Introduction to the Notebook

Welcome to this notebook on Loan Prediction Practice Problem. Below is a brief introduction of this notebook to get you acquainted with what you will be learning.

Objective of the Notebook

linkcode

This notebook is designed for people who want to solve binary classification problems using Python. By the end of this notebook, you will have the necessary skills and techniques required to solve such problems.

Expectation from the Notebook

This notebook is divided into the below sections:

1.Introduction to the problem.

2.Exploratory Data Analysis (EDA) and PreProcessing.

3.Feature engineering and Model building.

Table of Contents

Let’s look at the steps that we will follow in this notebook.

1.Problem Statement

2.Hypothesis Generation

3.Loading the data

4.Understanding the data

5.Exploratory Data Analysis (EDA)

i)Univariate Analysis

ii)Bivariate Analysis

6.Missing value and outlier treatment

7.Feature Engineering

8.Model Building :

i)Logistic Regression

ii) Decision Tree

iii) Random Forest

iv) Random Forest with Grid Search

v)XGBClassifier

Problem Statement

Understanding the problem statement is the first and foremost step. This would help you give an intuition of what you will face ahead of time. Let us see the problem statement -

Dream Housing Finance company deals in all home loans. They have presence across all urban, semi urban and rural areas. Customer first apply for home loan after that company validates the customer eligibility for loan. Company wants to automate the loan eligibility process (real time) based on customer detail provided while filling online application form. These details are Gender, Marital Status, Education, Number of Dependents, Income, Loan Amount, Credit History and others. To automate this process, they have given a problem to identify the customers segments, those are eligible for loan amount so that they can specifically target these customers.

This is a standard supervised classification task.A classification problem where we have to predict whether a loan would be approved or not. In a classification problem, we have to predict discrete values based on a given set of independent variable(s).Classification can be of two types:

Supervised: The labels are included in the training data and the goal is to train a model to learn to predict the labels from the features.

Binary Classification : In this classification we have to predict either of the two given classes. For example: classifying the gender as male or female, predicting the result as win or loss, etc.

Multiclass Classification : Here we have to classify the data into three or more classes. For example: classifying a movie's genre as comedy, action or romantic, classify fruits as oranges, apples, or pears, etc.

Loan prediction is a very common real-life problem that each retail bank faces atleast once in its lifetime. If done correctly, it can save a lot of man hours at the end of a retail bank.

Although this notebook is specifically built to give you a walkthrough of Loan Prediction problem, you can always refer the content to get a comprehensive overview to solve a classification problem.

Hypothesis Generation

After looking at the problem statement, we will now move into hypothesis generation. It is the process of listing out all the possible factors that can affect the outcome.

What is hypothesis generation?

This is a very important stage in any data science/machine learning pipeline. It involves understanding the problem in detail by brainstorming as many factors as possible which can impact the outcome. It is done by understanding the problem statement thoroughly and before looking at the data.

Below are some of the factors which I think can affect the Loan Approval (dependent variable for this loan prediction problem):

Salary: Applicants with high income should have more chances of loan approval.

Previous history: Applicants who have repayed their previous debts should have higher chances of loan approval.

Loan amount: Loan approval should also depend on the loan amount. If the loan amount is less, chances of loan approval should be high.

Loan term: Loan for less time period and less amount should have higher chances of approval.

EMI: Lesser the amount to be paid monthly to repay the loan, higher the chances of loan approval.

These are some of the factors which i think can affect the target variable, you can come up with many more factors

About The Data:

Given below is the description for each variable.

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)

Import required packages

In [1]:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")

Data

For this practice problem, we have been given two CSV files: train and test.

Train file will be used for training the model, i.e. our model will learn from this file. It contains all the independent variables and the target variable.

Test file contains all the independent variables, but not the target variable. We will apply the model to predict the target variable for the test data.

Read Train and Test Data

In [2]:

train = pd.read\_csv("../input/train.csv")

test = pd.read\_csv("../input/test.csv")

Copy of original data

Let’s make a copy of train and test data so that even if we have to make any changes in these datasets we would not lose the original datasets.

In [3]:

train\_original = train.copy()

test\_original = test.copy()

Understanding the Data

In this section, we will look at the structure of the train and test datasets. Firstly, we will check the features present in our data and then we will look at their data types.

In [4]:

train.columns

Out[4]:

Index(['Loan\_ID', 'Gender', 'Married', 'Dependents', 'Education',

'Self\_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',

'Loan\_Amount\_Term', 'Credit\_History', 'Property\_Area', 'Loan\_Status'],

dtype='object')

We have 12 independent variables and 1 target variable, i.e. Loan\_Status in the train dataset. Let’s also have a look at the columns of test dataset.

In [5]:

test.columns

Out[5]:

Index(['Loan\_ID', 'Gender', 'Married', 'Dependents', 'Education',

'Self\_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',

'Loan\_Amount\_Term', 'Credit\_History', 'Property\_Area'],

dtype='object')

We have similar features in the test dataset as the train dataset except the Loan\_Status. We will predict the Loan\_Status using the model that we will build using the train data.

Print the data types

In [6]:

train.dtypes

Out[6]:

Loan\_ID object

Gender object

Married object

Dependents object

Education object

Self\_Employed object

ApplicantIncome int64

CoapplicantIncome float64

LoanAmount float64

Loan\_Amount\_Term float64

Credit\_History float64

Property\_Area object

Loan\_Status object

dtype: object

We can see there are three format of data types:

object: Object format means variables are categorical. Categorical variables in our dataset are: Loan\_ID, Gender, Married, Dependents, Education, Self\_Employed, Property\_Area, Loan\_Status.

int64: It represents the integer variables. ApplicantIncome is of this format.

float64: It represents the variable which have some decimal values involved. They are also numerical variables. Numerical variables in our dataset are: CoapplicantIncome, LoanAmount, Loan\_Amount\_Term, and Credit\_History.

Shape of the dataset

In [7]:

print('Training data shape: ', train.shape)

train.head()

Training data shape: (614, 13)

Out[7]:

|  | 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 |

In [8]:

print('Test data shape: ', test.shape)

test.head()

Test data shape: (367, 12)

Out[8]:

|  | Loan\_ID | Gender | Married | Dependents | Education | Self\_Employed | ApplicantIncome | CoapplicantIncome | LoanAmount | Loan\_Amount\_Term | Credit\_History | Property\_Area |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | LP001015 | Male | Yes | 0 | Graduate | No | 5720 | 0 | 110.0 | 360.0 | 1.0 | Urban |
| 1 | LP001022 | Male | Yes | 1 | Graduate | No | 3076 | 1500 | 126.0 | 360.0 | 1.0 | Urban |
| 2 | LP001031 | Male | Yes | 2 | Graduate | No | 5000 | 1800 | 208.0 | 360.0 | 1.0 | Urban |
| 3 | LP001035 | Male | Yes | 2 | Graduate | No | 2340 | 2546 | 100.0 | 360.0 | NaN | Urban |
| 4 | LP001051 | Male | No | 0 | Not Graduate | No | 3276 | 0 | 78.0 | 360.0 | 1.0 | Urban |

We have 614 rows and 13 columns in the train dataset and 367 rows and 12 columns in test dataset.

