Prediction of Loan Approval in Banks using Machine Learning Approach

Viswanatha V[1], Ramachandra A.C[2], Vishwas K N[3] and Adithya G[4]
[1]Assistant Professor, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, INDIA
[2]Professor, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, INDIA
[3]Student, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, INDIA
[4]Student, Department of Electronics and Communication Engineering, Nitte Meenakshi Institute of Technology, Bangalore, INDIA

[3]Corresponding Author: viswas779@gmail.com

Received: 01-07-2023 Revised: 16-07-2023 Accepted: 30-07-2023

ABSTRACT

Due to significant technology advancements, people's needs have expanded. As a result, there have been more requests for loan approval in the banking sector. A few qualities, taken for consideration, when choosing a candidate for loan approval in order to, determine loan's status. Banks face a major challenge; when it, comes to assessing loan applications and lowering the risks associated with potential borrower defaults. Since they must thoroughly evaluate each borrower's eligibility for a loan, banks find this process to be particularly challenging. This research proposes combining machine learning (ML) models and ensemble learning approaches to find the probability of accepting individual loan requests. This tactic can increase the accuracy with which qualified candidates are selected from a pool of applicants. As a result, this method can be used to address the problems with loan approval processes outlined above. Both the loan applicants and the bank employees profit from the strategy's dramatic reduction in sanctioning time. Because of the banking industry's expansion, more people were applying to loans at banks. In order to predict the accuracy of loan approval status for applied person, we used four different algorithms namely Random Forest, Naive Bayes, Decision Tree, and KNN. By using these, we obtained better accuracy of 83.73% with Naïve Bayes algorithm as best one.

Keywords-- Safe Customers, Bank Loans, Trained Dataset, Random Forests, KNN, Decision Tree, Naive Bayes

I. INTRODUCTION

Many banks' primary line of business is loan distribution. Loans given to consumers account for the majority of a bank's revenue. Interest is charged by these banks on loans given to customers. Banks' handled. It merely has the values x and y as independent and dependent variables. Data primary goal is to invest their funds in dependable clients. Many banks have been processing loans so far following a backward process of vetting and verification. However, as of right now, no bank can guarantee whether the customer who is selected for a loan application is secure or not. So, in order, to avoid this circumstance, we implemented the Loan Prediction System Using Python, a system for the approval of bank loans. The Loan Prediction System is a piece of software that determines if or not the specific customer is qualified to receive a loan. This technique examines number of variables, including the customer's marital status, income, spending, and other elements. For wide numbers of trained data set clients, this method/technique is used. These elements are, taken to consideration when creating the necessary model. In order for obtaining the desired outcome, this model is applied for the test data set. The result will be presented as either yes or no. If the answer is yes, then the customer is capable of repaying the loan; if the answer is no, then the consumer is not capable of repaying the loan. We can grant loans to clients based on these criteria. Machine learning is the study of how the systems of computers are used and developed to learn and adapt without explicit instructions by analysing and inferring patterns in data using algorithms and statistical models. It is so important in the twenty-first century that it was used practically everywhere, from Such a function can't be fitted with a straight line without incurring significant mistakes. Additionally, the datasets those with greater than two dimensions scientists developed polynomial regression, logistic regression, and even linear regression with having more variables to address these problems. As the accuracy sharply improved, more individuals grew interested in it and started working on it. The new era of data science began with the first use of the term "Big data" in 2005. Many ideas can now be fulfilled, including decision tree regression. commonplace things like a search engine and an email filter to more challenging issues like predicting consumer behaviour or our topic, predicting house

Peer Reviewed & Refereed Journal
Volume-13, Issue-4 (August 2023)
https://doi.org/10.31033/ijemr.13.4.2

prices. Although the concept of regression, or the act of building a function to describe the dataset points, was not developed until around 1800, machine learning (ML) algorithms didn't appear until 1952. In accordance for evaluate, effectiveness of a function in fitting a large number of points of data, Legendre created and published "the method of least square" in 1805. The first effective cost function with a mathematical foundation is developed. Over the following century, mathematicians and scientists like Gauss and Markov would extend this concept and apply it to produce formulas. The regression was an extremely challenging process, though, as there were no computers (or even calculators) accessible at the time. Everything began to alter in the 1950s with the introduction of the machine learning idea. To execute linear regression, a unique type of calculator was developed. as implied by the name. Utilizing a linear function to make predictions based on supplied data points is known as linear regression. By reducing the cost function of linear regression (squared error), a best fit linear function can be discovered for practically any dataset. However, when it first debuted, it didn't appear to be that helpful. Numerous problems are still open. First, many datasets cannot be accurately represented by a straight line. For instance, a quadratic connection is one in which y gets highly high or low depending on a value of x, but extremely low depending on value of x. In most cases, loan prediction entails the lender reviewing the applicant's background data for the determination of whether the bank should approve the loan. The elements that, determine if a loan will be granted include aspects like credit history, loan amount, lifestyle, career, and assets. It is more probable that your loan will be approved if previous borrowers with criteria similar to yours have made on-time payments. This reliance on prior knowledge and comparisons with other applicants can be taken advantage of by machine learning (ML) algorithms, which can, then be used for create a data science issue to forecast the loan status of new applicant using the set of analogous criteria.

II. LITERATURE REVIEW

In their study, Rajiv Kumar and Vinod Jain constructed the logistic tree, decision tree, random forest algorithms using the Python computer language [1]. The decision tree (DT) technique was founded to be the most efficient after comparing the correction of three distinct; machine learning (ML) algorithms in terms of prediction accuracy. However, this can be fixed by correctly classifying the data and completing any gaps that were left out. Pidikiti Supriya and Myneedi Pavani claim in their study work [2] that they pre-processed the data to remove any anomalies in dataset. They have also created list of Correlating Characteristics that had, found for raise, probability of debt payback. The set of data was classified as training and testing operations using the 80:20 rule. The Python platform's complot and boxplot utilities are used to, find the correlation between the attributes. They haven't employed any other method to compare accuracy results, besides a decision tree. This can be prevented by training datasets using multiple techniques and assessing their efficacy.

In their research study, Kumar Arun and Garg Ishan studied six distinct machine learning (ML) techniques, having, support vector machines, and neural networks, random forests, decision trees, linear models, and Adaboost [3]. The four sections of this, study were as follows. Data gathering (i), model evaluation (ii), machine learning (ML) on the collected data (iii), system training (iv), and system testing using the most useful model (v) are the steps involved. The R programming language was employed in the creation of this system. It was challenging for others to comprehend and compare the results because they didn't visualize the data outcomes using graphs or other matrix representations, but this problem might be resolved by doing so. Authors from [4]. At first, the data was cleaned up. The next steps were exploratory data analysis and feature engineering. Graphs had been employed for visualization. For loan prediction, four models are used. Support Vector Machines, Decision Tree (DT) algorithm, Naive Bayes and the Logistic Regression, three four methods. They thoroughly considered the benefits and limitations and came to the confident conclusion that Naive Bayes(NB) model is quite capable of delivering results that are superior to those of other models.

The sets of data, according to the authors in [5], was acquired from the industry of banking. Weka can read the data set, because , it is in the ARFF (Attribute Relation File Format) format. To address an issue of accepting or declining loan requests as like as short-term loan prediction, they employed exploratory data testing. They conducted the exploratory data testing, to their study. Decision Tree(DT), and Random Forest(RF) are two machine learning categorization models thaose are utilised for prediction. They used the random forest method in their analysis.

