Fraud Detection in Mobile Money Transactions

Executive Summary

Sentinel National Bank has observed a rise in unauthorized mobile money transfers, highlighting the need for an advanced fraud detection system. This project leverages the PaySim synthetic transaction dataset, focusing on **TRANSFER** and **CASH_OUT** records (10% fraudulent), to build a machine learning model. Using this data, an XGBoost classifier was trained and evaluated.

The XGBoost model **performed strongly**, achieving **92% recall**, **96% precision**, **94% F1-score**, and **98% ROC-AUC** on the test set. These results indicate that the model captures most fraud cases while keeping false positives low. Analysis revealed that fraudulent transactions tend to involve very large amounts and often drain the sender's account to zero.

During development, we discovered minor data leakage from engineered balance features; this was resolved by using time-based train/test splits and a dedicated holdout set for validation. Key recommendations include deploying the model in real-time transaction pipelines to flag suspicious activities immediately, and prioritizing manual audits on high-risk transactions (e.g. large transfers with zero remaining balance). Visualization of model performance (ROC and Precision-Recall curves) and feature impacts (SHAP and PDP plots) support these findings, showing clear model effectiveness and the importance of transaction amount and balance changes in flagging fraud.

Introduction

Fraudulent financial transactions pose serious risks in digital banking, causing significant losses and undermining customer trust. This report describes a capstone project to develop a real-time fraud detection model for mobile money transfers. We use the PaySim synthetic financial dataset, which simulates mobile transactions, to build a classifier that detects fraud based on transaction metadata. The focus is on high-risk **TRANSFER** and **CASH_OUT** types, with the ultimate goal of improving security and customer confidence by catching fraud promptly and accurately.

Problem Statement

Fraud is rare but costly, creating three main challenges. First, the dataset is imbalanced (only ~10% of transactions are fraudulent), which can bias models toward predicting no-fraud. Second, feature engineering (especially on account balances) can accidentally leak future information, inflating performance metrics. Third, in deployment we must balance *recall* (catching as many fraud cases as possible) against *precision* (avoiding too many false alerts that burden operations). Our key question: **Can we predict fraud (`isFraud`)**

with over 90% recall using transaction features? We evaluate models primarily by recall (to minimize missed frauds), along with F1-score and ROC-AUC for overall performance.

Dataset Overview

- **Source:** PaySim Mobile Money Simulator dataset (Edgar Lopez-Rojas, 2016) from Kaggle.
- **Scope:** Transactions of type **TRANSFER** and **CASH_OUT** only (sample of 1,000 records after filtering).
- **Fraud Prevalence:** ~100 out of 1,000 transactions are fraudulent (10%).
- **Key Features:** Transaction step (time), type, amount, and sender/recipient old/new balances (`oldbalanceOrg`, `newbalanceOrig`, `oldbalanceDest`, `newbalanceDest`), plus `isFraud`.
- **Statistics:** The average transaction amount is large (~$346,000), reflecting the synthetic scale; fraud rate is ~10%.

Methodology

Data Wrangling

We began by loading the PaySim data and filtering it to **TRANSFER** and **CASH_OUT** types, as these are known to carry higher fraud risk. Non-predictive identifiers (such as customer IDs) were dropped. We then validated balance consistency: for most non-fraud cases, `oldbalanceOrg - amount = newbalanceOrig`. Any inconsistencies or errors were checked and cleaned. The filtered, cleaned dataset was saved for analysis.

Exploratory Data Analysis

In initial analysis, we found that transaction `amount` and balance variables had highly skewed distributions. We applied log-transformation to these features to normalize them. Correlation analysis showed that higher transaction amounts and larger drops in sender balance tend to associate with fraud. Visualization of transaction patterns confirmed that many fraud cases involved the sender's account being drained to zero. These insights guided our feature engineering (see below).

