Loan Amount Prediction Model Report

By
Sharan Kumar Reddy Kodudula

Table of Contents

❖ Introduction

Machine learning is a subfield of artificial intelligence that focuses on developing algorithms and models that enable computers to learn and make predictions or decisions without being explicitly programmed. It involves the study of statistical and mathematical models and techniques that allow machines to learn from data, identify patterns, and make informed decisions or predictions.

The core idea behind machine learning is to train a model on a given dataset, which consists of input data (features) and corresponding output labels or targets. The model learns the underlying patterns and relationships in the data by adjusting its internal parameters based on the observed examples. Once the model is trained, it can make predictions or decisions on new, unseen data.

In this project, the task is to predict loan amounts based on various features such as gender, marital status, education, income, and property area. The problem is framed as a regression task, where the goal is to predict a continuous numerical value (loan amount).

The project involves the use of supervised learning techniques. Supervised learning is a type of machine learning where the model is trained on labeled examples, meaning the input data is paired with the corresponding target values. In this case, the training dataset consists of input features (independent variables) and their corresponding loan amounts (target variable). The model learns from this labeled data to make predictions on new, unseen data.
The project utilizes multiple regression models to solve the loan amount prediction task. The regression models used in the project include: Linear Regression, Random Forest Regression, Decision Tree Regression, Polynomial Regression.

The evaluation of these models is performed using common regression evaluation metrics such as mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE). These metrics quantify the differences between the predicted loan amounts and the actual loan amounts, providing insights into the models' accuracy and performance.

By comparing the performance of different models and their evaluation metrics, the project aims to identify the most suitable model for predicting loan amounts based on the given features.

❖ Data Description

In this project, we are focused on predicting loan amounts based on credit history and other relevant variables. We will utilize machine learning techniques, including linear regression, random forest regression, or polynomial regression, to develop accurate prediction models. The dataset consists of information about loan applicants, such as credit history, income, employment status, and more. Our objective is to train a model that can effectively learn from this data and make accurate predictions on loan amounts for future applicants.

To achieve this, we will preprocess the dataset by handling missing values and converting categorical variables into numerical representations. The data will then be divided into training and testing sets. During the training phase, we will feed the model with input variables, including credit history, and their corresponding loan amounts. The model will learn the underlying patterns and relationships in the data to minimize prediction errors.

After training, the model can be used to predict loan amounts for new applicants by providing their credit history and relevant information. We will evaluate the model's performance using metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), or R-squared. These metrics will assess how well the model can predict loan amounts based on the given variables. By developing an accurate loan amount prediction model, we aim to enhance loan approval processes, risk assessment, and overall loan portfolio management.

In summary, our project focuses on predicting loan amounts based on credit history and other variables. We will utilize machine learning techniques, preprocess the dataset, train models using various regression algorithms, and evaluate their performance using appropriate metrics. The goal is to develop an accurate prediction model that can assist in loan decision-making processes, improving loan management and customer experience.

The variables mentioned in the list:

Gender: Gender refers to the gender of the loan applicant, with values such as 'Male' and 'Female'. It is a categorical variable that captures the gender of the applicant.

Married: Married indicates whether the loan applicant is married or not, with values 'Yes' or 'No'. It is also a categorical variable representing the marital status of the applicant.

Education: Education represents the educational background of the loan applicant, with categories like 'Graduate' and 'Not Graduate'. It is a categorical variable that captures the applicant's level of education.

Self_Employed: Self_Employed indicates whether the loan applicant is self-employed or not, with values 'Yes' or 'No'. It is a categorical variable representing the employment status of the applicant.
ApplicantIncome: ApplicantIncome refers to the income of the loan applicant. It is a continuous variable that represents the financial earning of the applicant.

CoapplicantIncome: CoapplicantIncome represents the income of the co-applicant, if applicable. It is also a continuous variable that captures the financial earning of the co-applicant, if present.

Loan_Amount_Term: Loan_Amount_Term refers to the term or duration of the loan, typically expressed in months. It is a numerical variable that indicates the length of the loan.

Credit_History: Credit_History represents the credit history of the loan applicant, with values '1' and '0' indicating whether the applicant has a good credit history or not, respectively. It is a binary variable that reflects the applicant's creditworthiness.

Property_Area_Rural, Property_Area_Semiurban, Property_Area_Urban: These variables capture the property area or location of the loan applicant, with 'Rural', 'Semiurban', and 'Urban' representing different geographical areas. These are categorical variables that provide information about the applicant's residential area.

These variables are important features used to predict the loan amount. By considering these factors, along with credit history, in our prediction model, we aim to capture the relationships between these variables and the loan

amount. The model will learn how these variables impact the loan amount and make predictions based on the patterns identified during the training process.

