INTRODUCTION

1.0 INTRODUTION

1.1 Project Overview

Predicting personal loan approval using machine learning analyses a borrower's financial data and credit history to determine the likelihood of loan approval. This can help financial institutions to make more informed decisions about which loan applications to approve and which to deny.

A loan is a sum of money that is borrowed and repaid over a period of time, typically with interest. There are various types of loans available to individuals and businesses, such as personal loans, mortgages, auto loans, student loans, business loans and many more.They are offered by banks, credit unions, and other financial institutions, and the terms of the loan, such as interest rate, repayment period, and fees, vary depending on the lender and the type of loan.

A personal loan is a type of unsecured loan that can be used for a variety of expenses such as home repairs, medical expenses, debt consolidation, and more. The loan amount, interest rate, and repayment period vary depending on the lender and the borrower's credit worthiness.To qualify for a personal loan, borrowers typically need to provide proof of income and have a good credit score.

Generally, loan prediction involves the lender looking at various background information about the applicant and deciding whether the bank should grant the loan. Parameters like credit score, loan amount, lifestyle, career, and assets are the deciding factors in getting the loan approved. If, in the past, people with parameters similar to yours have paid their dues timely, it is more likely that your loan would be granted as well. Machine learning algorithms can exploit this dependency on past experiences and comparisons with other applicants and formulate a data science problem to predict the loan status of a new applicant using similar rules. Several collections of data from past loan applicants use different features to decide the loan status. A machine learning model can look at this data, which could be static or time-series, and give a probability estimate of whether this loan will be approved. Let's look at some of these datasets.

INTRODUCTION TO MACHINE LEARNING

Machine Learning (ML) is that field of computer science with the help of which computer systems can provide sense to data in much the same way as human beings do. In simple words, ML is a type of artificial intelligence that extract patterns out of raw data by using an algorithm or method. The main focus of ML is to allow computer systems learn from experience without being explicitly programmed or human intervention. Two definitions of Machine Learning are offered. Arthur Samuel described it as: "the field of study that gives computers the ability to learn without being explicitly programmed." This is an older, informal definition.

We encounter machine learning models every day. For example, when Netflix recommends a show to you, they used a model based on what you and other users have watched to predict what you would like. When Amazon chooses a price for an item, they use a model based on how similar items have sold in the past. When your credit card company calls you because of suspicious activity, they use a model based on your past activity to recognize anomalous behavior. In the statistical context, Machine Learning is defined as an application of artificial intelligence where available information is used through algorithms to process or assist the processing of statistical data. While Machine Learning involves concepts of automation, it requires human guidance. Machine Learning involves a high level of generalization in order to get a system that performs well on yet unseen data instances.

In general, any machine learning problem can be assigned to one of two broad classifications: Supervised learning and Unsupervised learning.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.

INTRODUCTION TO UI INTERFACE

HTML stands for HyperText Markup Language. It is a standard markup language for web page creation. It allows the creation and structure of sections, paragraphs, and links using HTML elements (the building blocks of a web page) such as tags and attributes
HTML has a lot of use cases, namely:

Web development. Developers use HTML code to design how a browser displays web page elements, such as text, hyperlinks, and media files.
Internet navigation. Users can easily navigate and insert links between related pages and websites as HTML is heavily used to embed hyperlinks.
Web documentation. HTML makes it possible to organize and format documents, similarly to Microsoft Word.
CSS (Cascading Style Sheets) is used to style and layout web pages — for example, to alter the font, color, size, and spacing of your content, split it into multiple columns, or add animations and other decorative features. This module provides a gentle beginning to your path towards CSS mastery with the basics of how it works, what the syntax looks like, and how you can start using it to add styling to HTML.
Cascading Style Sheets (CSS) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.
User Interface (UI) defines the way humans interact with the information systems. In Layman's term, User Interface (UI) is a series of pages, screens, buttons, forms and other visual elements that are used to interact with the device. Every app and every website has a user interface.

2.0 LITERATURE SURVEY

2.0 LITERATURE SURVEY

Before going to replace or planning for a new system it is essential to have thorough knowledge about the existing system along with estimation or determination of how computers can be best used to make its operations more effective. System analysis is a process of collecting and interpreting fact diagnosing problems and using the information to recommend improvements to the system.

Accumulation of information about the existing system is called System Solution.
Study. Basically, system analysis is about understanding situation, not solving
problems.

2.1 EXISTING PROBLEM:

Bank employees check the details of applicant manually and give the loan to eligible applicant. Checking the details of all applicants takes lot of time. The artificial neural network model for predict the credit risk of a bank. The Feed- forward back propagation neural network is used to forecast the credit default. The method in which two or more classifiers are combined together to produce a ensemble model for the better prediction. They used the bagging and boosting techniques and then used random forest technique. The process of classifiers is to improve the performance of the data and it gives better efficiency. In this work, the authors describe various ensemble techniques for binary classification and also for multi class classification. The new technique that is described by the authors for ensemble is COB which gives effective performance of classification but it also compromised with noise and outlier data of classification. Finally they concluded that the ensemble based algorithm improves the results for training data set.

Disadvantages of the existing problem:

Checking details of all applicants consumes lot of time and efforts. There is chances of human error may occur due checking all details manually. There is possibility of assigning loan to ineligible applicant.

