Payment Fraud Detection using ML

Team:

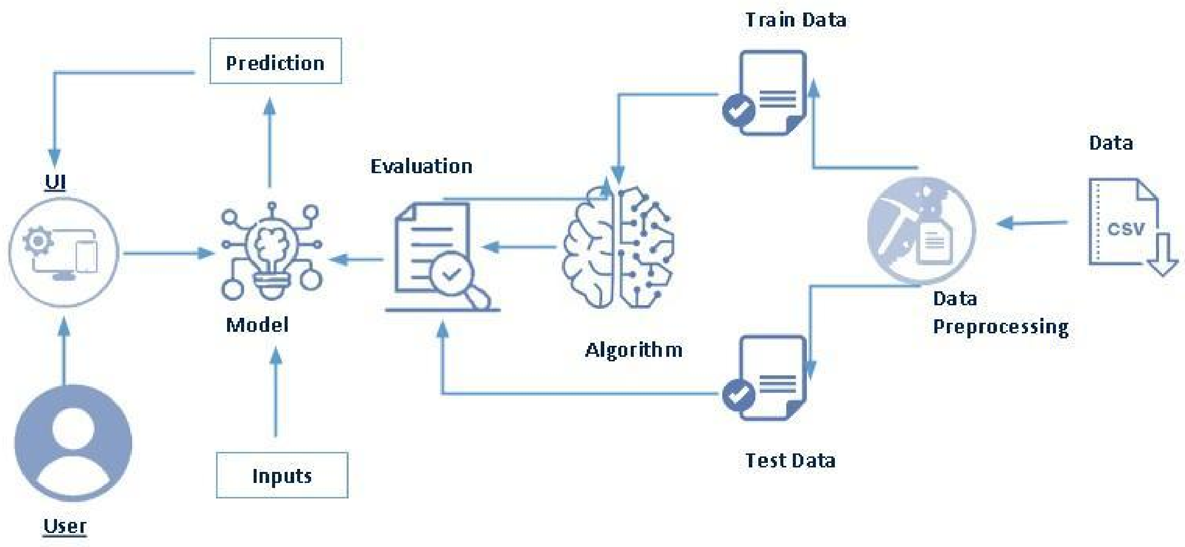
1. Yash Tyagi
2. Nethish Kumar C D
3. Yenesh Saxena
4. Raghu Venkata Pradh

Project Hand-out, Faculty Development Program – Shanmukhi

Online Payments Fraud Detection using Machine Learning

Online Payments Fraud Detection using Machine Learning is a proactive approach to identify and prevent fraudulent activities during online transactions. By leveraging historical transaction data, customer behavior patterns, and machine learning algorithms, this project aims to detect potential fraud in real time, ensuring secure and trustworthy online payment experiences for users and businesses alike.

Technical Architecture :



Project Flow

- The user interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- The predictions made by the model are 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
- Data pre-processing
    - Removing unnecessary columns
    - Checking for null values
- Visualizing and analyzing data
- Univariate analysis
- Bivariate analysis
- Descriptive analysis
- Model building
    - Handling categorical values
    - Dividing data into train and test sets
    - Import the model building libraries
    - Comparing the accuracy of various models
    - Hyperparameter tuning of the selected model
    - Evaluating the performance of models
    - Save the model
- Application Building
    - Create an HTML file
    - Build python code

Prior Knowledge

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

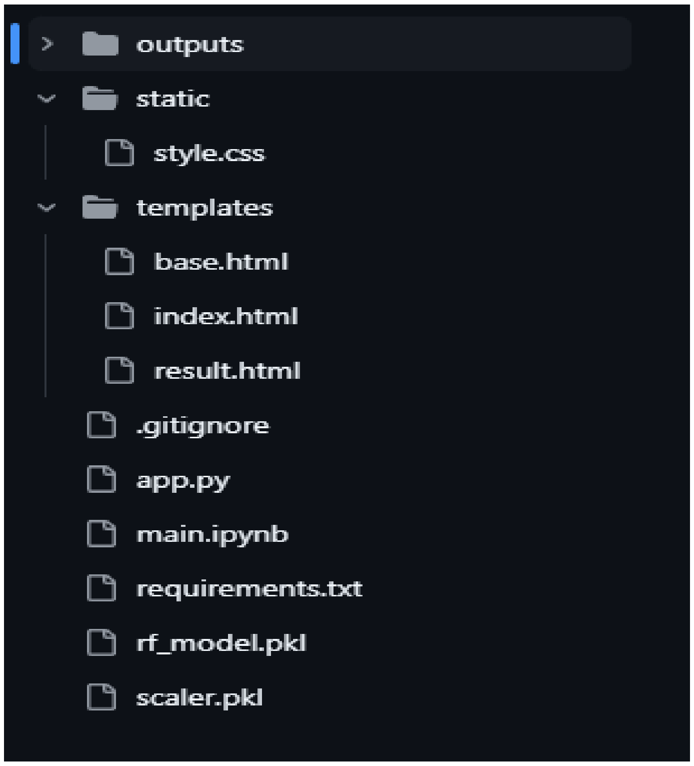
ML Concepts

Supervised learning:https://www.youtube.com/embed/QeKshry8pWQ
Unsupervised learning:https://www.youtube.com/embed/D6gtZrsYi6c
Metrics :https://www.youtube.com/embed/aWAnNHXIKww

- Flask :https://www.youtube.com/embed/lj4I_CvBnt0

Project Structure :

Create the Project folder which contains files as shown below



```
> outputs
v static
    style.css
v templates
    base.html
    index.html
    result.html
  .gitignore
  app.py
  main.ipynb
  requirements.txt
  rf_model.pkl
  scaler.pkl
```

- 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 : Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Real-time Fraud Monitoring

The system continuously monitors online payment transactions in real time. By analyzing transaction features such as transaction amount, location, device information, and user behavior, it can flag suspicious transactions for further investigation, preventing fraudulent activities before they occur.

Activity 3: Fraudulent Account Detection

Machine learning models can detect patterns indicative of fraudulent accounts or activities. By analyzing user behavior over time, such as unusual login times, multiple failed login attempts, or sudden changes in spending patterns, the system can identify and block potentially fraudulent accounts, protecting legitimate users and businesses.

Activity 4: Adamptive Fraud Prevention

The system adapts and improves its fraud detection capabilities over time. By continuously learning from new data and adjusting its algorithms, it can stay ahead of evolving fraud techniques and trends, providing ongoing protection against online payment fraud for businesses and their customers.

Milestone 2 : Data Collection/Preprocessing

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

Activity 1: Collect the dataset

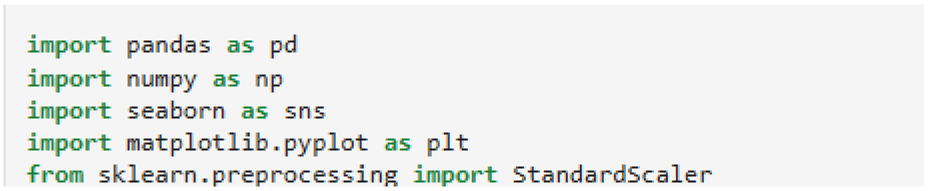
There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.
In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

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

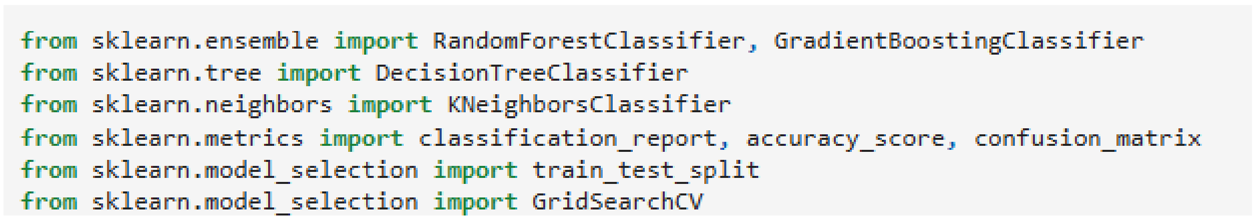
As the dataset is downloaded. Let us read and understand the data properly with the help of somevisualisation techniques and some analysing techniques.

