

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

category:Machine learning

Project Description:

introduction:

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

Pre-Requisites:

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

- Anaconda navigator:
 - Refer to the link to download the anaconda navigator

Python packages:

- o Open anaconda prompt as administrator
- o Type "pip install numpy" and click enter.
- o Type "pip install pandas" and click enter.
- o Type "pip install scikit-learn" and click enter.
- o Type "pip install matplotlib" and click enter.
- o Type "pip install pickle-mixin" and click enter.
- o Type "pip install seaborn" and click enter.
- o Type "pip install Flask" and click enter.

Refer this [link](#) to know about the above libraries.

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

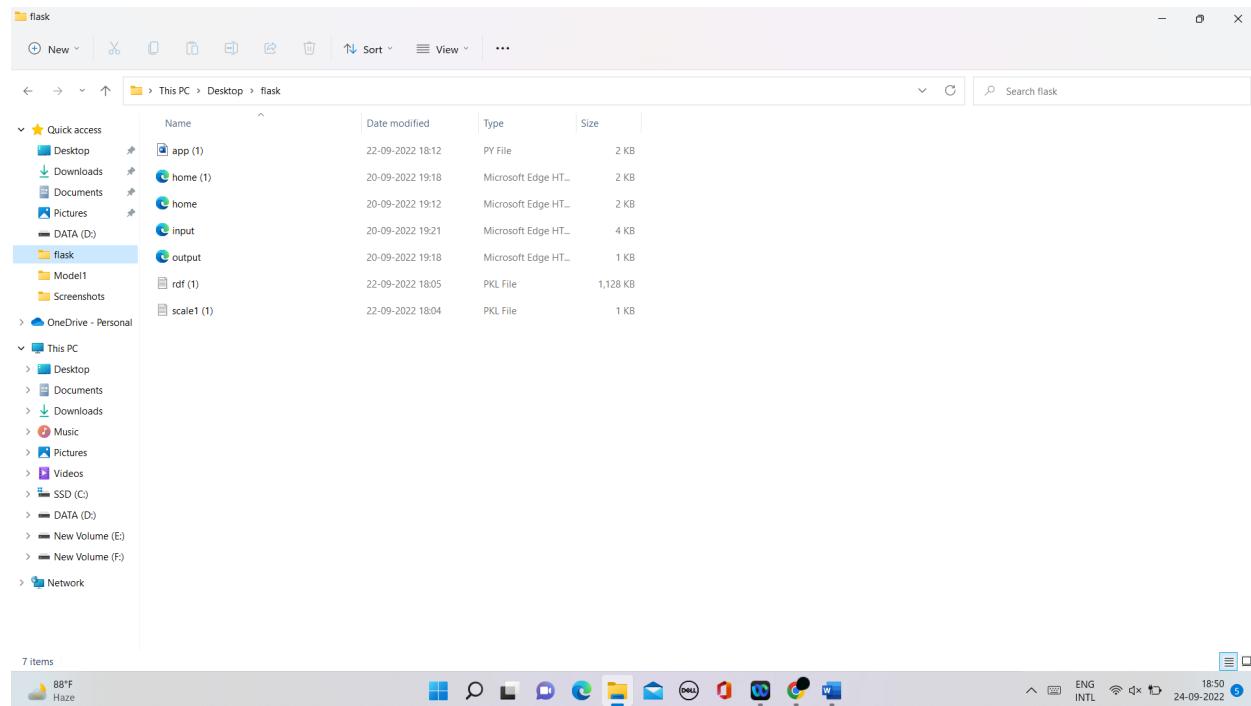
- Training and testing the model
- Evaluation of Model
- Saving the Model

Application Building

- Create an HTML file
- Build a Python Code

Final UI

Project Structure



- Dashboard Of the flask app.

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.

Importing The Libraries

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.

- **Numpy**- It is an open-source numerical Python library. It contains a multi-dimensional array and matrix data structures. It can be used to perform mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.
- **Pandas**- It is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **Seaborn**- Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Matplotlib**- Visualisation with python. It is a comprehensive library for creating static, animated, and interactive visualizations in Python

Sklearn – which contains all the modules required for model building

The screenshot shows a Jupyter Notebook window titled "dileep smartlender project - Jupyter Notebook". The URL is "localhost:8888/notebooks/dileep%20smartlender%20project.ipynb". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, Help, and a "Not Trusted" status. The Python kernel is set to "Python 3 (ipykernel)". The code cell In [1] contains imports for pandas, numpy, pickle, matplotlib.pyplot, seaborn, sklearn, and various classification models from sklearn. The code cell In [2] is titled "importing dataset" and contains the command `#importing the dataset which is in csv file` followed by `data = pd.read_csv(r"C:/Users/PRAVEEN/Downloads/loan_prediction.csv")`. The system tray at the bottom shows weather (88°F Haze), battery level (18:59 24-09-2022), and other system icons.

```
In [1]: 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
import os

In [2]: #importing the dataset which is in csv file
data = pd.read_csv(r"C:/Users/PRAVEEN/Downloads/loan_prediction.csv")
```

