Supervised Machine Learning: Classification Course Final Project Report

PREDICTIVE MODEL FOR CREDIT SCORING USING LOGISTIC REGRESSION, RANDOMFOREST AND XGBOOST

I. FRAME WORK

- Data source: Public Data from Github, link: https://raw.githubusercontent.com/NTTrung9204/assignment_basic/master/Credit%20Score/train.csv
- **Purpose of Modeling**: to classify personal customers into 3 labels: 'Good', 'Standard', 'Poor' base on some input features of that customer (features about income, debt situation and behaviors)
- **Application in real life**: perfom a credit scoring in a retail bank to assess customers' risk before banks give a loan to customers and in the time customers have a loan.

II. DATA EXPLORATION

- Information about Data:

  - Shape: 100.000 rows and 29 columns

  - Data have no null and no duplicated entries

  - Data have **sensitive information** included: ‘SSN’, ‘Name’ and **unusable information**: ‘Unnamed: 0’, ‘ID’, ‘Customer_ID’

  - Target variables: ‘Credit_Score’ with 3 nunique: 'Good', 'Standard', 'Poor'

```
Data columns (total 29 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Unnamed: 0                100000 non-null  int64
 1   ID                        100000 non-null  int64
 2   Customer_ID               100000 non-null  int64
 3   Month                     100000 non-null  int64
 4   Name                      100000 non-null  object
 5   Age                       100000 non-null  float64
 6   SSN                       100000 non-null  float64
 7   Occupation                100000 non-null  object
 8   Annual_Income             100000 non-null  float64
 9   Monthly_Inhand_Salary     100000 non-null  float64
 10  Num_Bank_Accounts         100000 non-null  float64
 11  Num_Credit_Card           100000 non-null  float64
 12  Interest_Rate             100000 non-null  float64
 13  Num_of_Loan               100000 non-null  float64
 14  Type_of_Loan              100000 non-null  object
 15  Delay_from_due_date       100000 non-null  float64
 16  Num_of_Delayed_Payment    100000 non-null  float64
 17  Changed_Credit_Limit      100000 non-null  float64
 18  Num_Credit_Inquiries      100000 non-null  float64
 19  Credit_Mix                100000 non-null  object
 20  Outstanding_Debt          100000 non-null  float64
 21  Credit_Utilization_Ratio  100000 non-null  float64
 22  Credit_History_Age        100000 non-null  float64
 23  Payment_of_Min_Amount     100000 non-null  object
 24  Total_EMI_per_month       100000 non-null  float64
 25  Amount_invested_monthly   100000 non-null  float64
 26  Payment_Behaviour         100000 non-null  object
 27  Monthly_Balance           100000 non-null  float64
 28  Credit_Score              100000 non-null  object
dtypes: float64(18), int64(4), object(7)
memory usage: 22.1+ MB
```

III. DATA CLEANING AND DETAILS EDA

- **Reducing size by listed high correlation numerical variables (use df.corr())**

  => Columns need to drop (sensitive, unusable and high correlation to another numerical variables)

  df = df.drop(columns=['ID' ,'Customer_ID', 'Name', 'SSN','Monthly_Inhand_Salary', 'Amount_invested_monthly', 'Monthly_Balance', 'Num_of_Loan', 'Interest_Rate', 'Num_Credit_Inquiries'])

- Check more details about nunique of every Dtypes (low nunique: nunique < 10% length of dataset)

  + high nunique and dtype = object : 0 columns

  + low nunique and dtype = object: columns: 'Occupation', 'Type_of_Loan', 'Credit_Mix', 'Payment_of_Min_Amount', 'Payment_Behaviour', 'Credit_Score'

  => 'Type_of_Loan' with nunique: 6261

  + high nunique and dtype = numeric: columns: 'Annual_Income', 'Outstanding_Debt', 'Credit_Utilization_Ratio', 'Total_EMI_per_month'

  + low nunique and dtype = numeric ['Month', 'Age', 'Num_Bank_Accounts', 'Num_Credit_Card', 'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit', 'Credit_History_Age']

  => Month : nunique 8

  => Num_Bank_Accounts : nunique 12

  => Num_Credit_Card : nunique 12

  => Num_of_Delayed_Payment : nunique 26

  => Age : nunique 43

- **Classify into 3 kinds of columns to preprocessing:**

  + Columns need to encode (numeric and objects): 'Month', 'Occupation', 'Credit_Mix', 'Payment_of_Min_Amount', 'Payment_Behaviour'

