SmartLender - Applicant Credibility Prediction for Loan Approval

TEAM MEMBERS -
Pranav Kalra - pranav.kalra2023@vitstudent.ac.in
Palak Goyal - palak.goyal2023@vitstudent.ac.in
Kashish Agrawal - kashish.agrawal2023@vitstudent.ac.in
Divyam Khetan - divyamhemant.khetan2023@vitstudent.ac.in

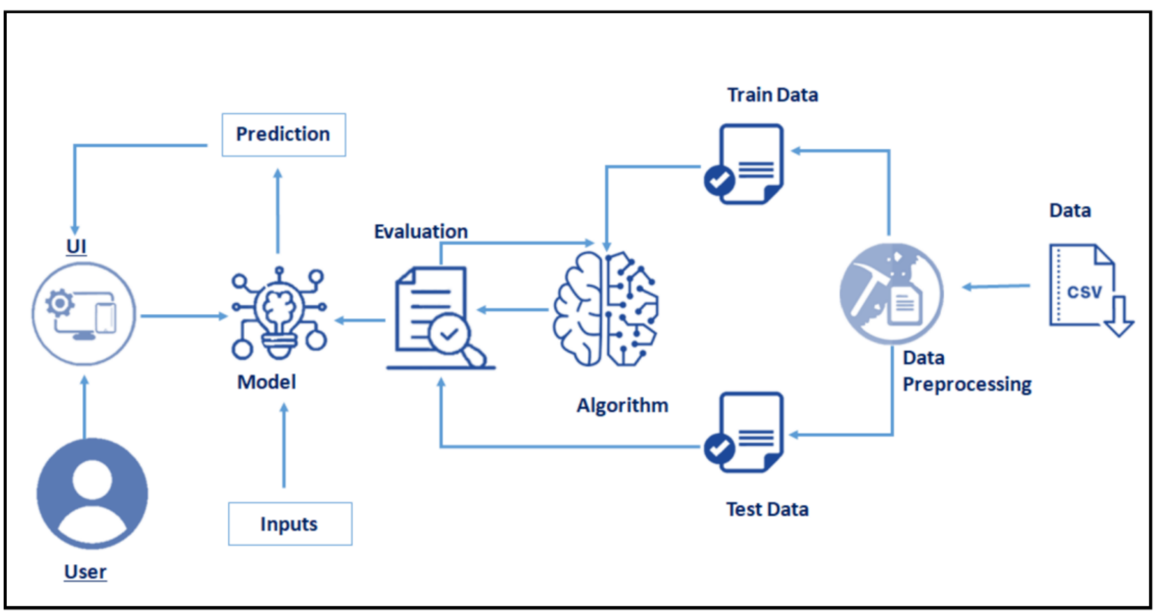
Smart Lender – Applicant Credibility Prediction for Loan Approval Using Machine Learning

In today's fast-moving financial environment, applying for a **loan** is a common step for individuals looking to meet personal, educational, or business goals. However, the loan approval process is often unclear and confusing. Many applicants face **rejection without understanding why**, which leads to disappointment and loss of trust in the system. Critical factors like **income, credit history, employment type, loan amount, and dependents** influence approval, but these are rarely explained to users in a transparent way.

To solve this issue, we developed a machine learning-based system called **Smart Lender**. This tool analyzes various applicant details such as **gender, marital status, education, applicant income, co-applicant income, loan amount, loan term, credit history, and property area** to predict whether a loan application is likely to be **approved or not approved**.

The system uses a trained machine learning model to give a **simple binary prediction** that acts as a pre-check. While it doesn't replace the actual loan process or give financial advice, it helps users understand their likelihood of approval **before** officially applying for a loan. This allows applicants to make more informed decisions and avoid unnecessary rejections.

Technical Architecture:



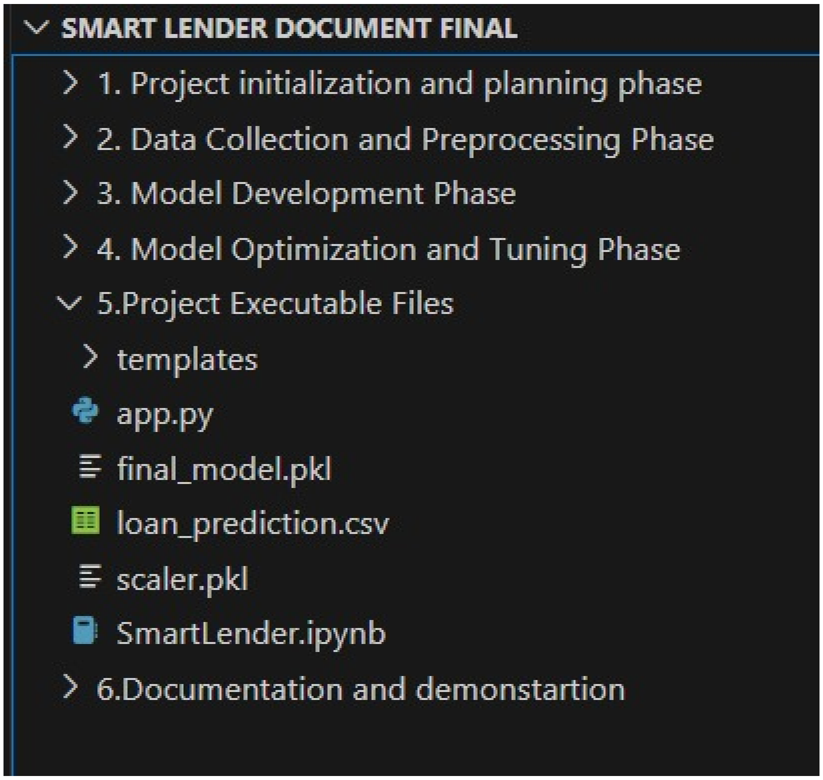
Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once the model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below:

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social or Business Impact
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Training the model in multiple algorithms
  - Testing the model
- Performance Testing & Hyperparameter Tuning
  - Testing model with multiple evaluation metrics
  - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
  - Save the best model
  - Integrate with Web Framework
- Project Demonstration & Documentation
  - Record explanation Video for project end to end solution
  - Project Documentation - Step by step project development procedure

Project Structure:



Milestone 1: Project Initialization and Planning Phase

Activity 1: Specify the Business Problem

This project, *Smart Lender - Applicant Credibility Prediction for Loan Approval using Machine Learning*, is aimed at helping financial institutions make faster and more accurate loan approval decisions. By analyzing a variety of applicant details such as income, credit history, employment status, and other relevant attributes, the system predicts whether a loan applicant is likely to repay the loan or default. This prediction helps in reducing financial risks for lenders and improves overall efficiency in loan processing.

Activity 2: Business Requirements

The Smart Lender project can have multiple business requirements, depending on the objectives of lending institutions. Some possible requirements may include:

- **Accurate and timely predictions**: The project should be able to analyze applicant data and generate reliable predictions that reflect their real-time creditworthiness. This helps ensure better decision-making for loan approvals.
- **Flexibility**: The prediction system should be flexible enough to adapt to different types of data and changing loan policies or financial environments.
- **Compliance**: The system must comply with all relevant financial and data protection regulations, such as RBI guidelines and ethical AI standards, to ensure fair and legal processing of applicant data.
- **User-friendly interface**: The system should be easy to use for bank staff or loan officers, showing clear credibility scores and insights that assist them in making informed decisions.

Activity 3: Literature Survey

A literature survey for this project would involve reviewing past research, case studies, and scholarly articles focused on the use of machine learning in financial decision-making. The survey will look into existing models used for credit scoring, such as logistic regression, decision trees, random forests, and neural networks, and analyze their performance and limitations.

It will also cover studies discussing challenges such as model interpretability, bias in predictions, and the ethical implications of using AI in finance. This information will help in understanding how to improve upon existing systems and develop a reliable and fair applicant credibility prediction model.

Activity 4: Social or Business Impact

Social Impact:
This project can support financial inclusion by helping people who don't have a traditional credit score—such as self-employed individuals or people from rural areas—gain access to loans. With a fair ML-based system, these applicants can still be evaluated based on alternative data.

Business Model / Impact:
For lending institutions, this system can help speed up the loan approval process and reduce default rates by predicting high-risk applicants in advance. It can also help in customizing loan offers based on applicant profiles, which ultimately improves profitability and reduces human errors in decision-making

<u>Milestone 2: Data Collection & Preparation</u>

Machine learning depends heavily on data — it is the most crucial component that enables algorithm training and effective prediction. In this milestone, we focus on data collection, understanding, and initial analysis.

Activity 1: Collect the Dataset

In this project, we have used the dataset provided in the **Skill Wallet by Smart Intern** as part of the internship resources. The file name of the dataset is **<u>loan_prediction.csv</u>**.

This dataset includes various fields related to loan applicants such as:

- **Gender**
- **Marital Status**
- **Dependents**
- **Education**
- **Self_Employed**
- **ApplicantIncome**
- **CoapplicantIncome**
- **LoanAmount**
- **Loan_Amount_Term**
- **Credit_History**
- **Property_Area**
- **Loan_Status**

Data Understanding & Analysis

Once the dataset is loaded, it is essential to explore and understand the structure and quality of the data. We used basic data exploration and visualization techniques to:

- Identify **missing values** or **null entries**.
- Analyze the **distribution** of numerical fields like income and loan amounts.
- Review **categorical values** like Gender, Education, etc., for consistency and class balance.
- Understand **correlations** between input features and the target variable (Loan_Status).

