	NAME_HOUSING_TYPE_Rented apartment	NAME_HOUSING_TYPE_With parents	AGE	EMPLOYMENT_YEARS
0	0	0	1	0	33.0	12.0
0	0	0	1	0	33.0	12.0
1	0	0	0	0	59.0	3.0
1	0	0	0	0	52.0	8.0
1	0	0	0	0	52.0	8.0
...	...	...	...	...	...	...
1	0	0	0	0	62.0	-1001.0
1	0	0	0	0	44.0	8.0
0	0	0	0	1	22.0	1.0
1	0	0	0	0	59.0	-1001.0
1	0	0	0	0	52.0	3.0

Driving new column number of employment from DAYS_EMPLOYED

```
1  #visualising EMPLOYMENT_YEARS for outliers
2  sns.boxplot(df_application['EMPLOYMENT_YEARS'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\seabo
a keyword arg: x. From version 0.12, the only valid positio
explicit keyword will result in an error or misinterpretati
  warnings.warn(
```

```
<AxesSubplot:xlabel='EMPLOYMENT_YEARS'>
```

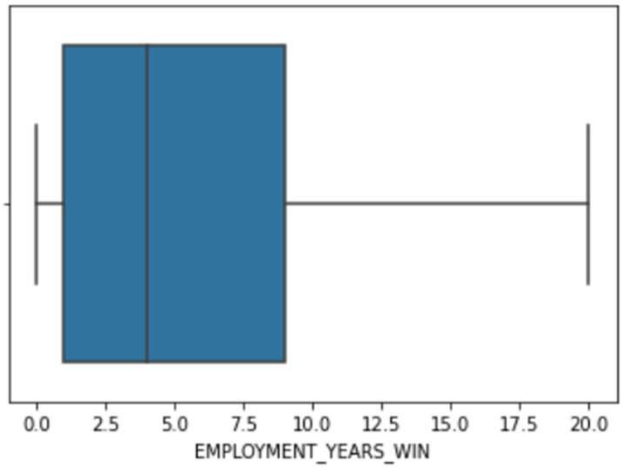


```
#use winsorize to eliminate the 17.5 & 5 percent of outliers & create new column EMPLOYMENT_YEARS_WIN
df_application['EMPLOYMENT_YEARS_WIN'] = winsorize((df_application['EMPLOYMENT_YEARS']),limits = [0.175,0.05],inplace = True
```

```
sns.boxplot(df_application['EMPLOYMENT_YEARS_WIN'])
```

```
C:\Users\shan2\Anaconda3\envs\mylib\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

```
<AxesSubplot:xlabel='EMPLOYMENT_YEARS_WIN'>
```



BINNING OCCUPATION TYPE COLUMN

```
def job_cat(col):

    if col['OCCUPATION_TYPE'] in ['Security staff','Sales staff','Laborers','Drivers','Core staff','High skill tech staff',
           'Cooking staff', 'Low-skill Laborers','Waiters/barmen staff']:
        return('Operational & Technical')
    if col['OCCUPATION_TYPE'] in ['Accountants','Medicine staff','Secretaries','HR staff', 'Realty agents','IT staff']:
        return('Professional')
    if col['OCCUPATION_TYPE'] in ['Managers']:
        return('Supervisors & Managers')
    if col['OCCUPATION_TYPE'] in ['Others']:
        return('Others')

```

```
df_application['OCCUPATION'] =  df_application.apply(lambda col: job_cat(col), axis=1)
df_application
```

_TYPE_Municipal apartment	NAME_HOUSING_TYPE_Office apartment	NAME_HOUSING_TYPE_Rented apartment	NAME_HOUSING_TYPE_With parents	AGE	EMPLOYMENT_YEARS_WIN	OCCUPATION
0	0	1	0	33.0	12.0	Others
0	0	1	0	33.0	12.0	Others
0	0	0	0	59.0	3.0	Operational & Technical
0	0	0	0	52.0	8.0	Operational & Technical
0	0	0	0	52.0	8.0	Operational & Technical
...	...	...	...	...	...	...
0	0	0	0	62.0	0.0	Others
0	0	0	0	44.0	8.0	Operational & Technical

ENCODING THE OCCUPATION COLUMN

