• Loan Prediction Machine Learning Blog:

1] Problem Definition:

Understanding the problem statement/ definition 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.

- Any Finance company deals in all types of loans loans. They have a presence across all urban, semi-urban and rural areas. Customers first apply for a loan after that company validates the customer's eligibility for a loan. The company wants to automate the loan eligibility process (real-time) based on customer detail provided while filling out the 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 customer segments, that are eligible for loan amounts so that they can specifically target these customers.

- It is a classification problem where we have to predict whether a loan would be approved or not. In these kinds of problems, we have to predict discrete values based on a given set of independent variables (s).

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

Hypothesis Generation:

This is a very important stage in a data science/machine learning pipeline. It involves understanding the problem in detail by brainstorming maximum possibilities that can impact the outcome. It is done by thoroughly understanding the problem statement before looking at the data.

2] Data Analysis :

Below are some of the factors which I think can affect the Loan Approval :

1. Loan_ID - This refer to the unique identifier of the applicant's affirmed purchases

2. Gender - This refers to either of the two main categories (male and female) into which applicants are divided on the basis of their reproductive functions

3. Married - This refers to applicant being in a state of matrimony

4. Dependents - This refers to persons who depends on the applicants for survival

5. Education - This refers to number of years in which applicant received systematic instruction, especially at a school or university

6. Self-employed - This refers to applicant working for oneself as a freelancer or the owner of a business rather than for an employer

7. Applicant Income - This refers to disposable income available for the applicant's use under State law.

8. CoapplicantIncome - This refers to disposable income available for the people that participate in the loan application process alongside the main applicant use under State law.

9. Loan Amount - This refers to the amount of money an applicant owe at any given time.

10. Loan_Amount_Term - This refers to the duration in which the loan is availed to the applicant

11. Credit History - This refers to a record of applicant's ability to repay debts and demonstrated responsibility in repaying them.

12. Property Area - This refers to the total area within the boundaries of the property as set out in Schedule.

13. Loan Status - This refers to whether applicant is eligible to be availed the Loan requested.

• Getting the System Ready and Loading the Data :

We will be using Python for this problem along with the below-listed libraries. The version of these libraries is mentioned below:

- ## Loading Packages :

import pandas as pd

import numpy as np #For mathematical calculations

import matplotlib.pyplot as plt #For plotting graphs

import seaborn as sns #For data visualization

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split

import warnings #To ignore any warnings

warnings.filterwarnings("ignore")

- ## Data :

For this problem, we have been given CSV file which contains all independent and dependent data.

- ## Reading Data :

#Loading dataset:-

df=pd.read_csv('loan_prediction.csv')

- ## Understanding the Data :

In this section, we will look at the dataset. Firstly, we will check the features present in our data, and then we will look at their data types.

df.columns

```
Index([‘Loan_ID’,'Gender', 'Married', 'Dependents', 'Education', 'Self_
Employed',
       'Applicant Income', 'CoapplicantIncome', 'Loan_Amount',
```

```
       'Loan_Amount_Term', 'Credit History', 'Property_Area', 'Loan_sta
tus'],
      dtype='object')
```

We have 12 independent variables and 1 target variable, i.e. Loan_Status in the 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	Loan approved (Y/N)

Print data types for each variable:

#As we drop index Loan_ID column because it just kind of id which have no effect on dataset:

df.drop('Loan_ID',inplace=True,axis=1)

df.dtypes

```
Gender               object
Married              object
```

```
Dependents            object
Education             object
Self_Employed         object
Applicant Income       int64
CoapplicantIncome    float64
Loan_Amount          float64
Loan_Amount_Term     float64
Credit History       float64
Property_Area         object
Loan_Status           object
dtype: object
```

We can see there are three formats of data types:

- **Object :** Object format means variables are categorical. Categorical variables in our dataset are: 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 that has some decimal values involved. They are also numerical variables. Numerical variables in our dataset are: CoapplicantIncome, LoanAmount, Loan_Amount_Term, and Credit_History

Let's look at the shape of the dataset:

#Checking the total rows and total columns:

print(df.shape)

```
(613, 12)
```

We have 613 rows and 12 columns in the dataset

3] EDA Concluding Remarks:

First We see the information of our dataset:

df.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 613 entries, 0 to 612
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Gender             600 non-null    object
 1   Married            610 non-null    object
 2   Dependents         598 non-null    object
 3   Education          613 non-null    object
 4   Self_Employed      581 non-null    object
 5   Applicant Income   613 non-null    int64
 6   CoapplicantIncome  613 non-null    float64
 7   Loan_Amount        592 non-null    float64
 8   Loan_Amount_Term   599 non-null    float64
 9   Credit History     563 non-null    float64
 10  Property_Area      613 non-null    object
 11  Loan_Status        613 non-null    object
dtypes: float64(4), int64(1), object(7)
memory usage: 57.6+ KB
```

* Univariate Analysis :

In this section, we will do a univariate analysis. It is the easiest form of analyzing data where we analyze each variable individually. For categorical features, we can use frequency tables or bar plots to calculate the number of each category in a particular variable. Probability Density Functions(PDF) can be used to look at the distribution of the numerical variables.

- ## Target Variable:

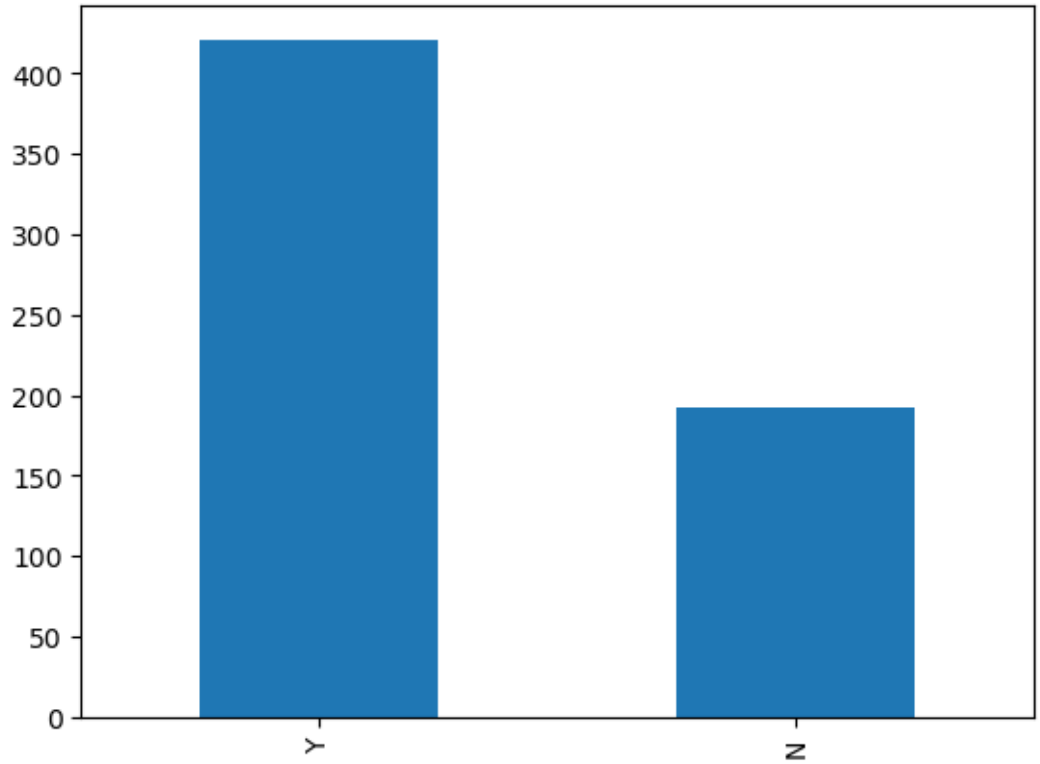
We will first look at the target variable, i.e., Loan_Status. As it is a categorical variable, let us look at its frequency table, percentage distribution, and bar plot.

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

df['Loan_Status'].value_counts()

```
Y    421
N    192
Name: Loan_Status, dtype: int64
```

```
df['Loan_Status'].value_counts().plot.bar()
```




```
422(around 69%) people out of 614 got the approval.
```

```
Now, let's 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)
- Ordinal features: Variables in categorical features having some order involved (Dependents, Education, Property_Area)
- Numerical features: These features have numerical values (ApplicantIncome, CoapplicantIncome, LoanAmount, Loan_Amount_Term)

Let's visualize the categorical and ordinal features first:-

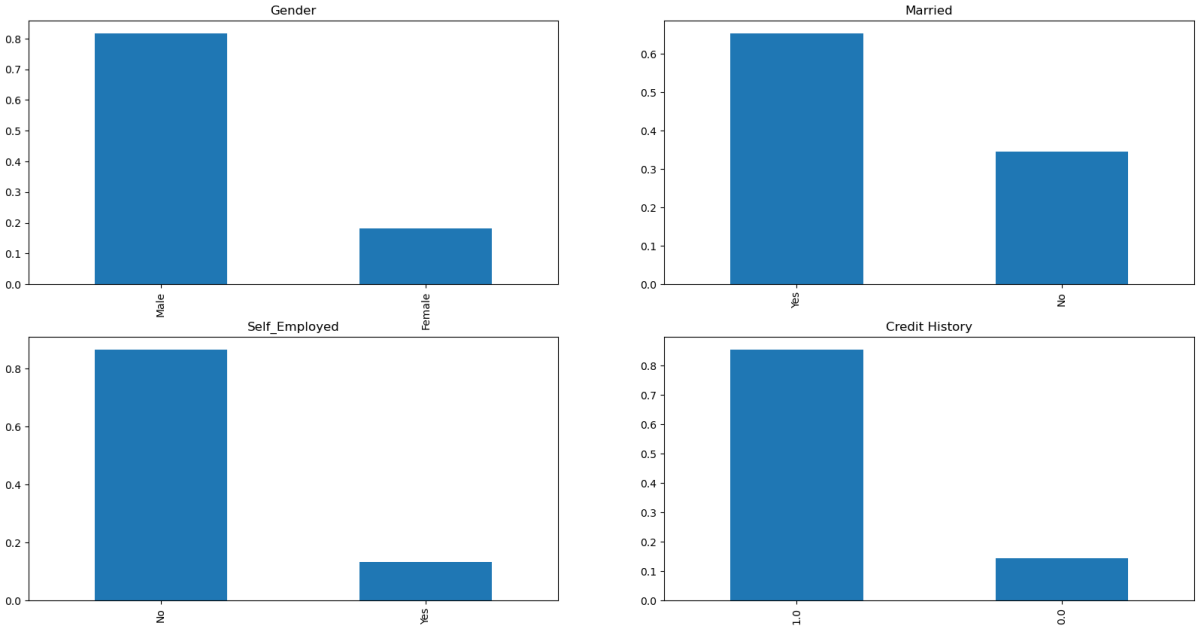
Independent Variable (Categorical) :

plt.figure(1)

plt.subplot(221)

df['Gender'].value_counts(normalize=True).plot.bar(figsize=(20,10), title= 'Gender')

```
plt.subplot(222)
df['Married'].value_counts(normalize=True).plot.bar(title= 'Married')
plt.subplot(223)
df['Self_Employed'].value_counts(normalize=True).plot.bar(title= 'Self_Employed')
plt.subplot(224)
df['Credit History'].value_counts(normalize=True).plot.bar(title= 'Credit History')
plt.show()
```



It can be inferred from the above bar plots that:

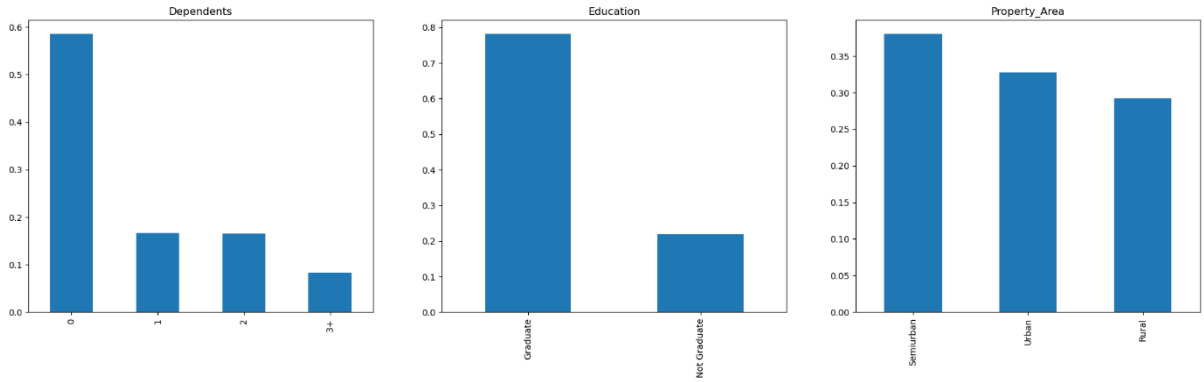
- 80% of applicants in the dataset are male.
- Around 65% of the applicants in the dataset are married.
- About 15% of applicants in the dataset are self-employed.
- About 85% of applicants have repaid their debts.

