Case Study — Loan Prediction

Hello friends, this is my first machine learning project. Recently I have participated in analytics-vidya hackathon. I am here to describe how i solved the case study in a very detailed manner.

Firstly let us look through problem statement.

Problem Statement

There is a company named Dream Housing Finance that 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. However doing this manually takes a lot of time. Hence it wants to automate the loan eligibility process (real time) based on customer information

So the final thing is to identify the factors/ customer segments that are eligible for taking loan. How will the company benefit if we give the customer segments is the immediate question that arises. The solution is ….Banks would give loans to only those customers that are eligible so that they can be assured of getting the money back. Hence the more accurate we are in predicting the eligible customers the more beneficial it would be for the Dream Housing Finance Company.

TYPE OF PROBLEM:

The above problem is a clear classification problem as we need to classify whether the Loan_Status is yes or no. So this can be solved by any of the classification techniques like

1. Logistic Regression .
2. Decision Tree Algorithm.
3. Random Forest Technique.

I have mentioned only few. We will be dealing with each of techniques later in this blog.

Description about the Data Columns:

There are 2 data sets that are given. One is training data and one is testing data. It's very useful to know about the data columns before getting in to the actual problem for avoiding confusion at a later state. Now let us understand the data columns (that has been already given by the company itself) first so that we will get a glance.

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)

There are altogether 13 columns in our data set. Of them Loan_Status is the response variable and rest all are the variables /factors that decide the approval of the loan or not.

Now let us look in to the each variable and can make some assumptions.(It's just assumptions right, there is no harm in just assuming few statements)

Loan ID -> As the name suggests each person should have a unique loan ID.

Gender -> In general it is male or female. No offence for not including the third gender.

Married -> Applicant who is married is represented by Y and not married is represented as N. The information regarding whether the applicant who is married is divorced or not has not been provided. So we don't need to worry regarding all these.

Dependents -> the number of people dependent on the applicant who has taken loan has been provided.

Education -> It is either non -graduate or graduate. The assumption I can make is " The probability of clearing the loan amount would be higher if the applicant is a graduate".

Self_Employed -> As the name suggests Self Employed means , he/she is employed for himself/herself only. So freelancer or having a own business might come in this category. An applicant who is self employed is represented by Y and the one who is not is represented by N.

Applicant Income -> Applicant Income suggests the income by Applicant.So the general assumption that i can make would be "The one who earns more have a high probability of clearing loan amount and would be highly eligible for loan "

Co Applicant income -> this represents the income of co-applicant. I can also assume that " If co applicant income is higher , the probability of being eligible would be higher "

Loan Amount -> This amount represents the loan amount in thousands. One assumption I can make is that " If Loan amount is higher , the probability of repaying would be lesser and vice versa"

Loan_Amount_Term -> This represents the number of months required to repay the loan.

Credit_History -> When I googled it , I got this information.
A **credit history** is a record of a borrower's responsible repayment of debts. It suggests → 1 denotes that the credit history is good and 0 otherwise.

Property_Area -> The area where they belong to is my general assumption as nothing more is told. Here it can be three types.
Urban or Semi Urban or Rural

Loan_Status -> If the applicant is eligible for loan it's yes represented by Y else it's no represented by N**.**

Exploratory Data Analysis

Well don't get to worry about the fancy names like exploratory data analysis and all. By looking at the columns description in the above paragraph, we can make many assumptions like

1. The one whose salary is more can have a greater chance of loan approval.
2. The one who is graduate has a better chance of loan approval.
3. Married people would have a upper hand than unmarried people for loan approval .
4. The applicant who has less number of dependents have a high probability for loan approval.
5. The lesser the loan amount the higher the chance for getting loan.

Why are we doing EDA?

Like these there are many more we can assume. But one basic question you may get it …"Why are we doing all these ? Why can't we do directly modeling the data instead of knowing all these….." Well in some cases we can easily come to conclusion if we just to do EDA. Then there is no necessary for going through next models.