2.2. PROPOSED SOLUTION:

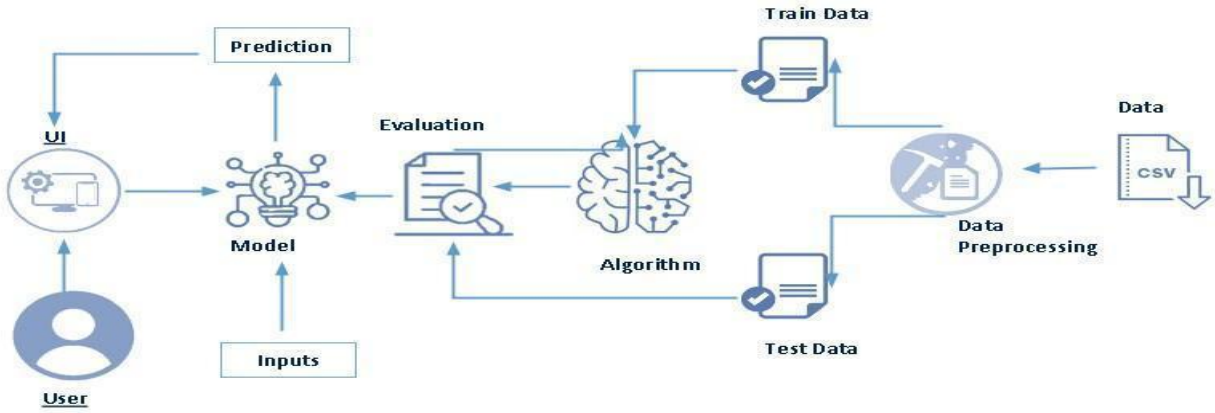
To deal with the problem, we developed automatic loan prediction using machine learning techniques. We will train the machine with previous dataset. so machine can analyse and understand the process . Then machine will check for eligible applicant and give us result.

Advantages

- Time period for loan sanctioning will be reduced
- Whole process will be automated , so human error will be avoided
- Eligible applicant will be sanctioned loan without any delay.

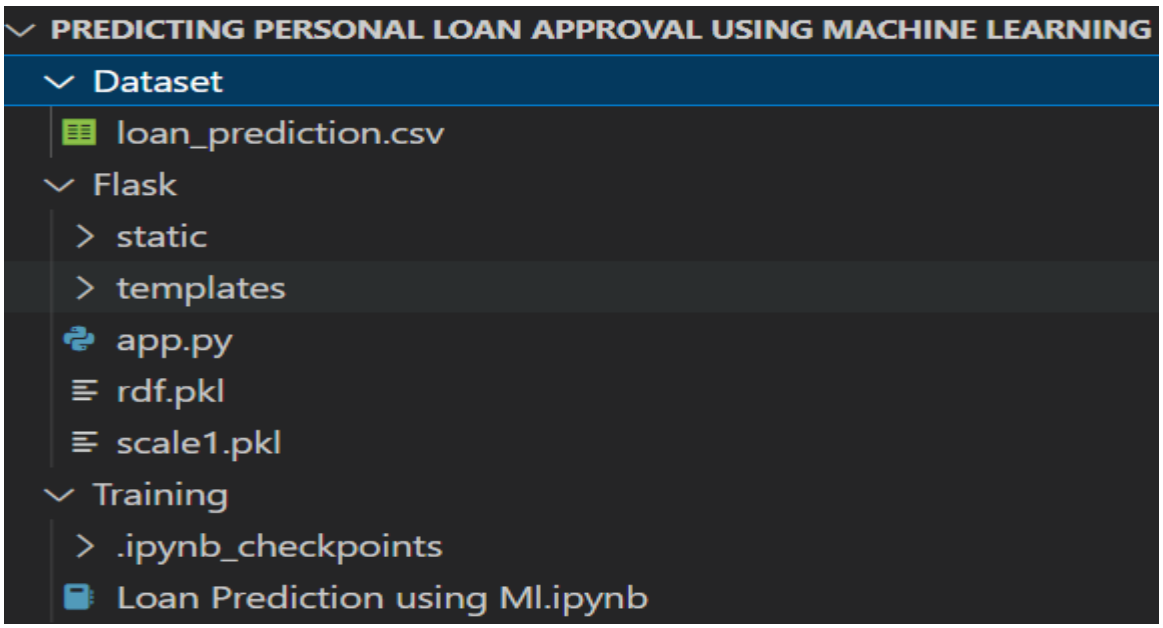
3.0 THEORITICAL ANALYSIS

3.1 BLOCK DIAGRAM



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.

3.1 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.

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

Activity 3.1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualization style as five thirty eight.

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.

```
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.

```
#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 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.

Activity3. 1.2: 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.

```
#changing the datype 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')
```

ACTIVITY 3.1.3 :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

```
#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 varialbles for the balnced 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 had 492 class 1, and 192 class items, after applying smote technique on the dataset the size has been changed for minority class.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in any data analytics project. It involves the initial analysis and visualization of the data to gain insights and understand the underlying patterns and relationships.

Descriptive Statistical

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

```
data.describe()
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

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.

