RAJALAKSHMI ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

AI19P71
Data Visualization Using Python

AI Powered Fraud Detection in Financial Transactions

Team Members

211501028 - Girish J R
211501030 - Gouri M P
211501031 - Harini B

Problem Statement

Fraud detection in financial transactions poses a major challenge for financial institutions as fraudulent activities cause significant financial losses and harm customer trust. Traditional systems struggle to keep up with evolving fraud tactics. The goal is to create an AI-driven solution that accurately identifies fraudulent transactions in real-time by leveraging machine learning algorithms to detect anomalies in large datasets. The system must minimize false positives to avoid disrupting legitimate users while being scalable and adaptive to new fraud patterns. Ensuring data security is crucial to protect sensitive information. The solution aims to improve financial system security, benefiting both institutions and consumers.

Dataset Source and Structure

Dataset Source : Kaggle - Synthetic Financial Datasets For Fraud Detection	
No of Features : 17	
No of Records : 10,127	

Dataset Feature Description

Feature	Description
step	An integer (int64) representing the step or sequence of the transaction within a session
type	A categorical variable (object) indicating the type of transaction (e.g., payment, transfer)
branch	The branch or location where the transaction was initiated (categorical)
amount	The monetary value (float64) of the transaction

Dataset Feature Description

Feature	Description
nameOrig	The origin account name (categorical), representing the account initiating the transaction
oldbalanceOrg	The balance of the origin account before the transaction (float64)
newbalanceOrig	The balance of the origin account after the transaction (float64)
nameDest	The destination account name (categorical), representing the account receiving the transaction

Dataset Feature Description

Feature	Description
oldbalanceDest	The balance of the destination account before the transaction (float64)
newbalanceDest	The balance of the destination account after the transaction (float64)
unusuallogin	A binary variable (int64) indicating whether the login is unusual
isFlaggedFraud	A binary variable (int64) indicating whether the transaction is flagged for possible fraud

Dataset Feature Description

Feature	Description
Acct type	The account type (categorical), such as checking or savings
Date of transaction	The date when the transaction occurred (categorical)
Time of day	The specific time the transaction took place (categorical)
isFraud	A binary target variable (float64) indicating whether the transaction is fraudulent (1) or not (0)

Data Acquisition and Cleaning

CODE

```
import pandas as pd
file_path = r"C:\Users\wonde\Downloads\Datasets.csv"
df = pd.read_csv(file_path)
df_info = df.info()
df_head = df.head()
df_info, df_head
print(df['isFlaggedFraud'].sum())
```

OUTPUT

```
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Unnamed: 0           10127 non-null  int64
 1   step                 10127 non-null  int64
 2   type                 10123 non-null  object
 3   branch               10127 non-null  object
 4   amount               10125 non-null  float64
 5   nameOrig             10121 non-null  object
 6   oldbalanceOrg        10125 non-null  float64
 7   newbalanceOrig       10127 non-null  float64
 8   nameDest             10121 non-null  object
 9   oldbalanceDest       10126 non-null  float64
 10  newbalanceDest       10125 non-null  float64
 11  unusuallogin         10127 non-null  int64
 12  isFlaggedFraud       10127 non-null  int64
 13  Acct type            10117 non-null  object
 14  Date of transaction  10120 non-null  object
 15  Time of day          10125 non-null  object
 16  isFraud              10125 non-null  float64
dtypes: float64(6), int64(4), object(7)
memory usage: 1.3+ MB
0
```

Data Acquisition and Cleaning

CODE

```
import pandas as pd
import numpy as np
print("Missing values in each column before handling:\n", df.isnull().sum())
numerical_cols = df.select_dtypes(include=[np.number]).columns
categorical_cols = df.select_dtypes(include=['object']).columns
for col in numerical_cols:
    if df[col].isnull().sum() > 0:
        median_value = df[col].median()
        df[col].fillna(median_value, inplace=True)
        print(f"Filled missing values in numerical column '{col}' with median value: {median_value}")
for col in categorical_cols:
    if df[col].isnull().sum() > 0:
        most_frequent = df[col].mode()[0]
        df[col].fillna(most_frequent, inplace=True)
        print(f"Filled missing values in categorical column '{col}' with most frequent value: '{most_frequent}'")
threshold = 0.5 * len(df)
df.dropna(thresh=threshold, axis=1, inplace=True)
df.dropna(thresh=len(df.columns) - 2, axis=0, inplace=True)
print("\nMissing values in each column after handling:\n", df.isnull().sum())
print("\nPreview of the dataset after handling missing values:\n", df.head())
print(df.isnull().sum())
```

OUTPUT

```
Missing values in each column before handling:
 Unnamed: 0              0
step                     0
type                     4
branch                   0
amount                   2
nameOrig                 6
oldbalanceOrg            2
newbalanceOrig           0
nameDest                 6
oldbalanceDest           1
newbalanceDest           2
unusuallogin             0
isFlaggedFraud           0
Acct type               10
Date of transaction      7
Time of day              2
isFraud                  2
dtype: int64
Filled missing values in numerical column 'amount' with median value: 12798.31
Filled missing values in numerical column 'oldbalanceOrg' with median value: 21363.0
Filled missing values in numerical column 'oldbalanceDest' with median value: 0.0
Filled missing values in numerical column 'newbalanceDest' with median value: 0.0
Filled missing values in numerical column 'isFraud' with median value: 0.0
Filled missing values in categorical column 'type' with most frequent value: 'PAYMENT'
...
Date of transaction    0
Time of day            0
isFraud                0
dtype: int64
```

