Fraud Detection Predicting Model

Roseanne Pham (np1029, Section 4), Esmeralda Bencosme (eb1024, Section 01)

GitHub Repo Link | Video Link

Abstract
Fraud detection systems must evolve to handle increasingly subtle and complex digital threats. Using a synthetic dataset that mimics real-world behavior, we tested several models and built a system that balances accuracy with interpretability. Our results show that this combined approach performs well and could offer a practical, scalable solution for real-world fraud detection.

1. Introduction

As online financial transactions become more complex, detecting fraud has become a growing concern for industries around the world. The growing volume of digital transactions introduces substantial risks, as conventional detection systems often struggle to maintain both accuracy and speed under dynamic conditions.

This study focuses on the development of a fraud detection system that was enhanced by supervised machine learning and manual pattern recognition, to identify suspicious transactions with high confidence.

2. Novelty and Importance:

Studies suggest that AI-driven fraud detection enhances security and reduces financial losses (Md Shakil Islam, and Nayem Rahman). By integrating real-time analytics and scalable data processing, the system aims to improve fraud detection rates while maintaining efficiency.

Accurate and efficient fraud detection is essential for maintaining trust in financial systems and protecting customers and financial institutions from financial losses. Key areas impacted by this system include:

Banking, Financial Services and E-commerce: In banking and financial services, this system helps prevent unauthorized access and fraudulent transactions, safeguarding customer assets and sensitive information while reducing financial losses from fraudulent activities. In e-commerce, it protects online merchants from chargebacks and fraudulent orders, enhancing the overall security of online transactions and improving customer trust by ensuring secure payment processing.

Insurance: For the insurance sector, fraud detection is crucial in identifying and preventing fraudulent claims, reducing the financial burden on insurance companies while ensuring fair and accurate claim processing for legitimate policyholders.

Public Sector and Government: In the public sector and government, the system can prevent fraudulent activities in public assistance programs and tax-related transactions, ensuring resources are distributed fairly and efficiently.

3. Related Work and Problem Statement

Prior research has highlighted the effectiveness of AI techniques in fraud detection. For instance, Islam and Rahman (2025) demonstrated that real-time, AI-driven systems can significantly reduce financial losses by quickly identifying suspicious activity. Despite these advances, many of these

models offer little to no interpretability, a critical drawback in fields that require transparency, like finance and law. This lack of effectiveness motivates our approach. We aim to design a system that combines machine learning with rule-based logic, allowing for both high predictive performance and greater transparency. Our question is: Can a hybrid fraud detection model that integrates algorithmic learning with rules made only by carefully observing outperform standalone methods in detecting financial fraud?

4. Methodology

We used a mix of approaches in our fraud detection system. Alongside standard supervised learning models, we added handcrafted rules based on trends we noticed in the data. Once again, the idea was to see if a hybrid system that combines both could improve fraud detection without sacrificing clarity.

4.1 Research Design

We used a synthetic dataset from Kaggle that was built to reflect the complexity of real-world financial transactions. Each entry was labeled as either genuine or fraudulent, which made it suitable for supervised learning. Our approach combined both traditional machine learning classifiers and manually constructed rules based on patterns we discovered in the data. The main goal was to compare the performance of pure algorithmic models, handcrafted logic, and a hybrid of both in detecting fraud.

To measure performance, we used common classification metrics: accuracy, precision, recall, and F1-score. For added transparency and to simulate a realistic data environment, we also structured our dataset into an SQL-based relational schema using SQLite. This allowed us to query users, transactions, and fraud flags in a more structured and insightful way. We used Python libraries like Pandas, Matplotlib, Seaborn, and Scikit-learn throughout our analysis, with SQL integration providing additional depth in data exploration.

4.2 Data Collection

The dataset includes 50,000 inputs and offers a wide set of attributes that reflect diverse behavioral, transactional, and contextual factors commonly involved in fraud analysis. These include monetary metrics (e.g., Transaction_Amount, Account_Balance), temporal factors (Timestamp, Is_Weekend), user behavior insights (Daily_Transaction_Count, Avg_Transaction_Amount_7d, Previous_Fraudulent_Activity), geolocation and device-based indicators (Location, IP_Address_Flag, Device_Type, Transaction_Distance), and categorical features such as Transaction_Type, Card_Type, Merchant_Category, and Authentication_Method. Additionally, a computed Risk_Score provides an estimate of the transaction's suspiciousness, which adds a valuable feature for modeling.