Importing The Dataset

- You might have your data in .csv files, .excel files
- Let's load a .csv data file into pandas using `read_csv()` function. We will need to locate the directory of the CSV file at first (it's more efficient to keep the dataset in the same directory as your program).
- If your dataset is in some other location, Then
- **Data=pd.read_csv(r"File_location/datasetname.csv")**
- If the dataset is in the same directory of your program, you can directly read it, without giving raw as r.
- Our Dataset `weatherAUS.csv` contains the following Columns
- Holiday - working day or holiday
- Temp- temperature of the day
- Rain and snow – whether it is raining or snowing on that day or not
- Weather = describes the weather conditions of the day

- Date and time = represents the exact date and time of the day
- Traffic volume – output column

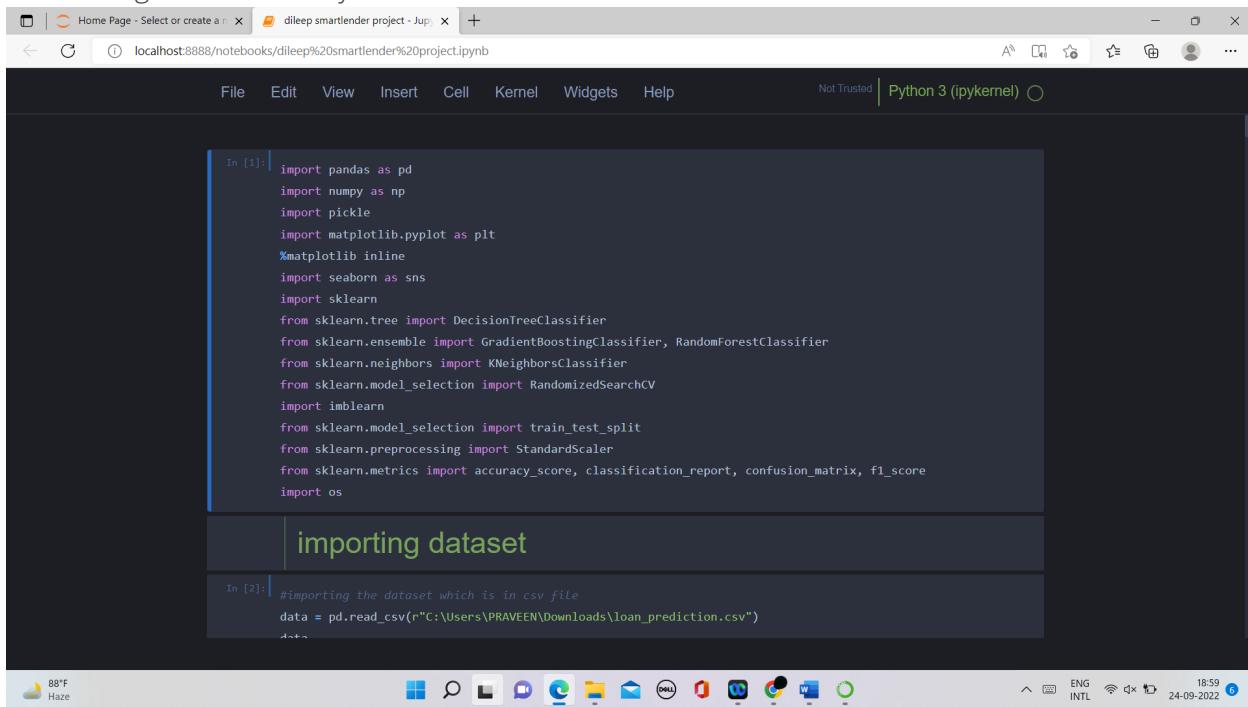
The output column to be predicted is Traffic volume. Based on the input variables we predict the volume of the traffic. The predicted output gives them a fair idea of the count of traffic

Analyse The Data

head() method is used to return top n (5 by default) rows of a DataFrame or series.

Reading 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.



The screenshot shows a Jupyter Notebook interface with the following details:

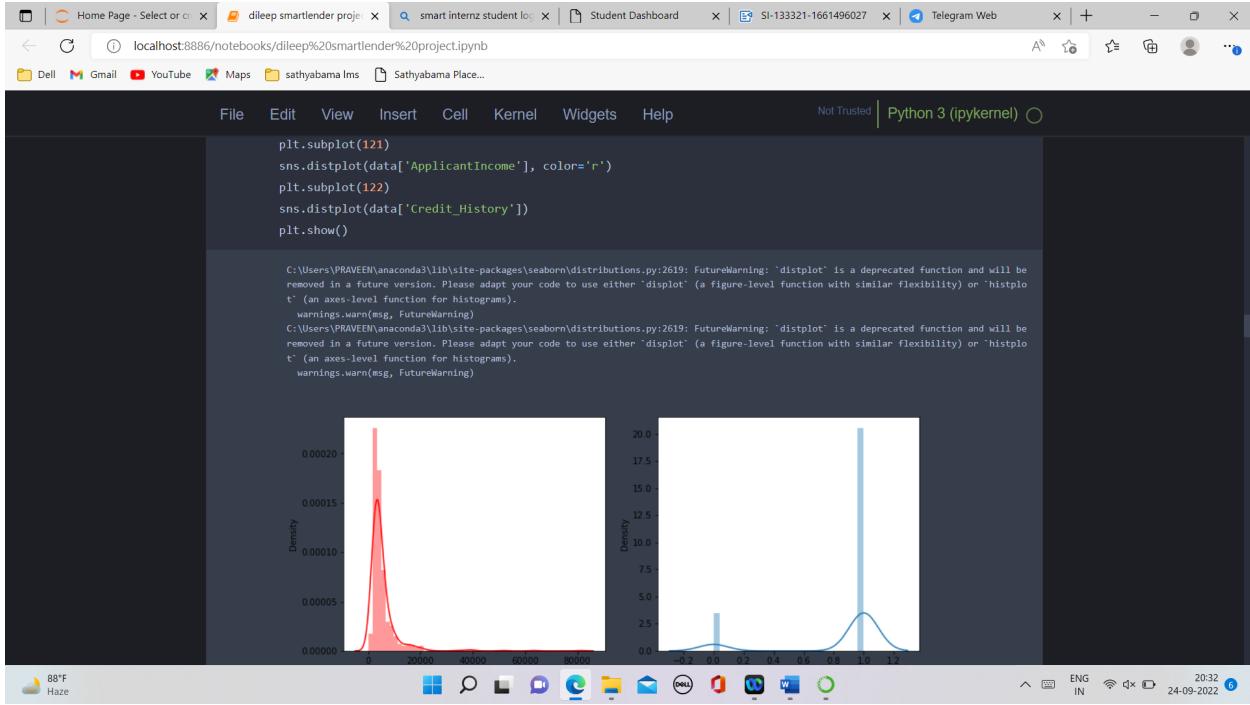
- Header:** Home Page - Select or create a new notebook | dileep smartlender project - Jupyter Notebook
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Status Bar:** Not Trusted | Python 3 (ipykernel)
- Code Cells:**
 - In [1]:** Contains imports for pandas, numpy, pickle, matplotlib.pyplot, seaborn, and various sklearn models. It also includes imports for imblearn, train_test_split, StandardScaler, accuracy_score, classification_report, confusion_matrix, f1_score, and os.
 - In [2]:** Contains the command `#importing the dataset which is in csv file` followed by `data = pd.read_csv(r"C:\Users\RAVEEN\Downloads\loan_prediction.csv")`.
- Bottom Status:** 88°F Haze, system icons, ENG INTL, 18:59, 24-09-2022

Uni-Variate Analysis

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

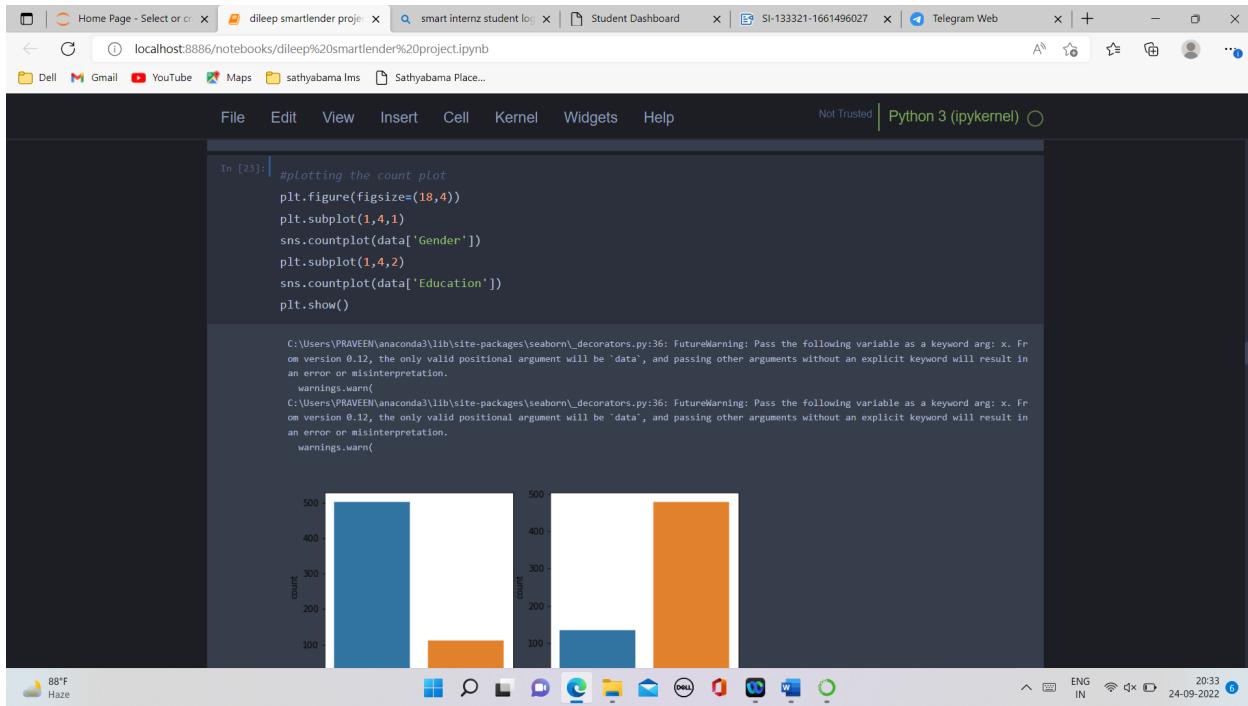
- Seaborn package provides a wonderful function `distplot`. With the help of `distplot`,

we can find the distribution of the feature. To make multiple graphs in a single plot, we use a subplot.