EDA THROUGH PYTHON

Now let me walk through the code. Firstly I just imported the necessary packages like pandas, numpy, seaborn etc. so that i can carry the necessary operations further.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Now I am going to upload or read the files/data-sets using pandas. For this we used **read_csv**

```
df= pd.read_csv('/content/sample_data/loan_dataset.csv')
df
```

Let me get the top 5 values. We can get using the head function. Hence the code would be train.head

	Loan-ID	Gender	Married	Dependents	Education	Self_Employed	ApplicationIncome	CoapplicationIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property Area	Loan Status	Loan Type
0	LP001001	Male	No	0	Graduate	No	5849	0	NAN	360	1	Urban	Y	Education Loan
1	LP001002	Male	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N	House Loan
2	LP001003	Male	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y	Health Loan
3	LP001004	Male	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y	Marriage Loan
4	LP001005	Male	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y	Education Loan
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1955	LP007060	Male	Yes	3+	Not Graduate	Yes	4009	1777	113	360	1	Urban	Y	Education Loan
1956	LP007061	Male	Yes	0	Graduate	No	4158	709	115	360	1	Urban	Y	Education Loan

train.columns *would give the list of columns of data-set train.* ***train.shape*** *gives the number of rows and columns.*

Now let us analyse the data using single variable.

```
df['Loan Status'].value_counts()
```

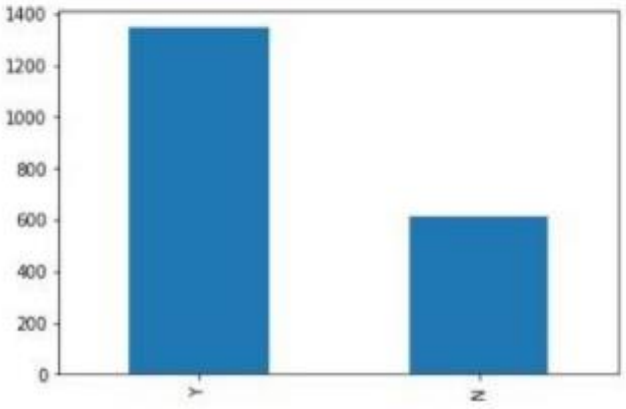
```
Y    1345
N     615
Name: Loan Status, dtype: int64
```

```
df['Loan Status'].value_counts(normalize=True)
```

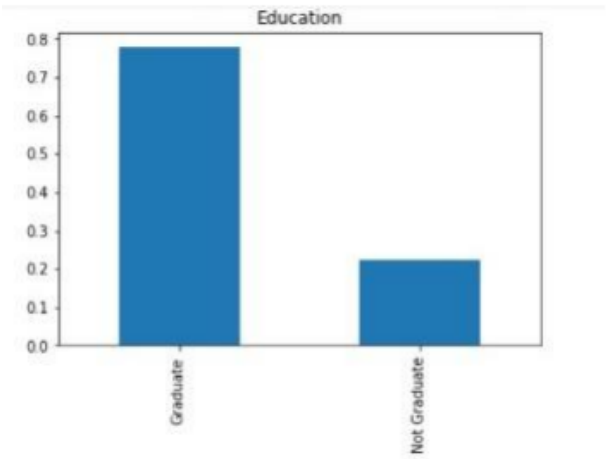
```
Y    0.686224
N    0.313776
Name: Loan Status, dtype: float64
```

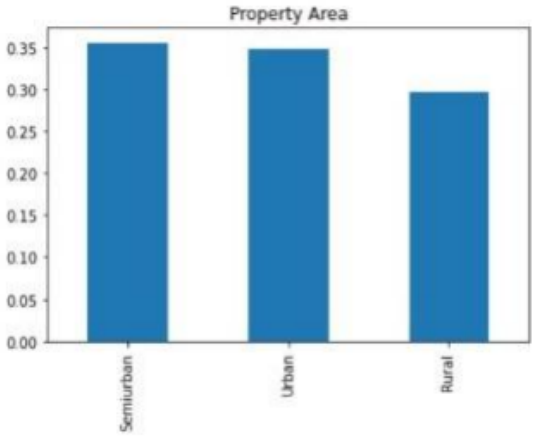
```
df['Loan Status'].value_counts().plot.bar()
```

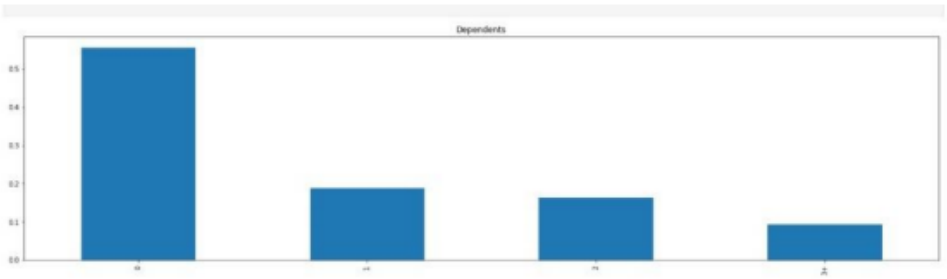
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd35a11fa10>
```




