COURSE CODE : INT-254

END TERM PROJECT REPORT

ON

LOAN APPROVAL PREDICTION

By

Nalli Shiva, Gatadi Varshith

Section: KM098

Roll Numbers: RKM098A19, RKM098A21

Department of Machine Learning

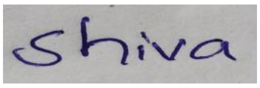
School of Computer Science and Engineering

Lovely Professional University, Jalandhar

11-2022

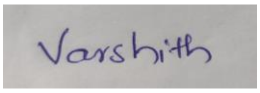
Student Declaration

This is to declare that this report has been written by N.Shiva, G.Varshith . No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. We aware that if any part of the report is found to be copied, we are shall take full responsibility for it.



Nalli Shiva

RKM098A19



G.Varshith

RKM098A21

Lovely Professional University

08-11-2022

TABLE OF CONTENTS

TITLE

BONAFIDE CERTIFICATE

Certified that this project report “ LOAN APPROVAL PREDICTION ” is the bonafide work of “ N.Shiva, G.Varshith ” who carried out the project work under my supervision.

<<Signature of the supervisior>>

Dr.Dhanpratap Singh

Introduction

In this project, we are going to solve the Loan Approval Prediction. This is a classification problem in which we need to classify whether the loan will be approved or not. Classification refers to a predictive modeling problem where a class label is predicted for a given example of input data. A few examples of classification problems are Spam Email detection, Cancer detection, Sentiment Analysis, etc.

In place of a class label, some might give us the prediction of a probability of class membership of a particular input and in such cases, the ROC curve can be a helpful indicator of how accurate one model is. There are mainly 4 different types of classification tasks that you might encounter in your day to day challenges. Generally, the different types of predictive models in machine learning are as follows :

- Binary classification
- Multi-Label Classification
- Multi-Class Classification
- Imbalanced Classification

Understanding the Problem Statement

Dream Housing Finance company deals in all kinds of home loans. They have a presence across all urban, semi-urban and rural areas. The customer first applies for a home loan and after that, the company validates the customer eligibility for the loan.

The company wants to automate the loan eligibility process (real-time) based on customer detail provided while filling out online application forms. These details are Gender, Marital Status, Education, number of Dependents, Income, Loan Amount, Credit History, and others.

To automate this process, they have provided a dataset to identify the customer segments that are eligible for loan amounts so that they can specifically target these customers.

As mentioned above this is a Binary Classification problem in which we need to predict our Target label which is “Loan Status”.

Loan status can have two values: Yes or NO.

Yes: if the loan is approved

NO: if the loan is not approved

So using the training dataset we will train our model and try to predict our target column that is “Loan Status” on the test dataset.

About the dataset

So train and test dataset would have the same columns except for the target column that is "Loan Status".

Train dataset:

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	(Target) Loan approved (Y/N)

Load Essential Python Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.model_selection import train_test_split
```

Load Training/ Test Dataset

```
train=pd.read_csv("/content/drive/MyDrive/train_ctrUa4K.csv")
test = pd.read_csv("/content/drive/MyDrive/test_lAUu6dG.csv")
ss = pd.read_csv("/content/drive/MyDrive/sample_submission_49d68Cx.csv")
```

Size of Train/Test Data

```
train.shape
```

```
(614, 13)
```

So we have 614 rows and 13 columns in our training dataset.

```
test.shape
```

```
(367, 12)
```

In test data, we have 367 rows and 12 columns because the target column is not included in the test data.

First look at the Dataset

```
train.head()
```

	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

Categorical Columns: Gender (Male/Female), Married (Yes/No), Number of dependents (Possible values:0,1,2,3+), Education (Graduate / Not Graduate), Self-Employed (No/Yes), credit history(Yes/No), Property Area (Rural/Semi-Urban/Urban) and Loan Status (Y/N)(i. e. Target variable)

Numerical Columns: Loan ID, Applicant Income, Co-applicant Income, Loan Amount, and Loan amount term

Data Preprocessing

Concatenating the train and test data for data preprocessing:

```
data = pd.concat([train,test])
```

dropping the unwanted column :

```
data.drop("Loan_ID",axis=1,inplace=True)
```

Identify missing values :

```
data.isnull().sum()
```

```
Gender                24
Married                3
Dependents            25
Education              0
Self_Employed         55
ApplicantIncome        0
CoapplicantIncome      0
LoanAmount            27
Loan_Amount_Term      20
Credit_History        79
Property_Area          0
Loan_Status          367
dtype: int64
```

Imputing the missing values:

```
for i in [data]:
    i["Gender"] = i["Gender"].fillna(data.Gender.dropna().mode()[0])
    i["Married"] = i["Married"].fillna(data.Married.dropna().mode()[0])
    i["Dependents"]=i["Dependents"].fillna(data.Dependents.dropna().mode()[0])
    i["Self_Employed"]=i["Self_Employed"].fillna(data.Self_Employed.dropna().mode()[0])
    i["Credit_History"]=i["Credit_History"].fillna(data.Credit_History.dropna().mode()[0])
```

- Fill null values with mode

Next, we will be using Iterative imputer for filling missing values of LoanAmount and Loan_Amount_Term

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

from sklearn.ensemble import RandomForestRegressor
data1 = data.loc[:, ['LoanAmount','Loan_Amount_Term']]

# Run imputer with a Random Forest estimator
imp = IterativeImputer(RandomForestRegressor(), max_iter=10, random_state=0)
data1 = pd.DataFrame(imp.fit_transform(data1), columns=data1.columns)
```

So now as we have imputed all the missing values we go on to mapping the categorical variables with the integers.

```
for i in [data]:
    i["Gender"] = i["Gender"].map({"Male":0,"Female":1}).astype(int)
    i["Married"] = i["Married"].map({'No':0,"Yes":1}).astype(int)
    i["Education"]=i["Education"].map({"Not Graduate":0,"Graduate":1}).astype(int)
    i["Self_Employed"]=i["Self_Employed"].map({'No':0,"Yes":1}).astype(int)
    i["Credit_History"]=i["Credit_History"].astype(int)
```

```
for i in [data]:
    i["Property_Area"] = i["Property_Area"].map({"Urban":0,"Rural":1,"Semiurban":2}).astype(int)

    i["Dependents"]=i["Dependents"].map({"0":0,"1":1,"2":2,"3+":3})
```

We map the values so that we can input the train data into the model as the model does not accept any string values.

Exploratory Data Analysis (EDA)

Splitting the data to new_train and new_test so that we can perform EDA.

```
new_train = data.iloc[:614]
new_test = data.iloc[614:]
```

