

Online Payments Fraud Detection using Machine learning

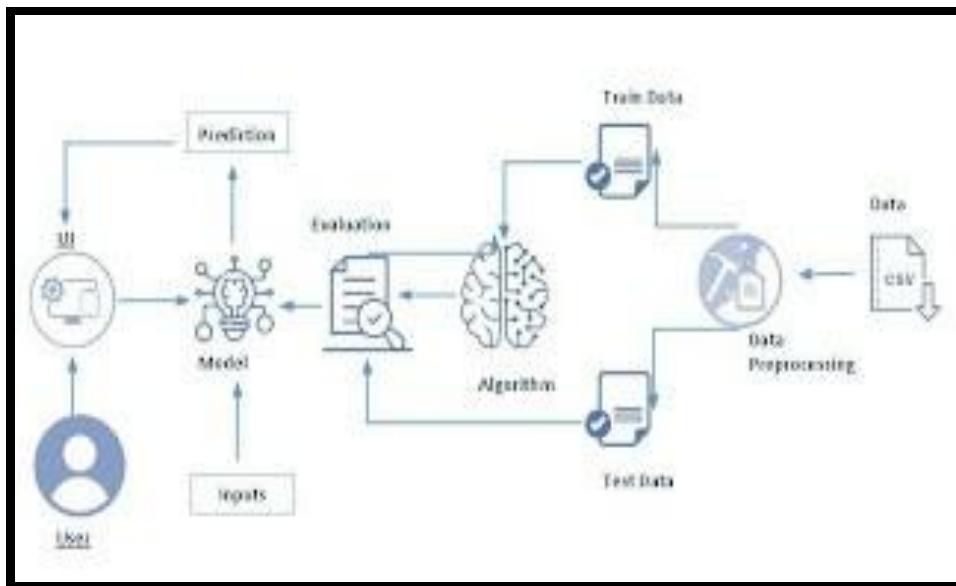
Project Description:

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

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

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

Technical Architecture:



Pre requisites:

To complete this project, you must required following software's, concepts and packages

- **Anaconda navigator and pycharm:**

- o Refer the link below to download anaconda navigator o Link : <https://youtu.be/1ra4zH2G4o0>

- **Python packages:**

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

Prior Knowledge:

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

- **ML Concepts**

- o Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
- o Unsupervised learning: <https://www.javatpoint.com/unsupervised-machinelearning>
- o Regression and classification
- o Decision tree: <https://www.javatpoint.com/machine-learning-decision-treeclassification-algorithm>
- o Random forest: <https://www.javatpoint.com/machine-learning-random-forestalgorithm>
- o KNN: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machinelearning>
- o Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
- o Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11important-model-evaluation-error-metrics/>

- **Flask Basics** : https://www.youtube.com/watch?v=lj4l_CvBnt0

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

Project Flow:

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

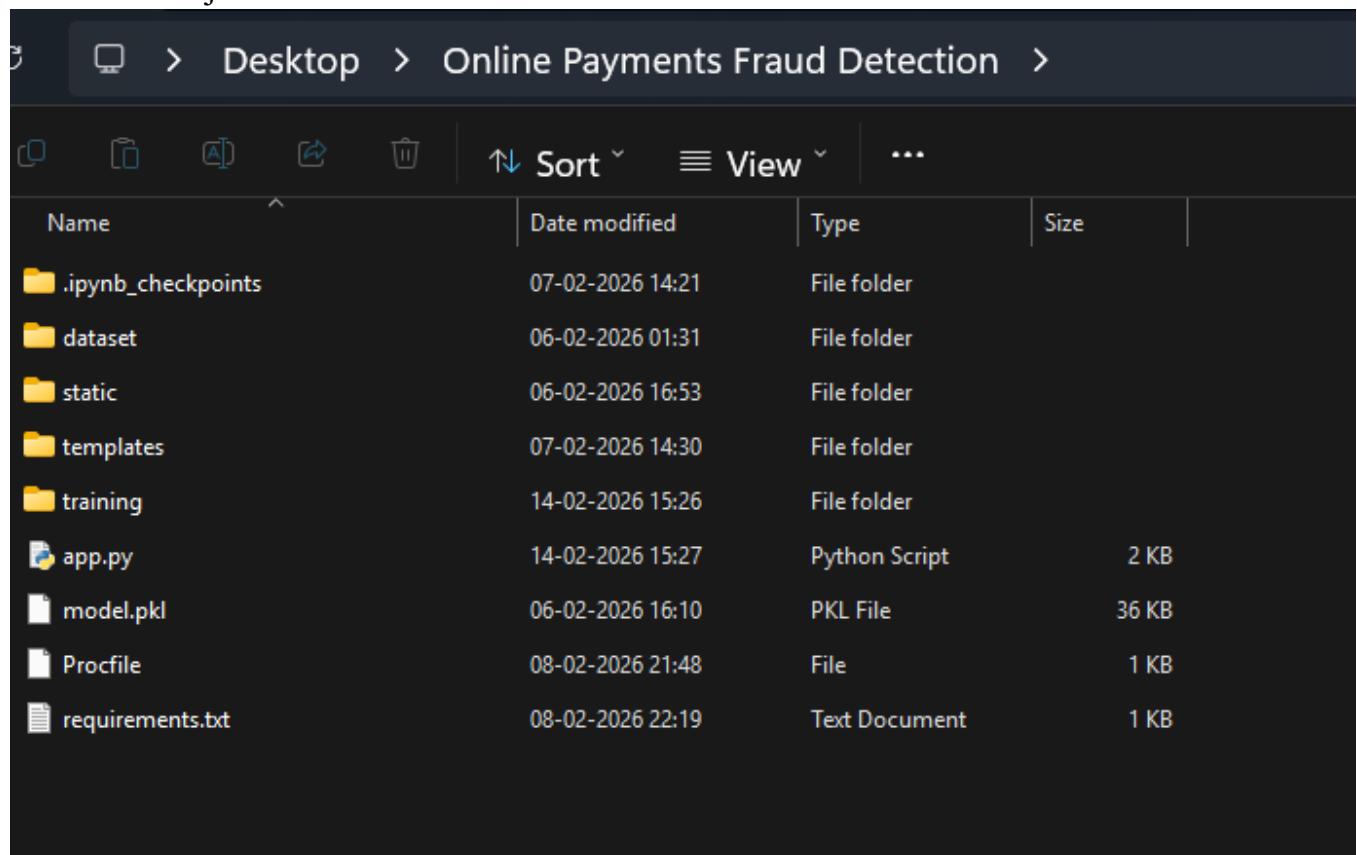
To accomplish this, we have to complete all the activities listed below,

- Data collection o Collect the dataset or create the dataset

- Visualizing and analyzing data o Univariate analysis o Bivariate analysis o Multivariate analysis o Descriptive analysis
- Data pre-processing o Checking for null values o Handling outlier o Handling categorical data o Splitting data into train and test
- Model building o Import the model building libraries o Initializing the model o Training and testing the model o Evaluating performance of model o Save the model
- Application Building o Create an HTML file o Build python code

Project Structure:

Create the Project folder which contains files as shown below



The screenshot shows a file explorer window with the following details:

Path: Desktop > Online Payments Fraud Detection

Toolbar: Includes icons for back, forward, search, and file operations.

Sort and View Options: Sort dropdown, View dropdown, and a three-dot menu icon.

Table Headers: Name, Date modified, Type, Size

Table Data:

Name	Date modified	Type	Size
.ipynb_checkpoints	07-02-2026 14:21	File folder	
dataset	06-02-2026 01:31	File folder	
static	06-02-2026 16:53	File folder	
templates	07-02-2026 14:30	File folder	
training	14-02-2026 15:26	File folder	
app.py	14-02-2026 15:27	Python Script	2 KB
model.pkl	06-02-2026 16:10	PKL File	36 KB
Procfile	08-02-2026 21:48	File	1 KB
requirements.txt	08-02-2026 22:19	Text Document	1 KB

- 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 model training files and training_ibm folder contains IBM deployment files.

Milestone 1: Data Collection

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

Activity 1: Download the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used drug200.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link:

https://docs.google.com/spreadsheets/d/1T4tlqMwRXEpYdWlneX2sY5UqrctsVIMMirgkTOKLd_kI/edit#gid=1373903219

Milestone 2: Visualizing and analysing the data

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

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

Activity 1: Importing the libraries

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

```
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 joblib
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
```

Activity 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 csv file.

```
Importing the dataset which is on online  
data = pd.read_csv('lens_predictions.csv')  
data
```

