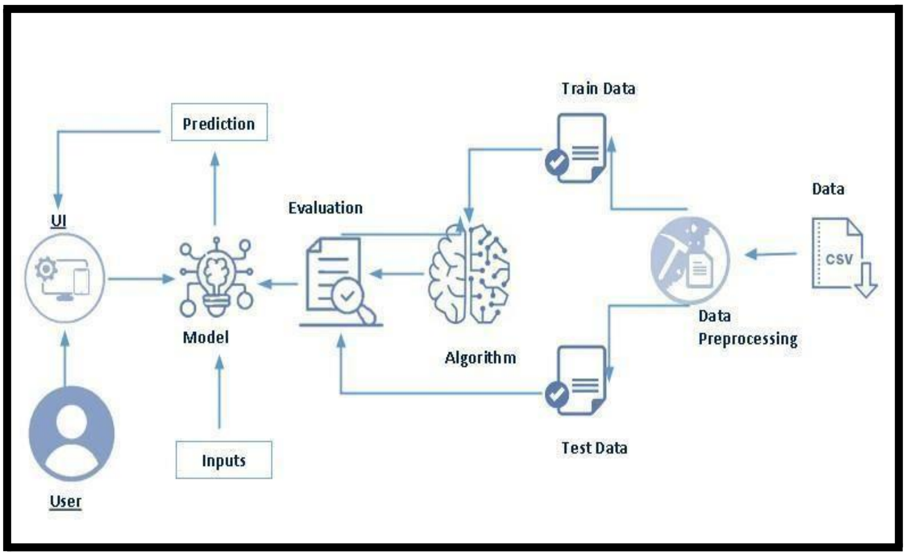
Predicting Personal Loan Approval Using Machine Learning

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 creditworthiness.To qualify for a personal loan, borrowers typically need to provide proof of income and have a good credit score.

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.

Technical Architecture:



Project Flow:

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

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social or Business Impact.
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis

    - Descriptive statistical
    - Visual Analysis
- Model Building
    - Training the model in multiple algorithms
    - Testing the model
- Performance Testing & Hyperparameter Tuning
    - Testing model with multiple evaluation metrics
    - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
    - Save the best model
    - Integrate with Web Framework
- Project Demonstration & Documentation
    - Record explanation Video for project end to end solution
    - Project Documentation-Step by step project development procedure

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- rdf.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains a model training file.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

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

Activity 3: Literature Survey (Student Will Write)

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

Activity 4: Social or Business Impact.

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

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

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset. **Activity 1: Collect the dataset**

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

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

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

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

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

Activity 1.2: Read the Dataset

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

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

```python
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
```

0s

```python
#importing the dataset which is in csv file
data = pd.read_csv('/content/test.csv')
data = pd.read_csv('/content/train.csv')
data
```

0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban
...	...	...	...	...	...	...	...	...	...	...	...	...
609	LP002978	Female	No	0	Graduate	No	2900	0.0	71.0	360.0	1.0	Rural
610	LP002979	Male	Yes	3+	Graduate	No	4106	0.0	40.0	180.0	1.0	Rural
611	LP002983	Male	Yes	1	Graduate	No	8072	240.0	253.0	360.0	1.0	Urban
612	LP002984	Male	Yes	2	Graduate	No	7583	0.0	187.0	360.0	1.0	Urban
613	LP002990	Female	No	0	Graduate	Yes	4583	0.0	133.0	360.0	0.0	Semiurban

614 rows × 13 columns

Activity 2: Data Preparation

Activity 2.1: Handling missing values

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

```
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])
```

```
[171] data['Married'] = data['Married'].fillna(data['Married'].mode()[0])
```

```
data['Dependents'] = data['Dependents'].str.replace('+','')
```

```
<ipython-input-172-e7493c2d1d37>:1: FutureWarning: The default value of regex will change from True to False in a future version. In addition, single character regular expressions will *not* b
  data['Dependents'] = data['Dependents'].str.replace('+','')
```

```
[173] data['Dependents'] = data['Dependents'].fillna(data['Dependents'].mode()[0])
```

```
[174] data['Self_Employed'] = data['Self_Employed'].fillna(data['Self_Employed'].mode()[0])
```

```
[175] data['LoanAmount'] = data['LoanAmount'].fillna(data['LoanAmount'].mode()[0])
```

```
[176] 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.

Activity 2.2: Handling Categorical Values

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

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

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

```python
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 2.3:Handling Imbalance Data

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

For Balancing the data we are using the SMOTE Method.

