



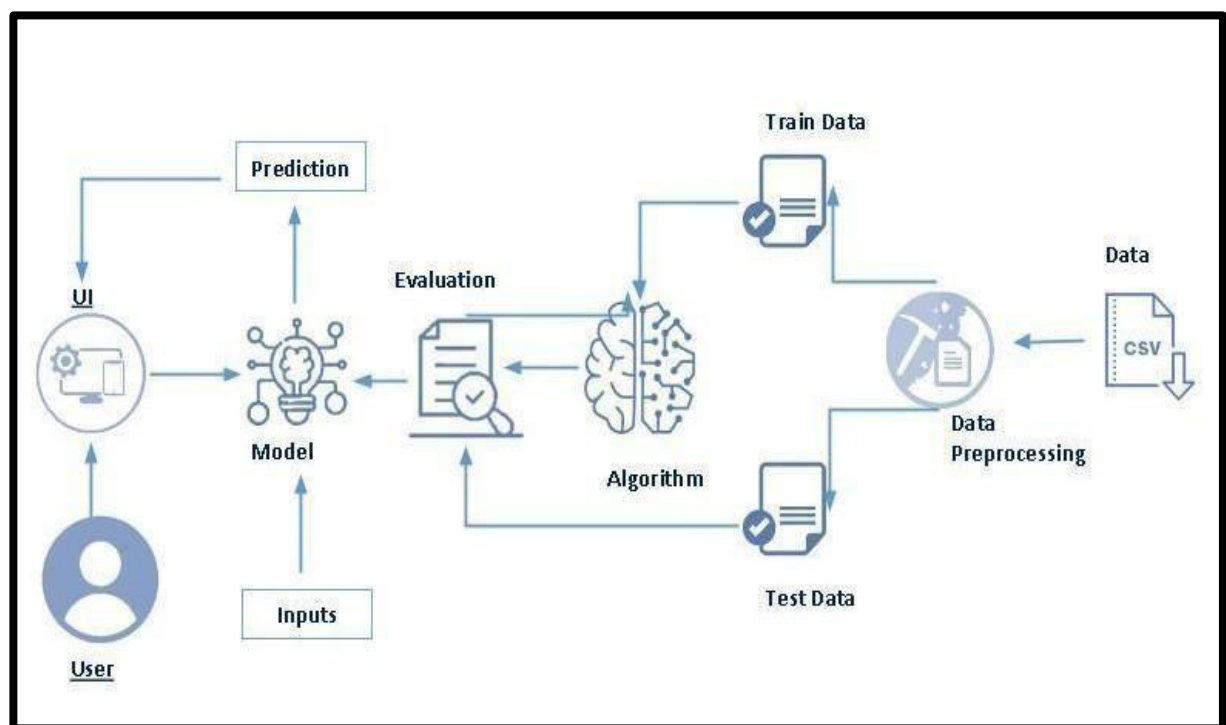
# SmartLender – Applicant Credibility Prediction for Loan Approval

Team ID- SWTID1750050475    Team Member-Yash Chugani, Yash Dharad,  
Aashish Kumar Mandhyani

## Applicant Credibility Prediction for loan approval using ML

In today's fast-paced financial ecosystem, ensuring efficient and accurate loan approval processes is essential for both lenders and applicants. However, traditional manual methods are often time-consuming, prone to errors, and may overlook key indicators of creditworthiness. A **Loan Approval Prediction System** leverages the power of machine learning to automate and optimize the loan evaluation process. By analyzing various applicant parameters such as income, employment status, credit history, and loan amount, the system can accurately predict whether a loan application should be approved or denied. This not only reduces the workload on financial institutions but also minimizes risks, enhances customer experience, and ensures fair decision-making. The goal of this project is to develop a predictive model that can classify loan applications as approved or rejected, ultimately supporting smarter and faster financial decisions.

### Technical Architecture:



## **Project Flow:**

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

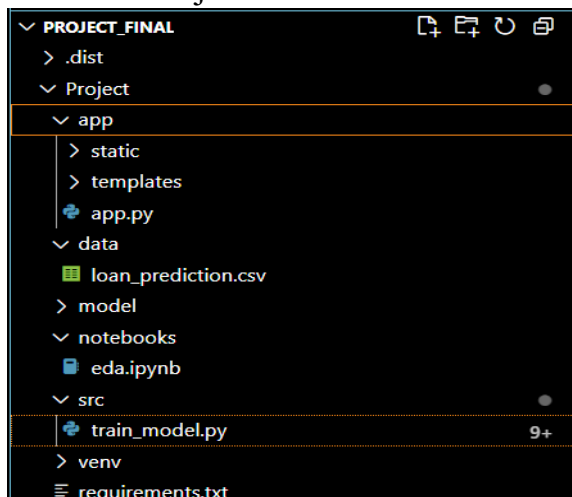
## **Prior Knowledge:**

You must have prior knowledge of following topics to complete this project.

