Studio
Shodwe

FASTAG FRAUD DETECTION SYSTEM

Presented by: Harshit Ghosh

Introduction

- FastTag is an electronic toll collection system in India.
- It uses Radio Frequency Identification (RFID) technology for automatic toll payments.
- With the increasing adoption of FastTags, fraudulent activities have also surged.
- This presentation outlines our approach to detecting and preventing FastTag fraud.

Introduction

Fastag, a convenient RFID-based toll payment method, is unfortunately susceptible to fraud. Malicious actors can misuse Fastags in several ways, such as:

- Using a Fastag meant for a lighter vehicle category on a heavier one, paying a lower toll.
- Tampering with RFID tags or using stolen ones to avoid paying tolls altogether.

These fraudulent activities lead to revenue loss for toll authorities and erode trust in the Fastag system.

Goals

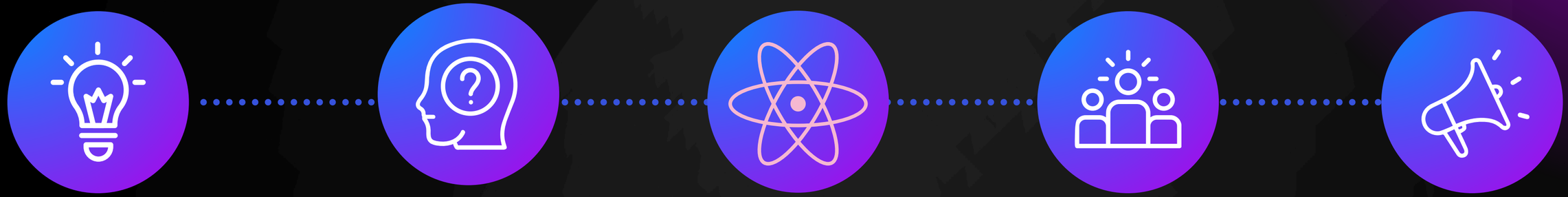
Objective 01

- To develop a robust system for detecting fraudulent activities in FastTag transactions.
- Ensure the security and reliability of the FastTag system for users and operators.

Objective 02

- Build a machine learning model to identify fraudulent transactions.
- Integrate the model into the FastTag system for real-time fraud detection.
- Minimize false positives and maximize the accuracy of fraud detection.

Project Timeline



01

Exploratory Data Analysis (EDA)

Uncover Patterns

02

Feature Engineering

Craft Useful Data

03

Model Building

Train Fraud Detection Model

04

Model Evaluation

Evaluate Model Performance

05

Deployment

Deploy for Real-Time Use

Exploratory Data Analysis (EDA)

- Understanding the dataset: transaction records, user details, toll plaza information, etc.
- Identifying patterns and anomalies in the data.
- Visualizing data to gain insights into common fraud characteristics.
- Example: Plotting the frequency of transactions by time and location.

Getting Insights

```
df.info()
```
Executed at 2024.06.14 16:55:12 in 9ms

```
<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.nunique()
```
Executed at 2024.06.14 16:55:12 in 12ms

```
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
```

```
df.dtypes
```
Executed at 2024.06.14 16:55:12 in 5ms

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

```
df.head()
```
Executed at 2024.06.14 16:55:12 in 10ms

5 rows | 5 rows × 13 columns | Static Output

	Transaction_ID	Timestamp	Vehicle_Type	FastagID	TollBoothID	Lane_Type	Vehicle_Dimensions	Transaction_Amount	Amount_paid	Geographical_Location
0	1	1/6/2023 11:20	Bus	FTG-001-ABC-121	A-101	Express	Large	350	120	13.059816123454882, 77.7
1	2	1/7/2023 14:55	Car	FTG-002-XYZ-451	B-102	Regular	Small	120	100	13.059816123454882, 77.7
2	3	1/8/2023 18:25	Motorcycle	NaN	D-104	Regular	Small	0	0	13.059816123454882, 77.7
3	4	1/9/2023 2:05	Truck	FTG-044-LMN-322	C-103	Regular	Large	350	120	13.059816123454882, 77.7
4	5	1/10/2023 6:35	Van	FTG-505-DEF-652	B-102	Express	Medium	140	100	13.059816123454882, 77.7

Getting Insights

```
df.describe()
```

Executed at 2024.06.14 16:55:12 in 12ms

8 rows × 4 columns

	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

```
# Check for missing values
print(df.isnull().sum())
```

Executed at 2024.06.14 16:55:12 in 9ms

```
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
```

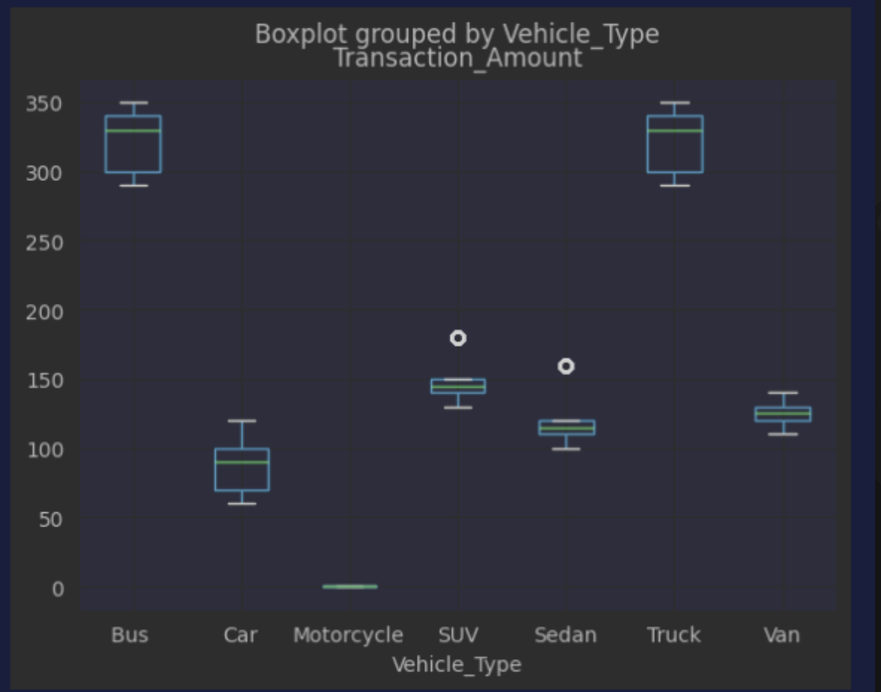
```
# Drop duplicates and missing values
df = df.drop_duplicates().dropna()
```

Executed at 2024.06.14 16:55:12 in 20ms

Getting Insights

```
df.boxplot(by="Vehicle_Type", column="Transaction_Amount")
plt.show()
```

