NAVEEN
POLIASETTY'S

LOAN ELIGIBILITY PREDICTION PROJECT

REPORT

Table of Contents

REPORT BY
NAVEEN POLIASETTY

Introduction

Loan Eligibility Prediction

Nowadays taking out a home loan is a decision that many people find tempting for a variety of reasons. Whether it's to purchase a dream home, invest in real estate, or simply upgrade to a bigger, better living space, the lure of homeownership is strong.

Home loans typically come with a fixed or variable interest rate, which is determined based on several factors such as the borrower's credit history, the loan amount, and the loan tenure. The interest rate on home loans can vary depending on market conditions, and borrowers may have the option to choose between a fixed or variable interest rate depending on their preference.

Since many of the people may show interest to take loans making an Machine learning model which can classify according to the clients information whether he/she is eligible for loan can be useful to have clear cut results and can fasten the process of loan approval

We will be using classification algorithms like Decision Trees , Gaussian Naive Bayes , Logistic Regression and Support Vector Machine from Scikit Learn and few libraries like pandas , NumPy and Matplotlib

DATA DESCRIPTION

The Most valuable asset of any company is users data . In here we will be dealing with such data which is actually powerful enough to make a model accurate and at the same inaccurate too. Maintaining the data in an appropriate way can make us happy all time.

We have two Data sets namely "Train Data set" and "Test Data set". We will be using the Train data for Training purpose and Test data for testing purpose . We import the data sets using pandas

```
df = pd.read_csv("train_loan.csv")
df
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0

614 rows × 13 columns

As we can see in the above we have the data set with 13 features in total and the rows denoting clients . Out of 13 features 11 are independent variables excluding Loan_ID , and Loan_Status . Let's dive deeper into the features and see what they have for us

- Loan_ID :
  For every client in the bank has this Loan_ID which cannot be assumed as an independent variable ,as this Loan_ID variables cannot affect the resultant Loan _Status . In order make easier for the model it is recommended to drop this column from the dataset when dealing with preprocessing and training the model .

- Gender :
  The Gender column gives us the gender of the client seeking for loan .

- Marriage:
  The Marriage column gives us the marital status of the client .

- Dependents :
  Dependents refer to the number of people who rely on the applicant for financial support. This can include children, elderly parents, or other family members who are financially dependent on the applicant. The number of dependents can be an important factor in determining the applicant's ability to repay the loan and may be taken into consideration by the lender when evaluating the loan application.

- Education:
  The Education column gives us the info whether the client is an graduate or not an graduate . The Education of the client affect the loan status which is one of the factor that lender takes into consideration when evaluating a loan application . Typically for having higher education can be seen as an positive factor as it often associated with higher income and financial stability

```
In [4]: 1 pd.crosstab(df["Education"], df["Loan_Status"])
```

Out[4]:

Loan_Status	N	Y
Education		
Graduate	140	340
Not Graduate	52	82

- Self Employed :
  The Self Employed column gives us whether the applicant is employed by someone else or is running his/her own business .This information can affect the loan status as self-employed individuals may have less predictable income streams than salaried individuals, which may make lenders perceive them as higher risk borrowers

