FAST TAG FRAUD DETECTION

USING MACHINE LEARNING FOR REAL-TIME FRAUD DETECTION

SONAL SONARGHARE

INTRODUCTION

OVERVIEW:

- FASTag is an electronic toll collection system in India.
- Fraud detection is crucial to prevent revenue loss.
- This project applies machine learning to detect fraudulent FASTag transactions.

PROBLEM STATEMENT

PROBLEM:

- Increasing fraudulent FASTag transactions.
- Manual detection methods are inefficient.
- Significant revenue losses due to undetected fraud.

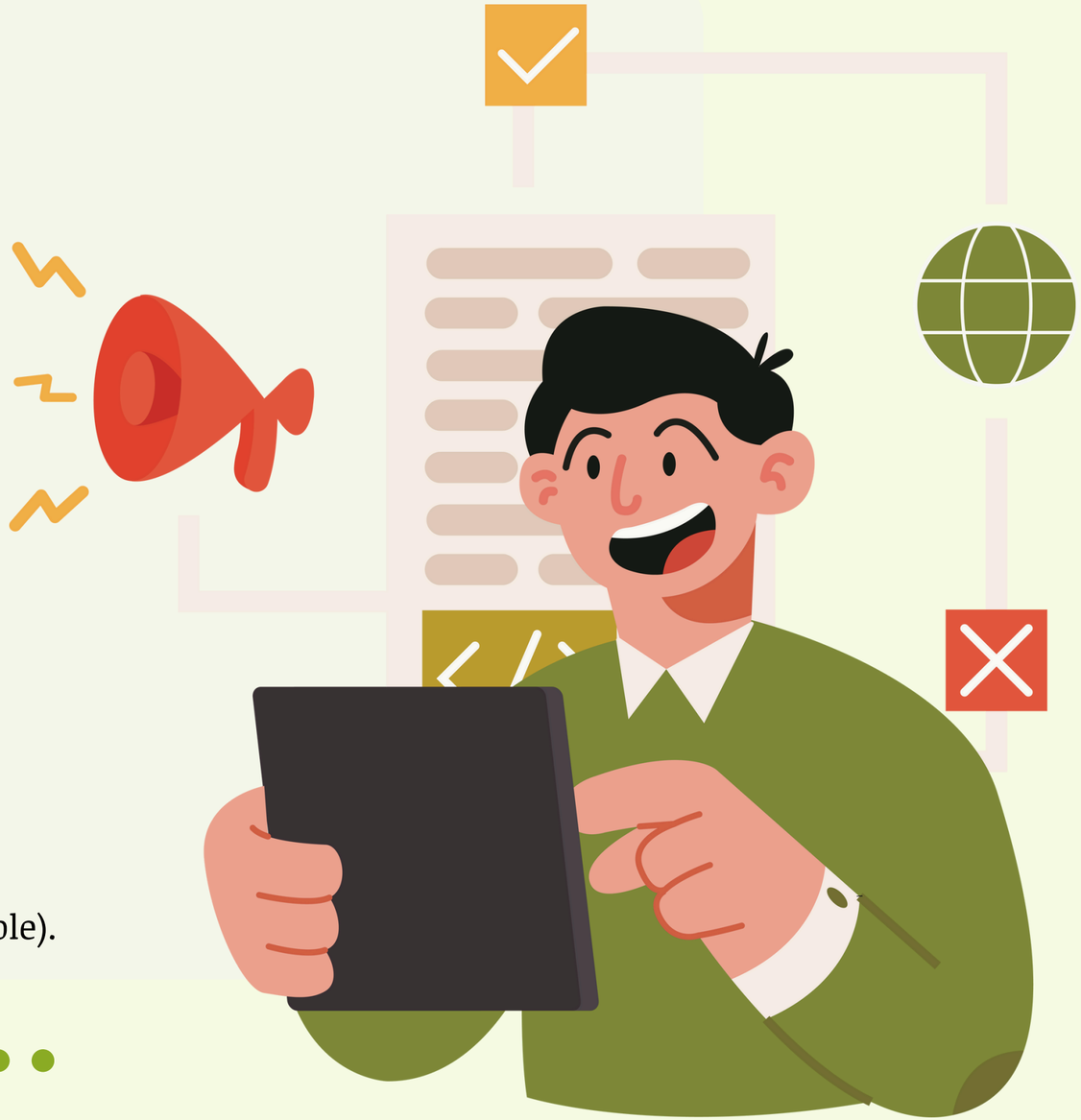
OBJECTIVE:

- Develop a machine learning solution for real-time fraud detection in FASTag transactions.

DATASET DESCRIPTION

DATASET DESCRIPTION:

1. **Transaction_ID:** Unique identifier for each transaction.
2. **Timestamp:** Date and time of the transaction.
3. **Vehicle_Type:** Type of vehicle involved in the transaction.
4. **FastagID:** Unique identifier for Fastag.
5. **TollBoothID:** Identifier for the toll booth.
6. **Lane_Type:** Type of lane used for the transaction.
7. **Vehicle_Dimensions:** Dimensions of the vehicle.
8. **Transaction_Amount:** Amount associated with the transaction.
9. **Amount_paid:** Amount paid for the transaction.
10. **Geographical_Location:** Location details of the transaction.
11. **Vehicle_Speed:** Speed of the vehicle during the transaction.
12. **Vehicle_Plate_Number:** License plate number of the vehicle.
13. **Fraud_indicator:** Binary indicator of fraudulent activity (target variable).



EXPLORATORY DATA ANALYSIS (EDA)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, auc
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings('ignore')
```

← Import Python Libraries

```
file_path = 'FastagFraudDetection.csv'
# Read the CSV file into a pandas DataFrame
df = pd.read_csv(file_path)
```

← Read CSV File

EXPLORATORY DATA ANALYSIS (EDA)

```
df.head()
```

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geographical_Location	Vehicle_Speed	Vehicle_Plate_Number	Fraud_indicator
0	1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	13.059816123454882, 77.77068662374292	65	KA11AB1234	Fraud
1	2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	13.059816123454882, 77.77068662374292	78	KA66CD5678	Fraud
2	3	1/8/2023 18:25	Motorcycle	NaN	D-104	Regular	Small	0	0	13.059816123454882, 77.77068662374292	53	KA88EF9012	Not Fraud

First few Rows

```
df.tail()
```

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geographical_Location	Vehicle_Speed	Vehicle_Plate_Number	Fraud_indicator
4995	4996	1/1/2023 22:18	Truck	FTG-445-EDC-765	C-103	Regular	Large	330	330	13.21331620748757, 77.55413526894684	81	KA74ST0123	Not Fraud
4996	4997	1/17/2023 13:43	Van	FTG-446-LMK-432	B-102	Express	Medium	125	125	13.21331620748757, 77.55413526894684	64	KA38UV3456	Not Fraud
4997	4998	2/5/2023 5:08	Sedan	FTG-447-PLN-109	A-101	Regular	Medium	115	115	13.21331620748757, 77.55413526894684	93	KA33WX6789	Not Fraud

Last few Rows

Shape & Info

```
df.shape

(5000, 13)


# Display basic information about the dataset
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Transaction_ID        5000 non-null   int64
 1   Timestamp             5000 non-null   object
 2   Vehicle_Type          5000 non-null   object
 3   FastagID              4451 non-null   object
 4   TollBoothID           5000 non-null   object
 5   Lane_Type             5000 non-null   object
 6   Vehicle_Dimensions    5000 non-null   object
 7   Transaction_Amount    5000 non-null   int64
 8   Amount_paid           5000 non-null   int64
 9   Geographical_Location 5000 non-null   object
 10  Vehicle_Speed         5000 non-null   int64
 11  Vehicle_Plate_Number  5000 non-null   object
 12  Fraud_indicator       5000 non-null   object
dtypes: int64(4), object(9)
memory usage: 507.9+ KB
```