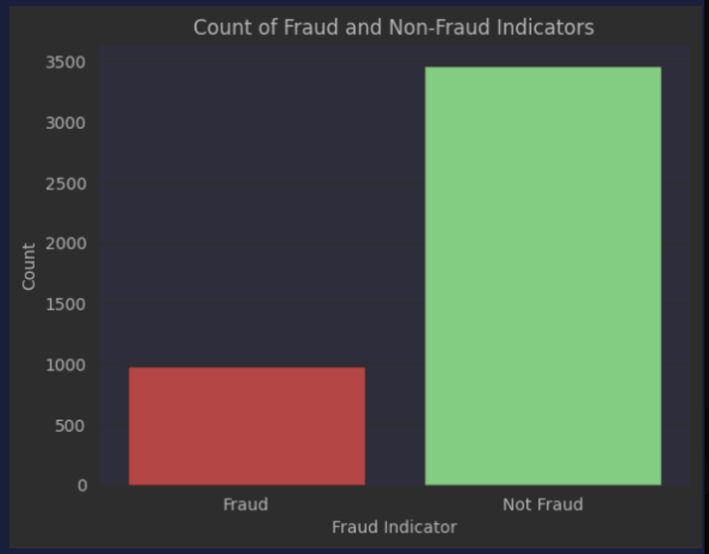
Executed at 2024.06.14 16:55:13 in 196ms




```
sns.countplot(x='Fraud_indicator', data=df, palette=['red', 'green'])
plt.xlabel('Fraud Indicator')
plt.ylabel('Count')
plt.title('Count of Fraud and Non-Fraud Indicators')
plt.show()
```

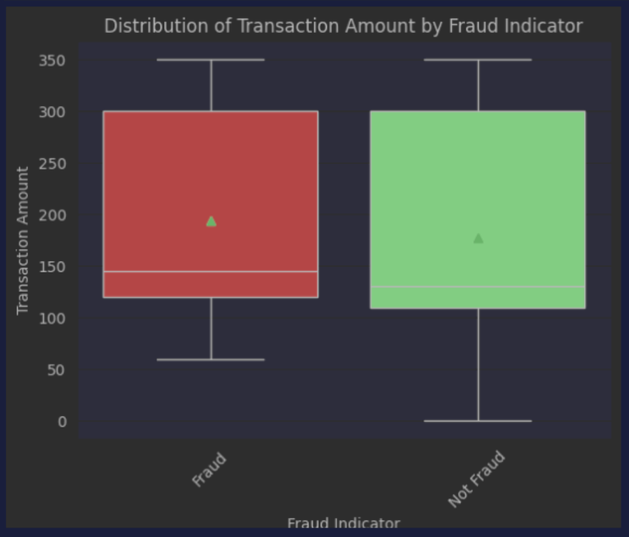
Executed at 2024.06.14 16:55:15 in 138ms

```
/tmp/ipykernel_141731/2535894053.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0

  sns.countplot(x='Fraud_indicator', data=df, palette=['red', 'green'])
```




```
sns.boxplot(
    x = "Fraud_indicator",
    y = "Transaction_Amount",
    showmeans=True,
    data=df,
    palette=["red", "green"]
)

plt.xlabel("Fraud Indicator")
plt.ylabel("Transaction Amount")
plt.title("Distribution of Transaction Amount by Fraud Indicator")
plt.xticks(rotation=45)
plt.show()
```

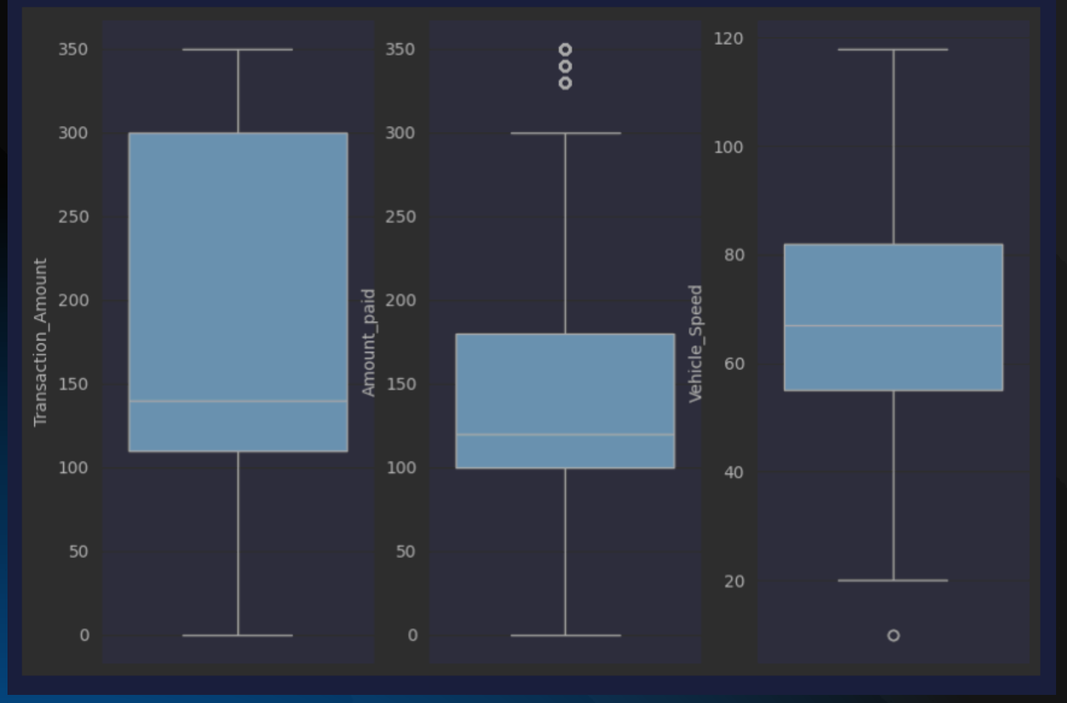
```
/tmp/ipykernel_141731/1729941162.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to

  sns.boxplot(
```



Getting Insights

```
plt.figure(figsize=(10, 7))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(1, 3, i)
    sns.boxplot(y=feature, data=df)
plt.show()
```

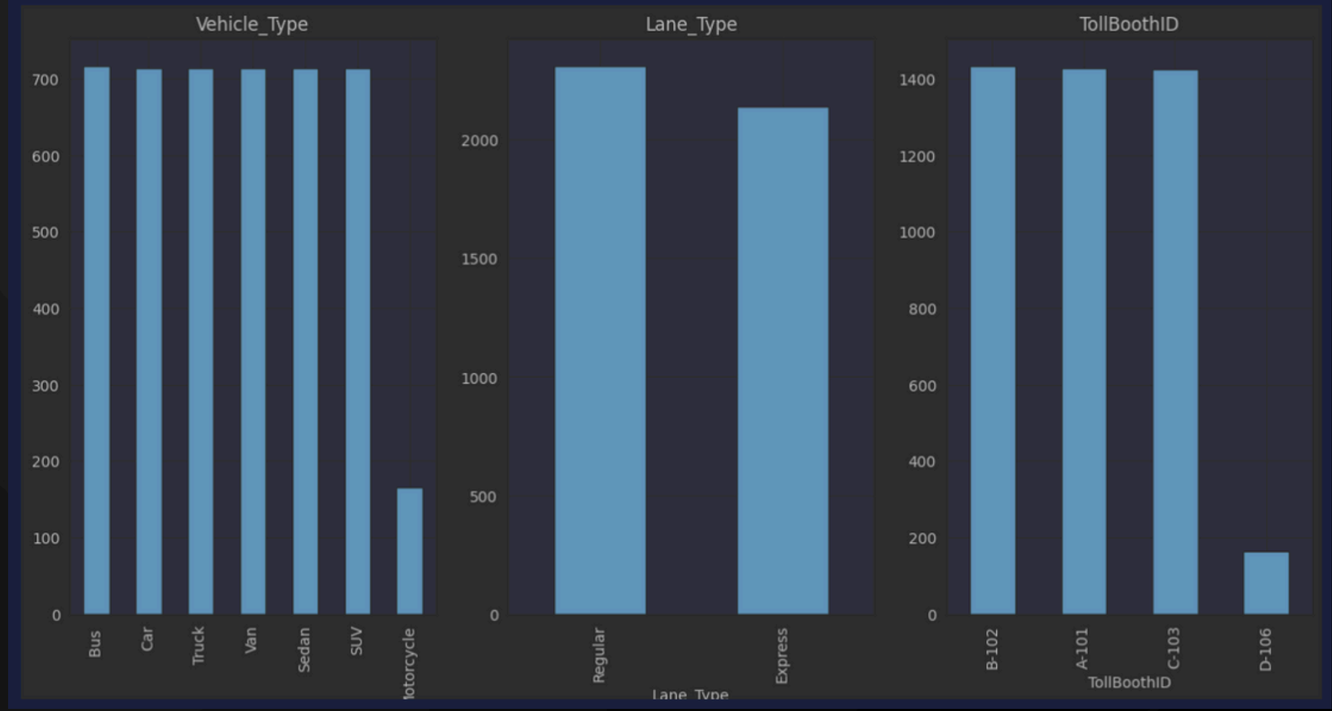
Executed at 2024.06.14 16:55:14 in 301ms



