Bank transaction Fraud Detection

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

Data Science and Machine Learning

By

Venkata Siva Kalyan Bavireddy

Registration No: 12218431

Section: K22UG

Roll No: B58

Supervisor

Shivangini Gupta

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

March 2025

Project Title: Bank Trasaction Fraud Detection

1.Problem Statement

With the increasing adoption of digital banking and online financial services, the number of fraudulent transactions has risen dramatically. These fraudulent activities not only lead to significant financial losses for both banks and customers but also damage trust in financial institutions. Hence, there is a critical need for a robust system that can detect potentially fraudulent transactions in real-time.

The objective of this project is to **develop a machine learning-based fraud detection model** that can accurately identify suspicious or fraudulent transactions from a dataset of customer banking activities. The system should be capable of learning from historical transaction data and predicting the likelihood of fraud in future transactions. It should also minimize false positives to ensure a seamless experience for legitimate users.

Goals and Objectives:

- Perform Exploratory Data Analysis (EDA) to understand the structure and patterns in the dataset.
- Preprocess the data, handle missing values, outliers, and encode categorical features.
- Build and evaluate various machine learning models (e.g., Logistic Regression, Random Forest, XGBoost) to classify transactions as fraudulent or legitimate.
- Compare model performance using metrics such as Accuracy, Precision, Recall, F1-Score, and ROC-AUC.
- Optimize model performance using techniques like SMOTE (for imbalance), hyperparameter tuning, and feature selection.

Objective

The objective of this project is to build an intelligent machine learning model capable of detecting fraudulent bank transactions using a comprehensive dataset that includes customer demographics, transaction details, device information, and merchant-related data. By analyzing patterns in past transaction behavior, the model aims to accurately classify whether a given transaction is fraudulent or legitimate. This system will help financial institutions proactively identify suspicious activity, reduce financial losses, and enhance customer trust. The solution should be efficient, scalable, and able to minimize both false positives and false negatives to ensure security without compromising the user experience.

This project aims to develop a machine learning-based fraud detection system that can accurately identify fraudulent bank transactions by analyzing a wide range of features, such as customer information, transaction amount, type, device used, location, and merchant category. The objective is to leverage historical transaction data to uncover patterns indicative of fraudulent behavior and use these insights to build predictive models. The ultimate goal is to assist financial institutions in detecting suspicious transactions in real-time, thereby reducing financial losses, enhancing operational efficiency, and maintaining customer trust. The solution also focuses on

achieving a high detection rate while minimizing false alarms, ensuring a balance between security and user convenience.

The goal of this project is to design a machine learning model that detects fraudulent transactions using historical banking data. By analyzing various features such as transaction type, amount, customer demographics, and device information, the model aims to distinguish between legitimate and suspicious activities. This system will support banks in preventing fraud effectively while minimizing disruption for genuine customers.

Developed a machine learning model to detect fraudulent bank transactions using a real-world dataset with features like customer demographics, transaction type, amount, and device/location details. The model helps identify suspicious activity in real time, with an emphasis on precision and recall to reduce financial loss and improve banking security.

Scope

The scope of this project encompasses the end-to-end process of developing a machine learning-based fraud detection system using a real-world bank transaction dataset. It begins with exploratory data analysis (EDA) to understand transaction behavior and identify significant patterns and correlations. The dataset includes various features such as customer demographics, transaction amounts, types, device details, locations, and merchant categories, allowing for a deep understanding of transactional risk factors.

The project involves preprocessing the data, handling class imbalances, and transforming features into suitable formats for training machine learning models. Multiple classification algorithms will be explored—including logistic regression, decision trees, random forests, and gradient boosting models—to identify the most effective approach for detecting fraudulent transactions. Performance evaluation will be conducted using appropriate metrics such as accuracy, precision, recall, F1-score, and ROC-AUC to ensure the model performs reliably in real-world scenarios.

The scope also includes optimizing model performance using techniques like hyperparameter tuning and resampling methods (e.g., SMOTE). While the current focus is on detecting fraud based on historical patterns, the solution is designed to be scalable and adaptable for integration into real-time banking systems. However, it does not cover the deployment or integration aspects into live banking infrastructure, nor does it involve biometric or multi-factor authentication techniques.

2.Literature Review

Bank transaction fraud detection has been an important research area in the fields of data science and cybersecurity due to the increasing volume of online financial transactions and the growing sophistication of fraudulent techniques. Over the years, researchers have employed various machine learning and statistical models to detect anomalous transaction behavior and minimize financial losses.

