Course Final Project Supervised Machine Learning: Classification

IBM Machine Learning Professional Certificate

By Omar Shariff Mailan

Contents

Dataset Description and Information

- Double A Housing Finance offers home loans and operates in cities, towns, and rural areas. When customers want to apply for a home loan, they first fill out an online application. The company then checks if the customer is eligible for a loan based on the information provided. This information includes details like gender, marital status, education, number of dependents, income, loan amount, and credit history.
- To make this process faster and easier, the company wants to automate the eligibility checks in real-time. They need help to identify which customer groups are eligible for loans so they can focus on reaching out to those people.
- This is a typical supervised classification task, where we aim to predict whether a loan will be approved or not. Below are the details of the dataset we will use.

Dataset Description and Information

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N)

Sample values:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Dataset Description and Information

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
```

Dataset Description and Information

- Finding for Null values

Loan_ID	0
Gender	13
Married	3
Dependents	15
Education	0
Self_Employed	32
ApplicantIncome	0
CoapplicantIncome	0
LoanAmount	22
Loan_Amount_Term	14
Credit_History	50
Property_Area	0
Loan_Status	0

Dataset Description and Information

- Preprocessing the dataset

```
# fill in the missing values for numerical terms using mean
df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].mean())
df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mean())
df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mean())
```

```
# fill in the missing values for categorical terms using mode
df['Gender'] = df["Gender"].fillna(df['Gender'].mode()[0])
df['Married'] = df["Married"].fillna(df['Married'].mode()[0])
df['Dependents'] = df["Dependents"].fillna(df['Dependents'].mode()[0])
df['Self_Employed'] = df["Self_Employed"].fillna(df['Self_Employed'].mode()[0])
```

Dataset Description and Information

- No more Null or missing values

