Fraud Detection System for Financial Transactions – Final Report

Problem Statement:

Financial institutions lose billions due to fraudulent transactions. The goal of this project is to detect fraud in real-time using historical transaction data and machine learning classification models. The solution should be able to predict potential fraud accurately and efficiently.

Tech Stack:

- Python
- Pandas, NumPy
- Matplotlib, Seaborn
- Scikit-learn, XGBoost
- Jupyter Notebook
- Streamlit
- Joblib (for model saving)

Data Overview:

- Source: Kaggle – Credit Card Fraud Detection dataset
- Total Rows: ~284,807 transactions
- Features: Time, Amount, V1–V28 (PCA components)
- Target: Class (0 = Non-Fraud, 1 = Fraud)
- Data Imbalance: Extremely skewed (fraud < 0.2%)

Data Handling:

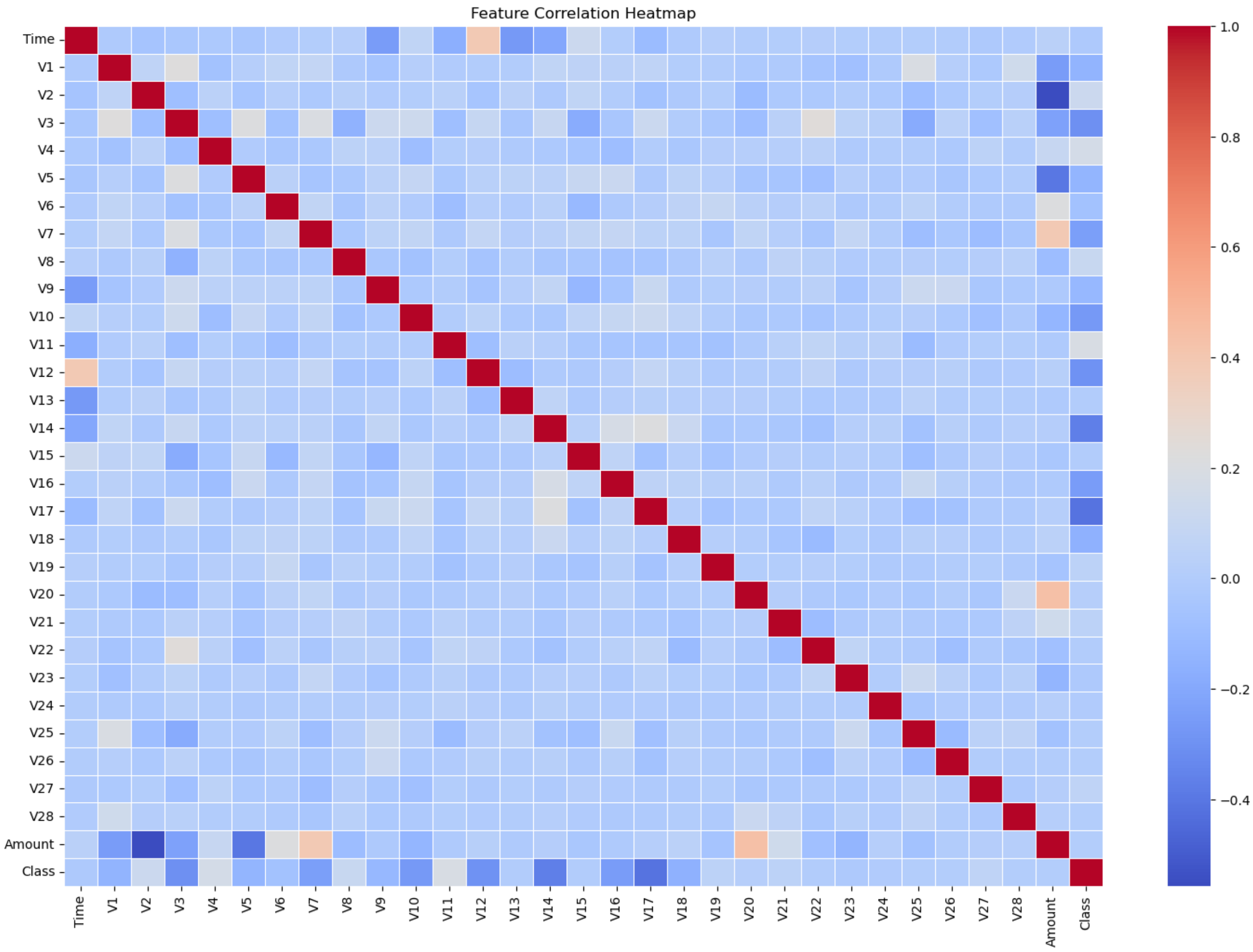
- Dropped columns irrelevant for modeling
- Scaled numerical columns ('Time' and 'Amount') using StandardScaler
- Created a balanced dataset using undersampling (Fraud : Non-Fraud = 1:5)

