Capstone Project: Multi-Stage AI Workflow

Name: Courage Dei
Module: Applied AI & Prompt Engineering
Specialization: Data Engineering

Automated Fraud Detection Pipeline

1. Problem Statement

In high-volume e-commerce environments, detecting fraudulent transactions is a time-critical challenge. Traditional methods often rely on retroactive analysis, where data analysts manually run SQL queries, export data to spreadsheets, and visually inspect rows to identify outliers.

The Challenge:

This manual process is:

Slow: There is a significant delay between the fraudulent transaction and its detection.

Repetitive: Analysts run the same queries repeatedly.

Scalability Limited: Manual review cannot keep pace with high transaction volumes.

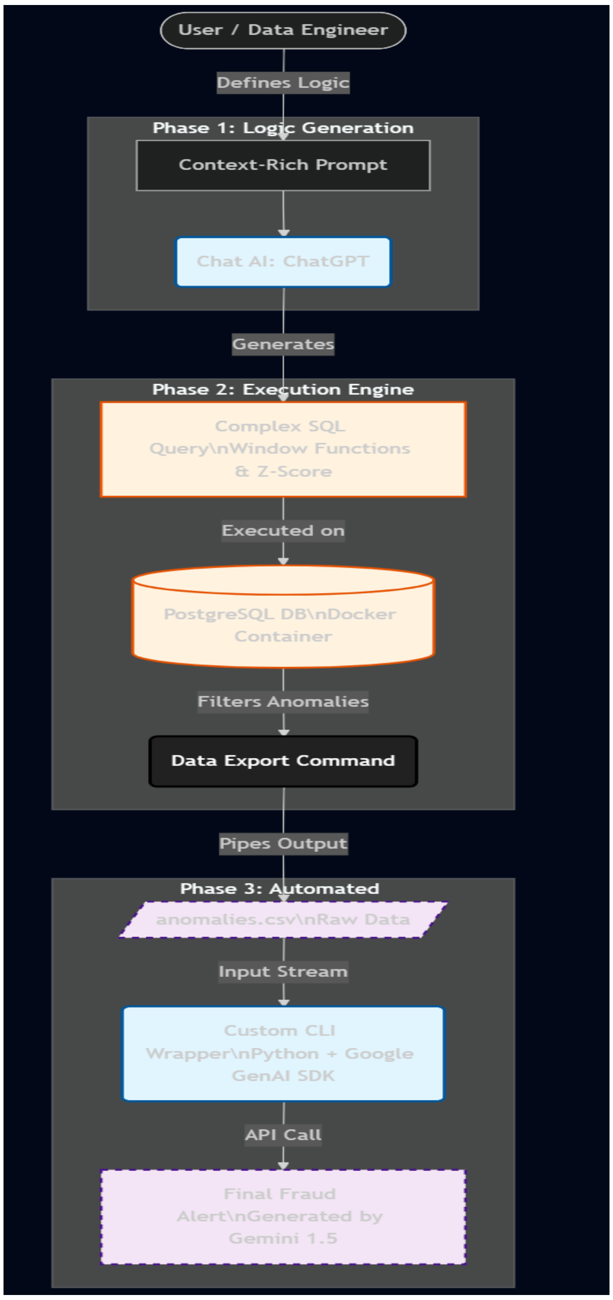
The Solution:

This project automates the entire detection lifecycle. I have designed a Multi-Stage AI Workflow that chains three distinct tools:

1. A Chat AI (ChatGPT) to generate complex statistical logic.
2. A Database Engine (PostgreSQL) to execute that logic and filter data.
3. A CLI AI (Google Gemini) to analyze the results and generate a natural language fraud alert.

2. Workflow Diagram

The following flowchart illustrates the automated data pipeline, moving from the user's initial prompt to the final critical alert.


User / Data Engineer
Defines Logic
Phase 1: Logic Generation
Context-Rich Prompt
Chat AI: ChatGPT
Generates
Phase 2: Execution Engine
Complex SQL Query\nWindow Functions & Z-Score
Executed on
PostgreSQL DB\nDocker Container
Filters Anomalies
Data Export Command
Pipes Output
Phase 3: Automated
anomalies.csv\nRaw Data
Input Stream
Custom CLI Wrapper\nPython + Google GenAI SDK
API Call
Final Fraud Alert\nGenerated by Gemini 1.5

3. Step-by-Step Implementation

Phase 1: Logic Generation (The "Architect")

The workflow begins by defining the statistical threshold for fraud. Instead of using basic filtering (e.g., "Amount > $500"), I utilized ChatGPT to generate a dynamic statistical model.

Goal: Calculate the Z-Score (Standard Deviation) for every user's spending history to find relative anomalies.

Action: The Chat AI generated a complex SQL query utilizing Window Functions: `AVG() OVER (PARTITION BY user_id)` and `STDDEV_SAMP()`.

