LOAN AMOUNT PREDICTION

Mini Project

Loan Amount Prediction

# Introduction:

Loan Prediction System is a software which checks the eligibility of a particular customer who is capable of paying loan or not. This method examines a number of variables, including the customer's marital status, income, spending, and other elements. Several users of the trained dataset use this approach. These elements are taken into account when creating the necessary model. In order to obtain the desired outcome, this model is applied to 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 theloan; if the answer is no, then the consumer is not capable of repaying the loan.

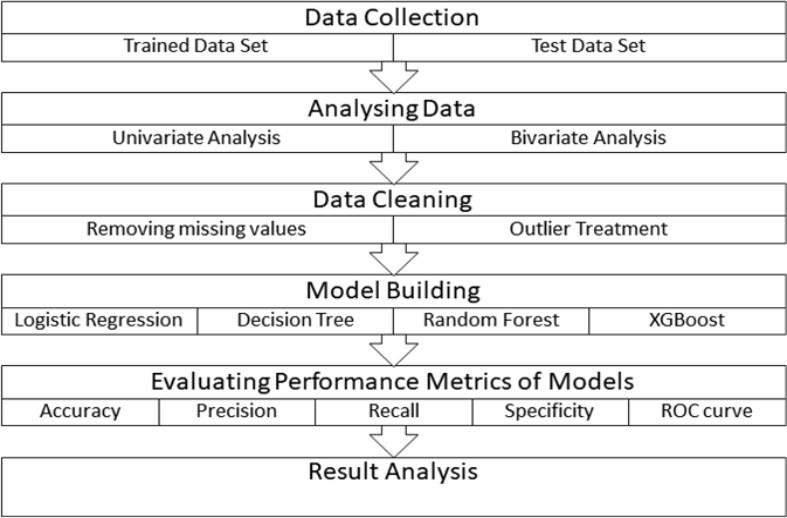
We are able to approve loans for clients based on these variables.

The Python programming language is used for the implementation of the code which has been developed in Colab and the html pages are developed for deployment of website using Visual Studio code. The proposed system can deliver high accuracy results and moderate loss fortraining and validate data. Finally, the results show the model implemented with high accuracy. Further, this work can be extended in order to improve the focus where the high accuracy can be obtained. The code created in Colab is implemented using the Python programming language, and Visual Studio code is utilised to create the html pages for the website's deployment. The suggested system can produce results with high accuracy and only a little amount of loss for training and validation data. Lastly, the outcomes demonstrate the model's high degree of accuracy. The scope of this effort may also be improved in order to achieve higher levels of accuracy.

# Data Analysis for Prediction of Loan Amount:

The primary goal of the report is to categorize the types of loan applicants. The report categorizes the clients based on specific parameters. There is classification through data analysisthat is exploratory. Exploratory data analysis is a technique that evaluates and summarizes the major aspects from training dataset.

# Methodology:



For the Model Building – Liner Regression is used.

For Evaluating the Performance: Mean Squared Error is Identified.

The above mentioned methodology is specified for the flow of the process (an example).

# Prediction of Loan Amount using Machine Learning Approach:

Machine learning is a phenomenon in which analytical model is built from the trained model. This model is applied on test data for providing of the accurate results. Here Linear Regression algorithms is used for prediction of loan. The main purpose of this report is to provide immediate and accurate results for the loan prediction. In banking sector there will be nnumber of people who apply loans. It is difficult to check customer’s eligibility through paper work. The system can provide accurate results for the n number of people.

# Libraries for Data Analysis:

## Pandas:

Pandas is a Python package to work with structured and time series data.The data from various file formats such as csv, json, sql etc can be imported using Pandas. It is a powerful open source tool used for data analysis and data manipulation operations suchas data cleaning, merging, selecting as well wrangling.

## Sklearn:

This python library is helpful for building machine learning and statistical models such as clustering, classification, regression etc. Though it can be used for reading, manipulating and summarizing the data as well, better libraries are there to perform these functions.

## Matplotlib:

Matplotlib is one of the plotting library in python which is however widely in use for machine learning application with its numerical mathematics extension- Numpy to createstatic, animated and interactive visualisations.

## Numpy:

NumPy can be used to perform a wide variety of mathematical operations on arrays. It adds powerful data structures to Python that guarantee efficient calculations with arraysand matrices and it supplies an enormous library of high-level mathematical functions that operate on these arrays and matrices.

# Understanding the Dataset:

The training data set is used to train the machine learning model. Each new applicant's application form information serves as a test data set. The model will forecast whether a loan would be accepted or not based on the training data sets. There are 13 features in all, 12 independent variables, 1 dependent variable, or Loan Status, in the train dataset, and 12 independent variables, or Loan Status, in the test dataset. There are categories for the Loan ID,Gender, Married, Dependents, Education, Self-Employed, Property Area, and Loan Status.

