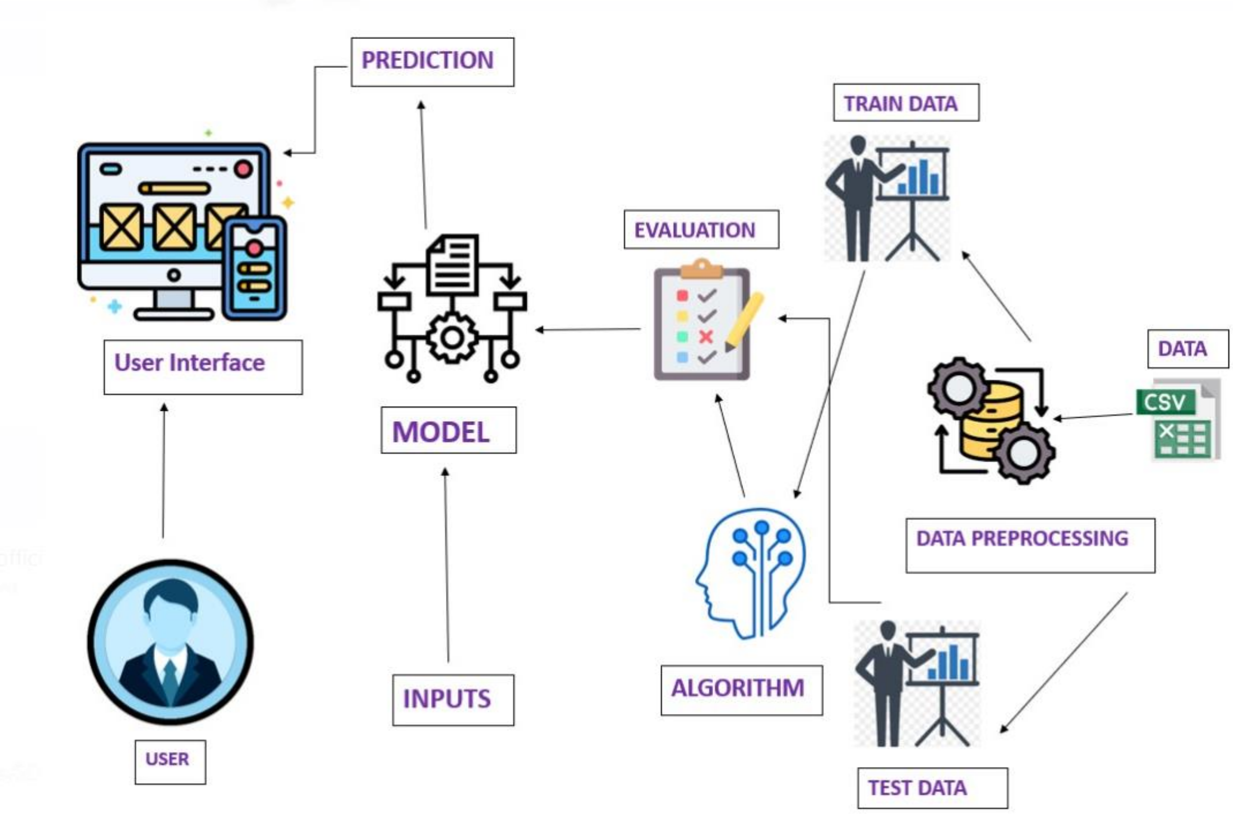
Online Payments Fraud Detection using ML

Project Description:

The growth in internet and e-commerce appears to involve the use of online credit/debit card transactions. The increase in the use of credit / debit cards is causing an increase in fraud. The frauds can be detected through various approaches, yet they lag in their accuracy and its own specific drawbacks. If there are any changes in the conduct of the transaction, the frauds are predicted and taken for further process. Due to large amount of data credit / debit card fraud detection problem is rectified by the proposed method

We will be using classification algorithms such as Decision tree, Random forest, svm, and Extra tree classifier, xgboost Classifier.We will train and test the data with these algorithms. From this the 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:**
    - Refer the link below to download anaconda navigator
    - Link : https://youtu.be/1ra4zH2G4o0
- **Python packages:**
    - Open anaconda prompt as administrator
    - Type"pip install numpy"and click enter.
    - Type"pip install pandas"andclickenter.
    - Type"pip install scikit-learn"andclickenter.
    - Type"pip install matplotlib"andclickenter.
    - Type"pip install scipy"andclickenter.
    - Type"pip install pickle-mixin"andclickenter.
    - Type"pip install seaborn"andclickenter.
    - Type"pipinstallFlask"and click enter.

Prior Knowledge:

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

- **ML Concepts**
    - Supervisedlearning: https://www.javatpoint.com/supervised-machine-learning
    - Unsupervisedlearning: https://www.javatpoint.com/unsupervised-machine-learning
    - Regression and classification
    - Decisiontree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
    - Randomforest:

https://www.javatpoint.com/machine-learning-random-forest-algorithm

- **xgboost Classifier**

  https://www.javatpoint.com/xgboost-classifier-algorithm-for-machine-learning

- Svm:

  https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-

  Evaluationmetrics:

  https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

- **Flask Basics** : https://www.youtube.com/watch?v=lj4I_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 visualisation concepts.

Project Flow:

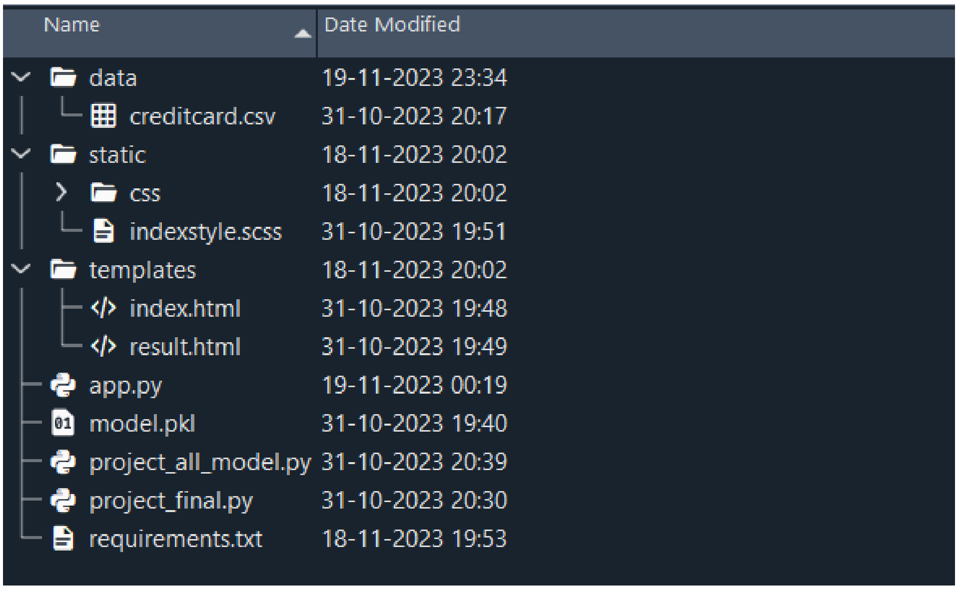
- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

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

- Data collection
  - Collect the dataset or create the dataset
- Visualising and analysing data
  - Importing the libraries
  - Read the Dataset
  - Univariate analysis
  - Bivariate analysis
  - Descriptive analysis
- Data pre-processing
  - Checking for null values
  - Handling outlier
  - Handling categorical(object) data
  - Splitting data into train and test
- Model building
  - Import the model building libraries
  - Initialising the model
  - Training and testing the model
  - Evaluating performance of model
  - Save the model
- Application Building
  - Create an HTML file
  - Build python code

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.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- requirements.txt is a common convention in Python projects to specify the dependencies required for a particular application. It typically contains a list of package names along with their versions. This file is used with package management tools like pip to install the necessary dependencies for a project.
- This lightweight web framework for Python, powers our application deployed seamlessly on Render's cloud platform.