  + Columns is numeric can scale: 'Age', 'Annual_Income', 'Num_Bank_Accounts', 'Num_Credit_Card', 'Delay_from_due_date', 'Num_of_Delayed_Payment', 'Changed_Credit_Limit', 'Outstanding_Debt', 'Credit_Utilization_Ratio', 'Credit_History_Age', 'Total_EMI_per_month'

  + Columns need to be sparsed: 'Type_of_Loan'

```
Top Absolute Correlations
Unnamed: 0              ID                        1.000000
Annual_Income           Monthly_Inhand_Salary     0.998154
Monthly_Inhand_Salary   Amount_invested_monthly   0.807326
Annual_Income           Amount_invested_monthly   0.806281
Num_of_Loan             Outstanding_Debt          0.638713
Interest_Rate           Num_Credit_Inquiries      0.632562
                        Outstanding_Debt          0.629414
Outstanding_Debt        Credit_History_Age        0.629320
Monthly_Inhand_Salary   Monthly_Balance           0.626631
Annual_Income           Monthly_Balance           0.625640
Num_Credit_Inquiries    Credit_History_Age        0.609679
Num_of_Loan             Credit_History_Age        0.605727
Num_Bank_Accounts       Num_of_Delayed_Payment    0.601842
dtype: float64
```

Correlation columns > 0.6

	Type_of_Loan
No Data	11408
Not Specified	1408
Credit-Builder Loan	1280
Personal Loan	1272
Debt Consolidation Loan	1264
Student Loan	1240
Payday Loan	1200
Mortgage Loan	1176
Auto Loan	1152
Home Equity Loan	1136
Personal Loan, and Student Loan	320
Not Specified, and Payday Loan	272
Mortgage Loan, and Home Equity Loan	264
Student Loan, and Payday Loan	256
Student Loan, and Credit-Builder Loan	248
Credit-Builder Loan, and Not Specified	248
Payday Loan, and Debt Consolidation Loan	240
Payday Loan, and Auto Loan	240
Mortgage Loan, and Not Specified	232
Payday Loan, and Personal Loan	232
Auto Loan, and Personal Loan	232
Payday Loan, and Student Loan	232
Credit-Builder Loan, and Mortgage Loan	232
Personal Loan, and Credit-Builder Loan	232
Auto Loan, and Payday Loan	224
Debt Consolidation Loan, and Home Equity Loan	224
Not Specified, and Auto Loan	216
Home Equity Loan, and Student Loan	216

Type of Loan value_counts

III. DATA CLEANING AND DETAILS EDA

- **Transformation with ‘Type_of_Loan’**

  + Drop ‘Type_of_Loan’ with value ‘No Data’

  + Spare into columns of every type of Loan with value: 0 and 1

  + Drop columns ‘Type_of_Loan’

Out[3]:

Unnamed: 0	Type_of_Loan_Auto Loan	Type_of_Loan_Credit-Builder Loan	Type_of_Loan_Debt Consolidation Loan	Type_of_Loan_Home Equity Loan	Type_
0	1	1	0	1	
1	1	1	0	1	
2	1	1	0	1	
3	1	1	0	1	
4	1	1	0	1	

Type_of_Loan after sparsed

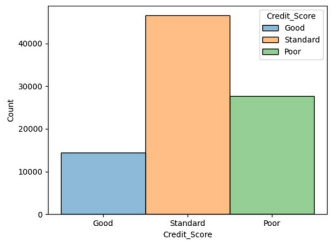
```
Data columns (total 26 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Type_of_Loan_Auto Loan                88592 non-null  int64
 1   Type_of_Loan_Credit-Builder Loan      88592 non-null  int64
 2   Type_of_Loan_Debt Consolidation Loan  88592 non-null  int64
 3   Type_of_Loan_Home Equity Loan         88592 non-null  int64
 4   Type_of_Loan_Mortgage Loan            88592 non-null  int64
 5   Type_of_Loan_Not Specified            88592 non-null  int64
 6   Type_of_Loan_Payday Loan              88592 non-null  int64
 7   Type_of_Loan_Personal Loan            88592 non-null  int64
 8   Type_of_Loan_Student Loan             88592 non-null  int64
 9   Month                                 88592 non-null  int64
 10  Age                                   88592 non-null  float64
 11  Occupation                            88592 non-null  object
 12  Annual_Income                         88592 non-null  float64
 13  Num_Bank_Accounts                     88592 non-null  float64
 14  Num_Credit_Card                       88592 non-null  float64
 15  Delay_from_due_date                   88592 non-null  float64
 16  Num_of_Delayed_Payment                88592 non-null  float64
```