```
df.describe()
```

	Transaction_ID	Transaction_Amount	Amount_paid	Vehicle_Speed
count	5000.000000	5000.00000	5000.000000	5000.000000
mean	2500.500000	161.06200	141.261000	67.851200
std	1443.520003	112.44995	106.480996	16.597547
min	1.000000	0.00000	0.000000	10.000000
25%	1250.750000	100.00000	90.000000	54.000000
50%	2500.500000	130.00000	120.000000	67.000000
75%	3750.250000	290.00000	160.000000	82.000000
max	5000.000000	350.00000	350.000000	118.000000

Summary Statistics

EXPLORATORY DATA ANALYSIS (EDA)

Check for Unique Values

```
df.nunique()
```

```
Transaction_ID          5000
Timestamp               4423
Vehicle_Type               7
FastagID                4451
TollBoothID                6
Lane_Type                  2
Vehicle_Dimensions         3
Transaction_Amount        20
Amount_paid               23
Geographical_Location      5
Vehicle_Speed             85
Vehicle_Plate_Number    5000
Fraud_indicator            2
dtype: int64
```

Check for Missing Values

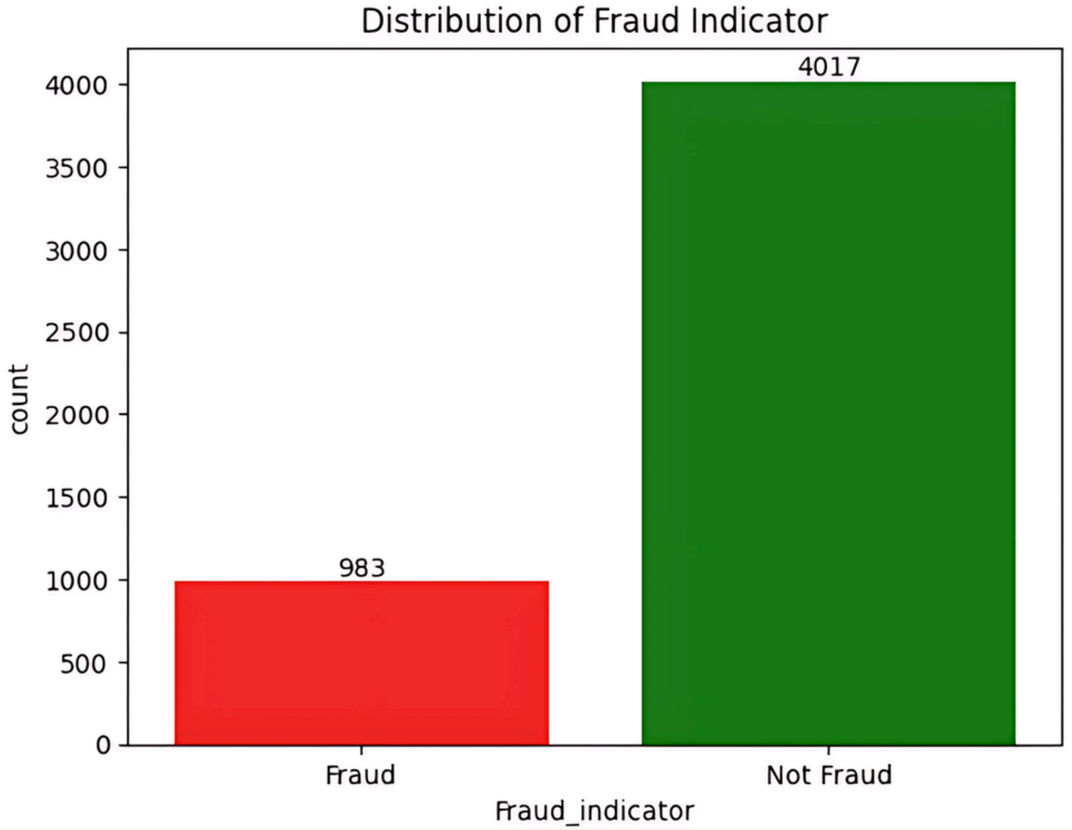
```
df.isnull().sum()
```

```
Transaction_ID            0
Timestamp                 0
Vehicle_Type              0
FastagID                549
TollBoothID               0
Lane_Type                 0
Vehicle_Dimensions        0
Transaction_Amount        0
Amount_paid               0
Geographical_Location     0
Vehicle_Speed             0
Vehicle_Plate_Number      0
Fraud_indicator           0
dtype: int64
```

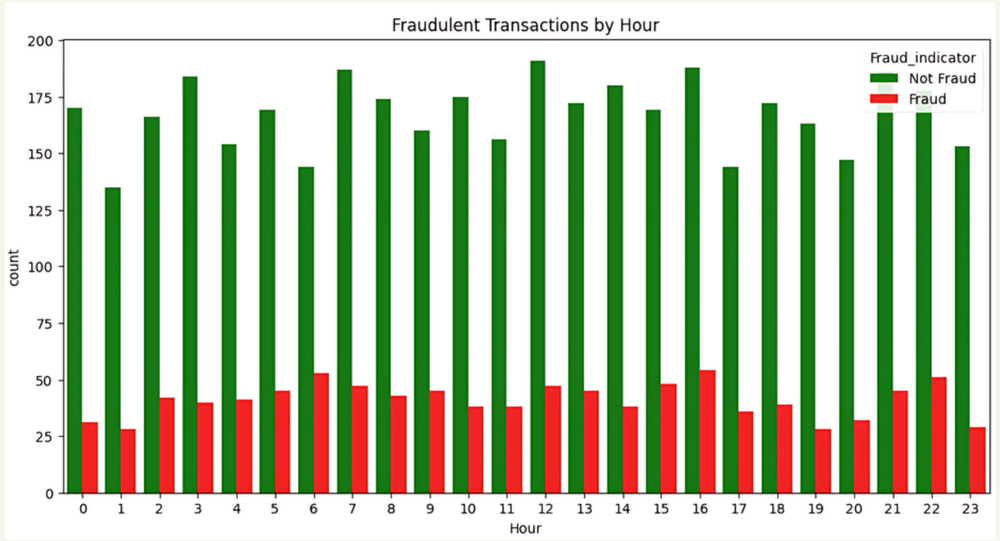
DATA VISUALIZATION

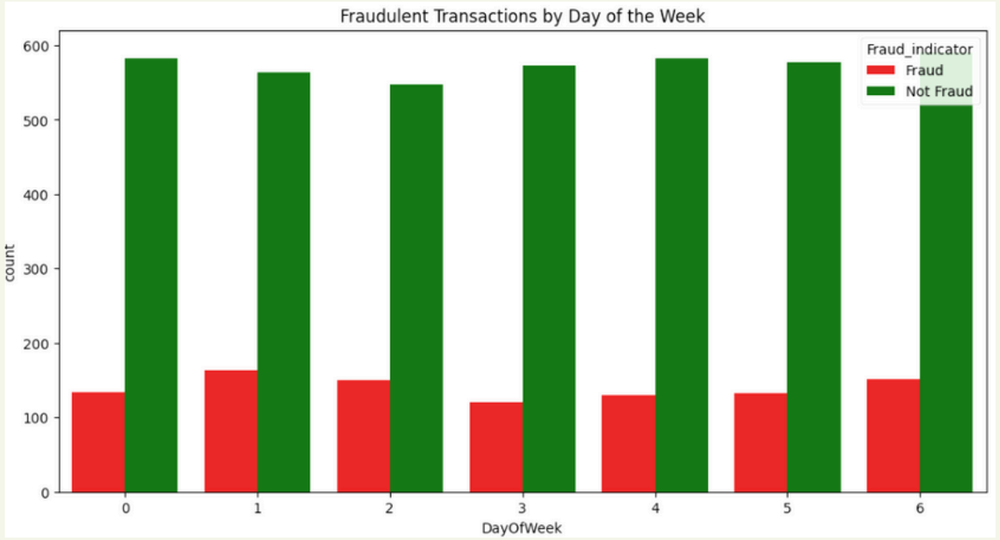
FRAUD DISTRIBUTION