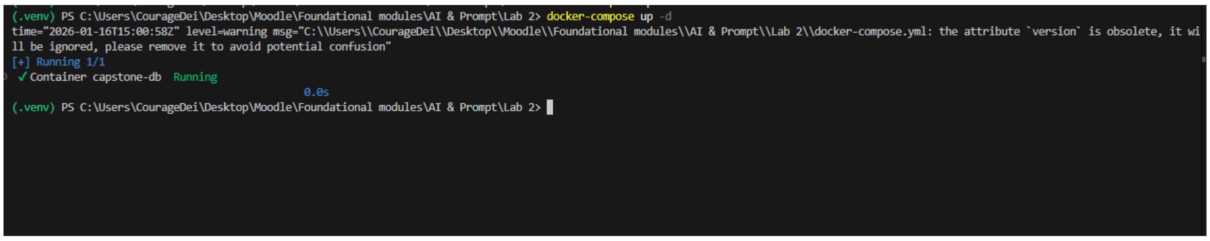
Phase 2: Execution & Extraction (The "Engine")

The logic is executed against a PostgreSQL database running in a Docker container. I used `docker-compose` to orchestrate the database and seed it with realistic transaction data.

Step 2.1: Environment Initialization

The database container is spun up using Docker, ensuring a consistent environment.

SCREENSHOT: 1 - Docker initialization.pn -: Docker container initializing successfully

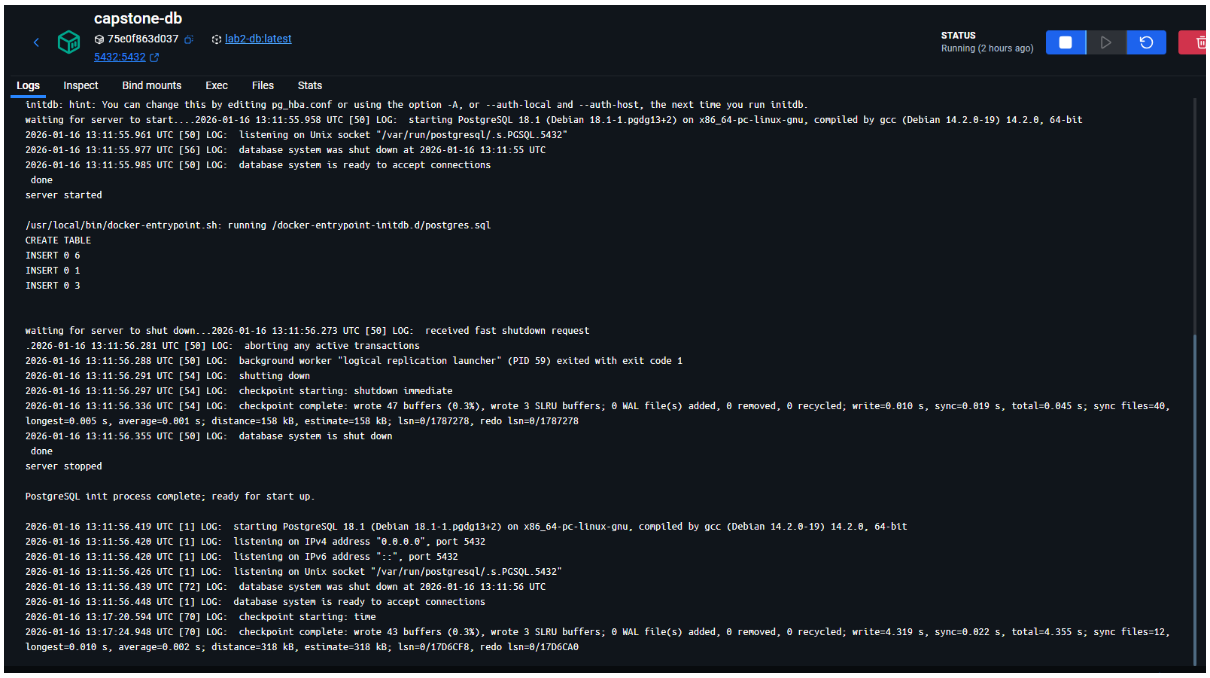


```
(.venv) PS C:\Users\CourageDei\Desktop\Moodle\Foundational modules\AI & Prompt\Lab 2> docker-compose up -d
time="2026-01-16T15:00:58Z" level=warning msg="C:\\Users\\CourageDei\\Desktop\\Moodle\\Foundational modules\\AI & Prompt\\Lab 2\\docker-compose.yml: the attribute `version` is obsolete, it wi
ll be ignored, please remove it to avoid potential confusion"
[+] Running 1/1
 ✓ Container capstone-db  Running
                                                  0.0s
(.venv) PS C:\Users\CourageDei\Desktop\Moodle\Foundational modules\AI & Prompt\Lab 2>
```

Step 2.2: Database Verification

Verifying that the PostgreSQL instance is active and ready for connections.