```
In [64]: 1 pd.crosstab(df["Self_Employed"], df["Loan_Status"])
```

Out[64]:

Loan_Status	N	Y
Self_Employed		
No	166	366
Yes	26	56

- ApplicantIncome :
  The ApplicantIncome column gives us the income of the client per month . The applicant's income is an important factor that affects the loan status. A higher income generally indicates that the applicant is capable of repaying the loan amount and may improve the chances of loan approval. Lenders often have a minimum income requirement for loan approval, and a higher income may also make the applicant eligible for a larger loan amount.

- CoapplicantIncome :
  The coapplicantincome is the income of the person who is collateral for the client . The coapplicant is the person who is ready to repay the loan if the applicant misbehaves or if in a condition where he cannot repay . Even this is one of the important factor which may affect the loan status .

- Loan Amount :
  The loan amount is the total amount of money borrowed by a borrower from a lender. It plays a significant role in determining the terms and conditions of the loan, as well as the overall financial impact of the loan on the borrower.

- Loan Amount Term :
  Loan Amount term or Loan Repayment Term is the amount that a borrower has to repay monthly . Typically, loans with larger amounts have longer repayment terms, while loans with smaller amounts have shorter repayment terms. The longer the repayment term, the lower the monthly payments .

- Credit history :
  Credit history refers to a record of a person's borrowing and repayment activities. It includes all credit accounts, such as credit cards, loans and their payment history. Credit history is used by lenders, banks, and other financial institutions to assess a person's creditworthiness and ability to repay loans.

- Property Area :
  The bank have a presence across all urban, semi-urban and rural areas ,this property columns gives us the info about which area does this person belong too.

- Loan Status :
  Loan_Status is the resultant column and dependent column , based on the features the loan status will be either YES or NO which mean that the applicant is eligible for loan or not eligible for loan respectively

Approach

So from the data description we had a clear understanding of each feature . As of now we have understood to classify the rows according to the client's information . In Machine learning the Classification part is satisfied with several classification alogorithms such as

1. Logistic Regression: It is a linear classification algorithm used for binary classification problems.
2. Decision Trees: It is a tree-based classification algorithm that partitions the dataset recursively into smaller subsets based on the value of the input features.
3. Naive Bayes: It is a probabilistic classification algorithm that uses Bayes' theorem to predict the class of an input sample.
4. Support Vector Machines (SVM): It is a linear or nonlinear classification algorithm that finds a hyperplane that separates the input samples into different classes.
5. K-Nearest Neighbors (KNN): It is a non-parametric classification algorithm that assigns the class of an input sample based on the k-nearest neighbors in the training dataset.
6. Random Forest: It is an ensemble learning method that combines multiple decision trees to improve the accuracy of the classification model.
7. Gradient Boosting: It is also an ensemble learning method that combines multiple weak classifiers to create a strong classifier.

Before applying the algorithms we must visualize , understand and preprocess the data. So after treating the data we apply these algorithms ,find the accuracy and then according to the accuracy scores we will be choosing one algorithm from the rest and predict the 'y' component

Data Visualization and Preprocessing

Visualizing the data is the foundation for machine learning applications although without understanding the data one cannot perform any tasks. There is quote and it goes like " A problem well-defined is a problem half-solved" ,and here the visualizing part was made easy with the help of "matplotlib" library. Moreover as of now we had a clear description of data , how banks process loans , what are the key factors and how one cannot attain loan . So what we actually should look into to the visualization part is whether the data is good enough to fit the model .

Step 1

Initially I have cross tabulated Credit History , Education and Self Employment as these features are considered as key features to attain Loan

```
In [3]: 1 pd.crosstab(df["Credit_History"], df["Loan_Status"])
```

Out[3]:

Credit_History	Loan_Status N	Y
0.0	82	7
1.0	97	378

```
In [4]: 1 pd.crosstab(df["Education"], df["Loan_Status"])
```

Out[4]:

Education	Loan_Status N	Y
Graduate	140	340
Not Graduate	52	82

