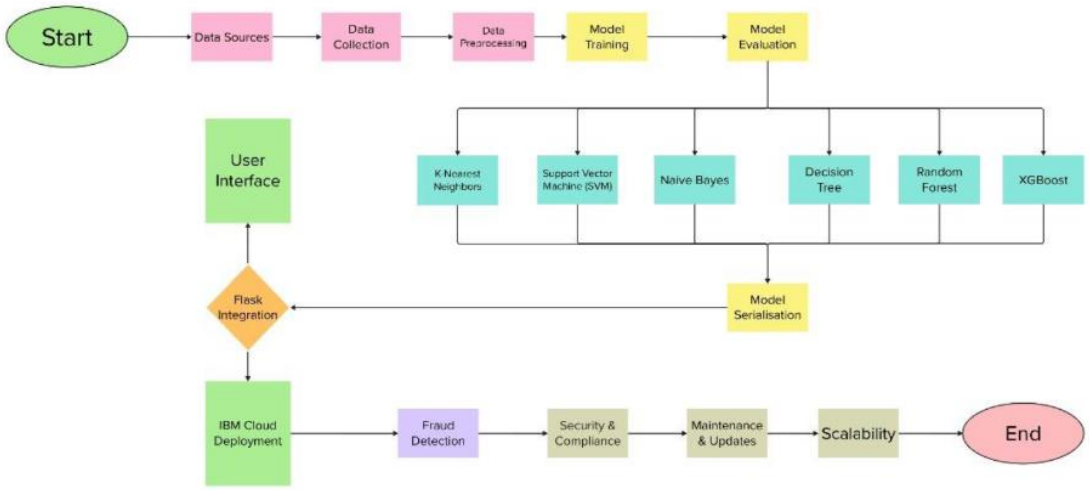
Project Description:-

The surge in internet usage and e-commerce has led to a significant uptick in online credit/debit card transactions, consequently fueling a rise in fraud. Detecting such fraud is challenging, with various approaches available but limited accuracy and specific drawbacks. The proposed method addresses this issue by anticipating and processing potential fraud cases through scrutiny of changes in transaction conduct.

To enhance credit/debit card fraud detection, the approach employs classification algorithms like Decision Tree, Random Forest, SVM, Extra Tree Classifier, and XGBoost. Training and testing with these algorithms identify the most effective model, saved in pkl format. The subsequent steps involve Flask integration and deployment on IBM, completing the strategy for bolstering fraud detection in online transactions.

Technical Architecture:-



Pre requisites:

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

1) **Anaconda navigator:-**
   Refer the link below to download anaconda navigator
   Link : https://youtu.be/1ra4zH2G4o0

2) **Python Packages:-**
   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.

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

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

   - o xgboost Classifier

     https://www.javatpoint.com/xgboost-classifier-algorithm-for-machine-le arning
   - o Svm: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to- understand-the-math-behind-Svm/
   - o Extra tree classifier:

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

   - o Evaluationmetrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-eva luation-error-metrics/
   - o 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

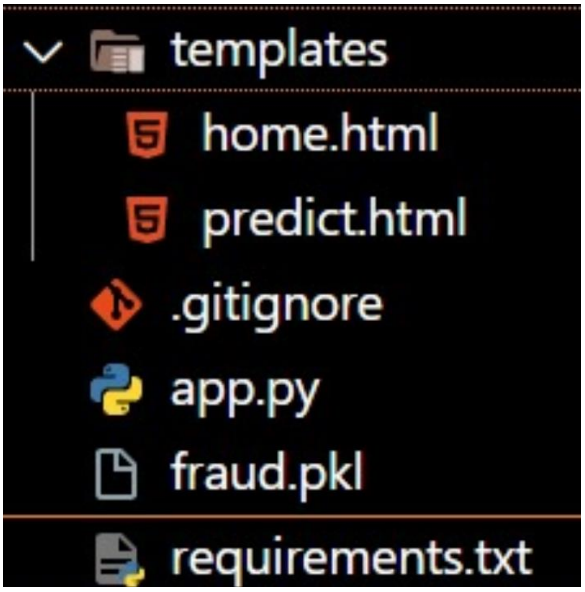
To accomplish this, we have to complete all the activities listed below,

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

Milestone 1: Data Collection

Machine learning relies extensively on data, constituting a pivotal factor that enables the training of algorithms. This section provides the opportunity to download the necessary dataset, a critical step in facilitating the training process for algorithms.

Collect the dataset or create the dataset or Download the dataset:

Various widely used open platforms are available for data collection, such as Kaggle.com and the UCI repository. In our project, we specifically employed the dataset named PS_20174392719_1491204439457_logs.csv, obtained through a download from Kaggle.com. To access and download the dataset, please consult the provided link below.

Link: https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

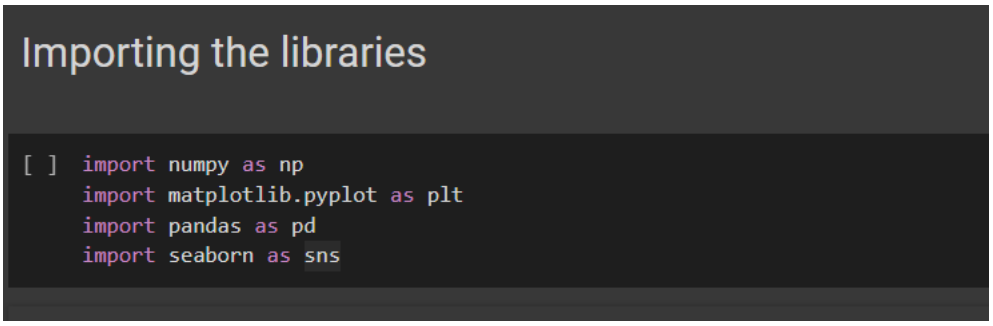
Milestone 2: Visualising and analysing data

Now that the dataset has been obtained, let's read and grasp the data effectively, employing visualization techniques and analysis methods. This will deepen our understanding of the dataset, offering valuable insights for additional exploration.