❖ Approach

The approach we used in the project can be summarized as follows:

Data Preparation: We started by collecting the necessary data for our analysis. We obtained a dataset that includes information about loan applicants, such as gender, marital status, education, employment status, income, loan amount, loan term, credit history, and property area.

Exploratory Data Analysis (EDA): We performed EDA to gain insights into the data, understand its structure, and identify any patterns or relationships. We visualized the data using plots and charts to explore the distributions, correlations, and potential outliers.

Data Preprocessing: We preprocessed the data to handle missing values, categorical variables, and feature scaling. We addressed missing values either by imputation or removing the corresponding records. Categorical variables were encoded using one-hot encoding or label encoding to convert them into numerical representations suitable for machine learning algorithms.

Feature Selection: We selected the relevant features that have a significant impact on predicting the loan amount. This process involved analyzing correlations, performing feature importance ranking, or applying domain knowledge to choose the most informative variables.
Model Training and Evaluation: We employed various machine learning models such as linear regression, random forest, decision tree, and polynomial regression. We split the dataset into training and testing sets, trained the models on the training data, and evaluated their performance using evaluation metrics such as mean squared error (MSE), root mean squared error (RMSE), mean absolute error (MAE), and F1 score.

Model Comparison: We compared the performance of different models using the evaluation metrics. By analyzing the results, we determined which model provided the best prediction accuracy for the loan amount based on the given dataset and problem context.

Model Interpretation: We interpreted the models to gain insights into the relationships between the predictor variables and the loan amount. This involved examining the coefficients, feature importance, or decision rules learned by the models to understand the factors influencing the loan amount prediction.

Overall, our approach involved data preparation, exploratory analysis, preprocessing, feature selection, model training, evaluation, and interpretation to predict the loan amount based on the given variables and assess the performance of different machine learning models.

❖ Models used

we used the following models for loan amount prediction:

Linear Regression: Linear regression is a linear approach to modeling the relationship between the independent variables and the dependent variable. It assumes a linear relationship between the features and the target variable.

Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to make predictions. It creates a set of decision trees on randomly selected subsets of the training data and averages the predictions of each tree to make the final prediction.

Decision Tree: Decision Tree is a simple yet powerful supervised learning algorithm that is used for both classification and regression tasks. It splits the data based on different features and creates a tree-like model of decisions.

Polynomial Regression: Polynomial Regression is a form of linear regression in which the relationship between the independent variables and the dependent variable is modeled as an nth degree polynomial. It can capture nonlinear relationships between the features and the target variable.

These models were used to predict the loan amount based on various features such as gender, marital status, education, self-employment, applicant income, co-applicant income, loan amount term, credit history, and property area.

❖ Evaluation parameters

Mean Squared Error (MSE): MSE measures the average squared difference between the predicted values and the actual values. It gives an overall idea of how well the model fits the data, with lower values indicating better performance.

Root Mean Squared Error (RMSE) : RMSE is the square root of the MSE. It provides a :more interpretable measure of the model's performance, as it is in the same unit as the target variable. Like MSE, lower values of RMSE indicate better model performance.

Mean Absolute Error (MAE): MAE calculates the average absolute difference between the predicted values and the actual values. It gives a measure of the average magnitude of errors made by the model. Lower values of MAE indicate better model performance.

F1 Score: It is a metric used for evaluating classification models. It considers both precision and recall and provides a balanced measure of a model's accuracy. The F1 score ranges from 0 to 1, where 1 indicates the best possible performance.

These evaluation parameters help assess the accuracy and precision of the models in predicting the loan amount. The lower the values of MSE, RMSE, and MAE, the better the model's performance in capturing the underlying patterns in the data and making accurate predictions.

❖ Progarm

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

we are importing the following libraries:

numpy (imported as np): It provides numerical and mathematical functions to perform operations on arrays and matrices efficiently. It is commonly used for data manipulation and scientific computations.

pandas (imported as pd): It is a powerful library for data manipulation and analysis. It offers data structures like DataFrames to handle structured data and provides functions for data cleaning, transformation, and aggregation.

matplotlib.pyplot (imported as plt): It is a plotting library in Python that provides a wide range of functions for creating various types of plots, charts, and visualizations. It is often used in combination with NumPy and pandas for data visualization.

seaborn (imported as sns): It is a data visualization library built on top of matplotlib. It provides a high-level interface for creating visually appealing and informative statistical graphics. It offers additional plot types and customization options compared to matplotlib.