Data Preprocessing
(I) Feature Engineering

CODE

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
df['log_amount'] = np.log(df['amount'] + 1)
df['balance_change_orig'] = df['oldbalanceOrg'] - df['newbalanceOrig']
df['balance_change_dest'] = df['oldbalanceDest'] - df['newbalanceDest']
df['Date of transaction 1'] = pd.to_datetime(df['Date of transaction'], format='%m/%d/%Y', errors='coerce')
df['day_of_week'] = df['Date of transaction 1'].dt.dayofweek
df.drop(columns=['Date of transaction 1'], inplace=True)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df[['oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']])
df[['scaled_oldbalanceOrg', 'scaled_newbalanceOrig', 'scaled_oldbalanceDest', 'scaled_newbalanceDest']] = scaled_features
df['amount_to_orig_balance'] = df['amount'] / (df['oldbalanceOrg'] + 1)
df['dest_balance_ratio'] = df['oldbalanceDest'] / (df['newbalanceDest'] + 1)
if 'Date of transaction' in df.columns:
    df['Date of transaction'] = pd.to_datetime(df['Date of transaction'], errors='coerce', dayfirst=True)
    df['transaction_year'] = df['Date of transaction'].dt.year
    df['transaction_month'] = df['Date of transaction'].dt.month
    df['transaction_day'] = df['Date of transaction'].dt.day
    df.drop(columns=['Date of transaction'], inplace=True)
df
```

OUTPUT

scaled_oldbalanceOrg	scaled_newbalanceOrig	scaled_oldbalanceDest	scaled_newbalanceDest	amount_to_orig_balance	dest_balance_ratio	transaction_year	transaction_month	transaction_day
-0.335828	-0.342963	-0.348597	-0.363849	0.057834	0.0	2018	1	3
-0.405916	-0.407904	-0.348597	-0.363849	0.087731	0.0	2018	1	5
-0.415834	-0.416838	-0.348597	-0.363849	0.994505	0.0	2018	1	7
-0.415834	-0.416838	-0.340686	-0.363849	0.994505	21182.0	2018	1	6
-0.396358	-0.403064	-0.348597	-0.363849	0.280788	0.0	2018	1	6
...	...	...	...	...	...	...	...	...
-0.400334	-0.401735	-0.348597	-0.363849	0.010194	0.0	2018	1	3
-0.400493	-0.404041	-0.348597	-0.363849	0.152685	0.0	2018	1	5
-0.392015	-0.398239	-0.348597	-0.363849	0.205291	0.0	2018	1	7
-0.415455	-0.416838	-0.348597	-0.363849	2.860780	0.0	2018	1	6
-0.412896	-0.416838	-0.348597	-0.363849	1.265054	0.0	2018	1	2

Data Preprocessing

EXPLANATION

Feature Engineering

Feature engineering involves creating new features or modifying existing ones to make the dataset more informative and suitable for machine learning models. The following new features are engineered:

1. **Log Transformation of Amount**: A new feature log_amount is created by applying a log transformation to the amount column (np.log(df['amount'] + 1)). This helps in normalizing highly skewed data.
2. **Balance Change Features**: The features balance_change_orig and balance_change_dest represent the change in balances for both the origin and destination accounts. They are calculated as the difference between the old and new balances.
3. **Day of the Week**: A new feature day_of_week is extracted from the Date of transaction column, indicating the day of the week on which the transaction occurred.
4. **Ratio Features:** New features amount_to_orig_balance and dest_balance_ratio are created to capture the relationship between the transaction amount and balances, which can indicate the significance of the transaction relative to account balances.

Data Preprocessing
(II) Label Encoding

CODE

```
from sklearn.preprocessing import LabelEncoder

columns_to_encode = ['type', 'branch', 'Acct type', 'day_of_week', 'nameOrig', 'nameDest', 'Time of day']

df_encoded = df.copy()

label_encoder = LabelEncoder()

for col in columns_to_encode:
    df_encoded[col] = label_encoder.fit_transform(df_encoded[col])

print(df_encoded)
import pandas as pd

null_values = df_encoded.isnull().sum()
print(df_encoded)
```

OUTPUT

```
       Unnamed: 0  step  type  branch     amount  nameOrig  oldbalanceOrg  \
0               0     1     3      55    9839.64      1174       170136.0
1               1     1     3      54    1864.28      3530        21249.0
2               2     1     4      54     181.00      1589          181.0
3               3     1     1       7     181.00      9281          181.0
4               4     1     3       7   11668.14      5543        41554.0
...           ...   ...   ...     ...        ...       ...            ...
10122       10122     7     3      31     337.50      2573        33107.0
10123       10123     7     3      78    5003.57      3348        32769.5
10124       10124     7     3      92   10424.89       131        50780.0
10125       10125     7     3      78    2823.59      6854          986.0
10126       10126     7     3      31    8126.71      3369         6423.0