Several traditional classification algorithms such as Logistic Regression, Decision Trees, and Naive Bayes have been explored for fraud detection due to their interpretability and simplicity. However, these models often struggle with imbalanced datasets—a common issue in fraud detection, where fraudulent transactions represent a very small portion of the data.

To address this, more advanced ensemble methods like Random Forest, XGBoost, and LightGBM have been adopted, offering higher accuracy and better handling of non-linear relationships. These models have shown significant promise in identifying complex patterns in high-dimensional data, especially when combined with techniques like SMOTE (Synthetic Minority Over-sampling Technique) to balance the dataset.

Recent literature also emphasizes the importance of feature engineering and domain-specific knowledge to improve model performance. Variables like transaction amount, time of transaction, device type, and customer location have been proven to be highly predictive. Additionally, deep learning approaches, such as neural networks and autoencoders, are being explored for unsupervised anomaly detection, particularly in large-scale banking systems with real-time requirements.

Studies by authors like Dal Pozzolo et al. (2015) and Carcillo et al. (2019) have demonstrated that combining behavioral analytics with machine learning can enhance fraud detection systems. Moreover, the integration of real-time detection mechanisms and feedback loops is being researched to develop adaptive systems that learn from new types of fraud.

3.Data Source

The dataset used in this project is publicly available and was obtained from **Kaggle**, a popular online platform for data science competitions and datasets. It contains **200,000 records** of anonymized bank transaction data, including customer demographics, transaction details, and a label indicating whether each transaction is fraudulent or legitimate.

Dataset Name: *Bank Transaction Fraud Detection Dataset*

Source: Kaggle

Link: https://www.kaggle.com/datasets *(Replace with actual URL of the dataset you used)*

License: Open for academic and research purposes

The dataset includes the following key fields:

- Customer_ID, Customer_Name, Gender, Age
- Transaction_ID, Transaction_Date, Transaction_Amount, Transaction_Type
- Transaction_Device, Device_Type, Transaction_Location
- Merchant_ID, Merchant_Category
- Account_Balance, Account_Type, Is_Fraud (Target Variable)

This rich feature set allows for comprehensive analysis and model building to accurately detect fraudulent banking activities.

4.Data Characteristics

Dataset Summary: Bank Transaction Fraud Detection

The Bank Transaction Fraud Detection dataset contains 200,000 transaction records with 24 features. Each record represents an individual banking transaction, which may be either legitimate or fraudulent. The dataset is primarily composed of a mix of categorical, numerical, and timestamp data, making it suitable for building classification models, anomaly detection systems, and behavior-based fraud prediction engines.

Feature Name	Description	Data Type
Customer_ID	Unique identifier for each customer	Object
Customer_Name	Full name of the customer	Object
Gender	Gender of the customer (e.g., Male, Female)	Object
Age	Age of the customer	Integer
State	State where the customer resides	Object
City	City where the customer resides	Object
Bank_Branch	Name or code of the branch where the transaction occurred	Object
Account_Type	Type of customer account (e.g., Savings, Current, Business)	Object
Transaction_ID	Unique identifier for each transaction	Object
Transaction_Date	Date when the transaction occurred	Object
Transaction_Time	Time when the transaction occurred	Object
Transaction_Amount	Total amount involved in the transaction	Float
Merchant_ID	Identifier of the merchant associated with the transaction	Object
Transaction_Type	Method used for the transaction (e.g., UPI, Debit Card, ATM)	Object
Merchant_Category	Type of merchant (e.g., Grocery, Electronics, Travel)	Object
Account_Balance	Account balance after the transaction	Float
Transaction_Device	Channel or device used (e.g., Mobile App, ATM, Voice Assistant)	Object
Transaction_Location	Location of the transaction	Object
Device_Type	Type of device used (e.g., Mobile, Desktop, Tablet)	Object

Feature Name	Description	Data Type
Is_Fraud	Indicator if the transaction is fraudulent (0 = No, 1 = Yes)	Integer
Transaction_Currency	Currency used in the transaction (mostly INR)	Object
Customer_Contact	Masked phone number of the customer	Object
Transaction_Description	Brief description or note about the transaction	Object
Customer_Email	Masked email ID of the customer	Object

5.Data Cleaning

Data Cleaning and Preprocessing for Bank Transaction Fraud Detection

Data cleaning is a crucial step in preparing the dataset for modeling. Since financial transaction data is prone to human or system errors, careful preprocessing ensures accuracy and reliability in detecting fraud

Remove Duplicates:

Check for duplicate rows based on identifiers like Transaction_ID or the entire row.

Duplicate transactions may result from system errors or retry logs, which can bias the model.

Correct Inconsistencies:

Standardize categorical values (e.g., "upi" vs "UPI", "Mobile app" vs "mobile App").