```
# Create the bar plot
colors = {'Fraud': 'red', 'Not Fraud': 'Green'}
ax = sns.countplot(data=df, x='Fraud_indicator', palette=colors)
ax.set_title('Distribution of Fraud Indicator')
for bars in ax.containers:
    ax.bar_label(bars)
```



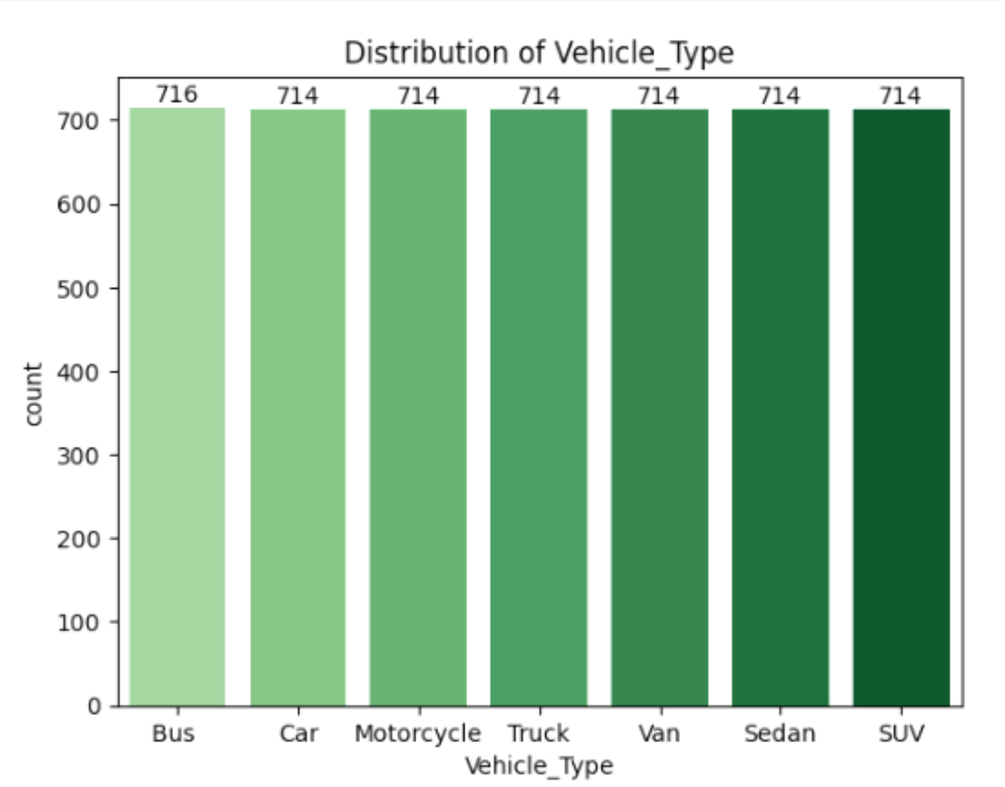
Fraudulent transactions by hour & DayOfWeek



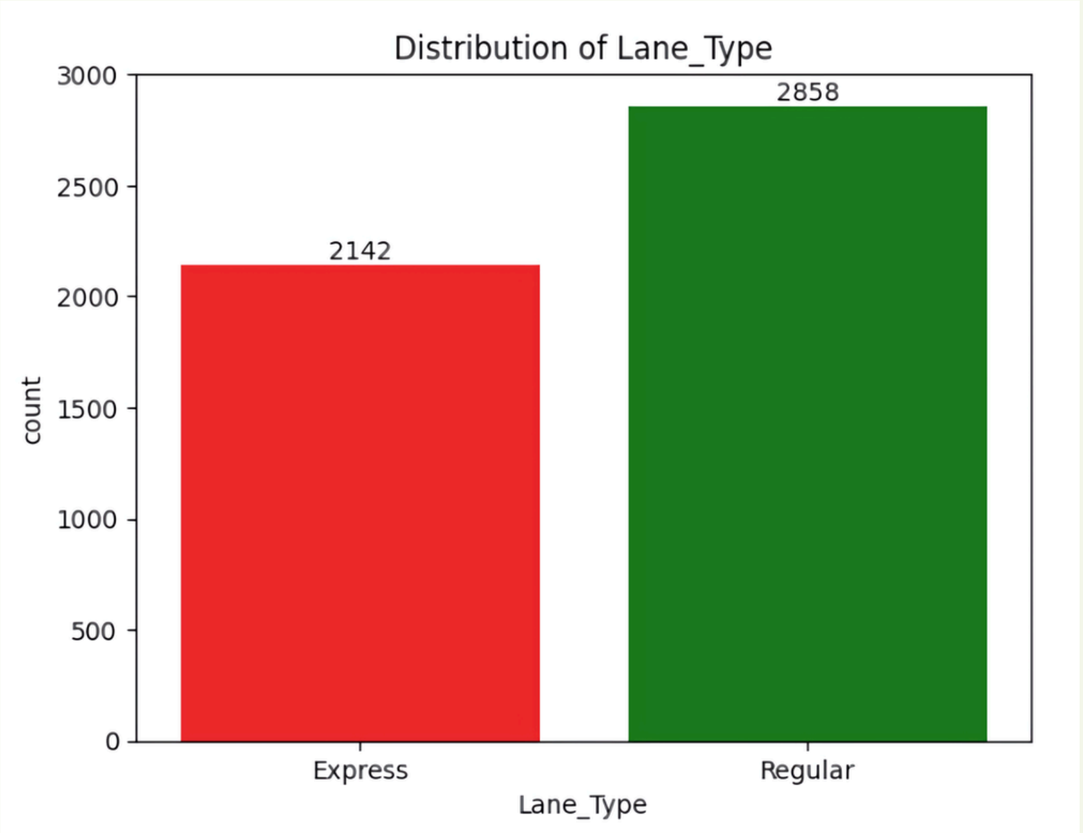


DATA VISUALIZATION

DISTRIBUTION OF VEHICLE TYPE

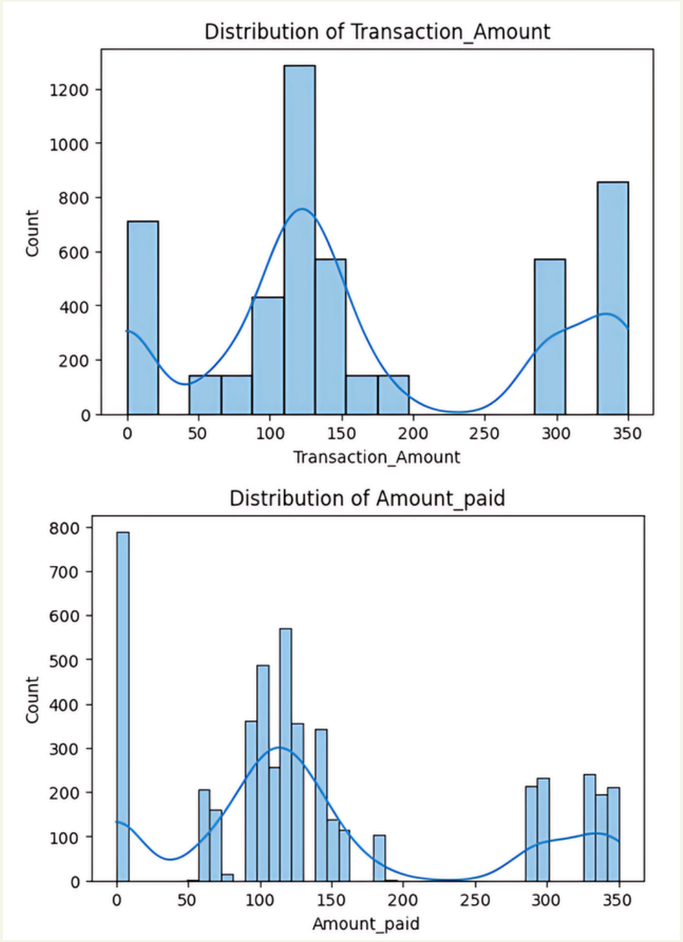


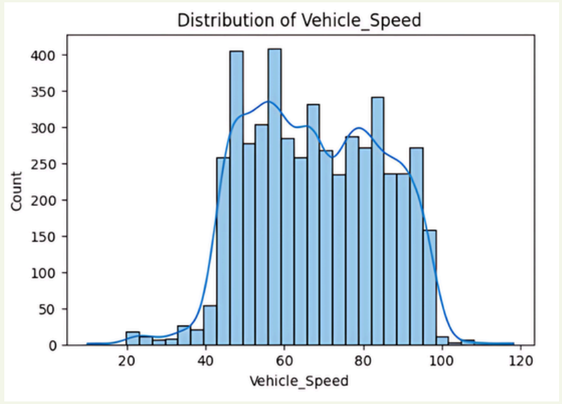
DISTRIBUTION OF LANE TYPE



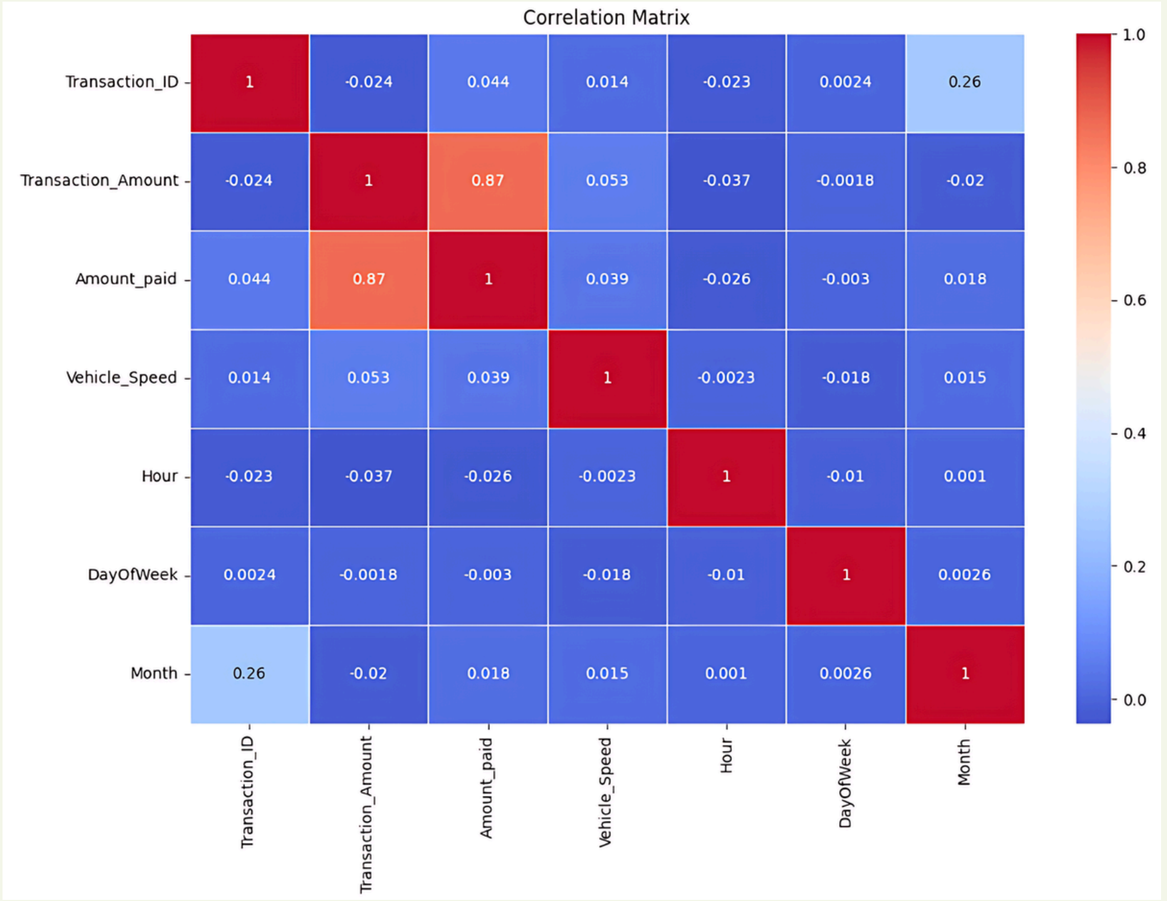
DATA VISUALIZATION

DISTRIBUTION OF NUMERICAL FEATURES





CORRELATION MATRIX



FEATURE ENGINEERING