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 fivethirtyeight.



Activity 2: Read the Dataset

The dataset may come in various formats like .csv, Excel, .txt, .json, etc., and Pandas offers the `read_csv()` function for CSV files, requiring the file path as a parameter. Equivalent functions like `read_excel()`, `read_table()`, or `read_json()` can be used for other formats, providing a flexible solution for data loading and manipulation in Pandas.

```
[ ] df = pd.read_csv("Data.csv")
    df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	TRANSFER	281425.06	C310276293	0.00	0.0	C1831477404	299995.48	247063.16	0	0
1	1	TRANSFER	18561.23	C1494277749	0.00	0.0	C1629911510	192136.35	15375.37	0	0
2	1	TRANSFER	39069.42	C1091107430	0.00	0.0	C22805895	238390.23	651524.92	0	0
3	1	TRANSFER	63186.93	C521342639	0.00	0.0	C1123629720	150527.05	46393.85	0	0
4	1	TRANSFER	419757.69	C2114629072	0.00	0.0	C998351292	559328.89	1015132.48	0	0
...	...	...	...	...	...	...	...	...	...	...	...
2636	95	CASH_OUT	56745.14	C526144262	56745.14	0.0	C79051264	51433.88	108179.02	1	0
2637	95	TRANSFER	33676.59	C732111322	33676.59	0.0	C1140210295	0.00	0.00	1	0
2638	95	CASH_OUT	33676.59	C1000086512	33676.59	0.0	C1759363094	0.00	33676.59	1	0
2639	95	TRANSFER	87999.25	C927181710	87999.25	0.0	C757947873	0.00	0.00	1	0
2640	95	CASH_OUT	87999.25	C409531429	87999.25	0.0	C1827219533	0.00	87999.25	1	0

2641 rows × 11 columns

The `df.info()` code provides a concise summary of the DataFrame `df`. It displays essential information such as the data types, non-null counts, and memory usage. This is valuable for a quick overview of the dataset's structure and characteristics.

```
[ ]  df.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 2641 entries, 0 to 2640
     Data columns (total 11 columns):
      #   Column          Non-Null Count  Dtype
     ---  ------          --------------  -----
      0   step            2641 non-null   int64
      1   type            2641 non-null   object
      2   amount          2641 non-null   float64
      3   nameOrig        2641 non-null   object
      4   oldbalanceOrg   2641 non-null   float64
      5   newbalanceOrig  2641 non-null   float64
      6   nameDest        2641 non-null   object
      7   oldbalanceDest  2641 non-null   float64
      8   newbalanceDest  2641 non-null   float64
      9   isFraud         2641 non-null   int64
      10  isFlaggedFraud  2641 non-null   int64
     dtypes: float64(5), int64(3), object(3)
     memory usage: 227.1+ KB
```

The code `df['isFraud'].value_counts()` counts the occurrences of unique values in the 'isFraud' column of the DataFrame `df`. It is useful for understanding the distribution of fraud and non-fraud instances in the dataset.

```
[ ]  df['isFraud'].value_counts()

     0    1499
     1    1142
     Name: isFraud, dtype: int64
```

The code `df.describe()` generates descriptive statistics of the numerical columns in the DataFrame `df`. It provides key statistical measures, including count, mean, standard deviation, minimum, and maximum values, offering a quick insight into the central tendency and spread of the data.

```
[ ]  df.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	2641.000000	2.641000e+03	2.641000e+03	2.641000e+03	2.641000e+03	2.641000e+03	2641.000000	2641.0
mean	21.441121	5.761542e+05	1.095526e+06	5.978885e+05	6.903365e+05	9.909810e+05	0.432412	0.0
std	29.337193	1.451956e+06	2.127746e+06	1.679177e+06	2.151393e+06	2.556171e+06	0.495505	0.0
min	1.000000	6.420000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
25%	1.000000	7.269740e+03	1.370400e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
50%	1.000000	7.938685e+04	1.157116e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
75%	41.000000	3.633788e+05	1.026209e+06	4.869947e+04	2.801579e+05	5.958335e+05	1.000000	0.0
max	95.000000	1.000000e+07	1.990000e+07	1.020000e+07	3.300000e+07	3.460000e+07	1.000000	0.0

The code `df = df.dropna()` removes rows with missing values from the DataFrame `df`. This operation is useful for ensuring data integrity and preparing the dataset for analysis by eliminating instances with incomplete information.

```
df = df.dropna()

Checking for null values
```