       newbalanceOrig  nameDest  oldbalanceDest  ...  scaled_oldbalanceOrg  \
0           160296.36      3763             0.0  ...             -0.335828
1            19384.72      3966             0.0  ...             -0.405916
2                0.00       732             0.0  ...             -0.415834
3                0.00       636         21182.0  ...             -0.415834
4            29885.86      1596             0.0  ...             -0.396358
...               ...       ...             ...  ...                   ...
10122        32769.50      2123             0.0  ...             -0.400334
10123        27765.93      3391             0.0  ...             -0.400493
10124        40355.11      3388             0.0  ...             -0.392015
10125            0.00      4532             0.0  ...             -0.415455
10126            0.00      1778             0.0  ...             -0.412896
...
```

Data Preprocessing

EXPLANATION

Label Encoding

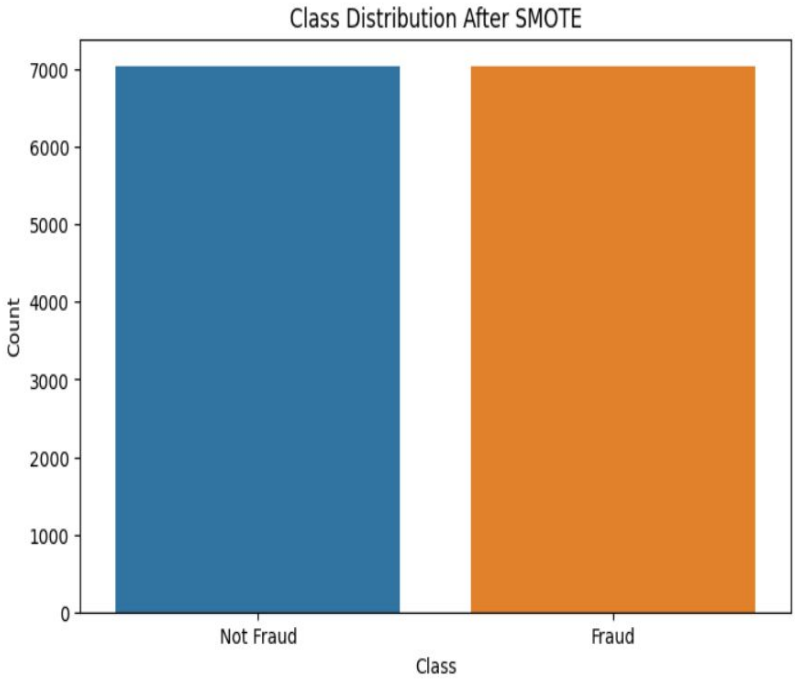
- Label encoding is a technique used to convert categorical text data into numerical values, making it suitable for machine learning models. In this process, the categorical columns such as type, branch, Acct type, day_of_week, nameOrig, nameDest, and Time of day are identified for encoding.
- A copy of the original DataFrame is created to preserve the original data. Using the LabelEncoder from the sklearn.preprocessing module, each categorical column is transformed into numerical values where each unique category is assigned a specific integer.
- The fit_transform() method is applied to each column, and the encoded values replace the original values in the DataFrame. After encoding, the dataset is checked for missing values to ensure no data integrity issues arise.
- The resulting df_encoded DataFrame contains the same structure as the original but with the categorical columns now represented as numerical values, making it ready for machine learning algorithms.

Data Preprocessing
(III) Data Balancing using SMOTE

CODE

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print(X_train_resampled)
plt.figure(figsize=(8, 5))
sns.countplot(x=y_train_resampled)
plt.title('Class Distribution After SMOTE')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1], labels=['Not Fraud', 'Fraud'])
plt.show()
print("Original training dataset shape:", y_train.value_counts())
print("Resampled training dataset shape:", pd.Series(y_train_resampled).value_counts())
import pandas as pd
# Combine the features and target into a single DataFrame
processed_dataset = pd.DataFrame(X_train_resampled, columns=X.columns)
processed_dataset['isFraud'] = y_train_resampled
# Save the DataFrame to a CSV file
processed_dataset.to_csv('processed_data.csv', index=False)
print("Processed data has been saved as 'processed_data.csv'")
```

OUTPUT



Data Preprocessing

EXPLANATION

Data Balancing using SMOTE

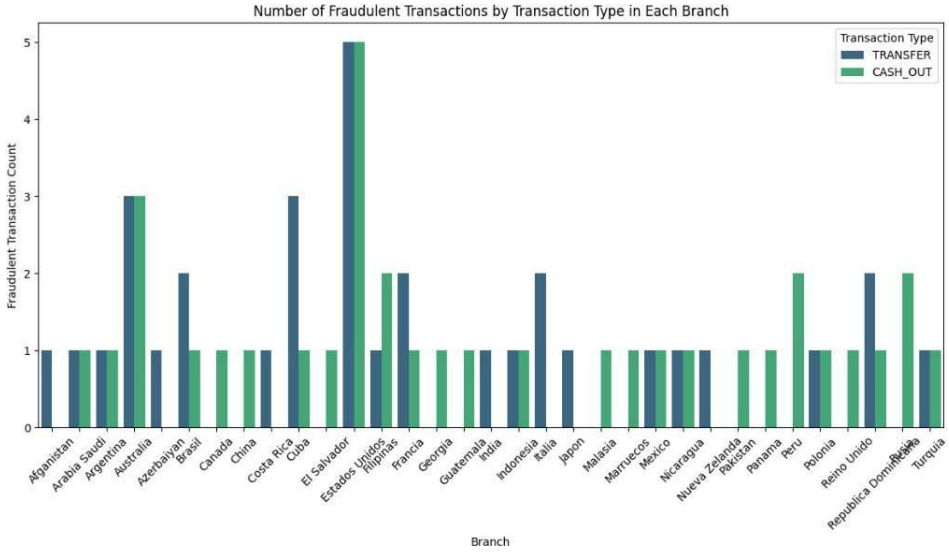
The Synthetic Minority Over-sampling Technique (SMOTE) is applied to the training dataset to balance the class distribution between fraudulent and non- fraudulent transactions. SMOTE generates synthetic samples of the minority class (fraudulent transactions) to prevent the model from being biased toward the majority class.