box for numeric

```
categorical_features = ['Vehicle_Type', 'Lane_Type', 'TollBoothID']
plt.figure(figsize=(15, 7))
for i, feature in enumerate(categorical_features, 1):
    plt.subplot(1, 3, i)
    df[feature].value_counts().plot(kind='bar')
    plt.title(feature)
plt.show()
```

Executed at 2024.06.14 16:55:15 in 477ms



box for cateogary

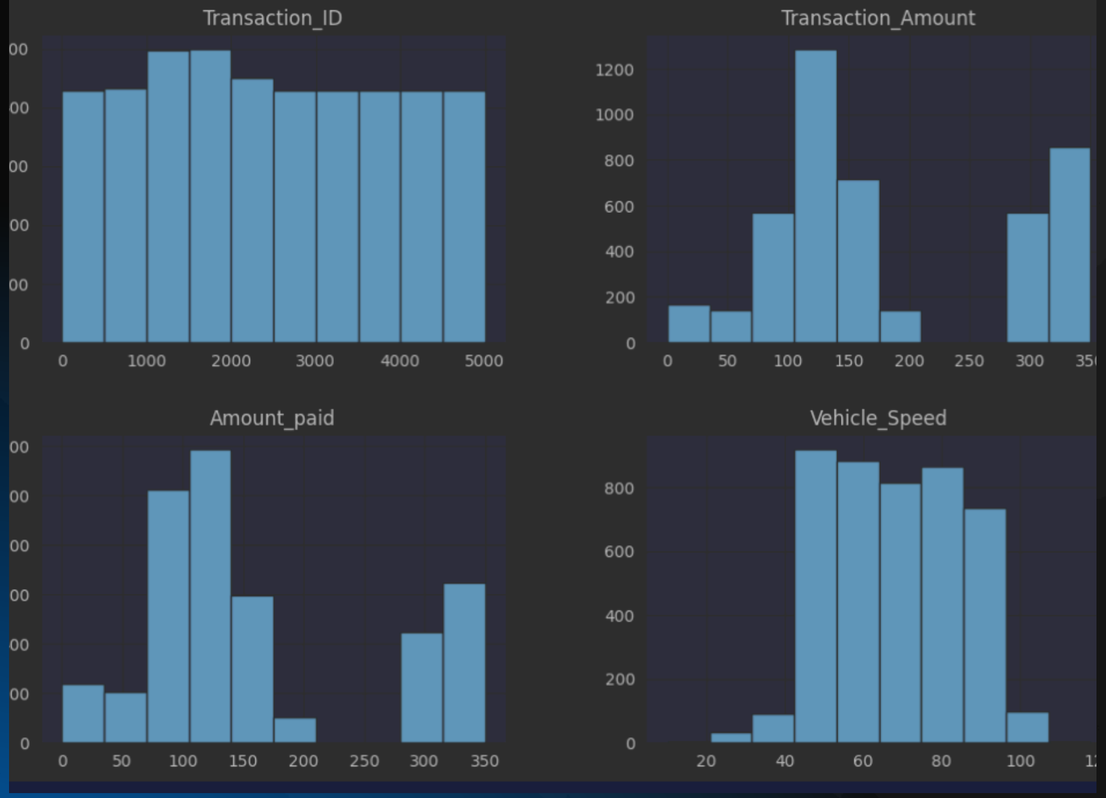
Getting Insights

```
# Correlation matrix
plt.figure(figsize=(12, 10))
corr_matrix = df.select_dtypes(include=['number']).corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```

Getting Insights

Feature Distributions

```
alyze feature distributions (histograms)
ist(figsize=(12, 8))
show()
ted at 2024.06.14 16:55:13 in 762ms
```



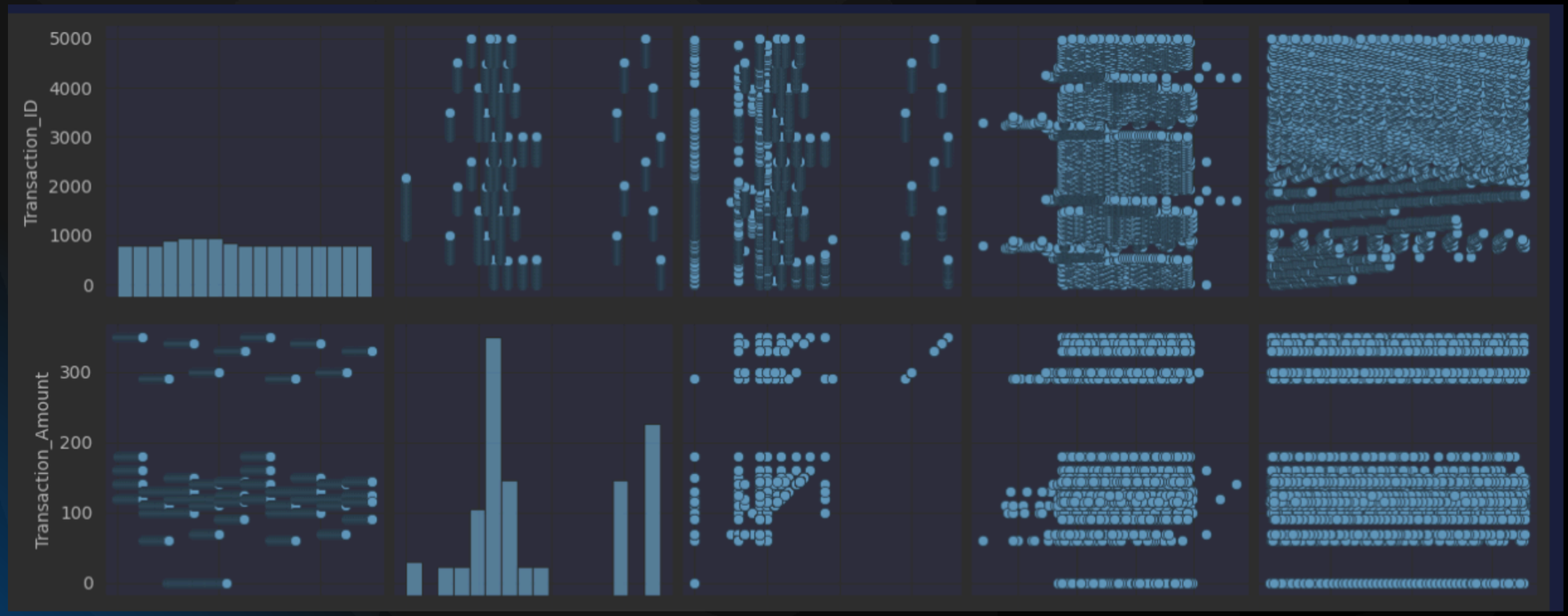