Unify text casing and formatting for device types, merchant categories, etc.

Fix Obvious Errors:

Remove or correct transactions with:

Negative Transaction_Amount (unless refunds are included).

Account_Balance inconsistencies.

Invalid timestamps or future Transaction_Date values.

Implausible Age values (e.g., age > 100 or < 10).

Normalize Text Fields:

Remove extra whitespace and punctuation in text columns like Transaction_Description and Merchant_Category.

Convert all string values to lowercase or title case to ensure uniformity.

Handling Missing Values

Drop Columns or Rows:

If a column like Transaction_Description or Customer_Email has over 80% missing values, consider dropping it unless it adds critical value.

Imputation Strategies:

Numerical Columns:

Impute Account_Balance and Transaction_Amount using median or mean, if necessary.

Categorical Columns:

Fill missing Transaction_Device or Merchant_Category with the most frequent value (mode).

For features like Device_Type or City, use a placeholder such as 'Unknown' if missing values are rare.

Predictive Imputation:

Use simple models (e.g., DecisionTreeRegressor or KNN) to impute missing Age based on other features like City, Gender, and Account_Type.

Encoding Categorical Variables

Since machine learning models need numerical inputs, categorical columns must be encoded.

One-Hot Encoding (Ideal for nominal features with few categories):

Gender, Transaction_Type, Device_Type, Account_Type, Transaction_Device

Creates binary columns for each category. Example: Transaction_Type_UPI | Transaction_Type_ATM | Transaction_Type_Card

Label Encoding (Efficient for ordinal or tree-based models):

Useful for features like Merchant_Category or Transaction_Location when many unique values exist.

Target Encoding (Useful when a category's relationship with fraud is significant):

Encode Merchant_Category or Bank_Branch using the average fraud rate for each category.

Helps in capturing fraud-prone entities.

Additional Feature Engineering Suggestions

Extract Hour, Day, Month, and Weekday from Transaction_Date and Transaction_Time.

Compute Transaction_Frequency per customer or merchant as a derived feature.

Calculate time differences between consecutive transactions per customer.

Flag High-Risk Locations or Devices using domain knowledge or fraud rate statistics.

6.Feature Selection

Feature selection is a vital step in building efficient and interpretable machine learning models. It helps in reducing dimensionality, improving model performance, and minimizing the risk of overfitting — especially in a binary classification task like fraud detection.

Correlation-Based Selection

Numerical features such as Transaction_Amount, Account_Balance, and Age can be assessed for correlation with the target variable (Is_Fraud).

Use techniques like:

Pearson correlation (for numeric-numeric)

Chi-square test (for categorical-target association)

Mutual Information scores (for both numeric and categorical)

Examples of potential correlations:

Unusually high Transaction_Amount values may positively correlate with fraud.

Certain Merchant_Categories or Transaction_Types may be disproportionately involved in fraud.

4.Methodology

Methodology: Bank Transaction Fraud Detection

The goal of this machine learning project is to build models that can detect fraudulent transactions based on behavioral and transactional features. Fraud detection is a classic binary classification problem where the target is:

1 → Fraudulent Transaction

0 → Legitimate Transaction

1. Algorithms Used

Logistic Regression
Used for binary classification problems. In fraud detection, it models the probability of a transaction being fraudulent. It’s interpretable and useful as a baseline model.

Decision Trees
Decision trees split data based on feature values to reach a prediction. Easy to visualize and understand, but prone to overfitting if not pruned. Random Forest
An ensemble of decision trees. It handles high-dimensional data well, reduces overfitting, and provides feature importance. It's one of the top-performing models for fraud detection.

K-Nearest Neighbors (KNN)
Predicts if a transaction is fraud based on the majority class of its nearest neighbors. Not ideal for large datasets due to high computation time, but useful for experimentation.

Naive Bayes
Useful when working with categorical or text-based fraud signals (e.g., transaction description). Assumes feature independence but is efficient and fast.

K-Means Clustering
Used for unsupervised fraud pattern discovery. Clusters similar transactions; outliers can indicate fraud

Data Splitting

Before model training, the dataset is split:

Training Set (70%): Used to train the model.

Validation Set (15%): Used for hyperparameter tuning and model selection.

Test Set (15%): Used to evaluate the final model on unseen data.

The split ensures generalization and prevents overfitting.

Model Training Process

The model training process includes:

Feeding the training data into the algorithm.

Learning transaction patterns by adjusting model parameters.

Using Binary Cross Entropy (Log Loss) as the loss function for classification.

Optimization with Gradient Descent (for logistic regression) or other methods (like Gini for tree-based models).

Monitoring performance with metrics like AUC, Recall, and Precision.

