

Stock Price Forecasting using Snowflake and Airflow

Dhruvkumar Patel, Drashti Shah

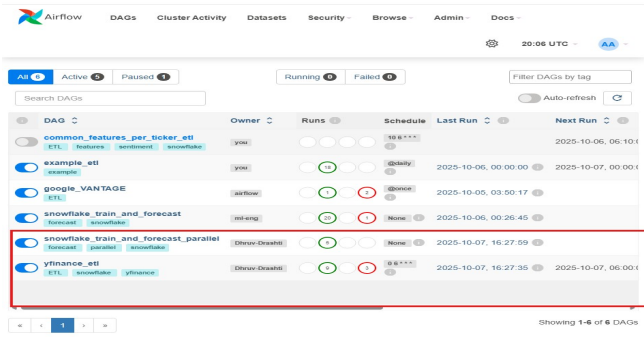
Abstract — This paper presents an automated end-to-end data pipeline for extracting, transforming, and loading (ETL) financial market data into Snowflake, followed by model training and forecasting using Snowflake's native machine learning capabilities. The implementation leverages Apache Airflow for orchestration, yFinance for data extraction, and Snowflake's ML.FORECAST for time series modeling. The architecture ensures data integrity, modularity, scalability, and parallelized forecasting for multiple stocks.

Index Terms — Airflow, ETL, yFinance, Snowflake, Time Series Forecasting, Parallel Processing, Machine Learning, Automation.

I. PROBLEM STATEMENT, REQUIREMENTS AND SPECIFICATIONS

The financial data analysis often involves the acquisition of large volumes of stock market data from public APIs, transformation into structured formats, and ingestion into analytical systems. Manual approaches are inefficient and error-prone. To address this, we propose a fully automated Airflow-based pipeline that integrates data ingestion from yFinance, ETL into Snowflake, and automated training and forecasting for multiple stock symbols.

The proposed system operates daily, downloading daily OHLCV (Open, High, Low, Close, Volume) data for a configurable list of stock symbols, storing them in Snowflake per-stock symbol tables, and triggering a secondary DAG that builds forecasting models and generates predictive data.



Dhruvkumar Patel is with Department of Applied Data Science, San Jose State University, California (dhruvkumarkamleshbhai.patel@sjsu.edu)

Drashti Shah is with Department of Applied Data Science, San Jose State University, California (dhrashti.shah@sjsu.edu)

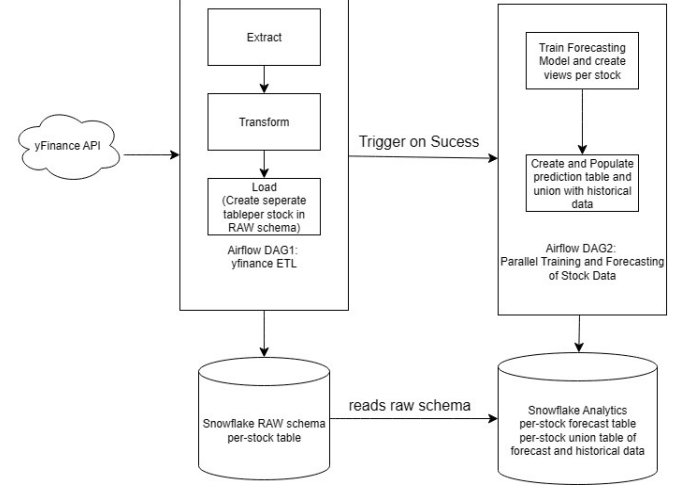


Fig. 1. System Architecture for ETL and Stock price prediction using Airflow and snowflake databases. Data managed by using separate schemas for dumping and forecasting of stock data.

II. SYSTEM ARCHITECTURE

The proposed system consists of two Apache Airflow Directed Acyclic Graphs (DAGs) and a Snowflake data warehouse organized into two schemas.

A. DAG 1 – yfinance_etl

ResolveConfiguration: This initial task gathers the operational parameters from Airflow's global variable store, including the list of stock symbols, the look-back duration, and the destination schema name. Centralizing these parameters guarantees that the workflow remains dynamic and configurable without code changes. The resolved configuration defines the temporal scope and symbol set for all downstream activities and is passed directly to the extraction stage.