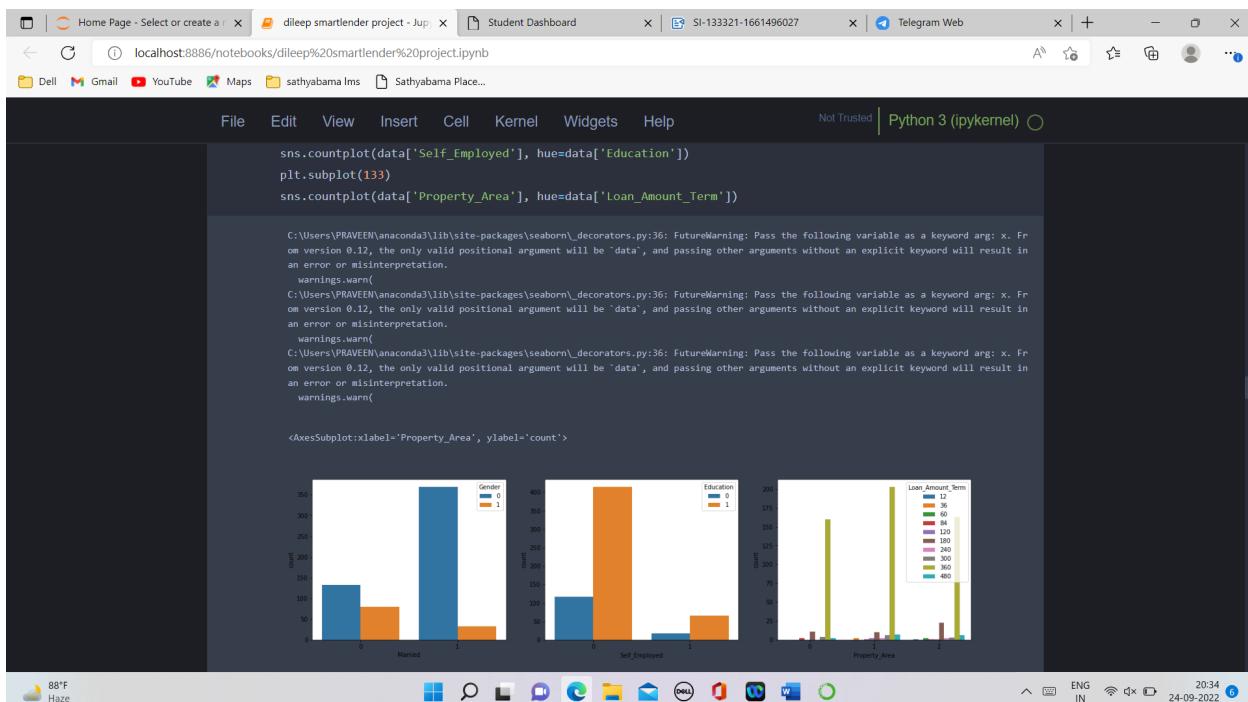
- In our dataset, we have some categorical features. With the count plot function, we are going to count the unique category in those features. We have created a dummy data frame with categorical features. With for loop and subplot, we have plotted the below graph.
- From the plot we came to know, Applicants' income is skewed towards the left side, whereas credit history is categorical with 1.0 and 0.0

Bivariate Analysis



Countplot:-

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



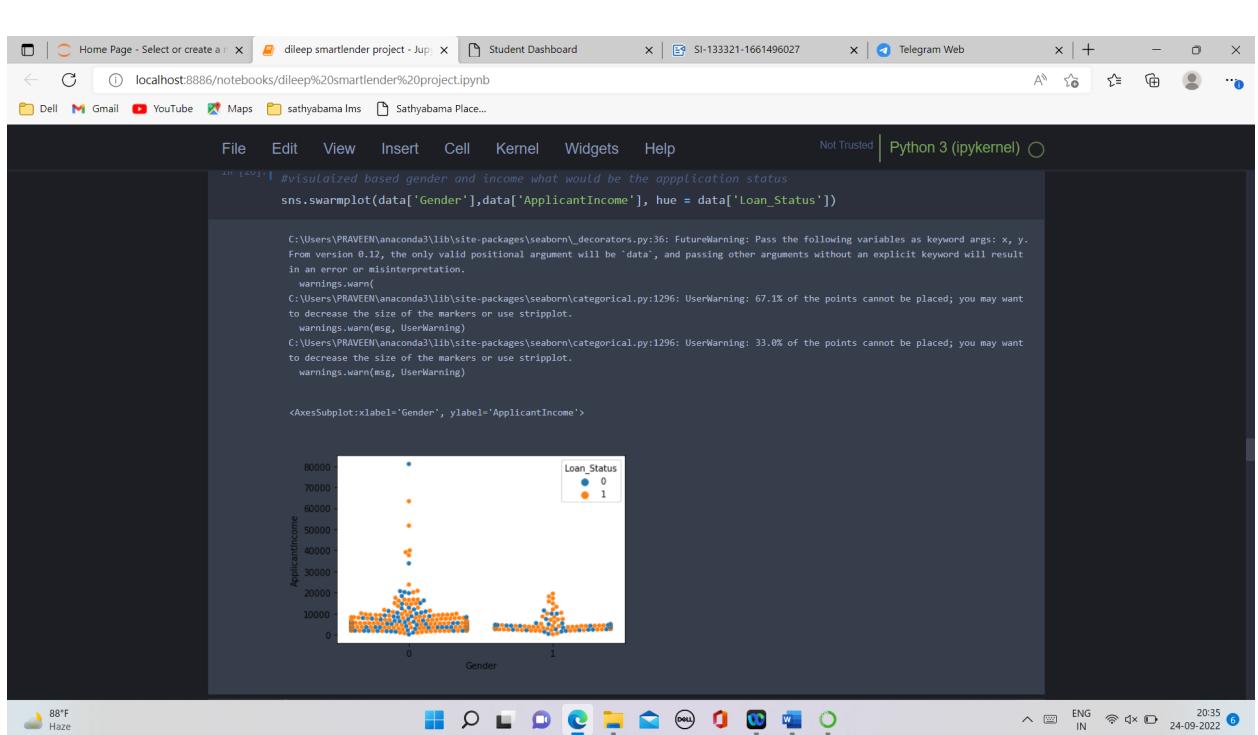
From the above graph, we can infer the analysis such as