```
1  df.isnull().sum()
```

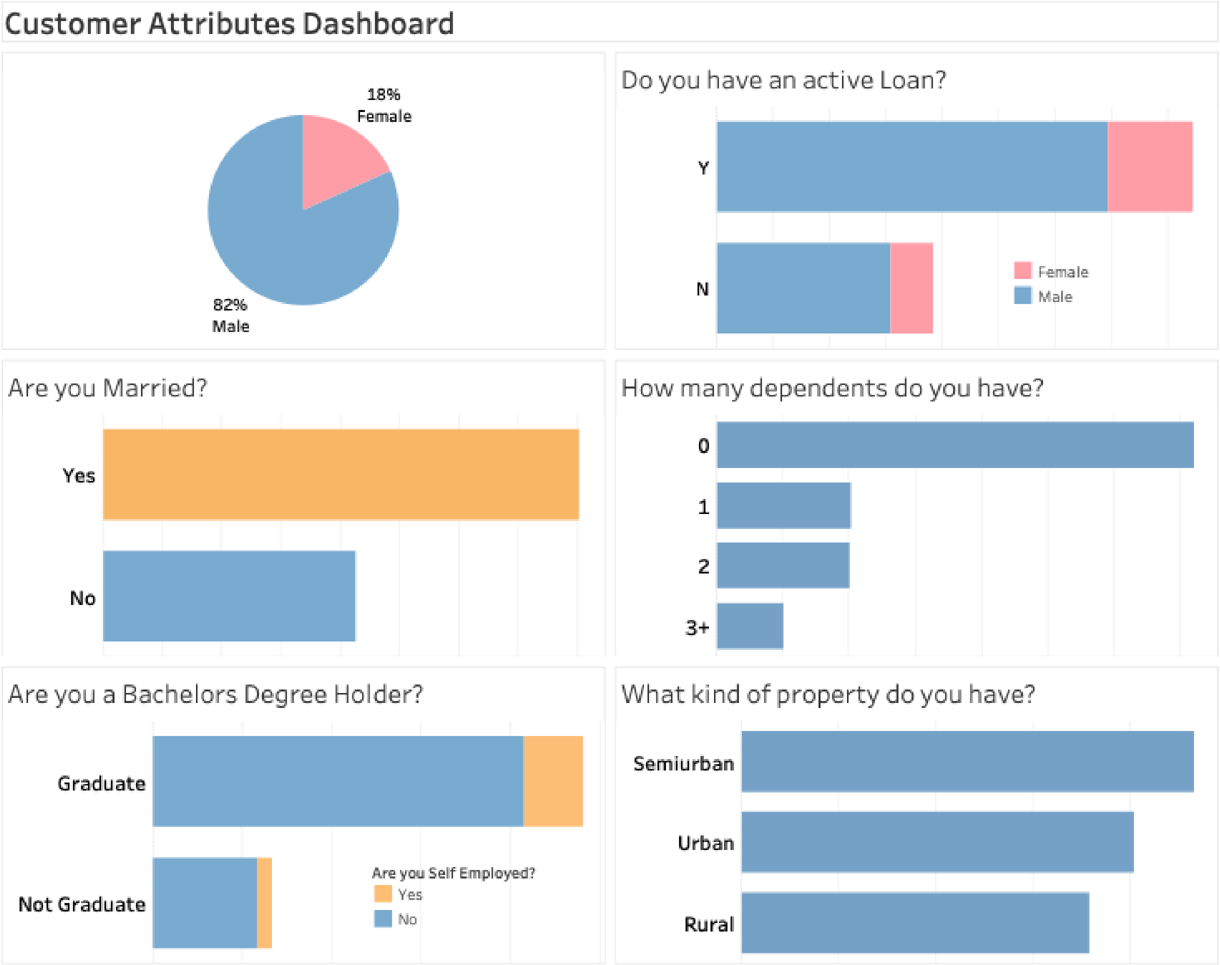
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

Main Objective of the Project

- To conduct a correlation matrix to determine significant variables or attributes that has a strong relationship and detect patterns or trends.
- To build different classification models to help identify faster on which customers are eligible for loans.
- To improve the applied classification models of at least 5%.

Exploratory Data Analysis

- Understanding customer attributes through Visualization Dashboard

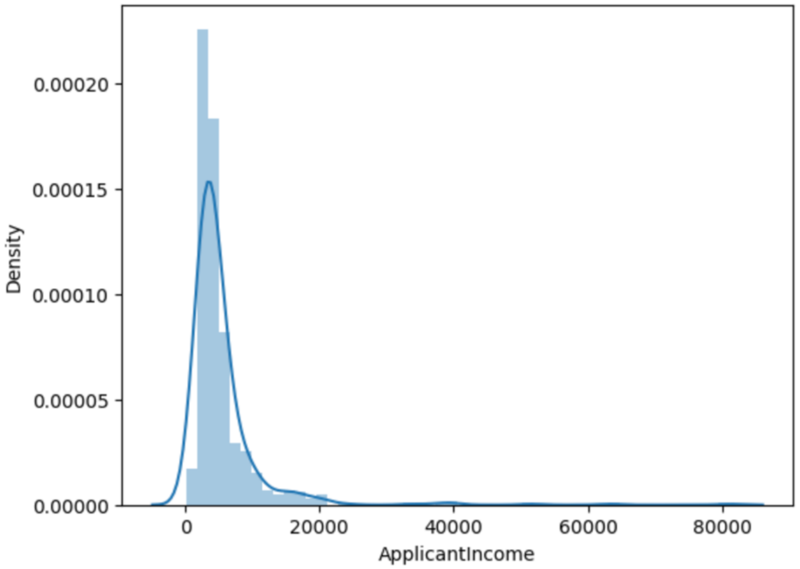


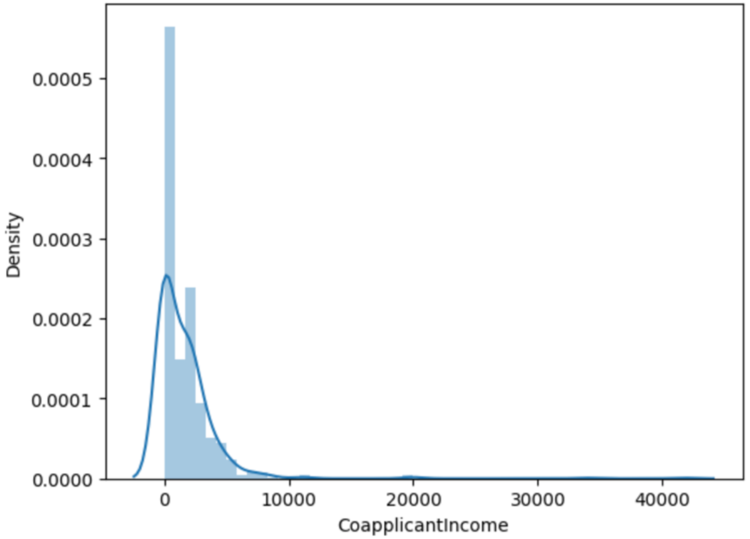
Categorical Attributes Results Showed:

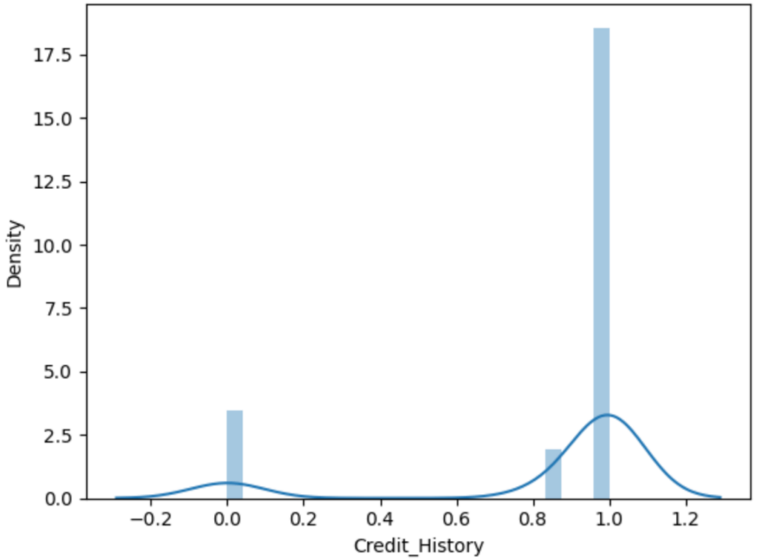
1. Male
2. Married
3. No dependents
4. A Bachelors Degree Holder
5. Private Employed
6. Has a Semi-Urban Property
7. Has an active loan

Exploratory Data Analysis

- Understanding numerical attributes of Applicants Income, Coapplicants Income and Credit History







Exploratory Data Analysis

Correlation Matrix

The Correlation Matrix was used to show the relationship between attributes that has a strong and significant factors in determining the eligibility for a loan.

Based on the results, it shows that Applicants Income, Loan Amount and its Total Income, has significant correlation and will be used for model testing

Exploratory Data Analysis

Removing unnecessary columns