Univariate Analysis

In this section, we will do univariate analysis. It is the simplest form of analyzing data where we examine each variable individually.

For categorical features we can use frequency table or bar plots which will calculate the number of each category in a particular variable.

For numerical features, probability density plots can be used to look at the distribution of the variable.

Target Variable

We will first look at the target variable, i.e., Loan\_Status.

As it is a categorical variable, let us look at its frequency table, percentage distribution and bar plot.

Frequency table of a variable will give us the count of each category in that variable.

In [9]:

*#train["Loan\_Status"].size*

train["Loan\_Status"].count()

Out[9]:

614

Size of our target variable is : 614

In [10]:

train["Loan\_Status"].value\_counts()

Out[10]:

Y 422

N 192

Name: Loan\_Status, dtype: int64

Among 614 Loan\_Status : Accepted : 422 Rejected : 192

In [11]:

*# Normalize can be set to True to print proportions instead of number*

train["Loan\_Status"].value\_counts(normalize=True)\*100

Out[11]:

Y 68.729642

N 31.270358

Name: Loan\_Status, dtype: float64

In [12]:

train["Loan\_Status"].value\_counts(normalize=True).plot.bar(title = 'Loan\_Status')

Out[12]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa342783160>

Logo, bar chart

Description automatically generated with medium confidence

The loan of 422(around 69%) people out of 614 was approved.

Now lets visualize each variable separately. Different types of variables are Categorical, ordinal and numerical.

Categorical features: These features have categories (Gender, Married, Self\_Employed, Credit\_History, Loan\_Status)

Independent Variable (Categorical)

Analysis on "Gender" variable :

In [13]:

train["Gender"].count()

Out[13]:

601

Size of our "Gender" variable is : 614

In [14]:

train["Gender"].value\_counts()

Out[14]:

Male 489

Female 112

Name: Gender, dtype: int64

Among 601 person: Male : 489 Female : 112

In [15]:

train['Gender'].value\_counts(normalize=True)\*100

Out[15]:

Male 81.364393

Female 18.635607

Name: Gender, dtype: float64

In [16]:

train['Gender'].value\_counts(normalize=True).plot.bar(title= 'Gender')

Out[16]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33cea8208>

Logo

Description automatically generated with medium confidence

In our train dataset the "Gender" variable contain Male : 81% Female: 19%

Analysis on "Married" variable :

In [17]:

train["Married"].count()

Out[17]:

611

Size of our "Married" variable is : 611

In [18]:

train["Married"].value\_counts()

Out[18]:

Yes 398

No 213

Name: Married, dtype: int64

Total number of people : 611

Married : 398

Unmarried : 213

In [19]:

train['Married'].value\_counts(normalize=True)\*100

Out[19]:

Yes 65.139116

No 34.860884

Name: Married, dtype: float64

In [20]:

train['Married'].value\_counts(normalize=True).plot.bar(title= 'Married')

Out[20]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33ce8d128>

Logo

Description automatically generated

From the Grapch we see that :

Number of married people : 65%

Number of unmarried people : 35%

Analysis on "Self\_Employed" variable :

In [21]:

train["Self\_Employed"].count()

Out[21]:

582

Size of our "Self\_Employed" variable is : 582

In [22]:

train["Self\_Employed"].value\_counts()

Out[22]:

No 500

Yes 82

Name: Self\_Employed, dtype: int64

Total number of people : 582

Self\_Employed : 82

Not\_Self\_Employed : 500

In [23]:

train['Self\_Employed'].value\_counts(normalize=True)\*100

Out[23]:

No 85.910653

Yes 14.089347

Name: Self\_Employed, dtype: float64

In [24]:

train['Self\_Employed'].value\_counts(normalize=True).plot.bar(title='Self\_Employed')

Out[24]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33ce01198>

Icon

Description automatically generated with low confidence

Among 582 people only 14% are Self\_Employed and rest of the 86% are Not\_Self\_Employed

Analysis on "Credit\_History" variable :

In [25]:

train["Credit\_History"].count()

Out[25]:

564

Size of our "Credit\_History" variable is : 564

In [26]:

train["Credit\_History"].value\_counts()

Out[26]:

1.0 475

0.0 89

Name: Credit\_History, dtype: int64

Total number of debts : 564

Repaid Debts : 475

Not Repaid Debts : 89

In [27]:

train['Credit\_History'].value\_counts(normalize=True)\*100

Out[27]:

1.0 84.219858

0.0 15.780142

Name: Credit\_History, dtype: float64

In [28]:

train['Credit\_History'].value\_counts(normalize=True).plot.bar(title='Credit\_History')

Out[28]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33cdd4470>

Bar chart

Description automatically generated with medium confidence

Around 84% applicants have repaid their debts.

Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property\_Area)

Independent Variable (Ordinal)

Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property\_Area)

Analysis on "Dependents" variable :

In [29]:

train['Dependents'].count()

Out[29]:

599

Size of our "Dependents" variable is : 599

In [30]:

train["Dependents"].value\_counts()

Out[30]:

0 345

1 102

2 101

3+ 51

Name: Dependents, dtype: int64

Number of 0 Dependent : 345

Number of 1 Dependent : 102

Number of 2 Dependesnt : 101

Number of 3+ Dependent : 51

In [31]:

train['Dependents'].value\_counts(normalize=True)\*100

Out[31]:

0 57.595993

1 17.028381

2 16.861436

3+ 8.514190

Name: Dependents, dtype: float64

In [32]:

train['Dependents'].value\_counts(normalize=True).plot.bar(title="Dependents")

Out[32]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33cd317b8>

Chart, bar chart

Description automatically generated

58% people have 0 dependent

17% people have 1 dependent

17% people have 2 dependent

8% people have 3+ dependent

Analysis on "Education" variable :

In [33]:

train["Education"].count()

Out[33]:

614

Size of Education variable : 614

In [34]:

train["Education"].value\_counts()

Out[34]:

Graduate 480

Not Graduate 134

Name: Education, dtype: int64

People who are Graduated : 480

People who are not Graduated : 134

In [35]:

train["Education"].value\_counts(normalize=True)\*100

Out[35]:

Graduate 78.175896

Not Graduate 21.824104

Name: Education, dtype: float64

In [36]:

train["Education"].value\_counts(normalize=True).plot.bar(title = "Education")

Out[36]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33cd137b8>

Bar chart

Description automatically generated with low confidence

Total number of People : 614

78% are Graduated and 22% are not Graduated

Analysis on "Property\_Area" variable :

In [37]:

train["Property\_Area"].count()

Out[37]:

614

Size of "Property\_Area" variable : 614

In [38]:

train["Property\_Area"].value\_counts()

Out[38]:

Semiurban 233

Urban 202

Rural 179

Name: Property\_Area, dtype: int64

Total number of People : 614

People from Semiurban area : 233

People from Urban area : 202

People from Rural area : 179

In [39]:

train["Property\_Area"].value\_counts(normalize=True)\*100

Out[39]:

Semiurban 37.947883

Urban 32.899023

Rural 29.153094

Name: Property\_Area, dtype: float64

In [40]:

train["Property\_Area"].value\_counts(normalize=True).plot.bar(title="Property\_Area")

Out[40]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33cc79828>

Bar chart

Description automatically generated

38% people from Semiurban area

33% people from Urban area

29% people from Rural area

Independent Variable (Numerical)

Numerical features: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount, Loan\_Amount\_Term)

"ApplicantIncome" distribution :

In [41]:

plt.figure(1)

plt.subplot(121)

sns.distplot(train["ApplicantIncome"]);

plt.subplot(122)

train["ApplicantIncome"].plot.box(figsize=(16,5))

plt.show()

Icon

Description automatically generated

It can be inferred that most of the data in the distribution of applicant income is towards left which means it is not normally distributed. We will try to make it normal in later sections as algorithms works better if the data is normally distributed.