Extract: This task retrieves daily OHLCV (Open, High, Low, Close, Volume) stock prices from the yFinance API for each symbol specified in the configuration. The extraction process ensures network reliability, data completeness, and consistency in the returned columns. Any missing or malformed data is logged for auditability. The normalized market records form the raw dataset consumed by the

transformation phase.

Transform: The transformation stage cleans and harmonizes the extracted data into a standardized structure compatible with the Snowflake warehouse. Data types are cast to their correct numeric formats, date fields are converted to ISO-compliant strings, and column names are aligned to the target schema conventions. This ensures schema compliance and numerical accuracy before insertion. As a result, a sequence of structured rows is then forwarded to the loading step.

Load: This is the core persistence task. It first guarantees that both the target schema and each per-stock symbol table exist, creating them if required to maintain independence from manual database setup. The second phase executes a transactional load: it inserts records into a temporary staging table, performs a MERGE operation into the main table, and commits the transaction. If any anomaly arises, the task rolls back changes to prevent partial updates. This ensures idempotency and integrity across multiple runs. Upon successful completion, it provides a status count to the triggering operator.

Trigger Forecasting DAG: Once all ETL tasks succeed, this operator programmatically triggers the second DAG responsible for model training and forecasting. It waits for a positive completion state before proceeding, ensuring that forecasts are always trained on the most up-to-date data. The orchestration link between DAGs enforces dependency ordering and isolates workflow responsibilities.

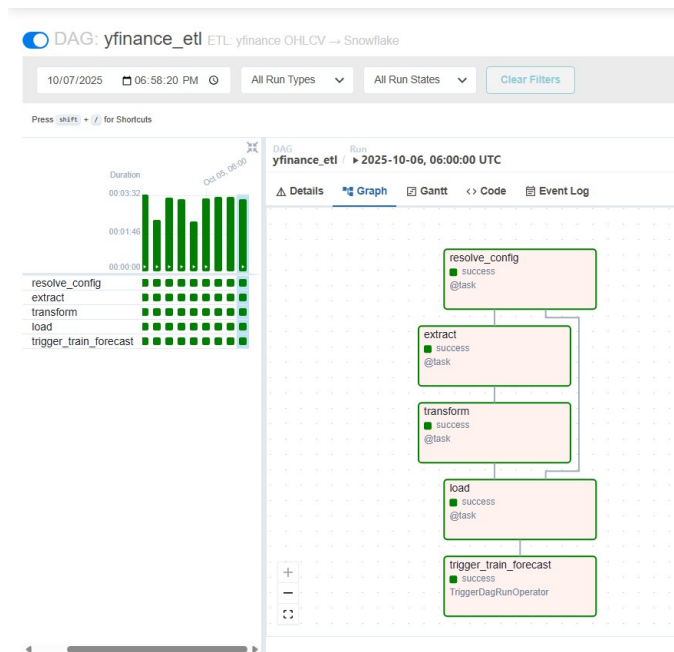


Fig. 2. Airflow ETL Data Pipeline for loading time series stock data into raw schema of database.

B. DAG 2 – snowflake_train_and_forecast_parallel

Resolve Configuration: This task reconstructs the runtime settings for the analytical stage, defining which schemas to read and write, the training and forecasting windows, the prediction interval, and the list of symbols. Its purpose is to make the forecasting process reproducible and environment-independent. The resolved parameters guide both the training and forecasting fan-out stages.

Fan-out for Training: This preparatory task expands the configuration into multiple per-symbol units of work. Each symbol receives an independent configuration dictionary that allows Airflow to spawn parallel tasks for model training. The significance of this design lies in scalability: multiple models can be trained simultaneously without resource contention, subject to the pool limits configured in Airflow. The output is a collection of symbol-specific instructions consumed by the training stage.

Fan-out for Forecasting: Similar to the previous fan-out, this task generates symbol-specific parameters for the forecasting stage, ensuring exact alignment between training and forecasting operations. It preserves ordering so that each forecasting task depends on its corresponding trained model. This enables Airflow’s dynamic task-mapping engine to coordinate “train-then-forecast” sequences per stock symbol.

Train-One (Per-Symbol Model Training): Each mapped training task executes independently within the Snowflake environment. It builds or refreshes a view that selects recent close prices and then trains a new time-series forecasting model using Snowflake’s native machine-learning procedure. Running inside an explicit SQL transaction guarantees atomicity—if the model creation fails, the operation is rolled back. The result is an updated forecasting model along with metadata about the training set size, which signals readiness for the next forecasting task.