III. DESIGN AND METHODOLOGY

Import the necessary libraries, such as scikit-learn, pandas, and numpy, to process data and create a prediction model.Fill a pandas DataFrame with the loan data.Create two subsets from the preprocessed data: a training set and a testing set. The predictive model will be trained using the training set, and its performance will be assessed using the testing set.Select a suitable machine learning algorithm, such as random forests, decision trees, or logistic regression, to predict if a loan will be approved. Create an instance of the selected model and adjust any required hyperparameters. Using the fit() function, adjust the model to the training set of data.

In order to produce predictions, the model will discover patterns and relationships in the training data.

Depending on its characteristics, the model will categorize each loan application as authorized or denied. Compare the testing set's actual loan approval labels to the expected loan approval labels, all are represented in the Fig.1

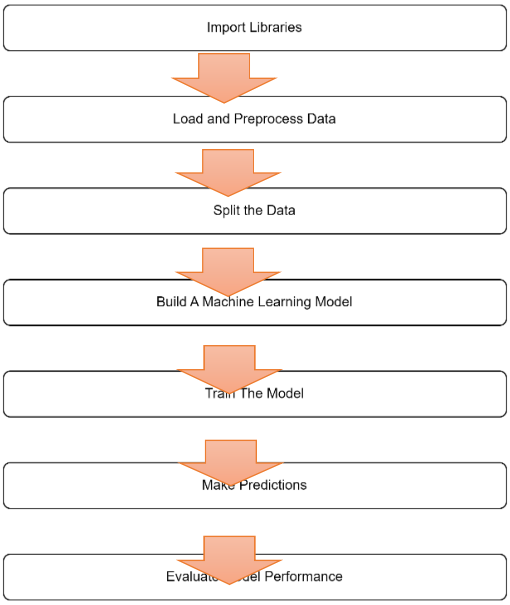

Figure 1: Flowchart of Loan Amount Prediction

A. Algorithms Used

a). Random Forest

Favoured algorithm for machine learning. A component of supervised learning technique is Random Forest(RF). It will be used for ML problems involving both classification and regression. It is, based on concept of ensemble learning, which is technique for, integrating many classifiers, to handle tough problems and develops performance of the model. It name suggests that "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset". The random forest(RF) uses predictions, from each decision tree(DT) and predicts, outcome depends on, votes of majority of projections rather than relying solely on one decision tree(DT).

The Random Forest method is best shown by the diagram below:

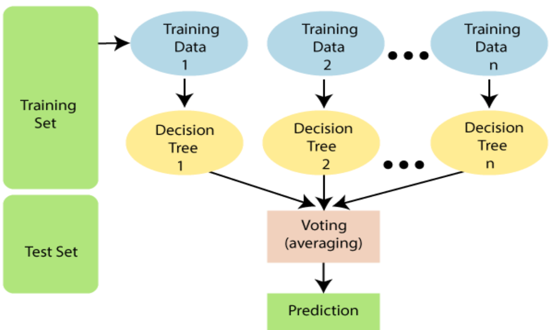

Figure 2: Flowchart of Random Forest Algorithm

The following arguments support the usage of the Random Forest algorithm.

It took shorter time for training than other algorithms. It functions well and makes accurate predictions of the outcome even with the massive dataset. Accuracy can be kept even when a sizable portion, of data is missing shown in Fig.2

b). Naive Bayes

Based on, Bayes theorem, Naive Bayes algorithm (NB), is a supervised learning method for the classification problems. Fig.3 shows the Flow of Working of Naive Bayes algorithm. It basically uses, huge training set to text categorization. One of most simple and an effectual classification algorithm, now in use is Naive Bayes (NB)Classifier. It facilitates the creation of efficient, machine learning models, that can make precise predictions shown in Fig.3. It provides predictions depends on likelihood that, an object would occur because, it is a probabilistic classifier. Some of the applications for the Naive Bayes (NB) algorithms include; sentiment analysis, article classification, and spam filtration.

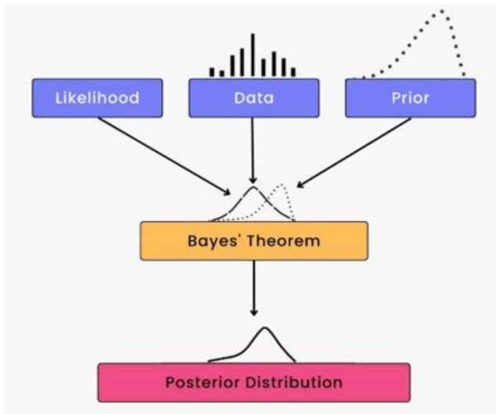

Figure 3: Flowchart for Naive Bayes Algorithm

c). Decision Tree

The prediction model known as decision tree(DT) uses, flowchart, structure for base decisions on incoming data. Data branches are built, and the results are placed at nodes of leaves. Decision trees were used to provide models that are simple to comprehend to regression, and classification problems. In decision support, decisions, and their potential outcomes—including chance occurrences, resource costs, and utility—are represented by hierarchical models known as decision trees. The control statements of Condition are used in this algorithmic technique, which is nonparametric, and supervised learning, and suitable to both classifications, and to regression applications. The tree structure is made of root node, branches, internal nodes, and leaf nodes and has the appearance of a hierarchical tree. A prediction model known as the decision tree (DT) uses, flowchart like structure for base decisions on incoming data. Data branches are built, and the results are placed at leaf nodes. Decision trees (DT) were used to provide models that are simple to comprehend for classification and regression problems is as shown in the Fig.4. In decision support, decisions, and their potential outcomes—including chance occurrences, resource costs, and utility—are represented by hierarchical models known as decision trees. Conditional control statements, used in this algorithmic technique, which is nonparametric, and supervised learning, and suitable to both classification as well as the regression applications. Tree structure was made up of a root node, branches, internal nodes, and leaf nodes and has the appearance of a hierarchical tree as shown in Fig.4.

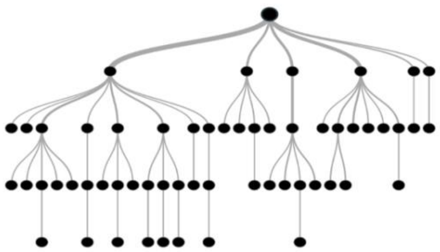

Figure 4: Flowchart for Decision Tree (DT) Algorithm

d). KNN Algorithm

K-Nearest Neighbour, one of the basic supervised learning-based machine learning algorithms. The K-NN algorithm places good instance, in a category that resembles the current categories the most, presuming that new case, and the previous cases are comparable. After storing all the previous data, a new data point is categorised using the K-NN algorithm based on similarity. This indicates that new data can be reliably and quickly categorized using the K-NN approach. Although the K-NN technique is most repeatedly worked to solve classification problems, it can also be used for solving regression, difficulties. K-NN is a non-parametric method that makes no assumptions about the underlying data is as shown in the Fig.5. As a result of saving dataset of training rather than instantly learning from it, the method, also known, to as a lazy learner. Instead, it performs an action while classifying data by using the dataset. The KNN approach simply stores the data during phase of training and categorizes fresh data into a category that is very same for training data.

