INDEX

❖ Problem Definition

Cyber Fraud Detection in Online Payments

The problem is to develop an effective machine learning model to detect cyber fraud in online payments. With the increasing prevalence of online transactions, there is a growing concern about cyber fraud activities, such as unauthorized access, identity theft, and fraudulent transactions. The goal is to build a system that can accurately identify fraudulent activities and prevent financial losses for individuals and businesses.

Key Objectives:

- ✓ Develop a machine learning model capable of distinguishing between legitimate and fraudulent online transactions.
- ✓ Minimize false positives (legitimate transactions misclassified as fraudulent) and false negatives (fraudulent transactions not detected).
- ✓ Provide real-time detection and alerts to prevent fraudulent transactions from being processed.
- ✓ Improve the security and trustworthiness of online payment systems to enhance user confidence and reduce financial risks.

❖ <u>Selection of the dataset</u>

About Data:
To detect online payment fraud using machine learning, we must train a model capable of classifying transactions as either fraudulent or non-fraudulent. This necessitates access to a dataset containing relevant information about online payment fraud, enabling us to discern patterns indicative of fraudulent activity. To this end, I sourced a dataset from Kaggle comprising historical data on fraudulent transactions, suitable for detecting online payment fraud. Below is a list of all the columns included in the dataset I'm utilizing for this purpose:

Feature	Description
step	tells about the unit of time
type	type of transaction done
amount	the total amount of transaction
nameOrg	account that starts the transaction
oldbalanceOrg	Balance of the account of sender before transaction
newbalanceOrg	Balance of the account of sender after transaction
nameDest	account that receives the transaction
oldbalanceDest	Balance of the account of receiver before transaction
newbalanceDest	Balance of the account of receiver after transaction
isFraud	The value to be predicted i.e. 0 or 1

Dataset Link:
https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection

❖ **Techniques of the Data Preprocessing**

➢ **Data Cleaning:** Remove duplicate entries.

Technique – 1

```
# Technique-1: Remove duplicate rows
data.drop_duplicates(inplace=True)
```

➢ **Handling Missing Values**: Replace missing values in numerical features with mean/median, and categorical features with mode.

Technique – 2

```
# Find rows with null values
rows_with_null = data[data.isnull().any(axis=1)]

# Display rows with null values
print("Rows with null values:")
rows_with_null
```

```
Rows with null values:
step    type    amount nameOrig       oldbalanceOrg  newbalanceOrig        nameDest
        oldbalanceDest newbalanceDest        isFraud  isFlaggedFraud
42270   9       CASH_OUT      271441.28      C2034845877   0.0      0.0      C71127 NaN
        NaN     NaN     NaN
```

```
data.columns
```

```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
     'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
     'isFlaggedFraud'],
    dtype='object')
```

```
# Technique-2: Replace missing values with median
for attr in data.columns:
  if((data[attr].dtype=='int64' or data[attr].dtype=='float64') and
data[attr].isna().sum()):
    data[attr].fillna(data[attr].median(), inplace=True)
```

```
# Find rows with null values
rows_with_null = data[data.isnull().any(axis=1)]

# Display rows with null values
print("Rows with null values:")
rows_with_null
```

```
Rows with null values:
step    type    amount nameOrig       oldbalanceOrg  newbalanceOrig        nameDest
        oldbalanceDest newbalanceDest        isFraud  isFlaggedFraud
```

Technique – 3

```
# Technique-3: Replace missing values with mean
for attr in data.columns:
  if((data[attr].dtype=='int64' or data[attr].dtype=='float64') and
data[attr].isna().sum()):
    data[attr].fillna(data[attr].mean(), inplace=True)
```

Technique – 4

```
# Technique-4: Replace missing values with mode
for attr in data.columns:
  if((data[attr].dtype=='int64' or data[attr].dtype=='float64') and
data[attr].isna().sum()):
    data[attr].fillna(data[attr].mode(), inplace=True)
```

Technique – 5

```
# Technique-5: Remove irrelevent column
if('isFlaggedFraud' in data):
  data.drop('isFlaggedFraud', axis=1, inplace=True)
  print("'isFlaggedFraud' column dropped suuccessfully from dataset.")
```

```
'isFlaggedFraud' column dropped suuccessfully from dataset.
```

Technique – 6

```
# Technique-6: Remove rows with null values
data_cleaned = data.dropna()
```

```
# Find rows with null values
rows_with_null = data_cleaned[data_cleaned.isnull().any(axis=1)]

# Display rows with null values
print("Rows with null values:")
rows_with_null
```

```
Rows with null values:
step    type    amount nameOrig    oldbalanceOrg  newbalanceOrig      nameDest
        oldbalanceDest newbalanceDest       isFraud
```

- **Feature Transformation**: Apply logarithmic transformation to skewed numerical features.

Technique – 7

```
# Technique-7: Logarithmic transformation
import numpy as np
if('log_feature' in data):
  data['log_feature'] = np.log(data['feature'])
```

- **Normalization**: Scale numerical features to a similar range using normalization techniques.

Technique – 8

```
# Technique-8: Min-Max Normalization
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Load the dataset
df_minmax = pd.read_csv('onlinefraud.csv')

# Extract numerical features
numerical_features = df_minmax.select_dtypes(include=['int',
'float']).columns

# Apply Min-Max normalization
scaler = MinMaxScaler()
df_minmax[numerical_features] =
scaler.fit_transform(df_minmax[numerical_features])
```

- **Outlier Detection and Removal**: Identify and remove outliers using z-score or IQR method.

Technique – 9

```
# Technique-9: z-score Normalization
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
df_zscore = pd.read_csv('onlinefraud.csv')

# Extract numerical columns for normalization
numerical_columns = df_zscore.select_dtypes(include=['int64',
'float64']).columns

# Apply z-score normalization
scaler = StandardScaler()
df_zscore[numerical_columns] =
scaler.fit_transform(df_zscore[numerical_columns])
```

- **Binning:** Bin numerical feature into categories

Technique – 10

```
# Technique-10: Binning (Discretization of Continuous Variables)
# Bin numerical feature into categories
```

```
if('numerical_feature' in data):
  data['binned_feature'] = pd.cut(data['numerical_feature'], bins=3,
labels=['low', 'medium', 'high'])
```

➢ **New Feature Generation:** Create a new feature by combining existing features.

Technique – 11

```
# Technique-11: Create a new feature by combining existing features
if('new_feature' in data):
  data['new_feature'] = data['feature1'] * data['feature2']
```

➢ **One-Hot Encoding**: Convert categorical variables into numerical format using one-hot encoding.

