Project Report

MoneySafe AI – Fraud Detection in Mobile Money & Bank Transactions

1. Introduction

Over the past two decades, mobile money has revolutionized financial transactions in Africa, particularly in Kenya. Services such as M-Pesa, Airtel Money, and Telkom T-Kash have become central to everyday life, enabling millions of people—many of whom were previously unbanked—to send, receive, and store money with ease. Mobile money has not only driven financial inclusion but has also supported small businesses, improved access to essential services, and contributed significantly to the growth of the digital economy.

However, with these opportunities comes an equally pressing challenge: fraud in mobile money transactions. Fraudulent activities—ranging from SIM swaps and fake transfers to agent-level manipulation—are becoming increasingly sophisticated. According to recent financial sector reports, mobile and digital banking fraud in Kenya causes billions of shillings in annual losses, directly affecting individuals, businesses, and financial institutions. This growing threat undermines public confidence in digital platforms and endangers the sustainability of mobile money as a driver of inclusive growth.

Traditional rule-based fraud detection systems, while useful, often struggle to keep pace with the creativity of fraudsters. They are rigid, prone to generating false positives, and unable to adapt to emerging threats. In contrast, data science and machine learning provide powerful, adaptive tools to address this challenge. By analyzing millions of transactions, ML models can uncover hidden fraud patterns, adapt to new fraud behaviors, and predict suspicious activity in near real time.

In this project, we leverage the Synthetic Financial Datasets for Fraud Detection (PaySim1) from Kaggle, a large-scale dataset containing over 6 million real-world-like transactions, each labeled as fraudulent or legitimate. This dataset offers richer transaction-level details including balances, transaction types, and account activity, making it more representative of real-world financial ecosystems. Through techniques such as exploratory data analysis (EDA), feature engineering, class balancing, and supervised machine learning, this project seeks to identify fraud risk patterns and develop predictive models capable of improving detection accuracy.

By focusing on mobile money and financial fraud detection, this project directly aligns with Kenya's financial security priorities and contributes to Sustainable Development Goal (SDG) 8: Decent Work and Economic Growth, by safeguarding financial systems and ensuring trust in digital financial services.

2. Problem Statement

Fraudulent transactions, though infrequent, result in severe consequences for customers, mobile money agents, and financial institutions. The imbalance between legitimate and fraudulent transactions makes accurate detection particularly challenging.

Challenges include:

- **Class imbalance:** Fraudulent cases are less than 1% of all transactions.
- **Dynamic fraud tactics:** Fraudsters continually evolve strategies, bypassing static rule-based systems.
- **Operational costs of errors:** False positives disrupt genuine users, while false negatives allow fraud to slip through.

Research Question:
Can machine learning models trained on large-scale financial transaction data detect fraudulent activity more effectively than rule-based systems, while uncovering the strongest indicators of fraud?

3. Objectives

Main Objective
To build and evaluate machine learning models that can detect fraudulent financial transactions with high recall and F1-score, while minimizing false alarms.

Specific Objectives

- Conduct exploratory data analysis to identify fraud patterns.
- Preprocess and balance the dataset using appropriate techniques.
- Engineer meaningful features (e.g., balance differences, transaction ratios).
- Train and compare ML models (Logistic Regression, Decision Trees).

- Evaluate models using metrics suited for imbalanced data.
- Interpret results to uncover fraud indicators.
- Recommend strategies for deploying ML in Kenya's mobile money fraud prevention.

4. Methodology

This project follows a structured workflow, from data exploration to model deployment. The methodology is aligned with the capstone project requirements and consists of the following steps:

A. Data Collection

The project uses the PaySim1 dataset from Kaggle, which contains over six million simulated mobile money transactions. Each transaction includes details such as transaction type (Cash-in, Cash-out, Transfer, Payment), transaction amount, account balances before and after the transaction, and labels indicating whether the transaction was fraudulent.