Loan_ID	Gender	Married	Dependents	Education	Not_Employed	ApplicantIncome	CosapplicantIncome	LoanAmount	L	
0	GP001002	Male	No	0	Graduate	No	5040	8.0	NaN	3
1	LP001003	Male	No	1	Graduate	No	4083	1500.0	128.0	3
2	LP001005	Male	No	0	Graduate	Yes	3000	8.0	85.0	3
3	LP001008	Male	Yes	0	Not Graduate	No	2585	2350.0	128.0	3
4	LP001009	Male	No	0	Graduate	No	6000	8.0	141.0	3
...
600	GP002078	Female	No	0	Graduate	No	3000	8.0	71.0	3
610	LP002079	Male	No	3+	Graduate	No	4000	8.0	40.0	1
611	LP002080	Male	No	1	Graduate	No	6072	240.0	253.0	3
612	LP002084	Male	Yes	2	Graduate	No	3583	8.0	85.0	3
613	LP002090	Female	No	0	Graduate	Yes	4583	8.0	153.0	3

834 rows × 12 columns

Activity 3: Univariate analysis

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

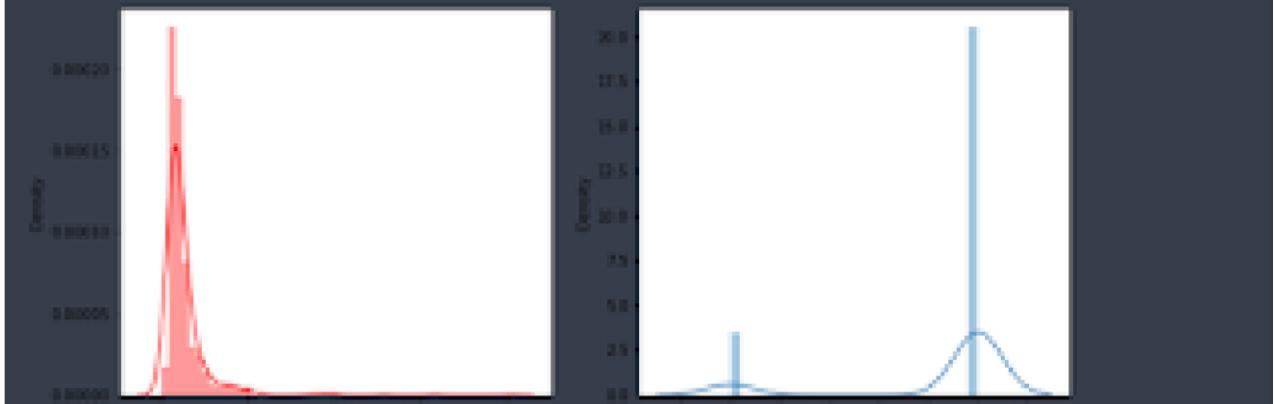
- Seaborn package provides a wonderful function `distplot`. With the help of `distplot`, we can find the distribution of the feature. To make multiple graphs in a single plot, we use `subplot`.

```

splitting the using distplot
plt.figure(figsize=(12,8))
plt.subplot(221)
sns.distplot(data['ApplicantIncome'], color='r')
plt.subplot(222)
sns.distplot(data['Credit_History'])
plt.show()

C:\Users\HP\Anaconda3\lib\site-packages\seaborn\distplots.py:269: FutureWarning: "distplot" is a deprecated function and will be removed in a future version. Please adapt your code to use either "kdeplot" (a figure-level function with either "kde" or "histplot" as main-level function for histograms),
warnings.warn(msg, FutureWarning)
C:\Users\HP\Anaconda3\lib\site-packages\seaborn\distplots.py:269: FutureWarning: "distplot" is a deprecated function and will be removed in a future version. Please adapt your code to use either "kdeplot" (a figure-level function with either "kde" or "histplot" as main-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, whereas 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 .

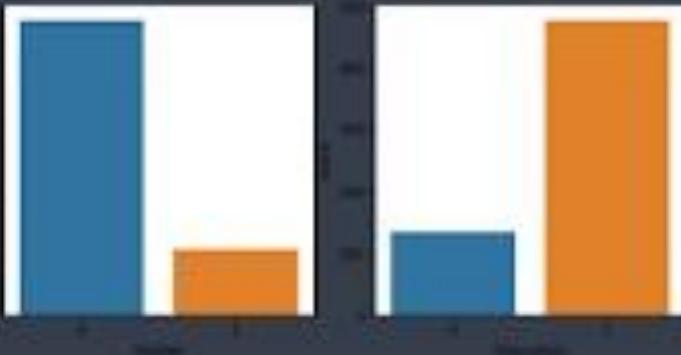
Activity 4: Bivariate analysis

