

# Smart Lender - Applicant Credibility Prediction For Loan Approval Using Machine Learning

**Category:** Machine Learning

**Skills Required:**

Python,Python For Data Analysis,Python For Data Visualization,Data Preprocessing Techniques,Machine Learning,IBM Watson

**Project Description:**

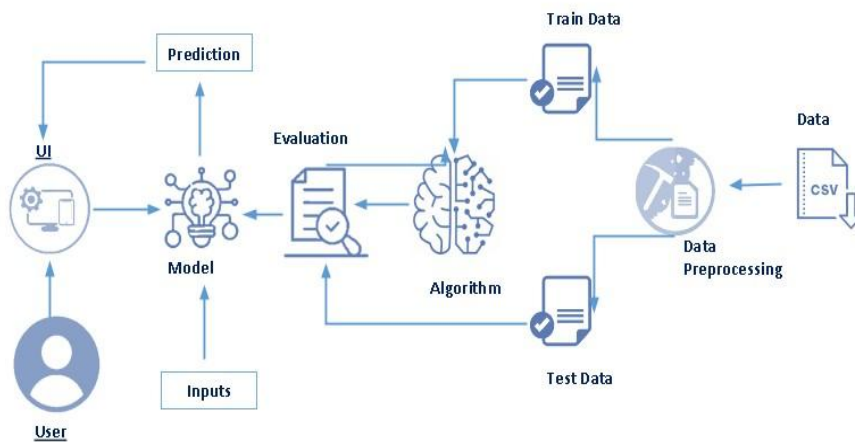
**Project Description:**

One of the most important factors which affect our country's economy and financial condition is the credit system governed by the banks. The process of bank credit risk evaluation is recognized at banks across the globe. "As we know credit risk evaluation is very crucial, there is a variety of techniques are used for risk level calculation. In addition, credit risk is one of the main functions of the banking community.

The prediction of credit defaulters is one of the difficult tasks for any bank. But by forecasting the loan defaulters, the banks definitely may reduce their loss by reducing their non-profit assets, so that recovery of approved loans can take place without any loss and it can play as the contributing parameter of the bank statement. This makes the study of this loan approval prediction important. Machine Learning techniques are very crucial and useful in the prediction of these types of data.

We will be using classification algorithms such as Decision tree, Random forest, KNN, and xgboost. We will train and test the data with these algorithms. From this best model is selected and saved in pkl format. We will be doing flask integration and IBM deployment

**Technical Architecture:**



## Pre-Requisites:

To complete this project, you must require the following software, concepts, and packages

- Anaconda navigator:

## Prior Knowledge

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

## Project Objectives

Write what are all the technical aspects that students would get if they complete this project.

1. Knowledge of Machine Learning Algorithms.
2. Knowledge of Python Language with Machine Learning
3. You'll be able to understand the problem to classify if it is a regression or a classification kind of problem.
4. You will be able to know how to pre-process/clean the data using different data pre-

processing techniques.

5. Applying different algorithms according to the dataset and based on visualization.
6. Real-Time Analysis of Project
7. Building ease of User Interface (UI)
8. Navigation of ideas towards other projects(creativity)
9. Knowledge of building ML models.
10. How to build web applications using the Flask framework.

## Project Flow

install Required Libraries.

Data Collection.

- Collect the dataset or Create the dataset

Data Preprocessing.

Import the Libraries.

- Importing the dataset.
- Understanding Data Type and Summary of features.
- Take care of missing data
- Data Visualization.
- Drop the column from DataFrame & replace the missing value.
- Splitting the Dataset into Dependent and Independent variables
- Splitting Data into Train and Test.

