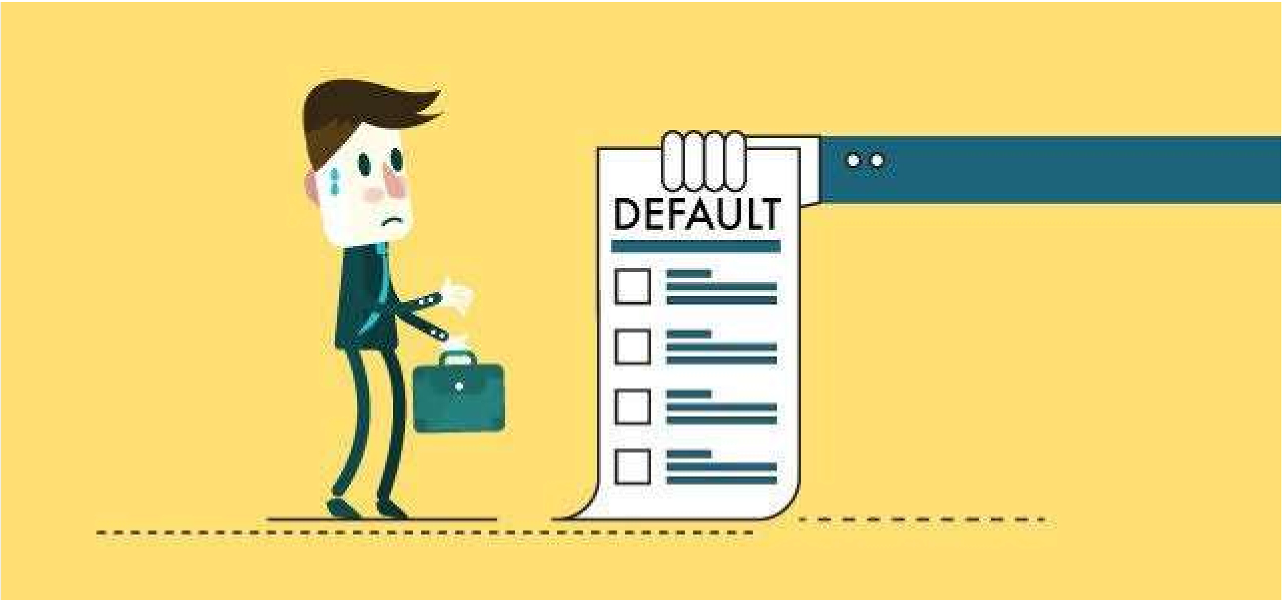
Project Report
on
Home Loan Allocation Predictor