Now let's visualize the ordinal variables:-

Independent Variable (Ordinal):

```
plt.figure(1)
```

```
plt.subplot(131)
df['Dependents'].value_counts(normalize=True).plot.bar(figsize=(24,6),title='Dependents')
plt.subplot(132)
df['Education'].value_counts(normalize=True).plot.bar(title= 'Education')
plt.subplot(133)
df['Property_Area'].value_counts(normalize=True).plot.bar(title= 'Property_Area')
plt.show()
```



Following inferences can be made from the above bar plots:

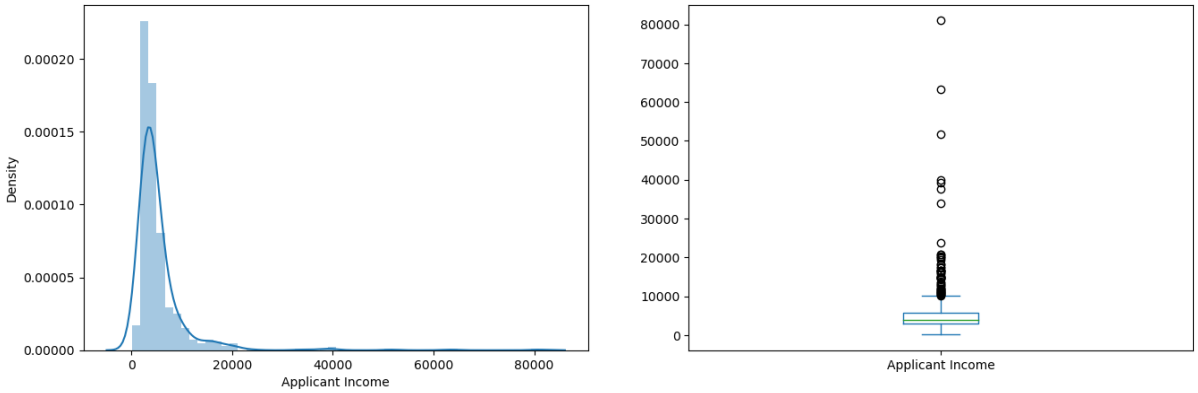
- Most of the applicants don't have dependents.
- About 80% of the applicants are graduates.
- Most of the applicants are from semi-urban areas.

Independent Variable (Numerical):

Till now we have seen the categorical and ordinal variables and now let's visualize the numerical variables. Let's look at the distribution of Applicant income first:-

```
plt.figure(1)
plt.subplot(121)
sns.distplot(df['Applicant Income']);
plt.subplot(122)
```

```
df['Applicant Income'].plot.box(figsize=(16,5))

plt.show()
```

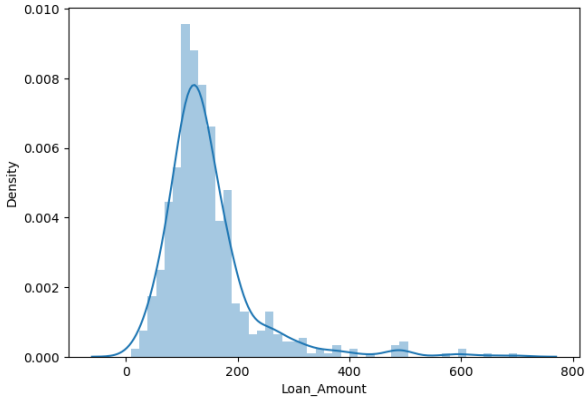


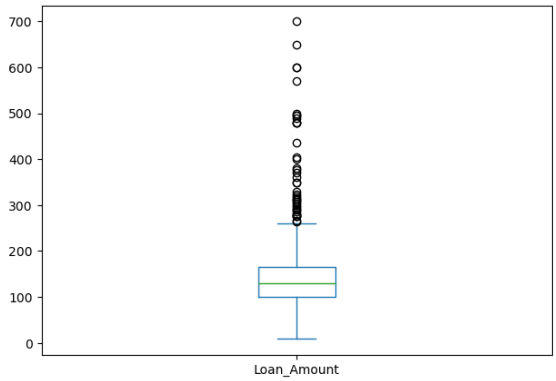
It can be inferred that most of the data in the distribution of applicant income are towards the left which means it is not normally distributed.

We will try to make it normal in later sections as algorithms work better if the data is normally distributed.

Let's look at the distribution of the LoanAmount variable:

```
plt.figure(1)

plt.subplot(121)

df=df.dropna()

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

plt.subplot(122)

df['Loan_Amount'].plot.box(figsize=(16,5))

plt.show()
```





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

Now we would like to know how well each feature correlates with Loan Status. So, in the next section, we will look at the bivariate analysis:-

Bivariate Analysis:-

- Applicants with high incomes should have more chances of loan approval.
- Applicants who have repaid their previous debts should have higher chances of loan approval.
- Loan approval should also depend on the loan amount. If the loan amount is less, the chances of loan approval should be high.
- Lesser the amount to be paid monthly to repay the loan, the higher the chances of loan approval.

Let's try to test the above-mentioned hypotheses using bivariate analysis:-

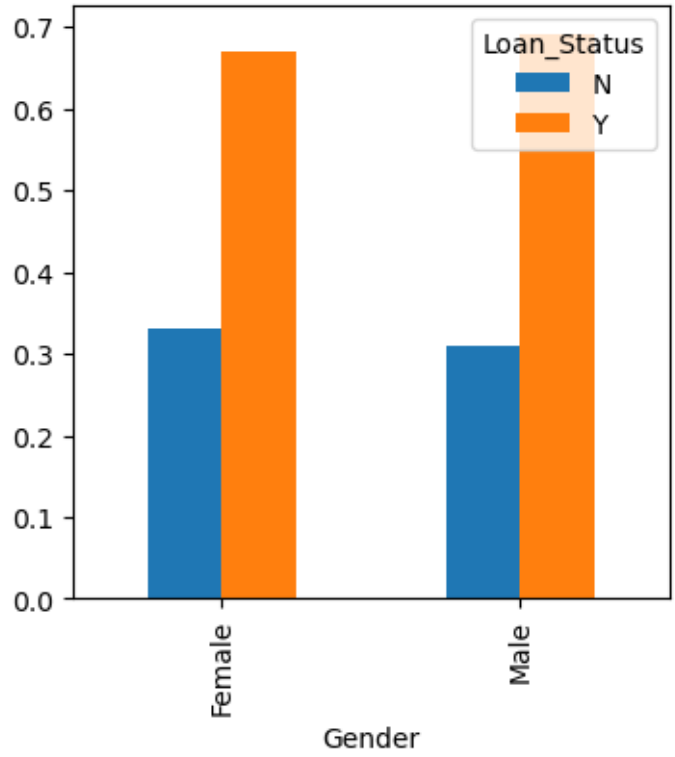
After looking at every variable individually in univariate analysis, we will now explore them again with respect to the target variable.

Categorical Independent Variable vs Target Variable:-

First of all, we will find the relation between the 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:

```
Gender=pd.crosstab(df['Gender'],df['Loan_Status'])
```

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



It can be inferred that the proportion of male and female applicants is more or less the same for both approved and unapproved loans.

Now let us visualize the remaining categorical variables vs the target variables:-

```
Married=pd.crosstab(df['Married'],df['Loan_Status'])
```

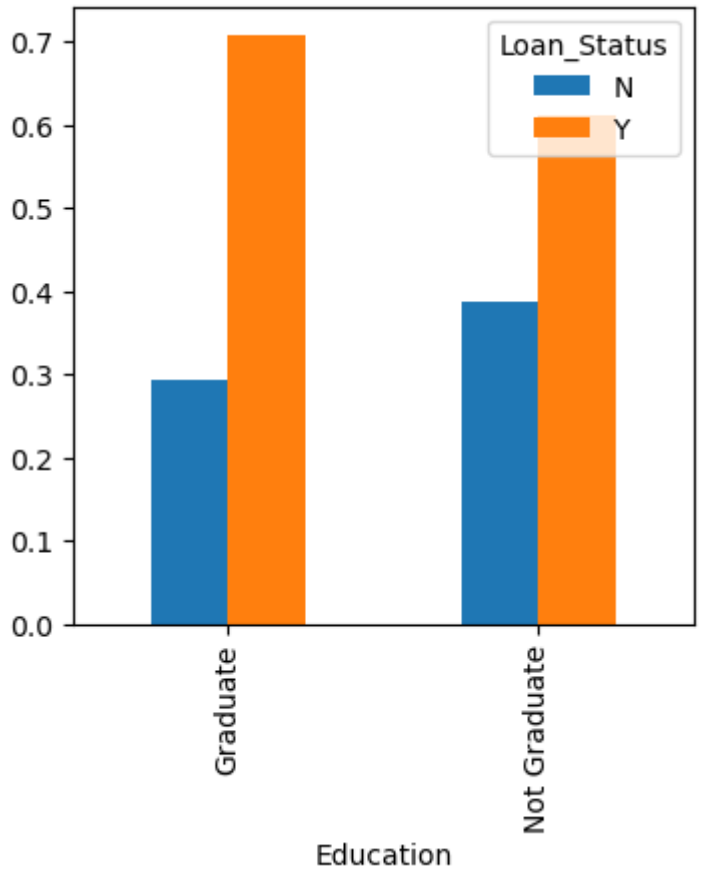
```
Dependents=pd.crosstab(df['Dependents'],df['Loan_Status'])
```

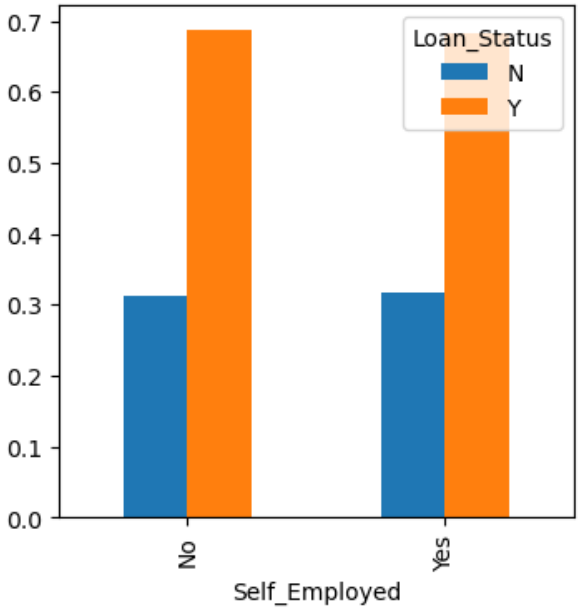
```
Education=pd.crosstab(df['Education'],df['Loan_Status'])
```

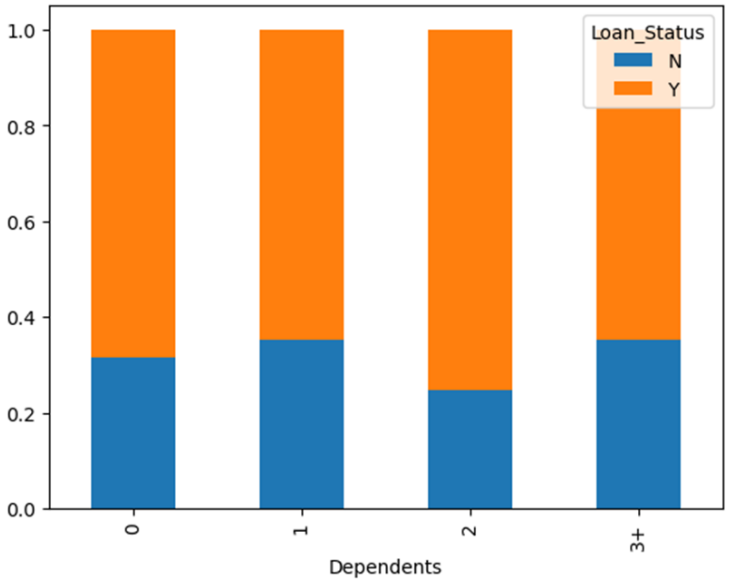
```
Self_Employed=pd.crosstab(df['Self_Employed'],df['Loan_Status'])
```

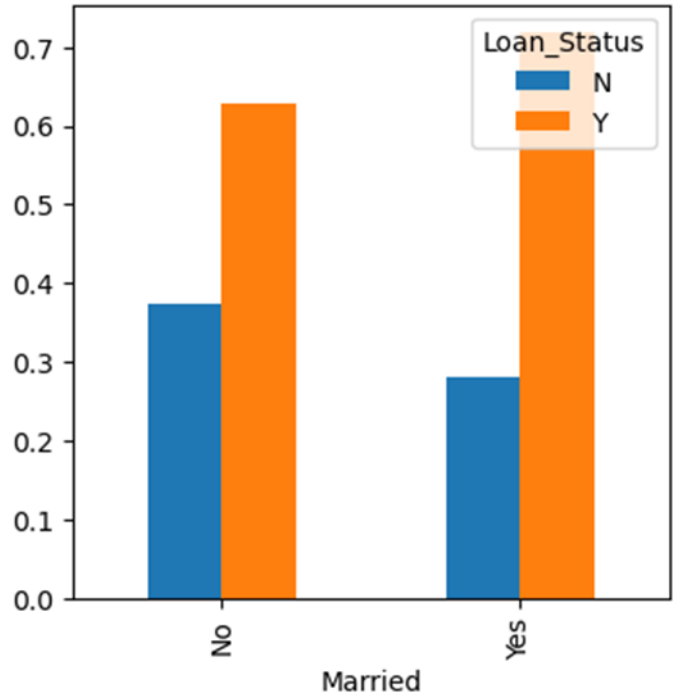
```
Married.div(Married.sum(1).astype(float), axis=0).plot(kind="bar", figsize=(4,4))
```

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




Loan_Status
N
Y
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
No
Yes
Self_Employed





- The proportion of married applicants is higher for the approved loans.
- The distribution of applicants with 1 or 3+ dependents is similar across both the categories of Loan_Status.
- There is nothing significant we can infer from Self_Employed vs Loan_Status plot.