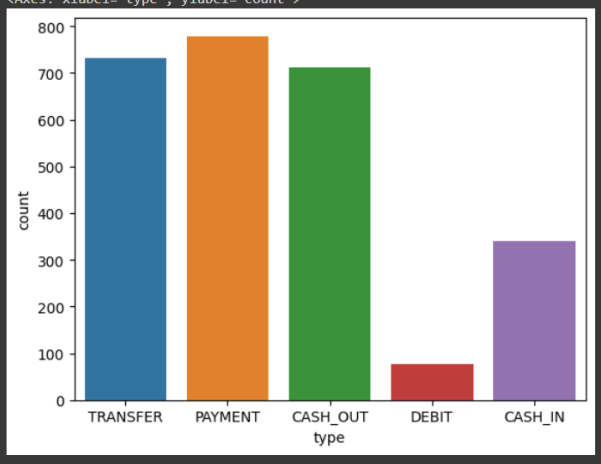
The code `df.isnull().sum()` calculates the count of missing values for each column in the DataFrame `df`. It provides a summary of the data's completeness, indicating how many null values exist in each column.

```
df.isnull().sum()

step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

Data Visualization:

The code `sns.countplot(x='type', data=df)` uses the Seaborn library to create a count plot based on the 'type' column in the DataFrame `df`. This visualization depicts the distribution of categories in the 'type' column, offering insights into the frequency of each category.

```
[ ] sns.countplot(x='type', data=df)

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



Retain 6 features and target variable:

The code `df = df[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFraud']]` selects specific columns ('amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', and 'isFraud') from the DataFrame `df`. This operation creates a new DataFrame with only the chosen columns, focusing on relevant features for further analysis or modeling.

```
[ ]  df = df[['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest',"isFraud"]]
```

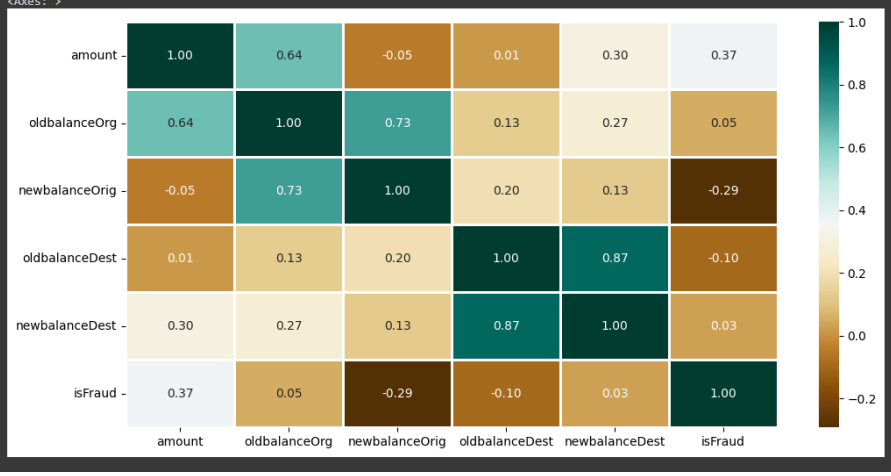
The provided code categorizes the columns in the DataFrame `df` based on their data types. It identifies and lists categorical variables, integer variables, and float variables separately. The output indicates the count of each variable type, providing a quick summary of the data's composition.

```
[ ]  obj = (df.dtypes == 'object')
     object_cols = list(obj[obj].index)
     print("Categorical variables:", len(object_cols))

     int_ = (df.dtypes == 'int')
     num_cols = list(int_[int_].index)
     print("Integer variables:", len(num_cols))

     fl = (df.dtypes == 'float')
     fl_cols = list(fl[fl].index)
     print("Float variables:", len(fl_cols))

     Categorical variables: 0
     Integer variables: 1
     Float variables: 5
```

The code uses Matplotlib and Seaborn libraries to create a heatmap of the correlation matrix for the DataFrame `df`. The figure size is set to 12 by 6 inches. The heatmap visualizes the pairwise correlations between numerical columns, with values annotated on the plot. It helps identify relationships between variables, with color intensity indicating the strength and direction of correlation. The 'BrBG' colormap is applied, and the format is set to display correlation values with two decimal places.

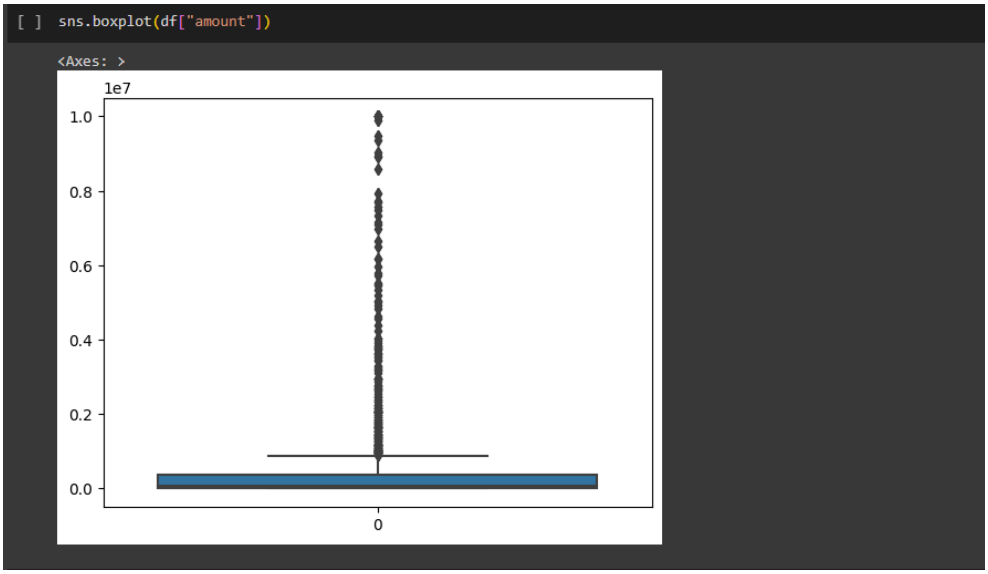
```
plt.figure(figsize=(12, 6))
sns.heatmap(df.corr(),
            cmap='BrBG',
            fmt='.2f',
            linewidths=2,
            annot=True)
```

```
<Axes: >
```