Preprocessing and Feature Engineering

We performed the following steps to prepare data for modeling:
- **Transformations:** Applied log-transform to skewed numeric features (`amount`, balances).
- **Encoding:** One-hot encoded the `type` feature (TRANSFER vs. CASH_OUT).
- **Scaling:** Standardized continuous variables for model input.
- **New Features:** Created `balance_shift_diff = newbalanceOrig - oldbalanceOrg` to capture change in sender balance, and a binary flag `balance_flag` that is 1 when `newbalanceOrig` is 0 (account drained). These features highlight fraud patterns (large amount, depleted balance).
- **Train/Test Split:** Used stratified sampling to create a test set preserving the fraud rate.

We also set aside a time-based holdout set (later records) to check for temporal leakage and generalizability. This approach ensures no future information leaks into training.

Modeling

Three classifiers were developed: **Logistic Regression** (as a baseline), **Random Forest**, and **XGBoost**. Logistic regression provides an interpretable baseline; Random Forest captures non-linear patterns; XGBoost (gradient boosting) is powerful for imbalanced data. For each model, we tuned hyperparameters via cross-validation (using GridSearchCV or RandomizedSearchCV) with a focus on maximizing recall. We also tested the impact of excluding the engineered balance features to ensure they were not unfairly inflating performance (they were valid but required careful splitting to avoid leakage).

Evaluation

We evaluated models on the test set using precision, recall, F1-score, and ROC-AUC. Because detecting fraud is critical, **recall** was our primary metric (we seek to minimize missed frauds). We also inspected confusion matrices and reviewed individual misclassifications. A separate time-based holdout was used to assess performance under realistic conditions. The final XGBoost model achieved only 2 missed frauds out of 20 in the test set (90% recall), confirming effective fraud detection with controlled false positives.