- 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:



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

Descriptive Analysis

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.

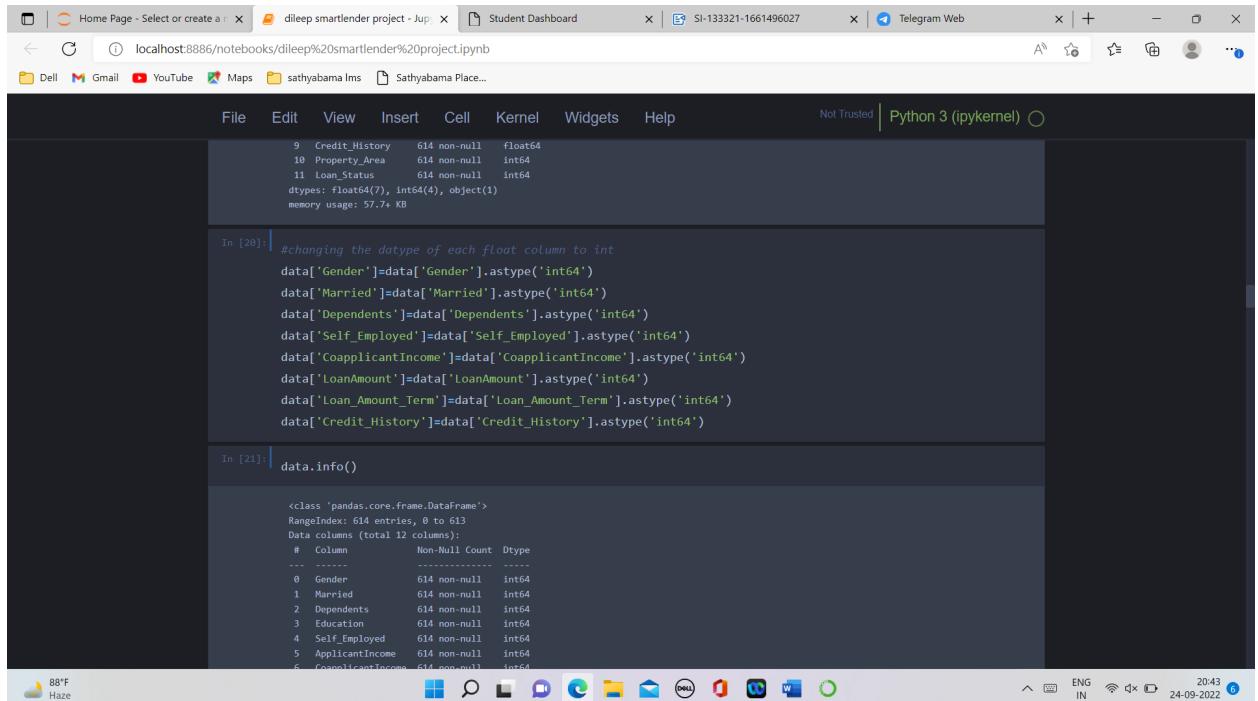
Data Pre-Processing

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.



The screenshot shows a Jupyter Notebook interface with the following content:

```
# 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')

In [21]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column      Non-Null Count Dtype  
 --- 
 0   Gender       614 non-null    int64  
 1   Married      614 non-null    int64  
 2   Dependents   614 non-null    int64  
 3   Education    614 non-null    int64  
 4   Self_Employed 614 non-null    int64  
 5   ApplicantIncome 614 non-null    int64  
 6   CoapplicantIncome 614 non-null    int64 
```

values present in our dataset. So we can skip the handling of the missing values step.

```

In [16]: data['Loan_Amount_Term'] = data['Loan_Amount_Term'].fillna(data['Loan_Amount_Term'].mode()[0])

In [17]: data['Credit_History'] = data['Credit_History'].fillna(data['Credit_History'].mode()[0])

In [18]: data.isnull().sum()

Gender      0
Married     0
Dependents  0
Education    0
Self_Employed 0
ApplicantIncome 0
CoapplicantIncome 0
LoanAmount   0
Loan_Amount_Term 0
Credit_History 0
Property_Area 0
Loan_Status   0
dtype: int64

In [19]: #getting bthe total info of the data after perfroming categorical to numericsal and replacing missssing value:
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
--- 
 0   Gender        614 non-null    float64
 1   Married       614 non-null    float64
 2   Dependents    614 non-null    int64  
 3   Education     614 non-null    int64  
 4   Self_Employed 614 non-null    int64  
 5   ApplicantIncome 614 non-null  int64  
 6   CoapplicantIncome 614 non-null int64  
 7   LoanAmount    614 non-null    int64  
 8   Loan_Amount_Term 614 non-null int64  
 9   Credit_History 614 non-null   int64  
 10  Property_Area 614 non-null   int64  
 11  Loan_Status   614 non-null   int64  
dtypes: float64(7), int64(4), object(1)
memory usage: 57.7+ KB

