Presented By:
1. Prachi Verma
   2023UCS1553
2. Debjanee Murdia
   2023UCS1559
3. Rudrakshi Bansal
   2023UCS1566
4. Dapinder Kaur
   2023UCS1579

Loan Status Prediction Model

Implementation Using Python and Support Vector Machines (SVM)

April, 2025

Problem

Banks receive many loan applications and must decide which applicants are eligible. Manual verification is time-consuming and can be inconsistent.

Goal

To build a machine learning model that predicts whether a loan should be approved based on applicant data using an SVM classifier.

Tech Stack

- **Language:** Python
- **Libraries:** Pandas, NumPy, Seaborn, Matplotlib, scikit-learn
- **Model:** Support Vector Machine (SVM)

Dataset

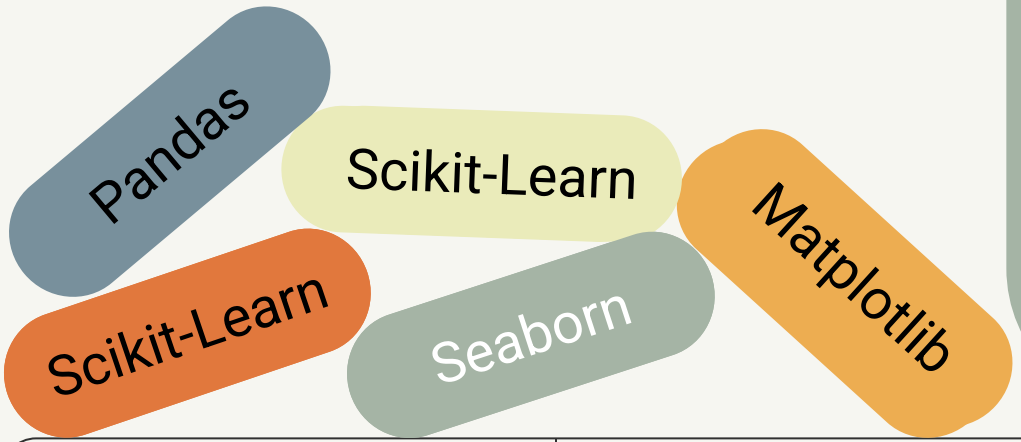
- **Source:** Loan dataset from Universal Bank (Kaggle/other).
- **Total Records:** 614 entries, 13 columns.
- **Target Variable:** `Loan_Status` (Y/N)
- **Feature Types:**
  - Categorical: Gender, Married, Education, etc.
  - Numerical: ApplicantIncome, CoapplicantIncome, LoanAmount, etc.

```
# number of missing values in each column
loan_dataset.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

Importing Libraries




```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report
```

pandas, **numpy** – data manipulation.

seaborn, **matplotlib** – visualization.

sklearn.model_selection – train-test split.

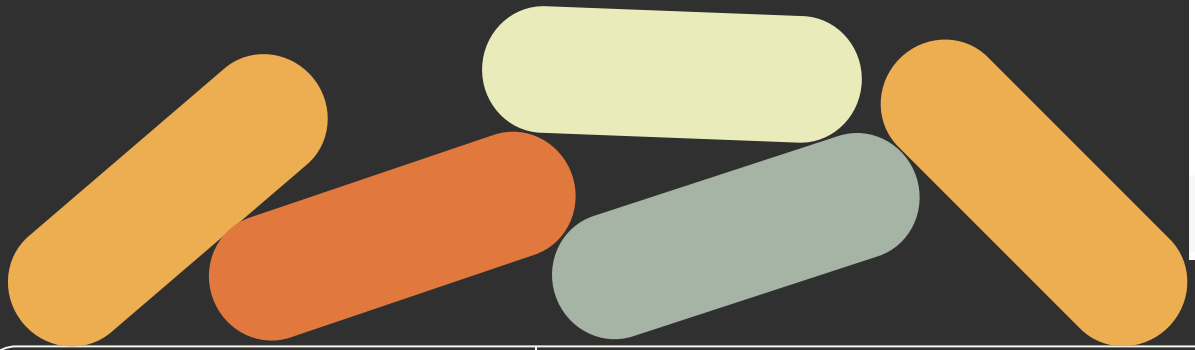
sklearn.svm – for SVM model.

sklearn.metrics – for evaluation.

Data Cleaning

Checked for missing values in the using `loan_dataset.isnull().sum()`

Dropped rows with missing values to ensure model accuracy and consistency



```
# number of missing values in each column
loan_dataset.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

```
[ ]  # dropping the missing values
     loan_dataset = loan_dataset.dropna()
```

Label Encoding

Converted categorical values into numerical ones using .replace.

