Report On :- Predicting Loan Approval Using Machine Learning Classification Algorithms

By- **Vinay Kumar Patel**

Introduction

Project Overview

This project aims to predict loan approval based on various applicant attributes using machine learning algorithms. The dataset used for this analysis is sourced from Kaggle. Using machine learning techniques, we can develop a model that can effectively assess loan applications and determine status of approval.

Problem Statement (AIM)

The primary objective of this project is to construct a predictive model capable of accurately classifying loan applicant(either approved or rejected). This model will assist banks, RBI and other financial institutions in making decisions regarding loan applications, reducing the risk of default and improving overall profit.

Research Questions to Answer

1. Which machine learning algorithms are most suitable for predicting loan approval?
2. How do the performance metrics (precision train accuracy, test accuracy and confusion matrics) of different algorithms compare?
3. What features are most influential in determining loan approval?
4. Can future Scope, techniques to enhance the model's performance?(Grid search CV, Parameter Tuning and standard scaling)

Approach

Step 1 : Data analysis and loading

In this step, the team loads the data and analyzes it to understand what information is provided in the dataset and how it can assist in further developing the model.

Step 2 : Data Preprocessing:

- Handle Missing Values: Identify and address missing values by summary and then fill them and make sure that there are no missing values in the data.

- Outlier Detection and Treatment: Detect and handle outliers in the dataset, which could impact the model's accuracy.

- Dummy variable creation (categorical data): Transform categorical variables into 0 or 1.

- Normalize Data: Normalize numerical features to bring them to a common scale to avoid any feature dominating the model.

Step 3: Train test split

- diving the data into x and y, then dividing it into train data and test data for traning and testing of the model. train test for 0.2 size and 0.1 size.

Step 4: Model Selection and Statistics for accuracy

- Choose different regression models that are available for the given task, like logistic regression, linear discriminant analysis, KNN (K-nearest neighbor), decision tree, random forest, boosting models, and bagging.

- Once the model is trained, compare the accuracy of each model using metrics like accuracy score and confusion matrix. checking overfitting in data by checking test and train both accuracy.

Step 5 : Model Comparison:

- Compare the performance of different models and select the one with the best accuracy and generalization.

- The model having the best accuracy score is the best to choose for making predictions of loan approval.

Step 6 : Further Improvement:

Consider additional techniques like changing hyperparameters, n estimators, and tuning.

Data Collection and Preprocessing

Dataset

The dataset used for this project was taken from Kaggle. It contains information of various loan applicants, including their loan_id, no_of_dependents, education, self_employed, income_annum, loan_amount, loan_term, cibil_score, residential_assets_value, luxury_assets_value, bank_asset_value.

Data Pre processing

Initial data exploration was conducted to understand the distribution of variables, identify missing values, and detect outliers. This involved statistical analysis, visualization, and data cleaning techniques.

Future Scope

To improve the model's performance, feature engineering techniques were applied. This involved creating new features or transforming existing ones to capture relevant information.

Model Development and Evaluation

Algorithm Selection

Several machine learning algorithms included:

- **Linear Discriminant Analysis (LDA):**

Logistic regression is a statistical method used in machine learning for binary classification problems, where the goal is to predict one of two possible outcomes (e.g., yes/no, 0/1). Despite the name "regression," it is actually a classification algorithm.

- **Logistic Regression:**

Linear Discriminant Analysis (LDA) is a classification and dimensionality reduction technique used to separate two or more classes in a dataset by finding a linear combination of features that best separates the classes. It is particularly effective when dealing with linearly separable data and is often used as a pre-processing step before classification tasks.

- **K-Nearest Neighbors (KNN)**

K-Nearest Neighbors (KNN) is a simple and widely used machine learning algorithm for both classification and regression tasks. It is a non-parametric and instance-based learning algorithm, meaning it does not assume any specific form for the underlying data distribution and makes decisions based on the instances closest to the given input.

- **Decision Tree**

A decision tree is a tree-based model that is used to predict the classification as well as the regression model. It is very close to human thinking to approach the task.

- **Random Forest**

A Random Forest model is an ensemble learning method used for classification and regression tasks. It operates by constructing multiple decision trees during training and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

- **Bagging**

Bagging, short for Bootstrap Aggregating, is an ensemble machine learning technique designed to improve the stability and accuracy of machine learning algorithms. It reduces variance and helps to prevent overfitting. The core idea behind bagging is to train multiple models on different subsets of the data and then combine their outputs to produce a more accurate and robust prediction.

- **AdaBoost**

AdaBoost, short for adaptive boosting, is an ensemble learning technique used for classification and regression tasks. It combines multiple weak learners, typically simple decision trees (called decision stumps), to create a strong predictive model. AdaBoost assigns weights to each training instance, increasing the weight of misclassified instances and decreasing the weight of correctly classified ones. In subsequent rounds, new models focus more on the harder-to-classify examples. The final prediction is a weighted majority vote of all the models.

- **Gradient Boosting**

Gradient boosting is an ensemble learning technique used for both classification and regression tasks. It builds a model sequentially by adding new models (usually decision trees) that correct errors made by the previous models. Each new model is trained to predict the residuals or errors of the previous models, and their predictions are combined to produce a final output.

- **XGBoost**

XGBoost, short for Extreme Gradient Boosting, is an advanced ensemble learning technique based on gradient boosting. It is designed to be highly efficient, flexible, and portable, often outperforming other algorithms due to its speed and accuracy. XGBoost uses a combination of decision trees and gradient boosting, optimizing model performance by minimizing errors more effectively.

Model Training

Each algorithm was trained on the cleaned and preprocessed dataset using appropriate hyperparameters. Model training involved fitting the algorithm to the data to learn the underlying patterns and relationships.