Class Imbalance Solved: SMOTE balances the minority and majority classes in the training set, which improves the model's ability to correctly identify minority class instances (e.g., fraud).

Training Dataset Enhanced: Synthetic data points are added only to the training dataset, leaving the testing dataset untouched to evaluate the model's performance realistically.

Data Preservation: The processed and balanced data is saved as a CSV file for reproducibility and easier integration into further steps.

EDA-INSIGHT-1-Number of Fraudulent Transactions by Transaction Type for Each Branch

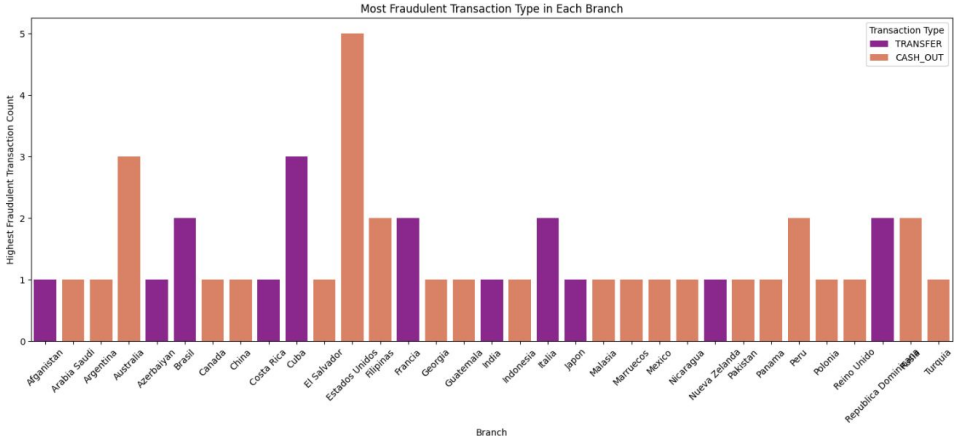


Inference: "TRANSFER" and "CASH_OUT," are more frequently associated with fraud in specific branches, suggesting patterns unique to those locations.

Observation: US, Cuba and Australia exhibits a higher count of fraudulent activities in specific transaction types, like "TRANSFER."

Recommendations: Implement targeted fraud prevention measures in branches with higher instances of fraud-prone transaction types and consider revising transaction monitoring protocols to identify suspicious patterns unique to those branches.

EDA-INSIGHT-2-Most Fraudulent Transaction Type in Each Branch

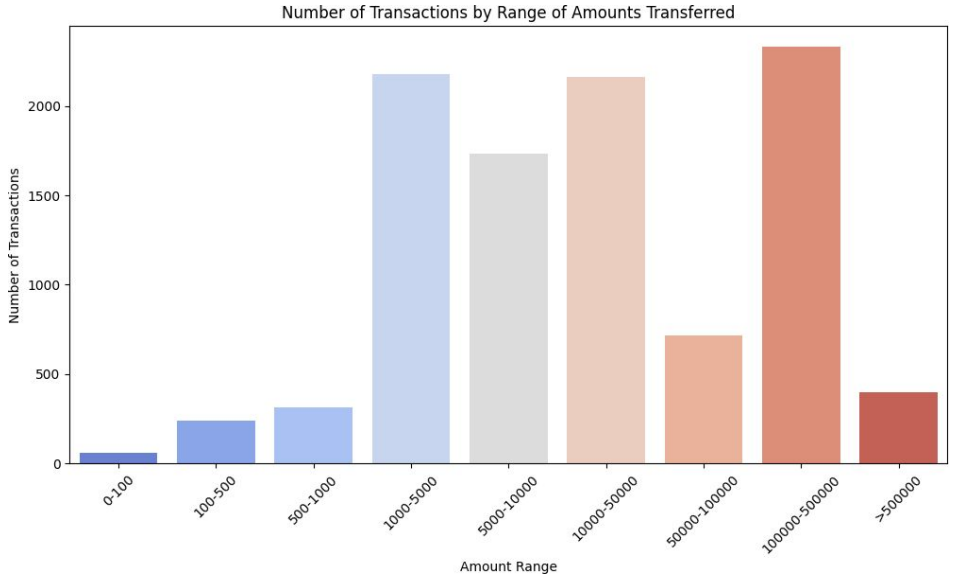


Inference: Each branch exhibits a predominant transaction type associated with the majority of fraudulent cases, such as "TRANSFER" or "CASH_OUT,"

Observation: branches have varied primary fraudulent transaction types, with US and Australia showing higher fraud incidents in "CASH_OUT" transactions, while Cuba has more fraudulent "TRANSFER" transactions.

Recommendations: Branches should strengthen fraud detection protocols specifically for the most common fraudulent transaction types they experience, and implement real-time monitoring to flag suspicious "TRANSFER" or "CASH_OUT" transactions

