Loan Approval Prediction

1. Introduction

Loan approval is a critical process in the financial sector. Predicting whether a loan applicant is eligible for approval based on key attributes can help banks and financial institutions make data-driven decisions. This project uses machine learning to build a predictive model that determines loan eligibility based on applicant details.

2. Problem Statement

Financial institutions face risks when approving loans, and manually evaluating applications is time-consuming. This project aims to develop a machine learning model to automate loan approval decisions using historical loan application data.

3. Data Description

The dataset consists of various attributes related to loan applicants, including:

- **Loan_ID**: Unique identifier for each loan
- **Gender**: Male or Female
- **Married**: Marital status
- **Dependents**: Number of dependents
- **Education**: Graduate or Not Graduate
- **Self_Employed**: Applicant is self-employed or not
- **ApplicantIncome**: Applicant's income
- **CoapplicantIncome**: Co-applicant's income
- **LoanAmount**: Loan amount applied for
- **Loan_Amount_Term**: Term of the loan in months
- **Credit_History**: Credit history of the applicant
- **Property_Area**: Urban, Semi-urban, or Rural
- **Loan_Status**: (Target Variable) Loan approval status (Yes/No)

4. Methodology

1. **Data Preprocessing**
   - Handle missing values
   - Convert categorical variables into numerical format
   - Normalize numerical features
2. **Exploratory Data Analysis (EDA)**
   - Visualize data distributions
   - Identify patterns and correlations
3. **Model Selection and Training**

    - Use machine learning models like Logistic Regression, Decision Tree, and Random Forest
    - Train models using training data
    - Evaluate performance using accuracy, precision, recall, and F1-score
4. **Model Evaluation**
    - Test the model on unseen data
    - Analyze performance metrics

5. Implementation Steps

This code is designed to **predict whether a loan application will be approved or not** using machine learning, specifically a **Logistic Regression model**. Let's break it down step by step.

1. Import Required Libraries

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score
```

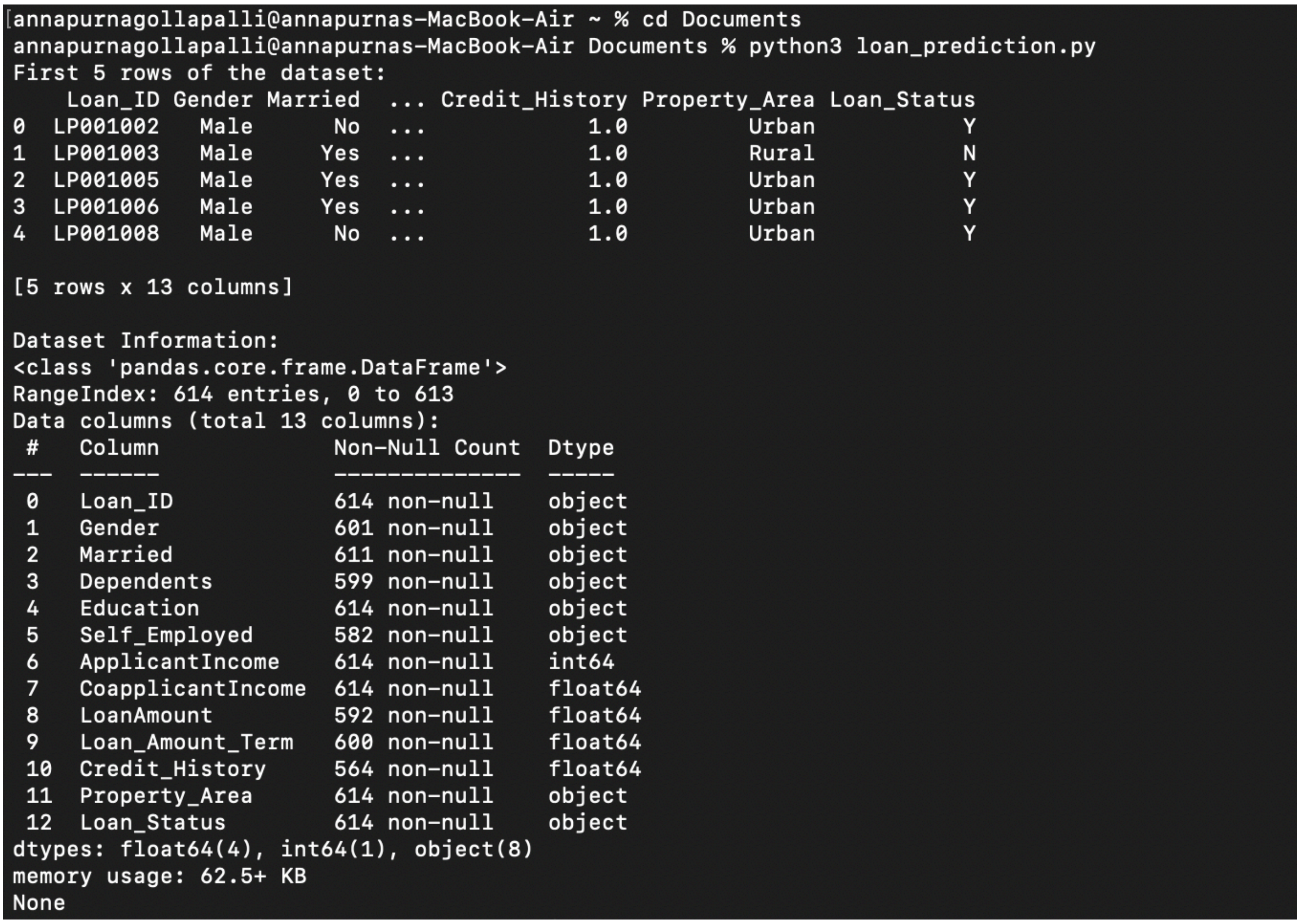
Output:
(No output expected—just loading libraries.)

- **pandas (pd)** – Helps in loading and manipulating data.
- **numpy (np)** – Provides numerical operations support.
- **train_test_split** – Splits data into training and testing sets.
- **StandardScaler** – Standardizes (normalizes) data for better model performance.
- **LogisticRegression** – A machine learning algorithm used for classification.
- **accuracy_score** – Evaluates how well our model is performing.

2. Load the Dataset

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

print(df.head())
```