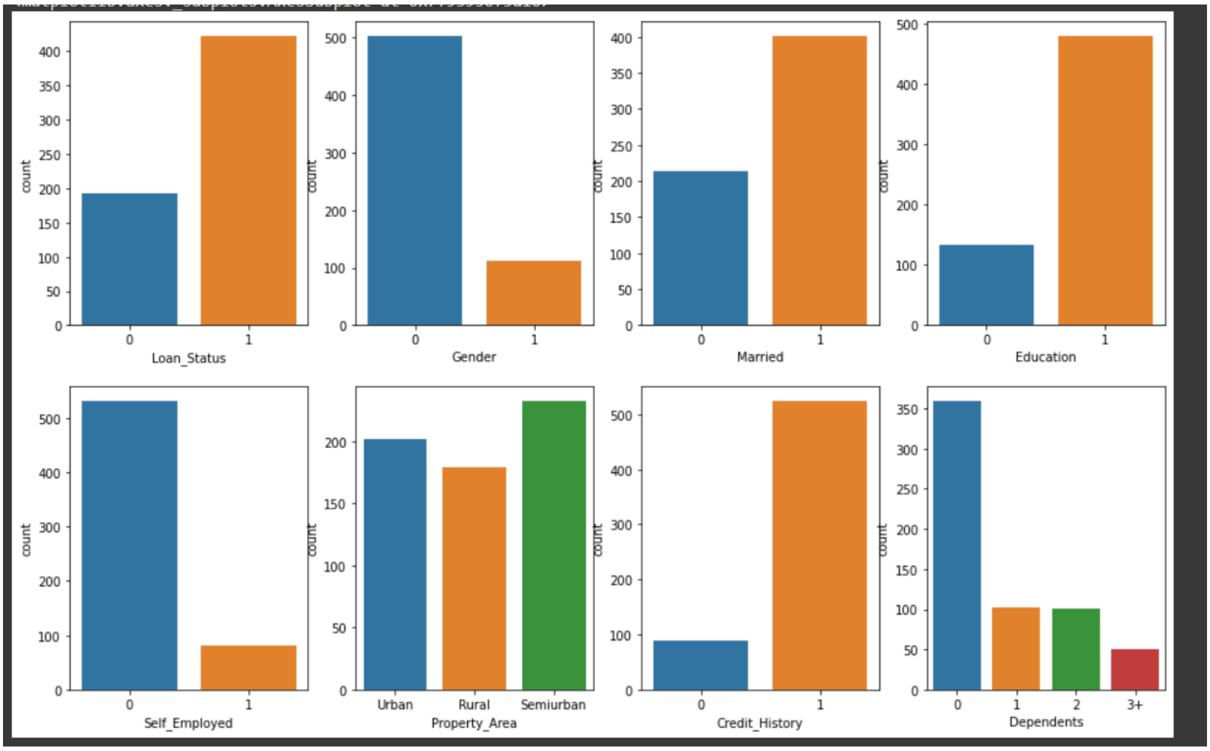
Mapping ‘N’ to 0 and ‘Y’ to 1

```
new_train["Loan_Status"] = new_train["Loan_Status"].map({'N':0,"Y":1}).astype(int)
```

Univariate Analysis:

```
fig,ax = plt.subplots(2,4,figsize=(16,10))
sns.countplot('Loan_Status',data=new_train,ax=ax[0][0])
sns.countplot('Gender',data=new_train,ax=ax[0][1])
sns.countplot('Married',data=new_train,ax=ax[0][2])
sns.countplot('Education',data=new_train,ax=ax[0][3])
sns.countplot('Self_Employed',data=new_train,ax=ax[1][0])
sns.countplot('Property_Area',data=new_train,ax=ax[1][1])
sns.countplot('Credit_History',data=new_train,ax=ax[1][2])
sns.countplot('Dependents',data=new_train,ax=ax[1][3])
```

Output:



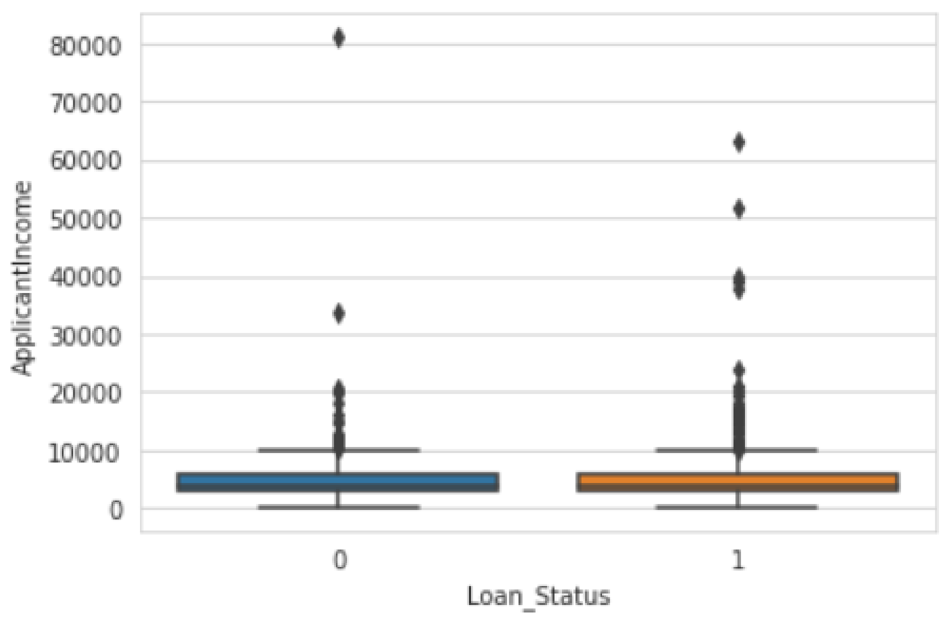
Univariate Analysis Observations

1. More Loans are approved Vs Rejected

2. Count of Male applicants is more than Female
3. Count of Married applicant is more than Non-married
4. Count of graduate is more than non-Graduate
5. Count of self-employed is less than that of Non-Self-employed
6. Maximum properties are located in Semiurban areas
7. Credit History is present for many applicants
8. The count of applicants with several dependents=0 is maximum.

Bivariate Analysis

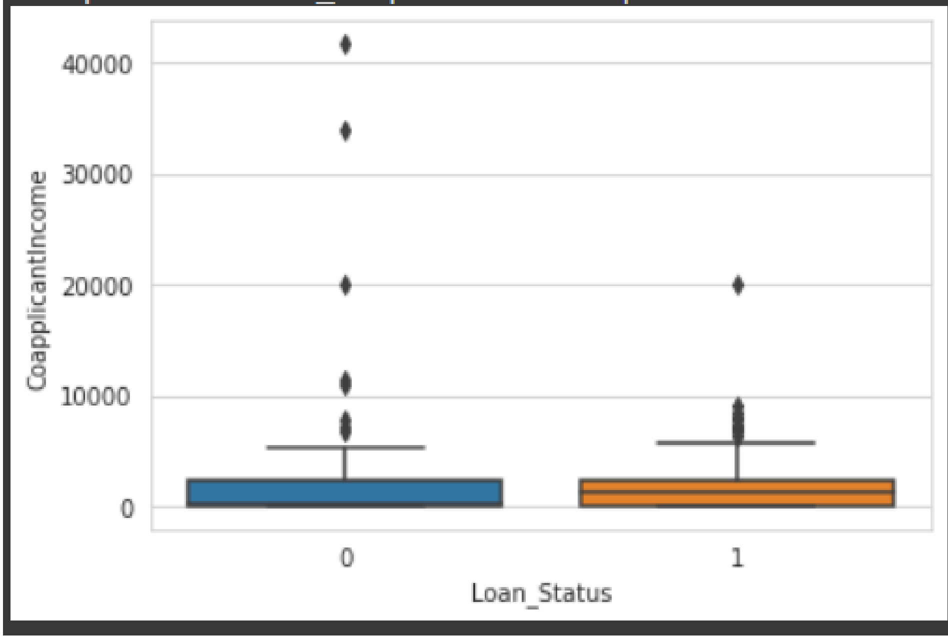
```
sns.boxplot(x='Loan_Status',y='ApplicantIncome',data=new_train)

<matplotlib.axes._subplots.AxesSubplot at 0x7f05d74e8550>
```



Mean ApplicantIncome of 0 and 1 are almost the same (o: no,1: Yes)

```
sns.boxplot(x='Loan_Status',y='CoapplicantIncome',data=new_train)
```