Activity 1.1: Importing the Libraries

Import the necessary libraries as shown in the image.



```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
```

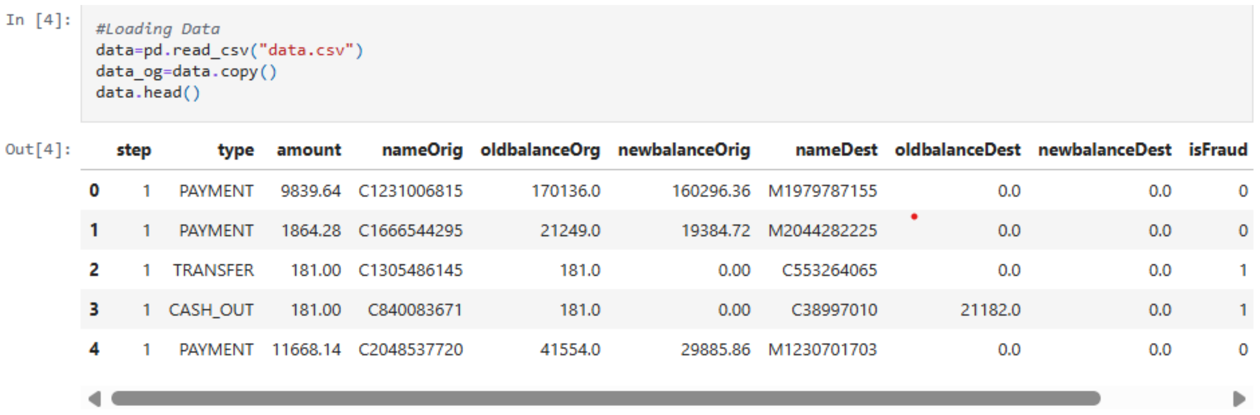


```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset withthe help of pandas.In pandas we have a function called read_csv() to read the dataset. Asa parameter we haveto give the directory of the csv file.

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



```
In [4]: #Loading Data
        data=pd.read_csv("data.csv")
        data_og=data.copy()
        data.head()
```

```
Out[4]:
   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36  M1979787155             0.0             0.0        0
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72  M2044282225             0.0             0.0        0
2     1  TRANSFER    181.00  C1305486145          181.0            0.00   C553264065             0.0             0.0        1
3     1  CASH_OUT    181.00   C840083671          181.0            0.00    C38997010         21182.0             0.0        1
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86  M1230701703             0.0             0.0        0
```

Activity 2: Data Preperation

As we have understood how the data is, let's pre-process the collected data.The download data set is not suitable for training the machine learning model as it might have so much 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 Outliers

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 2.1: Finding Null Values

For checking the null values, df.isna().any( ) function is used. To sum those null value we use .sum() function. From the below image we found that there are no null values present in our dataset.

```
In [9]:  #Handling Missing Data (There isnt any missing data
         data.isnull().sum()
```

```
Out[9]:  step              0
         type              0
         amount            0
         nameOrig          0
         oldbalanceOrg     0
         newbalanceOrig    0
         nameDest          0
         oldbalanceDest    0
         newbalanceDest    0
         isFraud           0
         isFlaggedFraud    0
         dtype: int64
```

Activity 2.2: Handling Outliers

```
In [8]:  #Anomalies/Outliers
         "Due to the extremely wide distribution and heavy-tailed nature of financial transaction data in this dataset, traditiona
```

```
Out[8]:  'Due to the extremely wide distribution and heavy-tailed nature of financial transaction data in this dataset, traditiona
         l outlier detection methods are not effective. The data spans several orders of magnitude (e.g., transaction amounts rang
         e from $0 to $92+ million), making it difficult to distinguish between legitimate large transactions and true anomalies u
         sing standard statistical methods.'
```

Activity 2.3: Object Data LabelEncoding

```
In [6]:  df_encoded = pd.get_dummies(df_clean, columns=['type'], prefix='type')
         df_encoded.head()
```

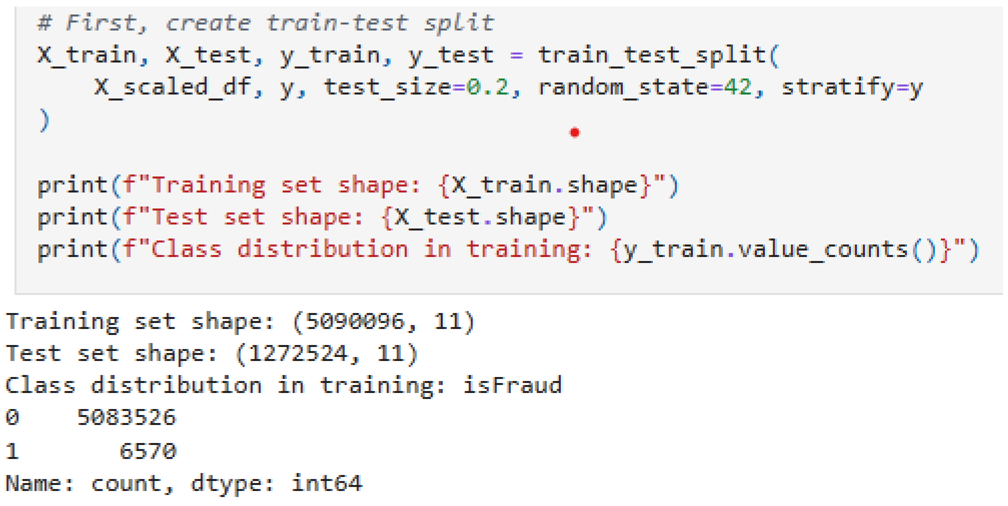
Out[6]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	type_CASH_IN	type_CASH_OUT	type_D
0	1	9839.64	170136.0	160296.36	0.0	0.0	0	False	False	
1	1	1864.28	21249.0	19384.72	0.0	0.0	0	False	False	
2	1	181.00	181.0	0.00	0.0	0.0	1	False	False	
3	1	181.00	181.0	0.00	21182.0	0.0	1	False	True	
4	1	11668.14	41554.0	29885.86	0.0	0.0	0	False	False	

```
In [7]:  X = df_encoded.drop('isFraud', axis=1)
         y = df_encoded['isFraud']
         scaler = StandardScaler()
         X_scaled = scaler.fit_transform(X)
         X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
         X_scaled_df.describe()
```

Activity 2.4: Train _Test split

Now let’s split the Dataset into train and test setsChanges: 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 my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.



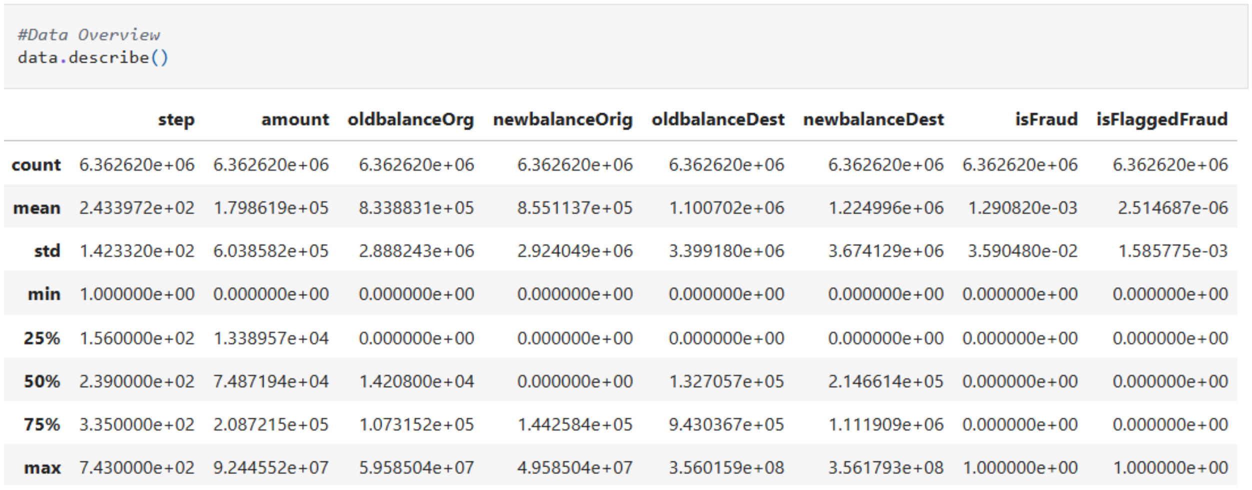
```
# First, create train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled_df, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")
print(f"Class distribution in training: {y_train.value_counts()}")
```

```
Training set shape: (5090096, 11)
Test set shape: (1272524, 11)
Class distribution in training: isFraud
0    5083526
1       6570
Name: count, dtype: int64
```

Milestone 3 : Exploratory Data Analysis

Activity 1: Descriptive statistical

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



```
#Data Overview
data.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03	2.514687e-06
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	1.585775e-03
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00	0.000000e+00
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00	0.000000e+00
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00	1.000000e+00