SCREENSHOT: 2 - Database running in Docker.png - Verification of the active database container

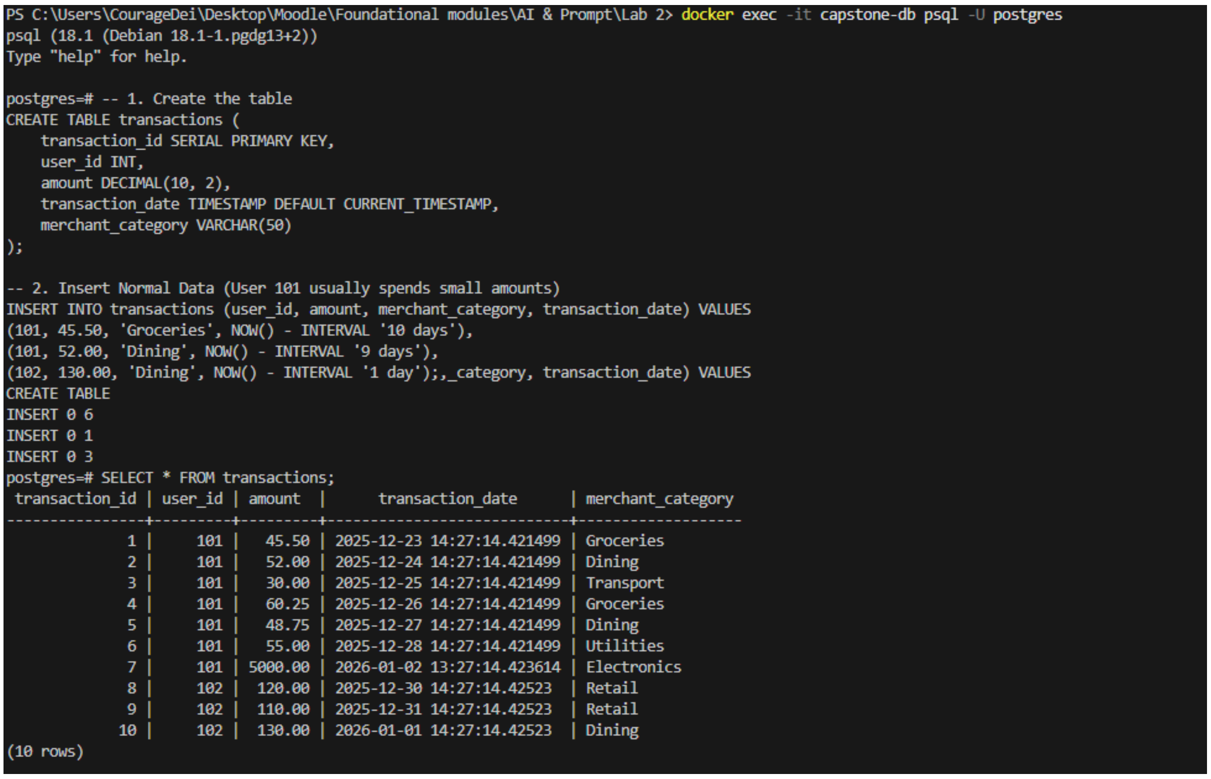


```
initdb: hint: You can change this by editing pg_hba.conf or using the option -A, or --auth-local and --auth-host, the next time you run initdb.
waiting for server to start....2026-01-16 13:11:55.958 UTC [50] LOG:  starting PostgreSQL 18.1 (Debian 18.1-1.pgdg13+2) on x86_64-pc-linux-gnu, compiled by gcc (Debian 14.2.0-19) 14.2.0, 64-bit
2026-01-16 13:11:55.961 UTC [50] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2026-01-16 13:11:55.977 UTC [56] LOG:  database system was shut down at 2026-01-16 13:11:55 UTC
2026-01-16 13:11:55.985 UTC [50] LOG:  database system is ready to accept connections
 done
server started

/usr/local/bin/docker-entrypoint.sh: running /docker-entrypoint-initdb.d/postgres.sql
CREATE TABLE
INSERT 0 6
INSERT 0 1
INSERT 0 3


waiting for server to shut down...2026-01-16 13:11:56.273 UTC [50] LOG:  received fast shutdown request
.2026-01-16 13:11:56.281 UTC [50] LOG:  aborting any active transactions
2026-01-16 13:11:56.288 UTC [50] LOG:  background worker "logical replication launcher" (PID 59) exited with exit code 1
2026-01-16 13:11:56.291 UTC [54] LOG:  shutting down
2026-01-16 13:11:56.297 UTC [54] LOG:  checkpoint starting: shutdown immediate
2026-01-16 13:11:56.336 UTC [54] LOG:  checkpoint complete: wrote 47 buffers (0.3%), wrote 3 SLRU buffers; 0 WAL file(s) added, 0 removed, 0 recycled; write=0.010 s, sync=0.019 s, total=0.045 s; sync files=40,
longest=0.005 s, average=0.001 s; distance=158 kB, estimate=158 kB; lsn=0/1787278, redo lsn=0/1787278
2026-01-16 13:11:56.355 UTC [50] LOG:  database system is shut down
 done
server stopped

PostgreSQL init process complete; ready for start up.

2026-01-16 13:11:56.419 UTC [1] LOG:  starting PostgreSQL 18.1 (Debian 18.1-1.pgdg13+2) on x86_64-pc-linux-gnu, compiled by gcc (Debian 14.2.0-19) 14.2.0, 64-bit
2026-01-16 13:11:56.420 UTC [1] LOG:  listening on IPv4 address "0.0.0.0", port 5432
2026-01-16 13:11:56.420 UTC [1] LOG:  listening on IPv6 address "::", port 5432
2026-01-16 13:11:56.426 UTC [1] LOG:  listening on Unix socket "/var/run/postgresql/.s.PGSQL.5432"
2026-01-16 13:11:56.439 UTC [72] LOG:  database system was shut down at 2026-01-16 13:11:56 UTC
2026-01-16 13:11:56.448 UTC [1] LOG:  database system is ready to accept connections
2026-01-16 13:17:20.594 UTC [70] LOG:  checkpoint starting: time
2026-01-16 13:17:24.948 UTC [70] LOG:  checkpoint complete: wrote 43 buffers (0.3%), wrote 3 SLRU buffers; 0 WAL file(s) added, 0 removed, 0 recycled; write=4.319 s, sync=0.022 s, total=4.355 s; sync files=12,
longest=0.010 s, average=0.002 s; distance=318 kB, estimate=318 kB; lsn=0/17D6CF8, redo lsn=0/17D6CA0
```