```
1 df_application =pd.get_dummies(data = df_application, columns=['OCCUPATION'],drop_first = True)
```

- BINNING THE UNIQUE VALUES IN THE OCCUPATION COLUMN WILL MAKE THE ENCODING EFFECTIVE
- THE DATASET WILL NOT BECOME A SPARSE DATASET

ID	MONTHS_	STATUS
5001711	0	X
5001711	-1	0
5001711	-2	0
5001711	-3	0
5001712	0	C
5001712	-1	C
5001712	-2	C
5001712	-3	C
5001712	-4	C
5001712	-5	C
5001712	-6	C
5001712	-7	C
5001712	-8	C
5001712	-9	0
5001712	-10	0
5001712	-11	0
5001712	-12	0
5001712	-13	0
5001712	-14	0
5001712	-15	0
5001712	-16	0
5001712	-17	0
5001712	-18	0
5001713	0	X
5001713	-1	X
5001713	-2	X
5001713	-3	X

Credit Record Dataset

- Credit dataset first 10 record

```
1 df_credit.head(10)
```

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
5	5001712	-1	C
6	5001712	-2	C
7	5001712	-3	C
8	5001712	-4	C
9	5001712	-5	C

No of Rows and Columns

```
1 df_credit.shape
```

```
(1048575, 3)
```

Column Information

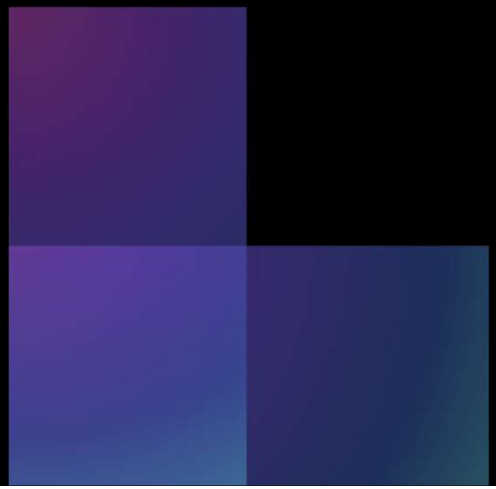
```
1 df_credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  object
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

- Missing/null values

```
1 df_credit.isna().sum()
```

```
ID                0
MONTHS_BALANCE    0
STATUS            0
dtype: int64
```



```
1  df_credit['STATUS'].value_counts()
```

```
C    442031
0    383120
X    209230
1     11090
5      1693
2       868
3       320
4       223
Name: STATUS, dtype: int64
```

- Label column has an imbalanced spread in unique values

- This will be handled later using stratify method

Unique values in the Label column

```
1 df_credit.STATUS.unique()
```

```
array(['X', '0', 'C', '1', '2', '3', '4', '5'], dtype=object)
```

Vintage Analysis

Create pivot table

```
df_credit_pivot = df_credit.pivot(index ='ID', columns = 'MONTHS_BALANCE',values = 'STATUS')
```

Feature Engineering

```
df_credit_pivot['opening_month'] = df_credit_grouped['MONTHS_BALANCE'].min()
```

```
df_credit_pivot['closing_month'] = df_credit_grouped['MONTHS_BALANCE'].max()
```

```
df_credit_pivot['ID'] = df_credit_pivot.index
```

```
df_credit_pivot = df_credit_pivot[['ID','opening_month','closing_month']]
```

- Create new columns for opening month and closing month
- These columns will be used to calculate the percentage of customers in each unique value in Label column

Merge the pivot table with the credit dataset

```
df_credit = pd.merge(df_credit,df_credit_pivot, on ='ID', how = 'left')
```

