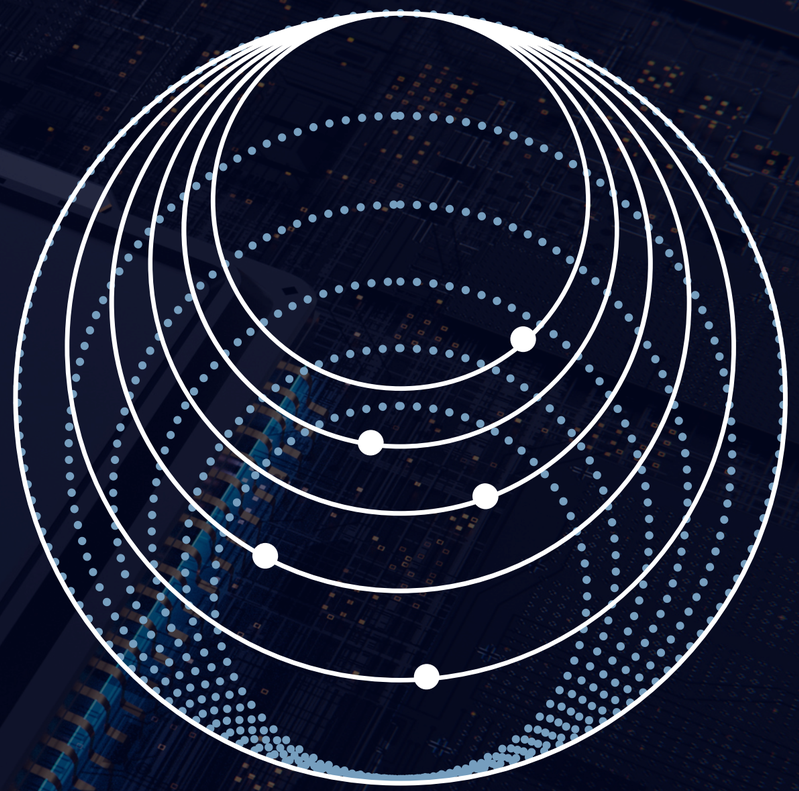
FASTAG FRAUD DETECTION

By Rana Alaa Ahmed

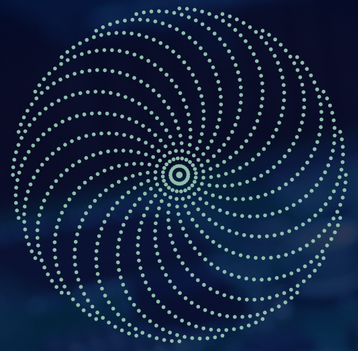
Project Overview



Develop a machine learning system to detect fraudulent Fastag transactions, ensuring secure toll operations. This involves analyzing data to understand fraud prevalence, engineering features for improved detection, training and evaluating models while addressing class imbalance, exploring real-time deployment, and identifying key factors contributing to fraud. Challenges include handling imbalanced data and effective feature engineering. Evaluation will focus on precision, recall, F1 score, and accuracy. Deliverables include a trained model, performance report, feature documentation, and an implementation guide. The expected outcome is a robust system that reduces financial losses and enhances transaction security.

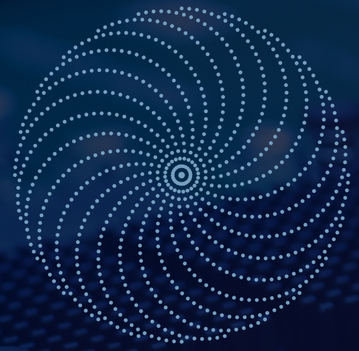
Methodology

Exploratory Data Analysis



Feature Engineering

Data Preprocessing



ML Pipeline

Model Evaluation

Exploratory Data Analysis

Explore the dataset to understand the distribution of features and the prevalence of fraud indicators.

```
# Display the first five rows of the DataFrame 'data'.
data.head().T
```

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

```
# Output a concise summary of the DataFrame 'data',
# including information about the index dtype and co
# non-null values, and memory usage.
data.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
```

```
# Summary statistics for numerical features
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Transaction_ID	5000.0	2500.5000	1443.520003	1.0	1250.75	2500.5	3750.25	5000.0
Transaction_Amount	5000.0	161.0620	112.449950	0.0	100.00	130.0	290.00	350.0
Amount_paid	5000.0	141.2610	106.480996	0.0	90.00	120.0	160.00	350.0
Vehicle_Speed	5000.0	67.8512	16.597547	10.0	54.00	67.0	82.00	118.0

```
print(data['Vehicle_Type'].value_counts())
print(data['Lane_Type'].value_counts())
```

```
Vehicle_Type
Bus           716
Car           714
Motorcycle    714
Truck         714
Van           714
Sedan         714
SUV           714
Name: count, dtype: int64
Lane_Type
Regular    2858
Express    2142
Name: count, dtype: int64
```

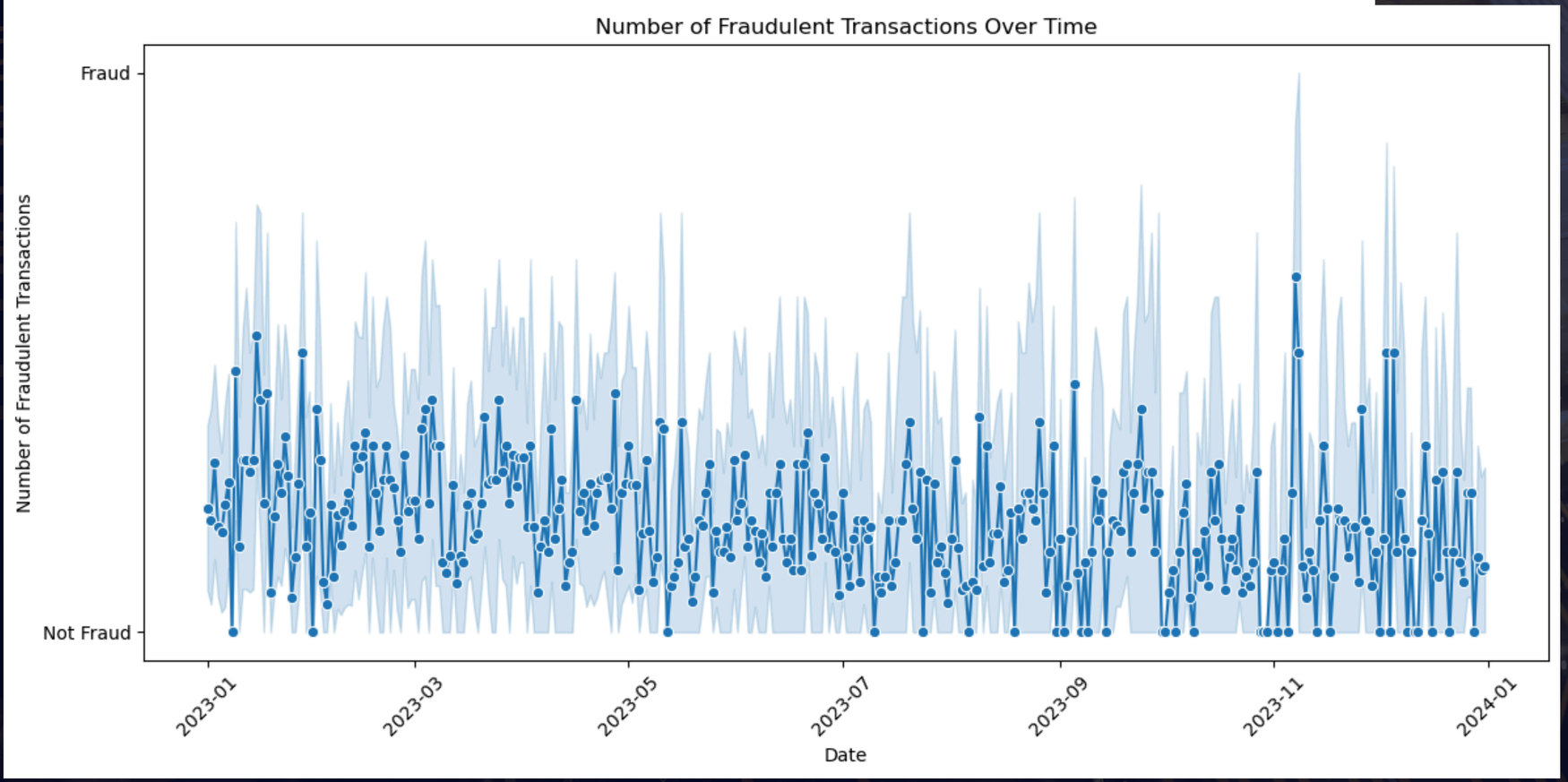
```
# Group by Toll Booth ID to calculate total transaction count and amount
summary = data.groupby('TollBoothID').agg({
    'Transaction_ID': 'count',  # Total transaction count
    'Transaction_Amount': 'sum'  # Total transaction amount
}).reset_index()