EDA-INSIGHT-3-Number of Transactions by Amount Range

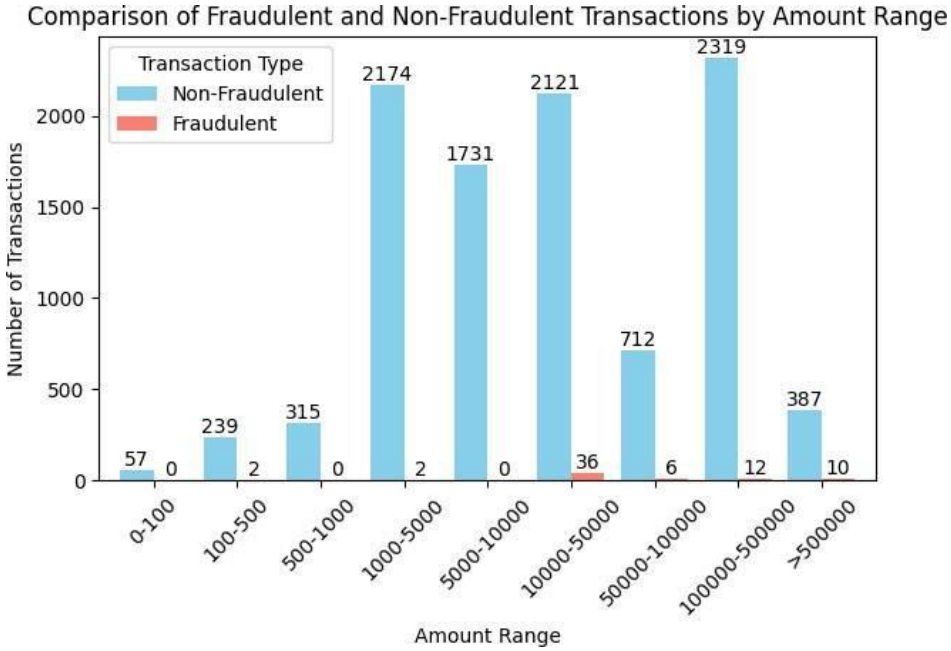


Inference: concentration of transactions within specific amount ranges, with a noticeably higher volume in mid-range amounts compared to very high or very low values.

Observation: Most transactions fall within 1000-500000 amount ranges, while high- value transactions are relatively rare. This trend may impact the visibility of high- value fraudulent activities if they blend in with lower-volume patterns.

Recommendations: Implement threshold-based monitoring and anomaly detection for transactions in both mid and high-value ranges to capture suspicious activity

EDA-INSIGHT-4-Fraudulent and Non-Fraudulent Transactions by Amount Range

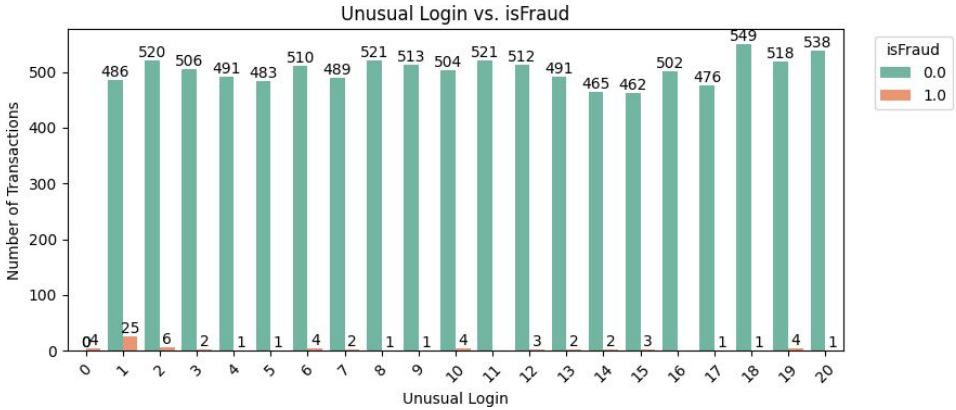


Inference: Fraudulent transactions tend to occur more frequently within 10000-50000 amount ranges.

Observation: Non-fraudulent transactions are widely distributed across all amount ranges, while fraudulent ones show higher concentrations in specific ranges. This clustering could reflect deliberate targeting of particular transaction amounts by fraudsters.

Recommendations: Adjust detection algorithms to flag transactions within identified high-risk amount ranges for closer inspection

EDA-INSIGHT-5-Fraudulent Transactions for Each Unusual Login

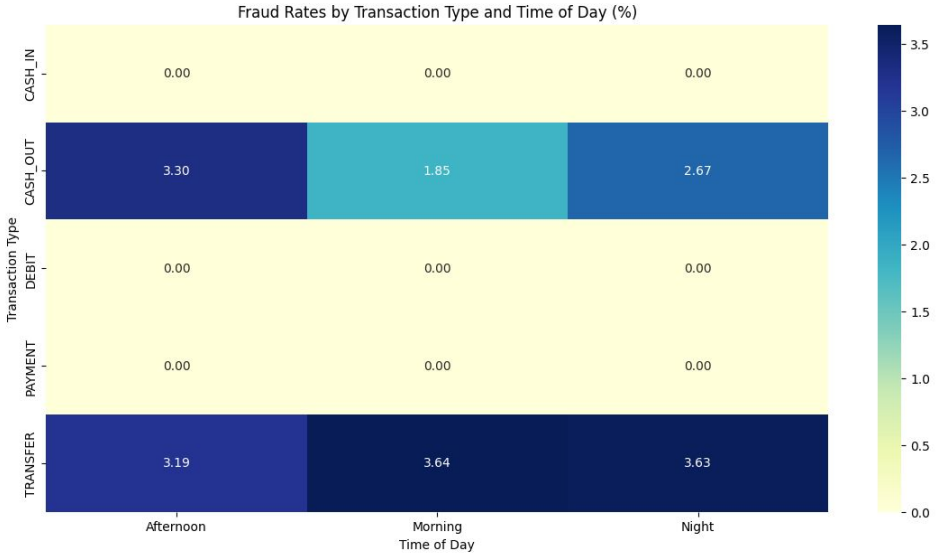


Inference: Fraudulent transactions tend to have a moderate incidence among accounts with unusual login patterns, indicating that such logins might be predictive of fraudulent behavior

Observation: Accounts with unusual logins are more likely to be associated with fraud cases, suggesting a correlation between login anomalies and fraudulent activities.