Step 2.3: Schema & Data Setup

The `transactions` table is created and seeded with dummy data, including the specific "fraud" scenario (User 101).

SCREENSHOT: 3 - Database table create and insert.png - SQL execution creating the schema and seeding transaction data



```
PS C:\Users\CourageDei\Desktop\Moodle\Foundational modules\AI & Prompt\Lab 2> docker exec -it capstone-db psql -U postgres
psql (18.1 (Debian 18.1-1.pgdg13+2))
Type "help" for help.

postgres=# -- 1. Create the table
CREATE TABLE transactions (
    transaction_id SERIAL PRIMARY KEY,
    user_id INT,
    amount DECIMAL(10, 2),
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    merchant_category VARCHAR(50)
);

-- 2. Insert Normal Data (User 101 usually spends small amounts)
INSERT INTO transactions (user_id, amount, merchant_category, transaction_date) VALUES
(101, 45.50, 'Groceries', NOW() - INTERVAL '10 days'),
(101, 52.00, 'Dining', NOW() - INTERVAL '9 days'),
(102, 130.00, 'Dining', NOW() - INTERVAL '1 day');,_category, transaction_date) VALUES
CREATE TABLE
INSERT 0 6
INSERT 0 1
INSERT 0 3
postgres=# SELECT * FROM transactions;
 transaction_id | user_id | amount  |      transaction_date      | merchant_category
----------------+---------+---------+----------------------------+-------------------
              1 |     101 |   45.50 | 2025-12-23 14:27:14.421499 | Groceries
              2 |     101 |   52.00 | 2025-12-24 14:27:14.421499 | Dining
              3 |     101 |   30.00 | 2025-12-25 14:27:14.421499 | Transport
              4 |     101 |   60.25 | 2025-12-26 14:27:14.421499 | Groceries
              5 |     101 |   48.75 | 2025-12-27 14:27:14.421499 | Dining
              6 |     101 |   55.00 | 2025-12-28 14:27:14.421499 | Utilities
              7 |     101 | 5000.00 | 2026-01-02 13:27:14.423614 | Electronics
              8 |     102 |  120.00 | 2025-12-30 14:27:14.42523  | Retail
              9 |     102 |  110.00 | 2025-12-31 14:27:14.42523  | Retail
             10 |     102 |  130.00 | 2026-01-01 14:27:14.42523  | Dining
(10 rows)
```

Step 2.4: The Handoff

To ensure efficiency, I utilized the `COPY (...) TO STDOUT` command. This effectively "pipes" the filtered anomaly data directly from the database container to a local CSV file (`anomalies.csv`), eliminating manual data entry.

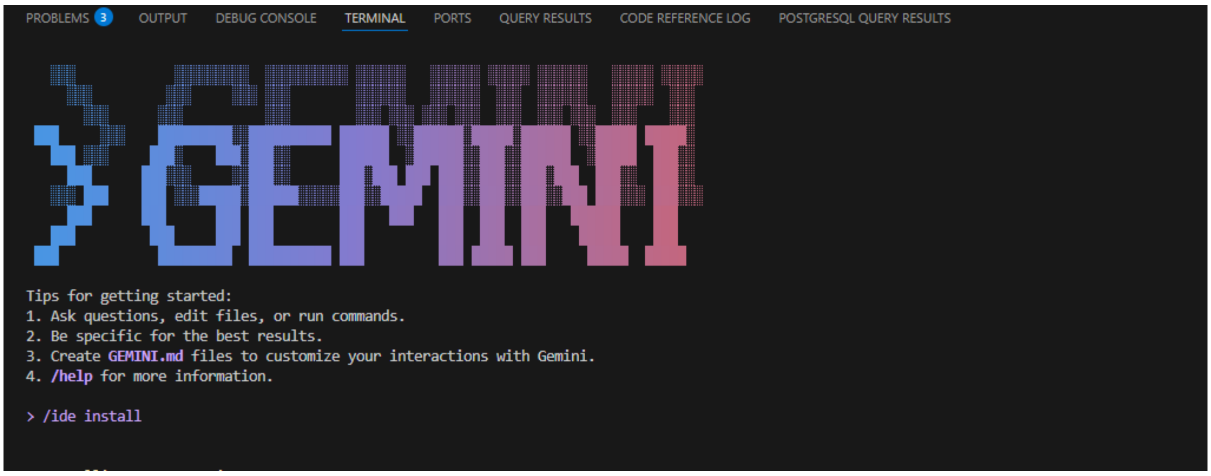
Phase 3: Automated Analysis (The "Analyst")

The final stage interprets the raw data. The CSV is fed into a custom CLI Analyzer powered by the Google Gemini API.