```python
# Encode categorical variables
categorical_features = ['Vehicle_Type', 'Lane_Type', 'Geographical_Location', 'Vehicle_Plate_Number','Vehicle_Dimensions', 'FastagID', 'TollBoothID','Fraud_indicator']
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    df[feature] = label_encoders[feature].fit_transform(df[feature])

# Check the balance of the target variable
print("\nTarget variable distribution:")
print(df['Fraud_indicator'].value_counts())

# Address imbalance using SMOTE
X = df.drop(columns=['Transaction_ID', 'Timestamp', 'Fraud_indicator'])
y = df['Fraud_indicator']

smote = SMOTE(random_state=42)
X_res, y_res = smote.fit_resample(X, y)

print("\nBalanced target variable distribution:")
print(pd.Series(y_res).value_counts())
```

Encode categorical variables using Label Encoding

Checking Target Variable Balance

Use SMOTE to handle class imbalance in the target variable.

```
Target variable distribution:
1    4017
0     983
Name: Fraud_indicator, dtype: int64

Balanced target variable distribution:
0    4017
1    4017
Name: Fraud_indicator, dtype: int64
```

Balancing Using SMOTE: Applies SMOTE to balance the classes by oversampling the minority class (fraudulent transactions) and undersampling the majority class (non-fraudulent transactions).

MODEL DEVELOPMENT & EVALUATION

SPLIT THE DATA INTO TRAINING AND TESTING SETS 70:30

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.3, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

TRAIN MULTIPLE MODELS:

1. Random Forest
2. MLPClassifier (Neural Network)
3. Gradient Boosting
4. SVC (Support Vector Classifier)

FINE-TUNING:

Hyperparameter tuning using RandomizedSearchCV and GridSearchCV.

EVALUATION METRICS:

1. Accuracy
2. Precision
3. Recall
4. F1 Score
5. ROC-AUC Curve

MODEL DEVELOPMENT & EVALUATION

RANDOM FOREST CLASSIFIER

```python
# Train a RandomForest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate the model
print("Random Forest Classifier")
print("\nClassification Report:")
report = classification_report(y_test, y_pred, digits=6)
print(report)

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

# Exact Calculation of Metrics
TP = conf_matrix[1, 1]
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]
FN = conf_matrix[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)

print(f"\nExact Accuracy: {accuracy:.10f}")
print(f"Exact Precision: {precision:.10f}")
print(f"Exact Recall: {recall:.10f}")
print(f"Exact F1 Score: {f1_score:.10f}")
```

```
Random Forest Classifier

Classification Report:
              precision    recall  f1-score   support

           0   1.000000  0.986622  0.993266      1196
           1   0.987002  1.000000  0.993459      1215

    accuracy                       0.993364      2411
   macro avg   0.993501  0.993311  0.993362      2411
weighted avg   0.993450  0.993364  0.993363      2411


Confusion Matrix:
[[1180   16]
 [   0 1215]]

