Stock Price Prediction for Amazon (AMZN) and Meta (META) using Airflow, yfinance, and Snowflake

Danish Waseem (019101511), Srinidhi Jaya Revanth Srirangarajapally (019123143)
Department of Applied Data Science,
College of Information, Data, and Society,
San Jose State University, San Jose, California, USA 95192
Email: *danish.waseem@sjsu.edu, srinidhijayarevanth.srirangarajapally@sjsu.edu*

Abstract

This paper presents a daily automated system that downloads stock prices for Amazon (AMZN) and Meta (META), stores them in Snowflake, and predicts prices for the next seven days using Snowflake ML Forecast. The workflows are built and scheduled using Apache Airflow and monitored through its web interface. This report describes the motivation, requirements, workflow design, database schema, and implementation details of the system. Screenshots, code repositories, and outputs are referenced inline and in the Appendix.

I. Team Introduction

Danish Waseem led ETL pipeline development, Snowflake schema, and orchestration.
Srinidhi Jaya Revanth Srirangarajapally implemented forecasting DAG, SQL modeling in Snowflake ML, and documentation.
Both collaborated on system design, testing, and report writing.

II. Problem Statement

Stock price forecasting is a key application in data analytics that requires processing large volumes of time-series data. Manual data collection and modeling are error-prone and time-consuming. This project automates the entire workflow: data extraction using `yfinance`, transformation and loading through Airflow, and predictive modeling using Snowflake ML. The system collects 180 days of OHLCV (Open, High, Low, Close, Volume) data for AMZN and META, stores it in Snowflake, and automatically forecasts prices for the next seven days.

III. Dataset(s)

We use **Yahoo Finance** via the `yfinance` Python package:

- **Symbols:** `AMZN`, `META`
- **Fields:** Open, High, Low, Close, Volume
- **Horizon:** Most recent 180 trading days (daily interval)
- **Access:** Public API wrapper; no keys required

IV. System Diagram

Figure 1 depicts the end-to-end flow from extraction to prediction. The system ensures data integrity, modularity, and full automation from raw data ingestion to forecast output.