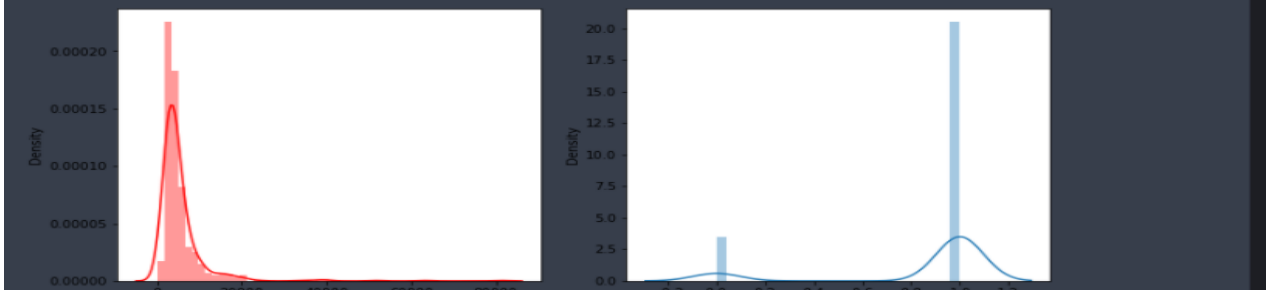
Activity3. 2.1: Univariate analysis

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.

```
#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 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:-

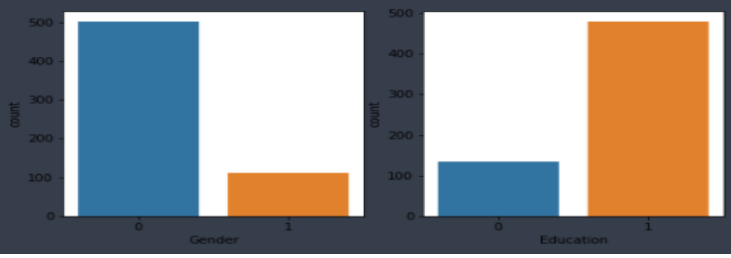
A count plot can be thought of as a histogram across a categorical, instead of quantitative, variable. The basic API and options are identical to those for barplot() , so you can compare counts across nested variables.

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 .

Activity3. 2.2: Bivariate analysis

```
#plotting the count plot
plt.figure(figsize=(18,4))
plt.subplot(1,4,1)
sns.countplot(data['Gender'])
plt.subplot(1,4,2)
sns.countplot(data['Education'])
plt.show()
```

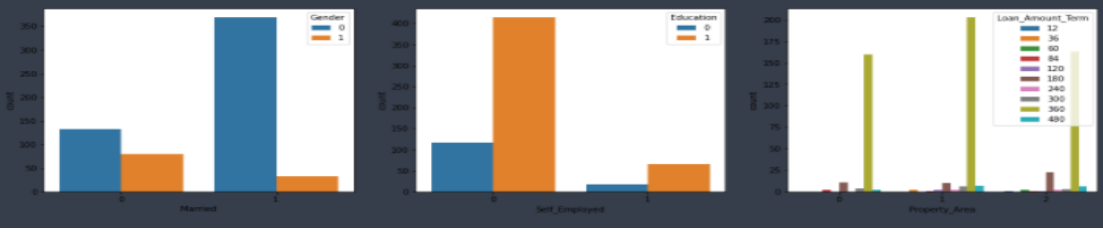
```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From ve
rsion 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an e
rror or misinterpretation.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



```
#visualising two columns against each other
plt.figure(figsize=(20,5))
plt.subplot(131)
sns.countplot(data['Married'], hue=data['Gender'])
plt.subplot(132)
sns.countplot(data['Self_Employed'], hue=data['Education'])
plt.subplot(133)
sns.countplot(data['Property_Area'], hue=data['Loan_Amount_Term'])
```

```
  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 v
ersion 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(

<AxesSubplot:xlabel='Property_Area', ylabel='count'>
```



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

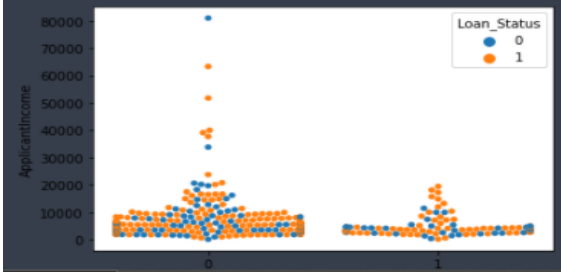
Activity 3.2.3: Multivariate analysis

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

```
#visualized based gender and income what would be the appplication status
sns.swarmplot(data['Gender'],data['ApplicantIncome'], hue = data['Loan_Status'])
```

```
C:\Users\HP\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variables as keyword args: x, y. 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\categorical.py:1296: UserWarning: 67.1% of the points cannot be placed; you may want to de
crease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
C:\Users\HP\anaconda3\lib\site-packages\seaborn\categorical.py:1296: UserWarning: 33.0% of the points cannot be placed; you may want to de
crease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)

<AxesSubplot:xlabel='Gender', ylabel='ApplicantIncome'>
```



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.

```
# perfroming feature Scaling op[eration 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 an array and we need to convert it back to a 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 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 sk learn. As parameters, we are passing x, y, test_size, random_state.

```
#splitting the dataset in train and test on balnmced 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

Model building is the process of developing a machine learning model to solve a specific problem or make predictions based on data. It involves selecting an appropriate algorithm, training the model on a dataset, and evaluating its performance.

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 3.3.1: Decision tree model

A function named decisionTree is created and train and test data are passed as the parameters. Inside the function, Decision Tre e Classifier algorithm is initialized 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.

```
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 3.3.2: 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.

```
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))
```

Activity 3.3.3: 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.

```
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))
```

Activity 3.3.4: 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.