Exact Accuracy: 0.9933637495
Exact Precision: 0.9870024370
Exact Recall: 1.0000000000
Exact F1 Score: 0.9934587081
```

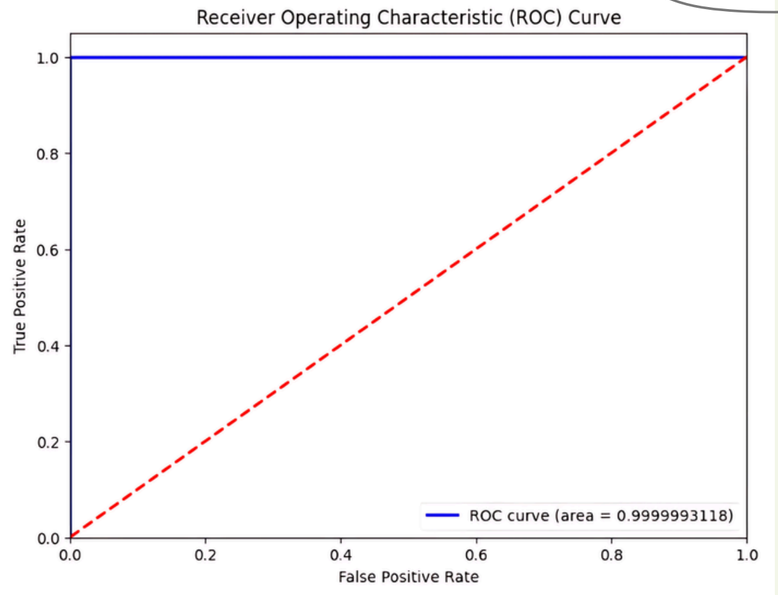
0.9933

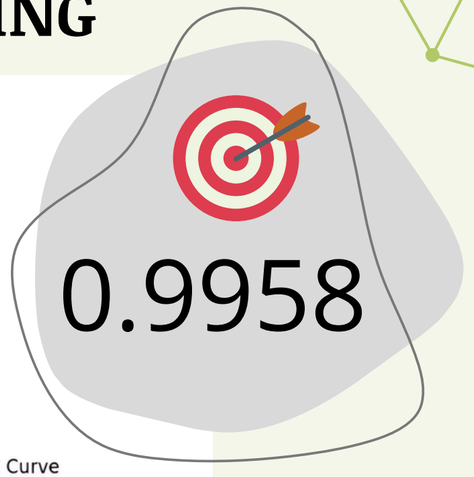
MODEL DEVELOPMENT & EVALUATION

RANDOM FOREST CLASSIFIER-HYPERTUNING

```python
from sklearn.model_selection import RandomizedSearchCV
# Define the parameter grid
param_distributions = {
    'n_estimators': [100, 200, 300, 400, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'bootstrap': [True, False]
}

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=clf, param_distributions=param_distributions,
                                   n_iter=100, cv=3, n_jobs=-1, verbose=2, random_state=42)

# Fit the random search to the data
random_search.fit(X_train, y_train)

# Print the best parameters found by RandomizedSearchCV
print("Best parameters found: ", random_search.best_params_)

# Train a RandomForest classifier with the best parameters
best_clf = random_search.best_estimator_
best_clf.fit(X_train, y_train)

# Make predictions
y_pred = best_clf.predict(X_test)

# Evaluate the model
print("\nClassification Report:")
report = classification_report(y_test, y_pred, digits=6)
print(report)

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)
```

```
Classification Report:
              precision    recall  f1-score   support

           0   1.000000  0.991639  0.995802      1196
           1   0.991837  1.000000  0.995902      1215

    accuracy                       0.995852      2411
   macro avg   0.995918  0.995819  0.995852      2411
weighted avg   0.995886  0.995852  0.995852      2411


Confusion Matrix:
[[1186   10]
 [   0 1215]]

Exact Accuracy: 0.9958523434
Exact Precision: 0.9918367347
Exact Recall: 1.0000000000
Exact F1 Score: 0.9959016393
```





MODEL DEVELOPMENT & EVALUATION

MULTI-LAYER PERCEPTRON (MLP)

```python
# Initialize and train MLPClassifier (Neural Network)
from sklearn.neural_network import MLPClassifier
mlp_clf = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=100, random_state=42)
mlp_clf.fit(X_train, y_train)

# Make predictions
mlp_y_pred = mlp_clf.predict(X_test)

# Evaluate MLPClassifier
print("\nMLPClassifier - Classification Report:")
print(classification_report(y_test, mlp_y_pred, digits=6))

print("\nMLPClassifier - Confusion Matrix:")
conf_martix1=confusion_matrix(y_test, mlp_y_pred)
print(conf_martix1)

# Exact Calculation of Metrics
TP = conf_martix1[1, 1]
TN = conf_martix1[0, 0]
FP = conf_martix1[0, 1]
FN = conf_martix1[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)

print(f"\nExact Accuracy: {accuracy:.10f}")
print(f"Exact Precision: {precision:.10f}")
print(f"Exact Recall: {recall:.10f}")
print(f"Exact F1 Score: {f1_score:.10f}")
```

```
MLPClassifier - Classification Report:
              precision    recall  f1-score   support

           0   0.980000  0.983278  0.981636      1196
           1   0.983485  0.980247  0.981863      1215

    accuracy                       0.981750      2411
   macro avg   0.981742  0.981762  0.981750      2411
weighted avg   0.981756  0.981750  0.981750      2411


MLPClassifier - Confusion Matrix:
[[1176   20]
 [  24 1191]]

Exact Accuracy: 0.9817503111
Exact Precision: 0.9834847234
Exact Recall: 0.9802469136
Exact F1 Score: 0.9818631492
```

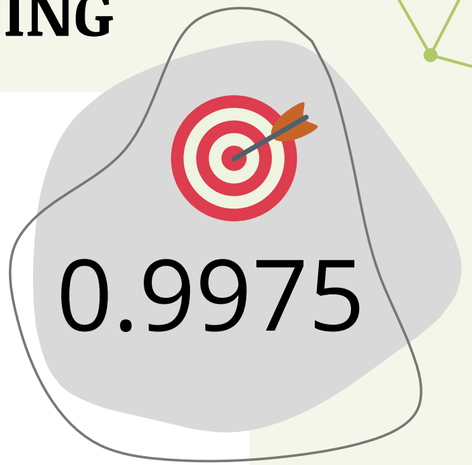
0.9817

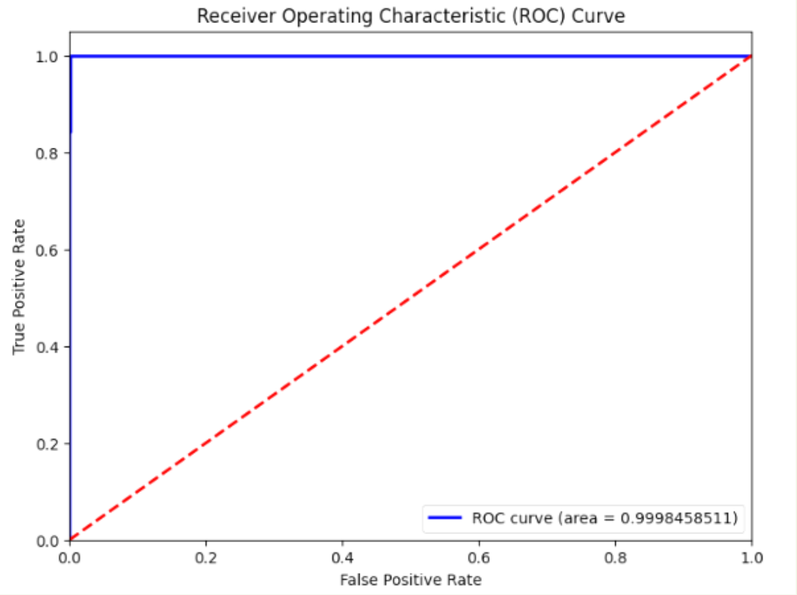
MODEL DEVELOPMENT & EVALUATION

Multi-Layer Perceptron (MLP)-HYPERTUNING

```python
# Define the parameter grid
param_distributions = {
    'hidden_layer_sizes': [(50,), (100,), (100, 50), (100, 100, 50)],
    'activation': ['tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'learning_rate': ['constant', 'adaptive'],
    'max_iter': [200, 300, 400, 500]
}

# Initialize the MLPClassifier
mlp_clf = MLPClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=mlp_clf, param_distributions=param_distributions
                                   n_iter=50, cv=3, n_jobs=-1, verbose=2, random_state=42)

