Building a Stock Price Prediction Analytics using Snowflake & Airflow

Yuyao Ding Naga Sai Haritha Gottumukkala

***Abstract*—This project focuses on the development of an automated stock price prediction analytics system that integrates Snowflake and Apache Airflow to streamline data ingestion and forecasting processes. Historical financial data for at least two companies (AAPL and NVDA) is collected from the yfinance API and systematically stored in Snowflake's RAW schema through a daily ETL pipeline. A second pipeline, built on Snowflake's ML Forecasting capabilities, is scheduled after the ETL process to predict the next seven days of closing prices.**

The final stage of the system combines the original stock data with the forecasted results into a unified analytics table using transactional SQL operations with error handling. This architecture ensures reliable scheduling, secure credential management through Airflow connections and variables, and the creation of an end-to-end automated workflow that supports repeatability and scalability. The resulting framework provides a structured foundation for time-series analysis and can be extended to additional symbols or longer forecast horizons, demonstrating the effectiveness of combining orchestration tools and cloud-based machine learning within a single data pipeline.

***Index Terms*—Stock price forecasting, Apache Airflow, Snowflake, ETL pipeline, time-series prediction, workflow orchestration**

I. Team Introduction

This project was collaboratively developed by two M.S. Data Analytics students as part of the Data226 course. The team jointly worked on designing, implementing, and documenting an automated stock price prediction system that integrates data engineering and machine learning components.

Team Members:

- **Yuyao Ding** — Focused on system design, data pipeline development, and overall project integration.
- **Naga Sai Haritha Gottumukkala** — Contributed to the ETL and machine learning components, as well as testing and documentation.

II. Problem Statement

Stock market data is inherently dynamic, with prices fluctuating by the second and massive volumes of historical records being generated daily. Analysts, traders, and financial institutions depend on reliable, up-to-date data to make informed investment and risk management decisions. Traditional approaches for collecting and forecasting stock prices often involve manual data downloads, spreadsheet processing, and locally executed scripts. These methods are slow, prone to human error, and lack scalability, making them unsuitable for continuous or multi-symbol monitoring.

Another challenge lies in the integration of data storage and predictive modeling. While APIs such as *yfinance* provide rich access to historical stock prices, the data alone does not provide actionable forecasts without additional processing. Furthermore, executing machine learning forecasting models in an ad hoc manner results in inconsistencies, poor reproducibility, and difficulty in tracking model performance over time.

Therefore, there is a clear need for a unified, automated system that:

- **Extracts and updates** stock price data for multiple companies on a regular schedule without manual intervention.
- **Stores** this data in a structured, queryable, and scalable warehouse environment (Snowflake) to support downstream analytics.
- **Executes forecasting models** directly within the data warehouse to predict future stock prices in a repeatable and standardized way.
- **Combines historical and predicted data** into one analytics-ready table for visualization and reporting.
- **Manages orchestration and dependencies** through a scheduling framework such as Apache Airflow to ensure that forecasting runs only after successful data ingestion.
- **Implements error handling and transactions** to guarantee data quality and pipeline reliability even in the presence of API failures or unexpected events.

By addressing these needs, the project aims to deliver a robust, end-to-end pipeline that eliminates manual effort, reduces the risk of errors, and provides a scalable platform for time-series forecasting of stock prices. The resulting system will be flexible enough to accommodate additional symbols, extended forecast horizons, or new modeling approaches in the future.

III. Solution Requirements

This section outlines both the functional and non-functional requirements of the system, as well as its intended usage and current limitations.

A. Functional Requirements

The system must meet the following core functional requirements:

1) **Data Ingestion from yfinance:** The system should fetch the latest 180 days of historical stock data for at least two stock symbols (AAPL, NVDA, TSLA, MSFT) using the public yfinance API. This data includes Open, High, Low, Close, and Volume (OHLCV) fields.

2) **ETL Pipeline via Apache Airflow:** The data ingestion process must be automated using an Airflow DAG. The DAG includes three stages:
   - *Extraction:* Pull raw data from the API.
   - *Transformation:* Format dates, handle missing values, and ensure schema consistency with Snowflake tables.
   - *Loading:* Insert the cleaned data into the Snowflake table `RAW.STOCK_PRICES`.
3) **ML Forecasting using Snowflake:** A second Airflow DAG trains a Snowflake ML Forecast model using the historical data and generates predictions for the next seven days for each stock symbol. Forecast results are stored in `MODEL.STOCK_FORECAST`.
4) **Data Merging:** The historical and forecasted data are combined into a final table `ANALYTICS.FINAL_STOCK_PRICES` using SQL transactions with error handling. This ensures data integrity and reliability.
5) **DAG Scheduling and Dependencies:** The ML Forecasting DAG should only execute after successful completion of the ETL DAG. Dependencies between them are managed using Airflow's `TriggerDagRunOperator` or `ExternalTaskSensor`.
6) **GitHub Integration:** All Airflow DAG scripts and SQL code are version-controlled and publicly accessible through a GitHub repository. The report includes references to these source files.