Activity 1.1: Importing the libraries

To begin working with our dataset and perform data preprocessing, visualization, model training, and evaluation, we need to import the necessary Python libraries. The following code snippet shows the libraries we used in this project.

```
# Import Basic Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Libraries to split , Encode data and to get final report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

#Models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# XGBoost
from xgboost import XGBClassifier

# For Scaling
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler

#To evaluate performance
from sklearn.model_selection import cross_val_score
```

Activity 1.2: Read the Dataset

The dataset used in our project is in `.csv` format and was provided by **SmartInternz** under the Skill Wallet program. It contains loan applicant information used to build the **Smart Lender - Applicant Credibility Prediction** system.

To read the dataset, we used the `read_csv()` function from the `pandas` library, which allows us to import data directly into a DataFrame for analysis and manipulation.

```
df = pd.read_csv("/content/loan_prediction.csv")
df.head()
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Te
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360

Activity 2: Data Preparation

After gaining a basic understanding of the dataset, the next important step is to **pre-process the data** so it can be effectively used to train machine learning models.

The raw dataset, as downloaded, may contain inconsistencies, missing values, or outliers. Therefore, cleaning and preparing the data is crucial to ensure accurate predictions and good model performance.

This activity includes the following key preprocessing steps:

- **Handling missing values**
- **Handling outliers**

Activity 2.1: Handling missing values

After reading the dataset, the next important step is to check for any missing (null) values. This helps in determining whether any cleaning or imputation is required before moving on to model training.

We used the following code:

```
df.isnull().sum()
```

	0
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

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive Statistical Analysis

Descriptive analysis helps us understand the **basic structure and summary statistics** of the dataset. It allows us to get a quick overview of the distribution, central tendency, and spread of the data, which is crucial before applying any machine learning models.

In our project, we used the `.describe()` function from the **pandas** library to generate descriptive statistics for numerical columns:

```
print(df.describe())
print(df.describe(include='object'))
```

```
       Dependents  ApplicantIncome  CoapplicantIncome  LoanAmount  \
count  614.000000       614.000000         614.000000  614.000000
mean     0.744300      5403.459283        1621.245798  145.752443
std      1.009623      6109.041673        2926.248369   84.107233
min      0.000000       150.000000           0.000000    9.000000
25%      0.000000      2877.500000           0.000000  100.250000
50%      0.000000      3812.500000        1188.500000  128.000000
75%      1.000000      5795.000000        2297.250000  164.750000
max      3.000000     81000.000000       41667.000000  700.000000