Recommendations: Integrate login pattern analysis into the fraud detection system, flagging transactions associated with unusual logins for further scrutiny to proactively identify potential fraud.

EDA-INSIGHT-6-Fraud Rates by Transaction Type and Time of Day

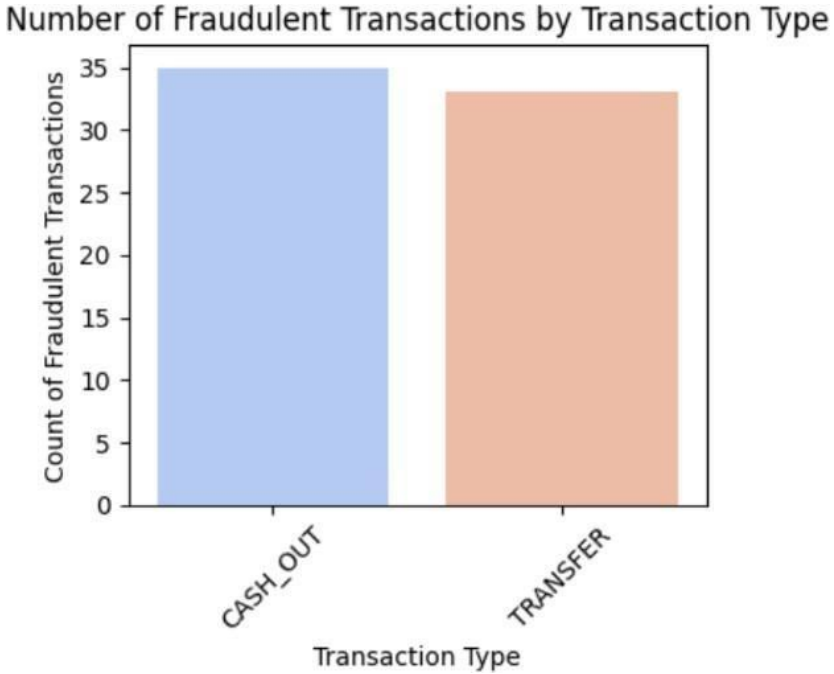


Inference: Higher fraud rates are observed for certain transaction types like "TRANSFER" and "CASH_OUT," and these are more frequent during specific times of the day, especially in the Afternoon and Evening

Observation: Fraudulent activities tend to increase with the "TRANSFER" and "CASH_OUT" transaction types in the Afternoon and Evening hours, suggesting that these times may present a higher vulnerability window for fraud.

Recommendations: enhancing security for "TRANSFER" and "CASH_OUT" transactions in the Afternoon and Evening by implementing stricter authentication checks, introducing dynamic fraud detection models that flag high-risk transactions based on time,

EDA-INSIGHT-7-Number of fraudulent transactions by transaction type

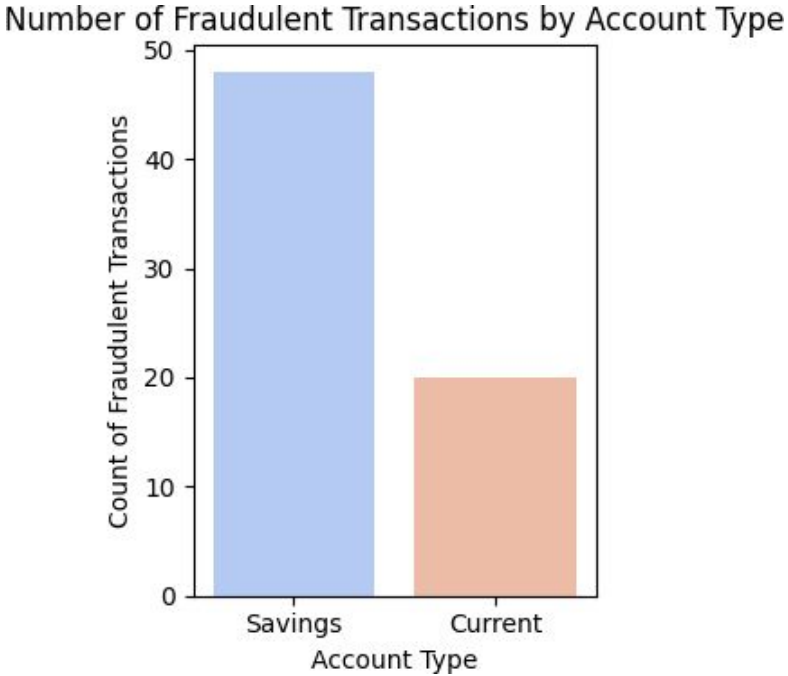


Inference: The dataset reveals that most fraudulent transactions occur in "CASH_OUT" and "TRANSFER" transaction types, with 35 and 33 fraudulent cases, respectively.

Observation: "CASH_OUT" transactions account for a slightly higher number of fraud cases than "TRANSFER" transactions.

Recommendations: Implement additional fraud detection measures focused on "CASH_OUT" and "TRANSFER" transactions, such as closer monitoring of these transactions for suspicious patterns or strengthening authentication processes specifically for these types.

