FASTAG FRAUD DETECTION

CONTENT

PROJECT OVERVIEW

PROJECT STEP'S

DATA DESCRIPTION

CODES & MODEL DEVELOPMENT, WEB API

KEY INSIGHTS

REFERNECES

PROJECT OVERVIEW

The project aims to develop a machine learning-based fraud detection system for Fastag transactions. Utilizing a dataset containing transaction details, vehicle information, geographical locations, and transaction amounts, the objective is to build a robust classification model. This model will accurately identify fraudulent activities, ensuring the security and integrity of Fastag transactions.

PROJECT STEPS

- **DATA EXTRACTION**:- Extract the necessary dataset from KAGGLE Repository. Load the dataset in Python .Use python codes to get a overview of the dataset.
- **DATA CLEANING**:- Address any missing, duplicate, data type conversion or inconsistent data. This includes writing codes to clean the data.
- **DATA ENCODING**:- Data encoding is a crucial step in preparing your dataset for machine learning models. It involves converting categorical data into a numerical format that can be used by the algorithms.
- **DATA UNDERSAMPLING**:- Undersampling involves reducing the number of instances in the majority class to match the number of instances in the minority class. This can help create a more balanced dataset, which can improve the performance of machine learning models.
- **DATA SPLITTING**:- Divide the data into training, validation, and test sets.
- **MODEL TRAINING**:- Train the chosen models on the training data.
- **MODEL EVALUATION**:- Choose appropriate metrics (e.g., accuracy, precision, recall, F1-score) to measure performance. Use the validation set to evaluate models performance.
- **TEST THE MODEL**:- Evaluate the models on the test set. Ensure the model performs well on unseen data.
- **FINAL MODEL**:- Select the model that you have to take forward for deployment purpose.
- **DEVELOP A WEB API:-**Using the required features, a web application is developed in the Fastag Fraud Detection model.

DATA DESCRIPTION

"FASTAG FRAUD DETECTION"

Columns:

- **Transaction_ID**: Unique identifier for each transaction.
- **Timestamp**: Date and time of the transaction.
- **Vehicle_Type**: Type of vehicle (e.g., Bus, Car, Motorcycle, Truck, Van).
- **FastagID**: Unique identifier for the Fastag used.
- **TollBoothID**: Identifier for the toll booth.
- **Lane_Type**: Type of lane (e.g., Express, Regular).
- **Vehicle_Dimensions**: Size category of the vehicle (e.g., Small, Medium, Large).
- **Transaction_Amount**: Total amount charged for the transaction.
- **Amount_paid**: Amount actually paid.
- **Geographical_Location**: Latitude and longitude of the transaction.
- **Vehicle_Speed**: Speed of the vehicle during the transaction.
- **Vehicle_Plate_Number**: License plate number of the vehicle.
- **Fraud_indicator**: Indicator of whether the transaction was fraudulent (Fraud, Not Fraud).

RAW DATA:- “FASTAG FRAUD DETECTION”

Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geographical_Location	Vehicle_Speed	Vehicle_Plate_Number	Fraud_indicator
1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	13.059816123454882, 77.77068662374292	65	KA11AB1234	Fraud
2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	13.059816123454882, 77.77068662374292	78	KA66CD5678	Fraud
3	1/8/2023 18:25	Motorcycle	NaN	D-104	Regular	Small	0	0	13.059816123454882, 77.77068662374292	53	KA88EF9012	Not Fraud
4	1/9/2023 2:05	Truck	FTG-044-LMN-322	C-103	Regular	Large	350	120	13.059816123454882, 77.77068662374292	92	KA11GH3456	Fraud
5	1/10/2023 6:35	Van	FTG-505-DEF-652	B-102	Express	Medium	140	100	13.059816123454882, 77.77068662374292	60	KA44IJ6789	Fraud