The boxplot confirms the presence of a lot of outliers/extreme values. This can be attributed to the income disparity in the society.

Part of this can be driven by the fact that we are looking at people with different education levels. Let us segregate them by Education:

In [42]:

train.boxplot(column='ApplicantIncome',by="Education" )

plt.suptitle(" ")

plt.show()

Chart

Description automatically generated

We can see that there are a higher number of graduates with very high incomes, which are appearing to be the outliers.

Let’s look at the "CoapplicantIncome" distribution:

In [43]:

plt.figure(1)

plt.subplot(121)

sns.distplot(train["CoapplicantIncome"]);

plt.subplot(122)

train["CoapplicantIncome"].plot.box(figsize=(16,5))

plt.show()

Icon

Description automatically generated with medium confidence

We see a similar distribution as that of the applicant income. Majority of coapplicant’s income ranges from 0 to 5000. We also see a lot of outliers in the coapplicant income and it is not normally distributed.

Let’s look at the distribution of "LoanAmount" variable :

In [44]:

plt.figure(1)

plt.subplot(121)

df=train.dropna()

sns.distplot(df['LoanAmount']);

plt.subplot(122)

train['LoanAmount'].plot.box(figsize=(16,5))

plt.show()

Icon

Description automatically generated with low confidence

We see a lot of outliers in this variable and the distribution is fairly normal. We will treat the outliers in later sections.

distribution of "LoanAmountTerm" variable :

In [45]:

plt.figure(1)

plt.subplot(121)

df = train.dropna()

sns.distplot(df["Loan\_Amount\_Term"]);

plt.subplot(122)

df["Loan\_Amount\_Term"].plot.box(figsize=(16,5))

plt.show()

A picture containing icon

Description automatically generated

We see a lot of outliers in this variable and the distribution is fairly normal. We will treat the outliers in later sections.

Bivariate Analysis

Lets recall some of the hypotheses that we generated earlier:

i)Applicants with high income should have more chances of loan approval.

ii)Applicants who have repaid their previous debts should have higher chances of loan approval.

iii)Loan approval should also depend on the loan amount. If the loan amount is less, chances of loan approval should be high.

iv)Lesser the amount to be paid monthly to repay the loan, higher the chances of loan approval.

Lets try to test the above mentioned hypotheses using bivariate analysis.

After looking at every variable individually in univariate analysis, we will now explore them again with respect to the target variable.

Categorical Independent Variable vs Target Variable

First of all we will find the relation between target variable and categorical independent variables. Let us look at the stacked bar plot now which will give us the proportion of approved and unapproved loans.

Relation between "Loan\_Status" and "Gender"

In [46]:

print(pd.crosstab(train["Gender"],train["Loan\_Status"]))

Gender = pd.crosstab(train["Gender"],train["Loan\_Status"])

Gender.div(Gender.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Gender")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

Gender

Female 37 75

Male 150 339

Chart, bar chart

Description automatically generated

Number of Female whose Loan was approed : 75

Number of Male whose Loan was approed : 339

Number of Female whose Loan was not approed : 37

Number of Male whose Loan was not approed : 150

Proportion of Male applicants is higher for the approved loans.

Relation between "Loan\_Status" and "Married"

In [47]:

print(pd.crosstab(train["Married"],train["Loan\_Status"]))

Married=pd.crosstab(train["Married"],train["Loan\_Status"])

Married.div(Married.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Married")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

Married

No 79 134

Yes 113 285

Chart, bar chart

Description automatically generated

Number of married people whose Loan was approed : 285

Number of married people whose Loan was not approed : 113

Number of unmarried people whose Loan was approed : 134

Number of unmarried people whose Loan was not approed : 79

Proportion of Married applicants is higher for the approved loans.

Relation between "Loan\_Status" and "Dependents"

In [48]:

print(pd.crosstab(train['Dependents'],train["Loan\_Status"]))

Dependents = pd.crosstab(train['Dependents'],train["Loan\_Status"])

Dependents.div(Dependents.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Dependents")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

Dependents

0 107 238

1 36 66

2 25 76

3+ 18 33

Chart

Description automatically generated

Number of dependents on the loan applicant : 0 and Loan was approed : 238

Number of dependents on the loan applicant : 0 and Loan was not approed : 107

Number of dependents on the loan applicant : 1 and Loan was approed : 66

Number of dependents on the loan applicant : 1 and Loan was not approed : 36

Number of dependents on the loan applicant : 2 and Loan was approed : 76

Number of dependents on the loan applicant : 2 and Loan was not approed : 25

Number of dependents on the loan applicant : 3+ and Loan was approed : 33

Number of dependents on the loan applicant : 3+ and Loan was not approed : 18

Distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan\_Status.

Relation between "Loan\_Status" and "Education"

In [49]:

print(pd.crosstab(train["Education"],train["Loan\_Status"]))

Education = pd.crosstab(train["Education"],train["Loan\_Status"])

Education.div(Education.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Education")

plt.ylabel("Percentage")

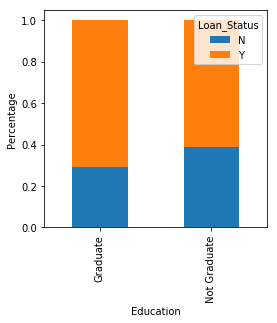
plt.show()

Loan\_Status N Y

Education

Graduate 140 340

Not Graduate 52 82



Number of people who are Graduate and Loan was approed : 340

Number of people who are Graduate and Loan was no approed : 140

Number of people who are Not Graduate and Loan was approed : 82

Number of people who are Not Graduate and Loan was not approed : 52

Proportion of Graduate applicants is higher for the approved loans.

Relation between "Loan\_Status" and "Self\_Employed"

In [50]:

print(pd.crosstab(train["Self\_Employed"],train["Loan\_Status"]))

SelfEmployed = pd.crosstab(train["Self\_Employed"],train["Loan\_Status"])

SelfEmployed.div(SelfEmployed.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Self\_Employed")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

Self\_Employed

No 157 343

Yes 26 56

Chart, bar chart

Description automatically generated

People who are Self\_Employed and Loan was approed : 56

People who are Self\_Employed and Loan was not approed : 26

People who are not Self\_Employed and Loan was approed : 343

People who are not Self\_Employed and Loan was not approed : 157

There is nothing significant we can infer from Self\_Employed vs Loan\_Status plot.