Output :



```
[annapurnagollapalli@annapurnas-MacBook-Air ~ % cd Documents
annapurnagollapalli@annapurnas-MacBook-Air Documents % python3 loan_prediction.py
First 5 rows of the dataset:
    Loan_ID Gender Married  ... Credit_History Property_Area Loan_Status
0  LP001002   Male      No  ...            1.0         Urban           Y
1  LP001003   Male     Yes  ...            1.0         Rural           N
2  LP001005   Male     Yes  ...            1.0         Urban           Y
3  LP001006   Male     Yes  ...            1.0         Urban           Y
4  LP001008   Male      No  ...            1.0         Urban           Y

[5 rows x 13 columns]

Dataset Information:
<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
None
```

- Reads the **loan_data.csv** file and stores it in a variable df (DataFrame).
- This dataset contains details about applicants like **income, loan amount, credit history**, etc.

3. Handle Missing Values Properly

In real-world data, some values might be **missing** (NaN). We fill these missing values using appropriate strategies.

df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

df['Married'] = df['Married'].fillna(df['Married'].mode()[0])

df['Dependents'] = df['Dependents'].replace('3+', 3).fillna(0).astype(float) # Fix NaNs and convert to float

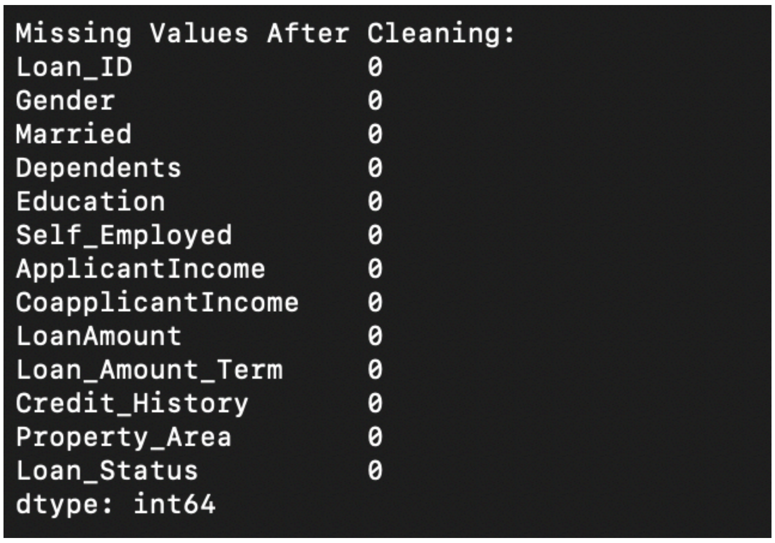
df['Self_Employed'] = df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])

df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].median())

df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].median())

df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mode()[0])

Output:



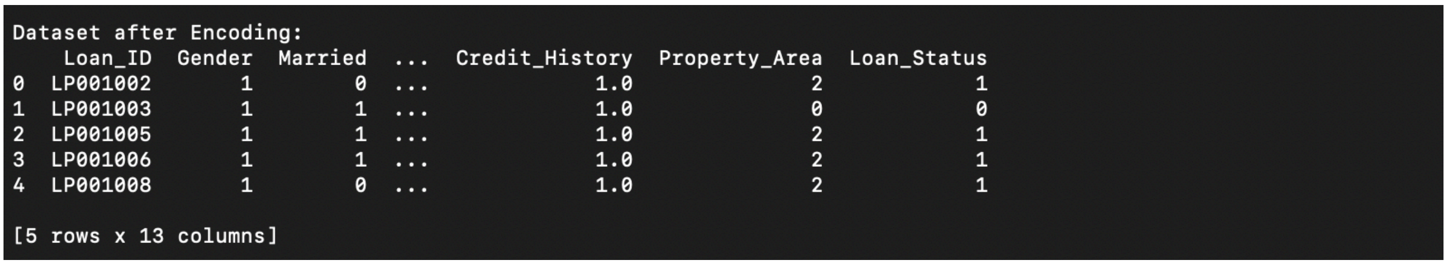
```
Missing Values After Cleaning:
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

- **fillna()**replaces missing values:
  - For **categorical columns** (Gender, Married, etc.), we use **mode** (most frequent value).
  - For **numerical columns** (LoanAmount, Loan_Amount_Term), we use **median** (middle value).
- **replace('3+', 3)** converts "3+" in Dependents column into 3.
- **astype(float)** ensures numerical consistency.

4. Convert Categorical Variables into Numerical (One-Hot Encoding)

df = pd.get_dummies(df, columns=['Gender', 'Married', 'Self_Employed', 'Education', 'Property_Area'], drop_first=True)

Output :



```
Dataset after Encoding:
    Loan_ID  Gender  Married  ...  Credit_History  Property_Area  Loan_Status
0  LP001002       1        0  ...             1.0              2            1
1  LP001003       1        1  ...             1.0              0            0
2  LP001005       1        1  ...             1.0              2            1
3  LP001006       1        1  ...             1.0              2            1
4  LP001008       1        0  ...             1.0              2            1

