High-level Overview

This documentation outlines the synthetic financial transaction data pipeline which contains 1 row per event (e.g. buying groceries, paying rent, etc). Data is stored in CSV files so it's easy to feed into scripts or ML models.

Each row has identifiers (IDs) and context fields that are derived from the ISO 20022 which is the bank messaging standard, but simplified for the scope of this project.

Core Fields (IDs) in our CSV

- transaction_id: unique transaction id, e.g. tx000123
- timestamp: UTC timestamp in ISO 8601, with trailing Z, e.g. 2025-12-18T08:31:10Z
- account_id: synthetic account id, e.g. ACCT100123
- payer_id: front-end payer identifier, usually tied to the account (e.g. card id)
- payee_id: synthetic merchant id, like MCHT_STARBUCKS
- amount: transaction amount as string with 2 decimals (e.g. 23.45)
- currency: ISO currency code (we use USD)
- merchant_category: high-level category (groceries, coffee, fuel, rent, p2p, etc.)
- merchant_name: human-readable merchant name (Starbucks, Costco Wholesale, etc.)
- country: 2-letter country code (US, CA, NG, IR, RU, …)
- channel: how the transaction was made (in_store, online, mobile_app, atm)
- device_id: synthetic device id tied to the account
- ip_hash: synthetic IP fingerprint token
- balance_before: balance before the transaction (string, 2 decimals)
- balance_after: balance after the transaction (string, 2 decimals)
- label: ground truth for evaluation
    - 0 = clean
    - 1 = suspicious
    - 2 = fraud
- notes: short human-readable explanation of what kind of behavior this row represents (e.g., grocery run, high amount potential fraud, multiple small purchases)

These are all the fields that the transaction generator knows. The fraud detector engine labels transactions with its own pred_label, and the debug script compares the two for testing and accuracy.

Scripts in The Pipeline

- **generate_transasctions_robust.py**
  - A synthetic financial data generator that contains tunable parameters such as: merchants, seed (deterministic entropy generator), etc
- **detect_fraud_robust.py**
  - A rule-based engine that reads the generated CSV and applies a prediction label (clean, suspicious, or fraudulent).
- **debug_fraud_data.py**
  - An evaluation script that compares the generator's label to the detector's pred_label and prints the distributions, confusion matrix, accuracy, precision, and recall.

Generating Synthetic Data

Script: generate_transactions_robust.py

This script creates realistic financial transactions with a mix of clean, suspicious, and fraudulent data. At the top of the file, you can:

1. edit default parameters with an IDE such as VS Code or any text editor such as nano on Linux or WSL.

```python
# here are a few tunable knobs at the top so you don't have to dig throughout the script
DEFAULT_N_TRANSACTIONS = 10_000

# label mix which get normalized but it's nice to control here
DEFAULT_LABEL_WEIGHTS = {
    "clean": 0.7,
    "suspicious": 0.2,
    "fraud": 0.1,
}

# if seed=None: new randomness each run (nice for entropy)
# change to an integer number for repeatable results (e.g. 42)
DEFAULT_SEED = None

# how many days to spread timestamps over
DEFAULT_DAYS_SPAN = 30

# starting money range in accounts (randomized per account)
ACCOUNT_START_BALANCE = (500, 5000)
```

2. Override them using flags when you run the script (in a command line)

```
python3 generate_transactions_robust.py \
  --n 10000 \
  -o tx_n10000_p70-20-10_seedRND.csv
/mnt/c/Users/bryso/SynologyDrive/Documents/ASU & CGCC/S7 Fall 2025 (15)/CSE485 (3)/Financial Fraud Data/generate_transactio
ns_robust.py:292: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future versio
n. Use timezone-aware objects to represent datetimes in UTC: datetime.datetime.now(datetime.UTC).
  base_time = datetime.utcnow()
generated 10000 transactions → tx_n10000_p70-20-10_seedRND.csv
label mix: clean 0.70, suspicious 0.20, fraud 0.10
```

The above highlighted command will generate 10,000 transactions (n 10000) with a weighted distribution of 70% clean, 20% suspicious, and 10% fraudulent.

3. File naming convention:
   tx_n**<num-of-rows>**_p**<clean-weight>**-**<suspicious>**-**<fraud>**_**<seed>**
   a. e.g. tx_n10000_p70-20-10_seedRNG
   b. ‘seedRNG’ means no seed wa s specified

Running The Fraud Detection Rules

Script: detect_fraud_robust.py

His script reads the generated CSV file, applies a set of rules, then writes a scored CSV with predictions. As above you can modify tunable parameters using an IDE (easiest) or text editor.

Run it using this command: python3 detect_fraud_robust.py tx_n10000_p70-20-10_seedRNG.csv --out tx_n1000_p70-20-10_seedRNG_scored.csv

It’s important to include ‘--out’ and not just write to the new CSV file because that ensures that the data retains the label and pred_label.

Evaluating Detector Accuracy & Performance

Script: debug_fraud_data.py

This script takes the scored CSV file from above and compares:

- Ground truth: the label from the generator
- Prediction: pred_label from the detector

Then it prints data such as:

- The distribution of true labels (did the generator really output the expected weights?)
- Distribution of the predicted labels (is the detector too sensitive or not sensitive enough?)
- A confusion matrix
- Accuracy, precision, recall

Once the detector script is ran and the scored csv file has been created, run the debug script using:
python3 debug_fraud_data.py tx_n10000_p70-20-10_seedRNG_scored.csv

End-to-end Workflow

1. Generate a dataset:
    a. python3 generate_transactions_robust.py -o tx_n10000_p70-20-10_seedRNG.csv
2. Run the fraud detector on that dataset
    a. Python3 detect_fraud_robust.py tx_n10000_p70-20-10_seedRNG.csv -o tx_n1000_p70-20-10_seedRNG_scored.csv
3. Evaluate detector accuracy
    a. Python3 debug_fraud_data.py tx_n1000_p70-20-10_seedRNG_scored.csv