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

```python
[187] from imblearn.combine import SMOTETomek
```

```python
[188] smote = SMOTETomek()
```

```python
[189] y = data['Loan_Status']
      x = data.drop(columns=['Loan_Status'],axis=1)
```

```python
[190] x.shape

      (614, 11)
```

```python
[191] y.shape

      (614,)
```

```python
[192] x_bal,y_bal = smote.fit_resample(x,y)
```

```python
[193] print(y.value_counts())
      print(y_bal.value_counts())

      1    422
      0    192
      Name: Loan_Status, dtype: int64
      1    356
      0    356
      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.

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

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

```
data.describe()
```

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
count	614.000000	614.000000	614.000000	614.000000	614.000000	614.000000	614.00000	614.000000	614.000000	614.000000	614.000000	614.000000
mean	0.182410	0.653094	0.744300	0.781759	0.133550	5403.459283	1621.24430	145.465798	342.410423	0.855049	1.037459	0.687296
std	0.386497	0.476373	1.009623	0.413389	0.340446	6109.041673	2926.24876	84.180967	64.428629	0.352339	0.787482	0.463973
min	0.000000	0.000000	0.000000	0.000000	0.000000	150.000000	0.00000	9.000000	12.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	1.000000	0.000000	2877.500000	0.00000	100.250000	360.000000	1.000000	0.000000	0.000000
50%	0.000000	1.000000	0.000000	1.000000	0.000000	3812.500000	1188.50000	125.000000	360.000000	1.000000	1.000000	1.000000
75%	0.000000	1.000000	1.000000	1.000000	0.000000	5795.000000	2297.25000	164.750000	360.000000	1.000000	2.000000	1.000000
max	1.000000	1.000000	3.000000	1.000000	1.000000	81000.000000	41667.00000	700.000000	480.000000	1.000000	2.000000	1.000000

Activity 2: Visual analysis

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

Activity 2.1: Univariate analysis

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

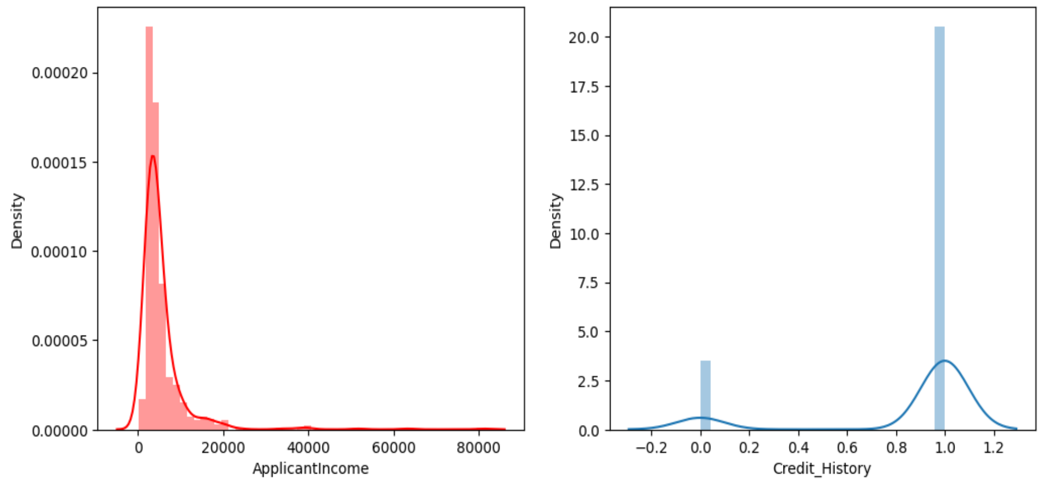
- The Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.

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

```
<ipython-input-182-4b78f43a4171>:3: UserWarning:

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

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

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

  sns.distplot(data['ApplicantIncome'], color='r')
<ipython-input-182-4b78f43a4171>:5: UserWarning:

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

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


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


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

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



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

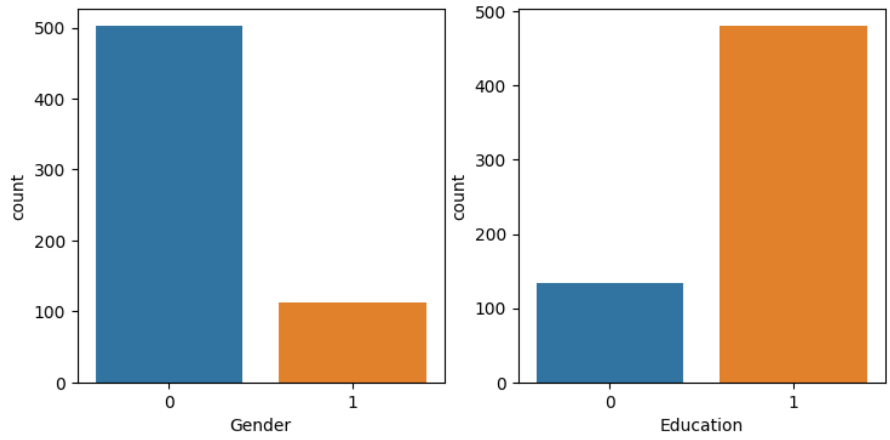
Countplot:-

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

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

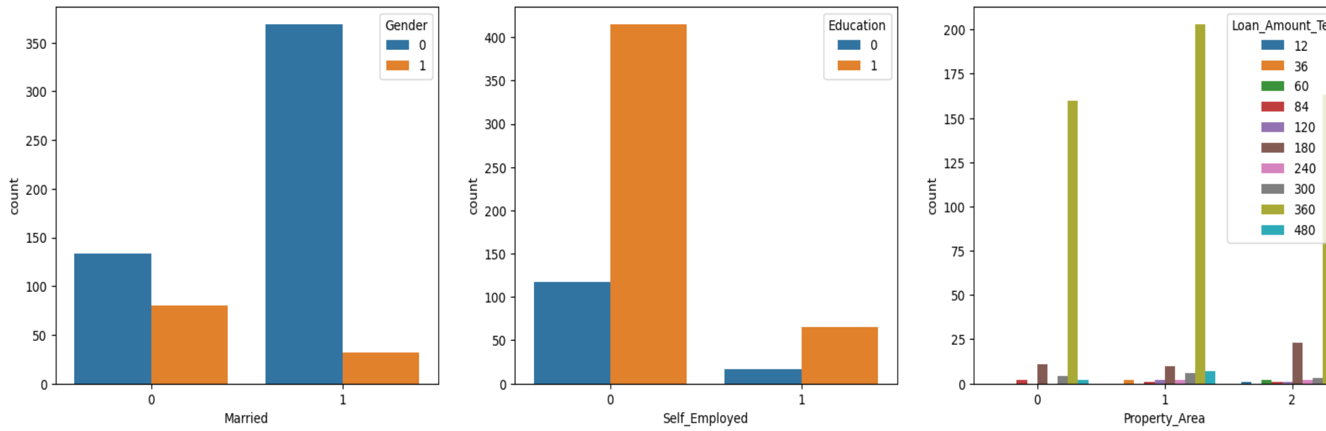
Activity 2.2: Bivariate analysis

```
plt.figure(figsize=(18,4))
plt.subplot(1,4,1)
sns.countplot(x = 'Gender',data = data)
plt.subplot(1,4,2)
sns.countplot(x = 'Education',data = data)
plt.show()
```




```
plt.figure(figsize=(20,5))
plt.subplot(131)
sns.countplot(x = 'Married', hue = 'Gender', data = data)
plt.subplot(132)
sns.countplot(x = 'Self_Employed', hue = 'Education', data = data)
plt.subplot(133)
sns.countplot(x = 'Property_Area', hue = 'Loan_Amount_Term', data = data)
```

<Axes: xlabel='Property_Area', ylabel='count'>



From the above graph we can infer the analysis such as

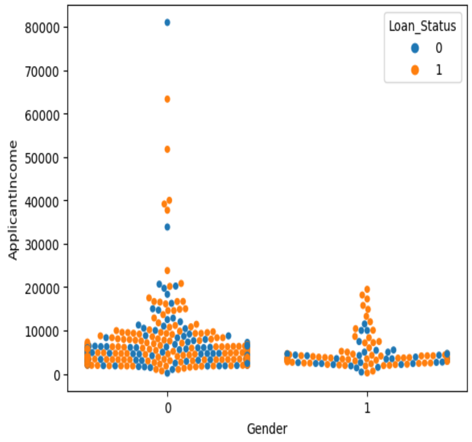
- Segmenting the gender column and married column based on bar graphs
- Segmenting the Education and Self-employed based on bar graphs ,for drawing insights such as educated people are employed.
- Loan amount term based on the property area of a person holding

Activity 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.

```
sns.swarmplot(x = "Gender",y = "ApplicantIncome", hue = "Loan_Status", data = data)
```

```
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 45.8% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
<Axes: xlabel='Gender',
ylabel='ApplicantIncome'>/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 61.6% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
/usr/local/lib/python3.9/dist-packages/seaborn/categorical.py:3544: UserWarning: 25.0% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```



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.