```
numerical_features = ['Transaction_Amount', 'Amount_paid', 'Vehicle_Speed']
df[numerical_features].hist(bins=30, figsize=(10, 7))
plt.show()
Executed at 2024.06.14 16:55:14 in 992ms
```

Getting Insights

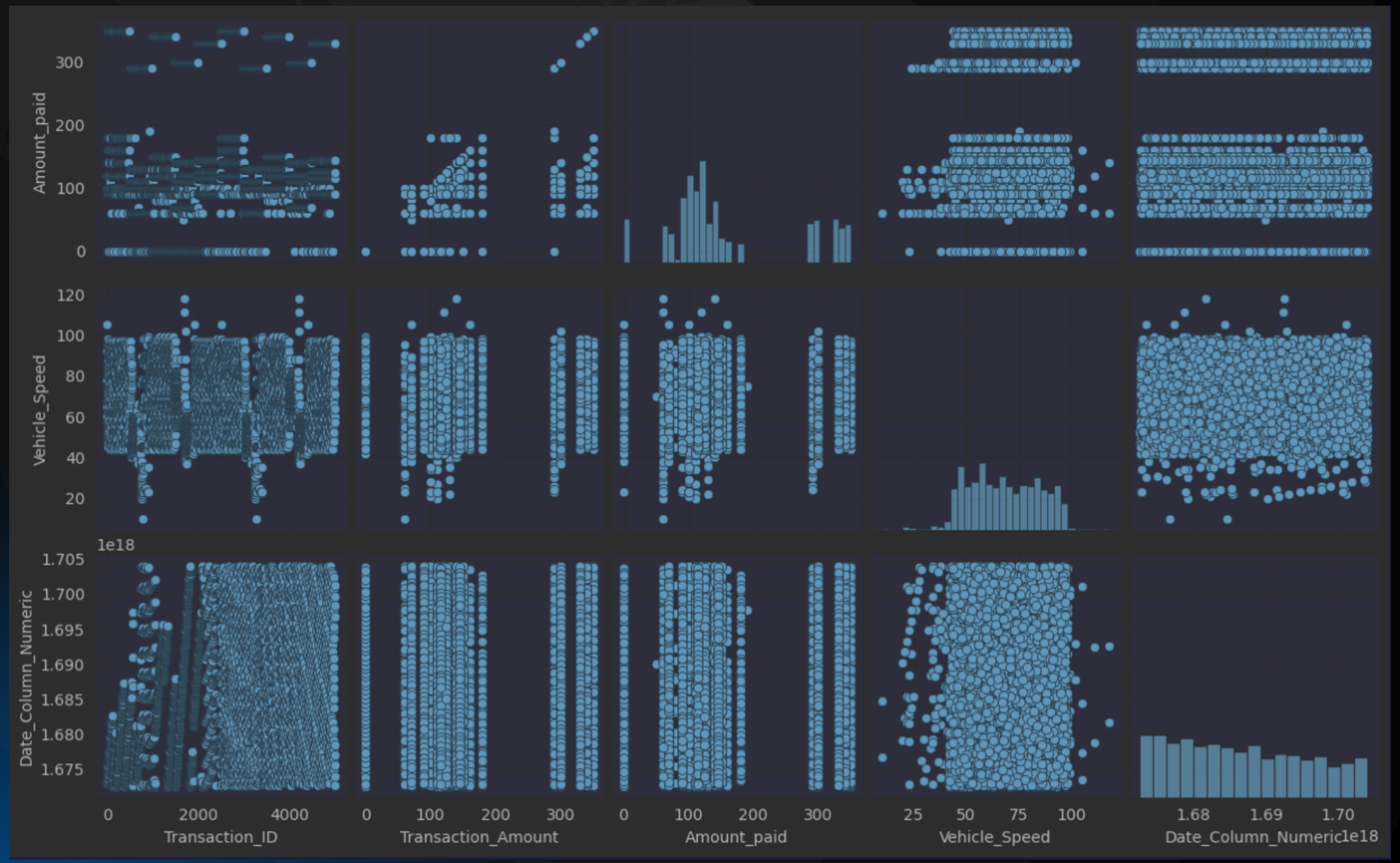
```
print("\nUnique values in categorical columns:")
for col in df.select_dtypes(include='object').columns:
    print(f"{col}: {df[col].unique()}")
sns.pairplot(df)
plt.show()
```

Executed at 2024.06.14 16:55:21 in 5s 14ms

Getting Unique Values



Getting Insights



Feature Engineering

Creating new features to enhance model performance.

- Examples of features
- Transaction time and frequency.
- Amount patterns.

Importance of selecting relevant and impactful features.

Data Conversion

```
# Convert date/time columns to numeric representation before calculating correlations
df['Timestamp'] = pd.to_datetime(df['Timestamp']) # Convert to datetime
df['Date_Column_Numeric'] = df['Timestamp'].astype(int)  # Convert to numeric timestamp

# Select only numeric columns for correlation calculation
numeric_df = df.select_dtypes(include=['number'])
correlation_matrix = numeric_df.corr(method="spearman")  # Now calculate correlations

print(correlation_matrix)
```

Executed at 2024.06.14 16:55:13 in 38ms

	Transaction_ID	Transaction_Amount	Amount_paid	\
Transaction_ID	1.000000	-0.024137	0.087752	
Transaction_Amount	-0.024137	1.000000	0.792911	
Amount_paid	0.087752	0.792911	1.000000	
Vehicle_Speed	0.010776	0.098390	0.075090	
Date_Column_Numeric	0.257582	-0.049760	0.009707	

	Vehicle_Speed	Date_Column_Numeric
Transaction_ID	0.010776	0.257582
Transaction_Amount	0.098390	-0.049760
Amount_paid	0.075090	0.009707
Vehicle_Speed	1.000000	0.016562
Date_Column_Numeric	0.016562	1.000000

Visualize categorical feature relationships (boxplots)

Time Extraction

```
# Extract time-based features
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df['Month'] = df['Timestamp'].dt.month
df['Week'] = df['Timestamp'].dt.isocalendar().week
```

```python
# Drop unnecessary columns
df = df.drop(columns=['Transaction_ID', 'FastagID', 'Vehicle_Plate_Number', 'Timestamp'])
```

Executed at 2024.06.14 16:55:21 in 10ms

```python
# Define feature columns
features = ['Transaction_Amount', 'Amount_paid', 'Vehicle_Type', 'TollBoothID',
            'Lane_Type', 'Vehicle_Dimensions', 'Geographical_Location',
            'Month', 'Week']
```

Executed at 2024.06.14 16:55:21 in 4ms