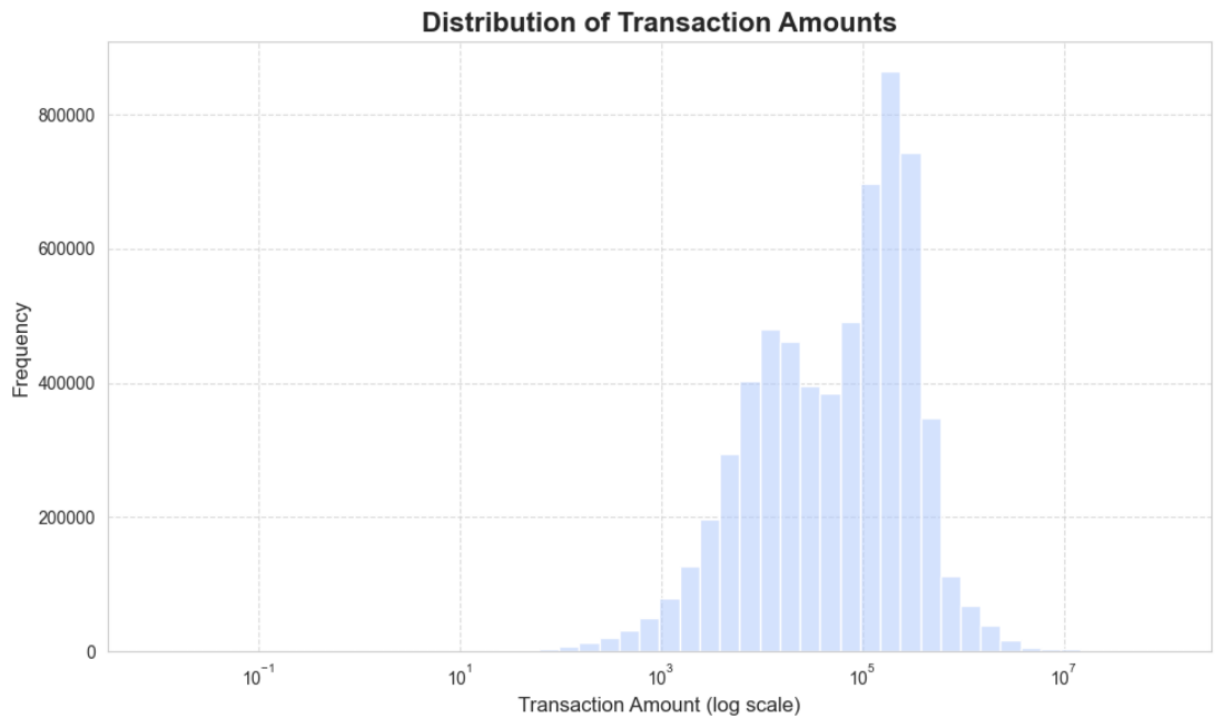
Activity 2: Visual analysis

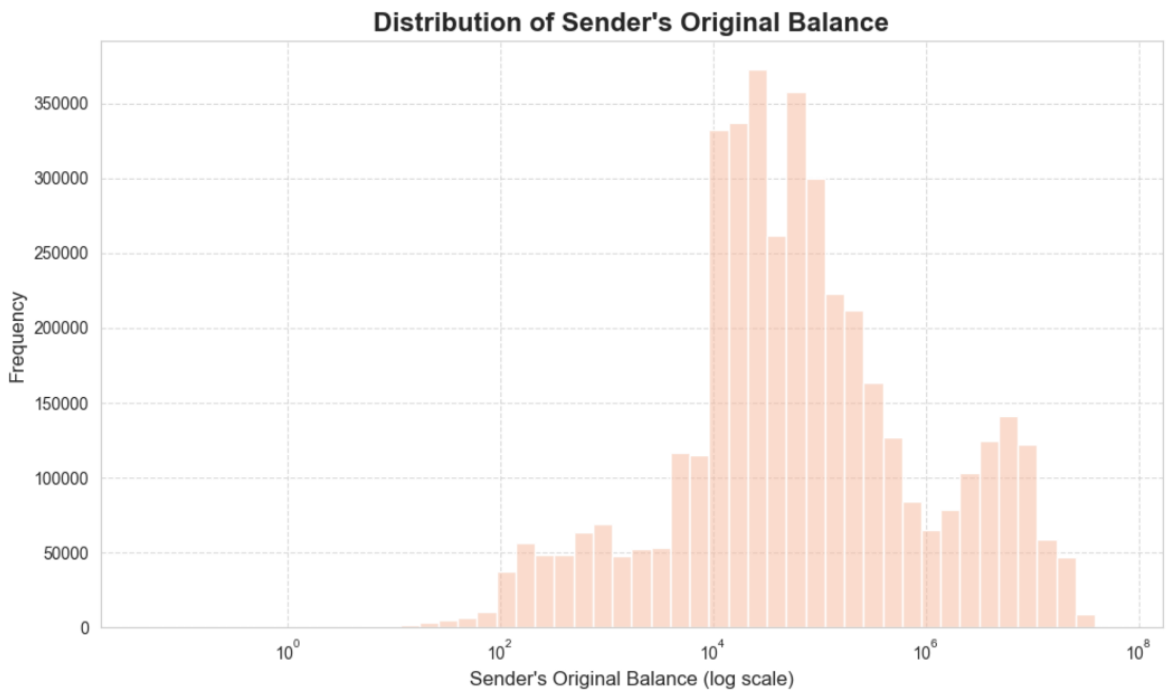
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

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

```python
#Univariate Analysis
sns.set_style('whitegrid')
palette = sns.color_palette('coolwarm', 2)

plt.figure(figsize=(10, 6))
sns.histplot(data['amount'], bins=50, log_scale=True, color=palette[0], kde=True)
plt.title('Distribution of Transaction Amounts', fontsize=16, fontweight='bold')
plt.xlabel('Transaction Amount (log scale)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 6))
sns.histplot(data['oldbalanceOrg'], bins=50, log_scale=True, color=palette[1], kde=True)
plt.title("Distribution of Sender's Original Balance", fontsize=16, fontweight='bold')
plt.xlabel("Sender's Original Balance (log scale)", fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

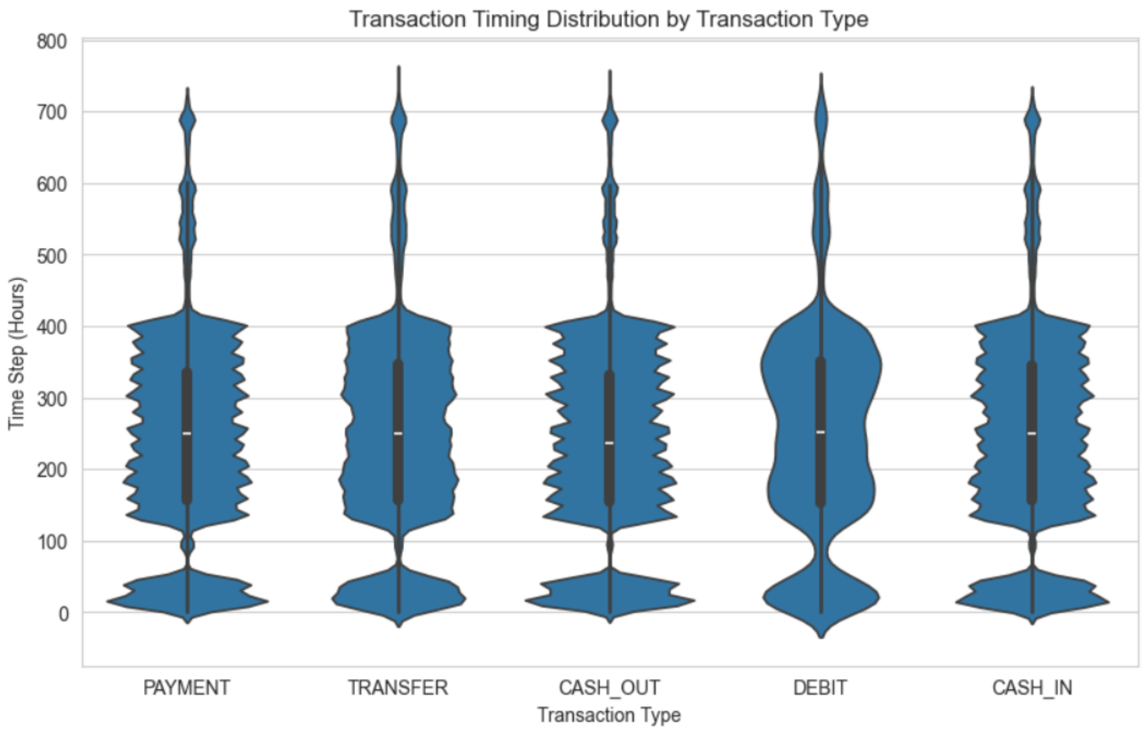


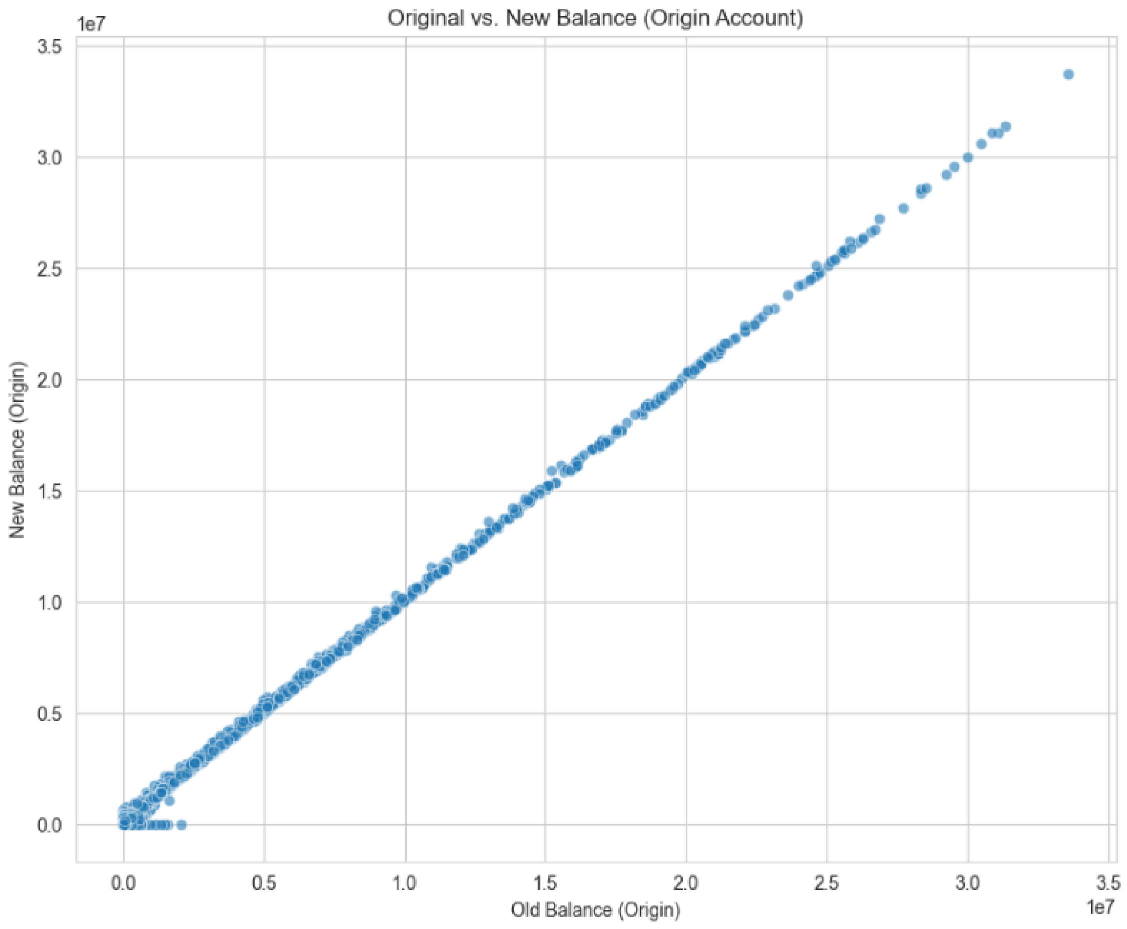


Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we can used barplot.