# Renaming columns for clarity
summary.columns = ['TollBoothID', 'TotalTransactionCount', 'TotalTransactionAmount']

summary
```

	TollBoothID	TotalTransactionCount	TotalTransactionAmount
0	A-101	1428	150180
1	B-102	1432	196070
2	C-103	1426	459060
3	D-104	40	0
4	D-105	104	0
5	D-106	570	0

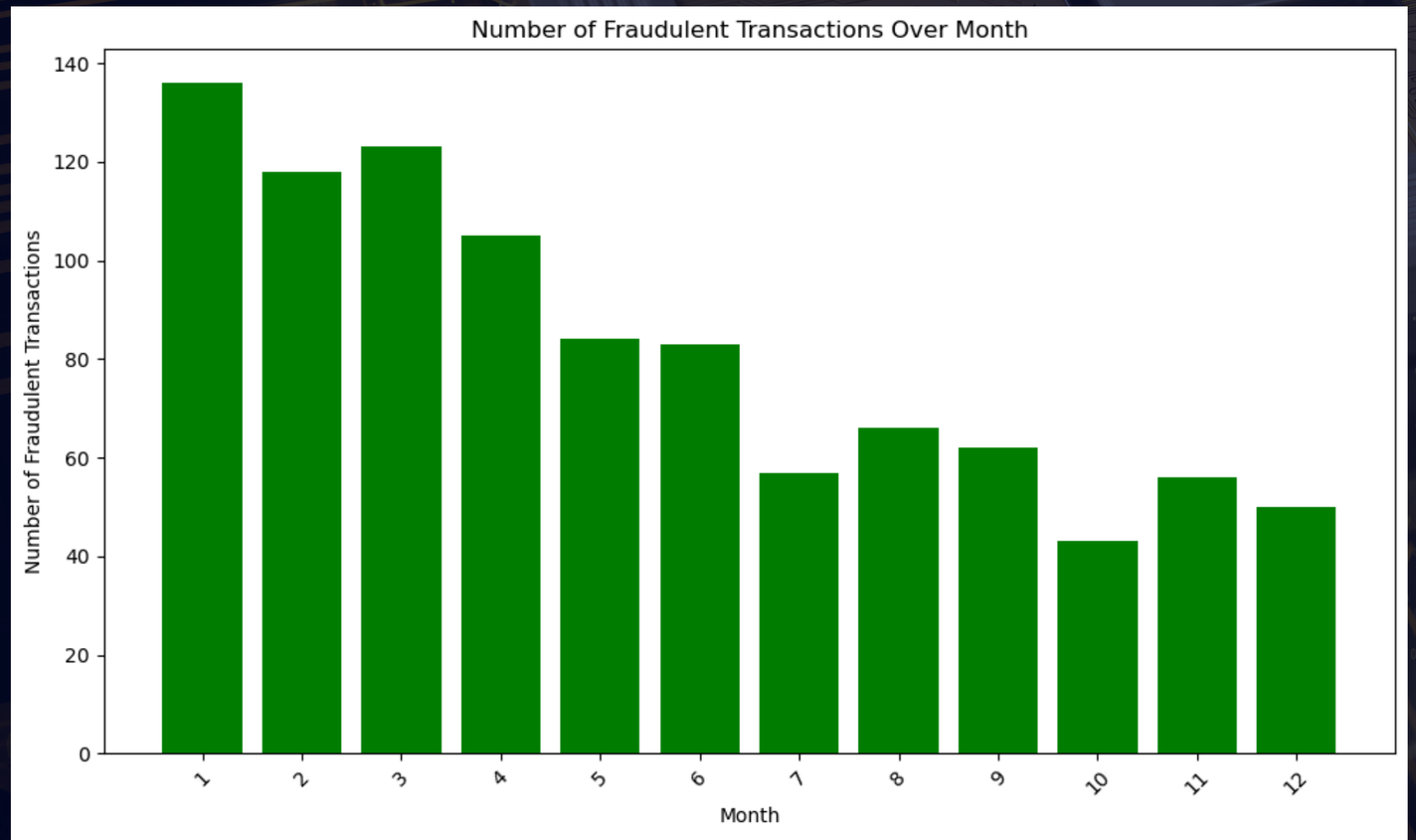
```
# Count the number of missing values
data.isnull().sum()
```

```
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
```

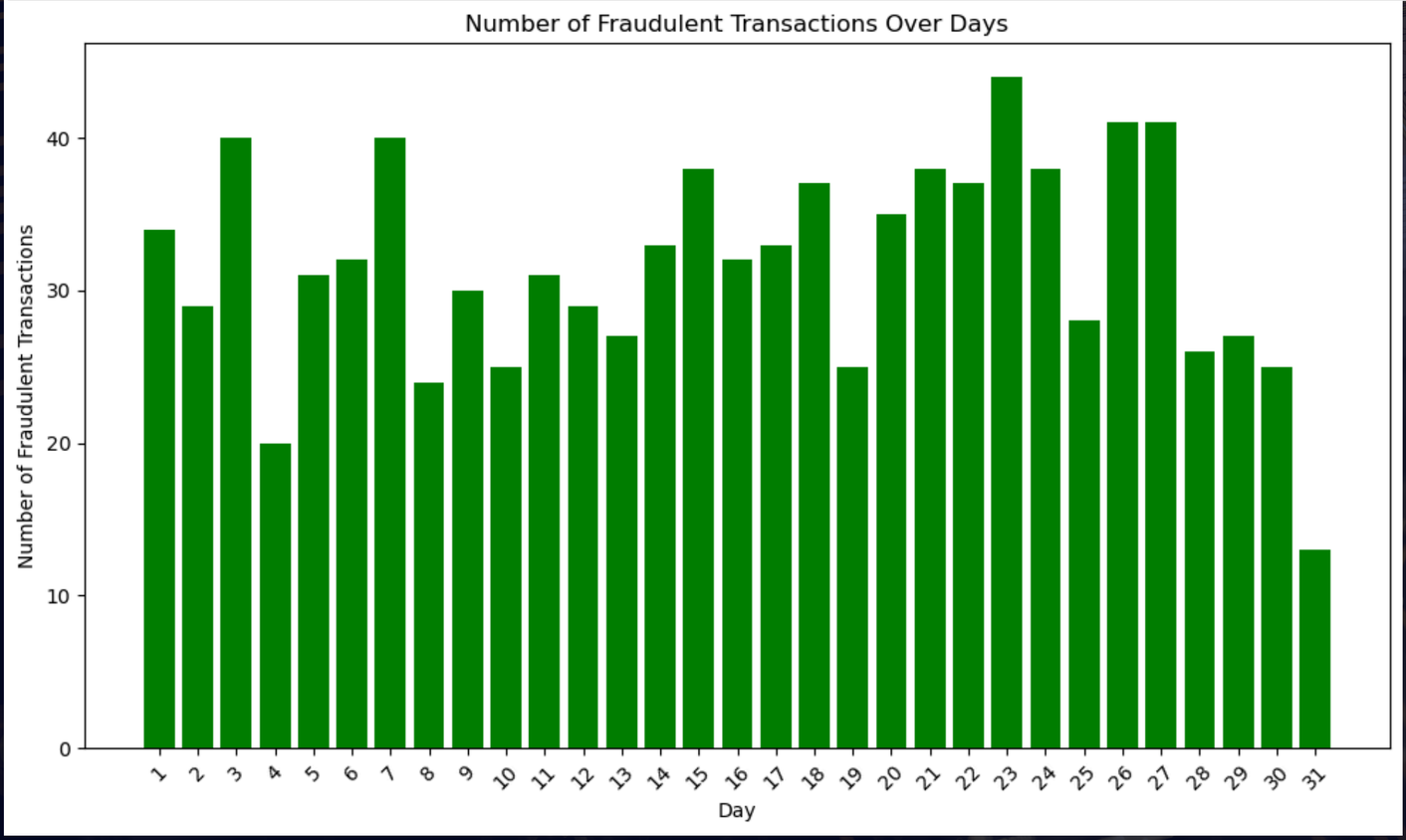
```python
# Plotting the number of fraudulent transactions over time
plt.figure(figsize=(12, 6))
sns.lineplot(x='Date', y='Fraud_indicator', data=data, marker='o')
plt.title('Number of Fraudulent Transactions Over Time')
plt.xlabel('Date')
plt.ylabel('Number of Fraudulent Transactions')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