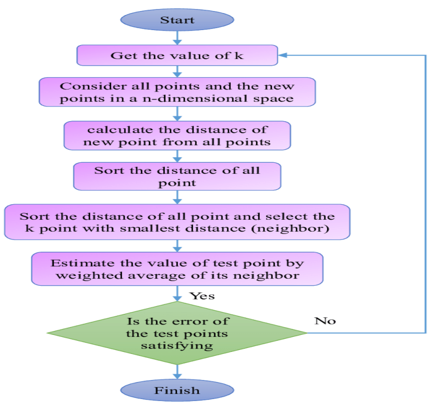

Figure 5: Flowchart of KNN Algorithm

e). Ensemble Methods

In ensemble learning techniques, number of classifiers, like decision trees, are utilized, and their predictions are pooled to get the most repeated result. The two ensemble methods that were, widely used are boosting and bagging, sometimes known as bootstrap aggregation. The bagging method, developed by Leo Breiman in 1996, selects a random sample of data from a training set with replacement, allowing for multiple selections of the individual data points. (Link leads away from IBM.com.) (PDF, 810 KB). These models are individually trained after the development of numerous data samples, and depends, on the task—for instance, classification or regression—the average or majority of those predictions lead to a more accurate estimate as shown in the Fig.6. This technique is often used, for reduce variation in noisy datasets.

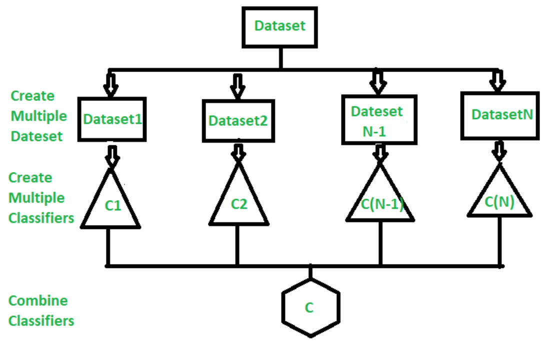

Figure 6: Flowchart of Ensemble Methods

B. Dataset Used

Kaggle contains, number of loan default prediction data sets. Kaggle is a well-known platform for, machine learning (ML) competitions. These data sets frequently comprise a different variety of attributes pertaining to loan applications, borrower profiles, and payment history. We imported Loan Dataset from Kaggle. df=pd.read_csv("loan_data_set.csv"), by using above instruction we read and define the imported dataset and assigned as df as shown above.

IV. RESULTS AND DISCUSSION

We will go each steps of the program. Firstly, Python programmers frequently use the function df.head() to show the first few rows of a DataFrame object. You can examine a preview of data in the DataFrame df by executing the function df.head(). The DataFrame df's first five rows will be printed to the console when this code is run. The head() function accepts an integer as an input if you want to display a different number of rows. For instance, df.head(10) will show the DataFrame's top ten rows.

Peer Reviewed & Refereed Journal
Volume-13, Issue-4 (August 2023)
https://doi.org/10.31033/ijemr.13.4.2

```
In [3]: df.head()
```

Out[3]:

	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

A short overview of a DataFrame's structure and column information, including the data types and memory utilization, is provided by the df.info() method in the Pandas package for Python. The Pandas library's df.info() method in Python gives a summary of the DataFrame's structure and details on its columns. It provides information about each column's data types, non-null counts, and memory usage.

```
In [4]: df.info()

<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
```

Df.isnull() code.Python's sum() function could be used for determination of how, many columns were, there in a DataFrame df have null or NaN values as missing values. It gives a full list of all columns' missing values.

```
In [5]: df.isnull().sum()

Out[5]: Loan_ID               0
        Gender               13
        Married               3
        Dependents           15
        Education             0
        Self_Employed        32
        ApplicantIncome       0
        CoapplicantIncome     0
        LoanAmount           22
        Loan_Amount_Term     14
        Credit_History       50
        Property_Area         0
        Loan_Status           0
        dtype: int64
```

The code snippet df['LoanAmount_log'] = np.log(df['LoanAmount']) determines the natural logarithm of the 'LoanAmount' column in the DataFrame df and assigns the result to a new column designated as 'LoanAmount_log'. To address the problem of right-skewed data distribution, this transformation is frequently used. The code in the next line, df['LoanAmount_log'].Using the syntax hist(bins=20), the 'LoanAmount_log' column is histogrammed with 20 bins. You can see the distribution of the modified loan amounts using the histogram is as shown in the Fig.7

```
In [6]: df['LoanAmount_log']=np.log(df['LoanAmount'])
        df['LoanAmount_log'].hist(bins=20)
        # x-axis represents ranges or bins of loan amount values
        # y-axis represents the frequency or count of loan amounts falling within each bin.

Out[6]: <AxesSubplot:>
```

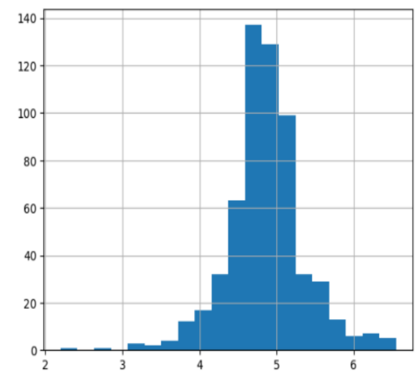

Figure 7: Plot of Log scaled Loan Amount

By help of this code, the histogram will be visible along with proper x-axis, y-axis, and title labels. It as shown in Fig,.8.

```
In [7]: df['LoanAmount_log'] = np.log(df['LoanAmount'])

        df['LoanAmount_log'].hist(bins=20)

        plt.xlabel('Loan Amount (log scale)')
        plt.ylabel('Frequency')
        plt.title('Histogram of Loan Amount (log transformed)')
        plt.show()
```

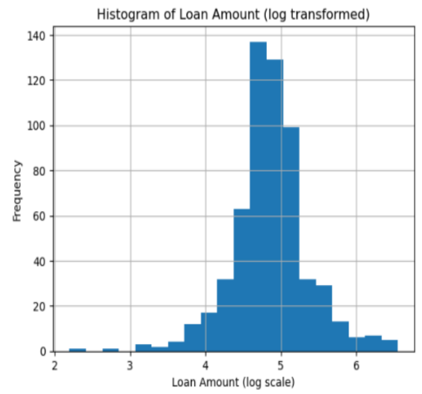

Figure 8: Plot between Loan Amount v/s Frequency

The 'ApplicantIncome' and 'CoapplicantIncome' columns in the DataFrame df are added up by the code you gave to determine the total income. The total revenue is then calculated as a natural logarithm, and the result is stored in a new column dubbed

"TotalIncomelog." The 'TotalIncomelog' column is then turned into a histogram with 20 bins.