```python
# Encode target variable
le = LabelEncoder()
df['Fraud_indicator'] = le.fit_transform(df['Fraud_indicator'])
```

X,Y Define and Train and Split

```
# Define X and y
X = df[features]
y = df['Fraud_indicator']
```

Executed at 2024.06.14 16:55:21 in 3ms

```
# Define ColumnTransformer
column_trans = ColumnTransformer(transformers=[
    ('onehot', OneHotEncoder(drop='first'), ['Vehicle_Type', 'TollBoothID', 'Lane_Type', 'Geographical_Location']),
    ('ordinal', OrdinalEncoder(), ['Vehicle_Dimensions'])
], remainder='passthrough')
```

Executed at 2024.06.14 16:55:21 in 3ms

```
# Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=45)
```

```
# Transform the features
X_train_transformed = column_trans.fit_transform(X_train)
X_test_transformed = column_trans.transform(X_test)
```

Executed at 2024.06.14 16:55:21 in 23ms

Model Evaluation

Analysis

- Evaluating the model using metrics such as accuracy, precision, recall, and F1-score.
- Conducting cross-validation to ensure model robustness.
- Analyzing the confusion matrix to understand false positives and false negatives.

Model Declaraion and Hyperparameter

```python
models = {
    'DecisionTreeClassifier': {
        'model': DecisionTreeClassifier(),
        'params': {'max_depth': [1, 3, 5, 7], 'criterion': ['gini', 'entropy']}
    },
    'RandomForestClassifier': {
        'model': RandomForestClassifier(),
        'params': {'max_depth': [2, 3, 5, 7], 'criterion': ['gini', 'entropy'], 'n_estimators': [100, 200, 300]}
    },
    'AdaBoostClassifier': {
        'model': AdaBoostClassifier(),
        'params': {'n_estimators': [100, 200, 300], 'learning_rate': [0.1, 0.01, 0.001]}
    },
    'XGBClassifier': {
        'model': XGBClassifier(),
        'params': {'max_depth': [2, 3, 5, 7], 'n_estimators': [100, 200, 300], 'learning_rate': [0.01, 0.1, 0.001]}
    },
    'KNeighborsClassifier': {
        'model': KNeighborsClassifier(),
        'params': {'n_neighbors': [2, 3, 5, 7]}
    },
    'SVM': {
        'model': svm.SVC(probability=True),
        'params': {'C': [5, 10, 20, 30], 'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
    }
}
```

Executed at 2024.06.14 16:55:21 in 5ms

Evaluation of Best Model

```python
# Evaluate each model using GridSearchCV
best_models = {}
for model_name in models:
    print(f"Training {model_name}...")
    grid_search = GridSearchCV(models[model_name]['model'], models[model_name]['params'], cv=5, n_jobs=-1, scoring='f1_weighted')
    grid_search.fit(X_train_transformed, y_train)
    best_models[model_name] = grid_search.best_estimator_
    print(f"Best Parameters for {model_name}: {grid_search.best_params_}")
    print(f"Best Cross-validation score for {model_name}: {grid_search.best_score_}")
```

```
Training DecisionTreeClassifier...
Best Parameters for DecisionTreeClassifier: {'criterion': 'gini', 'max_depth': 7}
Best Cross-validation score for DecisionTreeClassifier: 0.9925669264978341
Training RandomForestClassifier...
Best Parameters for RandomForestClassifier: {'criterion': 'entropy', 'max_depth': 7, 'n_estimators': 200}
Best Cross-validation score for RandomForestClassifier: 0.9590740385469838
Training AdaBoostClassifier...
Best Parameters for AdaBoostClassifier: {'learning_rate': 0.1, 'n_estimators': 300}
Best Cross-validation score for AdaBoostClassifier: 0.9639573170890389
Training XGBClassifier...
Best Parameters for XGBClassifier: {'learning_rate': 0.1, 'max_depth': 5, 'n_estimators': 300}
Best Cross-validation score for XGBClassifier: 0.9967805592764452
Training KNeighborsClassifier...
Best Parameters for KNeighborsClassifier: {'n_neighbors': 2}
Best Cross-validation score for KNeighborsClassifier: 0.9974242737273201
Training SVM...
Best Parameters for SVM: {'C': 20, 'kernel': 'linear'}
Best Cross-validation score for SVM: 0.9932167077153986
```

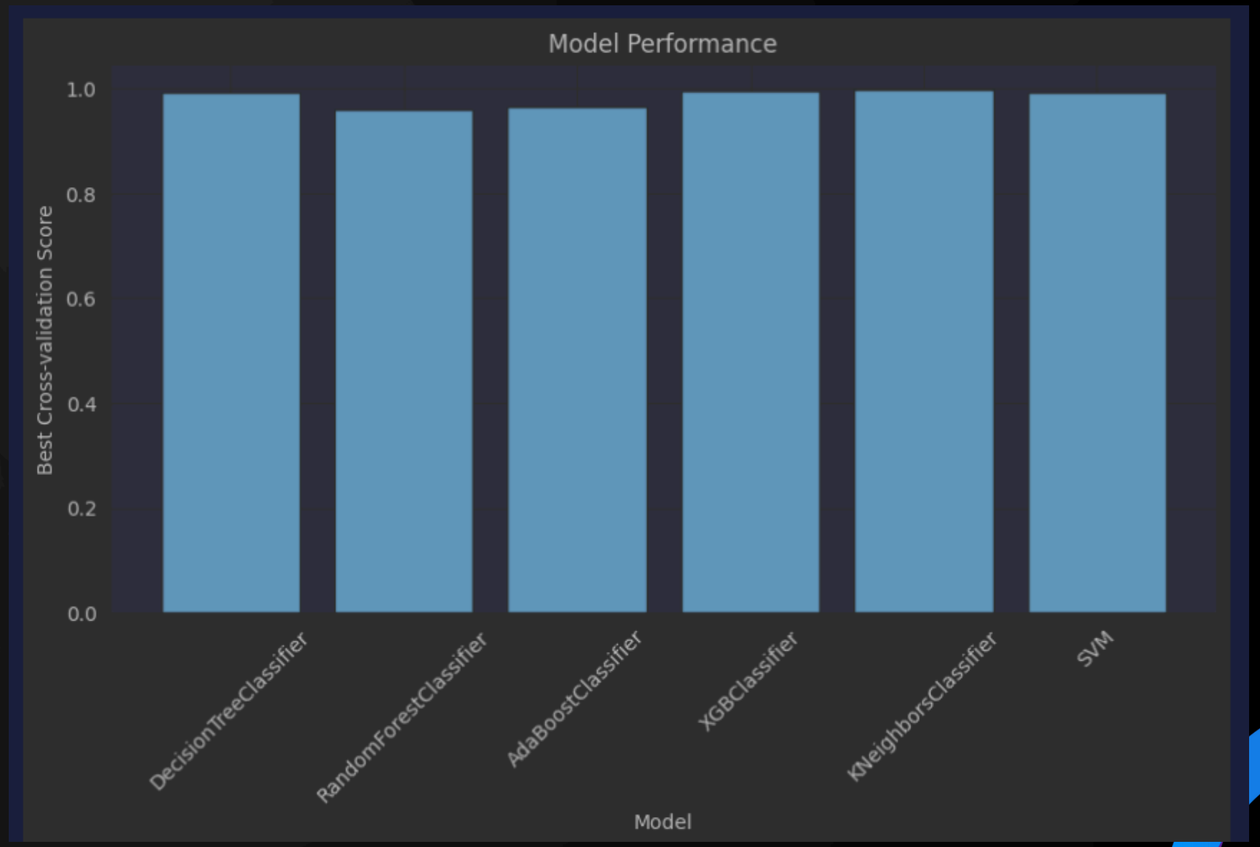
```python
for model_name, model in best_models.items():
    print(f"\nEvaluating {model_name}...")
    y_pred = model.predict(X_test_transformed)
    print(classification_report(y_test, y_pred))
    print(confusion_matrix(y_test, y_pred))
    f1 = f1_score(y_test, y_pred, average='weighted')
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, model.predict_proba(X_test_transformed)[:, 1])
    print(f'F1 Score: {f1}')
    print(f'Accuracy: {accuracy}')
    print(f'ROC AUC Score: {roc_auc}')
    # Plot ROC Curve
    fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test_transformed)[:, 1])
    plt.plot(fpr, tpr, label=f'{model_name} (area = {roc_auc:.2f})')
```

Getting All Model Score

Model	F1-Score	Accuracy	ROC AUC Score
Decision Tree	0.9895	0.9895	0.9939
Random Forest	0.9498	0.9521	0.9955
AdaBoost	0.9564	0.9581	0.9761
XGBoost	0.9947	0.9948	0.9973
KNeighborsClassifier	0.9978	0.9978	0.9948
SVM	0.9932	0.9933	0.9686

Plotting Score Of Each