B. Non-Functional Requirements

The system must also satisfy several non-functional requirements to ensure robustness, maintainability, and security:

1) **Security and Secrets Management:** Sensitive credentials such as Snowflake account information and API keys must be securely stored using Airflow Connections and Variables. Hardcoding credentials is not allowed.
2) **Reliability and Idempotency:** The system must prevent duplicate record insertion using SQL `MERGE` operations or primary key constraints. Error handling ensures smooth recovery from API or data inconsistencies.
3) **Scheduling and Automation:** The yfinance ETL DAG runs daily at a fixed time, and the Forecasting DAG follows automatically, ensuring fresh predictions are generated every day.
4) **Observability and Monitoring:** All DAGs and tasks must include clear logs within the Airflow Web UI. Failures or skipped runs can be traced through Airflow's task history.
5) **Scalability and Extensibility:** The architecture should allow easy addition of new stock symbols, changes in forecast duration, or alternative ML models without redesigning the pipeline.
6) **Reproducibility and Standardization:** All SQL operations follow standardized naming conventions, and forecast outputs maintain a consistent schema to facilitate integration with visualization tools.

C. Usage

The system runs automatically once deployed in Airflow. Daily stock data ingestion ensures the `RAW` table remains current without manual effort. Forecasting follows immediately after ingestion to produce seven-day predictions that are ready for analysis. The final analytics table (`ANALYTICS.FINAL_STOCK_PRICES`) can be connected directly to BI tools such as Tableau or Power BI for visualization and reporting.

D. Limitations

- Forecasting is limited to Snowflake's built-in ML Forecast function and does not include external features.
- Data comes exclusively from yfinance, which may occasionally experience downtime or rate limits.
- Each stock is modeled independently without considering inter-stock correlations.
- The historical window is fixed to 180 days, and the forecast horizon is limited to seven days.
- Events such as stock splits, dividends, or extreme market volatility are not explicitly handled in this version.

These limitations define the current scope of the project but also suggest clear directions for future improvements, such as incorporating external data sources or experimenting with more advanced forecasting models.

IV. Dataset

Our data source is the **yfinance** Python library, which provides access to Yahoo Finance stock market data. It is free to use, easy to integrate in Python.

Stock symbols in our analysis:

- NVDA (NVIDIA Corporation)
- AAPL (Apple Inc.)
- Additional tickers can by added by updating the Airflow Variable without code changes.

Fields we extract:

- Trading date
- Opening price
- Closing price
- Daily high and low
- Trading volume

Schema mapping to Snowflake. The transformed dataset is written into the target table with the required seven columns:

Source concept	Snowflake column
Ticker	`SYMBOL`
Trading date	`DATE`
Open price	`OPEN`
Close price	`CLOSE`
Daily low	`MIN`
Daily high	`MAX`
Volume	`VOLUME`

Each pipeline run fetches the previous 180 days of trading data. This window gives us sufficient historical context for the machine learning model to generate reliable forecasts.

V. System Diagram

Our architecture has two main Airflow DAGs that run sequentially (Fig. 1):

1) **ETL Pipeline:** Downloads 180 days of stock data, cleans it, and loads into Snowflake
2) **ML Forecast Pipeline:** Waits for ETL to finish, trains the forecasting model, generates predictions, and merges them with historical data

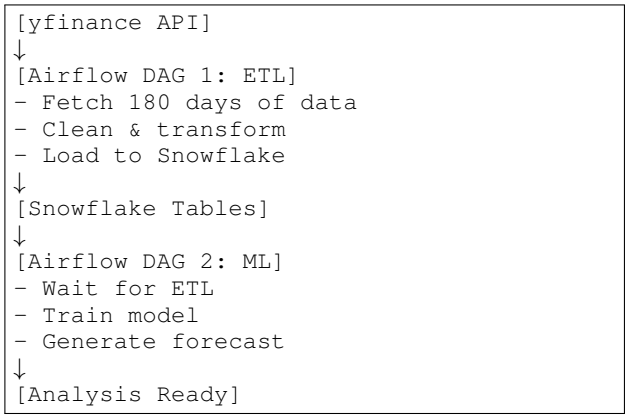

Fig. 1. System Architecture

Both DAGs execute daily. The ETL dag pulls fresh data first, then the ML pipeline waits for that to complete before running its forecasts.

VI. Tables

This project utilizes four core tables located in the Snowflake environment. These tables collectively manage historical stock data, machine learning forecasts, and final analytical outputs. Each table corresponds to a specific stage of the data pipeline and serves a distinct purpose in maintaining data flow, ensuring quality, and enabling insight generation.

A. FACT_STOCK_PRICE_DAILY

This is the primary raw data table that stores 180 days of historical stock prices for each symbol. It includes daily Open, Close, Min, Max, and Volume values. The table is created once and updated daily through the ETL pipeline defined in `etl_yfinance_stock.py`. The `MERGE` statement ensures updates without duplication.

```
CREATE OR REPLACE TABLE USER_DB_PIKACHU.LAB.
    FACT_STOCK_PRICE_DAILY (
    SYMBOL STRING NOT NULL,
    "DATE" DATE NOT NULL,
    "OPEN" FLOAT,
    "CLOSE" FLOAT,
    "MIN" FLOAT,
    "MAX" FLOAT,
    VOLUME NUMBER,
    CONSTRAINT UK_FACT_PRICE UNIQUE (SYMBOL, "DATE")
);
```

Listing 1. Fact Table Schema

Usage and Observations:

- Acts as training data for forecasting.
- Allows analysts to observe stock trends, volatility, and price behavior over time.
- Enables dashboards showing raw daily price movements per company.

—

B. FORECAST_LATEST

This table stores the most recent 7-day forecast results generated by `SNOWFLAKE.ML.FORECAST`. It is overwritten during each run to ensure that only the latest predictions are retained. Forecasts include predicted values and confidence intervals.

```
CREATE OR REPLACE TABLE USER_DB_PIKACHU.LAB.
    FORECAST_LATEST (
    SERIES STRING,
    TS TIMESTAMP_NTZ,
    FORECAST FLOAT,
    LOWER_BOUND FLOAT,
    UPPER_BOUND FLOAT
);
```

Listing 2. Forecast Results Table

Usage and Observations:

- Contains predicted stock prices with upper and lower bounds.
- Helps assess expected movement, forecast uncertainty, and market direction.
- Supports visualization in line charts or confidence interval plots.