Milestone 1: Data Collection

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 or create the dataset or 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 creditcard.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to downloadthe dataset.

Link:

Milestone 2: Visualising and analysing data

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: Importing the libraries

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

Importing Libraries¶

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
```

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 the csv file.

Here the dataset is loaded by reading the csv file imported through uploading and downloading from kaggle dataset through provided link above.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
```

```
In [2]: import sklearn
        import random
```

```
In [3]: from sklearn.utils import shuffle
```

```
In [4]: d=pd.read_csv('creditcard.csv')
```

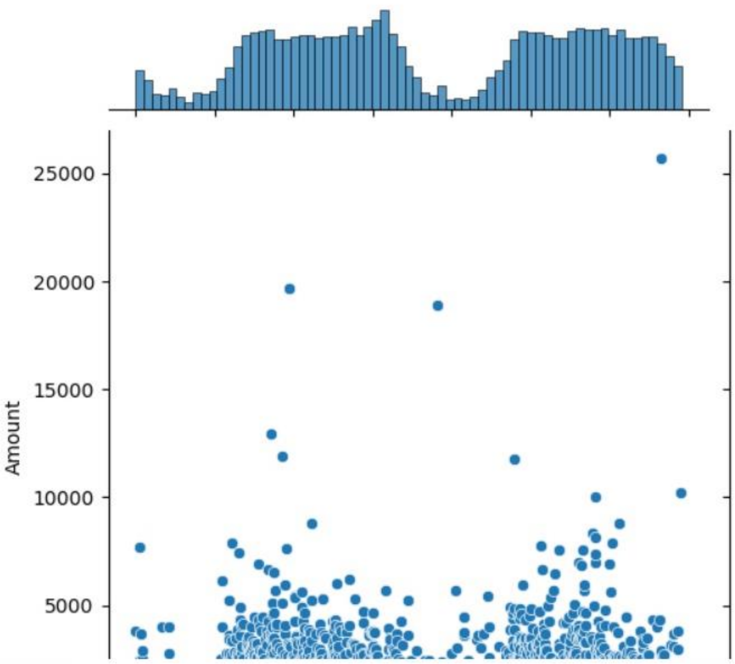
```
In [5]: d
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.06692
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.33984
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.68928
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.17557
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.14126
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.50934
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.01622

Activity 3: Data Exploration

```
In [8]: sns.jointplot(x= 'Time', y= 'Amount', data= d)
Out[8]: <seaborn.axisgrid.JointGrid at 0x1f481a71b50>
```




```
In [9]: class0 = d[d['Class']==0]

        len(class0)
Out[9]: 284315
```

```
In [10]: class1 = d[d['Class']==1]

         len(class1)
Out[10]: 492
```

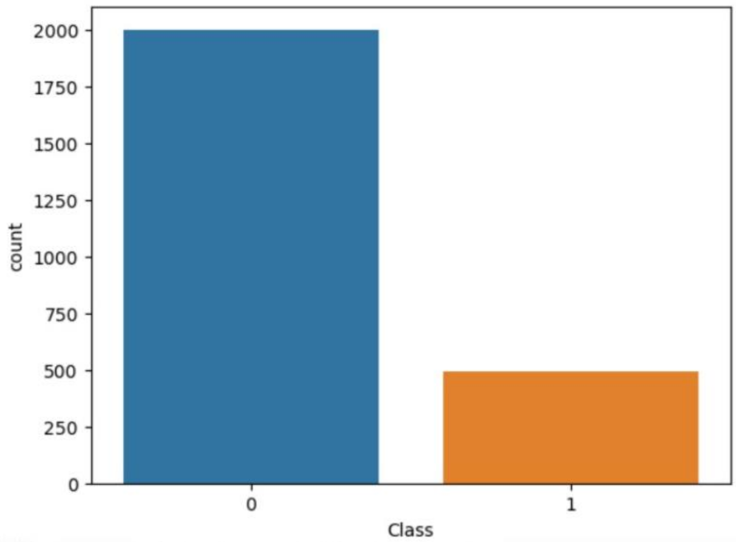
```
In [11]: class0
         temp = shuffle(class0)
```

```
In [13]: frames = [d1, class1]
         df_temp = pd.concat(frames)

         df_temp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2492 entries, 178005 to 281674
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    2492 non-null   float64
 1   V1      2492 non-null   float64
 2   V2      2492 non-null   float64
 3   V3      2492 non-null   float64
 4   V4      2492 non-null   float64
 5   V5      2492 non-null   float64
 6   V6      2492 non-null   float64
 7   V7      2492 non-null   float64
 8   V8      2492 non-null   float64
 9   V9      2492 non-null   float64
 10  V10     2492 non-null   float64
 11  V11     2492 non-null   float64
 12  V12     2492 non-null   float64
 13  V13     2492 non-null   float64
 14  V14     2492 non-null   float64
 15  V15     2492 non-null   float64
 16  V16     2492 non-null   float64
 17  V17     2492 non-null   float64
 18  V18     2492 non-null   float64
 19  V19     2492 non-null   float64
 20  V20     2492 non-null   float64
```

The joint plot helps uncover potential patterns or trends between transaction time and amount. The central scatter plot showcases the concentration of data points and any noticeable correlations. The histograms on the margins offer individual insights into the distribution of 'Time' and 'Amount’.

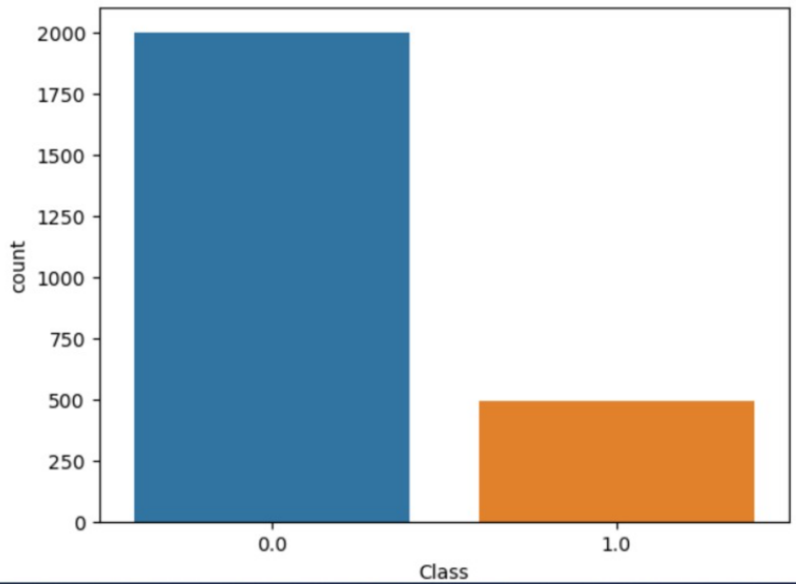
```
# Assuming 'Class' is a column in your DataFrame 'df'
sns.countplot(x='Class', data=df)