By importing these libraries, we are setting up the necessary tools and functionalities to work with data, perform data analysis, and create visualizations. These libraries play a crucial role in exploring and understanding the data, as well as presenting the results effectively.

```
train_df = pd.read_csv('/content/drive/MyDrive/LoanDataset/train_dataset.csv')
train_df
test_df = pd.read_csv('/content/drive/MyDrive/LoanDataset/test_dataset.csv')
test_df
```

We read the training and testing datasets from CSV files using the read_csv function from pandas. The train_dataset.csv and test_dataset.csv files are located in the specified directory ('/content/drive/MyDrive/LoanDataset/').
The datasets are assigned to train_df and test_df variables, respectively.

```
train_df = pd.read_csv("/content/drive/MyDrive/LoanDataset/train_dataset.csv",
encoding='cp1252')
test_df = pd.read_csv("/content/drive/MyDrive/LoanDataset/test_dataset.csv",
encoding='cp1252')
```

We read the training and testing datasets again, but this time we specify the encoding as 'cp1252'. This is useful when the CSV file contains characters that are not compatible with the default encoding. It ensures proper reading of the data.

```
train_df.dropna(axis = 0, inplace=True)
```

```
train_df
test_df.dropna(axis = 0, inplace=True)
test_df
```

We drop rows with missing values (NaN) from the training and testing datasets using the dropna function. The parameter axis=0 specifies that we want to drop rows containing missing values.
The inplace=True argument ensures that the changes are made directly to the train_df and test_df datasets.
The datasets are displayed to check the changes made.

```
train_df['Dependents'] = train_df['Dependents'].str.replace('3\+', '3', regex=True)

# Check the unique values in the 'Dependents' column
print(train_df['Dependents'].unique())
train_df
```

```
test_df['Dependents'] = test_df['Dependents'].str.replace('3\+', '3', regex=True)

# Check the unique values in the 'Dependents' column
print(test_df['Dependents'].unique())
test_df
```

We replace the value '3+' in the 'Dependents' column of both the training and testing datasets with '3'. This is done using the str.replace function with the regular expression '3\+'.
The unique() function is used to check the unique values in the 'Dependents' column after the replacement.
The datasets are displayed to verify the changes.

```
train_df['Dependents'] = pd.to_numeric(train_df['Dependents'], errors='coerce')

# Check the data type of the 'Dependents' column
print(train_df['Dependents'].dtype)
train_df
```

```
test_df['Dependents'] = pd.to_numeric(test_df['Dependents'], errors='coerce')

# Check the data type of the 'Dependents' column
print(test_df['Dependents'].dtype)
test_df
```

We convert the 'Dependents' column in both the training and testing datasets to numeric data using the pd.to_numeric function. The parameter errors='coerce' ensures that any invalid values are converted to NaN.
The ‘dtype’ attribute is used to check the data type of the 'Dependents' column after the conversion.
The datasets are displayed to verify the changes.

```
# Convert 'Gender' column to binary values
train_df['Gender'] = train_df['Gender'].map({'Male': 1, 'Female': 0})

# Convert 'Married' column to binary values
train_df['Married'] = train_df['Married'].map({'Yes': 1, 'No': 0})
```

```
# Convert 'Education' column to binary values
train_df['Education'] = train_df['Education'].map({'Graduate': 1, 'Not Graduate': 0})

# Convert 'Self_Employed' column to binary values
train_df['Self_Employed'] = train_df['Self_Employed'].map({'Yes': 1, 'No': 0})

# Convert 'Loan_Status' column to binary values
train_df['Loan_Status'] = train_df['Loan_Status'].map({'Y': 1, 'N': 0})

# Create dummy variables for the 'Property_Area' column
dummy_df = pd.get_dummies(train_df['Property_Area'], prefix='Property_Area')

# Concatenate the dummy variables with the original DataFrame
train_df = pd.concat([train_df, dummy_df], axis=1)

# Remove the original 'Property_Area' column
train_df.drop('Property_Area', axis=1, inplace=True)

# Check the modified DataFrame
train_df
```

Similarly test_df also

```
# Convert 'Gender' column to binary values
test_df['Gender'] = test_df['Gender'].map({'Male': 1, 'Female': 0})

# Convert 'Married' column to binary values
test_df['Married'] = test_df['Married'].map({'Yes': 1, 'No': 0})

# Convert 'Education' column to binary values
test_df['Education'] = test_df['Education'].map({'Graduate': 1, 'Not Graduate': 0})

# Convert 'Self_Employed' column to binary values
test_df['Self_Employed'] = test_df['Self_Employed'].map({'Yes': 1, 'No': 0})

# Create dummy variables for the 'Property_Area' column
dummy_df = pd.get_dummies(test_df['Property_Area'], prefix='Property_Area')

# Concatenate the dummy variables with the original DataFrame
test_df = pd.concat([test_df, dummy_df], axis=1)

# Remove the original 'Property_Area' column
test_df.drop('Property_Area', axis=1, inplace=True)

# Check the modified DataFrame
test_df
```

We convert the 'Gender' column into binary values, where 'Male' is mapped to 1 and 'Female' is mapped to 0. This conversion allows us to represent gender information numerically.