Relation between "Loan\_Status" and "Credit\_History"

In [51]:

print(pd.crosstab(train["Credit\_History"],train["Loan\_Status"]))

CreditHistory = pd.crosstab(train["Credit\_History"],train["Loan\_Status"])

CreditHistory.div(CreditHistory.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Credit\_History")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

Credit\_History

0.0 82 7

1.0 97 378

Chart, bar chart

Description automatically generated

People with credit history as 1 and loan was approved : 378

People with credit history as 1 and loan was not approved : 97

People with credit history as 0 and loan was approved : 7

People with credit history as 0 and loan was not approved : 82

It seems people with credit history as 1 are more likely to get their loans approved.

Relation between "Loan\_Status" and "Property\_Area"

In [52]:

print(pd.crosstab(train["Property\_Area"],train["Loan\_Status"]))

PropertyArea = pd.crosstab(train["Property\_Area"],train["Loan\_Status"])

PropertyArea.div(PropertyArea.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("Property\_Area")

plt.ylabel("Loan\_Status")

plt.show()

Loan\_Status N Y

Property\_Area

Rural 69 110

Semiurban 54 179

Urban 69 133

Chart, bar chart

Description automatically generated

People who are from Rural area and loan was approved : 110

People who are from Rural area and loan was not approved : 69

People who are from Semiurban area and loan was approved : 179

People who are from Semiurban area and loan was not approved : 54

People who are from Urban area and loan was approved : 133

People who are from Semiurban area and loan was not approved : 69

Proportion of loans getting approved in semiurban area is higher as compared to that in rural or urban areas.

Numerical Independent Variable vs Target Variable

Relation between "Loan\_Status" and "Income"

We will try to find the mean income of people for which the loan has been approved vs the mean income of people for which the loan has not been approved.

In [53]:

train.groupby("Loan\_Status")['ApplicantIncome'].mean().plot.bar()

Out[53]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33ca167f0>

Logo

Description automatically generated

Here the y-axis represents the mean applicant income. We don’t see any change in the mean income. So, let’s make bins for the applicant income variable based on the values in it and analyze the corresponding loan status for each bin.

In [54]:

bins=[0,2500,4000,6000,81000]

group=['Low','Average','High', 'Very high']

train['Income\_bin']=pd.cut(df['ApplicantIncome'],bins,labels=group)

In [55]:

print(pd.crosstab(train["Income\_bin"],train["Loan\_Status"]))

Income\_bin = pd.crosstab(train["Income\_bin"],train["Loan\_Status"])

Income\_bin.div(Income\_bin.sum(1).astype(float),axis=0).plot(kind="bar",stacked=True,figsize=(4,4))

plt.xlabel("ApplicantIncome")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

Income\_bin

Low 26 57

Average 51 123

High 32 79

Very high 39 73

Chart, bar chart

Description automatically generated

It can be inferred that Applicant income does not affect the chances of loan approval which contradicts our hypothesis in which we assumed that if the applicant income is high the chances of loan approval will also be high.

We will analyze the coapplicant income and loan amount variable in similar way.

In [56]:

bins=[0,1000,3000,42000]

group =['Low','Average','High']

train['CoapplicantIncome\_bin']=pd.cut(df["CoapplicantIncome"],bins,labels=group)

In [57]:

print(pd.crosstab(train["CoapplicantIncome\_bin"],train["Loan\_Status"]))

CoapplicantIncome\_Bin = pd.crosstab(train["CoapplicantIncome\_bin"],train["Loan\_Status"])

CoapplicantIncome\_Bin.div(CoapplicantIncome\_Bin.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsize=(4,4))

plt.xlabel("CoapplicantIncome")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

CoapplicantIncome\_bin

Low 3 15

Average 46 123

High 24 53

Chart, bar chart

Description automatically generated

It shows that if coapplicant’s income is less the chances of loan approval are high. But this does not look right. The possible reason behind this may be that most of the applicants don’t have any coapplicant so the coapplicant income for such applicants is 0 and hence the loan approval is not dependent on it. So we can make a new variable in which we will combine the applicant’s and coapplicant’s income to visualize the combined effect of income on loan approval.

In [58]:

train["TotalIncome"]=train["ApplicantIncome"]+train["CoapplicantIncome"]

In [59]:

bins =[0,2500,4000,6000,81000]

group=['Low','Average','High','Very High']

train["TotalIncome\_bin"]=pd.cut(train["TotalIncome"],bins,labels=group)

In [60]:

print(pd.crosstab(train["TotalIncome\_bin"],train["Loan\_Status"]))

TotalIncome = pd.crosstab(train["TotalIncome\_bin"],train["Loan\_Status"])

TotalIncome.div(TotalIncome.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsize=(2,2))

plt.xlabel("TotalIncome")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

TotalIncome\_bin

Low 14 10

Average 32 87

High 65 159

Very High 81 166

Chart

Description automatically generated

Whose TotalIncome was Low and loan was approved : 10

Whose TotalIncome was Low and loan was not approved : 14

Whose TotalIncome was Aerage and loan was apprvoed : 87

Whose TotalIncome was Average and loan was not approved : 32

Whose TotalIncome was High and loan was approved : 159

Whose TotalIncome was High and loan was not approved : 65

Whose TotalIncome was Very High and loan was approved : 166

Whose TotalIncome was Very High and loan was not approed : 81

We can see that Proportion of loans getting approved for applicants having low Total\_Income is very less as compared to that of applicants with Average, High and Very High Income.

Relation between "Loan\_Status" and "Loan Amount"

In [61]:

bins = [0,100,200,700]

group=['Low','Average','High']

train["LoanAmount\_bin"]=pd.cut(df["LoanAmount"],bins,labels=group)

In [62]:

print(pd.crosstab(train["LoanAmount\_bin"],train["Loan\_Status"]))

LoanAmount=pd.crosstab(train["LoanAmount\_bin"],train["Loan\_Status"])

LoanAmount.div(LoanAmount.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsize=(4,4))

plt.xlabel("LoanAmount")

plt.ylabel("Percentage")

plt.show()

Loan\_Status N Y

LoanAmount\_bin

Low 38 86

Average 83 207

High 27 39

Chart, bar chart

Description automatically generated

Whose Loan Amount was low and Loan was approved : 86

Whose Loan Amount was low and Loan was not approved : 38

Whose Loan Amount was Average and Loan was approved : 207

Whose Loan Amount was Average and Loan was not approved : 83

Whose Loan Amount was High and Loan was approved : 39

Whose Loan Amount was High and Loan was not approved : 27

It can be seen that the proportion of approved loans is higher for Low and Average Loan Amount as compared to that of High Loan Amount which supports our hypothesis in which we considered that the chances of loan approval will be high when the loan amount is less.