```
[196] sc=StandardScaler()
      x_bal=sc.fit_transform(x_bal)
```

```
[197] x_bal

array([[-0.42285689, -1.15186691, -0.71535408, ...,  0.29177308,
         0.6336993 ,  1.32799102],
       [-0.42285689,  0.86815585,  0.35019425, ...,  0.29177308,
         0.6336993 , -1.21717616],
       [-0.42285689,  0.86815585, -0.71535408, ...,  0.29177308,
         0.6336993 ,  1.32799102],
       ...,
       [-0.42285689, -1.15186691, -0.71535408, ...,  0.29177308,
         0.6336993 , -1.21717616],
       [-0.42285689,  0.86815585, -0.71535408, ...,  0.29177308,
        -1.57803551, -1.21717616],
       [-0.42285689,  0.86815585, -0.71535408, ...,  0.29177308,
        -1.57803551,  0.05540743]])
```

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

	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
0	-0.422857	-1.151867	-0.715354	0.638055	-0.332813	0.097527	-0.507924	-0.295570	0.291773	0.633699	1.327991
1	-0.422857	0.868156	0.350194	0.638055	-0.332813	-0.118137	-0.031761	-0.197191	0.291773	0.633699	-1.217176
2	-0.422857	0.868156	-0.715354	0.638055	3.004691	-0.387803	-0.507924	-0.959631	0.291773	0.633699	1.327991
3	-0.422857	0.868156	-0.715354	-1.567262	-0.332813	-0.458839	0.236634	-0.295570	0.291773	0.633699	1.327991
4	-0.422857	-1.151867	-0.715354	0.638055	-0.332813	0.123250	-0.507924	-0.037324	0.291773	0.633699	1.327991

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 sklearn. As parameters, we are passing x, y, test_size, random_state.

+ Code + Text

```
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
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

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

Activity 1.1: Decision tree model

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

```
[203] def decisionTree(x_train,x_test,y_train,y_test):
          dt=DecisionTreeClassifier()
          dt.fit(x_train,y_train)
          yPred = dt.predict(x_test)
          print('***DecisionTreeClassifier***')
          print('Confusion matrix')
          print(confusion_matrix(y_test,yPred))
          print('Classification report')
          print(classification_report(y_test,yPred))
```

Activity 1.2: Random forest model

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

```
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 1.3: KNN model

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

```
[205] def KNN(x_train,x_test,y_train,y_test):
          dt = 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 1.4: Xgboost model

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable.

For evaluating the model, confusion matrix and classification report is done.

```
[206] def xgboost(x_train,x_test,y_train,y_test):
          xg = GradientBoostingClassifier()
          xg.fit(x_train,y_train)
          yPred = xg.predict(x_test)
          print('***GradientBoostingClassifier***')
          print('Confusion matrix')
          print(confusion_matrix(y_test,yPred))
          print('Classification report')
          print(classification_report(y_test,yPred))
```

Activity 1.5: ANN model

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

```python
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```python
[208] classifier = Sequential()
```

```python
[209] classifier.add(Dense(units=100, activation='relu', input_dim=11))
```

```python
[210] classifier.add(Dense(units=50, activation='relu'))
```

```python
[211] classifier.add(Dense(units=1, activation='sigmoid'))
```

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

```python
model_history = classifier.fit(x_train, y_train, batch_size=100, validation_split=0.2, epochs=100)
```

```
Epoch 1/100
4/4 [==============================] - 1s 76ms/step - loss: 0.6869 - accuracy: 0.5486 - val_loss: 0.6538 - val_accuracy: 0.6667
Epoch 2/100
4/4 [==============================] - 0s 15ms/step - loss: 0.6423 - accuracy: 0.6824 - val_loss: 0.6258 - val_accuracy: 0.7292
Epoch 3/100
4/4 [==============================] - 0s 14ms/step - loss: 0.6066 - accuracy: 0.7480 - val_loss: 0.6010 - val_accuracy: 0.7396
Epoch 4/100
4/4 [==============================] - 0s 16ms/step - loss: 0.5734 - accuracy: 0.7585 - val_loss: 0.5801 - val_accuracy: 0.7604
Epoch 5/100
4/4 [==============================] - 0s 14ms/step - loss: 0.5465 - accuracy: 0.7638 - val_loss: 0.5602 - val_accuracy: 0.7917
Epoch 6/100
4/4 [==============================] - 0s 20ms/step - loss: 0.5217 - accuracy: 0.7979 - val_loss: 0.5446 - val_accuracy: 0.7917
Epoch 7/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4998 - accuracy: 0.7953 - val_loss: 0.5321 - val_accuracy: 0.7812
Epoch 8/100
4/4 [==============================] - 0s 15ms/step - loss: 0.4808 - accuracy: 0.7979 - val_loss: 0.5225 - val_accuracy: 0.7708
Epoch 9/100
4/4 [==============================] - 0s 23ms/step - loss: 0.4644 - accuracy: 0.8005 - val_loss: 0.5156 - val_accuracy: 0.7812
```