```python
#Bivariate Analysis
plt.figure(figsize=(10, 6))
sns.violinplot(x="type", y="step", data=data)
plt.title("Transaction Timing Distribution by Transaction Type")
plt.xlabel("Transaction Type")
plt.ylabel("Time Step (Hours)")
plt.show()


plt.figure(figsize=(10, 8))
sns.scatterplot(
    x="oldbalanceOrg",
    y="newbalanceOrig",
    data=data.sample(10000, random_state=42),
    alpha=0.6
)
plt.title("Original vs. New Balance (Origin Account)")
plt.xlabel("Old Balance (Origin)")
plt.ylabel("New Balance (Origin)")
plt.show()
```


Transaction Timing Distribution by Transaction Type
Time Step (Hours)
800
700
600
500
400
300
200
100
0
PAYMENT
TRANSFER
CASH_OUT
DEBIT
CASH_IN
Transaction Type

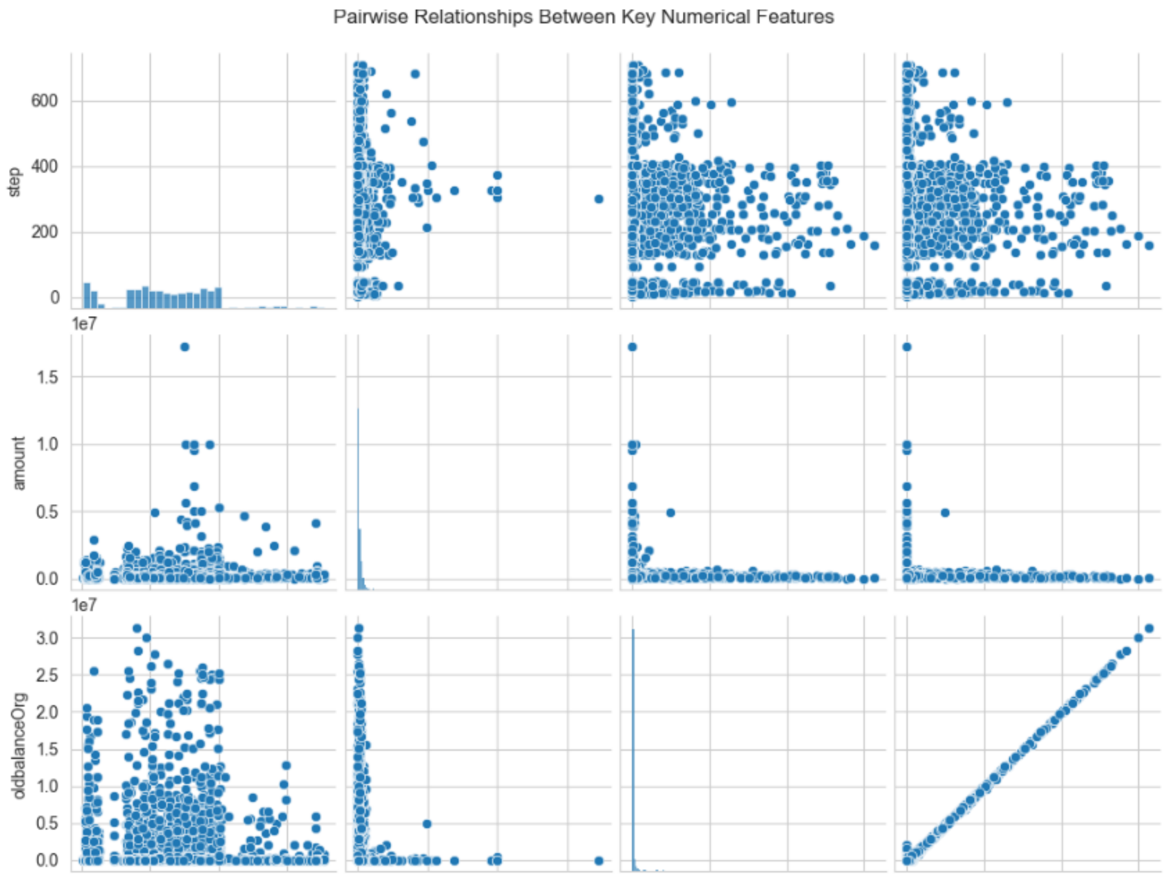

Original vs. New Balance (Origin Account)
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
New Balance (Origin)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e7
Old Balance (Origin)

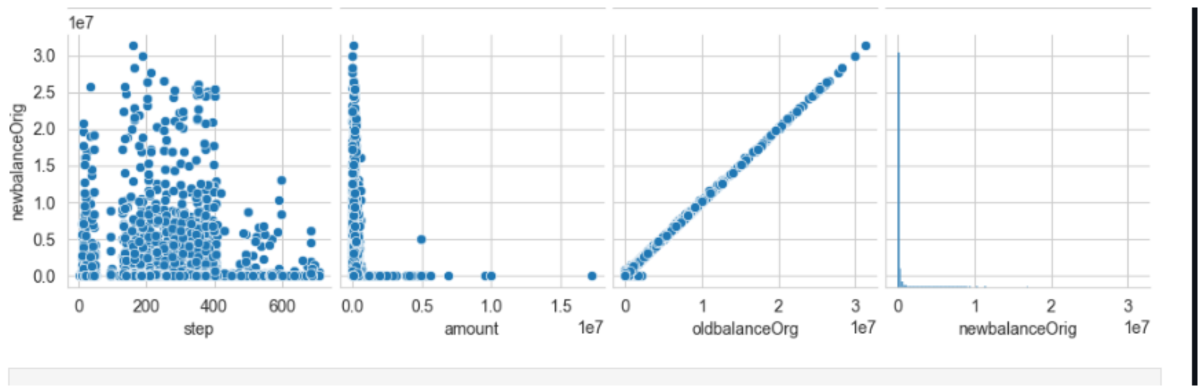
Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we check he pairise relationships between key numerical features.