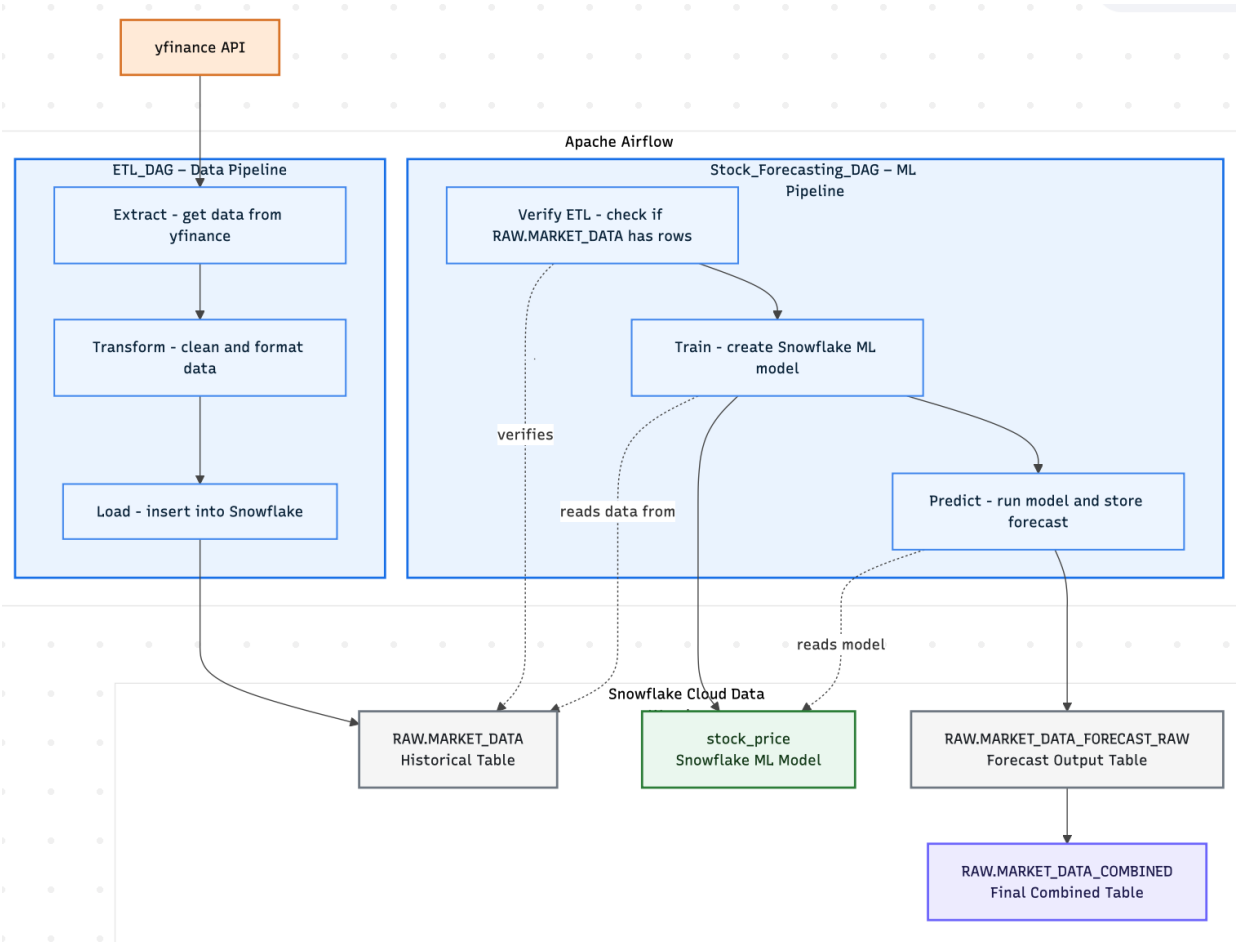

Fig. 1: System Architecture: yfinance → Airflow ETL DAG → Snowflake (RAW.MARKET_DATA) → Airflow Forecasting DAG → Final Output.

V. Tables

A. *RAW.MARKET_DATA (Historical Data)*

TABLE I: Snowflake Table: `RAW.MARKET_DATA`

Column	Type	Null	Constraints / Notes
`DATE`	`DATE`	No	Trading day (UTC). Logical uniqueness with `SYMBOL`.
`OPEN`	`FLOAT`	No	Open price for the day.
`HIGH`	`FLOAT`	No	High price for the day.
`LOW`	`FLOAT`	No	Low price for the day.
`CLOSE`	`FLOAT`	No	Close price (fallback to Adj Close if needed).
`VOLUME`	`BIGINT`	No	Total traded volume for the day.
`SYMBOL`	`VARCHAR(10)`	No	Ticker symbol (`AMZN`, `META`).
`CREATED_AT`	`TIMESTAMP_NTZ`	Yes	Ingestion timestamp; default `CURRENT_TIMESTAMP()`.
Logical Primary Key: (`SYMBOL`, `DATE`).			

A populated view of `RAW.MARKET_DATA` is shown in Figure 2.

B. *RAW.MARKET_DATA_COMBINED (History ∪ Forecast)*

A sample of the final combined output is shown in Figure 3.

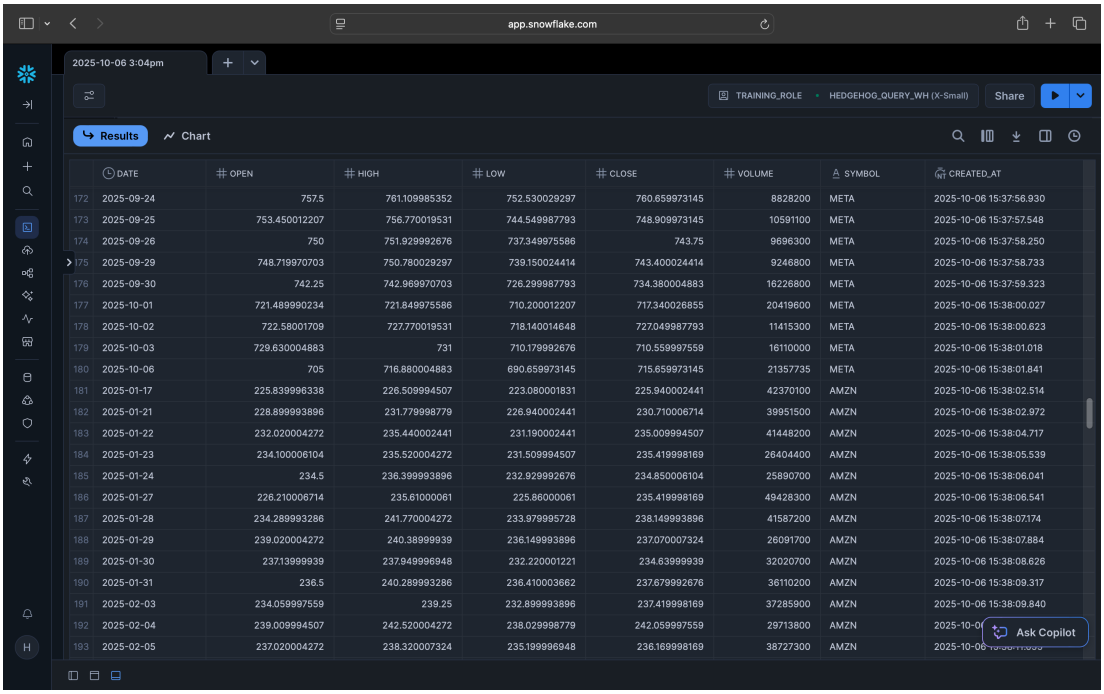

Fig. 2: Snowflake table `RAW.MARKET_DATA` populated by ETL_DAG.