✓ 0s completed at 4:37 PM

```python
model_history = classifier.fit(x_train, y_train, batch_size=100, validation_split=0.2, epochs=100)
```

```
4/4 [==============================] - 0s 14ms/step - loss: 0.4491 - accuracy: 0.8136 - val_loss: 0.5102 - val_accuracy: 0.7708
Epoch 11/100
4/4 [==============================] - 0s 21ms/step - loss: 0.4377 - accuracy: 0.8189 - val_loss: 0.5063 - val_accuracy: 0.7812
Epoch 12/100
4/4 [==============================] - 0s 20ms/step - loss: 0.4281 - accuracy: 0.8189 - val_loss: 0.5039 - val_accuracy: 0.7812
Epoch 13/100
4/4 [==============================] - 0s 22ms/step - loss: 0.4184 - accuracy: 0.8163 - val_loss: 0.5039 - val_accuracy: 0.7917
Epoch 14/100
4/4 [==============================] - 0s 14ms/step - loss: 0.4111 - accuracy: 0.8189 - val_loss: 0.5045 - val_accuracy: 0.7917
Epoch 15/100
4/4 [==============================] - 0s 15ms/step - loss: 0.4048 - accuracy: 0.8163 - val_loss: 0.5069 - val_accuracy: 0.7812
Epoch 16/100
4/4 [==============================] - 0s 14ms/step - loss: 0.3988 - accuracy: 0.8163 - val_loss: 0.5085 - val_accuracy: 0.7812
Epoch 17/100
4/4 [==============================] - 0s 19ms/step - loss: 0.3936 - accuracy: 0.8189 - val_loss: 0.5099 - val_accuracy: 0.7812
Epoch 18/100
4/4 [==============================] - 0s 13ms/step - loss: 0.3887 - accuracy: 0.8189 - val_loss: 0.5125 - val_accuracy: 0.7812
Epoch 19/100
4/4 [==============================] - 0s 17ms/step - loss: 0.3836 - accuracy: 0.8215 - val_loss: 0.5138 - val_accuracy: 0.7812
Epoch 20/100
4/4 [==============================] - 0s 15ms/step - loss: 0.3798 - accuracy: 0.8189 - val_loss: 0.5167 - val_accuracy: 0.7604
Epoch 21/100
```

```
Epoch 90/100
4/4 [==============================] - 0s 24ms/step - loss: 0.2070 - accuracy: 0.9213 - val_loss: 0.7023 - val_accuracy: 0.7188
Epoch 91/100
4/4 [==============================] - 0s 27ms/step - loss: 0.2058 - accuracy: 0.9213 - val_loss: 0.7048 - val_accuracy: 0.7083
Epoch 92/100
4/4 [==============================] - 0s 24ms/step - loss: 0.2050 - accuracy: 0.9239 - val_loss: 0.7072 - val_accuracy: 0.7083
Epoch 93/100
4/4 [==============================] - 0s 24ms/step - loss: 0.2028 - accuracy: 0.9213 - val_loss: 0.7078 - val_accuracy: 0.7292
Epoch 94/100
4/4 [==============================] - 0s 29ms/step - loss: 0.2012 - accuracy: 0.9213 - val_loss: 0.7164 - val_accuracy: 0.6979
Epoch 95/100
4/4 [==============================] - 0s 20ms/step - loss: 0.1989 - accuracy: 0.9213 - val_loss: 0.7188 - val_accuracy: 0.6979
Epoch 96/100
4/4 [==============================] - 0s 24ms/step - loss: 0.1983 - accuracy: 0.9160 - val_loss: 0.7208 - val_accuracy: 0.7188
Epoch 97/100
4/4 [==============================] - 0s 25ms/step - loss: 0.1953 - accuracy: 0.9213 - val_loss: 0.7240 - val_accuracy: 0.7083
Epoch 98/100
4/4 [==============================] - 0s 26ms/step - loss: 0.1939 - accuracy: 0.9239 - val_loss: 0.7274 - val_accuracy: 0.6875
Epoch 99/100
4/4 [==============================] - 0s 20ms/step - loss: 0.1935 - accuracy: 0.9265 - val_loss: 0.7266 - val_accuracy: 0.7292
Epoch 100/100
4/4 [==============================] - 0s 29ms/step - loss: 0.1921 - accuracy: 0.9239 - val_loss: 0.7311 - val_accuracy: 0.6979
```

