Fraud Detection in Online Transactions: Project Report

Executive Summary

This project analyzes a dataset of online transactions to detect fraudulent activities using machine learning techniques. The dataset contains over 6 million records, with a severe class imbalance (only 0.13% fraudulent transactions). Exploratory Data Analysis (EDA) reveals key insights into transaction types, balances, and correlations. Data preprocessing involves handling imbalances through sampling, feature selection, and scaling/encoding. A Logistic Regression model is attempted, but the notebook encounters import errors from scikit-learn. Despite this, the foundation for a fraud detection pipeline is established. Recommendations include fixing dependencies and evaluating advanced models like Random Forests or XGBoost for better performance.

The analysis was conducted using Python in a Jupyter Notebook environment with libraries such as Pandas, NumPy, Matplotlib, Seaborn, and scikit-learn.

## Introduction

### Project Objective

The goal is to build a predictive model for identifying fraudulent transactions in online payments. Fraud detection is critical in financial systems to minimize losses and enhance security. This project focuses on:

- Understanding the dataset structure and imbalances.

- Visualizing patterns in fraudulent vs. legitimate transactions.

- Preprocessing data for modeling.

- Training a baseline classifier (Logistic Regression with class weighting).

Dataset Overview

- Source: "AIML Dataset.csv" (likely a variant of the PaySim synthetic fraud dataset).

- Size: 6,362,620 rows × 11 columns.

- Features:

- `step`: Time step (unit of time, e.g., hours).

- `type`: Transaction type (e.g., PAYMENT, TRANSFER, CASH\_OUT).

- `amount`: Transaction amount.

- `nameOrig`: Originator's account ID.

- `oldbalanceOrg`: Originator's balance before transaction.

- `newbalanceOrig`: Originator's balance after transaction.

- `nameDest`: Recipient's account ID.

- `oldbalanceDest`: Recipient's balance before transaction.

- `newbalanceDest`: Recipient's balance after transaction.

- `isFraud`: Target variable (1 = Fraud, 0 = Legitimate).

- `isFlaggedFraud`: System-flagged fraud (very rare, only 16 instances).

- Key Statistics:

- No missing values.

- Fraudulent transactions: 8,213 (0.13%).

- Flagged fraud: 16 (near-zero).

- Transaction types: Dominated by CASH\_OUT and PAYMENT.

Methodology

Tools and Libraries

- Python 3.12+ environment.

- Data manipulation: Pandas, NumPy.

- Visualization: Matplotlib, Seaborn.

- Modeling: scikit-learn (for preprocessing and classification; note: notebook has unresolved import issues).

- Environment setup: Installed packages via pip (pandas, numpy, matplotlib, seaborn, scikit-learn).

Data Loading and Initial Exploration

The dataset was loaded into a Pandas DataFrame. Basic checks:

- `df.head()`: Showed sample rows with mixed numeric and categorical data.

- `df.info()`: Confirmed data types (5 floats, 3 ints, 3 objects) and no nulls.

- Class distribution:

- isFraud: 6,354,407 legitimate vs. 8,213 fraudulent.

- isFlaggedFraud: 6,362,604 unflagged vs. 16 flagged.

Exploratory Data Analysis (EDA)

EDA focused on distributions, relationships, and fraud patterns.

1. Fraud Distribution

- A pie chart visualized the class imbalance: ~99.87% legitimate, ~0.13% fraudulent.

- Insight: Severe imbalance requires techniques like oversampling, undersampling, or class weighting.

2. Transaction Types

- Bar plot of transaction counts by type: CASH\_OUT and PAYMENT are most common.

- Fraud by type: Bar plot showed fraud primarily in TRANSFER and CASH\_OUT types.

- Insight: Fraudsters prefer methods that quickly extract funds (e.g., transfers followed by cash-outs).

3. Amount and Balance Analysis

- Box plots compared amounts and balances for fraudulent vs. legitimate transactions.

- Fraudulent transactions often involve draining the originator's account (newbalanceOrig ≈ 0).