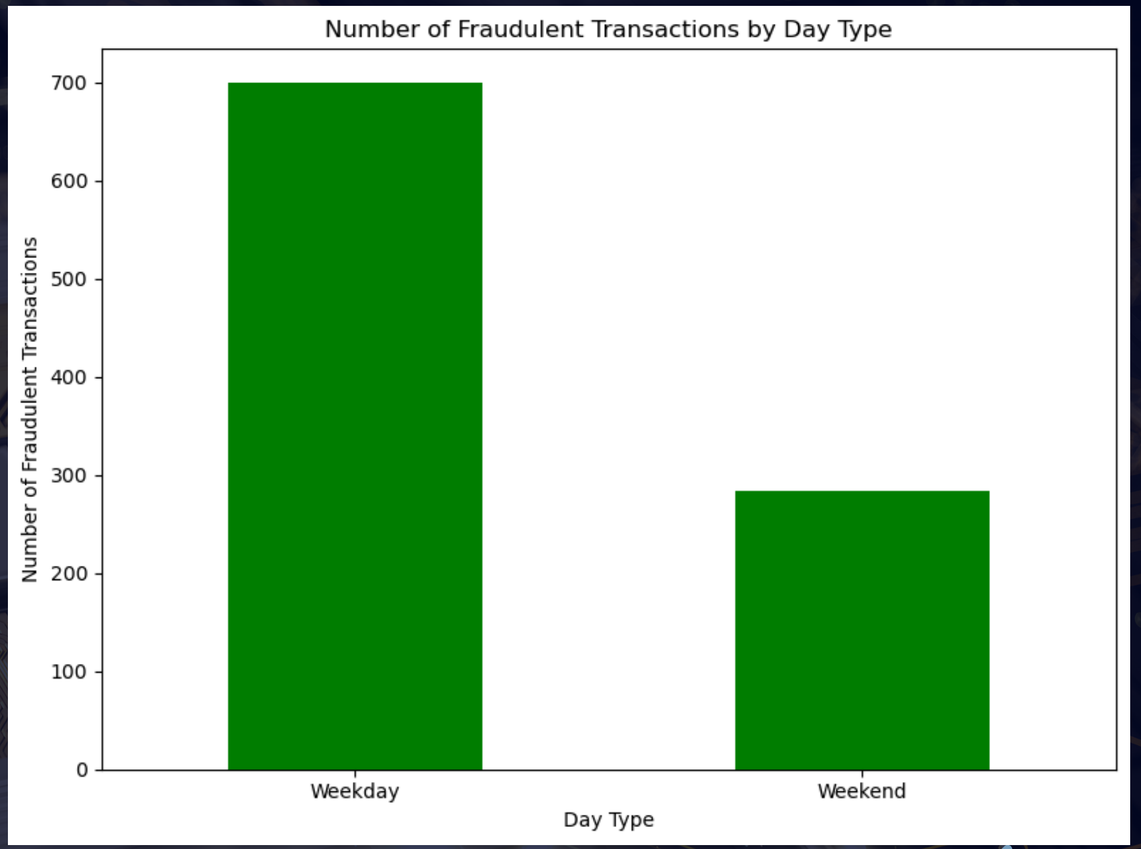
```python
# Group by Month and count fraudulent transactions
fraud_by_month = data[data['Fraud_indicator'] == 'Fraud'].groupby('Month').size()

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(fraud_by_month.index.astype(str), fraud_by_month.values, color='green')
plt.title('Number of Fraudulent Transactions Over Month')
plt.xlabel('Month')
plt.ylabel('Number of Fraudulent Transactions')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



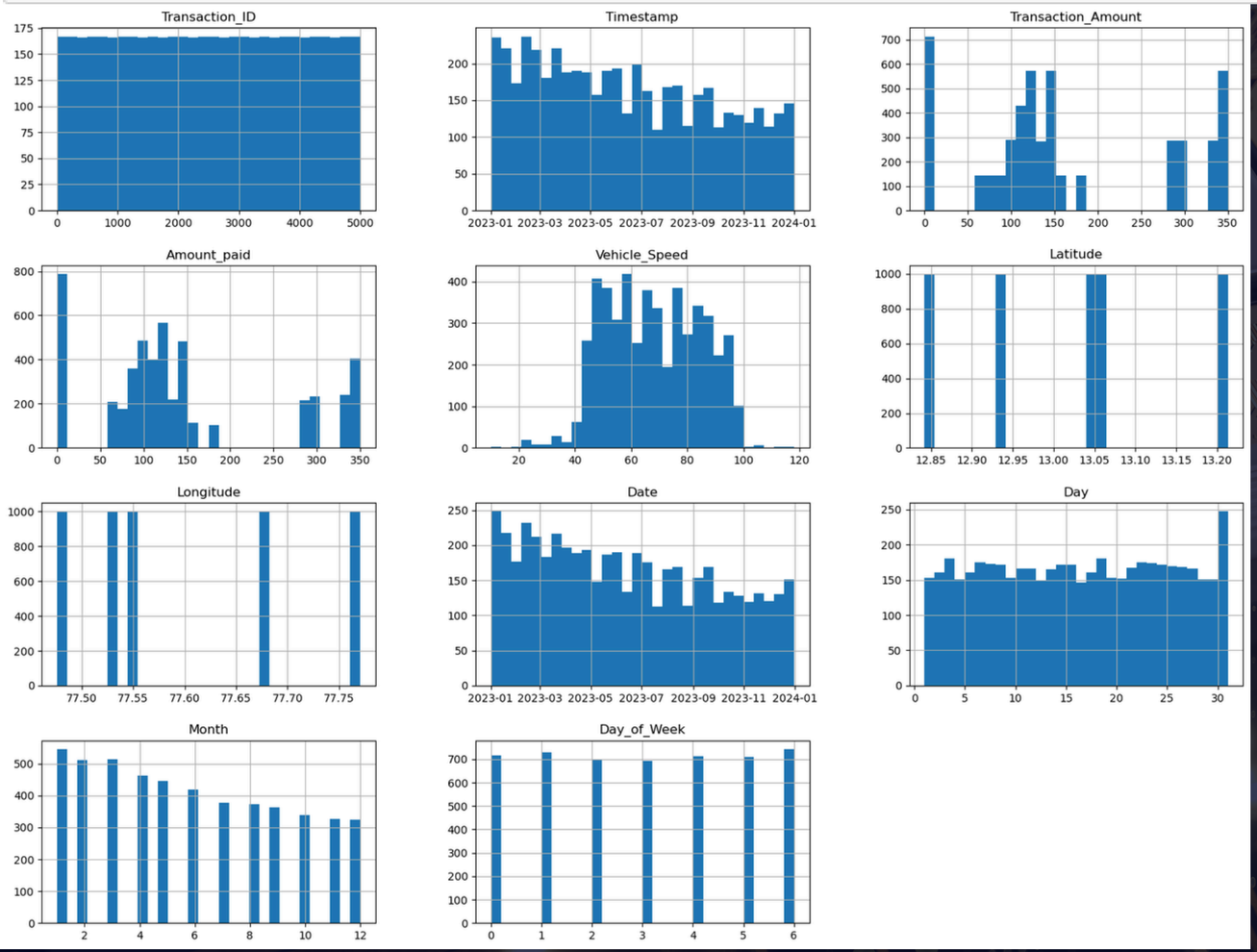
```python
# Group by day and count fraudulent transactions
fraud_by_day = data[data['Fraud_indicator'] == 'Fraud'].groupby('Day').size()