```
In [64]: 1 pd.crosstab(df["Self_Employed"], df["Loan_Status"])
```

Out[64]:

Self_Employed	Loan_Status N	Y
No	166	366
Yes	26	56

Step 2

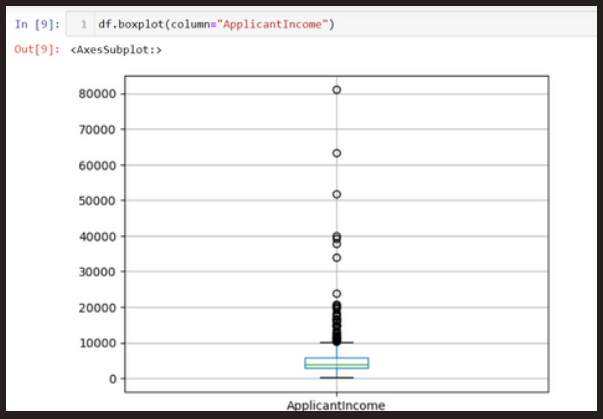
Diving deep into the data cells of features ,in order to have a quick look on data within a feature I am using info() method from pandas

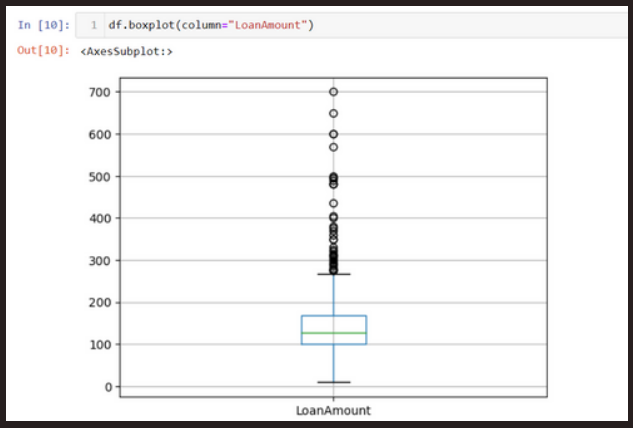
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Loan_ID            614 non-null    object
 1   Gender             601 non-null    object
 2   Married            611 non-null    object
 3   Dependents         599 non-null    object
 4   Education          614 non-null    object
 5   Self_Employed      582 non-null    object
 6   ApplicantIncome    614 non-null    int64
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object
 12  Loan_Status        614 non-null    object
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

as we can see have float , int and object datatypes within the features and even few NaN values are present we will look into it later in preprocessing Shape of the Data frame is (614 , 13) .

Step 3

Plotting using matplotlib will be performed on Loan amount and applicant Income since features like loan_ID , gender, married , dependents , education , self_employed , credit history and property area cannot be plotted . But features like coapplicant income and loan amount term can also be plotted , the reason why I am not taking them is because I am considering Applicant Income and Loan Amount as primary key features .



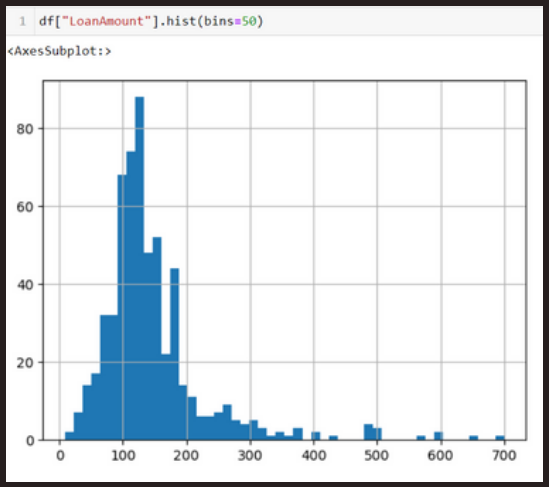


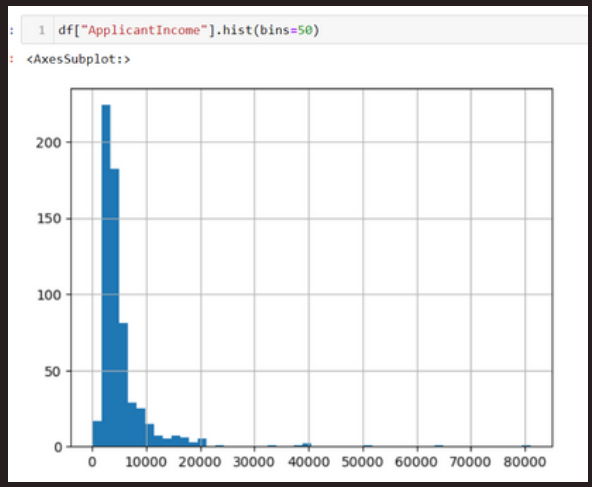
Box Plots

Box plots allows us to visualize the data in a statistical way for exploring the distribution of the dataset . Box plots provide a visual summary of the distribution of a dataset, including the median, quartiles, and potential outliers. The horizontal inside the box is the median, the box gives us the interquartile range . The upper whisker extending to the highest data point within 1.5 times the IQR of the box, and the lower whisker extending to the lowest data point within 1.5 times the IQR of the box. Any data points outside the whiskers are considered outliers and are represented by dots.

Histograms

Using histograms we can visualize the frequency distribution of a set of continuous or discrete data by grouping the data into bins and plotting the number of observations in each bin on the y-axis against the bin ranges on the x-axis , from the histograms we can see that the LoanAmount and Applicant Income both are right skewed . Further if deal with algorithms like Gaussian Naive bayes we must treat these features .





Filling the null values

If missing data is not handled properly, it can cause errors in machine learning models during training. Some machine learning algorithms may not be able to handle missing data, while others may produce biased or inaccurate results if missing data is ignored . Using fillna() from pandas can help in filling the null values . For categorical values we use the mode (the most occuring variable) and for numerical values we use mean(average) value

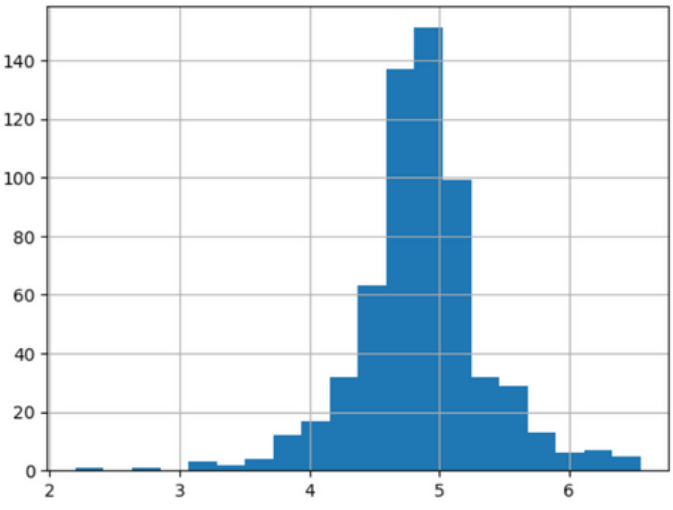
The mode [0] is used to extract the value where as "inplace = True" helps in permanently replacing the null value , after running the cell we can check whether the values are replaced or not by using the ' isnull() ' gives the info in an tabular format , where the " sum() " helps us in viewing in a summarized way

Normalizing the Data

Logarithmic transformation can help to normalize the distribution of data by reducing the impact of outliers and extreme values. This is because the logarithmic function compresses larger values more than smaller values, which can help to reduce the range of values and make the data more symmetrical . Logarithmic transformation can also help to reduce skewness in the data by transforming highly skewed data into a more normal distribution. This is because the logarithmic function is concave.