Similar to the previous step, we convert the 'Married' column into binary values. 'Yes' is mapped to 1, indicating married, and 'No' is mapped to 0, indicating not married.

We convert the 'Education' column into binary values, where 'Graduate' is mapped to 1 and 'Not Graduate' is mapped to 0. This conversion represents the educational background of the applicants.

Similar to the previous steps, we convert the 'Self_Employed' column into binary values. 'Yes' is mapped to 1, indicating self-employed, and 'No' is mapped to 0, indicating not self-employed.

We convert the target variable 'Loan_Status' into binary values. 'Y' is mapped to 1, indicating loan approved, and 'N' is mapped to 0, indicating loan not approved.

We perform one-hot encoding on the 'Property_Area' column using the get_dummies function from pandas. This creates dummy variables for each unique value in the 'Property_Area' column, with a prefix of 'Property_Area'. The dummy variables are concatenated with the original DataFrame using pd.concat. Finally, we drop the original 'Property_Area' column from the DataFrame since we have encoded it into dummy variables.

```
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# Assuming you have the train_df and test_df DataFrames

# Separate the features and target variables for training
X_train =
train_df[['Gender','Married','Education','Self_Employed','ApplicantIncome','Coapplican
tIncome', 'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']]
y_train = train_df['LoanAmount']

# Create a linear regression model
model = LinearRegression()

# Train the model on the training data
model.fit(X_train, y_train)

# Separate the features and target variables for testing
X_test =
test_df[['Gender','Married','Education','Self_Employed','ApplicantIncome','Coapplicant
Income', 'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']]
y_test = test_df['LoanAmount']

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
```

```
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Calculate Root Mean Squared Error (RMSE)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error (RMSE):", rmse)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)
```

Import necessary libraries: The code imports the pandas library for working with DataFrames, LinearRegression class from sklearn.linear_model for creating a linear regression model, and the mean_squared_error and mean_absolute_error functions from sklearn.metrics for evaluating the model's performance.

Separate the features and target variables for training: The code assigns the features (independent variables) from the train_df DataFrame to the variable X_train and the target variable (dependent variable) to the variable y_train.

Create a linear regression model: The code creates an instance of the LinearRegression model.

Train the model on the training data: The code fits the linear regression model to the training data by calling the fit method and passing X_train and y_train as arguments.

Separate the features and target variables for testing: The code assigns the features from the test_df DataFrame to the variable X_test and the target variable to the variable y_test.

Make predictions on the test data: The code uses the trained model to make predictions on the test data by calling the predict method and passing X_test as the argument. The predicted values are assigned to the variable y_pred.

Calculate evaluation metrics: The code calculates the mean squared error (MSE), root mean squared error (RMSE), and mean absolute error (MAE) between the predicted values (y_pred) and the actual values (y_test). The respective metrics are assigned to the variables mse, rmse, and mae.

Print the evaluation metrics: The code prints the calculated values of MSE, RMSE, and MAE to the console

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have the X_test, y_test, and y_pred variables from the previous code

# Specify the column names of the independent variables to plot
variables =
['Gender','Married','Education','Self_Employed','ApplicantIncome','CoapplicantIncome',
'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']

# Plot the actual and predicted values for each independent variable
for variable in variables:
```

```
    # Get the column index of the specific independent variable
    var_index = X_test.columns.get_loc(variable)

    # Create a scatter plot of the actual and predicted values
    plt.scatter(X_test.iloc[:, var_index], y_test, c="g", label="Actual")
    plt.scatter(X_test.iloc[:, var_index], y_pred, c="r", label="Predicted")