CODES & MODEL DEVELOPMENT

ONEHOT ENCODING

We employed a one-hot encoder to encode the vehicle type and subsequently dropped the following columns: 'Timestamp', 'FastagID', 'Vehicle_Type', 'TollBoothID', 'Geographical_Location', and 'Vehicle_Plate_Number'.

```
data['Vehicle_Type'].unique()
```

```
array(['Bus ', 'Car', 'Motorcycle', 'Truck', 'Van', 'Sedan', 'SUV'],
      dtype=object)
```

```
Lane_order=['Express', 'Regular']
Vehicle_Dimensions_order=['Large', 'Small', 'Medium']
Fraud_indicator_order=['Not Fraud','Fraud']
```

```
ohe = OneHotEncoder()
encode0 = ohe.fit_transform(data[['Vehicle_Type']]).toarray()
```

```
feature_labels = ohe.categories_
np.array(feature_labels).ravel()
```

```
array(['Bus ', 'Car', 'Motorcycle', 'SUV', 'Sedan', 'Truck', 'Van'],
      dtype=object)
```

```
feature_labels = np.array(feature_labels).ravel()
print(feature_labels)
```

```
['Bus ' 'Car' 'Motorcycle' 'SUV' 'Sedan' 'Truck' 'Van']
```

```
features = pd.DataFrame(encode0, columns = feature_labels)
```

```
df_new = pd.concat([data, features], axis=1)
```

ORDINAL ENCODING

We utilized an ordinal encoder to encode the columns "Lane_order," "Vehicle_Dimensions_order," and "Fraud_indicator_order.“ We used an ordinal encoder to encode "Lane_order," "Vehicle_Dimensions_order," and "Fraud_indicator_order," integrating the encoded values into the original table by replacing the existing columns with the new encoded columns.

```
encode1 = OrdinalEncoder(categories=[Lane_order])
encode2 = OrdinalEncoder(categories=[Vehicle_Dimensions_order])
encode3 = OrdinalEncoder(categories=[Fraud_indicator_order])
```

```
encode1.fit(new_dataset[['Lane_Type']])
encode2.fit(new_dataset[['Vehicle_Dimensions']])
encode3.fit(new_dataset[['Fraud_indicator']])
```

```
OrdinalEncoder(categories=[['Not Fraud', 'Fraud']])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
new_lane=pd.DataFrame(encode1.transform(new_dataset[['Lane_Type']]))
new_dimensuions=pd.DataFrame(encode2.transform(new_dataset[['Vehicle_Dimensions']]))
new_fraud_indicator=pd.DataFrame(encode3.transform(new_dataset[['Fraud_indicator']]))
```

```
new_dataset['Lane_Type']= new_lane
new_dataset['Vehicle_Dimensions']= new_dimensuions
new_dataset['Fraud_indicator']=new_fraud_indicator
```

UNDER SAMPLING CODES

Highly Unblanced dataset

0-> normal transaction

1-> Fraud transaction

```
# separating data for analysis
legit = data[data.Fraud_indicator == 0]
fraud = data[data.Fraud_indicator == 1]
```

```
print(legit.shape)
print(fraud.shape)
```

```
(4017, 14)
(983, 14)
```

```
#statistical method of the data
legit.Transaction_Amount.describe()
```

```
count    4017.000000
mean      153.110530
std       114.435986
min         0.000000
25%        90.000000
50%       125.000000
75%       290.000000
max       350.000000
Name: Transaction_Amount, dtype: float64
```

Under-sampling is a technique used in data analysis and machine learning to address class imbalance issues in datasets. Class imbalance occurs when one class (or category) of data significantly outnumbers the other classes. This can cause problems for machine learning algorithms, as they may become biased towards the majority class and perform poorly on the minority class. Under-sampling aims to balance the class distribution by reducing the number of instances in the majority class.

OUTPUT:-

Under Sampling

```
legit_sample = legit.sample(n=983)
```

Concatenating two DataFrames

```
new_dataset = pd.concat([legit_sample, fraud], axis=0)
```

```
new_dataset.head()
```

	Transaction_ID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Vehicle_Speed
2003	2004	1.0	1.0	90	90	90
1924	1925	0.0	0.0	140	140	58
3091	3092	1.0	0.0	290	290	49
3352	3353	0.0	2.0	100	100	45
109	110	1.0	2.0	140	140	87

```
#dristibution of legit and fraud transactions in new dataset
new_dataset['Fraud_indicator'].value_counts()
```

```
Fraud_indicator
0.0    983
1.0    983
Name: count, dtype: int64
```