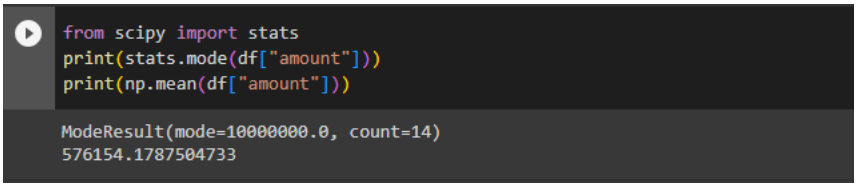


Handling outliers

The code `sns.boxplot(df["amount"])` utilizes Seaborn to create a boxplot for the 'amount' column in the DataFrame `df`. This visualization offers insights into the distribution of the 'amount' variable, displaying key statistical measures such as the median, quartiles, and potential outliers.



The code utilizes the SciPy library to calculate the mode of the 'amount' column in the DataFrame `df` using `stats.mode()`. Additionally, it computes the mean of the 'amount' column using NumPy's `np.mean()`. These statistical measures provide insights into the central tendency of the 'amount' variable, with the mode representing the most frequently occurring value and the mean indicating the average value.



The code calculates the interquartile range (IQR) for the 'amount' column in the DataFrame `df` and defines upper and lower bounds to identify potential outliers. It prints the values of the first quartile (q1), third quartile (q3), IQR, upper bound, and lower bound. The code then counts the number of data points beyond the upper and lower bounds, indicating potential skewed data points. This analysis helps identify and understand the distribution of the 'amount' variable in terms of outliers.

```
q1 = np.quantile(df['amount'],0.25)

q3 = np.quantile(df['amount'],0.75)

IQR = q3-q1

upper_bound = q3+(1.5*IQR)

lower_bound = q1-(1.5*IQR)

print('q1:',q1)

print('q3:',q3)

print('IQR:',IQR)

print('Upper Bound :', upper_bound)

print('Lower Bound :',lower_bound)

print('Skewed data :',len(df[df['amount']>upper_bound]))

print('Skewed data :',len(df[df['amount']<lower_bound]))
```

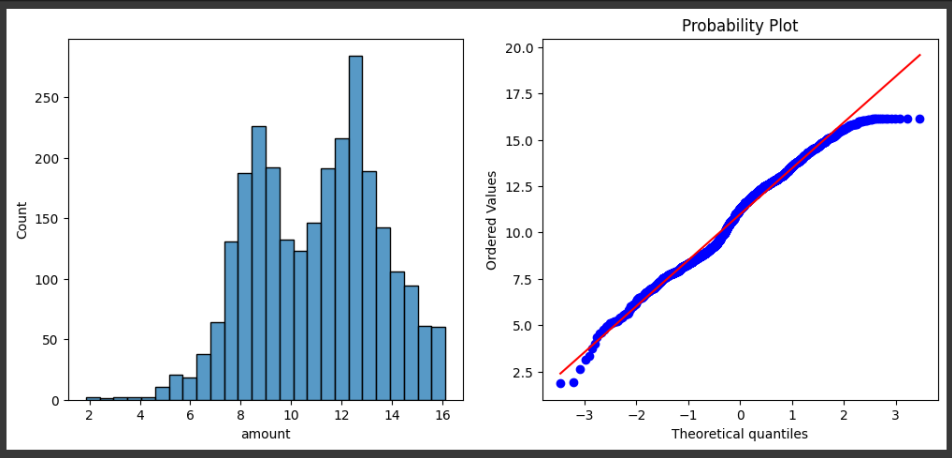
```
q1: 7269.74
q3: 363378.75
IQR: 356109.01
Upper Bound : 897542.265
Lower Bound : -526893.775
Skewed data : 383
Skewed data : 0
```

The code defines a function, `transformationPlot(feature)`, which generates a side-by-side plot. The left subplot is a histogram (using Seaborn) representing the distribution of the input feature, while the right subplot is a probability plot (using SciPy's `probplot` from the `stats` module). This function is designed to provide visual insights into the distribution and potential transformation of a given feature.

```
def transformationPlot(feature):

  plt.figure(figsize=(12,5))

  plt.subplot(1,2,1)

  sns.histplot(feature)

  plt.subplot(1,2,2)

  stats.probplot(feature, plot=plt)
```

The code applies the `transformationPlot` function to the logarithmically transformed 'amount' column (`np.log(df['amount'])`). This facilitates visualizing the impact of the logarithmic transformation on the distribution of the 'amount' variable, helping assess the effectiveness of this transformation for normalization or addressing skewness.

```
[ ] transformationPlot(np.log(df['amount']))
```



X and Y split:

The code creates a new DataFrame `X` by dropping the 'isFraud' column from the original DataFrame `df`. This operation is typically performed to isolate the independent variables, preparing the data for model training where 'isFraud' is the dependent variable.

```
[ ] X = df.drop('isFraud',axis=1)
```

The code creates a Series `y` by extracting the 'isFraud' column from the DataFrame `df`. This is typically done to define the dependent variable for a machine learning model, separating it from the independent variables stored in the DataFrame `X`.

```
[ ] y = df['isFraud']
```

Split the data into training and testing:

The code utilizes scikit-learn's `train_test_split` function to split the dataset into training and testing sets. It assigns 70% of the data to the training set (`X_train` and `y_train`) and 30% to the testing set (`X_test` and `y_test`). The parameter `random_state=42` ensures reproducibility of the split. This separation is essential for training and evaluating machine learning models.