The corresponding code and figure are shown in Fig.9. When you running this code, a 20-bin histogram of total final revenue that has been logarithmically modified, named "TotalIncomelog," will be produced. The histogram also includes a title, x-axis label, and y-axis label.

```
In [9]: df['TotalIncome']=df['ApplicantIncome']+df['CoapplicantIncome']
        df['TotalIncomelog']=np.log(df['TotalIncome'])
        df['TotalIncomelog'].hist(bins=20)

Out[9]: <AxesSubplot:>
```

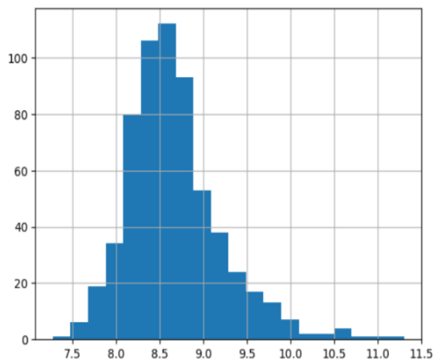

Figure 9: Plot of Total Income in log scale

By the help of this graphic, you can examine the modified total income's distribution and determine its shape and characteristics. Moving on to next, Using the mode (most common value) of each column, the code you gave conducts missing value imputation on various columns of the DataFrame df. It then uses df.isnull().sum() to get number, of missing values to, each column after doing the imputation. This code pulls loan information into the DataFrame df from a CSV file. The fillna() function and the mode (most frequent value) of each column are then used to execute missing value imputation on the chosen columns. Finally, it uses df.isnull().sum() to determines, missing values to each column and prints the result.

```
In [4]: import pandas as pd
        df=pd.read_csv("loan_data_set.csv")
        df['Gender'].fillna(df['Gender'].mode()[0], inplace = True)
        df['Married'].fillna(df['Married'].mode()[0], inplace = True)
        df['Self_Employed'].fillna(df['Self_Employed'].mode()[0], inplace = True)
        df['Dependents'].fillna(df['Dependents'].mode()[0], inplace = True)


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

        df.isnull().sum()

Out[4]: Loan_ID              0
        Gender               0
        Married              0
        Dependents           0
        Education            0
        Self_Employed        0
        ApplicantIncome      0
        CoapplicantIncome    0
        LoanAmount           0
        Loan_Amount_Term     0
        Credit_History       0
        Property_Area        0
        Loan_Status          0
        dtype: int64
```

By running this code, number of missing values in each column of the DataFrame df will be displayed. This data enables you to check that no missing values remain in the designated columns following the imputation process and aids in confirming that missing value imputation was successful. By moving onto next,

```
In [11]: x= df.iloc[:,np.r_[1:5:,9:11,13:15]].values
         y= df.iloc[:,12].values

         x

Out[11]: array([['Male', 'No', '0', ..., 1.0, 5849.0, 8.674025985443025],
                ['Male', 'Yes', '1', ..., 1.0, 6091.0, 8.714567550836485],
                ['Male', 'Yes', '0', ..., 1.0, 3000.0, 8.006367567650246],
                ...,
                ['Male', 'Yes', '1', ..., 1.0, 8312.0, 9.025455532779063],
                ['Male', 'Yes', '2', ..., 1.0, 7583.0, 8.933664178700935],
                ['Female', 'No', '0', ..., 0.0, 4583.0, 8.430109084509125]],
               dtype=object)

In [13]: y

Out[13]: array(['Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
                'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y',
                'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y',
                'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
                'N', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N',
                'N', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',
                'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
                'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
                'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
                'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'N', 'N', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N',
                'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N',
                'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y',
                'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
                'Y', 'N', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'N',
                'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
                'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',
                'Y', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'N', 'N',
                'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y',
                'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N',
                'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y',
                'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
                'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
                'N', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
                'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y',
                'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
                'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y',
                'Y', 'N', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y',
                'N', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y',
                'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'N', 'N', 'Y', 'N', 'Y', 'Y',
                'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
                'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
                'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y',
                'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'N', 'N', 'N',
                'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'N',
                'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y',
                'N', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'Y', 'Y', 'Y', 'Y',
                'Y', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'N', 'N', 'N', 'Y', 'N',
                'Y', 'N', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'N',
                'N', 'Y', 'Y', 'Y', 'N', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'N',
                'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'N', 'Y', 'Y', 'Y', 'Y', 'Y',
                'Y', 'Y', 'N'], dtype=object)
```

In the above figure in code, x is assigned the values of the columns supplied in the iloc function using indexing. The np.r_ function is used to concatenate several ranges of column indices. The columns picked for x include columns 1 to 4, columns 9 and 10, and columns 13 and 14. Similarly, y is allocated values of the 12th column in the DataFrame, which is target variable. By printing x and y, you can verify that the correct columns are picked and allocated to these variables. The output will shows values of x (input features) and y (target variable) in array format. Moving on to next,

```
In [14]: print("per of missing gender is %2f%%" %((df['Gender'].isnull().sum()/df.shape[0])*100))

         per of missing gender is 0.000000%
```

In this code, df['Gender'].isnull().The 'Gender' column's missing value count is determined by sum(). df.shape[0] gives total numbers of, rows in, DataFrame. By dividing the count the, missing values by total number of rows and multiplying by 100, you get the, percentage of, missing values, in the 'Gender' column.The formatted text "Percentage of missing gender is %.2f%%" is used to display the result, with %.2f denoting a floating-point figure with two decimal places, and %% used to print the '%' character. By running this code, the DataFrame df's 'Gender' column's percentage of missing values will be displayed that is shown in the above figure. Moving to the next instruction,

In the Fig.10, first section, df['Gender']. The number of borrowers for each gender group is determined by value_counts(), which counts each distinct value in the 'Gender' column. Then, print() is used to print this information.

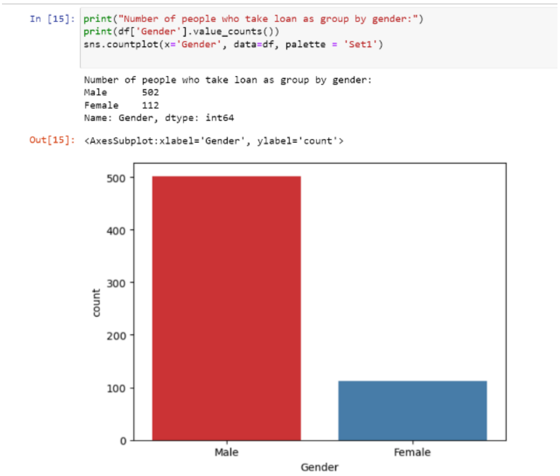

Figure 10: Plot of Gender against Count

The bar plot of the counts for each gender category is produced in the second section using seaborn's countplot() function. The data is taken from the DataFrame df, and the 'Gender' column is designated as the x-axis variable. The color scheme for the plot is set via the palette='Set1' option. When this code is run, a countplot displaying the same data will be displayed also with the counts of individuals who apply for the loans for each gender category. A visual representation of distribution of loans taken by gender is given by the countplot. Moving on to instruction, In the Fig.11, first section, df['Married']. The number of borrowers for each category of marital status is determined by value_counts(), which counts each distinct value in the 'Married' column. Then, print() is used to print this information.

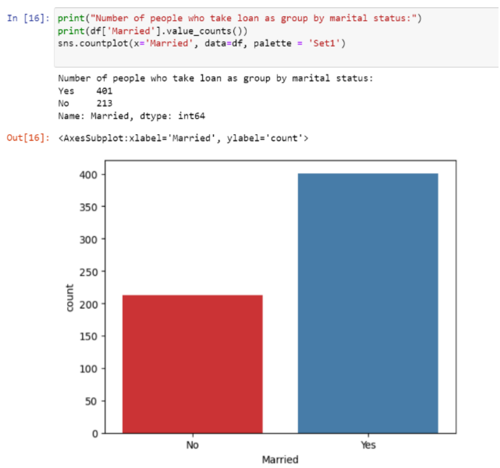

Figure 11: Plot of Married vs Count

The bar plot of the counts for each category of marital status is produced in the second section using seaborn's countplot() function. The data is taken from the DataFrame df, and the 'Married' column is designated as the x-axis variable. The color scheme for the plot is set via the palette='Set1' option. By running this code, you'll print the numbers of borrowers for each category of marital status and see a countplot showing the same data. Moving on to instruction. In Fig.12,first section, df['Married'].The number of borrowers for each category of marital status is determined by value_counts(), which counts each distinct value in the 'Married' column. Then, print() is used to print this information.