```python
# drop unnecessary columns
cols = ['ApplicantIncome', 'CoapplicantIncome', "LoanAmount", "Loan_Amount_Term", "Total_Income", 'Loan_ID', 'CoapplicantIncomeLog']
df = df.drop(columns=cols, axis=1)
df.head()
```

✓ 0.0s

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomeLog	LoanAmountLog	Loan_Amount_Term_Log	Total_Income_Log
0	Male	No	0	Graduate	No	1.0	Urban	Y	8.674197	4.993232	5.888878	8.674197
1	Male	Yes	1	Graduate	No	1.0	Rural	N	8.430327	4.859812	5.888878	8.714732
2	Male	Yes	0	Graduate	Yes	1.0	Urban	Y	8.006701	4.204693	5.888878	8.006701
3	Male	Yes	0	Not Graduate	No	1.0	Urban	Y	7.857094	4.795791	5.888878	8.505525
4	Male	No	0	Graduate	No	1.0	Urban	Y	8.699681	4.955827	5.888878	8.699681

Final Label Encoding

```python
from sklearn.preprocessing import LabelEncoder
cols = ['Gender',"Married","Education",'Self_Employed',"Property_Area","Loan_Status","Dependents"]
le = LabelEncoder()
for col in cols:
    df[col] = le.fit_transform(df[col])
```

✓ 0.2s

```python
df.head()
```

✓ 0.0s

	Gender	Married	Dependents	Education	Self_Employed	Credit_History	Property_Area	Loan_Status	ApplicantIncomeLog	LoanAmountLog	Loan_Amount_Term_Log	Total_Income_Log
0	1	0	0	0	0	1.0	2	1	8.674197	4.993232	5.888878	8.674197
1	1	1	1	0	0	1.0	0	0	8.430327	4.859812	5.888878	8.714732
2	1	1	0	0	1	1.0	2	1	8.006701	4.204693	5.888878	8.006701
3	1	1	0	1	0	1.0	2	1	7.857094	4.795791	5.888878	8.505525
4	1	0	0	0	0	1.0	2	1	8.699681	4.955827	5.888878	8.699681

Machine Learning Model Application

Train-Test Split

```
# specify input and output attributes
X = df.drop(columns=['Loan_Status'], axis=1)
y = df['Loan_Status']
```

0.0s

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

0.3s

Machine Learning Model Application

Model Training: Using Logistic Regression

```python
# classify function
from sklearn.model_selection import cross_val_score
def classify(model, x, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
    model.fit(x_train, y_train)
    print("Accuracy is", model.score(x_test, y_test)*100)
    # cross validation - it is used for better validation of model
    # eg: cv-5, train-4, test-1
    score = cross_val_score(model, x, y, cv=5)
    print("Cross validation is",np.mean(score)*100)
```

1] ✓ 0.0s

```python
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
classify(model, X, y)
```

2] ✓ 0.2s

```
Accuracy is 77.27272727272727
Cross validation is 80.9462881514061
```

Machine Learning Model Application

Model Training: Using Random Forest and Extra Trees Classifier

```
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
model = RandomForestClassifier()
classify(model, X, y)
```

[44] ✓ 0.5s

```
Accuracy is 79.87012987012987
Cross validation is 78.82980141276823
```

```
model = ExtraTreesClassifier()
classify(model, X, y)
```

[45] ✓ 0.3s

```
Accuracy is 74.02597402597402
Cross validation is 76.87724910035986
```

Machine Learning Model Application

Additional Model Training and Improvements for increase accuracy

Based on previous study others already performed model algorithms, but in this case, I applied K-Nearest Neighbors (KNN) algorithm and Feature Engineering for an increase accuracy of at least 5%

K-Nearest Neighbors (KNN) classification

```
# Define features and target variable
X = df.drop(columns=['Loan_Status'])  # Features
y = df['Loan_Status']  # Target variable

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

```

✓ 0.0s

```
from sklearn.preprocessing import StandardScaler

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

```

✓ 0.0s

Machine Learning Model Application

Additional Model Training and Improvements for increase accuracy

K-Nearest Neighbors (KNN)

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize KNN with a chosen number of neighbors (e.g., 5)
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the model on the training data
knn.fit(X_train, y_train)

# Make predictions on the test data
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

✓ 0.0s

```
Accuracy: 0.7621621621621621
Confusion Matrix:
 [[ 30  35]
 [  9 111]]
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.46      0.58        65
           1       0.76      0.93      0.83       120

    accuracy                           0.76       185
   macro avg       0.76      0.69      0.71       185
weighted avg       0.76      0.76      0.74       185
```

Machine Learning Model Application

Additional Model Training and Improvements for increase accuracy

Hyperparameter optimization or Tuning KNN using GridSearchCV

```
from sklearn.model_selection import GridSearchCV

# Define parameter grid
param_grid = {'n_neighbors': range(1, 20)}

# Initialize grid search
grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best parameters and score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)

```

✓ 0.2s

