Machine Learning for Credit Card Fraud Detection

Arnav Mehra
Department of Statistics
University of Michigan
Ann Arbor, USA
arnmehra@umich.edu

***Abstract*—Credit card fraud/anomaly detection is an important application of machine learning in the financial services industry. In this project, I have executed a complete data science project for fraud/anomaly detection, from raw transaction data to database design, feature engineering, model deployment, and evaluation. Using a dataset of 1.85 million+ credit card transactions, I implemented a normalized SQL relational database with a snowflake schema to manage personally identifiable information (PII) and created three machine learning models: Logistic Regression, Random Forest, and Isolation Forest. My Logistic Regression model achieved a 75.7% fraud detection rate, demonstrating the effectiveness of supervised learning when labeled data is available. While unsupervised learning alone is insufficient for fraud/anomaly detection, I also discuss potential solutions for scenarios where we do not have access to labelled data.**

***Index Terms*—fraud detection, anomaly detection, machine learning, supervised learning, unsupervised learning, database design, feature engineering, class imbalance**

I. Introduction

Credit card fraud poses a significant financial threat to banks, credit card companies, and their customers. Detecting fraudulent transactions requires processing large-scale transaction data, while balancing fraud detection accuracy against customer experience. This project demonstrates a complete data science workflow for building fraud detection models using Python and SQL.

A. Project Objectives

This project aims to: (1) Build a scalable data pipeline for managing transaction data with proper PII controls, (2) Implement supervised and unsupervised machine learning models for fraud detection, and (3) Evaluate model performance on severely imbalanced data (0.52% fraud rate).

B. Key Contributions

I provide: (1) A normalized SQL relational database design with snowflake schema for managing sensitive transaction data, (2) Comprehensive feature engineering pipeline to handle categorical and numerical features, and (3) Implementation of three fraud detection models with detailed performance analysis.

C. Related Work

Credit card fraud detection has been extensively studied using machine learning approaches. Recent literature demonstrates that Logistic Regression and ensemble methods, like Random Forest, remain effective baselines for this problem [1]. For unsupervised scenarios, Isolation Forest has emerged as a computationally efficient anomaly detection method with $O(n \log n)$ complexity [1].

Given severe class imbalance in transaction datasets (typically $<1\%$ fraud rate), standard accuracy metrics are misleading [2]. Recent studies emphasize evaluating models using Precision, Recall, and F1-score, which provide more meaningful assessment of minority class detection performance [2].

This project follows these established practices.

II. Method

Please refer to my GitHub [3] for a more detailed overview of this project and to view the complete project code.

A. Dataset

I utilized the Credit Card Transaction dataset from Hugging Face [4], consisting of two CSV files: 1,296,675 transactions from January 2019 - June 2020 and 555,719 transactions from June 2020 - December 2020, totaling 1,852,394 transactions. Each record contains 23 features, such as transaction amount, merchant category, customer demographics (age, gender, occupation), and geospatial coordinates for both customer and merchant locations.

B. Exploratory Data Analysis

Initial analysis revealed:

Class Imbalance: Only 9,651 fraudulent transactions (0.52%) versus 1,842,743 legitimate transactions (99.48%), which requires specialized class imbalance handling techniques.

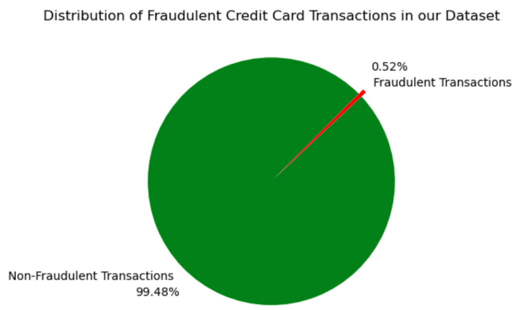


Fig. 1: Class distribution showing severe imbalance: 99.48% legitimate vs 0.52% fraudulent transactions.

Transaction Amounts: Distributions were heavily right-skewed (max: \$28,948.90, median: \$47.24). Fraudulent transactions showed higher median amounts (\$390.00) compared to legitimate transactions (\$47.24) but lower maximum values (\$1,376.04 vs \$28,948.90), validating the hypothesis that fraudsters avoid single large purchases to evade detection.