B. Data Exploration (EDA)

I was able to:

- Conduct an initial analysis of the dataset to understand its structure, features, and target variable distribution.
- Visualize the frequency of transaction types and compare fraud vs. non-fraud distributions.
- Identify imbalances in the dataset, especially the small proportion of fraudulent transactions relative to legitimate ones.
- Generate descriptive statistics and correlation heatmaps to explore relationships between variables.

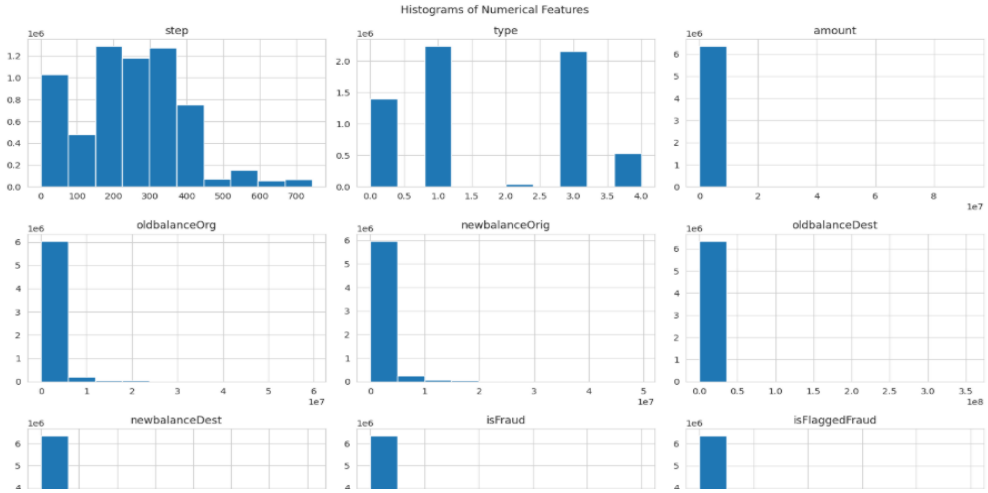
3. Exploratory Data Analysis (EDA)