       Loan_Amount_Term  Credit_History
count        614.000000      614.000000
mean         342.410423        0.855049
std           64.428629        0.352339
min           12.000000        0.000000
25%          360.000000        1.000000
50%          360.000000        1.000000
75%          360.000000        1.000000
max          480.000000        1.000000
       Gender Married Education Self_Employed Property_Area Loan_Status
count     614     614       614           614           614         614
unique      2       2         2             2             3           2
top      Male     Yes  Graduate            No     Semiurban           Y
freq      502     401       480           532           233         422
```

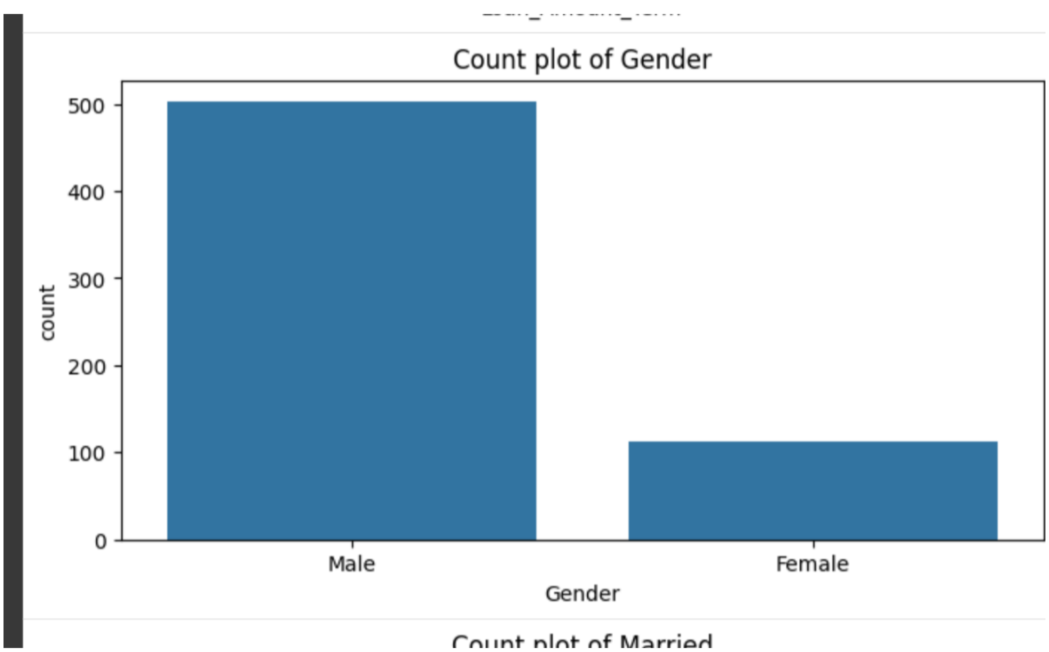
Activity 2: Visual Analysis

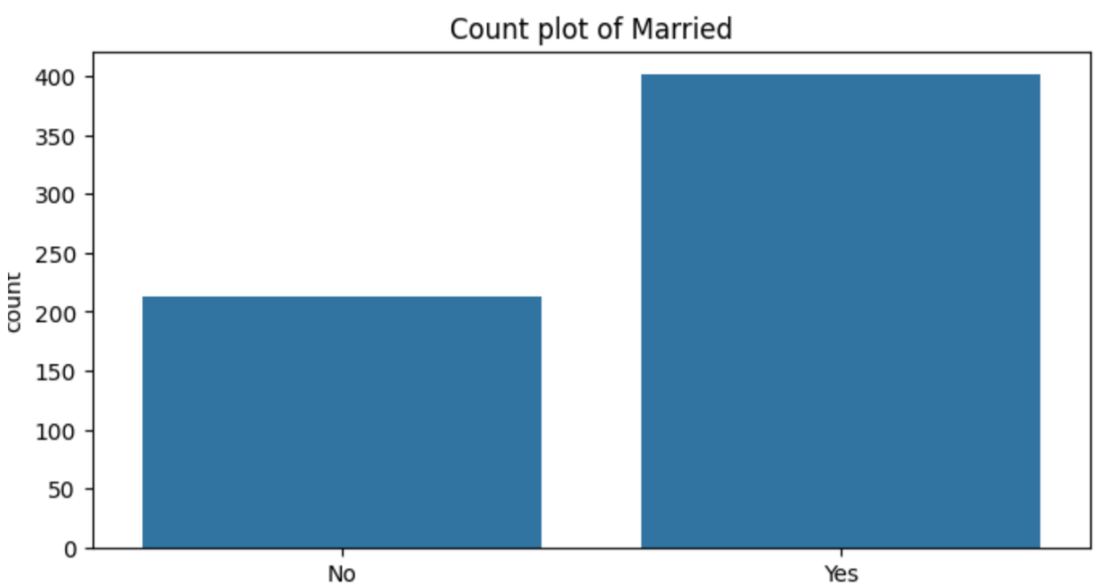
Visual analysis involves using charts, graphs, and plots to better understand the structure and distribution of the data. It helps in identifying **patterns, trends, class imbalance**, and **outliers** more effectively than just numerical summaries.

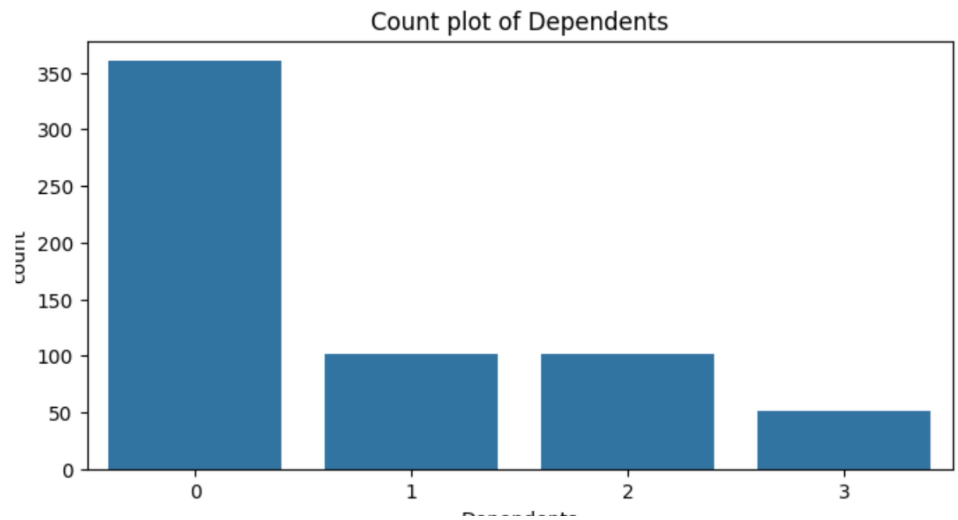
By visualizing the data, we can make **informed decisions** about how to process features and build models that are better aligned with the problem.

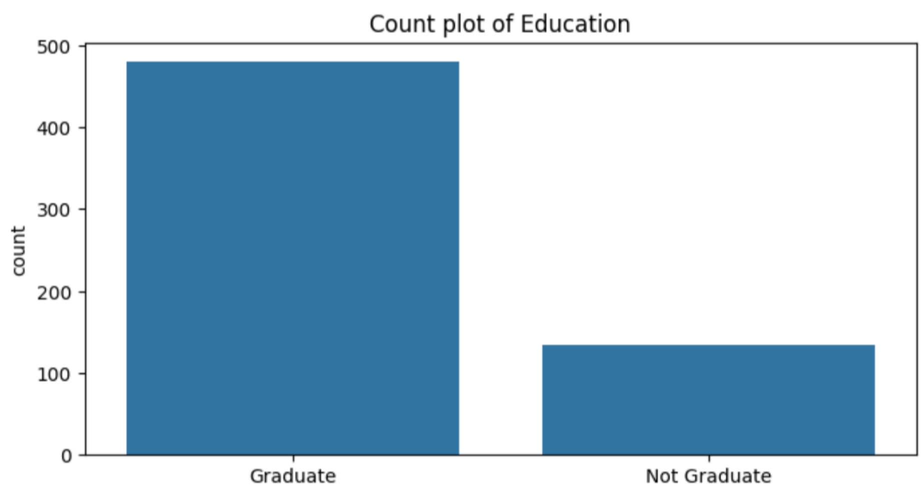
Activity 2.1: Univariate Analysis

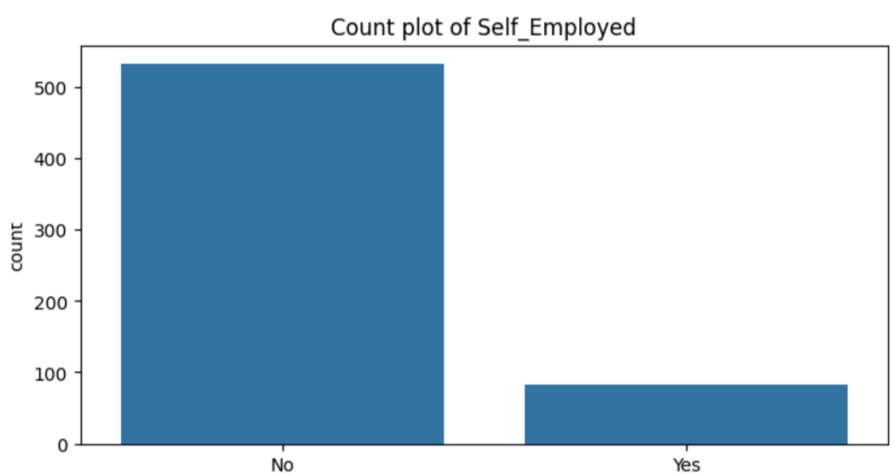
Univariate analysis focuses on analyzing **one variable at a time** to understand its distribution. This is especially useful for **categorical features** in our loan dataset such as:

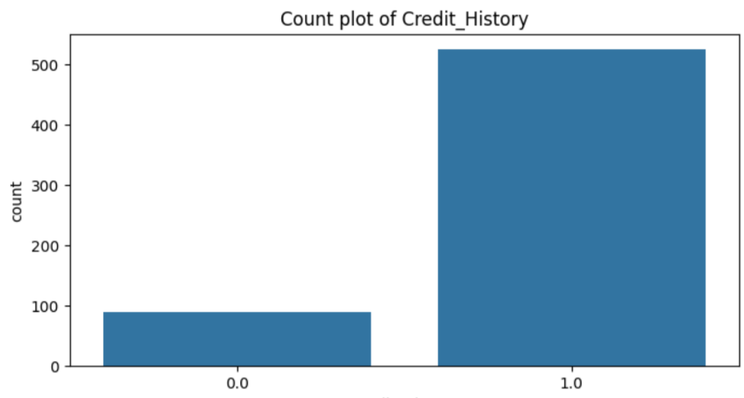


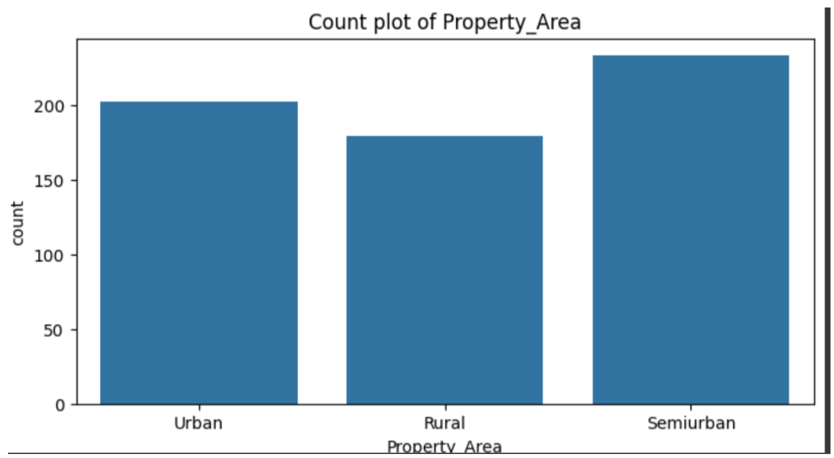



Count plot of Dependents
350
300
250
200
150
100
50
0
0
1
2
3


Count plot of Education
500
400
300
200
100
0
count
Graduate
Not Graduate

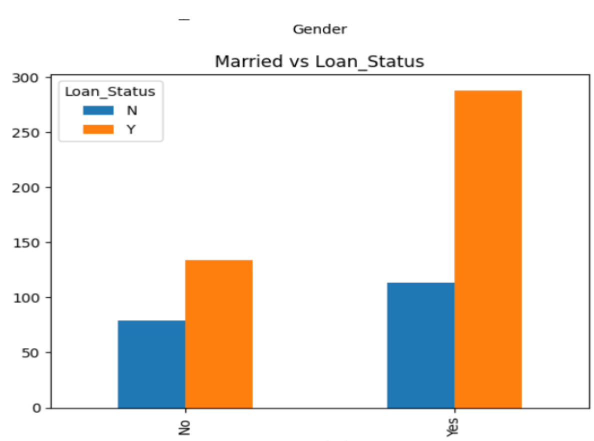

Count plot of Self_Employed
500
400
300
200
100
0
count
No
Yes

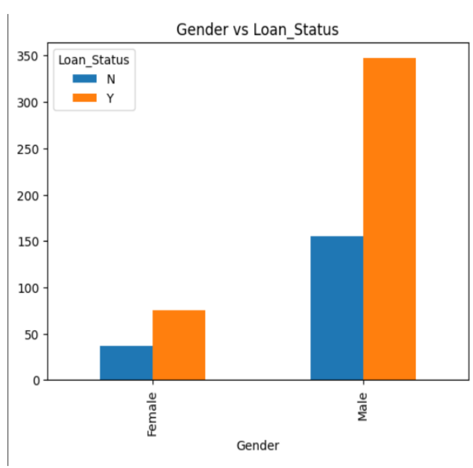




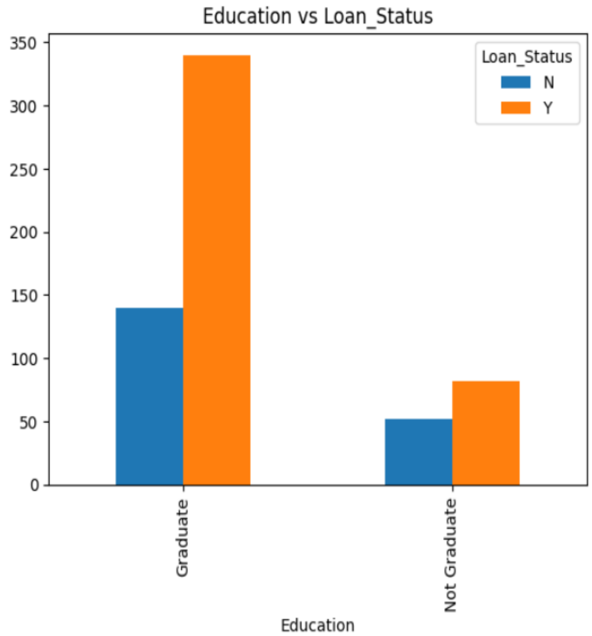
Activity 2.2: Bivariate analysis

```
#Categorical features
for col in ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area']:
    pd.crosstab(df[col], df['Loan_Status']).plot(kind='bar')
    plt.title(f'{col} vs Loan_Status')
    plt.show()