Model Evaluation

The performance of each model was evaluated using various metrics, including train accuracy, test accuracy and confusion matrics. The Over fitting is being checked using the difference between train accuracy and test accuracy. These metrics provide insights into the model's ability to correctly classify loan applications.

Hyperparameter Tuning

To optimize the performance of each algorithm, hyperparameter tuning was conducted using Grid scratch CV. This involved systematically exploring different combinations of hyperparameters to find the best-performing configuration.

Results and Discussion

Model Comparison

The performance of different algorithms was compared based on the evaluation metrics. The results indicated that Bagging Consistently out performed other algorithms in terms of accuracy score of test data and it also not having overfitting.

Feature Importance

The importance of different features in predicting loan approval was analyzed using techniques such as feature importance scores or permutation importance. The findings revealed that test accuracy score, train matrices and confusion matrices were the most important factors.

Model Interpretation

The interpretability of the best-performing model was to understand how it makes predictions. This involved analyzing the model's coefficients or the decision rules.

Conclusion

This project successfully developed a machine learning model capable of predicting loan approval with 0.98. The Bagging demonstrated superior performance compared to other algorithms. The identified important features provide valuable insights into the factors influencing loan approval.

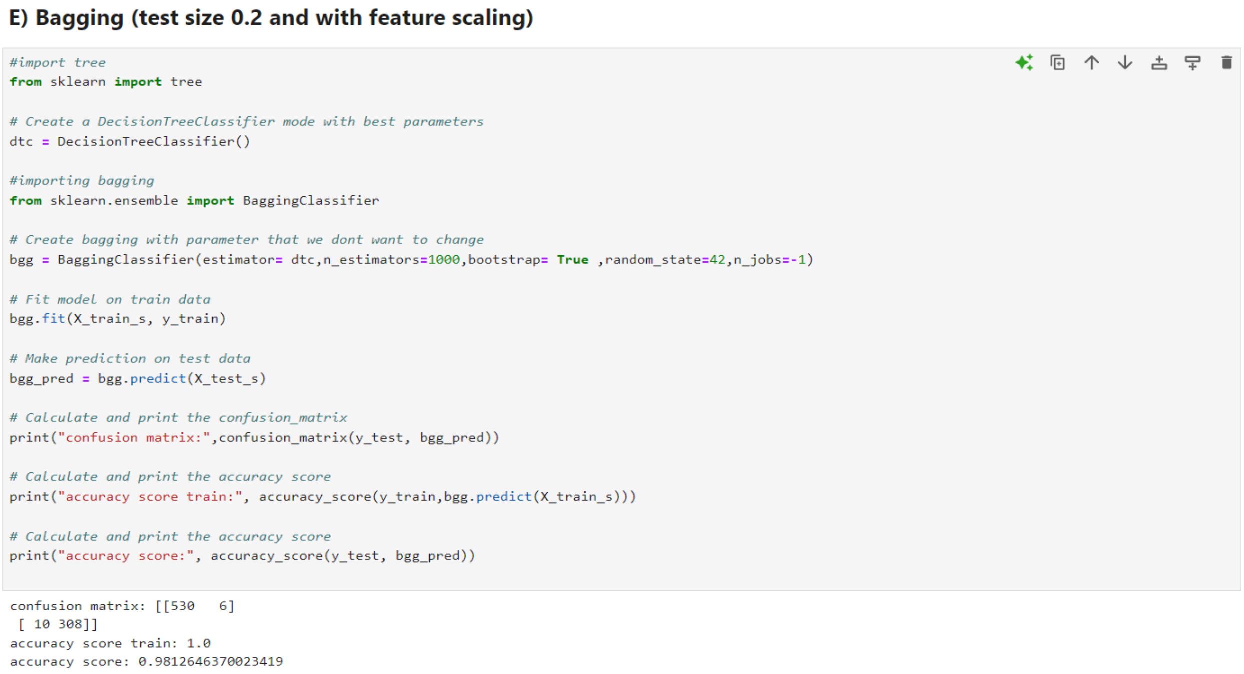
Future Scope

Future scope for the research could explore the following areas:

- Incorporate additional features or datasets to improve model performance.
- Experiment with different machine learning techniques or deep learning architectures.
- Develop more advanced feature engineering techniques.
- Investigate the impact of imbalanced data on model performance and explore techniques to address it.

Note: This is a general outline. The specific content and findings of the project will depend on the actual dataset, implementation details, and experimental results.