Transactior	User_ID	Transactior	Transactior	Timestamp	Account_B	Device_Typ	Location	Merchant_(	IP_Address	Previous_Fr	Daily_Trans	Avg_Transa	Failed_Tran	Card_Type	Card_Age	Transactior	Authentica	Risk_Score	Is_Weeken	Fraud_Label
TXN_3355	USER_183	39.79	POS	#########	93213.17	Laptop	Sydney	Travel	0	0	7	437.63	3	Amex	65	883.17	Biometric	0.8494	0	0
TXN_9427	USER_787	1.19	Bank Transf	#########	75725.25	Mobile	New York	Clothing	0	0	13	478.76	4	Mastercard	186	2203.36	Password	0.0959	0	1
TXN_199	USER_273	28.96	Online	#########	1588.96	Tablet	Mumbai	Restaurant	0	0	14	50.01	4	Visa	226	1909.29	Biometric	0.84	0	1
TXN_1244	USER_261	254.32	ATM Withdr	#########	76807.2	Tablet	New York	Clothing	0	0	8	182.48	4	Visa	76	1311.86	OTP	0.7935	0	1
TXN_3948	USER_201	31.28	POS	#########	92354.66	Mobile	Mumbai	Electronics	0	1	14	328.69	4	Mastercard	140	966.98	Password	0.3819	1	1
TXN_4272	USER_685	168.55	Online	#########	33236.94	Laptop	Tokyo	Restaurant	0	0	3	226.85	2	Discover	51	1725.64	OTP	0.0504	0	0
TXN_1082	USER_505	3.79	POS	#########	86834.18	Tablet	London	Restaurant	0	0	2	298.35	2	Mastercard	168	3757.19	Password	0.0875	0	0
TXN_4949	USER_466	7.08	ATM Withdr	#########	45826.27	Tablet	London	Restaurant	0	0	3	164.38	4	Discover	182	1764.66	Biometric	0.5326	0	1
TXN_4144	USER_158	34.25	ATM Withdr	#########	94392.35	Tablet	Tokyo	Clothing	0	0	7	90.02	3	Visa	24	550.38	Biometric	0.1347	1	0
TXN_3695	USER_949	16.24	POS	#########	91859.97	Mobile	Mumbai	Travel	0	0	6	474.42	1	Mastercard	124	720.91	PIN	0.3394	0	0
TXN_4310	USER_284	367.5	POS	#########	14640.09	Laptop	Mumbai	Electronics	0	0	4	397.58	0	Amex	136	292.36	PIN	0.643	0	0
TXN_3869	USER_686	50.44	ATM Withdr	#########	19962.22	Laptop	London	Travel	0	0	14	278.55	1	Discover	131	3993.62	PIN	0.4582	0	0
TXN_6188	USER_672	55.5	ATM Withdr	#########	89664.63	Laptop	London	Groceries	0	0	6	483.58	4	Discover	192	3721.54	OTP	0.5837	0	1

Because it's synthetic and to further get the unique dataset to push realism, we dirtied the data to further introduce variability that mimics real-world fraud complexity. First, we randomly flipped 3%

of the fraud labels to mimic mislabeling or human error. Outliers were injected into the Transaction_Amount column by assigning unusually high values to 1% of transactions. To reflect data incompleteness, we randomly inserted missing values (NaNs) into 10% of each numeric column. Additionally, small Gaussian noise was added to all numeric features (excluding the target variable) to prevent overfitting to overly clean patterns. We enforced realistic bounds on features such as Transaction_Amount and Risk_Score, clipping values to valid ranges and converting any interval-type values to their midpoints. Behavioral and categorical features like Is_Weekend, IP_Address_Flag, and Previous_Fraudulent_Activity were randomized using skewed probabilities to reflect realistic distributions (e.g., only 5% of transactions triggered the IP flag). These adjustments ensured that the dataset better reflected the ambiguity and irregularities commonly found in actual fraud detection environments.

This process required us to be extremely careful. Many of our features were binary flags rather than continuous variables, meaning we couldn't simply add noise, and changes had to remain logically consistent. We regenerated the synthetic data multiple times to ensure it reflected realistic ambiguities without completely breaking the signal we wanted the models to learn.

As the data is going to be stored in SQL, for now we only use panda and store it by creating an csv file for easier checking the data.

```
# Add small Gaussian noise to numeric features
numeric_cols = df.select_dtypes(include=['number']).columns.drop('Fraud_Label')
for col in numeric_cols:
    noise = np.random.normal(0, 0.05 * df[col].std(), size=len(df))
    df[col] += noise

# Enforce valid ranges and types
if 'Transaction_Amount' in df.columns:
    df['Transaction_Amount'] = df['Transaction_Amount'].clip(lower=0)

if 'Risk_Score' in df.columns:
    # Convert intervals to midpoint if needed
    df['Risk_Score'] = df['Risk_Score'].apply(
        lambda x: x.mid if isinstance(x, pd.Interval) else x
    )
    # Now clip safely
    df['Risk_Score'] = df['Risk_Score'].clip(lower=0, upper=1.0)

if 'IP_Address_Flag' in df.columns:
    df['IP_Address_Flag'] = np.random.choice([0, 1], size=len(df), p=[0.95, 0.05])

if 'Previous_Fraudulent_Activity' in df.columns:
    df['Previous_Fraudulent_Activity'] = np.random.choice([0, 1], size=len(df), p=[0.9, 0.1])

if 'Daily_Transaction_Count' in df.columns:
    df['Daily_Transaction_Count'] = df['Daily_Transaction_Count'].round().clip(lower=0)

if 'Avg_Transaction_Amount_7d' in df.columns:
    df['Avg_Transaction_Amount_7d'] = df['Avg_Transaction_Amount_7d'].clip(lower=0)
```

```
if 'Failed_Transaction_Count_7d' in df.columns:
    df['Failed_Transaction_Count_7d'] = df['Failed_Transaction_Count_7d'].round().clip(0, 4)

if 'Card_Age' in df.columns:
    df['Card_Age'] = df['Card_Age'].round().clip(lower=0)

if 'Transaction_Distance' in df.columns:
    df['Transaction_Distance'] = df['Transaction_Distance'].clip(lower=0)

if 'Is_Weekend' in df.columns:
    df['Is_Weekend'] = np.random.choice([0, 1], size=len(df), p=[0.7, 0.3])

# Flip 3% of the labels to simulate noise/mislabeling
flip_indices = df.sample(frac=0.03, random_state=42).index
df.loc[flip_indices, 'Fraud_Label'] = 1 - df.loc[flip_indices, 'Fraud_Label']

# Inject outliers into Transaction_Amount
if 'Transaction_Amount' in df.columns:
    outlier_indices = df.sample(frac=0.01, random_state=1).index
    df.loc[outlier_indices, 'Transaction_Amount'] *= np.random.randint(5, 10)

# Randomly drop 1% of values in each numeric column
for col in numeric_cols:
    missing_indices = df.sample(frac=0.01, random_state=0).index
    df.loc[missing_indices, col] = np.nan

# Export to file
df.to_csv("improved_dirty_fraud_dataset.csv", index=False)
print("Dirty version of synthetic dataset created")
```