- ML Concepts
  - o Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
  - o Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>
- Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
- Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
- KNN: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
- Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
- Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
- Flask Basics : [https://www.youtube.com/watch?v=lj4I\\_CvBnt0](https://www.youtube.com/watch?v=lj4I_CvBnt0)

## **Project Structure:**

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- final\_model.pkl is our saved model. Further we will use this model for flask integration.
- Data Folder contains the Dataset used.
- The Notebook file contains Exploratory Data Analysis of the dataset.

## **Milestone 1: Define Problem / Problem Understanding**

### **Activity 1: Specify the business problem**

In the financial sector, timely and accurate loan approval is crucial for effective lending and customer satisfaction. However, manual review of loan applications is often inefficient and prone to errors or bias. With the increasing volume of applications, it becomes difficult for banks and financial institutions to assess every applicant thoroughly. **SmartLender** aims to automate this process using machine learning, predicting whether a loan application should be approved or rejected based on various financial and personal parameters of the applicant. This reduces operational load, minimizes default risks, and improves turnaround time for applicants.

### **Activity 2: Business requirements**

A loan approval prediction project must meet certain business requirements to ensure practical applicability:

- **Accuracy & Reliability:** The prediction model must be highly accurate in classifying applications to minimize financial risk.
- **Scalability:** The system should be capable of handling a large volume of applications simultaneously.
- **Compliance:** It must comply with financial regulations, data protection standards, and fair lending practices.
- **Transparency:** The decisions made by the model should be explainable, especially in cases of rejections.
- **User Interface:** A clean, intuitive UI for both lenders and borrowers to interact with the system effectively.

### **Activity 3: Literature Survey (Student Will Write)**

A literature review on loan approval prediction reveals the widespread use of classification algorithms like Logistic Regression, Decision Trees, Random Forest, and XGBoost. Various datasets have been utilized, mostly from banks or open sources such as Kaggle. Research highlights include:

- Feature importance of credit history, income, loan amount, and employment status.
- The significance of balancing datasets for fair classification.
- The use of techniques like SMOTE for handling class imbalance and scaling for model accuracy.

Previous studies also emphasize the need for interpretable ML models to support trust in financial decisions.

## Activity 4: Social or Business Impact.

### Social Impact:

- **Financial Inclusion:** Helps underserved individuals get fair assessments and access to credit.
- **Bias Reduction:** Removes human bias from the loan approval process.

### Business Impact:

- **Improved Efficiency:** Automates application processing, saving time and cost.
- **Risk Management:** Accurately predicts risky applicants, reducing default rates.
- **Data-Driven Decisions:** Enhances strategic decision-making for financial institutions.

## **Milestone 2: Data Collection & Preparation**

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

### **Activity 1: Collect the dataset**

The dataset used in this project was obtained from [Kaggle](https://www.kaggle.com/datasets/vikasukani/loan-eligible-dataset?select=loan-train.csv) under the title **Loan Prediction Dataset**. It is in .csv format and contains historical loan application records with both numerical and categorical features.

Link: <https://www.kaggle.com/datasets/vikasukani/loan-eligible-dataset?select=loan-train.csv>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

**Note:** There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

#### **Activity 1.1: Importing the libraries**

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

#### **Activity 1.2: Read the Dataset**

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called `read_csv()` to read the dataset. As a parameter we have to give the directory of the csv file.

- For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function.

```
data=pd.read_csv("../data/loan_prediction.csv")
data
```

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0
...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0

614 rows x 13 columns

## Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

### Activity 2.1: Handling missing values

- For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function.

```
data.isnull().sum()
```

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	



```
# handling missing data values
df['Gender']=df['Gender'].fillna(df['Gender'].mode()[0])
df['Married']=df['Married'].fillna(df['Married'].mode()[0])
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'].mode()[0])
df['Credit_History']=df['Credit_History'].fillna(df['Credit_History'].mode()[0])
df['Dependents']=df['Dependents'].str.replace('+', ' ', regex=False)
df['Dependents']=df['Dependents'].fillna(df['Dependents'].mode()[0])
```