# Display the plot
plt.show()
```



Utilizes a count plot to visualize the distribution of fraud (Class 1) and non-fraud (Class 0) instances.Essential for understanding the class imbalance in the dataset.

```python
# Assuming 'd' is your data dictionary and 'names' is a list of column names
data = pd.DataFrame(d, columns=names)
sns.countplot(x='Class', data=data)
```

```
<Axes: xlabel='Class', ylabel='count'>
```



In [23]:
```python
data.describe()
```

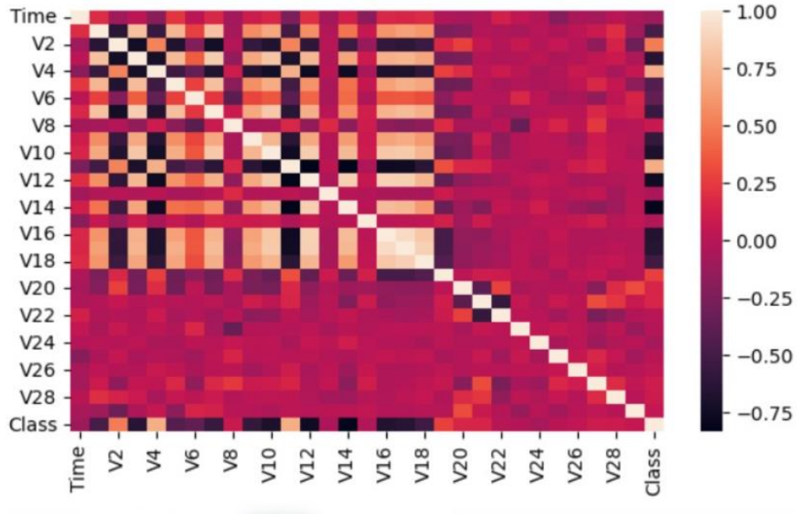
Out[23]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21
count	2492.000000	2492.000000	2492.000000	2492.000000	2492.000000	2492.000000	2492.000000	2492.000000	2492.000000	2492.000000	...	2492.000000
mean	91236.181782	-0.946977	0.727373	-1.383817	0.860228	-0.630729	-0.251022	-1.083121	0.074222	-0.474762	...	0.130843
std	47733.179149	3.971649	2.854086	4.409868	2.548462	2.917241	1.559170	4.018396	3.271067	1.813345	...	1.891749
min	74.000000	-30.821436	-35.616754	-31.103685	-4.345575	-22.105532	-13.360241	-43.557242	-41.044261	-13.434066	...	-22.797604
25%	50435.750000	-1.402012	-0.451572	-1.682002	-0.660915	-0.968541	-1.010898	-0.914349	-0.212450	-1.062483	...	-0.217592
50%	81924.000000	-0.298502	0.275900	-0.238059	0.278382	-0.194207	-0.395484	-0.087376	0.054171	-0.194030	...	0.020252
75%	137127.750000	1.217464	1.210635	0.799702	1.401317	0.537905	0.299020	0.488527	0.487872	0.491878	...	0.299808
max	172734.000000	2.349591	22.057729	3.664946	12.114672	18.611287	6.474115	9.303732	20.007208	7.929051	...	27.202839

[24]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2492 entries, 0 to 2491
Data columns (total 31 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Time    2492 non-null   float64
 1   V1      2492 non-null   float64
 2   V2      2492 non-null   float64
 3   V3      2492 non-null   float64
 4   V4      2492 non-null   float64
 5   V5      2492 non-null   float64
 6   V6      2492 non-null   float64
 7   V7      2492 non-null   float64
 8   V8      2492 non-null   float64
 9   V9      2492 non-null   float64
 10  V10     2492 non-null   float64
 11  V11     2492 non-null   float64
 12  V12     2492 non-null   float64
 13  V13     2492 non-null   float64
 14  V14     2492 non-null   float64
 15  V15     2492 non-null   float64
 16  V16     2492 non-null   float64
 17  V17     2492 non-null   float64
 18  V18     2492 non-null   float64
 19  V19     2492 non-null   float64
 20  V20     2492 non-null   float64
 21  V21     2492 non-null   float64
 22  V22     2492 non-null   float64
```

```
In [25]: plt.figure(figsize=(7,4))
         sns.heatmap(data.corr())

Out[25]: <Axes: >
```



The heatmap provides a quick overview of how each feature correlates with every other feature.

Darker colors indicate stronger correlations (positive or negative), while lighter colors suggest weaker or no correlations.

Milestone 3: Data Sampling and Preprocessing

```
In [26]: import math
         import sklearn.preprocessing

In [27]: from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import accuracy_score , classification_report, confusion_matrix, precision_recall_curve, f1_score, auc

In [28]: X_train, X_test, y_train, y_test = train_test_split(data.drop('Class', axis=1), data['Class'], test_size=0.3, random_state=42)

In [33]: d_scaled = pd.concat([scaled_col, d_temp], axis =1)

         d_scaled.head()

Out[33]:
```

	Time	Amount	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22	V23
0	-0.171219	-0.385323	1.157457	0.260843	0.383889	0.574981	-0.228116	-0.395403	-0.093067	0.071054	...	-0.111829	-0.195816	-0.568083	0.172627
1	1.587073	-0.374888	-3.028582	3.234577	-3.143391	-0.756203	-1.067800	-0.591526	-1.369818	2.654259	...	-0.033207	0.488533	1.102256	0.231338
2	0.177539	0.260394	0.202219	1.077172	-1.276218	1.737891	0.890508	-1.015050	1.884316	-0.552673	...	0.331340	-0.202064	-0.154461	0.122605
3	-0.415081	-0.365256	1.273667	0.348576	0.155665	0.610139	-0.179551	-0.860330	0.100726	-0.195347	...	-0.045382	-0.305978	-0.878975	0.078359
4	1.024144	0.169170	2.041205	-1.675773	-1.346314	-1.530424	-1.056192	-0.568938	-0.701911	-0.212398	...	-0.188928	-0.306894	-0.664945	0.177506

it stands for scikit-learn, and it provides simple and efficient tools for data analysis and modeling,

including various machine learning algorithms for classification, regression, clustering, and more.