Split the data into training and testing

```
[ ] from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state=42)
```

Model Building and Evaluation

Training the Naive Bayes model:

The code uses scikit-learn's `GaussianNB` to instantiate a Gaussian Naive Bayes classifier (`Cnb`). The classifier is then trained on the training sets (`X_train` and `y_train`) using the `fit` method. This establishes the Naive Bayes model for subsequent predictions based on the training data.

Training the Naive Bayes model

```
[ ] from sklearn.naive_bayes import GaussianNB
    Cnb = GaussianNB()
    Cnb.fit(X_train, y_train)
```

```
GaussianNB
GaussianNB()
```

The code utilizes scikit-learn's metrics module to evaluate the performance of a Gaussian Naive Bayes classifier (`Cnb`). It predicts the target variable (`y_pred`) on the test set (`X_test`) using the trained Naive Bayes model. The confusion matrix (`cm`) is calculated by comparing the predicted values to the actual values (`y_test`). Additionally, the accuracy score of the model is computed using the `accuracy_score` function, offering an overall measure of model performance.

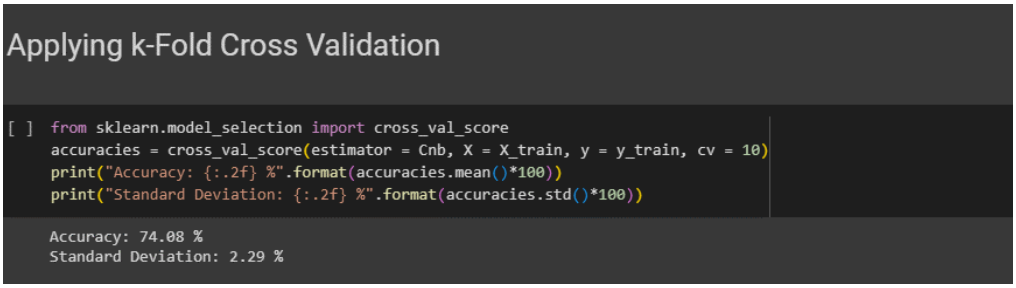
```
[ ] from sklearn.metrics import confusion_matrix, accuracy_score
    y_pred = Cnb.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    accuracy_score(y_test, y_pred)
```

```
[[445   7]
 [196 145]]
0.7440100882723834
```

Applying k-Fold Cross Validation:

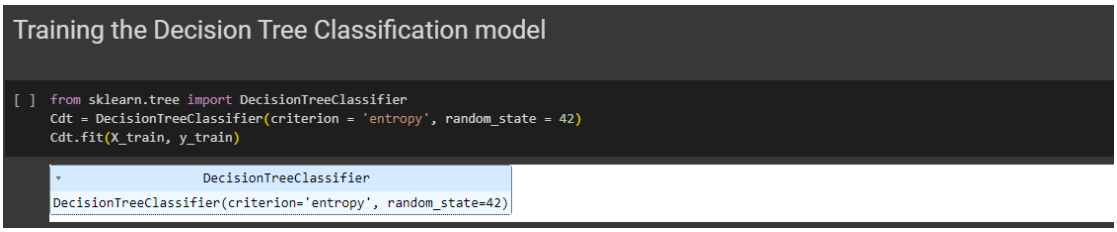
The code utilizes scikit-learn's `cross_val_score` to perform 10-fold cross-validation on a Gaussian Naive Bayes classifier (`Cnb`). It calculates accuracy scores for each fold, and the mean accuracy along with the

standard deviation of the accuracy scores is printed. This provides insights into the Naive Bayes model's generalization performance across different subsets of the training data.

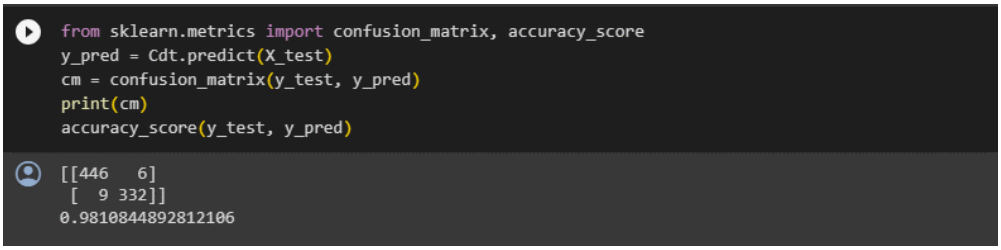


Training the Decision Tree Classification model:

The code uses scikit-learn's `DecisionTreeClassifier` to instantiate a Decision Tree classifier (`Cdt`). The classifier is configured with the entropy criterion and a specified random state for reproducibility. It is then trained on the training sets (`X_train` and `y_train`) using the `fit` method, establishing the Decision Tree model for subsequent predictions based on the training data.