```
import matplotlib.pyplot as plt
plt.figure(figsize=(8,4))
plt.subplot(1,4,1)
sns.countplot(data['Gender'])
plt.subplot(1,4,2)
sns.countplot(data['Education'])
plt.show()
```

```
C:\Users\HP\anaconda3\envs\datascientest\lib\site-packages\seaborn\categorical.py:63: FutureWarning: Pass the following variable as a keyword arg to pplot or
    raise ValueError("The only valid positional argument will be 'data', and passing other arguments without an explicit keyword will result in an
    error in matplotlib.",

C:\Users\HP\anaconda3\envs\datascientest\lib\site-packages\seaborn\categorical.py:63: FutureWarning: Pass the following variable as a keyword arg to pplot or
    raise ValueError("The only valid positional argument will be 'data', and passing other arguments without an explicit keyword will result in an
    error in matplotlib.",

warnings.warn
```



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.

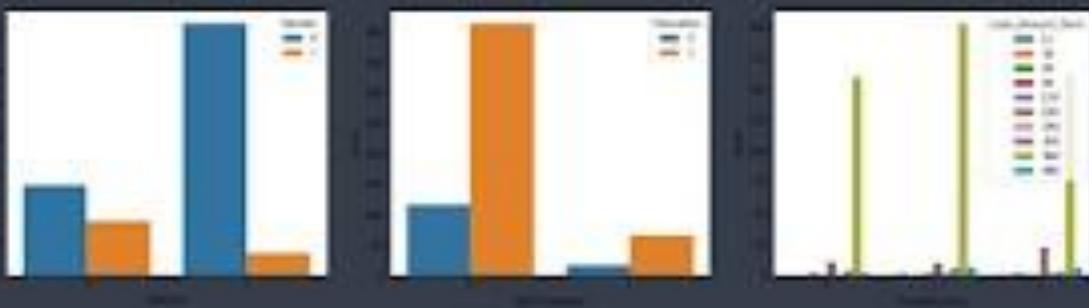
```

plt.figure(figsize=(20,8))
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_on_kwarginterpretation
warnings.warn(
    "The 'order' kwarg is deprecated. Use 'category_order' instead. FutureWarning: Pass the following variable as a keyword arg, e.g., from 0.4.0 onwards: 'order'. The 'order' kwarg parameter will be 'order', and passing other arguments除了 'category_order'之外 will result in an error or misinterpretation."
)
warnings.warn()