```
df["LoanAmount"] = np.log(df["LoanAmount"])
df["LoanAmount"].hist(bins=20)
```

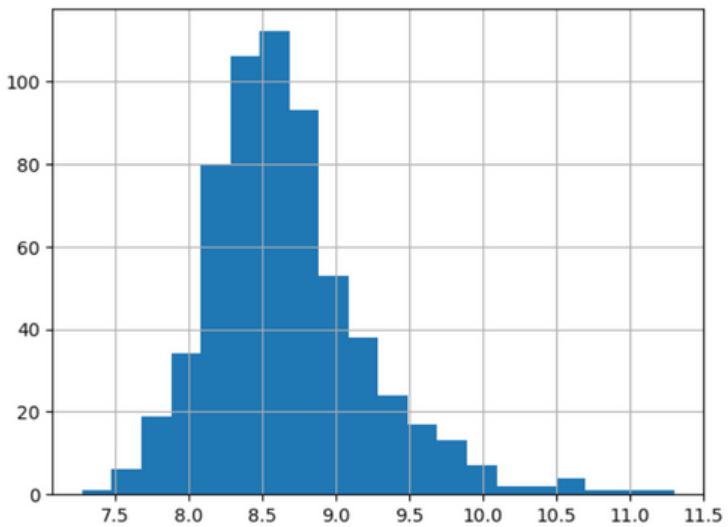
```
<AxesSubplot:>
```




```
df["totalIncome"] = df["ApplicantIncome"] + df["CoapplicantIncome"]
df["totalIncome"] = np.log(df["totalIncome"])
```

```
df["totalIncome"].hist(bins=20)
```

```
<AxesSubplot:>
```



I have created a another column called total income where the sum of applicant and co applicant income were took into consideration , so that it can be easy for model to find the relationships . With the help of the Logarithmic function now the data is normally distributed which can also help the algorithms with probabilistic classification. The column loan_ID is not going to help being inthe train dataset so we will be removing it .

Label Encoding

Label encoding is a technique used in machine learning to transform categorical data into numerical data, so that it can be used as input to machine learning models. Most machine learning algorithms are designed to work with numerical data. Categorical data in its original form cannot be used directly as input to these algorithms, so it needs to be transformed into numerical data. I converted them using a for loop to reduce the code

```
cols = [
    "Gender",
    "Dependents",
    "Married",
    "Education",
    "Self_Employed",
    "Property_Area",
    "Loan_Status",
]
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for x in cols:
    df[x] = le.fit_transform(df[x])
df
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Propert
0	1	0	0	0	0	5849	0.0	4.986426	360.0	1.0	
1	1	1	1	0	0	4583	1508.0	4.852030	360.0	1.0	
2	1	1	0	0	1	3000	0.0	4.189655	360.0	1.0	
3	1	1	0	1	0	2583	2358.0	4.787492	360.0	1.0	
4	1	0	0	0	0	6000	0.0	4.948760	360.0	1.0	
...	...	...	...	...	...	...	...	...	...	...	
609	0	0	0	0	0	2900	0.0	4.262680	360.0	1.0	
610	1	1	3	0	0	4106	0.0	3.688879	180.0	1.0	
611	1	1	1	0	0	8072	240.0	5.533389	360.0	1.0	
612	1	1	2	0	0	7583	0.0	5.231109	360.0	1.0	
613	0	0	0	0	1	4583	0.0	4.890349	360.0	0.0	