Exploratory Data Analysis:

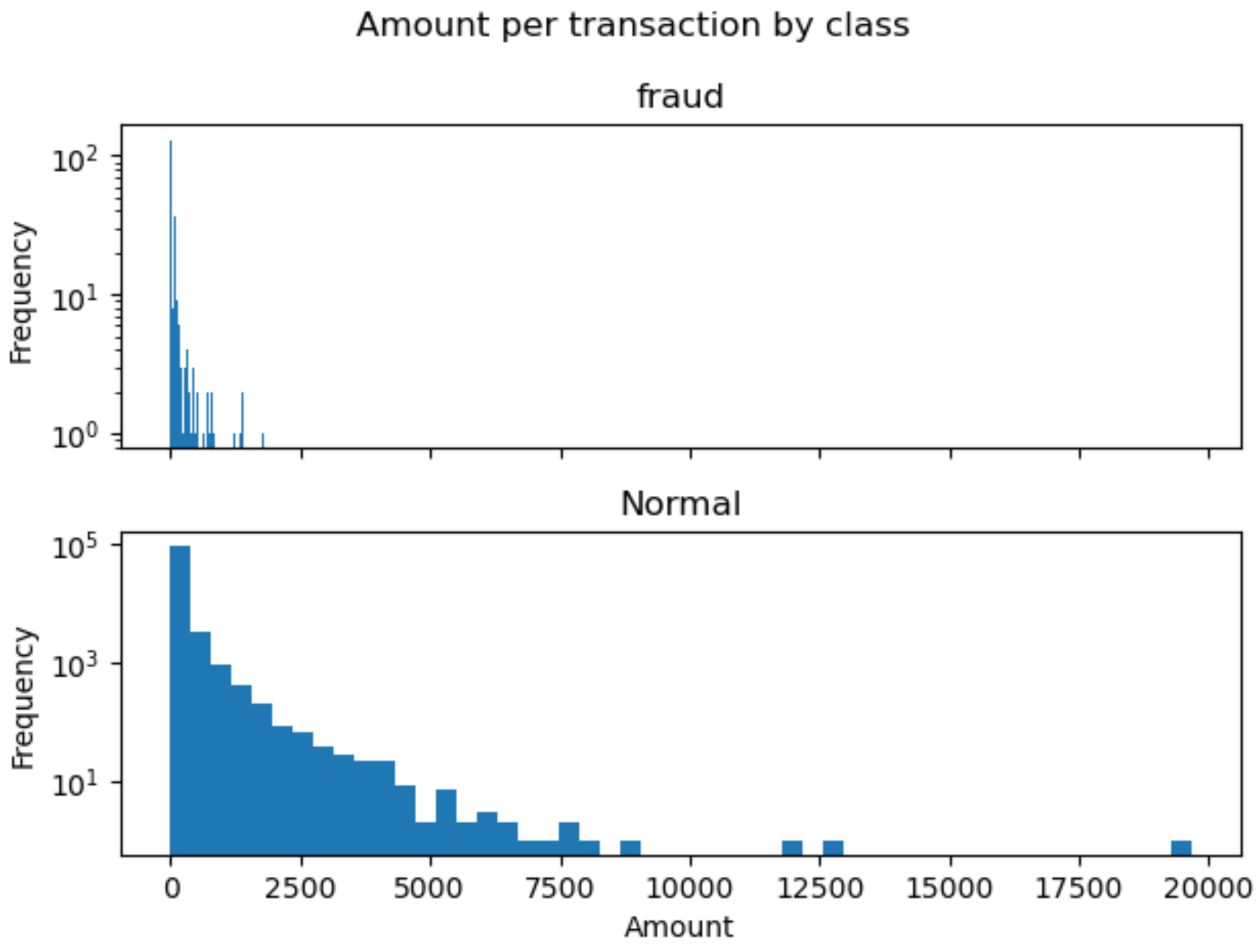
1. **Target Distribution**: Severe class imbalance observed.