Implementation: A Python script reads the CSV from Standard Input (STDIN).

Analysis: The script dynamically identifies the available Gemini model, constructs a prompt embedding the transaction data, and requests a professional fraud report.

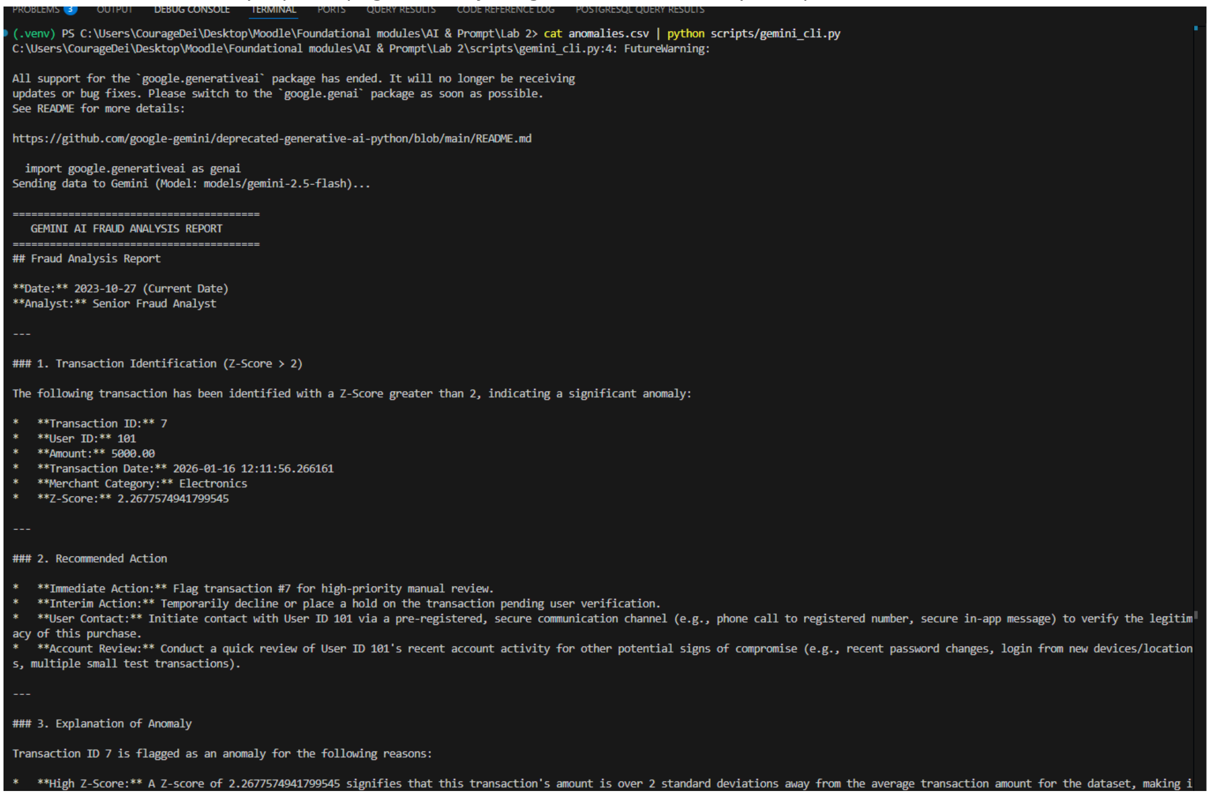
SCREENSHOT: 5 - gemini_cli.png - Execution of the custom Python CLI script piping data to the Gemini API



```
Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. Create GEMINI.md files to customize your interactions with Gemini.
4. /help for more information.

> /ide install
```