Calculate customer ratio

```
def calculate_rate(df_credit_pivot, command):

    df_credit_copy['status'] = None
    exec(command)
    sumagg = df_credit_copy.groupby('ID')['status'].agg(sum)
    df_credit_pivot = pd.merge(df_credit_pivot, sumagg, on = 'ID', how = 'left')
    df_credit_pivot.loc[df_credit_pivot['status'] > 1, 'status'] = 1
    rate = df_credit_pivot['status'].sum() / len(df_credit_pivot)

    return round(rate, 5)
```

```
command = "df_credit_copy.loc[(df_credit_copy['STATUS'] == '0') | (df_credit_copy['STATUS'] == '1') | (df_credit_copy['STATU
morethan1 = calculate_rate(df_credit_pivot, command)
```

```
command = "df_credit_copy.loc[(df_credit_copy['STATUS'] == '1') | (df_credit_copy['STATUS'] == '2') | (df_credit_copy['STATU
morethan30 = calculate_rate(df_credit_pivot, command)
```

```
command = "df_credit_copy.loc[(df_credit_copy['STATUS'] == '2') | (df_credit_copy['STATUS'] == '3' )| (df_credit_copy['STATU
morethan60 = calculate_rate(df_credit_pivot, command)
```

```
command = "df_credit_copy.loc[(df_credit_copy['STATUS'] == '3' )| (df_credit_copy['STATUS'] == '4' )| (df_credit_copy['STATU
morethan90 = calculate_rate(df_credit_pivot, command)
```

```
command = "df_credit_copy.loc[(df_credit_copy['STATUS'] == '4' )| (df_credit_copy['STATUS'] == '5'), 'status'] = 1"
morethan120 = calculate_rate(df_credit_pivot, command)
```

```
command = "df_credit_copy.loc[(df_credit_copy['STATUS'] == '5'), 'status'] = 1"
morethan150 = calculate_rate(df_credit_pivot, command)
```

```
summary_df = pd.DataFrame({'situation':[
                            'past due more than 1 day',
                               'past due more than 30 days',
                               'past due more than 60 days',
                               'past due more than 90 days',
                               'past due more than 120 days',
                               'past due more than 150 days'],
                    'bad customer ratio':[morethan1,
                               morethan30,
                               morethan60,
                               morethan90,
                               morethan120,
                               morethan150,
                    ]})
summary_df
```

	situation	bad customer ratio
0	past due more than 1 day	0.87054
1	past due more than 30 days	0.11634
2	past due more than 60 days	0.01450
3	past due more than 90 days	0.00720
4	past due more than 120 days	0.00528
5	past due more than 150 days	0.00424

- Table of Customer Ratio
- Past due more than 30 days will be considered bad customers
- From 1 to 5 will be labelled as 1 in Label column

```
df_credit_copy.replace(to_replace = 'X',
                       value = 0,inplace = True)
df_credit_copy.replace(to_replace = 'C',
                       value = 0,inplace = True)
```

```
df_credit_copy.STATUS = df_credit_copy.STATUS.astype('int8')
```

```
df_credit_copy.loc[(df_credit_copy.STATUS >= 1),'STATUS'] =1
```

Label column

- X – No loan for the month
- C- Paid off the loan for the month

- X and C have been labelled as 0

```
1 pd.read_csv('credit.csv',index_col = 0)
```

	ID	STATUS
0	5001711	0
1	5001712	0
2	5001713	0
3	5001714	0
4	5001715	0
...	...	...
45980	5150482	0
45981	5150483	0
45982	5150484	0
45983	5150485	0
45984	5150487	0

45985 rows × 2 columns

- Label Column

```
#merge Credit Dataset with Application Dataset
merge_dataset = pd.merge(df_application, credit, on = 'ID', how = 'right')
```

```
merge_dataset
```