```
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))
```

Activity3.3..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.

ANN

```
# Importing the Keras libraries and packages
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
[226] # Initialising the ANN
      classifier = Sequential()
```

```
[227] # Adding the input layer and the first hidden layer
      classifier.add(Dense(units=100, activation='relu', input_dim=11))
```

```
# Adding the second hidden layer
classifier.add(Dense(units=50, activation='relu'))
```

```
[229] # Adding the output layer
      classifier.add(Dense(units=1, activation='sigmoid'))
```

```
[230] # Compiling the ANN
      classifier.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

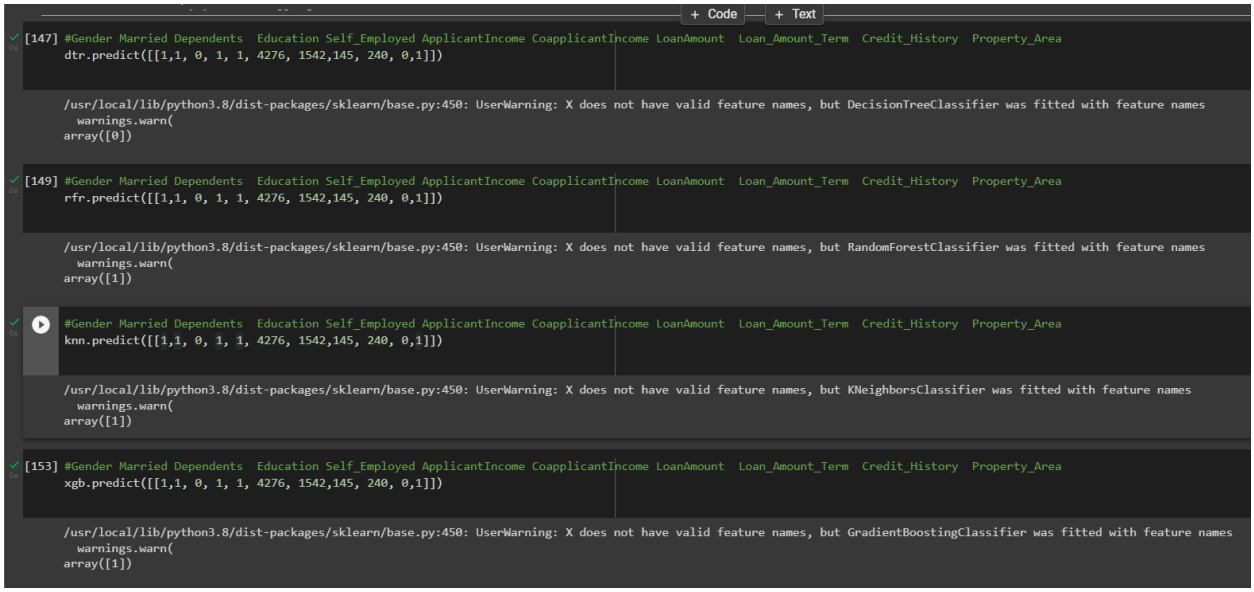
```
# Fitting the ANN to the Training set
model_history = classifier.fit(X_train, y_train, batch_size=100, validation_split=0.2, epochs=100)
```

```
Epoch 72/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4286 - accuracy: 0.7824 - val_loss: 0.7493 - val_accuracy: 0.6703
Epoch 73/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4252 - accuracy: 0.8017 - val_loss: 0.7592 - val_accuracy: 0.6703
Epoch 74/100
4/4 [==============================] - 0s 12ms/step - loss: 0.4244 - accuracy: 0.8017 - val_loss: 0.7638 - val_accuracy: 0.6703
Epoch 75/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4222 - accuracy: 0.7989 - val_loss: 0.7577 - val_accuracy: 0.6703
Epoch 76/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4200 - accuracy: 0.7934 - val_loss: 0.7586 - val_accuracy: 0.6703
Epoch 77/100
4/4 [==============================] - 0s 11ms/step - loss: 0.4181 - accuracy: 0.7989 - val_loss: 0.7657 - val_accuracy: 0.6703
```

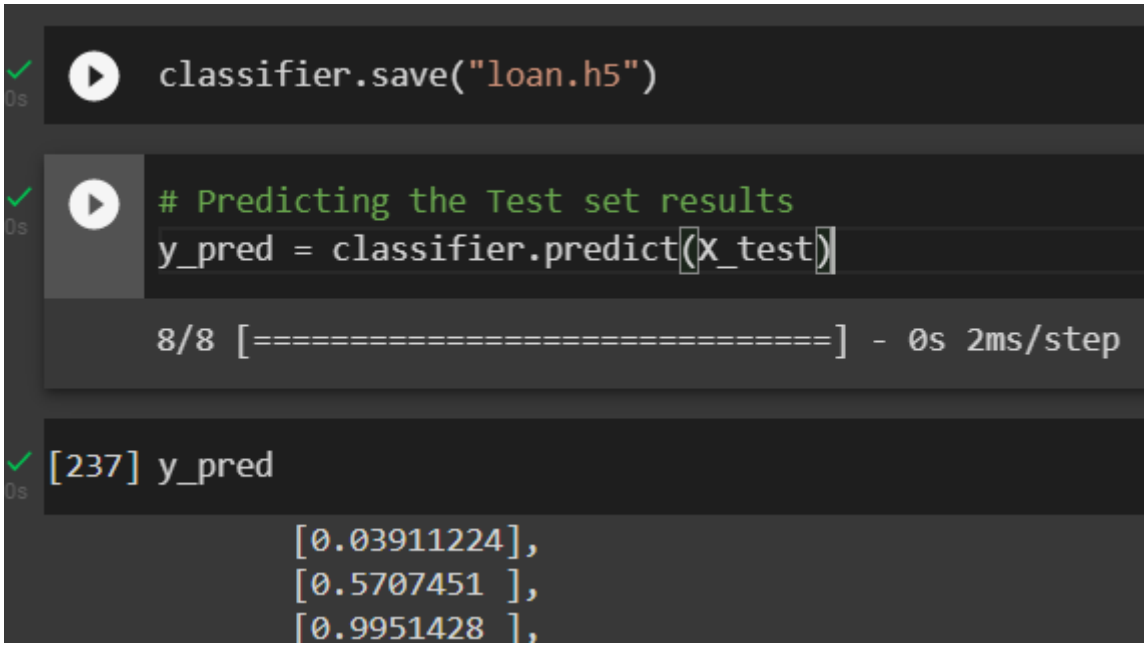
```
Epoch 95/100
4/4 [==============================] - 0s 17ms/step - loss: 0.3877 - accuracy: 0.8292 - val_loss: 0.8256 - val_accuracy: 0.6593
Epoch 96/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3858 - accuracy: 0.8292 - val_loss: 0.8253 - val_accuracy: 0.6593
Epoch 97/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3858 - accuracy: 0.8347 - val_loss: 0.8260 - val_accuracy: 0.6593
Epoch 98/100
4/4 [==============================] - 0s 12ms/step - loss: 0.3841 - accuracy: 0.8430 - val_loss: 0.8382 - val_accuracy: 0.6593
Epoch 99/100
4/4 [==============================] - 0s 12ms/step - loss: 0.3817 - accuracy: 0.8347 - val_loss: 0.8357 - val_accuracy: 0.6593
Epoch 100/100
4/4 [==============================] - 0s 11ms/step - loss: 0.3805 - accuracy: 0.8430 - val_loss: 0.8368 - val_accuracy: 0.6593
```