sklearn is part of the broader ecosystem for scientific computing and data science in Python.

```
In [34]: y = data['Class']

         d_scaled.head()
```

Out[34]:

	Time	Amount	V1	V2	V3	V4	V5	V6	V7	V8	...	V20	V21	V22	V23
0	-0.171219	-0.385323	1.157457	0.260843	0.383889	0.574981	-0.228116	-0.395403	-0.093067	0.071054	...	-0.111829	-0.195816	-0.568083	0.172627
1	1.587073	-0.374888	-3.028582	3.234577	-3.143391	-0.756203	-1.067800	-0.591526	-1.369818	2.654259	...	-0.033207	0.488533	1.102256	0.231338
2	0.177539	0.260394	0.202219	1.077172	-1.276218	1.737891	0.890508	-1.015050	1.884316	-0.552673	...	0.331340	-0.202064	-0.154461	0.122605
3	-0.415081	-0.365256	1.273667	0.348576	0.155665	0.610139	-0.179551	-0.860330	0.100726	-0.195347	...	-0.045382	-0.305978	-0.878975	0.078359
4	1.024144	0.169170	2.041205	-1.675773	-1.346314	-1.530424	-1.056192	-0.568938	-0.701911	-0.212398	...	-0.188928	-0.306894	-0.664945	0.177506

Activity1: Dimensionality Reduction

```
[35]: from sklearn.decomposition import PCA
```

```
[36]: pca = PCA(n_components=7)
```

```
[37]: X_temp_reduced = pca.fit_transform(d_scaled)
```

```
[38]: pca.explained_variance_ratio_

      pca.explained_variance_
```

```
:[38]: array([106.69768306,  14.50512346,  10.74676857,   5.46473049,
                4.76769311,   4.00649798,   2.61482255])
```

```
[39]: names=['Time','Amount','Transaction Method','Transaction Id','Location','Type of Card','Bank']
```

```
[40]: X_reduced= pd.DataFrame(X_temp_reduced,columns=names)
      X_reduced.head()
```

:[40]:

	Time	Amount	Transaction Method	Transaction Id	Location	Type of Card	Bank
0	-3.860818	-0.188792	-0.051130	0.168795	-0.439317	-0.705464	0.137812
1	-1.390245	2.979890	5.342280	1.269209	0.731550	-2.011317	-0.304936
2	-3.048058	-0.643302	-1.072993	3.540518	-0.562522	0.381300	-0.249209
3	-3.876920	-0.160903	-0.087560	0.092488	-0.475221	-0.832521	0.140371
4	-4.049599	-0.005896	0.336496	-0.423122	-0.655424	-0.591550	2.789950

Milestone 4: Model Training and Evaluation

```
In [41]: Y=d_scaled['Class']
```

```
In [42]: new_data=pd.concat([X_reduced,Y],axis=1)
         new_data.head()
         new_data.shape
```

```
Out[42]: (2492, 8)
```

```
In [43]: X_train, X_test, y_train, y_test= train_test_split(X_reduced, d_scaled['Class'], test_size = 0.30, random_state = 42)

         X_train.shape, X_test.shape
```

```
Out[43]: ((1744, 7), (748, 7))
```

```
Out[46]: {'C': 10, 'penalty': 'l2'}
```

```
In [47]: y_pred_lr3=grid_lr.predict(X_test)
         print(classification_report(y_test,y_pred_lr3))
```

```
              precision    recall  f1-score   support

         0.0       0.98      1.00      0.99       603
         1.0       0.98      0.90      0.94       145

    accuracy                           0.98       748
   macro avg       0.98      0.95      0.96       748
weighted avg       0.98      0.98      0.98       748
```

Support Vector Machine Classifier¶

A function named SupportVector is created and train and test data are passed as the parameters. Inside the function, the SupportVectorClassifier 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, confusion matrix and classification report is done

```
In [48]: from sklearn.svm import SVC
         svc=SVC(kernel='rbf')
         svc.fit(X_train,y_train)
         y_pred_svc=svc.predict(X_test)
         y_pred_svc
         print(classification_report(y_test,y_pred_svc))

                       precision    recall  f1-score   support

                  0.0       0.97      1.00      0.98       603
                  1.0       0.98      0.88      0.93       145

             accuracy                           0.97       748
            macro avg       0.97      0.94      0.96       748
         weighted avg       0.97      0.97      0.97       748

In [49]: print(confusion_matrix(y_test,y_pred_svc))

         [[600   3]
          [ 17 128]]

In [50]: from sklearn.model_selection import GridSearchCV
         parameters = [ {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.1, 1, 0.01, 0.0001 ,0.001]}]
         grid_search = GridSearchCV(estimator = svc,
                                    param_grid = parameters,
                                    scoring = 'accuracy',
                                    n_jobs = -1)
         grid_search = grid_search.fit(X_train, y_train)
         best_accuracy = grid_search.best_score_
         best_parameters = grid_search.best_params_
         print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
         print("Best Parameters:", best_parameters)

         svc_param=SVC(kernel='rbf',gamma=0.01,C=100)
         svc_param.fit(X_train,y_train)
         y_pred_svc2=svc_param.predict(X_test)
         print(classification_report(y_test,y_pred_svc2))

         Best Accuracy: 97.13 %
         Best Parameters: {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
                       precision    recall  f1-score   support

                  0.0       0.97      0.99      0.98       603
                  1.0       0.96      0.89      0.92       145

             accuracy                           0.97       748
            macro avg       0.97      0.94      0.95       748
         weighted avg       0.97      0.97      0.97       748
```

Decision tree Classifier

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

```
In [51]: from sklearn.tree import DecisionTreeClassifier
         dtree=DecisionTreeClassifier()
         dtree.fit(X_train,y_train)
         y_pred_dtree=dtree.predict(X_test)
         print(classification_report(y_test,y_pred_dtree))

                       precision    recall  f1-score   support

                  0.0       0.98      0.97      0.98       603
                  1.0       0.89      0.92      0.91       145

             accuracy                           0.96       748
            macro avg       0.93      0.95      0.94       748
         weighted avg       0.96      0.96      0.96       748
```

```
In [52]: print(confusion_matrix(y_test,y_pred_dtree))

         [[586  17]
          [ 11 134]]
```

```
In [53]: d_tree_param=DecisionTreeClassifier()
         tree_parameters={'criterion':['gini','entropy'],'max_depth':list(range(2,4,1)),
                          'min_samples_leaf':list(range(5,7,1))}
         grid_tree=GridSearchCV(d_tree_param,tree_parameters)
         grid_tree.fit(X_train,y_train)

         y_pred_dtree2=grid_tree.predict(X_test)

         print(classification_report(y_test,y_pred_dtree2))

                       precision    recall  f1-score   support

                  0.0       0.98      0.99      0.99       603
                  1.0       0.97      0.91      0.94       145

             accuracy                           0.98       748
            macro avg       0.97      0.95      0.96       748
         weighted avg       0.98      0.98      0.98       748
```

Random Forest classifier:

A function named RandomForest is created and train and test data are passed as the parameters. Inside the function, the RandomForestClassifier 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.