SnipShot of the Algorithm

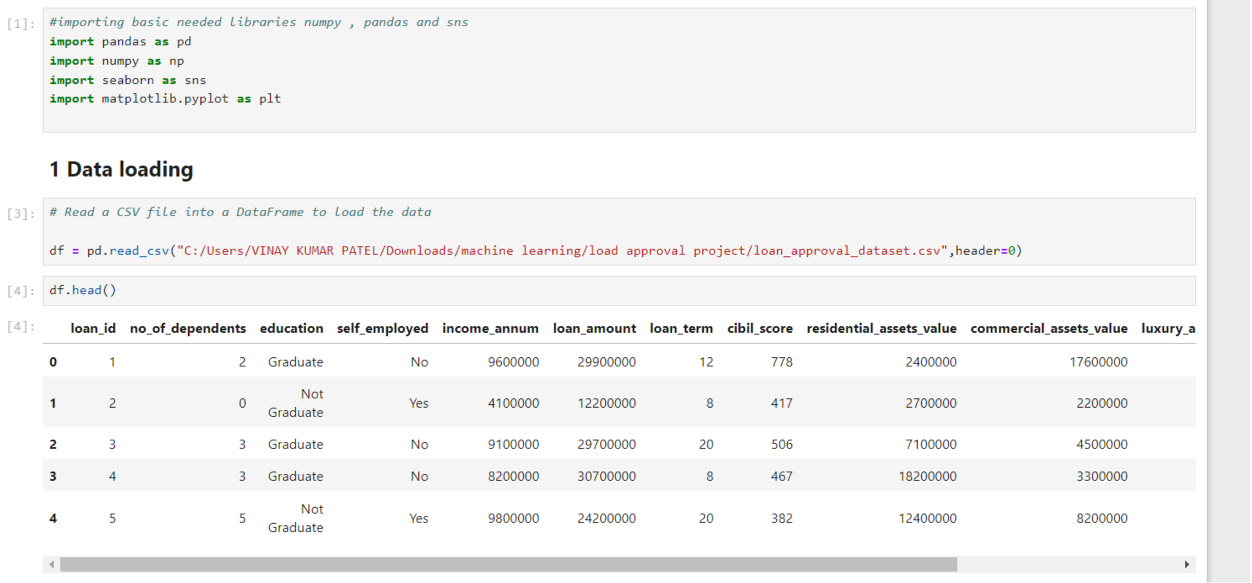


E) Bagging (test size 0.2 and with feature scaling)

```
#import tree
from sklearn import tree

# Create a DecisionTreeClassifier mode with best parameters
dtc = DecisionTreeClassifier()

#importing bagging
from sklearn.ensemble import BaggingClassifier

# Create bagging with parameter that we dont want to change
bgg = BaggingClassifier(estimator= dtc,n_estimators=1000,bootstrap= True ,random_state=42,n_jobs=-1)

# Fit model on train data
bgg.fit(X_train_s, y_train)

# Make prediction on test data
bgg_pred = bgg.predict(X_test_s)

# Calculate and print the confusion_matrix
print("confusion matrix:",confusion_matrix(y_test, bgg_pred))

# Calculate and print the accuracy score
print("accuracy score train:", accuracy_score(y_train,bgg.predict(X_train_s)))

# Calculate and print the accuracy score
print("accuracy score:", accuracy_score(y_test, bgg_pred))
```

```
confusion matrix: [[530   6]
 [ 10 308]]
accuracy score train: 1.0
accuracy score: 0.9812646370023419
```

A few snippets of Jupyter Notebook code



```
[1]: #importing basic needed libraries numpy , pandas and sns
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
```

1 Data loading

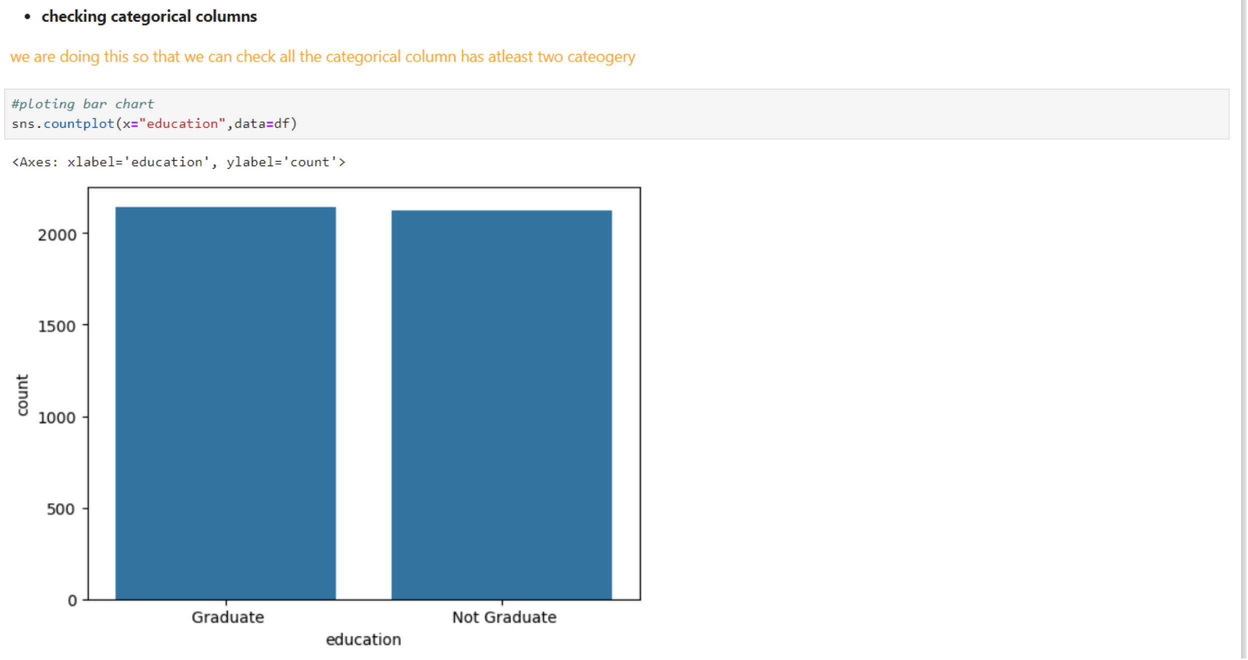
```
[3]: # Read a CSV file into a DataFrame to load the data

     df = pd.read_csv("C:/Users/VINAY KUMAR PATEL/Downloads/machine learning/load approval project/loan_approval_dataset.csv",header=0)
```

```
[4]: df.head()
```

[4]:

	loan_id	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_a
0	1	2	Graduate	No	9600000	29900000	12	778	2400000	17600000	
1	2	0	Not Graduate	Yes	4100000	12200000	8	417	2700000	2200000	
2	3	3	Graduate	No	9100000	29700000	20	506	7100000	4500000	
3	4	3	Graduate	No	8200000	30700000	8	467	18200000	3300000	
4	5	5	Not Graduate	Yes	9800000	24200000	20	382	12400000	8200000	