```
df['Dependents'].value_counts(normalize=True).plot.bar(figsize=(24,6),title='Dependents')
plt.show()
df['Education'].value_counts(normalize=True).plot.bar(title='Education')
plt.show()
df['Property Area'].value_counts(normalize=True).plot.bar(title='Property Area')
plt.show()
```





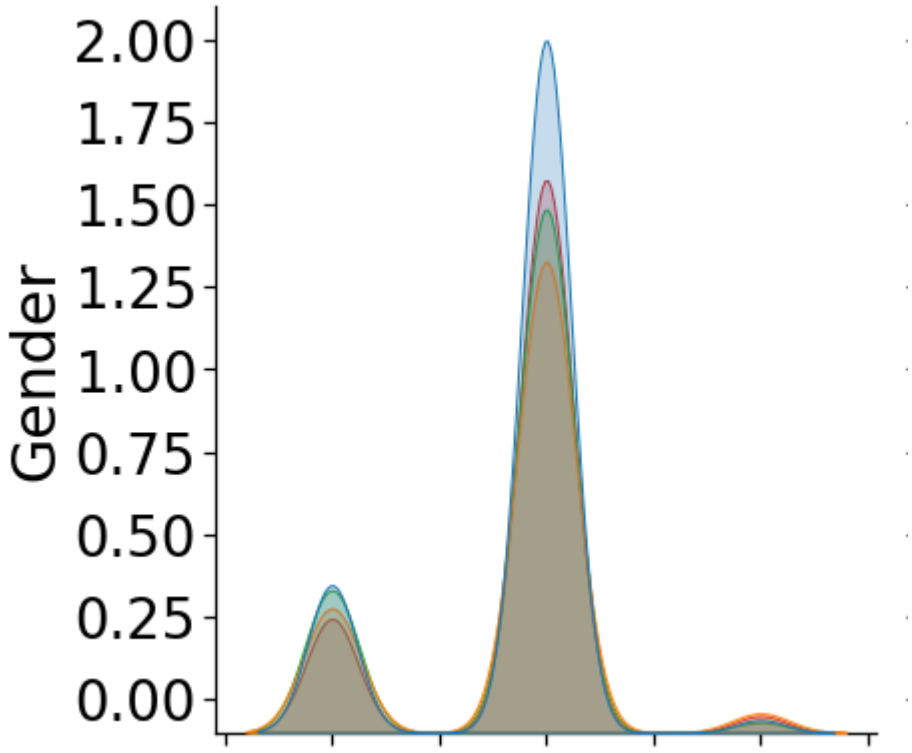


1. *We can see that approximately 81% are Male and 19% are female.*
2. *Percentage of applicants with no dependents is higher.*
3. *There are more number of graduates than non graduates.*
4. *Semi Urban people is slightly higher than Urban people among the applicants.*