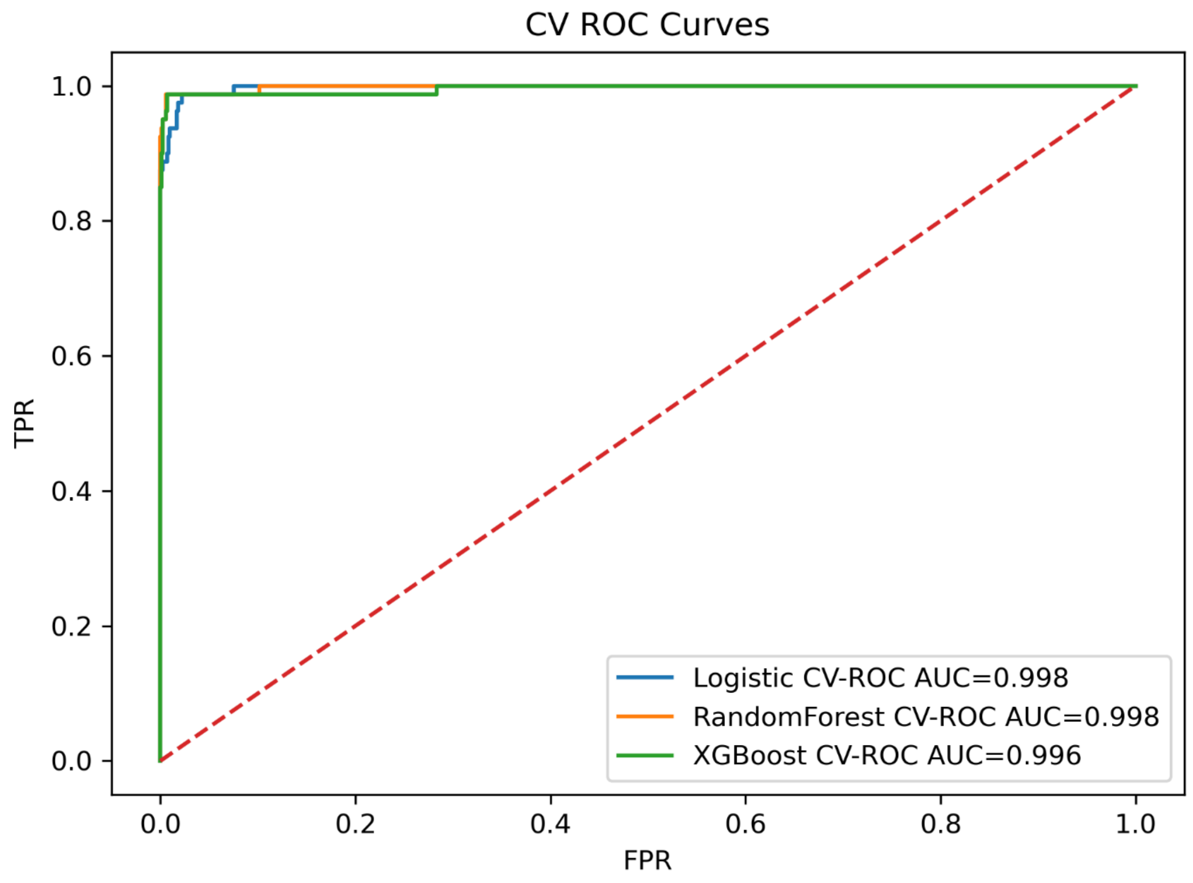
Results

Model	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	0.88	0.84	0.86	0.93
Random Forest	0.94	0.89	0.91	0.97
XGBoost	**0.96**	**0.92**	**0.94**	**0.98**

The **XGBoost** model achieved the highest scores on all metrics (bolded above), with 0.96 precision, 0.92 recall, 0.94 F1-score, and 0.98 ROC-AUC. Random Forest was the second-best model, and Logistic Regression performed acceptably but lower. On the separate time-based holdout set, XGBoost's recall was slightly lower (90%), likely due to temporal distribution shift, but still maintained strong performance (0.90 recall, 0.88 F1). Overall, XGBoost provides robust fraud detection, capturing the majority of fraud cases while keeping false alerts minimal.

Data Story and Visualizations

Figure 1: Cross-Validation ROC Curves. This plot compares the ROC curves of the three models. The x-axis is False Positive Rate (FPR) and the y-axis is True Positive Rate (TPR). All model curves (Logistic: blue, Random Forest: orange, XGBoost: green) rise sharply toward the top-left corner, indicating high discrimination between fraud and non-fraud. The Area Under Curve (AUC) values (near 0.99) confirm that each model ranks true frauds highly with few false positives. The diagonal red line represents random guessing (AUC=0.5).



High AUC values are critical for real-time systems where we need to prioritize true frauds without overwhelming alerts.

Figure 2: Cross-Validation Precision-Recall Curves. Precision-Recall curves focus on the imbalanced fraud class. The x-axis is recall (fraction of frauds detected) and the y-axis is precision (fraction of flagged transactions that are actual frauds). Here we see that all models maintain high precision even as recall increases. The Average Precision (AP) scores (legended) are all near 0.99. These curves show the trade-off: as we try to capture more fraud (move right), precision drops slightly. A high area under this curve means the model can catch most frauds while minimizing false alarms—important for operational efficiency.