#Numerical features
for col in ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']:
    df.boxplot(column=col, by='Loan_Status')
    plt.title(f'{col} by Loan_Status')
    plt.suptitle('')
    plt.show()
```

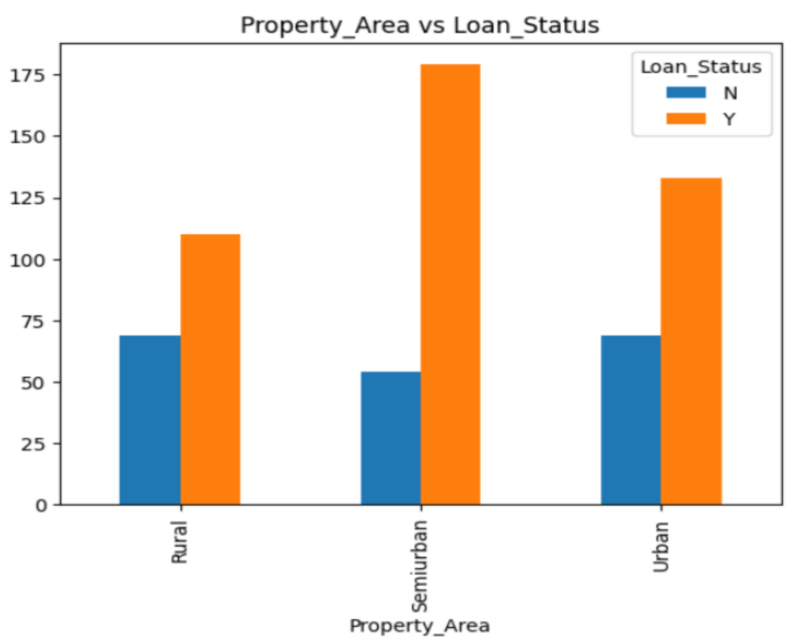


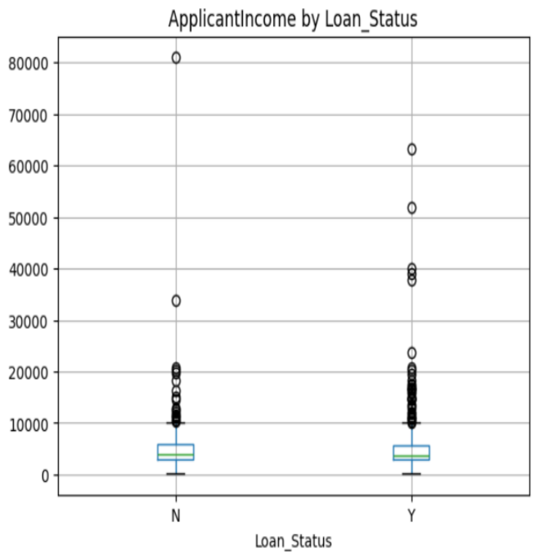


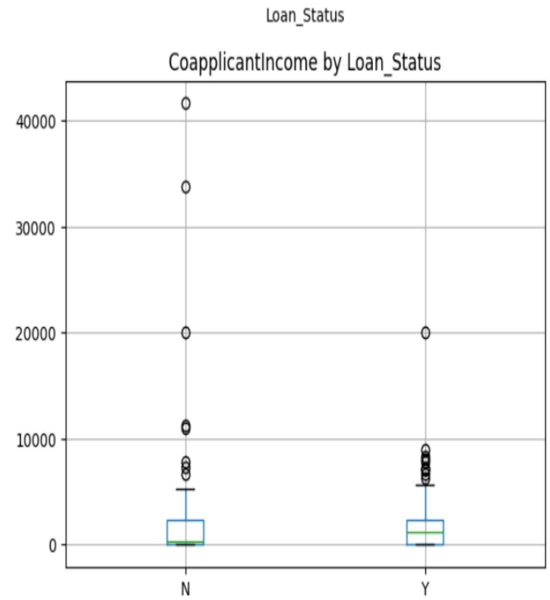
Education

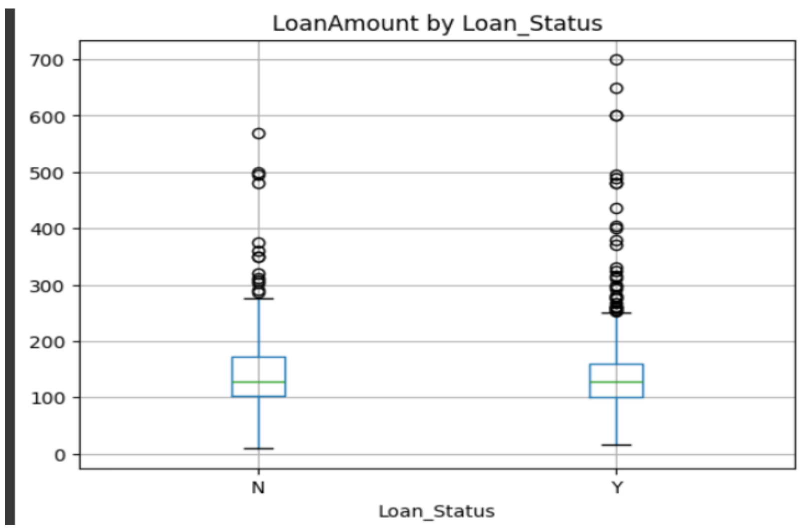

Education vs Loan_Status
Loan_Status
N
Y
350
300
250
200
150
100
50
0
Graduate
Not Graduate
Education


Self_Employed vs Loan_Status
Loan_Status
N
Y
350
300
250
200
150
100
50
0
No
Yes
Self_Employed


Property_Area vs Loan_Status
Loan_Status
N
Y
175
150
125
100
75
50
25
0
Rural
Semiurban
Urban
Property_Area



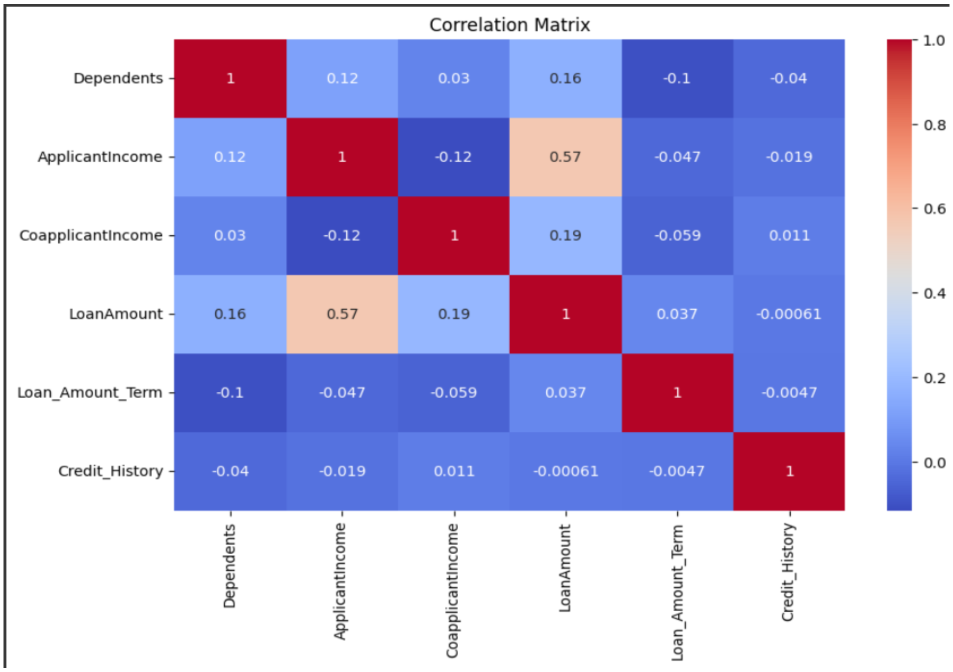


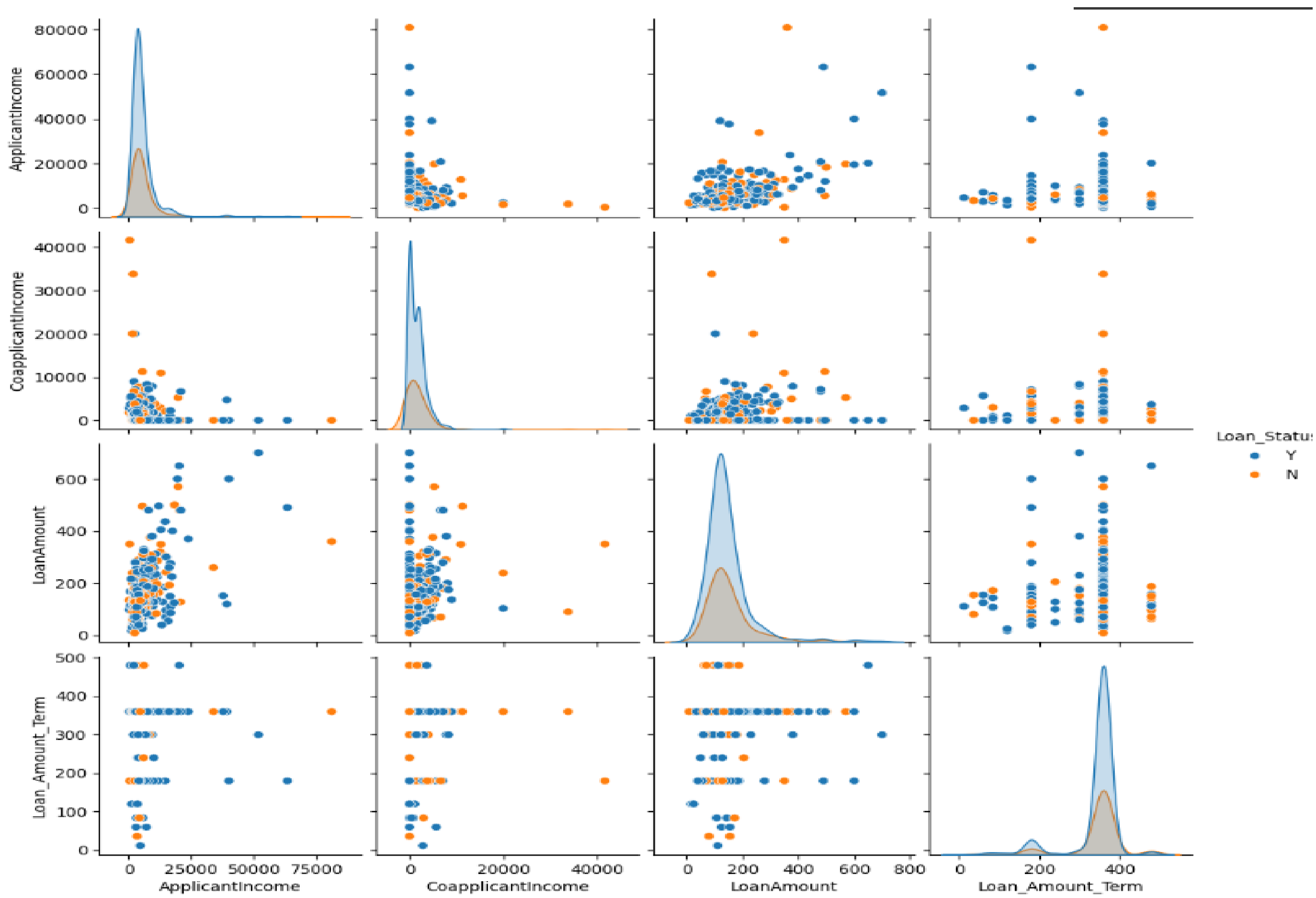


Activity 2.3: Multivariate analysis

Multivariate Analysis