5. *Larger Percentage of people have a good credit history.*

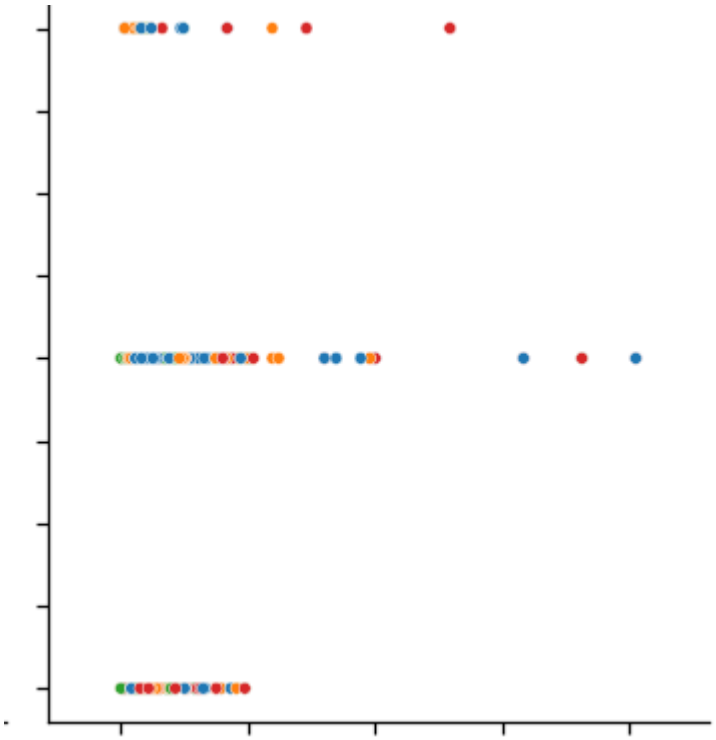
6. *The percentage of people that the loan has been approved has been higher rather than the percentage of applicant for which the loan has been declined.*

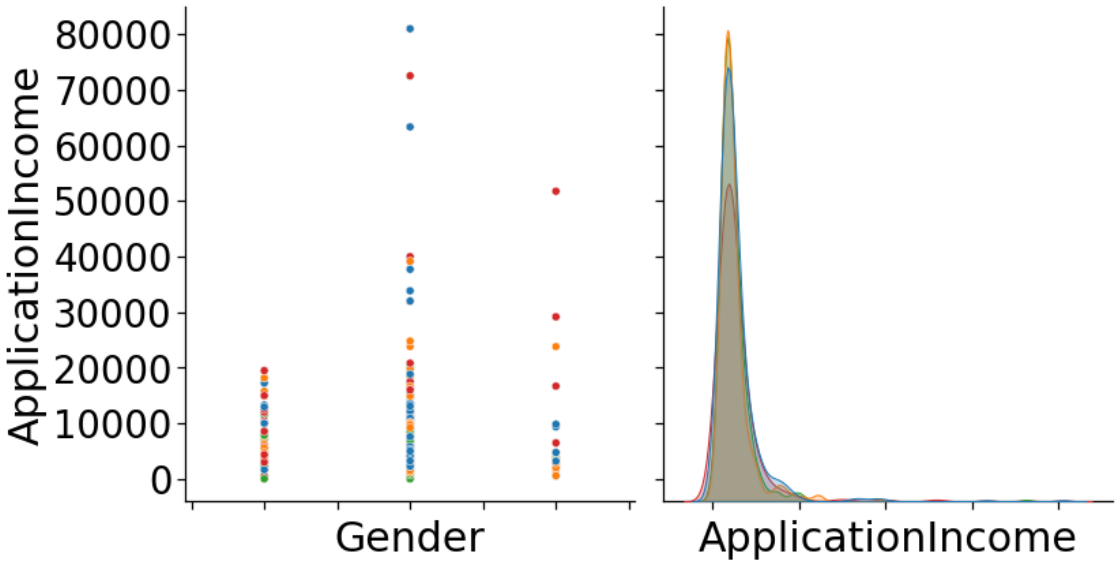
Now let me try different approaches to this problem. As our main target is Loan_Status Variable , let us try to find if Applicant income can exactly separate the Loan_Status. Suppose if i can find that if applicant income is above some X amount then Loan Status is yes .Else it is No. Firstly I am trying to plot the distribution plot based on Loan_Status.