## Activity 2.2: Handling Outliers

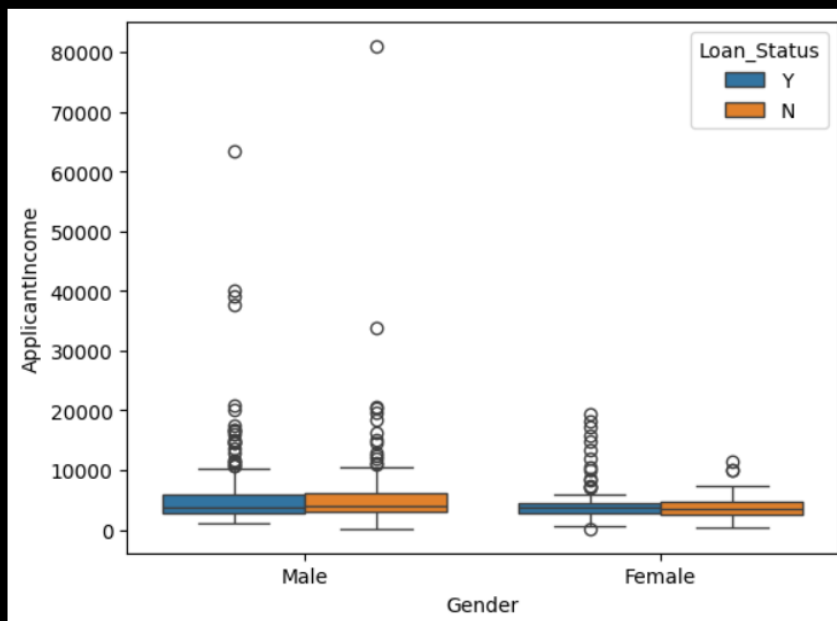
With the help of boxplot, outliers are visualized.

```
sns.boxplot(x='Gender', y='ApplicantIncome', hue='Loan_Status', data=data)
```

[24]

<Axes: xlabel='Gender', ylabel='ApplicantIncome'>

...



