Project Report

AAPL Stock Price & News Sentiment Analysis

An End-to-End ELT Data Engineering Project

Thanapon Nitchaphatchanakul 66102010168
Tammarat Sukkha 66102010170
Phimchanok Thongpool 66102010181
Phattharamongkolchan Puttiworrapong 66102010182

November 29, 2025

Contents

1 Executive Summary

This project implements an automated **ELT (Extract, Load, Transform)** data pipeline to analyze the correlation between Apple Inc. (AAPL) stock market performance and public news sentiment. Deviating from traditional ETL methods, this system adopts a modern "Local Data Stack" approach—leveraging **Docker, Apache Airflow, MinIO, and DuckDB**.

The system successfully ingests raw financial data, preserves it in a Data Lake (MinIO), processes it using Natural Language Processing (NLP), validates data quality via strict schemas (Pandera), and serves actionable insights through a real-time Dashboard (Streamlit) and REST API (Flask).

2 Project Objectives

The primary goals of this project are twofold:

1. **Analytical Objective:** To determine if daily news sentiment (positive/negative news) has a measurable correlation with AAPL's closing stock price and trading volume.
2. **Engineering Objective:** To construct a robust, reproducible data infrastructure that demonstrates best practices in:
   - **Containerization:** Ensuring the app runs identically on any machine.
   - **Data Lake Architecture:** Separating storage (MinIO) from compute (Python).
   - **Automated Orchestration:** Scheduling workflows without human intervention.
   - **Data Quality:** Implementing automated validation gates.

3 System Architecture

The project follows a modular Microservices architecture, where each component handles a specific responsibility.

3.1 Tech Stack Justification

- **Apache Airflow:** Used for orchestration. It ensures tasks run in a specific order (dependency management) and retries failed tasks automatically.
- **MinIO:** An S3-compatible object storage. It serves as our **Data Lake**, allowing us to save raw data immediately before processing.
- **DuckDB:** An in-process OLAP database. Chosen for its extreme speed in analytical queries (columnar storage) without the overhead of managing a heavy server like PostgreSQL.
- **Pandera:** A statistical validation library. It ensures no "garbage data" (e.g., negative stock prices) enters the analytical database.

3.2 High-Level Architecture Diagram

The data flows from external sources, through the ELT pipeline where merging occurs, and finally to the user interface.

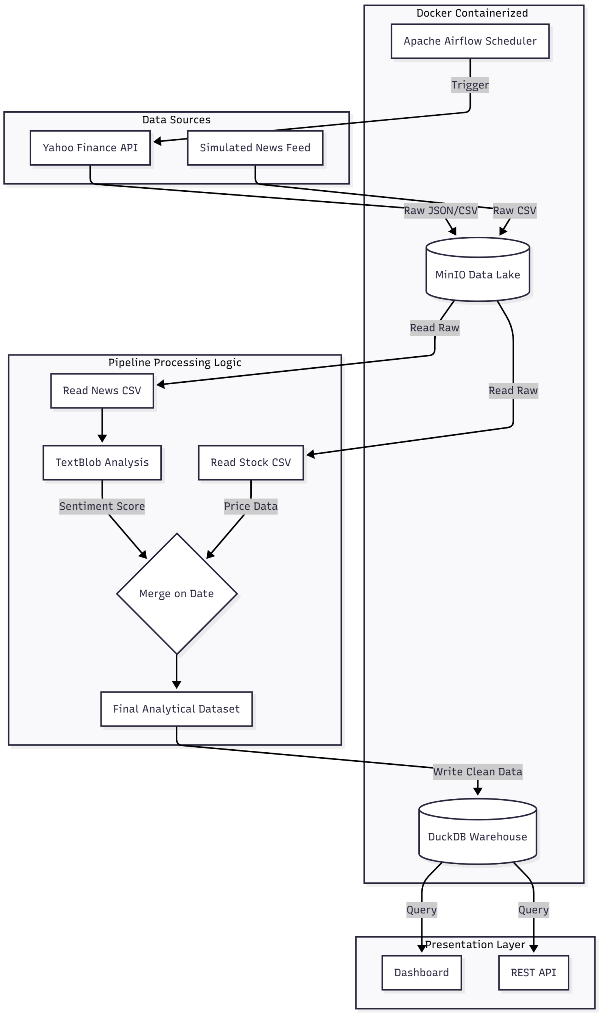

Figure 1: **High-Level System Architecture and Data Flow**

4 Methodology & Implementation Details

This section explains the logic implemented in the `pipeline.py` and supporting files.

4.1 Phase 1: Infrastructure Setup (`docker-compose.yaml`)

We use Docker Compose to spin up the entire environment.

- **Networking:** A dedicated network (`elt_network`) allows containers (Airflow, MinIO) to communicate securely using hostnames instead of IP addresses.
- **Volume Mounting:** Data persistence is handled via Docker Volumes, ensuring that if a container restarts, the database and MinIO files are not lost.

4.2 Phase 2: Extraction & Loading (The "EL" in ELT)

File: `pipeline.py` → `extract_and_load()`

Instead of processing data immediately, we follow the **ELT pattern**:

1. **Extraction:** The script fetches the last 7 days of AAPL data using the `yfinance` library.
2. **Loading (Raw Layer):** This data is immediately saved as a CSV file into the **MinIO bucket** (`raw-data`).
   - *Why this matters:* By saving the raw data first, we create a "Source of Truth." If our analysis code has a bug, we can fix the code and re-run it against this raw data without needing to fetch from the API again.