—

C. UNION_STOCK_PRICE

The final output table combines historical prices with forecasted values into a single unified structure. It adds metadata such as `DATA_TYPE` to distinguish between actual and forecast records and aligns both to a common date format.

```
CREATE OR REPLACE TABLE USER_DB_PIKACHU.LAB.
    UNION_STOCK_PRICE AS
SELECT
    SYMBOL,
    CAST("DATE" AS DATE) AS DATE,
    CLOSE AS ACTUAL,
    NULL::FLOAT AS FORECAST,
    NULL::FLOAT AS LOWER_BOUND,
    NULL::FLOAT AS UPPER_BOUND,
    'actual' AS DATA_TYPE
FROM USER_DB_PIKACHU.LAB.FACT_STOCK_PRICE_DAILY
WHERE "DATE" BETWEEN DATEADD(DAY, -180, CURRENT_DATE
    ()) AND CURRENT_DATE()
UNION ALL
SELECT
    REPLACE(SERIES, '\"', '') AS SYMBOL,
    CAST(TS AS DATE) AS DATE,
    NULL::FLOAT AS ACTUAL,
    FORECAST,
    LOWER_BOUND,
    UPPER_BOUND,
    'forecast' AS DATA_TYPE
FROM USER_DB_PIKACHU.LAB.FORECAST_LATEST
WHERE SERIES IS NOT NULL;
```

Listing 3. Unified Table for Historical and Forecasted Data

Usage and Observations:

- Ideal for dashboards, Power BI/Tableau charts, and timeline visualizations.
- Helps compare forecast results with actual performance.
- Enables KPI monitoring such as deviation, trend overlap, and forecast accuracy.

—

D. FORECAST_METADATA

This table stores summary-level metadata about the latest union operation. It tracks how many records were processed and ensures that both actual and forecast data were properly included.

```
CREATE OR REPLACE TABLE USER_DB_PIKACHU.LAB.
    FORECAST_METADATA AS
SELECT
    CURRENT_TIMESTAMP() AS LAST_UPDATED,
    CURRENT_DATE() AS BASE_DATE,
    COUNT(CASE WHEN DATA_TYPE = 'actual' THEN 1 END)
        AS ACTUAL_RECORDS,
    COUNT(CASE WHEN DATA_TYPE = 'forecast' THEN 1
        END) AS FORECAST_RECORDS,
    MIN(DATE) AS MIN_DATE,
    MAX(DATE) AS MAX_DATE
FROM USER_DB_PIKACHU.LAB.UNION_STOCK_PRICE;
```

Listing 4. Metadata Table for Forecast Tracking

Usage and Observations:

- Used for pipeline validation, logging, and alerting.
- Helps verify if forecasts succeeded and all data types were included correctly.
- Can be extended with additional fields such as job duration, error logs, or data quality checks.

VII. Functional Analysis

This section explains the functional components of the system and how the data pipelines, database operations, and machine-learning forecasting processes interact to achieve automated stock price prediction.

A. yfinance ETL Pipeline

The ETL pipeline is implemented in `etl_yfinance_stock.py` and registered in Airflow under the DAG ID `etl_yfinance_stock`. It fetches 180 days of OHLCV stock data from the *yfinance* API and upserts it into the Snowflake table `FACT_STOCK_PRICE_DAILY`. Data transformation includes column renaming, type casting, and removal of missing values. A SQL `MERGE` statement ensures idempotent upserts, so reruns update existing rows instead of creating duplicates.

This pipeline forms the foundation for all downstream analytics and forecasting tasks.

B. ML Forecasting Pipeline

The forecasting pipeline is implemented in `ml_forecast_stock_v2.py` with DAG ID `ml_forecast_stock_v2`. It executes only after the ETL DAG finishes successfully and uses

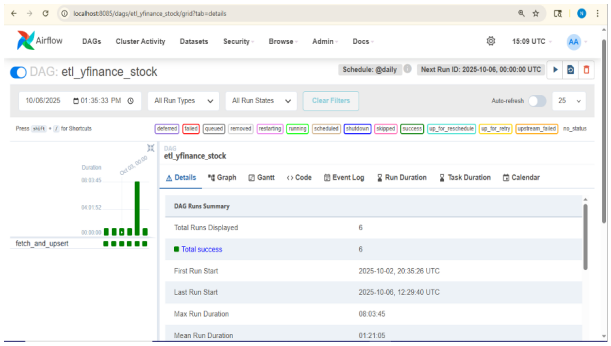

Fig. 2. Airflow ETL DAG showing successful data-ingestion run.

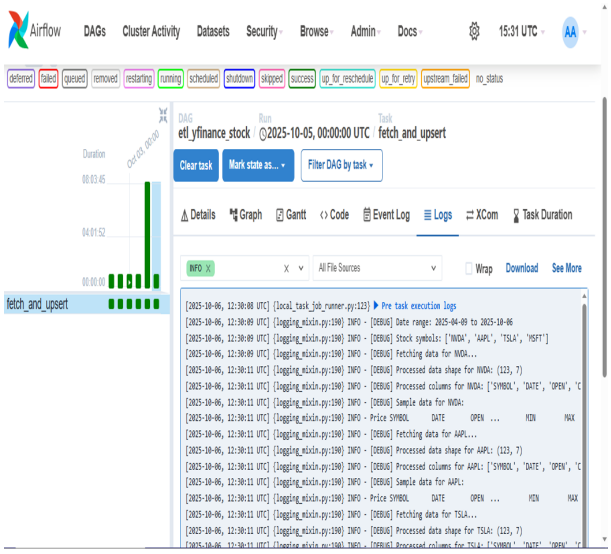

Fig. 3. Airflow task logs from `fetch_and_upsert()` confirming data written to Snowflake.