TABLE II: Snowflake Table: `RAW.MARKET_DATA_COMBINED`

Column	Type	Null	Constraints / Notes
`SYMBOL`	`VARCHAR(10)`	No	Ticker symbol.
`DATE`	`DATE`	No	History date or forecasted date (cast from timestamp).
`ACTUAL`	`FLOAT`	Yes	Close price for history rows; `NULL` for forecasts.
`FORECAST`	`FLOAT`	Yes	Model forecast for forecast rows; `NULL` for history.
`LOWER_BOUND`	`FLOAT`	Yes	Lower bound of 95% interval.
`UPPER_BOUND`	`FLOAT`	Yes	Upper bound of 95% interval.
Row Semantics: Exactly one of {`ACTUAL`, `FORECAST`} is non-`NULL`.			

VI. Implementation Details

A. Scheduling and Orchestration

Airflow schedules two DAGs:

- **ETL_DAG** at 02:30 daily (Extract → Transform → Load).
- **Stock_Forecasting_DAG** at 03:00 daily (Verify → Train → Predict).

B. ETL_DAG Pipeline

Extract uses `yfinance.Ticker().history()` for a clean DataFrame. **Transform** drops invalid rows. **Load** executes DDL and idempotent upserts (implemented as delete+insert) inside a transaction. Figure 4 shows the ETL DAG in the Airflow UI.

C. Forecasting Pipeline

Verify checks that `RAW.MARKET_DATA` has rows. **Train** creates `MARKET_DATA_v1` and the model `stock_price`. **Predict** calls `!FORECAST`, stores the raw output, creates a clean view, and builds the combined table. Figure 5 shows the Forecasting DAG.

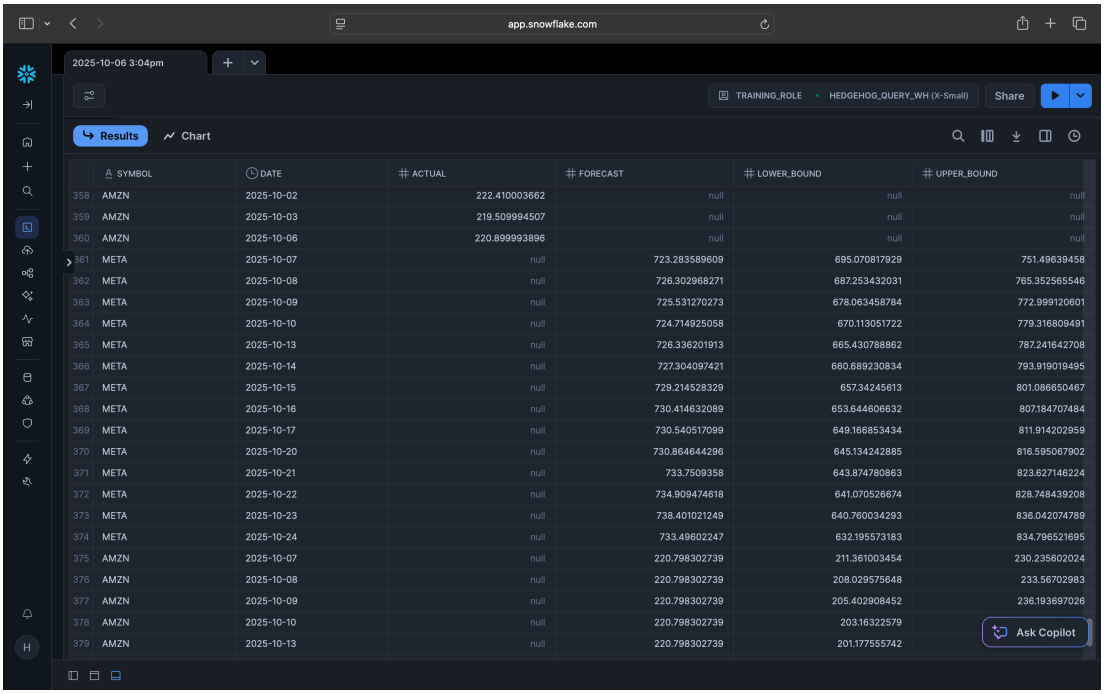

Fig. 3: Forecast output in `RAW.MARKET_DATA_COMBINED` (history ∪ forecast).