## Milestone 3: Exploratory Data Analysis

### Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
data.describe(include='all')
```

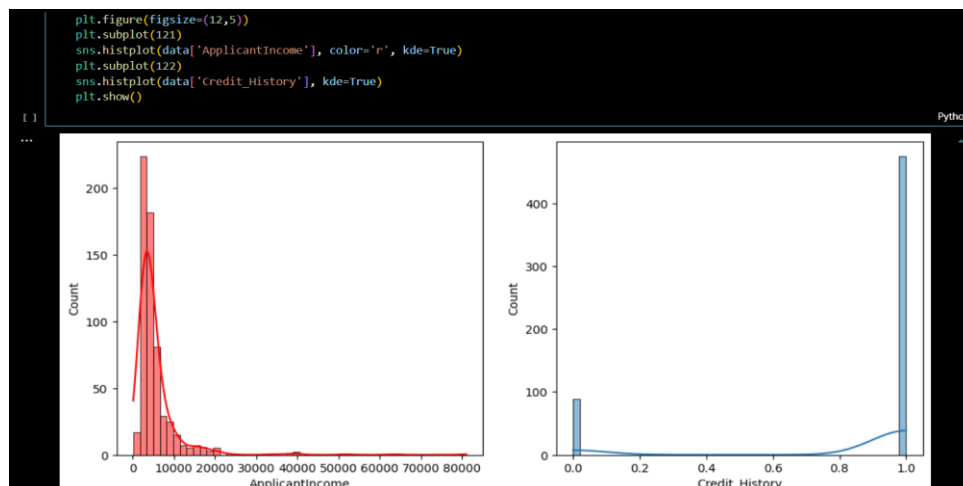
	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
count	614	601	611	599	614	582	614.000000	614.000000	592.000000	600.000000
unique	614	2	2	4	2	2	NaN	NaN	NaN	NaN
top	LP001002	Male	Yes	0	Graduate	No	NaN	NaN	NaN	NaN
freq	1	489	398	345	480	500	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	NaN	NaN	NaN	5403.459283	1621.245798	146.412162	342.000000
std	NaN	NaN	NaN	NaN	NaN	NaN	6109.041673	2926.248369	85.587325	65.12041
min	NaN	NaN	NaN	NaN	NaN	NaN	150.000000	0.000000	9.000000	12.000000
25%	NaN	NaN	NaN	NaN	NaN	NaN	2877.500000	0.000000	100.000000	360.000000
50%	NaN	NaN	NaN	NaN	NaN	NaN	3812.500000	1188.500000	128.000000	360.000000
75%	NaN	NaN	NaN	NaN	NaN	NaN	5795.000000	2297.250000	168.000000	360.000000
max	NaN	NaN	NaN	NaN	NaN	NaN	81000.000000	41667.000000	700.000000	480.000000

### Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

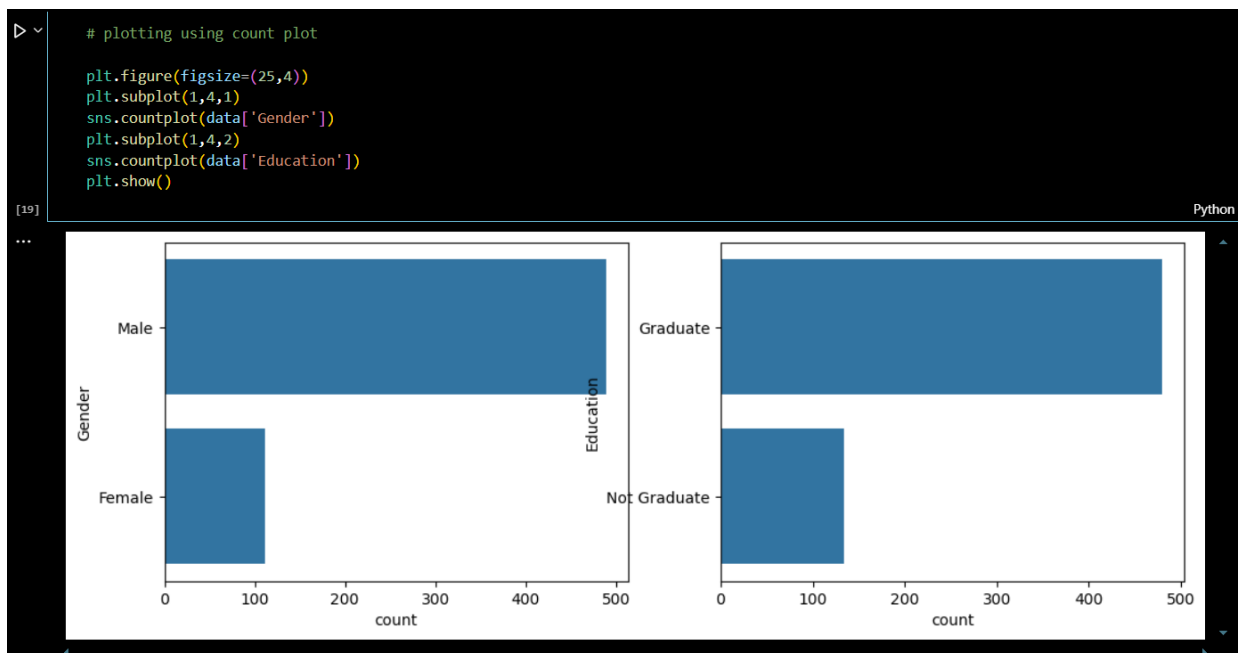
#### Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed a histogram plot

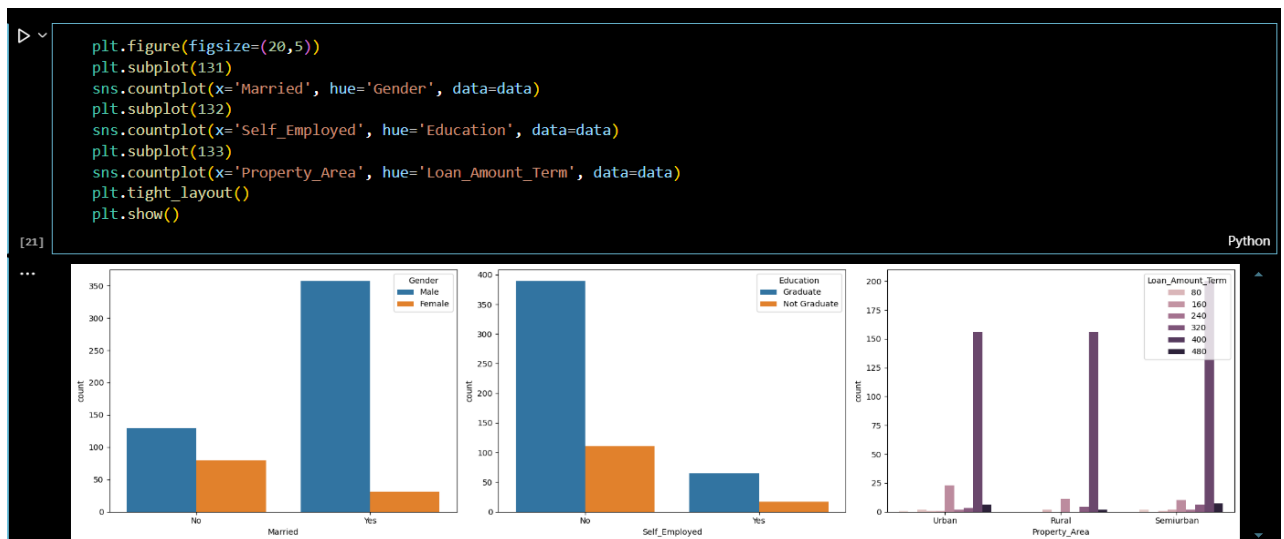


## Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we can use barplot.

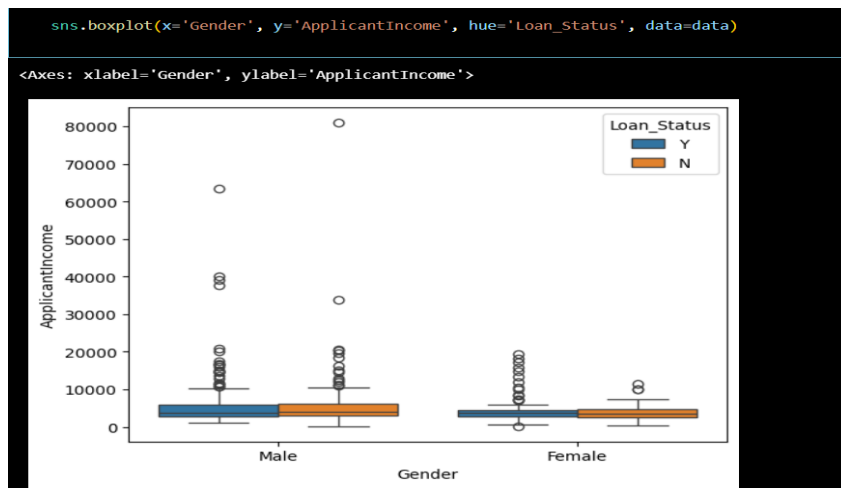
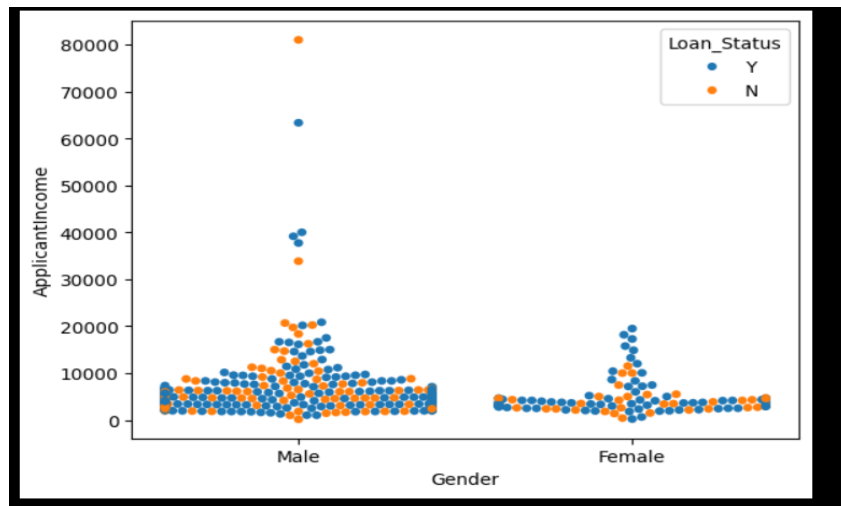


From the below barplot we can able to figure that Females are less married, less self-employed than Males across the different states.



## Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used swarmplot, violinplot, boxplot from seaborn package.



## Encoding the Categorical Features:

- The categorical Features are can't be passed directly to the Machine Learning Model. So we convert them into Numerical data based on their order. This Technique is called Encoding.
- Here we are importing Label Encoder from the Sklearn Library.
- Here we are applying fit\_transform to transform the categorical features to numerical features.