# The Dataset features are :

Index(['Loan\_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self\_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan\_Amount\_Term', 'Credit\_History', 'Property\_Area', 'Loan\_Status'], dtype='object')

# The Datatypes of features are as :

Loan\_ID object

Gender object

Married object

Dependents object

Education object Self\_Employed object ApplicantIncome int64 CoapplicantIncome float64 LoanAmount float64 Loan\_Amount\_Term float64 Credit\_History float64 Property\_Area object Loan\_Status

objec

tdtype: object

There are 614 rows ans 13 columns. 12 independent variable and target variable:(614, 13)

# There are three formats of data types mentioned in the given dataset :

* **Object:** Object format means variables are categorical. Categorical variables in our dataset are:Loan\_ID, Gender, Married, Dependents, Education, Self\_Employed, Property\_Area, Loan\_Status
* **int64:** It represents the integer variables. ApplicantIncome is of this format.
* **float64:** It represents the variable that has some decimal values involved. They are also numerical variables. Numerical variables in our dataset are: CoapplicantIncome, LoanAmount, Loan\_Amount\_Term, and Credit\_History

# Predicting the Loan Amount using Python:

**Step 1:** Importing the necessary libraries

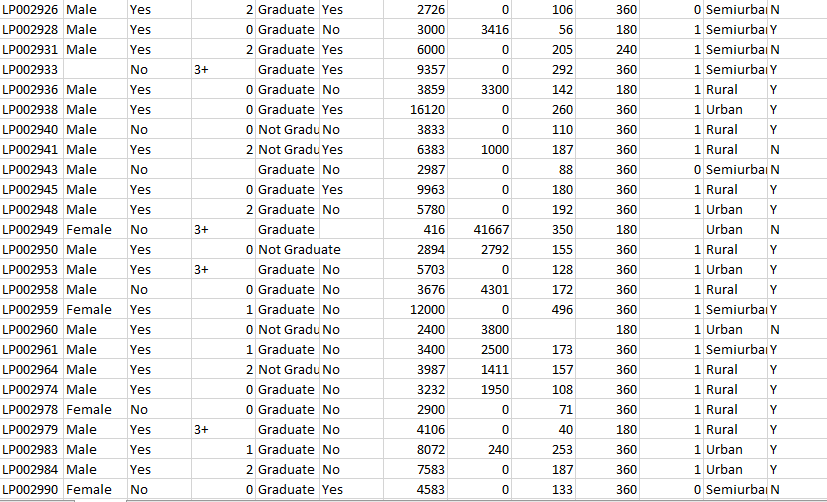
import pandas as pd import numpy as np

from sklearn.preprocessing import LabelEncoder from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error import matplotlib.pyplot as plt

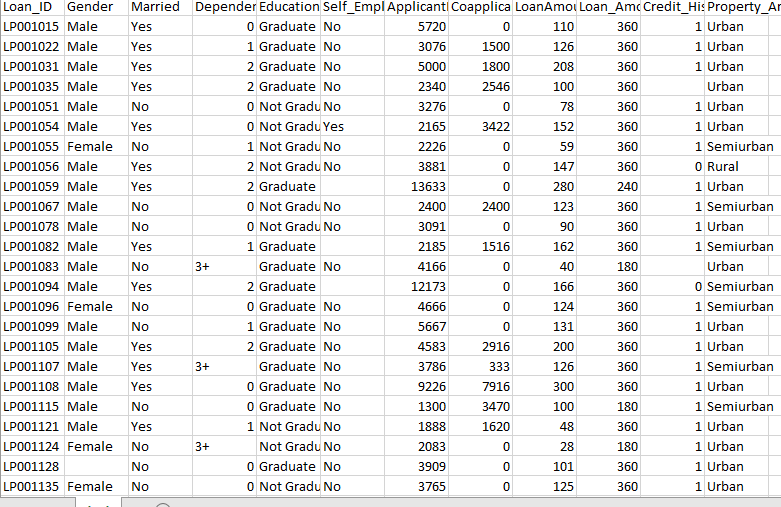
**Step2:** Loading the Training and Testing Data

train\_df = pd.read\_csv('/content/train.csv') test\_df = pd.read\_csv('/content/test.csv')

## Train Dataset Sample



**Test Dataset Sample:**



**Step3:** Dropping the irrelevant Columns

train\_df .drop(columns=['Loan\_ID','Loan\_Status'], inplace=True) test\_df .drop(columns=['Loan\_ID'], inplace=True)

**Step4:** Filling missing Values

There are missing values in Gender, Married, Dependents, Self\_Employed, LoanAmount, Loan\_Amount\_Term, and Credit\_History features.