Model Building

- Training and testing the model

- Evaluation of Model
- Saving the Model

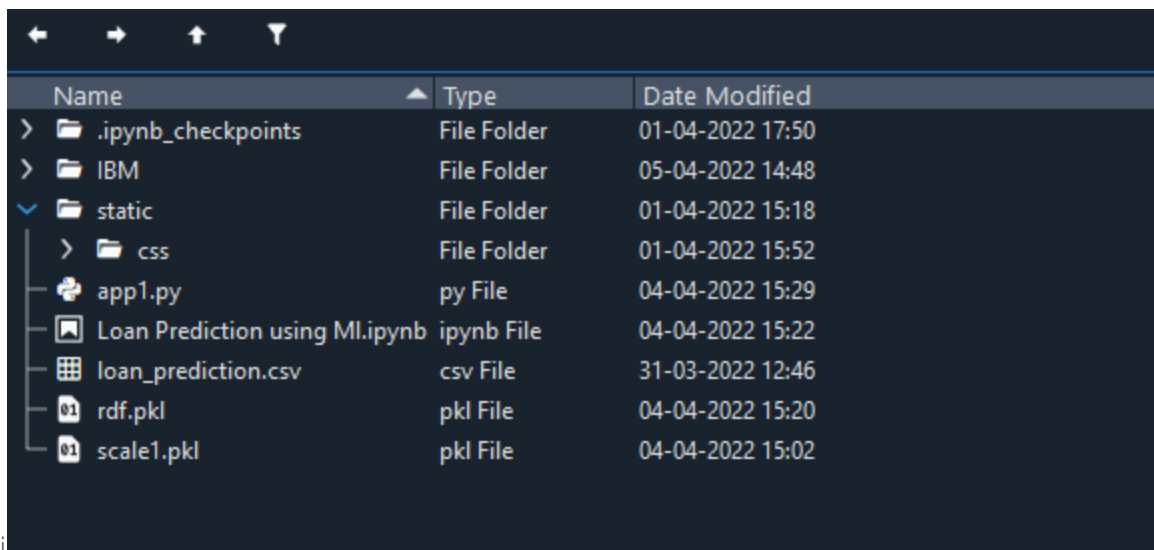
## Application Building

- Create an HTML file
- Build a Python Code

## Final UI

- Dashboard Of the flask app.

## Project Structure



Name	Type	Date Modified
> .ipynb_checkpoints	File Folder	01-04-2022 17:50
> IBM	File Folder	05-04-2022 14:48
✓ static	File Folder	01-04-2022 15:18
> css	File Folder	01-04-2022 15:52
app1.py	py File	04-04-2022 15:29
Loan Prediction using ML.ipynb	ipynb File	04-04-2022 15:22
loan_prediction.csv	csv File	31-03-2022 12:46
rdf.pkl	pkl File	04-04-2022 15:20
scale1.pkl	pkl File	04-04-2022 15:02

endpoi

- We are building a Flask Application that needs HTML pages "home.html", "index.html" stored in the templates folder and a python script app.py for server-side scripting
- The model is built in the notebook floods.ipynb
- We need the model which is saved and the saved model in this content is floods.save and transform.save
- The templates mainly used here are "home.html", "chance.html", "no chance.html", "index.html" for showcasing the UI
- The flask app is denoted as app.py
- IBM scoring endpoint consists of templates and app.py

## Data Collection

ML depends heavily on data, without data, it is impossible for an "AI" to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

### **Download The Dataset**

Duration: 0.1 Hrs

Skill Tags:

Download the dataset from the below link.

- You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository, etc.
- Here we are using a data set which you can find in the below link and you can download it from the link: [Link](#)

## **Visualizing And Analyzing The Data**

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

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

### **Importing The Libraries**

Duration: 0.1 Hrs

Skill Tags:

Import the necessary libraries as shown in the image

Import the required libraries for the model to run. The first step is usually importing the libraries that will be needed in the program.

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
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
```

## Reading The Dataset

Duration: 0.1 Hrs

Skill Tags:

- 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.

```
#importing the dataset which is in csv file
data = pd.read_csv('loan_prediction.csv')
data
```

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

614 rows x 13 columns

## Uni-Variate Analysis

Duration: 0.5 Hrs

Skill Tags:

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

- 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 a subplot.

