**CASE STUDY PYTHON**

**29-07-2025**

1 . import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

2. df = pd.read\_csv('LoanData (1).csv')

3. print(df.head()) # Displays the first few rows of the DataFrame

output:

Loan\_ID Gender Married Dependents Education Self\_Employed \

0 LP001002 Male No 0 Graduate No

1 LP001003 Male Yes 1 Graduate No

2 LP001005 Male Yes 0 Graduate Yes

3 LP001006 Male Yes 0 Not Graduate No

4 LP001008 Male No 0 Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \

0 5849 0.0 NaN 360.0

1 4583 1508.0 128.0 360.0

2 3000 0.0 66.0 360.0

3 2583 2358.0 120.0 360.0

4 6000 0.0 141.0 360.0

Credit\_History Property\_Area Loan\_Status

0 1.0 Urban Y

1 1.0 Rural N

2 1.0 Urban Y

3 1.0 Urban Y

4 1.0 Urban Y

4. # Check the shape of the dataset

print(df.shape)

# Display the first few rows

print(df.head())

output :

(614, 13)

Loan\_ID Gender Married Dependents Education Self\_Employed \

0 LP001002 Male No 0 Graduate No

1 LP001003 Male Yes 1 Graduate No

2 LP001005 Male Yes 0 Graduate Yes

3 LP001006 Male Yes 0 Not Graduate No

4 LP001008 Male No 0 Graduate No

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \

0 5849 0.0 NaN 360.0

1 4583 1508.0 128.0 360.0

2 3000 0.0 66.0 360.0

3 2583 2358.0 120.0 360.0

4 6000 0.0 141.0 360.0

Credit\_History Property\_Area Loan\_Status

0 1.0 Urban Y

1 1.0 Rural N

2 1.0 Urban Y

3 1.0 Urban Y

4 1.0 Urban Y

5. # Check for missing values

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

output :

Loan\_ID 0

Gender 13

Married 3

Dependents 15

Education 0

Self\_Employed 32

ApplicantIncome 0

CoapplicantIncome 0

LoanAmount 22

Loan\_Amount\_Term 14

Credit\_History 50

Property\_Area 0

Loan\_Status 0

dtype: int64

6. # Filling missing values for LoanAmount

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

# Filling missing values for Loan\_Amount\_Term

df['Loan\_Amount\_Term'] = df['Loan\_Amount\_Term'].fillna(df['Loan\_Amount\_Term'].mode()[0])

# Filling missing values for Credit\_History

df['Credit\_History'] = df['Credit\_History'].fillna(1) # Most common value is 1

7. print(df.describe())

output:

ApplicantIncome CoapplicantIncome LoanAmount Loan\_Amount\_Term \

count 614.000000 614.000000 614.000000 614.000000

mean 5403.459283 1621.245798 145.752443 342.410423

std 6109.041673 2926.248369 84.107233 64.428629

min 150.000000 0.000000 9.000000 12.000000

25% 2877.500000 0.000000 100.250000 360.000000

50% 3812.500000 1188.500000 128.000000 360.000000

75% 5795.000000 2297.250000 164.750000 360.000000

max 81000.000000 41667.000000 700.000000 480.000000

Credit\_History

count 614.000000

mean 0.855049

std 0.352339

min 0.000000

25% 1.000000

50% 1.000000

75% 1.000000

max 1.000000

8. import seaborn as sns

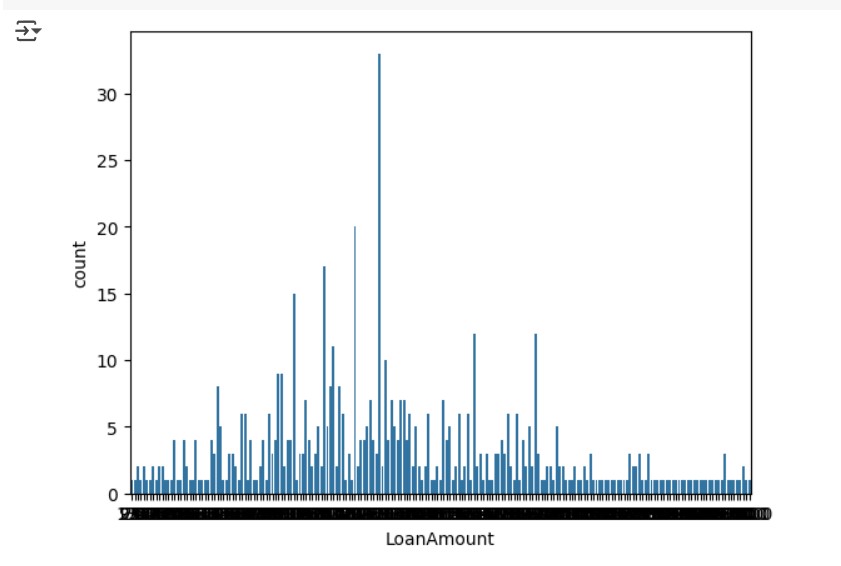
import matplotlib.pyplot as plt

# Example visualization

sns.countplot(x='LoanAmount', data=df)

plt.show()

output:



**DATA PREPROCESSING**

9.# Import necessary libraries

from sklearn.preprocessing import OneHotEncoder

# Create a OneHotEncoder instance

encoder = OneHotEncoder(drop='first', sparse\_output=False)  # Use sparse\_output instead of sparse

# Select categorical columns to encode

categorical\_cols = ['Gender', 'Married', 'Education', 'Self\_Employed', 'Property\_Area']

# Fit and transform the categorical columns

encoded\_features = encoder.fit\_transform(df[categorical\_cols])

# Create a DataFrame from the encoded features

encoded\_df = pd.DataFrame(encoded\_features, columns=encoder.get\_feature\_names\_out(categorical\_cols))

# Drop original categorical columns and concatenate the new encoded DataFrame

df = df.drop(categorical\_cols, axis=1)

df = pd.concat([df, encoded\_df], axis=1)

NO OUTPUT

b)from sklearn.preprocessing import StandardScaler

# Initialize the scaler

scaler = StandardScaler()

# Select numerical columns to scale

numerical\_cols = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']

# Scale the numerical features

df[numerical\_cols] = scaler.fit\_transform(df[numerical\_cols])

NO OUTPUT

**EXPLORATORY DATA ANALYSIS (EDA)**

import seaborn as sns

import matplotlib.pyplot as plt

# Drop non-numeric columns (e.g., Loan\_ID)

df\_numeric = df.select\_dtypes(include=['float64', 'int64'])

# Correlation heatmap

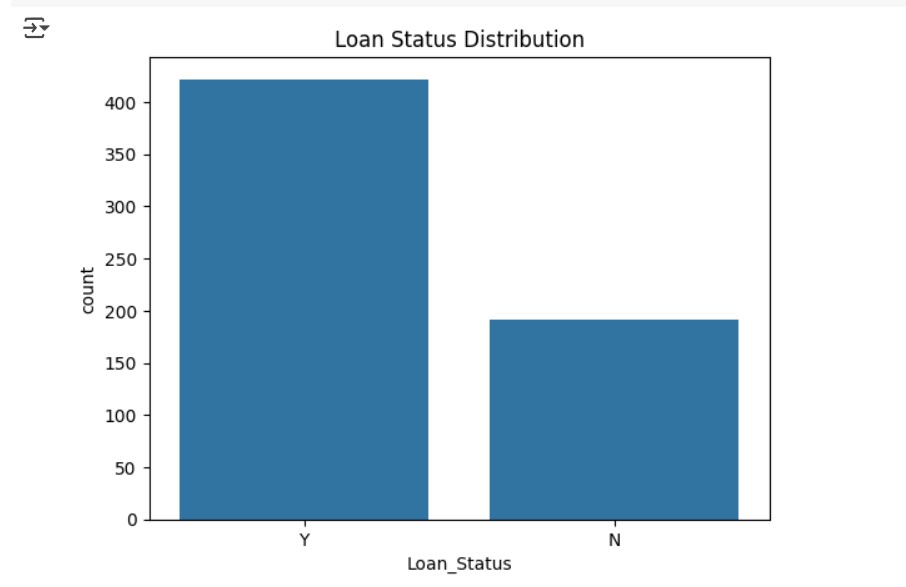
plt.figure(figsize=(10, 6))

