# Anomaly Detection Using Autoencoders


## Abstract

This document describes the implementation of an anomaly detection system using autoencoders to identify fraudulent merchant behaviors. It includes synthetic data generation, feature engineering, model development, and anomaly scoring mechanisms.


## 1 Data Generation

### 1.1 Merchant Profile Generation

Merchant profiles are synthesized with attributes such as ID, name, business type, registration date, GST status, and transaction volumes. The code generates data for normal trading patterns (80% of merchants) and specific fraud patterns (20% of merchants):

- Late night transactions
- High velocity spikes
- Customer concentration

### 1.2 Code Snippet

Listing 1: Data Generation

```
import random
import datetime
from faker import Faker
import json

fake = Faker()
```

```python
def generate_merchant_profile(merchant_id):
    return {
        "merchant_id": merchant_id,
        "business_name": fake.company(),
        "business_type": random.choice(["Electronics", "Fashion", "Gro
        "registration_date": fake.date_between(start_date="-5y", end_d
        "gst_status": random.choice([True, False]),
        "average_ticket_size": round(random.uniform(500, 10000), 2)
    }
```

# 2 Feature Engineering

## 2.1 Features Calculated

Features extracted from the transaction dataset include:

- Transaction velocity metrics
- Time-based patterns
- Amount distributions
- Customer concentration ratios

## 2.2 Normalization Pipeline

A pipeline ensures all features are normalized to a standard range to support efficient training of the autoencoder.

# 3 Model Development

## 3.1 Autoencoder Architecture

An autoencoder model is implemented to reconstruct normal merchant behaviors. Key steps include:

- Training on normal merchant data
- Calculating reconstruction error thresholds for anomaly detection
- Implementing anomaly scoring

# 4 Fraud Pattern Detection

## 4.1 Specific Detection Rules

Fraudulent patterns are detected based on the following rules:

- High velocity detection: Identifying transactions exceeding calculated thresholds.
- Odd-hour pattern detection: Flagging transactions occurring between 11 PM and 4 AM.
- Customer concentration analysis: Detecting merchants with unusually low customer-to-transaction ratios.

## 4.2 Code Snippet

Listing 2: Fraud Pattern Detection

```
from sklearn.metrics import confusion_matrix, precision_score, recall_s

# Step 1: Extract the `Is_Anomalous` and `label` columns from the `tra
y_true = transactions['label']
y_pred = transactions['Is_Anomalous'].astype(int)

# Step 2: Calculate the confusion matrix
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

# Step 3: Calculate the scores
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred)
recall = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)

# Display the results
print("Confusion Matrix (TN, FP, FN, TP):", (tn, fp, fn, tp))
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

# 5 Anomaly Scoring Mechanism

## 5.1 Weighted Scoring

Fraud scores are calculated using weighted flags:

- High Velocity Flag: 0.5
- Odd Hour Flag: 0.3
- High Concentration Flag: 0.2

Anomalies are flagged based on a threshold calculated from fraud scores.

## 5.2 Code Snippet

Listing 3: Fraud Score Calculation

```
# Assign weights to flags
weights = {
    'High_Velocity_Flag': 0.5,
    'Odd_Hour_Flag': 0.3,
    'High_Concentration_Flag': 0.2
}

# Calculate fraud score per transaction
df['Fraud_Score'] = (
    weights['High_Velocity_Flag'] * df['High_Velocity_Flag'].astype(in
    weights['Odd_Hour_Flag'] * df['Odd_Hour_Flag'].astype(int) +
    weights['High_Concentration_Flag'] * df['High_Concentration_Flag']
)

# Filter for each flag and calculate mean and std fraud score
mean_fraud_score_high_velocity = df[df['High_Velocity_Flag'] == 1]['Fr
std_fraud_score_high_velocity = df[df['High_Velocity_Flag'] == 1]['Fra
print("High Velocity Detection - Mean:", mean_fraud_score_high_velocit
```

# 6 Evaluation and Results

## 6.1 Performance Metrics

The performance of the anomaly detection system is evaluated using standard metrics. These include accuracy, precision, recall, and F1-score, calculated as follows:

- **Accuracy:** Measures the overall correctness of predictions.
- **Precision:** Indicates the percentage of correctly identified anomalies.
- **Recall:** Reflects the ability to identify all actual anomalies.
- **F1-Score:** Balances precision and recall for overall effectiveness.

### 6.2 Example Metrics Output

An example output from the evaluation is:

```
Confusion Matrix (TN, FP, FN, TP): (850, 25, 30, 95)
Accuracy: 0.9405
Precision: 0.7917
Recall: 0.7600
F1 Score: 0.7756
```

### 6.3 Insights

From the metrics, it is observed:

- High accuracy ensures that most transactions are correctly classified.
- Precision highlights the system's robustness in minimizing false positives.
- Balanced recall ensures comprehensive anomaly detection.

These insights indicate the system effectively detects fraud patterns while maintaining reliability.

## Conclusion

This document outlines a comprehensive pipeline for detecting anomalies in merchant transactions. By leveraging synthetic data, feature engineering, and an autoencoder model, the system efficiently flags suspicious patterns and aids in fraud prevention.