```
#plotting the using distplot
```

```
plt.figure(figsize=(12,5))
```

```
plt.subplot(121)
```

```
sns.distplot(data['ApplicantIncome'], color='r')
```

```
plt.subplot(122)
```

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

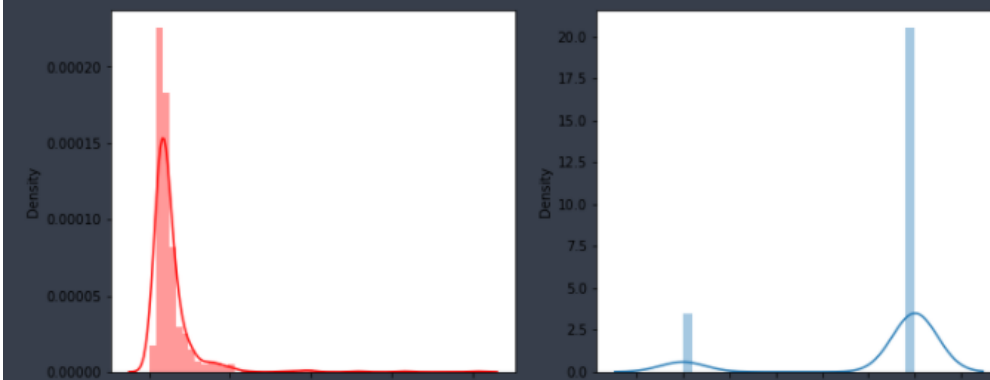
```
plt.show()
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: 'distplot' is a deprecated function and will be removed in a future version. Please adapt your code to use either 'displot' (a figure-level function with similar flexibility) or 'histplot' (an axes-level function for histograms).
```

```
warnings.warn(msg, FutureWarning)
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: 'distplot' is a deprecated function and will be removed in a future version. Please adapt your code to use either 'displot' (a figure-level function with similar flexibility) or 'histplot' (an axes-level function for histograms).
```

```
warnings.warn(msg, FutureWarning)
```



- 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 the below graph.
- From the plot we came to know, Applicants' income is skewed towards the left side, whereas credit history is categorical with 1.0 and 0.0

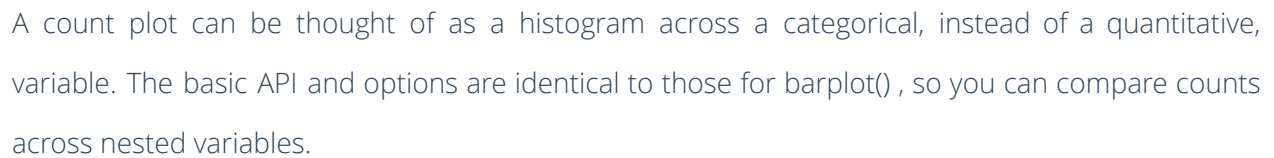
## Bivariate Analysis

Duration: 0.5 Hrs

Skill Tags:



```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```





From the above graph, we can infer the analysis such as

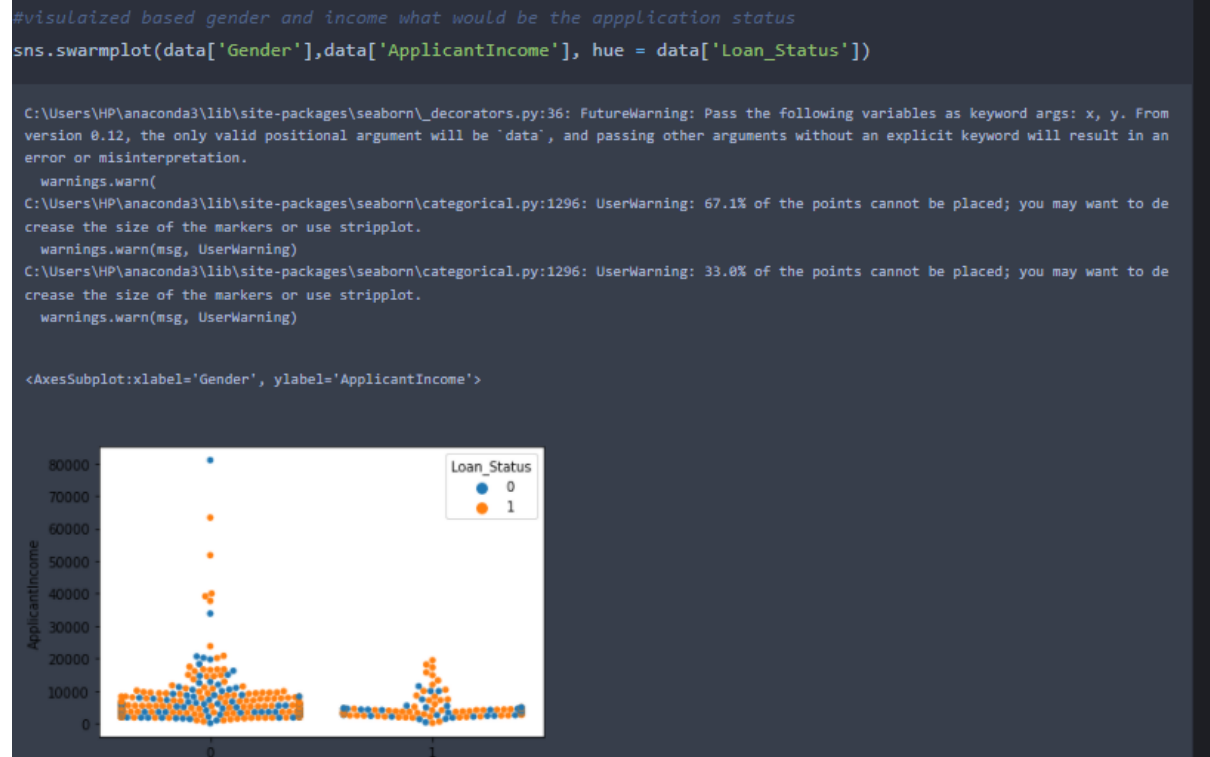
- Segmenting the gender column and married column based on bar graphs
- Segmenting the Education and Self-employed based on bar graphs, for drawing insights such as educated people are employed.
- The loan amount term is based on the property area of a person holding

## Multivariate Analysis

Duration: 0.5 Hrs

Skill Tags:

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



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

## Descriptive Analysis

Duration: 0.5 Hrs

Skill Tags:

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have 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()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.000000	564.000000
mean	5403.459283	1621.245798	146.412162	342.000000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.000000	0.000000
25%	2877.500000	0.000000	100.000000	360.000000	1.000000
50%	3812.500000	1188.500000	128.000000	360.000000	1.000000
75%	5795.000000	2297.250000	168.000000	360.000000	1.000000
max	81000.000000	41667.000000	700.000000	480.000000	1.000000

## Data Pre-Processing

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 of 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 outliers
- Scaling Techniques

- Splitting dataset into training and test set

**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.

## Checking For Null Values

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

```
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 to it. From the below image we found that there are no null values present in our dataset. So we can skip the handling of the missing values step.

```
#finding the sum of null values in each column  
data.isnull().sum()
```

```
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.

```
data['Gender'] = data['Gender'].fillna(data['Gender'].mode()[0])  
  
data['Married'] = data['Married'].fillna(data['Married'].mode()[0])  
  
#replacing + with space for filling the nan values  
data['Dependents'] = data['Dependents'].str.replace('+','')  
  
data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])  
  
data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])  
  
data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].mode()[0])  
  
data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0])  
  
data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mode()[0])
```

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

# 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.



converting string datatype variables into integer data type

```
#changing the datatype of each float column to int  
data['Gender']=data['Gender'].astype('int64')  
data['Married']=data['Married'].astype('int64')  
data['Dependents']=data['Dependents'].astype('int64')  
data['Self_Employed']=data['Self_Employed'].astype('int64')  
data['CoapplicantIncome']=data['CoapplicantIncome'].astype('int64')  
data['LoanAmount']=data['LoanAmount'].astype('int64')  
data['Loan_Amount_Term']=data['Loan_Amount_Term'].astype('int64')  
data['Credit_History']=data['Credit_History'].astype('int64')
```

## Balancing The Dataset

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 biased results, which means our model is able to predict only one class element

