Loan Approval Prediction using Logistic Regression

1. Database Creation and Variable Separation

Let’s create a **sample dataset** (for illustration). In real cases, you’ll use a CSV file.

```
# Import libraries

import pandas as pd


# Create a sample dataset

data = {

    'Applicant_ID': [1, 2, 3, 4, 5, 6, 7, 8],

    'Credit_Score': [750, 680, 710, 600, 720, 590, 800, 670],

    'Applicant_Income': [60000, 45000, 52000, 40000, 70000, 38000, 75000, 46000],

    'Loan_Amount': [500000, 800000, 600000, 900000, 400000, 950000, 350000, 700000],

    'Loan_Term': [15, 20, 10, 30, 15, 30, 10, 20],

    'Loan_Approved': [1, 0, 1, 0, 1, 0, 1, 0]

}


df = pd.DataFrame(data)


# Display dataset

print(df)
```

✅ **Output:**

Applicant_ID	Credit_Score	Applicant_Income	Loan_Amount	Loan_Term	Loan_Approved
1	750	60000	500000	15	1

2	680	45000	800000	20	0
3	710	52000	600000	10	1
4	600	40000	900000	30	0
5	720	70000	400000	15	1
6	590	38000	950000	30	0
7	800	75000	350000	10	1
8	670	46000	700000	20	0

Separate Input (X) and Output (Y) Variables

```
# X -> Independent variables
X = df[['Credit_Score', 'Applicant_Income', 'Loan_Amount', 'Loan_Term']]

# y -> Dependent variable
y = df['Loan_Approved']

print("Input Features (X):")
print(X.head())

print("\nTarget Variable (y):")
print(y.head())
```

2. Data Cleaning and Preprocessing

Data cleaning ensures quality before model training.

2.1 Checking for Missing Values

Check for missing values

print(df.isnull().sum())

✅ **Output:**

Applicant_ID 0

Credit_Score 0

Applicant_Income 0

Loan_Amount 0

Loan_Term 0

Loan_Approved 0

dtype: int64

(No missing data here, but let's show how to handle it.)

2.2 Handling Missing Values (Example)

Suppose Credit_Score had missing values, we can fill them with mean:

df['Credit_Score'].fillna(df['Credit_Score'].mean(), inplace=True)

2.3 Checking Data Types and Duplicates

Check data types

print(df.dtypes)

Check duplicate records

print("Duplicate rows:", df.duplicated().sum())

2.4 Scaling / Normalizing Data

Logistic Regression performs better when numerical features are scaled.

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("\nScaled Data:")
print(X_scaled[:5])
```

3. Exploratory Data Analysis (EDA)

We’ll explore data using visualization.

3.1 Import Visualization Libraries

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

3.2 Univariate Analysis

Histogram of Credit Score

```python
plt.hist(df['Credit_Score'], bins=5, edgecolor='black')
plt.title("Distribution of Credit Scores")
plt.xlabel("Credit Score")
plt.ylabel("Count")
plt.show()
```

Box Plot of Applicant Income

```python
sns.boxplot(x=df['Applicant_Income'])
```

```
plt.title("Boxplot of Applicant Income")

plt.show()
```

3.3 Bivariate Analysis

Correlation Heatmap

```
plt.figure(figsize=(6,4))

sns.heatmap(df.corr(), annot=True, cmap='coolwarm')

plt.title("Correlation between Features")

plt.show()
```

Scatter Plot: Credit Score vs Loan Amount

```
sns.scatterplot(x='Credit_Score', y='Loan_Amount', hue='Loan_Approved', data=df)

plt.title("Credit Score vs Loan Amount")

plt.show()
```

3.4 Pair Plot

```
sns.pairplot(df, hue='Loan_Approved')

plt.show()
```

4. Model Building: Logistic Regression

4.1 Import Required Libraries

```
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

4.2 Split Data into Training and Test Sets

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=42)

print("Training Size:", X_train.shape)

print("Testing Size:", X_test.shape)
```

4.3 Create and Train Logistic Regression Model

```
# Create model

model = LogisticRegression()


# Train model

model.fit(X_train, y_train)
```

4.4 Make Predictions

```
y_pred = model.predict(X_test)


print("Predicted Values:", y_pred)

print("Actual Values:", list(y_test))
```

4.5 Evaluate Model Performance

Accuracy Score

```
accuracy = accuracy_score(y_test, y_pred)

print("Model Accuracy:", accuracy)
```

✅ **Example Output:**

Model Accuracy: 1.0

(Since our sample dataset is small, accuracy may appear perfect — larger datasets give more realistic values.)

Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.title("Confusion Matrix")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.show()
```

Classification Report

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

✅ **Example Output:**

```
          precision    recall  f1-score   support



        0       1.00      1.00      1.00         1

        1       1.00      1.00      1.00         1


    accuracy                          1.00         2

   macro avg       1.00      1.00      1.00         2

weighted avg       1.00      1.00      1.00         2
```

5. Conclusion

- Logistic Regression was applied successfully to predict **loan approval** using applicant data.
- Important factors influencing approval were **Credit Score, Applicant Income, and Loan Amount**.
- Model achieved **high accuracy** on test data (for small dataset = 100%).

- For real-world applications:
    - Use larger datasets.
    - Apply **cross-validation** for better generalization.
    - Analyze feature importance for decision-making transparency.

\Why You Choose logistic regression only

1. Nature of the Problem

- The loan approval task is a binary classification problem — an applicant's loan is either:
    - Approved (1), or
    - Rejected (0)

Logistic Regression is specifically designed for such binary outcome prediction problems. It estimates the probability of a loan being approved, making it ideal for yes/no type decisions.

Logistic Regression is mathematically simple, computationally fast, and works very well on small to medium-sized datasets.

It's the baseline model in most financial prediction pipelines — you always start with Logistic Regression to understand relationships before moving to complex models.