Let’s drop the bins which we created for the exploration part. We will change the 3+ in dependents variable to 3 to make it a numerical variable

In [63]:

train=train.drop(["Income\_bin","CoapplicantIncome\_bin","LoanAmount\_bin","TotalIncome","TotalIncome\_bin"],axis=1)

In [64]:

*#train['Dependents'].replace(('0', '1', '2', '3+'), (0, 1, 2, 3),inplace=True)*

*#test['Dependents'].replace(('0', '1', '2', '3+'), (0, 1, 2, 3),inplace=True)*

train['Dependents'].replace('3+',3,inplace=True)

test['Dependents'].replace('3+',3,inplace=True)

train['Loan\_Status'].replace('N', 0,inplace=True)

train['Loan\_Status'].replace('Y', 1,inplace=True)

Now lets look at the correlation between all the numerical variables. We will use the heat map to visualize the correlation. Heatmaps visualize data through variations in coloring. The variables with darker color means their correlation is more.

In [65]:

matrix = train.corr()

f, ax = plt.subplots(figsize=(10, 12))

sns.heatmap(matrix, vmax=.8, square=True, cmap="BuPu",annot=True);

Graphical user interface, application, Teams

Description automatically generated

We see that the most correlated variables are (ApplicantIncome - LoanAmount) and (Credit\_History - Loan\_Status).

Missing Value and Outlier Treatment

After exploring all the variables in our data, we can now impute the missing values and treat the outliers because missing data and outliers can have adverse effect on the model performance.

Missing value imputation

Let’s list out feature-wise count of missing values.

In [66]:

train.isnull().sum()

Out[66]:

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

dtype: int64

There are missing values in Gender, Married, Dependents, Self\_Employed, LoanAmount, Loan\_Amount\_Term and Credit\_History features.

We will treat the missing values in all the features one by one.

We can consider these methods to fill the missing values:

For numerical variables: imputation using mean or median

For categorical variables: imputation using mode

There are very less missing values in Gender, Married, Dependents, Credit\_History and Self\_Employed features so we can fill them using the mode of the features

In [67]:

train["Gender"].fillna(train["Gender"].mode()[0],inplace=True)

train["Married"].fillna(train["Married"].mode()[0],inplace=True)

train['Dependents'].fillna(train["Dependents"].mode()[0],inplace=True)

train["Self\_Employed"].fillna(train["Self\_Employed"].mode()[0],inplace=True)

train["Credit\_History"].fillna(train["Credit\_History"].mode()[0],inplace=True)

Now let’s try to find a way to fill the missing values in Loan\_Amount\_Term. We will look at the value count of the Loan amount term variable.

In [68]:

train["Loan\_Amount\_Term"].value\_counts()

Out[68]:

360.0 512

180.0 44

480.0 15

300.0 13

84.0 4

240.0 4

120.0 3

36.0 2

60.0 2

12.0 1

Name: Loan\_Amount\_Term, dtype: int64

It can be seen that in loan amount term variable, the value of 360 is repeating the most. So we will replace the missing values in this variable using the mode of this variable.

In [69]:

train["Loan\_Amount\_Term"].fillna(train["Loan\_Amount\_Term"].mode()[0],inplace=True)

In [70]:

train["Loan\_Amount\_Term"].value\_counts()

Out[70]:

360.0 526

180.0 44

480.0 15

300.0 13

84.0 4

240.0 4

120.0 3

36.0 2

60.0 2

12.0 1

Name: Loan\_Amount\_Term, dtype: int64

Now we will see the LoanAmount variable. As it is a numerical variable, we can use mean or median to impute the missing values.

We will use median to fill the null values as earlier we saw that loan amount have outliers so the mean will not be the proper approach as it is highly affected by the presence of outliers.

In [71]:

train["LoanAmount"].fillna(train["LoanAmount"].median(),inplace=True)

Now lets check whether all the missing values are filled in the dataset.

In [72]:

train.isnull().sum()

Out[72]:

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

As we can see that all the missing values have been filled in the train dataset.

Let’s fill all the missing values in the test dataset too with the same approach

In [73]:

test.isnull().sum()

Out[73]:

Loan\_ID 0

Gender 11

Married 0

Dependents 10

Education 0

Self\_Employed 23

ApplicantIncome 0

CoapplicantIncome 0

LoanAmount 5

Loan\_Amount\_Term 6

Credit\_History 29

Property\_Area 0

dtype: int64

In [74]:

test["Gender"].fillna(test["Gender"].mode()[0],inplace=True)

test['Dependents'].fillna(test["Dependents"].mode()[0],inplace=True)

test["Self\_Employed"].fillna(test["Self\_Employed"].mode()[0],inplace=True)

test["Loan\_Amount\_Term"].fillna(test["Loan\_Amount\_Term"].mode()[0],inplace=True)

test["Credit\_History"].fillna(test["Credit\_History"].mode()[0],inplace=True)

test["LoanAmount"].fillna(test["LoanAmount"].median(),inplace=True)

In [75]:

test.isnull().sum()

Out[75]:

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

dtype: int64

In [76]:

sns.distplot(train["LoanAmount"]);

A picture containing histogram

Description automatically generated

In [77]:

train['LoanAmount'].hist(bins=20)

Out[77]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c4a2be0>

Chart, histogram

Description automatically generated

Due to these outliers bulk of the data in the loan amount is at the left and the right tail is longer. This is called right skewness.

One way to remove the skewness is by doing the log transformation. As we take the log transformation, it does not affect the smaller values much, but reduces the larger values.

So, we get a distribution similar to normal distribution.

Let’s visualize the effect of log transformation.

We will do the similar changes to the test file simultaneously.

In [78]:

train['LoanAmount\_log'] = np.log(train['LoanAmount'])

train['LoanAmount\_log'].hist(bins=20)

Out[78]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c55a6a0>

Chart, histogram

Description automatically generated

In [79]:

sns.distplot(train["LoanAmount\_log"])

Out[79]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c446eb8>

Histogram

Description automatically generated

Let's have a look in test set [LoanAmount]

In [80]:

test["LoanAmount\_log"]=np.log(train["LoanAmount"])

test['LoanAmount\_log'].hist(bins=20)

Out[80]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c419a20>

Chart, histogram

Description automatically generated

In [81]:

sns.distplot(test["LoanAmount\_log"])

Out[81]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c3e6048>

Chart, histogram

Description automatically generated

Feature Engineering

Based on the domain knowledge, we can come up with new features that might affect the target variable. We will create the following three new features:

Total Income - As discussed during bivariate analysis we will combine the Applicant Income and Coapplicant Income. If the total income is high, chances of loan approval might also be high.

EMI - EMI is the monthly amount to be paid by the applicant to repay the loan. Idea behind making this variable is that people who have high EMI’s might find it difficult to pay back the loan. We can calculate the EMI by taking the ratio of loan amount with respect to loan amount term.

Balance Income - This is the income left after the EMI has been paid. Idea behind creating this variable is that if this value is high, the chances are high that a person will repay the loan and hence increasing the chances of loan approval.

In [82]:

train["TotalIncome"]=train["ApplicantIncome"]+train["CoapplicantIncome"]

Just have a look of train dataset "TotalIncome"

In [83]:

train[["TotalIncome"]].head()

Out[83]:

|  | TotalIncome |
| --- | --- |
| 0 | 5849.0 |
| 1 | 6091.0 |
| 2 | 3000.0 |
| 3 | 4941.0 |
| 4 | 6000.0 |

In [84]:

test["TotalIncome"]=test["ApplicantIncome"]+test["CoapplicantIncome"]

Just have a look of test dataset "TotalIncome"

In [85]:

test[["TotalIncome"]].head()

Out[85]:

|  | TotalIncome |
| --- | --- |
| 0 | 5720 |
| 1 | 4576 |
| 2 | 6800 |
| 3 | 4886 |
| 4 | 3276 |

Let’s check the distribution of train dataset Total Income.