In [54]:
```
from sklearn.ensemble import RandomForestClassifier
randomforest=RandomForestClassifier(n_estimators=5)
randomforest.fit(X_train,y_train)
y_pred_rf=randomforest.predict(X_test)
print(confusion_matrix(y_test,y_pred_rf))

print(classification_report(y_test,y_pred_rf))
```

```
[[593  10]
 [ 14 131]]
              precision    recall  f1-score   support

         0.0       0.98      0.98      0.98       603
         1.0       0.93      0.90      0.92       145

    accuracy                           0.97       748
   macro avg       0.95      0.94      0.95       748
weighted avg       0.97      0.97      0.97       748
```

In [55]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train,y_train)
y_pred_knn=knn.predict(X_test)
y_pred_knn

print(classification_report(y_test,y_pred_knn))
```

```
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98       603
         1.0       0.96      0.88      0.92       145

    accuracy                           0.97       748
   macro avg       0.96      0.94      0.95       748
weighted avg       0.97      0.97      0.97       748
```

In [56]:
```
print(confusion_matrix(y_test,y_pred_knn))
```

```
[[597   6]
 [ 17 128]]
```

[60]:
```
knn.fit(X_train,y_train)
pred_knn2 = knn.predict(X_test)
```

[61]:
```
print('WITH K=3')
print('\n')
print(confusion_matrix(y_test,pred_knn2))
print('\n')
print(classification_report(y_test,pred_knn2))
```

```
WITH K=3


[[599   4]
 [ 18 127]]


              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98       603
         1.0       0.97      0.88      0.92       145

    accuracy                           0.97       748
   macro avg       0.97      0.93      0.95       748
weighted avg       0.97      0.97      0.97       748
```

xgboost Classifier:

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, the xgboostClassifier 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, confusion matrix and classification report is done.

```
In [62]:
from xgboost import XGBClassifier
xgb=XGBClassifier()
```

```
In [63]: xgb.fit(X_train,y_train)
y_pred_xg=xgb.predict(X_test)
print(classification_report(y_test,y_pred_xg))
```

```
              precision    recall  f1-score   support

         0.0       0.98      0.99      0.98       603
         1.0       0.94      0.91      0.92       145

    accuracy                           0.97       748
   macro avg       0.96      0.95      0.95       748
weighted avg       0.97      0.97      0.97       748
```

```
In [64]: import lightgbm as lgb
```

```
In [65]: lgb_train = lgb.Dataset(X_train, y_train, free_raw_data= False)

lgb_test = lgb.Dataset(X_test, y_test, reference=lgb_train, free_raw_data= False)

parameters = {'num_leaves': 2**8,
              'learning_rate': 0.1,
              'is_unbalance': True,
              'min_split_gain': 0.1,
              'min_child_weight': 1,
              'reg_lambda': 1,
              'subsample': 1,
              'objective':'binary',
              #'device': 'gpu', # comment this line if you are not using GPU
              'task': 'train'
              }
num_rounds = 300

lgb_train = lgb.Dataset(X_train, y_train)

lgb_test = lgb.Dataset(X_test, y_test)

clf = lgb.train(parameters, lgb_train, num_boost_round=num_rounds)

y_prob = clf.predict(X_test)
y_pred = sklearn.preprocessing.binarize(np.reshape(y_prob, (-1,1)), threshold= 0.5)

accuracy_score(y_test, y_pred)

print(classification_report(y_test,y_pred))
```

ROC Curve:

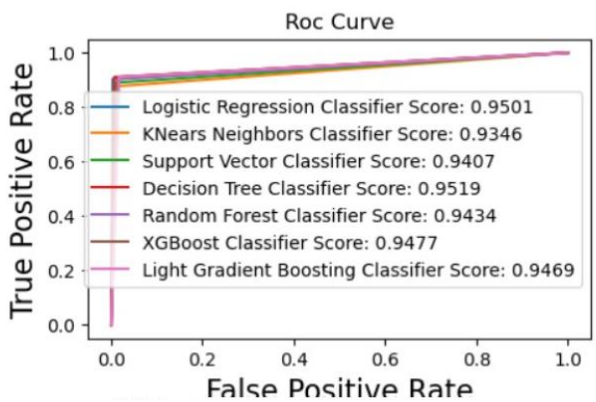
Plots Receiver Operating Characteristic (ROC) curves for each classifier.

Visualizes the trade-off between true positive rate and false positive rate for model comparison.

```
In [66]: from sklearn.metrics import roc_curve,roc_auc_score
         lg_fpr,lg_tpr,lg_threshold=roc_curve(y_test,y_pred_lr3)
         svc_fpr,svc_tpr,svc_threshold=roc_curve(y_test,y_pred_svc2)
         dtree_fpr,dtree_tpr,dtree_threshold=roc_curve(y_test,y_pred_dtree2)
         rf_fpr,rf_tpr,rf_threshold=roc_curve(y_test,y_pred_rf)
         knn_fpr,knn_tpr,rf_threshold=roc_curve(y_test,pred_knn2)
         xg_fpr,xg_tpr,xg_threshold=roc_curve(y_test,y_pred_xg)
         lgb_fpr,lgb_tpr,lgb_threshold=roc_curve(y_test,y_pred)
```

The code compares different classifiers and displays their performance metrics.

```
In [70]: plt.figure(figsize=(5,3))
         plt.title("Roc Curve")
         plt.plot(lg_fpr,lg_tpr, label='Logistic Regression Classifier Score: {:.4f}'.format(roc_auc_score(y_test, y_pred_lr3)))
         plt.plot(knn_fpr,knn_tpr, label='KNears Neighbors Classifier Score: {:.4f}'.format(roc_auc_score(y_test, pred_knn2)))
         plt.plot(svc_fpr, svc_tpr, label='Support Vector Classifier Score: {:.4f}'.format(roc_auc_score(y_test, y_pred_svc2)))
         plt.plot(dtree_fpr, dtree_tpr, label='Decision Tree Classifier Score: {:.4f}'.format(roc_auc_score(y_test, y_pred_dtree2)))
         plt.plot(rf_fpr,rf_tpr, label='Random Forest Classifier Score: {:.4f}'.format(roc_auc_score(y_test, y_pred_rf)))
         plt.plot(xg_fpr,xg_tpr, label='XGBoost Classifier Score: {:.4f}'.format(roc_auc_score(y_test, y_pred_xg)))
         plt.plot(lgb_fpr,lgb_tpr, label='Light Gradient Boosting Classifier Score: {:.4f}'.format(roc_auc_score(y_test, y_pred)))
         plt.xlabel('False Positive Rate', fontsize=16)
         plt.ylabel('True Positive Rate', fontsize=16)
         plt.legend()
         plt.show()
```



```
In [73]: from sklearn.decomposition import PCA
```

```
In [74]: pca = PCA(n_components=7)