Now we will look at the relationship between the remaining categorical independent variables and Loan_Status:-

Credit_History=pd.crosstab(df['Credit History'],df['Loan_Status'])

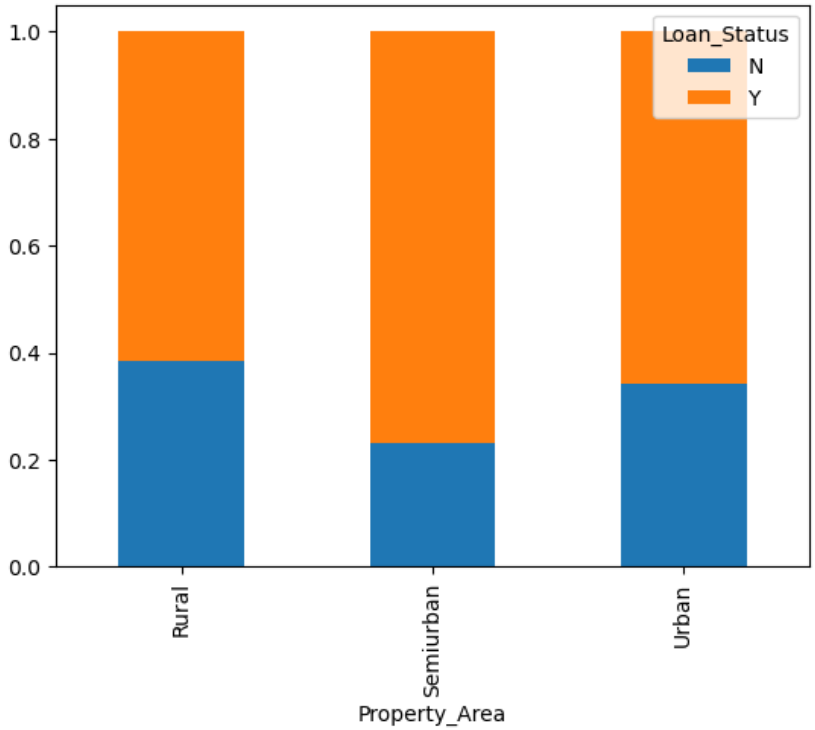
Property_Area=pd.crosstab(df['Property_Area'],df['Loan_Status'])

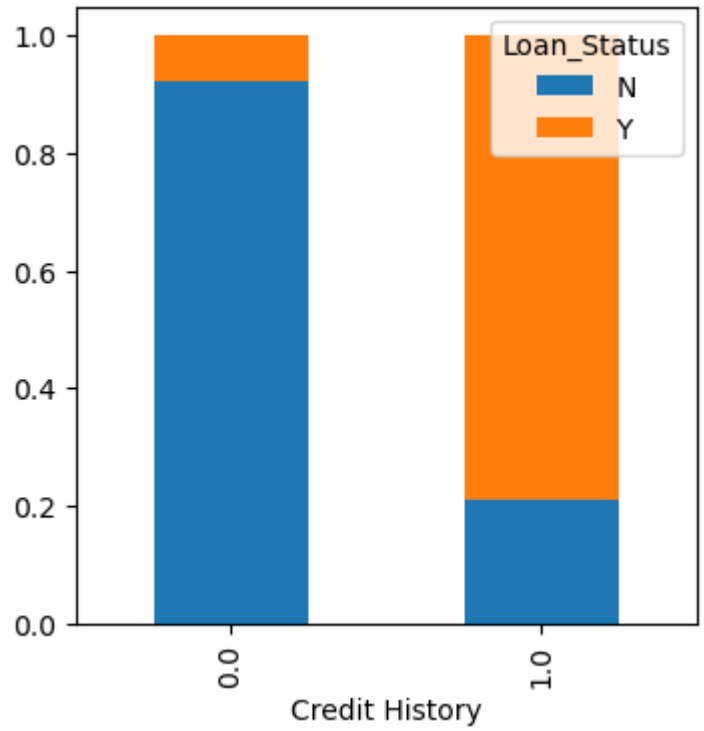
Credit_History.div(Credit_History.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(4,4))

plt.show()

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

plt.show()





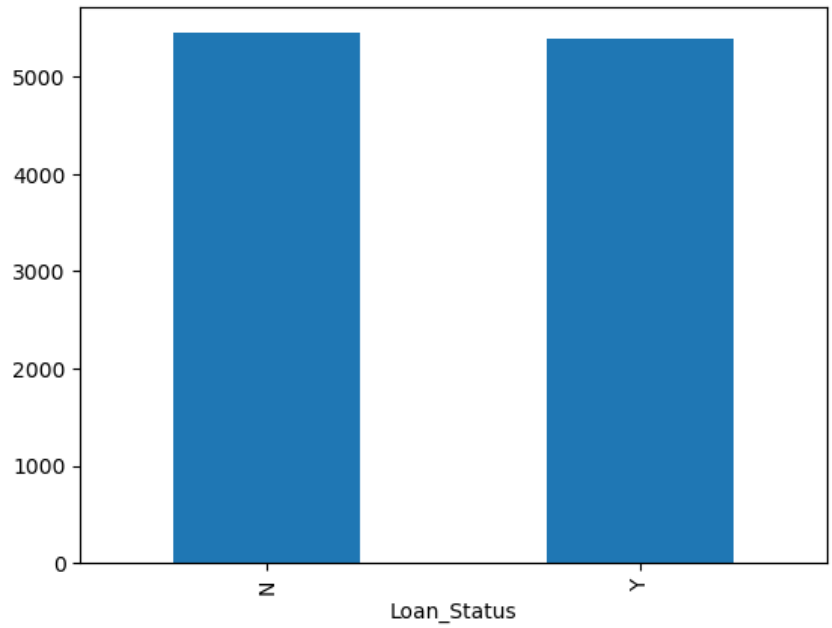
- It seems people with a credit history of 1 are more likely to get their loans approved.
- The proportion of loans getting approved in semi-urban areas is higher as compared to that in rural or urban areas.

Now let's visualize numerical independent variables with respect to the target variable:-

Numerical Independent Variable vs Target Variable:-

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:-

df.groupby('Loan_Status')['Applicant Income'].mean().plot.bar()

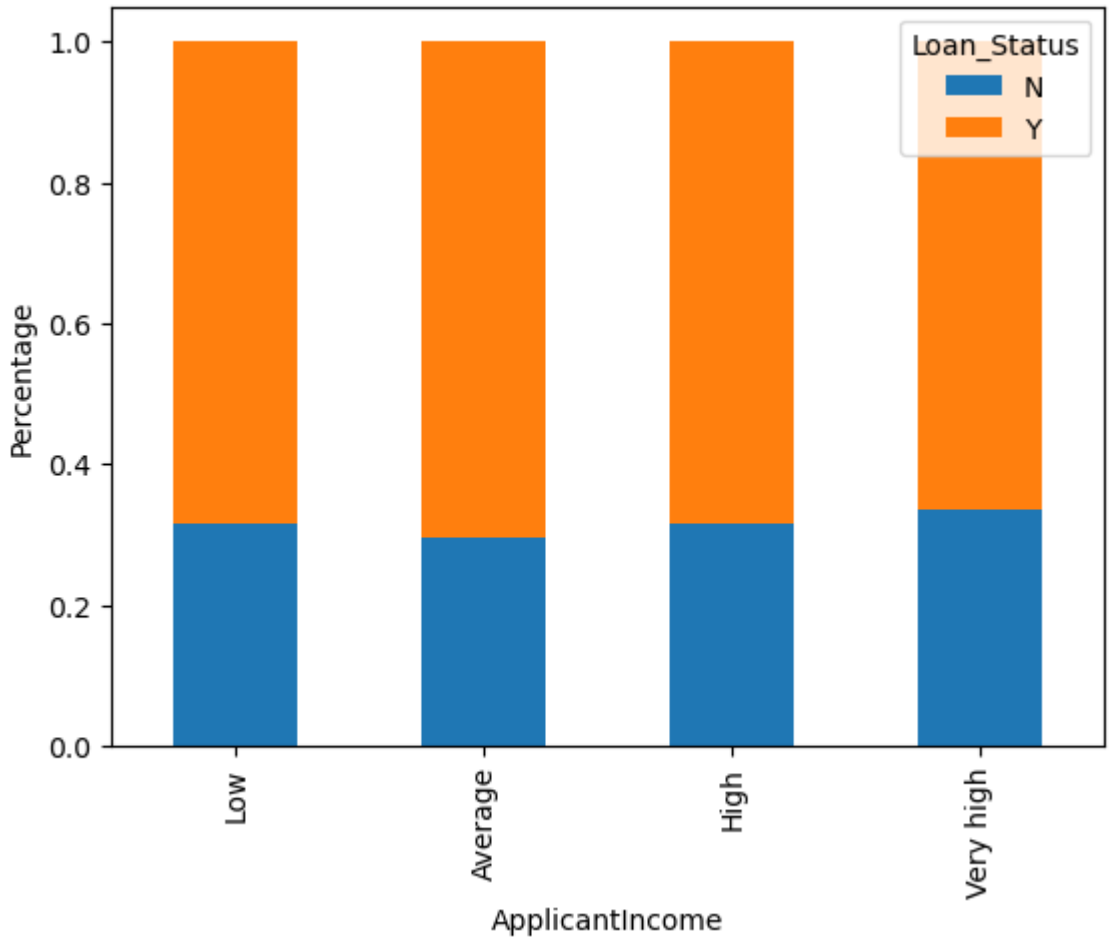


Here the y-axis represents the mean applicant income. We don't see any change in the mean income.

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:-

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

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

df['Income_bin']=pd.cut(df['Applicant Income'],bins,labels=group)

Income_bin=pd.crosstab(df['Income_bin'],df['Loan_Status'])

Income_bin.div(Income_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)
```

```
plt.xlabel('ApplicantIncome')
P = plt.ylabel('Percentage')
```

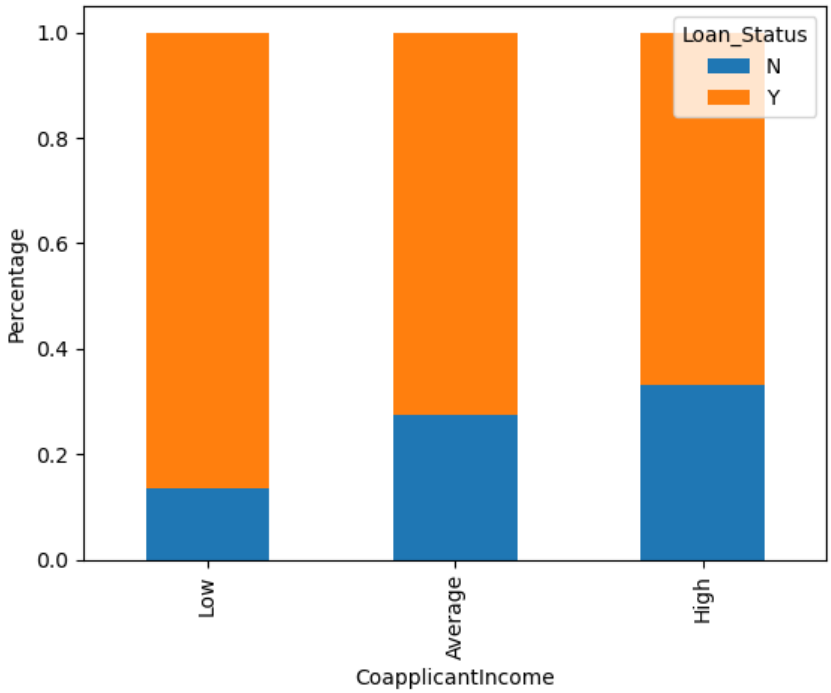


It can be inferred that Applicant's income does not affect the chances of loan approval which contradicts our hypothesis in which we assumed that if the applicant's income is high the chances of loan approval will also be high.

We will analyze the applicant's income and loan amount variable in a similar manner:-

```
bins=[0,1000,3000,42000]
group=['Low','Average','High']
df['Coapplicant_Income_bin']=pd.cut(df['CoapplicantIncome'],bins,labels=group)
Coapplicant_Income_bin=pd.crosstab(df['Coapplicant_Income_bin'],df['Loan_Status'])
```

```
Coapplicant_Income_bin.div(Coapplicant_Income_bin.sum(1).astype(float),
axis=0).plot(kind="bar", stacked=True)

plt.xlabel('CoapplicantIncome')