Forecast-One (Per-Symbol Inference and Table Union): For each symbol, this task generates predictions using the trained model. It begins by cleaning previous forecasts for dates equal to or after the current day, thereby maintaining idempotency. Next, it calls the Snowflake forecasting function to produce predicted close prices and confidence bounds for the defined horizon. These predictions are inserted into a forecast table, and finally, a new **final table** is created by **unioning** recent historical data from the RAW schema with the newly generated forecast data. The transaction is committed only after all statements succeed. This produces a single, authoritative dataset combining both actual and predicted values.

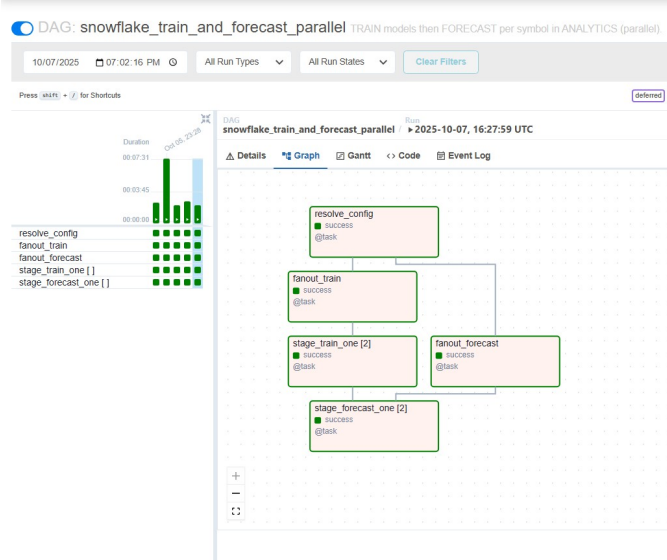


Fig. 3. Execution graph for the `snowflake_train_and_forecast_parallel` pipeline. The workflow resolves configuration, fans out perstock, trains a Snowflake ML model per stock, and then forecasts, with each `stage_train_one[i]` preceding its corresponding `stage_forecast_one[i]`.

III. PARALLELIZATION AND CONCURRENCY CONTROL

The training–inference workload is **embarrassingly parallel** across symbols, so the pipeline decomposes the problem into independent, per-symbol tasks using Airflow’s **dynamic task mapping**. At runtime, a single configuration is expanded into a set of identically shaped work items—one per stock symbol—so the scheduler can execute many homogeneous tasks concurrently. Dependency edges enforce **per-symbol ordering** (i.e., each forecast strictly waits for its corresponding train), while allowing different symbols to progress in parallel. This approach yields near-linear scalability with respect to the number of available worker slots and warehouse capacity, shortens wall-clock time compared to a sequential baseline, and improves fault isolation because a failure for one symbol neither blocks nor corrupts the others.

Concurrency is governed by an Airflow **Pool** that caps the number of simultaneous Snowflake sessions, providing **back-pressure** so database resources are not saturated. Each mapped task executes within a **transaction** to preserve atomicity (commit on success, rollback on error), and tasks are designed to be **idempotent** (e.g., today-and-forward forecasts are replaced), enabling safe retries without duplication. Resource isolation is reinforced by per-symbol tables and views, minimizing lock contention; observability is maintained via structured logs at the task level, enabling targeted reruns for only the affected symbols. Together, dynamic mapping plus pool-based throttling deliver a controlled, cost-aware parallel execution model that preserves correctness while maximizing throughput.

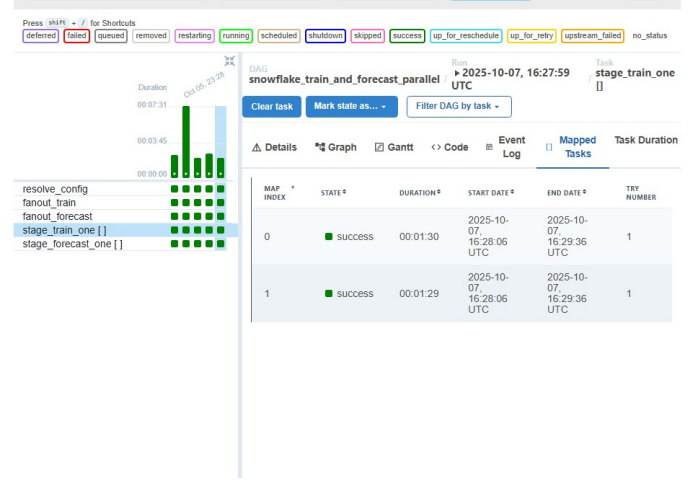


Fig. 4. Execution view of the `snowflake_train_and_forecast_parallel` DAG in Apache Airflow. The figure shows successful completion of dynamically mapped tasks (`stage_train_one`) for two symbols, each running in parallel under the Snowflake resource pool. Task states, durations, and start/end times are displayed, confirming parallel model training and orchestration consistency.