# Plotting
plt.figure(figsize=(10, 6))
plt.bar(fraud_by_day.index.astype(str), fraud_by_day.values, color='green')
plt.title('Number of Fraudulent Transactions Over Days')
plt.xlabel('Day')
plt.ylabel('Number of Fraudulent Transactions')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



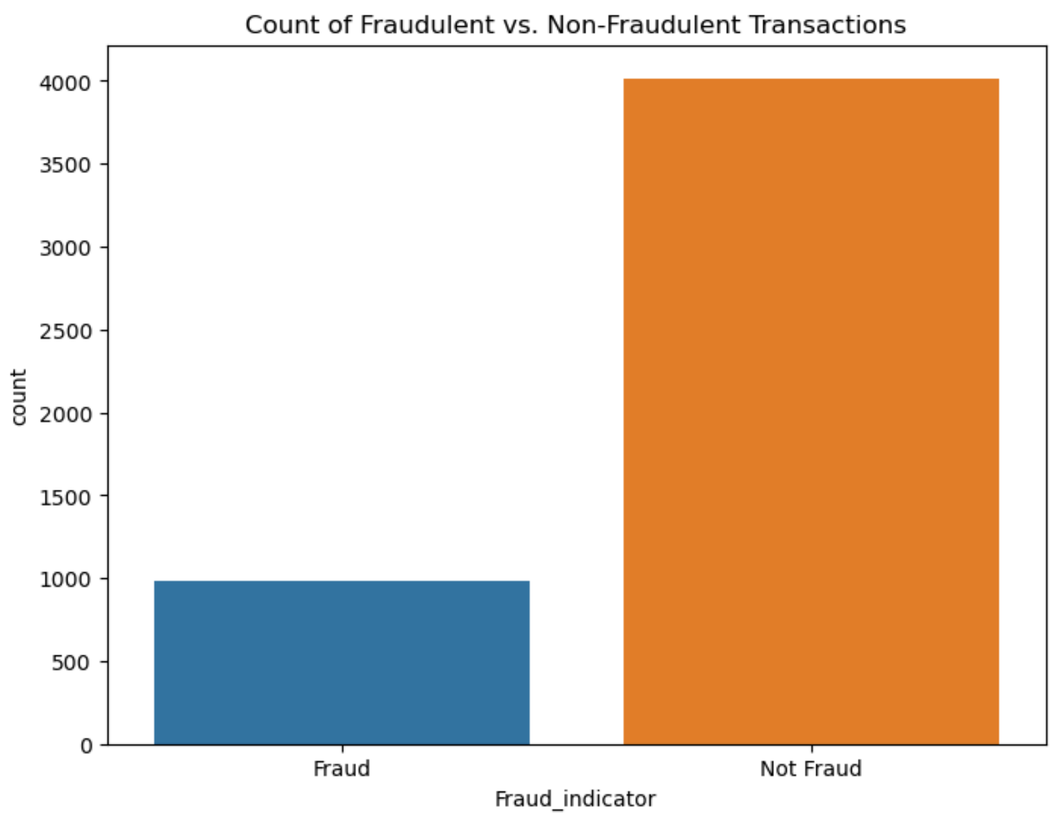
```python
# Extract day of the week (0 = Monday, 6 = Sunday)
data['Day_of_Week'] = data['Date'].dt.dayofweek

# Define function to categorize weekdays (Monday to Friday) and weekends (Saturday and Sunday)
def categorize_day(day):
    if day < 5:
        return 'Weekday'
    else:
        return 'Weekend'

# Apply the categorize_day function to create a new column 'Day_Type'
data['Day_Type'] = data['Day_of_Week'].apply(categorize_day)