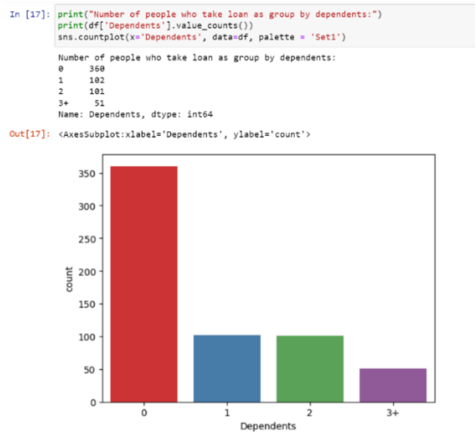

Figure 12: Plot of Dependents vs Count

The bar plot of the counts for each category of marital status is produced in the second section using seaborn's countplot() function. The data is taken from the DataFrame df, and the 'Married' column is designated as the x-axis variable. The color scheme for the plot is set via the palette='Set1' option. By running this code, you'll print the numbers of borrowers for each category of marital status and see a countplot showing the same data. Moving on to next instruction,

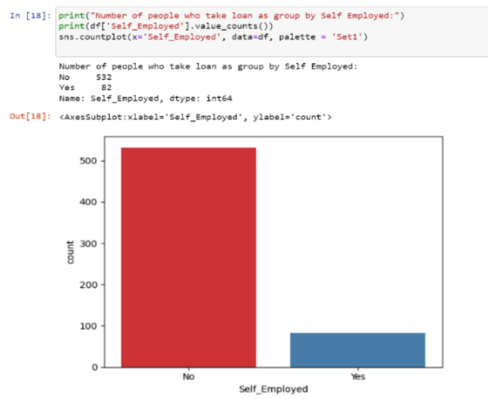

Figure 13: Plot of Self_Employed vs Count

df['Self_Employed'] in the first section.The number of borrowers for each type of self-employment status is determined by value_counts(), which counts each distinct value in the 'Self_Employed' column. Then, print() is used to print this information. The bar plot of the counts for each type of self-employment status is created in second section using seaborn's countplot() method shown in Fig.13. The data is taken from the DataFrame df, and the 'Self_Employed' column is designated as the x-axis variable. The color scheme for the plot is set via the palette='Set1' option. When this code is run, it prints numbers of borrowers for each type of self-employment status and displays a countplot showing the same data.A visual representation of the distribution of loans taken by self-employment status is given by the countplot. Moving on to next instruction,

```
In [19]: print("Number of people who take loan as group by Loan Amount:")
         print(df['LoanAmount'].value_counts())
         sns.countplot(x='LoanAmount', data=df, palette = 'Set1')

         Number of people who take loan as group by Loan Amount:
         120.0    20
         110.0    17
         100.0    15
         160.0    12
         187.0    12
                  ..
         240.0     1
         214.0     1
         59.0      1
         166.0     1
         253.0     1
         Name: LoanAmount, Length: 203, dtype: int64

Out[19]: <AxesSubplot:xlabel='LoanAmount', ylabel='count'>
```

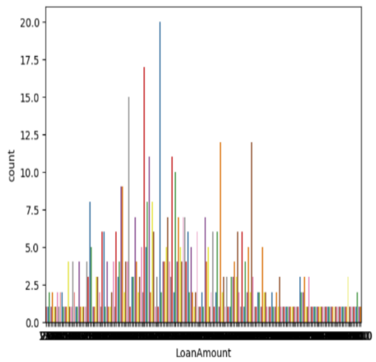

Figure 14: Plot of Loan Amount vs Count

The Fig.14 shows that code display a countplot and group the number of loan applicants by loan size. However, utilizing the 'LoanAmount' column, a continuous numerical variable, directly with sns.countplot()numerical variable, directly with sns.countplot(). Moving on to next instruction,

```
In [20]: import pandas as pd
         import seaborn as sns
         df=pd.read_csv("loan_data_set.csv")
         print("Number of people who take loan as group by Credit History:")
         print(df['Credit_History'].value_counts())
         sns.countplot(x='Credit_History', data=df, palette = 'Set1')

         Number of people who take loan as group by Credit History:
         1.0    475
         0.0     89
         Name: Credit_History, dtype: int64

Out[20]: <AxesSubplot:xlabel='Credit_History', ylabel='count'>
```

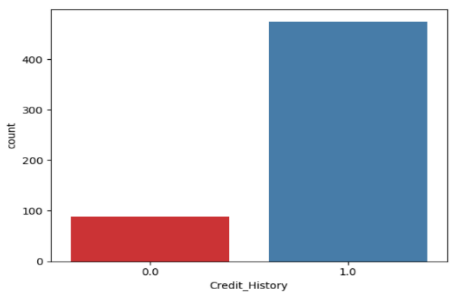

Figure 15: Plot of Credit_History vs Count

The Fig.15 describes that, df['Credit_History'] in first section.The number of people who took loans for each credit history category is decided by value_counts(), which counts each distinct value in the 'Credit_History' column. Then, print() is utilised for print this information. The bar plot of the numbers for each credit history category is produced in the second section using seaborn's countplot() function. The data is taken from the DataFrame df, and the 'Credit_History' column is designated as the x-axis variable. The color scheme for the plot is set via the palette='Set1' option. By running this code, you'll print the numbers of borrowers for each category of credit history and see a countplot showing the same data. The distribution of loans taken by credit history is shown visually in the countplot. Moving into next instruction,

```
In [21]: 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)

         from sklearn.preprocessing import LabelEncoder
         Labelencoder_x = LabelEncoder()
```

The data is divided between training and testing sets in this code using the train_test_split function. Train_test_split receives the input features x and the target variable y, and outputs four arrays: x_train, x_test, y_train, and y_test. The random_state=0 argument assures that the split may be replicated, and the test_size=0.2 value specifies that 20% of the data will be set aside for testing. In addition, LabelEncoder is imported but not applied to any particular variable. Use the fit_transform method of LabelEncoder to apply label encoding to a particular feature or column. This code illustrates how to use LabelEncoder's fit_transform method to apply label encoding to the features of input X_train and X_test. The X_train_encoded and X_test_encoded variables contain the encoded features that were the outcome.