D. Key SQL (Snowflake)

Model creation:

```
CREATE OR REPLACE SNOWFLAKE.ML.FORECAST stock_price(
INPUT_DATA => SYSTEM$REFERENCE('VIEW', 'MARKET_DATA_v1'),
SERIES_COLNAME => 'SYMBOL',
TIMESTAMP_COLNAME => 'DATE_v1',
TARGET_COLNAME => 'CLOSE',
CONFIG_OBJECT => { 'ON_ERROR': 'SKIP' }
);
```

Forecast execution and capture:

```
BEGIN
CALL stock_price!FORECAST(
FORECASTING_PERIODS => 14,
CONFIG_OBJECT => {'prediction_interval': 0.95}
);
LET x := SQLID;
CREATE OR REPLACE TABLE RAW.MARKET_DATA_FORECAST_RAW AS
SELECT * FROM TABLE(RESULT_SCAN(:x));
END;
```

Final union:

```
CREATE OR REPLACE TABLE RAW.MARKET_DATA_COMBINED AS
SELECT SYMBOL, DATE, CLOSE AS ACTUAL, NULL AS FORECAST, NULL AS LOWER_BOUND, NULL AS
    UPPER_BOUND
FROM RAW.MARKET_DATA
WHERE SYMBOL IN ('META','AMZN')
```

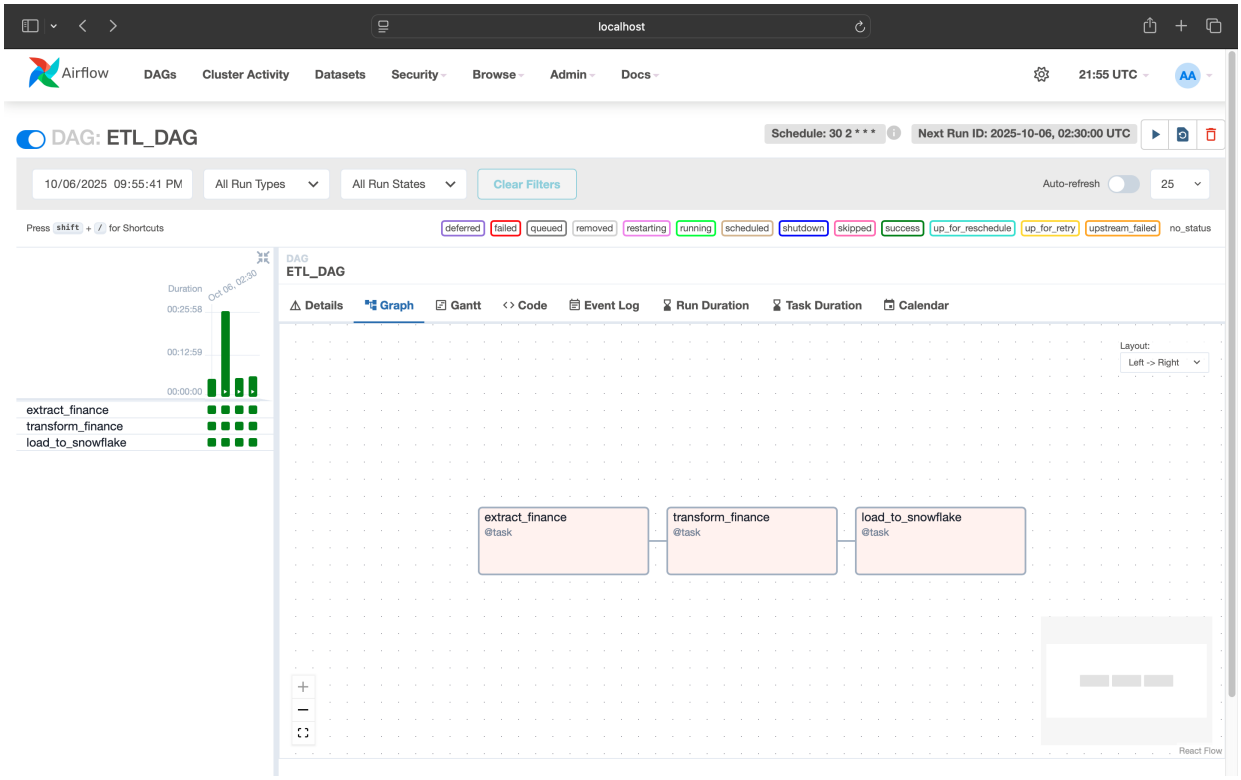

Fig. 4: Airflow ETL DAG: Extract → Transform → Load.

```
UNION ALL
SELECT SYMBOL, CAST(DATE_v1 AS DATE), NULL AS ACTUAL, FORECAST, LOWER_BOUND,
    UPPER_BOUND
FROM MARKET_DATA_FORECAST_CLEAN;
```

VII. Lessons

- Using `Ticker().history()` avoids MultiIndex issues from `yf.download()` for single symbols.
- Always wrap Snowflake writes in transactions for safe retries.
- Validate data before model training (our Verify task prevents empty-train runs).
- Keep DAGs modular; it simplifies debugging and re-runs.

VIII. Future Work

- Add more symbols and longer forecasting windows.
- Include technical indicators (moving averages, Bollinger Bands) for trend analysis.

IX. Conclusion

Two Airflow DAGs automate stock ingestion and forecasting for AMZN and META. Airflow provides scheduling/observability, while Snowflake ML enables fast model training and inference. The Verify–Train–Predict flow maintains data quality and reproducibility.