# Group by 'Day_Type' and count fraudulent transactions
fraud_by_day = data[data['Fraud_indicator'] == 'Fraud'].groupby('Day_Type').size()

# Plotting
plt.figure(figsize=(8, 6))
fraud_by_day.plot(kind='bar', color='green')
plt.title('Number of Fraudulent Transactions by Day Type')
plt.xlabel('Day Type')
plt.ylabel('Number of Fraudulent Transactions')
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```



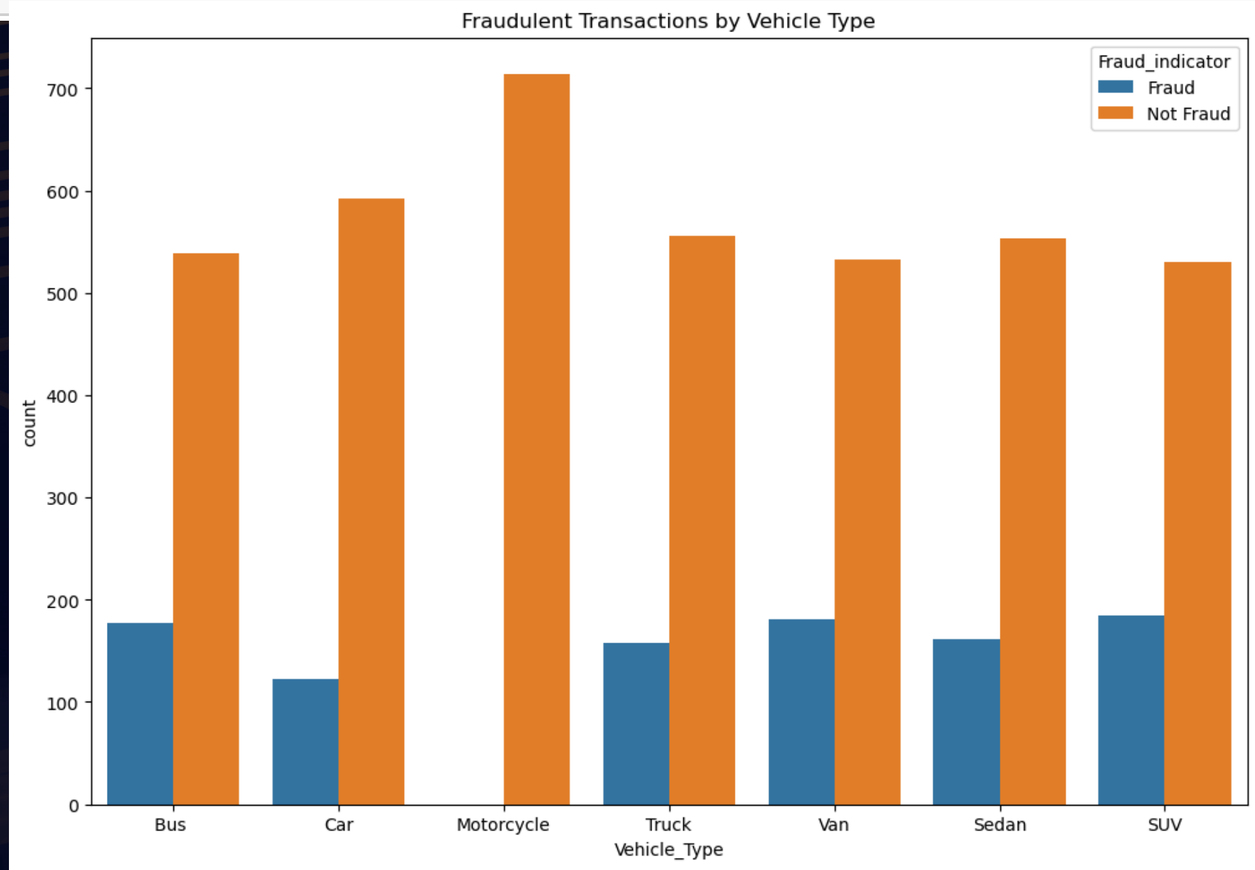
```python
# Display histograms for all numerical features in the DataFrame 'data'.
data.hist(bins=30, figsize=(20, 15))
plt.show()
```



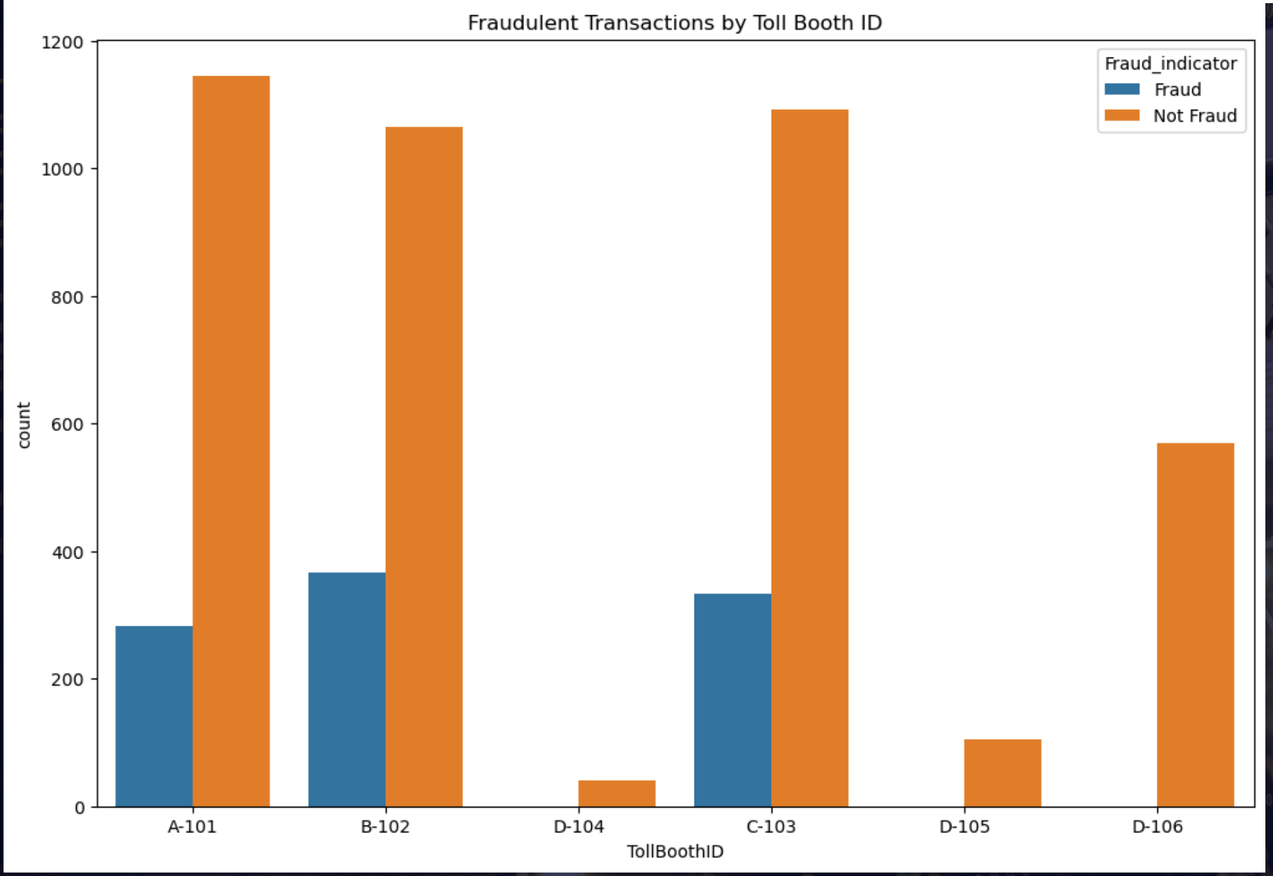
```
# Count of Fraudulent vs. Non-Fraudulent Transactions
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='Fraud_indicator')
plt.title('Count of Fraudulent vs. Non-Fraudulent Transactions')
plt.show()
```



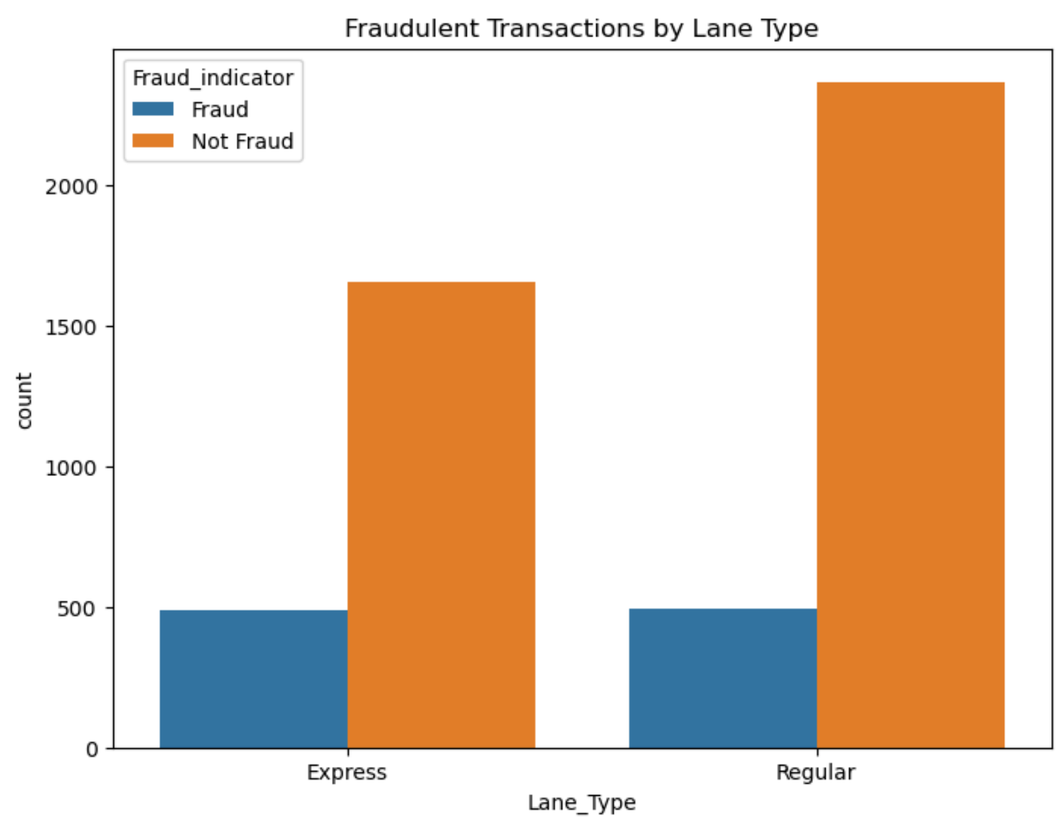