```
#Multivariate Analysis
plt.figure(figsize=(12, 10))
sns.pairplot(data[['step', 'amount', 'oldbalanceOrg', 'newbalanceOrig']].sample(5000,random_state=42))
plt.suptitle('Pairwise Relationships Between Key Numerical Features', y=1.02)
plt.show()
```

```
Figure size 1200x1000 with 0 Axes>
```



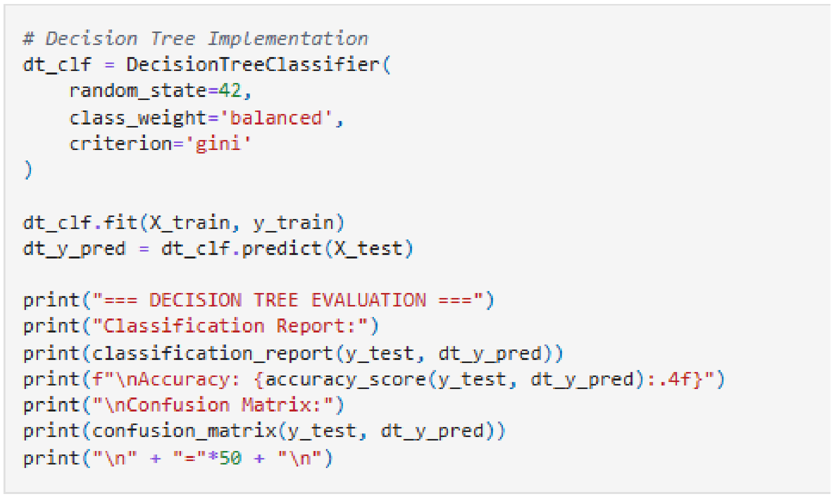


Milestone 4 : Model Building

Activity 1: Training the model in multiple algorithms Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance

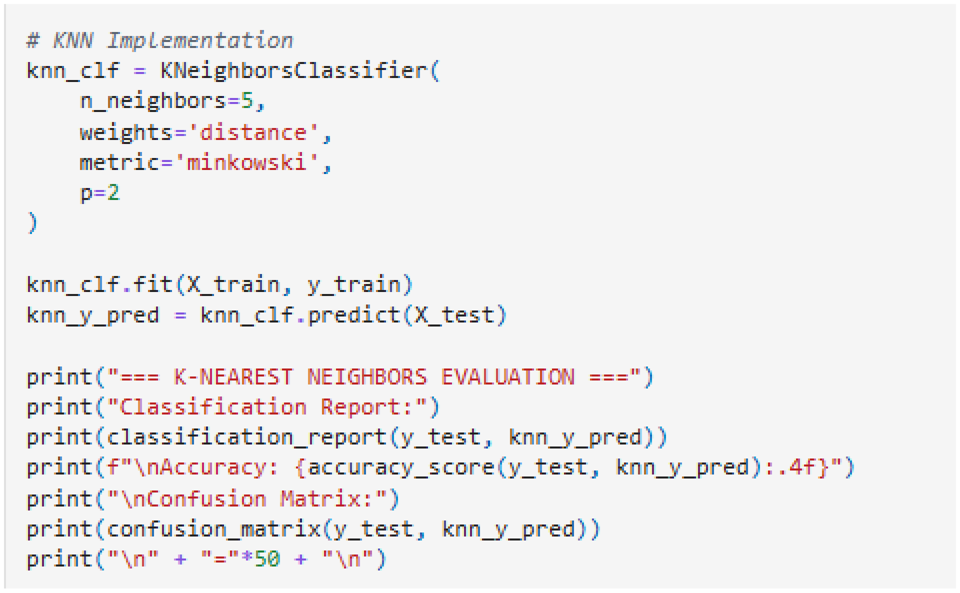
Activity 1.1: Decision tree model

First Decision Tree is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.


```
# Decision Tree Implementation
dt_clf = DecisionTreeClassifier(
    random_state=42,
    class_weight='balanced',
    criterion='gini'
)

dt_clf.fit(X_train, y_train)
dt_y_pred = dt_clf.predict(X_test)

print("=== DECISION TREE EVALUATION ===")
print("Classification Report:")
print(classification_report(y_test, dt_y_pred))
print(f"\nAccuracy: {accuracy_score(y_test, dt_y_pred):.4f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, dt_y_pred))
print("\n" + "="*50 + "\n")
```

Activity 1.2: KNN model

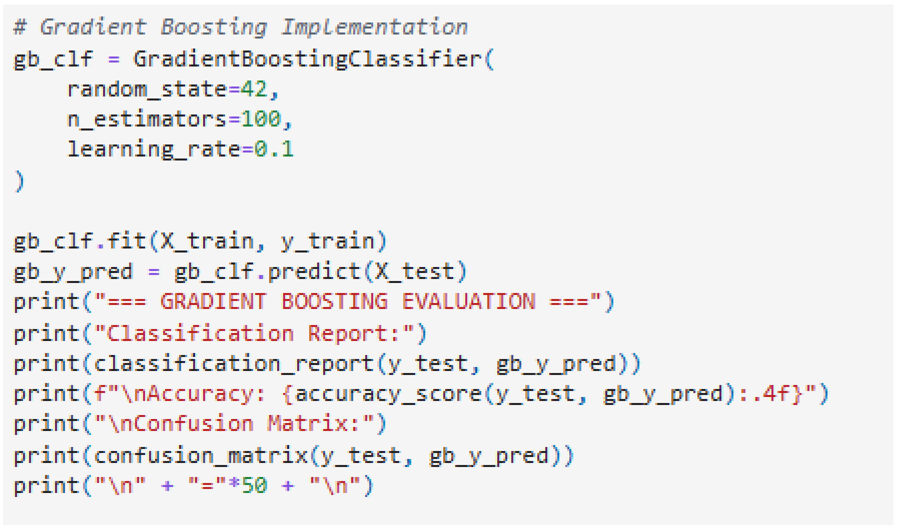
KNN Model is imported from sklearn Library then KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.



```
# KNN Implementation
knn_clf = KNeighborsClassifier(
    n_neighbors=5,
    weights='distance',
    metric='minkowski',
    p=2
)

knn_clf.fit(X_train, y_train)
knn_y_pred = knn_clf.predict(X_test)

print("=== K-NEAREST NEIGHBORS EVALUATION ===")
print("Classification Report:")
print(classification_report(y_test, knn_y_pred))
print(f"\nAccuracy: {accuracy_score(y_test, knn_y_pred):.4f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, knn_y_pred))
print("\n" + "="*50 + "\n")
```

Activity 1.3: Gradient boosting

Gradient boosting is an ensemble learning technique that combines multiple "weak" learners, typically decision trees, to create a strong predictive model. It works by sequentially training these learners, where each new model tries to correct the errors made by the previous ones. This iterative process continues until a desired level of accuracy is reached.



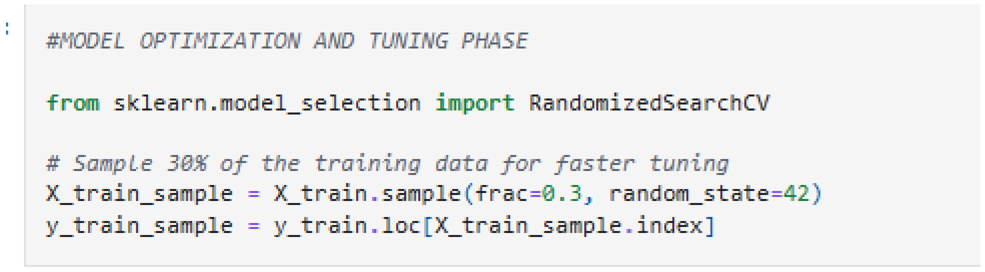
```python
# Gradient Boosting Implementation
gb_clf = GradientBoostingClassifier(
    random_state=42,
    n_estimators=100,
    learning_rate=0.1
)

gb_clf.fit(X_train, y_train)
gb_y_pred = gb_clf.predict(X_test)
print("=== GRADIENT BOOSTING EVALUATION ===")
print("Classification Report:")
print(classification_report(y_test, gb_y_pred))
print(f"\nAccuracy: {accuracy_score(y_test, gb_y_pred):.4f}")
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, gb_y_pred))
print("\n" + "="*50 + "\n")
```

Milestone 5 : Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

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

Activity 2: Applying hyperparameter tuning



```python
#MODEL OPTIMIZATION AND TUNING PHASE

from sklearn.model_selection import RandomizedSearchCV

# Sample 30% of the training data for faster tuning
X_train_sample = X_train.sample(frac=0.3, random_state=42)
y_train_sample = y_train.loc[X_train_sample.index]
```

This code performs hyperparameter tuning for a Random Forest classifier using RandomizedSearchCV on a 30% sample of the training data. It searches for optimal parameters (n_estimators, max_depth, and class_weight) and evaluates the best model on test data using F1 score, accuracy, classification report, and confusion matrix.

```
#Random Forest Fine Tuning

# Define a small parameter distribution
param_dist = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10],
    'class_weight': ['balanced']
}

rf_clf = RandomForestClassifier(random_state=42)

print("Tuning Random Forest with RandomizedSearchCV on a sample...")
rf_random_search = RandomizedSearchCV(
    rf_clf, param_distributions=param_dist, n_iter=4,
    scoring='f1', cv=2, n_jobs=1, verbose=1, random_state=42
)

rf_random_search.fit(X_train_sample, y_train_sample)

print("=== RANDOM FOREST OPTIMIZATION RESULTS ===")
print("Best Parameters:")
print(rf_random_search.best_params_)
print(f"Best Cross-Validation Score: {rf_random_search.best_score_:.4f}")

best_rf = rf_random_search.best_estimator_
rf_y_pred_tuned = best_rf.predict(X_test)

print("\nOptimized Random Forest Evaluation:")
print("Classification Report:")
print(classification_report(y_test, rf_y_pred_tuned))
print(f"Accuracy: {accuracy_score(y_test, rf_y_pred_tuned):.4f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, rf_y_pred_tuned))
print("\n" + "="*70 + "\n")
```

This code performs decision tree fine tuning:

```
# Decision Tree Fine Tuning

# Define a small parameter distribution
dt_param_dist = {
    'max_depth': [None, 10, 15],
    'min_samples_split': [2, 10],
    'min_samples_leaf': [1, 5],
    'criterion': ['gini', 'entropy'],
    'class_weight': ['balanced', {0: 1, 1: 10}]
}

dt_clf = DecisionTreeClassifier(random_state=42)

print("Tuning Decision Tree with RandomizedSearchCV on a sample...")
dt_random_search = RandomizedSearchCV(
    dt_clf, param_distributions=dt_param_dist, n_iter=6,
    scoring='f1', cv=2, n_jobs=1, verbose=1, random_state=42
)

dt_random_search.fit(X_train_sample, y_train_sample)

print("=== DECISION TREE OPTIMIZATION RESULTS ===")
print("Best Parameters:")
print(dt_random_search.best_params_)
print(f"Best Cross-Validation Score: {dt_random_search.best_score_:.4f}")

best_dt = dt_random_search.best_estimator_
dt_y_pred_tuned = best_dt.predict(X_test)

print("\nOptimized Decision Tree Evaluation:")
print("Classification Report:")
print(classification_report(y_test, dt_y_pred_tuned))
print(f"Accuracy: {accuracy_score(y_test, dt_y_pred_tuned):.4f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, dt_y_pred_tuned))
print("\n" + "="*70 + "\n")
```

This code performs KNN fine tuning:

```
# KNN Fine Tuning

# Define a small parameter distribution
knn_param_dist = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
}

knn_clf = KNeighborsClassifier()

print("Tuning KNN with RandomizedSearchCV on a sample...")
knn_random_search = RandomizedSearchCV(
    knn_clf, param_distributions=knn_param_dist, n_iter=4,
    scoring='f1', cv=2, n_jobs=1, verbose=1, random_state=42
)

knn_random_search.fit(X_train_sample, y_train_sample)

print("=== KNN OPTIMIZATION RESULTS ===")
print("Best Parameters:")
print(knn_random_search.best_params_)
print(f"Best Cross-Validation Score: {knn_random_search.best_score_:.4f}")

best_knn = knn_random_search.best_estimator_
knn_y_pred_tuned = best_knn.predict(X_test)

print("\nOptimized KNN Evaluation:")
print("Classification Report:")
print(classification_report(y_test, knn_y_pred_tuned))
print(f"Accuracy: {accuracy_score(y_test, knn_y_pred_tuned):.4f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, knn_y_pred_tuned))
print("\n" + "="*70 + "\n")
```

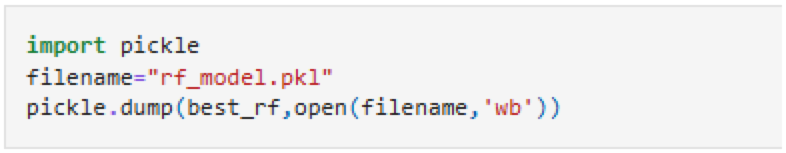
This code performs gradient boosting fine tuning:

```
# Gradient Boosting Fine Tuning

# Define a small parameter distribution
gb_param_dist = {
    'n_estimators': [50, 100],
    'learning_rate': [0.05, 0.1],
    'max_depth': [3, 5],
    'subsample': [0.8, 1.0]
}

gb_clf = GradientBoostingClassifier(random_state=42)

print("Tuning Gradient Boosting with RandomizedSearchCV on a sample...")
gb_random_search = RandomizedSearchCV(
    gb_clf, param_distributions=gb_param_dist, n_iter=4,
    scoring='f1', cv=2, n_jobs=1, verbose=1, random_state=42
)

gb_random_search.fit(X_train_sample, y_train_sample)

print("=== GRADIENT BOOSTING OPTIMIZATION RESULTS ===")
print("Best Parameters:")
print(gb_random_search.best_params_)
print(f"Best Cross-Validation Score: {gb_random_search.best_score_:.4f}")

best_gb = gb_random_search.best_estimator_
gb_y_pred_tuned = best_gb.predict(X_test)

print("\nOptimized Gradient Boosting Evaluation:")
print("Classification Report:")
print(classification_report(y_test, gb_y_pred_tuned))
print(f"Accuracy: {accuracy_score(y_test, gb_y_pred_tuned):.4f}")
print("Confusion Matrix:")
print(confusion_matrix(y_test, gb_y_pred_tuned))
print("\n" + "="*70 + "\n")
```

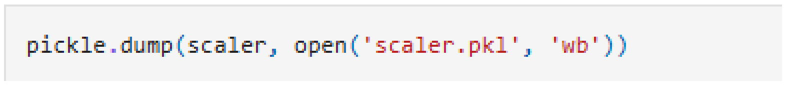
Milestone 6: Model Deployment

Activity 1: Save the best model

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