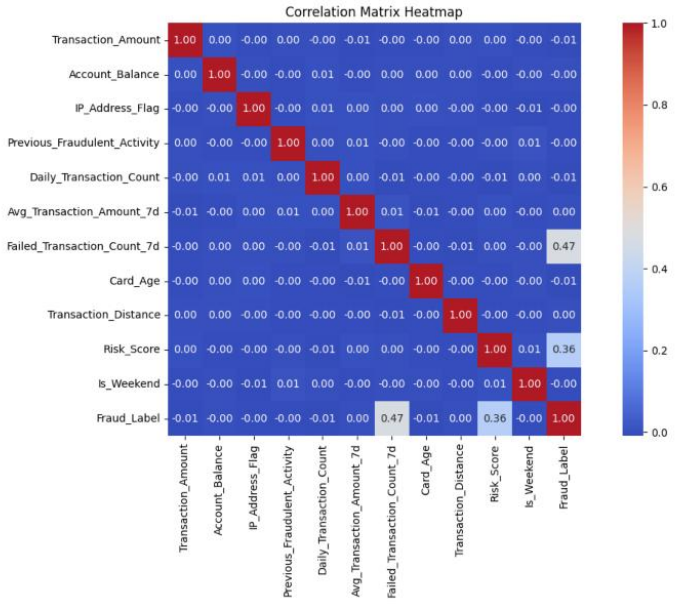
The dataset we used was designed to reflect realistic fraud detection scenarios. It includes time-based features and behavioral patterns, which help simulate how fraud changes over time. This makes it easier to test models that can adapt to new fraud tactics instead of just working on static data. It also includes detailed information like transactions and user behavior, giving a more complete picture. This makes it especially useful for tackling real-world challenges like class imbalance and catching rare but important fraud cases, something banks and financial systems deal with all the time.

4.3 Data Preprocessing

The first step involved data cleaning. We began by removing duplicates, dropping columns that are not needed and verifying the consistency of each column by removing outliers. Data types were standardized, and categorical features were reviewed to ensure proper formatting for model input. While IQR-based outlier removal was initially considered to clean the dataset, it resulted in the loss of rare but potentially important transactions, especially high-value ones. Since fraud often occurs in these edge cases, we chose to retain all data points for clustering and analysis to preserve critical behavioral signals.

4.4 Tools and Techniques

Using Matplotlib and Seaborn, we conducted exploratory data analysis (EDA) to identify patterns. Features like Risk_Score, Transaction_Amount, and Failed_Transaction_Count showed meaningful correlations with fraud.



Correlation Matrix Heatmap: Most features showed weak correlation with the fraud label, except for Failed_Transaction_Count_7d (0.47) and Risk_Score (0.36), supporting their importance in both manual rules and models. The rest — including Transaction_Amount, Card_Age, and Is_Weekend — exhibit minimal correlation, suggesting that while they may not be strong individual predictors, they could still hold value through interactions or in engineered combinations.

Categorical Features: Most categorical attributes showed about 35% fraud rate across the board, indicating they had minimal contribution to fraud detection in this dataset and therefore not heavily weighted in our models.