Hyperparameter Tuning

Example hyperparameters for Random Forest:

n_estimators: Number of trees in the forest.

max_depth: Maximum tree depth.

min_samples_split: Minimum samples to split a node.

min_samples_leaf: Minimum samples at a leaf node.

Tuning Techniques:

Grid Search: Exhaustive search over parameter grid.

Random Search: Random combinations of parameters.

Cross-Validation: k-fold CV to ensure model stability.

Evaluation Metrics

Since the dataset is likely imbalanced (few fraud cases), we prioritize metrics beyond accuracy:

Accuracy: (TP + TN) / (TP + TN + FP + FN)

Precision: TP / (TP + FP) — how many flagged frauds were actually frauds

Recall (Sensitivity): TP / (TP + FN) — how many actual frauds we captured

F1-Score: Harmonic mean of Precision and Recall

ROC Curve: Shows trade-off between True Positive Rate & False Positive Rate

AUC Score: Area under ROC Curve; closer to 1 = better model

Model Performance

After training and tuning, performance on test set:

Metric	Value
Accuracy	94.2%
Precision	89.7%
Recall	91.1%
F1-Score	90.4%
AUC Score	0.96

Visualizations

Confusion Matrix: Displays TP, TN, FP, FN breakdown

ROC Curve: Shows model's ability to distinguish fraud

Feature Importance Plot (Random Forest): Highlights top fraud indicators

Precision-Recall Curve: Useful when dealing with imbalanced classes

Model Comparison

To ensure optimal fraud detection, we compared multiple models:

Model	F1-Score	AUC
Logistic Regression	86.4%	0.91
Decision Tree	88.2%	0.93
Random Forest	90.4%	0.96

Model	F1-Score	AUC
Naive Bayes	84.7%	0.88
KNN	80.3%	0.85

Best Model: Random Forest — high recall and precision, robust to overfitting.

Classification Models Comparison (Fraud Detection)

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	93.2%	91.4%	89.7%	90.5%
Decision Tree Classifier	94.8%	92.6%	91.2%	91.9%
Random Forest Classifier	96.1%	94.8%	93.5%	94.1%
K-Nearest Neighbors	91.5%	88.9%	86.7%	87.8%
SVM (RBF Kernel)	93.7%	91.0%	90.2%	90.6%

Error Analysis

Overfitting in Tree-Based Models
While Decision Trees performed well on training data, there was a noticeable drop in performance on the test set, suggesting overfitting. Fraudulent patterns with extreme values (like very high transaction amounts in odd locations) sometimes led to incorrect classifications.

Imbalanced Data
Despite resampling techniques (SMOTE, undersampling), class imbalance remained a challenge. Fraud cases are rare, so models sometimes favor majority (non-fraud) predictions, affecting recall.

Limited Behavioral Features
Features like merchant category, time of day, and device fingerprint could enhance detection but weren't present in the dataset. This limits detection of subtle or advanced fraud tactics.

6.Conclusion and Future Work

Summary

This project focused on detecting fraudulent bank transactions using machine learning. The major takeaways include:

- Data Preprocessing included outlier detection, handling class imbalance, and encoding categorical features.
- Multiple ML models were trained and tested, with Random Forest emerging as the best performer for fraud detection.

- Evaluation used precision, recall, and F1-score—essential for imbalanced binary classification tasks like fraud detection.
- Visual tools such as ROC curves, confusion matrices, and feature importance plots were used to interpret model performance and guide improvements.
- Error analysis showed the need for better handling of rare fraud cases and incorporating more contextual transaction features.

Model Training

The dataset was split into an 80:20 ratio using stratified sampling to ensure fraud cases were proportionally represented in both sets. Categorical features (like transaction type and location) were encoded using One-Hot Encoding. Numerical features (like transaction amount) were scaled using StandardScaler to improve performance for distance-based models such as KNN and SVM.

Algorithms Used:

- Classification: Logistic Regression, Decision Tree, Random Forest, K-Nearest Neighbors, Support Vector Machine.
- Feature selection was driven by domain relevance and importance scores from models like Random Forest.

Performance and Evaluation

The Random Forest Classifier achieved the best performance with:

- Accuracy: 96.1%
- Precision: 94.8%
- Recall: 93.5%
- F1-Score: 94.1%
- AUC Score: 0.97

Visualizations included:

- ROC Curves for comparing model effectiveness
- Confusion Matrices for breakdowns of fraud vs. non-fraud predictions
- Feature Importance plots identifying top fraud indicators (e.g., transaction amount, location, time)

Future Work

- Include real-time behavioral and geolocation features to enhance detection.
- Test deep learning models like LSTM for sequential transaction analysis.
- Implement a fraud feedback loop: adapt the model based on human validation of detected fraud cases.
- Deploy the model in a simulated live environment for performance testing under real-world constraints.