```
import pickle
filename="rf_model.pkl"
pickle.dump(best_rf,open(filename,'wb'))
```



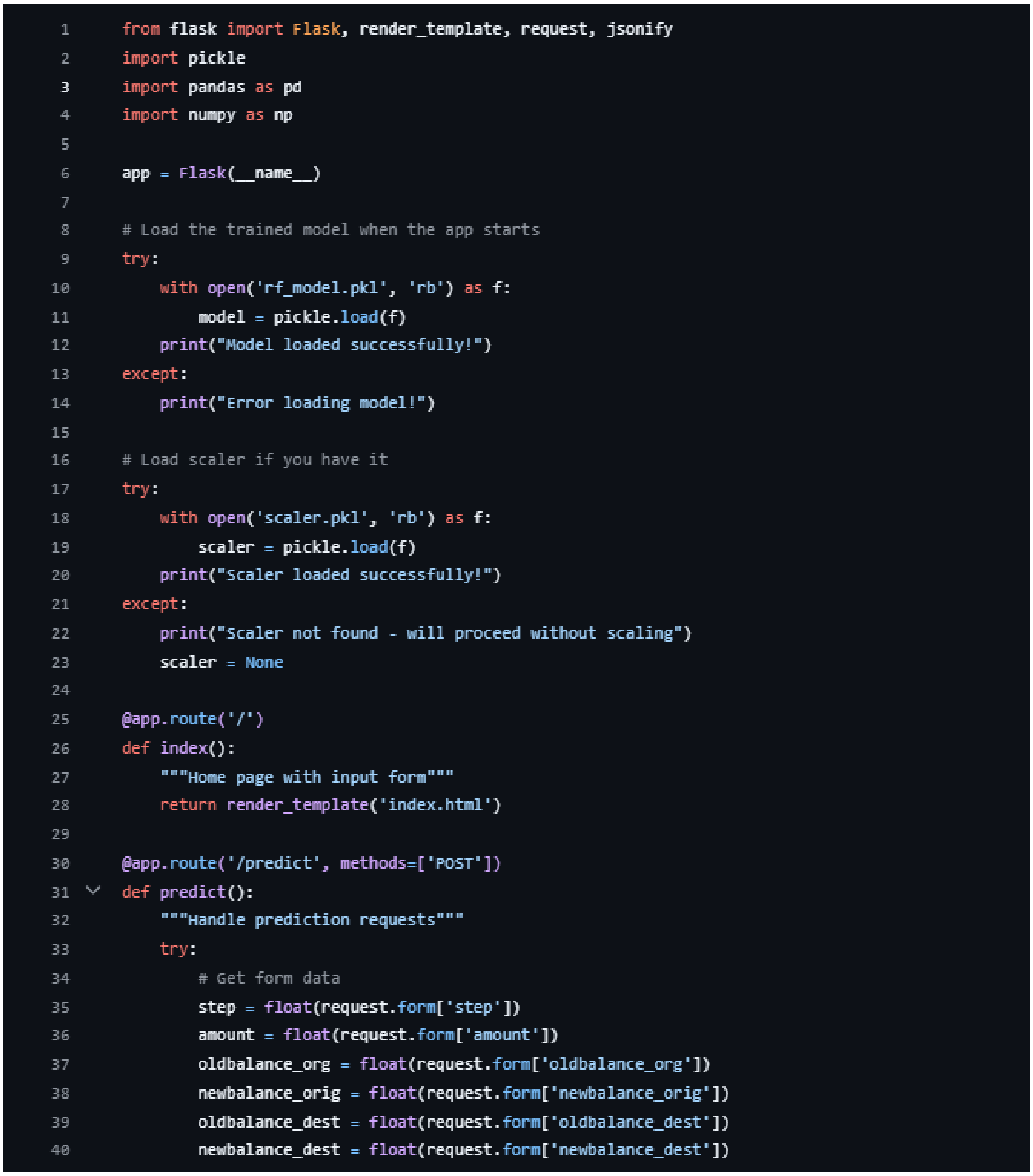
```
pickle.dump(scaler, open('scaler.pkl', 'wb'))
```

Activity 2: Integrate with Web Framework

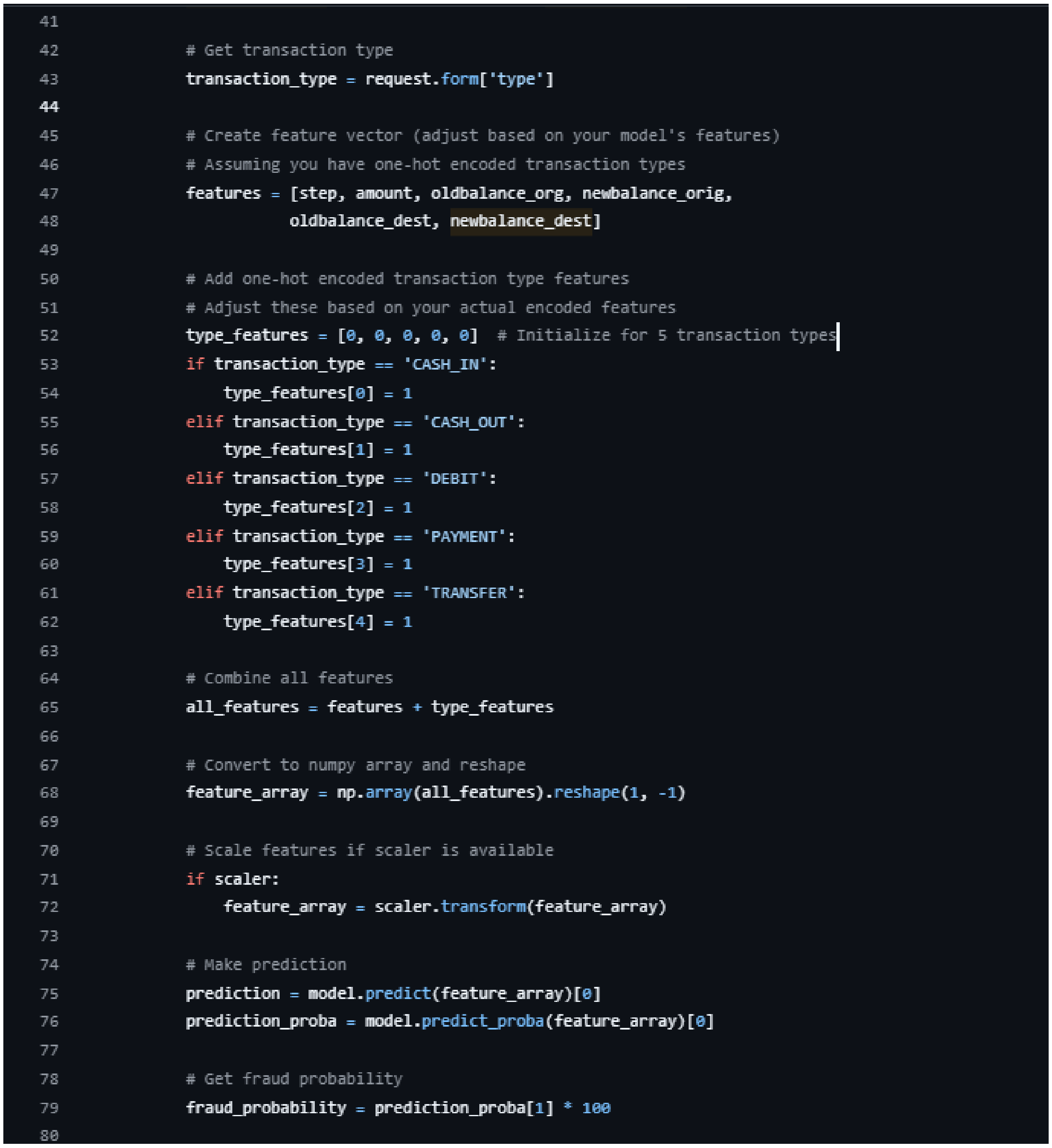
In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

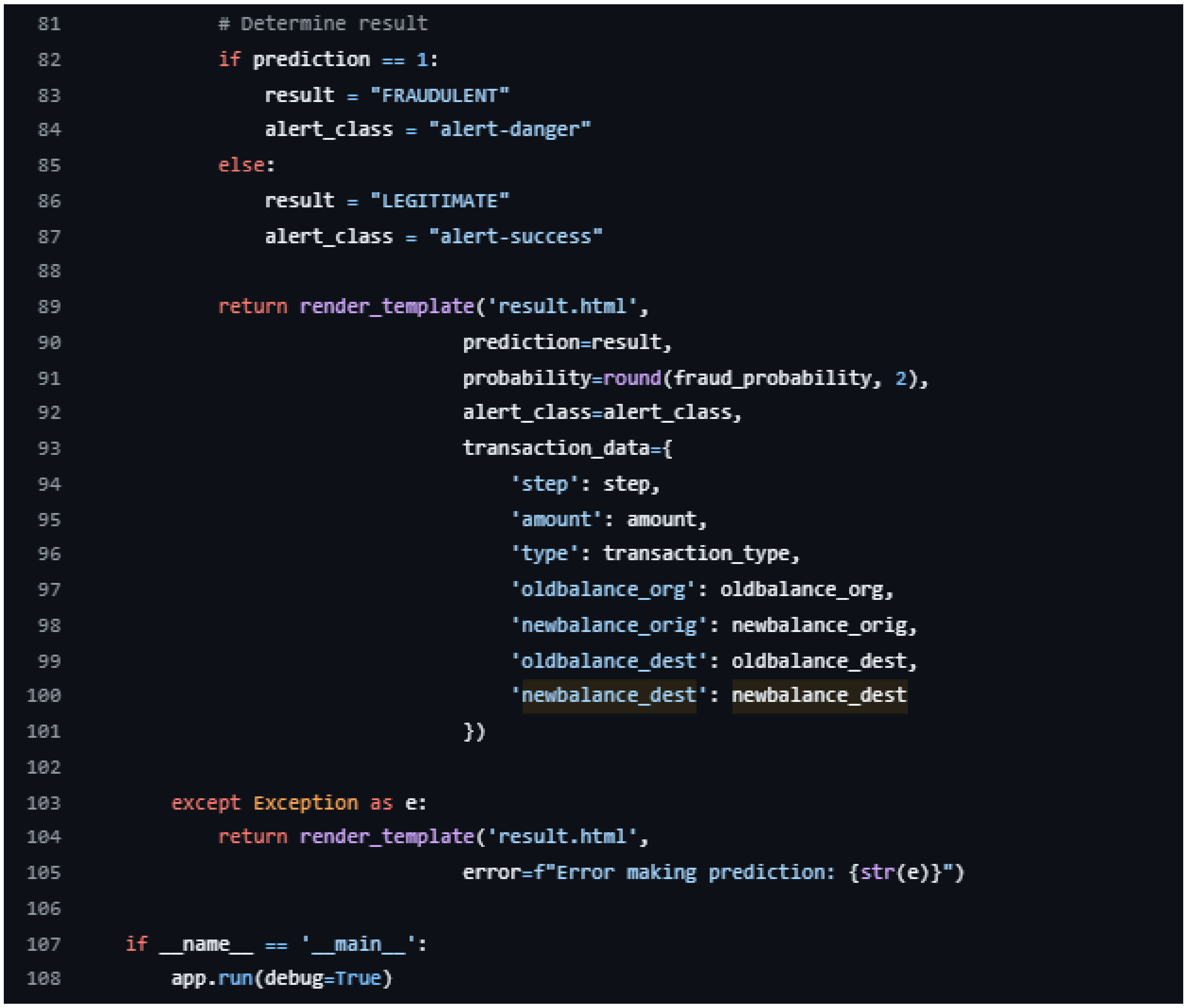
---> Activity 2.1 Import the libraries

---> Activity 2.2 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.



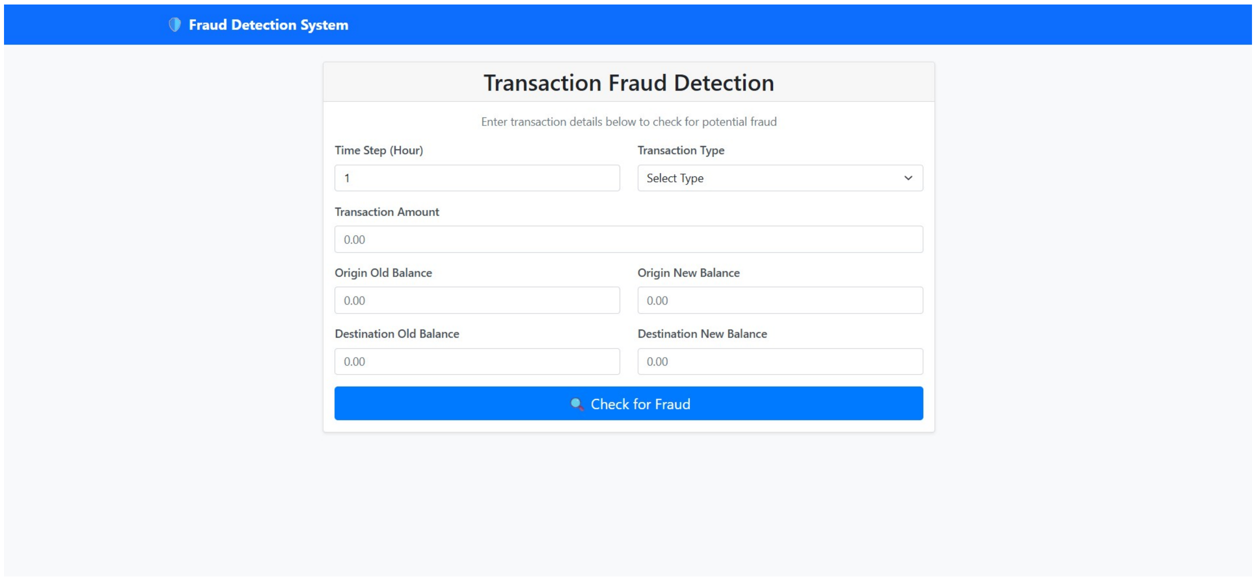
```python
from flask import Flask, render_template, request, jsonify
import pickle
import pandas as pd
import numpy as np

app = Flask(__name__)

# Load the trained model when the app starts
try:
    with open('rf_model.pkl', 'rb') as f:
        model = pickle.load(f)
    print("Model loaded successfully!")
except:
    print("Error loading model!")