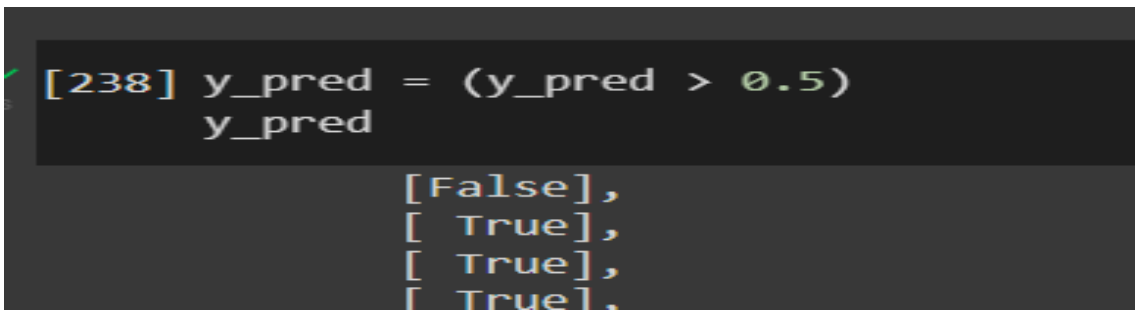

```
Epoch 95/100
4/4 [==============================] - 0s 17ms/step - loss: 0.3877 - accuracy: 0.8292 - val_loss: 0.8256 - val_accuracy: 0.6593
Epoch 96/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3858 - accuracy: 0.8292 - val_loss: 0.8253 - val_accuracy: 0.6593
Epoch 97/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3858 - accuracy: 0.8347 - val_loss: 0.8260 - val_accuracy: 0.6593
Epoch 98/100
4/4 [==============================] - 0s 12ms/step - loss: 0.3841 - accuracy: 0.8430 - val_loss: 0.8382 - val_accuracy: 0.6593
Epoch 99/100
4/4 [==============================] - 0s 12ms/step - loss: 0.3817 - accuracy: 0.8347 - val_loss: 0.8357 - val_accuracy: 0.6593
Epoch 100/100
4/4 [==============================] - 0s 11ms/step - loss: 0.3805 - accuracy: 0.8430 - val_loss: 0.8368 - val_accuracy: 0.6593
```

Testing The Model



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





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.

```
[244] def predict_exit(sample_value):

      # Convert list to numpy array
        sample_value = np.array(sample_value)

      # Reshape because sample_value contains only 1 record
        sample_value = sample_value.reshape(1, -1)

      # Feature Scaling
        sample_value = sc.transform(sample_value)

        return classifier.predict(sample_value)
```

```
# Predictions
# Value order 'CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','France','Germany','Spain','Female',Male'.
sample_value = [[1,1, 0, 1, 1, 4276, 1542,145, 240, 0,1]]
if predict_exit(sample_value)>0.5:
    print('Prediction: High chance of Loan Approval!')
else:
    print('Prediction: Low chance Loan Approval.')
```

```
1/1 [==============================] - 0s 18ms/step
Prediction: Low chance Loan Approval.
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

```
# Predictions
# Value order 'CreditScore','Age','Tenure','Balance','NumOfProducts','HasCrCard','IsActiveMember','EstimatedSalary','France','Germany','Spain','Female',Male'.
sample_value = [[1,0, 1, 1, 1, 45, 14,45, 240, 1,1]]
if predict_exit(sample_value)>0.5:
    print('Prediction: High chance of Loan Approval!')
else:
    print('Prediction: Low chance of Loan Approval.')
```