For Balancing the data we are using 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.

```
#Balancing the dataset by using smote
from imblearn.combine import SMOTETomek

smote = SMOTETomek(0.90)

C:\Users\HP\AppData\Roaming\Python\Python39\site-packages\imblearn\utils\_validation.py:587: FutureWarning: Pass sampling_strategy=0.9
keyword args. From version 0.9 passing these as positional arguments will result in an error
  warnings.warn(

#dividing the dataset into dependent and independent y and x respectively
y = data['Loan_Status']
x = data.drop(columns=['Loan_Status'],axis=1)

#creating a new x and y variables for the balanced set
x_bal,y_bal = smote.fit_resample(x,y)

#printing the values of y before balancing the data and after
print(y.value_counts())
print(y_bal.value_counts())

1    422
0    192
Name: Loan_Status, dtype: int64
1    351
0    308
Name: Loan_Status, dtype: int64
```

From the above picture, we can infer that previously our dataset is having 492 class 1, and 192 class items, after applying smote technique on the dataset the size has been changed for minority class.

## 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.



```
# performing feature Scaling operation using standard scaller on X part of the dataset becau
# there different type of values in the columns
sc=StandardScaler()
x_bal=sc.fit_transform(x_bal)

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

We will perform scaling only on the input values

Once the dataset is scaled, it will be converted into array and we need to convert it back to dataframe.

## 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 the x variable, df is passed by 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, and random\_state.

```
#splitting the dataset in train and test on balnmcad datasew
X_train, X_test, y_train, y_test = train_test_split(
    x_bal, y_bal, test_size=0.33, random_state=42)
```

## Model Building

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 by **Xgboost Model**

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

```
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))
```

Now let's see the performance of all the models and save the best model based on its performance.

## Decision Tree Model

A function named `decision tree` is created and train and test data are passed as the parameters. Inside the function, the `DecisionTreeClassifier` algorithm is initialized and training data is passed to the model with `.fit()` function. Test data is predicted with the `.predict()` function and saved in the new variable. For evaluating the model, a confusion matrix and classification report are done.

```
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))
```

## Random Forest Model

A function named `randomForest` is created and train and test data are passed as the parameters. Inside the function, the `RandomForestClassifier` algorithm is initialized 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 are done.

```
def randomForest(x_train, x_test, y_train, y_test):  
    rf = RandomForestClassifier()  
    rf.fit(x_train,y_train)  
    yPred = rf.predict(x_test)  
    print('***RandomForestClassifier***')  
    print('Confusion matrix')  
    print(confusion_matrix(y_test,yPred))  
    print('Classification report')  
    print(classification_report(y_test,yPred))
```

## KNN Model

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

```
def KNN(x_train, x_test, y_train, y_test):  
    knn = KNeighborsClassifier()  
    knn.fit(x_train,y_train)  
    yPred = knn.predict(x_test)  
    print('***KNeighborsClassifier***')  
    print('Confusion matrix')  
    print(confusion_matrix(y_test,yPred))  
    print('Classification report')  
    print(classification_report(y_test,yPred))
```

## Xgboost Model

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

```
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))
```

Now let's see the performance of all the models and save the best model

## Compare The Model

For comparing the above four models compareModel function is defined.

```
#printing the train accuracy and test accuracy res
RandomForest(X_train,X_test,y_train,y_test)

1.0
0.8165137614678899
```

```
#printing the train accuracy and test accuracy respectively
decisionTree(X_train,X_test,y_train,y_test)

1.0
0.7339449541284404
```

```
#printing the train accuracy and test accuracy respectively
KNN(X_train,X_test,y_train,y_test)

0.8299319727891157
0.7706422018348624
```

```
#printing the train accuracy and test accuracy respectively
XGB(X_train,X_test,y_train,y_test)

0.9478458049886621
0.8119266055045872
```

After calling the function, the results of models are displayed as output. From the four model Xgboost is performing well. From the below image, We can see the accuracy of the model. Xgboost is giving the accuracy of 94.7% with training data , 81.1% accuracy for the testing data.so we considering xgboost and deploying this model.

## Evaluating Performance Of The Model And Saving 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 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')

0.9679166666666668

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

np.mean(cv)

0.985
```

```
#saviung the model by using pickle function  
pickle.dump(model,open('rdf.pkl','wb'))
```