In [86]:

sns.distplot(train["TotalIncome"])

Out[86]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33cb94eb8>

A picture containing shape

Description automatically generated

We can see it is shifted towards left, i.e., the distribution is right skewed. So, let’s take the log transformation to make the distribution normal.

In [87]:

train["TotalIncome\_log"]=np.log(train["TotalIncome"])

sns.distplot(train["TotalIncome\_log"])

Out[87]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c4574e0>

Chart, histogram

Description automatically generated

Now the distribution looks much closer to normal and effect of extreme values has been significantly subsided.

Let’s check the distribution of test dataset Total Income.

In [88]:

sns.distplot(test["TotalIncome"])

Out[88]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c9dd9b0>

A picture containing shape

Description automatically generated

We can see it is shifted towards left, i.e., the distribution is right skewed. So, let’s take the log transformation to make the distribution normal.

In [89]:

test["TotalIncome\_log"] = np.log(train["TotalIncome"])

sns.distplot(test["TotalIncome\_log"])

Out[89]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c82f278>

Chart, histogram

Description automatically generated

Now the distribution looks much closer to normal and effect of extreme values has been significantly subsided.

Now create the EMI feature.

In [90]:

train["EMI"]=train["LoanAmount"]/train["Loan\_Amount\_Term"]

test["EMI"]=test["LoanAmount"]/test["Loan\_Amount\_Term"]

Have a look of train dataset "EMI"

In [91]:

train[["EMI"]].head()

Out[91]:

|  | EMI |
| --- | --- |
| 0 | 0.355556 |
| 1 | 0.355556 |
| 2 | 0.183333 |
| 3 | 0.333333 |
| 4 | 0.391667 |

Again have a look of test dataset "EMI"

In [92]:

test[["EMI"]].head()

Out[92]:

|  | EMI |
| --- | --- |
| 0 | 0.305556 |
| 1 | 0.350000 |
| 2 | 0.577778 |
| 3 | 0.277778 |
| 4 | 0.216667 |

Let’s check the distribution of EMI variable.

In [93]:

sns.distplot(train["EMI"])

Out[93]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33ca46278>

Shape

Description automatically generated

In [94]:

sns.distplot(test["EMI"])

Out[94]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa33c7cd7f0>

Shape, square

Description automatically generated

Let's create Balance Income feature now and check its distribution.

In [95]:

train["Balance\_Income"] = train["TotalIncome"]-train["EMI"]\*1000 *# To make the units equal we multiply with 1000*

test["Balance\_Income"] = test["TotalIncome"]-test["EMI"]

Have a look of train dataset "Balance Income"

In [96]:

train[["Balance\_Income"]].head()

Out[96]:

|  | Balance\_Income |
| --- | --- |
| 0 | 5493.444444 |
| 1 | 5735.444444 |
| 2 | 2816.666667 |
| 3 | 4607.666667 |
| 4 | 5608.333333 |

Have a look of test dataset "Balance Income"

In [97]:

test[["Balance\_Income"]].head()

Out[97]:

|  | Balance\_Income |
| --- | --- |
| 0 | 5719.694444 |
| 1 | 4575.650000 |
| 2 | 6799.422222 |
| 3 | 4885.722222 |
| 4 | 3275.783333 |

Let us now drop the variables which we used to create these new features. Reason for doing this is, the correlation between those old features and these new features will be very high and logistic regression assumes that the variables are not highly correlated. We also wants to remove the noise from the dataset, so removing correlated features will help in reducing the noise too.

In [98]:

train=train.drop(["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan\_Amount\_Term"],axis=1)

In [99]:

train.head()

Out[99]:

|  | Loan\_ID | Gender | Married | Dependents | Education | Self\_Employed | Credit\_History | Property\_Area | Loan\_Status | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | LP001002 | Male | No | 0 | Graduate | No | 1.0 | Urban | 1 | 4.852030 | 5849.0 | 8.674026 | 0.355556 | 5493.444444 |
| 1 | LP001003 | Male | Yes | 1 | Graduate | No | 1.0 | Rural | 0 | 4.852030 | 6091.0 | 8.714568 | 0.355556 | 5735.444444 |
| 2 | LP001005 | Male | Yes | 0 | Graduate | Yes | 1.0 | Urban | 1 | 4.189655 | 3000.0 | 8.006368 | 0.183333 | 2816.666667 |
| 3 | LP001006 | Male | Yes | 0 | Not Graduate | No | 1.0 | Urban | 1 | 4.787492 | 4941.0 | 8.505323 | 0.333333 | 4607.666667 |
| 4 | LP001008 | Male | No | 0 | Graduate | No | 1.0 | Urban | 1 | 4.948760 | 6000.0 | 8.699515 | 0.391667 | 5608.333333 |

In [100]:

test = test.drop(["ApplicantIncome","CoapplicantIncome","LoanAmount","Loan\_Amount\_Term"],axis=1)

In [101]:

test.head()

Out[101]:

|  | Loan\_ID | Gender | Married | Dependents | Education | Self\_Employed | Credit\_History | Property\_Area | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | LP001015 | Male | Yes | 0 | Graduate | No | 1.0 | Urban | 4.852030 | 5720 | 8.674026 | 0.305556 | 5719.694444 |
| 1 | LP001022 | Male | Yes | 1 | Graduate | No | 1.0 | Urban | 4.852030 | 4576 | 8.714568 | 0.350000 | 4575.650000 |
| 2 | LP001031 | Male | Yes | 2 | Graduate | No | 1.0 | Urban | 4.189655 | 6800 | 8.006368 | 0.577778 | 6799.422222 |
| 3 | LP001035 | Male | Yes | 2 | Graduate | No | 1.0 | Urban | 4.787492 | 4886 | 8.505323 | 0.277778 | 4885.722222 |
| 4 | LP001051 | Male | No | 0 | Not Graduate | No | 1.0 | Urban | 4.948760 | 3276 | 8.699515 | 0.216667 | 3275.783333 |

Model Building :

After creating new features, we can continue the model building process. So we will start with logistic regression model and then move over to more complex models like RandomForest and XGBoost

We will build the following models in this section.

i)Logistic Regression

ii)Decision Tree

iii)Random Forest

iv)Random Forest with Grid Search

v)XGBClassifier

Let’s prepare the data for feeding into the models.

Let's drop the "Loan\_ID" variable as it do not have any effect on the loan status. We will do the same changes to the test dataset which we did for the training dataset.