```
with sns.plotting_context("notebook",font_scale=2.5):
  g=sns.pairplot(df[['Gender','Married','Dependents','Education','Self_Employed','ApplicationIncome','CoapplicationIncome','LoanAmount','Loan_Amount_Term','Credit_History','Property Area','Lo
g.set(xticklabels=[]);
```

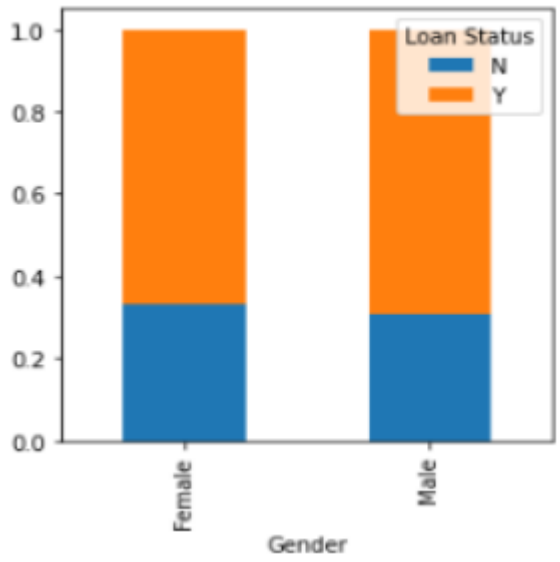


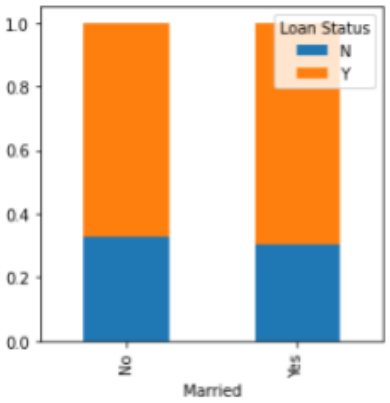
Unfortunately I cannot segregate based on Applicant Income alone. The Same is the case with Co-applicant Income and Loan-Amount. Let me try different visualization technique so that we can understand better

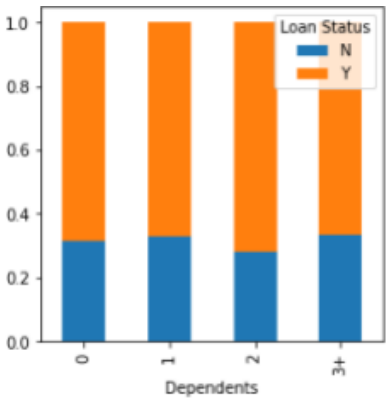


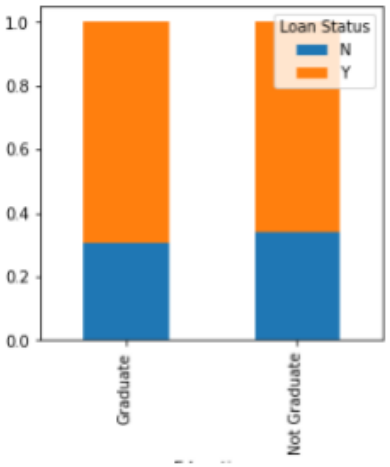


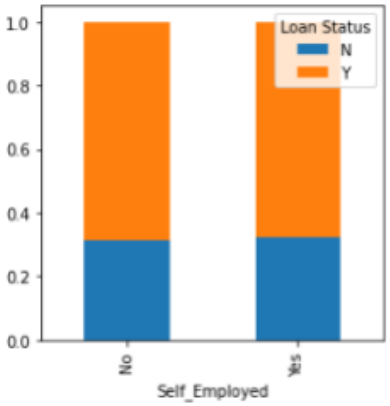
In the above one I tried to know whether we can segregate the Loan Status based on Applicant Income and Credit_History. Now Can I say to some extent that Applicant income which is less than 20,000 and Credit History which is 0 can be segregated as NO for Loan_Status. I don't think I can as it not dependent on Credit History itself at least for income less than 20,000. Hence even this approach did not make a great sense. Now we will move on to cross tab plot.











Conclusions:

We can infer that percentage of married people who have got their loan approved is higher when compared to non- married people.

The percentage of applicants with either 0 or 2 dependents have got their loan approved is higher.

The percentage of applicants who are graduates have got their loan approved rather than the one who are not graduates.

There is hardly any correlation between Loan_Status and Self_Employed applicants. So in short we can say that it doesn't matter whether the applicant is self employed or not.

Despite seeing some data analysis, unfortunately we could not figure out what factors exactly would distinguish the Loan Status column. Hence we go to next step which is nothing but Data Cleaning.

DATA CLEANING AND STRUCTURING—

Before we go for modeling the data, we have to check whether the data is cleaned or not. And after cleaning part, we have to structure the Data. For cleaning part, First I have to check whether there exists any missing values. For that I am using the code snippet **isnull()**

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