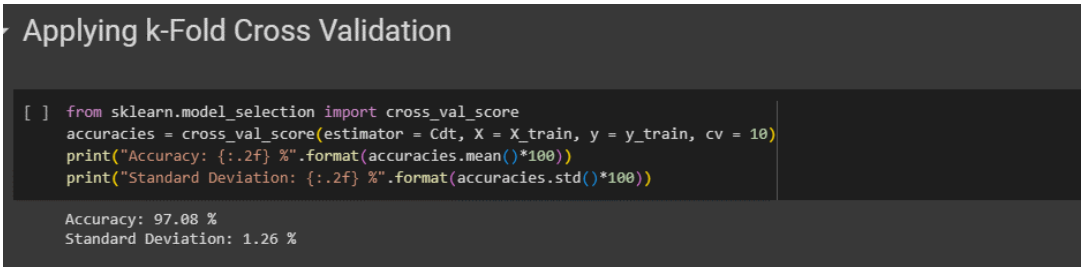


The code utilizes scikit-learn's metrics module to evaluate the performance of a Decision Tree classifier (`Cdt`). It predicts the target variable (`y_pred`) on the test set (`X_test`) using the trained Decision Tree model. The confusion matrix (`cm`) is calculated by comparing the predicted values to the actual values (`y_test`). Additionally, the accuracy score of the model is computed using the `accuracy_score` function, offering an overall measure of model performance.



Applying k-Fold Cross Validation:

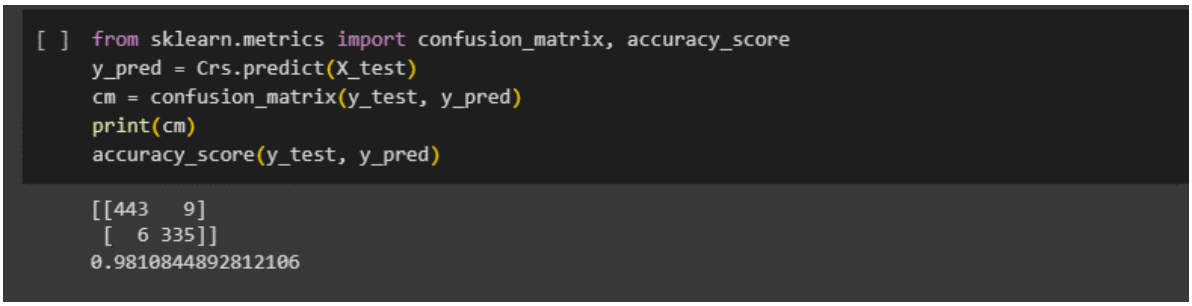
The code uses scikit-learn's `cross_val_score` to perform 10-fold cross-validation on a Decision Tree classifier (`Cdt`). It calculates accuracy scores for each fold, and the mean accuracy along with the standard deviation of the accuracy scores is printed. This provides insights into the Decision Tree model's generalization performance across different subsets of the training data.



Training the Random Forest Classification model:

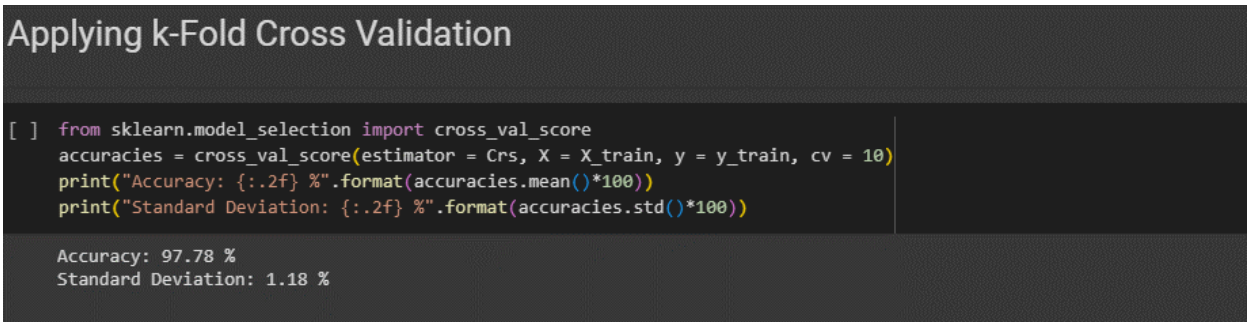
The code uses scikit-learn's `RandomForestClassifier` to instantiate a Random Forest classifier (`Crs`). The classifier is configured with 100 estimators, the entropy criterion, and a specified random state for reproducibility. It is then trained on the training sets (`X_train` and `y_train`) using the `fit` method, establishing the Random Forest model for subsequent predictions based on the training data.

The code uses scikit-learn's metrics module to evaluate the performance of a Random Forest classifier (`Crs`). It predicts the target variable (`y_pred`) on the test set (`X_test`) using the trained Random Forest model. The confusion matrix (`cm`) is calculated by comparing the predicted values to the actual values (`y_test`). Additionally, the accuracy score of the model is computed using the `accuracy_score` function, providing an overall measure of model performance.