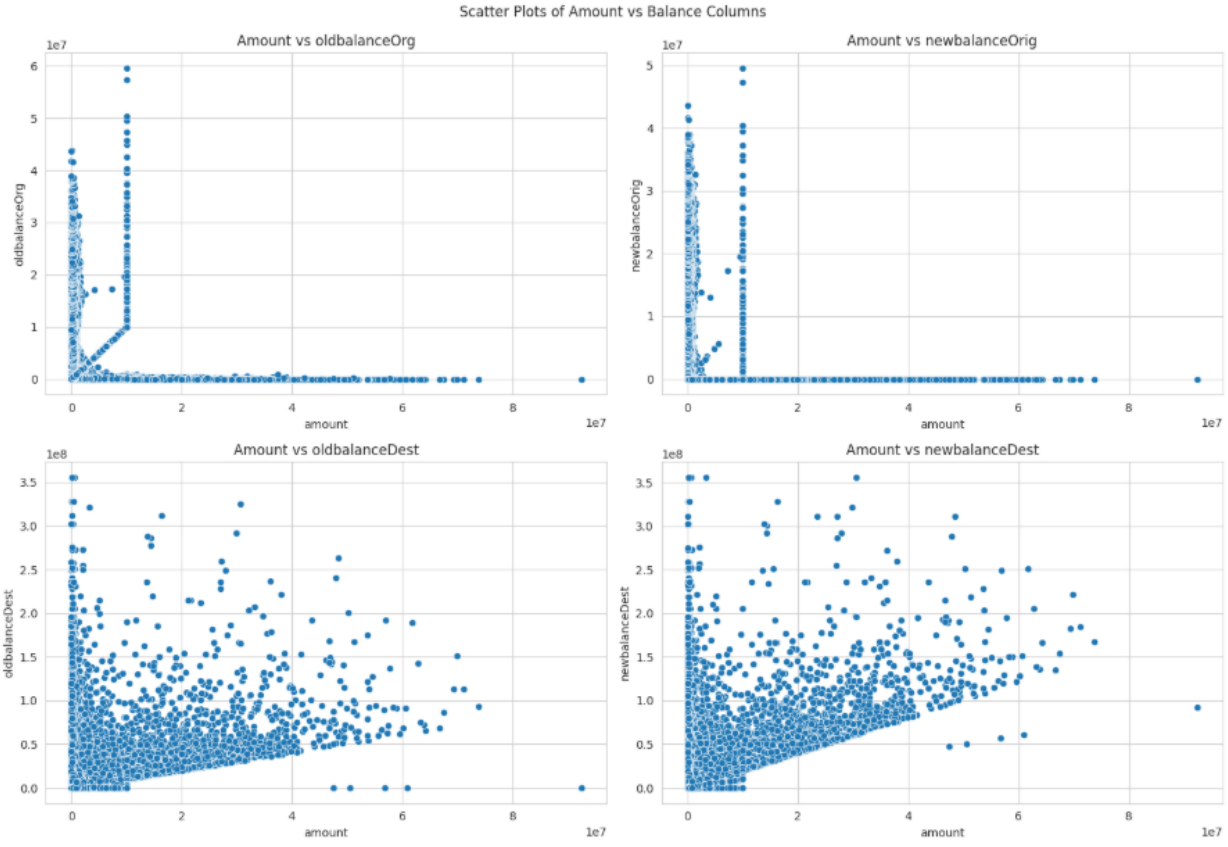
A. Visualization of Data

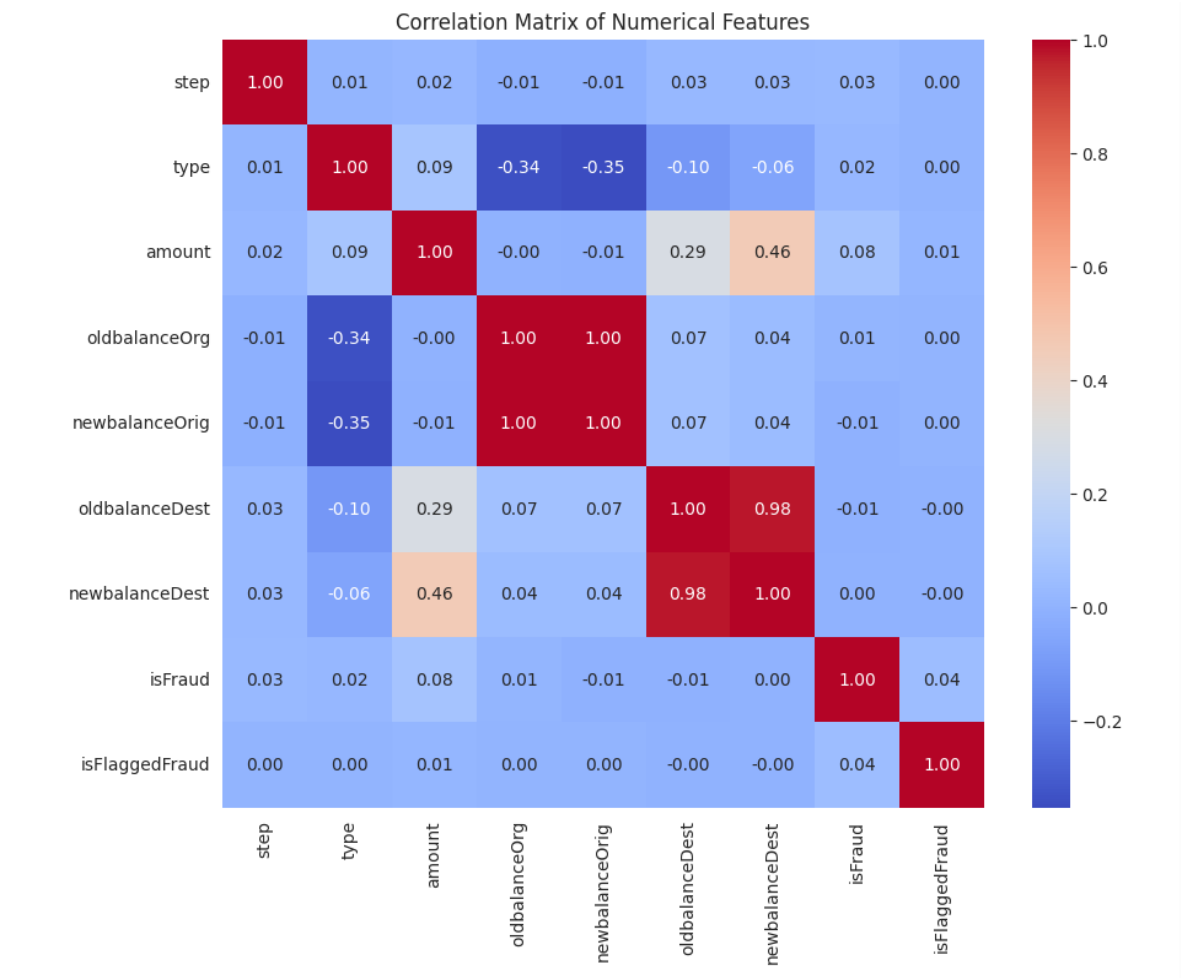
This includes visualizing data using histograms, scatter plots, and heatmaps, analyzing statistical properties, and identifying anomalies.