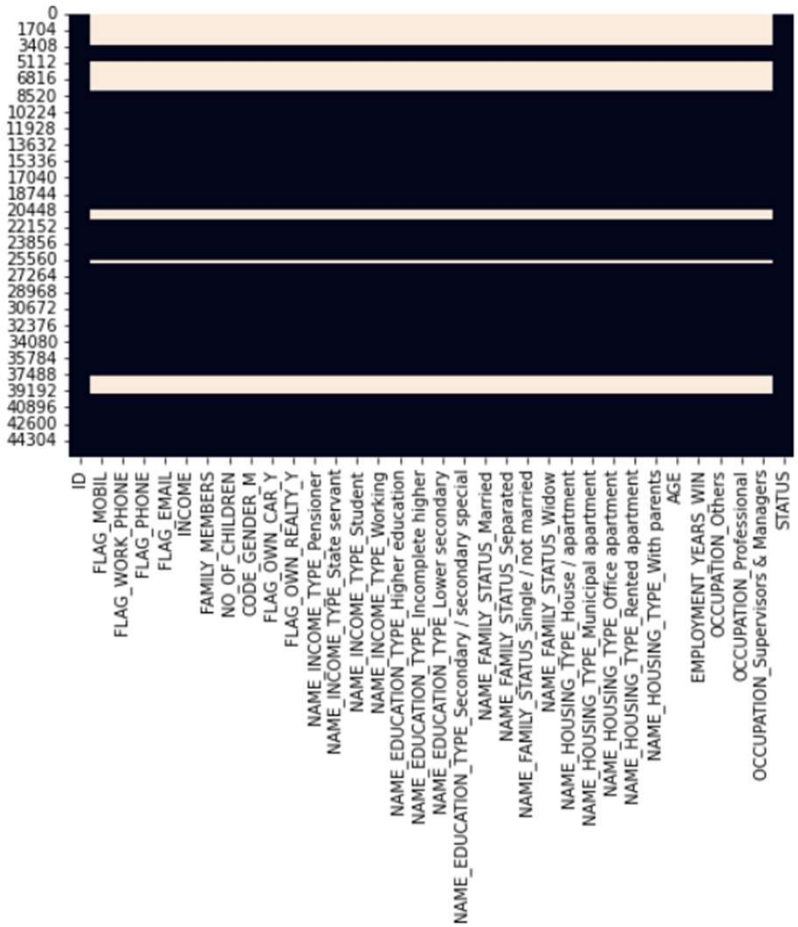
'PE_Rented apartment	NAME_HOUSING_TYPE_With parents	AGE	EMPLOYMENT_YEARS_WIN	OCCUPATION_Others	OCCUPATION_Professional	OCCUPATION_Supervisors & Managers	STATUS
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
NaN	NaN	NaN	NaN	NaN	NaN	NaN	0
...	...	...	...	...	...	...	...
0.0	0.0	30.0	5.0	0.0	0.0	0.0	0
0.0	0.0	30.0	5.0	0.0	0.0	0.0	0
0.0	0.0	30.0	5.0	0.0	0.0	0.0	0
0.0	0.0	30.0	5.0	0.0	0.0	0.0	0
0.0	0.0	53.0	6.0	0.0	0.0	0.0	0

- MERGED DATASET

- Used right join to merge the datasets
- Reason - credit dataset had less data than application data
- Left join will result in null values
- If null values in Label column filled with zero or one, can create bias in dataset

```
#visualise the nul values
fig = plt.figure(figsize = (8,5))
sns.heatmap(merge_dataset.isnull(),cbar=False)
```

```
<AxesSubplot:>
```



- Null value rows are 9528
- Null values in the merged dataset have been dropped to maintain the integrity of the dataset

```
1  X = merge_dataset.drop(['STATUS'],axis =1)
```

```
1  y = merge_dataset.STATUS
```

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=123,stratify=y)
```

SPILTTING THE DATASET

FEATURE SCALING

- Normalization is suitable for a dataset that does not follow Gaussian distribution
- Standardization is good when the dataset follows Gaussian distribution
- Feature scaling applied only to train dataset to avoid data leakage

Option-1