Figure 1. Correlation Matrix Heatmap

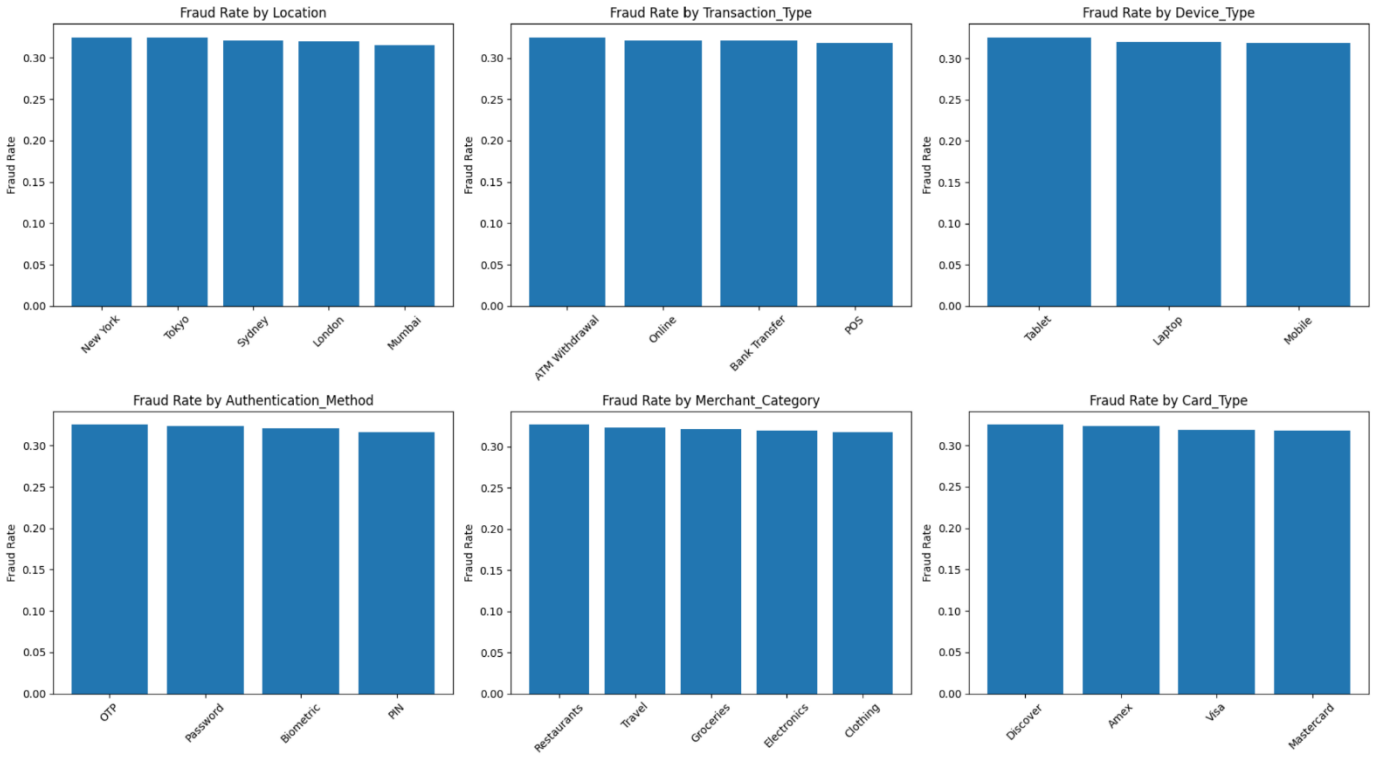


Figure 6. Fraud Rate by Categorical Features

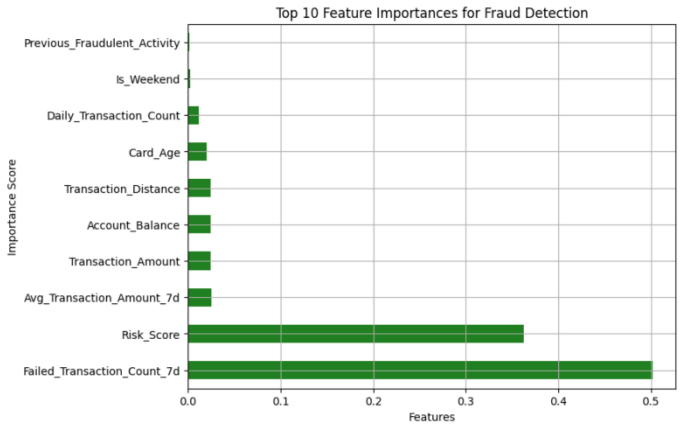


Figure 2. Top 10 Important Features

Feature Importance Plot: The top 10 feature importance chart revealed that Failed_Transaction_Count_7d and Risk_Score accounted for over 90% of decision weight. Other features such as Card_Age and Account_Balance had limited influence.

Risk Score by Fraud Label: The density plot showed fraud cases spiking in the 0.85–1.0 range, validating the High_Risk_Score flag threshold.

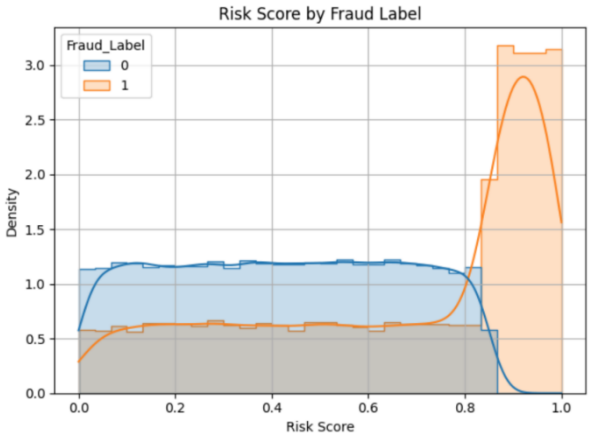


Figure 3. Risk Score by Fraud Label

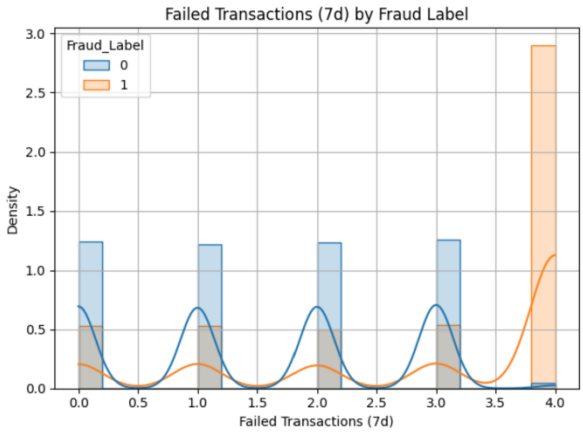


Figure 4. Failed Transactions by Fraud Label

Failed Transactions by Fraud Label: Fraudulent transactions were highly concentrated at 4 failed attempts, while non-fraud cases were more spread out. This provided strong support for creating a new column where a value above 3 triggers a risk score boost. The visual makes clear that exceeding this failure threshold is rare in normal behavior but common in fraud, supporting its inclusion as a manual rule.