countplot(x='Property_Area', palette='magma')

```

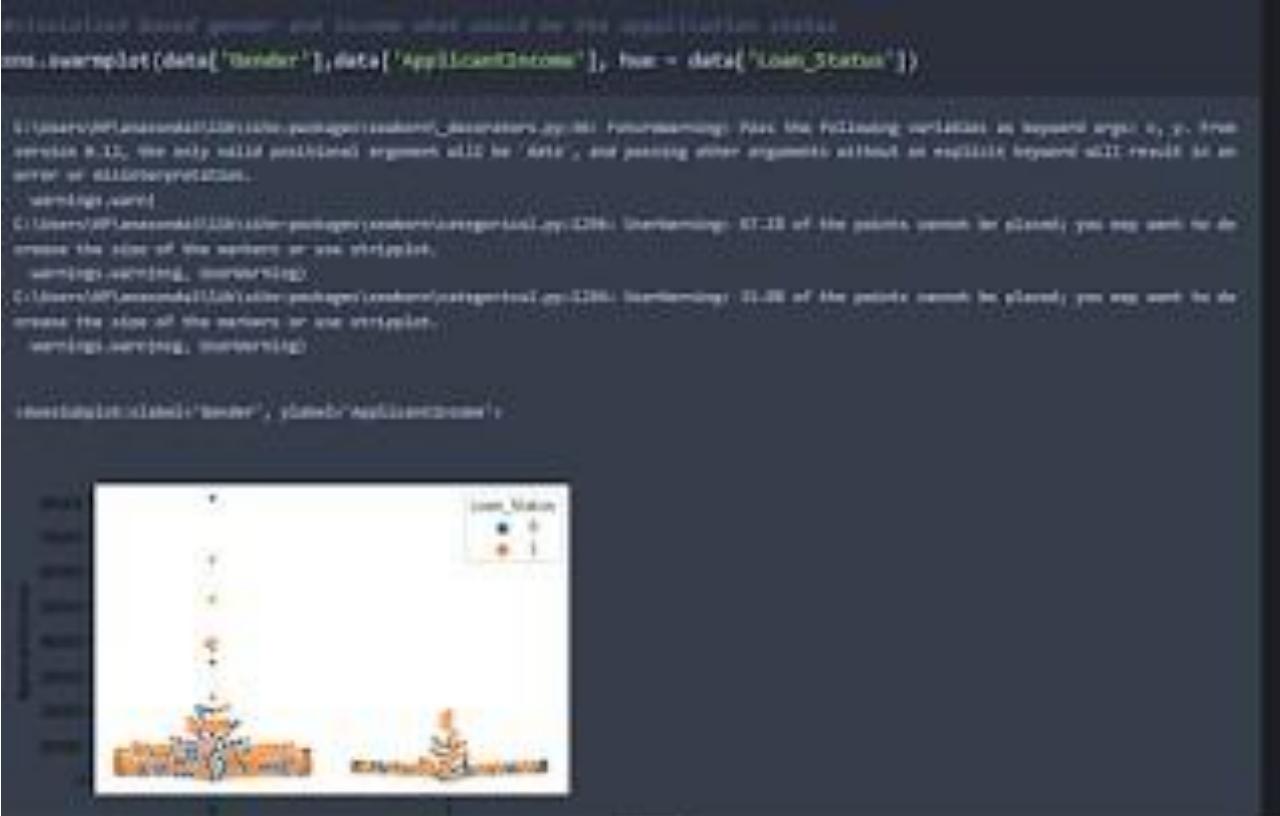


From the above graph we can infer the analysis such as

- 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 5: Multivariate analysis

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



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

Activity 6: Descriptive analysis

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.

	ApplicantIncome	CosapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	582.000000	680.000000	564.000000
mean	5489.458283	1021.345798	146.412142	342.900000	0.842199
std	6189.941673	3926.348360	65.587325	65.13941	0.394829
min	150.000000	0.000000	0.000000	12.000000	0.000000
25%	2877.500000	8.000000	100.000000	360.000000	1.000000
50%	3612.500000	1184.500000	120.000000	360.000000	1.000000
75%	5795.000000	2367.250000	160.000000	360.000000	1.000000
max	81830.000000	41067.000000	790.000000	480.000000	1.000000

Milestone 3: Data Pre-processing

As we have understood how the data is lets pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much of randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling categorical data
- Handling outliers
- Scaling Techniques
- Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 1: Checking for null values

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

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
 ...   ...            
 0   Loan_ID          614 non-null    object 
 1   Gender           601 non-null    object 
 2   Married          611 non-null    object 
 3   Dependents       599 non-null    object 
 4   Education        614 non-null    object 
 5   Self_Employed    542 non-null    object 
 6   ApplicantIncome  614 non-null    int64  
 7   CoapplicantIncome 614 non-null    float64
 8   LoanAmount        592 non-null    float64
 9   Loan_Amount_Term  600 non-null    float64
 10  Credit_History    564 non-null    float64
 11  Property_Area     614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

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

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

Gender           33
Married          3
Dependents      15
Education         8
Self_Employed   32
ApplicantIncome    0
CoapplicantIncome  0
LoanAmount        22
Loan_Amount_Term  34
Credit_History    58
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 tern 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 non values
data['Dependents']=data['Dependents'].str.replace('+','')

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

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

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

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

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

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

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

Activity 3: Balancing the dataset:

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

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

```

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

smote = SMOTETomek(0.99)

# UserWarning: /Applications/Python/Python38/lib/python3.8/site-packages/imbalance/_smote.py:107: FutureWarning: Non-tuple args... Note: version 0.9 passing these as positional arguments will result in an error
# warnings.warn()

# Balancing the dataset, this argument and 'oversampling' and 'n_neighbors'
y = data['Loan_Status']
x = data.drop(columns=['Loan_Status'], axis=1)

# Initiating a new class variable for the balanced set
x_bal,y_bal = smote.fit_resample(x,y)

# Verifying the success of a before balancing the data and after
print(y.value_counts())
print(y_bal.value_counts())

1    492
0    192
Name: Loan_Status, dtype: int64
1    492
0    492
Name: Loan_Status, dtype: int64

```

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

Activity 4: Scaling the Data

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

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

```

# performing feature Scaling by normalizing standard deviation & mean of the dataset
# there are different type of values in the columns
sc=StandardScaler()
x_bal=sc.fit_transform(x_bal)

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

```

We will perform scaling only on the input values

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

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

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

Milestone 4: Model Building

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

Activity 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 initialized 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 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 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 initialized 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 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: KNN model

A function named KNN is created and train and test data are passed as the parameters. Inside the function, KNeighborsClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in 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 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 initialized 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)

```

← ML - 0059-Smart Lender -...

```
print(classification_report(y_test,yPred))
```

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

Activity 5: Compare the model

For comparing the above four models compareModel function is defined.