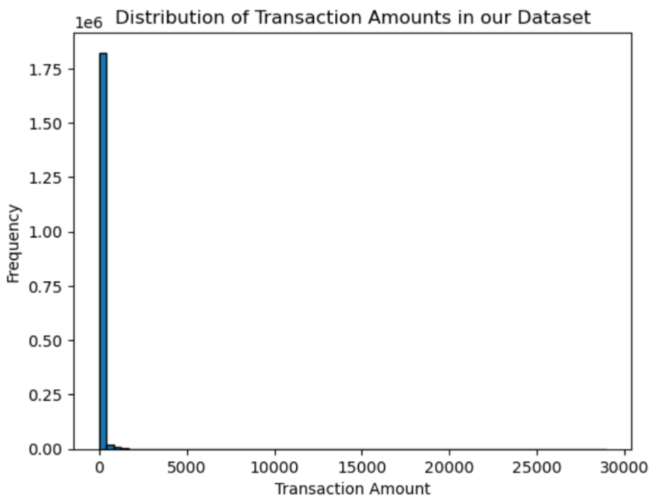


Fig. 2: Distribution of transaction amounts showing heavy right skew with median of \$47.24 and maximum of \$28,948.90.

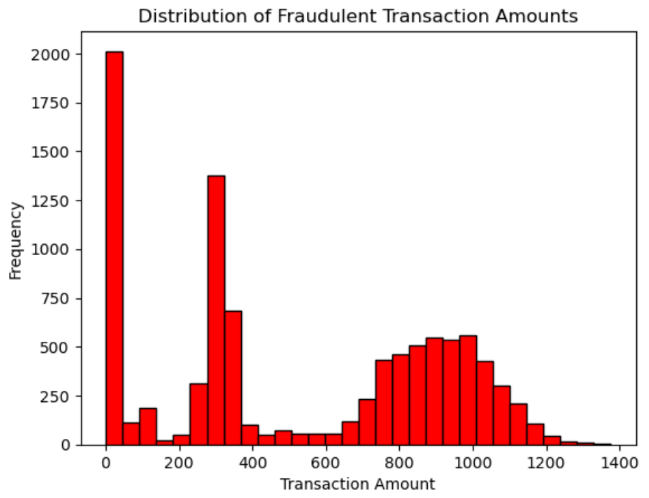


Fig. 3: Distribution of fraudulent transaction amounts (median: \$390.00, max: \$1,376.04) showing fraudsters avoid large single purchases.

Missing Data: The merchant zipcode field had 279,956 missing values (15.1%). Analysis showed equal missing rates across fraud (14.7%) and non-fraud (15.1%) classes, indicating that the missing values were missing completely at random. I retained these records, using merchant latitude/longitude as geospatial features instead.

C. Database Design

Rather than using a single flat table, I implemented a normalized [5] relational database [6] using SQLite to:

1) **Separate PII from analytics data** for access control
2) **Reduce data redundancy** through normalization
3) **Enable flexible querying** via SQL views
4) **Simulate real-world data infrastructure** where analysts query databases rather than working with CSV files

Snowflake Schema: I designed a snowflake schema [7], as opposed to a star schema [8] with one fact table, two dimension tables, and one additional dimension table:

- **TransactionDetails** (fact table): trans_num (PK), trans_date_trans_time, cc_num (FK), merchant_id (FK), category, amt, unix_time, is_fraud
- **CustomerInformation** (dimension table): cc_num (PK/FK), first, last, gender, job, dob
- **CustomerAddressInformation** (additional dimension table): cc_num (PK), street, city, state, zip, lat, long, city_pop
- **MerchantInformation** (dimension table): merchant_id (PK), merchant, merch_lat, merch_long, merch_zipcode, missing_merch_zipcode

I created a merchant_id primary key by concatenating merchant name and category fields to uniquely identify each merchant-category combination.

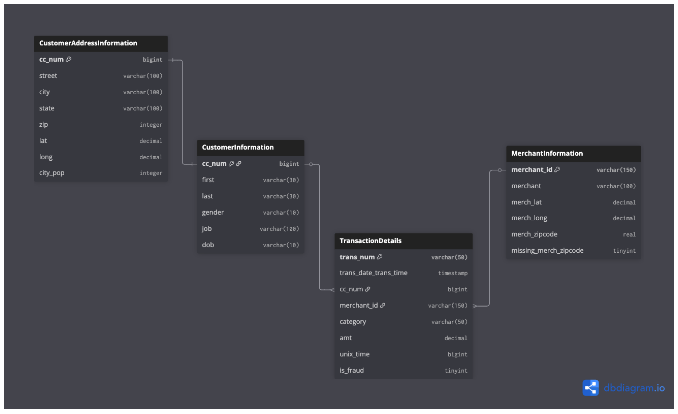


Fig. 4: Snowflake schema showing TransactionDetails fact table connected to CustomerInformation, CustomerAddressInformation, and MerchantInformation dimension tables.

ML View: I created an SQL view by joining relevant tables, while excluding PII fields (first, last, street) and redundant features (unix_time, merchant_id, merch_zipcode). This view contained 19 feature columns, which is suitable for machine learning, while maintaining the separation between analytics and sensitive data.

D. Feature Engineering

One-Hot Encoding: Applied one-hot encoding [9] to low-cardinality features: category (14 values), gender (2 values), and state (51 values). I used $drop\_first = True$ to avoid multicollinearity.