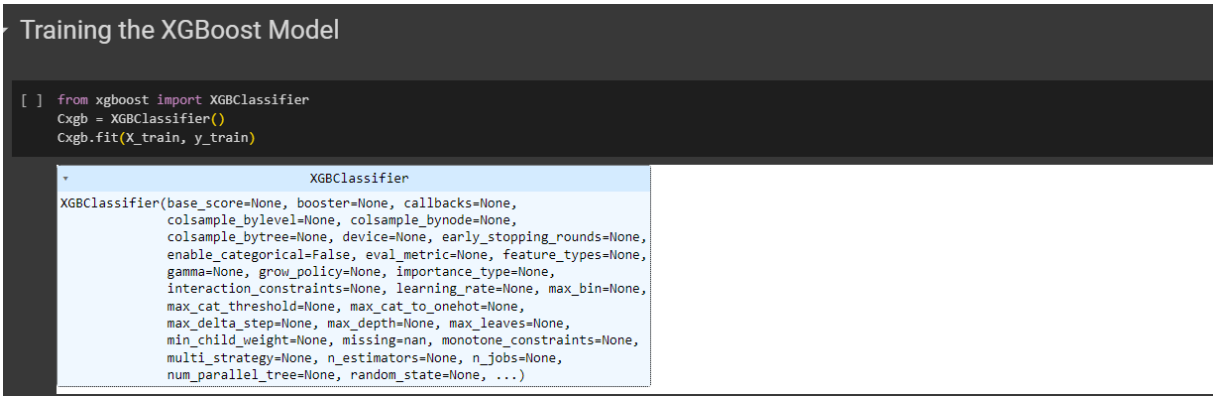
Applying k-Fold Cross Validation:

The code utilizes scikit-learn's `cross_val_score` to perform 10-fold cross-validation on a Random Forest classifier (`Crs`). It calculates accuracy scores for each fold, and the mean accuracy along with the standard deviation of the accuracy scores is printed. This provides insights into the Random Forest model's generalization performance across different subsets of the training data.



Training the XGBoost Model:

The code uses the XGBoost library to instantiate an XGBoost classifier (`Cxgb`). The classifier is then trained on the training sets (`X_train` and `y_train`) using the `fit` method. This establishes the XGBoost model for subsequent predictions based on the training data. XGBoost is a powerful gradient boosting algorithm known for its efficiency and performance.



The code utilizes scikit-learn's metrics module to evaluate the performance of an XGBoost classifier (`Cxgb`). It predicts the target variable (`y_pred`) on the test set (`X_test`) using the trained XGBoost model. The confusion matrix (`cm`) is calculated by comparing the predicted values to the actual values (`y_test`). Additionally, the accuracy score of the model is computed using the `accuracy_score` function, providing an overall measure of model performance. XGBoost is recognized for its effectiveness in various machine learning tasks.

```
[ ] from sklearn.metrics import confusion_matrix, accuracy_score
    y_pred = Cxgb.predict(X_test)
    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    accuracy_score(y_test, y_pred)

    [[443   9]
     [  9 332]]
    0.9773013871374527
```

Applying k-Fold Cross Validation:

The code uses scikit-learn's `cross_val_score` to perform 10-fold cross-validation on an XGBoost classifier (`Cxgb`). It calculates accuracy scores for each fold, and the mean accuracy along with the standard deviation of the accuracy scores is printed. This provides insights into the XGBoost model's generalization performance across different subsets of the training data.

Applying k-Fold Cross Validation

```
[ ] from sklearn.model_selection import cross_val_score
    accuracies = cross_val_score(estimator = Cxgb, X = X_train, y = y_train, cv = 10)
    print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
    print("Standard Deviation: {:.2f} %".format(accuracies.std()*100))

    Accuracy: 97.56 %
    Standard Deviation: 1.36 %
```

The code uses the `pickle` library to save the trained XGBoost classifier (`Cxgb`) as a binary file named 'fraud.pkl'. This allows for later retrieval and reuse of the model without needing to retrain it. The 'wb' mode indicates writing in binary mode.

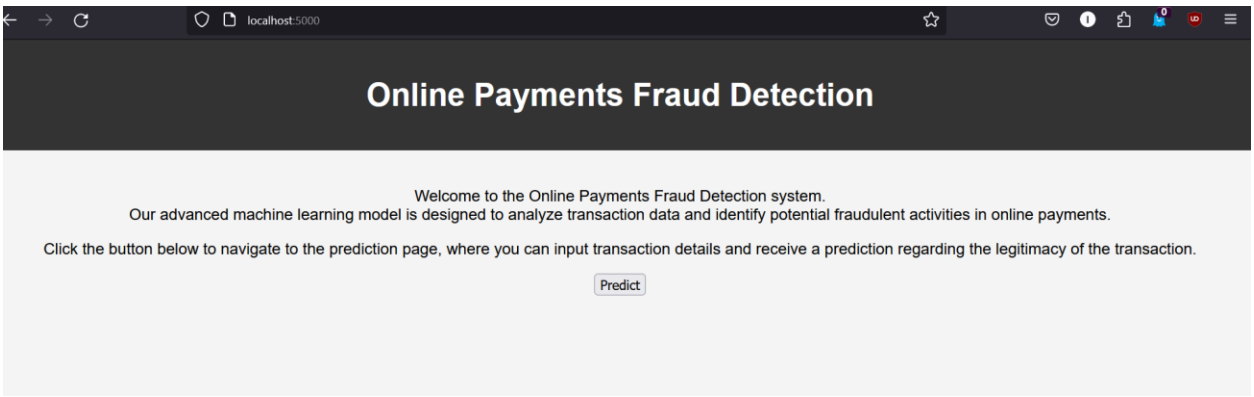
```
[ ] import pickle
    pickle.dump(Cxgb,open('fraud.pkl','wb'))
```