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

1.0
0.8165137614678899
```

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

1.0
0.7339449541284484
```

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

```
0.8299339727883157
0.7706422018348624
```

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

```
0.9478458849886621
0.8119266855045872
```

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

Activity 6: Evaluating performance of the model and saving the model

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

Note: To understand cross validation, refer this link.

<https://towardsdatascience.com/crossvalidation-explained-evaluating-estimator-performance-e51e5430ff85>.

```
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
```

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

Milestone 5: Application Building

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

This section has the following tasks

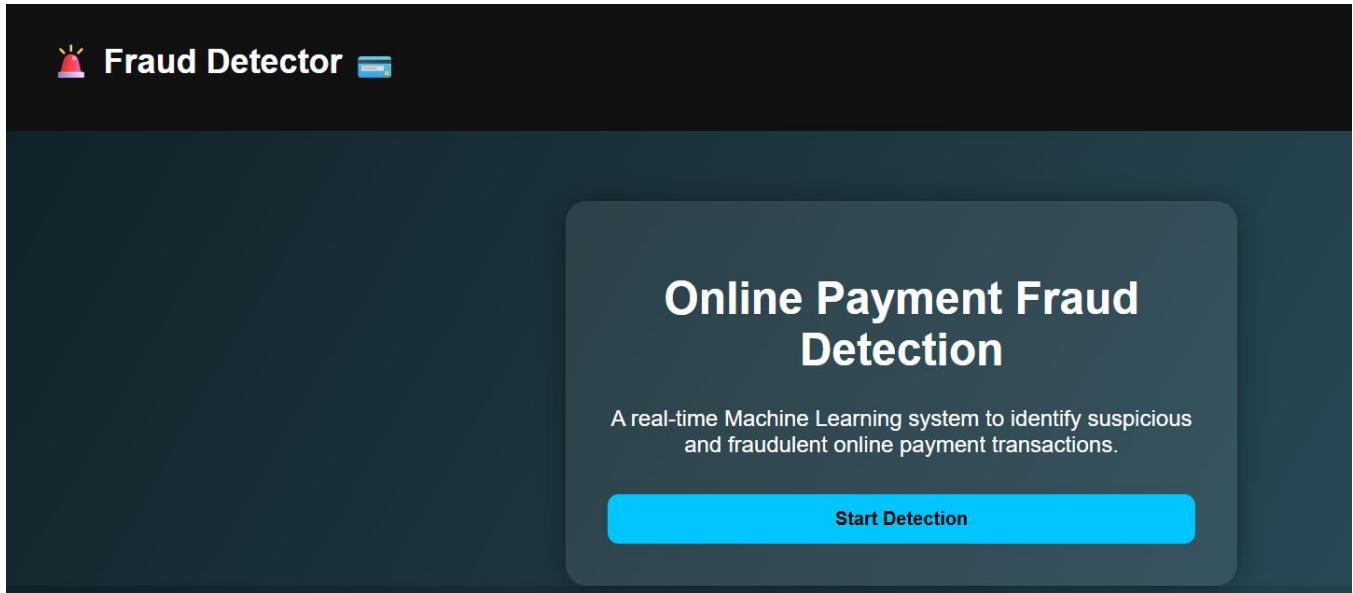
- Building HTML Pages
- Building serverside script

Activity1: Building Html Pages:

For this project create three HTML files namely

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

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



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

Lets look how our predict.html file looks like:

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

Lets look how our submit.html file looks like:

Enter Transaction Details

1

2

5000

4000

6000

1000

4000

0

Analyze Transaction

Activity 2: Build Python code:

Import the libraries

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

Load the saved model. Importing 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 declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with home.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/submit',methods=['POST','GET'])# route to show the predictions in a web UI
def submit():
    # reading the inputs given by the user
    input_feature=[int(x) for x in request.form.values()]
    input_feature = np.reshape(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)

    # 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: Run the 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 top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