P = plt.ylabel('Percentage')
```

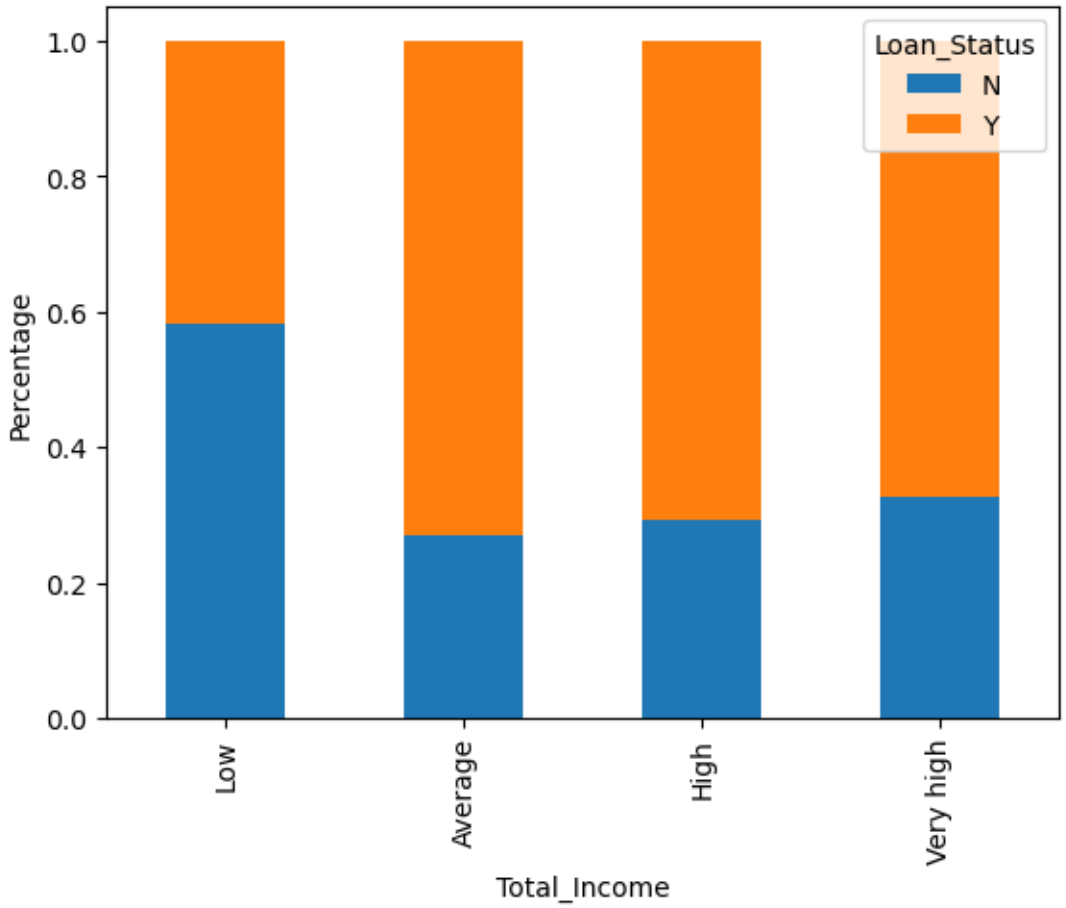


It shows that if co-applicants income is less the chances of loan approval are high. But this does not look right. The possible reason behind this may be that most of the applicants don't have any co-applicant so the co-applicant income for such applicants is 0 and hence the loan approval is not dependent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

Let us combine the Applicant Income and Co-applicant Income and see the combined effect of Total Income on the Loan_Status:-

```
df['Total_Income']=df['Applicant Income']+df['CoapplicantIncome']

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

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

df['Total_Income_bin']=pd.cut(df['Total_Income'],bins,labels=group)

Total_Income_bin=pd.crosstab(df['Total_Income_bin'],df['Loan_Status'])
```

```
Total_Income_bin.div(Total_Income_bin.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True)

plt.xlabel('Total_Income')

P = plt.ylabel('Percentage')
```

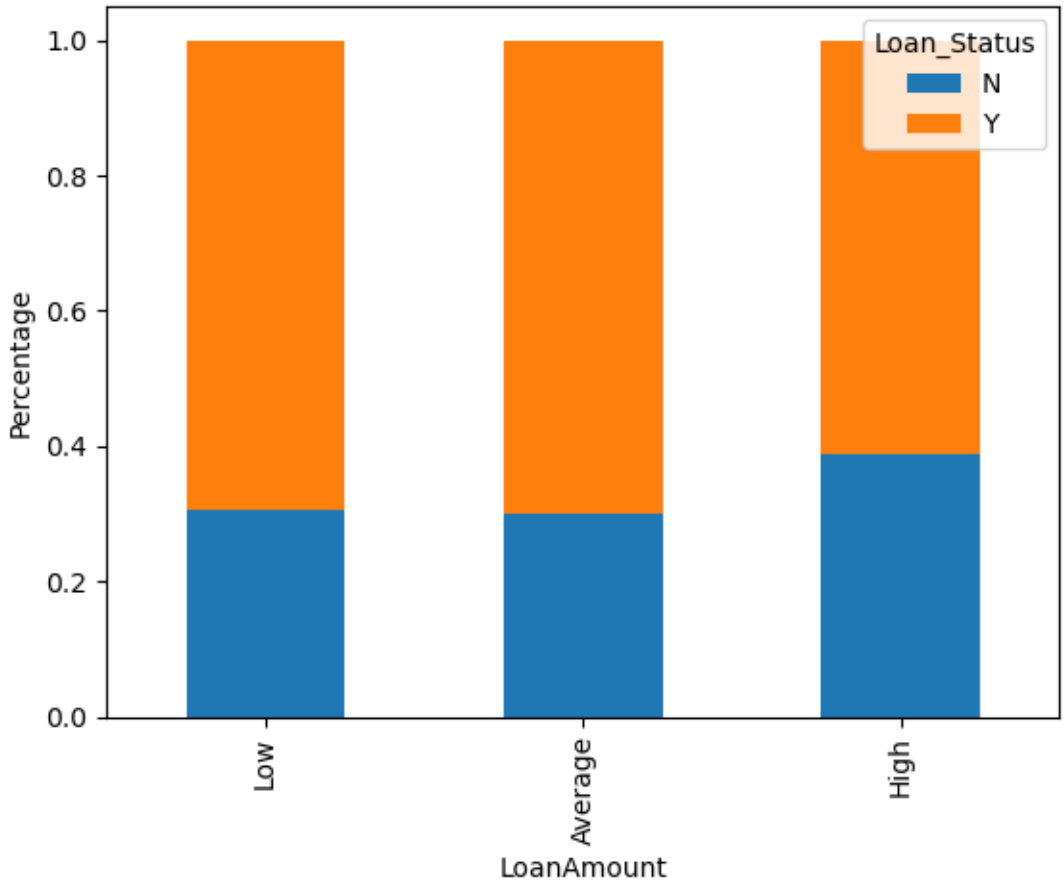


We can see that Proportion of loans getting approved for applicants having low Total_Income is very less as compared to that of applicants with Average, High, and Very High Income.

Let’s visualize the Loan amount variable:-

```
bins=[0,100,200,700]

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

df['LoanAmount_bin']=pd.cut(df['Loan_Amount'],bins,labels=group)

LoanAmount_bin=pd.crosstab(df['LoanAmount_bin'],df['Loan_Status'])
```

```
LoanAmount_bin.div(LoanAmount_bin.sum(1).astype(float), axis=0).plot(kind="bar",
stacked=True)

plt.xlabel('LoanAmount')

P = plt.ylabel('Percentage')
```



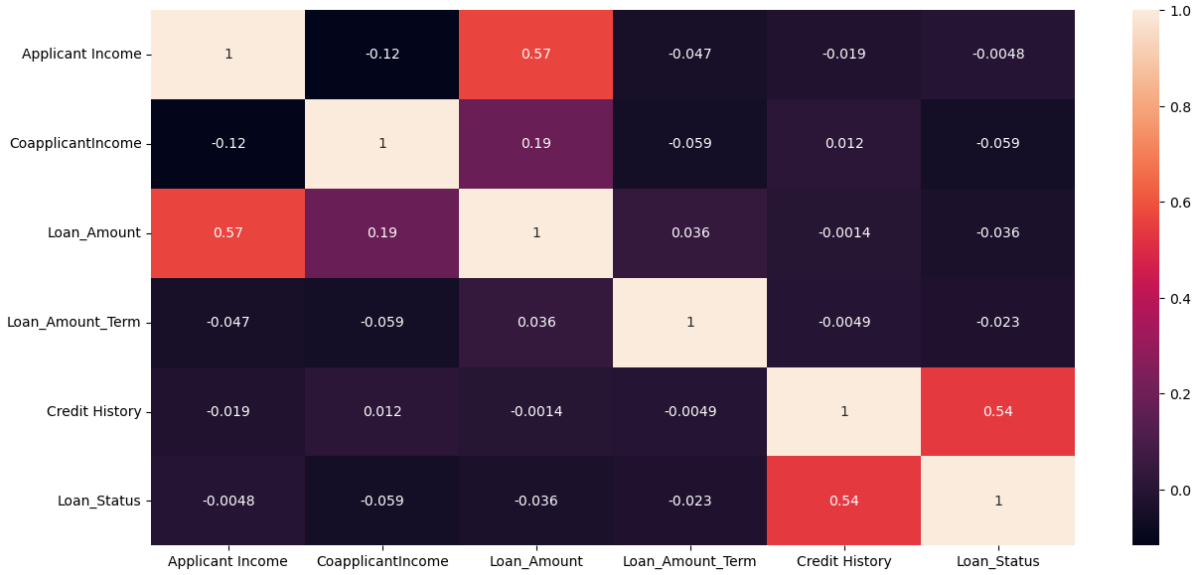
It can be seen that the proportion of approved loans is higher for Low and Average Loan Amounts as compared to that of High Loan Amounts which supports our hypothesis which considered that the chances of loan approval will be high when the loan amount is less.

```
df=df.drop(['Income_bin', 'Coapplicant_Income_bin',

 'LoanAmount_bin', 'Total_Income_bin', 'Total_Income'], axis=1)

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

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

df['Loan_Status'].replace('N', 0,inplace=True)

df['Loan_Status'].replace('Y', 1,inplace=True)
```

Now let’s 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 colors mean their correlation is more:-

#checking for correlation in the dataset:-

```
cor_mat=df.corr()

fig=plt.figure(figsize=(15,7))

sns.heatmap(cor_mat,annot=True)
```



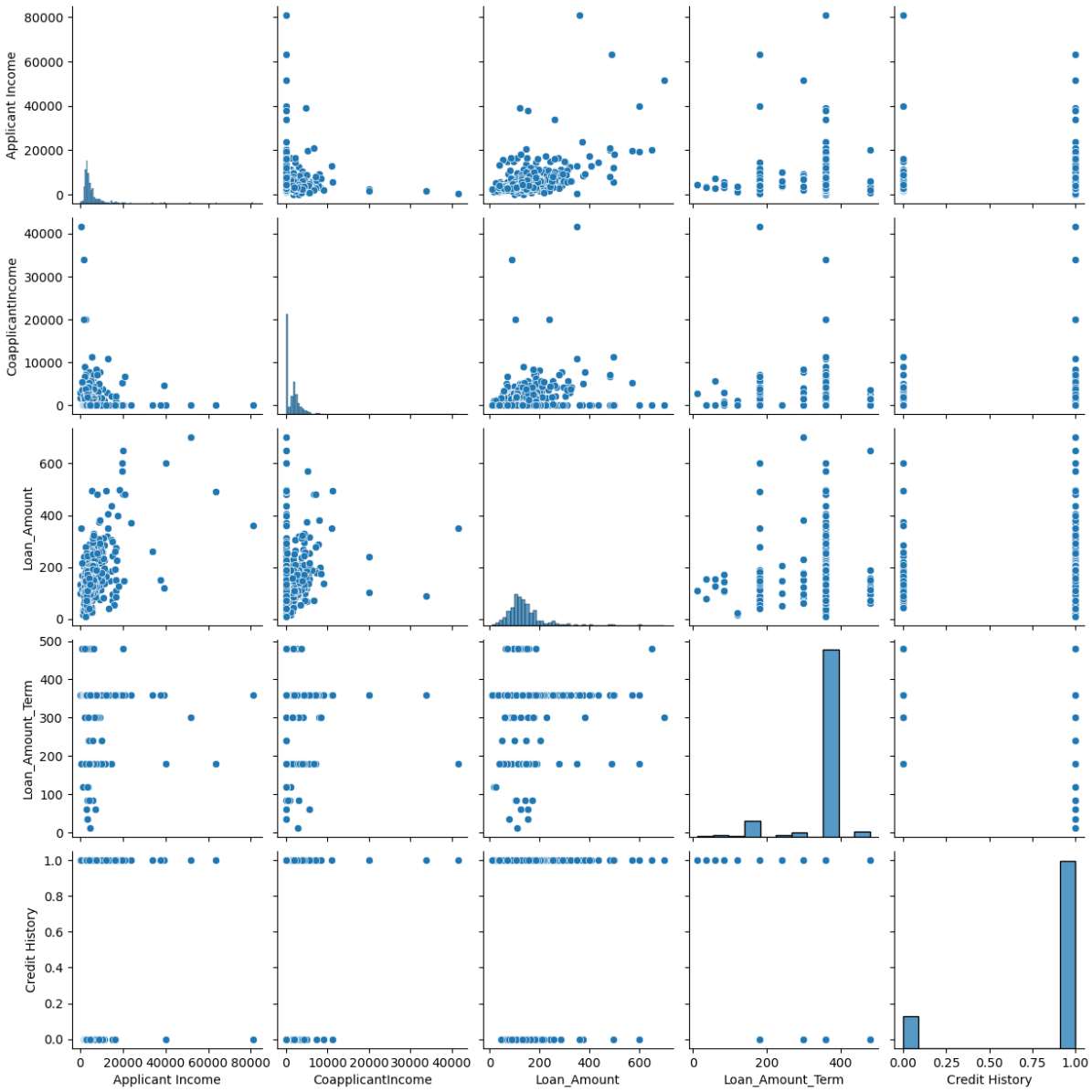
We see that the most correlated variables are (ApplicantIncome – LoanAmount) and (Credit_History – Loan_Status). LoanAmount is also correlated with CoapplicantIncome.

Pairplot:

#Using the PairPlot:

```
sns.pairplot(df)
```

plt.show()



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 adversely affect the model performance.

#checking null values in dataset:-

print(df.isna().sum())

```
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
Applicant Income      0
CoapplicantIncome     0
Loan_Amount          21
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.