Label Encoding: Applied label encoding [9] to high-cardinality features to avoid the curse of dimensionality: job (497 unique occupations) and merchant (693 unique merchants).

Temporal Features: From transaction timestamps, I extracted: hour (0-23), month (1-12), and day (1-31). These capture temporal fraud patterns, such as unusual transaction hours.

Age Calculation: Converted date of birth to age as of December 31, 2020 (dataset end date).

Log Transformation: Applied $log(amt)$ to transaction amounts to address the right-skewed distribution. Since minimum amount was \$1.00, $log1p$ was unnecessary.

Standardization: All numerical features (amt_log, lat, long, city_pop, merch_lat, merch_long, age, hour, month, day, job_encoded, merchant_encoded) were standardized using StandardScaler to ensure equal contribution to model learning.

E. Feature Selection

I retained geospatial features (latitude, longitude, city population) while excluding: transaction/customer identifiers (trans_num, cc_num), redundant location fields (city, zip given lat/long availability), and the missing zipcode flag. The final feature set contained 76 dimensions.

F. Train-Test Split

I employed an 80-20 train-test split (1,481,915 training / 370,479 test samples) with stratification to maintain the 0.52% fraud rate in both sets. This approach was chosen over k-fold cross-validation due to computational constraints and the large dataset size providing sufficient statistical power with a single split.

G. Handling Class Imbalance

Rather than synthetic oversampling (SMOTE) or undersampling, I used built-in class weighting:

Supervised Models: $class\_weight =' balanced'$ parameter automatically penalizes minority class misclassification more heavily.

Isolation Forest: $contamination = 0.0052$ informed the model that 0.52% of data represents anomalies.

H. Model Selection

Logistic Regression: Logistic Regression [10] was chosen as a simple, interpretable baseline for binary classification with built-in probability estimates.

Random Forest: Random Forest [11] was selected for high-dimensional data handling and feature importance analysis. I trained 200 trees with $class\_weight =' balanced'$. Random Forest was preferred over XGBoost due to faster training times given project constraints.

Isolation Forest: Isolation Forest [12] was implemented to simulate real-world scenarios without labeled data. Isolation Forest has $O(n \log n)$ complexity versus $O(n^2)$ for Local Outlier Factor, making it more suitable for 1.85M+ transactions.

All models used default hyperparameters as parameter tuning was beyond project scope.

III. Results

A. Evaluation Metrics

Given the high cost of false negatives (undetected fraud), I prioritized **Recall** (True Positive Rate):

$$Recall = \frac{TP}{TP + FN} = 1 - FNR \tag{1}$$

where TP = true positives (detected fraud), FN = false negatives (missed fraud), FNR = false negative rate. I also report Precision and F1-score for a comprehensive assessment. To evaluate these metrics, I utilized a Classification Report [13] and a Confusion Matrix [14]

B. Model Performance

Table I summarizes test set performance (370,479 transactions, 1,930 fraudulent).

TABLE I: Model Performance on Test Set

Model	**Recall**	**FNR**	**Precision**
Logistic Regression	**75.70%**	**24.30%**	2%
Random Forest	71.71%	28.29%	**98%**
Isolation Forest	1.19%	98.81%	1%

Logistic Regression achieved highest recall (75.70%), correctly identifying 1,461 of 1,930 fraudulent transactions while missing 469 (lowest false negative count). However, low precision (2%) resulted in 86,321 false positives. This aggressive detection strategy minimizes the most critical error type (missed fraud) at the cost of customer inconvenience.

Random Forest demonstrated strong precision (98%) with 71.71% recall, producing only 22 false positives but missing 546 fraudulent transactions. This represents a favorable precision-recall trade-off for scenarios prioritizing customer experience.

Isolation Forest performed catastrophically (1.19% recall), missing 1,907 of 1,930 fraudulent transactions. This demonstrates that unsupervised methods alone are insufficient when labeled data is available.