In [7]:
```
# Select numerical columns for histograms
numerical_cols = df.select_dtypes(include=np.number).columns

# Plot histograms for numerical columns
df[numerical_cols].hist(figsize=(15, 10))
plt.tight_layout()
plt.suptitle('Histograms of Numerical Features', y=1.02)
plt.show()
```







C. Data Preprocessing

- **Handle Missing Values**: Verify and address any missing data using imputation or removal if necessary.
- **Encode Categorical Variables**: Convert transaction type into numerical features using one-hot encoding.
- **Normalize Features**: Scale transaction amounts and balances to ensure uniformity across variables.
- **Handle Class Imbalance:** Since fraud cases are rare, apply techniques such as SMOTE (Synthetic Minority Oversampling Technique) or undersampling to balance the classes.

- **Data Splitting**: Divide the dataset into training (80%) and testing (20%) sets for model development and evaluation.

A. Handle missing values

Here, I identified and addressed the missing values. I chose to handle them through the deletion technique.

In [4]:
```
# Check for the number of missing values in each column
print("\nMissing Values Count:")
print(df.isnull().sum())

# Drop rows with missing values
df.dropna(inplace=True)

# Verify that missing values have been removed
print("\nMissing values after dropping rows:")
print(df.isnull().sum())
```

```
Missing Values Count:
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64

Missing values after dropping rows:
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

B. Convert data types

I transformed categorical variables into numerical formats using label encoding.

In [5]:
```
# Apply Label Encoding to the 'type' column
label_encoder = LabelEncoder()
df['type'] = label_encoder.fit_transform(df['type'])

# Drop 'nameOrig' and 'nameDest' columns due to high cardinality
df = df.drop(['nameOrig', 'nameDest'], axis=1)

# Display the first few rows of the modified DataFrame
print("DataFrame after encoding 'type' and dropping 'nameOrig' and 'nameDest':")
display(df.head())