```
# Vehicle Type Analysis
plt.figure(figsize=(12, 8))
sns.countplot(data=data, x='Vehicle_Type', hue='Fraud_indicator')
plt.title('Fraudulent Transactions by Vehicle Type')
plt.show()
```

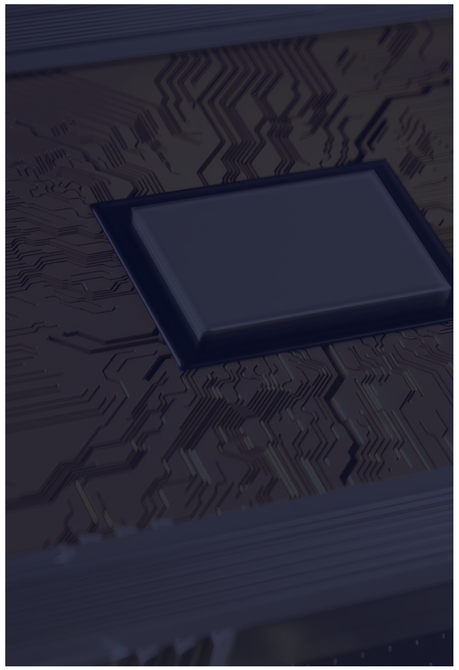


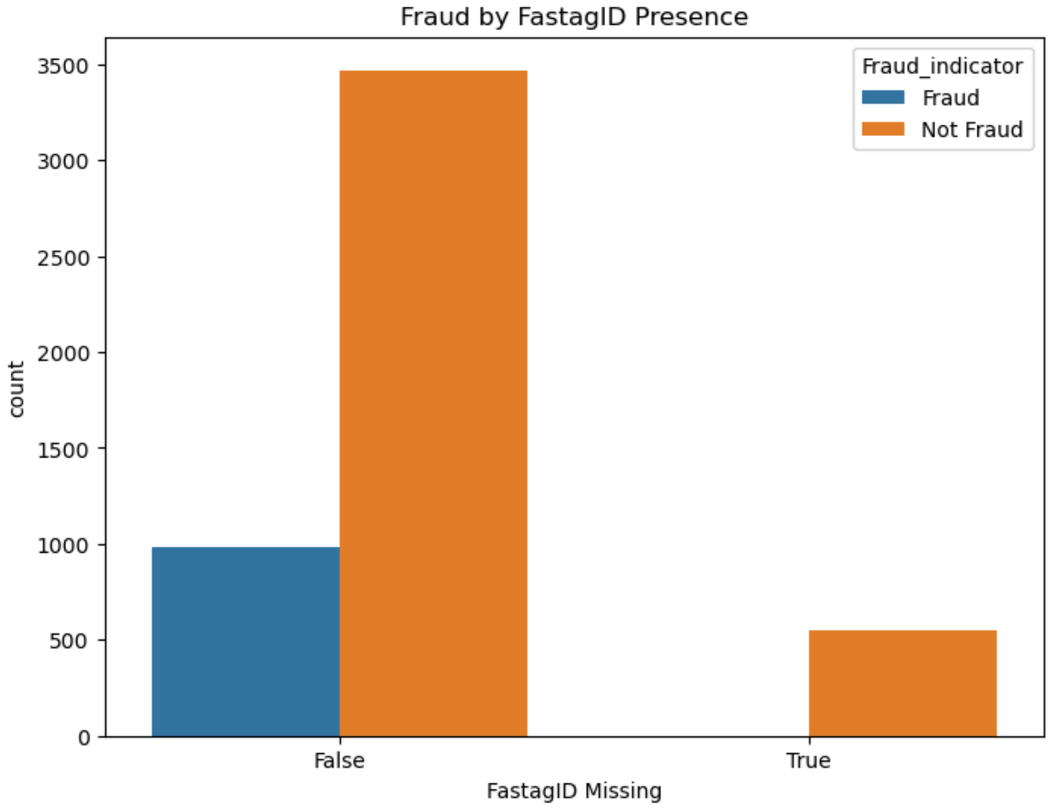
```python
# Toll Booth ID Analysis
plt.figure(figsize=(12, 8))
sns.countplot(data=data, x='TollBoothID', hue='Fraud_indicator')
plt.title('Fraudulent Transactions by Toll Booth ID')
plt.show()
```



```python
# Lane Type Analysis
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='Lane_Type', hue='Fraud_indicator')
plt.title('Fraudulent Transactions by Lane Type')
plt.show()
```

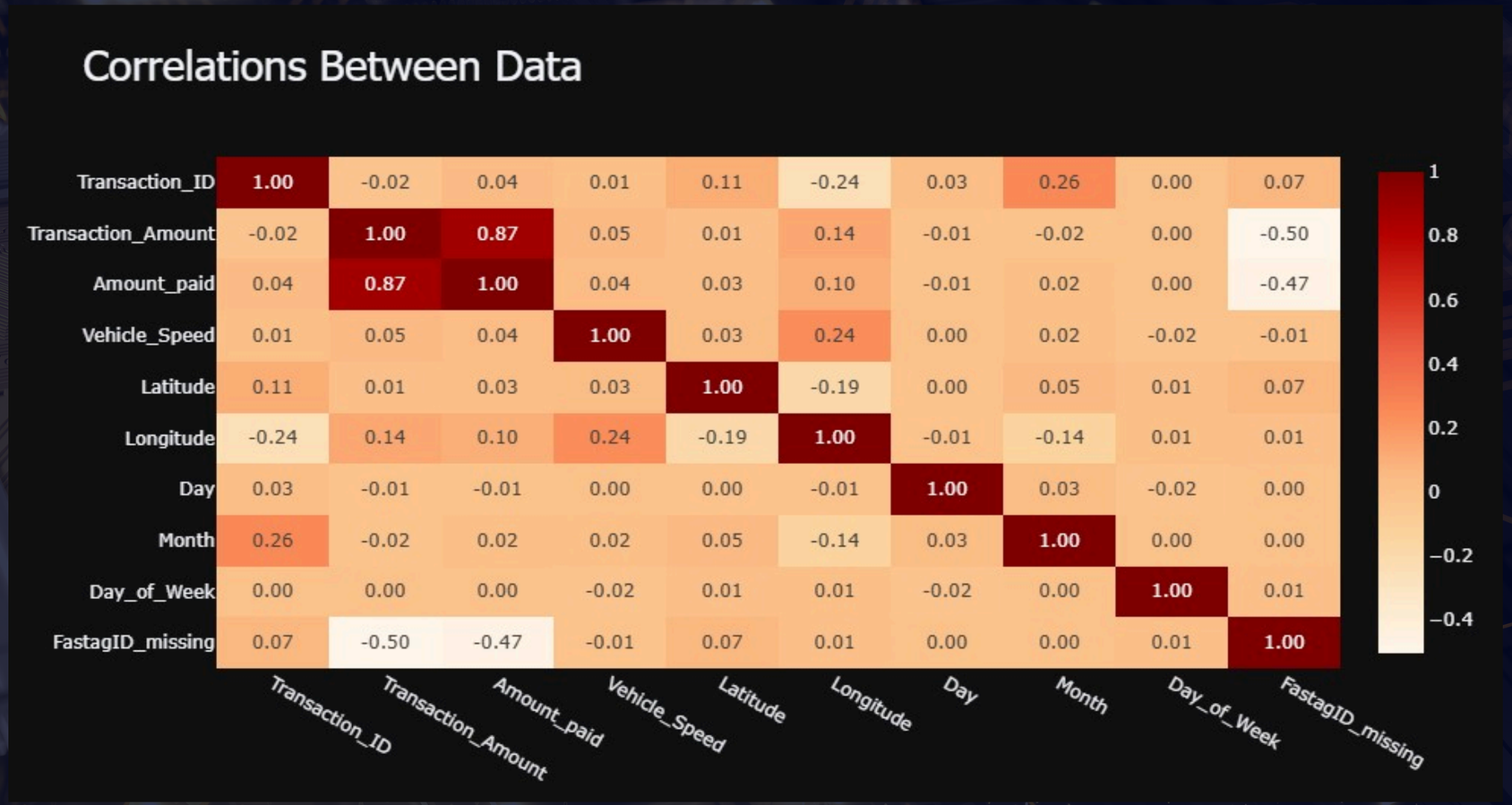