Technique – 12

```
# Technique-12: One-Hot Encoding (Convert Categorical Variables)
type_new = pd.get_dummies(data_cleaned['type'], drop_first=True)
data_new = pd.concat([data_cleaned, type_new], axis=1)
data_new.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	CASH_OUT	DEBIT	PAYMENT	TRANSFER
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0.0	False	False	True	False
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0.0	False	False	True	False
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0	1.0	False	False	False	True
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0	1.0	True	False	False	False
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0.0	False	False	True	False

❖ Descriptive Statistical Analysis

Descriptive statistical analysis involves summarizing and describing the main characteristics of a dataset. It provides insights into the central tendency, spreadness, and shape of the data distribution. Here's a brief overview of the main components of descriptive statistical analysis:

1. **Measures of Central Tendency**:
   - ➢ These measures indicate where the center of the data distribution lies.
   - ➢ The most commonly used measures of central tendency are the mean, median, and mode.
   - ➢ The mean (average) is the sum of all values divided by the total number of values.
   - ➢ The median is the middle value of a sorted dataset.
   - ➢ The mode is the value that appears most frequently in the dataset.

2. **Measures of Spreadness (Dispersion)**:
   - ➢ These measures describe how spread out the data values are from the center.
   - ➢ Common measures of spreadness include the range, variance, standard deviation, and interquartile range (IQR).
   - ➢ The range is the difference between the maximum and minimum values.
   - ➢ The variance is the average of the squared differences from the mean.
   - ➢ The standard deviation is the square root of the variance and provides a measure of the average deviation from the mean.
   - ➢ The interquartile range (IQR) is the difference between the third quartile (Q3) and the first quartile (Q1).

3. **Skewness and Kurtosis**:
   - ➢ Skewness measures the asymmetry of the data distribution.
   - ➢ A positive skewness indicates that the right tail of the distribution is longer, while a negative skewness indicates a longer left tail.
   - ➢ Kurtosis measures the heaviness of the tails and the peakedness of the distribution.
   - ➢ A higher kurtosis value indicates heavier tails and a sharper peak, while a lower value indicates lighter tails and a flatter peak.

4. **Normality Tests**:
   - ➢ Normality tests, such as the Shapiro-Wilk test, assess whether the data follows a normal distribution.
   - ➢ A normal distribution is symmetrical and bell-shaped, with the mean, median, and mode all equal.
   - ➢ Deviations from normality may impact the validity of certain statistical analyses and models.

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42271 entries, 0 to 42270
Data columns (total 10 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   step            42271 non-null  int64  
 1   type            42271 non-null  object 
 2   amount          42271 non-null  float64
 3   nameOrig        42271 non-null  object 
 4   oldbalanceOrg   42271 non-null  float64
 5   newbalanceOrig  42271 non-null  float64
 6   nameDest        42271 non-null  object
 7   oldbalanceDest  42271 non-null  float64
 8   newbalanceDest  42271 non-null  float64
 9   isFraud         42271 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 3.2+ MB
```

```
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:", len(object_cols))

int_ = (data.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:", len(num_cols))

fl = (data.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:", len(fl_cols))
```

```
Categorical variables: 3
Integer variables: 1
Float variables: 6
```

```
data.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
count	42271.000000	4.227100e+04	4.227100e+04	4.227100e+04	4.227100e+04	4.227100e+04	42271.000000
mean	7.171087	1.490108e+05	7.727734e+05	7.874065e+05	8.450782e+05	1.158377e+06	0.002224
std	2.118037	3.145005e+05	2.284003e+06	2.321941e+06	2.466717e+06	2.960478e+06	0.047105
min	1.000000	1.770000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
25%	7.000000	7.031875e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
50%	8.000000	2.751283e+04	1.856733e+04	3.339900e+02	0.000000e+00	0.000000e+00	0.000000
75%	8.000000	1.794321e+05	1.421965e+05	1.500335e+05	4.337929e+05	8.013660e+05	0.000000
max	9.000000	1.000000e+07	2.854724e+07	2.861740e+07	2.958454e+07	3.130692e+07	1.000000

```
# Calculate mean for a specific column
mean = data['step'].mean()
print("Mean:", mean)

# Calculate median for a specific column
median = data['amount'].median()
print("Median:", median)

# Calculate mode for a specific column
mode = data['step'].mode()[0]
print("Mode:", mode)
```

```
Mean: 7.171086560526129
Median: 27512.83
Mode: 8
```

```
# Calculate skewness for a specific column
skewness = data['isFraud'].skew()
print("Skewness:", skewness)