614 rows × 13 columns

Splitting the dataset between X and y

Splitting Independent variable columns as X and Dependent variable as y is first step to to begin with using SciKit Learn library . After the separation of dependent and independent variables ,I have flattened the y component for smooth processing , actually the Naive bayes expects a single dimensioned input .

```
y = df[["Loan_Status"]]
y = y.values.ravel()
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Train Test Split

Train test split from sklearn is an function which split the passed dataset as training and test datasets where we can pass the test size ex: 0. 2 which denotes that the 20% of the entire data to be splitted as test dataset and remaining 80% as train dataset , also we can give the random state which accordingly picks the data for splittings

Standard Scaling

Standard scaling is a useful preprocessing technique that can improve the performance of machine learning algorithms and simplify the interpretation of the resulting models. Many machine learning algorithms assume that the features are normally distributed. By standardizing the features, we can ensure that they are closer to a normal distribution, which can improve the performance of these algorithms. Here the word 'st 'is denoted as an obejct for StandardScalar() For any task by the method can be satisfied using this object

```
from sklearn.preprocessing import StandardScaler

st = StandardScaler()
X_train = st.fit_transform(X_train)
X_test = st.fit_transform(X_test)
```

Algorithms

Decision Tree Classifier

- Decision trees are a tree-based classification algorithm that partitions the dataset recursively into smaller subsets based on the value of the input features.
- Decision trees are easy to understand and interpret, and they can handle both categorical and numerical features.

From Scikit Learn we import the Decision tree classifier and create an object for it the criterion will be entropy .

Why the criterion we taking is entropy ?

When using entropy as the criterion for a decision tree, the algorithm tries to split the data in a way that maximizes the information gain, which is the reduction in entropy achieved by splitting the data at a given node. Intuitively, this means that the algorithm tries to create splits that result in subsets of the data that are as pure as possible

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(criterion="entropy", random_state=0)
dtc.fit(X_train, y_train)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
y_pred = dtc.predict(X_test)
y_pred
```

```
array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1])
```

```
from sklearn import metrics

print(
    "The Accuracy gained by Decision Treee is:", metrics.accuracy_score(y_pred, y_test)
)
```

```
The Accuracy gained by Decision Treee is: 0.7723577235772358
```

After passing the parameters the model will ready to use , predict the y_test by using the predict method and find the accuracy of the model prediction and actual y_test using metrics.accuracy.score by passing the predicted_y and y_test

Accuracy : 77.235%

Gaussian Naive Bayes

- In Gaussian Naive Bayes, it is assumed that the input features are continuous variables that follow a Gaussian (normal) distribution. This assumption allows for the calculation of the probability density function (PDF) of each input feature, which is then used to calculate the posterior probability of each class given the input features

```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred2 = nb.predict(X_test)
print("The Accuracy gained by Naive Bayes is:", metrics.accuracy_score(y_pred2, y_test))
```

```
The Accuracy gained by Naive Bayes is: 0.8455284552845529
```

Accuracy : 84.55%

Logistic Regression

- Logistic regression is a linear classification algorithm used for binary classification problems, i.e., classifying an input into one of two classes.
- It uses a logistic function to map the input features to the probabilities of belonging to each class.

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_pred3 = lr.predict(X_test)
```

```
metrics.accuracy_score(y_test, y_pred3)
```

```
0.8373983739837398
```

Accuracy : 83.73%

Support Vector Machine

Support Vector Machine (SVM) is a powerful and popular supervised learning algorithm that can be used for classification and regression tasks. SVM aims to find the best possible hyperplane that separates the data into different classes, where a hyperplane is a boundary that maximizes the margin between the closest data points from different classes

```
from sklearn.svm import SVC

svm = SVC()
svm.fit(X_train, y_train)
y_pred4 = svm.predict(X_test)
```

```
SVC()
```

```
print(
    "the accuracy gained using Support Vector Machine is :",
    metrics.accuracy_score(y_pred4, y_test),
)
```

```
the accuracy gained using Support Vector Machine is : 0.8048780487804879
```

Accuracy : 80.48%

Random Forest Classifier

Random Forest is a popular and powerful ensemble learning algorithm used for both classification and regression tasks. It is an extension of the decision tree algorithm that constructs multiple decision trees and combines their predictions to produce a final prediction

```
from sklearn.ensemble import RandomForestClassifier

ref = RandomForestClassifier()
ref.fit(X_train, y_train)
yik = ref.predict(X_test)
```