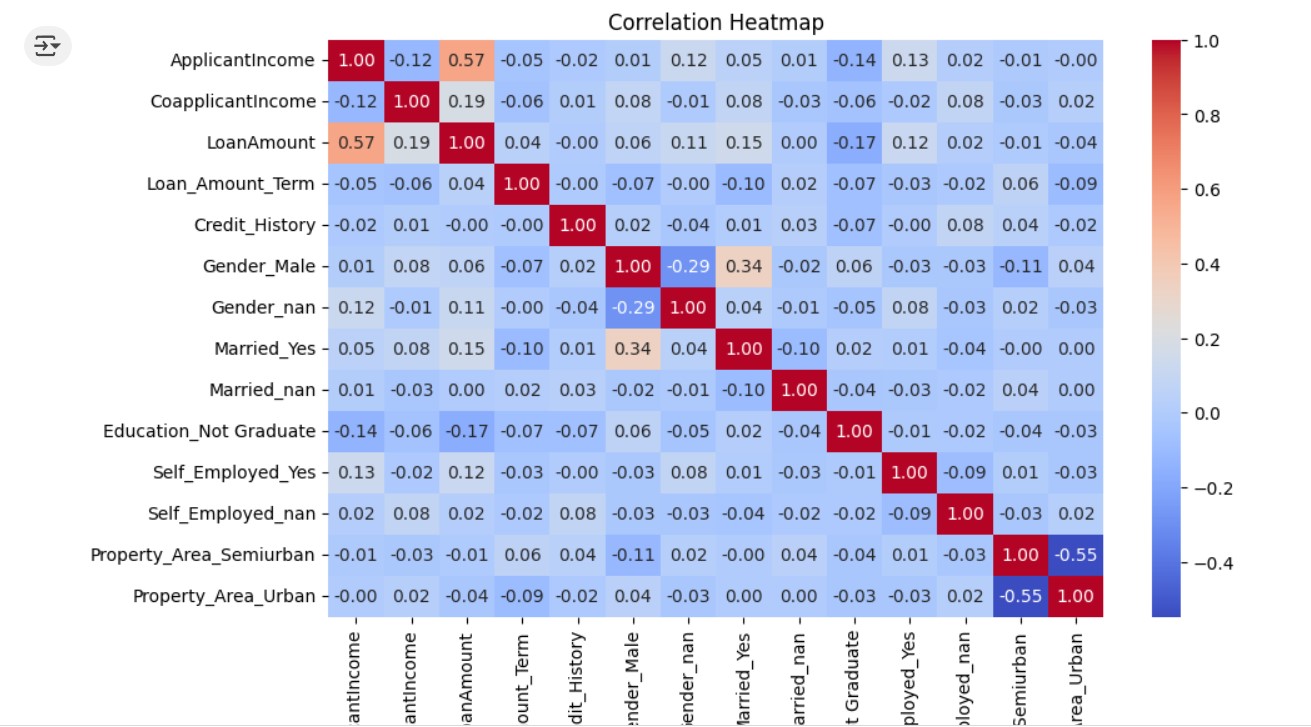
sns.heatmap(df\_numeric.corr(), annot=True, fmt=".2f", cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

OUTPUT:





**MODEL BUILDING**

from sklearn.model\_selection import train\_test\_split

# Define features and target variable

X = df.drop('Loan\_Status', axis=1)

y = df['Loan\_Status'].map({'Y': 1, 'N': 0}) # Convert Loan\_Status to binary

# Split the dataset 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)

NO OUTPUT

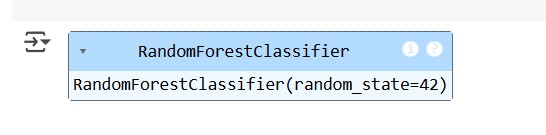
11. # Convert 'Dependents' to numeric after replacing values

X\_train['Dependents'] = X\_train['Dependents'].replace({'0': 0, '1': 1, '2': 2, '3+': 3}).astype(float)

# Now you can train the model

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

OUTPUT:



**MODEL EVALUATION**

**Performance Metrics**

from sklearn.ensemble import RandomForestClassifier

# Initialize the model

model = RandomForestClassifier(random\_state=42)

# Replace and convert 'Dependents' to numeric

X\_train['Dependents'] = X\_train['Dependents'].replace({'0': 0, '1': 1, '2': 2, '3+': 3}).astype(float)

# Train the model using the existing X\_train and y\_train

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

# Optionally, you can print the model's feature importances

print("Feature importances:", model.feature\_importances\_)

OUTPUT:

Feature importances: [5.50629308e-02 1.99081025e-01 1.08024181e-01 1.81769477e-01

5.08332462e-02 2.41528548e-01 2.11506819e-02 8.32376687e-03

2.63166992e-02 1.32363219e-04 2.41972732e-02 1.86704258e-02

1.14368602e-02 3.31012357e-02 2.03712848e-02]

**FINAL CODE AS A WHOLE WITH HYPERPARAMETER TUNING**

# ============== 1. IMPORTS & SETUP ==============

import pandas as pd

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.preprocessing import OneHotEncoder

import seaborn as sns

import matplotlib.pyplot as plt

from flask import Flask, request, jsonify

# ============== 2. DATA PREPROCESSING ==============

# Load data

df = pd.read\_csv('LoanData (1).csv')

# Handle missing values (example)

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

df['Credit\_History'] = df['Credit\_History'].fillna(1)

# Encode categorical variables

encoder = OneHotEncoder(drop='first', sparse\_output=False)

categorical\_cols = ['Gender', 'Married', 'Education', 'Self\_Employed', 'Property\_Area']

encoded\_features = encoder.fit\_transform(df[categorical\_cols])

encoded\_df = pd.DataFrame(encoded\_features, columns=encoder.get\_feature\_names\_out(categorical\_cols))

df = pd.concat([df.drop(categorical\_cols, axis=1), encoded\_df], axis=1)

# Convert 'Dependents' to numeric

df['Dependents'] = df['Dependents'].replace({'0':0, '1':1, '2':2, '3+':3}).astype(float)

# Drop Loan\_ID before splitting

df = df.drop(['Loan\_ID'], axis=1, errors='ignore')

# Split data

X = df.drop(['Loan\_Status'], axis=1) # Ensure Loan\_Status is the only target variable

y = df['Loan\_Status'].map({'Y':1, 'N':0})

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# ============== 3. MODEL TRAINING ==============

model = RandomForestClassifier(random\_state=42)

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

# ============== 4. EVALUATION ==============

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

print(classification\_report(y\_test, y\_pred))

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues')

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.show()

# ============== 5. HYPERPARAMETER TUNING ==============

param\_grid = {

'n\_estimators': [100, 200],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5]

}

grid\_search = GridSearchCV(RandomForestClassifier(random\_state=42), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

best\_model = grid\_search.best\_estimator\_

# ============== 6. DEPLOYMENT ==============

app = Flask(\_\_name\_\_)

@app.route('/predict', methods=['POST'])

def predict():

data = request.json

# Preprocess the input data similar to training data

input\_data = pd.DataFrame([data]) # Wrap data in a list to create a DataFrame

# Drop Loan\_ID if it exists

input\_data = input\_data.drop(['Loan\_ID'], axis=1, errors='ignore')

# Convert 'Dependents' to numeric

if 'Dependents' in input\_data.columns:

input\_data['Dependents'] = input\_data['Dependents'].replace({'0': 0, '1': 1, '2': 2, '3+': 3}).astype(float)

# One-hot encode categorical variables

encoded\_input = encoder.transform(input\_data[categorical\_cols])

encoded\_input\_df = pd.DataFrame(encoded\_input, columns=encoder.get\_feature\_names\_out(categorical\_cols))

# Concatenate the encoded features with the rest of the input data

input\_data = pd.concat([input\_data.drop(categorical\_cols, axis=1), encoded\_input\_df], axis=1)

# Make prediction

prediction = best\_model.predict(input\_data)

return jsonify({'Loan\_Status': 'Y' if prediction[0] == 1 else 'N'})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True) # Set debug=False in production

OUTPUT:

