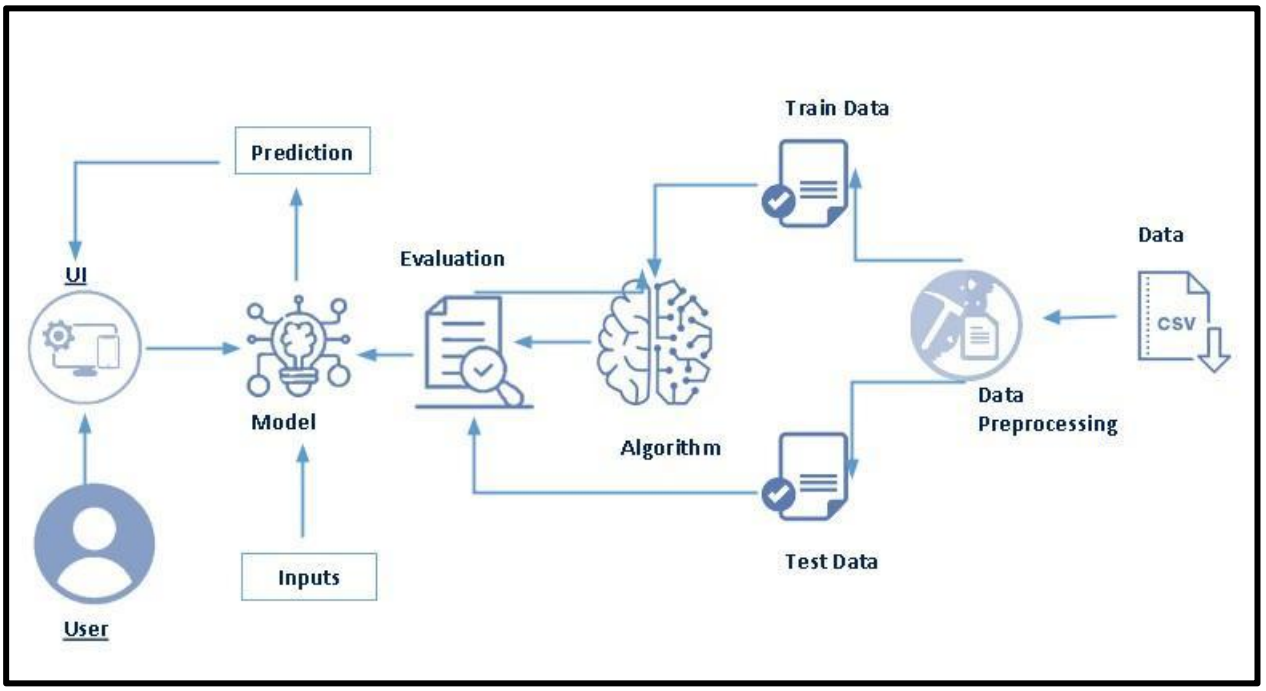
Smart Lender Loan Approval Using Machine Learning

Loans are like financial tools that allow people and businesses to borrow money, which they must then repay over time, usually with some extra money added as interest. There are many types of loans out there, like personal loans, mortgages, car loans, student loans, and business loans. These loans are offered by banks, credit unions, and other financial institutions, each with its own unique rules and requirements. Things like interest rates, how long you have to pay the loan back, and any additional charges can all be different depending on the lender and the specific type of loan.

Now, let's focus on personal loans. Think of them as a flexible way to borrow money without needing to put up any collateral. People use personal loans for all sorts of things, like fixing up their homes, covering medical bills, or consolidating their debts. How much money you can borrow, the interest rate you'll be charged, and how long you have to repay the loan will depend on where you get the loan and how trustworthy you appear to the lender. Generally, to get a personal loan, yo need to show that you have a regular income and a good credit history.

Now, here's where it gets interesting. Machine learning, a type of computer technology, can be used to predict whether someone will be approved for a personal loan. It does this by crunching numbers and analyzing the person's financial info and credit history. This helps banks and other money-lending places make smarter choices about which loan applications they should say "yes" to and which ones they should turn down. So, in a nutshell, it's a bit like having a virtual financial expert to help decide who should get a personal loan and who shouldn't.

Technical Architecture:



Project Flow:

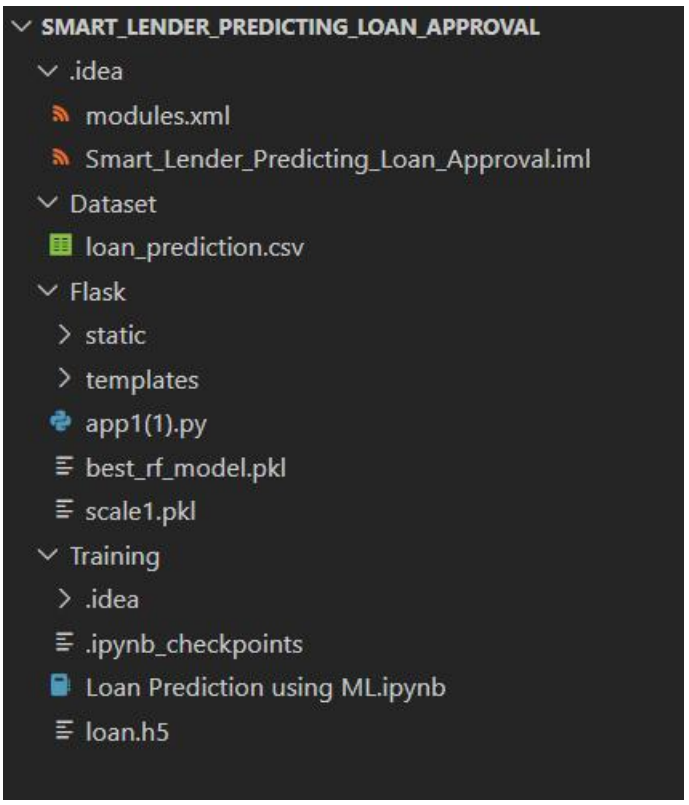
- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

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

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.
- rdf.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains a model training file.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

The business requirements for a machine learning model to predict Smart lender predicting loan approval include the ability to accurately predict loan approval based on applicant information,Minimise the number of false positives (approved loans that default) and false negatives(rejected loans that would have been successful).Provide an explanation for the model's decision, to comply with regulations and improve transparency.

Activity 3: Literature Survey

As the data is increasing daily due to digitization in the banking sector, people want to apply for loans through the internet. Machine Learning (ML), as a typical method for information investigation, has gotten more consideration increasingly. Individuals of various businesses are utilising ML calculations to take care of the issues dependent on their industry information. Banks are facing a significant problem in the approval of the loan. Daily there are so many applications that are challenging to manage by the bank employees, and also the chances of some mistakes are high.Most banks earn profit from the loan, but it is risky to choose deserving customers from the number of applications.There are various algorithms that have been used with varying levels of success. Logistic regression, decision tree, random forest, and neural networks have all been used and have been able to accurately predict loan defaults. Commonly used features in these studies include credit score, income, and employment history, sometimes also other features like age, occupation, and education level.

Activity 4: Social or Business Impact.

Social Impact :- Personal loans can stimulate economic growth by providing individuals with the funds they need to make major purchases, start businesses, or invest in their Education

Business Model/Impact :- Personal loan providers may charge fees for services such as loan origination, processing, and late payments.Advertising the brand awareness and marketing to reach out to potential borrowers to generate revenue.

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

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset

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

```
# Importing the Necessary Libraries
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import RandomizedSearchCV
import imblearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score
```

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.

```
# Reading the Dataset
# Importing the dataset which is in csv file

data = pd.read_csv('F:\VIT AP\DATA SCIENCE EXTERNSHIP\SmartIntern Data Science Project\Smart_Lender_Predicting_Loan_Approval\Dataset\loan_prediction.csv')
data
```

Python

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Ar
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urb
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Ru
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urb
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urb
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urb
...	...	...	...	...	...	...	...	...	...	...	...	
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Ru
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Ru
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urb
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urb
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurb

```
[5]
    # Checking the size of the dataset
    data.shape

...  (614, 13)
```

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 categorical data
- Handling Imbalance Data

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

- Let's find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```
[6]
    # Handling the missing value
    data.info()

...  <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 614 entries, 0 to 613
     Data columns (total 13 columns):
      #   Column             Non-Null Count  Dtype
     ---  ------             --------------  -----
      0   Loan_ID            614 non-null    object
      1   Gender             601 non-null    object
      2   Married            611 non-null    object
      3   Dependents         599 non-null    object
      4   Education          614 non-null    object
      5   Self_Employed      582 non-null    object
      6   ApplicantIncome    614 non-null    int64
      7   CoapplicantIncome  614 non-null    float64
      8   LoanAmount         592 non-null    float64
      9   Loan_Amount_Term   600 non-null    float64
      10  Credit_History     564 non-null    float64
      11  Property_Area      614 non-null    object
      12  Loan_Status        614 non-null    object
     dtypes: float64(4), int64(1), object(8)
     memory usage: 62.5+ KB
```

- For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
[8]  # finding the sum of null values in each column
     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
```

- From the above code of analysis, we can infer that columns such as gender ,married,dependents,self employed ,loan amount, loan amount term and credit history are having the missing values, we need to treat them in a required way.

- We will fill in the missing values in the numeric data type using the mean value of that particular column and categorical data type using the most repeated value

```
#Filling all Nan values with mode of respective variable
data["Gender"].fillna(data["Gender"].mode()[0],inplace=True)
data["Married"].fillna(data["Married"].mode()[0],inplace=True)
data["Self_Employed"].fillna(data["Self_Employed"].mode()[0],inplace=True)
data["Loan_Amount_Term"].fillna(data["Loan_Amount_Term"].mode()[0],inplace=True)
data["Dependents"].fillna(data["Dependents"].mode()[0],inplace=True)
data["Credit_History"].fillna(data["Credit_History"].mode()[0],inplace=True)
```

Activity 2.2: Handling Categorical Values

As we can see our dataset has categorical data we must convert the categorical data to integer encoding or binary encoding.

To convert the categorical features into numerical features we use encoding techniques. There are several techniques but in our project we are using manual encoding with the help of list comprehension.

- In our project, Gender ,married,dependents,self-employed,co-applicants income,loan amount ,loan amount term, credit history With list comprehension encoding is done.

```
#All values of "Dependents" columns were of "str" form now converting to "int" form.
data["Dependents"] = data["Dependents"].replace('3+',int(3))
data["Dependents"] = data["Dependents"].replace('1',int(1))
data["Dependents"] = data["Dependents"].replace('2',int(2))
data["Dependents"] = data["Dependents"].replace('0',int(0))

data["LoanAmount"].fillna(data["LoanAmount"].median(),inplace=True)
```

Activity 2.3:Handling Imbalance Data

Data Balancing is one of the most important step, which need to be performed for classification models, because when we train our model on imbalanced dataset ,we will get biassed results, which means our model is able to predict only one class element

For Balancing the data we are using the SMOTE Method.

SMOTE: Synthetic minority over sampling technique, which will create new synthetic data points for under class as per the requirements given by us using KNN method.

```
sc=StandardScaler()
x_bal=sc.fit_transform(x_bal)


x_bal=pd.DataFrame(x_bal,columns=names)
```

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
```

[11]

Python

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	128.0	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y
...	...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural	Y
610	LP002979	Male	Yes	3	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural	Y
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban	Y
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban	Y
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban	N

614 rows × 13 columns

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.

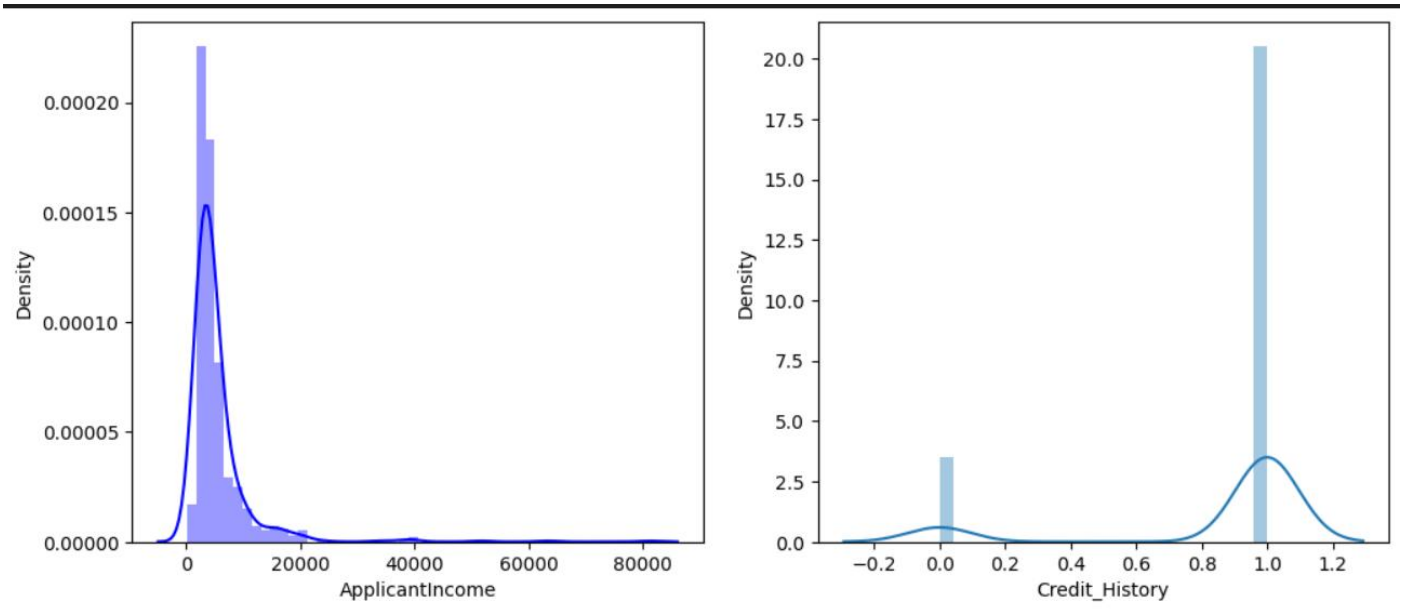
Activity 2.1: Univariate analysis

In simple words, univariate analysis is understanding the data with a single feature. Here we have displayed two different graphs such as distplot and countplot.

- The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

```
plt.figure(figsize=(12,5))
plt.subplot(121)
sns.distplot(data['ApplicantIncome'], color='b')
plt.subplot(122)
sns.distplot(data['Credit_History'])
plt.show()
```

```
C:\Users\thara\AppData\Local\Temp\ipykernel_17444\1608621340.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['ApplicantIncome'], color='b')
C:\Users\thara\AppData\Local\Temp\ipykernel_17444\1608621340.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(data['Credit_History'])
```



● In our dataset we have some categorical features. With the count plot function, we are going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot we have plotted this below graph.

● From the plot we came to know, Applicants income is skewed towards left side, where as credit history is categorical with 1.0 and 0.0

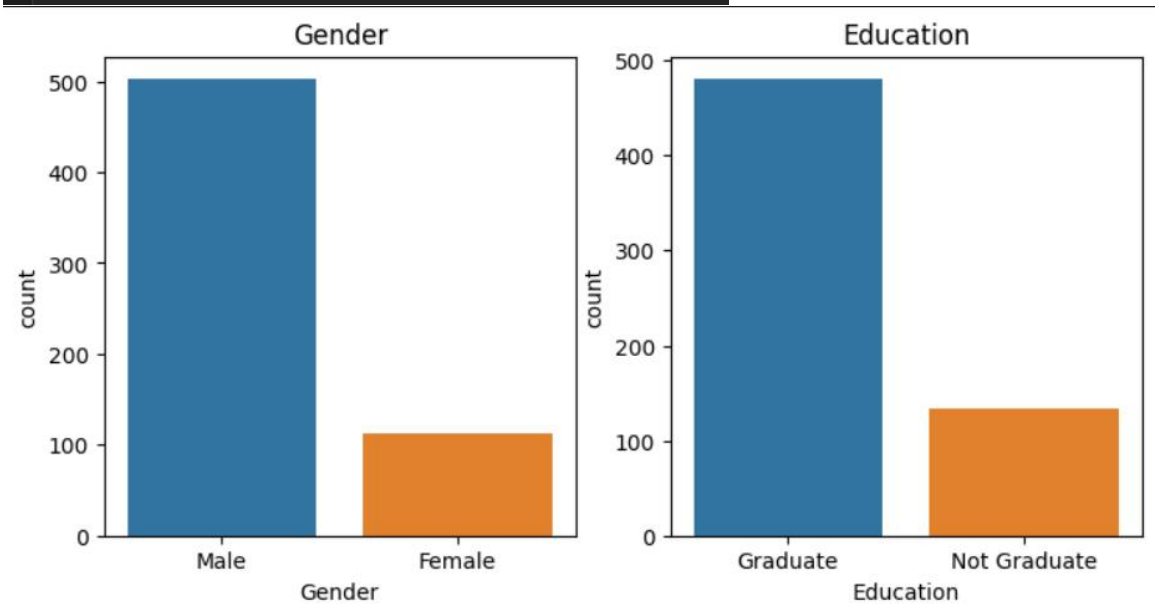
Countplot:-
A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot() , so you can compare counts across nested variables.

From the graph we can infer that , gender and education is a categorical variables with 2 categories , from gender column we can infer that 0-category is having more weightage than category-1,while education with 0,it means no education is a underclass when compared with category -1, which means educated

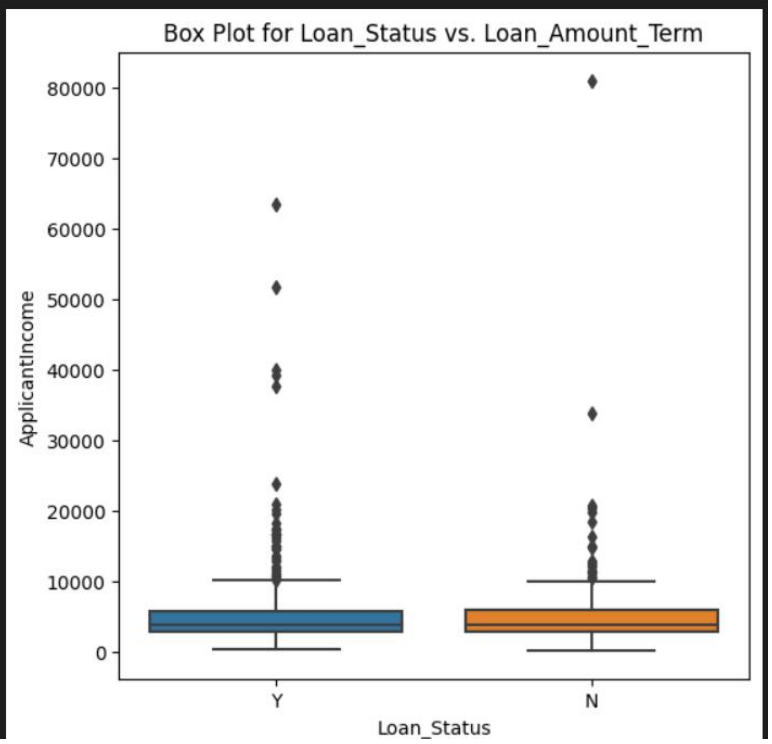
Activity 2.2: Bivariate analysis

BIVARIATE ANALYSIS

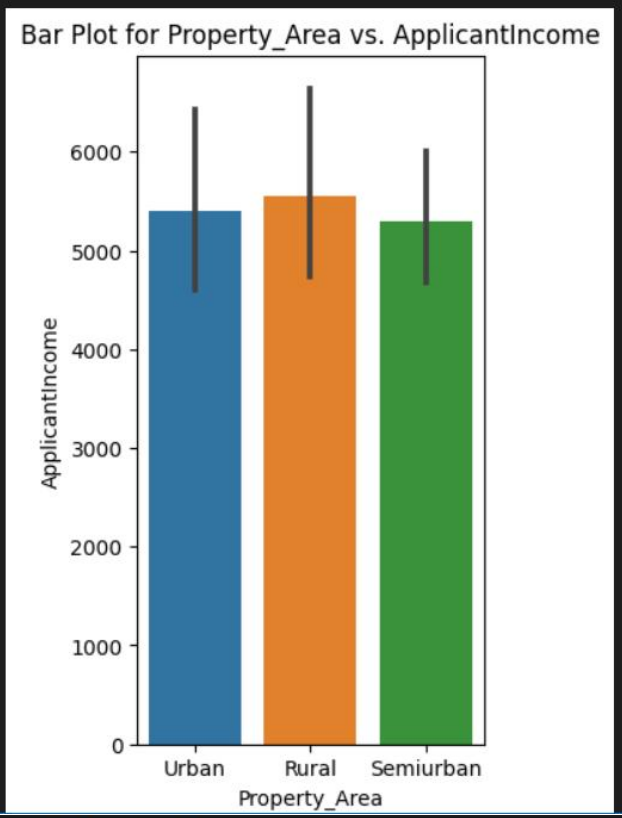
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(18, 4))

# Subplot 1: Gender
plt.subplot(1, 4, 1)
sns.countplot(data=data, x='Gender')
plt.title('Gender')

# Subplot 2: Education
plt.subplot(1, 4, 2)
sns.countplot(data=data, x='Education')
plt.title('Education')

plt.show()
```



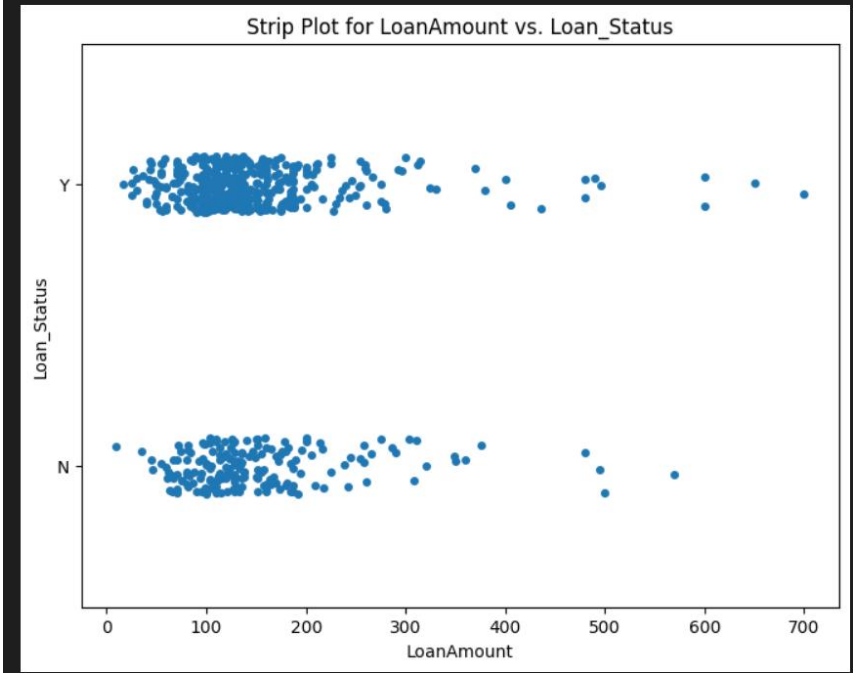
```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 6))
sns.boxplot(x='Loan_Status', y='ApplicantIncome', data=data)
plt.title('Box Plot for Loan_Status vs. Loan_Amount_Term')
plt.show()
```



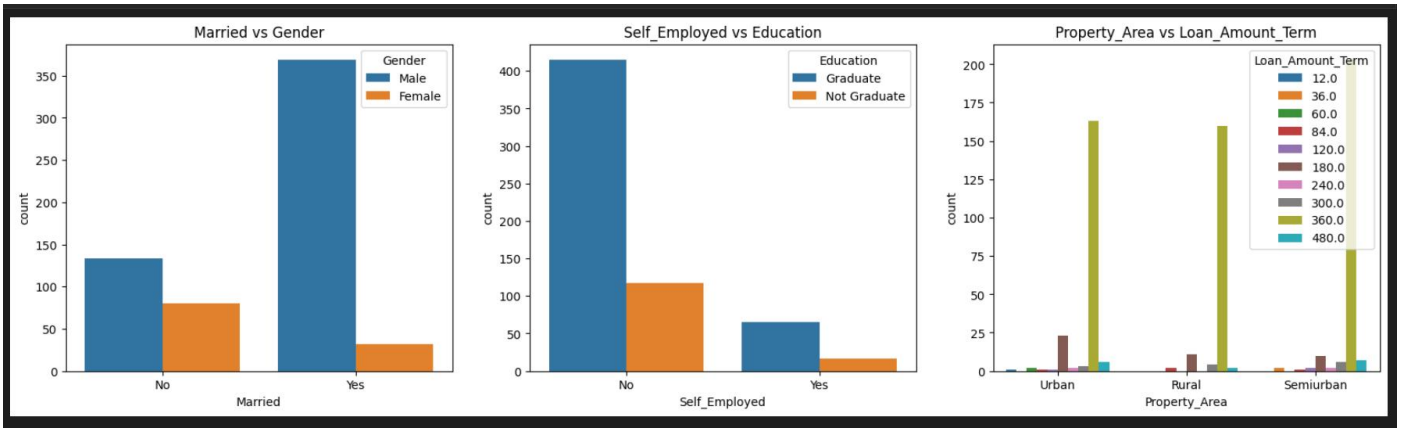
```python
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(3, 6))
sns.barplot(x='Property_Area', y='ApplicantIncome', data=data)
plt.title('Bar Plot for Property_Area vs. ApplicantIncome')
plt.show()
```




```python
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
sns.stripplot(x='LoanAmount', y='Loan_Status', data=data)
plt.title('Strip Plot for LoanAmount vs. Loan_Status')
plt.show()
```



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

# Replace 'data' with your DataFrame
plt.figure(figsize=(20, 5))

# Subplot 1: Married vs Gender
plt.subplot(131)
sns.countplot(data=data, x="Married", hue="Gender")
plt.title("Married vs Gender")

# Subplot 2: Self_Employed vs Education
plt.subplot(132)
sns.countplot(data=data, x="Self_Employed", hue="Education")
plt.title("Self_Employed vs Education")

# Subplot 3: Property_Area vs Loan_Amount_Term
plt.subplot(133)
sns.countplot(data=data, x="Property_Area", hue="Loan_Amount_Term")
plt.title("Property_Area vs Loan_Amount_Term")

plt.show()
```



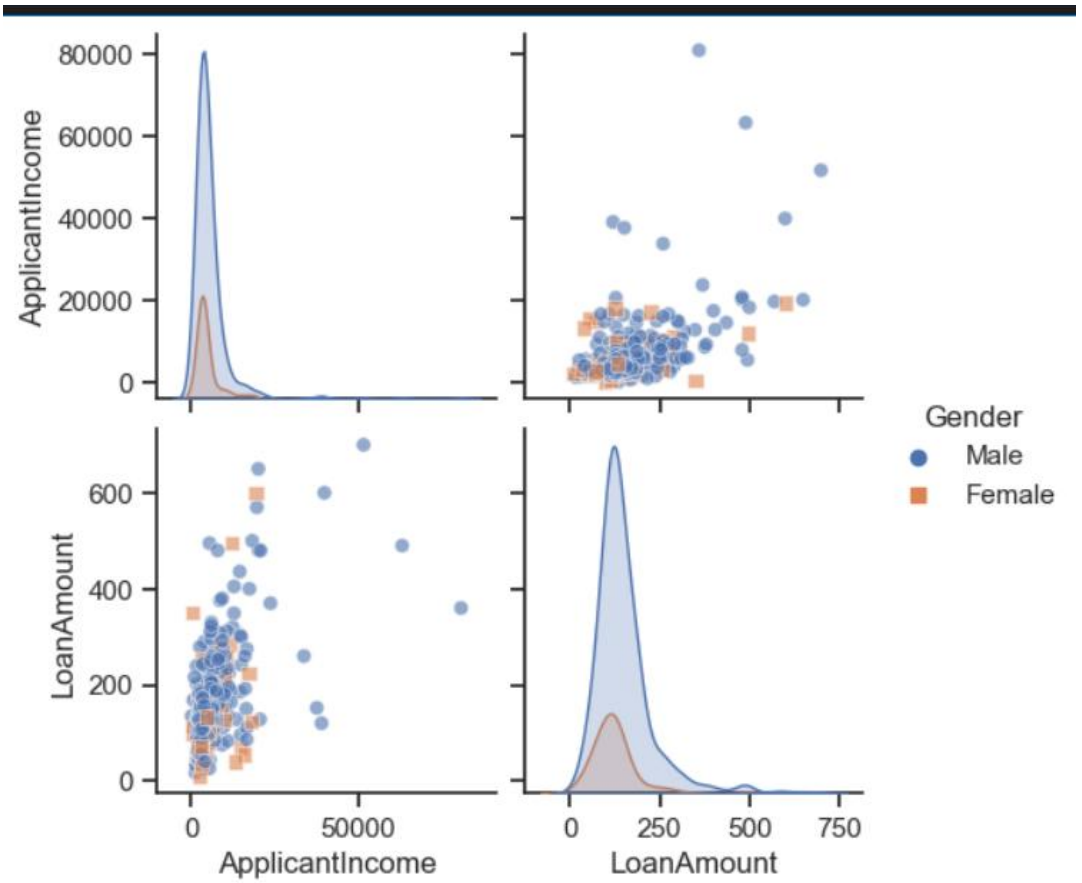
From the above graph we can infer the analysis such as

- Segmenting the gender column and married column based on bar graphs
- Segmenting the Education and Self-employed based on bar graphs ,for drawing insights such as educated people are employed.
- Loan amount term based on the property area of a person holding

Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used a swarm plot from the seaborn package.

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="ticks")
sns.pairplot(data, hue="Gender", vars=['ApplicantIncome', 'LoanAmount'], diag_kind="kde", markers=["o", "s"], plot_kws={'alpha': 0.6})
plt.show()
```



From the above graph we are plotting the relationship between the Gender, applicants income and loan status of the person.

Now, the code would be normalising the data by scaling it to have a similar range of values, and then splitting that data into a training set and a test set for training the model and testing its performance, respectively.

Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as KNN, Logistic regression need scaled data, as they follow distance based method and Gradient Descent concept

ONE-HOT ENCODING

```
# List of categorical columns for one-hot encoding
categorical_columns = ['Gender', 'Married', 'Education', 'Self_Employed', 'Property_Area']

# Apply one-hot encoding
data = pd.get_dummies(data, columns=categorical_columns, drop_first=True)
```

LABEL ENCODER

```python
from sklearn.preprocessing import LabelEncoder

# Assuming 'data' is your DataFrame
label_encoder = LabelEncoder()

# Apply label encoding to the 'Loan_Status' column
data['Loan_Status'] = label_encoder.fit_transform(data['Loan_Status'])
```

SCALING THE DATA

```python
from sklearn.preprocessing import StandardScaler

# Initialize the StandardScaler
scaler = StandardScaler()

# Specify the columns to scale (assuming all columns need to be scaled)
columns_to_scale = ['Dependents','ApplicantIncome','CoapplicantIncome','LoanAmount','Loan_Amount_Term']

# Fit and transform the data using the StandardScaler
data[columns_to_scale] = scaler.fit_transform(data[columns_to_scale])
```

Python

[29]
```python
# Handling Imbalance Data
from imblearn.combine import SMOTETomek
```

[30]
```python
smote = SMOTETomek(sampling_strategy=0.90)
```

[31]
```python
# Dividing the data into dependent and independent y and x respectively
y = data['Loan_Status']
x = data.drop(columns=['Loan_Status','Loan_ID'],axis=1)
```

[32]
```python
data['Loan_Status'].value_counts()
```

```
Loan_Status
1    422
0    192
Name: count, dtype: int64
```

[33]
```python
# creating a new x and y variables for the balanced set
X_resampled,Y_resampled = smote.fit_resample(x,y)
```

Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: 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 the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

SPLITTING DATA INTO TRAIN AND TEST

```
[35]  X_train, X_test, y_train, y_test = train_test_split(X_resampled, Y_resampled, test_size=0.33, random_state=42)
```

```
[36]  X_train.head()
```

Python

	Dependents	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Gender_M
114	-0.737806	-0.180775	0.134327	0.574111	0.273231	1.0	
665	-0.737806	0.163292	-0.554487	-0.415697	0.273231	0.0	
721	0.934376	0.519356	6.802746	2.422586	-0.687076	1.0	
664	-0.737806	-0.520932	0.473169	-0.481895	0.273231	0.0	
423	1.244745	-0.850820	0.443165	-0.568220	0.273231	1.0	F

```
[37]  X_train.columns
```

```
Index(['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'],
      dtype='object')
```

```
[38]  X_train.shape
```

```
(496, 12)
```

```
[39]  y_train.head()

...   114    1
      665    0
      721    0
      664    0
      423    1
      Name: Loan_Status, dtype: int32
```

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 four classification algorithms. The best model is saved based on its performance.

Activity 1.1: Decision tree model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, 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. For evaluating the model, a confusion matrix and classification report is done.

```
[40]  # Decision Tree Model
      def decisionTree(X_train,X_test,y_train,y_test):
          dt = DecisionTreeClassifier()
          dt.fit(X_train,y_train)
          yPred = dt.predict(X_test)
          print('***DecisionTreeClassifier***')
          print('Confusion Matrix')
          print(confusion_matrix(y_test,yPred))
          print('Classification report')
          print(classification_report(y_test,yPred))
```

Activity 1.2: Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, 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. For evaluating the model, a confusion matrix and classification report is done.

```
# Random Forest Model
def randomForest(X_train,X_test,y_train,y_test):
    rfr = RandomForestClassifier()
    rfr.fit(X_train,y_train)
    yPred = rfr.predict(X_test)
    print('***RandomForestClassifier***')
    print('Confusion Matrix')
    print(confusion_matrix(y_test,yPred))
    print('classification_report')
    print(classification_report(y_test,yPred))
```

Activity 1.3: KNN model

A function named KNN is created and train and test data are passed as the parameters. Inside the function, 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.

```
# KNN Model
def KNN(X_train,X_test,y_train,y_test):
    knn = KNeighborsClassifier()
    knn.fit(X_train,y_train)
    yPred =knn.predict(X_test)
    print('***KNeighborsClassifer***')
    print('Confusion Matrix')
    print(confusion_matrix(y_test,yPred))
    print('classification_report')
    print(classification_report(y_test,yPred))
```

Activity 1.4: Xgboost model

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier 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.

```python
# Xgboost Model
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train)
    yPred = xg.predict(x_test)
    print(' *** GradientBoostingClassifier *** ')
    print('Confusion matrix')
    print(confusion_matrix(y_test, yPred))
    print('Classification report')
    print(classification_report(y_test, yPred))
```

Activity 1.5: ANN model

Building and training an Artificial Neural Network (ANN) using the Keras library with TensorFlow as the backend. The ANN is initialised as an instance of the Sequential class, which is a linear stack of layers. Then, the input layer and two hidden layers are added to the model using the Dense class, where the number of units and activation function are specified. The output layer is also added using the Dense class with a sigmoid activation function. The model is then compiled with the Adam optimizer, binary cross-entropy loss function, and accuracy metric. Finally, the model is fit to the training data with a batch size of 100, 20% validation split, and 100 epochs.

```python
# ANN model
from tensorflow import keras
from tensorflow.keras import layers

classifier = keras.Sequential([
    layers.Dense(units=100, activation='relu', input_dim=12),  # Input layer with 12 features
    layers.Dense(units=50, activation='relu'),
    layers.Dense(units=1, activation='sigmoid')
])

classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Fit the model to the training set
model_history = classifier.fit(X_train, y_train, batch_size=100, validation_split=0.2, epochs=100)
```

```
Epoch 1/100
4/4 [==============================] - 2s 137ms/step - loss: 0.6866 - accuracy: 0.5884 - val_loss: 0.6689 - val_accuracy: 0.6600
Epoch 2/100
4/4 [==============================] - 0s 26ms/step - loss: 0.6670 - accuracy: 0.6591 - val_loss: 0.6597 - val_accuracy: 0.6800
Epoch 3/100
4/4 [==============================] - 0s 29ms/step - loss: 0.6518 - accuracy: 0.7146 - val_loss: 0.6521 - val_accuracy: 0.6900
Epoch 4/100
4/4 [==============================] - 0s 36ms/step - loss: 0.6376 - accuracy: 0.7424 - val_loss: 0.6409 - val_accuracy: 0.7300
Epoch 5/100
4/4 [==============================] - 0s 34ms/step - loss: 0.6237 - accuracy: 0.7626 - val_loss: 0.6291 - val_accuracy: 0.7600
Epoch 6/100
4/4 [==============================] - 0s 35ms/step - loss: 0.6098 - accuracy: 0.7626 - val_loss: 0.6181 - val_accuracy: 0.7600
Epoch 7/100
4/4 [==============================] - 0s 33ms/step - loss: 0.5956 - accuracy: 0.7727 - val_loss: 0.6063 - val_accuracy: 0.7600
Epoch 8/100
4/4 [==============================] - 0s 40ms/step - loss: 0.5808 - accuracy: 0.7753 - val_loss: 0.5938 - val_accuracy: 0.7600
Epoch 9/100
4/4 [==============================] - 0s 40ms/step - loss: 0.5661 - accuracy: 0.7727 - val_loss: 0.5819 - val_accuracy: 0.7600
Epoch 10/100
4/4 [==============================] - 0s 43ms/step - loss: 0.5512 - accuracy: 0.7828 - val_loss: 0.5730 - val_accuracy: 0.7600
Epoch 13/100
...
Epoch 99/100
4/4 [==============================] - 0s 41ms/step - loss: 0.2058 - accuracy: 0.9217 - val_loss: 0.6758 - val_accuracy: 0.7400
Epoch 100/100
4/4 [==============================] - 0s 44ms/step - loss: 0.2060 - accuracy: 0.9217 - val_loss: 0.6804 - val_accuracy: 0.7400
```

Activity 2: Testing the model

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor()
dtr.fit(x, y)
predictions = dtr.predict([[1, 1, 0, 1, 1, 4276, 1542, 145, 240, 0, 1, 0]])
print(predictions)
```

```
[0.]
```

```
from sklearn.ensemble import RandomForestRegressor
rfr = RandomForestRegressor()
rfr.fit(x, y)
predictions = rfr.predict([[1, 1, 0, 1, 1, 4276, 1542, 145, 240, 0, 1, 0]])
print(predictions)
```

```
[0.76]
```

```
from sklearn.neighbors import KNeighborsRegressor
(variable) predictions: ndarray
predictions = knn.predict([[1, 1, 0, 1, 1, 4276, 1542, 145, 240, 0, 1, 0]])
print(predictions)
```

```
[48]
...
[0.6]
```

In ANN we first have to save the model to the test the inputs

```
[50]  classifier.save("loan.h5")

...   C:\Users\thara\AppData\Local\Packages\Pytho
        saving_api.save_model(
```

```
X_test= np.array(X_test, dtype=np.float32)
```

```
# Predict the test data using the trained model
y_pred = classifier.predict(X_test)
# Convert predicted probabilities to binary labels (0 or 1)
y_pred_binary = np.round(y_pred)
# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy:.2f}")
# Create a confusion matrix
confusion = confusion_matrix(y_test, y_pred_binary)
print("Confusion Matrix:")
print(confusion)
# Generate a classification report
classification_report = classification_report(y_test, y_pred_binary, target_names=["Class 0", "Class 1"])
print("Classification Report:")
print(classification_report)
```

```
...  8/8 [==============================] - 0s 4ms/step
Accuracy: 0.75
Confusion Matrix:
[[85 28]
 [33 99]]
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.72      0.75      0.74       113
     Class 1       0.78      0.75      0.76       132

    accuracy                           0.75       245
   macro avg       0.75      0.75      0.75       245
weighted avg       0.75      0.75      0.75       245
```

```
y_pred
```

```
array([[7.5612742e-01],
       [5.3672010e-01],
       [8.0714762e-01],
       [5.4747224e-01],
       [9.8572671e-01],
       [5.0689077e-01],
       [8.8916522e-01],
       [4.4941515e-01],
       [5.6102759e-01],
       [4.9683532e-01],
       [9.4559437e-01],
       [5.6683576e-01],
       [1.4354399e-02],
       [9.9338758e-01],
       [1.9491999e-01],
```

```
y_pred = (y_pred > 0.5)
y_pred
```

```
array([[ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
       [ True],
       [False],
       [False],
       [ True],
       [ True],
       [ True],
       [ True],
       [ True],
       [False],
       [ True],
       [ True],
       [False],
```

This code defines a function named "predict_exit" which takes in a sample_value as an input. The function then converts the input sample_value from a list to a numpy array. It reshapes the sample_value array as it contains only one record. Then, it applies feature scaling to the reshaped sample_value array using a scaler object 'sc' that should have been previously defined and fitted. Finally, the function returns the prediction of the classifier on the scaled sample_value.

```
def predict_exit(sample_value):
    sample_value = np.array(sample_value)
    sample_value = sample_value.reshape(1,-1)
    sample_value = scalar.transform(sample_value)
    return classifier.predict(sample_value)
```

[ ]

```
# Sample value for prediction
sample_value = [[1, 1, 0, 1, 1, 4276, 1542, 145, 240, 0, 1, 0]]
# Make a prediction using your model
predicted_probability = classifier.predict(sample_value)
# Assuming that 'classifier' is your trained model (e.g., a Keras model)
# It may return a probability; you can interpret it as the likelihood of loan approval.
# Set a threshold to determine the prediction result
threshold = 0.5
if predicted_probability > threshold:
    print("Prediction: High chance of Loan Approval!")
else:
    print("Prediction: Low chance of Loan Approval!")
```

```
1/1 [==============================] - 0s 280ms/step
Prediction: High chance of Loan Approval!
```

```
# Sample value for prediction
sample_value = [[1, 0, 1, 1, 1, 45, 14, 45, 240, 1, 1, 0]]
# Make a prediction using your model
predicted_probability = classifier.predict(sample_value)
# Assuming that 'classifier' is your trained model (e.g., a Keras model)
# It may return a probability; you can interpret it as the likelihood of loan approval.
# Set a threshold to determine the prediction result
threshold = 0.5
if predicted_probability > threshold:
    print("Prediction: High chance of Loan Approval!")
else:
    print("Prediction: Low chance of Loan Approval!")
```

```
1/1 [==============================] - 0s 165ms/step
Prediction: Low chance of Loan Approval!
```

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

For comparing the above four models, the compareModel function is defined.

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def train_decision_tree(X_train, y_train):
    dt = DecisionTreeClassifier()
    dt.fit(X_train, y_train)
    return dt

def train_random_forest(X_train, y_train):
    rfr = RandomForestClassifier()
    rfr.fit(X_train, y_train)
    return rfr

def train_xgboost(X_train, y_train):
    xg = GradientBoostingClassifier()
    xg.fit(X_train, y_train)
    return xg

def train_knn(X_train, y_train):
    knn = KNeighborsClassifier()
    knn.fit(X_train, y_train)
    return knn

def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print('Accuracy:', accuracy)
    print('Confusion Matrix:')
    print(confusion_matrix(y_test, y_pred))
    print('Classification Report:')
    print(classification_report(y_test, y_pred))
    return accuracy
```

```python
def compareModel(X_train, X_test, y_train, y_test):
    model_names = ['Decision Tree', 'Random Forest', 'XGBoost', 'KNN']
    models = [train_decision_tree(X_train, y_train), train_random_forest(X_train, y_train),
              train_xgboost(X_train, y_train), train_knn(X_train, y_train)]

    accuracies = []

    for model_name, model in zip(model_names, models):
        print(f'*** {model_name} ***')
        accuracy = evaluate_model(model, X_test, y_test)
        accuracies.append((model_name, accuracy))
        print('-' * 100)

    max_accuracy_model = max(accuracies, key=lambda x: x[1])
    print(f'Highest Accuracy Model: {max_accuracy_model[0]} with Accuracy {max_accuracy_model[1]}')

    highest_accuracy = max_accuracy_model[1]  # Store the highest accuracy in a separate variable
    return highest_accuracy

# Split your data into training and testing sets (X_train, X_test, y_train, y_test)
highest_accuracy = compareModel(X_train, X_test, y_train, y_test)
print(f'Highest Accuracy: {highest_accuracy}')
```

```
*** Decision Tree ***
Accuracy: 0.6979591836734694
Confusion Matrix:
[[76 37]
 [37 95]]
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.67      0.67       113
           1       0.72      0.72      0.72       132

    accuracy                           0.70       245
   macro avg       0.70      0.70      0.70       245
weighted avg       0.70      0.70      0.70       245

--------------------------------------------------------------------------------------------------
*** Random Forest ***
Accuracy: 0.8285714285714286
Confusion Matrix:
[[ 79  34]
 [  8 124]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.70      0.79       113
...

--------------------------------------------------------------------------------------------------
Highest Accuracy Model: Random Forest with Accuracy 0.8285714285714286
Highest Accuracy: 0.8285714285714286
```

```
    compareModel(X_train, X_test, y_train, y_test)
```

```
*** Decision Tree ***
Accuracy: 0.7061224489795919
Confusion Matrix:
[[77 36]
 [36 96]]
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.68      0.68       113
           1       0.73      0.73      0.73       132

    accuracy                           0.71       245
   macro avg       0.70      0.70      0.70       245
weighted avg       0.71      0.71      0.71       245

--------------------------------------------------------------------------------
*** Random Forest ***
Accuracy: 0.8326530612244898
Confusion Matrix:
[[ 81  32]
 [  9 123]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.72      0.80       113
...
weighted avg       0.73      0.73      0.73       245

--------------------------------------------------------------------------------
Highest Accuracy Model: Random Forest with Accuracy 0.8326530612244898
```

```python
# Predict the test data using the trained model
y_pred = classifier.predict(X_test)

# Convert predicted probabilities to binary labels (0 or 1)
y_pred_binary = np.round(y_pred)

# Calculate the accuracy of the model on the test data
accuracy = accuracy_score(y_test, y_pred_binary)
print(f"Accuracy: {accuracy:.2f}")

# Create a confusion matrix
confusion = confusion_matrix(y_test, y_pred_binary)
print("Confusion Matrix:")
print(confusion)

# Generate a classification report
report = classification_report(y_test, y_pred_binary, target_names=["Class 0", "Class 1"])
print("Classification Report:")
print(report)
```

```
8/8 [==============================] - 0s 4ms/step
Accuracy: 0.75
Confusion Matrix:
[[85 28]
 [33 99]]
Classification Report:
              precision    recall  f1-score   support

     Class 0       0.72      0.75      0.74       113
     Class 1       0.78      0.75      0.76       132

    accuracy                           0.75       245
   macro avg       0.75      0.75      0.75       245
weighted avg       0.75      0.75      0.75       245
```

After calling the function, the results of models are displayed as output. From the five models RandomForest is performing well. From the below image, We can see the accuracy of the model. RandomForest is giving the accuracy of 82.85% with training data , 82.2% accuracy for the testing data.