Importing libraries and loading data



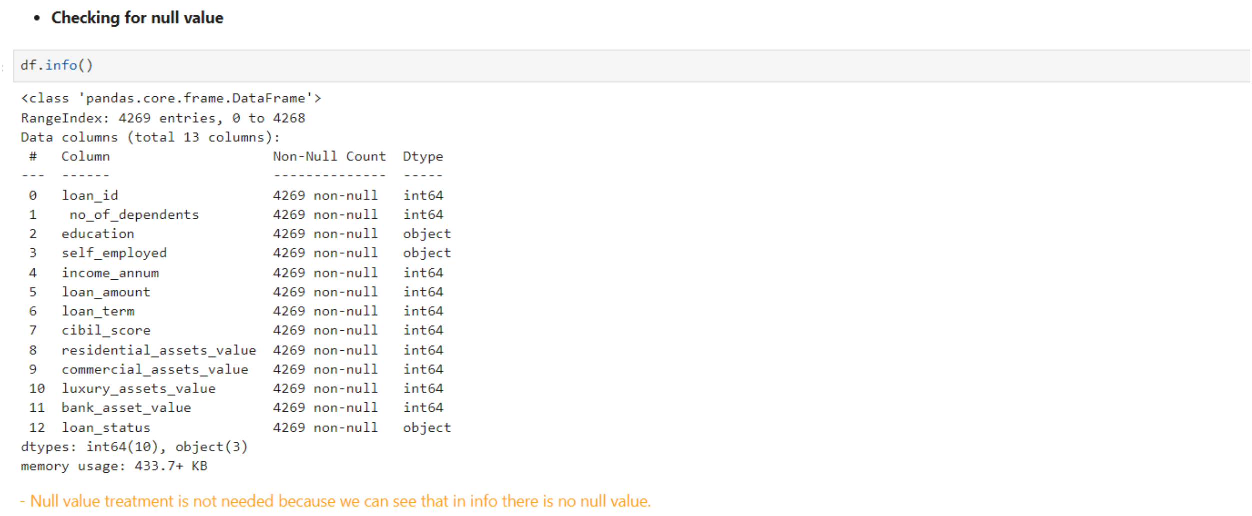
- **checking categorical columns**

we are doing this so that we can check all the categorical column has atleast two cateogery

```
#ploting bar chart
sns.countplot(x="education",data=df)
```

```
<Axes: xlabel='education', ylabel='count'>
```