```

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

```

9   Credit_History 614 non-null    float64
10  Property_Area 614 non-null    int64
11  Loan_Status   614 non-null    int64
dtypes: float64(7), int64(4), object(1)
memory usage: 57.7+ KB

In [20]: #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')

In [21]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
--- 
 0   Gender        614 non-null    int64  
 1   Married       614 non-null    int64  
 2   Dependents    614 non-null    int64  
 3   Education     614 non-null    int64  
 4   Self_Employed 614 non-null    int64  
 5   ApplicantIncome 614 non-null  int64  
 6   CoapplicantIncome 614 non-null int64  
 7   LoanAmount    614 non-null    int64  
 8   Loan_Amount_Term 614 non-null int64  
 9   Credit_History 614 non-null   int64  
 10  Property_Area 614 non-null   int64  
 11  Loan_Status   614 non-null   int64  
dtypes: int64(12)
memory usage: 57.7+ KB

```

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.

The screenshot shows a Jupyter Notebook interface with multiple tabs at the top. The active tab is 'dileep smartlender project - Jupyter Notebook'. Below the tabs, there's a toolbar with various icons. The main area contains two code cells and their outputs.

Cell 20 Output:

```
9 Credit_History    614 non-null    float64
10 Property_Area   614 non-null    int64
11 Loan_Status      614 non-null    int64
dtypes: float64(7), int64(4), object(1)
memory usage: 57.7+ KB
```

Cell 21 Output:

```
#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')
```

```
In [21]: data.info()
```

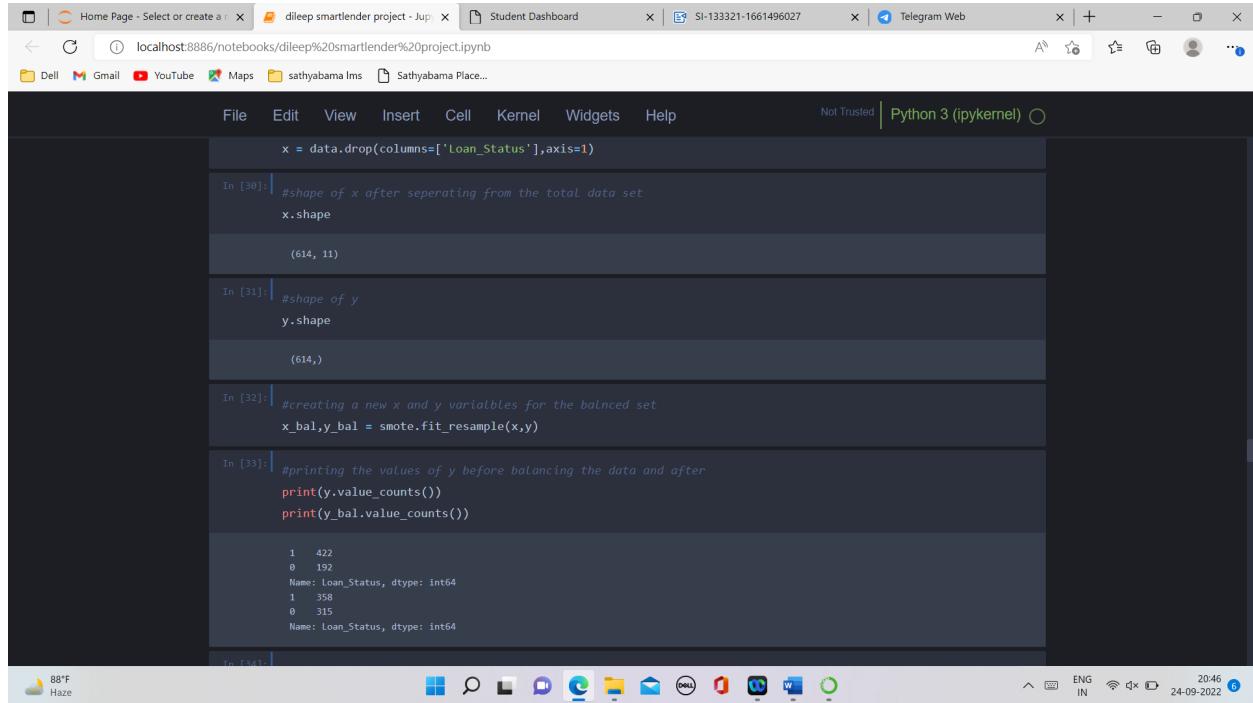
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Gender            614 non-null    int64  
 1   Married           614 non-null    int64  
 2   Dependents        614 non-null    int64  
 3   Education         614 non-null    int64  
 4   Self_Employed     614 non-null    int64  
 5   ApplicantIncome   614 non-null    int64  
 6   CoapplicantIncome 614 non-null    int64  
 7   LoanAmount        614 non-null    float64
 8   Loan_Amount_Term  614 non-null    int64  
 9   Credit_History    614 non-null    int64  
 10  Property_Area    614 non-null    int64  
 11  Loan_Status       614 non-null    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.