Activity 2:Comparing model accuracy before & after applying hyperparameter tuning

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. So, we are saving the model by pickle.dump().

Note: To understand cross validation, refer to this link

```
from sklearn.model_selection import cross_val_score
```

```
#Random forest model is selected
rf= RandomForestClassifier()
rf.fit(X_train,y_train)
yPred=rf.predict(X_test)
```

```
f1_score(yPred,y_test,average='weighted')
```

```
0.8071105365223011
```

```
cv=cross_val_score(rf,x,y,cv=5)
```

```
np.mean(cv)
```

```
0.7850593096094897
```

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

# Define the hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4]  # Minimum number of samples required to be at a leaf node
}

# Create a Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Use Grid Search to find the best hyperparameters
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

# Get the best model with optimal hyperparameters
best_rf_model = grid_search.best_estimator_

# Fit the best model to the training data
best_rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred = best_rf_model.predict(X_test)

# Calculate the accuracy on the test set
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8163265306122449
```

```
    print('After Tunning Accuracy:' , accuracy)
    print('Before Tunning Accuracy:', highest_accuracy)
    randomForest(X_train,X_test,y_train,y_test)
```

```
After Tunning Accuracy: 0.8163265306122449
Before Tunning Accuracy: 0.8285714285714286
***RandomForestClassifier***
Confusion Matrix
[[ 81  32]
 [ 13 119]]
classification_report
              precision    recall  f1-score   support

           0       0.86      0.72      0.78       113
           1       0.79      0.90      0.84       132

    accuracy                           0.82       245
   macro avg       0.82      0.81      0.81       245
weighted avg       0.82      0.82      0.81       245
```

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 and saving its weights and configuration. 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.

```
    import pickle

    # Assuming that grid_search contains the best trained model
    best_rf_model = grid_search.best_estimator_

    # Save the model to a file
    with open('best_rf_model.pkl', 'wb') as model_file:
        pickle.dump(best_rf_model, model_file)
```

```
    # Load the saved model
    with open('best_rf_model.pkl', 'rb') as model_file:
        loaded_model = pickle.load(model_file)

    # Now, you can use loaded_model to make predictions
```

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 Pages:

For this project create two HTML files namely

- home.html
- predict.html

and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

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

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.

```
app = Flask(__name__)
model = pickle.load(open('C:/Users/Tanmay Anand/OneDrive/Desktop/Smart_Lender_Predicting_Loan_Approval/Smart_Lender_Predicting_Loan_Approval/Flask/best_rf_model.pkl', 'rb'))
scale = pickle.load(open('C:/Users/Tanmay Anand/OneDrive/Desktop/Smart_Lender_Predicting_Loan_Approval/Smart_Lender_Predicting_Loan_Approval/Flask/scale1.pkl','rb'))
```

Render HTML page:

```
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
@app.route('/predict',methods=["POST","GET"]) # rendering the html template
def predict() :
    return render_template("input.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('/submit',methods=["POST","GET"])# route to show the predictions in a web UI
def submit():
    #  reading the inputs given by the user
    input_feature=[int(x) for x in request.form.values() ]
    #input_feature = np.transpose(input_feature)
    input_feature=[np.array(input_feature)]
    print(input_feature)
    names = ['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']
    data = pandas.DataFrame(input_feature,columns=names)
    print(data)

    #data_scaled = scale.fit_transform(data)
    #data = pandas.DataFrame(,columns=names)




    # predictions using the loaded model file
    prediction=model.predict(data)
    print(prediction)
    prediction = int(prediction)
    print(type(prediction))

    if (prediction == 0):
       return render_template("output.html",result ="Loan wiil Not be Approved")
    else:
       return render_template("output.html",result = "Loan will be Approved")
    # showing the prediction results in a UI
```

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__":

    # app.run(host='0.0.0.0', port=8000,debug=True)    # running the app
    port=int(os.environ.get('PORT',5000))
    app.run(debug=False)
```

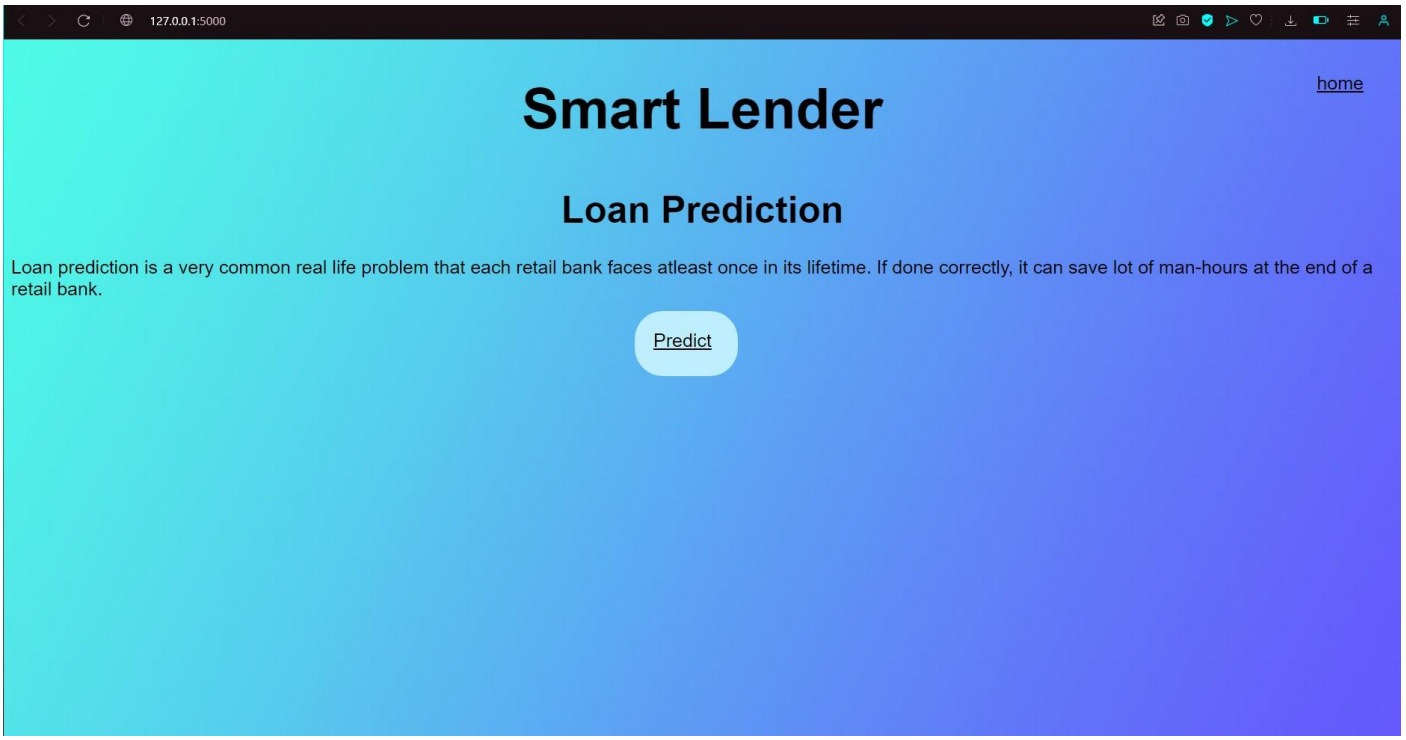
Activity 2.3: Run the web application

- 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.