EDA-INSIGHT-8-Account Type and Number of Fraudulent Transactions

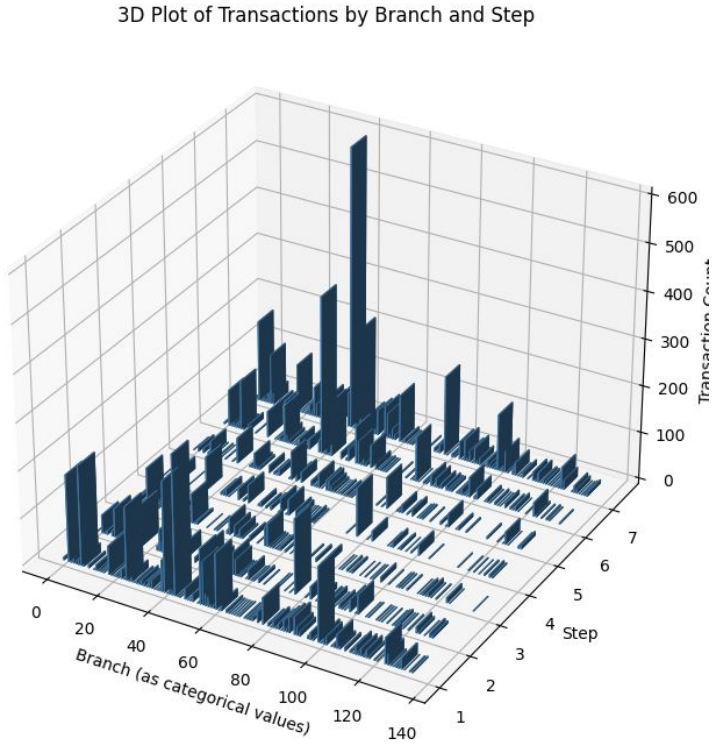


Inference: The distribution of fraudulent transactions varies significantly between account types, with one type ("savings") experiencing a notably higher number of fraudulent cases.

Observation: Analysis shows that a specific account type has a higher vulnerability to fraud, indicating that fraudulent activities may be more prevalent or more easily conducted within this account category.

Recommendations: Implement targeted fraud prevention measures for the more susceptible account type, such as enhanced authentication and real-time anomaly detection, to reduce fraud incidents and improve overall account security

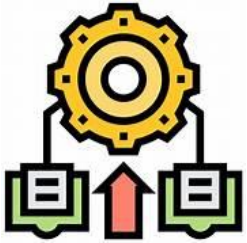
EDA-INSIGHT-10-Transactions by branch and step



Inference: The US branch has the highest transaction counts, particularly in steps 7 and 6 followed by France in step 1.

Observation: Steps 7 and 6 in the US branch have significantly higher transaction counts than other steps or branches.

Recommendations: Monitoring and optimizing transactions in step 7 and step 6 to enhance efficiency and detect any potential anomalies.



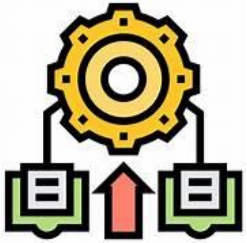
Machine Learning Model

Logistic Regression

- A linear model used for binary classification, predicting the probability of transactions being fraudulent.
- Simple and interpretable, it performs well with linearly separable data.
- Utilizes techniques like regularization to prevent overfitting.
- Selected for its simplicity and interpretability, making it easy to understand how features contribute to predicting fraudulent transactions.

Challenges

1. Struggles with non-linear relationships, which are common in fraud detection datasets.
2. Requires feature engineering to capture complex patterns effectively.



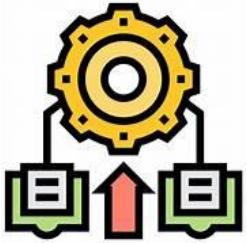
Machine Learning Model

Decision Tree Classifier

- A non-linear model that splits the data into subsets based on feature values, forming a tree structure.
- Easily interpretable and handles both numerical and categorical data effectively.
- Provides clear decision rules, making it interpretable and suitable for identifying key factors contributing to fraud.
- Chosen for its ability to capture non-linear patterns in the data, which are common in fraudulent transaction behavior.

Challenges

Prone to overfitting, especially on imbalanced datasets where fraudulent transactions are rare.



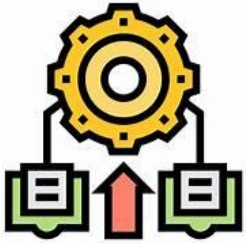
Machine Learning Model

Random Forest Classifier

- An ensemble of decision trees, combining their predictions to improve accuracy and generalization.
- Reduces overfitting by averaging outputs and is robust to noise and imbalanced data.
- Handles high-dimensional datasets and complex relationships well.
- Selected for its ensemble nature, which combines multiple decision trees to improve accuracy and robustness.

Challenges

1. Interpretability is reduced compared to single decision trees, making it harder to explain results.
2. May require careful hyperparameter tuning to balance performance and overfitting.



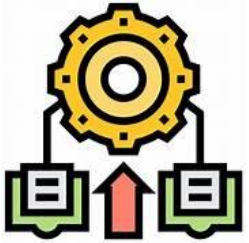
Machine Learning Model

XGBoost

- A gradient boosting algorithm designed for high efficiency and performance.
- Employs iterative tree-building and advanced regularization techniques to optimize model performance.
- Particularly effective for imbalanced datasets and large-scale problems, making it ideal for fraud detection.

Challenges

Highly sensitive to hyperparameter settings, requiring extensive tuning for optimal performance.



Model Evaluation

Accuracy: The ratio of correctly predicted observations to the total observations. It is a measure of how often the model is correct.