```
1/1 [==============================] - 0s 50ms/step
Prediction: High chance of Loan Approval!
/usr/local/lib/python3.8/dist-packages/sklearn/base.py:450: UserWarning: X does not have valid feature names, but StandardScaler was fitted with feature names
  warnings.warn(
```

Performance Testing & Hyperparameter Tuning

Performance testing and hyperparameter tuning are critical steps in building an effective machine learning model. These steps are used to optimize the performance of the model and ensure that it is accurate and reliable.

Performance testing involves evaluating the performance of the model on a separate test dataset, which has not been used in the model training process. The goal of performance testing is to determine how well the model performs on new data, and to identify any issues or areas for improvement. Common metrics used to evaluate the performance of a machine learning model include accuracy, precision, recall, F1 score, and area under the curve (AUC).

Hyperparameter tuning involves fine-tuning the parameters of the machine learning algorithm to optimize its performance. Hyperparameters are parameters that are set before the model training process begins, and they can have a significant impact on the performance of the model. Examples of hyperparameters include the learning rate, regularization parameter, and number of hidden layers in a neural network.

There are several techniques for hyperparameter tuning, including grid search, random search, and Bayesian optimization. Grid search involves trying out all possible combinations of hyperparameters, while random search involves randomly sampling hyperparameters from a defined range. Bayesian optimization uses probabilistic models to estimate the performance of different hyper parameter configurations.

When conducting performance testing and hyperparameter tuning, it's important to use best practices such as cross-validation to ensure that the results are reliable and robust. It's also important to be aware of the trade-off between model complexity and model performance, and to choose the simplest model that adequately solves the problem at hand.

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.

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

Compare the model

```
def compareModel(X_train,X_test,y_train,y_test):
  decisionTree(X_train,X_test,y_train,y_test)
  print('-'*100)
  RandomForest(X_train,X_test,y_train,y_test)
  print('-'*100)
  XGB(X_train,X_test,y_train,y_test)
  print('-'*100)
  KNN(X_train,X_test,y_train,y_test)
  print('-'*100)
```

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

1.0
0.7822222222222223
Decision Tree
Confusion_Matrix
[[83 24]
 [25 93]]
Classification Report
              precision    recall  f1-score   support

           0       0.77      0.78      0.77       107
           1       0.79      0.79      0.79       118

    accuracy                           0.78       225
   macro avg       0.78      0.78      0.78       225
weighted avg       0.78      0.78      0.78       225

----------
```

```
----------
1.0
0.8088888888888889
Random Forest
Confusion_Matrix
[[ 78  29]
 [ 14 104]]
Classification Report
              precision    recall  f1-score   support

           0       0.85      0.73      0.78       107
           1       0.78      0.88      0.83       118

    accuracy                           0.81       225
   macro avg       0.81      0.81      0.81       225
weighted avg       0.81      0.81      0.81       225

----------
```

```
0.933920704845815
0.8222222222222222
XGBoost
Confusion_Matrix
[[ 78  29]
 [ 11 107]]
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.73      0.80       107
           1       0.79      0.91      0.84       118

    accuracy                           0.82       225
   macro avg       0.83      0.82      0.82       225
weighted avg       0.83      0.82      0.82       225
```

```
0.7665198237885462
0.6666666666666666
KNN
Confusion_Matrix
[[60 47]
 [28 90]]
Classification Report
              precision    recall  f1-score   support

           0       0.68      0.56      0.62       107
           1       0.66      0.76      0.71       118

    accuracy                           0.67       225
   macro avg       0.67      0.66      0.66       225
weighted avg       0.67      0.67      0.66       225
```

```
yPred = classifier.predict(X_test)
print(accuracy_score(y_pred,y_test))
print("ANN Model")
print("Confusion_Matrix")
print(confusion_matrix(y_test,y_pred))
print("Classification Report")
print(classification_report(y_test,y_pred))
```

```
8/8 [==============================] - 0s 4ms/step
0.6844444444444444
ANN Model
Confusion_Matrix
[[63 44]
 [27 91]]
Classification Report
              precision    recall  f1-score   support

           0       0.70      0.59      0.64       107
           1       0.67      0.77      0.72       118

    accuracy                           0.68       225
   macro avg       0.69      0.68      0.68       225
weighted avg       0.69      0.68      0.68       225
```

```
0.933920704845815
0.8222222222222222
XGBoost
Confusion_Matrix
[[ 78  29]
 [ 11 107]]
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.73      0.80       107
           1       0.79      0.91      0.84       118

    accuracy                           0.82       225
   macro avg       0.83      0.82      0.82       225
weighted avg       0.83      0.82      0.82       225
```

After calling the function, the results of models are displayed as output. From the five models Xgboost is performing well. From the below image, We can see the accuracy of the model. Xgboost is giving the accuracy of 93.39% with training data , 82.2% accuracy for the testing data.

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().

```
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
```

```
0.9691629955947136
0.8222222222222222
Random Forest
Confusion_Matrix
[[ 77  30]
 [ 10 108]]
Classification Report
              precision    recall  f1-score   support

           0       0.89      0.72      0.79       107
           1       0.78      0.92      0.84       118

    accuracy                           0.82       225
   macro avg       0.83      0.82      0.82       225
weighted avg       0.83      0.82      0.82       225

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:    0.0s
```

Model Deployment

Model deployment is the process of integrating a machine learning model into a production environment, where it can make predictions on new data. This involves taking the trained model and making it available to other systems or applications, such as a web application, mobile app, or IOT devices.

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.

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

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 3.4.1: Building Html Pages:

For this project create two HTML files namely

home.html

predict.html

and save them in the templates folder.