Acknowledgment

The authors thank Professor Keeyong Han and SA Jeff Chong for their valuable feedback and guidance throughout the project.

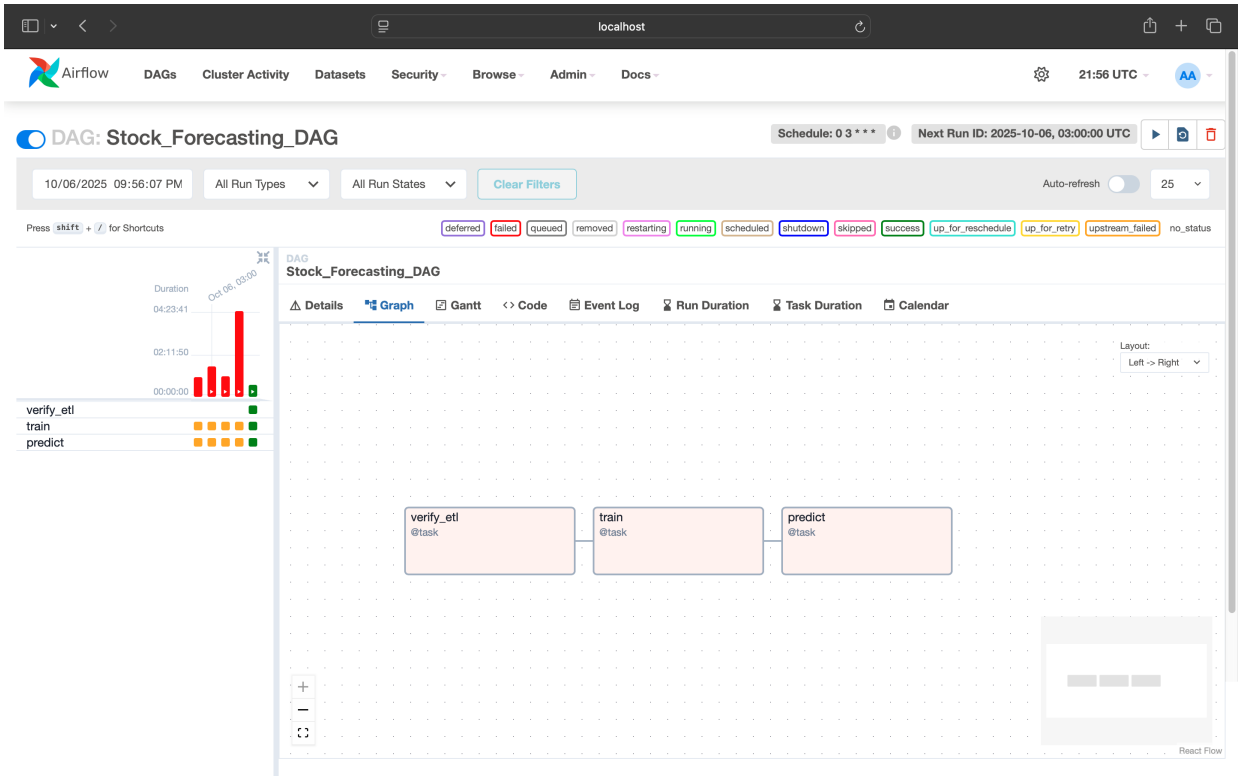

Fig. 5: Airflow Stock Forecasting DAG: Verify → Train → Predict.

REFERENCES

1) yfinance: https://pypi.org/project/yfinance/
2) Snowflake ML Forecast: https://docs.snowflake.com/en/sql-reference/ml-forecast
3) Apache Airflow: https://airflow.apache.org/docs/