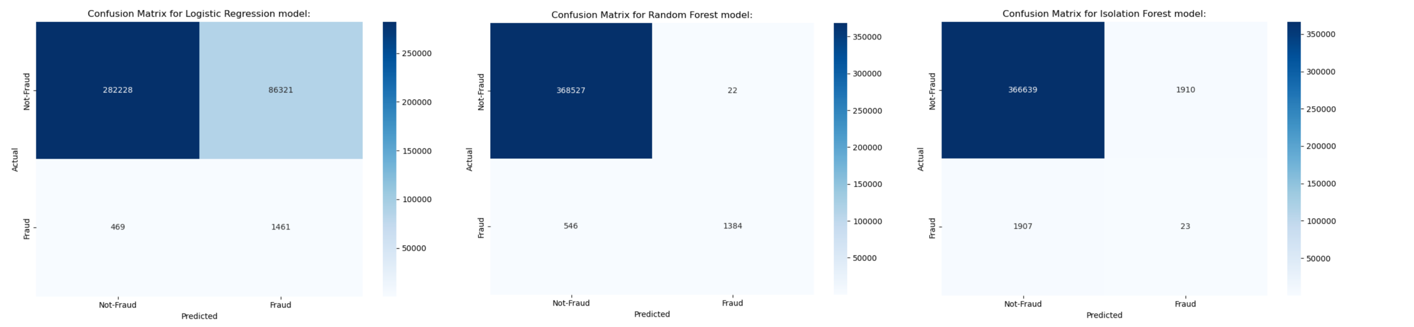
C. Business Recommendations

Logistic Regression is recommended for maximum fraud detection, accepting high false positive rates. This suits risk-averse organizations prioritizing fraud prevention.

Random Forest suits scenarios balancing fraud detection with customer experience, producing minimal false alarms while maintaining a 71.71% fraud detection rate.

Isolation Forest's Failure highlights the cold-start problem [15]. Organizations without labeled data should:

1) Deploy rule-based systems initially (typically 20-30% detection: amount $>$ \$5000, international transactions, unusual hours)
2) Use active learning to bootstrap labeled datasets via manual review and customer feedback
3) Transition to supervised models once $>$1000 labeled examples exist
4) Continuously collect labels from disputes and chargebacks for model retraining



(a) Logistic Regression (b) Random Forest (c) Isolation Forest

Fig. 5: Confusion matrices comparing test set performance (370,479 transactions, 1,930 fraudulent). Logistic Regression achieves highest recall (75.70%) with 469 false negatives, Random Forest balances precision (98%) and recall (71.71%) with only 22 false positives, while Isolation Forest fails catastrophically with 1,907 missed fraud cases.

IV. Conclusion

This project demonstrated a complete data science pipeline for fraud detection, from database design to model deployment and evaluation. Key findings include:

Database Design: A normalized snowflake schema enables PII separation, reduces redundancy, and provides flexible querying for analytics.

Supervised Superiority: Supervised models vastly outperformed unsupervised approaches when labels were available.

Precision-Recall Trade-off: Logistic Regression maximizes fraud detection while Random Forest balances detection with minimal customer friction.

Labeled Data Value: The 60x performance gap between supervised and unsupervised methods underscores the importance of label collection.

A. Future Work

Extensions include: (1) Hyperparameter tuning to optimize model performance, (2) Advanced feature engineering (customer-merchant distance, transaction velocity, time-between-transactions), and (3) Ensemble methods combining model predictions

B. Limitations

Time constraints precluded hyperparameter tuning and extensive feature engineering. I used default model parameters, which may be suboptimal. Additional features like sequential transaction patterns, distance and velocity metrics could improve fraud detection rates.

References

[1] "A Systematic Review of Machine Learning in Credit Card Fraud Detection Under Original Class Imbalance," *Computers*, vol. 14, no. 10, p. 437, Oct. 2025. [Online]. Available: https://www.mdpi.com/2073-431X/14/10/437

[2] "Enhancing credit card fraud detection with a stacking-based hybrid machine learning approach" *PubMed Central*, 2025. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC12453863/

[3] A. Mehra, "STATS 507 Final Project: Credit Card Anomaly Detection," GitHub Repository, 2025. [Online]. Available: https://github.com/arnmehra/STATS-507

[4] Pointe77, "Credit Card Transaction," Hugging Face Datasets, 2025. [Online]. Available: https://huggingface.co/datasets/pointe77/credit-card-transaction

[5] "Database Normalization," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Database_normalization

[6] "Database Schema," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Database_schema

[7] "Snowflake Schema," Databricks Glossary. [Online]. Available: https://www.databricks.com/glossary/snowflake-schema

[8] "Star Schema," Databricks Glossary. [Online]. Available: https://www.databricks.com/glossary/star-schema

[9] "Types of Categorical Data Encoding," Analytics Vidhya. [Online]. Available: https://www.analyticsvidhya.com/blog/2020/08/types-of-categorical-data-encoding/

[10] "Logistic Regression in Machine Learning," GeeksforGeeks Machine Learning. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/understanding-logistic-regression/

[11] "Random Forest Algorithm in Machine Learning," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/random-forest-algorithm-in-machine-learning/

[12] "Anomaly Detection Using Isolation Forest," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/anomaly-detection-using-isolation-forest/

[13] "Understanding a Classification Report for Your Machine Learning Model," Medium. [Online]. Available: https://medium.com/@kohlishivam5522/understanding-a-classification-report-for-your-machine-learning-model-88815e2ce397

[14] "Compute Classification Report and Confusion Matrix in Python," GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org/machine-learning/compute-classification-report-and-confusion-matrix-in-python/

[15] "Cold Start (Recommender Systems)," Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)