# Calculate kurtosis for a specific column
kurtosis = data['isFraud'].kurtosis()
print("Kurtosis:", kurtosis)
```

```
Skewness: 21.13587997232341
Kurtosis: 444.74646483028175
```

Summary:
descriptive statistical analysis involves summarizing data using various measures and techniques to gain insights into its characteristics and distribution. These insights are essential for understanding the dataset and informing subsequent analyses and modeling decisions.

❖ **Visual Analysis**

Visual analysis is an essential part of data exploration and involves using graphical representations to gain insights into the dataset. Visualizations provide a way to understand patterns, trends, relationships, and distributions within the data. Here are some common types of visualizations used in data analysis:

1. **Histograms**: Histograms are used to visualize the distribution of a single numerical variable. They display the frequency or count of data points within different bins or intervals.

2. **Box Plots (Box-and-Whisker Plots)**: Box plots are used to visualize the distribution, spread, and skewness of numerical data. They show the median, quartiles, and potential outliers in the data.

3. **Scatter Plots**: Scatter plots are used to visualize the relationship between two numerical variables. Each data point is represented as a dot on the plot, with one variable on the x-axis and the other on the y-axis.

4. **Line Plots**: Line plots are used to visualize trends over time or sequential data. They are commonly used for time series analysis or to show changes in a variable over a continuous range.

5. **Bar Plots**: Bar plots are used to visualize the distribution of categorical variables or to compare different categories. They display the frequency or count of each category as bars.

6. **Pie Charts**: Pie charts are used to visualize the relative proportions or percentages of different categories within a dataset. Each category is represented as a slice of the pie, with the size of the slice proportional to its percentage of the whole.

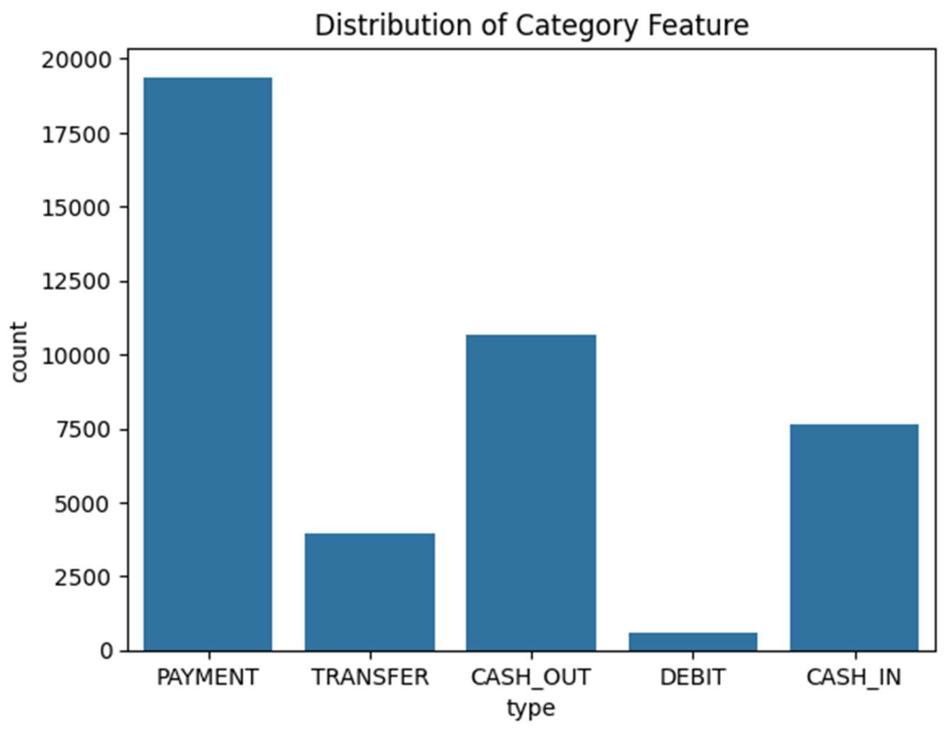
7. **Heatmaps**: Heatmaps are used to visualize the correlation between variables in a dataset. They display a matrix of colors, with darker colors indicating stronger correlations.

8. **Pair Plots**: Pair plots are used to visualize pairwise relationships between numerical variables in a dataset. They show scatter plots for each pair of variables along with histograms for each variable on the diagonal.

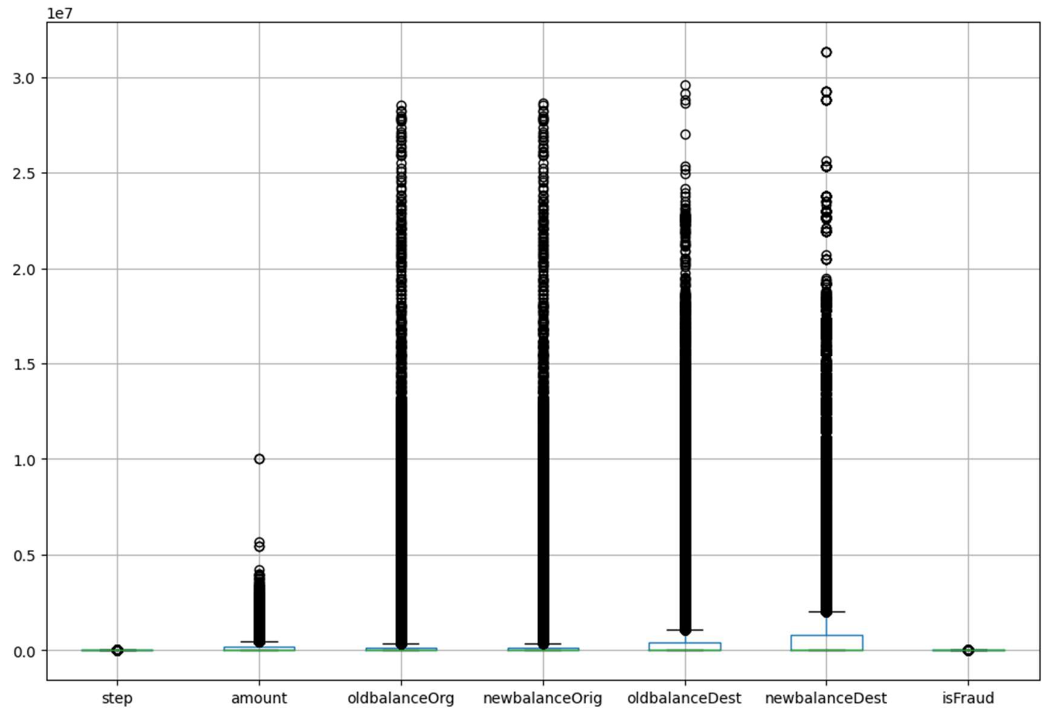
9. **Violin Plots**: Violin plots combine the features of box plots and kernel density plots to visualize the distribution of numerical data. They provide insights into the spread and shape of the distribution.

10. **Correlation Matrix**: A correlation matrix is a heatmap that visualizes the correlation coefficients between all pairs of numerical variables in a dataset. It helps identify patterns and relationships between variables.

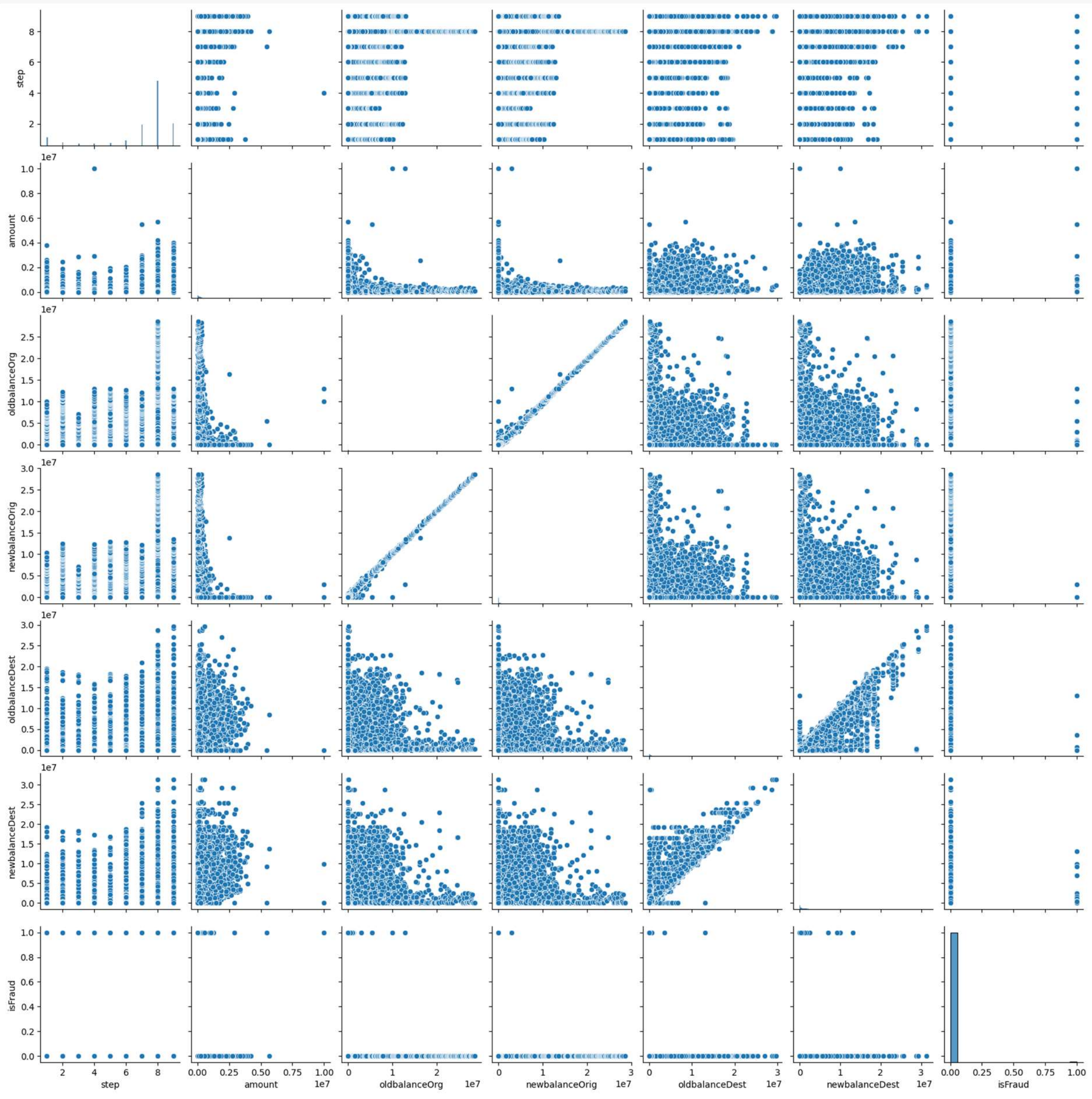
```
# Count plots to visualize the distribution of categorical features
sns.countplot(x='type', data=data)
plt.title('Distribution of Category Feature')
plt.show()
```



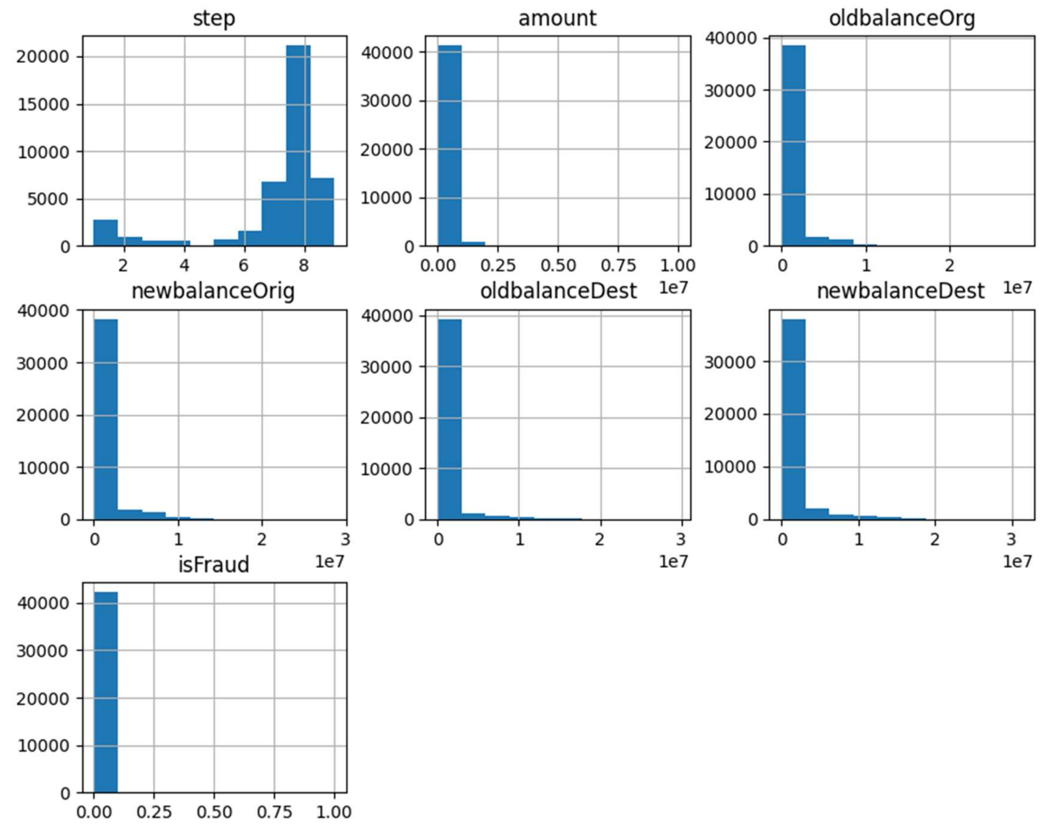
```
# Box plots to detect outliers in numerical features
plt.figure(figsize=(12, 8))
data.boxplot()
plt.show()
```



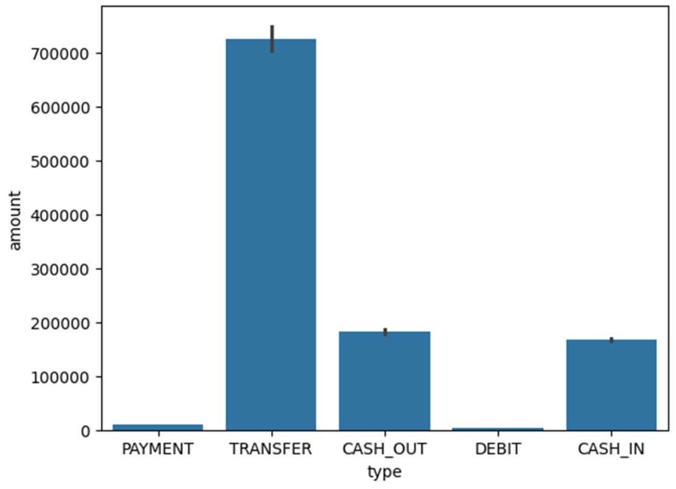
```
# Pairplot to visualize pairwise relationships between numerical
variables
sns.pairplot(data)
plt.show()
```