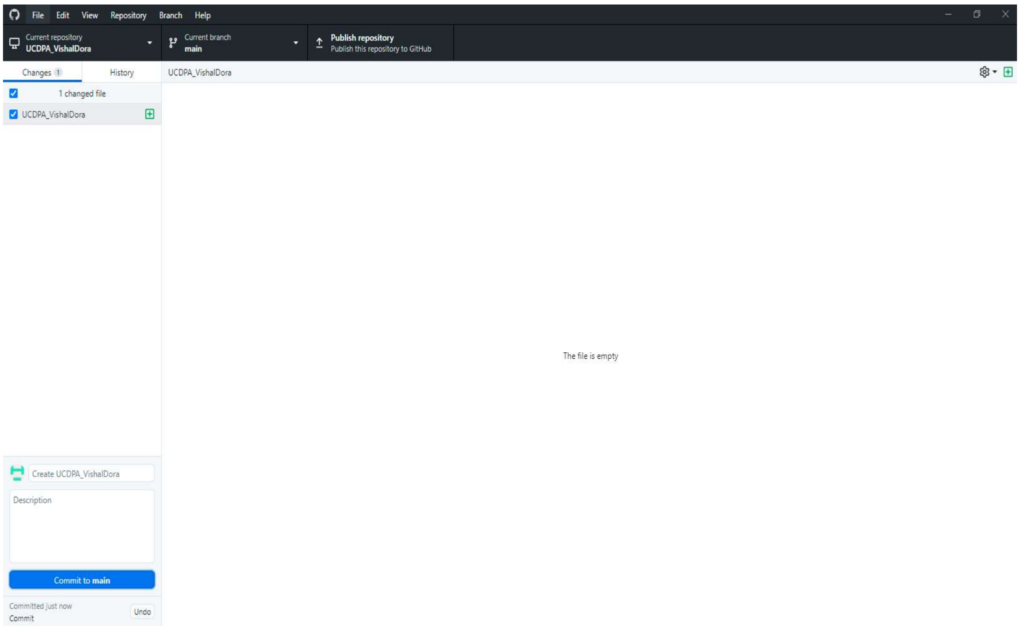


Name: Vishal Dora
Mobile Number : +91-9896373987
E-mail ID: vishal.dora@gmail.com
Batch : Fidelity International – Data Analytics Training (CIDAP 22-04-25)

Table of Contents

Project Report GitHub URL

https://github.com/VishalDora/UCDPA_VishalDora



Abstract

The case study pertains to banking and financial services industry wherein the task is to estimate the likelihood of a bank house loan default for a borrower based on the multiple factors like Applicant's Income, Co-applicant's Income, Loan Amount, Gender, Credit History, Property location and other such given metrics.

In a nutshell, The major aim of this project is to predict which of the customers will have their loan approved.

Introduction

I chose this project because I belong to finance domain as of now. In my past experiences I have worked with Banking sector organisations and I was keen to learn on which basis the loan is approved / rejected for any individual / organisations. When I started learning Python, it was from start of the training that I will use my learning to work on the project.

This project is helpful in all the banking sectors where loan is allocated to individuals/organizations. Loans can be of any shape/value such as home, personal, study etc.

I was keen to learn and implement supervised learning on basis of loan allocations history.

This model can be edited/modified for cards allocation.

Multiple Supervised models have been used to get the highest accuracy of results.

Dataset

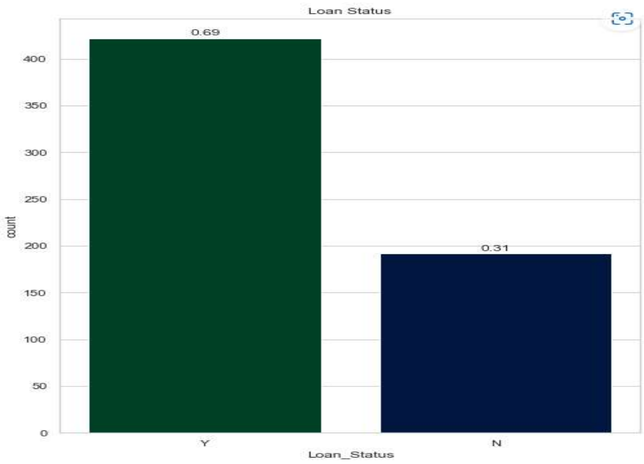
The dataset has been chosen because it has all the required columns such as Applicant's income, credit history, property location , Loan term, dependents etc.

The Training and testing dataset is enriched with information required for Machine learning. The training dataset is approximately of double size as test dataset.

The training dataset has 13 columns and testing dataset has 12 columns. The testing/training dataset has same columns.