df after transform Type_of_Loan

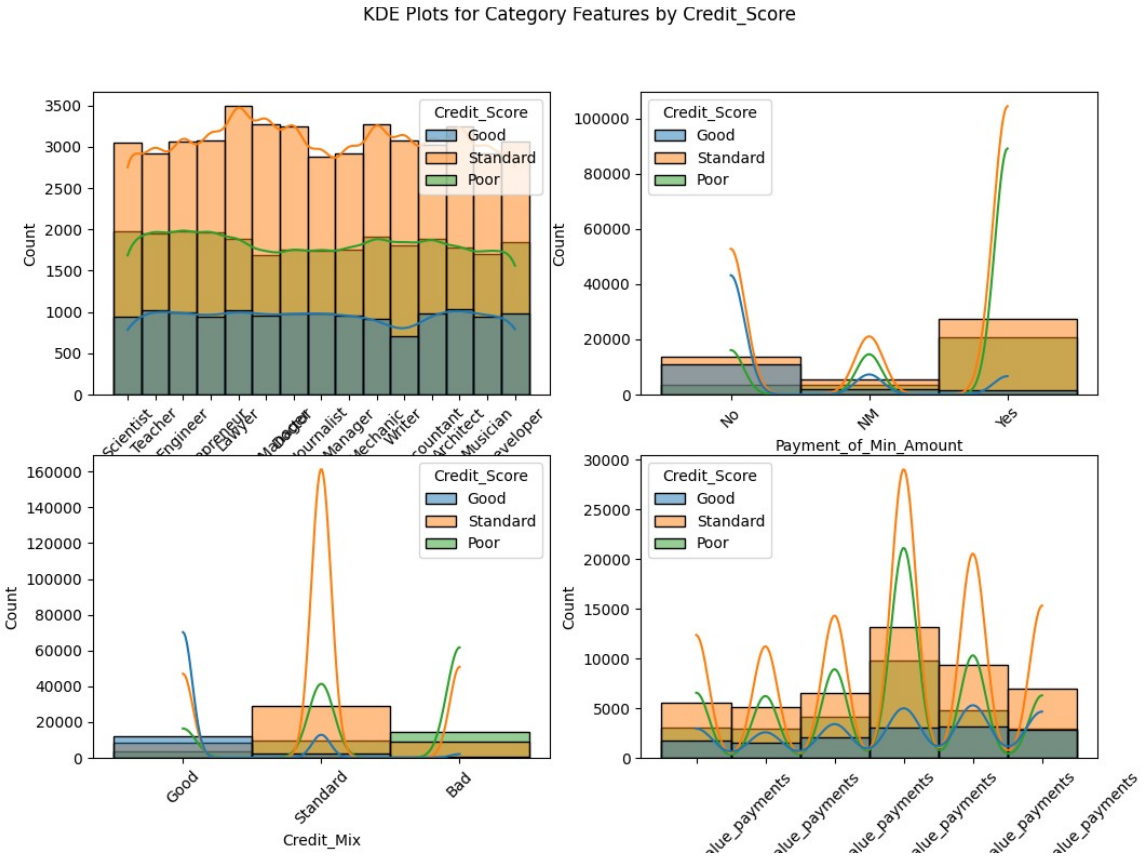
III. DATA CLEANING AND DETAILS EDA

1. **Plot value_counts of ‘Credit_Score’**

=> Imbalance labels



2. Plot X features by every label of ‘Credit_Score’ to check the type of relationship between X and y

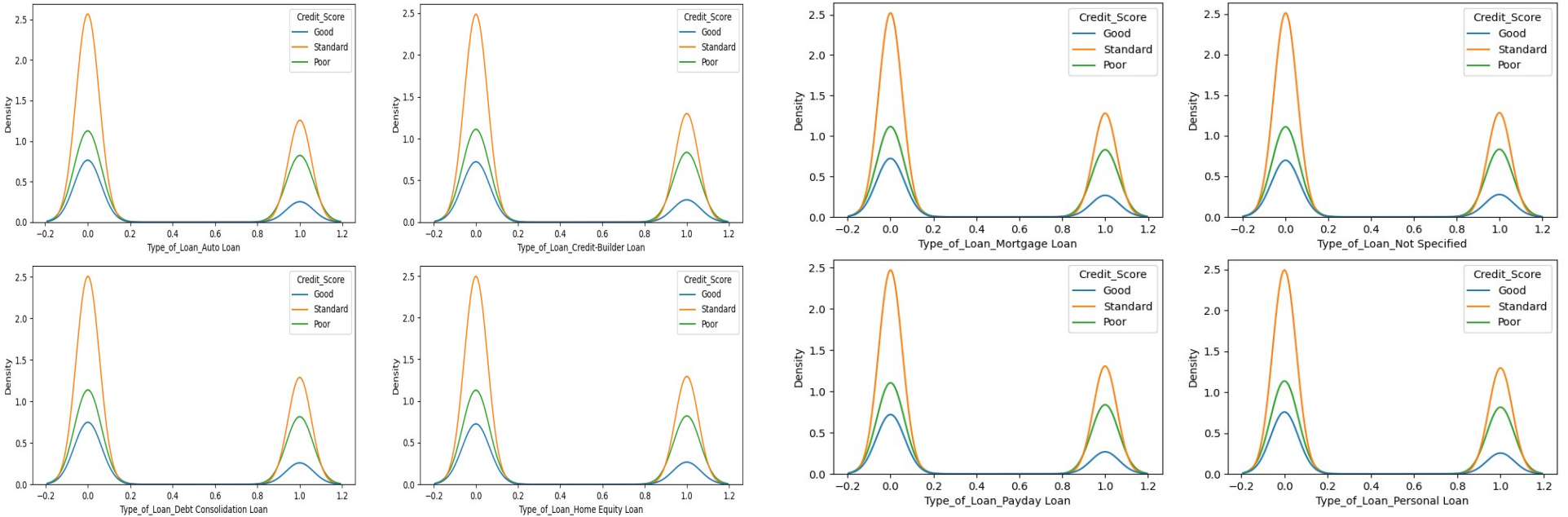


Category variables with by every labels of Credit_Score

III. DATA CLEANING AND DETAILS EDA