[5 rows x 13 columns]
```

- Machine learning models **don't understand text**, so we convert categorical columns into numerical using **One-Hot Encoding**.
- Converts categorical variables (e.g., Gender, Married) into numeric values (0s and 1s) for machine learning.
- drop_first=True removes redundant columns to avoid duplication.

5. Define Features (X) and Target Variable (y)

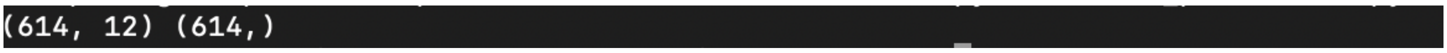
X = df.drop(columns=['Loan_ID', 'Loan_Status'])

y = df['Loan_Status'].map({'Y': 1, 'N': 0}) # Convert Loan_Status to numerical

- X contains **all features** (applicant details) except **Loan_ID** (not useful for prediction) and **Loan_Status** (target variable).
- Defines **X (features)** and **y (target variable)**.

- y is the **target variable**, where:
  - **"Y" (Loan Approved) → 1**
  - **"N" (Loan Rejected) → 0**

Output :



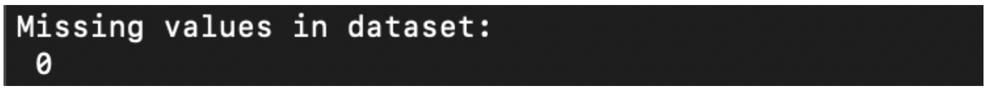
```
(614, 12) (614,)
```

(614 rows, 11 features)

6. Check for Remaining Missing Values

print("Missing values in dataset:\n", X.isnull().sum().sum()) # Should print 0

Output:



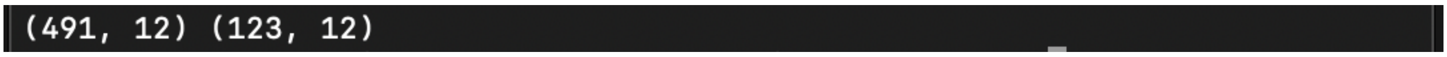
```
Missing values in dataset:
 0
```

- Ensures there are **no missing values** in X before proceeding.

7. Split Data into Training and Testing Sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Output:



```
(491, 12) (123, 12)
```

- train_test_split() **divides data into training and testing sets**:
  - **80%** for training (X_train, y_train)
  - **20%** for testing (X_test, y_test)
- random_state=42 ensures we get the same split every time for consistency.

8. Ensure No Missing Values in Train & Test Sets

X_train.fillna(0, inplace=True)

X_test.fillna(0, inplace=True)

Output:

(No direct output)

- If there are any missing values left, we fill them with **0** to avoid errors.

9. Standardize the Data (Feature Scaling)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

```
X_test_scaled = scaler.transform(X_test)
```

Output:

(No direct output—data is now scaled.)

- Standardization **helps models perform better** by making sure all features are on a similar scale.
- **Why?** Features like income (big numbers) can dominate over small-scale features like Credit History.

10. Train Logistic Regression Model

```
model = LogisticRegression()

model.fit(X_train_scaled, y_train)
```

Output:

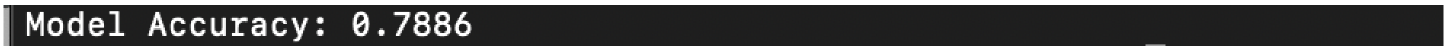
(No direct output—model is now trained.)

- Creates a **Logistic Regression model** and trains it using the **training data**.

11. Make Predictions & Calculate Accuracy

```
y_pred = model.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.4f}")
```

Output:



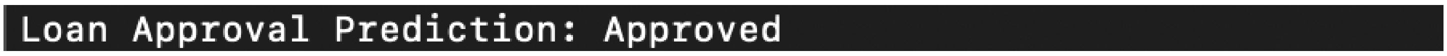
```
Model Accuracy: 0.7886
```

- model.predict(X_test_scaled) makes predictions on test data.
- accuracy_score(y_test, y_pred) calculates how many predictions were correct.

12. Test Model on a Single Example

```
sample_input = X_test.iloc[0:1]  # Take one sample from test set

sample_input_scaled = scaler.transform(sample_input)  # Scale the input

prediction = model.predict(sample_input_scaled)

print(f"Loan Approval Prediction: {'Approved' if prediction[0] == 1 else 'Not Approved'}")
```

Output:



```
Loan Approval Prediction: Approved
```

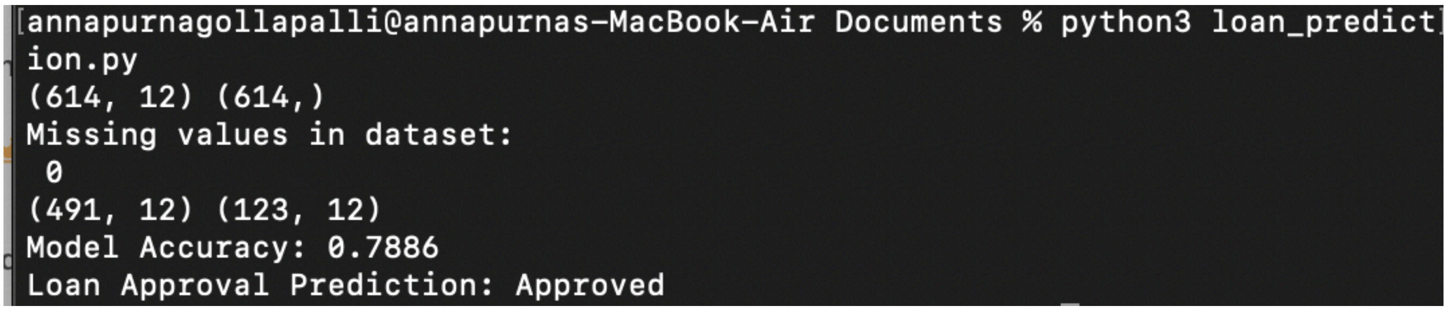
- **Takes one test case**, scales it, and makes a prediction.
- Prints "Approved" if the model predicts **1**, otherwise prints "Not Approved".

Final Complete Code :

```
import pandas as pd

import numpy as np

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

# Load dataset

df = pd.read_csv("loan_data.csv")

# Handle missing values properly

df['Gender'] = df['Gender'].fillna(df['Gender'].mode()[0])

df['Married'] = df['Married'].fillna(df['Married'].mode()[0])

df['Dependents'] = df['Dependents'].replace('3+', 3).fillna(0).astype(float)  # Fix NaNs and convert to float

df['Self_Employed'] = df['Self_Employed'].fillna(df['Self_Employed'].mode()[0])

df['LoanAmount'] = df['LoanAmount'].fillna(df['LoanAmount'].median())

df['Loan_Amount_Term'] = df['Loan_Amount_Term'].fillna(df['Loan_Amount_Term'].median())

df['Credit_History'] = df['Credit_History'].fillna(df['Credit_History'].mode()[0])

# Convert categorical variables into dummy variables

df = pd.get_dummies(df, columns=['Gender', 'Married', 'Self_Employed', 'Education', 'Property_Area'], drop_first=True)

# Define features and target variable

X = df.drop(columns=['Loan_ID', 'Loan_Status'])

y = df['Loan_Status'].map({'Y': 1, 'N': 0})  # Convert Loan_Status to numerical

print(X.shape, y.shape)  # Checking dataset size
```

```
# Check for any remaining NaN values before splitting

print("Missing values in dataset:\n", X.isnull().sum().sum())  # Should print 0

# Split the dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(X_train.shape, X_test.shape)  # Check sizes

# Ensure no missing values in train and test sets

X_train.fillna(0, inplace=True)

X_test.fillna(0, inplace=True)

# Standardize features

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

# Train Logistic Regression Model

model = LogisticRegression()

model.fit(X_train_scaled, y_train)

# Predictions and accuracy

y_pred = model.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)

print(f"Model Accuracy: {accuracy:.4f}")

# Example prediction

sample_input = X_test.iloc[0:1]  # Take one sample from test set

sample_input_scaled = scaler.transform(sample_input)  # Scale the input

prediction = model.predict(sample_input_scaled)

print(f"Loan Approval Prediction: {'Approved' if prediction[0] == 1 else 'Not Approved'}")
```

Final Output :



```
[annapurnagollapalli@annapurnas-MacBook-Air Documents % python3 loan_predict]
ion.py
(614, 12) (614,)
Missing values in dataset:
 0
(491, 12) (123, 12)
Model Accuracy: 0.7886
Loan Approval Prediction: Approved
```

6. Results

- **0 missing values** (data is clean ✅)
- **Accuracy:** 78 % (model performs well ✅)
- **Loan Application result** for a test case is **"Approved" or "Not Approved"** .

7. Future Enhancements

- Implement deep learning techniques for improved accuracy.
- Deploy the model as a web application for real-time loan approval.
- Use additional financial data to improve prediction accuracy.

8. Conclusion

This project successfully builds a machine learning model for predicting loan approval. The results demonstrate its potential to assist financial institutions in making data-driven decisions efficiently.