We will treat the missing values in all the features one by one. We can consider these methods to fill the missing values:

* For numerical variables: imputation using mean or median
* For categorical variables: imputation using mode

There are very less missing values in Gender, Married, Dependents, Credit\_History, and Self\_Employed features so we can fill them using the mode of the features.

train\_df['Gender'].fillna(value='Male', inplace=True) train\_df['Married'].fillna(value='Yes', inplace=True) train\_df['Dependents'].fillna(value='0', inplace=True) train\_df['Self\_Employed'].fillna(value='No', inplace=True) train\_df['LoanAmount'].fillna(value=train\_df['LoanAmount'].median(), inplace=True)

train\_df['Loan\_Amount\_Term'].fillna(value=train\_df['Loan\_Amount\_Term'].median(), inplace= True)

train\_df['Credit\_History'].fillna(value=1.0, inplace=True)

test\_df['Gender'].fillna(value='Male', inplace=True) test\_df['Married'].fillna(value='Yes', inplace=True) test\_df['Dependents'].fillna(value='0', inplace=True) test\_df['Self\_Employed'].fillna(value='No', inplace=True) test\_df['LoanAmount'].fillna(value=test\_df['LoanAmount'].median(), inplace=True)

test\_df['Loan\_Amount\_Term'].fillna(value=test\_df['Loan\_Amount\_Term'].median(), inplace=Tr ue)

test\_df['Credit\_History'].fillna(value=1.0, inplace=True)

**Step5:** Encoding the Categorical variables

So, the next step would be to map these categories to their binary alternative. For instance, "No" is mapped to 0, and "Yes" is mapped to 1. For non-binary values, like Property area with options like semiurban, rural or urban areas, we use the get\_dummies function of Pandas to automatically one-hot encode them. Similarly, the variable for male and female applicants will be separated into two dummy columns: Gender\_male and Gender\_female. Once we do this, we can see that there are dummy variables for each category of every categorical variable

le = LabelEncoder()

test\_df['Married'] = le.fit\_transform(test\_df['Married'])

test\_df['Gender'] = le.fit\_transform(test\_df['Gender']) test\_df['Education'] = le.fit\_transform(test\_df['Education']) test\_df['Property\_Area'] = le.fit\_transform(test\_df['Property\_Area']) test\_df['Self\_Employed'] = le.fit\_transform(test\_df['Self\_Employed'])

train\_df['Married'] = le.fit\_transform(train\_df['Married']) train\_df['Gender'] = le.fit\_transform(train\_df['Gender']) train\_df['Education'] = le.fit\_transform(train\_df['Education']) train\_df['Property\_Area'] = le.fit\_transform(train\_df['Property\_Area']) train\_df['Self\_Employed'] = le.fit\_transform(train\_df['Self\_Employed'])

**Step6:** Applying the conversion function to each value in the column

test\_df['Dependents'] = test\_df['Dependents'].apply(convert\_range) train\_df['Dependents'] = train\_df['Dependents'].apply(convert\_range)

**Step7:** Splitting the train and test data

X\_train = train\_df.drop(columns=['LoanAmount']) y\_train = train\_df['LoanAmount']

X\_test = test\_df.drop(columns=['LoanAmount']) y\_test = test\_df['LoanAmount']

**Step8:** Creating the Linear Regression Model

model = LinearRegression()

**Step9**: Train the model on the training dataset

model.fit(X\_train, y\_train)

**Step10:** Make predictions on the test data

y\_pred = model.predict(X\_test)

**Step11:** Evaluate the model using mean squared error

mse = mean\_squared\_error(y\_test, y\_pred)

**Step12:** Print the Mean Squared Error

print("Mean squared error: ", mse)

**Step13:** Plotting and Printing the prediction

plt.scatter(y\_test, y\_pred) plt.xlabel('Actual Loan Amount') plt.ylabel('Predicted Loan Amount') plt.show()

print("Coefficients:", model.coef\_) print("Intercept:", model.intercept\_)

r\_squared = model.score(X\_test, y\_test) print("R-squared:", r\_squared)

# Complete Python Code:

import pandas as pd import numpy as np

from sklearn.preprocessing import LabelEncoder from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error

import matplotlib.pyplot as plt

# Load train and test data

train\_df = pd.read\_csv('/content/train.csv') test\_df = pd.read\_csv('/content/test.csv')

# drop irrelevant columns

train\_df .drop(columns=['Loan\_ID','Loan\_Status'], inplace=True) test\_df .drop(columns=['Loan\_ID'], inplace=True)