MODELS CODE

· As the Ordinal encoding and balancing of data is done. Then data is divided into train and test, then later back into x and y. 80% percent of the data is used as train data. After this the model will be created . Four machine learning models—the decision tree, random forest, logistic regression, and SVM classification on the fastag data—have been created, and we predict each one of them. The model's output is expressed as accuracy, f1 score, precision, and recall.

```python
Models = {
    "Decision Tree":DecisionTreeClassifier(),
    "Random Forest":RandomForestClassifier(),
    "Logistic Regression":LogisticRegression(),
    "SVM Classification": SVC()
}

for i in range (len(list(Models))):
    Model=list(Models.values())[i]

    #train Model
    Model.fit(X_train, Y_train)

    #Make predictions
    Y_train_pred = Model.predict(X_train)
    Y_test_pred = Model.predict(X_test)

    #Training Performance
    model_train_Accuracy = accuracy_score(Y_train, Y_train_pred)
    model_train_Precision = precision_score(Y_train, Y_train_pred)
    model_train_recall = recall_score(Y_train, Y_train_pred)
    model_train_F1 = f1_score(Y_train, Y_train_pred, average='weighted')

    #Testing Performance
    model_test_Accuracy = accuracy_score(Y_test, Y_test_pred)
    model_test_Precision = precision_score(Y_test, Y_test_pred)
    model_test_recall = recall_score(Y_test, Y_test_pred)
    model_test_F1 = f1_score(Y_test, Y_test_pred, average='weighted')

    print(list(Models.keys())[i])

    print("Models Performance for Training Set")
    print("- Accuracy: {:.4f}".format(model_train_Accuracy))
    print("- Precision: {:.4f}".format(model_train_Precision))
    print("- Recall: {:.4f}".format(model_train_recall))
    print("- F1 Score: {:.4f}".format(model_train_F1))


    print("-------------------------")

    print("Models Performance for Testing Set")
    print("- Accuracy: {:.4f}".format(model_test_Accuracy))
    print("- Precision: {:.4f}".format(model_test_Precision))
    print("- Recall: {:.4f}".format(model_test_recall))
    print("- F1 Score: {:.4f}".format(model_test_F1))

    print('='*35)
    print('\n')
```

```
Decision Tree
Models Performance for Training Set
- Accuracy: 1.0000
- Precision: 1.0000
- Recall: 1.0000
- F1 Score: 1.0000
--------------------------
Models Performance for Testing Set
- Accuracy: 0.9924
- Precision: 1.0000
- Recall: 0.9848
- F1 Score: 0.9924
===================================


Random Forest
Models Performance for Training Set
- Accuracy: 1.0000
- Precision: 1.0000
- Recall: 1.0000
- F1 Score: 1.0000
--------------------------
Models Performance for Testing Set
- Accuracy: 0.9797
- Precision: 0.9948
- Recall: 0.9645
- F1 Score: 0.9797
===================================


Logistic Regression
Models Performance for Training Set
- Accuracy: 0.9771
- Precision: 1.0000
- Recall: 0.9542
- F1 Score: 0.9771
--------------------------
Models Performance for Testing Set
- Accuracy: 0.9822
- Precision: 1.0000
- Recall: 0.9645
- F1 Score: 0.9822
===================================
```