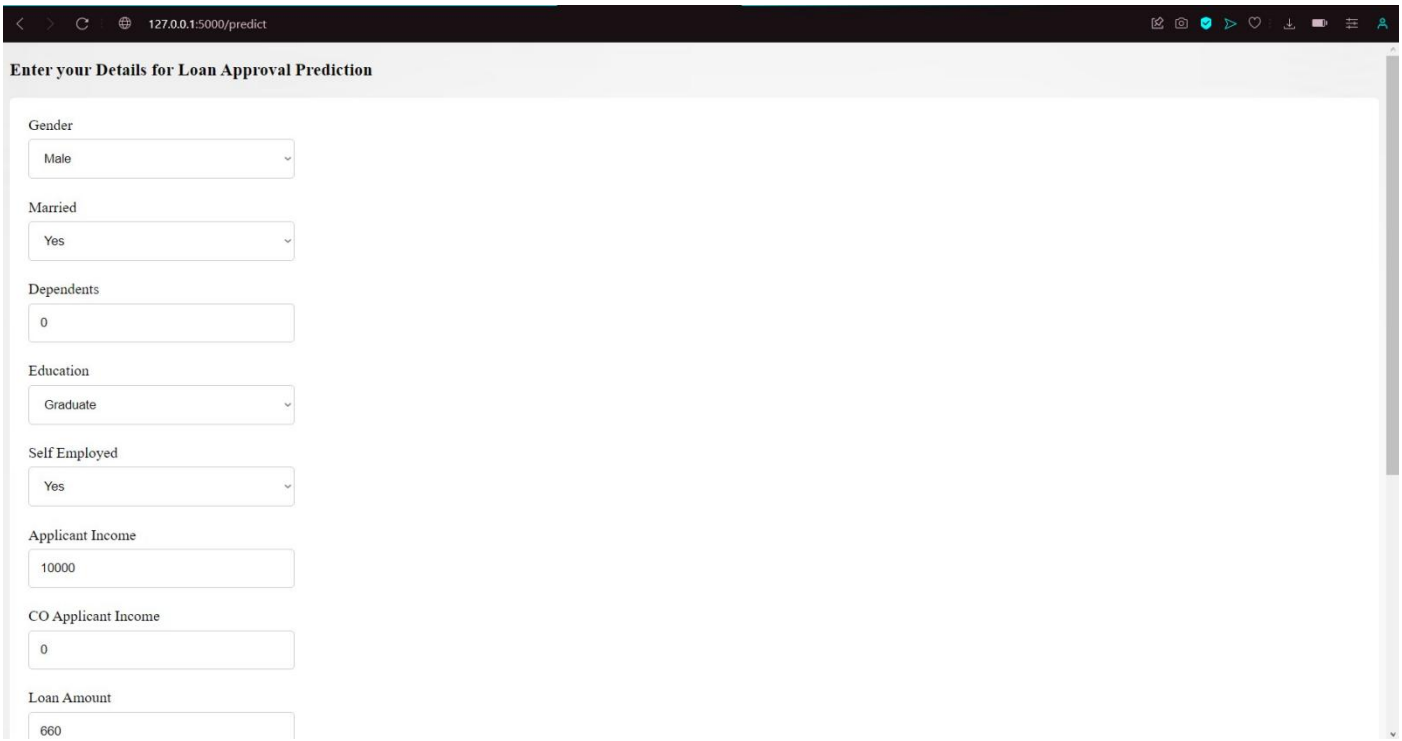
```
 * Serving Flask app 'app1(1)'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

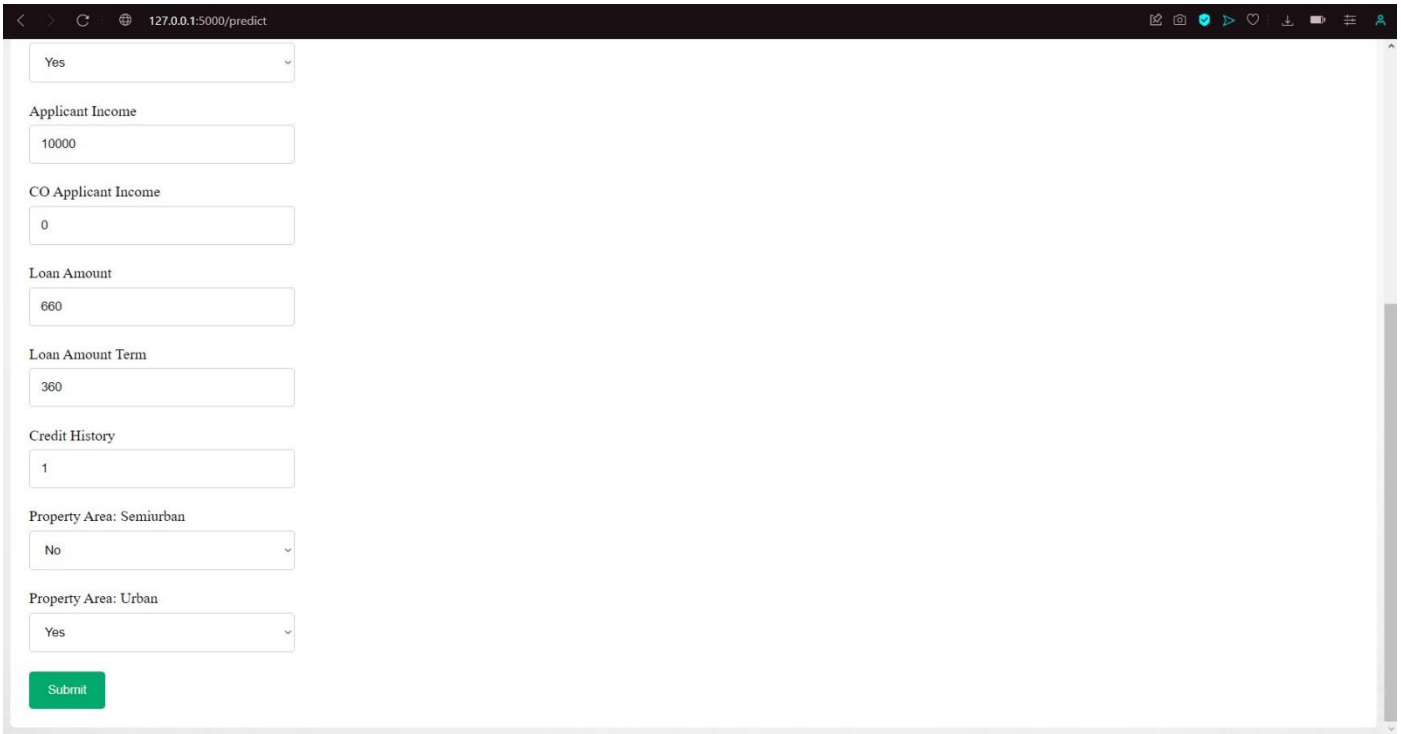
Now,Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result.



Now,when you click on click me to predict the button from the banner you will get redirected to the prediction page.

Input 1- Now, the user will give inputs to get the predicted result after clicking onto the submit button.


127.0.0.1:5000/predict
Enter your Details for Loan Approval Prediction
Gender
Male
Married
Yes
Dependents
0
Education
Graduate
Self Employed
Yes
Applicant Income
10000
CO Applicant Income
0
Loan Amount
660


127.0.0.1:5000/predict
Yes
Applicant Income
10000
CO Applicant Income
0
Loan Amount
660
Loan Amount Term
360
Credit History
1
Property Area: Semiurban
No
Property Area: Urban
Yes
Submit

Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure

The project demonstration video is uploaded in the Github.