```
# Histograms to visualize the distribution of numerical features
data.hist(figsize=(10, 8))
plt.show()
```



```
# Bar plot
sns.barplot(x='type', y='amount', data=data)
<Axes: xlabel='type', ylabel='amount'>
```

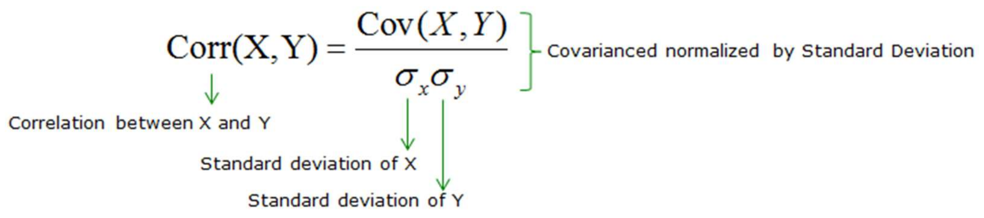


Summary:
By using these visualizations, analysts can explore the dataset, identify patterns and trends, detect outliers, and make informed decisions for further analysis. Each type of visualization has its strengths and is suitable for different types of data and analysis goals.

❖ <u>**Feature Extraction based on correlation, covariance**</u>

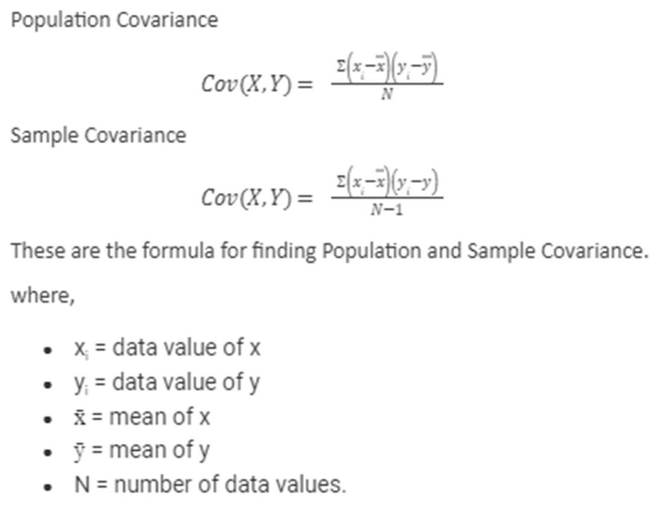
Feature extraction based on correlation and covariance is a technique used in data analysis and machine learning to identify and select relevant features from a dataset. This process involves analyzing the relationships between features (variables) to determine their importance or contribution to the target variable or overall dataset.

1. **Correlation**: Correlation measures the strength and direction of the linear relationship between two variables. The correlation coefficient, typically denoted by *ρ* (rho) or *r*, ranges from -1 to 1.
   - A correlation coefficient of 1 indicates a perfect positive linear relationship, meaning that as one variable increases, the other variable also increases proportionally.
   - A correlation coefficient of -1 indicates a perfect negative linear relationship, meaning that as one variable increases, the other variable decreases proportionally.
   - A correlation coefficient of 0 indicates no linear relationship between the variables.
   - Formula:


$\mathrm{Corr(X,Y)} = \dfrac{\mathrm{Cov}(X,Y)}{\sigma_x \sigma_y}$
Covarianced normalized by Standard Deviation
Correlation between X and Y
Standard deviation of X
Standard deviation of Y

Feature extraction based on correlation involves identifying features that are highly correlated with the target variable or with each other. Highly correlated features may contain redundant information and can negatively impact the performance of some machine learning algorithms. Therefore, it's common to select only one feature from a highly correlated group or to use techniques such as principal component analysis (PCA) to reduce dimensionality.

2. **Covariance**: Covariance measures the degree to which two variables vary together. It is similar to correlation but does not standardize the values, so it is affected by the scale of the variables. The covariance between two variables *X* and *Y* is calculated as:



Population Covariance

$$Cov(X,Y) = \frac{\Sigma(x_i - \bar{x})(y_i - \bar{y})}{N}$$

Sample Covariance

$$Cov(X,Y) = \frac{\Sigma(x_i - \bar{x})(y_i - y)}{N-1}$$

These are the formula for finding Population and Sample Covariance.

where,

- x_i = data value of x
- y_i = data value of y
- $\bar{x}$ = mean of x
- $\bar{y}$ = mean of y
- N = number of data values.

Feature extraction based on covariance involves identifying features with high covariance with the target variable. However, covariance alone does not provide information about the strength or direction of the relationship between variables. Therefore, it's often used in conjunction with correlation analysis.