2. Plot X features by every label of ‘Credit_Score’ to check the type of relationship between X and y

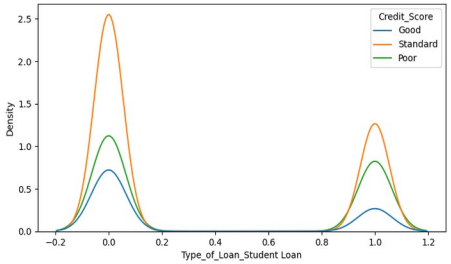
Numerical variables with by every labels of Credit_Score

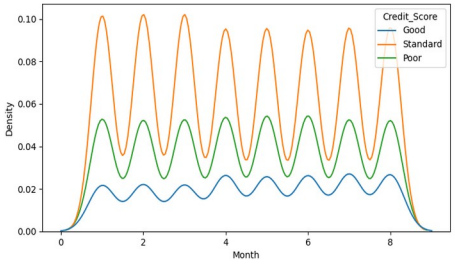


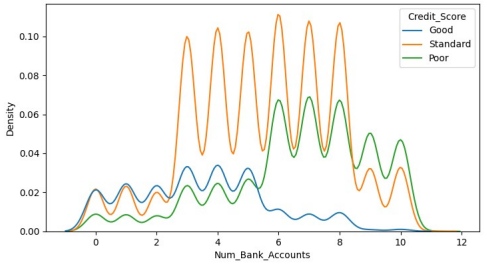
III. DATA CLEANING AND DETAILS EDA

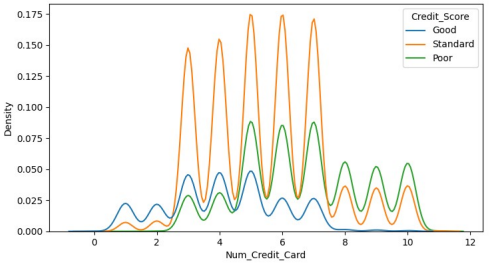
2. Plot X features by every label of ‘Credit_Score’ to check the type of relationship between X and y