2. **Amount & Time**: Log-scaled plots showed different patterns for fraud vs. non-fraud.

3. **Heatmap**: Showed that features V10, V14, and V17 had strongest correlations with fraud.

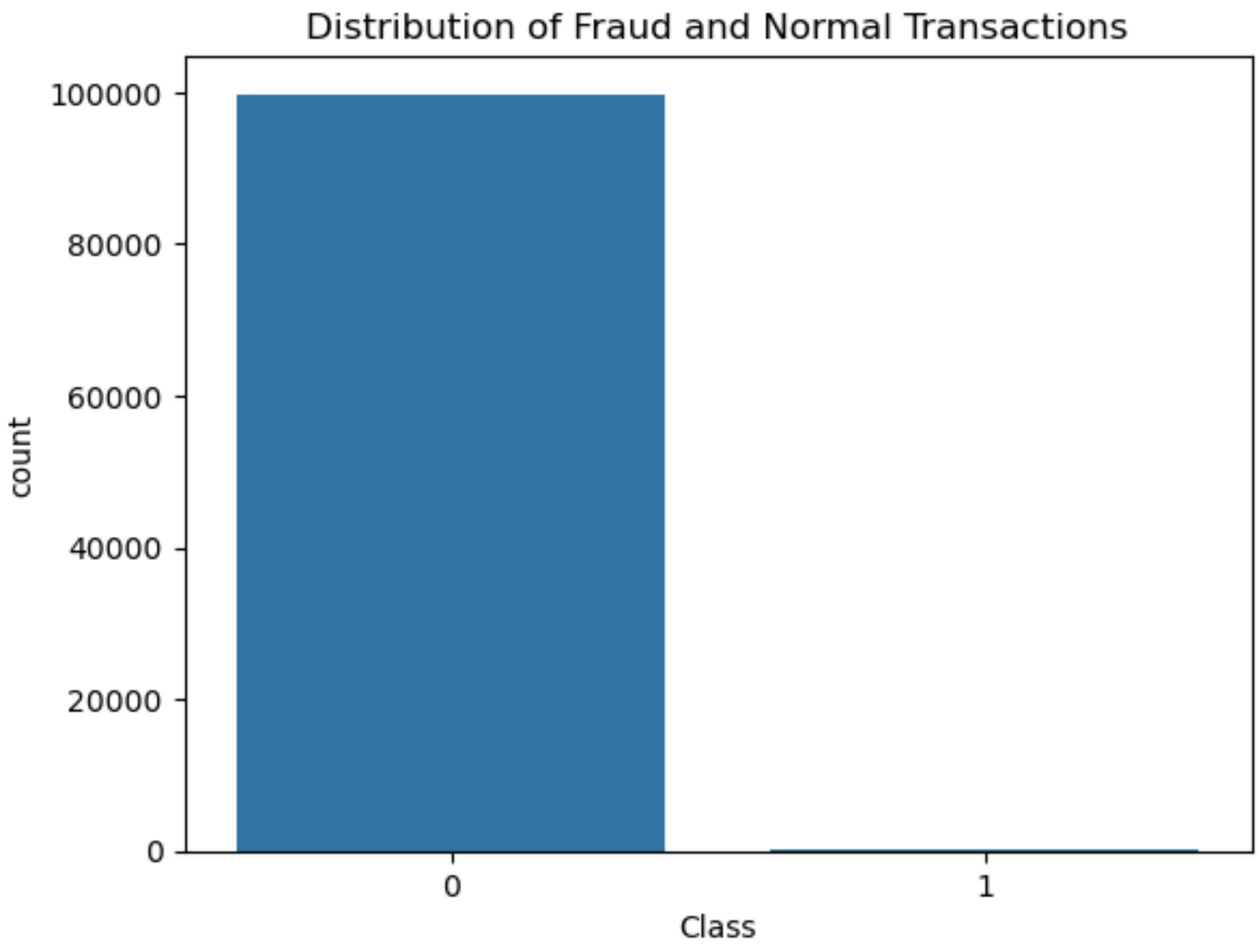


4. Amount per Transaction by Class:



- Logarithmic scale was used to highlight frequency differences.

5. **Fraud vs Non-Fraud Distribution**: Clear difference in transaction patterns.



Feature Engineering:

- Applied StandardScaler to 'Time' and 'Amount'
- Retained V1–V28 as they were already PCA-transformed

Model Training and Evaluation:

Three models were trained:

1. **Logistic Regression**
    - Baseline linear model
    - Performance limited by complex data structure

```
=== Logistic Regression ===
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       223
           1       0.98      0.93      0.95        45

    accuracy                           0.99       268
   macro avg       0.98      0.96      0.97       268
weighted avg       0.99      0.99      0.98       268

AUC-ROC Score: 0.964424514200299

=== Decision Tree ===
              precision    recall  f1-score   support
```

2. **Decision Tree Classifier**
   - Captures non-linear patterns
   - Tuned with max depth and minimum samples per split

```
AUC-ROC Score: 0.964424514200299

=== Decision Tree ===
              precision    recall  f1-score   support

           0       0.99      0.96      0.98       223
           1       0.84      0.93      0.88        45

