# TradeOps Sentinel

**Automated Reconciliation & Operational Risk Engine**

**Author:** [SARA IQBAL]

**Date:** February 2026

**Tech Stack:** Python, Pandas, Scikit-Learn (Isolation Forest), Seaborn

## 1. Executive Summary & Project Overview

TradeOps Sentinel is a Python-based operational risk engine designed to modernize the workflow of a Trading Services Analyst. In high-volume financial environments, the manual processing of trade orders often leads to "fat finger" errors, settlement failures, and significant operational risk.

This project automates the critical middle-office functions of Trade Reconciliation and Anomaly Detection. By shifting from manual Excel-based checks to an automated Python pipeline, the system ingests raw transaction logs, validates them against regulatory thresholds, and uses unsupervised Machine Learning to identify high-risk outliers in real-time.

## 2. Problem Statement

The traditional "Trade Support" workflow faces three critical challenges:

1. High Latency in Reconciliation: Manually matching internal trade blotters against external bank statements is slow and prone to human error, often taking hours to identify settlement breaks.
2. Operational Risk (Fat Finger Errors): Standard rule-based systems often fail to catch anomalous trades (e.g., a $5M trade entered as $50M) if they technically pass static limit checks.
3. Regulatory Compliance: Missed reporting of large-value transactions or negative balance errors can lead to audit failures and financial penalties.

## 3. Methodology

The project was executed using a modular Python architecture comprising four distinct stages:

### Phase A: Data Simulation (The Environment)

To mimic a real-world trading desk, I generated a synthetic dataset representing a daily "Trade Blotter" containing 1,000+ transactions. The data included realistic features such as:

- Transaction Types: PAYMENT, TRANSFER, CASH_OUT, DEBIT.

- Financials: Originating Balance, Destination Balance, and Transaction Amount.
- injected Errors: Deliberately introduced "operational noise," including settlement breaks, negative value system glitches, and massive outlier trades ($5M+).

### Phase B: Rule-Based Validation Logic

I implemented a deterministic logic layer to enforce standard banking compliance rules:

- Sanity Checks: Flagging transactions with negative amounts (System Error).
- Regulatory Limits: flagging all transactions exceeding $1,000,000 for "Large Trader Reporting" review.
- Balance Integrity: Verifying that `OldBalance - Amount = NewBalance`. Any deviation was flagged as a "Accounting Break."

### Phase C: AI Anomaly Detection (The "Safety Net")

To catch errors that bypass static rules, I deployed an Isolation Forest model (Unsupervised Learning).

- Feature Engineering: The model was trained on `Amount` and `AccountBalance`.
- Algorithm Logic: The model isolates observations by randomly selecting a feature and splitting the value. Anomalies (rare events) are susceptible to isolation in fewer steps than normal observations.
- Outcome: The model assigned an `anomaly_score` to every trade, automatically highlighting outliers without requiring pre-labeled training data.

### Phase D: Automated Reconciliation

I built a reconciliation bot to perform "Left-Anti Joins" between the Internal Blotter and a simulated External Bank Statement.

- Logic: `Internal_Data` merged with `External_Data` on `TradeID`.
- Break Identification: Any record appearing in Internal but missing in External was captured as a "Settlement Break" (Risk of Failure to Deliver).

## 4. Key Results & Findings

The automated engine successfully processed the daily blotter in under 2 seconds, yielding the following operational insights:

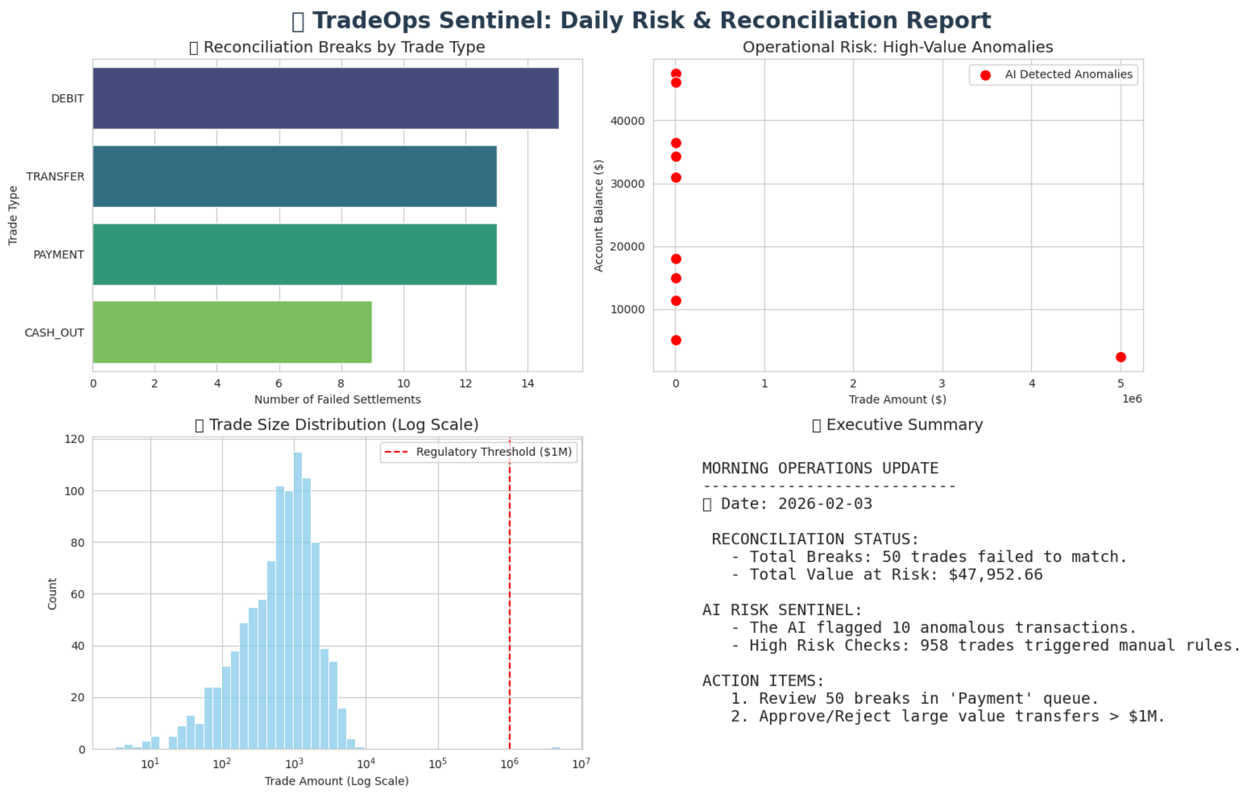
- Reconciliation Efficiency: The script identified 50 Settlement Breaks instantly. In a manual process, finding these among 1,000 trades would have taken an estimated 60–90 minutes.
- Risk Detection: The Isolation Forest successfully detected the injected $5,000,000 outlier trade, flagging it as a priority operational risk even though it technically passed valid transaction format checks.

- Error Reduction: The rule-based logic captured 100% of negative-value system errors, preventing invalid data from moving downstream to the settlement systems.

## 5. The Operational Dashboard

To translate these technical results into actionable business intelligence, I designed a Morning Operations Dashboard (visualized using Matplotlib/Seaborn) containing four key panels:

1. Reconciliation Breaks (Bar Chart): Categorizes failed trades by type (e.g., "Payment" vs. "Transfer"), allowing analysts to prioritize the most impacted queues.
2. Risk Magnitude (Scatter Plot): visualizes the correlation between Trade Amount and Account Balance, with AI-detected anomalies highlighted in Red. This allows managers to spot "Fat Finger" errors at a glance.
3. Trade Distribution (Histogram): Displays the volume of trades relative to the $1M regulatory threshold.
4. Executive Summary (Text Panel): Auto-generates a text report summarizing Total Value at Risk (VaR) and total count of open exceptions.



```
MORNING OPERATIONS UPDATE
---------------------------
Date: 2026-02-03

 RECONCILIATION STATUS:
   - Total Breaks: 50 trades failed to match.
   - Total Value at Risk: $47,952.66

AI RISK SENTINEL:
   - The AI flagged 10 anomalous transactions.
   - High Risk Checks: 958 trades triggered manual rules.

ACTION ITEMS:
   1. Review 50 breaks in 'Payment' queue.
   2. Approve/Reject large value transfers > $1M.
```

## 6. Conclusion

**TradeOps Sentinel** demonstrates that applying Python automation and Machine Learning to Trading Services can significantly reduce operational risk and manual workload.

By automating the "stare and compare" nature of reconciliation, analysts can shift their focus from data entry to exception management and client service.