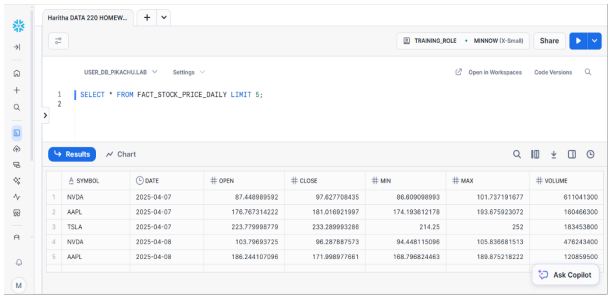

Fig. 4. Snowflake table `FACT_STOCK_PRICE_DAILY` containing 180 days of stock data.

SNOWFLAKE.ML.FORECAST to predict closing prices for the next seven days. The process consists of four major stages: (1) creating a training view, (2) building or replacing the ML model, (3) generating forecasts, and (4) merging the results with historical data to create an analytics-ready table.

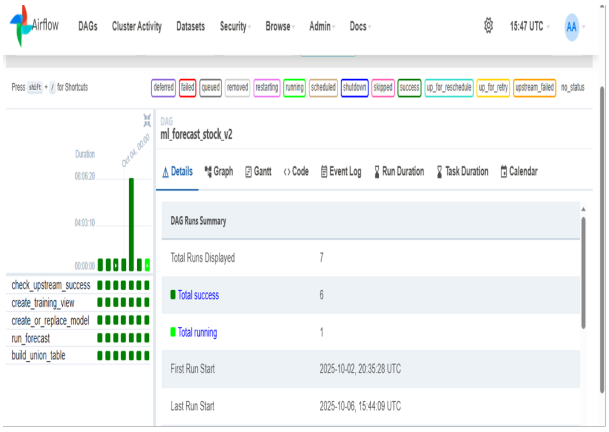

Fig. 5. Airflow UI showing both ETL and ML Forecasting DAGs.

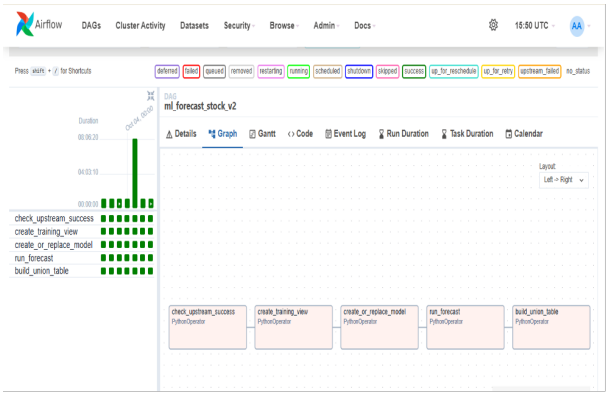

Fig. 6. Airflow task-flow graph for the ML Forecasting DAG.

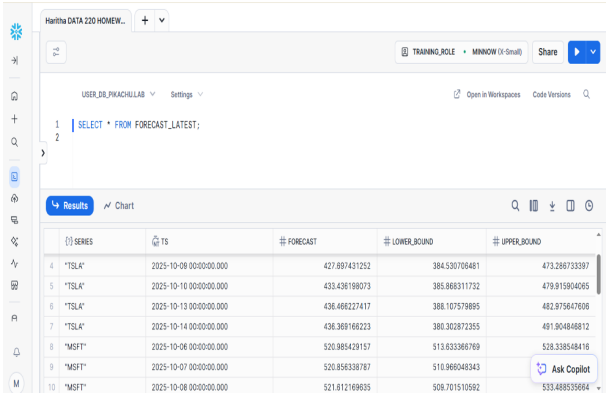

Fig. 7. Snowflake table FORECAST_LATEST storing the most recent seven-day forecast results.

The forecasting DAG produces standardized tables that store model outputs and merge them with actual data in Snowflake.

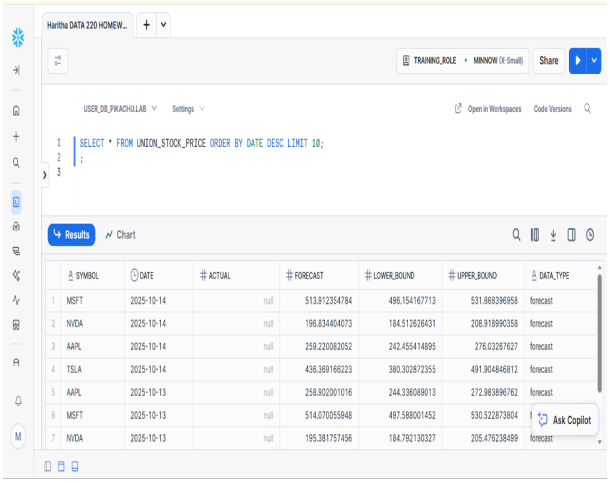

Fig. 8. Final union table UNION_STOCK_PRICE combining historical and forecasted data.