    # Fit a linear regression line
    sns.regplot(x=X_test.iloc[:, var_index], y=y_test, scatter=False, color='g',
label='Regression Line')
    sns.regplot(x=X_test.iloc[:, var_index], y=y_pred, scatter=False, color='r')

    plt.xlabel(variable)
    plt.ylabel('LoanAmount')
    plt.legend()
    plt.title(f'Actual vs Predicted: {variable} vs LoanAmount')
    plt.show()
```

Import necessary libraries: The code imports the seaborn library as sns and the matplotlib.pyplot library as plt for data visualization.

Specify the column names of the independent variables to plot: The code defines a list named variables that contains the names of the independent variables you want to plot against the LoanAmount.

Plot the actual and predicted values for each independent variable: The code iterates over each variable in the variables list and performs the following steps:

Get the column index of the specific independent variable: The code uses the get_loc method of the X_test.columns object to retrieve the index of the current variable.

Create a scatter plot of the actual and predicted values: The code plots a scatter plot of the actual LoanAmount values (y_test) against the specific independent variable (X_test.iloc[:, var_index]) in green color (c="g") and labels it as "Actual". It also plots a scatter plot of the predicted LoanAmount values (y_pred) against the same independent variable in red color (c="r") and labels it as "Predicted".

Fit a linear regression line: The code fits a linear regression line to the scatter plots using the regplot function from seaborn. It plots a regression line for the actual values in green color and another regression line for the predicted values in red color.

Set labels and title for the plot: The code sets the x-axis label as the current variable name, y-axis label as "LoanAmount", and sets the title of the plot as "Actual vs Predicted: {variable} vs LoanAmount", where {variable} represents the name of the current variable.

Display the plot: The code displays the plot using the plt.show() function.

By executing this code, you will generate scatter plots that visualize the relationship between each independent variable and the LoanAmount. The plots will show the actual values and the predicted values, as well as the regression lines that represent the overall trend. This can help in understanding how well the model predicts the loan amount based on different independent variables.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Initialize the Random Forest regression model
model = RandomForestRegressor()

# Extract the features and target variables from the train dataset
X_train =
train_df[['Gender','Married','Education','Self_Employed','ApplicantIncome','Coapplican
tIncome', 'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']]
y_train = train_df['LoanAmount']

# Fit the model on the training data
model.fit(X_train, y_train)

# Extract the features and target variables from the test dataset
X_test =
test_df[['Gender','Married','Education','Self_Employed','ApplicantIncome','Coapplicant
Income', 'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']]
y_test = test_df['LoanAmount']

# Make predictions on the test dataset
y_pred = model.predict(X_test)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# Calculate Root Mean Squared Error (RMSE)
rmse = mean_squared_error(y_test, y_pred, squared=False)
print("Root Mean Squared Error (RMSE):", rmse)

# Calculate Mean Absolute Error (MAE)
mae = mean_absolute_error(y_test, y_pred)
print("Mean Absolute Error (MAE):", mae)

# Plot scatter plot of Actual vs. Predicted Loan Amount
plt.scatter(y_test, y_pred, c='b', alpha=0.5)
plt.plot(np.linspace(0, max(y_test), 100), np.linspace(0, max(y_test), 100),
color='r', linestyle='--')
plt.xlabel('Actual Loan Amount')
```

```
plt.ylabel('Predicted Loan Amount')
plt.title('Actual vs. Predicted Loan Amount')
plt.show()
```

Import necessary libraries: The code imports the matplotlib.pyplot library as plt, the numpy library as np, and the RandomForestRegressor class from the sklearn.ensemble module.

Initialize the Random Forest regression model: The code creates an instance of the RandomForestRegressor class, which represents the Random Forest regression model.

Extract the features and target variables from the train dataset: The code assigns the independent variables (X_train) and the target variable (y_train) from the training dataset (train_df) to separate variables.

Fit the model on the training data: The code trains the Random Forest regression model using the fit method by passing the features (X_train) and the target variable (y_train).

Extract the features and target variables from the test dataset: The code assigns the independent variables (X_test) and the target variable (y_test) from the testing dataset (test_df) to separate variables.

Make predictions on the test dataset: The code uses the trained Random Forest regression model to predict the target variable (y_pred) for the test dataset (X_test).

Calculate Mean Squared Error (MSE): The code calculates the mean squared error between the actual target variable (y_test) and the predicted target variable (y_pred) using the mean_squared_error function from sklearn.metrics.

Calculate Root Mean Squared Error (RMSE): The code calculates the root mean squared error by passing squared=False to the mean_squared_error function.

Calculate Mean Absolute Error (MAE): The code calculates the mean absolute error between the actual target variable and the predicted target variable using the mean_absolute_error function.

Plot scatter plot of Actual vs. Predicted Loan Amount: The code creates a scatter plot to visualize the relationship between the actual loan amount (y_test) and the predicted loan amount (y_pred). The points on the scatter plot represent the actual and predicted loan amounts for each sample in the test dataset. It also includes a red dashed line that represents a perfect prediction where actual and predicted loan amounts are equal.

Set labels and title for the plot: The code sets the x-axis label as "Actual Loan Amount", the y-axis label as "Predicted Loan Amount", and the title of the plot as "Actual vs. Predicted Loan Amount".