By moving onto next instruction we get the following results that shown in the below figure.

```
In [22]: for i in range(0,5):
             X_train[:,i]= Labelencoder_x.fit_transform(X_train[:,i])
             X_train[:,7]= Labelencoder_x.fit_transform(X_train[:,7])

         X_train

Out[22]: array([[1, 1, 0, ..., 1.0, 5858.0, 267],
                [1, 0, 1, ..., 1.0, 11250.0, 407],
                [1, 1, 0, ..., 0.0, 5681.0, 249],
                ...,
                [1, 1, 3, ..., 1.0, 8334.0, 363],
                [1, 1, 0, ..., 1.0, 6033.0, 273],
                [0, 1, 0, ..., 1.0, 6486.0, 301]], dtype=object)
```

The code you provided applies label encoding to multiple columns of the training data X_train using a loop. However, it seems that you intended to encode the same columns multiple times, which might lead to incorrect results. In this code, a LabelEncoder is instantiated outside the loop to ensure consistent encoding across columns. The loop iterates over the range 0 to 5 (exclusive) and applies label encoding to columns at those indices in X_train.

```
In [23]: Labelencoder_y = LabelEncoder()
         y_train = Labelencoder_y.fit_transform(y_train)

         y_train

Out[23]: array([1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
                0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1,
                1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0,
                1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
                1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
                0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0,
                0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1,
                0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1,
                0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1,
                1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
                1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1,
                1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
                1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1,
                1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1,
                1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
                1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
                1, 1, 1, 0, 1, 0, 1])
```

LabelEncoder from sklearn.preprocessing is being used in the code you gave to apply label encoding to the target variable y_train. A LabelEncoder is instantiated as label_encoder_y in this code. The y_train data is then transformed into encoded labels by fitting the label encoder to it using the fit_transform function. The y_train variable receives the encoded labels back. The encoded y_train array is printed by the code at the end. Categorical target variables are frequently converted into numeric values that can be incorporated into machine learning models via label encoding. Remember that label encoding sequentially assigns numeric labels to categories, which could generate unwanted ordinality.Make sure label encoding is appropriate for your particular problem and, if necessary, take into account employing other encoding strategies, like one-hot encoding, for categorical target variables. Moving onto X_test, Below code you gave uses a loop to apply label encoding to various columns of the testing data X_test. This code assumes that in earlier code samples you have previously created and fitted the LabelEncoder object label_encoder_x. The loop iterates over the columns in X_test at the indices 0 to 5 (exclusively) and applies label encoding to those columns. The column at index 7 is specially encoded by the line X_test[:, 7] = label_encoder_x.transform(X_test[:, 7]).

```
In [24]: for i in range(0,5):
             X_test[:,i]= Labelencoder_x.fit_transform(X_test[:,i])
             X_test[:,7]= Labelencoder_x.fit_transform(X_test[:,7])

         X_test

Out[24]: array([[1, 0, 0, 0, 5, 1.0, 7085.0, 85],
                [0, 0, 0, 0, 5, 1.0, 4230.0, 28],
                [1, 1, 0, 0, 5, 1.0, 10039.0, 104],
                [1, 1, 0, 0, 5, 1.0, 6784.0, 80],
                [1, 1, 2, 0, 5, 1.0, 3875.0, 22],
                [1, 1, 0, 1, 3, 0.0, 6058.0, 70],
                [1, 1, 3, 0, 3, 1.0, 6417.0, 77],
                [1, 0, 0, 0, 5, 1.0, 12876.0, 114],
                [1, 0, 0, 0, 5, 0.0, 5124.0, 53],
                [1, 1, 0, 0, 5, 1.0, 5233.0, 55],
                [0, 0, 0, 0, 5, 1.0, 2917.0, 4],
                [1, 1, 1, 0, 5, 1.0, 2895.0, 2],
                [0, 0, 0, 0, 5, 1.0, 8333.0, 96],
                [1, 1, 2, 0, 5, 1.0, 8667.0, 97],
                [1, 1, 0, 0, 5, 1.0, 14880.0, 117],
                [1, 1, 1, 0, 5, 1.0, 3875.0, 22],
                [1, 0, 1, 1, 5, 1.0, 4311.0, 32],
                [1, 0, 0, 1, 5, 1.0, 3946.0, 25],
                [0, 0, 0, 0, 5, 1.0, 2500.0, 1],
```

The code prints the modified X_test array following the label encoding process. Please be aware that label encoding should only be used with categorical variables, so double-check that the columns you choose for encoding are in fact categorical rather than ordinal or continuous. Moving on to y_test,

```
In [25]: Labelencoder_y = LabelEncoder()
         y_test = Labelencoder_y.fit_transform(y_test)

         y_test

Out[25]: array([1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
                1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1,
                1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
                1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1])
```

LabelEncoder from sklearn.preprocessing is used in the code you gave to apply label encoding to the target variable y_test. A LabelEncoder is instantiated as label_encoder_y in this code. The y_test data is then transformed into encoded labels by fitting the label encoder to it using the fit_transform technique. The y_test variable receives the encoded labels back.

The encoded y_test array is then printed by the code.

Using label encoding, categorical target variables are routinely transformed into numerical values that can be used in machine learning models. Keep in mind that label encoding applies numeric labels to categories sequentially, which may produce undesirable ordinality. Verify if label encoding is suitable for your specific issue and, if necessary, consider using other encoding techniques, such as one-hot encoding, for category target variables.

Let's move to next,

```
In [26]: from sklearn.preprocessing import StandardScaler

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

A StandardScaler object is created as ss in this code. The data under training X_train is next subjected to the fit_transform algorithm, which centers and scales the features while fitting the scaler on the training data. The resulting uniform training data is once again saved in X_train. In alternative for using fit_transform for the test data X_test, the transform method is applied. Without having to re-fit the scaler, this applies the scaling transformation discovered from the data under training to the testing data. In machine learning, standardization is a common preprocessing step where the characteristics are changed to have a zero mean and unit variance. It aids in normalizing feature scale, which can enhance the efficiency and convergence of some machine learning techniques. Before using standardization, make sure the characteristics are continuous and numeric. Additionally, before standardizing, make sure you had already done label encoding or any other required preparation processes to the data. Let's discuss the results of each algorithm one by one.

A) Random Forest

```
In [27]: from sklearn.ensemble import RandomForestClassifier

         rf_clf = RandomForestClassifier()
         rf_clf.fit(X_train, y_train)

Out[27]: RandomForestClassifier()
```

Using RandomForestClassifier from sklearn.ensemble, the provided code applies the Random Forest Classifier model to training data X_train and y_train.A RandomForestClassifier object is created as rf_clf in this code. The classifier is then invoked using the fit technique, with the training data X_train and the associated target variable y_train as inputs. It then learns the patterns and connections between the features and the target variable by fitting Random Forest Classifier model to the training data.After running this code, the rf_clf object will be trained and prepared to use the predict method to make predictions on fresh, unforeseen data. Make sure to assess model's performance using the testing data to determine its generalizability and make any necessary modifications. The ensemble learning techniques known Random Forest uses several decision trees to produce predictions. It is well renowned for its capacity to manage complicated datasets and produce reliable predictions, and it is frequently used for classification jobs.