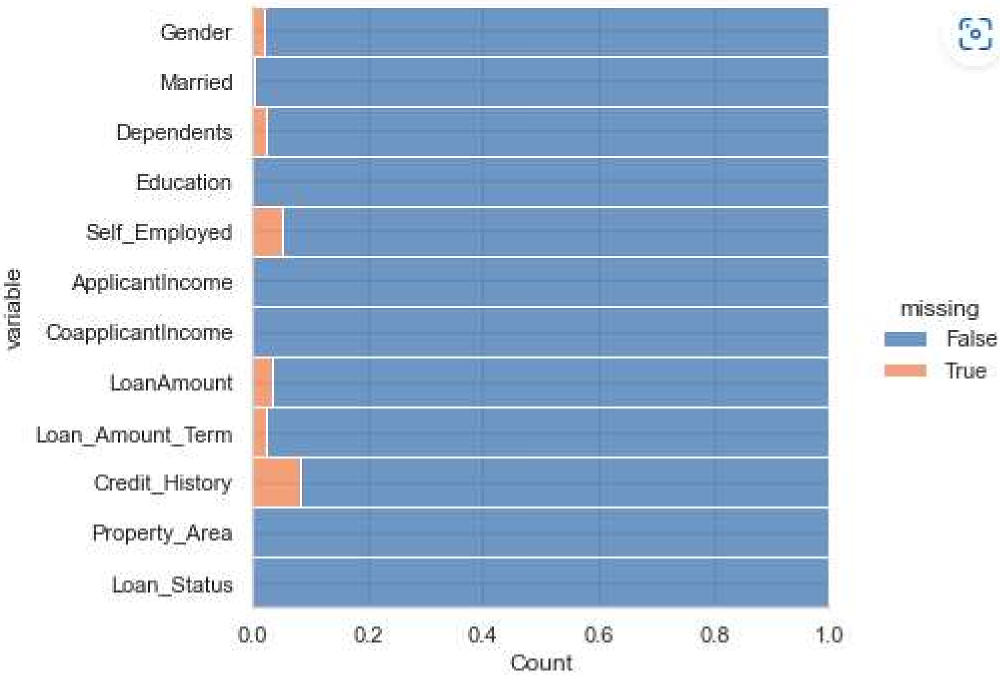
Implementation Process

From the given sample training data, the response variable is 'Default' and the other variables are taken as the predictor variables.
This a type of classification problem with an imbalanced dataset wherein the proportion of allottees to non-allottees is 31% which needs to be taken care off while making the model.



Imbalance in Sample Train Dataset

Upon analysis of the train dataset, it was found that the dataset had missing values for some of the features. The percentage of these missing values was found to be less than 3% for the dataset. Hence, we used the technique of **deletion of rows** to deal with the missing values. As all customer entities are independent of each other, so we haven't imputed the missing values by using other techniques.



The given data consists of categorical variables like Gender, Marital Status, dependents, education, employed status, Applicant's income, loan amount and loan_status. It is essential to encode these categorical features into numerical values i.e. replacing categorical data with numerical variables for easier interpretation. Better

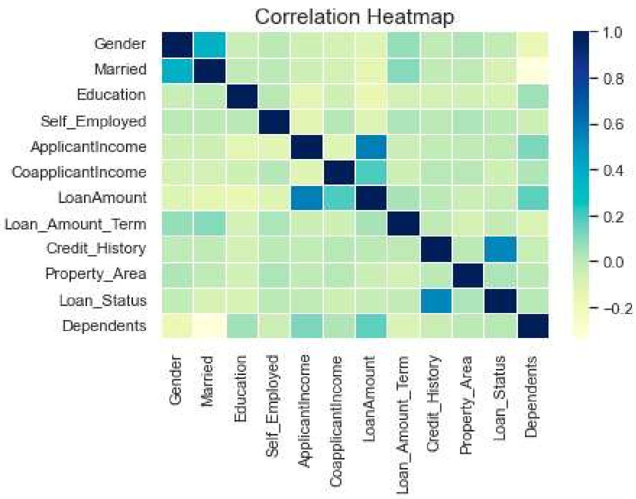
encoding of categorical data can mean better model performance. Label Encoding technique have been used for the same.
The following is the matrix for the encoding followed:

'Male': 1, 'Female': 2, 'Yes': 1, 'No': 2, 'Graduate': 1, 'Not Graduate': 2, 'Urban': 3, 'Semiurban': 2,'Rural': 1, 'Y': 1, 'N': 0, '3+': 3

```
#converting categorical values to numbers
to_numeric = {'Male': 1, 'Female': 2,
'Yes': 1, 'No': 2,
'Graduate': 1, 'Not Graduate': 2,
'Urban': 3, 'Semiurban': 2,'Rural': 1,
'Y': 1, 'N': 0,
'3+': 3}

# adding the new numeric values from the to_numeric variable to both datasets
training_dataframe = training_dataframe.applymap(lambda lable: to_numeric.get(lable) if lable in to_numeric else lable)
testing_dataframe = testing_dataframe.applymap(lambda lable: to_numeric.get(lable) if lable in to_numeric else lable)

# convertind the Dependents column
Dependents_ = pd.to_numeric(training_dataframe.Dependents)
Dependents__ = pd.to_numeric(testing_dataframe.Dependents)

# dropping the previous Dependents column
training_dataframe.drop(['Dependents'], axis = 1, inplace = True)
testing_dataframe.drop(['Dependents'], axis = 1, inplace = True)

# concatination of the new Dependents column with both datasets
training_dataframe = pd.concat([training_dataframe, Dependents_], axis = 1)
testing_dataframe = pd.concat([testing_dataframe, Dependents__], axis = 1)

# checking the our manipulated dataset for validation
print(f"training dataset (row, col): {training_dataframe.shape}\n\ntesting dataset (row, col): {testing_dataframe.shape}\n")
print(training_dataframe.info(), "\n\n", testing_dataframe.info())
```