```
#NORMALIZATION
scaler = MinMaxScaler()
```

Option-2

```
#STANDARDIZATION
# scaler = StandardScaler()
```

```
X_train_scaled =pd.DataFrame(scaler.fit_transform(X_train), columns=X_train.columns)
```

- For this dataset fitting Normalization gives a better result of AUC than Standardization
- This could be due to the dataset not following Gaussian distribution

The Big Question – Normalize or Standardize?

Normalization vs. standardization is an eternal question among machine learning newcomers. Let me elaborate on the answer in this section.

- Normalization is good to use when you know that the distribution of your data does not follow a Gaussian distribution. This can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors and Neural Networks.
- Standardization, on the other hand, can be helpful in cases where the data follows a Gaussian distribution. However, this does not have to be necessarily true. Also, unlike normalization, standardization does not have a bounding range. So, even if you have outliers in your data, they will not be affected by standardization.

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. **You can always start by fitting your model to raw, normalized and standardized data and compare the performance for best results.**

Source: https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/

Difference between Normalization and Standardization

S.NO.	Normalization	Standardization
1.	Minimum and maximum value of features are used for scaling	Mean and standard deviation is used for scaling.
2.	It is used when features are of different scales.	It is used when we want to ensure zero mean and unit standard deviation.
3.	Scales values between [0, 1] or [-1, 1].	It is not bounded to a certain range.
4.	It is really affected by outliers.	It is much less affected by outliers.
5.	Scikit-Learn provides a transformer called `MinMaxScaler` for Normalization.	Scikit-Learn provides a transformer called `StandardScaler` for standardization.
6.	This transformation squishes the n-dimensional data into an n-dimensional unit hypercube.	It translates the data to the mean vector of original data to the origin and squishes or expands.
7.	It is useful when we don't know about the distribution	It is useful when the feature distribution is Normal or Gaussian.
8.	It is a often called as Scaling Normalization	It is a often called as Z-Score Normalization.

Source: https://www.geeksforgeeks.org/normalization-vs-standardization/

Option-1

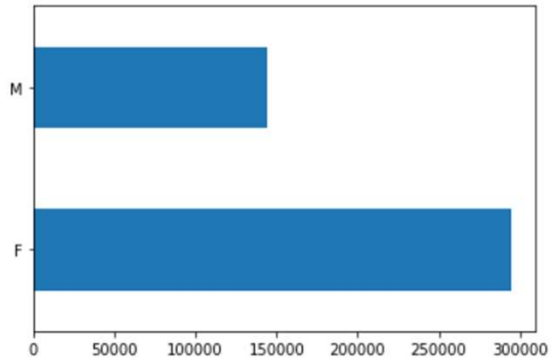
```
ros = RandomOverSampler(sampling_strategy = 'not majority')
X_res,y_res = ros.fit_resample(X_train,y_train)
```

Option-2 ¶

```
# sm = SMOTENC(random_state=123, categorical_features=[31])
# X_res, y_res = sm.fit_resample(X_train_scaled,y_train)
```

```
df_application['CODE_GENDER'].value_counts().plot(kind='barh')
```

```
<AxesSubplot:>
```

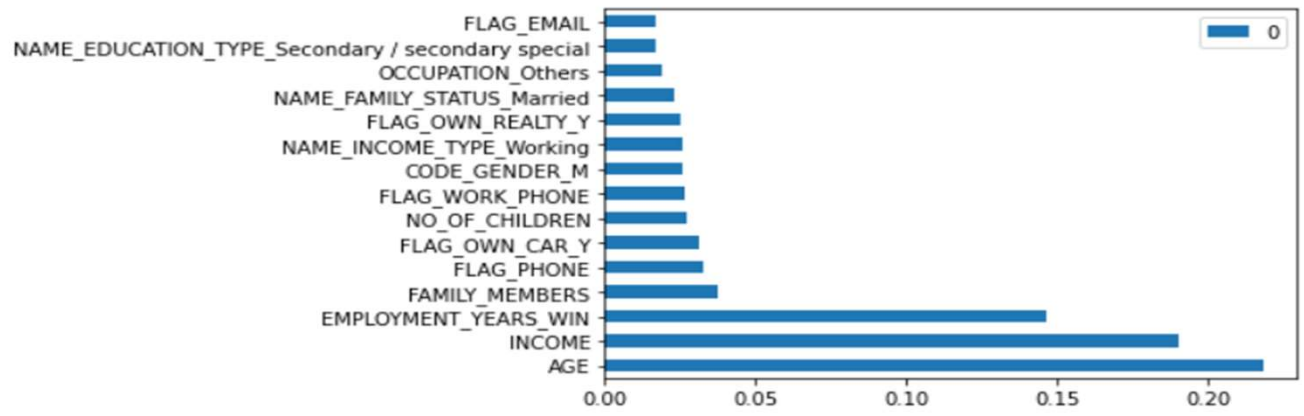


HANDING IMBALANCED DATA

- For the dataset, Random Sampler resulted in high AUC than SMOTNC

- Possible reason could be the algorithm fitted for the dataset does not work well with the syntactic data points created by SMOTENC

```
# best = ExtraTreesClassifier()
best = RandomForestClassifier()
best_fi = best.fit(X_res, y_res)
df_best_fi = pd.DataFrame(best_fi.feature_importances_,index=X_res.columns)
df_best_fi.nlargest(15,df_best_fi.columns).plot(kind='barh')
plt.show()
```



FEATURE SELECTION

- Feature selection helps to reduce the number of features and reduce the complexity of the model
- Model performance can be improved

MODEL SELECTION USING PYCARET

```
1 best_model = compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
dummy	Dummy Classifier	0.8834	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.1220
lightgbm	Light Gradient Boosting Machine	0.8823	0.6432	0.1216	0.4823	0.1939	0.1542	0.1980	0.9550
ada	Ada Boost Classifier	0.8819	0.5294	0.0013	0.0203	0.0025	-0.0008	-0.0031	2.2030
gbc	Gradient Boosting Classifier	0.8817	0.5638	0.0111	0.3049	0.0213	0.0133	0.0376	5.7250
et	Extra Trees Classifier	0.8729	0.7492	0.3370	0.4414	0.3814	0.3122	0.3161	3.5420
rf	Random Forest Classifier	0.8726	0.7624	0.3498	0.4419	0.3896	0.3198	0.3229	3.0690
dt	Decision Tree Classifier	0.8630	0.7108	0.3602	0.4023	0.3795	0.3030	0.3038	0.4530
knn	K Neighbors Classifier	0.7590	0.7084	0.5386	0.2511	0.3423	0.2181	0.2421	1.2120
lr	Logistic Regression	0.5600	0.5238	0.4654	0.1257	0.1979	0.0175	0.0246	5.2280
ridge	Ridge Classifier	0.5415	0.0000	0.5081	0.1287	0.2054	0.0237	0.0348	0.1830
lda	Linear Discriminant Analysis	0.5411	0.5393	0.5081	0.1286	0.2052	0.0235	0.0345	0.5430
svm	SVM - Linear Kernel	0.4233	0.0000	0.6000	0.0700	0.1253	0.0000	0.0000	2.4490
qda	Quadratic Discriminant Analysis	0.3743	0.5163	0.6747	0.1181	0.2010	0.0031	0.0063	0.2130
nb	Naive Bayes	0.3655	0.5199	0.6993	0.1197	0.2044	0.0066	0.0144	0.1850

SELECTING TOP FOUR ALGORITHMS

```
# clf = KNeighborsClassifier()    #Lowest
# clf = RandomForestClassifier()  #Same
# clf = ExtraTreesClassifier()    #Same
clf = DecisionTreeClassifier()    #Hightest


clf.fit(X_res_best, y_res)
```

```
DecisionTreeClassifier()
```

```
predicted= clf.predict(X_test_best)
```

- DecisionTreeClassifier gives the highest AUC of the four algorithms
- Possible reasons for the other 3 algorithms' performance:
  - Need a larger dataset
  - The way data has been pre-processed