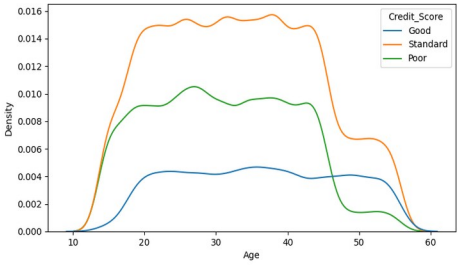
Numerical variables with by every labels of Credit_Score

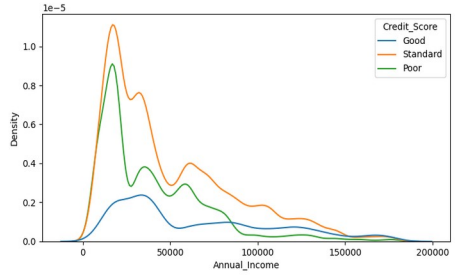


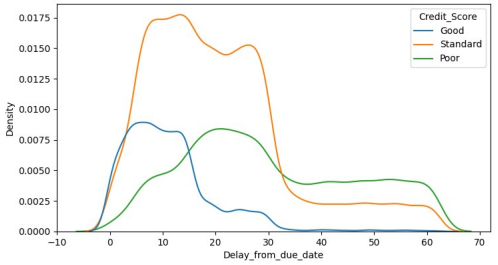


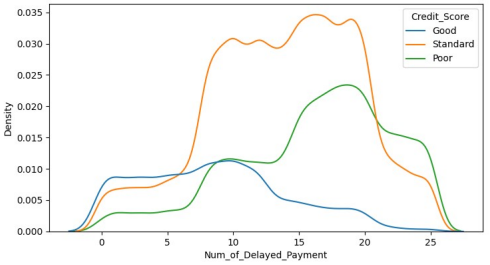






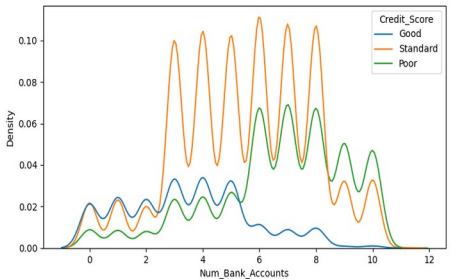


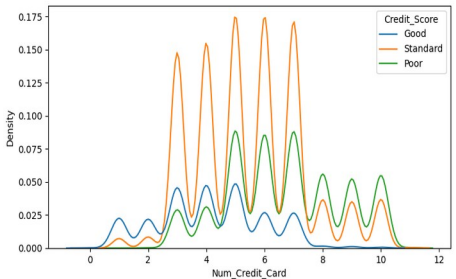


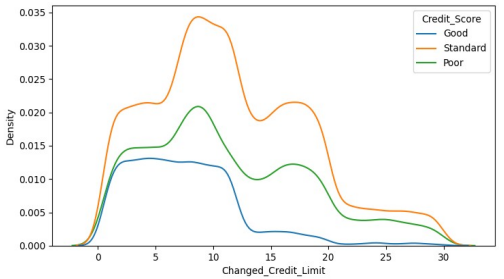


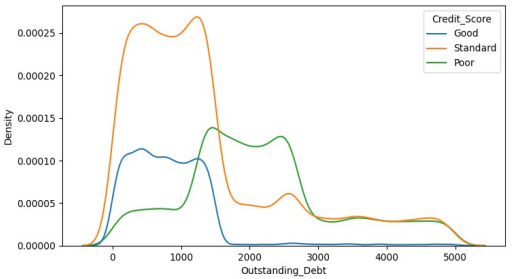
III. DATA CLEANING AND DETAILS EDA

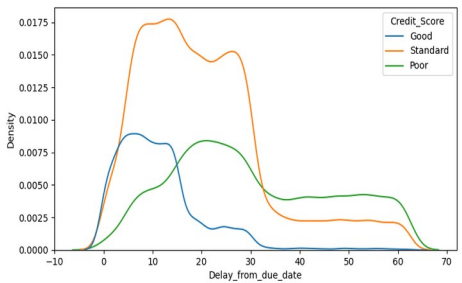
2. Plot X features by every label of ‘Credit_Score’ to check the type of relationship between X and y