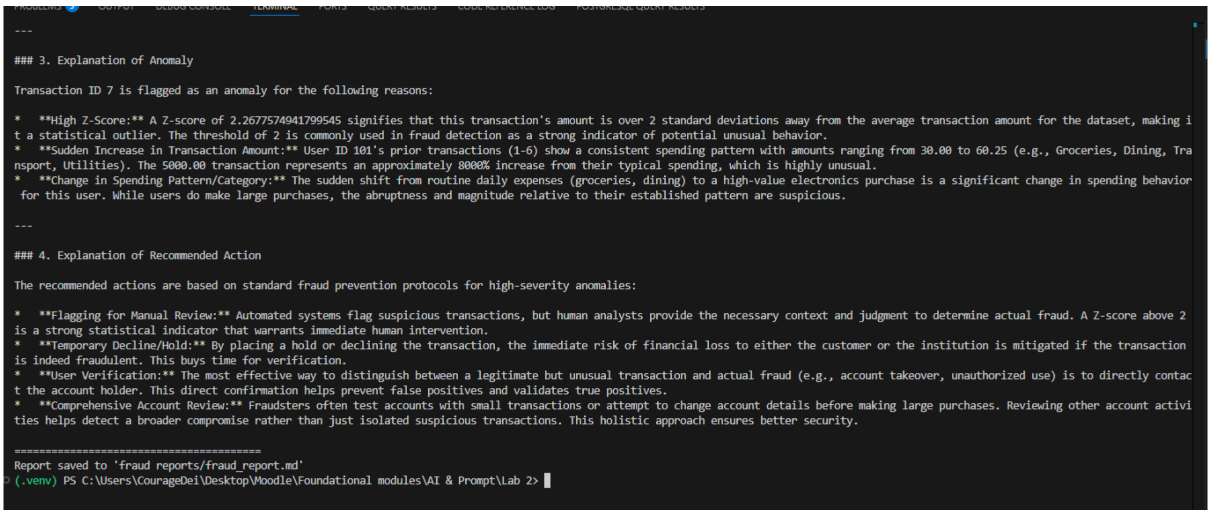
Output: The system generates a `fraud_report.md` file containing the specific reason for the flag and a recommended action (e.g., "Freeze Account").

SCREENSHOT: 6 - Anomaly report 1.png - Part 1 of the generated Fraud Analysis Report



```
(.venv) PS C:\Users\CourageDei\Desktop\Moodle\Foundational modules\AI & Prompt\Lab 2> cat anomalies.csv | python scripts/gemini_cli.py
C:\Users\CourageDei\Desktop\Moodle\Foundational modules\AI & Prompt\Lab 2\scripts\gemini_cli.py:4: FutureWarning:

All support for the `google.generativeai` package has ended. It will no longer be receiving
updates or bug fixes. Please switch to the `google.genai` package as soon as possible.
See README for more details:

https://github.com/google-gemini/deprecated-generative-ai-python/blob/main/README.md

  import google.generativeai as genai
Sending data to Gemini (Model: models/gemini-2.5-flash)...

========================================
   GEMINI AI FRAUD ANALYSIS REPORT
========================================
## Fraud Analysis Report

**Date:** 2023-10-27 (Current Date)
**Analyst:** Senior Fraud Analyst

---

### 1. Transaction Identification (Z-Score > 2)

The following transaction has been identified with a Z-Score greater than 2, indicating a significant anomaly:

*   **Transaction ID:** 7
*   **User ID:** 101
*   **Amount:** 5000.00
*   **Transaction Date:** 2026-01-16 12:11:56.266161
*   **Merchant Category:** Electronics
*   **Z-Score:** 2.2677574941799545

---

### 2. Recommended Action

*   **Immediate Action:** Flag transaction #7 for high-priority manual review.
*   **Interim Action:** Temporarily decline or place a hold on the transaction pending user verification.
*   **User Contact:** Initiate contact with User ID 101 via a pre-registered, secure communication channel (e.g., phone call to registered number, secure in-app message) to verify the legitim
acy of this purchase.
*   **Account Review:** Conduct a quick review of User ID 101's recent account activity for other potential signs of compromise (e.g., recent password changes, login from new devices/location
s, multiple small test transactions).

---

### 3. Explanation of Anomaly

Transaction ID 7 is flagged as an anomaly for the following reasons:

*   **High Z-Score:** A Z-score of 2.2677574941799545 signifies that this transaction's amount is over 2 standard deviations away from the average transaction amount for the dataset, making i
```

SCREENSHOT: 7 - Anomaly report 2.png - Part 2 of the Report showing the Statistical Insight and Recommendation