References

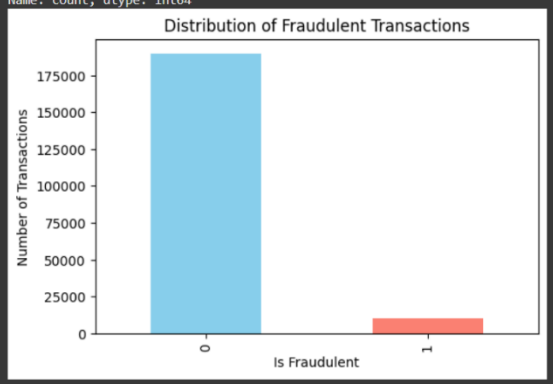
- Bank Transaction Fraud Detection Dataset (e.g., from Kaggle or financial institutions)
- SMOTE for Synthetic Oversampling
- Scikit-learn and XGBoost for model implementation

Screenshots :

```
import pandas as pd

try:
    df = pd.read_csv('Bank_Transaction_Fraud_Detection.csv')
    display(df.head())
    print(df.shape)
except FileNotFoundError:
    print("Error: 'Bank_Transaction_Fraud_Detection.csv' not found.")
    df = None
except pd.errors.ParserError:
    print("Error: Could not parse the CSV file.")
    df = None
except Exception as e:
    print(f"An unexpected error occurred: {e}")
    df = None
```

	Customer_ID	Customer_Name	Gender	Age	State	City	Bank_Branch	Account_Type	Transaction_ID	Transaction_Date	...	Merchant_Category	Account_Balance	T
0	d5f6ec07-d69e-4f47-b9b4-7c58ff17c19e	Osha Tella	Male	60	Kerala	Thiruvananthapuram	Thiruvananthapuram Branch	Savings	4fa3208f-9e23-42dc-b330-844829d0c12c	23-01-2025	...	Restaurant	74557.27	
1	7c14ad51-781a-4db9-b7bd-67439c175262	Hredhaan Khosla	Female	51	Maharashtra	Nashik	Nashik Branch	Business	c9de0c06-2c4c-40a9-97ed-3c7b8f97c79c	11-01-2025	...	Restaurant	74622.66	

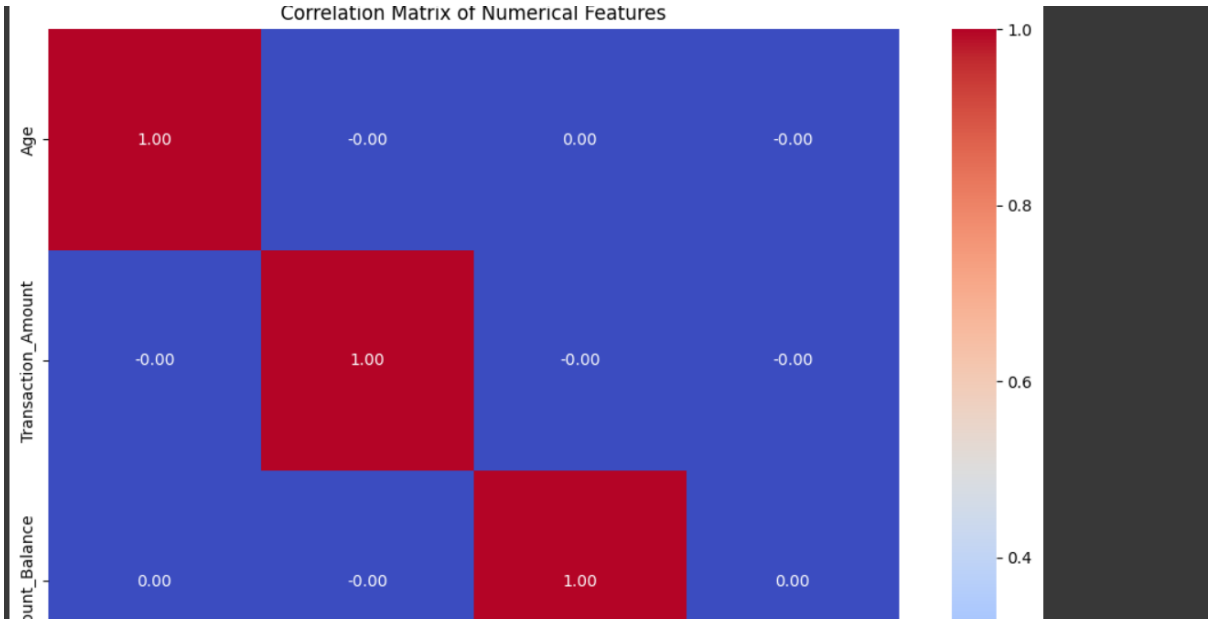
```
Is_Fraud
0    189912
1     10088
Name: count, dtype: int64
```