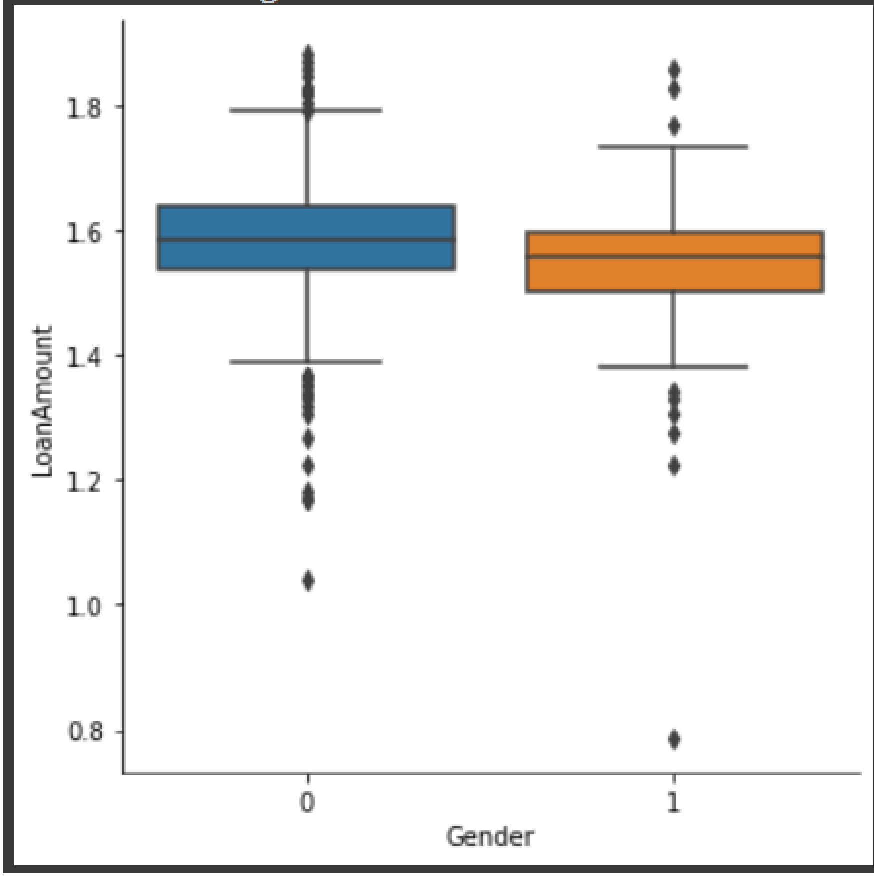
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05d8736210>
```



Mean Co- ApplicantIncome of 1 is slightly more than 0 (o: no,1 Yes)

```
sns.catplot(x='Gender',y='LoanAmount',data=new_train,kind='box'
```

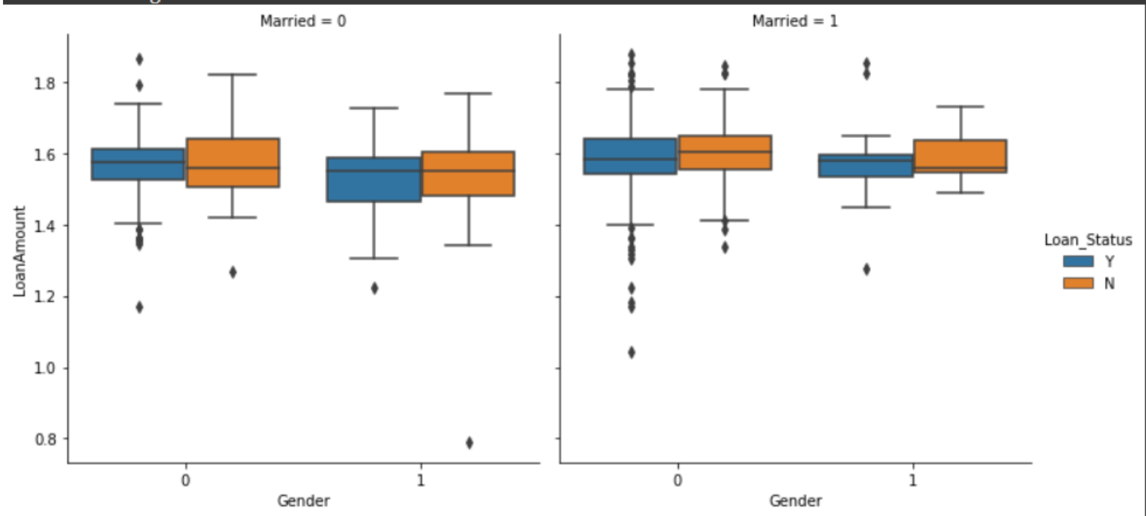
```
<seaborn.axisgrid.FacetGrid at 0x7f9395876dd0>
```



The mean value of Loan Amount applied by males (0) is slightly higher than Females(1).