EVALUATION OF THE MODEL

ACCURACY

```
:  1  print("Accuracy:",metrics.accuracy_score(y_test, predicted))
```

```
Accuracy: 0.7888096544157981
```

Accuracy is 79%, but for the imbalanced dataset, accuracy is not a good measure

Because the accuracy score will be high, even the model fails to predict the minority event outcome in the dataset

Precision and Recall

```
1 print(classification_report(y_test, predicted))
```

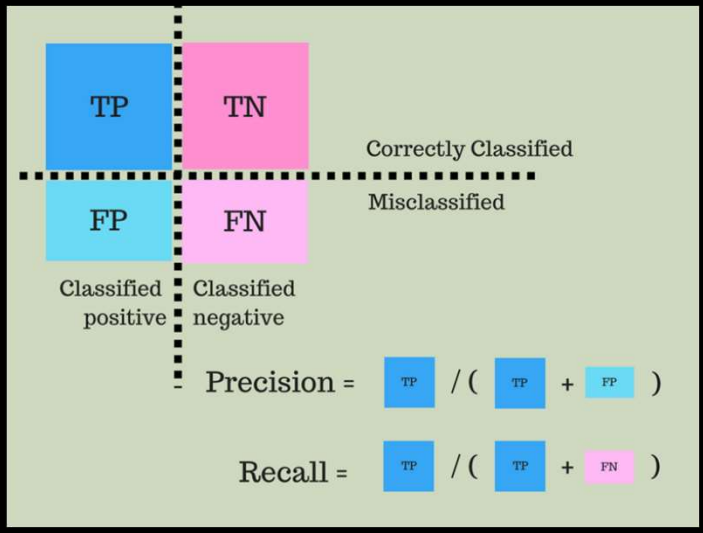
	precision	recall	f1-score	support
0	0.94	0.81	0.87	8042
1	0.30	0.61	0.40	1073
accuracy			0.79	9115
macro avg	0.62	0.71	0.64	9115
weighted avg	0.86	0.79	0.82	9115

Recall = TP/(TP+FN)

Precision = TP/(TP + FP)

Recall for 0 (good applicants): high
Recall for 1 (bad applicants): low
Label 1 has more false negatives

Precision for 0 (good applicants) : high
Precision for 1 (bad applicants) : very low
Label 1 has more false positives