```
# Select numeric columns
numeric_data = data_new.select_dtypes(include=['number'])

# Identify highly correlated features
correlation_matrix = numeric_data.corr().abs()
high_corr_var=np.where(correlation_matrix>0.8)
high_corr_var=[(correlation_matrix.columns[x],correlation_matrix.column
s[y]) for x,y in zip(*high_corr_var) if x!=y and x<y]
print(high_corr_var)
```

```
[('oldbalanceOrg',          'newbalanceOrig'),          ('oldbalanceDest',
'newbalanceDest')]
```

```
X = data_new.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = data_new['isFraud']

X.shape, y.shape
```

((42271, 10), (42271,))

Summary:

eature extraction based on correlation and covariance helps reduce dimensionality, improve model performance, and enhance interpretability by selecting the most informative features for analysis or modeling. It's important to carefully consider the relationships between variables and the specific requirements of the analysis or model when performing feature extraction.

❖ Models Creation

Creating machine learning models in Python for data science involves several steps, including model selection, training, testing and evaluation. Here's a general outline of how you can create machine learning models in Python for data science tasks:

1. **Model Selection**:
   - Choose an appropriate model based on the nature of the problem (classification, regression, clustering, etc.) and the characteristics of the data.
   - Consider factors such as interpretability, scalability, and performance requirements.

2. **Split Data**:
   - Split the dataset into training and testing sets to evaluate the performance of the model on unseen data.
   - Optionally, perform cross-validation for more robust evaluation.

3. **Train Model**:
   - Train the selected model on the training data using the fit() method.
   - Provide input features and corresponding target labels (for supervised learning tasks).

1. **Make Predictions**:
   - Use the trained model to make predictions on new data.
   - Provide input features and obtain predicted target labels or values.

```
# Import train_test_split function
from sklearn.model_selection import train_test_split

# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((29589, 10), (12682, 10), (29589,), (12682,))
```

```
#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics
```

Linear Regression, AdaBoostClassifier, and RandomForestClassifier are three popular machine learning models used for both data science and machine learning tasks. Here's a brief overview of each model along with its theory:

Linear Regression:

- **Theory**: Linear regression is a simple and widely-used supervised learning algorithm used for predicting a continuous target variable based on one or more input features. It assumes a linear relationship between the input features and the target variable, represented by the equation:

$y = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n + \epsilon$

where:

- y is the target variable,
- $x_1, x_2, ..., x_n$ are the input features,
- $w_0, w_1, w_2, ..., w_n$ are the coefficients (weights) to be learned,
- ϵ represents the error term.

- **Usage**: Linear regression is commonly used for tasks such as predicting house prices, stock prices, sales forecasting, and other regression problems where the relationship between the input features and the target variable is assumed to be linear.

Model - 1 : LinearRegression

```
from sklearn.linear_model import LinearRegression

# Create Linear Regression object
lr = LinearRegression()

# Train Adaboost Classifer
model_lr = lr.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_lr = model_lr.predict(X_test)

# Model Accuracy
print("Accuracy of LinearRegression
model:",metrics.accuracy_score(y_test, y_pred_lr.astype(int)))
```

```
Accuracy of LinearRegression model: 0.9971613310203438
```

AdaBoostClassifier:

- **Theory**: AdaBoost (Adaptive Boosting) is an ensemble learning method that combines multiple weak learners (typically decision trees) to create a strong classifier. It iteratively trains weak learners on the dataset, giving more weight to misclassified samples in each iteration. The final classifier is a weighted sum of the weak learners' predictions, where the weights are determined based on their performance.
- **Usage**: AdaBoost is commonly used for classification tasks, especially when dealing with imbalanced datasets or when high accuracy is required. It is effective in improving the performance of weak learners and reducing bias and variance.

Model - 2 : AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier

# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50)

# Train Adaboost Classifer
model_abc = abc.fit(X_train, y_train)

#Predict the response for test dataset
```

```
y_pred_abc = model_abc.predict(X_test)

# Model Accuracy
print("Accuracy of AdaBoostClassifier
model:",metrics.accuracy_score(y_test, y_pred_abc))
Accuracy of AdaBoostClassifier model: 0.998028702097461
```

RandomForestClassifier:

- **Theory**: RandomForest is another ensemble learning method that consists of a collection of decision trees. Each tree in the forest is trained independently on a random subset of the training data and a random subset of the features. The final prediction is made by aggregating the predictions of all the trees (classification via voting or averaging for regression).
- **Usage**: RandomForest is widely used for both classification and regression tasks. It is robust to overfitting, handles high-dimensional data well, and provides feature importance scores, making it useful for understanding the importance of different features in the dataset.