```
# Correlation matrix
corr = df.select_dtypes(include=np.number).corr()
plt.figure(figsize=(10,6))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()

# Pairplot for numerical features
sns.pairplot(df[num_cols + ['Loan_Status']], hue='Loan_Status')
plt.show()
```


Correlation Matrix
Dependents
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History


ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Loan_Status
Y
N

Encoding the Categorical Features

In our loan prediction dataset, many of the features are **categorical**, such as:

- `Gender`
- `Married`
- `Education`
- `Self_Employed`
- `Property_Area`
- `Loan_Status` (target)

Machine learning models **cannot directly work with categorical values**, so we need to convert them into numerical form — a process known as **encoding**

```
[79] df['Loan_Status'] = df['Loan_Status'].map({'Y': 1, 'N': 0})
     df['Dependents'] = df['Dependents'].replace('3+', 3).astype(int)

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

Splitting Data into Train and Test Sets

Once our dataset is cleaned and all features are encoded into numerical values, the next step is to **split the data** into **training** and **testing** sets. This is essential for evaluating how well our machine learning model performs on unseen data.

```
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']

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

Handling Imbalanced Dataset

In machine learning, **imbalanced data** is a situation where the target variable has a **disproportionate number of observations in each class**. For example, in our case, the number of applicants whose loan is approved (`Loan_Status = 1`) may be significantly higher than those whose loan is not approved (`Loan_Status = 0`).

Balancing And Scaling

```python
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']

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

print("Train Target Distribution:")
print(y_train.value_counts())

print("Test Target Distribution:")
print(y_test.value_counts())
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train_smote, y_train_smote, test_size=0.2, random_state=42
)
```

Scaling

Scaling is an important **preprocessing step** in machine learning. It is used to bring all features to the **same scale** so that no particular feature dominates or biases the model because of its magnitude.

```python
X = df.drop('Loan_Status', axis=1)
y = df['Loan_Status']

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

print("Train Target Distribution:")
print(y_train.value_counts())

print("Test Target Distribution:")
print(y_test.value_counts())
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

X_train_final, X_val, y_train_final, y_val = train_test_split(
    X_train_smote, y_train_smote, test_size=0.2, random_state=42
)
```

```
Train Target Distribution:
Loan_Status
1    337
0    154
Name: count, dtype: int64
Test Target Distribution:
Loan_Status
1    85
0    38
Name: count, dtype: int64
```

Milestone 4: Model Building

Activity 1: Training the Model Using Multiple Algorithms

Now that our data is fully preprocessed (cleaned, balanced, scaled, and split), we move on to **training machine learning models**. For this project, we applied **three different classification algorithms** to predict whether a loan should be approved or not.

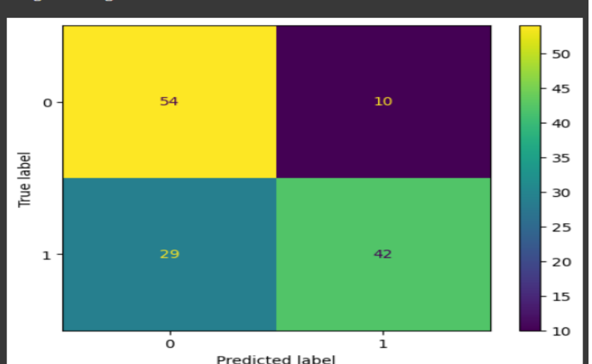
We trained each model, evaluated their performance, and selected the **best-performing model** for final use based on accuracy and other metrics.

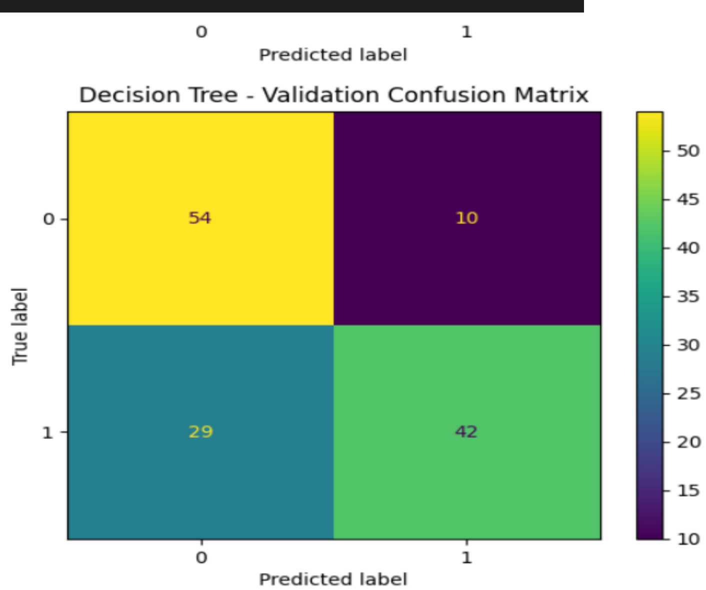
Activity 1.1: Decision Tree Model

For our first model, we used the **Decision Tree Classifier** from the `sklearn` library. A decision tree works by splitting data based on feature values in a tree-like structure until it arrives at a decision.

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
import matplotlib.pyplot as plt

# Model dictionary
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "KNN": KNeighborsClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
}

# Dictionary to store accuracy results
results = {}

# Loop through each model
for name, model in models.items():
    print(f"\n🔷 Training {name}...")

    # Train on SMOTE-balanced training data
    model.fit(X_train_final, y_train_final)

    # -------- Validation Performance --------
    val_preds = model.predict(X_val)
    print(f"\n📋 {name} - Validation Results:")
    print(classification_report(y_val, val_preds))
    ConfusionMatrixDisplay.from_predictions(y_val, val_preds).plot()
    plt.title(f"{name} - Validation Confusion Matrix")
    plt.show()
```

```
🔷 Training Decision Tree...

📋 Decision Tree - Validation Results:
              precision    recall  f1-score   support

           0       0.65      0.84      0.73        64
           1       0.81      0.59      0.68        71

    accuracy                           0.71       135
   macro avg       0.73      0.72      0.71       135
weighted avg       0.73      0.71      0.71       135
```



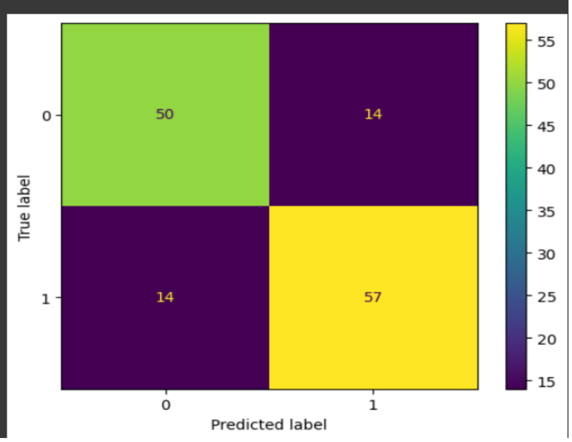


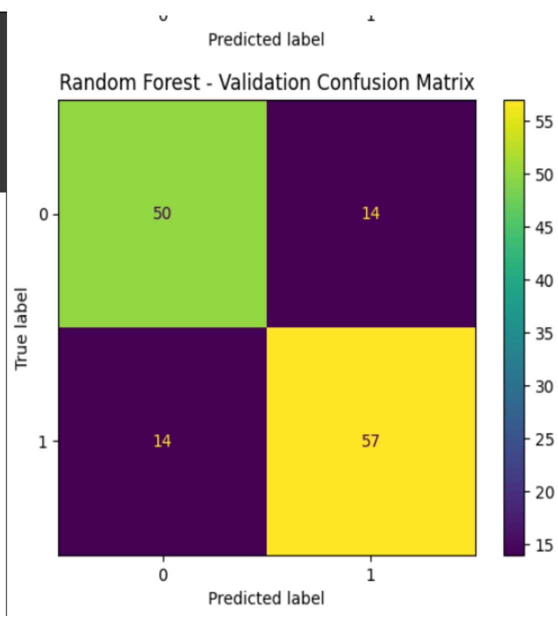
Activity 1.2: Random Forest Model

In this activity, we used the **Random Forest Classifier**, which is an ensemble learning method. It builds multiple decision trees and combines their predictions to get better performance and avoid overfitting.

```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ConfusionMatrixDisplay
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
import matplotlib.pyplot as plt

# Model dictionary
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "KNN": KNeighborsClassifier(),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
}

# Dictionary to store accuracy results
results = {}

# Loop through each model
for name, model in models.items():
    print(f"\n🔹 Training {name}...")

    # Train on SMOTE-balanced training data
    model.fit(X_train_final, y_train_final)

    # -------- Validation Performance --------
    val_preds = model.predict(X_val)
    print(f"\n📋 {name} - Validation Results:")
    print(classification_report(y_val, val_preds))
    ConfusionMatrixDisplay.from_predictions(y_val, val_preds).plot()
    plt.title(f"{name} - Validation Confusion Matrix")
    plt.show()
```