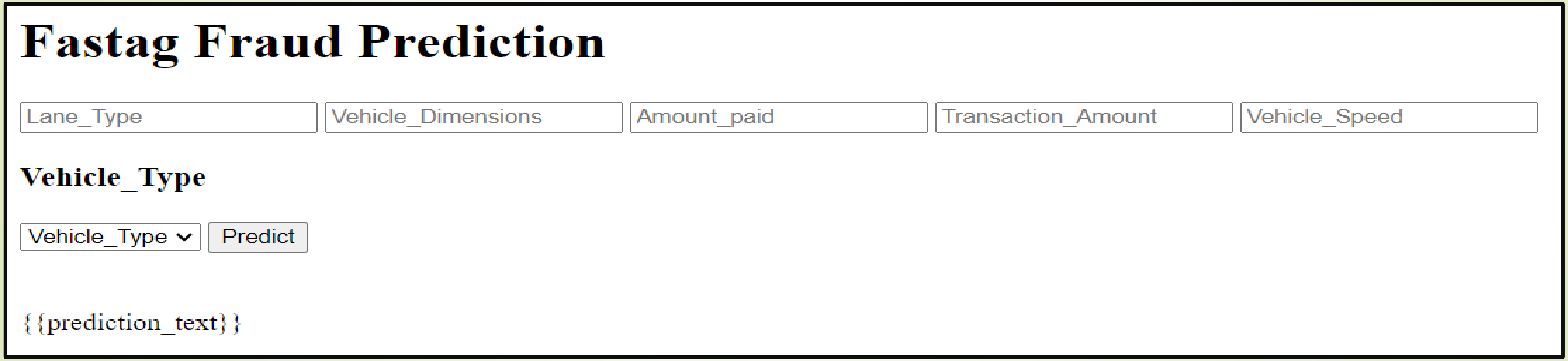
OUTPUTS

```
SVM Classification
Models Performance for Training Set
- Accuracy: 0.6838
- Precision: 0.7213
- Recall: 0.5992
- F1 Score: 0.6816
--------------------------
Models Performance for Testing Set
- Accuracy: 0.7259
- Precision: 0.7947
- Recall: 0.6091
- F1 Score: 0.7221
===================================
```

After getting an accurate model now we have created a pickle file as **“random_forest_model.pkl”** which further will be used in flask.

WEB API

A web API is an application programming interface for either a web server or a web browser. As a web development concept, it can be related to a web application's client side. A web API is an application programming interface (API) for either a web server or a web browser. As a web development concept, it can be related to a web application's client side (including any web frameworks being used). Using the required features, a web application is developed in the Fastag Fraud Detection model. It will forecast whether fraud is occurring or not by utilizing the hidden data.



HTML CODE:-

```
<!DOCTYPE html>
<html >
<!--From https://codepen.io/frytyler/pen/EGdtg-->
<head>
  <meta charset="UTF-8">
  <title>ML API</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300' rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300' rel='stylesheet' type='text/css'>

</head>
<body>
 <div class="login">

      <!-- Main Input For Receiving Query to our ML -->
    <form action="{{ url_for('predict')}}"method="post">
      <input type="text" name="Lane_Type" placeholder="Lane_Type" required="required" />
        <input type="text" name="Vehicle_Dimensions" placeholder="Vehicle_Dimensions" required="required" />
    <input type="text" name="Amount_paid" placeholder="Amount_paid" required="required" />
        <input type="text" name="Transaction_Amount" placeholder="Transaction_Amount" required="required" />
        <input type="text" name="Vehicle_Speed" placeholder="Vehicle_Speed" required="required" />
        <h3>Vehicle_Type</h3>
            <select id="sixth" class="form-input align-center" name="Vehicle_Type">
                <option value="None">Vehicle_Type</option>
                <option value="Bus">Bus</option>
                <option value="Car">Car</option>
                <option value="Motorcyle">Motorcyle</option>
                <option value="SUV">SUV</option>
                <option value="Sedan">Sedan</option>
                <option value="Truck">Truck</option>
                <option value="Van">Van</option>
              </select>

        <button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
    </form>

   <br>
   <br>
   {{prediction_text}}

 </div>


</body>
</html>
```

KEY INSIGHTS

- Decision Tree and Random Forest have a 100% in Training and Test data accuracy than Logistic Regression of 99% and an SVC of 69.09%
- When comparing precision & recall for 4 models, Here the Decision tree and Random forest performed much better than the Logistics Regression and SVC as we can see that the detection of fraud cases is around 100 % and 98 %, and Logistics Regression and SVC of 72% and 62%.
- So overall Decision tree and Random Forest Method performed much better in determining the fraud cases which is 100%.
- We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense. We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases.

REFERENCES

Data Source:
thegoanpanda. "Fastag Fraud Detection Datasets." Kaggle, 2024. Available at: Fastag Fraud Detection Datasets

Data Access Method:
The dataset was directly downloaded using the Kaggle with the following link:
https://www.kaggle.com/datasets/thegoanpanda/fastag-fraud-detection-datesets fictitious?resource=download

The dataset can also be accessed and downloaded using the Kaggle API with the following command:
kaggle datasets download -d thegoanpanda/fastag-fraud-detection-datesets-fictitious

This commands allows for easy programmatic access and integration of the dataset into data analysis workflows.

THANK
YOU