For Example:

- 'Male' → 1, 'Female' → 0
- 'Graduate' → 1, 'Not Graduate' → 0
- 'Loan_Status': 'Y' → 1, 'N' → 0

```
# label encoding
loan_dataset.replace({"Loan_Status":{'N':0,'Y':1}},
                     inplace=True)
```

```
# convert categorical columns to numerical values
loan_dataset.replace({'Married':{'No':0,'Yes':1},
                      'Gender':{'Male':1,'Female':0},
                      'Self_Employed':{'No':0,'Yes':1},
                      'Property_Area':{'Rural':0,'Semiurban':1,'Urban':2},
                      'Education':{'Graduate':1,'Not Graduate':0}},
                     inplace=True)
```

EDA

Exploratory Data Analysis & Visualisation

Understanding the Data:

Before training the model, it's important to explore patterns and relationships within the data.

Tools Used:

`seaborn.countplot()`: for categorical comparisons

`matplotlib.pyplot`: for plotting and customization

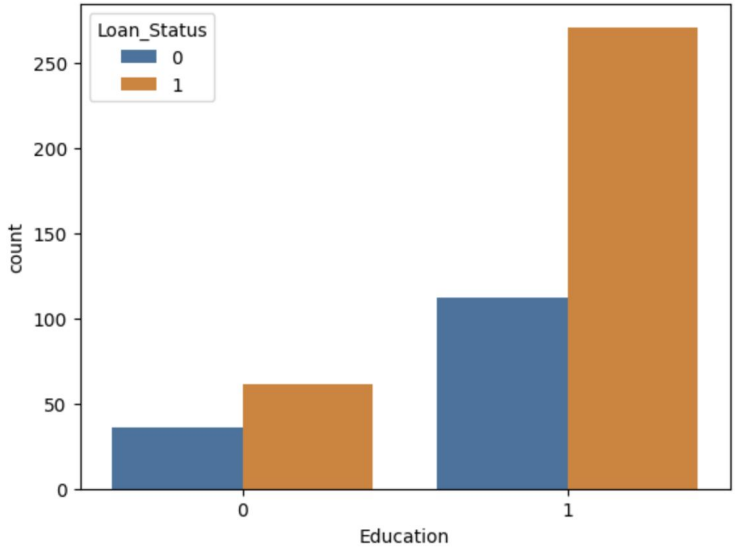
Understanding the Data:

Count plots were used to visually analyze how features like education and marital status relate to loan approval.

Helped identify which features might be important predictors.

```
# education & Loan Status
sns.countplot(x='Education',hue='Loan_Status',data=loan_dataset)
```

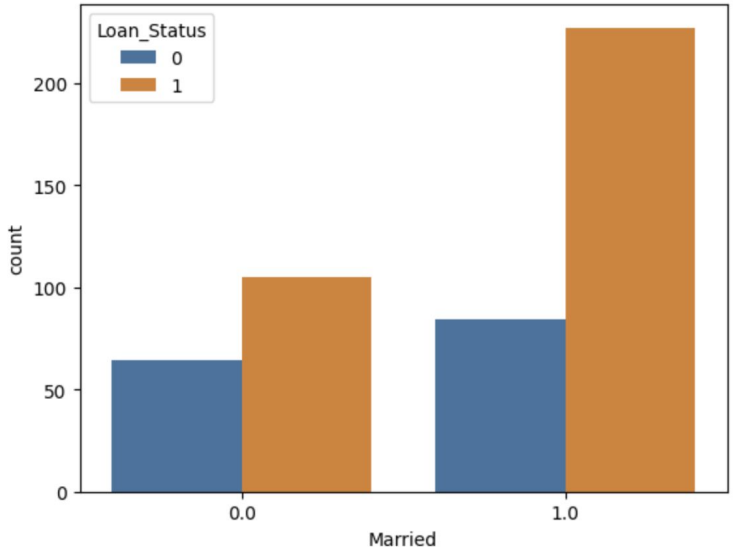
```
<Axes: xlabel='Education', ylabel='count'>
```



A bar chart shows that graduates have a slightly higher chance of loan approval compared to non-graduates.

```
# marital status & Loan Status
sns.countplot(x='Married',hue='Loan_Status',data=loan_dataset)
```

```
<Axes: xlabel='Married', ylabel='count'>
```



Married applicants tend to receive more approvals than unmarried ones.