```
Loan-ID                0
Gender                 0
Married                0
Dependents             0
Education              0
Self_Employed          0
ApplicationIncome      0
CoapplicationIncome    0
LoanAmount            55
Loan_Amount_Term      40
Credit_History       296
Property Area          0
Loan Status            0
dtype: int64
```

In the above code, missing values of Loan-Amount is replaced by 128 which is nothing but the median. Loan_Amount_Term is a continuous variable. Here also I can replace with median. However the most occurring value is 360 which is nothing but 30 years. I just saw if there is any difference between median and mode values for this data. However there is no difference, hence I chose 360 as the term that has to be replaced for missing values. After replacing let us check if there are further any missing values by the following code train1.isnull().sum().

```
df['Loan_Amount_Term'].value_counts()
```

```
360.0    1650
180.0     132
480.0      46
300.0      40
240.0      16
84.0       13
120.0      12
600.0       8
350.0       2
840.0       1
Name: Loan_Amount_Term, dtype: int64
```

As we already know that there are 614 rows in our train data set, there should be 614 unique Loan_ID's. Fortunately there are no duplicate values. We can also see that for Gender, Married, Education and Self_Employed columns, the values are only 2 which is evident after cleaning the data-set.

Till now we have cleaned only our train data set, we have to apply the same strategy to test data set too.

As the data cleaning and data structuring are done, we will be going to our next section which is nothing but Model Building.

DEALING CATEGORICAL VARIABLES:

As our target variable is Loan_Status. We are storing it in a variable called y. But before doing all these we are dropping LOAN_ID column in both the data sets. Here it goes.

```
'''x_train = df.iloc[:3,:-1]
y_train = df.iloc[:3,-1:]
x_test  =df.iloc[3:,:-1]
y_test  =df.iloc[3:,-1:]'''
```

```
'x_train = df.iloc[:3,:-1]\ny_train = df.iloc[:3,-1:]\nx_test  =df.iloc[3:,:-1]\ny_test  =df.iloc[3:,-1:]'
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.5,random_state=0)
```

```
x_train= df[['Property Area','Gender','Married','Education']]
y_train = df['Loan Status']
x_test= df[['Property Area','Gender','Married','Education']]
y_test = df['Loan Status']
```

```
x_train
```

As we are having a lot of categorical variables that are affecting Loan Status. We need to convert each of them in to numeric data for modeling.

	Gender	Married	Dependents	Education	Self_Employed	ApplicationIncome	CoapplicationIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property Area	Loan Status	Loan Type
0	1	No	0	Graduate	No	5849	0	NAN	360	1	Urban	Y	Education Loan
1	1	Yes	1	Graduate	No	4583	1508	128	360	1	Rural	N	House Loan
2	1	Yes	0	Graduate	Yes	3000	0	66	360	1	Urban	Y	Health Loan
3	1	Yes	0	Not Graduate	No	2583	2358	120	360	1	Urban	Y	Marriage Loan
4	1	Yes	2	Graduate	Yes	5417	4196	267	360	1	Urban	Y	Education Loan

```
from sklearn.preprocessing import LabelEncoder
var_mod = ['Gender','Married','Dependents','Education','Self_Employed','Property Area','Loan Status']
le = LabelEncoder()
for i in var_mod:
    df[i] = le.fit_transform(df[i])
df.head()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicationIncome	CoapplicationIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property Area	Loan Status	Loan Type
0	1	1	0	0	1	5849	0	NAN	360	1	2	1	Education Loan
1	1	2	1	0	1	4583	1508	128	360	1	0	0	House Loan
2	1	2	0	0	2	3000	0	66	360	1	2	1	Health Loan
3	1	2	0	1	1	2583	2358	120	360	1	2	1	Marriage Loan
4	1	2	2	0	2	5417	4196	267	360	1	2	1	Education Loan

For handling categorical variables, there are many methods like One Hot Encoding or Dummies. In one hot encoding method we can specify which categorical data needs to be converted . However as in my case, as I need to convert every categorical variable in to numerical, I have used get_dummies method.

If we observe clearly, the main column will be disappeared. For example Gender has two types male and female. The column Gender has disappeared and been converted to Gender_Male and Gender_Female. Similar is the case with Married , Dependents and each other categorical variable.

Getting new Data Columns

We have two columns named applicant income and co-applicant income. It may be the case that total income might have a great impact on Loan Status. This is just a guess. It may or may not work. Also It may be the case that EMI would have a greater impact on Loan Status as it combines Loan Amount and Loan Amount Term. So I am just using some common sense to find new variables that can impact. Well this concept in short is known as Feature Engineering. (It's not as easy as what is explained here….But to make our model better ..we are approaching this way…..)

Hence the formula that can be used is

Total Income = Applicant Income + Co-applicant Income

EMI = Loan Amount / Loan Amount Term

Of-course EMI is not calculated this way. As it is for home loans let's check the interest rates in google for several banks.

By considering the above link, I have found that on an average it would be around 8.5% to 9.5%. Hence for safe-side I am assuming that 9% is the interest rate.