checking categorical column



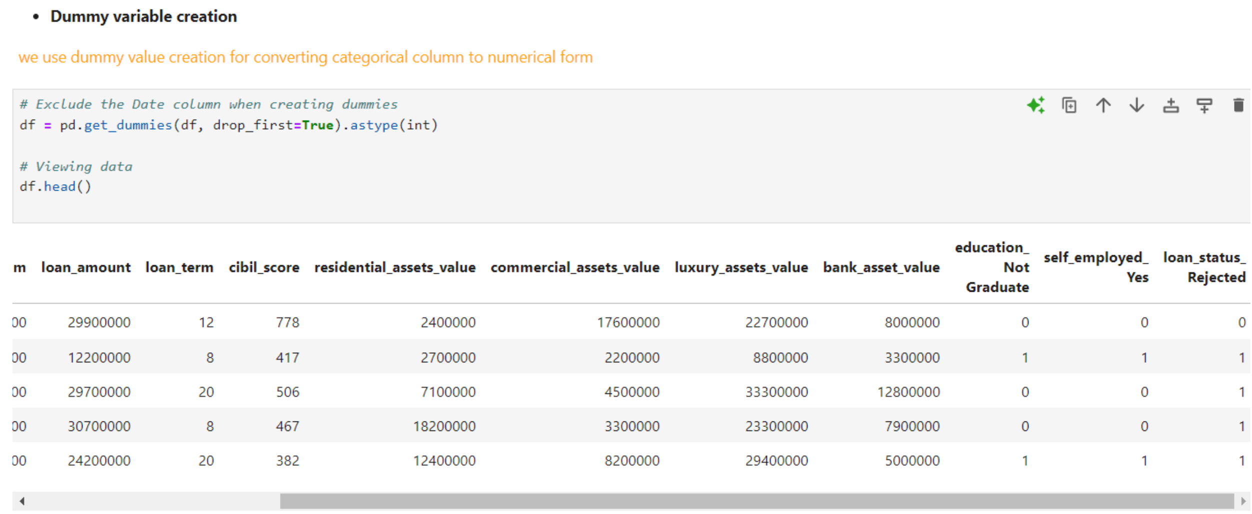
- **Checking for null value**

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4269 entries, 0 to 4268
Data columns (total 13 columns):
 #   Column                     Non-Null Count  Dtype 
---  ------                     --------------  ----- 
 0   loan_id                    4269 non-null   int64 
 1    no_of_dependents          4269 non-null   int64 
 2    education                 4269 non-null   object
 3    self_employed             4269 non-null   object
 4    income_annum              4269 non-null   int64 
 5    loan_amount               4269 non-null   int64 
 6    loan_term                 4269 non-null   int64 
 7    cibil_score               4269 non-null   int64 
 8    residential_assets_value  4269 non-null   int64 
 9    commercial_assets_value   4269 non-null   int64 
 10   luxury_assets_value       4269 non-null   int64 
 11   bank_asset_value          4269 non-null   int64 
 12   loan_status               4269 non-null   object
dtypes: int64(10), object(3)
memory usage: 433.7+ KB
```

- Null value treatment is not needed because we can see that in info there is no null value.

Checking for null value



- **Dummy variable creation**

we use dummy value creation for converting categorical column to numerical form

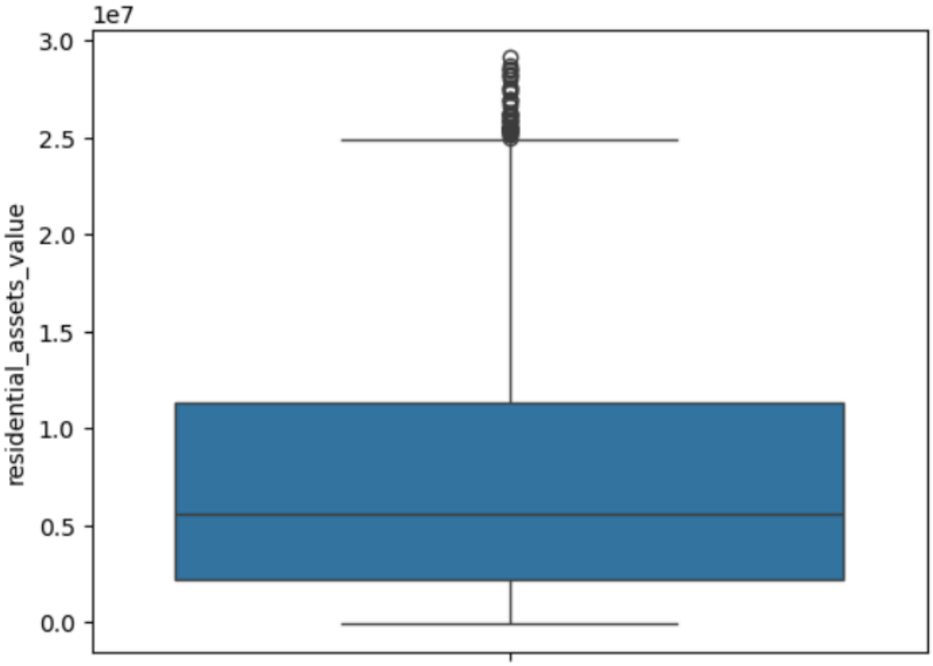
```
# Exclude the Date column when creating dummies
df = pd.get_dummies(df, drop_first=True).astype(int)

# Viewing data
df.head()
```

m	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value	education_Not Graduate	self_employed_Yes	loan_status_Rejected
00	29900000	12	778	2400000	17600000	22700000	8000000	0	0	0
00	12200000	8	417	2700000	2200000	8800000	3300000	1	1	1
00	29700000	20	506	7100000	4500000	33300000	12800000	0	0	1
00	30700000	8	467	18200000	3300000	23300000	7900000	0	0	1
00	24200000	20	382	12400000	8200000	29400000	5000000	1	1	1

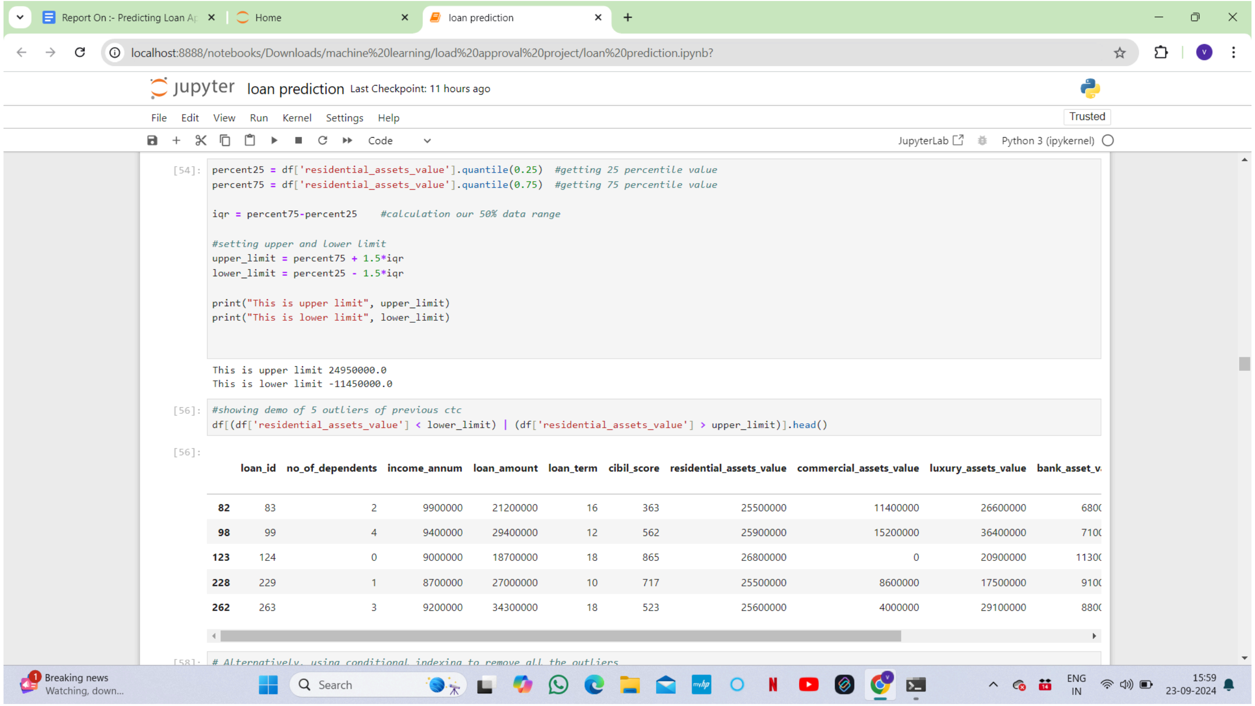
Dummy variable creation