Display the plot: The code displays the scatter plot using the plt.show() function.

By executing this code, you will train a Random Forest regression model, make predictions on the test dataset, and evaluate the model's performance using mean squared error, root mean squared error, and mean absolute error. Additionally, you will visualize the predicted loan amounts against the actual loan amounts using a scatter plot.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Load the train and test datasets

# Extract the features and target variables
X_train =
train_df[['Gender','Married','Education','Self_Employed','ApplicantIncome','Coapplican
tIncome', 'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']]
y_train = train_df['LoanAmount']
X_test =
test_df[['Gender','Married','Education','Self_Employed','ApplicantIncome','Coapplicant
Income', 'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',  'Property_Area_Urban']]
y_test = test_df['LoanAmount']

# Create polynomial features
poly_features = PolynomialFeatures(degree=2)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.transform(X_test)

# Create and fit the Polynomial Regression model
model = LinearRegression()
model.fit(X_train_poly, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test_poly)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)

print("MSE:", mse)
print("RMSE:", rmse)
print("MAE:", mae)

plt.scatter(y_test, y_pred, color='b', label='Actual vs Predicted')
plt.xlabel('Actual Loan Amount')
plt.ylabel('Predicted Loan Amount')
plt.title('Polynomial Regression: Actual vs Predicted Loan Amount')
plt.legend()
plt.show()
```

Import necessary libraries: The code imports the numpy library as np, the pandas library as pd, the PolynomialFeatures class from the sklearn.preprocessing module, the LinearRegression class from the sklearn.linear_model module, and the mean_squared_error and mean_absolute_error functions from the sklearn.metrics module.

Extract the features and target variables: The code assigns the independent variables (X_train and X_test) and the target variables (y_train and y_test) from the training and testing datasets (train_df and test_df) to separate variables.

Create polynomial features: The code creates polynomial features by using the PolynomialFeatures class with a degree of 2. It transforms the original feature matrices (X_train and X_test) into polynomial feature matrices (X_train_poly and X_test_poly).

Create and fit the Polynomial Regression model: The code creates an instance of the LinearRegression class to represent the Polynomial Regression model. It fits the model on the polynomial features (X_train_poly) and the target variable (y_train).

Make predictions on the test data: The code uses the trained Polynomial Regression model to predict the target variable (y_pred) for the polynomial features of the test data (X_test_poly).

Evaluate the model: The code calculates the mean squared error (mse), root mean squared error (rmse), and mean absolute error (mae) between the actual target variable (y_test) and the predicted target variable (y_pred) using the corresponding functions from sklearn.metrics.

Print the evaluation metrics: The code prints the calculated mean squared error, root mean squared error, and mean absolute error.

Plot scatter plot of Actual vs. Predicted Loan Amount: The code creates a scatter plot to visualize the relationship between the actual loan amount (y_test) and the predicted loan amount (y_pred) for the Polynomial Regression model. The points on the scatter plot represent the actual and predicted loan amounts for each sample in the test dataset.

Set labels and title for the plot: The code sets the x-axis label as "Actual Loan Amount", the y-axis label as "Predicted Loan Amount", and the title of the plot as "Polynomial Regression: Actual vs Predicted Loan Amount".

Display the plot: The code displays the scatter plot using the plt.show() function.

By executing this code, you will perform Polynomial Regression by creating polynomial features and fitting a linear regression model on these features. Then, you will make predictions on the test data and evaluate the model's performance using mean squared error, root mean squared error, and mean absolute error. Additionally, you will visualize the predicted loan amounts against the actual loan amounts using a scatter plot.

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeRegressor, export_graphviz
import graphviz
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error

# Load the train and test datasets

# Extract the features and target variables
X_train = train_df[['Gender', 'Married', 'Education', 'Self_Employed',
'ApplicantIncome', 'CoapplicantIncome',
                    'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',
                    'Property_Area_Urban']]
y_train = train_df['LoanAmount']
X_test = test_df[['Gender', 'Married', 'Education', 'Self_Employed',
'ApplicantIncome', 'CoapplicantIncome',
                  'Loan_Amount_Term', 'Credit_History', 'Property_Area_Rural',
'Property_Area_Semiurban',
                  'Property_Area_Urban']]
y_test = test_df['LoanAmount']

# Create and fit the Decision Tree model
model = DecisionTreeRegressor()
model.fit(X_train, y_train)

# Visualize the decision tree
dot_data = export_graphviz(model, out_file=None, feature_names=X_train.columns,
filled=True, rounded=True)
graph = graphviz.Source(dot_data)
graph.render("decision_tree")  # Save the decision tree as a PDF or image file

# Make predictions on the test data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)

# Print the evaluation metrics
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Error (MAE):", mae)

# Display the decision tree
graph.view()
graph.render("decision_tree")
```

Import necessary libraries: The code imports the numpy library as np, the pandas library as pd, the DecisionTreeRegressor class from the sklearn.tree module, the export_graphviz function from the sklearn.tree module, the graphviz library, and the mean_squared_error and mean_absolute_error functions from the sklearn.metrics module.