IV. AIRFLOW CONFIGURATION AND SNOWFLAKE CONNECTIONS

This section describes the configuration setup of Apache Airflow and its integration with the Snowflake data warehouse used in the proposed ETL and forecasting system.

A. Airflow Variables Configuration

Airflow Variables are used to store global parameters required by the DAGs.

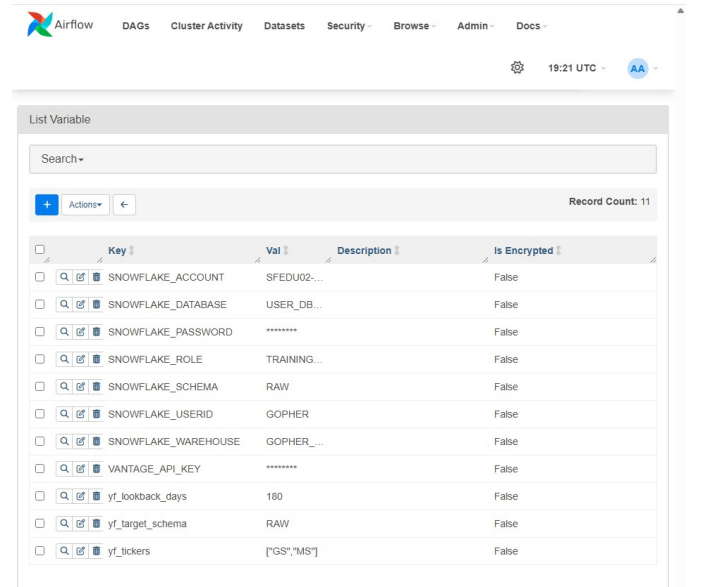
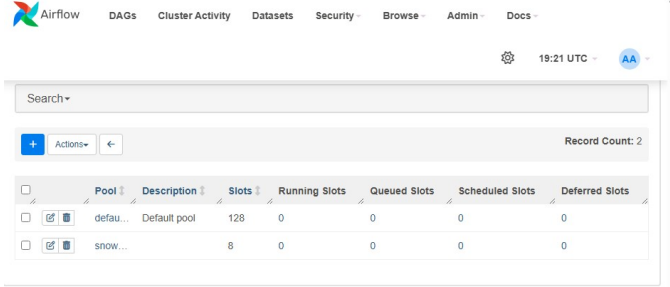


Fig. 5 illustrates the List Variable view in the Airflow UI, where the Snowflake credentials and ETL parameters are maintained. The variables include connection details such as `SNOWFLAKE_ACCOUNT`, `SNOWFLAKE_DATABASE`, and `SNOWFLAKE_ROLE`, as well as operational settings like `yf_lookback_days`, `yf_target_schema`, and `yf_tickers`.

B. Airflow Pool Configuration

Airflow Pools are configured to control task concurrency and manage resource allocation. As shown in Fig. 5, two pools are defined: the default pool with 128 slots and the snowflake_pool with 8 slots. The latter is specifically used to throttle parallel training and forecasting tasks within the Snowflake environment, preventing resource contention.



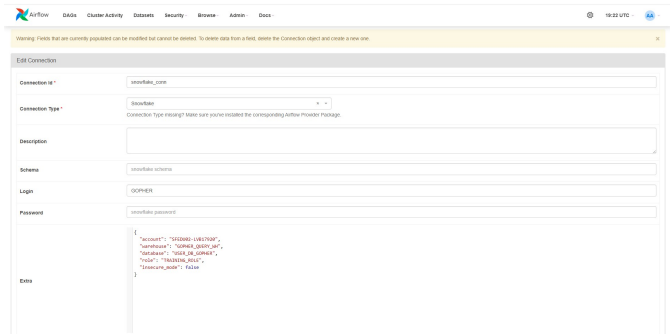
Pool	Description	Slots	Running Slots	Queued Slots	Scheduled Slots	Deferred Slots
default	Default pool	128	0	0	0	0
snow...		8	0	0	0	0

Fig. 5. Airflow Pools configuration controlling task concurrency for Snowflake-based operations.