```
# handling categorical values
binary_cols = ['Gender', 'Married', 'Self_Employed']
le = LabelEncoder()
# Label encoding binary features
for col in binary_cols:
    df[col] = le.fit_transform(df[col])

# OneHot encoding multi-class features
df = pd.get_dummies(df, columns=['Education', 'Property_Area'], drop_first=True)
# Encoding target variable
df['Loan_Status'] = le.fit_transform(df['Loan_Status'])
```



## Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train\_test\_split() function from sklearn. As parameters, we are passing x, y, test\_size, random\_state.

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

## Handling Imbalanced dataset

- Imbalanced data is a common problem in machine learning and data analysis, where the number of observations in one class is significantly higher or lower than the other class. Handling imbalanced data is important to ensure that the model is not biased towards the majority class and can accurately predict the minority class.
- Here we are using SMOTE Technique.

```
# Balancing dataset using SMOTETomek
smk=SMOTETomek(random_state=42)
X_resampled, y_resampled = smk.fit_resample(X_scaled,y)
X_resampled = pd.DataFrame(X_resampled, columns=X.columns)
y_resampled = pd.Series(y_resampled, name='Loan_Status')
```

## Scaling

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.
- This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by:

$$X\_scaled = (X - X\_mean) / X\_std$$

```
# Feature Scaling the data
scaler=StandardScaler()
X_scaled= scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)
```

## **Milestone 4: Model Building**

### **Activity 1: Training the model in multiple algorithms**

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

#### **Activity 1.1: Decision tree model**

First Decision Tree is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X\_train and X\_test.

#### **Activity 1.2: Random forest model**

First Random Forest Model is imported from sklearn Library then RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X\_train and X\_test.

#### **Activity 1.3: KNN model**

KNN Model is imported from sklearn Library then KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

#### **Activity 1.4: XGBoost model**

XGBoost Classifier model is imported from the xgboost library. Then, the XGBoost algorithm is initialized using the XGBClassifier() function. The training data is passed to the model using the .fit() method, allowing the model to learn patterns from the input features and corresponding labels. After training, the model is used to predict outcomes on the test dataset with the .predict() function, and the results are stored in a new variable. To evaluate the performance of the model, a confusion matrix is created, which provides insight into the number of correct and incorrect predictions made by the classifier. This helps in assessing the accuracy and reliability of the XGBoost model.