         X_temp_reduced = pca.fit_transform(d_scaled)

         pca.explained_variance_ratio_

         pca.explained_variance_
```

```
Out[74]: array([106.69768306,  14.50512346,  10.74676857,   5.46473051,
                  4.7676926 ,   4.00649845,   2.61477527])
```

```
In [75]: names=['Time','Amount','Transaction Method','Transaction Id','Location','Type of Card','Bank']

         X_reduced= pd.DataFrame(X_temp_reduced,columns=names)
         X_reduced.head()
```

Out[75]:

	Time	Amount	Transaction Method	Transaction Id	Location	Type of Card	Bank
0	-3.860818	-0.188805	-0.051146	0.169318	-0.439922	-0.706279	0.152602
1	-1.390245	2.979890	5.342285	1.268939	0.730881	-2.011175	-0.294796
2	-3.048058	-0.643304	-1.072999	3.540501	-0.562806	0.381967	-0.250936
3	-3.876920	-0.160911	-0.087567	0.092719	-0.475687	-0.833864	0.152510
4	-4.049599	-0.005868	0.336549	-0.423728	-0.652822	-0.589819	2.756834

```
In [76]: Y=d_scaled['Class']

         new_data=pd.concat([X_reduced,Y],axis=1)
         new_data.head()
         new_data.shape
```

```
Out[76]: (2492, 8)
```

```
In [77]: new_data.to_csv('finaldata.csv')

         X_train, X_test, y_train, y_test= train_test_split(X_reduced, d_scaled['Class'], test_size = 0.30, random_state = 42)

         X_train.shape, X_test.shape
```

```
Out[77]: ((1744, 7), (748, 7))
```

```
In [78]: from sklearn.metrics import classification_report,confusion_matrix
```

```
In [79]: from sklearn.svm import SVC
         svc=SVC(kernel='rbf',probability=True)
         svc.fit(X_train,y_train)
         y_pred_svc=svc.predict(X_test)
         y_pred_svc
```

```
Out[79]: array([0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 1., 1., 0.,
                0., 1., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0.,
                0., 0., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 1.,
                0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                1., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
                0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 1.,
                0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
                1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0.,
                0., 0., 1., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0., 0., 0., 1., 0.,
                0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0.,
                0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
                0., 0., 0., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
                0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0.,
                0., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1.,
                1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0.,
                0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 1., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 1., 0., 0., 0., 0., 0.,
```

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

```
[80]: type(X_test)
      X_test.to_csv('testing.csv')
      from sklearn.model_selection import GridSearchCV
      parameters = [ {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.1, 1, 0.01, 0.0001 ,0.001]}
      grid_search = GridSearchCV(estimator = svc,
                                 param_grid = parameters,
                                 scoring = 'accuracy',
                                 n_jobs = -1)
      grid_search = grid_search.fit(X_train, y_train)
      best_accuracy = grid_search.best_score_
      best_parameters = grid_search.best_params_
      print("Best Accuracy: {:.2f} %".format(best_accuracy*100))
      print("Best Parameters:", best_parameters)

      svc_param=SVC(kernel='rbf',gamma=0.01,C=100,probability=True)
      svc_param.fit(X_train,y_train)

      Best Accuracy: 97.13 %
      Best Parameters: {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}

[80]:           SVC
      SVC(C=100, gamma=0.01, probability=True)
```

```
In [81]: import pickle
         # Saving model to disk
         pickle.dump(svc_param, open('model.pkl','wb'))