Feature Selection and Data Splitting

```
# separating the data and label
X = loan_dataset.drop(columns=['Loan_ID','Loan_Status'],axis=1)
Y = loan_dataset['Loan_Status']
```

Removed unnecessary columns like 'Loan_ID'.

```
X_train = X_train.dropna()
Y_train = Y_train[X_train.index]
X_train, X_test,Y_train,Y_test = train_test_split(X,Y,
          test_size=0.1,stratify=Y,random_state=2)
```

Splitting done with stratified sampling to preserve class balance.

Model Evaluation

Model Performance:

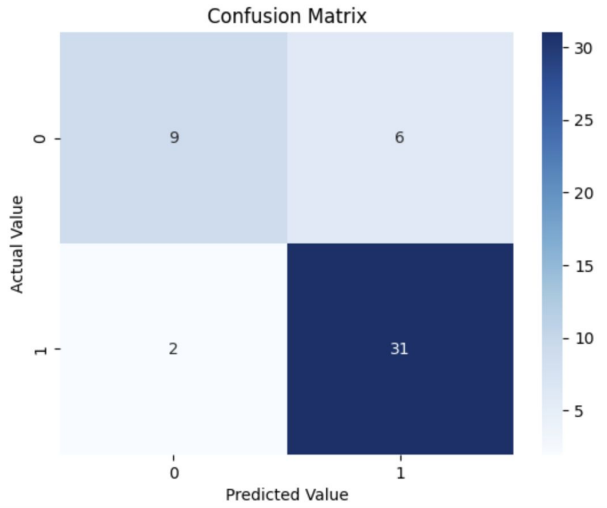
- **Training Accuracy:** ~80%
- **Testing Accuracy:** ~83%

```
Y_pred = model.predict(X_test)
print("Classification Report:\n", classification_report(Y_test, Y_pred))
sns.heatmap(pd.crosstab(Y_test, Y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Value')
plt.ylabel('Actual Value')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.60      0.69        15
           1       0.84      0.94      0.89        33

    accuracy                           0.83        48
   macro avg       0.83      0.77      0.79        48
weighted avg       0.83      0.83      0.83        48
```



Model Training

```
[ ]  model = svm.SVC(kernel='linear')

[ ]  #training the support Vector Machine model
     model.fit(X_train,Y_train)
```

SVC

SVC(kernel='linear')

Why SVM?
SVM is effective in high-dimensional spaces and works well for classification tasks like ours.

Training the Model:

- Used a linear kernel for simplicity and speed
- Trained on cleaned and encoded data
-

Key Parameters:

- `kernel='linear'`
- Default regularization (C) and gamma values

Predictive System

Step 1

Takes a list of inputs (mimicking user form data).

Step 2

Reshapes and encodes it like the training data

Step 3

Passes input to trained model to predict output

```python
def predict_loan_approval(input_data):
    # Convert input data to a numpy array
    input_data_array = np.asarray(input_data)
    # Reshape data for a single prediction
    input_data_reshaped = input_data_array.reshape(1, -1)
    # Predict using the trained model
    prediction = model.predict(input_data_reshaped)
    if prediction[0] == 1:
        return 'Loan Approved ✅'
    else:
        return 'Loan Not Approved ❌'

# Step 9: Test the Predictive System (eg: Gender=1, Married=1,
#Dependents=0, Education=1, Self_Employed=0, ApplicantIncome=5000,
#CoapplicantIncome=0.0, LoanAmount=150.0, Loan_Amount_Term=360.0,
#Credit_History=1.0, Property_Area=2)
test_input = [1, 1, 0, 1, 0, 5000, 0.0, 150.0, 360.0, 1.0, 2]

result = predict_loan_approval(test_input)
print("Prediction Result:", result)
```

```
Prediction Result: Loan Approved ✅
```

Conclusion

Successfully built an SVM-based model for loan status prediction

Achieved strong performance with ~83% test accuracy

Developed a simple predictive system for real-time eligibility checks

Developed a **user-friendly predictive system** that accepts user inputs and returns instant loan status.

Integrated the model with a **frontend website** using HTML and CSS, making it accessible for end-users.