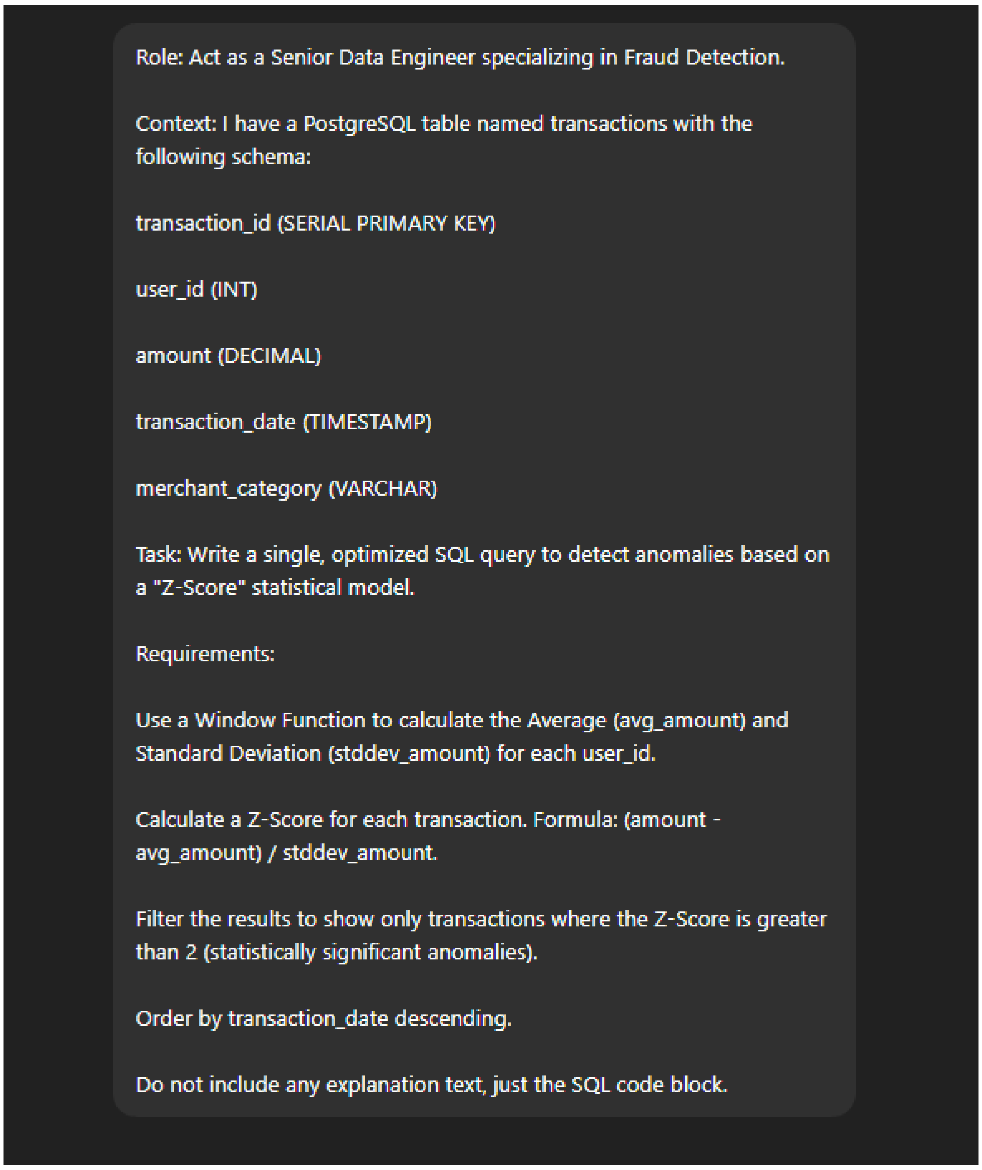


```
---

### 3. Explanation of Anomaly

Transaction ID 7 is flagged as an anomaly for the following reasons:

*   **High Z-Score:** A Z-score of 2.2677574941799545 signifies that this transaction's amount is over 2 standard deviations away from the average transaction amount for the dataset, making i
t a statistical outlier. The threshold of 2 is commonly used in fraud detection as a strong indicator of potential unusual behavior.
*   **Sudden Increase in Transaction Amount:** User ID 101's prior transactions (1-6) show a consistent spending pattern with amounts ranging from 30.00 to 60.25 (e.g., Groceries, Dining, Tra
nsport, Utilities). The 5000.00 transaction represents an approximately 8000% increase from their typical spending, which is highly unusual.
*   **Change in Spending Pattern/Category:** The sudden shift from routine daily expenses (groceries, dining) to a high-value electronics purchase is a significant change in spending behavior
 for this user. While users do make large purchases, the abruptness and magnitude relative to their established pattern are suspicious.

---

### 4. Explanation of Recommended Action

The recommended actions are based on standard fraud prevention protocols for high-severity anomalies:

*   **Flagging for Manual Review:** Automated systems flag suspicious transactions, but human analysts provide the necessary context and judgment to determine actual fraud. A Z-score above 2
is a strong statistical indicator that warrants immediate human intervention.
*   **Temporary Decline/Hold:** By placing a hold or declining the transaction, the immediate risk of financial loss to either the customer or the institution is mitigated if the transaction
is indeed fraudulent. This buys time for verification.
*   **User Verification:** The most effective way to distinguish between a legitimate but unusual transaction and actual fraud (e.g., account takeover, unauthorized use) is to directly contac
t the account holder. This direct confirmation helps prevent false positives and validates true positives.
*   **Comprehensive Account Review:** Fraudsters often test accounts with small transactions or attempt to change account details before making large purchases. Reviewing other account activi
ties helps detect a broader compromise rather than just isolated suspicious transactions. This holistic approach ensures better security.

========================================
Report saved to 'fraud reports/fraud_report.md'
(.venv) PS C:\Users\CourageDei\Desktop\Moodle\Foundational modules\AI & Prompt\Lab 2>
```

4. Prompt Engineering Strategy

Stage 1: Logic Generation Prompt (ChatGPT)

The following context-rich prompt was used to generate the SQL logic:


Role: Act as a Senior Data Engineer specializing in Fraud Detection.

Context: I have a PostgreSQL table named transactions with the following schema:

transaction_id (SERIAL PRIMARY KEY)

user_id (INT)

amount (DECIMAL)

transaction_date (TIMESTAMP)

merchant_category (VARCHAR)

Task: Write a single, optimized SQL query to detect anomalies based on a "Z-Score" statistical model.

Requirements:

Use a Window Function to calculate the Average (avg_amount) and Standard Deviation (stddev_amount) for each user_id.