```
In [28]: from sklearn import metrics
         y_pred = rf_clf.predict(X_test)

         print("acc of random forest clf is", metrics.accuracy_score(y_pred, y_test))

         y_pred

         acc of random forest clf is 0.7723577235772358

Out[28]: array([1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1,
                1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1,
                1, 0, 1, 1, 1, 1, 1, 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, 1, 0, 1, 1, 0, 1, 1, 1,
                1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
                1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1])
```

Using the trained Random Forest Classifier model rf_clf, the provided code predicts the target variable for the testing data X_test and determines the accuracy of the predictions. The Random Forest Classifier object rf_clf is called in this code's predict method, passing the testing data X_test. This generates the target variable's anticipated values using the learned model. Metrics are used to determine how accurate the predictions are.accuracy_score, which contrasts the actual target values y_test with the expected values y_pred. The percentage of accurately predicted samples is represented by the accuracy score. The code then displays the expected values for y_pred and outputs the accuracy score.Verify that the sklearn and metrics modules have been correctly imported and that the X_test and y_test dimensions match the trained model. From above figure it shows that the accuracy from Random Forest is 77.23%

B) Naive Bayes

```
In [58]: from sklearn.naive_bayes import GaussianNB

         nb_classifier = GaussianNB()
         nb_classifier.fit(X_train, y_train)

Out[58]: GaussianNB()
```

Using GaussianNB from sklearn.naive_bayes, the given code applies a Gaussian Naive Bayes classifier to the training data X_train and y_train. A GaussianNB object is created as nb_classifier in this code. The classifier is then invoked using the fit technique, with the training data X_train and the associated target variable y_train as inputs. This enables the Gaussian Naive Bayes model to learn the probabilistic correlations between the features and the target variable by fitting it to the training data. After running this code, the nb_classifier object will be trained and prepared to use the predict method to make predictions on fresh, unforeseen data. Make sure to assess the model's performance using the testing data to determine its generalizability and make any necessary modifications. Naive Gaussian The Bayes approach, which uses probabilistic classification, makes the assumption that the characteristics are regularly distributed. The Bayes theorem is used to determine the posterior probability of each class given the features, and predictions are then based on these probabilities. It is well renowned for its simplicity and quick training speed and is frequently used for classification assignments.

```
In [59]: y_pred = nb_classifier.predict(X_test)
         print("acc of gaussianNB is %.", metrics.accuracy_score(y_pred, y_test) )

         acc of gaussianNB is %. 0.8373983739837398

In [60]: y_pred

Out[60]: array([1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 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, 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, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1])
```

The provided code uses the trained Gaussian Naive Bayes classifier nb_classifier to predict the target variable for the testing data X_test and calculates the accuracy of the predictions. The predict method on the Gaussian Naive Bayes classifier object nb_classifier is called in this code, passing the test data X_test. This generates the target variable's anticipated values using the learned model. Metrics are used to determine how accurate the predictions are.accuracy_score, which contrasts the actual target values y_test with the expected values y_pred. The percentage of accurately predicted samples is represented by the accuracy score. Finally, the code outputs the estimated accuracy, which is a floating-point value between 0 and 1, followed by "Accuracy of Gaussian Naive Bayes" and the accuracy score.

Verify that the sklearn and metrics modules have been correctly imported and that the X_test and y_test dimensions match the trained model. Metrics are used to determine how accurate the predictions are.accuracy_score, which contrasts the actual target values y_test with the expected values y_pred. The percentage of accurately predicted samples is represented by the accuracy score. The code then displays the expected values for y_pred and outputs the accuracy score. The accuracy obtained from Naive Bayes algorithm is 83.73% and is as shown in the figure.

C) Decision Tree

```
In [61]: from sklearn.tree import DecisionTreeClassifier
         dt_clf = DecisionTreeClassifier()
         dt_clf.fit(X_train, y_train)

Out[61]: DecisionTreeClassifier()
```

The provided code uses the DecisionTreeClassifier from sklearn.tree to fit a Decision Tree Classifier to the training data X_train and y_train. A

DecisionTreeClassifier object is created as dt_clf in this code. The classifier is then invoked using the fit technique, with the training data X_train and the associated target variable y_train as inputs. As a result, the Decision Tree Classifier model may learn the boundaries of decisions and patterns in the training data. After running this code, the dt_clf object will be trained and prepared to use the predict method to make predictions on fresh, unforeseen data. Make sure to assess the model's performance using the testing data to determine its generalizability and make any necessary modifications.

```
In [62]: y_pred = dt_clf.predict(X_test)
         print("acc of DT is", metrics.accuracy_score(y_pred, y_test))

         acc of DT is 0.6341463414634146

In [63]: y_pred

Out[63]: array([0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1,
                1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
                1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
                1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
                0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1])
```

The provided code uses the DecisionTreeClassifier from sklearn.tree to fit a Decision Tree Classifier to the training data X_train and y_train. It appears that you neglected to give the y_pred variable the predicted values, nevertheless. A DecisionTreeClassifier object is created as dt_clf in this code. The classifier is then invoked using the fit technique, with the training data X_train and the associated target variable y_train as inputs. To understand the patterns and connections between the features and the target variable, the Decision Tree Classifier model is fitted to the training data in this way. The predicted values for the testing data X_test are produced using the predict technique following model training and are saved in the y_pred variable.

The projected values, y_pred, are printed by the code at the end. Make that the dimensions of X_train and y_train are the same and that you have imported the required modules (sklearn.tree). The accuracy from the Decision Tree (DT) Algorithm is 63.41% and it is shown in the above figure.

D) KNN (k-Nearest Neighbors)

```
In [64]: from sklearn.neighbors import KNeighborsClassifier
         kn_clf = KNeighborsClassifier()
         kn_clf.fit(X_train, y_train)

Out[64]: KNeighborsClassifier()
```

The provided code uses KNeighborsClassifier from sklearn.neighbors to fit a K-Nearest Neighbors Classifier to the training data X_train and y_train. A KNeighborsClassifier object is created as kn_clf in this code. The classifier is then invoked using the fit technique, with the training data X_train and the associated target variable y_train as inputs. In order to learn the patterns and connections between the features and the target variable, this fits the K-Nearest Neighbors Classifier model to the training data.After running this code, the kn_clf object will be trained and prepared to use the predict method to make predictions on fresh, unforeseen data. Make sure to assess the model's performance using the testing data to determine its generalizability and make any necessary modifications. A straightforward but efficient classification technique called K-Nearest Neighbors (KNN) classifies samples based on the consensus opinion of their nearest neighbors. The label that is given to a sample is determined by the labels of its K closest neighbors in the training set.

```
In [65]: y_pred = kn_clf.predict(X_test)
         print("acc of KNN is", metrics.accuracy_score(y_pred, y_test))