```
sns.boxplot(df['residential_assets_value'])
```

```
<Axes: ylabel='residential_assets_value'>
```



Outlier detection



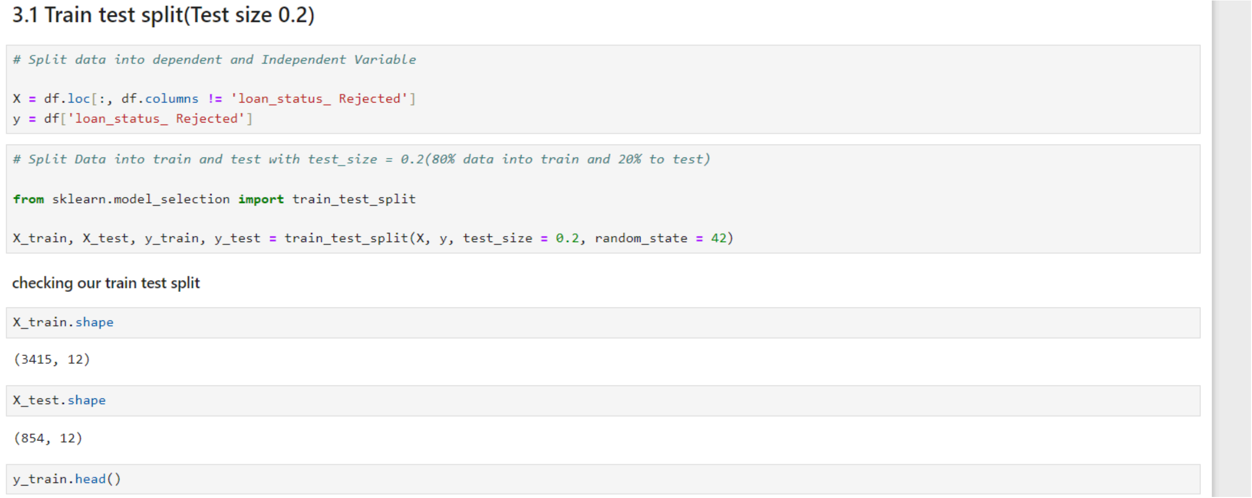
```
[54]: percent25 = df['residential_assets_value'].quantile(0.25)  #getting 25 percentile value
      percent75 = df['residential_assets_value'].quantile(0.75)  #getting 75 percentile value

      iqr = percent75-percent25    #calculation our 50% data range

      #setting upper and lower limit
      upper_limit = percent75 + 1.5*iqr
      lower_limit = percent25 - 1.5*iqr

      print("This is upper limit", upper_limit)
      print("This is lower limit", lower_limit)

      This is upper limit 24950000.0
      This is lower limit -11450000.0

[56]: #showing demo of 5 outliers of previous ctc
      df[(df['residential_assets_value'] < lower_limit) | (df['residential_assets_value'] > upper_limit)].head()
```

[56]:

	loan_id	no_of_dependents	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_v
82	83	2	9900000	21200000	16	363	25500000	11400000	26600000	680
98	99	4	9400000	29400000	12	562	25900000	15200000	36400000	710
123	124	0	9000000	18700000	18	865	26800000	0	20900000	1130
228	229	1	8700000	27000000	10	717	25500000	8600000	17500000	910
262	263	3	9200000	34300000	18	523	25600000	4000000	29100000	880

```
[58]: # Alternatively, using conditional indexing to remove all the outliers
```

Outlier treatment



3.1 Train test split(Test size 0.2)

```
# Split data into dependent and Independent Variable

X = df.loc[:, df.columns != 'loan_status_ Rejected']
y = df['loan_status_ Rejected']
```

```
# Split Data into train and test with test_size = 0.2(80% data into train and 20% to test)

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 = 42)
```

checking our train test split

```
X_train.shape
```

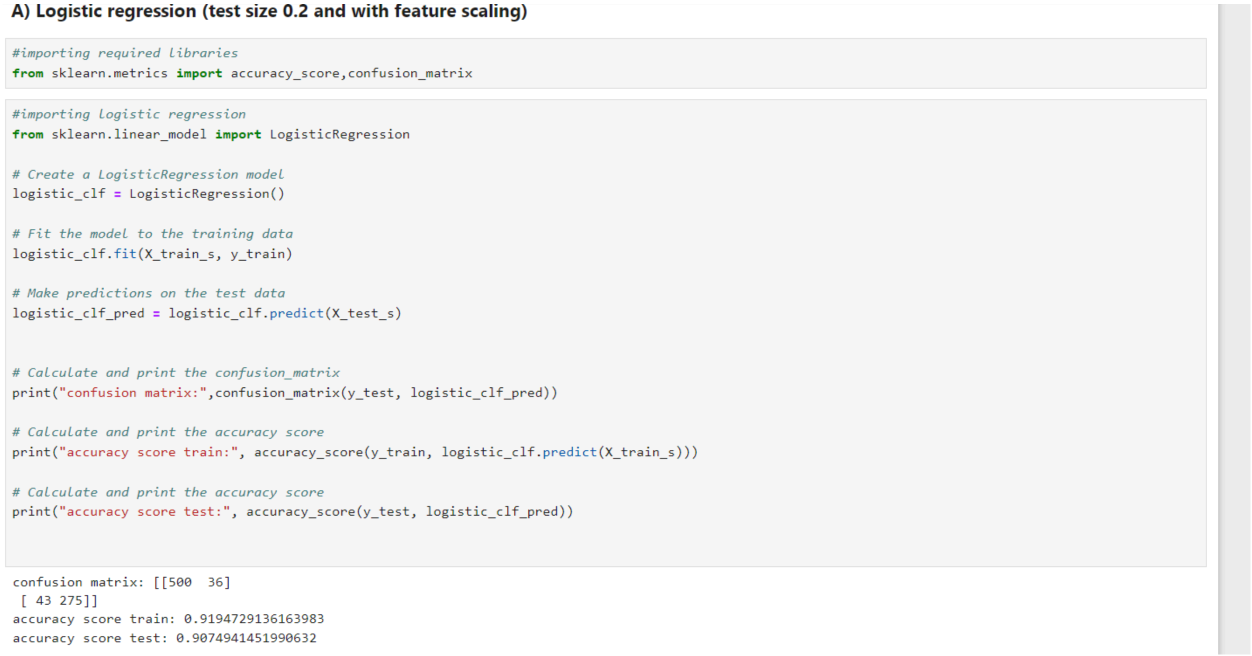
```
(3415, 12)
```