# Load scaler if you have it
try:
    with open('scaler.pkl', 'rb') as f:
        scaler = pickle.load(f)
    print("Scaler loaded successfully!")
except:
    print("Scaler not found - will proceed without scaling")
    scaler = None

@app.route('/')
def index():
    """Home page with input form"""
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    """Handle prediction requests"""
    try:
        # Get form data
        step = float(request.form['step'])
        amount = float(request.form['amount'])
        oldbalance_org = float(request.form['oldbalance_org'])
        newbalance_orig = float(request.form['newbalance_orig'])
        oldbalance_dest = float(request.form['oldbalance_dest'])
        newbalance_dest = float(request.form['newbalance_dest'])
```


```

        # Get transaction type
        transaction_type = request.form['type']

        # Create feature vector (adjust based on your model's features)
        # Assuming you have one-hot encoded transaction types
        features = [step, amount, oldbalance_org, newbalance_orig,
                    oldbalance_dest, newbalance_dest]

        # Add one-hot encoded transaction type features
        # Adjust these based on your actual encoded features
        type_features = [0, 0, 0, 0, 0]  # Initialize for 5 transaction types
        if transaction_type == 'CASH_IN':
            type_features[0] = 1
        elif transaction_type == 'CASH_OUT':
            type_features[1] = 1
        elif transaction_type == 'DEBIT':
            type_features[2] = 1
        elif transaction_type == 'PAYMENT':
            type_features[3] = 1
        elif transaction_type == 'TRANSFER':
            type_features[4] = 1

        # Combine all features
        all_features = features + type_features

        # Convert to numpy array and reshape
        feature_array = np.array(all_features).reshape(1, -1)

        # Scale features if scaler is available
        if scaler:
            feature_array = scaler.transform(feature_array)

        # Make prediction
        prediction = model.predict(feature_array)[0]
        prediction_proba = model.predict_proba(feature_array)[0]

        # Get fraud probability
        fraud_probability = prediction_proba[1] * 100

```



```python
        # Determine result
        if prediction == 1:
            result = "FRAUDULENT"
            alert_class = "alert-danger"
        else:
            result = "LEGITIMATE"
            alert_class = "alert-success"

        return render_template('result.html',
                             prediction=result,
                             probability=round(fraud_probability, 2),
                             alert_class=alert_class,
                             transaction_data={
                                 'step': step,
                                 'amount': amount,
                                 'type': transaction_type,
                                 'oldbalance_org': oldbalance_org,
                                 'newbalance_orig': newbalance_orig,
                                 'oldbalance_dest': oldbalance_dest,
                                 'newbalance_dest': newbalance_dest
                             })

    except Exception as e:
        return render_template('result.html',
                             error=f"Error making prediction: {str(e)}")

if __name__ == '__main__':
    app.run(debug=True)
```

Activity 2.3: Run the web application

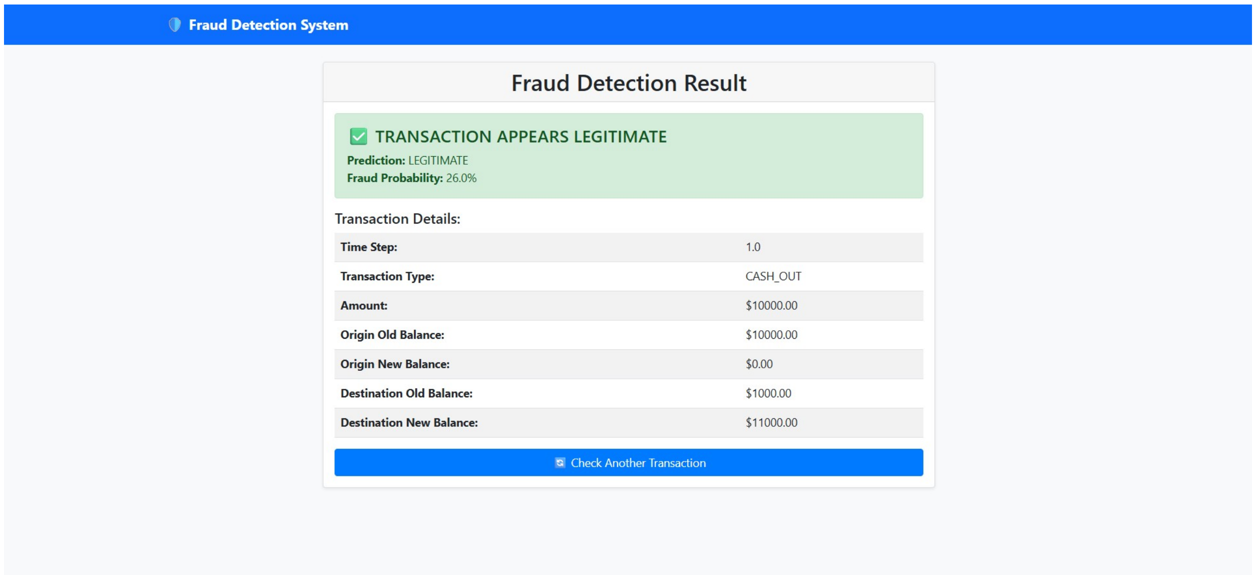


Fraud Detection System

Transaction Fraud Detection

Enter transaction details below to check for potential fraud

Time Step (Hour)	Transaction Type
1	Select Type
Transaction Amount	
0.00	
Origin Old Balance	Origin New Balance
0.00	0.00
Destination Old Balance	Destination New Balance
0.00	0.00



Fraud Detection System

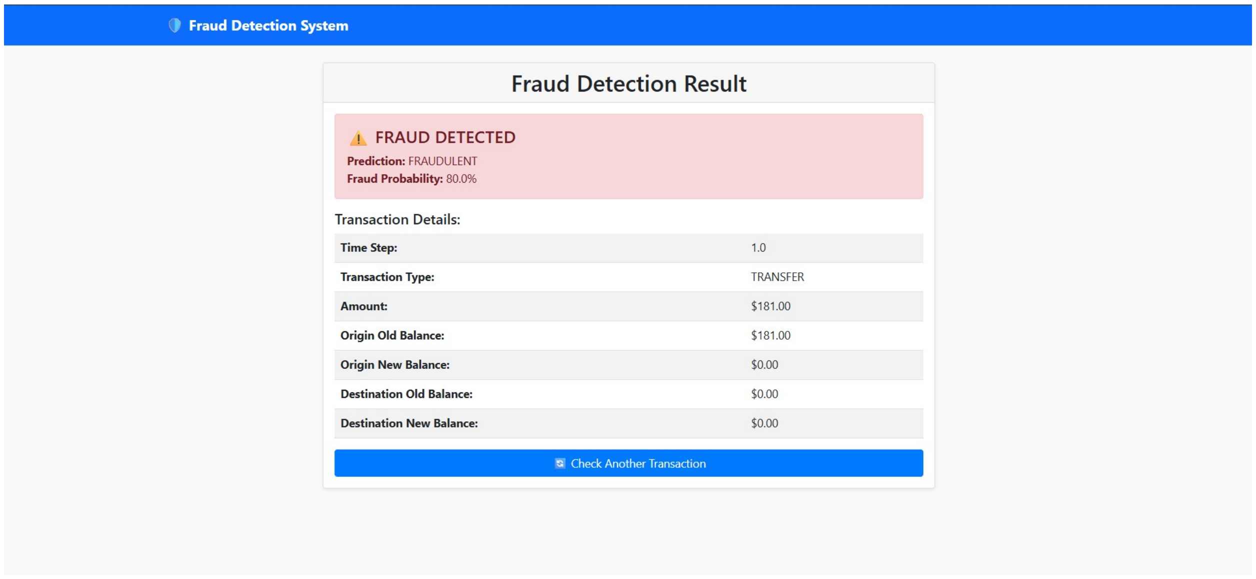
Fraud Detection Result

TRANSACTION APPEARS LEGITIMATE

Prediction: LEGITIMATE
Fraud Probability: 26.0%

Transaction Details:

Time Step:	1.0
Transaction Type:	CASH_OUT
Amount:	$10000.00
Origin Old Balance:	$10000.00
Origin New Balance:	$0.00
Destination Old Balance:	$1000.00
Destination New Balance:	$11000.00



Fraud Detection Result

⚠ FRAUD DETECTED

Prediction: FRAUDULENT
Fraud Probability: 80.0%

Transaction Details:

Time Step:	1.0
Transaction Type:	TRANSFER
Amount:	$181.00
Origin Old Balance:	$181.00
Origin New Balance:	$0.00
Destination Old Balance:	$0.00
Destination New Balance:	$0.00