A = P*R*(1+R)^N

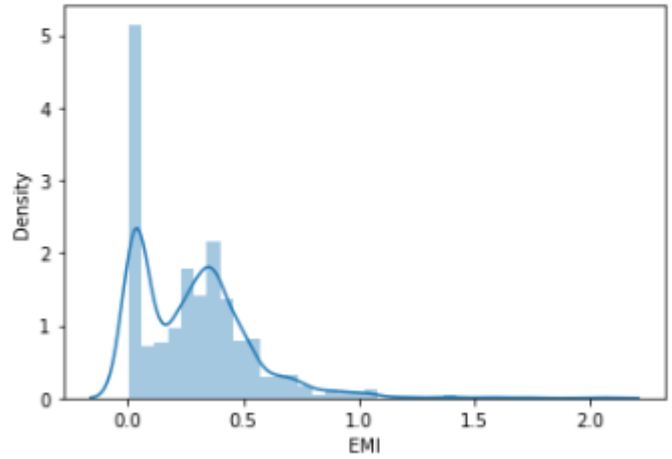
B = (1+R)^(N-1)

EMI = A/B.

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

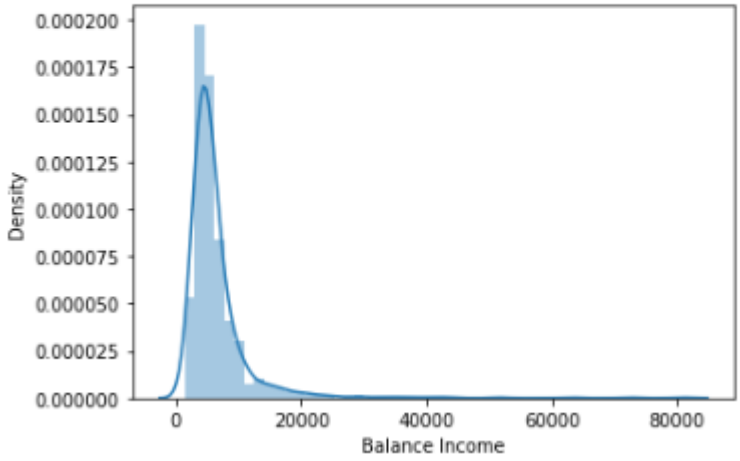
```
sns.distplot(df['EMI'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd359e1aa10>
```



```
df['Balance Income'] = df['Total_Income']-(df['EMI']*1000)
sns.distplot(df['Balance Income'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd365308310>
```



Logistic Regression:

As here we wan't to classify between the people who have taken loan or not we have used Logistic Regression. The purpose of this algorithm is to find a plane that separates two types. Y variable belongs to 1 or 0. In 2–d we have to figure out a line that exactly separates two classes.

Here is the code for Logistic Regression and Stratified Sampling.

```
[ ] from sklearn.tree import DecisionTreeClassifier
    tree = DecisionTreeClassifier()
    tree.fit(x_train,y_train)
```

```
DecisionTreeClassifier()
```

```
print('predicted training data',tree.score(x_train,y_train))
print('predicted testing data',tree.score(x_test,y_test))
```

```
predicted training data 0.8345481049562682
predicted testing data 0.5255102040816326
```

```
[ ] accuracy=tree.score(x,y)*100
    print(accuracy)
```

```
74.18367346938776
```

```
[ ] from sklearn.svm import SVC
    svm_clf = SVC()
    svm_clf.fit(x_train,y_train)
```

```
SVC()
```

```
[ ] print('predicted training data',svm_clf.score(x_train,y_train))
    print('predicted testing data',svm_clf.score(x_test,y_test))
```

```
predicted training data 0.6895043731778425
predicted testing data 0.6887755102040817
```

```
[ ] accuracy=svm_clf.score(x,y)*100
    print(accuracy)
```

```
68.92857142857143
```

```
from sklearn.ensemble import RandomForestClassifier
forest=RandomForestClassifier()
forest.fit(x_train,y_train)
```

```
RandomForestClassifier()
```

```
print('predicted training data',forest.score(x_train,y_train))
print('predicted testing data',forest.score(x_test,y_test))
```

```
predicted training data 0.8345481049562682
predicted testing data 0.5663265306122449
```

```
accuracy=forest.score(x,y)*100
print(accuracy)
```

```
75.40816326530613
```

```
from sklearn.neighbors import KNeighborsClassifier
model1=KNeighborsClassifier()
model1.fit(x_train,y_train)
```

```
KNeighborsClassifier()
```

```
print("Predicted value for training data:",model1.score(x_train,y_train))
print("Predicted value for testing data:",model1.score(x_test,y_test))
```

```
Predicted value for training data: 0.7128279883381924
Predicted value for testing data: 0.6292517006802721
```

```
accuracy=model1.score(x,y)*100
print(accuracy)
```

```
68.77551020408164
```

```
from sklearn.linear_model import LogisticRegression
model2=LogisticRegression()
model2.fit(x_train,y_train)
```

```
LogisticRegression()
```

```
print("Predicted value for training data:",model2.score(x_train,y_train))
print("Predicted value for testing data:",model2.score(x_test,y_test))
```

```
Predicted value for training data: 0.685131195335277
Predicted value for testing data: 0.6921768707482994
```

```
accuracy=model2.score(x,y)*100
print(accuracy)
```

```
68.72448979591836
```

```
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(x_train, y_train)
```

```
XGBClassifier()
```

```
print("Predicted value for training data:",model.score(x_train,y_train))
print("Predicted value for testing data:",model.score(x_test,y_test))
```

```
Predicted value for training data: 0.7244897959183674
Predicted value for testing data: 0.6581632653061225
```