```
metrics.accuracy_score(yik, y_test)
```

```
0.7967479674796748
```

Accuracy : 79.67%

So far we performed several classification models out of which Gaussian Naive Bayes gives an accuracy of 84.55% so we choosing Naive Bayes for further loan eligibility predictions

Predicting Loan Eligibility

We have an dataset which has Independent variable columns and we will be predicting the dependent y using the previous model 'Naive Bayes'

As we discussed before we import the dataset to the jupyter notebook retrieve its information, find the null values count and fill them using fillna() method for categorical columns we use mode function and for numerical coulmns we use mean average

```
df1 = pd.read_csv("test_loan.csv")
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 367 entries, 0 to 366
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            367 non-null    object 
 1   Gender             356 non-null    object 
 2   Married            367 non-null    object 
 3   Dependents         357 non-null    object 
 4   Education          367 non-null    object 
 5   Self_Employed      344 non-null    object 
 6   ApplicantIncome    367 non-null    int64  
 7   CoapplicantIncome  367 non-null    int64  
 8   LoanAmount         362 non-null    float64
 9   Loan_Amount_Term   361 non-null    float64
 10  Credit_History     338 non-null    float64
 11  Property_Area      367 non-null    object 
dtypes: float64(3), int64(2), object(7)
memory usage: 34.5+ KB
```

```
df1
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
0	LP001015	Male	Yes	0	Graduate	No	5720	0	110.0	360.0	1.0
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	126.0	360.0	1.0
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	208.0	360.0	1.0
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	100.0	360.0	NaN
4	LP001051	Male	No	0	Not Graduate	No	3276	0	78.0	360.0	1.0
...	...	...	...	...	...	...	...	...	...	...	...
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	1777	113.0	360.0	1.0
363	LP002975	Male	Yes	0	Graduate	No	4158	709	115.0	360.0	1.0
364	LP002980	Male	No	0	Graduate	No	3250	1993	126.0	360.0	NaN
365	LP002986	Male	Yes	0	Graduate	No	5000	2393	158.0	360.0	1.0
366	LP002989	Male	No	0	Graduate	Yes	9200	0	98.0	180.0	1.0

367 rows × 12 columns

```
df1["Gender"].fillna(df1["Gender"].mode()[0], inplace=True)
df1["Dependents"].fillna(df1["Dependents"].mode()[0], inplace=True)
df1["Self_Employed"].fillna(df1["Self_Employed"].mode()[0], inplace=True)
```

```
df1["LoanAmount"].fillna(df1["LoanAmount"].mean(), inplace=True)
df1["Loan_Amount_Term"].fillna(df1["Loan_Amount_Term"].mean(), inplace=True)
df1["ApplicantIncome"].fillna(df1["ApplicantIncome"].mean(), inplace=True)
df1["CoapplicantIncome"].fillna(df1["CoapplicantIncome"].mean(), inplace=True)
df1["Credit_History"].fillna(df1["Credit_History"].mean(), inplace=True)
```

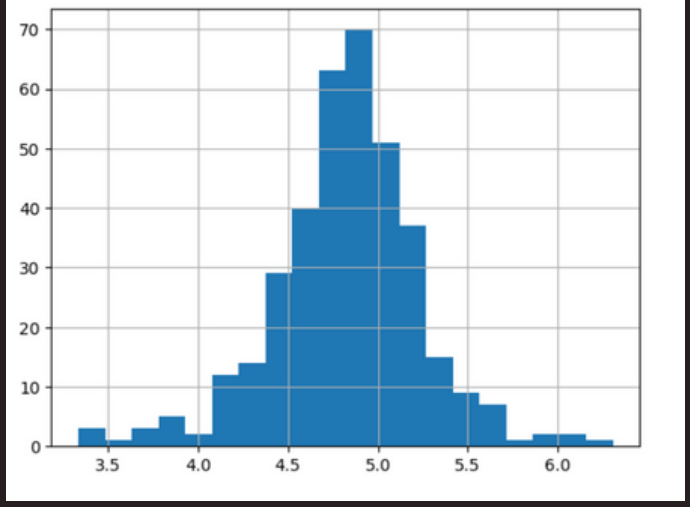
```
df1.isnull().sum()
```

```
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
```

Treating the "Loan Amount" and "Total Income " columns for better results using Logarithmic functions