Drop "Loan\_ID"

In [102]:

train=train.drop("Loan\_ID",axis=1)

test=test.drop("Loan\_ID",axis=1)

After drop train dataset will look :

In [103]:

train.head(3)

Out[103]:

|  | Gender | Married | Dependents | Education | Self\_Employed | Credit\_History | Property\_Area | Loan\_Status | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Male | No | 0 | Graduate | No | 1.0 | Urban | 1 | 4.852030 | 5849.0 | 8.674026 | 0.355556 | 5493.444444 |
| 1 | Male | Yes | 1 | Graduate | No | 1.0 | Rural | 0 | 4.852030 | 6091.0 | 8.714568 | 0.355556 | 5735.444444 |
| 2 | Male | Yes | 0 | Graduate | Yes | 1.0 | Urban | 1 | 4.189655 | 3000.0 | 8.006368 | 0.183333 | 2816.666667 |

After drop test dataset will look :

In [104]:

test.head(3)

Out[104]:

|  | Gender | Married | Dependents | Education | Self\_Employed | Credit\_History | Property\_Area | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Male | Yes | 0 | Graduate | No | 1.0 | Urban | 4.852030 | 5720 | 8.674026 | 0.305556 | 5719.694444 |
| 1 | Male | Yes | 1 | Graduate | No | 1.0 | Urban | 4.852030 | 4576 | 8.714568 | 0.350000 | 4575.650000 |
| 2 | Male | Yes | 2 | Graduate | No | 1.0 | Urban | 4.189655 | 6800 | 8.006368 | 0.577778 | 6799.422222 |

We will use scikit-learn (sklearn) for making different models which is an open source library for Python. It is one of the most efficient tool which contains many inbuilt functions that can be used for modeling in Python.

Sklearn requires the target variable in a separate dataset. So, we will drop our target variable from the train dataset and save it in another dataset.

droping the target variable "Loan\_Status"

In [105]:

X=train.drop("Loan\_Status",1)

In [106]:

X.head(2)

Out[106]:

|  | Gender | Married | Dependents | Education | Self\_Employed | Credit\_History | Property\_Area | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | Male | No | 0 | Graduate | No | 1.0 | Urban | 4.85203 | 5849.0 | 8.674026 | 0.355556 | 5493.444444 |
| 1 | Male | Yes | 1 | Graduate | No | 1.0 | Rural | 4.85203 | 6091.0 | 8.714568 | 0.355556 | 5735.444444 |

save the target variable "Loan\_Status" in another dataset

In [107]:

y=train[["Loan\_Status"]]

In [108]:

y.head(2)

Out[108]:

|  | Loan\_Status |
| --- | --- |
| 0 | 1 |
| 1 | 0 |

Now we will make dummy variables for the categorical variables. Dummy variable turns categorical variables into a series of 0 and 1, making them lot easier to quantify and compare.

Let us understand the process of dummies first:

Consider the “Gender” variable. It has two classes, Male and Female.

As logistic regression takes only the numerical values as input, we have to change male and female into numerical value.

Once we apply dummies to this variable, it will convert the “Gender” variable into two variables(Gender\_Male and Gender\_Female), one for each class, i.e. Male and Female.

Gender\_Male will have a value of 0 if the gender is Female and a value of 1 if the gender is Male.

In [109]:

X = pd.get\_dummies(X)

In [110]:

X.head(3)

Out[110]:

|  | Credit\_History | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income | Gender\_Female | Gender\_Male | Married\_No | Married\_Yes | Dependents\_3 | Dependents\_0 | Dependents\_1 | Dependents\_2 | Education\_Graduate | Education\_Not Graduate | Self\_Employed\_No | Self\_Employed\_Yes | Property\_Area\_Rural | Property\_Area\_Semiurban | Property\_Area\_Urban |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1.0 | 4.852030 | 5849.0 | 8.674026 | 0.355556 | 5493.444444 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1.0 | 4.852030 | 6091.0 | 8.714568 | 0.355556 | 5735.444444 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1.0 | 4.189655 | 3000.0 | 8.006368 | 0.183333 | 2816.666667 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

In [111]:

train=pd.get\_dummies(train)

test=pd.get\_dummies(test)

In [112]:

train.head(3)

Out[112]:

|  | Credit\_History | Loan\_Status | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income | Gender\_Female | Gender\_Male | Married\_No | Married\_Yes | Dependents\_3 | Dependents\_0 | Dependents\_1 | Dependents\_2 | Education\_Graduate | Education\_Not Graduate | Self\_Employed\_No | Self\_Employed\_Yes | Property\_Area\_Rural | Property\_Area\_Semiurban | Property\_Area\_Urban |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1.0 | 1 | 4.852030 | 5849.0 | 8.674026 | 0.355556 | 5493.444444 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1.0 | 0 | 4.852030 | 6091.0 | 8.714568 | 0.355556 | 5735.444444 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 1.0 | 1 | 4.189655 | 3000.0 | 8.006368 | 0.183333 | 2816.666667 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |

In [113]:

test.head(3)

Out[113]:

|  | Credit\_History | LoanAmount\_log | TotalIncome | TotalIncome\_log | EMI | Balance\_Income | Gender\_Female | Gender\_Male | Married\_No | Married\_Yes | Dependents\_3 | Dependents\_0 | Dependents\_1 | Dependents\_2 | Education\_Graduate | Education\_Not Graduate | Self\_Employed\_No | Self\_Employed\_Yes | Property\_Area\_Rural | Property\_Area\_Semiurban | Property\_Area\_Urban |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1.0 | 4.852030 | 5720 | 8.674026 | 0.305556 | 5719.694444 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1.0 | 4.852030 | 4576 | 8.714568 | 0.350000 | 4575.650000 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 2 | 1.0 | 4.189655 | 6800 | 8.006368 | 0.577778 | 6799.422222 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

Now we will train the model on training dataset and make predictions for the test dataset. But can we validate these predictions? One way of doing this is we can divide our train dataset into two parts:train and validation. We can train the model on this train part and using that make predictions for the validation part. In this way we can validate our predictions as we have the true predictions for the validation part (which we do not have for the test dataset).

We will use the train\_test\_split function from sklearn to divide our train dataset. So, first let us import train\_test\_split.

In [114]:

from sklearn.model\_selection import train\_test\_split

In [115]:

x\_train,x\_cv,y\_train,y\_cv=train\_test\_split(X,y,test\_size=0.3,random\_state=1)

The dataset has been divided into training and validation part.

70% data will use for train the model and rest of the 30% data will use for checking validation of the model.

Logistic Regression

Let's import LogisticRegression and accuracy\_score from sklearn and fit the logistic regression model.

In [116]:

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

In [117]:

logistic\_model = LogisticRegression(random\_state=1)

In [118]:

logistic\_model.fit(x\_train,y\_train)

Out[118]:

LogisticRegression(C=1.0, class\_weight=None, dual=False, fit\_intercept=True,

intercept\_scaling=1, max\_iter=100, multi\_class='warn',

n\_jobs=None, penalty='l2', random\_state=1, solver='warn',

tol=0.0001, verbose=0, warm\_start=False)

Let’s predict the Loan\_Status for validation set

In [119]:

pred\_cv\_logistic=logistic\_model.predict(x\_cv)

Now calculate how accurate our predictions are by calculating the accuracy.

In [120]:

score\_logistic =accuracy\_score(pred\_cv\_logistic,y\_cv)\*100

In [121]:

score\_logistic

Out[121]:

78.91891891891892

So our predictions are almost 79% accurate, i.e. we have identified 79% of the loan status correctly for our logistic regression model.