Activity 3.4.2: 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 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 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 = ['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 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 3.4.3: Run the web application

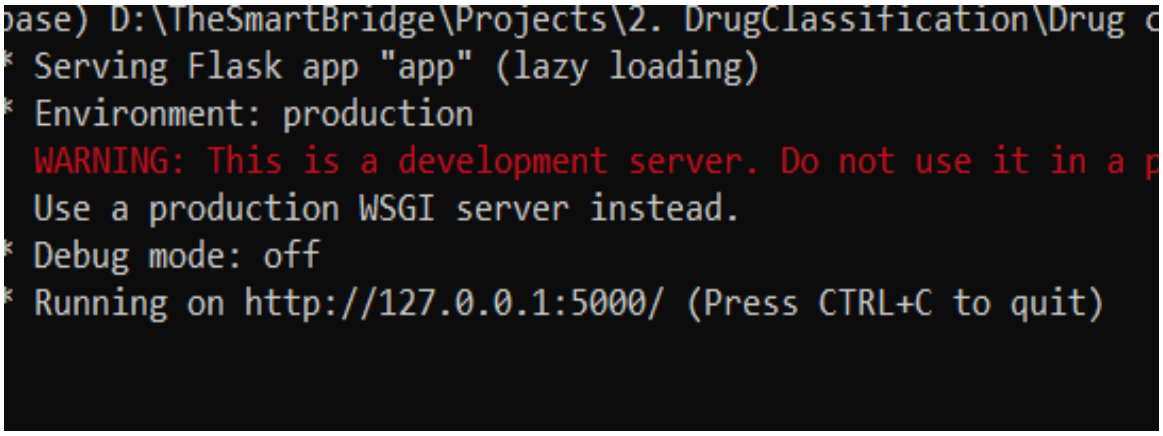
Open anaconda prompt from the start menu

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.