```

#-----
# MODEL TRAINING AND EVALUATION
# -----
def train_models(X_train, X_test, y_train, y_test):
    models = {
        'Decision Tree': DecisionTreeClassifier(),
        'Random Forest': RandomForestClassifier(),
        'KNN': KNeighborsClassifier(),
        'Gradient Boosting': GradientBoostingClassifier()
    }

    results = {}

    for name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)

        # 5-Fold Cross Validation on training data
        cv_scores = cross_val_score(model, X_train, y_train, cv=5)
        cv_mean = cv_scores.mean()
        cv_std = cv_scores.std()

        results[name] = {
            'model': model,
            'accuracy': acc,
            'f1_score': f1,
            'cv_mean': cv_mean,
            'cv_std': cv_std
        }

```

```

print(f"Model: {name}")
print(f"Test Accuracy: {acc:.4f}")
print(f"F1 Score: {f1:.4f}")
print(f"Cross-Validation Mean Accuracy: {cv_mean:.4f}")
print(f"Cross-Validation Std Dev: {cv_std:.4f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("-" * 60)

model_selection(results)

```

## Activity 2: Testing the model

Test with all the algorithms using the predict() function

## Milestone 5: Performance Testing & Hyperparameter Tuning

### Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

#### Activity 1.1: Compare the model

```
Model: Decision Tree
Best Params: {'splitter': 'random', 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': None, 'criterion': 'entropy'}
Test Accuracy: 0.7708
F1 Score: 0.7718
Cross-Validation Mean Accuracy: 0.7518
Cross-Validation Std Dev: 0.0249

Confusion Matrix:
[[92 37]
 [18 93]]

Classification Report:
      precision    recall  f1-score   support

     0       0.84      0.71      0.77       129
     1       0.72      0.84      0.77       111

 accuracy          0.77       240
 macro avg          0.78       240
 weighted avg       0.78       240
```

```
Training Random Forest...
Fitting 5 folds for each of 20 candidates, totalling 100 fits

Model: Random Forest
Best Params: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_depth': None, 'criterion': 'entropy'}
Test Accuracy: 0.8292
F1 Score: 0.8367
Cross-Validation Mean Accuracy: 0.8107
Cross-Validation Std Dev: 0.0442

Confusion Matrix:
[[ 94  35]
 [  6 105]]

Classification Report:
      precision    recall  f1-score   support

     0       0.94      0.73      0.82       129
     1       0.75      0.95      0.84       111

 accuracy          0.83       240
 macro avg          0.84       240
 weighted avg       0.85       240
```

```

Model: KNN
Best Params: {'weights': 'distance', 'p': 1, 'n_neighbors': 5}
Test Accuracy: 0.7958
F1 Score: 0.7860
Cross-Validation Mean Accuracy: 0.8286
Cross-Validation Std Dev: 0.0249

```

Confusion Matrix:

```

[[101  28]
 [ 21  90]]

```

Classification Report:

	precision	recall	f1-score	support
0	0.83	0.78	0.80	129
1	0.76	0.81	0.79	111
accuracy			0.80	240
macro avg	0.80	0.80	0.80	240
weighted avg	0.80	0.80	0.80	240

```

Model: Gradient Boosting
Best Params: {'subsample': 1.0, 'n_estimators': 50, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_depth': 4, 'learning_rate': 0.2}
Test Accuracy: 0.8083
F1 Score: 0.8099
Cross-Validation Mean Accuracy: 0.8000
Cross-Validation Std Dev: 0.0360

```

Confusion Matrix:

```

[[96 33]
 [13 98]]

```

Classification Report:

	precision	recall	f1-score	support
0	0.88	0.74	0.81	129
1	0.75	0.88	0.81	111
accuracy			0.81	240
accuracy			0.81	240
macro avg	0.81	0.81	0.81	240
macro avg	0.81	0.81	0.81	240
weighted avg	0.82	0.81	0.81	240

From the above models, Random Forest is performing the best

## Activity 2: Comparing model accuracy before & after applying hyperparameter tuning (Hyperparameter tuning is optional. For this project it is not required.)

Evaluating performance of the model From sklearn, cross\_val\_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well.

```

def train_models(X_train, X_test, y_train, y_test, scaler):
    # Hyper-parameter Tuning the models
    models = {
        'Decision Tree': {
            'model': DecisionTreeClassifier(),
            'params': {
                'criterion': ['gini', 'entropy'],
                'splitter': ['best', 'random'],
                'max_depth': [None, 10, 20, 30, 40, 50],
                'min_samples_split': [2, 5, 10],
                'min_samples_leaf': [1, 2, 4]
            }
        },
        'Random Forest': {
            'model': RandomForestClassifier(),
            'params': {
                'n_estimators': [50, 100, 200],
                'criterion': ['gini', 'entropy'],
                'max_depth': [None, 10, 20, 30],
                'min_samples_split': [2, 5, 10],
                'min_samples_leaf': [1, 2, 4]
            }
        },
        'KNN': {
            'model': KNeighborsClassifier(),
            'params': {
                'n_neighbors': [3, 5, 7, 9],
                'weights': ['uniform', 'distance'],
                'p': [1, 2] # p=1: Manhattan, p=2: Euclidean
            }
        },
        'Gradient Boosting': {

```