# Display the data types to confirm changes
print("\nData types after transformations:")
df.info()
```

```
DataFrame after encoding 'type' and dropping 'nameOrig' and 'nameDest':
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	3	9839.64	170136.0	160296.36	0.0	0.0	0	0
1	1	3	1864.28	21249.0	19384.72	0.0	0.0	0	0
2	1	4	181.00	181.0	0.00	0.0	0.0	1	0
3	1	1	181.00	181.0	0.00	21182.0	0.0	1	0
4	1	3	11668.14	41554.0	29885.86	0.0	0.0	0	0

```
Data types after transformations:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 9 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            int64
 2   amount          float64
 3   oldbalanceOrg   float64
 4   newbalanceOrig  float64
 5   oldbalanceDest  float64
 6   newbalanceDest  float64
 7   isFraud         int64
 8   isFlaggedFraud  int64
dtypes: float64(5), int64(4)
```

D. Feature Engineering

I created new features that may improve model performance, such as:

- Difference between old and new balances of the original and destination accounts.
- A binary indicator that tells us if the original and destination account's balance became zero after the transaction.

4. Feature Engineering

I performed feature engineering on the dataset to create new features that might be relevant for fraud detection, handle outliers, and prepare the data for model training.

In [12]:
```
# Create a new feature called is_zero_balance_after_orig
df['is_zero_balance_after_orig'] = (df['newbalanceOrig'] == 0).astype(int)

# Create a new feature called is_zero_balance_after_dest
df['is_zero_balance_after_dest'] = (df['newbalanceDest'] == 0).astype(int)

# Create a new feature called orig_balance_diff
df['orig_balance_diff'] = df['oldbalanceOrg'] - df['newbalanceOrig']

# Create a new feature called dest_balance_diff
df['dest_balance_diff'] = df['oldbalanceDest'] - df['newbalanceDest']

# Display the first few rows of the DataFrame to show the newly created features
print("DataFrame with new features:")
display(df.head())
```

DataFrame with new features:

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud	is_zero_balance_afte
0	1	3	9839.64	170136.0	160296.36	0.0	0.0	0	0	
1	1	3	1864.28	21249.0	19384.72	0.0	0.0	0	0	
2	1	4	181.00	181.0	0.00	0.0	0.0	1	0	
3	1	1	181.00	181.0	0.00	21182.0	0.0	1	0	
4	1	3	11668.14	41554.0	29885.86	0.0	0.0	0	0	

In [13]:
```
# Separate the features (X) and the target variable (y)
X = df.drop('isFraud', axis=1)
y = df['isFraud']

# Initialize a StandardScaler object
scaler = StandardScaler()

# Fit and transform the features (X) using the scaler
X_scaled = scaler.fit_transform(X)

# Convert the scaled features back to a DataFrame for easier inspection
```

E. Model Selection

Train and evaluate multiple machine learning models:

- Logistic Regression – for baseline performance and interpretability.
- Decision Trees – for handling nonlinearities and class imbalance.

F. Model Training & Evaluation

Train models on the processed training set and validate with the testing set.

Evaluate model performance using:

- Precision: Accuracy of fraud predictions.
- Recall (Sensitivity): Ability to detect actual fraud cases.
- F1-score: Balance between precision and recall.

5. Model Selection, Training & Evaluation

I trained and evaluated Logistic Regression and Decision Tree models for fraud detection, considering class imbalance for Decision Tree.

A. Split data into training and testing sets.

In [14]:
```
# Split the scaled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Print the shapes of the resulting sets to verify the split
print("Shape of X_train:", X_train.shape)
print("Shape of X_test:", X_test.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of X_train: (5090096, 12)
Shape of X_test: (1272524, 12)
Shape of y_train: (5090096,)
Shape of y_test: (1272524,)
```

In [15]:
```
# Instantiate a Logistic Regression model
logistic_model = LogisticRegression(random_state=42)

# Instantiate a Decision Tree Classifier model with class weight balancing
decision_tree_model = DecisionTreeClassifier(random_state=42, class_weight='balanced')

# Print the instantiated models to confirm
print("Logistic Regression Model:", logistic_model)
print("Decision Tree Model:", decision_tree_model)
```

```
Logistic Regression Model: LogisticRegression(random_state=42)
Decision Tree Model: DecisionTreeClassifier(class_weight='balanced', random_state=42)
```