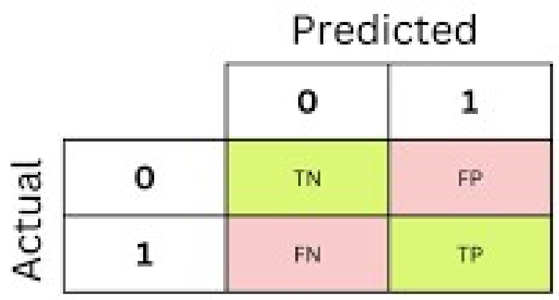
Model - 3 : RandomForestClassifier

```
from sklearn.ensemble import RandomForestClassifier

# Create RandomForest classifer object
rfc = RandomForestClassifier(n_estimators=60)

# Train RandomForest Classifer
model_rfc = rfc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_rfc = model_rfc.predict(X_test)

# Model Accuracy
print("Accuracy of RandomForestClassifier
model:",metrics.accuracy_score(y_test, y_pred_rfc))
Accuracy of RandomForestClassifier model: 0.9979498501813594
```

❖ Evaluation of the model based on various metrics

Performance Metrics: Calculate various performance metrics to evaluate the model's performance. Common metrics include:

- **Accuracy**: The proportion of correctly classified instances.
- **Precision**: The proportion of true positive predictions among all positive predictions.
- **Recall**: The proportion of true positive predictions among all actual positive instances.
- **F1-score**: The harmonic mean of precision and recall, providing a balance between the two.
- **Confusion Matrix**: A table showing the counts of true positives, true negatives, false positives, and false negatives.



$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix

# Assuming y_true are the true labels and y_pred are the predicted
labels

def evaluate_model(y_pred):

  # Calculate accuracy
  accuracy = accuracy_score(y_test, y_pred)
  print("Accuracy:\t", accuracy)

  # Calculate precision
  precision = precision_score(y_test, y_pred)
  print("Precision:\t", precision)

  # Calculate recall
  recall = recall_score(y_test, y_pred)
  print("Recall:\t\t", recall)

  # Calculate F1-score
  f1 = f1_score(y_test, y_pred)
  print("F1-score:\t", f1)

  # Calculate confusion matrix
  conf_matrix = confusion_matrix(y_test, y_pred)
  print("Confusion Matrix:")
  print(conf_matrix)
```

```
# Evaluate Linear Regression model
print("Linear Regression Model Performance:")
evaluate_model(y_pred_lr.astype(int))
```

```
Linear Regression Model Performance:
Accuracy:    0.9971613310203438
Precision:   0.0
Recall:            0.0
F1-score:    0.0
Confusion Matrix:
[[12646     0]
 [   36     0]]
```

```
# Evaluate AdaBoostClassifier model
print("AdaBoostClassifier Model Performance:")
evaluate_model(y_pred_abc)
```

```
AdaBoostClassifier Model Performance:
Accuracy:    0.998028702097461
Precision:   0.72
Recall:            0.5
F1-score:    0.5901639344262295
Confusion Matrix:
[[12639     7]
 [   18    18]]
```

```
# Evaluate RandomForestClassifier model
print("RandomForestClassifier Model Performance:")
evaluate_model(y_pred_rfc)
```

```
RandomForestClassifier Model Performance:
Accuracy:    0.9979498501813594
Precision:   0.9166666666666666
Recall:            0.3055555555555556
F1-score:    0.4583333333333333
Confusion Matrix:
[[12645     1]
 [   25    11]]
```

❖ Final Selection of the model

When selecting a model based on evaluation metrics such as accuracy, precision, recall, and F1-score, it's important to consider the specific requirements and characteristics of your problem. Here's a general approach for selecting a model based on these metrics:

Define Evaluation Criteria: Determine which evaluation metric(s) are most important for your problem. For example:

- If you want to prioritize correctly identifying positive cases (e.g., detecting fraud), focus on metrics like precision and recall.
- If you want to balance the trade-off between precision and recall, consider using the F1-score.
- If you want a general measure of overall correctness, accuracy can be used.

Select model based on Accuracy of the Model:

```
✓ Linear Regression Model Performance:
    o Accuracy:   0.9971613310203438

✓ AdaBoostClassifier Model Performance:
    o Accuracy:   0.998028702097461

✓ RandomForestClassifier Model Performance:
    o Accuracy:   0.9979498501813594
```

Here, Accuracy of the AdaBoostClassifier Model is more out of all so that AdaBoostClassifier Model is select as the best model for given dataset and problem.

Other criteria for model selection are .....

In Python, evaluating machine learning models based on errors involves analyzing the discrepancies between the model's predictions and the actual target values. This process helps assess the model's performance and identify areas for improvement. Here are some common techniques for evaluating models based on errors in Python:

1. **Mean Absolute Error (MAE)**:
    - Computes the average absolute difference between the predicted and actual values.
    - Lower values indicate better performance.

```
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_true, y_pred)
```

2. **Mean Squared Error (MSE)**:
    - Computes the average of the squared differences between the predicted and actual values.
    - Lower values indicate better performance.

```
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_true, y_pred)
```

3. **Root Mean Squared Error (RMSE)**:
    - Computes the square root of the MSE.
    - Lower values indicate better performance.

```
import numpy as np
rmse = np.sqrt(mse)
```

4. **Mean Absolute Percentage Error (MAPE)**:
    - Computes the mean of the absolute percentage differences between the predicted and actual values.
    - Lower values indicate better performance.

```
def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100

mape = mean_absolute_percentage_error(y_true, y_pred)
```

5. **R-squared (R²)**:
    - Measures the proportion of the variance in the dependent variable that is predictable from the independent variables.
    - R² score ranges from 0 to 1, where 1 indicates a perfect fit and 0 indicates no improvement over the mean.
    - Higher values indicate better performance.