```

            'model': GradientBoostingClassifier(),
            'params': {
                'n_estimators': [50, 100, 200],
                'learning_rate': [0.01, 0.1, 0.2],
                'max_depth': [3, 4, 5],
                'min_samples_split': [2, 5, 10],
                'min_samples_leaf': [1, 2, 4],
                'subsample': [0.8, 1.0]
            }
        }
    }

```

## **Milestone 6: Model Deployment**

### **Activity 1: Save the best model**

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
# Saving best model
model_name = "final_model.pkl"
with open(f"model/{model_name}", 'wb') as f:
    pickle.dump(best_model, 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 user where he has to enter the values for predictions. The entered 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

- index.html

and save them in the templates folder.

#### **Activity 2.2: Build Python code:**

Import the libraries

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

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (name\_) as argument.

```
# Loading the saved model and scaler
model_path = os.path.join("../", "model", "final_model.pkl")
scaler_path = os.path.join("../", "model", "scaler.pkl")

with open(model_path, 'rb') as f:
    model = pickle.load(f)

with open(scaler_path, 'rb') as f:
    scaler = pickle.load(f)
```

Render HTML page:

```
# -----
# Routes
# -----

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

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

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

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 index.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:

```
def submit():
    try:
        # Extracting form data
        gender = 1 if request.form['Gender'] == 'Male' else 0
        married = 1 if request.form['Married'] == 'Yes' else 0
        dependents = request.form['Dependents'].strip()
        dependents = 3 if dependents == '3+' else int(dependents)
        education = 1 if request.form['Education'] == 'Graduate' else 0
        self_employed = 1 if request.form['Self_Employed'] == 'Yes' else 0
        applicant_income = float(request.form['ApplicantIncome'])
        coapplicant_income = float(request.form['CoapplicantIncome'])
        loan_amount = float(request.form['LoanAmount'])
        loan_term = float(request.form['Loan_Amount_Term'])
        credit_history = float(request.form['Credit_History'])

        prop_area = request.form['Property_Area']
        # One-hot encoding for Property_Area
        prop_urban = 1 if prop_area == 'Urban' else 0
        prop_semiurban = 1 if prop_area == 'Semiurban' else 0
        # If neither, Rural = [0, 0]

        # Final feature vector (must match training order!)
        features = [[
            gender, married, dependents, education, self_employed,
            applicant_income, coapplicant_income, loan_amount,
            loan_term, credit_history,
            prop_semiurban, prop_urban
        ]]
```

```

# Scale the features
features_scaled = scaler.transform(features)

# Make prediction
prediction = model.predict(features_scaled)[0]
proba = model.predict_proba(features_scaled)[0]
print("Model Prediction:", prediction)
print("Prediction Probabilities:", proba)

# Choose response content
if prediction == 1:
    return render_template('submit.html', status="Approved")
else:
    return render_template('submit.html', status="Rejected")

except Exception as e:
    print("Error during prediction:", e)
    return "Something went wrong during prediction. Please try again."

```

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:

```

# Run the app (for local dev only)
if __name__ == '__main__':
    app.run(debug=True)