```
X_test.shape
```

```
(854, 12)
```

```
y_train.head()
```

Train test split 0.2



A) Logistic regression (test size 0.2 and with feature scaling)

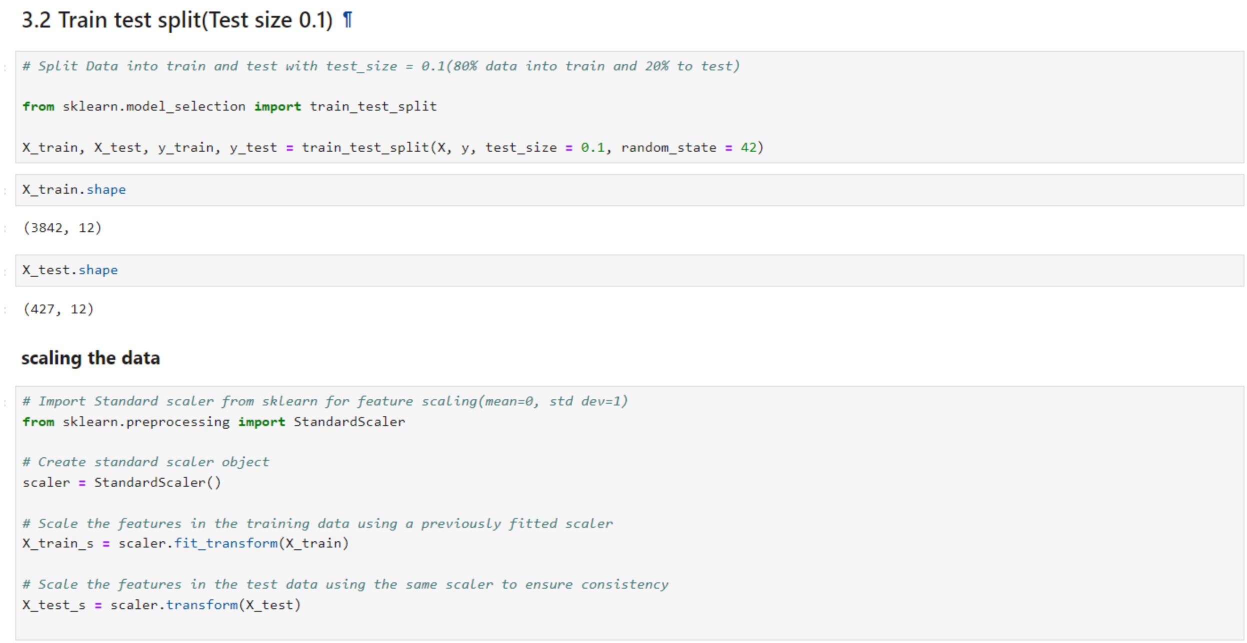
```
#importing required libraries
from sklearn.metrics import accuracy_score,confusion_matrix
```

```
#importing logistic regression
from sklearn.linear_model import LogisticRegression

# Create a LogisticRegression model
logistic_clf = LogisticRegression()

# Fit the model to the training data
logistic_clf.fit(X_train_s, y_train)

# Make predictions on the test data
logistic_clf_pred = logistic_clf.predict(X_test_s)


# Calculate and print the confusion_matrix
print("confusion matrix:",confusion_matrix(y_test, logistic_clf_pred))

# Calculate and print the accuracy score
print("accuracy score train:", accuracy_score(y_train, logistic_clf.predict(X_train_s)))

# Calculate and print the accuracy score
print("accuracy score test:", accuracy_score(y_test, logistic_clf_pred))
```

```
confusion matrix: [[500  36]
 [ 43 275]]
accuracy score train: 0.9194729136163983
accuracy score test: 0.9074941451990632
```

Implementing logistic regression model

Similarly all models are done one by one



3.2 Train test split(Test size 0.1) ¶

```python
# Split Data into train and test with test_size = 0.1(80% data into train and 20% to test)