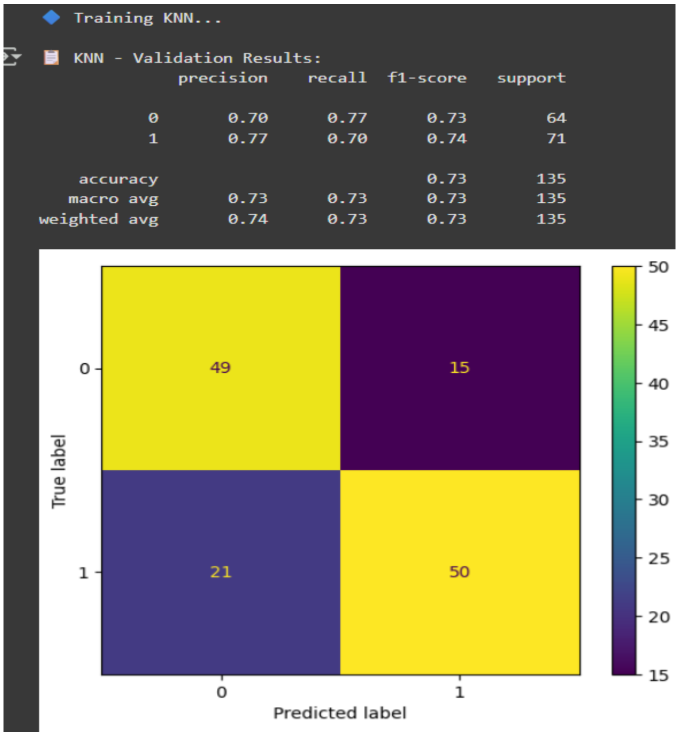
```
📋 Random Forest - Validation Results:
              precision    recall  f1-score   support

           0       0.78      0.78      0.78        64
           1       0.80      0.80      0.80        71

    accuracy                           0.79       135
   macro avg       0.79      0.79      0.79       135
weighted avg       0.79      0.79      0.79       135
```

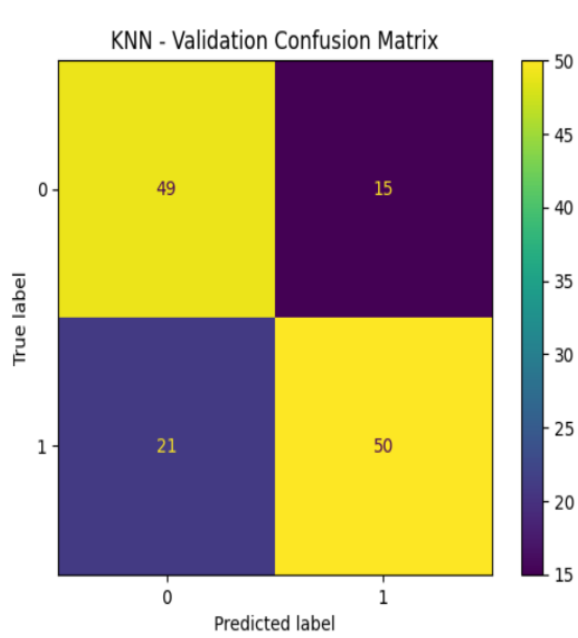




Activity 1.3: KNN Model

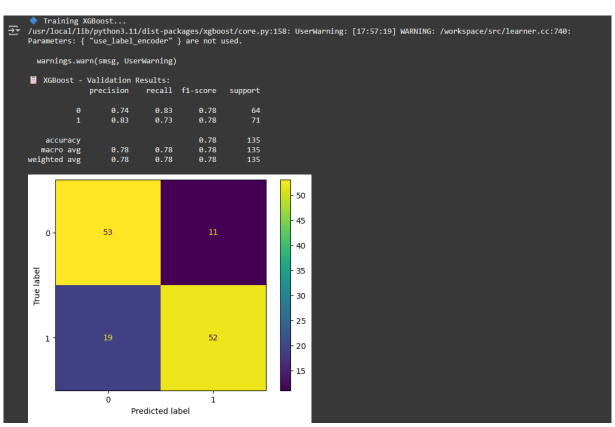
The third model we applied to our loan approval prediction task is **K-Nearest Neighbors (KNN)**. It's a simple yet powerful algorithm that predicts the class of a data point based on the **majority vote of its nearest neighbors**.

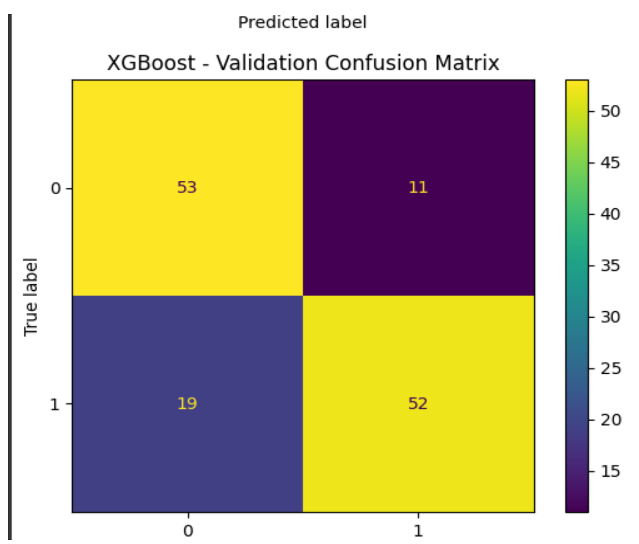




Activity 1.4: XGBoost Model

In this part, we trained our data using the **XGBoost (Extreme Gradient Boosting)** algorithm. XGBoost is a highly efficient and accurate classification model based on decision tree ensembles. It performs exceptionally well in many real-world machine learning competitions and is well-suited for tabular datasets like ours.





Activity 2: Testing the Model

After training the models, the next step is to **test them** using the **test dataset** that was separated earlier during the train-test split.

In our project, we tested the predictions using the `.predict()` function on **all the models** — including **Decision Tree, Random Forest, KNN, and XGBoost**.

```
        # -------- Validation Performance --------
        val_preds = model.predict(X_val)
        print(f"\n📋 {name} - Validation Results:")
        print(classification_report(y_val, val_preds))
        ConfusionMatrixDisplay.from_predictions(y_val, val_preds).plot()
        plt.title(f"{name} - Validation Confusion Matrix")
        plt.show()

        # -------- Test Performance --------
        test_preds = model.predict(X_test_scaled)
        print(f"\n📋 {name} - Test Results:")
        print(classification_report(y_test, test_preds))
        ConfusionMatrixDisplay.from_predictions(y_test, test_preds).plot()
        plt.title(f"{name} - Test Confusion Matrix")
        plt.show()
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing the Model with Multiple Evaluation Metrics

Evaluating a model using just one metric like accuracy might not give a complete picture. That's why we used **multiple evaluation metrics** to understand how well our classification models are performing from different perspectives.

In our project, we used:

- **Accuracy** – Overall correctness of the model
- **Precision** – How many predicted positives were actually positive
- **Recall** – How many actual positives were correctly predicted
- **F1-score** – Harmonic mean of precision and recall
- **Support** – Number of actual samples in each class

These metrics help assess whether the model is **balanced** and not biased toward the majority class (which is especially important in loan approval decisions).

Activity 1.1: Compare the Models

To make comparison easier, we defined a function `compareModel()` that prints all key metrics for each model:

```
    # -------- Validation Performance --------
    val_preds = model.predict(X_val)
    print(f"\n📋 {name} - Validation Results:")
    print(classification_report(y_val, val_preds))
    ConfusionMatrixDisplay.from_predictions(y_val, val_preds).plot()
    plt.title(f"{name} - Validation Confusion Matrix")
    plt.show()

    # -------- Test Performance --------
    test_preds = model.predict(X_test_scaled)
    print(f"\n📋 {name} - Test Results:")
    print(classification_report(y_test, test_preds))
    ConfusionMatrixDisplay.from_predictions(y_test, test_preds).plot()
    plt.title(f"{name} - Test Confusion Matrix")
    plt.show()

    # Accuracy on test set
    acc = accuracy_score(y_test, test_preds)
    results[name] = acc
    print(f"✅ {name} Test Accuracy: {acc:.4f}")

# -------- Compare All Accuracies --------
print("\n📊 Final Model Accuracies on Test Set:")
for name, score in results.items():
    print(f"{name}: {score:.4f}")

# Bar Chart
plt.bar(results.keys(), results.values(), color=['skyblue', 'orange', 'green', 'red'])
plt.title("Model Accuracy Comparison (Test Set)")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()
```

DECISION TREE - TEST RESULT

```
📋 Decision Tree - Test Results:
              precision    recall  f1-score   support