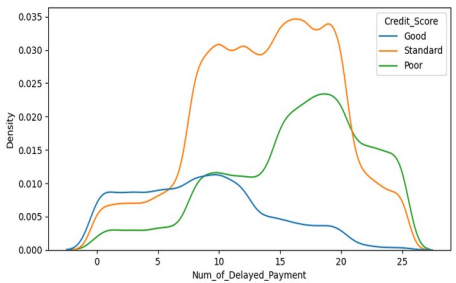


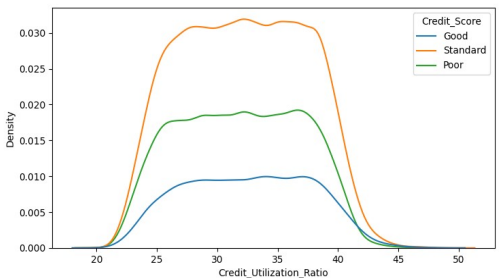


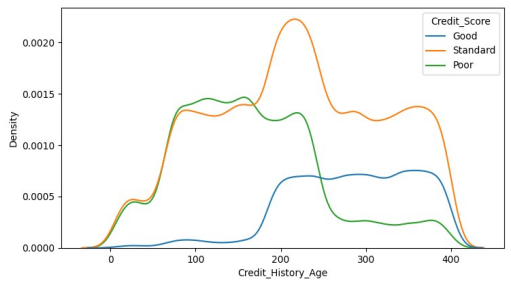


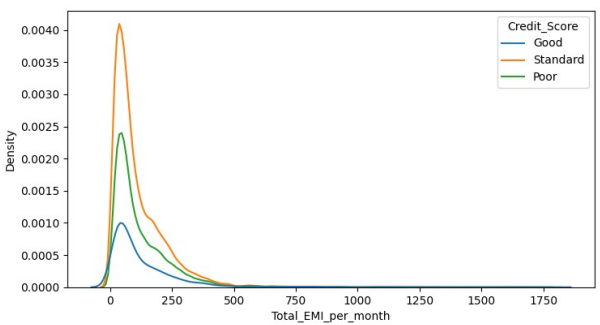












Numerical variables with by every labels of Credit_Score

III. DATA CLEANING AND DETAILS EDA

Conclusion on Visualization:

+ Target variable: **‘Credit_Score’ is imbalance**
+ Relationship between **X features and y: non-linear**=> should use polynomial features to catch the relationship when using Linear Model
+ feature **‘Month’ have no relationship with ‘Credit_Score’** => need to drop from dataset

Dataset after cleaning and EDA:

```
Data columns (total 25 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Type_of_Loan_Auto Loan                88592 non-null  int64
 1   Type_of_Loan_Credit-Builder Loan      88592 non-null  int64
 2   Type_of_Loan_Debt Consolidation Loan  88592 non-null  int64
 3   Type_of_Loan_Home Equity Loan         88592 non-null  int64
 4   Type_of_Loan_Mortgage Loan            88592 non-null  int64
 5   Type_of_Loan_Not Specified            88592 non-null  int64
 6   Type_of_Loan_Payday Loan              88592 non-null  int64
 7   Type_of_Loan_Personal Loan            88592 non-null  int64
 8   Type_of_Loan_Student Loan             88592 non-null  int64
 9   Age                                   88592 non-null  float64
 10  Occupation                            88592 non-null  object
 11  Annual_Income                         88592 non-null  float64
 12  Num_Bank_Accounts                     88592 non-null  float64
 13  Num_Credit_Card                       88592 non-null  float64
 14  Delay_from_due_date                   88592 non-null  float64
 15  Num_of_Delayed_Payment                88592 non-null  float64
 16  Changed_Credit_Limit                  88592 non-null  float64
 17  Credit_Mix                            88592 non-null  object
 18  Outstanding_Debt                      88592 non-null  float64
 19  Credit_Utilization_Ratio              88592 non-null  float64
 20  Credit_History_Age                    88592 non-null  float64
 21  Payment_of_Min_Amount                 88592 non-null  object
 22  Total_EMI_per_month                   88592 non-null  float64
 23  Payment_Behaviour                     88592 non-null  object
 24  Credit_Score                          88592 non-null  object
dtypes: float64(11), int64(9), object(5)
memory usage: 17.6+ MB
```