Training set will be used to fit the model, and Test set will be used to evaluate the best model to get an estimation of generalization error.
And then we will use our model to make the prediction of new borrowers whether they will default on the home loan will be allocated or not.

Finding the correlation between the categorical values is as below:



As none of the selected independent variables are highly correlated to each other, so we can pick up all the variables to train our models.

After that, I have split the given sample data into training set (65%) and test set (35%).

```
y = training_dataframe['Loan_Status']
X = training_dataframe.drop('Loan_Status', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.35, random_state = 0)
```

Model Algorithm Selection

Model- Since this is a classification problem, I started by building the decision tree model on the sample train dataset. The decision tree is an interpretable model because it makes classifications much like we do: I ask a sequence of queries about the available data until we arrive at a decision. Along with that, to avoid the over fitting of the decision tree model, i built the random forest model, xgboost and logistic Regression as well.

Decision Tree Model: A decision tree is a non-parametric supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes.

```
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)

y_predict = DT.predict(X_test)

# Prediction Summary by species
print(classification_report(y_test, y_predict))

# Accuracy score
DT_SC = accuracy_score(y_predict,y_test)
print(f"{round(DT_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.45      0.53      0.49        51
           1       0.81      0.75      0.78       134

    accuracy                           0.69       185
   macro avg       0.63      0.64      0.63       185
weighted avg       0.71      0.69      0.70       185

69.19% Accurate
```

```
Decision_Tree=pd.DataFrame({'y_test':y_test,'prediction':y_predict})
Decision_Tree.to_csv("Data/Dection Tree.csv")
```

Logistic Regression model : In statistics, the (binary) logistic model (or logit model) is a statistical model that models the probability of one event (out of two alternatives) taking place by having the log-odds (the logarithm of the odds) for the event be a linear combination of one or more independent variables ("predictors"). In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (the coefficients in the linear combination).

```
LR = LogisticRegression()
LR.fit(X_train, y_train)

y_predict = LR.predict(X_test)

# Prediction Summary by species
print(classification_report(y_test, y_predict))

# Accuracy score
LR_SC = accuracy_score(y_predict,y_test)
print(f"{round(accuracy_score(y_predict,y_test)*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.92      0.43      0.59        51
           1       0.82      0.99      0.89       134

    accuracy                           0.83       185
   macro avg       0.87      0.71      0.74       185
weighted avg       0.85      0.83      0.81       185

83.24% Accurate
```

```
Logistic_Regression=pd.DataFrame({'y_test':y_test,'prediction':y_predict})
Logistic_Regression.to_csv("Data/Logistic Regression.csv")
```

Random Forest Classifier Model: Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

```
RF = RandomForestClassifier()
RF.fit(X_train, y_train)

y_predict = RF.predict(X_test)

# Prediction Summary by species
print(classification_report(y_test, y_predict))

# Accuracy score
RF_SC = accuracy_score(y_predict,y_test)
print(f"{round(RF_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.72      0.41      0.53        51
           1       0.81      0.94      0.87       134

    accuracy                           0.79       185
   macro avg       0.77      0.68      0.70       185
weighted avg       0.78      0.79      0.77       185

79.46% Accurate
```

```
Random_Forest=pd.DataFrame({'y_test':y_test,'prediction':y_predict})
Random_Forest.to_csv("Data/Random Forest.csv")
```