- 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.

df['Gender'].fillna(df['Gender'].mode()[0], inplace=True)

df['Married'].fillna(df['Married'].mode()[0], inplace=True)

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

df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace=True)

df['Credit_History'].fillna(df['Credit_History'].mode()[0], inplace=True)

df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].mode()[0], inplace=True)

df['LoanAmount'].fillna(df['LoanAmount'].median(), inplace=True)

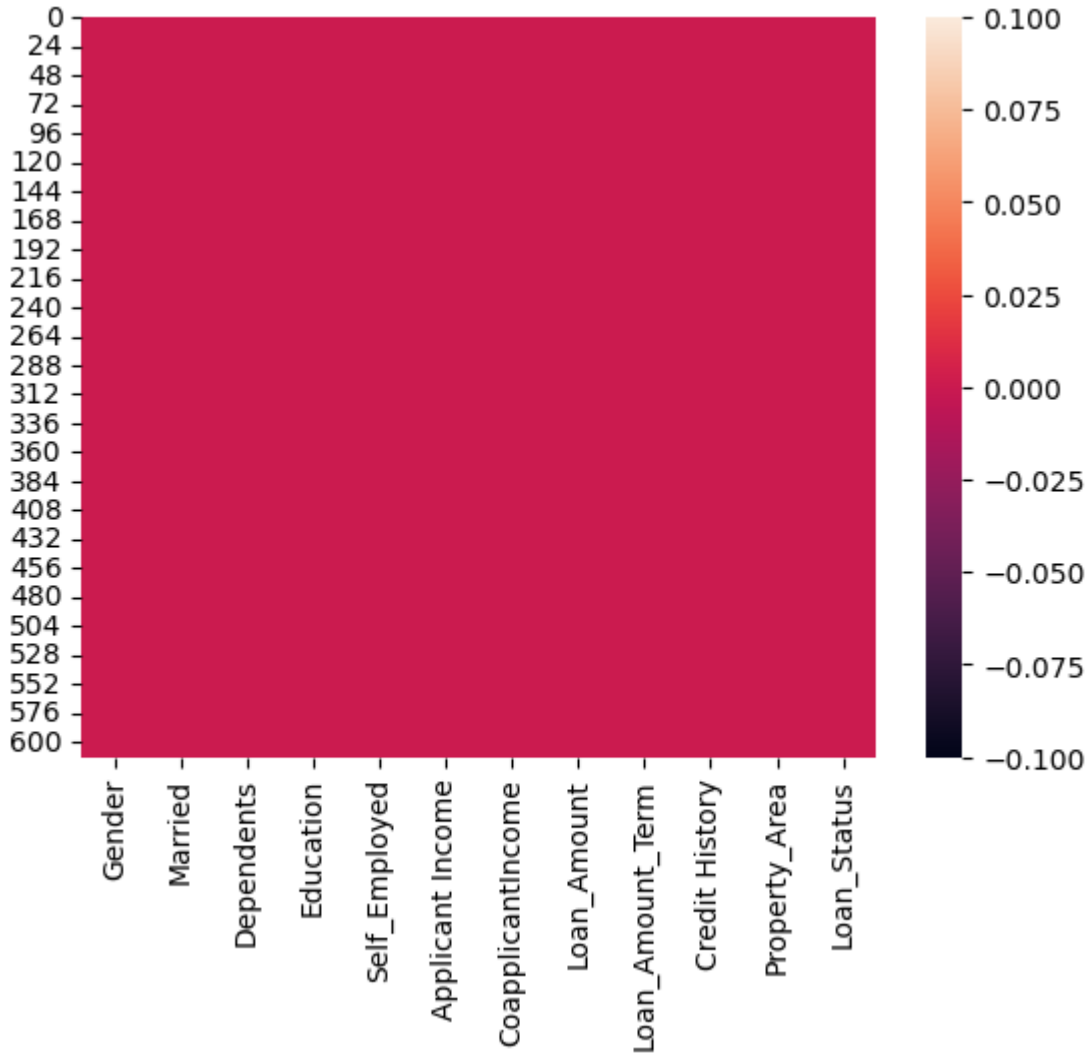
Now let's check whether all the missing values are filled in the dataset:-

print(df.isna().sum())

```
Gender                0
Married               0
Dependents            0
Education             0
Self_Employed         0
Applicant Income      0
CoapplicantIncome     0
Loan_Amount           0
Loan_Amount_Term      0
Credit History        0
Property_Area         0
Loan_Status           0
dtype: int64
```

Heatmap For Null Values:

```
sns.heatmap(df.isnull())
```



Outlier Treatment:

As we saw earlier in univariate analysis, LoanAmount contains outliers so we have to treat them as the presence of outliers affects the distribution of the data. Let's examine what can happen to a data set with outliers. For the sample data set:

1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4

We find the following: mean, median, mode, and standard deviation

Mean = 2.58

Median = 2.5

Mode = 2

Standard Deviation = 1.08

If we add an outlier to the data set:

1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 400

The new values of our statistics are:

Mean = 35.38

Median = 2.5

Mode = 2

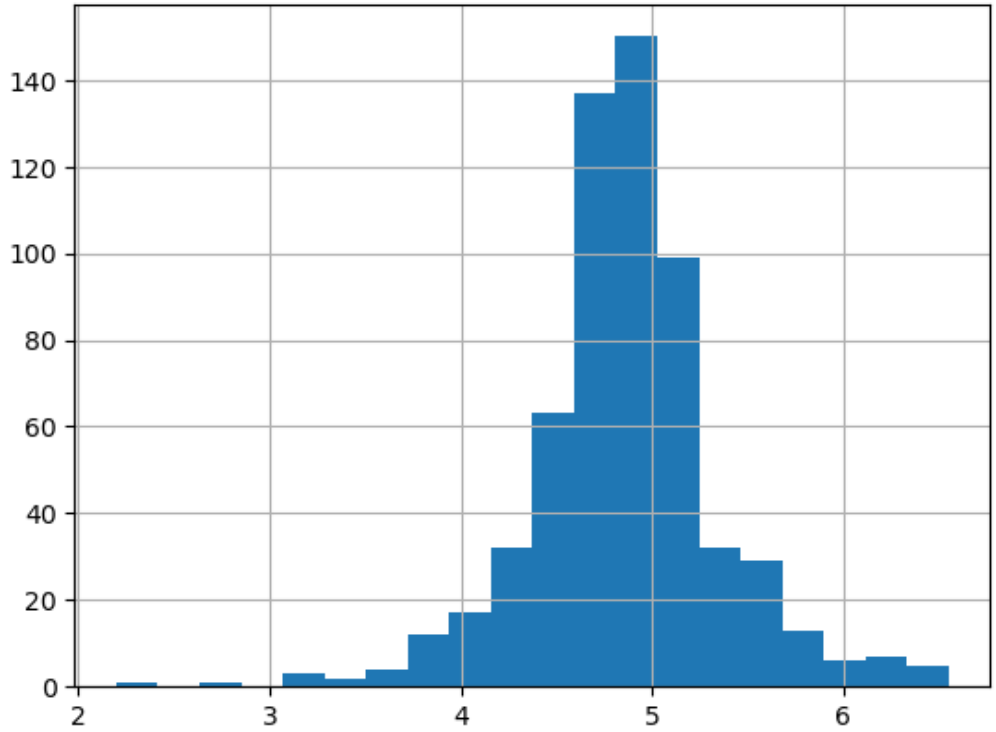
Standard Deviation = 114.74

It can be seen that having outliers often has a significant effect on the mean and standard deviation and hence affecting the distribution. We must take steps to remove outliers from our data sets.

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 the normal distribution.

Let’s visualize the effect of log transformation. We will do similar changes to the test data simultaneously:-

```
df['Loan Amount_log'] = np.log(df['Loan_Amount'])
df['Loan Amount_log'].hist(bins=20)
```



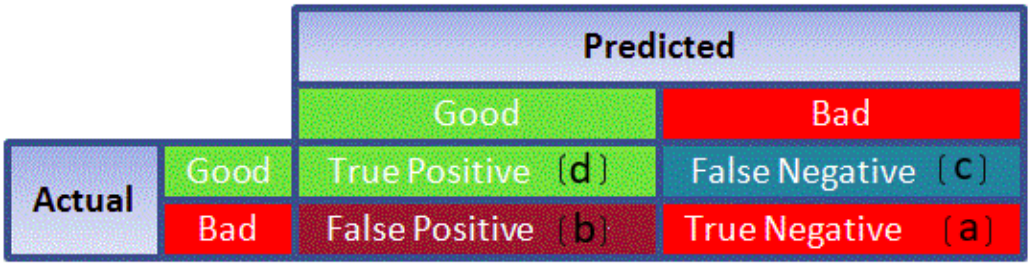
Now the distribution looks much closer to normal and the effect of extreme values has been significantly subsided. Let’s build a logistic regression model and make predictions for the test dataset.

Evaluation Metrics for Classification:-

The process of model building is not complete without the evaluation of model performance. Suppose we have the predictions from the model, how can we decide whether the predictions are accurate? We can plot the results and compare them with the actual values, i.e. calculate the distance between the predictions and actual values. The lesser this distance more accurate will be the predictions. Since this is a

classification problem, we can evaluate our models using any one of the following evaluation metrics:

- **Accuracy:-** Let us understand it using the confusion matrix which is a tabular representation of Actual vs Predicted values. This is what a confusion matrix looks like:



- True Positive – Targets which are actually true(Y) and we have predicted them as true(Y)
- True Negative – Targets that are actually false(N) and we have predicted them as false(N)
- False Positive – Targets that are actually false(N) but we have predicted them as true(T)
- False Negative – Targets that are actually true(T) but we have predicted them as false(N)

Using these values, we can calculate the accuracy of the model. The accuracy is given by:-

$$\frac{\text{True Positive + True Negatives}}{\text{True Positive + True Negatives + False Positives + False Negatives}}$$

- **Precision:-**: It is a measure of correctness achieved in true prediction i.e. of observations labeled as true, how many are actually labeled true

$$\text{Precision} = TP / (TP + FP)$$

- **Recall(Sensitivity**):- It is a measure of actual observations which are predicted correctly i.e. how many observations of true class are labeled correctly. It is also known as 'Sensitivity'.

  Recall = TP / (TP + FN)

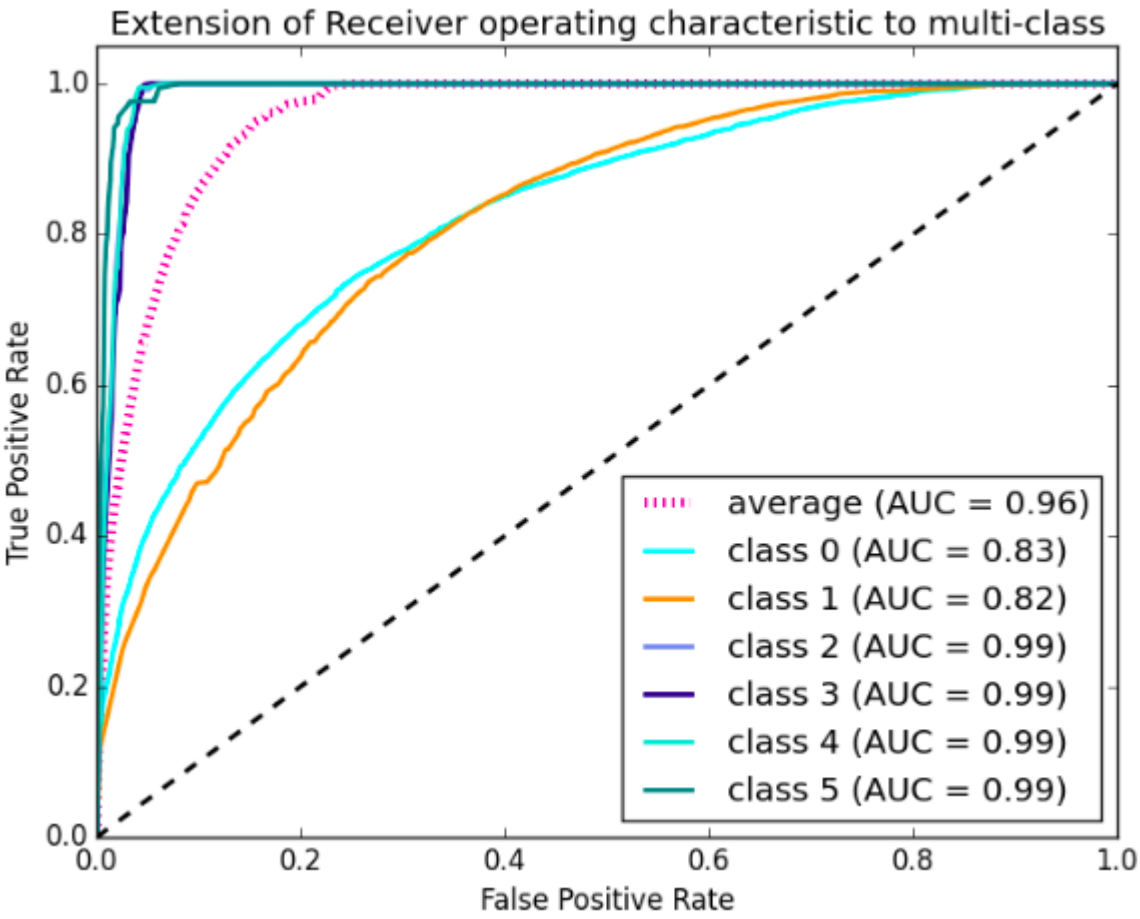
- **Specificity:**- It is a measure of how many observations of false class are labeled correctly.