    accuracy                           0.96       268
   macro avg       0.91      0.95      0.93       268
weighted avg       0.96      0.96      0.96       268

AUC-ROC Score: 0.9487294469357249
```

3. **XGBoost Classifier**
   - Best performance
   - Tuned with GridSearchCV (n_estimators, max_depth, learning_rate)

```
=== XGBoost ===
...
weighted avg       0.99      0.99      0.99       268

AUC-ROC Score: 0.9777777777777779
```

Metrics Used:

- F1 Score
- Precision
- Recall
- ROC-AUC Score

Best Model: XGBoost (ROC-AUC > 0.98, High F1 Score)

Deployment – Streamlit App:

- Created a lightweight frontend using Streamlit
- Model loaded with joblib (`fraud_model.pkl`)
- User inputs transaction data
- App displays real-time prediction (Fraud/Not Fraud)

Risks & Limitations:

- **Class Imbalance**: Even after balancing, risk of model bias remains.
- **False Negatives**: Undetected frauds are critical risks in finance.
- **PCA Components (V1–V28)**: No interpretability of features.
- **Real-Time Data**: Streamlit app uses static features, lacks user behavior history.

Conclusion:

This project successfully demonstrates the use of machine learning for fraud detection. The XGBoost model provided highly accurate results. With a lightweight frontend built in Streamlit, the system is capable of performing real-time fraud classification on user-input data.

The system can be extended with real-world transactional attributes, richer user profiling, and continuous learning pipelines in future iterations.