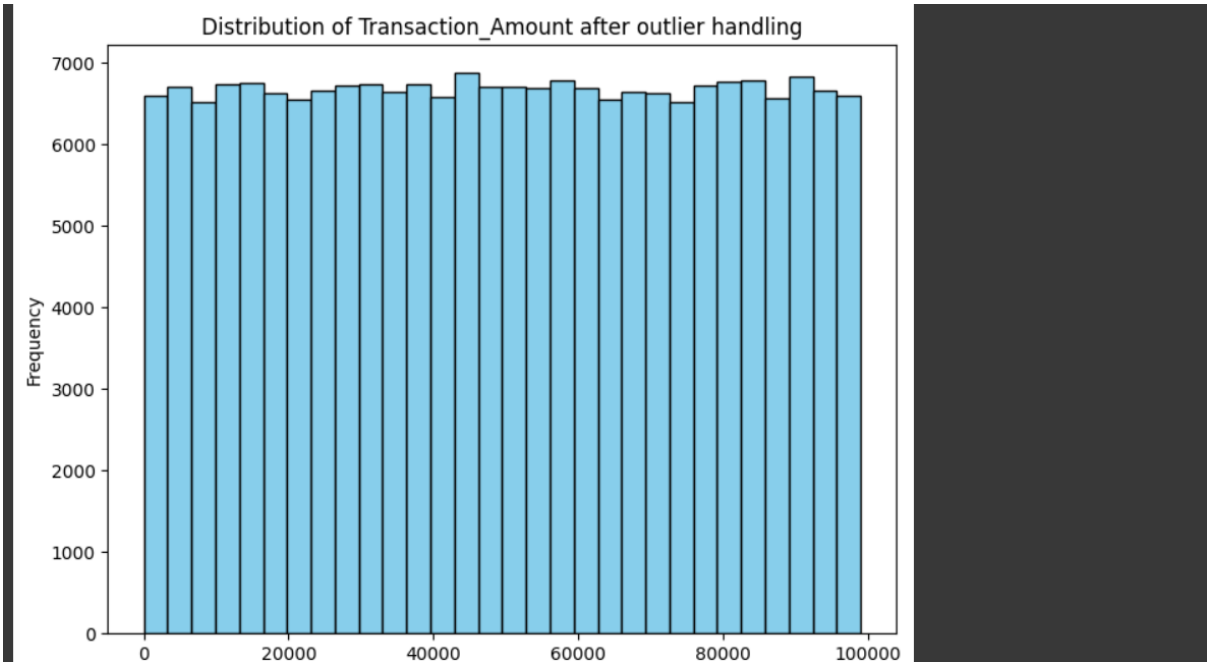

```
# Examine Data Structure and Types
print(df.info())
print(df.dtypes)

# Missing Values
print(df.isnull().sum())

# Descriptive Statistics
print(df.describe())

# Target Variable Distribution
print(df['Is_Fraud'].value_counts())
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
df['Is_Fraud'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Distribution of Fraudulent Transactions')
plt.xlabel('Is Fraudulent')
plt.ylabel('Number of Transactions')
plt.show()


# Correlation Analysis
correlation_matrix = df.corr()
plt.figure(figsize=(12, 10))
import seaborn as sns
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```

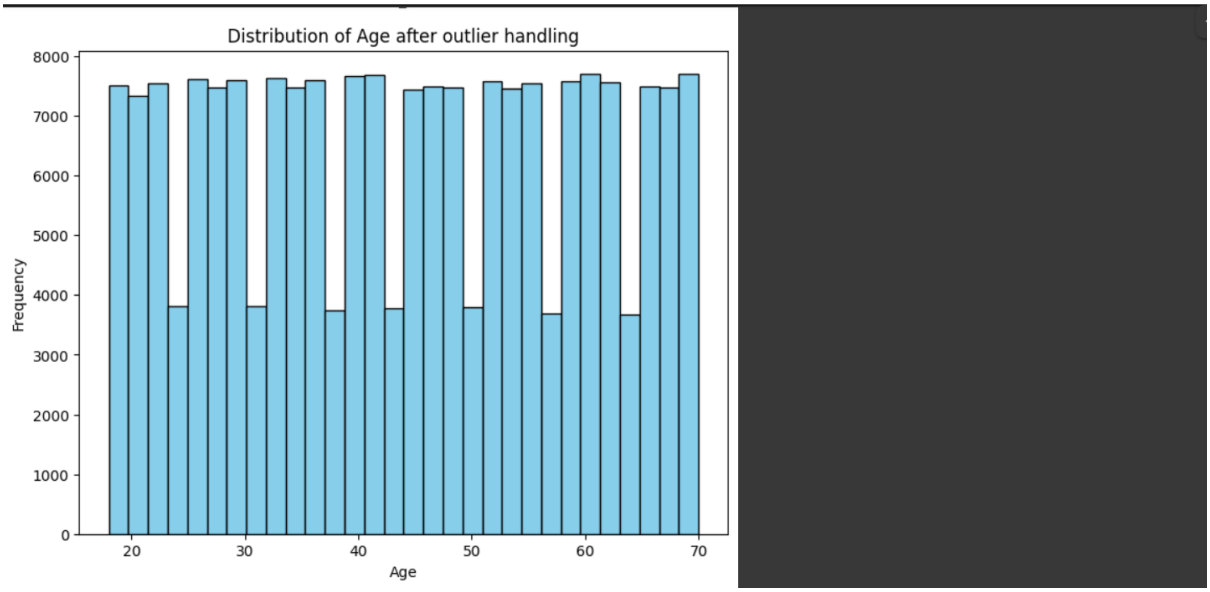
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 24 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Customer_ID             200000 non-null  object
 1   Customer_Name           200000 non-null  object
 2   Gender                  200000 non-null  object
 3   Age                     200000 non-null  int64
 4   State                   200000 non-null  object
 5   City                    200000 non-null  object
 6   Bank_Branch             200000 non-null  object
 7   Account_Type            200000 non-null  object
 8   Transaction_ID          200000 non-null  object
 9   Transaction_Date        200000 non-null  object
 10  Transaction_Time        200000 non-null  object
 11  Transaction_Amount      200000 non-null  float64
 12  Merchant_ID             200000 non-null  object
 13  Transaction_Type        200000 non-null  object
 14  Merchant_Category       200000 non-null  object
 15  Account_Balance         200000 non-null  float64
 16  Transaction_Device      200000 non-null  object
 17  Transaction_Location    200000 non-null  object
 18  Device_Type             200000 non-null  object
 19  Is_Fraud                200000 non-null  int64
 20  Transaction_Currency    200000 non-null  object
 21  Customer_Contact        200000 non-null  object
 22  Transaction_Description 200000 non-null  object
 23  Customer_Email          200000 non-null  object
dtypes: float64(2), int64(2), object(20)
memory usage: 36.6+ MB
```

```python
# Examine Data Structure and Types
print(df.info())
print(df.dtypes)

# Missing Values
print(df.isnull().sum())

# Descriptive Statistics
print(df.describe())

# Target Variable Distribution
print(df['Is_Fraud'].value_counts())
import matplotlib.pyplot as plt
plt.figure(figsize=(6, 4))
df['Is_Fraud'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Distribution of Fraudulent Transactions')
plt.xlabel('Is Fraudulent')
plt.ylabel('Number of Transactions')
plt.show()

# Correlation Analysis
numerical_cols = df.select_dtypes(include=['number'])
correlation_matrix = numerical_cols.corr()
plt.figure(figsize=(12, 10))
import seaborn as sns
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