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = 42)
```

```python
X_train.shape
```

```
(3842, 12)
```

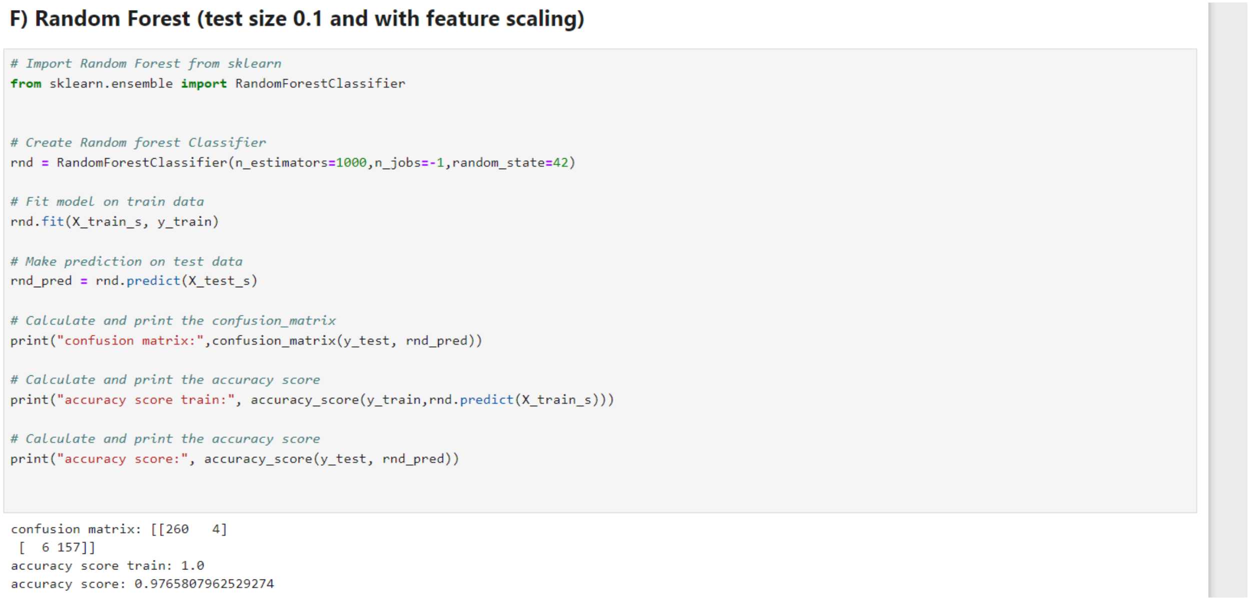
```python
X_test.shape
```

```
(427, 12)
```

scaling the data

```python
# Import Standard scaler from sklearn for feature scaling(mean=0, std dev=1)
from sklearn.preprocessing import StandardScaler

# Create standard scaler object
scaler = StandardScaler()

# Scale the features in the training data using a previously fitted scaler
X_train_s = scaler.fit_transform(X_train)

# Scale the features in the test data using the same scaler to ensure consistency
X_test_s = scaler.transform(X_test)
```

Train test split for 0.1 test size



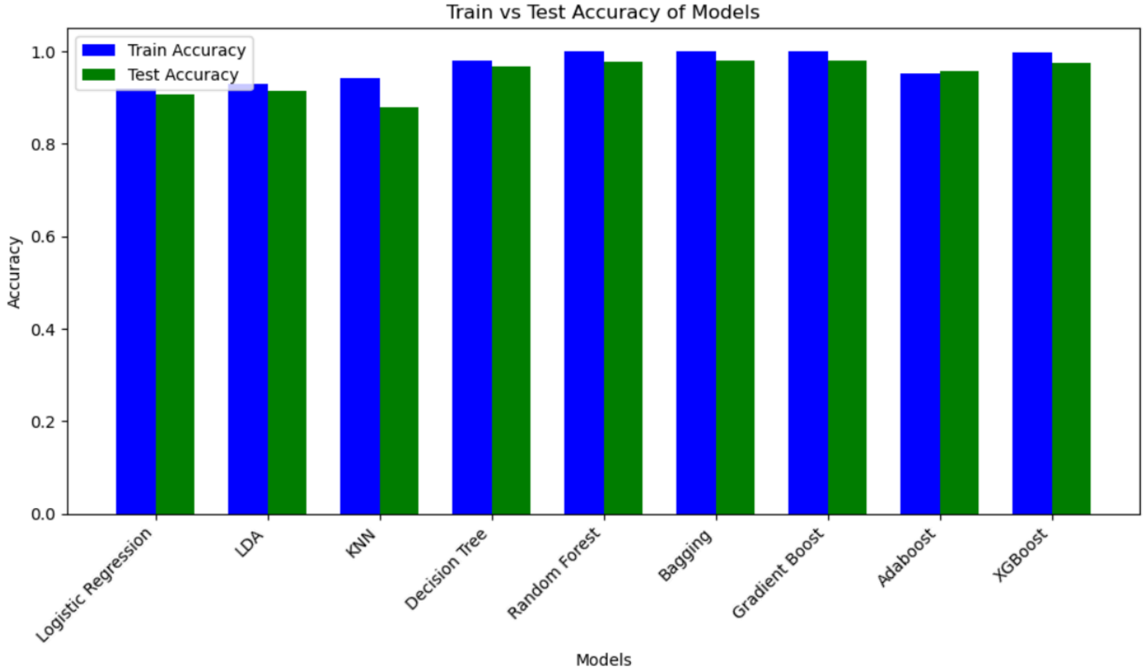
F) Random Forest (test size 0.1 and with feature scaling)

```python
# Import Random Forest from sklearn
from sklearn.ensemble import RandomForestClassifier


# Create Random forest Classifier
rnd = RandomForestClassifier(n_estimators=1000,n_jobs=-1,random_state=42)

# Fit model on train data
rnd.fit(X_train_s, y_train)

# Make prediction on test data
rnd_pred = rnd.predict(X_test_s)

# Calculate and print the confusion_matrix
print("confusion matrix:",confusion_matrix(y_test, rnd_pred))

# Calculate and print the accuracy score
print("accuracy score train:", accuracy_score(y_train,rnd.predict(X_train_s)))

# Calculate and print the accuracy score
print("accuracy score:", accuracy_score(y_test, rnd_pred))
```

```
confusion matrix: [[260   4]
 [  6 157]]
accuracy score train: 1.0
accuracy score: 0.9765807962529274
```

implementing Random forest algorithm

Similarly for all models will be performed



Model comparison to check best