           0       0.56      0.71      0.63        38
           1       0.85      0.75      0.80        85

    accuracy                           0.74       123
   macro avg       0.71      0.73      0.71       123
weighted avg       0.76      0.74      0.75       123
```

RANDOM FOREST - TEST RESULT

📋 Random Forest - Test Results:

	precision	recall	f1-score	support
0	0.71	0.63	0.67	38
1	0.84	0.88	0.86	85
accuracy			0.80	123
macro avg	0.77	0.76	0.76	123
weighted avg	0.80	0.80	0.80	123

KNN - TEST RESULT

📋 KNN - Validation Results:

	precision	recall	f1-score	support
0	0.70	0.77	0.73	64
1	0.77	0.70	0.74	71
accuracy			0.73	135
macro avg	0.73	0.73	0.73	135
weighted avg	0.74	0.73	0.73	135

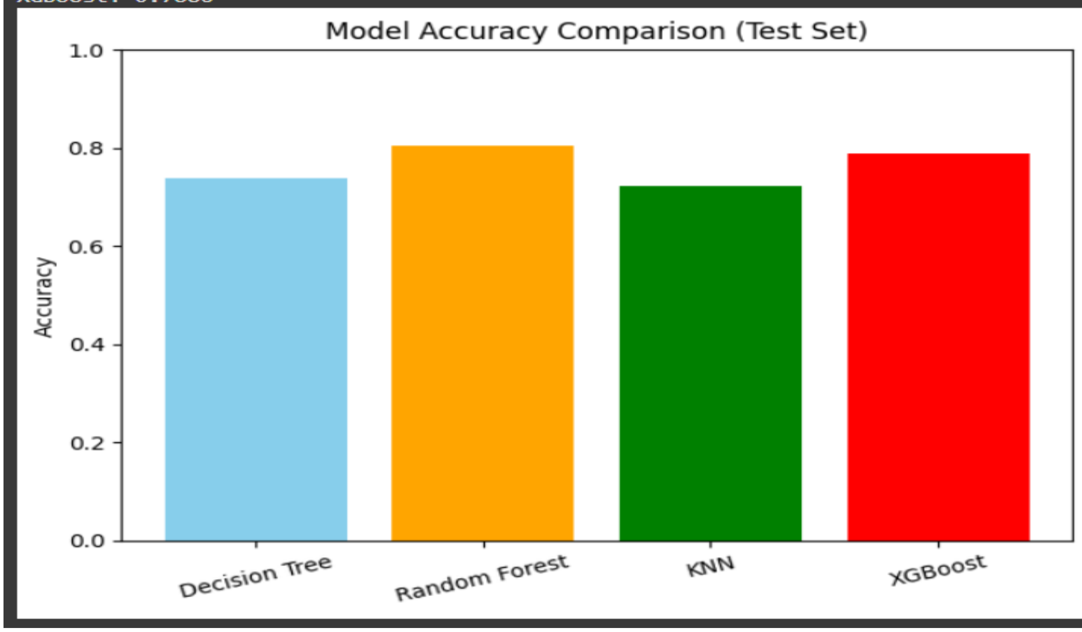
XGBOOST - TEST RESULT

📋 XGBoost - Test Results:

	precision	recall	f1-score	support
0	0.64	0.71	0.68	38
1	0.86	0.82	0.84	85
accuracy			0.79	123
macro avg	0.75	0.77	0.76	123
weighted avg	0.80	0.79	0.79	123

FINAL MODEL ACCURACIES OF ALL MODELS -

```
📊 Final Model Accuracies on Test Set:
Decision Tree: 0.7398
Random Forest: 0.8049
KNN: 0.7236
XGBoost: 0.7886
```



Activity 2: Comparing Model Accuracy Before & After Applying Hyperparameter Tuning

In this activity, we performed **hyperparameter tuning** to improve the performance of our **Random Forest model**. Although it was optional for the project, tuning helped us explore the best possible combination of parameters to boost accuracy and generalization.

Hyper parameter tuning

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

rf = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid,
                           scoring='accuracy', cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train_final, y_train_final)

print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation accuracy: {:.4f}".format(grid_search.best_score_))
```

OUTPUT -

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best parameters found:  {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}
Best cross-validation accuracy: 0.7809
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528: FitFailedWarning:
540 fits failed out of a total of 1080.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
540 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py", line 866, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1382, in wrapper
    estimator._validate_params()
  File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 436, in _validate_params
    validate_parameter_constraints(
  File "/usr/local/lib/python3.11/dist-packages/sklearn/utils/_param_validation.py", line 98, in validate_parameter_constraints
    raise InvalidParameterError(
sklearn.utils._param_validation.InvalidParameterError: The 'max_features' parameter of RandomForestClassifier must be an int in the range [1, inf), a float in the range (0.0, 1.0], a str among {'log2', '

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are non-finite: [        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.77538941 0.77720665 0.78092766
 0.77724126 0.77727587 0.77724126 0.76048806 0.76607823 0.7753721
 0.76607823 0.76420907 0.75867082 0.77165109 0.76239183 0.76424368
 0.76050537 0.76239183 0.76239183 0.76235722 0.75124611 0.75678435
 0.76235722 0.75124611 0.75678435 0.74567324 0.74941156 0.75683628
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan 0.76055729 0.77352025 0.77722395
 0.76796469 0.76235722 0.77163378 0.76240914 0.76607823 0.77535479
```

Milestone 6: Model Deployment

Activity 1: Save the Best Model

After training and evaluating multiple machine learning models — including **Decision Tree, Random Forest, KNN, and XGBoost** — we selected the **best-performing model** based on **accuracy, precision, recall, and F1-score**. In our case, the **Random Forest model with hyperparameter tuning** gave the most balanced and accurate results.

```python
import pickle
#Save model and scaler for Flask
with open('final_model.pkl', 'wb') as f:
    pickle.dump(voting_clf, f)

with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks

● Building HTML Pages

● Building server-side script
● Run the web application

Activity 2.1: Building Html Page:

For this project create HTML file namely
· home.html
. predict.html
. output.html

Activity 2.2: Build Python code:

Import the libraries

```
from flask import Flask, render_template, request
import numpy as np
import pandas as pd
import pickle
import os
```

Load the saved model.An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (name) as argument.

```
app = Flask(__name__)

model = pickle.load(open('final_model.pkl', 'rb'))
scaler = pickle.load(open('scaler.pkl', 'rb'))
```

Render HTML page:

```
@app.route('/')
def home():
    return render_template('home.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.
In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered.
Whenever you enter the values from the html page the values can be retrieved using POST Method.
Retrieves the value from UI:

```
@app.route('/predict')
def predict():
    return render_template('predict.html')

@app.route('/submit', methods=['POST', 'GET'])
def submit():
    if request.method == 'POST':
        # Extract form data
        form = request.form

        # Construct the input dict manually
        input_dict = {
            'Dependents': int(form.get("dependents")),
            'ApplicantIncome': float(form.get("applicant_income")),
            'CoapplicantIncome': float(form.get("coapplicant_income")),
            'LoanAmount': float(form.get("loan_amount")),
            'Loan_Amount_Term': float(form.get("loan_term")),
            'Credit_History': float(form.get("credit_history")),
            'Gender_Male': 1 if form.get("gender") == 'Male' else 0,
            'Married_Yes': 1 if form.get("married") == 'Yes' else 0,
            'Education_Not Graduate': 1 if form.get("education") == 'Not Graduate' else 0,
            'Self_Employed_Yes': 1 if form.get("self_employed") == 'Yes' else 0,
            'Property_Area_Semiurban': 1 if form.get("property_area") == 'Semiurban' else 0,
            'Property_Area_Urban': 1 if form.get("property_area") == 'Urban' else 0
        }
        input_df = pd.DataFrame([input_dict])

        # Ensure correct column order
        expected_columns = [
            'Dependents', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term',
            'Credit_History', 'Gender_Male', 'Married_Yes', 'Education_Not Graduate',
            'Self_Employed_Yes', 'Property_Area_Semiurban', 'Property_Area_Urban'
        ]
        input_df = input_df[expected_columns]

        # Scale the input
        input_scaled = scaler.transform(input_df)

        # Make prediction
        prediction = model.predict(input_scaled)[0]
        print("User input dict:\n", input_dict)
        print("DataFrame before scaling:\n", input_df)
        print("Scaled input:\n", input_scaled)
        print("Prediction result:", prediction)
        # Show result
        result = "✅ Loan will be Approved" if prediction == 1 else "❌ Loan will Not be Approved"
        return render_template('output.html', result=result)

    return render_template('home.html')
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