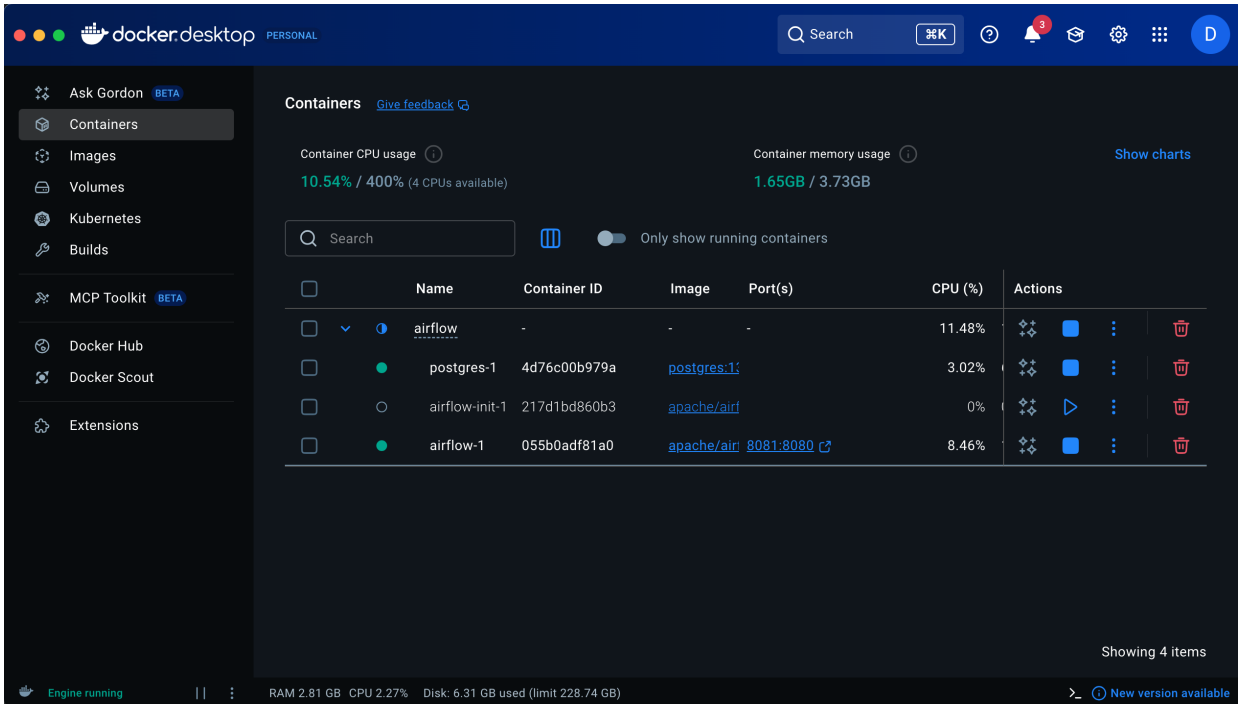

Fig. 6: Docker Desktop

Appendix: Source Code and Repositories

All Python DAGs, SQL scripts, and supplementary files are hosted on GitHub:

- Danish Waseem (Main Repo): https://github.com/danwaseem/SJSU-DATA226/tree/main/LAB1
- Srinidhi Jaya Revanth Srirangarajapally (Main Repo): https://github.com/Revanth0211/DATA-226-Lab-1-
- Danish Waseem (Airflow DAGs): https://github.com/danwaseem/SJSU-DATA226/tree/main/Airflow
- Srinidhi Jaya Revanth Srirangarajapally (Airflow DAGs): https://github.com/Revanth0211/Stock-Prices-ETL-Airflo

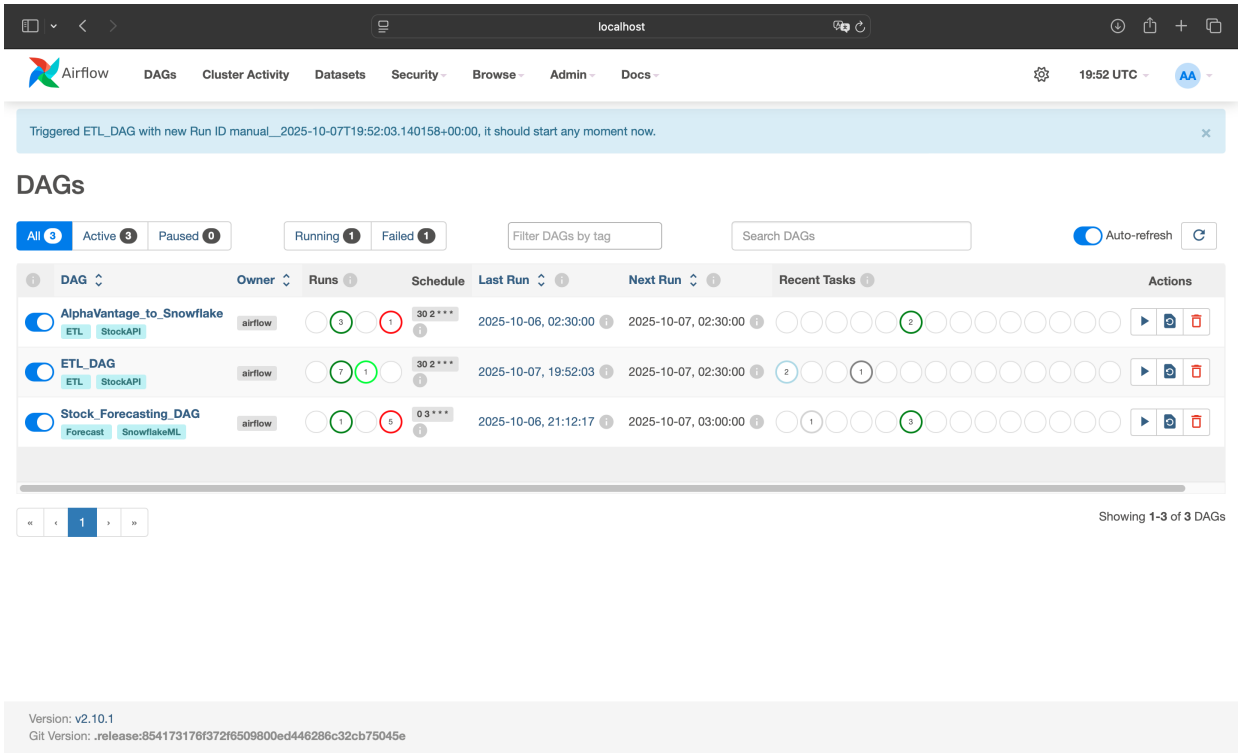

Fig. 7: DAGS

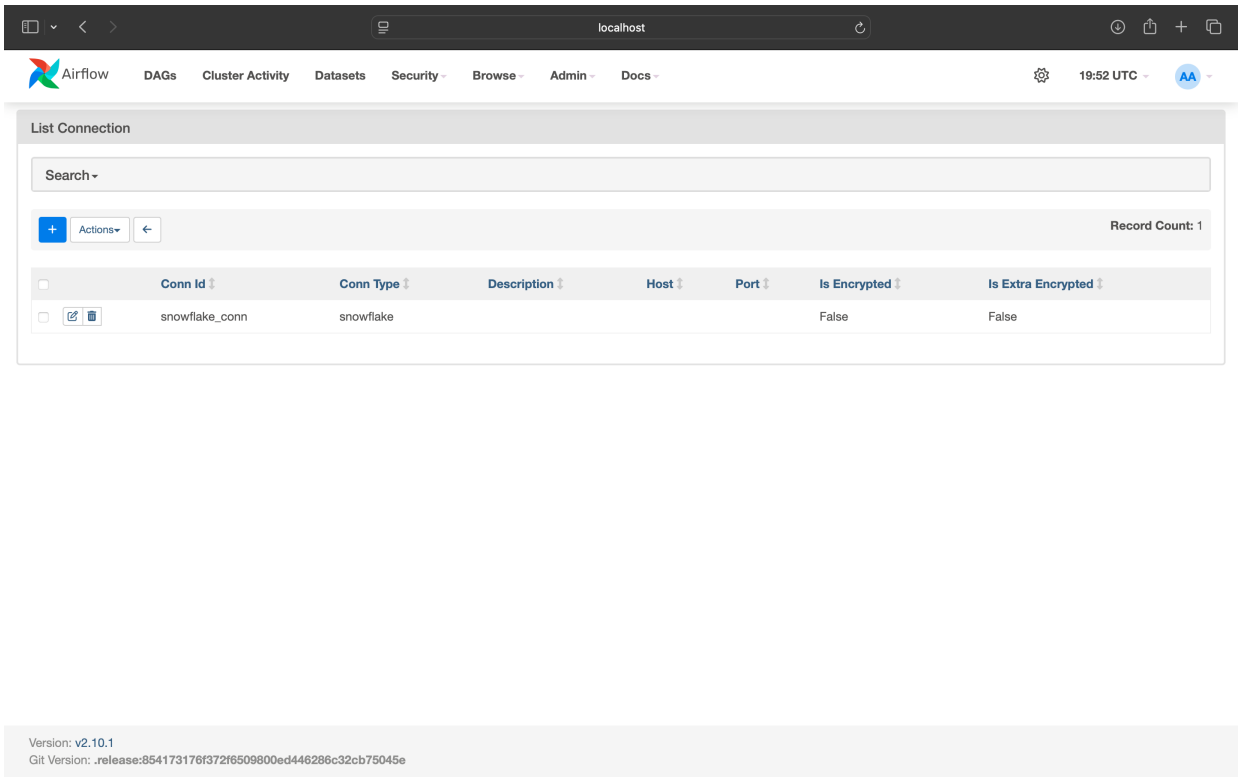

Fig. 8: Snowflake Connection in Airflow

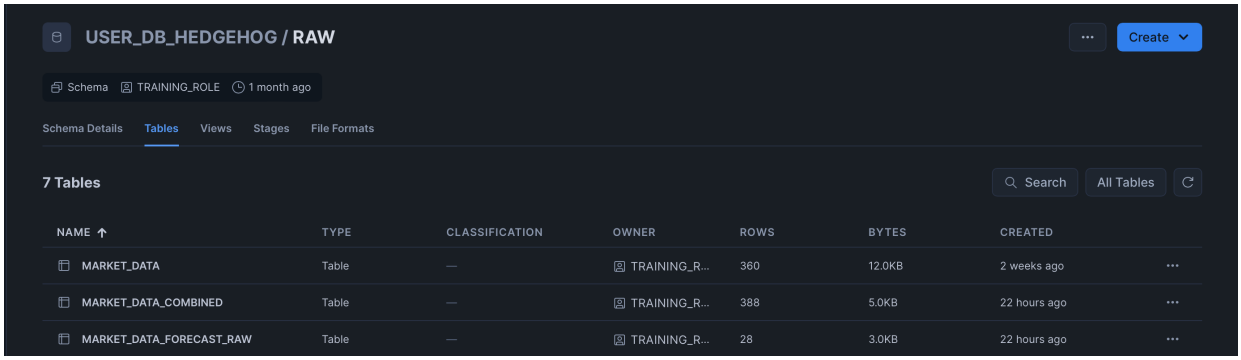