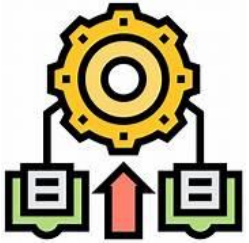
- Formula: (True Positives+True Negatives)/Total Observations
- Limitation: Can be misleading in imbalanced datasets.

Precision: The ratio of correctly predicted positive observations to the total predicted positives. It measures the accuracy of positive predictions.

- Formula: True Positives/(True Positives+False Positives)
- High precision means fewer false positives.

Recall: The ratio of correctly predicted positive observations to all the actual positives. It shows how well the model can capture the positive class.

- Formula: True Positives/(True Positives+False Negatives)
- High recall means fewer false negatives.



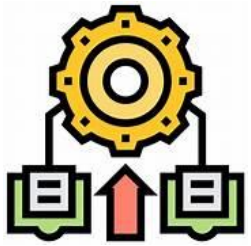
Model Evaluation

F1-Score: The harmonic mean of precision and recall, providing a balance between them. It is useful when you need to balance both precision and recall.

- Formula: $2 \times Precision \times Recall / (Precision + Recall)$

AUC (Area Under the Curve): The area under the ROC curve (Receiver Operating Characteristic curve), which plots the true positive rate (recall) against the false positive rate. AUC represents the model's ability to distinguish between positive and negative classes.

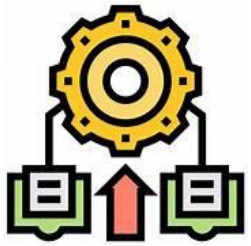
- AUC value ranges from 0 to 1, with 1 being a perfect classifier and 0.5 indicating a random model.



Model Evaluation

```
Model: LogisticRegression
Accuracy: 0.9936
Precision: 1.0000
Recall: 0.2353
F1-Score: 0.3810
AUC: 0.9823
Confusion Matrix:
 [[2009    0]
 [  13    4]]
==================================================
```

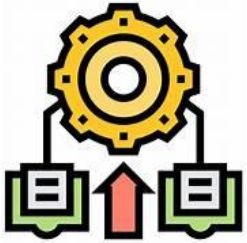
```
Model: DecisionTreeClassifier
Accuracy: 0.9990
Precision: 1.0000
Recall: 0.8824
F1-Score: 0.9375
AUC: 0.9409
Confusion Matrix:
 [[2009    0]
 [   2   15]]
==================================================
```



Model Evaluation

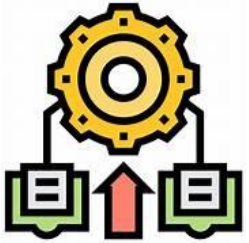
```
Model: RandomForestClassifier
Accuracy: 0.9961
Precision: 1.0000
Recall: 0.5294
F1-Score: 0.6923
AUC: 1.0000
Confusion Matrix:
 [[2009    0]
 [   8    9]]
==================================================
```

```
Model: XGBClassifier
Accuracy: 0.9985
Precision: 1.0000
Recall: 0.8235
F1-Score: 0.9032
AUC: 1.0000
Confusion Matrix:
 [[2009    0]
 [   3   14]]
==================================================
```



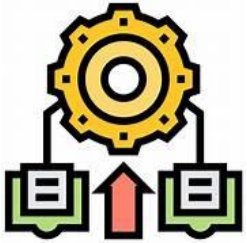
Model Evaluation

```
Cross-Validation AUC Scores: [0.95375494 0.99990683 0.85       0.99962733 0.94909938]
Mean CV AUC Score: 0.9504776962168264
Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      2009
         1.0       1.00      0.88      0.94        17

    accuracy                           1.00      2026
   macro avg       1.00      0.94      0.97      2026
weighted avg       1.00      1.00      1.00      2026

Performance Metrics:
Accuracy: 0.9990
Precision: 1.0000
Recall: 0.8824
F1-Score: 0.9375
AUC: 0.9409
Confusion Matrix:
[[2009    0]
 [   2   15]]
```



Summary of the Findings

1. **Model Performance**: The model achieved a high overall **accuracy** of **99.90%**, indicating that it can classify the majority of transactions correctly.
2. **Fraud Detection Precision**: The precision for detecting fraudulent transactions is **1.0000**, meaning every transaction flagged as fraudulent was indeed fraudulent, ensuring no false alarms.
3. **Fraud Detection Recall**: The recall is **0.8824**, signifying that the model successfully identified **88.24%** of all fraudulent transactions, though a small portion of fraud cases (2) was missed.
4. **Balanced Performance**:The **F1-score** of **0.9375** highlights the model's balance between precision and recall, ensuring effective fraud detection with minimal trade-offs.



Summary of the Findings

5. **AUC-ROC:** The **AUC score** of **0.9409** demonstrates the model's strong ability to distinguish between fraudulent and non-fraudulent transactions.
6. **Cross-Validation Consistency:** Cross-validation AUC scores show variability between folds, ranging from **0.85 to 0.9999**, with a mean of **0.9505**, indicating consistent but slightly variable performance across different data splits.
7. **Confusion Matrix Insights:** The model correctly classified all **2,009 non-fraudulent transactions** and identified **15 out of 17 fraudulent transactions**, missing only 2 cases.
8. **Classification Report Highlights:** The macro average scores (**precision: 1.0000**, **recall: 0.94**, **f1-score: 0.97**) confirm robust performance across both classes, with weighted averages close to 1.0000, emphasizing strong reliability.

Thanks!