IV. PREPROCESSING

- **Devide data in train, validation and test set** by StratifiedShuffleSplit (devide df into train set – test set,then devide train set into train – validation)
- **Check skew of X features and scale**:

  + skewed columns (abs(skew) >=1): 'Total_EMI_per_month', 'Annual_Income', 'Outstanding_Debt'

  + all skewed columns is positive skew => apply log transform to X_train, X_val and X_test

  + Standard scale to X_train, X_val, X_test

- **Encode category variables**: Apply OneHotEncoder from category_encoders library and LabelEncoder from sklearn.preprocessing
- **Create another train, validation, test set from orginal but apply Polinomial features (degree = 2)** to use for Logistic Regression: X_train_poly, X_val_poly and X_test_poly

```
Train set (56698, 47) (56698,)
Test set (17719, 47) (17719,)
Val set (14175, 47) (14175,)
```

Shape of train, test, validation set

```
In[8]: X_train[scale_skew_cols].skew()
Out[8]:
Total_EMI_per_month    3.972286
Annual_Income          1.177540
Outstanding_Debt       1.081688
dtype: float64
```

Skewness before transform

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 56698 entries, 98170 to 69117
Data columns (total 47 columns):
 #   Column                                  Non-Null Count  Dtype
---  ------                                  --------------  -----
 0   Type_of_Loan_Auto Loan                  56698 non-null  int64
 1   Type_of_Loan_Credit-Builder Loan        56698 non-null  int64
 2   Type_of_Loan_Debt Consolidation Loan    56698 non-null  int64
 3   Type_of_Loan_Home Equity Loan           56698 non-null  int64
 4   Type_of_Loan_Mortgage Loan              56698 non-null  int64
 5   Type_of_Loan_Not Specified              56698 non-null  int64
 6   Type_of_Loan_Payday Loan                56698 non-null  int64
 7   Type_of_Loan_Personal Loan              56698 non-null  int64
 8   Type_of_Loan_Student Loan               56698 non-null  int64
 9   Age                                     56698 non-null  float64
 10  Occupation_Lawyer                       56698 non-null  int64
 11  Occupation_Developer                    56698 non-null  int64
 12  Occupation_Architect                    56698 non-null  int64
 13  Occupation_Manager                      56698 non-null  int64
 14  Occupation_Engineer                     56698 non-null  int64
 15  Occupation_Mechanic                     56698 non-null  int64
```

Final X_train

V. MODELING

LOGISTIC REGRESSION MODEL

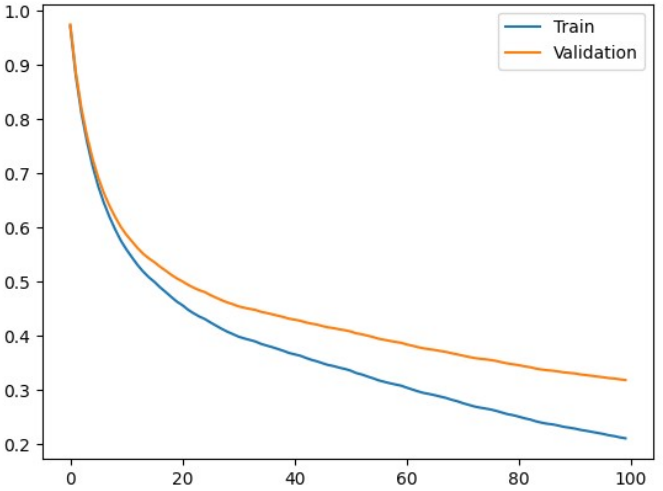
By plotting KDE plot between X features and y, => many non linear relationship
we will fit polynomial X features to Logistic Regression