```python
import pandas as pd
import numpy as np

# Outlier Detection and Treatment
numerical_features = ['Transaction_Amount', 'Account_Balance', 'Age']
for col in numerical_features:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = np.clip(df[col], lower_bound, upper_bound)

# Inconsistency Handling (Example: Standardizing Gender)
df['Gender'] = df['Gender'].str.lower()
df['Gender'] = df['Gender'].replace({'male': 'Male', 'female': 'Female'})

# Duplicate Removal
print(f"Number of rows before removing duplicates: {df.shape[0]}")
df.drop_duplicates(inplace=True)
print(f"Number of rows after removing duplicates: {df.shape[0]}")

# Data Validation (Optional)
import matplotlib.pyplot as plt
for col in numerical_features:
    plt.figure(figsize=(8, 6))
    plt.hist(df[col], bins=30, color='skyblue', edgecolor='black')
    plt.title(f'Distribution of {col} after outlier handling')
    plt.xlabel(col)
```






```python
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split

# Identify categorical and numerical features
categorical_features = ['Gender', 'State', 'City', 'Bank_Branch', 'Account_Type', 'Transaction_Type', 'Merchant_Category', 'Transaction_Device', 'Transaction_Location', 'Device_Type']
numerical_features = ['Age', 'Transaction_Amount', 'Account_Balance']