Activity 2: Testing the model

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

```
[214] y_pred = classifier.predict(x_test)

8/8 [==============================] - 0s 2ms/step
```

```
[215] y_pred

[6.17431641e-01],
[9.16451156e-01],
[5.26232362e-01],
```

```
y_pred = (y_pred > 0.5)
y_pred

array([[False],
```

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

Finally, the function returns the prediction of the classifier on the scaled sample_value.

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

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

Activity 1.1: Compare the model

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

```
def RandomForest(x_train,x_test,y_train,y_test):
    model = RandomForestClassifier(verbose= 4, n_estimators= 68,max_features= 'auto',max_depth= 8,criterion= 'entropy')
    model.fit(x_train,y_train)
    y_tr = model.predict(x_train)
    print("Training Accuracy")
    print(accuracy_score(y_tr,y_train))
    yPred = model.predict(x_test)
    print('Testing Accuracy')
    print(accuracy_score(yPred,y_test))
```

```
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
0.49361702127659574
ANN Model
Confusion_Matrix
[[116   0]
 [119   0]]
Classification Report
              precision    recall  f1-score   support

           0       0.49      1.00      0.66       116
           1       0.00      0.00      0.00       119

    accuracy                           0.49       235
   macro avg       0.25      0.50      0.33       235
weighted avg       0.24      0.49      0.33       235
```

```
    _warn_prf(average, modifier, msg_start, len(result))

[218] rf = RandomForestClassifier()

[219] parameters = {
                      'n_estimators' : [1,20,30,55,68,74,90,120,115],
                       'criterion':['gini','entropy'],
                       'max_features' : ["auto", "sqrt", "log2"],
              'max_depth' : [2,5,8,10], 'verbose' : [1,2,3,4,6,8,9,10]
      }

RCV  = RandomizedSearchCV(estimator=rf,param_distributions=parameters,cv=10,n_iter=4)
```

```
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))
```

```
0.49361702127659574
ANN Model
Confusion_Matrix
[[116   0]
 [119   0]]
Classification Report
              precision    recall  f1-score   support

           0       0.49      1.00      0.66       116
           1       0.00      0.00      0.00       119

    accuracy                           0.49       235
   macro avg       0.25      0.50      0.33       235
weighted avg       0.24      0.49      0.33       235
```

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.

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

Evaluating performance of the model From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model by pickle.dump().

Note: To understand cross validation, refer to this link

Milestone 6: Model Deployment

Activity 1:Save the best model

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

```
[228] pickle.dump(model,open('rdf.pkl','wb'))
```

```
[229] pickle.dump(sc,open('scale.pkl','wb'))
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server side script
- Run the web application

Activity 2.1: Building Html Pages:

For this project create two HTML files namely

- home.html
- predict.html and save them in the templates folder.

Activity 2.2: Build Python code:

Import the libraries

```
[230] from flask import Flask
      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.

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

Render HTML page:

```
[232] @app.route('/') # rendering the html templet
      def home():
        return render_template('home.html')

      @app.route('/submit',methods=["POST","GET"])# route to show
      def submit():
```

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:

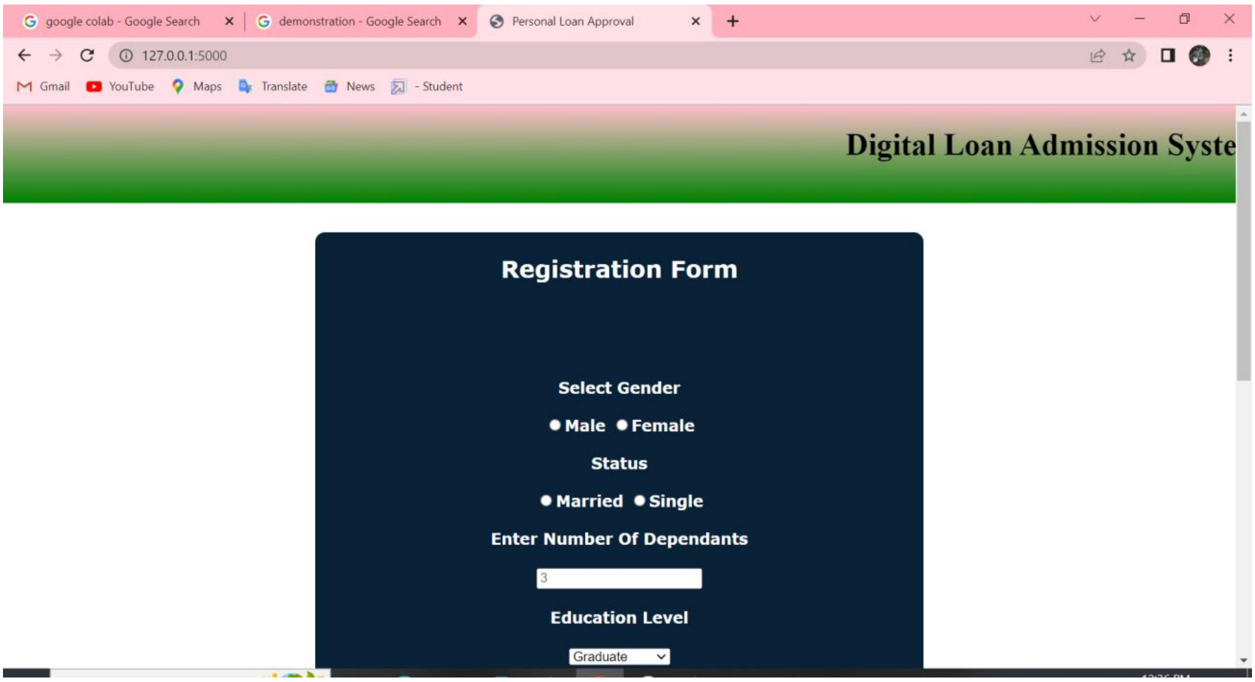
```
@app.route('/submit',methods=["POST","GET"])# route to show the prediction in a UI
def submit():
  # reading the inputs given by the user
  input_feature=[int(X) for x in request.form.value()]
  #input_feature = np.transpose(input_feature)
  input_feature=[np.array(input_feature)]
  print(input_feature)
  names = ['Gender', 'Married', 'Departments', 'Education', 'Self_Empolyed', 'ApplicantIncome',
           'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History','Property_Area']
  print(data)



   # predictions using the loaded model file
  prediction=model.predict(date)
  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
```

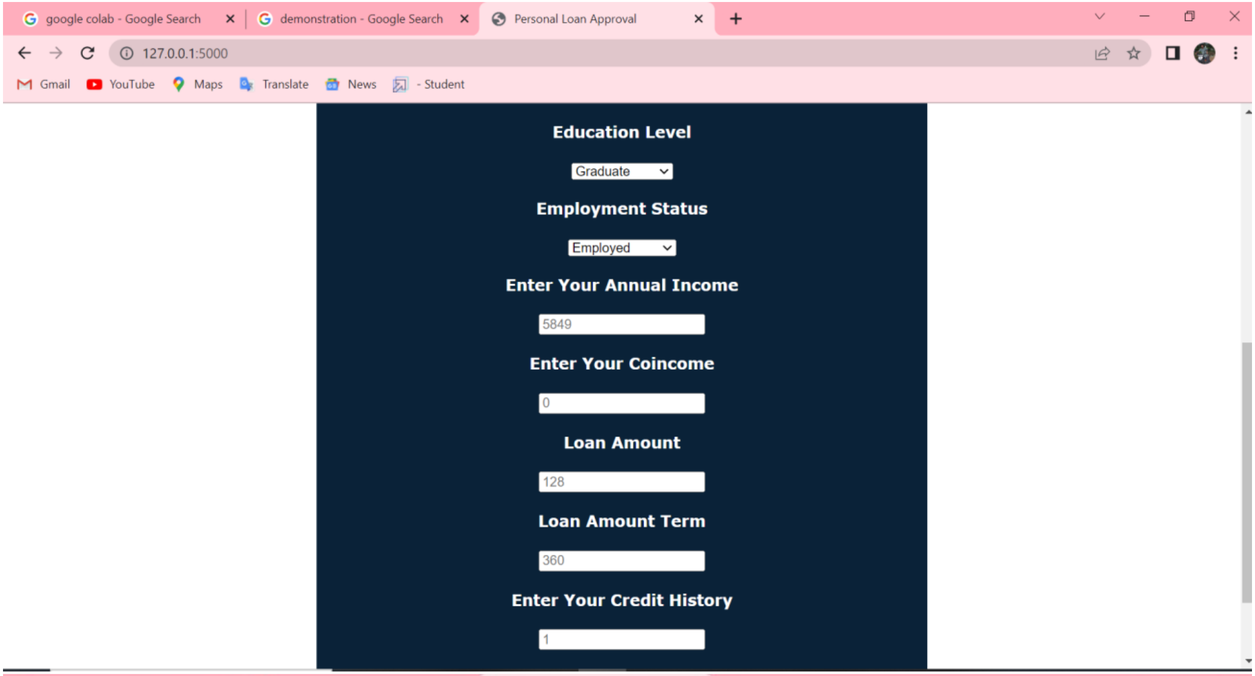
Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that