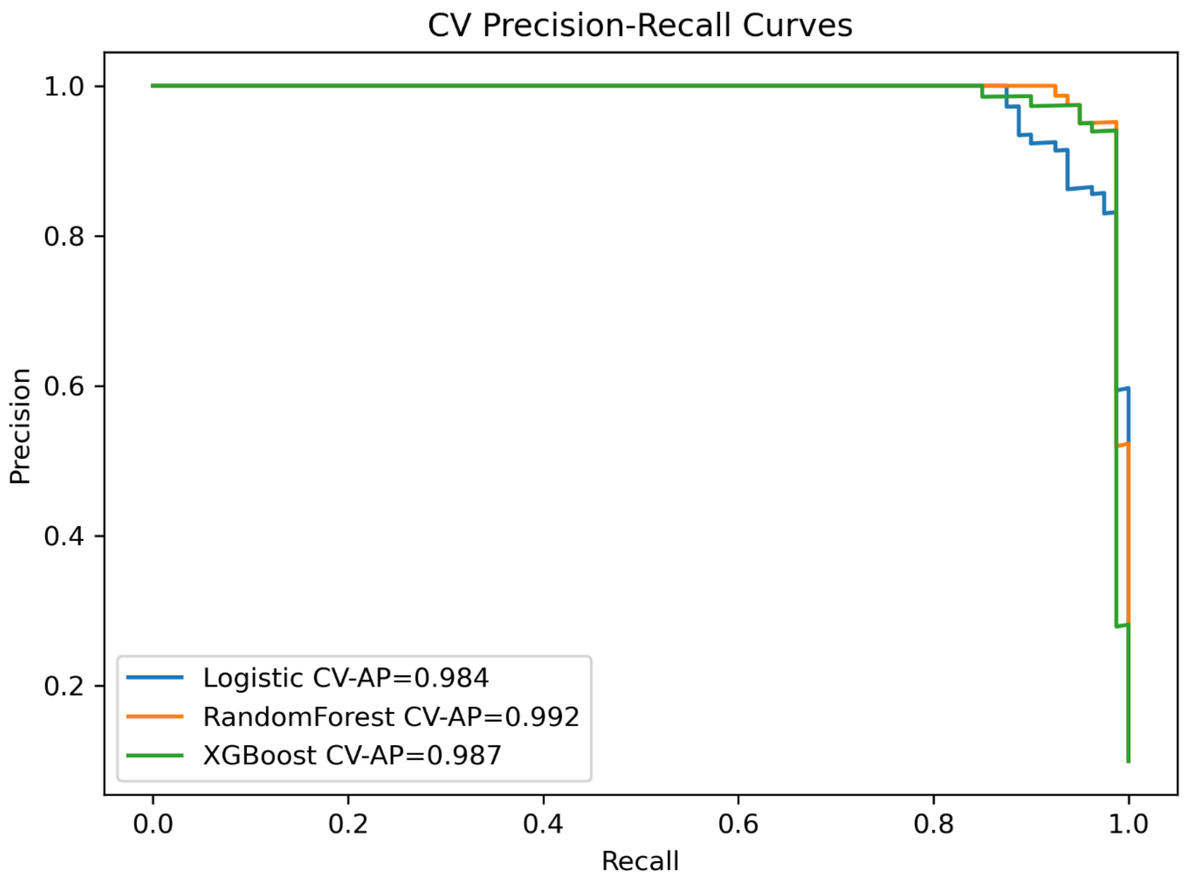
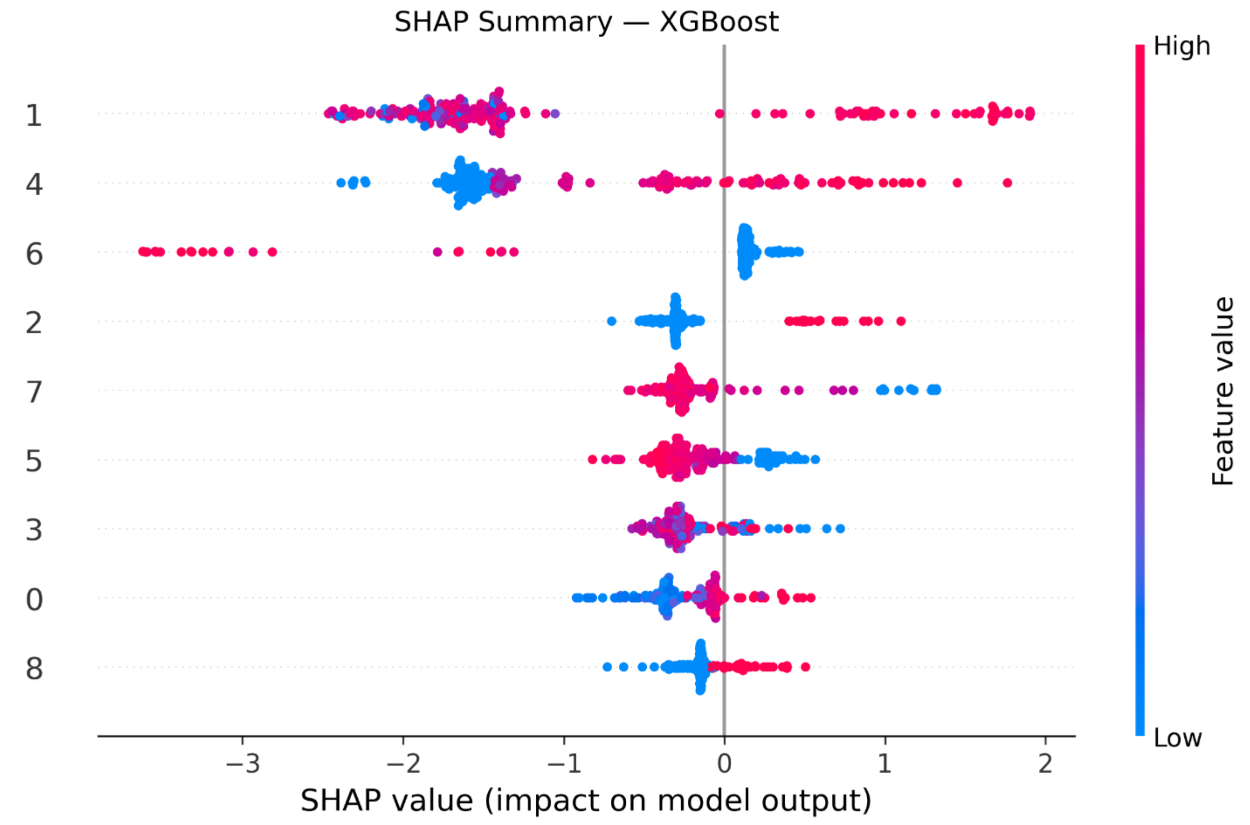