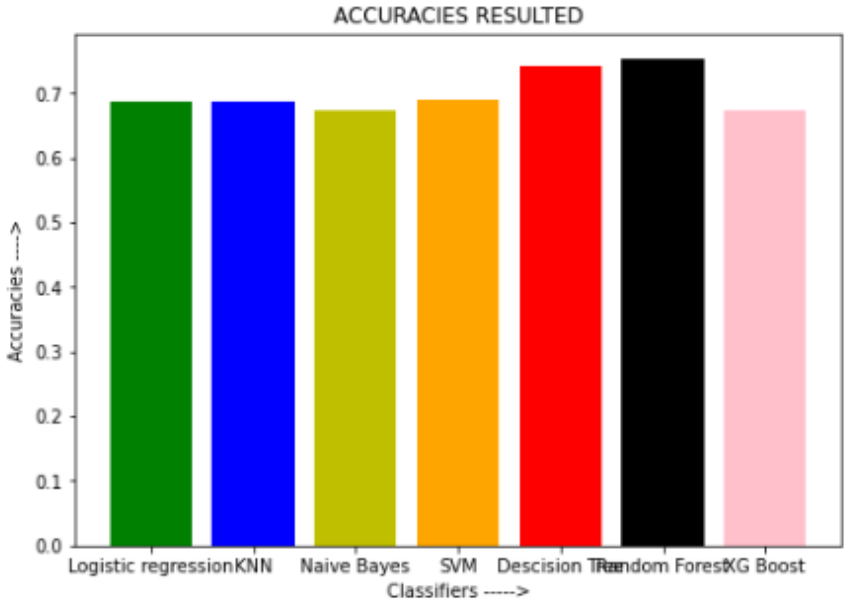
```
accuracy=model.score(x,y)*100
print(accuracy)
```

```
70.45918367346938
```

```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
```

```
GaussianNB()
```

```
import matplotlib.pyplot as plt

fig = plt.figure()

ax = fig.add_axes([1,1,1,1])

algo = ['Logistic regression','KNN','Naive Bayes','SVM','Descision Tree','Random Forest','XG Boost']

accuracy = [model2.score(x,y),model1.score(x,y),model.score(x,y),svm_clf.score(x,y),tree.score(x,y),forest.score(x,y),model.score(x,y)]

ax.bar(algo[0],accuracy[0],color = 'g')
ax.bar(algo[1],accuracy[1],color = 'b')
ax.bar(algo[2],accuracy[2],color = 'y')
ax.bar(algo[3],accuracy[3],color = 'orange')
ax.bar(algo[4],accuracy[4],color = 'red')
ax.bar(algo[5],accuracy[5],color = 'black')
ax.bar(algo[6],accuracy[6],color = 'pink')

plt.xlabel('Classifiers ----->')
plt.ylabel('Accuracies ---->')
plt.title('ACCURACIES RESULTED')
plt.show()
```



I have tried various techniques like Random Forest, Decision Tree, Decision Tree etc. and came to conclusion that the above code gave maximum accuracy. However there is still a lot of room to enhance accuracy which I have to figure it out still.

As we can see average accuracy is 75% I have used the same thing for predicting test data variable. However, how much ever i try i ended up with maximum accuracy 75%