```

### Activity 2.3: Run the web application

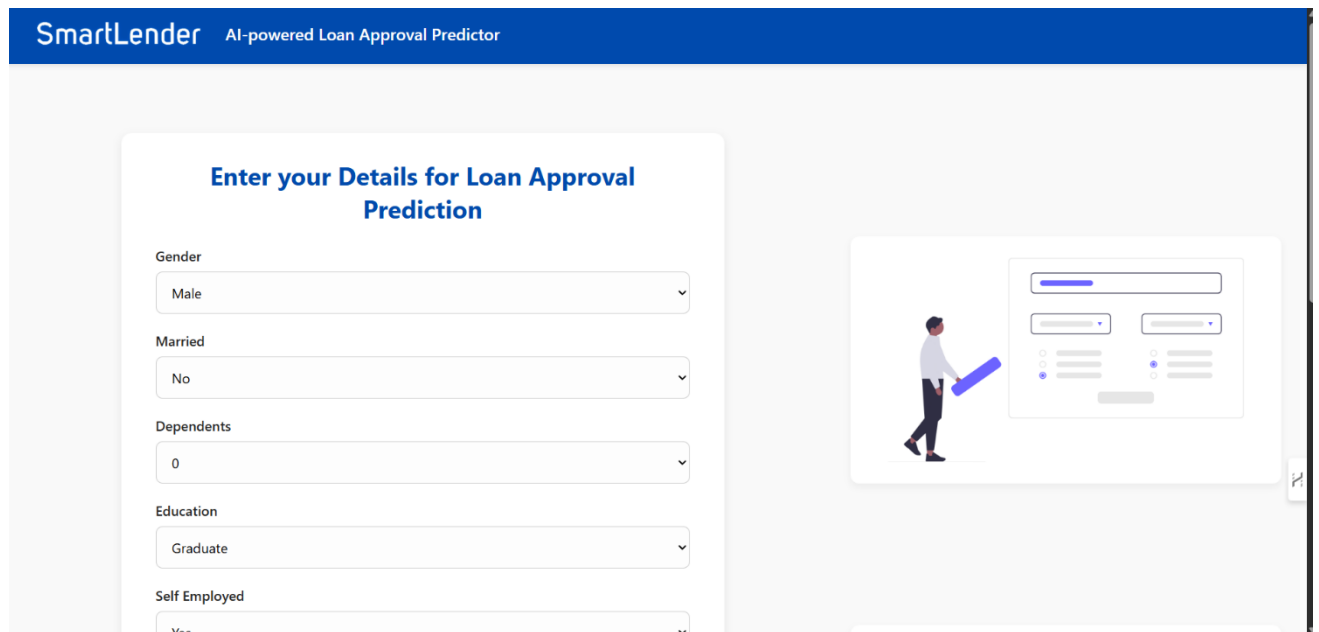
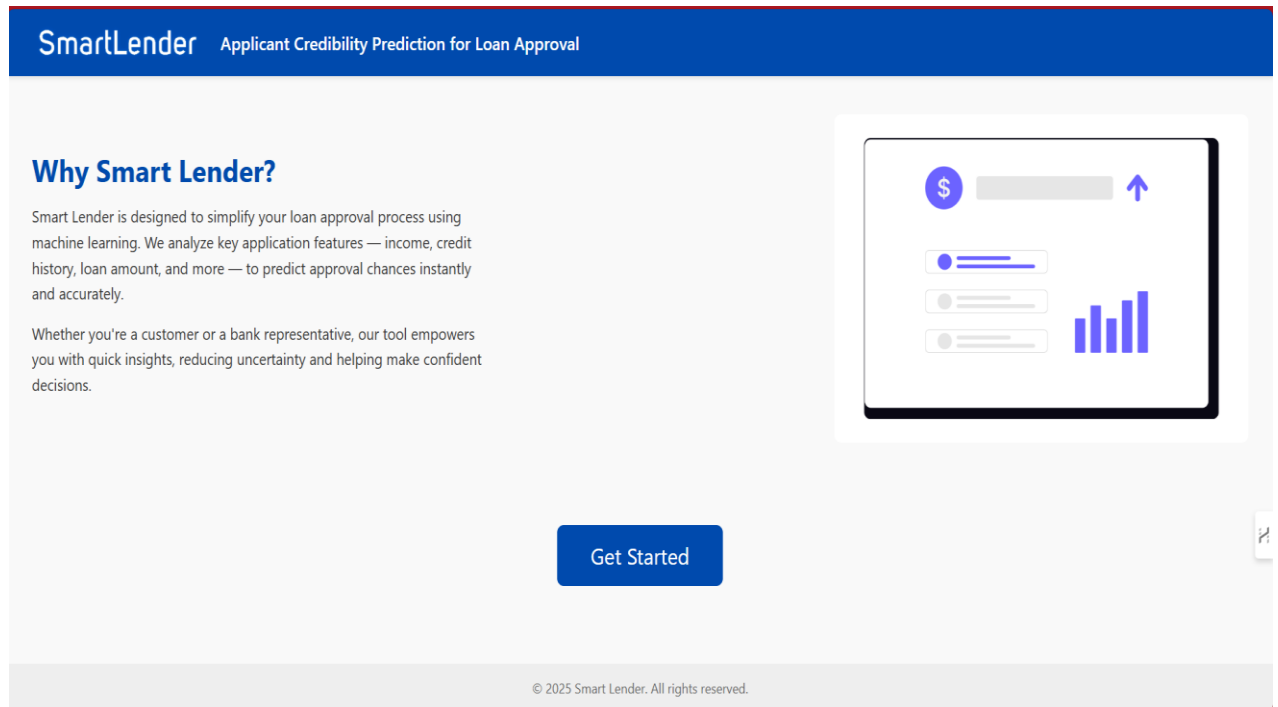
- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see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

```

PS C:\Users\yashd\Downloads\Project_final\Project\app> python app.py
* 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
* Debugger is active!
* Debugger PIN: 112-522-547

```

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





Yes

Applicant Income

10000

Co-applicant Income

5000

Loan Amount

50000

Loan Amount Term

500


Credit History

1.0 - Good

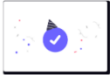
Property Area

Semiurban

Submit



SmartLender AI-powered Loan Approval Predictor



### Congratulations!

Your loan is **likely to be approved** based on the information provided.

Back to Home

© 2025 Smart Lender. All rights reserved.

## Milestone 7: Project Demonstration & Documentation

GitHub Repository: <https://github.com/YashChugani/Smart-Lender-ML-Project>

Project Website: <https://smart-lender.onrender.com/>

Project Demonstration Video: <https://youtu.be/IxInFLzu038>