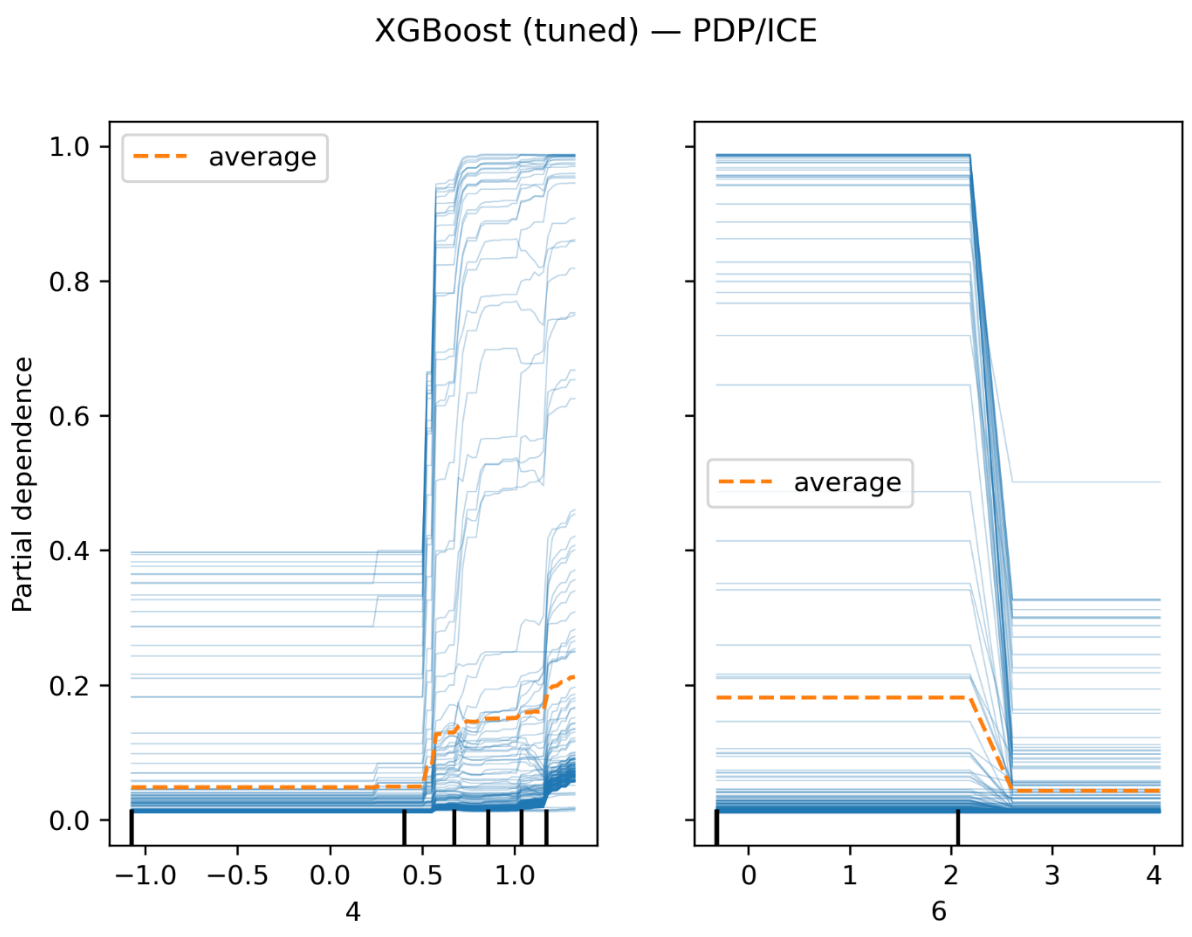