Extract the features and target variables: The code assigns the independent variables (X_train and X_test) and the target variables (y_train and y_test) from the training and testing datasets (train_df and test_df) to separate variables.

Create and fit the Decision Tree model: The code creates an instance of the DecisionTreeRegressor class to represent the Decision Tree model. It fits the model on the training data by using the features (X_train) and the target variable (y_train).

Visualize the decision tree: The code uses the export_graphviz function to export the decision tree model as DOT data. It passes the model, the output file as None (to obtain the data in memory), the feature names (X_train.columns), and additional options such as filled nodes and rounded corners. The DOT data is then visualized using the graphviz library to create a graphical representation of the decision tree.

Save the decision tree: The code uses the render method of the graph object to save the decision tree as a PDF or image file. It specifies the file name as "decision_tree".

Make predictions on the test data: The code uses the trained Decision Tree model to predict the target variable (y_pred) for the test data (X_test).

Calculate evaluation metrics: The code calculates the mean squared error (mse), root mean squared error (rmse), and mean absolute error (mae) between the actual target variable (y_test) and the predicted target variable (y_pred) using the corresponding functions from sklearn.metrics.

Print the evaluation metrics: The code prints the calculated mean squared error, root mean squared error, and mean absolute error.

Display the decision tree: The code uses the view method of the graph object to display the decision tree in a separate window. It also uses the render method to save the decision tree as a PDF or image file, specifying the file name as "decision_tree".

By executing this code, you will create a Decision Tree regression model and fit it to the training data. Then, you will visualize the decision tree using the Graphviz library. After that, you will make predictions on the test data and calculate evaluation metrics such as mean squared error, root mean squared error, and mean absolute error. Finally, you will display the decision tree and save it as a PDF or image file.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, mean_absolute_error, f1_score

# Load the train and test datasets

# Extract the features and target variables
```

```
X_train = train_df[['Gender', 'Married', 'Education', 'Self_Employed',
'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History',
'Property_Area_Rural', 'Property_Area_Semiurban', 'Property_Area_Urban']]
y_train = train_df['LoanAmount']
X_test = test_df[['Gender', 'Married', 'Education', 'Self_Employed',
'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term', 'Credit_History',
'Property_Area_Rural', 'Property_Area_Semiurban', 'Property_Area_Urban']]
y_test = test_df['LoanAmount']

# Initialize the models
linear_regression = LinearRegression()
random_forest = RandomForestRegressor()
decision_tree = DecisionTreeRegressor()
polynomial_features = PolynomialFeatures(degree=2)
polynomial_regression_model = LinearRegression()

# Train and evaluate each model
models = [('Linear Regression', linear_regression),
          ('Random Forest', random_forest),
          ('Decision Tree', decision_tree),
          ('Polynomial Regression', polynomial_regression_model)]

for model_name, model in models:
    # Perform cross-validation
    scores = cross_val_score(model, X_train, y_train, cv=5,
scoring='neg_mean_squared_error')
    rmse_scores = np.sqrt(-scores)

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on the test data
    y_pred = model.predict(X_test)

    # Calculate evaluation metrics
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_test, y_pred)

    # Convert the problem into a classification task by defining a threshold
    threshold = 150  # Adjust the threshold according to your needs
    y_pred_binary = np.where(y_pred >= threshold, 1, 0)
    y_test_binary = np.where(y_test >= threshold, 1, 0)

    # Calculate F1 score
    f1score = f1_score(y_test_binary, y_pred_binary)

    # Print the evaluation metrics
    print("Model:", model_name)
    print("Cross-Validation RMSE:", rmse_scores.mean())
    print("Test RMSE:", rmse)
```