The screenshot shows a Jupyter Notebook interface with several code cells and their outputs:

- In [30]:
x = data.drop(columns=['Loan_Status'], axis=1)
x.shape

Output:
(614, 11)
- In [31]:
y.shape

Output:
(614,)
- In [32]:
#creating a new x and y variables for the balanced set
x_bal, y_bal = smote.fit_resample(x, y)
- In [33]:
#printing the values of y before balancing the data and after
print(y.value_counts())
print(y_bal.value_counts())

Output:
1 422
0 192
Name: Loan_Status, dtype: int64
1 358
0 315
Name: Loan_Status, dtype: int64

From the above picture, we can infer that previously our dataset is having 492 class 1, and 192 class 0 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.

The screenshot shows a Jupyter Notebook window titled "scaling the dataset". The code in the notebook performs feature scaling using StandardScaler and splits the dataset into training and testing sets. The output shows the shapes of the resulting arrays.

```
# performing feature Scaling operation using standard scaller on X part of the dataset because
# 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)

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

X_train.shape

(450, 11)

X_test.shape

(223, 11)

y_train.shape, y_test.shape
```

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.

The screenshot shows a Jupyter Notebook interface with several cells of code. The code includes importing pandas, splitting a dataset, checking shapes, and defining a Random Forest classifier function. A section titled 'model building' is highlighted.

```
In [36]: x_bal = pd.DataFrame(x_bal,columns=names)

In [37]: #splitting the dataset in train and test on balanced dataset
        X_train, X_test, y_train, y_test = train_test_split(
            x_bal, y_bal, test_size=0.33, random_state=42)

In [38]: X_train.shape

(450, 11)

In [39]: X_test.shape

(223, 11)

In [40]: y_train.shape, y_test.shape

((450,), (223,))

model building

In [41]: #importing and building the random forest model
        def RandomForest(X_train,X_test,y_train,y_test):
            model = RandomForestClassifier()
```

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.

The screenshot shows a Jupyter Notebook interface with several cells of Python code. The first cell (In [42]) contains code to build a Random Forest model and print its accuracy. The output shows 1.0 and a long numerical string. The second cell (In [43]) contains code to build a Decision Tree model and print its accuracy. The output shows 1.0 and another long numerical string. The third cell (In [44]) contains the same Decision Tree code as cell 43, and its output also shows 1.0 and a long numerical string.

```
y_tr = model.predict(X_train)
print(accuracy_score(y_tr,y_train))
yPred = model.predict(X_test)
print(accuracy_score(yPred,y_test))

In [42]: #printing the train accuracy and test accuracy respectively
RandomForest(X_train,X_test,y_train,y_test)

1.0
0.8609865470852018

In [43]: #importing and building the Decision tree model
def decisionTree(X_train,X_test,y_train,y_test):
    model = DecisionTreeClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))

In [44]: #printing the train accuracy and test accuracy respectively
decisionTree(X_train,X_test,y_train,y_test)

1.0
0.7668161434977578
```

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.

The screenshot shows a Jupyter Notebook interface with three code cells. Cell [41] imports and builds a Random Forest model. Cell [42] prints the train and test accuracy. Cell [43] imports and builds a Decision Tree model. A battery saver notification is visible on the right.

```
In [41]: #importing and building the random forest model
def RandomForest(X_train,X_test,y_train,y_test):
    model = RandomForestClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))

In [42]: #printing the train accuracy and test accuracy respectively
RandomForest(X_train,X_test,y_train,y_test)

1.0
0.869955470852018

In [43]: #importing and building the Decision tree model
def decisionTree(X_train,X_test,y_train,y_test):
    model = DecisionTreeClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
```

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.

The screenshot shows a Jupyter Notebook interface with four code cells (In [44], In [45], In [46], In [47]) and their corresponding outputs.

- In [44]:**

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

1.0
0.766816143497578
- In [45]:**

```
#importing and building the KNN model
def KNN(X_train,X_test,y_train,y_test):
    model = KNeighborsClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```
- In [46]:**

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

0.8355555555555556
0.726457399103139
- In [47]:**

```
#importing and building the Xg boost model
def XGB(X_train,X_test,y_train,y_test):
```

The notebook is running on Python 3 (ipykernel). The system tray at the bottom shows weather (88°F Haze), battery level (20:55), and network status (ENG IN).

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.

The screenshot shows a Jupyter Notebook interface with three code cells. Cell [46] contains KNN model code, printing train and test accuracy. Cell [47] contains XGBoost model code, also printing accuracy. Cell [48] contains Random Forest model code. A search bar at the top contains the text "hyper parameter tuning". The status bar at the bottom shows system information like battery level, network, and date.