This integration provides analysts with analytics-ready datasets that can be directly queried or visualized in BI tools such as Tableau or Power BI.

C. Scheduling and Dependency Management

The ML Forecasting DAG is triggered only after the ETL DAG completes successfully. Dependency control is handled using a custom Airflow function check_upstream_success() and the ExternalTaskSensor. This ensures the forecasting process always uses the latest available data.

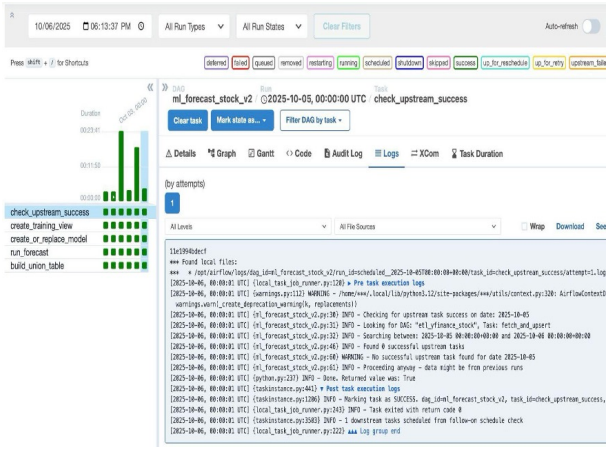

Fig. 9. Airflow log showing successful ETL–ML dependency validation.

This dependency management guarantees correct task sequencing, data consistency, and fully automated end-to-end operation of the pipeline.

VIII. Implementation & Code Listings

This section summarizes the implementation of the automated data pipeline and presents key code excerpts.

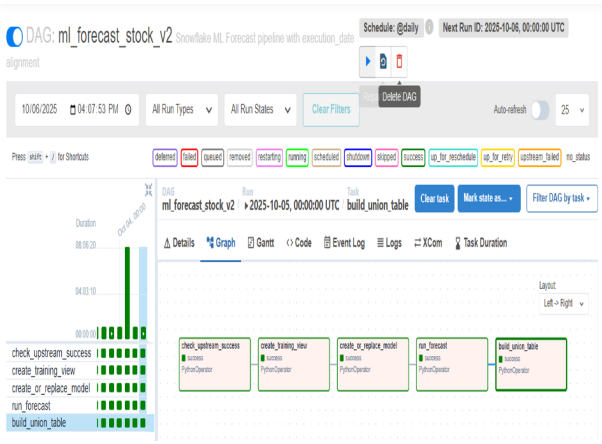

Fig. 10. Combined DAG graph view illustrating sequential execution of ETL and ML workflows.

A. ETL Implementation (Airflow DAG)

The ETL pipeline is implemented in `etl_yfinance_stock.py` under the DAG ID `etl_yfinance_stock`. This DAG automatically downloads the last 180 days of OHLCV stock data from the *yfinance* API, cleans and formats it, and loads it into the Snowflake table `FACT_STOCK_PRICE_DAILY`. The process runs daily, ensuring the data is always up to date for downstream forecasting.

The DAG consists of three main tasks:

1) **Extraction:** Pulls data for each stock symbol listed in Airflow Variables.
2) **Transformation:** Cleans missing values, converts datatypes, and standardizes columns.
3) **Loading:** Writes processed data into Snowflake using a transactional `MERGE` operation.

```
MERGE INTO fact_stock_price_daily t
USING tmp_fact_stock_price_daily s
  ON t.SYMBOL = s.SYMBOL AND t."DATE" = s."DATE"
WHEN MATCHED THEN UPDATE SET
  "OPEN"=s."OPEN", "CLOSE"=s."CLOSE",
  "MIN"=s."MIN", "MAX"=s."MAX", VOLUME=s.VOLUME
WHEN NOT MATCHED THEN INSERT
  (SYMBOL, "DATE", "OPEN", "CLOSE", "MIN", "MAX",
      VOLUME)
VALUES (s.SYMBOL, s."DATE", s."OPEN", s."CLOSE", s."
    MIN", s."MAX", s.VOLUME);
```

Listing 5. Transactional MERGE statement ensuring idempotent data loads

This ensures the pipeline is idempotent: rerunning the DAG will update existing records instead of creating duplicates. All SQL operations are wrapped in `BEGIN/COMMIT/ROLLBACK` transactions to guarantee data integrity.

Overall, this ETL process establishes a robust and reproducible data foundation for forecasting. By leveraging Airflow for orchestration and Snowflake for data storage, the system ensures reliability, scalability, and maintainability across daily runs.

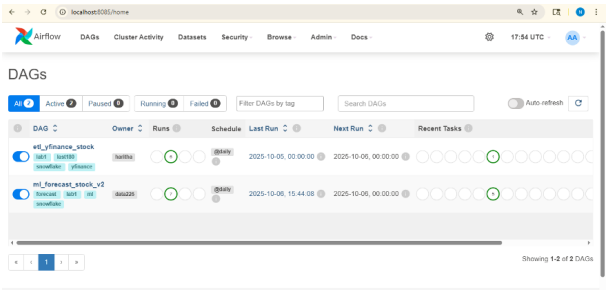

Fig. 11. Airflow overview showing the ETL DAG for yfinance and the ML forecasting DAG.