```python
import matplotlib.pyplot as plt

# Set the figure size
plt.figure(figsize=(10, 5))

# Create a bar plot of the best scores
plt.bar(results_df['Model'], results_df['Best Score'])

# Add labels and title
plt.xlabel('Model')
plt.ylabel('Best Cross-validation Score')
plt.title('Model Performance')
# Tilt the x-axis labels
plt.xticks(rotation=45)

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



Model Evaluation Metrics

The table shows three key performance metrics for various models:

- F1-Score: The harmonic mean of precision and recall, providing a balance between the two.
- Accuracy: The ratio of correctly predicted instances to the total instances.
- ROC AUC Score: The Area Under the Receiver Operating Characteristic Curve, measuring the ability of the model to distinguish between classes.

Detailed Analysis of Each Model

Decision Tree:

The Decision Tree performs well across all metrics, but its simplicity might make it prone to overfitting.

Random Forest:

Random Forest shows good performance, particularly in terms of ROC AUC Score, indicating strong capability in distinguishing between classes. However, its F1-Score is comparatively lower.

AdaBoost:

AdaBoost has a decent balance of all three metrics but does not outperform others significantly in any single metric.

Detailed Analysis of Each Model

XGBoost:

XGBoost performs exceptionally well, especially in ROC AUC Score. Its F1-Score and Accuracy are also very high, making it a strong contender.

KNeighborsClassifier:

KNeighborsClassifier shows excellent performance in F1-Score and Accuracy, with a slightly lower ROC AUC Score compared to XGBoost. Its dependency on the choice of 'k' and potential computational cost might be a consideration.

Support Vector Machine (SVM):

SVM offers a high F1-Score and Accuracy, indicating a good balance between precision and recall, and overall accuracy. While its ROC AUC Score is slightly lower than some other models, it is still robust.

Choosing the Best Model

Why Choose SVM?

High F1-Score and Accuracy:

- *SVM has an F1-Score of 0.9932 and Accuracy of 0.9933, both of which are among the highest, indicating it can accurately identify fraudulent transactions with a good balance of precision and recall.*

Robust Performance:

- *Despite having a slightly lower ROC AUC Score (0.9686), the SVM model's performance in F1-Score and Accuracy makes it a reliable choice for practical applications where precision and recall are crucial.*

Choosing the Best Model

Why Choose SVM?

Complexity Handling:

- *SVM is effective in high-dimensional spaces and is particularly useful when the number of dimensions exceeds the number of samples. This is advantageous in fraud detection scenarios where transaction data might be high-dimensional.*

Generalization Capability:

- *SVMs are known for their capacity to generalize well to unseen data, which is critical in fraud detection to avoid overfitting and ensure the model performs well on new transactions.*

Choosing the Best Model

Why Choose SVM?

Margin Maximization:

- *SVM focuses on maximizing the margin between classes, which helps in creating a more defined decision boundary, reducing the risk of misclassification.*

Comparison with XGBoost and KNeighborsClassifier:

- ***XGBoost:*** *While XGBoost has a slightly better ROC AUC Score and comparable F1-Score and Accuracy, SVM's performance is more balanced and may generalize better in real-world scenarios.*

- ***KNeighborsClassifier:*** *Despite the high F1-Score and Accuracy, KNeighborsClassifier might be computationally intensive and less scalable compared to SVM.*

Classification

```python
from sklearn.metrics import confusion_matrix, classification_report
import pandas as pd

# Predict the test set results
y_pred = pipe.predict(X_test)

# Create a DataFrame from the confusion matrix
cm = confusion_matrix(y_test, y_pred)
cm_df = pd.DataFrame(cm,
                     index = ['Actual Negative', 'Actual Positive'],
                     columns = ['Predicted Negative', 'Predicted Positive'])

# Print the confusion matrix in tabular format
print("Confusion Matrix:")
print(cm_df)

# Print the classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))
```

Executed at 2024.06.14 17:20:00 in 17ms

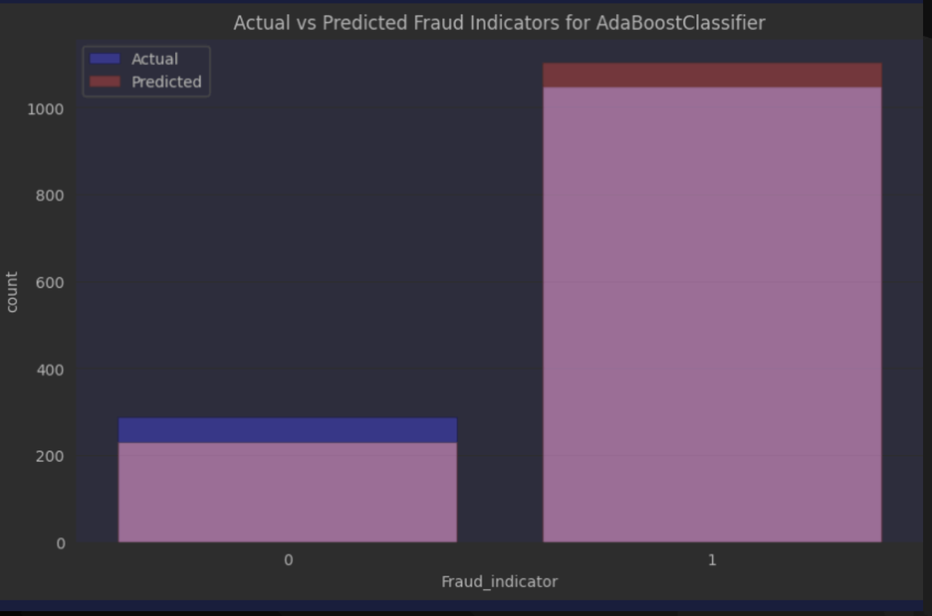
```
Confusion Matrix:
                 Predicted Negative  Predicted Positive
Actual Negative                 287                   0
Actual Positive                   0                1049

Classification Report:
              precision    recall  f1-score   support

    Negative       1.00      1.00      1.00       287
    Positive       1.00      1.00      1.00      1049

    accuracy                           1.00      1336
   macro avg       1.00      1.00      1.00      1336
weighted avg       1.00      1.00      1.00      1336
```