```
In [46]: #printing the train accuracy and test accuracy respectively
KNN(X_train,X_test,y_train,y_test)

0.8355555555555556
0.726457399103139

In [47]: #importing and building the Xg boost model
def XGB(X_train,X_test,y_train,y_test):
    model = GradientBoostingClassifier()
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))

In [48]: #printing the train accuracy and test accuracy respectively
XGB(X_train,X_test,y_train,y_test)

0.9244444444444444
0.8385658224215246
```

Compare The Model

For comparing the above four models compareModel function is defined.

The screenshot shows two code cells for model comparison. The first cell runs Random Forest and prints accuracy values. The second cell runs a Decision Tree model and prints accuracy values. The status bar at the bottom shows system information.

```
#printing the train accuracy and test accuracy respectively
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
0.7339449541284484
```

```
#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](#)

The screenshot shows a Jupyter Notebook interface with several cells of Python code. The first cell contains code for a Random Forest classifier with specific parameters. The second cell, labeled 'In [58]:', shows the result 'bt_score' with the value 0.7933333333333333. The third cell, labeled 'In [59]:', contains a function definition for training and testing an XGBoost model. The fourth cell, labeled 'In [60]:', contains another function definition for a Random Forest classifier. The bottom of the screen shows a Windows taskbar with various icons and system status.

```
'max_features': 'log2',
'max_depth': 10,
'criterion': 'entropy'}
```

```
In [58]: bt_score
```

```
0.7933333333333333
```

```
In [59]: # training and test the xg boost model on the best parameters got from the randomized cv
def RandomForest(X_train,X_test,y_train,y_test):
    model = RandomForestClassifier(verbose= 10, n_estimators= 120, max_features= 'log2',max_depth= 10,criterion='entropy')
    model.fit(X_train,y_train)
    y_tr = model.predict(X_train)
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(X_test)
    print(accuracy_score(yPred,y_test))
```

```
In [60]: model = RandomForestClassifier(verbose= 10, n_estimators= 120, max_features= 'log2',max_depth= 10,criterion='entropy')
model.fit(X_train,y_train)
```

The screenshot shows a Jupyter Notebook interface with a data table and code. The data table displays 450 rows of numerical values. The code cell below it uses the 'pickle' module to save a model named 'rdf.pkl'. The bottom of the screen shows a Windows taskbar with various icons and system status.

203	-0.429953	0.870161	0.383532	0.627196	-0.335256	-0.331763	-0.514931	-0.582777	0.276204
198	-0.429953	0.870161	-0.687494	0.627196	-0.335256	-0.261068	-0.514931	-1.264330	-2.473804
...
71	-0.429953	0.870161	1.454559	-1.594398	-0.335256	-0.328590	0.151161	-0.544913	0.276204
106	2.325838	-1.149213	-0.687494	0.627196	-0.335256	-0.265299	-0.514931	-0.620641	0.276204
270	-0.429953	0.870161	2.525585	0.627196	-0.335256	-0.150705	-0.451985	-0.229379	0.276204
435	-0.429953	0.870161	2.525585	0.627196	2.982794	0.084654	-0.040673	-0.532291	0.276204
102	2.325838	-1.149213	-0.687494	0.627196	-0.335256	-0.181733	-0.514931	-1.239087	0.276204

```
450 rows x 11 columns
```

```
saving the model
```

```
In [63]: #saving the model by using pickle function
pickle.dump(model,open('rdf.pkl','wb'))
```

```
In [ ]:
```

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 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 serverside script

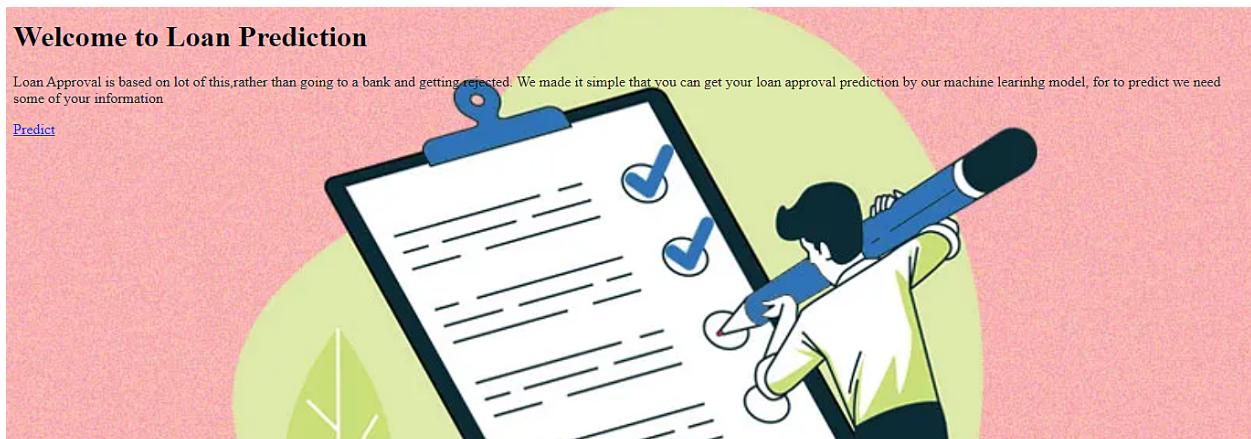
Building Html Pages

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:



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



Build Python Code

Import the libraries

```
y ×  
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
```

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 classification
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production
         environment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Loan Approval Prediction

Loan will be Approved