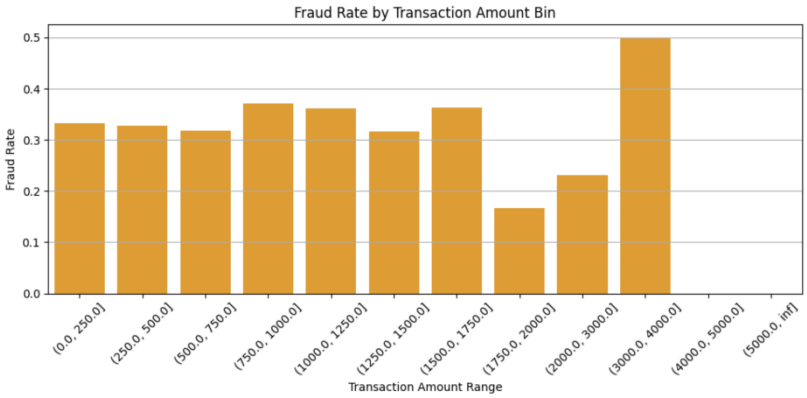
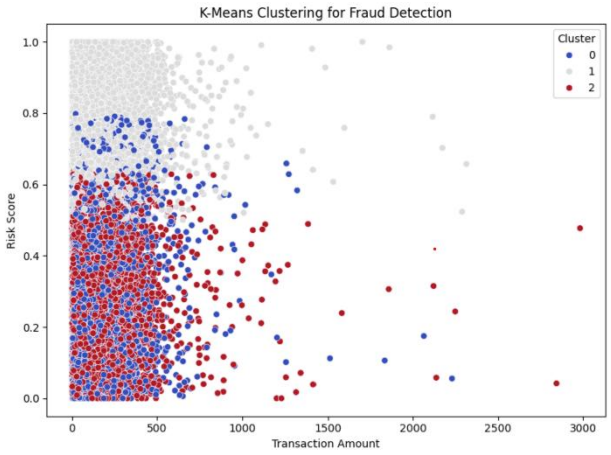


Figure 5. Fraud Rate by Transaction Amount

Fraud Rate by Transaction Amount: Fraud rate is slightly higher at $750-1500 range, with a fraud rate around 35%, after which it drops off significantly and raised again around 50% in range $3000-5000. While Transaction_Amount alone had low importance in our feature ranking, this plot showed that it still plays a meaningful role when combined with other indicators, making it valuable for layered detection logic.

K-Means Clustering Implementation



To uncover hidden patterns in transaction behavior, we applied K-Means clustering using Transaction_Amount, Risk_Score, and Failed_Transaction_Count_7d after standardizing the data. We chose 3 clusters to keep the results interpretable and compared them against actual fraud labels. Cluster 1 had the highest fraud rate (~52.6%), Cluster 2 had a moderate rate (~37.8), and Cluster 0 had the lowest, which explains as it is more scattered (~3%). This showed that K-Means was effective at grouping transactions by fraud risk, even without using the fraud labels during clustering.

Approach 1: All Models Compared

For model development, we tested a diverse range of classifiers to identify which techniques would best suit this fraud detection problem.

- LightGBM
- CatBoost
- Bagging Classifier
- XGBoost
- Extra Trees
- AdaBoost
- Gradient Boosting
- Random Forest
- Decision Tree
- Support Vector Machine (SVM)
- Naive Bayes
- K-Nearest Neighbors (KNN)

The dataset was split into 50% training and 50% testing subsets. After determining through visualization that categorical columns had minimal influence on fraud prediction, we dropped them and applied feature scaling before training.

Each model was trained under consistent conditions and evaluated using the same dataset and feature set. Performance metrics included accuracy, precision, recall, and F1-score. Random Forest, XGBoost, and Bagging Classifier were among the top performers, with accuracies exceeding 96%. We noticed that simpler models like Decision Tree (92.45%), KNN (87.97%), and Naive Bayes (86.49%) showed lower accuracy, likely due to the nuanced and complex feature interactions required in this context.