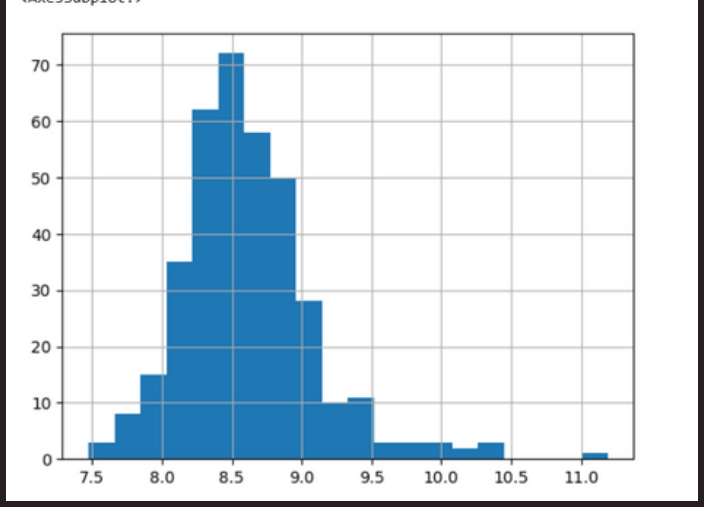
```
df1["LoanAmount"] = np.log(df1["LoanAmount"])
df1["LoanAmount"].hist(bins=20)
```

```
<AxesSubplot:>
```




```
df1["TotalIncome"] = df1["ApplicantIncome"] + df1["CoapplicantIncome"]
df1["TotalIncome"] = np.log(df1["TotalIncome"])
df1["TotalIncome"].hist(bins=20)
```

```
<AxesSubplot:>
```



Preserving a copy of the datast before removing the "Loan_ID" column for further requirements

```
df2 = df1.copy()
df1 = df1.drop(columns=["Loan_ID"])
```

```
li = ["Gender", "Dependents", "Married", "Education", "Self_Employed", "Property_Area"]
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for x in li:
    df1[x] = le.fit_transform(df1[x])
df1
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_
0	1	1	0	0	0	5720	0	4.700480	360.0	1.000000	
1	1	1	1	0	0	3076	1500	4.836282	360.0	1.000000	
2	1	1	2	0	0	5000	1800	5.337538	360.0	1.000000	
3	1	1	2	0	0	2340	2546	4.605170	360.0	0.825444	
4	1	0	0	1	0	3276	0	4.356709	360.0	1.000000	
...	...	...	...	...	...	...	...	...	...	...	...
362	1	1	3	1	1	4009	1777	4.727388	360.0	1.000000	
363	1	1	0	0	0	4158	709	4.744932	360.0	1.000000	
364	1	0	0	0	0	3250	1993	4.836282	360.0	0.825444	
365	1	1	0	0	0	5000	2393	5.062595	360.0	1.000000	
366	1	0	0	0	1	9200	0	4.584967	180.0	1.000000	

Performed Standard Scaling to simplify the interpretation and Label encoding for transforming categorical values into numerical values and
Predicting the dependent variable using the model

```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
df1 = ss.fit_transform(df1)
```

```
finpred = nb.predict(df1)
finpred
```

```
array([1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

Converting the nd array to a data frame called endres and naming the index column as "Index" the numerical values other than index are the resultant column so naming the column can be more useful for identification the column is named as "Loan_Status"

```
endres = pd.DataFrame(finpred)
endres.index.name = "Index"
```

```
endres = endres.rename(columns={0: "Loan_Status"})
endres
```

	Loan_Status
0	1
1	1
2	1
3	1
4	1
...	...
362	1
363	1
364	1
365	1
366	1

367 rows × 1 columns

Similarly naming the index column as "Index" so that from both the datasets we can have a common column which helps us on merging them

```
df2.index.name = "Index"
df2
```

Index	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Histo
0	LP001015	Male	Yes	0	Graduate	No	5720	0	4.700480	360.0	1.0000
1	LP001022	Male	Yes	1	Graduate	No	3076	1500	4.836282	360.0	1.0000
2	LP001031	Male	Yes	2	Graduate	No	5000	1800	5.337538	360.0	1.0000
3	LP001035	Male	Yes	2	Graduate	No	2340	2546	4.605170	360.0	0.8254
4	LP001051	Male	No	0	Not Graduate	No	3276	0	4.356709	360.0	1.0000
...	...	...	...	...	...	...	...	...	...	...	
362	LP002971	Male	Yes	3+	Not Graduate	Yes	4009	1777	4.727388	360.0	1.0000
363	LP002975	Male	Yes	0	Graduate	No	4158	709	4.744932	360.0	1.0000
364	LP002980	Male	No	0	Graduate	No	3250	1993	4.836282	360.0	0.8254
365	LP002986	Male	Yes	0	Graduate	No	5000	2393	5.062595	360.0	1.0000
366	LP002989	Male	No	0	Graduate	Yes	9200	0	4.584967	180.0	1.0000