XG Boost Classifier Model: XGBoost is driving force behind the algorithms that win massive ML competitions. Its speed and performance are unparalleled and it consistently outperforms any other algorithms aimed at supervised learning tasks.

```
XGB = XGBClassifier()
XGB.fit(X_train, y_train)

y_predict = XGB.predict(X_test)

#  Prediction Summary by species
print(classification_report(y_test, y_predict))

# Accuracy score
XGB_SC = accuracy_score(y_predict,y_test)
print(f"{round(XGB_SC*100,2)}% Accurate")
```

```
              precision    recall  f1-score   support

           0       0.65      0.55      0.60        51
           1       0.84      0.89      0.86       134

    accuracy                           0.79       185
   macro avg       0.74      0.72      0.73       185
weighted avg       0.79      0.79      0.79       185

79.46% Accurate
```

Final model selection- So finally, I decided to use Logistic Regression model as our predictive model to evaluate the allocation decision for every new loan borrower based on the above diagnosis of the models.

Results

The analysis was run on 4 models to get the maximum accuracy for the test dataset.

The best results were received from Logistic regression model resulting in 83% accuracy. The most suitable and accurate decision is provided and can be used for allocation on home loan to the borrowers.

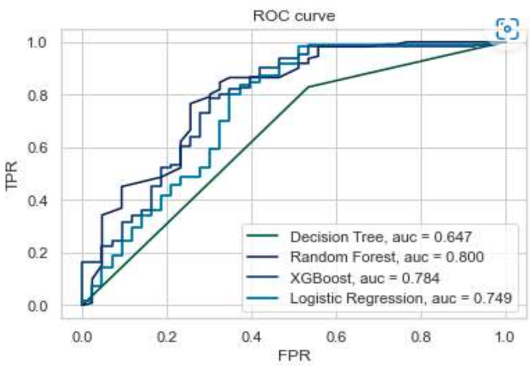
The results have been stored in csv files which can be used to see the allocation results.

```python
score = [DT_SC,RF_SC,XGB_SC,LR_SC]
Models = pd.DataFrame({
    'Model': ["Decision Tree","Random Forest","XGBoost", "Logistic Regression"],
    'Score': score})
Models.sort_values(by='Score', ascending=False)
```

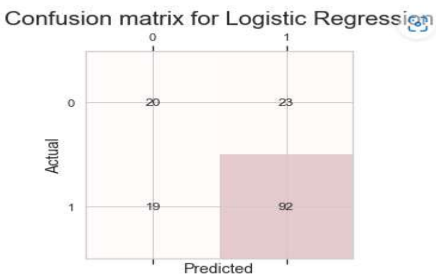
	Model	Score
3	Logistic Regression	0.832432
2	XGBoost	0.794595
1	Random Forest	0.778378
0	Decision Tree	0.702703

Insights

ROC curve



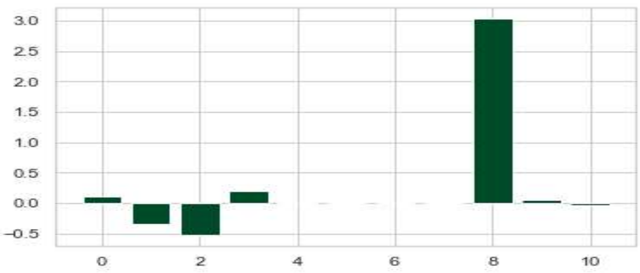
Confusion matrix



Feature score in Logistic Regression

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

```
Feature: 0, Score: 0.11882
Feature: 1, Score: -0.33638
Feature: 2, Score: -0.52876
Feature: 3, Score: 0.20101
Feature: 4, Score: 0.00001
Feature: 5, Score: -0.00007
Feature: 6, Score: -0.00120
Feature: 7, Score: -0.00341
Feature: 8, Score: 3.04162
Feature: 9, Score: 0.05294
Feature: 10, Score: -0.02582
```



1. As we look at the ROC curve chart, Logistic Regression has maximum area under curve and is most accurate for use on this dataset.
2. XG boost is the 2nd best model that can be used for this dataset.
3. The confusion matrix helps us understand that in our final selected model, False positive cases are not that much high i.e. ~19 , which is a good sign.
4. Based on feature scoring of logistic model, if any individual has good credit score, he/she can be given loan easily without much risk
5. If any individual is not graduated, he/she can be considered as highly risky person to give loan to.