  Specificity = TN / (TN + FP)

Specificity and Sensitivity play a crucial role in deriving the ROC curve.

- ROC curve
- Receiver Operating Characteristic(ROC) summarizes the model's performance by evaluating the trade-offs between true positive rate (sensitivity) and false-positive rate(1- specificity).
- The area under the curve (AUC), referred to as an index of accuracy(A) or concordance index, is a perfect performance metric for the ROC curve. Higher the area under the curve, the better the prediction power of the model.

This is what a ROC curve looks like:



- The area of this curve measures the ability of the model to correctly classify true positives and true negatives. We want our model to predict the true classes as true and false classes as false.
- So it can be said that we want the true positive rate to be 1. But we are not concerned with the true positive rate only but the false positive rate too. For example in our problem, we are not only concerned about predicting the Y classes as Y but we also want N classes to be predicted as N.
- We want to increase the area of the curve which will be maximum for classes 2,3,4 and 5 in the above example.
- For class 1 when the false positive rate is 0.2, the true positive rate is around 0.6. But for class 2 the true positive rate is 1 at the same false-positive rate. So, the AUC for class 2 will be much more as compared to the AUC for class 1. So, the model for class 2 will be better.
- The class 2,3,4 and 5 models will predict more accurately as compared to the class 0 and 1 models as the AUC is more for those classes.

Catagorical Data Handling:-

Apply label encoder:-

#converting data to numerical type using LabelEncoder:
from sklearn.preprocessing import LabelEncoder
label=LabelEncoder()

label.fit(df.Gender.drop_duplicates())
df.Gender= label.transform(df.Gender)

label.fit(df.Married.drop_duplicates())
df.Married= label.transform(df.Married)

label.fit(df.Education.drop_duplicates())
df.Education= label.transform(df.Education)

label.fit(df.Self_Employed.drop_duplicates())
df.Self_Employed= label.transform(df.Self_Employed)

label.fit(df.Loan_Status.drop_duplicates())
df.Loan_Status= label.transform(df.Loan_Status)

df.dtypes

```
Gender                int32
Married               int32
Dependents           object
Education             int32
Self_Employed         int32
Applicant Income      int64
CoapplicantIncome   float64
Loan_Amount         float64
Loan_Amount_Term    float64
Credit History      float64
Property_Area        object
Loan_Status           int32
dtype: object
```

Apply ordinal encoder:-

```
from sklearn.preprocessing import OrdinalEncoder

enc=OrdinalEncoder()
df['Dependents']=enc.fit_transform(df[['Dependents']])
df['Property_Area']=enc.fit_transform(df[['Property_Area']])
```

df.dtypes

```
Gender                int32
Married               int32
Dependents          float64
Education             int32
Self_Employed         int32
Applicant Income      int64
CoapplicantIncome   float64
Loan_Amount         float64
Loan_Amount_Term    float64
Credit History      float64
Property_Area       float64
Loan_Status           int32
dtype: object
```

Checking for skewness:

df_new.skew()

```
Gender              -1.620181
Married             -0.635432
Dependents           1.049515
Education            1.303909
Self_Employed        2.249864
Applicant Income     2.149534
CoapplicantIncome    1.348577
Loan_Amount          1.113223
Loan_Amount_Term    -2.096104
Credit History      -1.973183
Property_Area       -0.052313
Loan_Status         -0.819913
dtype: float64
```

Skewness Handling:

#Removing skewness of the data:

```
from sklearn.preprocessing import PowerTransformer
PT=PowerTransformer()
for i in df_new.columns:
    if abs(df_new.loc[:,i].skew())>0.5:
        df_new.loc[:,i]=PT.fit_transform(df_new.loc[:,i].values.reshape(-1,1))
df_new.skew()
```

```
Gender              -1.620181
Married             -0.635432
Dependents           0.476020
Education            1.303909
Self_Employed        2.249864
Applicant Income     0.027642
CoapplicantIncome   -0.195199
Loan_Amount          0.047431
Loan_Amount_Term     0.725684
Credit History      -1.973183
Property_Area       -0.052313
Loan_Status         -0.819913
dtype: float64
```

5] Building Machine Learning Models:-

Let us make our first model predict the target variable. We will start with Logistic Regression which is used for predicting binary outcomes.

- Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables.
- Logistic regression is an estimation of the Logit function. The logit function is simply a log of odds in favor of the event.
- This function creates an S-shaped curve with the probability estimate, which is very similar to the required stepwise function

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 tools 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 training dataset and save it in another dataset:-

```
x=df_new.drop('Loan_Status',axis=1) #List of all feature

y=df_new['Loan_Status'] #Label


print(x.shape)

print(y.shape)
```

```
(576, 11)
(576,)
```

Feature Scaling:-

```
scaler=StandardScaler()
X=pd.DataFrame(scaler.fit_transform(x),columns=x.columns)
X
```

Multicollinearity using Variance_inflation_factor:-

```
import statsmodels
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif['vif']=[variance_inflation_factor(X.values,i)for i in range(X.shape[1])]
vif['Features']=X.columns
vif
```

Balancing the target column using SMOTE:-

```
from collections import Counter
from imblearn.over_sampling import SMOTE
```

```
sm=SMOTE()
print('unbalanced data: ',Counter(y))
X_sm,y_sm=sm.fit_resample(X,y)
print('balanced data: ',Counter(y_sm))
```

```
unbalanced data:  Counter({1: 397, 0: 179})
balanced data:  Counter({0: 397, 1: 397})
```

We can train the model on this training 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.

We will use the train_test_split function from sklearn to divide our train dataset. So first, let us import train_test_split:

#Creating the data sets:

```
X_train,X_test,y_train,y_test=train_test_split(X_sm,y_sm,test_size=0.25,random_state=42,shuffle=True)
```

The dataset has been divided into training and validation parts. Let us import LogisticRegression and other some classifiers and accuracy_score from sklearn and fit the logistic regression model and other classifiers models :-

#import libraries:-

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score

from sklearn.svm import SVC
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import cross_val_score,GridSearchCV
```

- # Logistic Regression:-

```
LR= LogisticRegression()

#training the model:-
LR.fit(X_train,y_train)

#Let’s predict the Loan_Status for the validation set and calculate its accuracy.:-

pred=LR.predict(X_test)

# Let us calculate how accurate our predictions are by calculating the accuracy:

acc_score=(accuracy_score(y_test,pred))
print("Accuracy_score:",acc_score)

#confusion matrix:-
print('Confusion matrix:\n',confusion_matrix(y_test,pred))

#classification report:-
class_report=classification_report(y_test,pred)
print('n\Classification Report:\n',class_report)

#cross validation score:-
cv_score=(cross_val_score(LR,X,y,cv=5).mean())
print("Cross Validation Score:",cv_score)

#Result of accuracy minus cv scores:-
```

result=acc_score - cv_score

print("\n Accuracy Score - cross validation score is",result)

```
Accuracy_score: 0.7934673366834171
Confusion matrix:
 [[51 51]
 [10 87]]
n\Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.50      0.73       102
           1       0.73      0.90      0.84        97

    accuracy                           0.79       199
   macro avg       0.83      0.70      0.78       199
weighted avg        0.84      0.69      0.78        199

Cross Validation Score: 0.8194602698650673

 Accuracy Score - cross validation score is -0.12599293318165017
```

So our predictions are almost 80% accurate, i.e. we have identified 80% of the loan status correctly.

Logistic Regression Using Stratified k-folds Cross-validation:-

To check how robust our model is to unseen data, we can use Validation. It is a technique that involves reserving a particular sample of a dataset on which you do not train the model. Later, you test your model on this sample before finalizing it. Some of the common methods for validation are listed below:

- The validation set approach
- k-fold cross-validation
- Leave one out cross-validation (LOOCV)

In this section we will learn about k-fold cross-validation.

- ## 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 Co-applicant Income. If the total income is high, the chances of loan

approval might also be high.

- **EMI:**- EMI is the monthly amount to

be paid by the applicant to repay the loan. The 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. The idea behind creating

this variable is that if this value is high, the chances are high that a

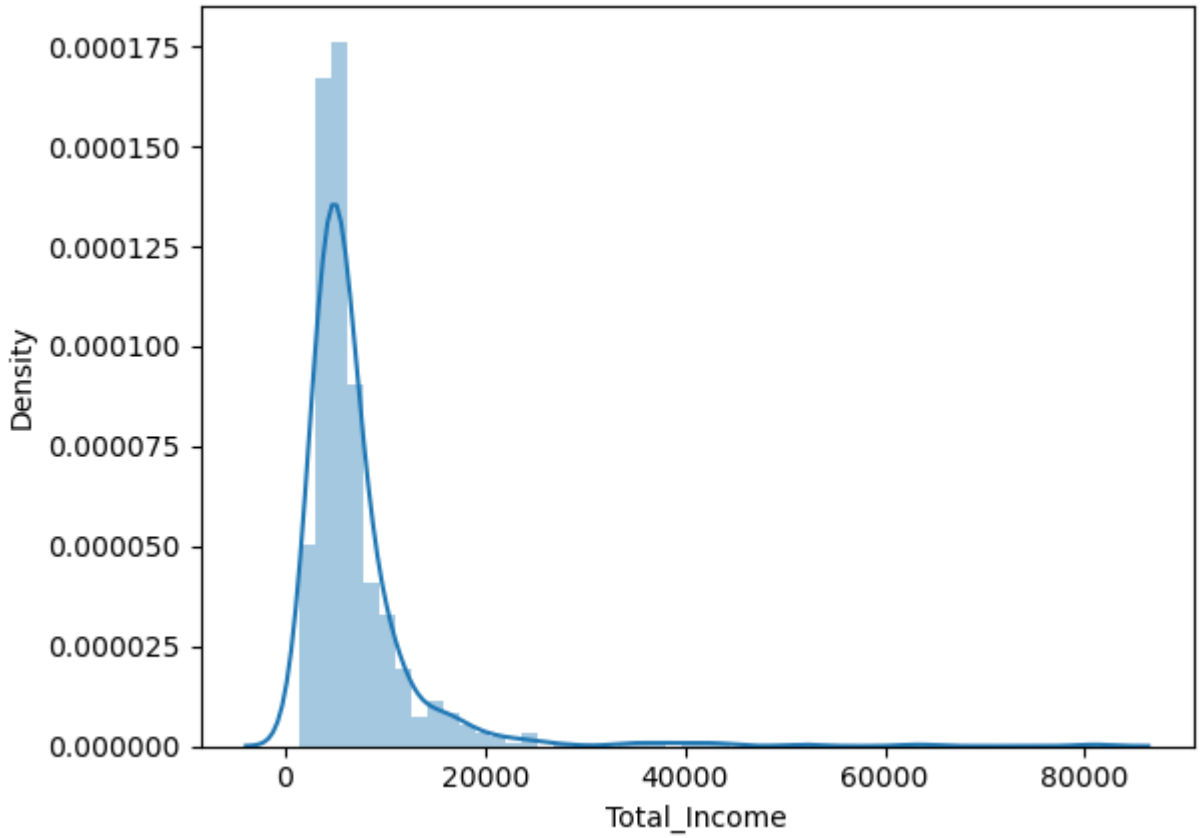
person will repay the loan and hence increasing the chances of loan

approval.