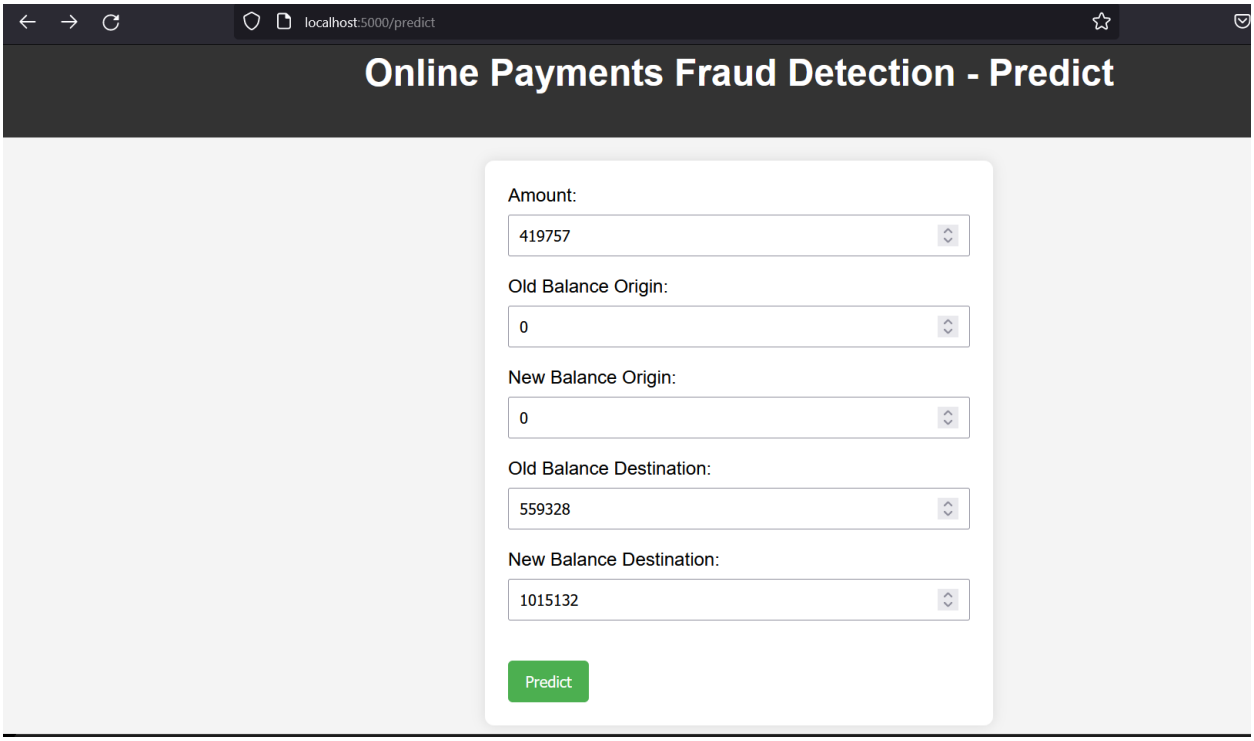
Flask integration:

```
import numpy as np

from flask import Flask, request, render_template
import pickle
```

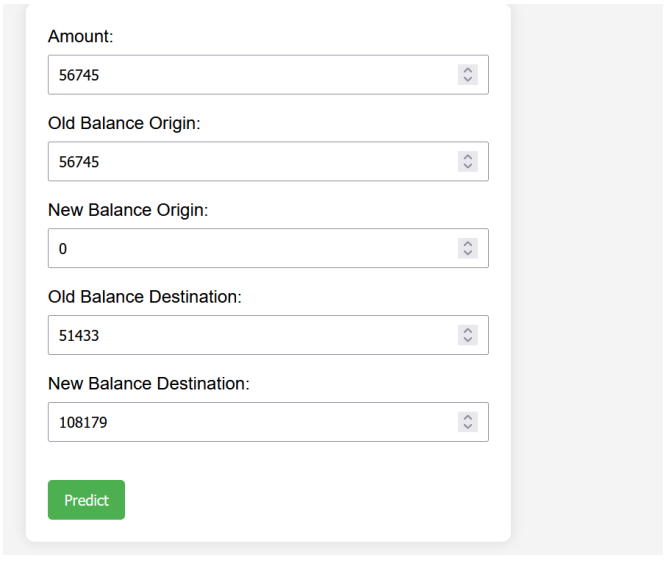
```
app = Flask(__name__)

model = pickle.load(open('fraud.pkl', 'rb'))
@app.route('/')
def home():
    return render_template('home.html')
@app.route('/predict')
def home1():
    return render_template('predict.html')
@app.route('/pred', methods=['POST', 'GET'])
def predict():


    int_features = [float(x) for x in request.form.values()]
    final_features = [np.array(int_features)]
    prediction = model.predict(final_features)

    if prediction == 0:
        return render_template('predict.html',
                               prediction_text='Low chances of transaction being
fraud'.format(
                                   prediction),
                                   )
    else:
        return render_template('predict.html',
                               prediction_text='High chances of transaction being
fraud'.format(
                                   prediction),
                                   )
if __name__ == "__main__":
    app.run(debug=False, host='0.0.0.0')
```

ScreenShots:





Low chances of transaction being fraud



High chances of transaction being fraud