Let’s make predictions for the test dataset.

In [122]:

pred\_test\_logistic = logistic\_model.predict(test)

Decision Tree

Let's import Decison Tree Classifier

In [123]:

from sklearn.tree import DecisionTreeClassifier

In [124]:

tree\_model = DecisionTreeClassifier(random\_state=1)

Fit the model :

In [125]:

tree\_model.fit(x\_train,y\_train)

Out[125]:

DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,

max\_features=None, max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, presort=False, random\_state=1,

splitter='best')

In [126]:

pred\_cv\_tree=tree\_model.predict(x\_cv)

In [127]:

score\_tree =accuracy\_score(pred\_cv\_tree,y\_cv)\*100

In [128]:

score\_tree

Out[128]:

71.35135135135135

our predictions are almost 71% accurate, i.e. we have identified 71% of the loan status correctly for our Decision tree model.

Let’s make predictions for the test dataset.

In [129]:

pred\_test\_tree = tree\_model.predict(test)

Random Forest

i)RandomForest is a tree based bootstrapping algorithm wherein a certain no. of weak learners (decision trees) are combined to make a powerful prediction model.

ii)For every individual learner, a random sample of rows and a few randomly chosen variables are used to build a decision tree model.

iii)Final prediction can be a function of all the predictions made by the individual learners.

Let's import Random Forest Classifier

In [130]:

from sklearn.ensemble import RandomForestClassifier

In [131]:

forest\_model = RandomForestClassifier(random\_state=1,max\_depth=10,n\_estimators=50)

In [132]:

forest\_model.fit(x\_train,y\_train)

Out[132]:

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=10, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=50, n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False)

In [133]:

pred\_cv\_forest=forest\_model.predict(x\_cv)

In [134]:

score\_forest = accuracy\_score(pred\_cv\_forest,y\_cv)\*100

In [135]:

score\_forest

Out[135]:

77.83783783783784

our predictions are almost 78% accurate, i.e. we have identified 78% of the loan status correctly for our Random Forest model.

Let’s make predictions for the test dataset.

In [136]:

pred\_test\_forest=forest\_model.predict(test)

Random Forest with Grid-search

We will try to improve the accuracy by tuning the hyperparameters for this model. We will use grid search to get the optimized values of hyper parameters. Grid-search is a way to select the best of a family of hyper parameters, parametrized by a grid of parameters.

We will tune the max\_depth and n\_estimators parameters. max\_depth decides the maximum depth of the tree and n\_estimators decides the number of trees that will be used in random forest model.

Let's import GridSearchCV

In [137]:

from sklearn.model\_selection import GridSearchCV

Provide range for max\_depth from 1 to 20 with an interval of 2 and from 1 to 200 with an interval of 20 for n\_estimators.

In [138]:

paramgrid = {'max\_depth': list(range(1,20,2)),'n\_estimators':list(range(1,200,20))}

In [139]:

grid\_search = GridSearchCV(RandomForestClassifier(random\_state=1),paramgrid)

Fit the grid search model

In [140]:

grid\_search.fit(x\_train,y\_train)

/opt/conda/lib/python3.6/site-packages/sklearn/model\_selection/\_search.py:841: DeprecationWarning: The default of the `iid` parameter will change from True to False in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.

DeprecationWarning)

Out[140]:

GridSearchCV(cv='warn', error\_score='raise-deprecating',

estimator=RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=None, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators='warn', n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False),

fit\_params=None, iid='warn', n\_jobs=None,

param\_grid={'max\_depth': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19], 'n\_estimators': [1, 21, 41, 61, 81, 101, 121, 141, 161, 181]},

pre\_dispatch='2\*n\_jobs', refit=True, return\_train\_score='warn',

scoring=None, verbose=0)

Find out the optimized value

In [141]:

grid\_search.best\_estimator\_

Out[141]:

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=3, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=101, n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False)

The optimized value for the max\_depth variable is 3 and for n\_estimator is 101,random\_state = 1. Now let’s build the model using these optimized values.

In [142]:

grid\_forest\_model = RandomForestClassifier(random\_state=1,max\_depth=3,n\_estimators=101)

In [143]:

grid\_forest\_model.fit(x\_train,y\_train)

Out[143]:

RandomForestClassifier(bootstrap=True, class\_weight=None, criterion='gini',

max\_depth=3, max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=101, n\_jobs=None,

oob\_score=False, random\_state=1, verbose=0, warm\_start=False)

In [144]:

pred\_grid\_forest = grid\_forest\_model.predict(x\_cv)

In [145]:

score\_grid\_forest = accuracy\_score(pred\_grid\_forest,y\_cv)\*100

In [146]:

score\_grid\_forest

Out[146]:

76.75675675675676

our predictions are almost 77% accurate, i.e. we have identified 77% of the loan status correctly for our Random Forest model with grid search.

In [147]:

pred\_grid\_forest\_test = grid\_forest\_model.predict(test)

XGBoost

XGBoost is a fast and efficient algorithm.XGBoost works only with numeric variables and we have already replaced the categorical variables with numeric variables.

Let’s have a look at the parameters that we are going to use in our model

Let's import XGBClassifier and fit the model.

n\_estimator: This specifies the number of trees for the model.

max\_depth: We can specify maximum depth of a tree using this parameter.

In [148]:

from xgboost import XGBClassifier

In [149]:

xgb\_model = XGBClassifier(n\_estimators=50,max\_depth=4)

In [150]:

xgb\_model.fit(x\_train,y\_train)

Out[150]:

XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1,

colsample\_bytree=1, gamma=0, learning\_rate=0.1, max\_delta\_step=0,

max\_depth=4, min\_child\_weight=1, missing=None, n\_estimators=50,

n\_jobs=1, nthread=None, objective='binary:logistic', random\_state=0,

reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, seed=None,

silent=True, subsample=1)

In [151]:

pred\_xgb=xgb\_model.predict(x\_cv)

In [152]:

score\_xgb = accuracy\_score(pred\_xgb,y\_cv)\*100

In [153]:

score\_xgb

Out[153]:

78.37837837837837

our predictions are almost 78% accurate, i.e. we have identified 78% of the loan status correctly for our XGBClassifier model.

Logistic Regression model gives : 79% prediction accuracy

Decision Tree model gives : 71% prediction accuracy

Random Forest model gives : 78% prediction accuracy

Random Forest with Grid Search model gives : 77% prediction accuracy

XGBClassifier model gives : 78% prediction accuracy

Find the important feature

Let us find the feature importance now, i.e. which features are most important for this problem. We will use feature\_importances\_attribute of sklearn to do it.

As 'LogisticRegression' object has no attribute 'feature*importances*' so we choose next high accuracy predictive model. Random Forest model is 2nd highest model.

Using Random Forest model we can find out most important feature among the features.

In [154]:

importances = pd.Series(forest\_model.feature\_importances\_,index=X.columns)

importances.plot(kind='barh', figsize=(12,8))

Out[154]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fa332b8a828>

A picture containing bar chart

Description automatically generated

We can find out that 'Credit\_History','Balance Income' feature are most important. So, feature engineering helped us in predicting our target variable.