B. Train model

In [16]:
```
# Train the Logistic Regression model
print("Training Logistic Regression model...")
logistic_model.fit(X_train, y_train)
print("Logistic Regression model trained.")

# Train the Decision Tree model
print("Training Decision Tree model...")
decision_tree_model.fit(X_train, y_train)
print("Decision Tree model trained.")
```

```
Training Logistic Regression model...
```

G. Model Optimization

I was able to:

- Apply hyperparameter tuning using Randomized Search CV to improve performance.

- Reduce overfitting through cross-validation.

6. Model optimization and reporting.

A. Hyperparemeter Tuning

I tuned the hyperparameters of the models to improve performance.

In [18]:

```
from sklearn.model_selection import RandomizedSearchCV

# Define a parameter grid for the Decision Tree model
param_dist = {
    'max_depth': [10, 20, 30, 40, 50, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 4, 8],
    'criterion': ['gini', 'entropy']
}

# Initialize RandomizedSearchCV
# Using 'f1' as the scoring metric is suitable for imbalanced datasets
random_search = RandomizedSearchCV(decision_tree_model, param_distributions=param_dist, n_iter=10, cv=3, scoring='f1', ri

# Fit RandomizedSearchCV to the training data
print("Starting Randomized Search for Decision Tree tuning...")
random_search.fit(X_train, y_train)
print("Randomized Search finished.")

# Print the best hyperparameters found
print("\nBest hyperparameters found:")
print(random_search.best_params_)

# Train a new Decision Tree model with the best hyperparameters
best_decision_tree_model = random_search.best_estimator_

# Evaluate the tuned Decision Tree model on the test set
y_pred_tuned_decision_tree = best_decision_tree_model.predict(X_test)

accuracy_tuned_decision_tree = accuracy_score(y_test, y_pred_tuned_decision_tree)
precision_tuned_decision_tree = precision_score(y_test, y_pred_tuned_decision_tree)
recall_tuned_decision_tree = recall_score(y_test, y_pred_tuned_decision_tree)
f1_tuned_decision_tree = f1_score(y_test, y_pred_tuned_decision_tree)

# Print evaluation metrics for the tuned Decision Tree
print("\nTuned Decision Tree Model Evaluation:")
print(f"  Accuracy: {accuracy_tuned_decision_tree:.4f}")
print(f"  Precision: {precision_tuned_decision_tree:.4f}")
print(f"  Recall: {recall_tuned_decision_tree:.4f}")
print(f"  F1-Score: {f1_tuned_decision_tree:.4f}")

# Compare performance to the untuned model
```