```
df['Total_Income']=df['Applicant Income']+df['CoapplicantIncome']
```

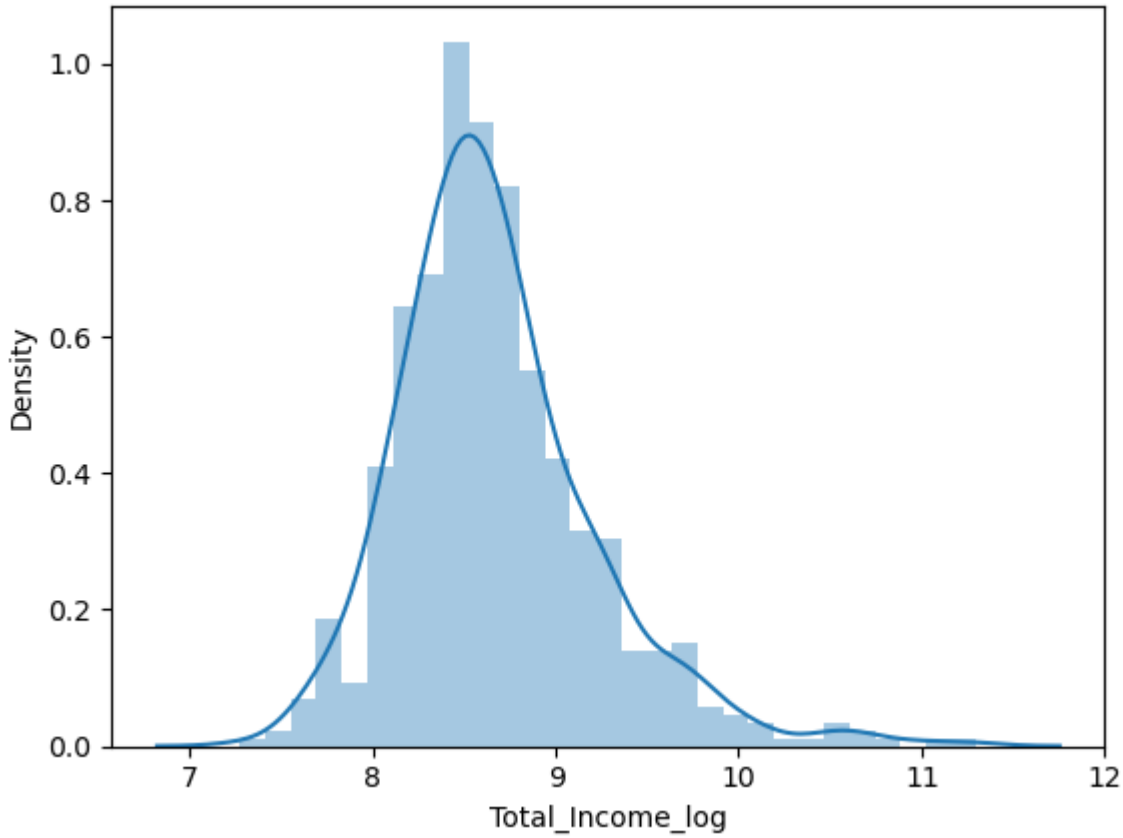
Let's check the distribution of Total Income:-

```
sns.distplot(df['Total_Income']);
```



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

```
df['Total_Income_log'] = np.log(df['Total_Income'])

sns.distplot(df['Total_Income_log']);
```

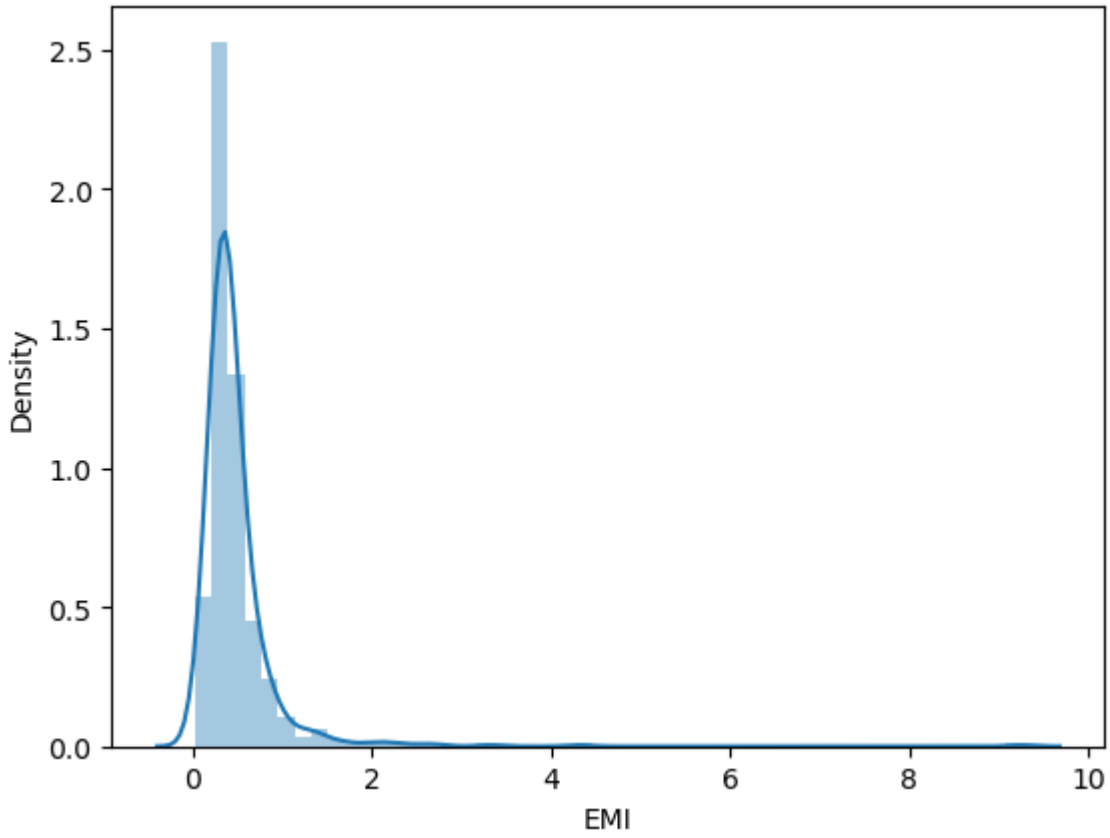


Now the distribution looks much closer to normal and the effect of extreme values has been significantly subsided. Let's create the EMI feature now:-

```
df['EMI']=df['Loan_Amount']/df['Loan_Amount_Term']
```

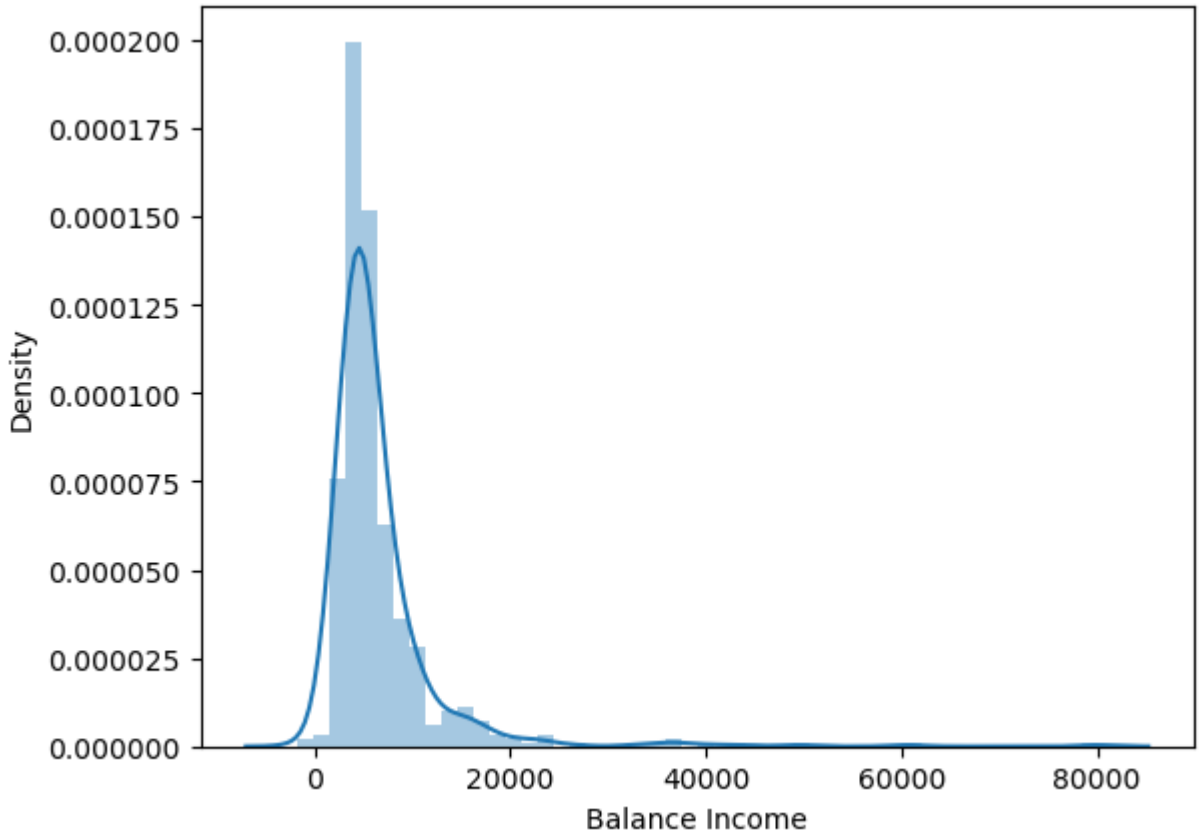
Let's check the distribution of the EMI variable:-

```
sns.distplot(df['EMI']);
```



Let us create the Balance Income feature now and check its distribution:-

```
df['Balance Income']=df['Total_Income']-(df['EMI']*1000)
# Multiply with 1000 to make the units equal
sns.distplot(df['Balance Income']);
```



We will build the following models in this section.

- Decision Tree
- Random Forest
- XGBoost

• Decision Tree Classifier:-

A decision tree is a type of supervised learning algorithm(having a pre-defined target variable) that is mostly used in classification problems. In this technique, we split the population or sample into two or more homogeneous sets(or sub-populations) based on the most significant splitter/differentiator in input variables.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant

sub-nodes. In other words, we can say that purity of the node increases with respect to the target variable.

```
DTC= DecisionTreeClassifier()


#training the model:-

DTC.fit(X_train,y_train)

pred=DTC.predict(X_test)


#Checking Accuracy Score:

acc_score=(accuracy_score(y_test,pred))

print("Accuracy_score:",acc_score)


#confusion matrix:-

print('Confusion matrix:\n',confusion_matrix(y_test,pred))


#classification report:-

class_report=classification_report(y_test,pred)

print('n\Classification Report:\n',class_report)


#cross validation score:-

cv_score=(cross_val_score(DTC,X,y,cv=5).mean())

print("Cross Validation Score:",cv_score)


#Result of accuracy minus cv scores:-

result=acc_score - cv_score
```

print("\n Accuracy Score - cross validation score is",result)

```
Accuracy_score: 0.7336683417085427
Confusion matrix:
 [[71 31]
 [22 75]]
n\Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.70      0.73       102
           1       0.71      0.77      0.74        97

    accuracy                           0.73       199
   macro avg       0.74      0.73      0.73       199
weighted avg       0.74      0.73      0.73       199

Cross Validation Score: 0.7221889055472264

 Accuracy Score - cross validation score is 0.011479436161316303
```

We got an accuracy of 0.73. So let's build another model, i.e. Random Forest, a tree-based ensemble algorithm and try to improve our model by improving the accuracy.

- # Random forest classifier:-

- RandomForest is a tree-based bootstrapping algorithm wherein a certain no. of weak learners (decision trees) are combined to make a powerful prediction model.
- For every individual learner, a random sample of rows and a few randomly chosen variables are used to build a decision tree model.
- The final prediction can be a function of all the predictions made by the individual learners.
- In the case of a regression problem, the final prediction can be the mean of all the predictions.

RFC= RandomForestClassifier()

```
#training the model:-
RFC.fit(X_train,y_train)
pred=RFC.predict(X_test)

#Checking Accuracy Score:
acc_score=(accuracy_score(y_test,pred))
print("Accuracy_score:",acc_score)

#confusion matrix:-
print('Confusion matrix:\n',confusion_matrix(y_test,pred))

#classification report:-
class_report=classification_report(y_test,pred)
print('n\Classification Report:\n',class_report)

#cross validation score:-
cv_score=(cross_val_score(RFC,X,y,cv=5).mean())
print("Cross Validation Score:",cv_score)

#Result of accuracy minus cv scores:-
result=acc_score - cv_score
print("\n Accuracy Score - cross validation score is",result)
```

```
Accuracy_score: 0.8190954773869347
Confusion matrix:
 [[78 24]
 [12 85]]
n\Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.76      0.81       102
           1       0.78      0.88      0.83        97

    accuracy                           0.82       199
   macro avg       0.82      0.82      0.82       199
weighted avg       0.82      0.82      0.82       199

Cross Validation Score: 0.8038230884557722

 Accuracy Score - cross validation score is 0.01527238893116245
```

- **XGBClassifier:-**

XGBoost is a fast and efficient algorithm and has been used by the winners of many data science competitions. It's a boosting algorithm and you may refer to the below article to know more about boosting:

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.

- n_estimator: This specifies the number of trees for the model.
- max_depth: We can specify the maximum depth of a tree using this parameter.

```
from xgboost import XGBClassifier

from sklearn.naive_bayes import GaussianNB

XGB= XGBClassifier()

#training the model:-
XGB.fit(X_train,y_train)
pred=XGB.predict(X_test)

#Checking Accuracy Score:
acc_score=(accuracy_score(y_test,pred))
print("Accuracy_score:",acc_score)

#confusion matrix:-
print('Confusion matrix:\n',confusion_matrix(y_test,pred))

#classification report:-
class_report=classification_report(y_test,pred)
print('n\Classification Report:\n',class_report)
```

#cross validation score:-

cv_score=(cross_val_score(XGB,X,y,cv=5).mean())

print("Cross Validation Score:",cv_score)

#Result of accuracy minus cv scores:-

result=acc_score - cv_score

print("\n Accuracy Score - cross validation score is",result)

```
Accuracy_score: 0.7989949748743719
Confusion matrix:
 [[77 25]
 [15 82]]
n\Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.75      0.79       102
           1       0.77      0.85      0.80        97

    accuracy                           0.80       199
   macro avg       0.80      0.80      0.80       199
weighted avg       0.80      0.80      0.80       199

Cross Validation Score: 0.7761019490254872

 Accuracy Score - cross validation score is 0.02289302584888464
```

- # ExtraTreesClassifier:-

The Extra Trees classifier, also known as Extremely Randomized Trees, is an ensemble learning technique that aggregates the results of multiple de-correlated decision trees to improve predictive accuracy and control over-fitting. Here are some of its uses:

- Feature Selection: It can be used to identify the most important features from a dataset by computing the Gini importance of each feature1.
- Handling Large Datasets: It's efficient for large datasets as it works by fitting a number of randomized decision trees on various sub-samples of the dataset.
- Predictive Modeling: It can be used for both classification and regression tasks, providing a predictive model based on the input data.
- Variety of Criteria: It supports different criteria like Gini impurity, entropy, and log loss for measuring the quality of splits in the decision trees.

Extra Trees classifier is particularly useful when you have a large amount of data and need a robust model that can handle the complexity and noise in the data. It's a powerful tool in the machine learning toolkit for various practical applications.