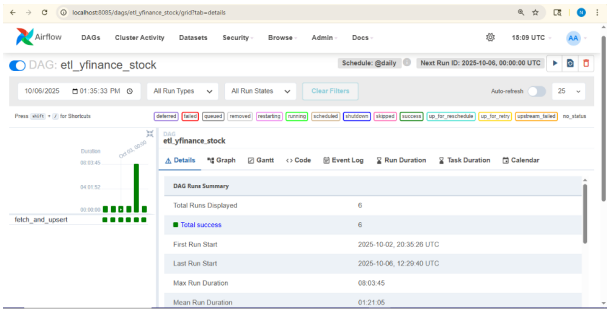

Fig. 12. Airflow DAG `etl_yfinance_stock` with successful daily execution.

B. Forecasting Implementation (ML DAG)

The forecasting pipeline is implemented in `ml_forecast_stock_v2.py` with DAG ID `ml_forecast_stock_v2`. This DAG runs only after the ETL pipeline completes successfully and automates model training, forecasting, and result consolidation inside Snowflake. It leverages the `SNOWFLAKE.ML.FORECAST` function to predict the next seven days of closing prices for each stock symbol.

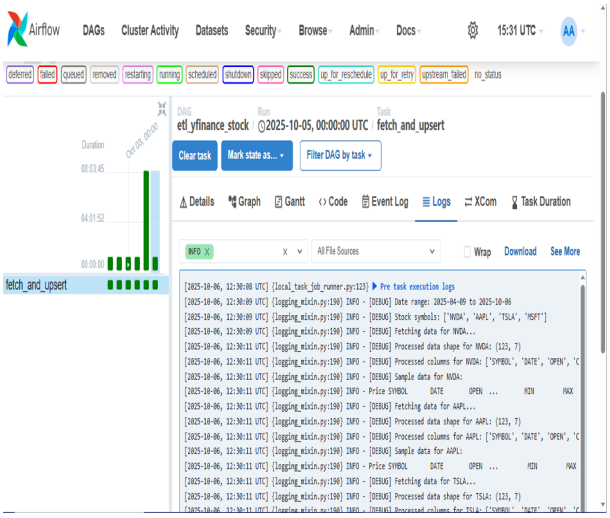

Fig. 13. Airflow task log for `fetch_and_upsert()` confirming successful data ingestion.

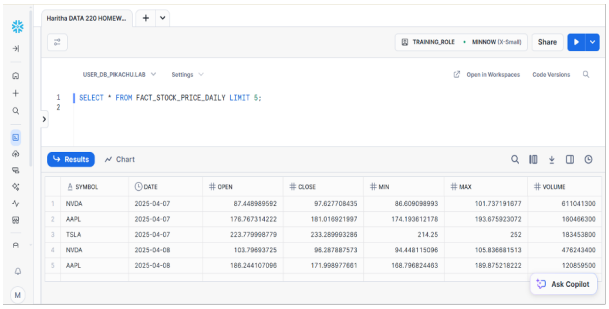

Fig. 14. Snowflake table `FACT_STOCK_PRICE_DAILY` containing 180 days of historical data.

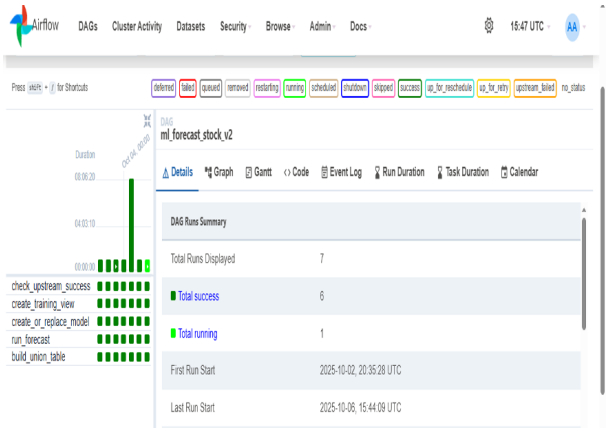

Fig. 15. Airflow UI showing successful execution of the ML Forecasting DAG.

The DAG consists of five key tasks:

1) **Check Upstream Success:** Confirms that the ETL DAG finished successfully before forecasting begins.
2) **Create Training View:** Builds a temporary view containing the latest 180 days of closing prices for each stock symbol.
3) **Create or Replace Model:** Defines and trains the `SNOWFLAKE.ML.FORECAST` model on the prepared data.
4) **Run Forecast:** Executes the model to produce seven-day predictions with upper and lower confidence bounds.
5) **Build Union Table:** Merges actual and forecasted data into a unified table for downstream analytics.

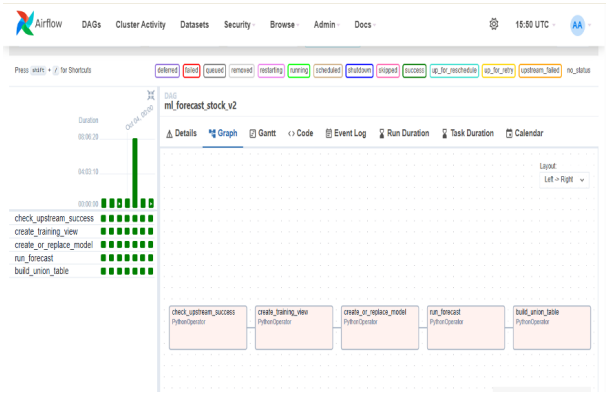

Fig. 16. Task flow view of the ML Forecasting DAG in Airflow.

The following figures illustrate log outputs and Snowflake results from each stage of the ML pipeline.