Model	Accuracy	Precision	Recall	F1 Score	AUC-ROC
Decision Tree	0.922747	0.880053	0.888199	0.884107	0.914049
Random Forest	0.960040	0.953415	0.924735	0.938856	0.960975
Gradient Boosting	0.960121	0.951613	0.926927	0.939108	0.960182
AdaBoost	0.958990	0.941798	0.934113	0.937939	0.960287
Support Vector Machine	0.951152	0.935123	0.916332	0.925632	0.960544
K-Nearest Neighbors	0.878303	0.858009	0.758738	0.805326	0.921056
Naive Bayes	0.862101	0.819271	0.749726	0.782957	0.913793
Extra Trees	0.960202	0.953211	0.925466	0.939134	0.960729
XGBoost	0.958909	0.949513	0.925344	0.937273	0.959819
Bagging Classifier	0.960485	0.952912	0.926684	0.939615	0.960785
CatBoost	0.958222	0.948058	0.924735	0.936252	0.959953
LightGBM	0.957899	0.948119	0.923639	0.935719	0.959703

At first, we were surprised by how well the models performed, many reached almost perfect accuracy on both training and test sets. This raised concerns about potential overfitting, prompting us to conduct multiple validation checks. However, after running the correlation check and seeing that the data are all low to moderate, we concluded that overfitting wasn't the cause.

Instead, we realized that the original synthetic dataset was too “clean” or was already preprocessed. To address this, we spent a significant amount of time dirtying the dataset, injecting randomness, overlapping edge cases, and softening clear feature boundaries as has been mentioned previously in previous steps.

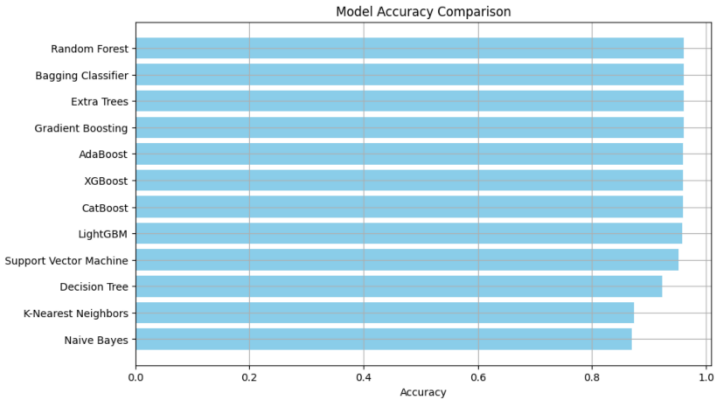


Figure 7. Model Accuracy Comparison

Approach 2: Handcrafted Rule-Based Model

To take this project a step further, we decided not to stop at existing models. Instead, we challenged ourselves to create a fraud detection system entirely by hand, based on what we observed through data visualization and analysis. Without using any machine learning model, we defined our own logic and conditions rooted in the patterns we uncovered.

We constructed custom binary flags like High_Risk_Score", "Mid_High_Amount", "FailedTx_Heavy", "High_Amount_Fraud_Pattern, based on thresholds derived from visual trends. These were then used in a rule-based function that classifies transactions as fraudulent or not, entirely independent of any algorithmic model.

```
# Binary flags based on visual analysis
df["High_Risk_Score"] = (df["Risk_Score"] > 0.85).astype(int)
df["Extreme_Risk_Score"] = (df["Risk_Score"] > 0.95).astype(int)
df["Mid_High_Amount"] =(
    (df["Transaction_Amount"] >= 750) &
    (df["Transaction_Amount"] <= 1500)).astype(int)
df["Risk_Transaction"] = (
    (df["Risk_Score"] > 0.8) &
    (df["Transaction_Amount"] <= 500)
).astype(int)
df["FailedTx_Heavy"] = (df["Failed_Transaction_Count_7d"] > 3).astype(int)
df["High_Amount_Fraud_Pattern"] = ((df["Transaction_Amount"] >= 750) &
                                   (df["Failed_Transaction_Count_7d"] > 3)).astype(int)
```

```
# Rule-based model
def final_rule_from_flags(row):
    if row["Extreme_Risk_Score"] == 1:
        return 1
    score = (
        2 * row["High_Risk_Score"] +
        0.5 * row["Mid_High_Amount"] +
        2 * row["FailedTx_Heavy"] +
        1 * row["High_Amount_Fraud_Pattern"] +
        1 * row["Risk_Transaction"]
    )
    return 1 if score >= 2 else 0

df["Predicted_Final"] = df.apply(final_rule_from_flags, axis=1)
```

```
true_labels = df["Fraud_Label"]
predictions = df["Predicted_Final"]

accuracy = accuracy_score(true_labels, predictions)
precision = precision_score(true_labels, predictions, zero_division=0)
recall = recall_score(true_labels, predictions, zero_division=0)
f1 = f1_score(true_labels, predictions, zero_division=0)
auc = roc_auc_score(true_labels, predictions)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("ROC AUC:", auc)
```

```
Accuracy: 0.9616363636363636
Precision: 0.9576782794617029
Recall: 0.9254937708903069
F1 Score: 0.9413109991655593
ROC AUC: 0.9525638017713753
```

This approach allowed us to test whether manual logic, purely taken by insight, could perform competitively in a fully transparent way. Risk_Score was weighted with a score of 2; Transaction_Amount ranging in $750-1750 was given a score of 0.5; Failed_Transaction_Count > 3 was scored at 2 and later combined with extra factors. We spent a lot of time experimenting with over 1,000 combinations of flags and weights to derive the most predictive rules. The final handcrafted model achieved 96.16% accuracy with great recall and f1 score and precision. Given its simplicity and full interpretability, the model is decent.