Source: https://nlpforhackers.io/classification-performance-metrics/

AUC

```
1  auc = metrics.roc_auc_score(y_test, predicted)
2  auc
```

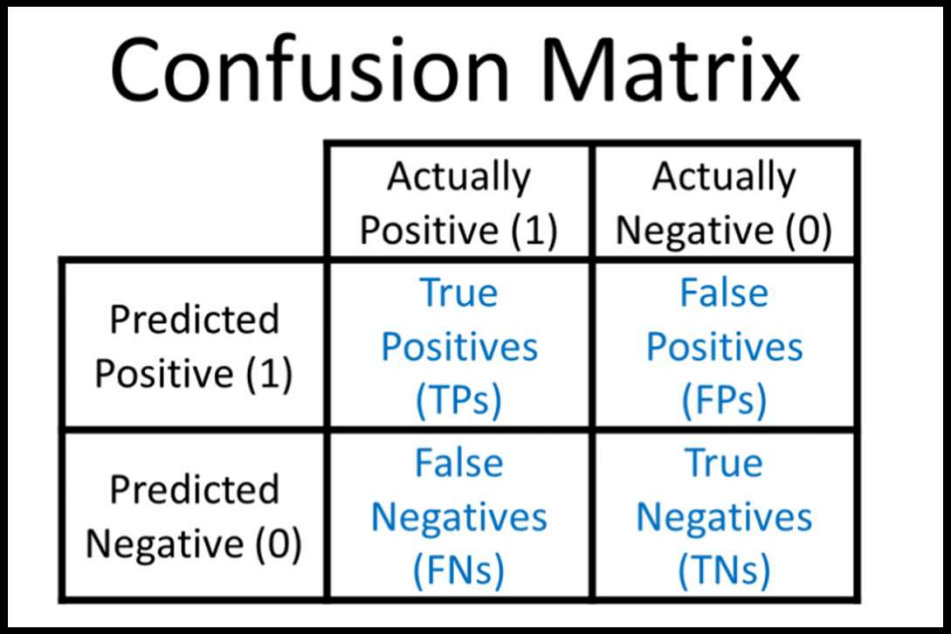
```
0.7107157947337521
```

- ROC AUC is used to measure the performance of an imbalanced dataset model
- Reason: The curve is plotted with True Positive Rate & False Positive Rate
- Higher AUC curve higher the model ‘s performance

CONFUSION MATRIX

```
1 print(confusion_matrix(y_test, predicted))
```

```
[[6537 1505]
 [ 420  653]]
```



Source: https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/

HYPERPARAMETER TUNING

DEFINING PARAMETER

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(50, 200, num = 11)]
# Minimum number of samples required to split a node
min_samples_split = [5, 10, 15]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [100, 311, 522, 733, 944, 1155, 1366, 1577, 1788, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [50, 6
5, 80, 95, 110, 125, 140, 155, 170, 185, 200], 'min_samples_split': [5, 10, 15], 'min_samples_leaf': [1, 2, 4], 'bootstrap': [T
rue, False]}
```

```
rf = RandomForestRegressor()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 75, cv = 3, verbose=2, random_sta
rf_random.fit(X_res_best, y_res)
```

```
Fitting 3 folds for each of 75 candidates, totalling 225 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  6.6min
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed: 43.4min
[Parallel(n_jobs=-1)]: Done 225 out of 225 | elapsed: 79.0min finished
```

```
RandomizedSearchCV(cv=3, estimator=RandomForestRegressor(), n_iter=75,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [50, 65, 80, 95, 110, 125,
                                                      140, 155, 170, 185, 200],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [5, 10, 15],
                                        'n_estimators': [100, 311, 522, 733,
                                                         944, 1155, 1366, 1577,
                                                         1788, 2000]},
                   random_state=42, verbose=2)
```

RandomizedSearchCV to find the best hyperparameters for the model

```
1 rf_random.best_params_
```

```
{'n_estimators': 733,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 140,
 'bootstrap': False}
```

```
1 rf_random.best_estimator_
```

```
RandomForestRegressor(bootstrap=False, max_depth=140, max_features='sqrt',
                      min_samples_split=5, n_estimators=733)
```

```
1 rf_random.best_score_
```

```
0.012039084609565412
```

- The appropriate hyperparameters to use to get the best performance by the model

```
clf_tune = DecisionTreeClassifier(
 min_samples_split= 5,
 min_samples_leaf= 1,
 max_features = 'sqrt',
 max_depth= 140,
)

clf_tune.fit(X_res_best, y_res)
```

```
DecisionTreeClassifier(max_depth=140, max_features='sqrt', min_samples_split=5)
```

```
predicted= clf_tune.predict(X_test_best)
```

```
print("Accuracy:",metrics.accuracy_score(y_test, predicted))
```

```
Accuracy: 0.7897970378496983
```

```
print(classification_report(y_test, predicted))
```

```
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      8042
           1       0.30      0.61      0.41      1073

    accuracy                           0.79      9115
   macro avg       0.62      0.71      0.64      9115
weighted avg       0.87      0.79      0.82      9115
```

```
auc = metrics.roc_auc_score(y_test, predicted)
auc
```

```
0.7120829763035768
```

```
print(confusion_matrix(y_test, predicted))
```

```
[[6544 1498]
 [ 418  655]]
```

- AUC did not change at all compared to the untuned model
- Tuned Model AUC – 71
- Untuned Model AUC – 71
- The reason could be not enough data in the dataset,
- It could be due to overfitting

If I redo the project:

I will look for more data to increase the reliability of the model

I would like to do feature engineering like debt to income ratio

Github link: https://github.com/y2131/Machine-Learning-Project-on-Credit-Card-Applications.git

THANK YOU