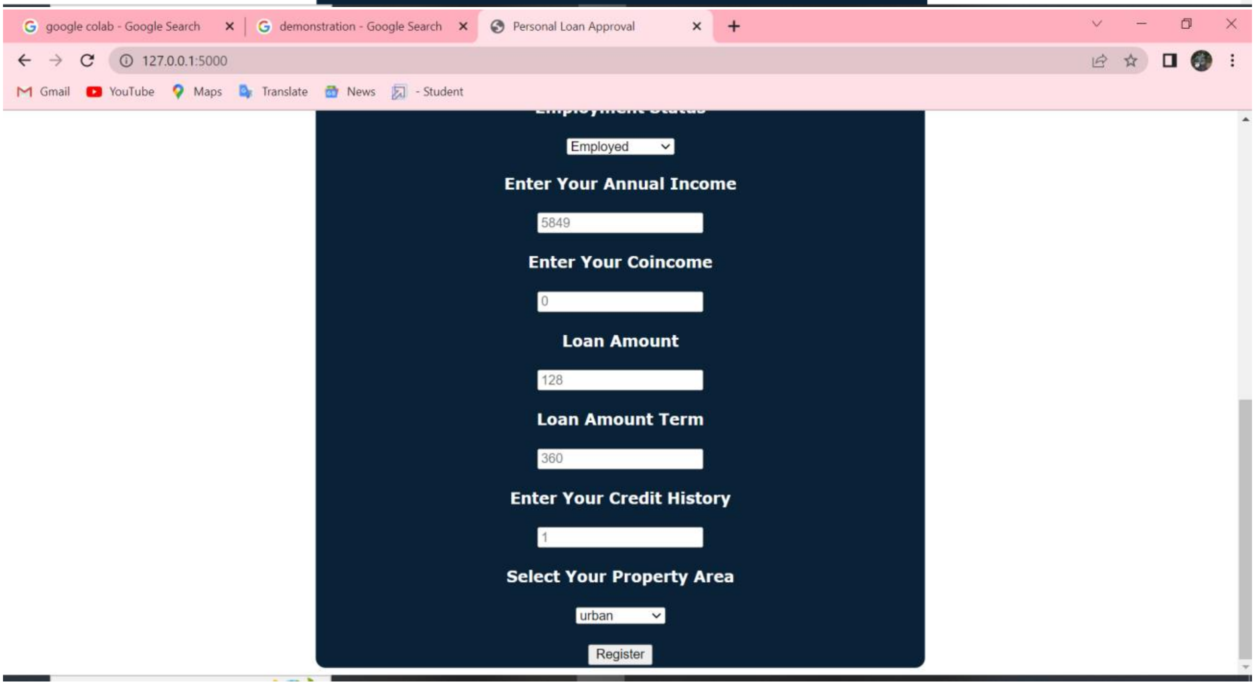
```
[234] if __name__=="__main__":
        def os():
         # app.run(host='0.0.0.0', port=8000,debug=True)    # running the app
         port=int(os.environ.get('PORT',5000))
         app.run(debug=False)
```

0s completed at 5:28 PM

Activity 2.3: Run the web application




google colab - Google Search
demonstration - Google Search
Personal Loan Approval
127.0.0.1:5000
Gmail
YouTube
Maps
Translate
News
- Student
Education Level
Graduate
Employment Status
Employed
Enter Your Annual Income
5849
Enter Your Coincome
0
Loan Amount
128
Loan Amount Term
360
Enter Your Credit History
1


google colab - Google Search
demonstration - Google Search
Personal Loan Approval
127.0.0.1:5000
Gmail
YouTube
Maps
Translate
News
- Student
Employed
Enter Your Annual Income
5849
Enter Your Coincome
0
Loan Amount
128
Loan Amount Term
360
Enter Your Credit History
1
Select Your Property Area
urban
Register