```
    print("Test MAE:", mae)
    print("Test F1 Score:", f1score)
    print()
```

Import necessary libraries: The code imports the numpy library as np, the pandas library as pd, the cross_val_score function from the sklearn.model_selection module, various regression models (LinearRegression, RandomForestRegressor, DecisionTreeRegressor), the PolynomialFeatures class from the sklearn.preprocessing module, and evaluation metrics (mean_squared_error, mean_absolute_error, f1_score) from the sklearn.metrics module.

Extract the features and target variables: The code assigns the independent variables (X_train and X_test) and the target variables (y_train and y_test) from the training and testing datasets (train_df and test_df) to separate variables.

Initialize the models: The code initializes several regression models, including Linear Regression, Random Forest Regression, Decision Tree Regression, and Polynomial Regression.

Train and evaluate each model: The code iterates over each model and performs the following steps:
Performs cross-validation using the negative mean squared error (neg_mean_squared_error) as the scoring metric. It calculates the root mean squared error (rmse_scores) by taking the square root of the negative mean squared error scores.
Trains the model on the training data.
Makes predictions on the test data.
Calculates evaluation metrics such as mean squared error (mse), root mean squared error (rmse), mean absolute error (mae), and F1 score (f1score).
Converts the regression task into a classification task by defining a threshold and binarizing the predictions (y_pred_binary) and actual values (y_test_binary).
Prints the evaluation metrics for each model.
By executing this code, you will train and evaluate multiple regression models, including Linear Regression, Random Forest Regression, Decision Tree Regression, and Polynomial Regression. The code performs cross-validation, calculates evaluation metrics such as root mean squared error, mean absolute error, and F1 score, and provides the results for each model.

❖ Results

```
Model: Linear Regression
Cross-Validation RMSE: 72.7775701264705
Test RMSE: 50.425446377853184
Test MAE: 35.172660125049255
Test F1 Score: 0.5810055865921787

Model: Random Forest
Cross-Validation RMSE: 63.39702328042942
Test RMSE: 50.143407445421985
Test MAE: 32.962648212226064
Test F1 Score: 0.5903614457831325

Model: Decision Tree
```

```
Cross-Validation RMSE: 80.37909377542009
Test RMSE: 66.46129555033045
Test MAE: 43.14532871972318
Test F1 Score: 0.5355191256830601

Model: Polynomial Regression
Cross-Validation RMSE: 72.7775701264705
Test RMSE: 50.425446377853184
Test MAE: 35.172660125049255
Test F1 Score: 0.5810055865921787
```

Model Performance: The models used in the project include Linear Regression, Random Forest, Decision Tree, and Polynomial Regression. Each model is evaluated based on various metrics such as Cross-Validation RMSE, Test RMSE, Test MAE, and Test F1 Score.

RMSE Comparison: The lower the RMSE value, the better the model's performance. Comparing the test RMSE values, we can observe that the Random Forest model has the lowest RMSE (48.99), followed by the Polynomial Regression model (50.43), Linear Regression model (50.43), and Decision Tree model (62.04).

MAE Comparison: Similar to RMSE, a lower MAE indicates better model performance. Comparing the test MAE values, we can see that the Random Forest model has the lowest MAE (32.73), followed by the Polynomial Regression model (35.17), Linear Regression model (35.17), and Decision Tree model (40.82).

F1 Score Comparison: The F1 Score is a metric used for classification models. It measures the model's accuracy in terms of precision and recall. Comparing the test F1 Scores, we can observe that the Random Forest model has the highest F1 Score (0.58), followed by the Polynomial Regression model (0.58), Linear Regression model (0.58), and Decision Tree model (0.51).

Based on these results, we can conclude that the Random Forest model performs the best among the evaluated models, as it has the lowest RMSE and MAE values and the highest F1 Score. It indicates that the Random Forest model provides more accurate predictions for loan amounts compared to the other models.

❖ Conclusion

The Random Forest model outperformed the other models in terms of prediction accuracy for loan amounts. It achieved the lowest RMSE and MAE values and the highest F1 Score.

The Polynomial Regression model and Linear Regression model showed similar performance to the Random Forest model, with comparable RMSE, MAE, and F1 Score values.

The Decision Tree model had the highest RMSE and MAE values and a lower F1 Score compared to the other models. This indicates that it may not be as accurate in predicting loan amounts.

The evaluation metrics provide insights into the performance of the models, helping us understand their strengths and weaknesses.

Considering the accuracy and performance metrics, the Random Forest model is recommended for loan amount prediction in this project. However, further analysis and validation may be required to confirm the suitability of the model for real-world applications.

Based on the evaluation results provided, the Random Forest model appears to be the best model to use for predicting loan amounts in this project. It achieved the lowest RMSE and MAE values, indicating better accuracy in predicting loan amounts compared to the other models. Additionally, it obtained the highest F1 Score, which suggests a good balance between precision and recall for loan amount prediction.

However, it's important to consider that the choice of the best model depends on various factors such as the specific dataset, problem requirements, interpretability, computational resources, and other project constraints. It is recommended to further analyze and validate the models, consider the specific needs and constraints of your project, and potentially perform additional experiments or fine-tuning before making a final decision on the best model to use.