Fig. 9: Snowflake Tables

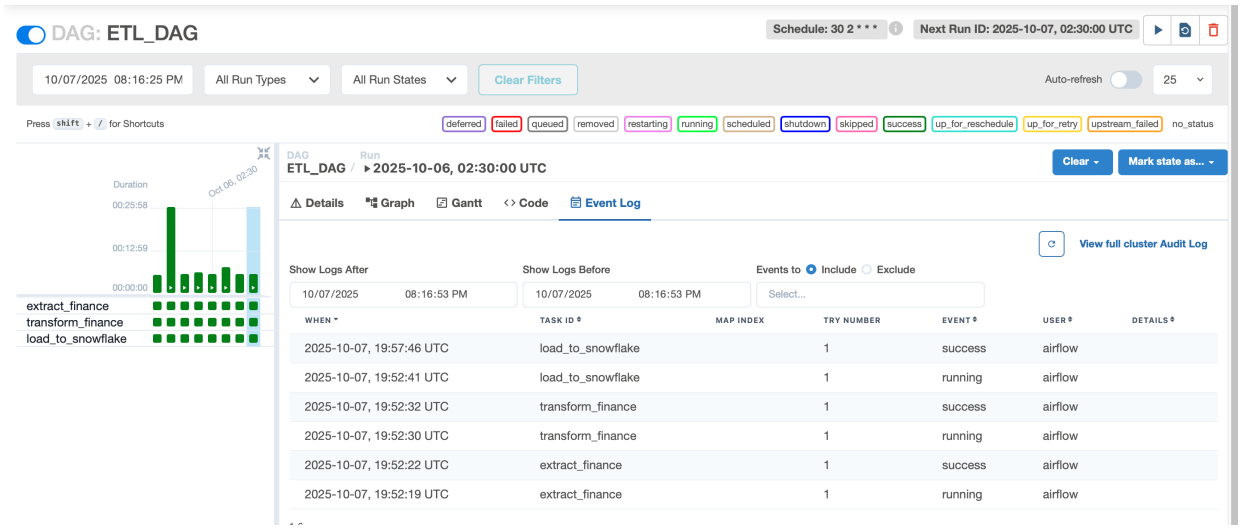

Fig. 10: ETL DAG Event Log

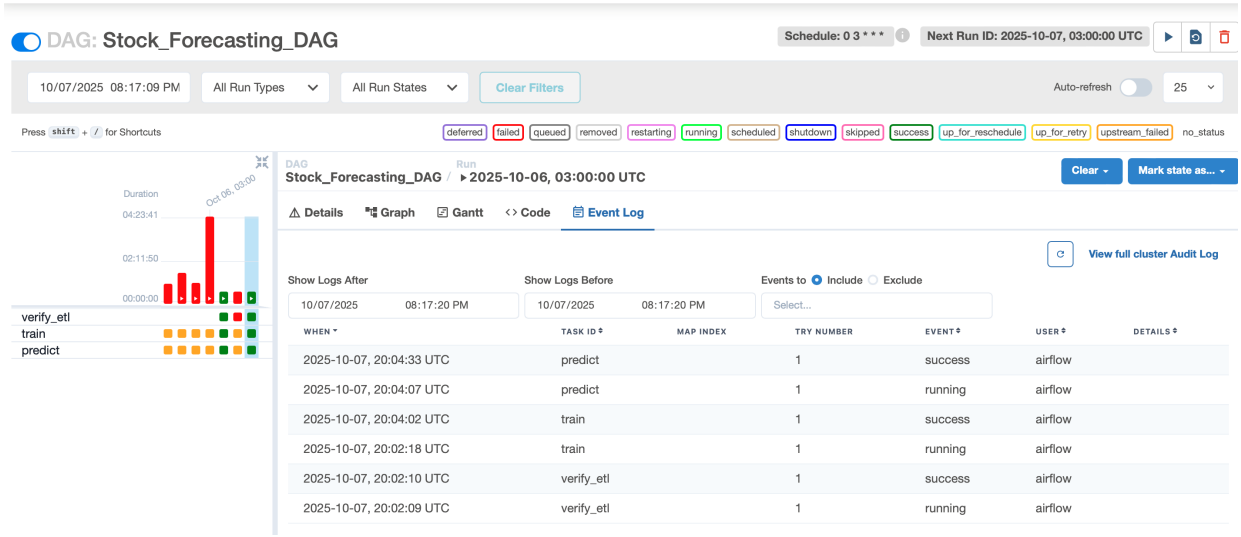

Fig. 11: Stock Forecasting DAG Event Log