367 rows × 13 columns

The loan_status column is being Converting or replacing those 1's and 0's as Yes and No for human understanding about the loan status

```
df2["Loan_Status"] = df2["Loan_Status"].map({1: "YES", 0: "NO"})
df2
```

nts	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	TotalIncome	Loan_Status
0	Graduate	No	5720	0	4.700480	360.0	1.000000	Urban	8.651724	YES
1	Graduate	No	3076	1500	4.836282	360.0	1.000000	Urban	8.428581	YES
2	Graduate	No	5000	1800	5.337538	360.0	1.000000	Urban	8.824678	YES
2	Graduate	No	2340	2546	4.605170	360.0	0.825444	Urban	8.494129	YES
0	Not Graduate	No	3276	0	4.356709	360.0	1.000000	Urban	8.094378	YES
...	...	...	...	...	...	...	...	...	...	...
3+	Not Graduate	Yes	4009	1777	4.727388	360.0	1.000000	Urban	8.663196	YES
0	Graduate	No	4158	709	4.744932	360.0	1.000000	Urban	8.490233	YES
0	Graduate	No	3250	1993	4.836282	360.0	0.825444	Semiurban	8.564649	YES
0	Graduate	No	5000	2393	5.062595	360.0	1.000000	Rural	8.908289	YES
0	Graduate	Yes	9200	0	4.584967	180.0	1.000000	Rural	9.126959	YES

Additionally I have wrote a piece of code where user can enter his/her Loan_Id and can know the Loan_status (just to make use of the Loan_Status Column) Lets see how the actually works the execution starts by expecting the loan_ID of the user , then the user id will be matched to the loan_id column and row which matches with the id it extracts the Loan_Status column element and converts the object to string . If Else block checks if the b is empty then returns the print statement "Loan ID not found" else the status variable extracts the value with respect to the index, the next line prints the statement with f'string which reserves a place for status variable to be filled and then if the value is YES the statement "You are ELIGIBLE for loan " will be printed else the statement "Sorry , you are NOT ELIGIBLE for loan will be printed

```
id = input("Enter the Loan_ID to know the Loan_Status :")
b = df2.loc[df2["Loan_ID"] == id, "Loan_Status"].astype(str)
if b.empty:
    print("Loan ID not found")
else:
    status = b.iloc[0]
    print(f"The loan status for ID {id} is: {status}")
    if status == "YES":
        print("You are ELIGIBLE for loan!")
    else:
        print("Sorry, you are NOT ELIGIBLE for loan.")
```

```
Enter the Loan_ID to know the Loan_Status :LP002989
The loan status for ID LP002989 is: YES
You are ELIGIBLE for loan!
```

Conclusion

So this is how I done my first ever machine learning model using Machine learning algorithms few libraries like pandas ,NumPy, Matplotlib and of course Sklearn (Scikit learn) . Summarizing the project we have performed visualization , Exploratory Data Analysis, Data Preprocessing and using SKlearn functions .

Difficulties Faced

I gave try to reduce the code by using loops in filling the null values But I couldn't do so. The interpreter the throwing the SettingWithCopyingWarning which means that the code is trying to when we are trying to modify a slice of a Pandas DataFrame or a NumPy array using a new DataFrame or array. Going through the documentation I found that when I pass a nested tuple of to a single call to __getitem__. This allows pandas to deal with this as a single entity. Furthermore this order of operations can be significantly faster, and allows one to index both axes if so desired. So here when using the chained indexing the order and type of the indexing operation partially determine whether the result is a slice into the original object, or a copy of the slice. pandas has the SettingWithCopyWarning because assigning to a copy of a slice is frequently not intentional, but a mistake caused by chained indexing returning a copy where a slice was expected.

```
for x in Mo:
    df[x].fillna(df[x].mode()[0], inplace=True)
```

```
C:\Users\iamna\AppData\Local\Temp\ipykernel_13228\4107378536.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df[x].fillna(df[x].mode()[0], inplace=True)
C:\Users\iamna\AppData\Local\Temp\ipykernel_13228\4107378536.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df[x].fillna(df[x].mode()[0], inplace=True)
C:\Users\iamna\AppData\Local\Temp\ipykernel_13228\4107378536.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df[x].fillna(df[x].mode()[0], inplace=True)
C:\Users\iamna\AppData\Local\Temp\ipykernel_13228\4107378536.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-ve
rsus-a-copy
  df[x].fillna(df[x].mode()[0], inplace=True)
<IPython.core.display.Javascript object>
```

Moreover I have learned a lot from this project and looking forward to make more and more projects and gain practical experience this is how I am going to end this report

THANK YOU