```
from sklearn.ensemble import ExtraTreesClassifier

ETC= ExtraTreesClassifier()

#training the model:-
ETC.fit(X_train,y_train)
pred=ETC.predict(X_test)

#Checking Accuracy Score:
acc_score=(accuracy_score(y_test,pred))
print("Accuracy_score:",acc_score)

#confusion matrix:-
print('Confusion matrix:\n',confusion_matrix(y_test,pred))

#classification report:-
class_report=classification_report(y_test,pred)
print('n\Classification Report:\n',class_report)

#cross validation score:-
cv_score=(cross_val_score(ETC,X,y,cv=5).mean())
print("Cross Validation Score:",cv_score)

#Result of accuracy minus cv scores:-
result=acc_score - cv_score
print("\n Accuracy Score - cross validation score is",result)
```

```
Accuracy_score: 0.8391959798994975
Confusion matrix:
 [[83 19]
 [13 84]]
n\Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.86      0.81      0.84       102
           1       0.82      0.87      0.84        97

    accuracy                           0.84       199
   macro avg       0.84      0.84      0.84       199
weighted avg       0.84      0.84      0.84       199

Cross Validation Score: 0.7690854572713643

 Accuracy Score - cross validation score is 0.07011052262813311
```

Hyperparameter Tunning:

We will try to improve the accuracy by tuning the hyperparamet ers for this model. We will use grid search to get the optimiz ed values of hyperparameters. Grid-search is a way to select t he best of a family of hyperparameters, parametrized by a grid of parameters.

- **Applying Grid Search:-**

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import KFold,cross_val_score
from numpy import mean

#Hyper parameter Tuning on the best ML model:

par={'n_estimators':range(100,500,100),'criterion':['gini','entropy'],'max_depth':range(0,10,5),'min_samples_split':range(2,3),'min_samples_leaf':range(1,5),'max_features':['auto']}

grid=GridSearchCV(ExtraTreesClassifier(),param_grid=par,verbose=2,n_jobs=-1)

grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
                                                                Out[79]:
GridSearchCV(estimator=ExtraTreesClassifier(), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
```

```
                         'max_depth': range(0, 10, 5), 'max_features':
['auto'],
                         'min_samples_leaf': range(1, 5),
                         'min_samples_split': range(2, 3),
                         'n_estimators': range(100, 500, 100)},
             verbose=2)
```

print("Best score:",grid.best_score_)

print("Best estimator:",grid.best_estimator_)

print("Best parameters:",grid.best_params_)

```
Best score: 0.7495798319327731
Best estimator: ExtraTreesClassifier(max_depth=5, max_features='auto')
Best parameters: {'criterion': 'gini', 'max_depth': 5, 'max_features':
'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
100}
```

ETC_final=ExtraTreesClassifier(criterion='entropy',max_depth=5,min_samples_leaf=1,min_samples_split=2,max_features='auto',n_estimators=300,random_state=42)

ETC_final.fit(X_train,y_train)

predETC=ETC_final.predict(X_test)

print('accuracy:',accuracy_score(y_test,predETC))

print('confusion matrix:\n',confusion_matrix(y_test,predETC))

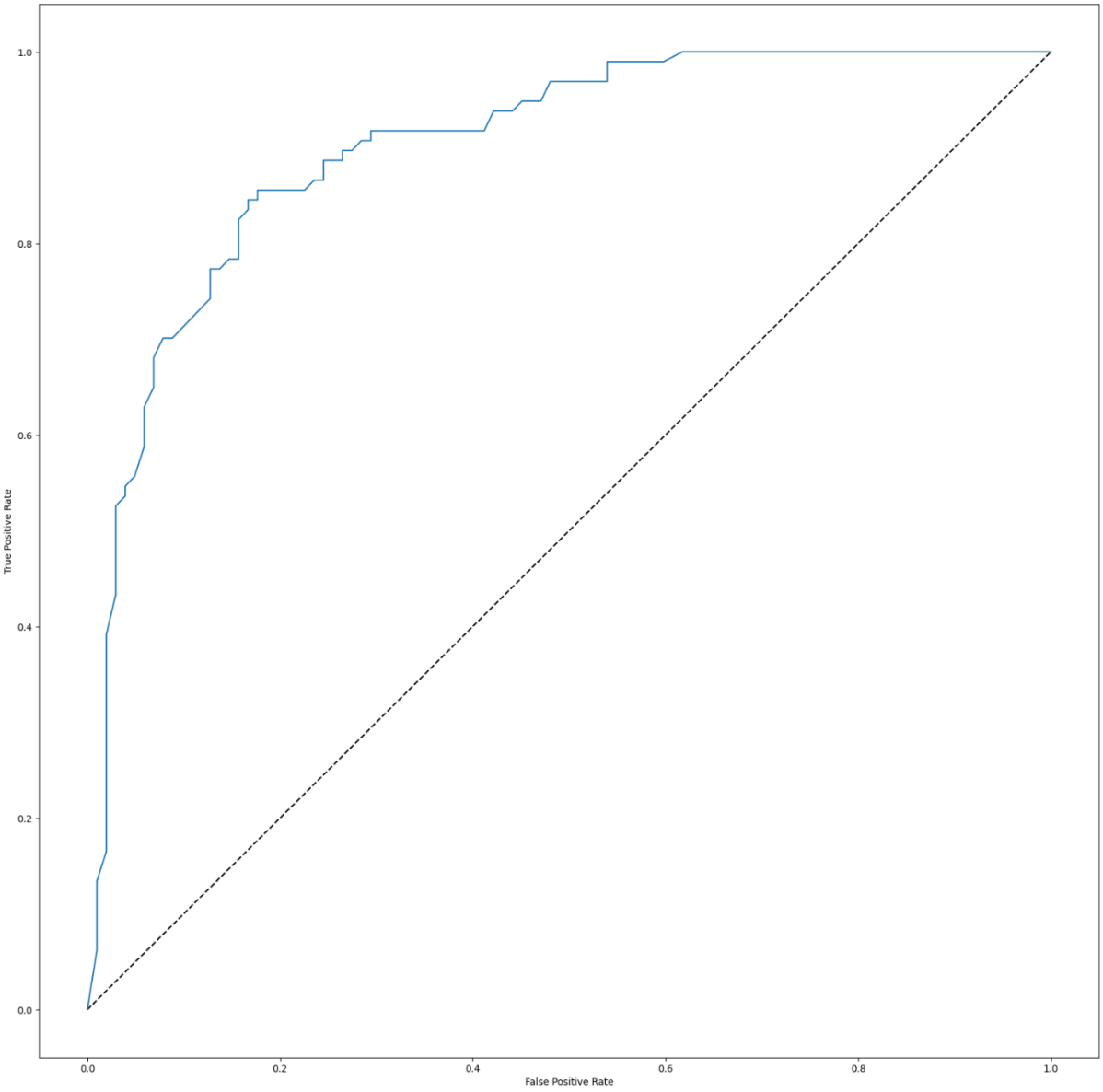
print('classification report:\n',classification_report(y_test,predETC))

```
accuracy: 0.7336683417085427
confusion matrix:
 [[50 52]
 [ 1 96]]
classification report:
              precision    recall  f1-score   support

           0       0.98      0.49      0.65       102
           1       0.65      0.99      0.78        97

    accuracy                           0.73       199
   macro avg       0.81      0.74      0.72       199
weighted avg       0.82      0.73      0.72       199
```

#After tweaking parameters we are getting the best accuracy score is 0.7336 which is less than the accuracy score of ExtraTreesClassifier(0.8391) without hyper parameter tunning,so lets save the model in by default Extratrees model.

- **AUC ROC Curve:-**

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn import metrics

#predicting the probability of having 1 in the x_test:
y_pred_prob=ETC.predict_proba(X_test)[:,1]

fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)

plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr,label='ExtraTreesClassifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.show()
auc_score=roc_auc_score(y_test,ETC.predict(X_test))
print('Score:',auc_score)
```

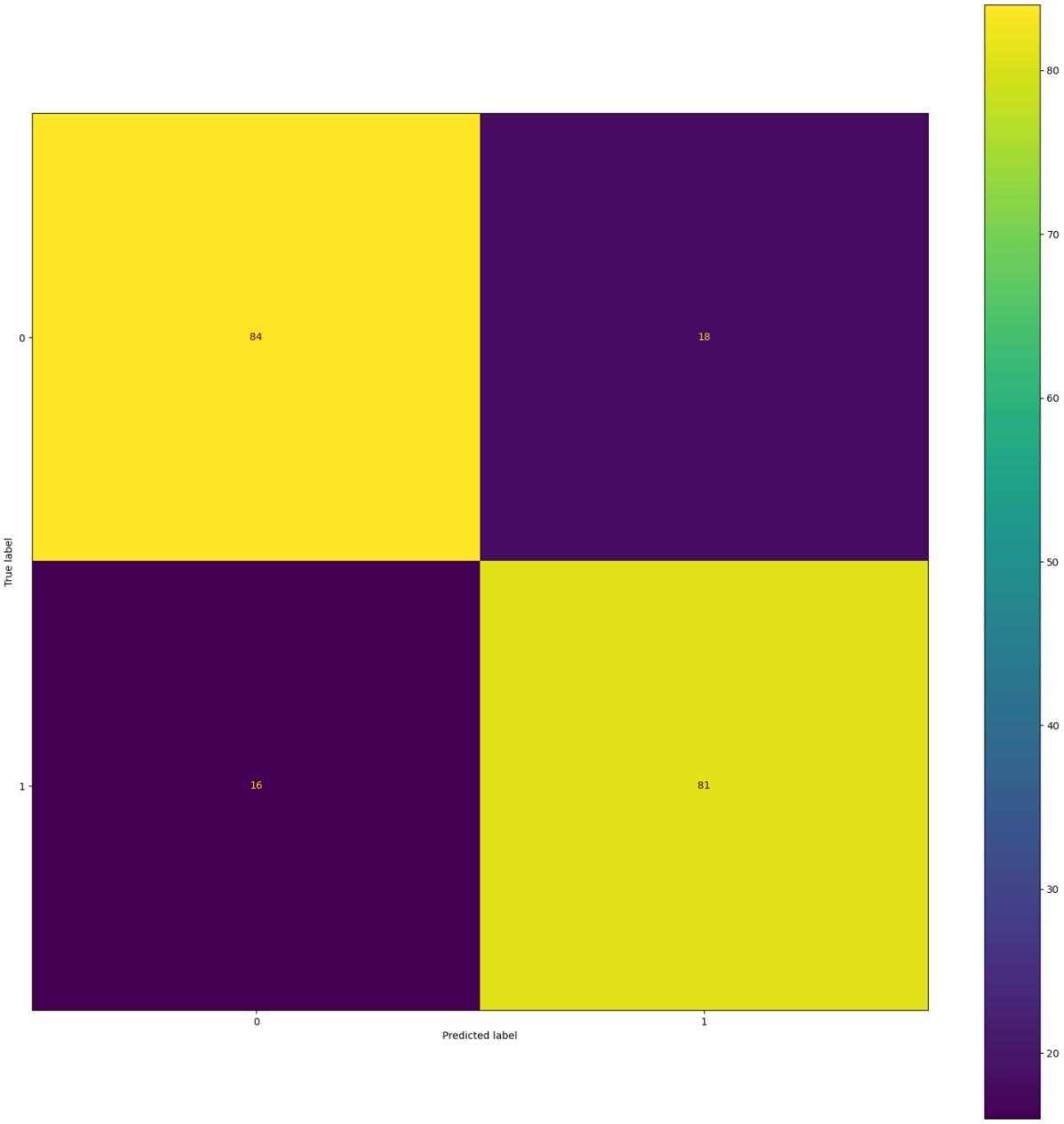


- **Confusion matrix for ExtraTreesclassifier:-**

```
ETC.fit(X_train,y_train)
y_pred=ETC.predict(X_test)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

confusion_matrix(y_test,y_pred)
```

```
array([[84, 18],
       [16, 81]], dtype=int64)
```

```
predictions=ETC.predict(X_test)
cm=confusion_matrix(y_test,predictions,labels=ETC.classes_)
disp=ConfusionMatrixDisplay(confusion_matrix=cm,display_labels=ETC.classes_)
disp.plot()
plt.show()
print(classification_report(y_test,y_pred))
```



- #With the help of above matrix we are able to understand the number of times we got the correct outputs and the number of times our ML model missed to provide the correct prediction.

Conclusion:-

- The result clearly demonstrate that the ExtraTreesClassifier algorithm outperforms the other methods,achieving the highest accuracy among all evaluated classifiers.
- RandomForest and gradient boosting also performed admirably, showing competitive performance with an accuracy close to ExtraTreesClassifier.

There are still quite a many things that can be tried to improve our models' predictions. We create and add more variables, try different models with a different subset of features and/or rows, etc. Some of the ideas are listed below

- We can train the XGBoost model using grid search to optimize its hyperparameters and improve its accuracy.
- We can combine the applicants with 1,2,3 or more dependents and make a new feature as discussed in the EDA part.
- We can also make independent vs independent variable visualizations to discover some more patterns.
- We can also arrive at the EMI using a better formula which may include interest rates as well.
- We can even try ensemble modeling (a combination of different models). To read more about ensemble techniques you can refer to these articles.

THANK YOU.