# Fit the random search to the data
random_search.fit(X_train, y_train)

# Print the best parameters found by RandomizedSearchCV
print("Best parameters found: ", random_search.best_params_)

# Train a MLPClassifier with the best parameters
best_mlp_clf = random_search.best_estimator_
best_mlp_clf.fit(X_train, y_train)

# Make predictions
mlp_y_pred = best_mlp_clf.predict(X_test)

# Evaluate the model
print("\nMLPClassifier - Classification Report:")
report = classification_report(y_test, mlp_y_pred, digits=6)
print(report)

print("\nMLPClassifier - Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, mlp_y_pred)
print(conf_matrix)
```

```
MLPClassifier - Classification Report:
              precision    recall  f1-score   support

           0   1.000000  0.994983  0.997485      1196
           1   0.995086  1.000000  0.997537      1215

    accuracy                       0.997511      2411
   macro avg   0.997543  0.997492  0.997511      2411
weighted avg   0.997524  0.997511  0.997511      2411


MLPClassifier - Confusion Matrix:
[[1190    6]
 [   0 1215]]

Exact Accuracy: 0.9975114061
Exact Precision: 0.9950859951
Exact Recall: 1.0000000000
Exact F1 Score: 0.9975369458
```





MODEL DEVELOPMENT & EVALUATION

GRADIENT BOOSTING CLASSIFIER

```python
# Initialize and train GradientBoostingClassifier
gb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb_clf.fit(X_train, y_train)

# Make predictions
gb_y_pred = gb_clf.predict(X_test)

# Evaluate GradientBoostingClassifier
print("\nGradientBoostingClassifier - Classification Report:")
print(classification_report(y_test, gb_y_pred, digits=6))

print("\nGradientBoostingClassifier - Confusion Matrix:")
conf_martix2=confusion_matrix(y_test, gb_y_pred)
print(conf_martix2)

# Exact Calculation of Metrics
TP = conf_martix2[1, 1]
TN = conf_martix2[0, 0]
FP = conf_martix2[0, 1]
FN = conf_martix2[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)

print(f"\nExact Accuracy: {accuracy:.10f}")
print(f"Exact Precision: {precision:.10f}")
print(f"Exact Recall: {recall:.10f}")
print(f"Exact F1 Score: {f1_score:.10f}")
```

```
GradientBoostingClassifier - Classification Report:
              precision    recall  f1-score   support

           0   1.000000  0.988294  0.994113      1196
           1   0.988609  1.000000  0.994272      1215

    accuracy                       0.994193      2411
   macro avg   0.994304  0.994147  0.994192      2411
weighted avg   0.994259  0.994193  0.994193      2411


GradientBoostingClassifier - Confusion Matrix:
[[1182   14]
 [   0 1215]]

Exact Accuracy: 0.9941932808
Exact Precision: 0.9886086249
Exact Recall: 1.0000000000
Exact F1 Score: 0.9942716858
```

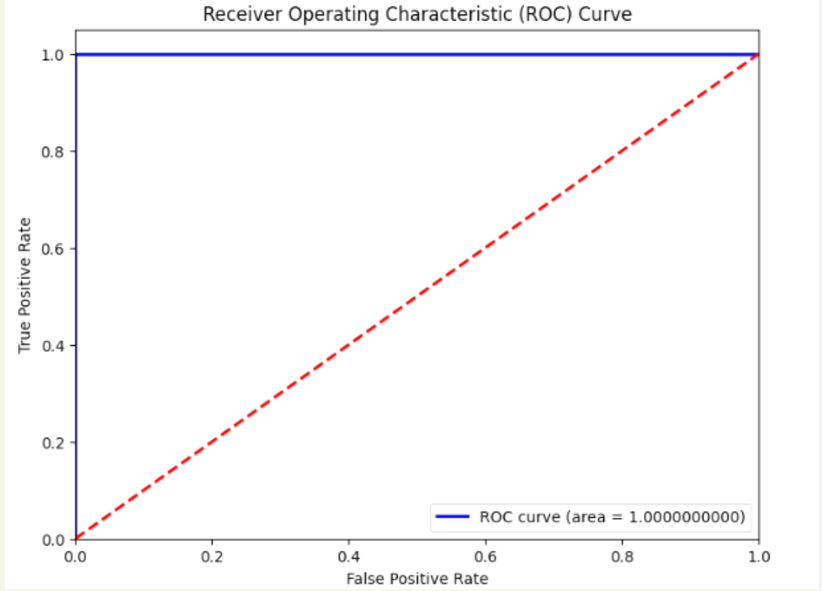
0.9941

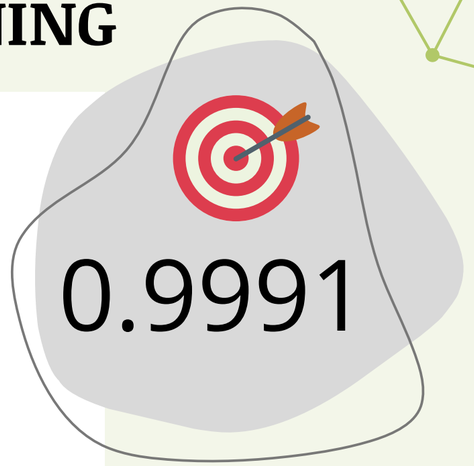
MODEL DEVELOPMENT & EVALUATION

GRADIENT BOOSTING CLASSIFIER- HYPERTUNING

```python
param_distributions = {
    'n_estimators': [100, 200, 300, 400, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3],
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.8, 0.9, 1.0],
    'max_features': ['auto', 'sqrt', 'log2', None]
}

# Initialize the GradientBoostingClassifier
gb_clf = GradientBoostingClassifier(random_state=42)

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=gb_clf, param_distributions=param_distributions,
                                   n_iter=100, cv=3, n_jobs=-1, verbose=2, random_state=42)

# Fit the random search to the data
random_search.fit(X_train, y_train)

# Print the best parameters found by RandomizedSearchCV
print("Best parameters found: ", random_search.best_params_)

# Train a GradientBoostingClassifier with the best parameters
best_gb_clf = random_search.best_estimator_
best_gb_clf.fit(X_train, y_train)

# Make predictions
gb_y_pred = best_gb_clf.predict(X_test)

# Evaluate the model
print("\nGradientBoostingClassifier - Classification Report:")
report = classification_report(y_test, gb_y_pred, digits=6)
print(report)

print("\nGradientBoostingClassifier - Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, gb_y_pred)
print(conf_matrix)

# Exact Calculation of Metrics
TP = conf_matrix[1, 1]
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]
FN = conf_matrix[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)
```

```
GradientBoostingClassifier - Classification Report:
              precision    recall  f1-score   support

           0   1.000000  0.998328  0.999163      1196
           1   0.998357  1.000000  0.999178      1215

    accuracy                       0.999170      2411
   macro avg   0.999178  0.999164  0.999170      2411
weighted avg   0.999172  0.999170  0.999170      2411


GradientBoostingClassifier - Confusion Matrix:
[[1194    2]
 [   0 1215]]

Exact Accuracy: 0.9991704687
Exact Precision: 0.9983566146
Exact Recall: 1.0000000000
Exact F1 Score: 0.9991776316
```





MODEL DEVELOPMENT & EVALUATION

SUPPORT VECTOR CLASSIFIER

```python
# Initialize and train SVC (Support Vector Classifier)
svm_clf = SVC(kernel='rbf', probability=True, random_state=42)
svm_clf.fit(X_train, y_train)

# Make predictions
svm_y_pred = svm_clf.predict(X_test)

# Evaluate SVC
print("\nSVC - Classification Report:")
print(classification_report(y_test, svm_y_pred, digits=6))

print("\nSVC - Confusion Matrix:")
conf_martix3 = confusion_matrix(y_test, svm_y_pred)
print(conf_martix3)

# Exact Calculation of Metrics
TP = conf_martix3[1, 1]
TN = conf_martix3[0, 0]
FP = conf_martix3[0, 1]
FN = conf_martix3[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)

print(f"\nExact Accuracy: {accuracy:.10f}")
print(f"Exact Precision: {precision:.10f}")
print(f"Exact Recall: {recall:.10f}")
print(f"Exact F1 Score: {f1_score:.10f}")
```