4.0 SYSTEM SPECIFICATIONS

4.1. SOFTWARE REQUIREMENTS

Language Tools used: MS-EXCEL

- ♣ Anaconda Navigator
- ♣ Jupyter Notebook

Operating System : Windows or linux

Language : python,html

Python ide :python 2.7x and above

Documentation too l: Ms-word 2007

4.2. MINIMUM HARDWARE REQUIREMENTS

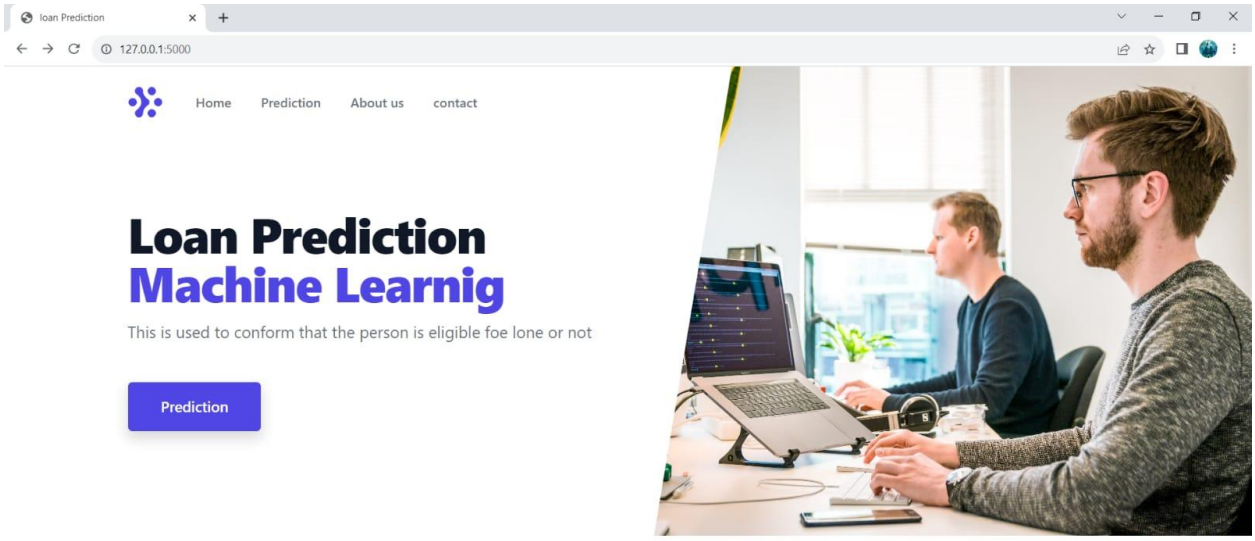
Hard Disk : 500GB minimum

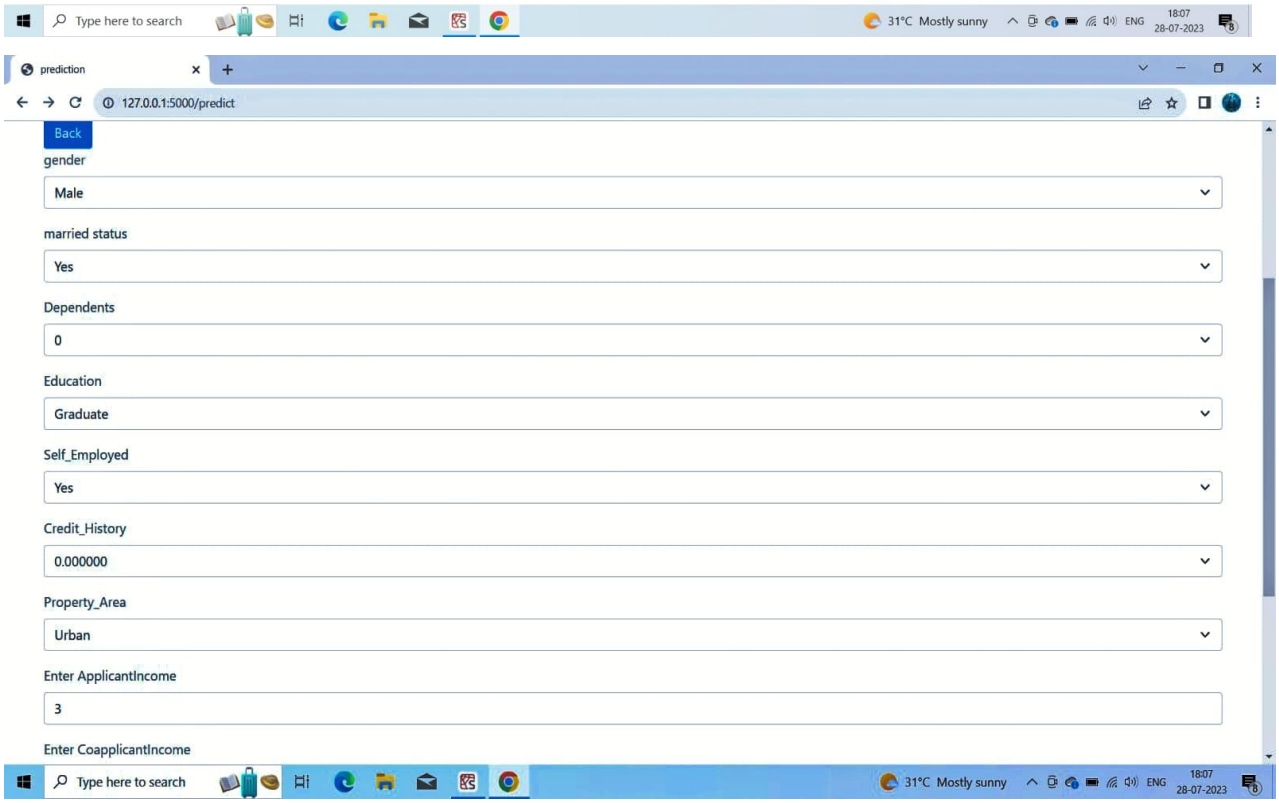
PROCESSOR : Intel i3 and above

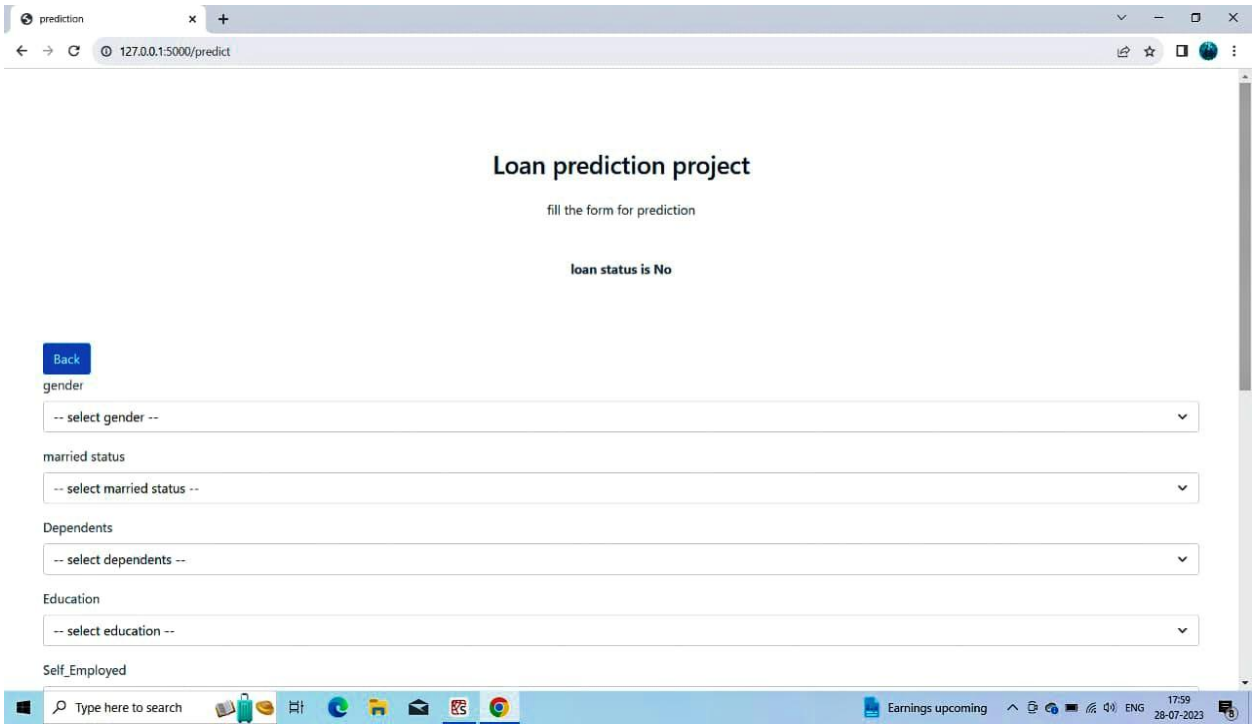
Ram : 4 GB and Higher

RESULT :

INPUT & OUTPUT







PREDICTING THE OUTCOMES:

Using decision tree algorithm, the outcomes of all applicant can be stored in any file. Algorithm:

1. Import all the required python modules
2. Import the database for both TESTING and TRAINING.
3. Check any NULLVALUES are exists
4. If NULLVALUES exits ,fill the table with corresponding coding
5. Exploratory Data Analysis for all ATTRIBUTES from the table
6. Plot all graphs using MATPLOTLIB module
7. Build the DECISIONTREE MODEL for the coding
8. Send that output to CSV FILE

CONCLUSION

From a proper analysis of positive points and constraints on the member, it can be safely concluded that the product is a considerably productive member. This use is working duly and meeting to all Banker requisites. This member can be freely plugged in numerous other systems. There have been mathematics cases of computer glitches, violations in content and most important weight of features is fixed in automated prophecy system, so in the near future the so – called software could be made more secure, trustworthy and dynamic weight conformation. In near future this module of prophecy can be integrated with the module of automated processing system. The system is trained on old training dataset in future software can be made resembling that new testing date should also take part in training data after some fix time.