```
sns.catplot(x='Gender',y='LoanAmount',data=data,kind='box',hue='Loan_Status', col='Married')
```

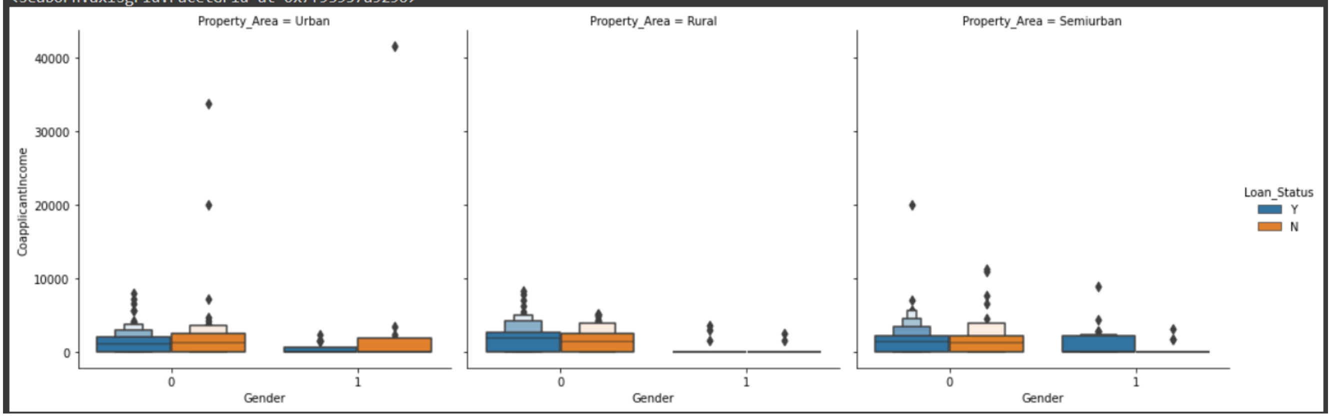
```
<seaborn.axisgrid.FacetGrid at 0x7f93950fcdd0>
```



If you are married then the loan amount requested is slightly higher than non-married