```
from sklearn.metrics import r2_score
r2 = r2_score(y_true, y_pred)
```

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum (y_i - \hat{y})^2}{\sum (y_i - \bar{y})^2}$$

Where,

$\hat{y}$ − *predicted value of y*

$\bar{y}$ − *mean value of y*

Summary:

These are some common error metrics used to evaluate machine learning models in Python. Depending on the specific problem and context, you may choose one or more of these metrics to assess the performance of your models.

❖ Storytelling

In this era of Cyber Fraud Detection in Online Payments, brave warriors armed themselves with the tools of data science and machine learning to protect the realm from these nefarious deeds. Their quest was noble: to safeguard the integrity of online transactions and preserve the trust of countless individuals and organizations.

Our journey begins with a dataset. This dataset held within it the secrets of fraudulent transactions, encoded in rows and columns, waiting to be unearthed by the vigilant eyes of data scientists and analysts.As we delved into the depths of this data, we encountered a myriad of features—timestamps marking the passage of time, transaction amounts revealing financial exchanges, and categorical variables hinting at transaction types and origins. Each data point held a story, a clue waiting to be deciphered in the quest for truth.

With the wisdom of our predecessors and the power of modern technology, we embarked on a grand adventure of exploration and discovery. We plotted histograms, box plots and revealing patterns and anomalies hidden within the data. We calculated means, medians, and standard deviations, seeking insights into centrality, spreadness and normalcy.

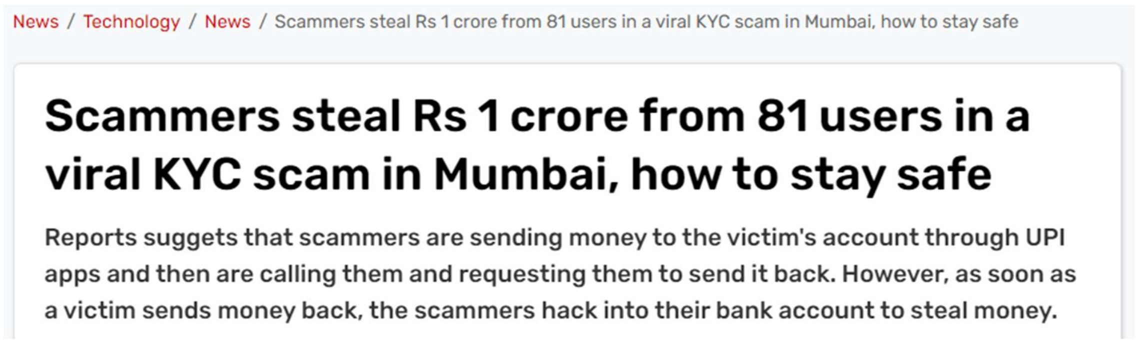
We trained Linear Regression models to predict transaction amounts, Logistic Regression models to classify fraud, and ensemble methods like Random Forests and AdaBoost to harness the collective wisdom of decision trees.

With each model trained, we entered the realm of evaluation, where metrics such as accuracy, precision, recall, and F1-score served as our guiding stars. We navigated the landscape of errors—mean absolute errors, mean squared errors, and root mean squared errors—seeking to minimize the chasm between prediction and reality.

But our journey did not end there, for the path of a data scientist is one of continuous learning and refinement. We iterated, experimented, and fine-tuned our models, striving to achieve ever greater levels of performance and efficacy.

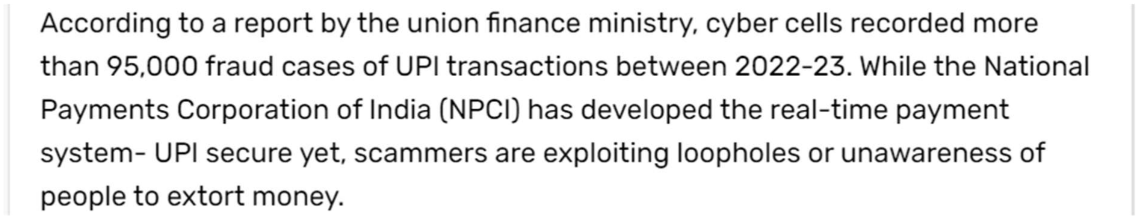
Conclusion:
Our story concludes not with a final victory, but with a steadfast commitment—to remain vigilant guardians of the digital realm, to adapt and evolve in the face of new threats, and to uphold the principles of integrity and trust in the world of online commerce. For in the ongoing saga of Cyber Fraud Detection in Online Payments, our quest for justice and security knows no end.

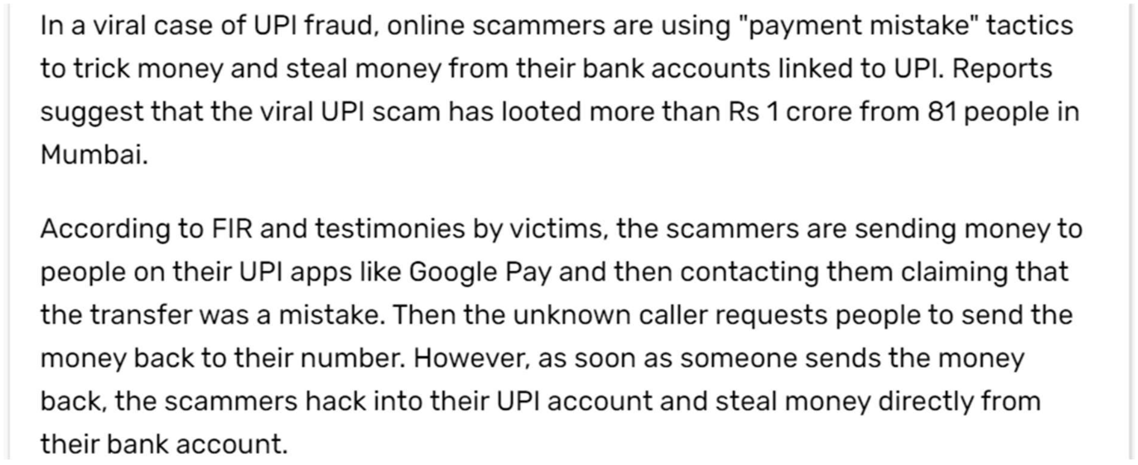
❖ <u>Real Life Example</u>


News / Technology / News / Scammers steal Rs 1 crore from 81 users in a viral KYC scam in Mumbai, how to stay safe

Scammers steal Rs 1 crore from 81 users in a viral KYC scam in Mumbai, how to stay safe

Reports suggets that scammers are sending money to the victim's account through UPI apps and then are calling them and requesting them to send it back. However, as soon as a victim sends money back, the scammers hack into their bank account to steal money.


According to a report by the union finance ministry, cyber cells recorded more than 95,000 fraud cases of UPI transactions between 2022-23. While the National Payments Corporation of India (NPCI) has developed the real-time payment system- UPI secure yet, scammers are exploiting loopholes or unawareness of people to extort money.


In a viral case of UPI fraud, online scammers are using "payment mistake" tactics to trick money and steal money from their bank accounts linked to UPI. Reports suggest that the viral UPI scam has looted more than Rs 1 crore from 81 people in Mumbai.

According to FIR and testimonies by victims, the scammers are sending money to people on their UPI apps like Google Pay and then contacting them claiming that the transfer was a mistake. Then the unknown caller requests people to send the money back to their number. However, as soon as someone sends the money back, the scammers hack into their UPI account and steal money directly from their bank account.

In Short:

- Scammers are using a new "payment mistake" tactic to steal money.
- They are calling UPI users to return their money back which they have sent to them my mistake.
- In Mumbai, the viral KYC Scam has made 81 users loose around Rs 1 crore.

News Link:
https://www.indiatoday.in/technology/news/story/scammers-steal-rs-1-crore-from-81-users-in-mumbai-while-making-upi-payment-how-to-stay-safe-2351854-2023-03-27

❖ <u>**Complete Code**</u>

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

#Load the dataset

```
data = pd.read_csv('onlinefraud.csv')
data.head()
data.info()
data.isna().sum()
```

#1. At Least 10 techniques of the Data preprocessing

Technique - 1

```
# Technique-1: Remove duplicate rows
data.drop_duplicates(inplace=True)
```

Technique - 2

```
# Find rows with null values
rows_with_null = data[data.isnull().any(axis=1)]

# Display rows with null values
print("Rows with null values:")
rows_with_null

data.columns
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')

# Technique-2: Replace missing values with median
for attr in data.columns:
  if((data[attr].dtype=='int64' or data[attr].dtype=='float64') and data[attr].isna().sum()):
    data[attr].fillna(data[attr].median(), inplace=True)

# Find rows with null values
rows_with_null = data[data.isnull().any(axis=1)]

# Display rows with null values
print("Rows with null values:")
rows_with_null
```

Technique - 3

```
# Technique-3: Replace missing values with mean
for attr in data.columns:
  if((data[attr].dtype=='int64' or data[attr].dtype=='float64') and data[attr].isna().sum()):
    data[attr].fillna(data[attr].mean(), inplace=True)
```

Technique - 4