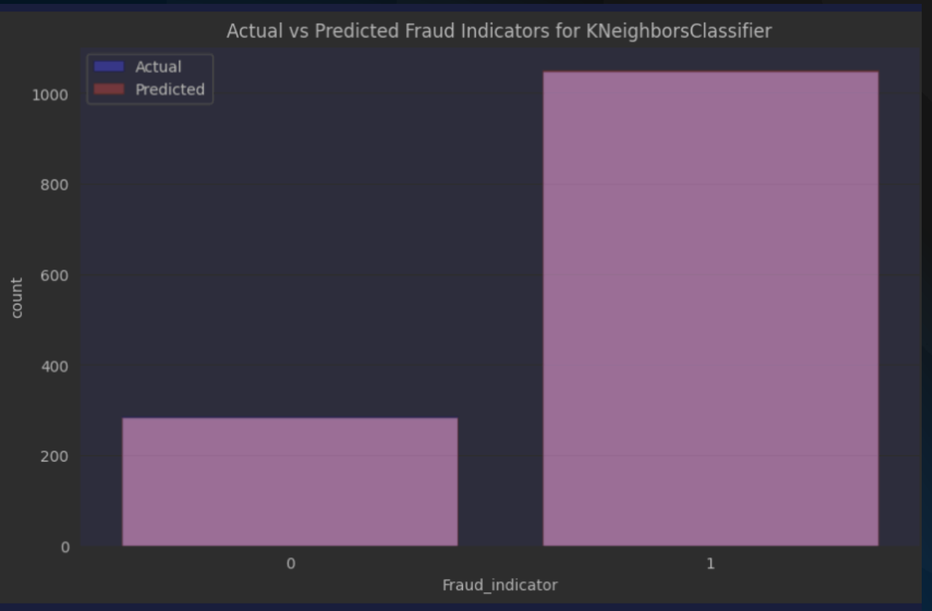
Actual vs Fraud

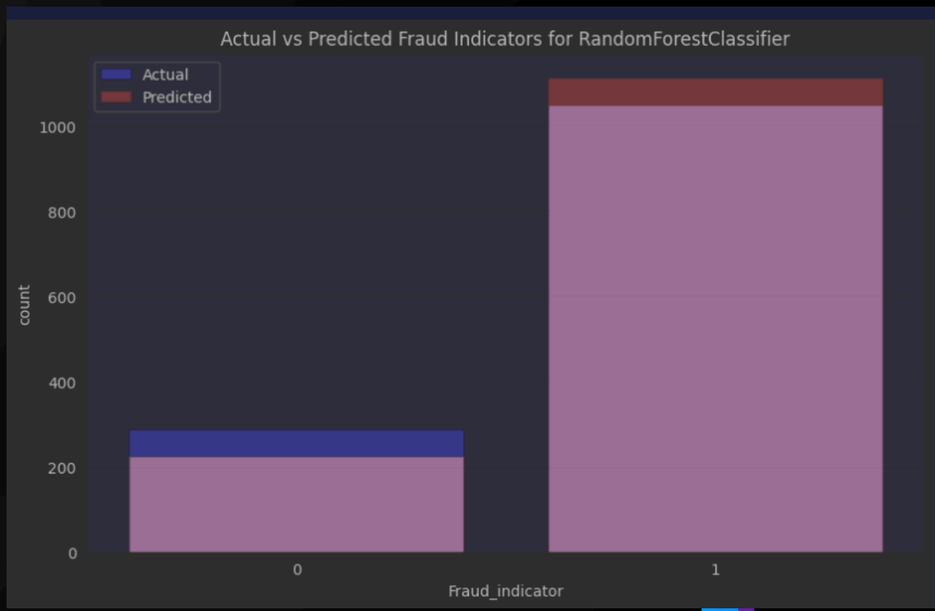
```
# Iterate over each model in the best_models dictionary
for model_name, model in best_models.items():
    # Generate predictions
    y_pred_final = model.predict(X_test_transformed)

    # Create a new figure for each model
    plt.figure(figsize=(10, 6))

    # Create countplots for actual and predicted values
    sns.countplot(x=y_test, label='Actual', color='blue', alpha=0.6)
    sns.countplot(x=y_pred_final, label='Predicted', color='red', alpha=0.4)

    # Add legend and title
    plt.legend()
    plt.title(f'Actual vs Predicted Fraud Indicators for {model_name}')

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

Executed at 2024.06.14 17:20:00 in 1s 432ms

Actual vs Fraud








Actual vs Predicted Fraud Indicators for SVM
Actual
Predicted
1000
800
600
400
200
0
count
0
1
Fraud_indicator


Actual vs Predicted Fraud Indicators for XGBClassifier
Actual
Predicted
1000
800
600
400
200
0
count
0
1
Fraud_indicator

Model Building

- Selection of appropriate machine learning algorithms (e.g., SVM, Random Forest, Neural Networks).
- Training the model on historical data with known fraud labels.

```python
# Define the model
model = svm.SVC(gamma='auto', C=5, kernel='poly')
```

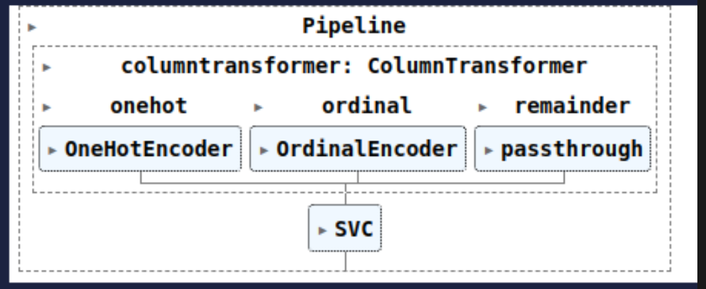
Executed at 2024.06.14 17:20:00 in 2ms

```python
# Create pipeline
pipe = make_pipeline(column_trans, model)
```

Executed at 2024.06.14 17:20:00 in 5ms

```python
# Fit the model
pipe.fit(X_train, y_train)
```

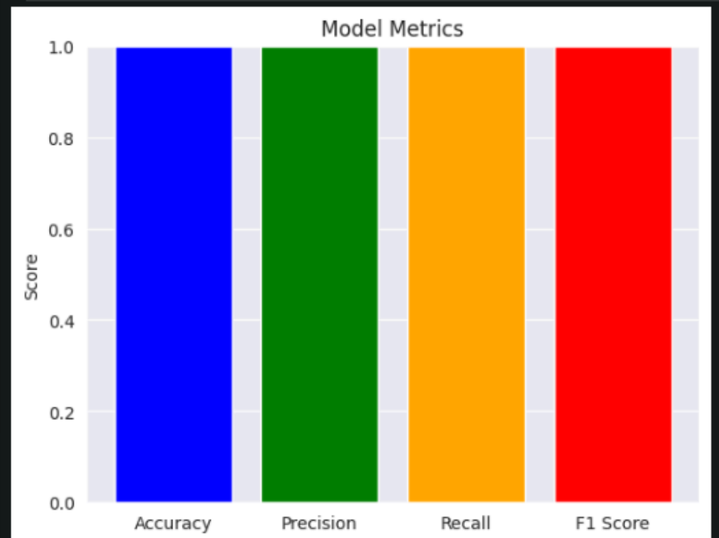
Executed at 2024.06.14 17:20:00 in 618ms