```
SVC - Classification Report:
              precision    recall  f1-score   support

           0   0.994686  0.938963  0.966022      1196
           1   0.943058  0.995062  0.968362      1215

    accuracy                       0.967234      2411
   macro avg   0.968872  0.967012  0.967192      2411
weighted avg   0.968668  0.967234  0.967201      2411


SVC - Confusion Matrix:
[[1123   73]
 [   6 1209]]

Exact Accuracy: 0.9672335131
Exact Precision: 0.9430577223
Exact Recall: 0.9950617284
Exact F1 Score: 0.9683620344
```

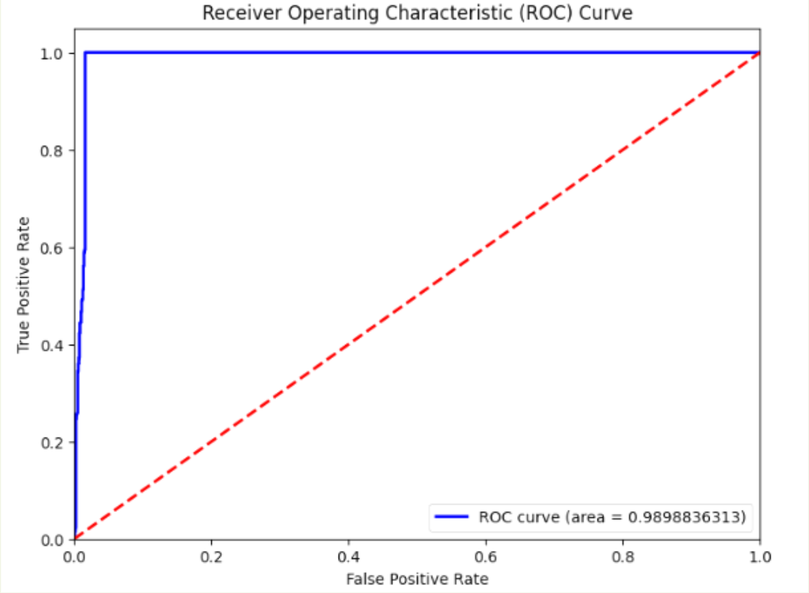
0.9672

MODEL DEVELOPMENT & EVALUATION

SUPPORT VECTOR CLASSIFIER- HYPERTUNING

```python
param_grid = {
    'C': [0.1, 1, 10, 100],
    'gamma': [1, 0.1, 0.01, 0.001],
    'kernel': ['rbf']
}

# Initialize the SVC
svm_clf = SVC(probability=True, random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=svm_clf, param_grid=param_grid,
                           cv=3, n_jobs=-1, verbose=2)

# Fit GridSearchCV to the data
grid_search.fit(X_train, y_train)

# Print the best parameters found by GridSearchCV
print("Best parameters found: ", grid_search.best_params_)

# Train a SVC with the best parameters
best_svm_clf = grid_search.best_estimator_
best_svm_clf.fit(X_train, y_train)

# Make predictions
svm_y_pred = best_svm_clf.predict(X_test)

# Evaluate the model
print("\nSVC - Classification Report:")
report = classification_report(y_test, svm_y_pred, digits=6)
print(report)

print("\nSVC - Confusion Matrix:")
conf_matrix = confusion_matrix(y_test, svm_y_pred)
print(conf_matrix)

# Exact Calculation of Metrics
TP = conf_matrix[1, 1]
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]
FN = conf_matrix[1, 0]

accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)
```

```
SVC - Classification Report:
              precision    recall  f1-score   support

           0   1.000000  0.976589  0.988156      1196
           1   0.977474  1.000000  0.988609      1215

    accuracy                       0.988387      2411
   macro avg   0.988737  0.988294  0.988382      2411
weighted avg   0.988648  0.988387  0.988384      2411


SVC - Confusion Matrix:
[[1168   28]
 [   0 1215]]