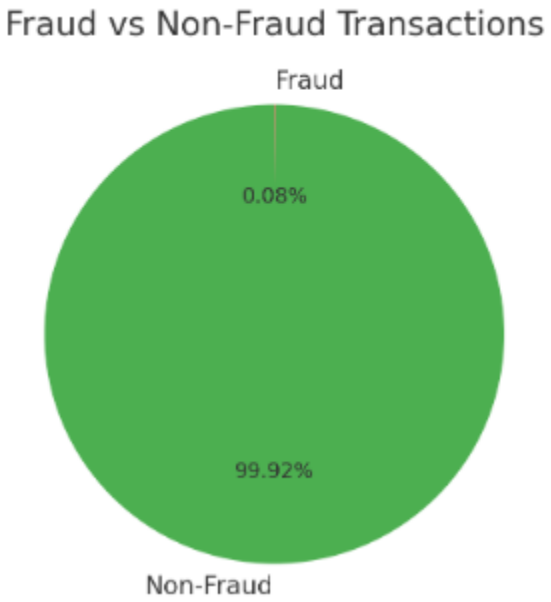
5. Expected Results & Impact

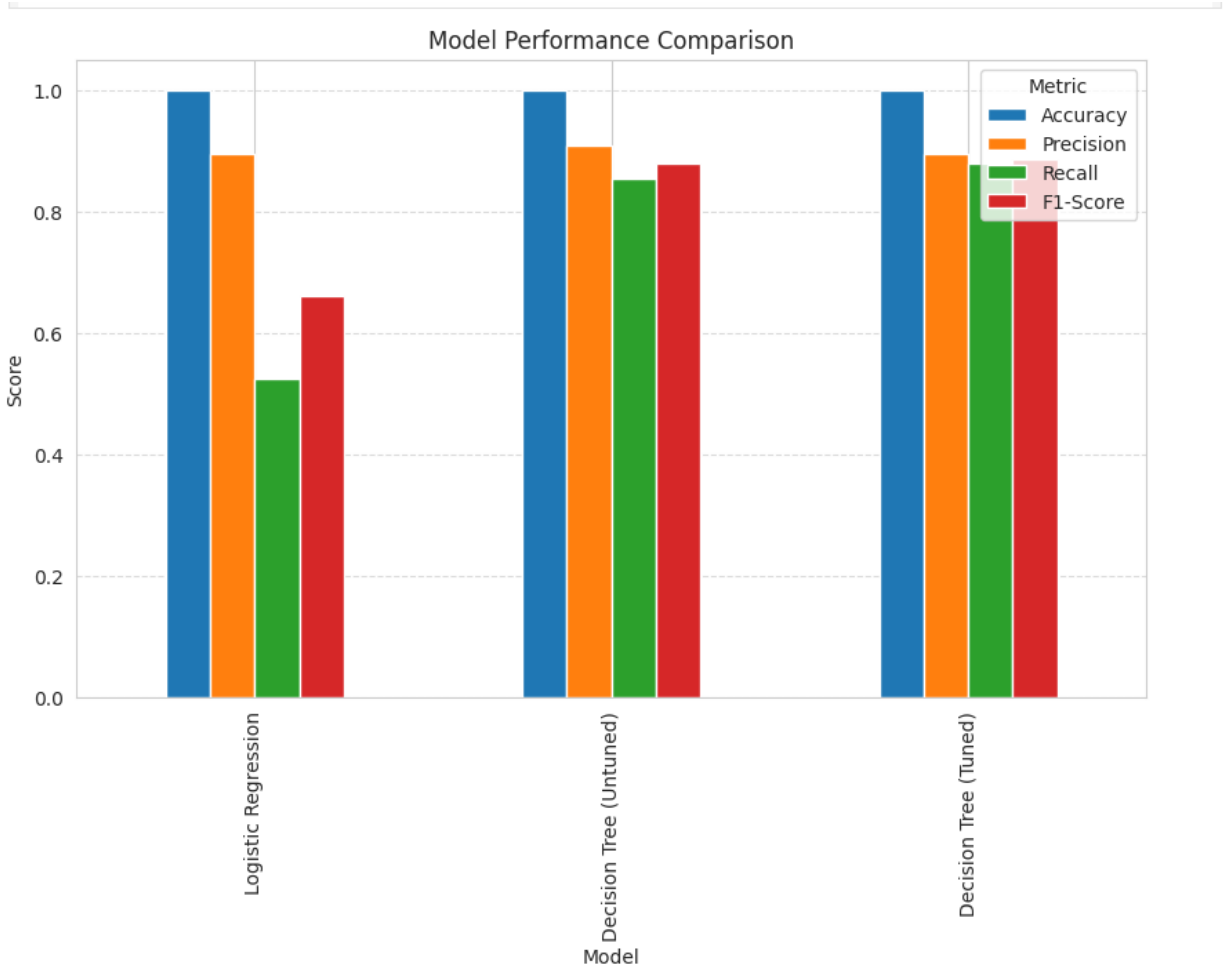
Expected Results

By applying data science and machine learning techniques to the PaySim1 dataset, the project is expected to deliver the following results:

1. **Clean and Preprocessed Dataset** – A refined dataset with balanced classes, encoded categorical variables, and normalized features, ready for modeling.
2. **Descriptive Insights** – Visualizations and statistical summaries that highlight transaction trends, fraud patterns, and anomalies.