```
sns.catplot(x='Gender',y='CoapplicantIncome',data=data,kind='boxen',hue='Loan_Status', col='Property_Area')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f93957a9290>
```

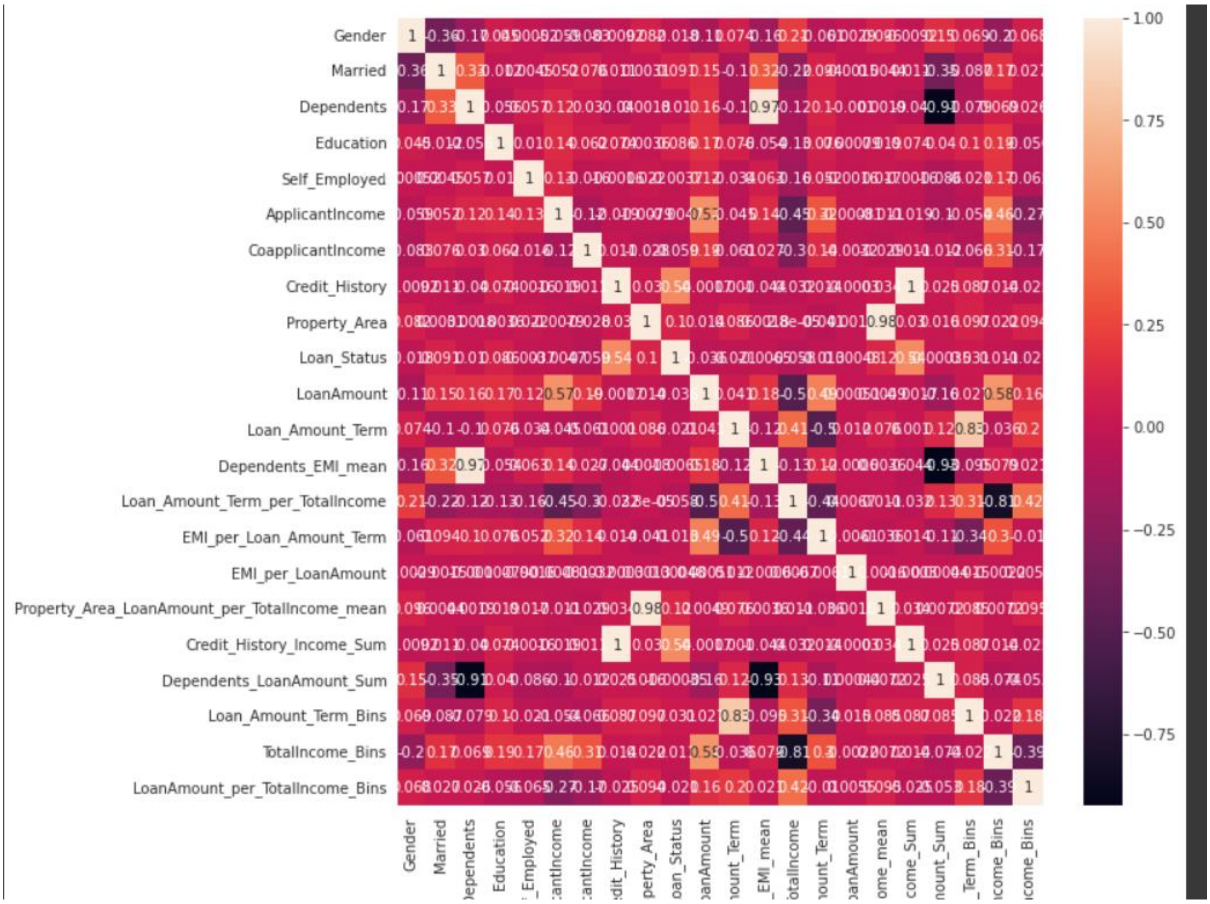


Male have higher Co-applicant income than females in all three property areas

Correlation matrix

```
plt.figure(figsize = (10,10))
correlation_matrix = new_train.corr()
sns.heatmap(correlation_matrix,annot=True)
plt.show
```

Output:



Feature Engineering

Total Income :

```
for i in [data]:
    i["TotalIncome"] = i["ApplicantIncome"]+i["CoapplicantIncome"]
```

EMI :

Lets assume that interest rate=10.0 # hence r = ((10/12)/100) = 0.00833

```
r = 0.00833
data['EMI']=data.apply(lambda x: (x['LoanAmount']*r*((1+r)**x['Loan_Amount_Term']))/((1+r)**((x['Loan_Amount_Term'])-1)),axis=1)
```

Additional Features :

```
data['Dependents_EMI_mean']=data.groupby(['Dependents'])['EMI'].transform('mean')

# LoanAmount_per_TotalIncome
data['LoanAmount_per_TotalIncome']=data['LoanAmount']/data['TotalIncome']

# Loan_Amount_Term_per_TotalIncome
data['Loan_Amount_Term_per_TotalIncome']=data['Loan_Amount_Term']/data['TotalIncome']

# EMI_per_Loan_Amount_Term
data['EMI_per_Loan_Amount_Term']=data['EMI']/data['Loan_Amount_Term']

# EMI_per_LoanAmount
data['EMI_per_LoanAmount']=data['EMI']/data['LoanAmount']

# Categorical variables wise mean of LoanAmount_per_TotalIncome
data['Property_Area_LoanAmount_per_TotalIncome_mean']=data.groupby(['Property_Area'])['LoanAmount_per_TotalIncome'].transform('mean')

# Credit_History wise sum of TotalIncome
data['Credit_History_Income_Sum']=data.groupby(['Credit_History'])['TotalIncome'].transform('sum')