Approach 3: Hybrid Model

```
# Train-Test Split & Scaling
# ----------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.5, random_state=42, stratify=y
)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# ----------------------
# Train Hybrid Model
# ----------------------
model = GradientBoostingClassifier(
    n_estimators=100, learning_rate=0.1, max_depth=4, random_state=42
)
model.fit(X_train_scaled, y_train)

# ----------------------
# Evaluate Model
# ----------------------
preds = model.predict(X_test_scaled)
probs = model.predict_proba(X_test_scaled)[:, 1]

accuracy = accuracy_score(y_test, preds)
precision = precision_score(y_test, preds)
recall = recall_score(y_test, preds)
f1 = f1_score(y_test, preds)
auc = roc_auc_score(y_test, probs)

print("Final Hybrid Model Results")
print("Accuracy:", round(accuracy, 4))
print("Precision:", round(precision, 4))
print("Recall:", round(recall, 4))
print("F1 Score:", round(f1, 4))
print("AUC-ROC:", round(auc, 4))
```

```
Final Hybrid Model Results
Accuracy: 0.9609
Precision: 0.9556
Recall: 0.9253
F1 Score: 0.9402
AUC-ROC: 0.9619
```

And then we think. Since the model is already performing well, is it possible to maximize the results even further? To do that, we introduced our handcrafted logic and feed it directly as a feature into one of the existing models we tested earlier, the Gradient Boosting model. This allowed the algorithm to consider both raw features and our customed flags. This hybrid approach improved the model's accuracy to 96.09%, catching even better precision, recall, and f1-score than when we use the model alone by itself. Rather than requiring the model to discover all patterns from scratch, this method provided guided learning, allowed us to help the model "see" what a human might already know, but without overriding the flexibility of algorithmic decision-making. If choosing the wrong logic, the result can be really bad even if you use good models like this. We believe it will perform even better as more time and effort is spent to train and test even more trials and combinations. The more concise and careful with the conditions we add, the higher the accuracy will be. We also tested with XGBoost, but that had to use multiple loops until it reached the amount needed. That failed to align with our goal as it is not really efficient and has not outperformed this model.

4.5 Key Findings and Results

As predicted, the two conditions where risk score is higher and number of failed transactions >3 have a high chance of being fraud. We checked them and tested random 15 transactions that satisfies the conditions and only 1 of the transactions is actually fraud.

```
['Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 0',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1',
 'Input: (1.00, 4.00), Actual Label: 1']
```

5. SQL-Based Analysis of High-Risk Transactions

To deepen the interpretability of our fraud detection logic, we introduced SQL into our pipeline by simulating a relational database environment using sqlite3. We structured our synthetic dataset into three normalized tables:

- A users table with User_ID, Location, and Device_Type,
- A transactions table including key attributes like Transaction_ID, User_ID, Transaction_Amount, Risk_Score, Failed_Transaction_Count_7d, and Is_Weekend,

- And a flags table consisting of our handcrafted binary indicators such as High_Risk_Score, Mid_High_Amount, FailedTx_Heavy, and High_Amount_Fraud_Pattern.

By inserting these tables into memory and performing SQL joins, we were able to run focused queries that mimic how real-world fraud analysts might retrieve records of interest.

One such query filtered for transactions where: Risk_Score exceeds 0.8, High_Risk_Score flag is triggered (1), and FailedTx_Heavy is also flagged (1), indicating more than 3 failed attempts in the last 7 days. This logical conjunction retrieved transactions that match multiple fraud signals and suggest the user with not only transaction details but also associated user locations and fraud flags.

5.1. Structured Validation of Rule-Based Logic

At this stage, we weren't just trying to detect fraud, but we wanted to find out who kept doing it. We specifically filtered for users who triggered our fraud flags more than once, simulating what it would be like to catch repeat offenders in a real system, as the IP_Address_Flag feature, one of the most hopeful features out of all, does not seem to help much with catching actual frauds.

This query listed 323 high-risk transactions made by users who repeatedly satisfied the conditions we set up. Among them, we found that at least 310 transactions were confirmed fraud, linked to 155 unique users. That means it's roughly 96% precision. This result gave us confidence that our rule-based system was consistently flagging users who kept coming back for more.

	User_ID	Transaction_ID	Risk_Score	Failed_Transaction_Count_7d	High_Risk_Score	FailedTx_Heavy	Fraud_Label
0	USER_1003	TXN_18367	0.888925	4.0	1	1	1
1	USER_1003	TXN_39240	0.838262	4.0	1	1	1
2	USER_1026	TXN_24358	0.810272	4.0	1	1	1
3	USER_1026	TXN_5299	0.880760	4.0	1	1	1
4	USER_1058	TXN_13496	0.905622	4.0	1	1	1
...	...	...	...	...	...	...	...
317	USER_9885	TXN_40121	0.924701	4.0	1	1	1
318	USER_9906	TXN_24383	0.815809	4.0	1	1	1
319	USER_9906	TXN_1691	0.879930	4.0	1	1	1
320	USER_9946	TXN_38033	0.874713	4.0	1	1	1
321	USER_9946	TXN_37570	0.997715	4.0	1	1	1

322 rows × 7 columns

5.2. Insight into User Behavior Patterns

SQL's relational design allowed us to group and join tables in ways that revealed deeper behavioral trends. Specifically, we identified:

- Repeat offenders: who triggered high-risk flags multiple times
- Transaction-level recurrence: how often those users matched our fraud rules

- Fraud-label alignment: the degree to which flagged behavior matched actual fraud

We also found that 8,697 users had made more than one transaction in general, but only a fraction (1,800 users) ever triggered our fraud logic. This contrast revealed that while repeated user activity is normal, only a small, specific group repeatedly exhibits fraud-like behavior, providing a clearer picture of how fraud evolves over time across user accounts.

```
query = """
WITH flagged_txns AS (
    SELECT
        t.User_ID,
        t.Transaction_ID,
        t.Risk_Score,
        t.Failed_Transaction_Count_7d,
        f.High_Risk_Score,
        f.FailedTx_Heavy,
        t.Fraud_Label
    FROM transactions t
    JOIN flags f ON t.Transaction_ID = f.Transaction_ID
    WHERE t.Risk_Score > 0.8
      AND f.High_Risk_Score = 1
      AND f.FailedTx_Heavy = 1
)
SELECT *
FROM flagged_txns
WHERE User_ID IN (
    SELECT User_ID
    FROM flagged_txns
    GROUP BY User_ID
    HAVING COUNT(*) > 1
)
ORDER BY User_ID;
"""

repeat_users_with_fraud_labels = pd.read_sql_query(query, conn)
repeat_users_with_fraud_labels
```

5.2. Contextual Filtering and Robustness Checks

This layered filtering allowed us to test our rule not just in narrow extremes, but also under broader behavioral contexts. It confirmed that our fraud detection rules perform reliably in detecting persistent fraud users even outside of traditional model training environments.

In combination, these SQL-driven insights not only validated the core logic behind our rule-based detection system but also provided a practical foundation for real-world deployment, forensic inspection, and integration with human fraud analysis workflows.

6. Addressing Class Imbalance

One notable challenge we encountered was the imbalance between fraudulent and non-fraudulent transactions in our dataset. Although our dataset was synthetic and customized, we intentionally made fraudulent cases much less frequent like in real-world banking data, so the model would learn to catch rare but high-impact events. This class imbalance made it easy for some models to become biased toward predicting the majority class (non-fraud), leading to deceptively high accuracy but poor fraud detection.

Rather than using oversampling techniques like SMOTE, which may not work well on synthetic data, we focused on boosting-based models that handle imbalance more gracefully. Models such as Gradient Boosting and XGBoost showed strong performance, with high recall and precision even when trained on the original, imbalanced dataset.

We also used feature engineering to give the models stronger fraud-related signals (e.g., binary flags based on risky behavior), which helped it stay sensitive to unusual cases. When we tested our handcrafted rule-based model, we noticed it was naturally focused on fraud detection, despite the imbalance, because it was designed using patterns directly observed in the data. By combining these strategies, engineered fraud indicators, and hybrid modeling, we were able to create a system that could detect fraud reliably without being so overwhelmed by the dominant non-fraud class.

7. Limitations and Future Work

7.1. Limitations

While our models performed well on the synthetic dataset, there are important limitations to consider. The dataset itself, although designed to mimic real-world patterns, remains randomized. This means that although we introduced specific correlations between features and fraud behavior, the underlying relationships may not fully represent real-life financial fraud complexity. As a result, our models, the handcrafted logic, may perform differently when tested on actual transaction data.

Another key limitation is that some of the variables used may hold artificial influence due to how the synthetic data was constructed. While the inclusion of fraud-like behaviors such as high Risk_Score and multiple failed transactions helped train effective models, these indicators might not generalize across diverse datasets where fraud manifests differently.

7.2. Future Work

For future work, it would be valuable to test these models on real-world financial datasets with secure access or anonymized logs. Incorporating streaming data for real-time fraud detection, as well as techniques such as explainable AI (e.g., SHAP or LIME), could enhance both the robustness and interpretability of the system. Exploring federated learning across institutions may also allow broader fraud prevention efforts while protecting user privacy.

As a next step, we plan to turn this system into a working website that can predict fraud based on uploaded or real-time transaction inputs. Ideally, this would be backed by a database that stores ongoing activity and allows the model to keep learning and improving over time—making the tool both accessible and continuously smarter.

Acknowledgement

We sincerely appreciate Professor Naina Chaturvedi for her engaging lectures and invaluable guidance in CS 439: Introduction to Data Science. Her dedication to teaching and willingness to assist us whenever needed have greatly enhanced my understanding of the subject.

We would also like to thank our Teaching Assistants for their support and assistance throughout the course. Their guidance has been instrumental in clarifying concepts and reinforcing our learning.

References

Dataset Used: Fraud Detection Transactions Dataset, https://www.kaggle.com/datasets/samayashar/fraud-detection-transactions-dataset

Fang, Gavin, “Fraud Detection ML Model Comparison.” *Kaggle.com*, Kaggle, 24 Feb. 2025, www.kaggle.com/code/kangxunfang/fraud-detection-ml-model-comparison

Md Shakil Islam, and Nayem Rahman. “AI-Driven Fraud Detections in Financial Institutions: A Comprehensive Study.” Journal of Computer Science and Technology Studies, vol. 7, no. 1, 28 Jan. 2025, https://www.researchgate.net/publication/388462459_AI-Driven_Fraud_Detections_in_Financial _Institutions_A_Comprehensive_Study