# fill missing values train\_df['Gender'].fillna(value='Male', inplace=True) train\_df['Married'].fillna(value='Yes', inplace=True) train\_df['Dependents'].fillna(value='0', inplace=True)

train\_df['Self\_Employed'].fillna(value='No', inplace=True) train\_df['LoanAmount'].fillna(value=train\_df['LoanAmount'].median(), inplace=True) train\_df['Loan\_Amount\_Term'].fillna(value=train\_df['Loan\_Amount\_Term'].median(), inplace= True)

train\_df['Credit\_History'].fillna(value=1.0, inplace=True)

test\_df['Gender'].fillna(value='Male', inplace=True) test\_df['Married'].fillna(value='Yes', inplace=True) test\_df['Dependents'].fillna(value='0', inplace=True) test\_df['Self\_Employed'].fillna(value='No', inplace=True) test\_df['LoanAmount'].fillna(value=test\_df['LoanAmount'].median(), inplace=True)

test\_df['Loan\_Amount\_Term'].fillna(value=test\_df['Loan\_Amount\_Term'].median(), inplace=Tr ue)

test\_df['Credit\_History'].fillna(value=1.0, inplace=True)

#encoding

le = LabelEncoder()

test\_df['Married'] = le.fit\_transform(test\_df['Married']) test\_df['Gender'] = le.fit\_transform(test\_df['Gender']) test\_df['Education'] = le.fit\_transform(test\_df['Education']) test\_df['Property\_Area'] = le.fit\_transform(test\_df['Property\_Area']) test\_df['Self\_Employed'] = le.fit\_transform(test\_df['Self\_Employed'])

train\_df['Married'] = le.fit\_transform(train\_df['Married']) train\_df['Gender'] = le.fit\_transform(train\_df['Gender']) train\_df['Education'] = le.fit\_transform(train\_df['Education']) train\_df['Property\_Area'] = le.fit\_transform(train\_df['Property\_Area']) train\_df['Self\_Employed'] = le.fit\_transform(train\_df['Self\_Employed'])

def convert\_range(value):

lower\_bound = float(value.replace("+", "")) upper\_bound = 1000

# replace with the maximum value allowed in your problem return lower\_bound

# apply the conversion function to each value in the column test\_df['Dependents'] = test\_df['Dependents'].apply(convert\_range) train\_df['Dependents'] = train\_df['Dependents'].apply(convert\_range)

# Split the training and test data into features (X) and target (y) X\_train = train\_df.drop(columns=['LoanAmount'])

y\_train = train\_df['LoanAmount']

X\_test = test\_df.drop(columns=['LoanAmount']) y\_test = test\_df['LoanAmount']

# Create a linear regression model model = LinearRegression()

# Train the model on the training set model.fit(X\_train, y\_train)

#Make predictions on the test set y\_pred = model.predict(X\_test)

# Evaluate the model using mean squared error mse = mean\_squared\_error(y\_test, y\_pred)

# Print the mean squared error print("Mean squared error: ", mse)

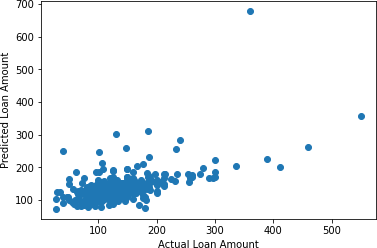
plt.scatter(y\_test, y\_pred) plt.xlabel('Actual Loan Amount') plt.ylabel('Predicted Loan Amount') plt.show()

print("Coefficients:", model.coef\_) print("Intercept:", model.intercept\_)

r\_squared = model.score(X\_test, y\_test) print("R-squared:", r\_squared)

**Output:**

Mean squared error: 2669.5437299479454



Coefficients: [ 4.72357880e+00 1.35560571e+01 5.82607743e+00 -1.51793608e+01

1.07332071e+01 7.82347440e-03 7.10953043e-03 1.14354531e-01

9.66818118e-01 -4.32072660e+00] Intercept: 41.27969123222486

R-squared: 0.27966284186792056

# Conclusion:

From the proper view of analysis this system can be used perfect for loan prediction. The software is working perfect and can be used for all banking requirements. This system can be easily uploaded in any operating system. Since the technology is moving towards online, this system has more scope for the upcoming days. This system is more secure and reliable. Since wehave used Linear Regression Algorithm the system returns very accurate results and the flow of graph. There is no issue if there are many no of customers applying for loan. This system acceptsdata for N no. of customers. In future we can add more algorithms to this system for getting moreaccurate results.

Exploratory data Analysis on the features of this dataset and saw how each feature is distributedis done. Analysed each variable to check if data is cleaned and normally distributed.

The data is cleaned and removed NA values. Created dummy variables for constructing the

model. There are still quite a many things that can be tried to improve the models’ predictions. We create and add more variables, try different models with a different subset of features and/orrows, etc. Some of the ideas are listed below

We can train the XGBoost model using grid search to optimize its hyperparameters and improveits accuracy.

We can combine the applicants with 1,2,3 or more dependents and make a new feature as discussed in the EDA part.

We can also make independent vs independent variable visualizations to discover some more patterns.

We can even try ensemble modeling (a combination of different models).