# Dependents wise sum of LoanAmount
data['Dependents_LoanAmount_Sum']=data.groupby(['Dependents'])['LoanAmount'].transform('sum')
```

Bin Information :

```
from sklearn.preprocessing import KBinsDiscretizer

Loan_Amount_Term_discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
data['Loan_Amount_Term_Bins'] = Loan_Amount_Term_discretizer.fit_transform(data['Loan_Amount_Term'].values.reshape(-1,1)).astype(float)

TotalIncome_discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
data['TotalIncome_Bins'] = TotalIncome_discretizer.fit_transform(data['TotalIncome'].values.reshape(-1,1)).astype(float)

LoanAmount_per_TotalIncome_discretizer = KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='quantile')
data['LoanAmount_per_TotalIncome_Bins'] = LoanAmount_per_TotalIncome_discretizer.fit_transform(data['LoanAmount_per_TotalIncome'].values.reshape(-1,1)).astype(float)
```

Drop Unwanted Column :

```
data=data.drop(['EMI'],axis=1)
data=data.drop(['TotalIncome'],axis=1)
data=data.drop(['LoanAmount_per_TotalIncome'],axis=1)
```

Size after feature engineering :

```
new_train.shape
```

```
(614, 22)
```

We have added 8 new features

Building Machine Learning Model:

Creating X (input variables) and Y (Target Variable) from the new_train data.

```python
x = new_train.drop("Loan_Status",axis=1)

y = new_train["Loan_Status"]
```

Using train test split on the training data for validation

```python
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)

x_train.shape
```

```
(429, 21)
```

```python
x_test.shape
```

```
(185, 21)
```

We have a (70:30) split on the training data.

Using ML algorithm for training

We have used multiple algorithms for training purposes like Decision Tree, Random Forest, SVC, Logistic Regression, XGB Regressor, etc.

Among all the algorithms logistic regression performs best on the validation data with an accuracy score of **82.7%**.

```
log_clf = LogisticRegression()
from sklearn.model_selection import cross_val_score
cross_val_score(log_clf,x_train,y_train,scoring=make_scorer(accuracy_score),cv=3)
```

```
array([0.8041958 , 0.79020979, 0.7972028 ])
```

```
predo = log_clf.fit(x_train,y_train).predict(x_test)
accuracy_score(predo,y_test)
```

```
0.827027027027027
```

After getting an accuracy of 82.7% I tried fine-tuning it to improve my accuracy score using GridSearchCV.

```
from sklearn.model_selection import GridSearchCV
LRparam_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    'penalty': ['l1', 'l2'],
    'max_iter': list(range(100,800,100)),
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}
LR_search  = GridSearchCV(LogisticRegression(), LRparam_grid, refit = True, verbose = 3, cv=5)
LR_search.fit(x_train , y_train)
LR_search.best_params_
# summarize
print('Mean Accuracy: %.3f' % LR_search.best_score_)
print('Config: %s' % LR_search.best_params_)
```

The best parameters I got after fine-tuning were:

```
Config: {'C': 0.001, 'max_iter': 100, 'penalty': 'l1', 'solver': 'liblinear'}
```

After fine-tuning the logistic regression model the accuracy score improved from 82.7% to 83.24%.

```
l=LR_search.predict(x_test)
accuracy_score(l,y_test)
```

```
0.8324324324324325
```

Predicting on test data:

```
hj=LR_search.predict(new_test)
```

Prepare Sumbisson file:

```
test_df = pd.DataFrame(data = hj,columns=["Loan_Status"])
final_pred = pd.concat([ss['Loan_ID'],test_df],axis=1)
final_pred['Loan_Status']=final_pred['Loan_Status'].map({1:'Y',0:'N'})
final_pred.to_csv("final38.csv",index=False)
```

Conclusion

After the Final Submission of test data, my accuracy score was 78%.

Feature engineering helped me increase my accuracy.

Work Division

- Nalli Shiva
    - Project code
    - Data Set
    - Project Implementation
- Gatadi Varshith
    - Project code
    - Helps In Finding of DataSet and Report