```
# Technique-4: Replace missing values with mode
for attr in data.columns:
  if((data[attr].dtype=='int64' or data[attr].dtype=='float64') and data[attr].isna().sum()):
    data[attr].fillna(data[attr].mode(), inplace=True)
```

Technique - 5

```
# Technique-5: Remove irrelevent column
if('isFlaggedFraud' in data):
  data.drop('isFlaggedFraud', axis=1, inplace=True)
  print("'isFlaggedFraud' column dropped suuccessfully from dataset.")
'isFlaggedFraud' column dropped suuccessfully from dataset.
```

Technique - 6

```
# Technique-6: Remove rows with null values
data_cleaned = data.dropna()

# Find rows with null values
rows_with_null = data_cleaned[data_cleaned.isnull().any(axis=1)]

# Display rows with null values
print("Rows with null values:")
rows_with_null
```

Technique - 7

```
# Technique-7: Logarithmic transformation
import numpy as np
if('log_feature' in data):
  data['log_feature'] = np.log(data['feature'])
```

Technique - 8

```
# Technique-8: Min-Max Normalization
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Load the dataset
df_minmax = pd.read_csv('onlinefraud.csv')

# Extract numerical features
numerical_features = df_minmax.select_dtypes(include=['int', 'float']).columns
```

Technique - 9

```
# Technique-9: z-score Normalization
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
df_zscore = pd.read_csv('onlinefraud.csv')

# Extract numerical columns for normalization
numerical_columns = df_zscore.select_dtypes(include=['int64', 'float64']).columns
```

```
# Apply z-score normalization
scaler = StandardScaler()
df_zscore[numerical_columns] = scaler.fit_transform(df_zscore[numerical_columns])
```

Technique - 10

```
# Technique-10: Binning (Discretization of Continuous Variables)
# Bin numerical feature into categories

if('numerical_feature' in data):
  data['binned_feature'] = pd.cut(data['numerical_feature'], bins=3, labels=['low', 'medium', 'high'])
```

Technique - 11

```
# Technique-11: Create a new feature by combining existing features
if('new_feature' in data):
  data['new_feature'] = data['feature1'] * data['feature2']
```

Technique - 12

```
# Technique-12: One-Hot Encoding (Convert Categorical Variables)
type_new = pd.get_dummies(data_cleaned['type'], drop_first=True)
data_new = pd.concat([data_cleaned, type_new], axis=1)
data_new.head()
```

#2. Descriptive Statistical Analysis

```
data.info()
obj = (data.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:", len(object_cols))

int_ = (data.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:", len(num_cols))

fl = (data.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:", len(fl_cols))

data.describe()

# Calculate mean for a specific column
mean = data['step'].mean()
print("Mean:", mean)

# Calculate median for a specific column
median = data['amount'].median()
print("Median:", median)

# Calculate mode for a specific column
mode = data['step'].mode()[0]

# Calculate skewness for a specific column
```

```
skewness = data['isFraud'].skew()
print("Skewness:", skewness)

# Calculate kurtosis for a specific column
kurtosis = data['isFraud'].kurtosis()
print("Kurtosis:", kurtosis)
```

#3. Visual Analysis

```
# Count plots to visualize the distribution of categorical features
sns.countplot(x='type', data=data)
plt.title('Distribution of Category Feature')
plt.show()

# Box plots to detect outliers in numerical features
plt.figure(figsize=(12, 8))
data.boxplot()
plt.show()

# Pairplot to visualize pairwise relationships between numerical variables
sns.pairplot(data)
plt.show()

# Histograms to visualize the distribution of numerical features
data.hist(figsize=(10, 8))
plt.show()

# Bar plot
sns.barplot(x='type', y='amount', data=data)
```

#4. Feature Extraction based on correlation, covariance

```
# Select numeric columns
numeric_data = data_new.select_dtypes(include=['number'])

# Identify highly correlated features
correlation_matrix = numeric_data.corr().abs()
high_corr_var=np.where(correlation_matrix>0.8)
high_corr_var=[(correlation_matrix.columns[x],correlation_matrix.columns[y]) for x,y in
zip(*high_corr_var) if x!=y and x<y]
print(high_corr_var)

X = data_new.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = data_new['isFraud']
X.shape, y.shape
```

#5. Models Creation (At Least two)

```
# Import train_test_split function
from sklearn.model_selection import train_test_split
```

```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape

#Import scikit-learn metrics module for accuracy calculation
from sklearn import metrics

#Model - 1 : LinearRegression
from sklearn.linear_model import LinearRegression

# Create Linear Regression object
lr = LinearRegression()

# Train Adaboost Classifer
model_lr = lr.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_lr = model_lr.predict(X_test)

# Model Accuracy
print("Accuracy of LinearRegression model:",metrics.accuracy_score(y_test, y_pred_lr.astype(int)))

#Model - 2 : AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier

# Create adaboost classifer object
abc = AdaBoostClassifier(n_estimators=50)

# Train Adaboost Classifer
model_abc = abc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_abc = model_abc.predict(X_test)

# Model Accuracy
print("Accuracy of AdaBoostClassifier model:",metrics.accuracy_score(y_test, y_pred_abc))

#Model - 3 : RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

# Create RandomForest classifer object
rfc = RandomForestClassifier(n_estimators=60)

# Train RandomForest Classifer
model_rfc = rfc.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_rfc = model_rfc.predict(X_test)

# Model Accuracy
```

```
print("Accuracy of RandomForestClassifier model:",metrics.accuracy_score(y_test, y_pred_rfc))
```

#6. Evaluation of the model based on various metrics

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix

# Assuming y_test are the true labels and y_pred are the predicted labels
def evaluate_model(y_pred):

  # Calculate accuracy
  accuracy = accuracy_score(y_test, y_pred)
  print("Accuracy:\t", accuracy)

  # Calculate precision
  precision = precision_score(y_test, y_pred)
  print("Precision:\t", precision)

  # Calculate recall
  recall = recall_score(y_test, y_pred)
  print("Recall:\t\t", recall)

  # Calculate F1-score
  f1 = f1_score(y_test, y_pred)
  print("F1-score:\t", f1)

  # Calculate confusion matrix
  conf_matrix = confusion_matrix(y_test, y_pred)
  print("Confusion Matrix:")
  print(conf_matrix)

# Evaluate Linear Regression model
print("Linear Regression Model Performance:")
evaluate_model(y_pred_lr.astype(int))

# Evaluate AdaBoostClassifier model
print("AdaBoostClassifier Model Performance:")
evaluate_model(y_pred_abc)

# Evaluate RandomForestClassifier model
print("RandomForestClassifier Model Performance:")
evaluate_model(y_pred_rfc)
```

#7. Final Selection of the model

```
#AdaBoostClassifier model is selected because Accuracy is more

from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_true, y_pred)

from sklearn.metrics import mean_squared_error
```

```
mse = mean_squared_error(y_true, y_pred)

import numpy as np
rmse = np.sqrt(mse)

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred) / y_true)) * 100
mape = mean_absolute_percentage_error(y_true, y_pred)

from sklearn.metrics import r2_score
r2 = r2_score(y_true, y_pred)
```