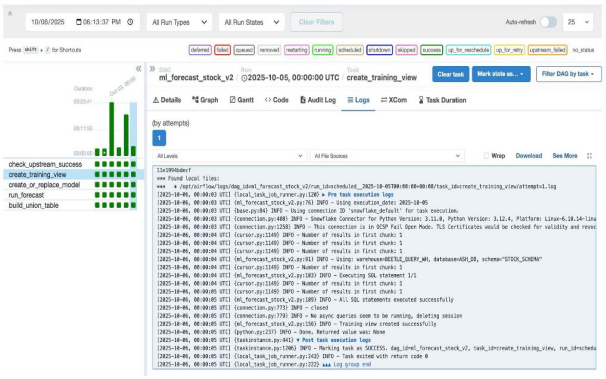

Fig. 17. Airflow log for the `create_training_view` task, confirming successful view creation.

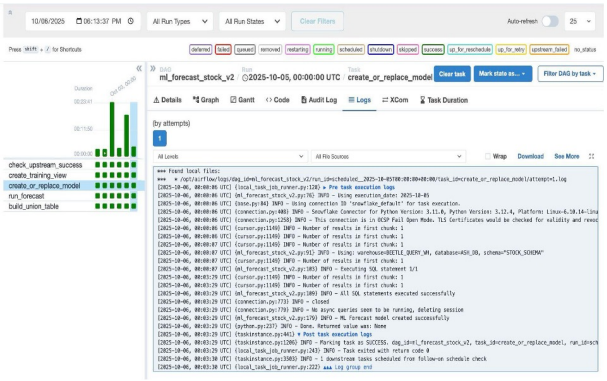

Fig. 18. Model creation log showing the successful definition of `SNOWFLAKE.ML.FORECAST`.

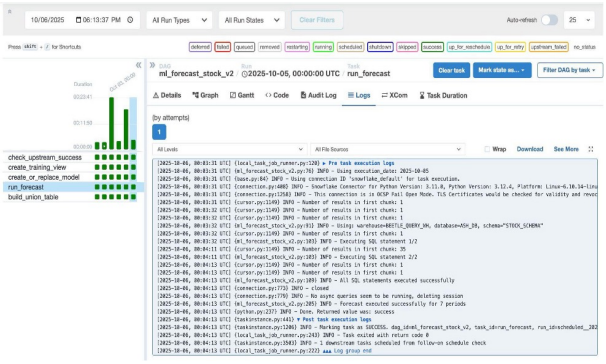

Fig. 19. Airflow log for the `run_forecast` task, generating 7-day predictions.

This forecasting pipeline demonstrates a fully automated, end-to-end process that integrates machine learning directly within the data warehouse. By using Airflow for orchestration and

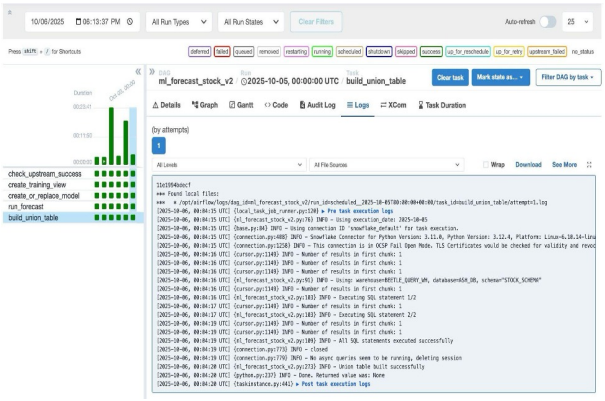

Fig. 20. Airflow log confirming successful creation of the final union table.

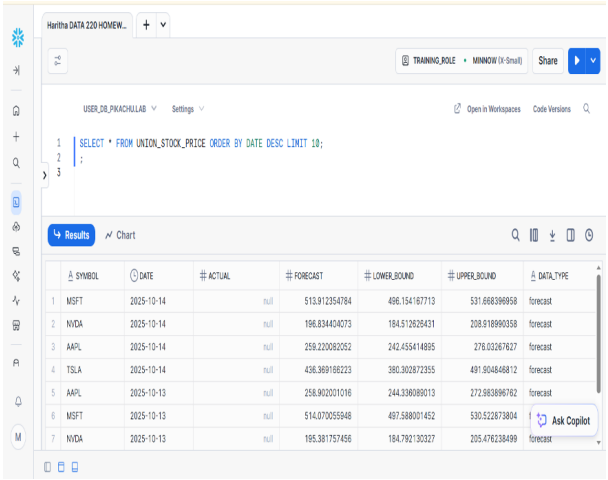

Fig. 21. Snowflake table `UNION_STOCK_PRICE` combining actual historical and forecasted values.

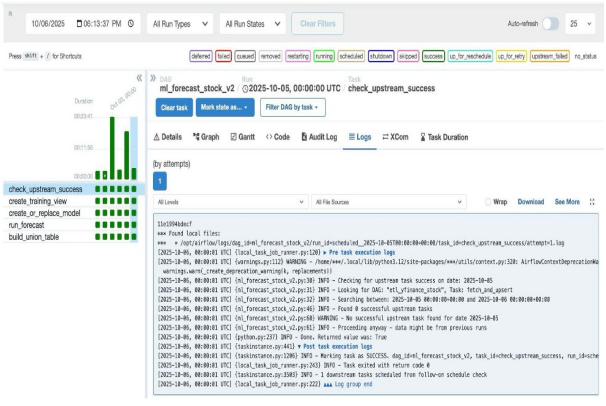

Fig. 22. Upstream success check log ensuring proper ETL-to-ML task dependency.

Snowflake for computation, the system achieves high scalability, reproducibility, and minimal maintenance overhead.