Calculate a Z-Score for each transaction. Formula: (amount - avg_amount) / stddev_amount.

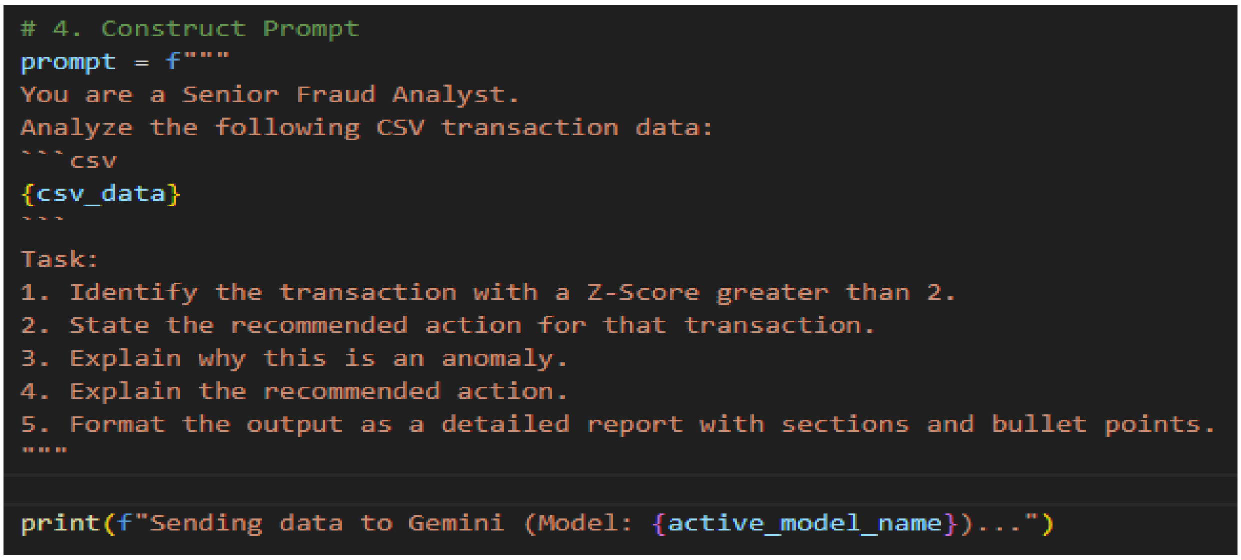
Filter the results to show only transactions where the Z-Score is greater than 2 (statistically significant anomalies).

Order by transaction_date descending.

Do not include any explanation text, just the SQL code block.

Stage 2: Analytical Prompt (Google Gemini)

The following prompt is dynamically constructed within the Python CLI script:


~~~
# 4. Construct Prompt
prompt = f"""
You are a Senior Fraud Analyst.
Analyze the following CSV transaction data:
```csv
{csv_data}
```
Task:
1. Identify the transaction with a Z-Score greater than 2.
2. State the recommended action for that transaction.
3. Explain why this is an anomaly.
4. Explain the recommended action.
5. Format the output as a detailed report with sections and bullet points.
"""

print(f"Sending data to Gemini (Model: {active_model_name})...")
~~~

5. Adaptability & Technical Decisions

A key requirement for this capstone was ensuring the workflow was not dependent on a single proprietary tool.

The Pivot:

Initially, the workflow was designed to utilize the GitHub Copilot CLI. However, during the implementation phase, I discovered that the specific Copilot CLI extension had been deprecated and was no longer functioning reliably.

The Solution:

Instead of abandoning the workflow, I demonstrated adaptability by engineering a custom CLI wrapper using the Google Gemini API (`google-genai`).

Benefit: This modification made the pipeline more robust, as it interacts directly with the API rather than relying on a third-party shell extension.

Automation: The script was further enhanced to automatically detect available models (`gemini-1.5-flash`), ensuring it works regardless of regional model availability.

6. Reflection & Challenges

What I Learned

Through the execution of this capstone, I gained significant practical experience in orchestrating multi-tool workflows.

- API Integration: I moved beyond simple "chat" interfaces and learned how to programmatically interact with Large Language Models using Python and the Google GenAI SDK.
- Infrastructure as Code: Working with docker-compose taught me the value of reproducible environments. I learned how to spin up a database, seed it with data, and tear it down using a single configuration file, which is a standard DevOps practice.
- Prompt Engineering for Code: I learned that generating complex SQL (like Window Functions) requires highly specific, context-rich prompts. Iterating on the prompt to get the correct Z-score calculation was a key learning moment in understanding how to "steer" the AI effectively.

Challenges Faced & Solutions

The most significant challenge was the deprecation of the GitHub Copilot CLI.

- The Problem: Mid-project, I discovered that the tool I intended to use for the analysis phase (gh copilot) was no longer supported, breaking my planned workflow.

- The Solution: Instead of downgrading the project to a manual step, I engineered a custom solution. I wrote a Python script (gemini_cli.py) that pipes standard input to the Google Gemini API.
- The Outcome: This not only solved the immediate problem but resulted in a more robust and adaptable tool that I can customize further, teaching me that adaptability is as important as technical skill in Data engineering.