```python
import numpy as np

# Transform the test data
X_test_transformed = column_trans.transform(X_test)

# Make predictions
y_pred_prob = model.predict(X_test_transformed)

# Convert probabilities to binary predictions
y_pred = np.round(y_pred_prob)

# Print accuracy metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print accuracy metrics
print("Accuracy: {:.2f}%".format(accuracy * 100))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1))
```

```
Accuracy: 100.00%
Precision: 1.00
Recall: 1.00
F1 Score: 1.00
```

```python
#Accuracy metrics
import matplotlib.pyplot as plt


metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score']
values = [accuracy, precision, recall, f1]

plt.bar(metrics, values, color=['blue', 'green', 'orange', 'red'])
plt.ylabel('Score')
plt.title('Model Metrics')
plt.ylim(0, 1)
plt.show()
```




```python
with open('SVM_Fast_best.pkl', 'rb') as f:
    model = pickle.load(f)
```

Executed at 2024.06.14 17:33:26 in 2ms

```python
# Make predictions
predictions = model.predict(X)
```

Executed at 2024.06.14 17:33:26 in 23ms

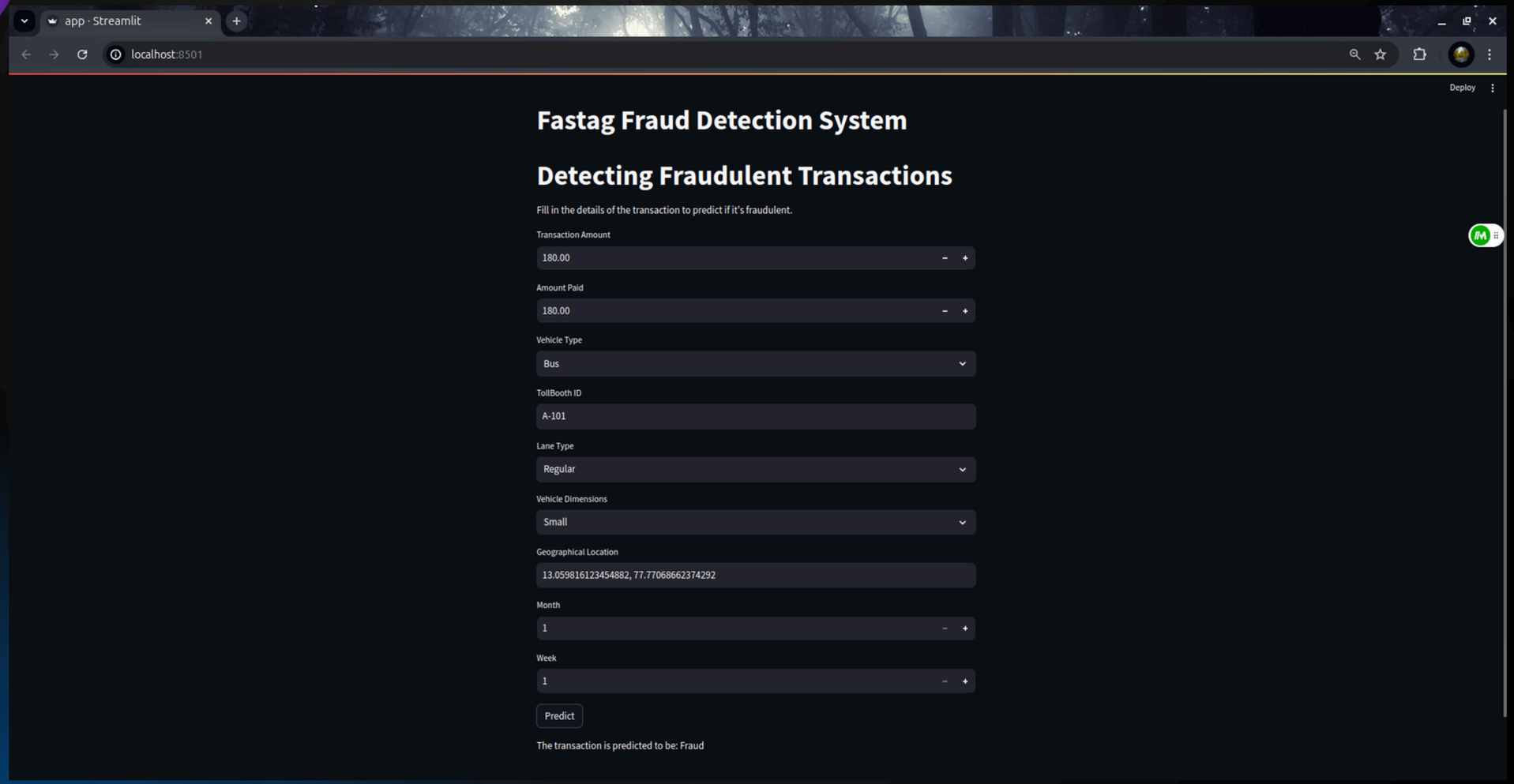
```python
# Print whether each transaction is a fraud or not
for i, prediction in enumerate(predictions):
    print(f'Transaction {i+1} is {"a fraud" if prediction == 1 else "not a fraud"}')
```

Executed at 2024.06.14 17:33:27 in 39ms

```
Transaction 13 is not a fraud
Transaction 14 is not a fraud
Transaction 15 is not a fraud
Transaction 16 is not a fraud
Transaction 17 is a fraud
Transaction 18 is not a fraud
Transaction 19 is a fraud
Transaction 20 is not a fraud
Transaction 21 is not a fraud
Transaction 22 is a fraud
Transaction 23 is not a fraud
Transaction 24 is not a fraud
```

Deployment

- Integrating the trained model into the FastTag transaction processing system.
- Setting up real-time monitoring for fraud detection alerts.
- Ensuring scalability and efficiency for high-volume transaction processing.


app · Streamlit
localhost:8501
Deploy
Fastag Fraud Detection System
Detecting Fraudulent Transactions
Fill in the details of the transaction to predict if it's fraudulent.
Transaction Amount
180.00
Amount Paid
180.00
Vehicle Type
Bus
TollBooth ID
A-101
Lane Type
Regular
Vehicle Dimensions
Small
Geographical Location
13.059816123454882, 77.77068662374292
Month
1
Week
1
Predict
The transaction is predicted to be: Fraud

Benefits

- Enhanced security for FastTag users.
- Reduction in fraudulent transactions and financial losses.
- Improved trust and reliability in the FastTag system.
- Streamlined toll collection process with real-time fraud detection.
- Data-driven decision-making for toll plaza operators and administrators.

Conclusion

- FastTag fraud detection is crucial for maintaining the integrity of electronic toll collection systems.
- Our approach leverages machine learning to effectively identify and prevent fraudulent transactions.
- Ongoing monitoring and model updates will ensure sustained performance and adaptability to new fraud patterns.
- Together, we can create a safer and more efficient toll collection experience for all stakeholders.

THANK YOU

For watching this presentation