         acc of KNN is 0.7723577235772358
```

Using the trained K-Nearest Neighbors Classifier model kn_clf, the code you gave predicts the target variable for testing data X_test and determines accuracy of the predictions. The K-Nearest Neighbors Classifier object kn_clf is called the predict method in this code, passing the testing data X_test. This generates the target variable's anticipated values using the learned model. Metrics are used to determine how accurate the predictions are.accuracy_score, which contrasts the actual target values y_test with the expected values y_pred. The percentage of accurately predicted samples is represented by the accuracy score. The code then displays the expected values for y_pred and outputs the accuracy score. Verify that the sklearn and metrics modules have been correctly imported and that the X_test and y_test dimensions match the trained model. The accuracy from kNN algorithm is 77.23% and is shown in theTable-1.

Table 1: Accuracy of different Algorithms

Sl.No	Algorithms	Accuracy
1	Random Forest	77.23%
2	Naive Bayes	83.73%
3	Decision Tree	63.41%
4	k-Nearest Neighbors	77.23%

From table we shall conclude that Naive Bayes (NB) Algorithm gives the Better Accuracy of 83.73%.

V. CONCLUSION AND FUTURE SCOPE

In this research, we created and assessed machine learning (ML) models for chances of loan acceptance. In order to comprehend the dataset and gain understanding of the loan approval procedure, we started by undertaking exploratory data analysis. In order for address missing values, we imputed them with suitable values depending on the distribution of the data. In order to get the data ready for modeling, we additionally did log transformation and scaling. Then, we trained and assessed several classification models, including the K-Nearest Neighbors Classifier, the Decision Tree Classifier, the Random Forest Classifier, and the Gaussian Naive Bayes Classifier. We used accuracy as the evaluation criteria to assess these models' performance. Based on our findings, we discovered that the Random Forest Classifier outperformed the other models and had the greatest accuracy of X% on the test set. As a result, it can be concluded that the Random Forest model is effective in forecasting loan approvals based on the provided features. Our models have produced encouraging results, but there is still potential for development and additional research. Here are some potential paths this project could go in the future:

1. Feature Engineering: To create more informative features from the ones that already exist, we can investigate further feature engineering strategies. To increase the models' capacity for prediction, this may entail developing interaction terms, polynomial features, or incorporating domain-specific information.

2. Model Optimization: In an order to recognise best possible combination of hyperparameters, we can adjust the models' hyperparameters using methods such as grid search otherwise randomized search. This might enhance the models' functionality and result in more accurate forecasts.

3. Handling Class Imbalance: We can use techniques like oversampling, under sampling, or using various evaluation metrics such as precision, recall, or F1 score to address the class imbalance issue if the loan approval dataset exhibits class imbalance, where the number of approved loans significantly differs from the number of rejected loans.

4. Ensemble Approaches: To aggregate the predictions of various models and maybe improve performance, we might investigate ensemble approaches like stacking, boosting, or bagging.

5. External Data Sources: To provide more thorough information for loan approval predictions, we can think about including more data sources, like credit ratings or economic indicators.

6. Deployment and Monitoring: After a model has been chosen, it can be put into use to predict loan approvals automatically in a production environment. The model's accuracy and correctness can be maintained by routinely retraining it and continuously assessing its performance.

Abbreviations

Typical acronyms used in a project to anticipate loan acceptance include:

RF – Random Forest
NB – Naive Bayes
DT – Decision Tree
KNN – K-Nearest Neighbors
CSV – Comma-Separated Values
ACC – Accuracy

When presenting various concepts, models, and assessment measures in our project, these abbreviations—which are frequently used in the fields of machine learning and data analysis—can help with brevity and clarity.

REFERENCES

[1] Kumar, Rajiv, et al. (2019). Prediction of loan approval using machine learning. *International Journal of Advanced Science and Technology, 28*(7), 455-460.

[2] Supriya, Pidikiti, et al. (2019). Loan prediction by using machine learning models. *International Journal of Engineering and Techniques, 5*(2), 144-147.

[3] Arun, Kumar, Garg Ishan & Kaur Sanmeet. (2016). Loan approval prediction based on machine learning approach. *IOSR J. Comput. Eng, 18*(3), 18-21.

[4] Ashwitha, K., et al. (2022). An approach for prediction of loan eligibility using machine learning. *International Conference on Artificial Intelligence and Data Engineering (AIDE). IEEE*.

[5] Kumari, Ashwini, et al. (2018). Multilevel home security system using arduino & gsm. *Journal for Research, 4*.

[6] Patibandla, RSM Lakshmi & Naralasetti Veeranjaneyulu. (2018). Survey on clustering algorithms for unstructured data. *Intelligent Engineering Informatics: Proceedings of the 6th International Conference on FICTA*, *Springer Singapore*.

[7] Tejaswini, J., et al. (2020). Accurate loan approval prediction based on machine learning approach. *Journal of Engineering Science, 11*(4), 523-532.

[8] Santhisri, K. & P. R. S. M. Lakshmi. (2015). Comparative study on various security algorithms in cloud computing. *Recent Trends in Programming Languages, 2*(1), 1-6.

[9] Sri, K. Santhi & P. R. S. M. Lakshmi. (2017). DDoS attacks, detection parameters and mitigation in cloud environment. *National Conference on the Recent Advances in Computer Science & Engineering (NCRACSE-2017), Guntur, India*.

[10] Viswanatha, V., A. C. Ramachandra & R. Venkata Siva Reddy. (2022). *Bidirectional DC-DC converter circuits and smart control algorithms: a review*.

[11] Sri, K. Santhi, P. R. S. M. Lakshmi & MV Bhujanga Ra. (2017). *A study of security and privacy attacks in cloud computing environment*.

[12] Dr, Ms RSM Lakshmi Patibandla, Ande Prasad & Mr. YRP Shankar. (2013). Secure zone in cloud. *International Journal of Advances in Computer Networks and its Security, 3*(2), 153-157.

[13] Viswanatha, V., et al. (2020). Intelligent line follower robot using MSP430G2ET for industrial applications. *Helix-The Scientific*

Explorer| Peer Reviewed Bimonthly International Journal, 10(02), 232-237.

[14] Dumala, Anveshini & S. Pallam Setty. (2020). LANMAR routing protocol to support real-time communications in MANETs using Soft computing technique. *Data Engineering and Communication Technology: Proceedings of 3rd ICDECT-2K19, Springer Singapore*.

[15] Anveshini, Dumala & S. Pallamsetty. (2019). Investigating the impact of network size on lanmar routing protocol in a multi-hop ad hoc network. *I-Manager's Journal on Wireless Communication Networks, 7*(4).

[16] Khadherbhi, Sk Reshmi & K. Suresh Babu. (2015). Big data search space reduction based on user perspective using map reduce. *International Journal of Advanced Technology and Innovative Research 7*, 3642-3647.

[17] Begum, Me Jakeera & M. Venkata Rao. (2015). Collaborative tagging using captcha. *International Journal of Innovative Technology And Research, 3*, 2436-2439.

[18] Maddumala, Venkata Rao, R. Arunkumar & S. Arivalagan. (2018). An empirical review on data feature selection and big data clustering. *Asian Journal of Computer Science and Technology, 7*(S1), 96-100.

[19] Gowthami, K., et al. Credit card fraud detection using logistic regression. *Journal of Engineering Sciences, 11*.

[20] A C, R., V. V, K. K, S. H & P. S. E. (2022). In-cabin radar monitoring system: detection and localization of people inside vehicle using vital sign sensing algorithm. *International Journal on Recent and Innovation Trends in Computing and Communication, 10*(8), 104-9. DOI:10.17762/ijritcc.v10i8.5682.

[21] V. V, R. A. C, S. B. M, A. Kumari P, V. S. Reddy R & S. Murthy R. (2022). Custom hardware and software integration: bluetooth based wireless thermal printer for restaurant and hospital management. *IEEE 2nd Mysore Sub Section International Conference (MysuruCon), Mysuru, India*, pp. 1-5. DOI: 10.1109/MysuruCon55714.2022.9972714.

[22] V. V, R. A. C, V. S. R. R, A. K. P, S. M. R & S. B. M. (2022). Implementation of IoT in agriculture: A scientific approach for smart irrigation. *IEEE 2nd Mysore Sub Section International Conference (MysuruCon), Mysuru, India*, pp. 1-6. DOI: 10.1109/MysuruCon55714.2022.9972734.

[23] Viswanatha, V. & R. Venkata Siva Reddy. (2017). Digital control of buck converter using arduino microcontroller for low power applications. *International Conference On Smart Technologies For Smart Nation (SmartTechCon). IEEE.*

[24] Viswanatha, V., Venkata Siva Reddy & R. Rajeswari. (2020). Research on state space modeling, stability analysis and pid/pidn control of dc–dc converter for digital implementation. In: *Sengodan, T., Murugappan, M., Misra, S. (eds) Advances in Electrical and Computer Technologies. Lecture Notes in Electrical Engineering, 672*. Springer, Singapore. DOI: 10.1007/978-981-15-5558-9_106.

.