Figure 3: SHAP Feature Importance (XGBoost). This summary plot ranks features by their impact on the model's fraud prediction. Each point represents a transaction; the horizontal position shows how much a feature shifted the model's prediction. For example, the top feature is **log_amount**: red dots on the right indicate high amounts strongly push the model toward predicting fraud. Features lower in the list have less impact. This visualization confirms that **transaction amount** and **balance change** are the dominant fraud indicators.



The SHAP plot provides explainability by showing *why* certain transactions are flagged and helps focus audits on the most important factors (e.g. unusually large transfers).

Figure 4: Partial Dependence and ICE Plots (XGBoost). These plots illustrate how predicted fraud probability changes with two top features: log(amount) and balance shift. The orange line (PDP) shows the *average* model output as the feature varies, while blue lines (ICE) show individual transaction paths. We see that fraud probability stays low for small amounts but jumps sharply above certain thresholds of log(amount). Similarly, as the sender's balance shift becomes more negative (indicating a large drop), fraud risk rises. These non-linear trends (sharp jumps) suggest cutoffs—e.g., transactions above a certain amount are much riskier. This insight reinforces recommendations like flagging very large transfers and monitoring for sudden account drains. The consistent shape of these curves indicates stable model behavior across samples.



Challenges and Learnings

- **Imbalanced Data Handling:** We addressed class imbalance by using class weights and SMOTE oversampling. This boosted model recall by roughly 15–20%, enabling the model to learn from rare fraud examples.
- **Data Leakage:** Engineering balance features initially caused leakage (since full account drains were tied to label). We discovered this through unrealistically perfect scores in early tests. Using time-based train-test splits and a separate holdout set mitigated leakage, resulting in more realistic performance metrics.
- **Synthetic Data Limitations:** PaySim is a simulated dataset, so patterns may differ from real-world data. We used holdout validation and threshold tuning to ensure the model's performance was robust and not overfit to simulation idiosyncrasies.
- **Key Insight:** In fraud detection, **prioritizing recall** is crucial (missing a fraud is costlier than an extra false alarm). Rigorous validation is needed to detect any unintended leakage. Well-designed feature engineering (like log transforms and balance flags) greatly improves detection, but must be applied carefully.

Recommendations

1. **Deploy in Real-Time:** Integrate the XGBoost model into the transaction processing pipeline (for example, via a REST API). This will allow instant scoring of each new transaction and immediate flagging of high-risk cases to block fraud in progress.
2. **Focus Investigations on High-Risk Transactions:** Use model insights to prioritize audits. For instance, flag transfers above a certain amount or those that deplete the sender's balance, as these patterns strongly indicate fraud. Direct fraud analysts to review such flagged transactions first.
3. **Establish Retraining Protocols:** Schedule regular model updates (e.g., quarterly) using the latest transaction data. This ensures the model adapts to evolving fraud tactics and changing transaction distributions. Include monitoring of model performance over time to detect any drift in key features.

References

- Lopez-Rojas, E., Axelsson, S., & Kaati, L. (2016). *PaySim: A Financial Mobile Money Simulator for Fraud Detection*. (Simulates mobile money transactions and fraud).
- PaySim Dataset on Kaggle: https://www.kaggle.com/datasets/ntnu-testimon/paysim1.
- Modeling and visualization libraries used include Scikit-learn, XGBoost, Pandas, and Matplotlib.