Exact Accuracy: 0.9883865616
Exact Precision: 0.9774738536
Exact Recall: 1.0000000000
Exact F1 Score: 0.9886086249
```



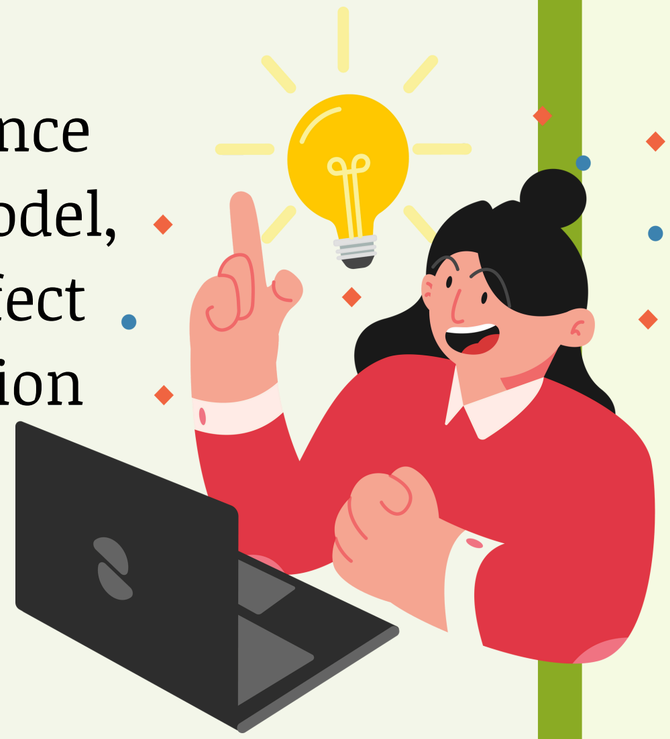
0.9883

MODEL SELECTION

SELECTED MODEL: GRADIENT BOOSTING CLASSIFIER

REASON:

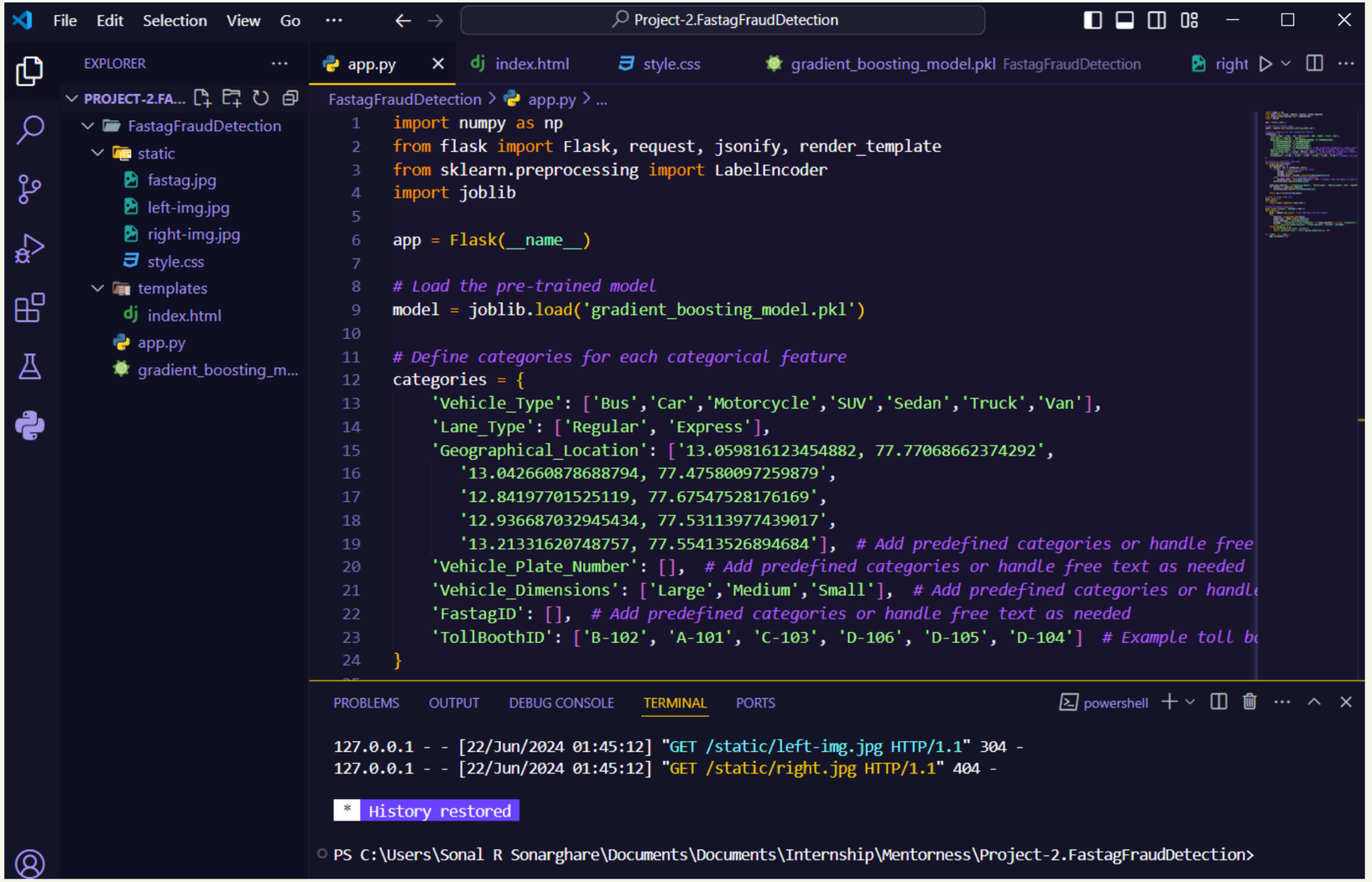
This model was selected based on its superior performance across the evaluation metrics. The Gradient Boosting model, fine-tuned with Randomized Search, achieved near-perfect performance with accuracy(0.9991). Ensures high precision and recall, critical for minimizing false negatives and maintaining transaction integrity.



- **Exact Metrics:**
  - Accuracy: 0.9991704687
  - Precision: 0.9983566146
  - Recall: 1.0000000000
  - F1 Score: 0.9991776316
- **ROC Curve:**
  - AUC: 1.0000
  - Demonstrates excellent discrimination between fraudulent and non-fraudulent transactions

REAL TIME FRAUD DETECTION

DEPLOYMENT ON FLASK



REAL TIME FRAUD DETECTION

Fraud Detection

Vehicle Type: Bus

Lane Type: Regular

Geographical Location: 13.059816123454882, 77.77068662374292

Vehicle Plate Number: KA11GH3456

Vehicle Dimensions: Large

Fastag ID: FTG-707-JKL-210

TollBooth ID: D-105

Transaction Amount: 500

Amount Paid: 200

Vehicle Speed: 56

Hour: 5

DayOfWeek: 7

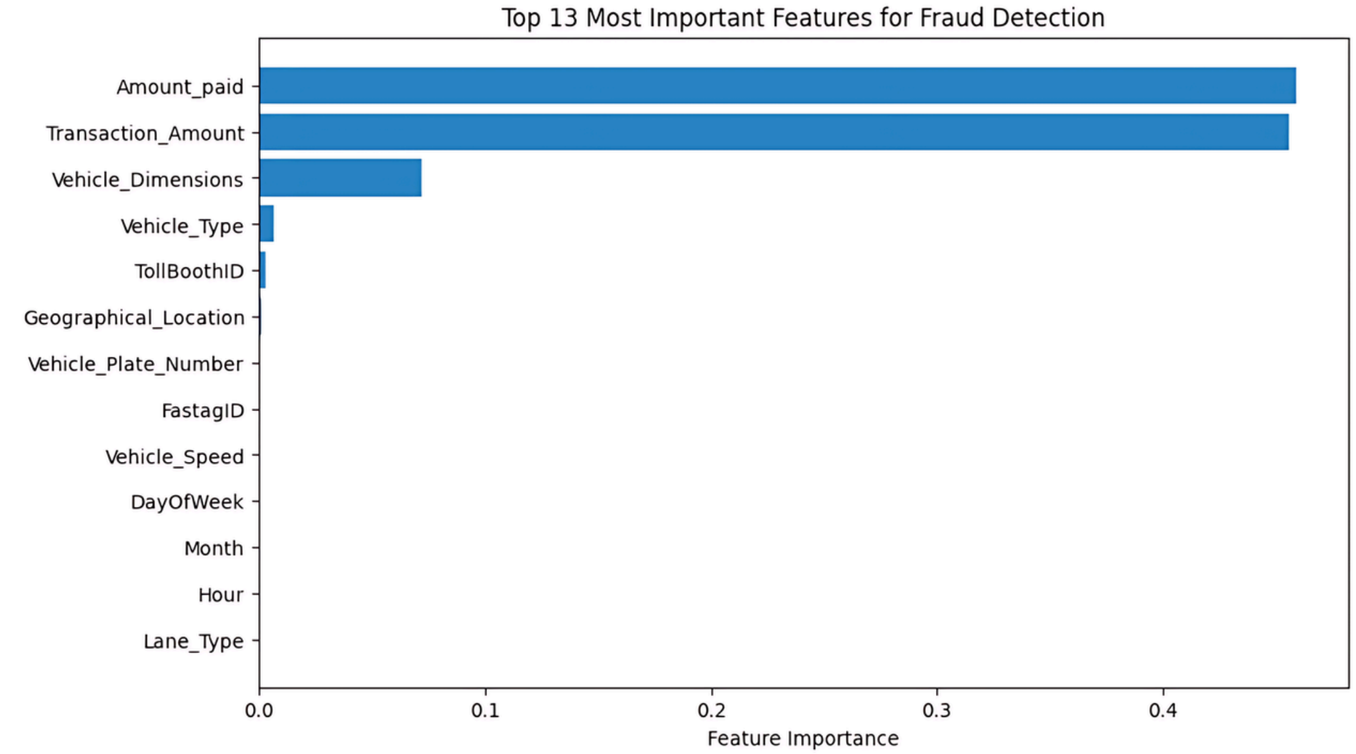
Month: 3

Submit

Fraud Indicator: Fraud
Transaction is fraudulent

EXPLANTORY ANALYSIS

```
Amount_paid: 0.4593686619
Transaction_Amount: 0.4562961305
Vehicle_Dimensions: 0.0719769523
Vehicle_Type: 0.0069367856
TollBoothID: 0.0030344974
Geographical_Location: 0.0012082614
Vehicle_Plate_Number: 0.0005814439
FastagID: 0.0004597983
Vehicle_Speed: 0.0000912652
DayOfWeek: 0.0000175068
Month: 0.0000175064
Hour: 0.0000111883
Lane_Type: 0.0000000021
```



The chart shows that the most critical features for fraud detection are:

Amount_paid (0.459)
Transaction_Amount (0.456)
Vehicle_Dimensions (0.072)

These three features have the highest importance scores, indicating they are the most influential in identifying fraud. Other features, such as Vehicle_Type, TollBoothID, and Geographical_Location, have minimal importance, while features like Vehicle_Speed, DayOfWeek, Month, Hour, and Lane_Type contribute very little to fraud detection.

CONCLUSION

Model Selected:

Gradient Boosting Classifier

Performance:

Achieved near-perfect accuracy (99.92%) in detecting fraudulent FASTag transactions.
High precision (99.84%) and recall (100%) rates ensure minimal false positives and negatives.

Why Gradient Boosting:

- Superior in handling complex patterns and imbalanced data.
- Effective feature importance analysis.
- Proven success in similar financial fraud detection applications.

Impact:

- Enhanced security and integrity of FASTag transactions.
- Significant reduction in potential revenue losses due to fraud.

Future Work:

- Continuous model monitoring and updates with new data.
- Explore additional features and advanced ensemble techniques for further improvements.

THANK YOU

SONAL SONARGHARE
sonalsonarghare30@gmail.com
A.P SHAH INSTITUTE OF TECHNOLOGY