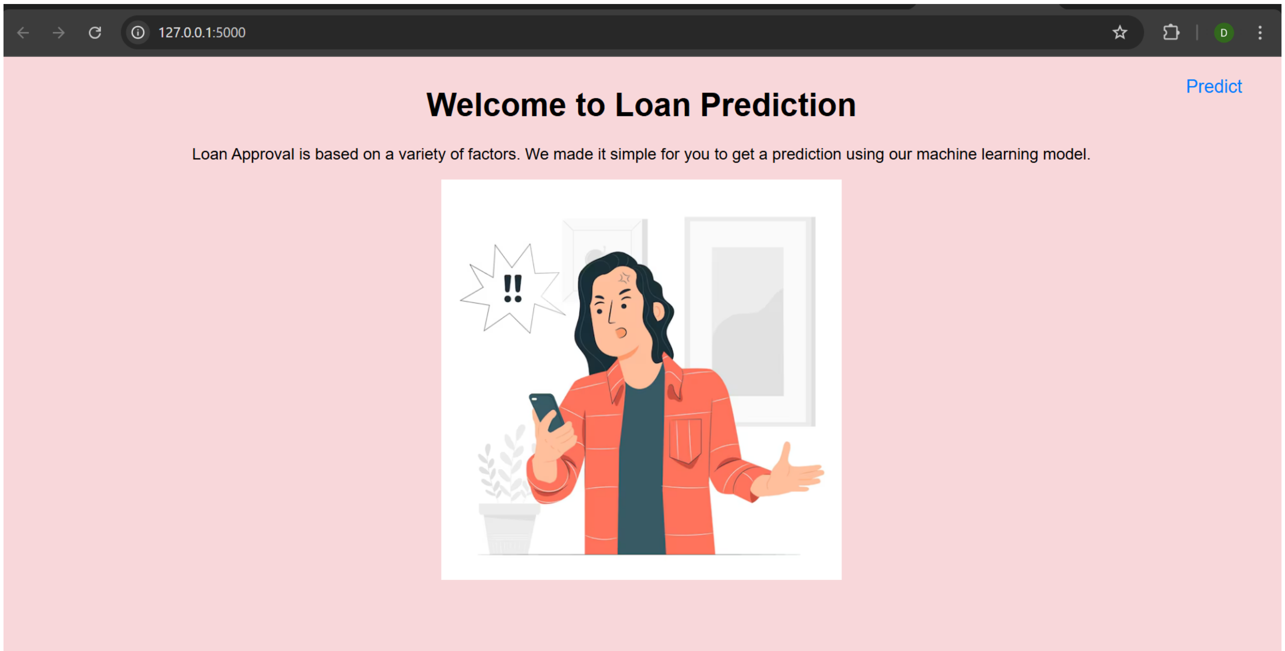
```
if __name__ == "__main__":
    port = int(os.environ.get('PORT', 5000))
    app.run(debug=True, port=port)
```

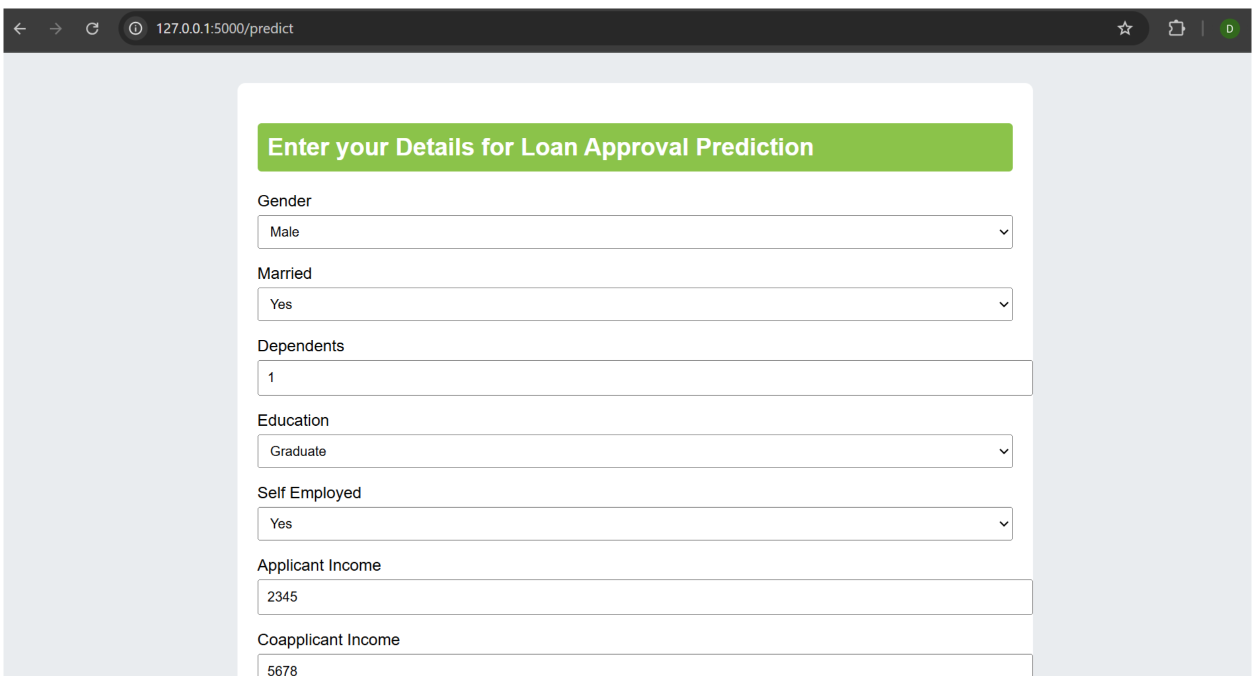
Activity 2.3: Run the web application

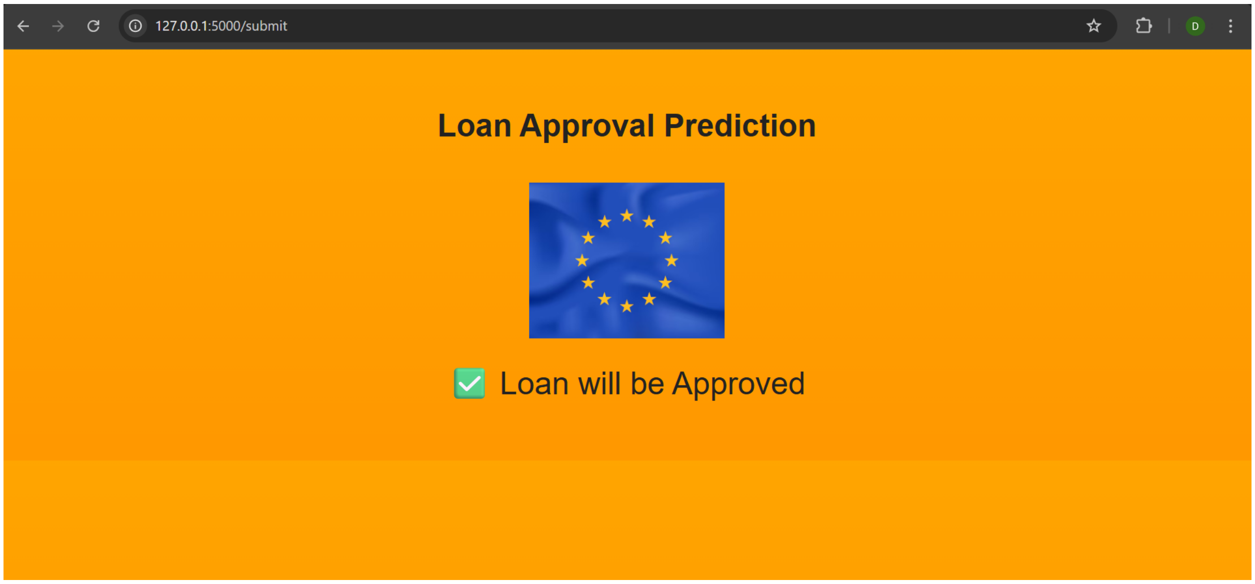
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
  warnings.warn(
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
```

Now,Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result


127.0.0.1:5000
Predict
Welcome to Loan Prediction
Loan Approval is based on a variety of factors. We made it simple for you to get a prediction using our machine learning model.
!!


127.0.0.1:5000/predict
Enter your Details for Loan Approval Prediction
Gender
Male
Married
Yes
Dependents
1
Education
Graduate
Self Employed
Yes
Applicant Income
2345
Coapplicant Income
5678



Milestone 7: Advantages and Disadvantages

Advantages

- Clear modular design: Easy to manage and update individual components (UI, model, preprocessing).
- Automated data preprocessing: Ensures clean and consistent data before model training.
- Reusable components: UI, model, and algorithm are separated, allowing flexible improvements.
- User-friendly interface: Simplifies data input and result viewing for applicants.
- Integrated model evaluation step: Ensures performance is validated before deployment.
- Supports iterative training: Loop between training and testing allows continuous optimization.

Disadvantages

- Not real-time adaptive: Model doesn't update automatically with new data unless retrained manually.
- Possible tight coupling between UI and model may make independent changes harder.
- No feedback loop: Doesn't learn from actual loan approval outcomes to improve predictions.

- CSV-based data input: Indicates offline or static data source, limiting real-time scalability.
- Scalability issues: Architecture may not handle large datasets or multiple concurrent users efficiently.
- Data security not addressed: No mention of encryption, privacy, or compliance with data protection laws.

Milestone 7: Conclusion

The Smart Lender – Applicant Credibility Prediction System is designed to bring transparency and efficiency to the loan approval process. By leveraging machine learning, the system evaluates key applicant attributes such as gender, marital status, education, income, loan amount, credit history, and property area to predict the likelihood of loan approval.

Among various models tested, XGBoost emerged as the most effective due to its high accuracy, ability to handle complex patterns, and built-in regularization. The system offers a user-friendly interface that provides applicants with a preliminary assessment, helping them make informed decisions before applying for a loan.

While the system does not replace formal financial evaluations, it serves as a valuable decision-support tool. Future improvements may include real-time data integration, enhanced interpretability, and continuous model updates to adapt to changing financial trends.

APPENDIX

Github link- https://github.com/Kash0205/Loan_Prediction_Project_SmartLender

Demo Video link-
https://drive.google.com/file/d/1fMS0r9wPWcGhA0lfe6Mvbz6TTmRCOIlY/view?usp=sharing