## Application Building

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 serverside script

## Building Html Pages

For this project create three HTML files namely

- home.html
- predict.html
- submit.html

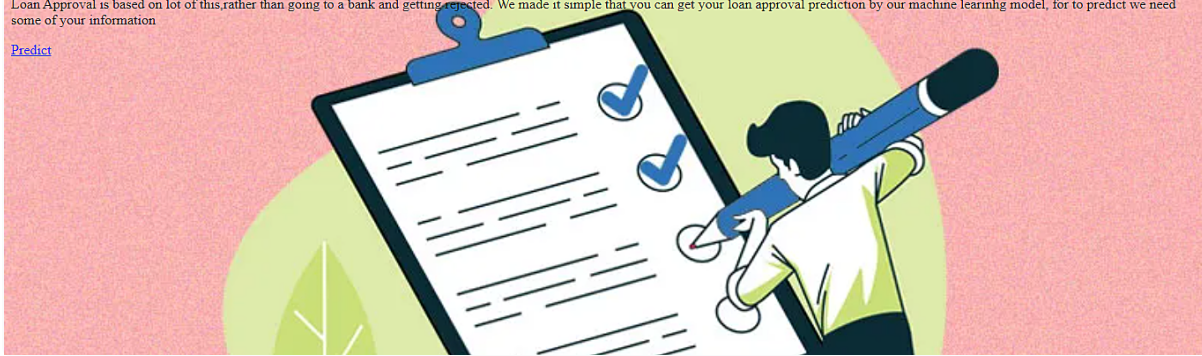
and save them in the templates folder.

Let's see how our home.html page looks like:

## Welcome to Loan Prediction

Loan Approval is based on lot of this, rather than going to a bank and getting rejected. We made it simple that you can get your loan approval prediction by our machine learning model, for to predict we need some of your information

[Predict](#)



Now when you click on predict button from top right corner you will get redirected to `predict.html`

Lets look how our `predict.html` file looks like:

**Enter your Details for Loan Approval Prediction**

Gender	<input type="text" value="Male"/>
Married	<input type="text" value="Yes"/>
Dependents	<input type="text" value="No of Dependents on you....."/>
Education	<input type="text" value="Graduate"/>
Self Employed	<input type="text" value="Yes"/>
Applicant Income	<input type="text" value="Your Income..."/>
CO Applicant Income	<input type="text" value="Your Co Applicant Income..."/>
Loan Amount	<input type="text" value="Enter the Loan Amount ..."/>
Loan Amount Term	<input type="text" value="Enter the Term Loan Amount in days ..."/>
Credit History	<input type="text" value="Enter the Your Previous Credit History ..."/>
Property Area	<input type="text" value="Urban"/>
<input type="button" value="Submit"/>	

Now when you click on submit button from left bottom corner you will get redirected to `submit.html`

Lets look how our `submit.html` file looks like:

# Laon Approval Prediction

{{result}}



## Build Python Code

Import the libraries

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

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 an argument.

```
app = Flask(__name__)
model = pickle.load(open(r'rdf.pkl', 'rb'))
scale = pickle.load(open(r'scale1.pkl', 'rb'))
```

Render HTML page:



```
@app.route('/') # rendering the html template
def home():
    return render_template('home.html')
```

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in 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 = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome',
             'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area']
    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 will Not be Approved")
    else:
        return render_template("output.html",result = "Loan will be Approved")
    # showing the prediction results in a UI
    if page == 0:
        page = 1
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using a 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)
```

## Build Python Code

Import the libraries

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

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 an argument.

```
app = Flask(__name__)  
model = pickle.load(open(r'rdf.pkl', 'rb'))  
scale = pickle.load(open(r'scale1.pkl', 'rb'))
```

Render HTML page:

```
@app.route('/') # rendering the html template  
def home():  
    return render_template('home.html')
```

Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with home.html function. Hence, when the home page of the web server is opened in 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 = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome',
             'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area']
    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 will Not be Approved")
    else:
        return render_template("output.html",result = "Loan will be Approved")
    # showing the prediction results in a UI
    if __name__ == "__main__":
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using a 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)
```

## Run The Application

## Run the application

- Open the 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.

```
base) D:\TheSmartBridge\Projects\2. DrugClassification\Drug c
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a p
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

## Laon Approval Prediction

Loan will be Approved