         model=pickle.load(open('model.pkl','rb'))
```

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 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

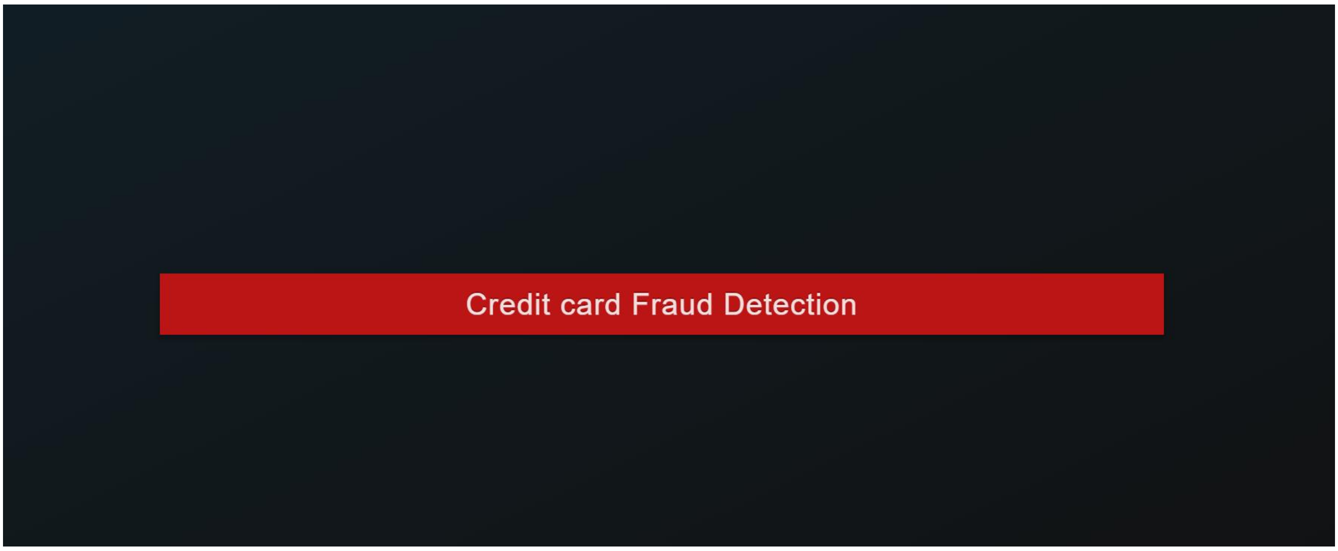
Activity1: Building Html Pages:

For this project create three HTML files namely

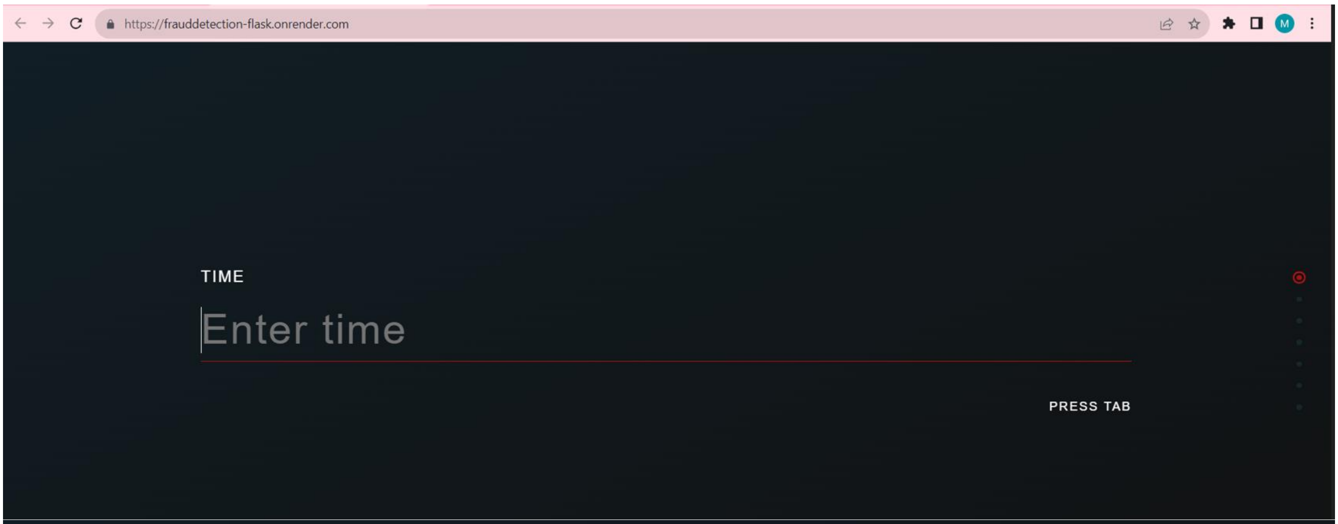
- index.html
- result.html

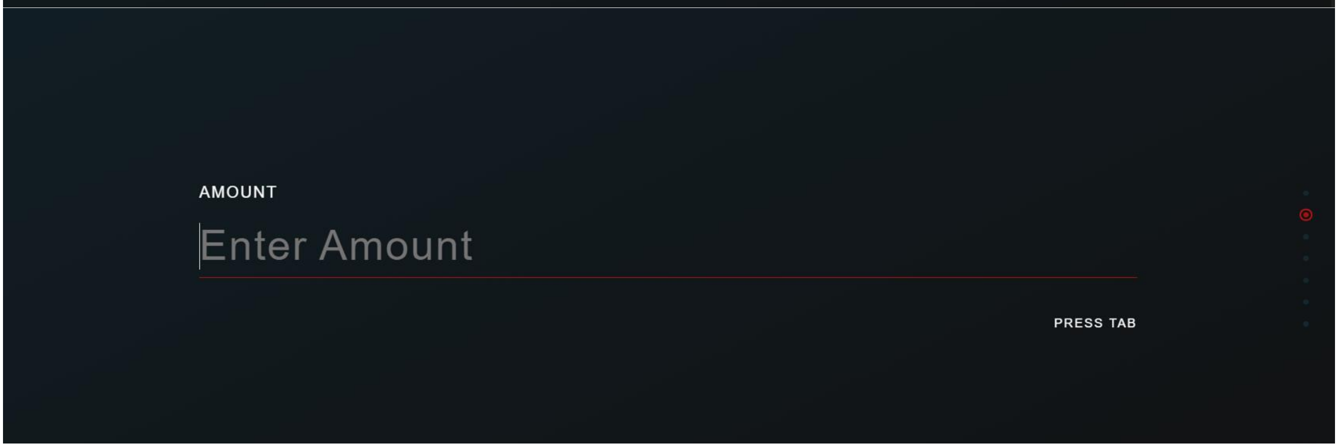
and save them in the templates folder.

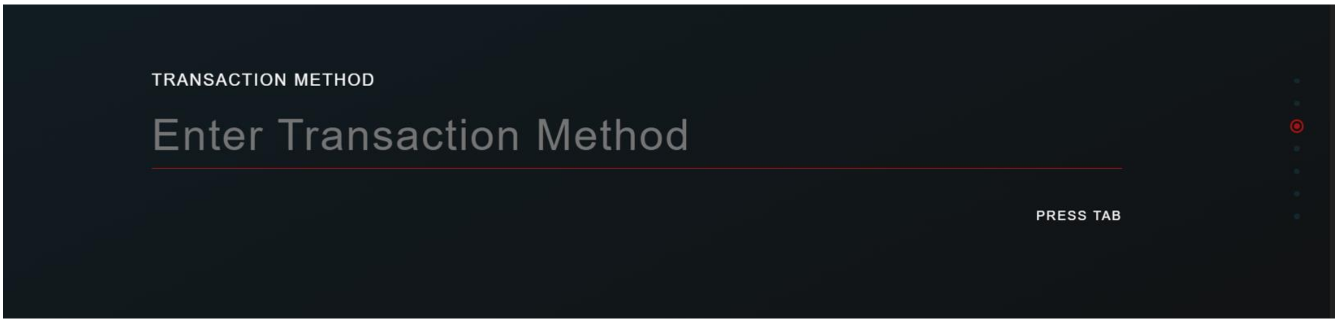
Let’s see how our index.html page looks like:

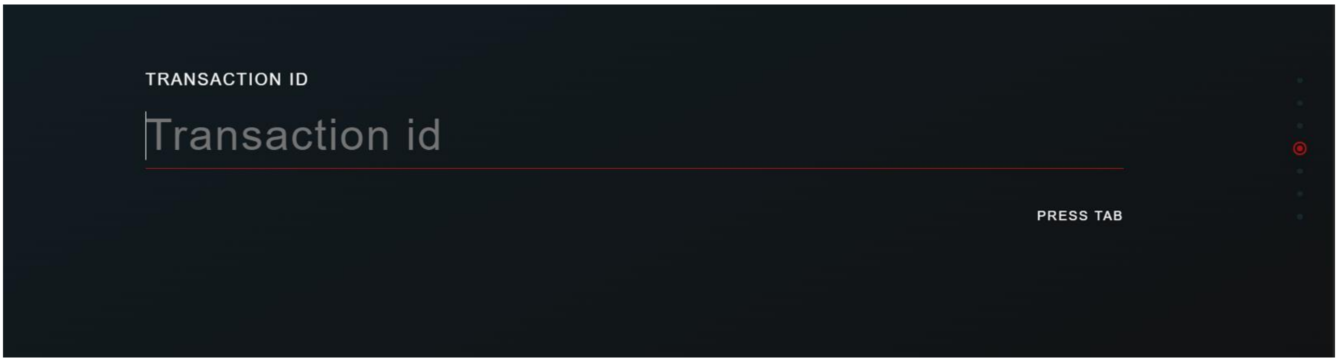


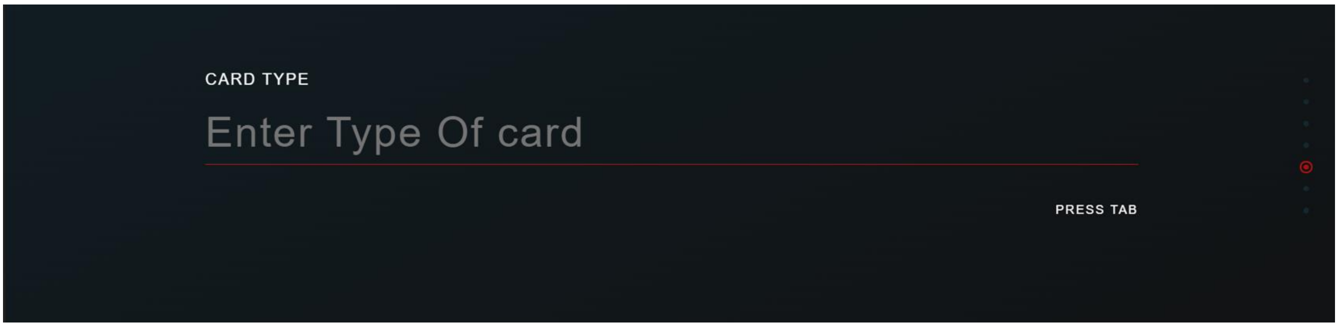
By pressing tab button we will redirecting to different slides asking input of transaction details of respective columns.

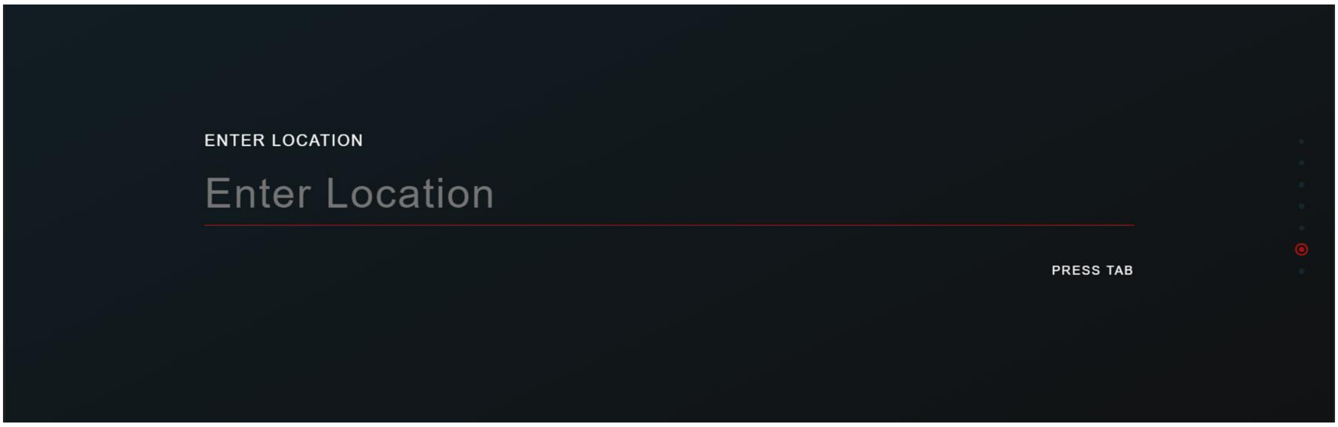








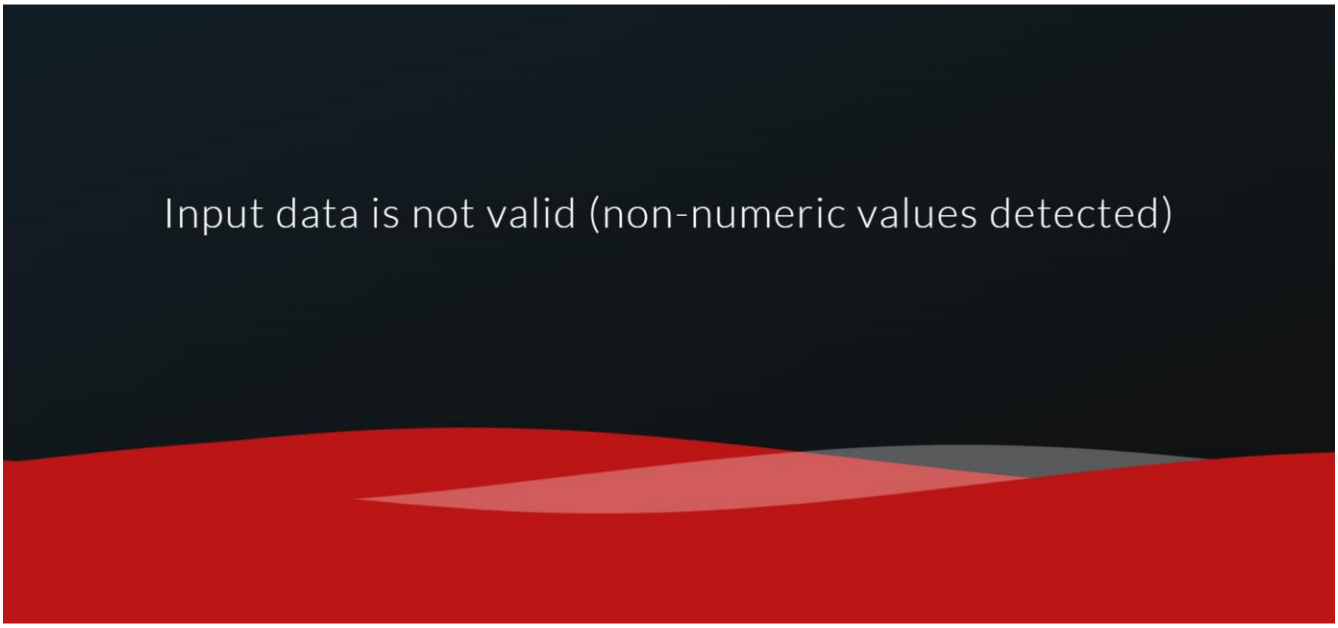




We need to input the data according the transactions and it will be predicting whether it is fraud.

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

Let's look how our result.html file looks like:



As we given non numeric input it gives the output as above.

If the given transaction details matches with the conditions of fraud then it predicts as fraud and vice versa with not fraud.

Activity 2: Build Python code:

Import the libraries

```
import numpy as np
from flask import Flask, request, jsonify, render_template
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__)
# prediction function
def ValuePredictor(to_predict_list):
    to_predict = np.array(to_predict_list).reshape(1, 7)
    loaded_model = pickle.load(open("model.pkl", "rb"))
    result = loaded_model.predict(to_predict)
    return result[0]
```

Render HTML page:

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('/')
def home():
    return render_template("index.html")

@app.route('/predict', methods=['POST', 'GET'])
def predict():
    if request.method == 'POST':
        to_predict_list = request.form.to_dict()
        to_predict_list = list(to_predict_list.values())

        try:
            to_predict_list = list(map(float, to_predict_list))
            result = ValuePredictor(to_predict_list)
            if int(result) == 1:
                prediction = 'Given transaction is fraudulent'
            else:
                prediction = 'Given transaction is NOT fraudulent'
        except ValueError:
            prediction = 'Input data is not valid (non-numeric values detected)'

        return render_template("result.html", prediction=prediction)
```

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 the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
In [1]: runfile('D:/fraudapp/app.py', wdir='D:/fraudapp')
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production
deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
 * Restarting with watchdog (windowsapi)
```

Finally deployed the flask application in render through the bash and linux commands.