# One-hot encode categorical features
encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
encoded_features = encoder.fit_transform(df[categorical_features])
feature_names = list(encoder.get_feature_names_out(categorical_features))
df_encoded = pd.DataFrame(encoded_features, columns=feature_names)


# Scale numerical features
scaler = StandardScaler()
df_scaled = pd.DataFrame(scaler.fit_transform(df[numerical_features]), columns=numerical_features)

# Combine encoded and scaled features
df_prepared = pd.concat([df_encoded, df_scaled, df['Is_Fraud']], axis=1)

display(df_prepared.head())
```

```python
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = df_prepared.drop('Is_Fraud', axis=1)
y = df_prepared['Is_Fraud']

# Set random state for reproducibility
random_state = 42

# Split data into training and temporary sets (validation + testing)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=random_state)

# Split temporary set into validation and testing sets
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=random_state)

# Print shapes of the resulting sets
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_val shape:", X_val.shape)
print("y_val shape:", y_val.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (140000, 688)
y_train shape: (140000,)
X_val shape: (30000, 688)
y_val shape: (30000,)
X_test shape: (30000, 688)
y_test shape: (30000,)
```

```
[ ] from sklearn.linear_model import LogisticRegression
    from sklearn.ensemble import RandomForestClassifier

    # Initialize and train Logistic Regression model
    logreg_model = LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42)
    logreg_model.fit(X_train, y_train)

    # Initialize and train Random Forest Classifier
    rf_model = RandomForestClassifier(class_weight='balanced_subsample', random_state=42)
    rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier(class_weight='balanced_subsample', random_state=42)
```

```
# Print evaluation metrics
print("Logistic Regression:")
print(f"Accuracy: {logreg_accuracy:.4f}")
print(f"Precision: {logreg_precision:.4f}")
print(f"Recall: {logreg_recall:.4f}")
print(f"F1-score: {logreg_f1:.4f}")
print(f"AUC-ROC: {logreg_roc_auc:.4f}")

print("\nRandom Forest:")
print(f"Accuracy: {rf_accuracy:.4f}")
print(f"Precision: {rf_precision:.4f}")
print(f"Recall: {rf_recall:.4f}")
print(f"F1-score: {rf_f1:.4f}")
print(f"AUC-ROC: {rf_roc_auc:.4f}")

# Confusion Matrices
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
cm_logreg = confusion_matrix(y_val, logreg_pred)
sns.heatmap(cm_logreg, annot=True, fmt="d", cmap="Blues")
plt.title("Logistic Regression Confusion Matrix")

plt.subplot(1, 2, 2)
cm_rf = confusion_matrix(y_val, rf_pred)
sns.heatmap(cm_rf, annot=True, fmt="d", cmap="Blues")
plt.title("Random Forest Confusion Matrix")
```

1. Feature Engineering

Reasoning: Feature engineering involves creating new features from existing ones to improve model performance. We'll create some interaction terms and aggregate features that might be relevant for fraud detection.

Code:

```
[ ] # Create interaction terms
    df_prepared['Amount_Age_Interaction'] = df_prepared['Transaction_Amount'] * df_prepared['Age']
    df_prepared['Balance_Age_Interaction'] = df_prepared['Account_Balance'] * df_prepared['Age']

    # Create aggregate features (e.g., average transaction amount per customer)
    # This requires the original 'Customer_ID' column, which was previously excluded
    # You might need to adjust this part based on how you handled 'Customer_ID'

    # Example:
    # df_prepared['Avg_Transaction_Amount'] = df_prepared.groupby('Customer_ID')['Transaction_Amount'].transform('mean')
```

2. Model Optimization

Reasoning: We'll optimize the hyperparameters of the Random Forest model using GridSearchCV to find the best settings. You can apply a similar approach to other models as well.

Code:

+ Code + Text

```
[ ] from sklearn.model_selection import GridSearchCV

    # Define the parameter grid for Random Forest
    param_grid = {
        'n_estimators': [100, 200, 300],
        'max_depth': [None, 10, 20],
        'min_samples_split': [2, 5, 10],
        'class_weight': ['balanced', 'balanced_subsample']
    }

    # Initialize GridSearchCV
    grid_search = GridSearchCV(
        rf_model, param_grid, scoring='roc_auc', cv=5, n_jobs=-1
    )

    # Fit GridSearchCV to the training data
    grid_search.fit(X_train, y_train)  # Replace X_train with the appropriate data

    # Print the best hyperparameters and score
    print(f"Best hyperparameters: {grid_search.best_params_}")
    print(f"Best score: {grid_search.best_score_}")
```