4.3 Phase 3: Transformation (The "T" in ELT)

File: `pipeline.py` → `transform_sentiment()`

Once the data is safely in the Data Lake, we transform it:

1. **Sentiment Scoring:** We use **TextBlob**, a simple NLP library, to analyze news headlines. It assigns a polarity score:
   - `-1.0`: Very Negative
   - `0.0`: Neutral
   - `+1.0`: Very Positive
2. **Aggregation:** Since there may be multiple news articles in one day, we group them by date and calculate the **mean (average) sentiment**.

3. **Data Merging (Merge on Date):** A critical step in this pipeline is the unification of the two distinct datasets: **Stock Market Data** and **News Sentiment Data**. This is achieved through a 'Merge on Date' operation (specifically an **Inner Join**).

   - **Mechanism:** The system aligns the `Date` index of the stock dataframe with the `Date` column of the aggregated sentiment dataframe.
   - **Handling Non-Trading Days:** Stock markets are closed on weekends and holidays, whereas news is generated daily. By performing an Inner Join, the pipeline automatically filters out news data from non-trading days (e.g., Saturday/Sunday) where no corresponding stock price exists. This ensures that the correlation analysis is strictly performed on active market days, preventing data skew.
   - **Verification:** The subsequent `validate_data` function employs a `Pandera` schema that checks both `close_price` and `daily_sentiment` within a single DataFrame, confirming that the merge operation was successful.

4.4 Phase 4: Data Validation & Warehousing

File: `pipeline.py` → `validate_data()` & `load_to_warehouse()`

Before saving to the database, the system acts as a gatekeeper:

1. **Validation:** Using **Pandera**, we define a strict schema:
   - `close_price`: Must be greater than 0.
   - `daily_sentiment`: Must be between -1 and 1.
   - *Result:* If data violates these rules, the pipeline stops immediately, preventing corrupt analytics.
2. **Warehousing:** Validated data is written to `aapl_warehouse.db` (DuckDB). We use the `CREATE OR REPLACE TABLE` command to ensure the table always reflects the latest pipeline run.

5 Presentation Layer

To make the data accessible to end-users, we implemented two interfaces.

5.1 Interactive Dashboard (`dashboard.py`)

Built with **Streamlit**, this tool allows non-technical users to explore the data.

- **Key Feature:** A dual-axis chart (created with Plotly) overlays the Stock Price (Line chart) against the Sentiment Score (Bar chart).
- **Interactivity:** Users can trigger the pipeline manually via a "Run Pipeline" button, which executes `pipeline.main()` in the background.
- **Performance:** We use the `@st.cache_data` decorator. This stores the result of database queries in memory, making the dashboard load instantly on subsequent visits.

6 Group Contributions

This project was a collaborative effort involving full-stack data engineering. Based on the project files (`pipeline.py`, `dashboard.py`, `api.py`, `docker-compose.yaml`), the responsibilities were divided to cover Infrastructure, Data Pipeline, Quality Assurance, and Application layers.

- **Thanapon Nitchaphatchanakul (66102010168): Project Planning & Coordination**
  - **Workflow Planning:** Structured the project roadmap and defined sequential development steps to ensure the entire team had a clear understanding of the pipeline architecture.
  - **Task Allocation:** Managed the division of labor by assigning tasks based on individual member aptitudes, ensuring efficient project progression.
  - **Infrastructure Support:** Assisted in the planning and setup of the foundational infrastructure required for the project.
- **Tammarat Sukkha (66102010170): Lead Developer & DevOps**
  - **Core Pipeline Development:** Served as the main developer for the ELT architecture, responsible for writing the core logic for data extraction and loading.
  - **DevOps & Automation:** Implemented the containerization strategy using Docker and configured the Apache Airflow environment for automated orchestration.
  - **System Integration:** Ensured seamless connectivity and data flow between the various services (MinIO, DuckDB, Python scripts).
- **Phimchanok Thongpool (66102010181): Conceptualization & QA**
  - **Transformation Strategy:** Initiated ideas for the Data Transformation phase, specifically selecting Sentiment Analysis as a key analytical component.
  - **Quality Assurance (QA):** Defined the standards for data quality and designed validation schemas (using Pandera) to ensure the integrity of the dataset.
  - **Logic Validation:** Verified the correctness of the transformation logic to ensure accurate analytical results.
- **Phattharamongkolchan Puttiworrapong (66102010182): Application Layer Architect**
  - **Dashboard Design:** Led the design and development of the Application Layer, specifically the interactive Dashboard, ensuring an intuitive user experience.
  - **API Implementation:** Developed the REST API to expose processed data to external systems.
  - **Visualization:** Selected and implemented the visualization tools to effectively present the correlation between stock prices and sentiment.

6.1 REST API (`api.py`)

Built with **Flask**, this API allows other software systems to consume our data.

- **Endpoint:** `/api/v1/stock_summary` returns JSON data.
- **Reliability:** The database connection logic includes a **retry mechanism** (exponential backoff). Since DuckDB is file-based, it can be locked if multiple processes access it simultaneously. This retry logic ensures the API doesn't crash during a write operation.

7 Conclusion

This project successfully demonstrates the capability to build a production-grade data pipeline on a local machine. By adhering to the **ELT paradigm**, we ensured that:

1. **Data Lineage** is preserved (via MinIO Raw storage).
2. **Data Quality** is guaranteed (via Pandera validation).
3. **Scalability** is inherent (via Docker containerization).

The correlation analysis provided by the dashboard offers a foundation for further financial modeling, proving that modern open-source tools can effectively handle complex financial data engineering tasks.