```python
# Create a new column indicating whether FastagID is missing or not
data['FastagID_missing'] = data['FastagID'].isnull()

# Plot using the new column
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='FastagID_missing', hue='Fraud_indicator')
plt.title('Fraud by FastagID Presence')
plt.xlabel('FastagID Missing')
plt.show()
```

```python
# Calculate the correlation matrix for
# numerical columns in the DataFrame 'data'.
correlation = data.corr(numeric_only=True)

# Plot the correlation matrix using Plotly.
fig = px.imshow(
    correlation,
    template="plotly_dark",
    text_auto="0.2f",
    aspect=1,
    color_continuous_scale="orrd",
    title="Correlations Between Data"
)

# Update the layout of the plot.
fig.update_layout(
    title={
        "font": {
            "size": 28,
            "family": "tahoma"
        }
    }
)
fig
```



Feature Engineering

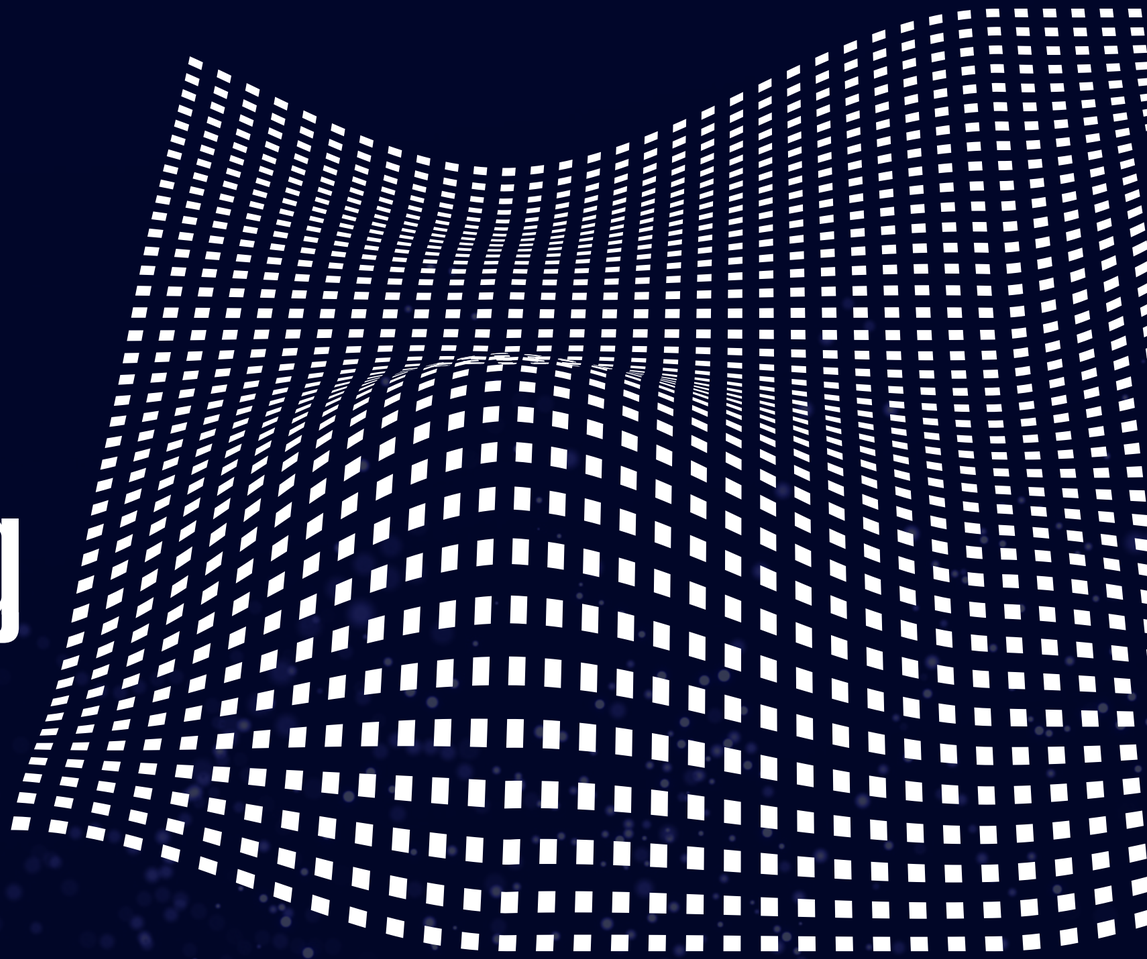
Identify and engineer relevant features that contribute to fraud detection accuracy.

```
# Split Geographical_Location into latitude and longitude
data[['Latitude', 'Longitude']] = data['Geographical_Location'].str.split(',', expand=True)
data['Latitude'] = data['Latitude'].astype(float)
data['Longitude'] = data['Longitude'].astype(float)
```

```
# Convert 'Timestamp' column to datetime format
data['Timestamp'] = pd.to_datetime(data['Timestamp'])

# Create separate date and time columns
data['Date'] = data['Timestamp'].dt.date
data['Time'] = data['Timestamp'].dt.time
```

```
# Convert 'Date' column to datetime format
data['Date'] = pd.to_datetime(data['Date'])

# Split 'Date' into day and month columns
data['Day'] = data['Date'].dt.day
data['Month'] = data['Date'].dt.month
```

Data Preprocessing



- Prepare the data for model training by :
  - Drop irrelevant columns
  - Handle missing values
  - Encode categorical variables
  - scale or normalize features as needed.

```python
# Replace missing FastagID with a unique category
data['FastagID'].fillna('Missing', inplace=True)
```

```python
# Drop unnecessary columns
data.drop(['Transaction_ID', 'Timestamp', 'Geographical_Location', 'Vehicle_Plate_Number', 'FastagID'], axis=1, inplace=True)
```

```python
# Map 'Fraud_indicator' to binary labels: 'Fraud' -> 1, 'Not Fraud' -> 0
data['Fraud_indicator'] = data['Fraud_indicator'].map({'Fraud': 1, 'Not Fraud': 0})
```

```python
# Split data into features and target variable
X = data.drop('Fraud_indicator', axis=1)
y = data['Fraud_indicator']
```

```python
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

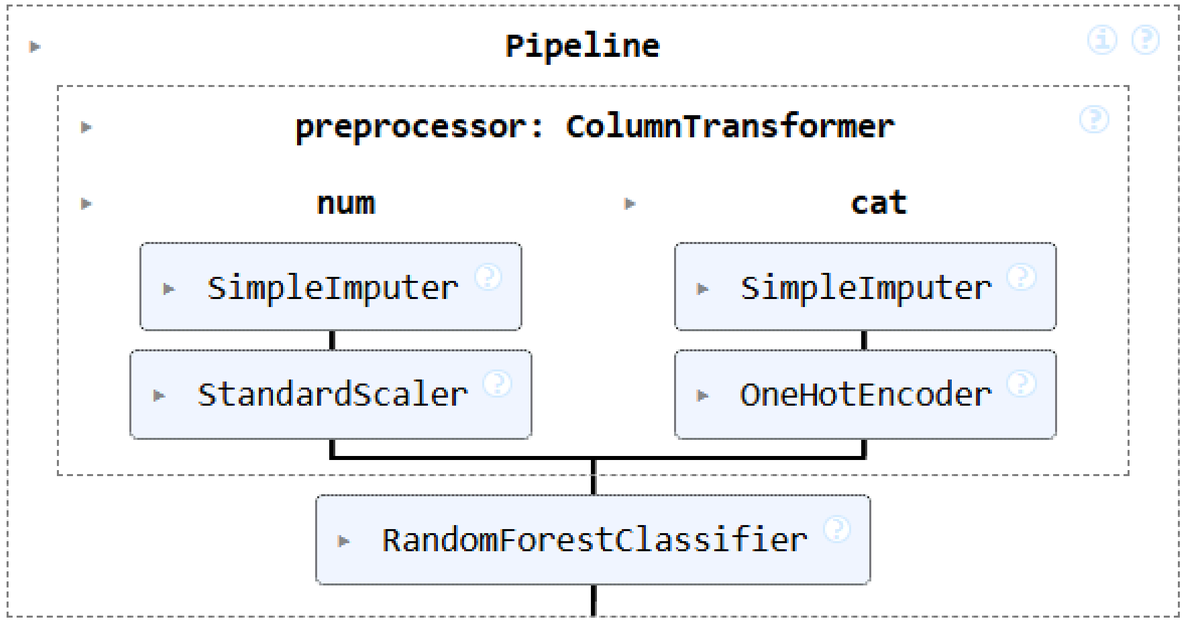
Create ML Pipeline to streamline the end-to-end machine learning process, from data preprocessing to model training.