- High amounts are more common in fraud.

4. Correlation Analysis

- Heatmap of correlations (using Seaborn):

- Strong positive correlation between oldbalanceOrg and newbalanceOrig.

- Amount correlates with changes in balances.

- isFraud shows weak correlations overall, but notable with balance drains.

- Insight: Features like balance differences (e.g., oldbalanceOrg - newbalanceOrig) could be engineered for better prediction.

5. Time-Based Analysis

- Line plot of transactions over steps: Spikes in certain time periods, but no clear fraud trend without further grouping.

Additional Visualizations

- Scatter plots: Amount vs. oldbalanceOrg, colored by fraud status.

- Pairplot: Subset of features to spot multivariate patterns (computationally intensive due to dataset size).

Data Preprocessing

- Feature Selection: Dropped irrelevant columns (nameOrig, nameDest – as they are IDs; isFlaggedFraud – due to rarity).

- Handling Imbalance: Undersampled the majority class to create a balanced subset (e.g., 10,000 samples each).

- Train-Test Split: 80-20 split using stratified sampling to preserve class ratios.

- Feature Types:

- Numeric: step, amount, oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest.

- Categorical: type.

- Pipeline Setup:

- ColumnTransformer: StandardScaler for numerics, OneHotEncoder (drop first) for categoricals.

- Full Pipeline: Preprocessor + LogisticRegression (balanced weights, max\_iter=1000).

- Issue Noted: Notebook errors due to missing imports (e.g., from sklearn.compose import ColumnTransformer; from sklearn.pipeline import Pipeline; from sklearn.preprocessing import StandardScaler, OneHotEncoder; from sklearn.linear\_model import LogisticRegression; from sklearn.model\_selection import train\_test\_split). These need resolution for execution.

Model Building and Evaluation

- Baseline Model: Logistic Regression with class weighting to handle imbalance.

- Training: Attempted on sampled data, but halted by import errors.

- Prediction and Metrics: y\_pred generation failed; in a complete run, evaluate using:

- Accuracy, Precision, Recall, F1-Score (focus on Recall for fraud detection to minimize false negatives).

- Confusion Matrix.

- ROC-AUC (ideal for imbalanced data).

- Expected Challenges: Overfitting on small sampled data; need full dataset with SMOTE or similar for better generalization.

- Alternatives Suggested: Try Decision Trees, Random Forests, or Gradient Boosting for non-linear patterns.

Results and Findings

- EDA Key Insights:

- Fraud is rare but patterned: Often involves full account drainage via TRANSFER/CASH\_OUT.

- No strong linear correlations with fraud, suggesting need for feature engineering (e.g., balance delta = oldbalanceOrg - amount).

- Time steps show cyclic patterns, but fraud isn't time-specific.

- Model Performance: Incomplete due to errors. Hypothetical: Logistic Regression typically achieves ~90% recall on similar datasets with proper handling.

- Limitations:

- Dataset size (6M+ rows) causes memory issues; sampling reduces representativeness.

- Synthetic data may not fully mimic real-world fraud.

- Unresolved code errors prevent full modeling.

Conclusion and Recommendations

This project demonstrates a solid EDA pipeline for fraud detection, highlighting the importance of imbalance handling. The model setup is promising but requires fixing scikit-learn imports and running on a machine with sufficient resources.

\*\*Recommendations\*\*:

- Import necessary sklearn modules and rerun the pipeline.

- Use advanced sampling (e.g., SMOTE) or ensemble models for better accuracy.

- Deploy as a real-time system with monitoring.

- Future Work: Incorporate external data (e.g., user behavior) and deep learning (e.g., Autoencoders for anomaly detection).

References

- Dataset: Inspired by PaySim fraud simulation.

- Libraries: Pandas (data handling), Seaborn/Matplotlib (visuals), scikit-learn (modeling).

- Notebook: "analysis\_model.ipynb" (provided).

This report can be hosted online (e.g., via GitHub Pages or Google Docs) for sharing. If you need the full corrected notebook or HTML export, provide more details!