```
print('Modeling by Logistic Regression: ')
penalty = 'l2'
multi_class = 'multinomial'
solver = 'lbfgs'
max_iter = 1000
model = LogisticRegression(random_state = 42, penalty = penalty, multi_class = multi_class, solver = solver, max_iter = max_iter)
model.fit(X_train_poly, y_train)
print('Evaluating  train', evaluate_metrics(yt=y_train, yp=model.predict(X_train_poly)))
print('Evaluating  val', evaluate_metrics(yt=y_val, yp=model.predict(X_val_poly)))
print('Evaluating  test', evaluate_metrics(yt=y_test, yp=model.predict(X_test_poly)))
```

```
Modeling by Logistic Regression:
Evaluating  train {'accuracy': 0.7131644855197714, 'recall': 0.7131644855197714, 'precision': 0.7133027240670498, 'f1score': 0.7128992354438679}
Evaluating  val {'accuracy': 0.7070194003527337, 'recall': 0.7070194003527337, 'precision': 0.7071370705700517, 'f1score': 0.7065047294892783}
Evaluating  test {'accuracy': 0.6987414639652351, 'recall': 0.6987414639652351, 'precision': 0.698842342897399, 'f1score': 0.6983619004241316}
```

V. MODELING

XGBOOST GRIDSEARCH



Mlogloss decrease after each epoch

```
param_grid = {'learning_rate': [0.2,0.4, 0.5],
              'n_estimators' : [70, 100, 150],
              'max_depth': [10, 15, 20, 30],
              }
early_stopping_rounds = 10
objective = 'multi:softmax'
eval_metric = 'mlogloss'
num_class = 3
eval_set = [(X_val, y_val)]
model = XGBClassifier(objective = objective, eval_metric = eval_metric, num_class = num_class, early_stopping_rounds =
early_stopping_rounds )
search = GridSearchCV(estimator=model, param_grid=param_grid, scoring= 'neg_log_loss', cv = 3)
search.fit(X_train, y_train, eval_set = eval_set)
```

```
{'learning_rate': 0.2, 'max_depth': 10, 'n_estimators': 100}
Evaluating  train {'accuracy': 0.9569649723094289, 'recall': 0.9569649723094289, 'precision': 0.9575399905669776, 'f1score': 0.9570095380348368}
Evaluating  val {'accuracy': 0.8986948853615521, 'recall': 0.8986948853615521, 'precision': 0.8990318599186896, 'f1score': 0.898692832174243}
Evaluating  test {'accuracy': 0.8004966420226876, 'recall': 0.8004966420226876, 'precision': 0.8008268161048924, 'f1score': 0.80043391514344}
```

V. MODELING

RANDOMFOREST GRIDSEARCH

```
param_grid = {'max_depth': [10,20,30]
              }
model = RandomForestClassifier(oob_score= True)
search = GridSearchCV(estimator=model, param_grid=param_grid, cv = 3)
search.fit(X_train, y_train)
print(search.best_score_)
print(search.best_params_)
```

```
{'max_depth': 30}
Evaluating  train {'accuracy': 0.9998412642421249, 'recall': 0.9998412642421249, 'precision': 0.9998412662369854, 'f1score': 0.9998412617430169}
Evaluating  val {'accuracy': 0.9295238095238095, 'recall': 0.9295238095238095, 'precision': 0.9296384081415531, 'f1score': 0.9295144990517926}
Evaluating  test {'accuracy': 0.8084542017043851, 'recall': 0.8084542017043851, 'precision': 0.8089636578137642, 'f1score': 0.8083539936471259}
```

VI. ASSESSMENT AND NEXT STEP

Best performance on validation and test set is RandomForest:

*- **RandomForest: 'accuracy': 0.806873977086743, 'recall': 0.806873977086743, 'precision': 0.8073300644667447, 'f1score': 0.8067633622128491***
- XGBoost: 'accuracy': 0.8004966420226876, 'recall': 0.8004966420226876, 'precision': 0.8008268161048924, 'f1score': 0.80043391514344
- LogisticRegression: 'accuracy': 0.6987414639652351, 'recall': 0.6987414639652351, 'precision': 0.698842342897399, 'f1score': 0.6983619004241316

Notice:
Although RandomForest have the best performance. Because of its complexity, the interpretable is lower than Logistic Regression
Still Logistic model will be higher interpretable and can be improved performance by using WOE to segment the continous variables to catch up more non-linear relationship

Decision:
*On this current stage of this project, **use RandomForest model to classify ('max_depth' = 30)**.*
Next step, we will improve performace of Logistic Regression.