```
Best Parameters: {'n_neighbors': 9}
Best Score: 0.8158139534883722
```

Machine Learning Model Application

Additional Model Training and Improvements for increase accuracy

Application of Feature Engineering in adding an Income-to-Loan ratio

```
import numpy as np

# Assuming df is your DataFrame with the specified columns

# Step 1: Exponentiate the log-transformed income and loan amount to get back their original values
df['ApplicantIncome'] = np.exp(df['ApplicantIncomeLog'])
df['LoanAmount'] = np.exp(df['LoanAmountLog'])

# Step 2: Calculate the Income-to-Loan ratio
df['Income_to_Loan_Ratio'] = df['ApplicantIncome'] / df['LoanAmount']

# Optional: Take the log of the ratio to maintain the logarithmic transformation
df['Income_to_Loan_Ratio_Log'] = np.log(df['Income_to_Loan_Ratio'])

# Step 3: Check if the new feature has been added correctly
print(df[['ApplicantIncomeLog', 'LoanAmountLog', 'ApplicantIncome', 'LoanAmount', 'Income_to_Loan_Ratio', 'Income_to_Loan_Ratio_Log']].head())

✓ 0.0s
```

```
# Define features (X) and target variable (y)
X = df.drop(columns=['Loan_Status'])  # Assuming Loan_Status is the target
y = df['Loan_Status']

# Optional: Check feature set after adding new feature
print(X.head())


5]  ✓ 0.0s
```

Machine Learning Model Application

Additional Model Training and Improvements for increase accuracy

Application of Feature Engineering in adding an Income-to-Loan ratio

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Optional: Check if scaling is applied correctly
print(X_train[:5])

```

✓ 0.0s

Machine Learning Model Application

Additional Model Training and Improvements for increase accuracy

Application of Feature Engineering in adding an Income-to-Loan ratio

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Initialize KNN with a chosen number of neighbors (e.g., 5)
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the model on the training data
knn.fit(X_train, y_train)

(variable) y_pred: ndarray t data
y_pred = knn.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

✓ 0.0s

```
Accuracy: 0.7675675675675676
Confusion Matrix:
 [[ 27  38]
 [  5 115]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.42      0.56        65
           1       0.75      0.96      0.84       120

    accuracy                           0.77       185
   macro avg       0.80      0.69      0.70       185
weighted avg       0.78      0.77      0.74       185
```

Analysis and Insights

- Profile shows that majority of the clients are Male, Married, with minimal dependents, a degree holder, owns a semi-urban property, privately employed and has an active loan. This can be used as a premium target for the right loan application.
- Numerical Attributes shows that an average income of at least 5,500, Coapplicants Income of at least 1,500 and a credit history of less than 0.84
- It also shows that Applicants Income, Loan Amount and its Total Income, has significant correlation impact in terms of loan approval and was used for model training and testing.

Analysis and Insights

- Model Application of K-Nearest Neighbors (KNN) model helps us determine which clients or LoanID will have a Loan Approval. Further fine tuning the KNN Model using GridSearchCV increases its accuracy from 0.76 to 0.82, about 11% increase improvement.
- Application of this model helps the company make faster decision making by helping their process faster and easier in determining their client groups that are eligible for loans.

Model Flaws and Next Steps

- One thing that needs to be addressed in this case was the percent accuracy that needs to be further improve. Accuracy of a model should be greater than 0.90 to have an accurate result with minimal errors. Kindly apply Gradient Boosting Machines (GBM), Support Vector Machines (SVM), AdaBoost to achieve higher accuracy.
- Another one, is biased testing in terms of homogeneity because it only assumes nearby points that are similar. We should consider the type of industry that it reflects, in this case it’s a financial business, we need to carefully assess other aspects such as credit history, property location and etc. This was not conducted due to limited time constraint.
- Lastly, would be the Data Imbalance checking, if the dataset has more approved loans than the rejected one, this could result to biases to majority class leading to false-positive.