C. Scheduling and Robustness

DAG dependencies are enforced through `ExternalTaskSensor`. Each SQL block runs within `BEGIN/COMMIT/ROLLBACK`, with limited retries to ensure idempotency.

D. Airflow Variables and Connections

Airflow's **Variables** and **Connections** are used to make the pipeline both flexible and secure. Variables store runtime parameters such as stock symbols, warehouse, and schema names, allowing easy updates without changing code.

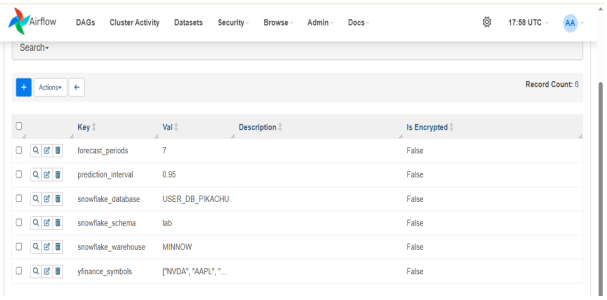

Fig. 23. Airflow Variables configuring stock symbols and Snowflake settings.

Connections securely manage credentials for Snowflake and other external systems. The DAGs reference these credentials through the `snowflake_default` connection ID.

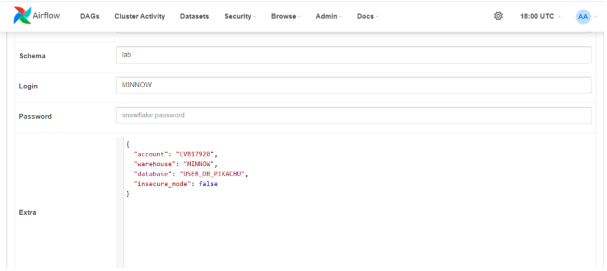

Fig. 24. Airflow Connection setup for Snowflake credentials.

This setup simplifies configuration management and ensures sensitive information remains protected.

IX. Lessons Learned

A. What Worked Well

- Using the `MERGE` command in Snowflake turned out to be very effective — it automatically handled duplicates whenever the pipeline was re-run.
- Airflow Variables made the pipeline flexible; adding or changing stock symbols no longer required modifying the code itself.
- After some initial trial and error, Snowflake's built-in ML forecasting became straightforward once we understood the SQL structure.

- Keeping the workflow in two separate DAGs — one for ETL and one for ML — made the logic cleaner and easier to troubleshoot.

B. Challenges

- Data quality from *yfinance* was occasionally inconsistent, especially around market holidays or missing trading days.
- Learning Snowflake's ML syntax required experimentation since the documentation examples did not always match our project.
- Coordinating the two DAGs correctly required careful scheduling and dependency management.
- Handling errors gracefully was another challenge — a single failed stock symbol could interrupt the entire pipeline if not managed properly.

X. Future Work

A. Short-Term Enhancements

- Integrate additional technical indicators such as RSI, MACD, and Bollinger Bands to enhance the analytical depth of the model.
- Implement automated alert mechanisms to notify users of abnormal or unexpected price movements.
- Introduce stronger data validation and quality checks during ingestion to detect issues earlier rather than after processing.
- Extend system capability to handle a broader range of stock symbols, including those from international exchanges.

B. Long-Term Extensions

- Experiment with alternative forecasting approaches such as ARIMA or Prophet to compare performance with Snowflake's built-in model.
- Incorporate external data sources, such as financial news sentiment or social media data, to improve predictive accuracy.
- Develop a web-based dashboard for intuitive visualization and interactive exploration of stock trends and forecasts.
- Explore portfolio optimization techniques to expand the system beyond single-stock forecasting and into investment strategy analysis.

XI. Conclusion

The implementation of this end-to-end data pipeline highlights the significance of automating data ingestion, transformation, and predictive analytics. The integration of Airflow with Snowflake and yfinance enabled reliable scheduling, extraction, and forecasting for multiple stock symbols. Key takeaways include the use of the SnowflakeHook for seamless SQL execution, Airflow's ability to manage DAG dependencies, and the forecasting capabilities of SNOWFLAKE.ML.FORECAST. The use of SQL transactions and exception handling within Airflow ensures robustness and data integrity. Challenges encountered included time synchronization across DAGs, handling missing dates in financial data, and debugging Airflow Docker setup. These were addressed through retry logic, precise data windowing, and controlled environment deployment using Docker Compose.

In the future, this system can be extended to incorporate additional data sources, integrate more advanced machine learning models, and support real-time forecasting for broader financial analytics applications.

XII. GitHub Repository

Project code and documentation available at:

Repository URL: https://github.com/Ashley96yy/Data226-Stock-Price-Prediction-Analytics

The repository includes:

- Airflow DAG implementations for ETL and ML pipelines
- SQL scripts for table creation and schema setup
- Docker configuration and deployment files
- Setup instructions and environment requirements
- Sample Airflow variables and connection configurations
- Documentation

References

[1] Apache Software Foundation, "Airflow Documentation." [Online]. Available: https://airflow.apache.org/

[2] Snowflake Inc., "Snowflake Documentation: Forecasting." [Online]. Available: https://docs.snowflake.com/en/user-guide/ml-powered-forecasting

[3] Yahoo Finance, "yfinance Python Library." [Online]. Available: https://pypi.org/project/yfinance/

[4] Docker Inc., "Docker Desktop Installation Guide." [Online]. Available: https://docs.docker.com/get-dock/

[5] Keeyong Han, "SJSU Data226 GitHub Repository." [Online]. Available: https://github.com/keeyong/sjsu-data226-FA25/blob/main/docker-compose.yaml