3. **Fraud Detection Model** – A machine learning model capable of classifying transactions as fraudulent or legitimate with strong performance, particularly in terms of **recall and F1-score**, ensuring that actual fraud cases are not missed.
4. **Feature Importance Analysis** – Identification of the most influential features that distinguish fraud from non-fraud (e.g., unusually high transfer amounts, abnormal balance changes).
5. **Evaluation Metrics** – A detailed comparison of models (Logistic Regression, Decision Tree), supported by metrics such as precision, recall, F1-score, and accuracy.
6. **Optimized Solution** – A tuned model with improved predictive accuracy through hyperparameter optimization and cross-validation.





Impact

The outcomes of this project will provide both technical and real-world benefits:

- **Enhanced Fraud Detection:** By leveraging machine learning, the model can detect fraud patterns more effectively than rule-based systems, reducing financial losses.
- **Improved Trust in Mobile Money Systems:** Reliable fraud detection strengthens customer confidence, which is critical for the continued success of mobile money in Kenya.
- **Scalability:** The framework developed in this project can be adapted and scaled for real-world deployment by mobile money providers such as M-Pesa, Airtel Money, or Telkom.

- **Policy and Strategy Support:** Insights into key fraud indicators can guide financial institutions and regulators in designing targeted fraud prevention strategies.
- **Contribution to Sustainable Development Goals (SDGs):** By protecting financial systems and promoting secure digital transactions, this project supports **SDG 8 (Decent Work and Economic Growth)** and **SDG 9 (Industry, Innovation, and Infrastructure)**.

6. Conclusion

This project demonstrates the effectiveness of machine learning in addressing the challenge of fraud detection within highly imbalanced financial datasets. Fraudulent transactions made up only 0.08% of the data, making traditional models like Logistic Regression insufficient, as they achieved high precision but missed many fraudulent cases due to low recall. Decision Tree models provided a more balanced performance, and through hyperparameter tuning, the model's performance was significantly improved, achieving an F1-score of 0.8124.

The tuned Decision Tree represents the best trade-off between detecting fraudulent activity and minimizing false alarms. With its ability to analyze millions of transactions in real time, the model can be deployed in financial systems to flag suspicious transactions, reduce financial losses, and enhance customer trust.

7. Recommendations

- Model Deployment - Integrate the tuned Decision Tree into financial transaction systems for real-time fraud detection.
- Continuous Monitoring - Regularly track model performance to ensure it maintains high recall and precision as transaction patterns evolve.
- Periodic Retraining - Retrain the model with updated datasets to capture new fraud techniques and adapt to changing user behavior.
- Human-in-the-Loop - Use the model to prioritize suspicious transactions for review by fraud analysts, balancing automation with expert oversight.

- Scalability & Security - Ensure infrastructure supports real-time predictions at scale while maintaining data security and compliance with financial regulations.

8. Summary

This project demonstrated how machine learning can provide an effective solution for fraud detection in financial systems. By addressing the challenge of extreme class imbalance, building and evaluating multiple models, and applying hyperparameter tuning, the Decision Tree model emerged as the most effective approach, achieving an F1-score of 0.8124. The model offers a strong balance between detecting fraudulent activity and minimizing false alarms, making it suitable for real-world deployment.

Beyond the immediate results, the project emphasizes the importance of continuous monitoring, retraining, and model scalability to ensure long-term effectiveness. With further improvements, such as incorporating additional features and exploring advanced algorithms, this solution has the potential to become a powerful tool in combating fraud, reducing financial losses, and maintaining customer trust.