ML Pipeline

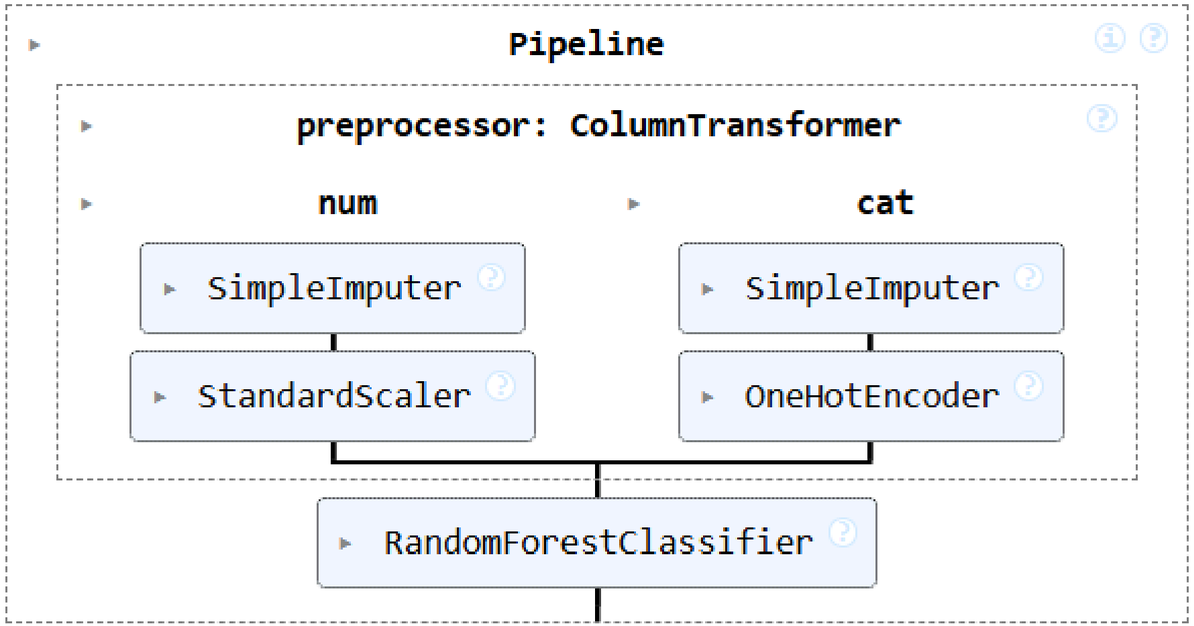
```python
# Preprocessing pipeline for numerical and categorical features
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_features = X.select_dtypes(include=['object']).columns
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

```python
# Append classifier to preprocessing pipeline
rf_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('classifier', RandomForestClassifier(random_state=42))])
```

```python
# Fit the pipeline to the training data
rf_pipeline.fit(X_train, y_train)
```




```python
# Fit the pipeline with interaction features to the training data
rf_pipeline.fit(X_train_interact, y_train)
```




```python
# Feature engineering: Creating interaction features
X_train_interact = X_train.copy()
X_test_interact = X_test.copy()

interaction_features = [('Transaction_Amount', 'Vehicle_Speed')]

for feature1, feature2 in interaction_features:
    interaction_name = f"{feature1}_x_{feature2}"
    X_train_interact[interaction_name] = X_train[feature1] * X_train[feature2]
    X_test_interact[interaction_name] = X_test[feature1] * X_test[feature2]
```

Model Evaluation

he process of using different evaluation metrics to understand a machine learning model's performance

```python
# Evaluation on test data
y_pred = rf_pipeline.predict(X_test_interact)
y_pred_proba = rf_pipeline.predict_proba(X_test_interact)[:, 1]
```

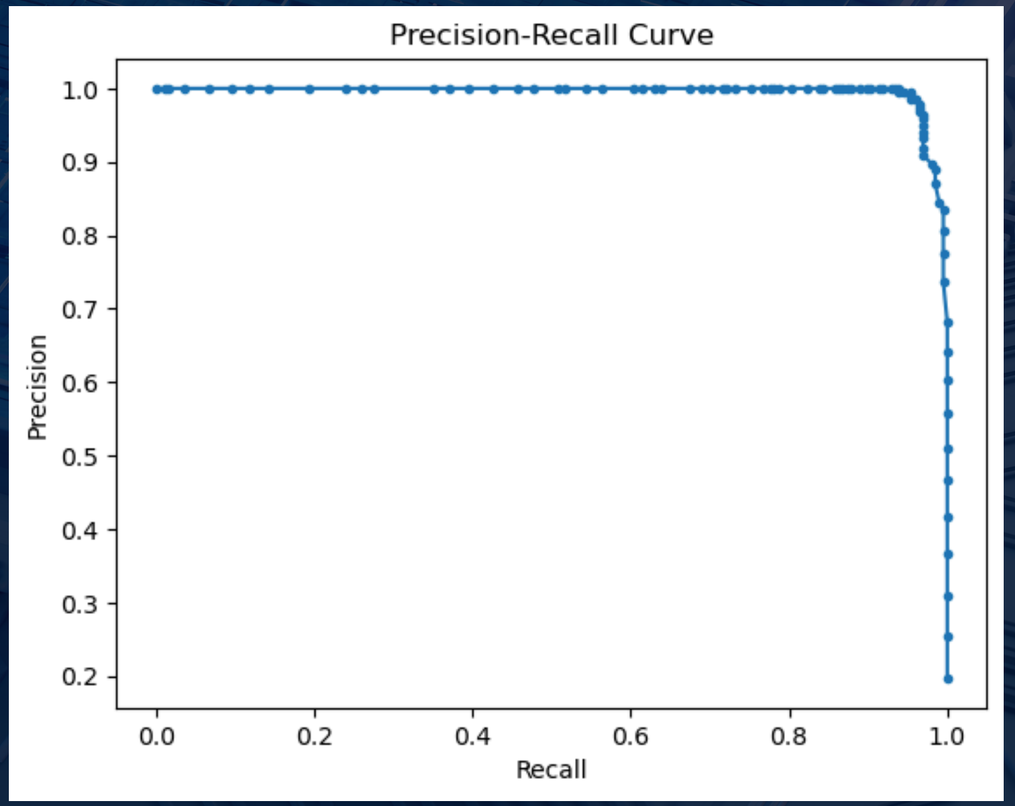
```python
# Display evaluation metrics
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
```

```
Accuracy: 0.978
Precision: 1.0
Recall: 0.8883248730964467
F1 Score: 0.9408602150537635
```

```python
# Evaluate model performance
print('Classification Report:\n', classification_report(y_test, y_pred))
print('ROC-AUC Score:', roc_auc_score(y_test, y_pred_proba))
```

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

           0       0.97      1.00      0.99       803
           1       1.00      0.89      0.94       197

    accuracy                           0.98      1000
   macro avg       0.99      0.94      0.96      1000
weighted avg       0.98      0.98      0.98      1000

ROC-AUC Score: 0.998539739934636
```

```python
# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)
plt.plot(recall, precision, marker='.')
plt.title('Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```



THANK YOU