C. Snowflake Connection Setup

Airflow connects to Snowflake through a defined Connection object.

Fig. 6 displays the snowflake_conn configuration, which specifies connection parameters such as the Snowflake account, warehouse, database, role, and user credentials. These are securely stored and retrieved by Airflow tasks during runtime to execute SQL, DDL, and ML operations in Snowflake.



```

{
  "account": "snowflake-account",
  "warehouse": "snowflake-warehouse",
  "database": "snowflake-database",
  "role": "snowflake-role",
  "schema": "snowflake-schema",
  "login": "snowflake-user",
  "password": "snowflake-password"
}

```

Fig. 6. Airflow Snowflake connection definition (snowflake_conn) specifying account, warehouse, and schema parameters.

V. DATABASE TABLE STRUCTURE AND SCHEMA DESIGN

The Snowflake data warehouse is divided into two logical schemas: **RAW** and **ANALYTICS**. Each pipeline execution creates or updates schema objects automatically based on the input tickers. The **RAW** schema stores historical OHLCV data, while the **ANALYTICS** schema contains training views, forecast outputs, and final merged datasets.

RAW Schema

In the **RAW** schema, a dedicated table is created for each stock symbol (e.g., AAPL, GOOG, MSFT) whenever the ETL pipeline runs. These tables contain daily historical market data.

Table – RAW.<STOCK_SYMBOL> (e.g., RAW.AAPL)

Field Name	Data Type	Constraint	Description
SYMBOL	VARCHAR	PRIMARY KEY (with DT)	Stock ticker symbol
DT	DATE	PRIMARY KEY (with SYMBOL)	Trading date
OPEN	FLOAT	—	Opening price
HIGH	FLOAT	—	Highest price of the day
LOW	FLOAT	—	Lowest price of the day
CLOSE	FLOAT	—	Closing price
VOLUME	NUMBER(38,0)	—	Daily trading volume

Fig. 7. Table structure of the RAW schema showing per-symbol OHLCV fields and primary key constraint.

ANALYTICS Schema -Forecast Tables

After model training, forecast tables are generated per symbol within the **ANALYTICS** schema. Each forecast table stores the predicted price and confidence intervals for upcoming dates.

Table – ANALYTICS.<SYMBOL>_FORECAST (e.g., ANALYTICS.AAPL_FORECAST)

Field Name	Data Type	Constraint	Description
SYMBOL	VARCHAR	—	Stock ticker symbol
TS	TIMESTAMP_NTZ(9)	—	Timestamp of input used for prediction
CLOSE_PRED	FLOAT	—	Predicted closing price
LOWER_BOUND	FLOAT	—	Lower bound of confidence interval
UPPER_BOUND	FLOAT	—	Upper bound of confidence interval
PREDICTION_FOR	DATE	—	Date for which prediction is made
CREATED_AT	TIMESTAMP_NTZ(9)	DEFAULT CURRENT_TIMESTAMP	Record creation timestamp

Fig. 8. Forecast table under ANALYTICS schema storing predicted closing prices and confidence intervals.

ANALYTICS Schema -Training Views

For each stock, a training view is created to feed the Snowflake ML forecasting model. The view references recent records from the RAW schema filtered by the lookback period.

Field Name	Data Type	Source	Description
TS	TIMESTAMP_NTZ	Derived from DT	Timestamp version of trading date
CLOSE	FLOAT	From RAW.<SYMBOL>	Closing price used for model training

Condition: Includes records where DT >= CURRENT_DATE() - LOOKBACK_DAYS and CLOSE IS NOT NULL.

Fig. 9. Analytical training view selecting recent close prices for model input.

ANALYTICS Schema -Final Table

After forecasting, both historical and predicted data are merged into a unified FINAL table per symbol. This table simplifies visualization and downstream analytics.

Field Name	Data Type	Constraint	Description
SYMBOL	VARCHAR	—	Stock ticker symbol
DT	DATE	—	Date (historical or forecasted)
OPEN	FLOAT	—	Opening price (from RAW)
HIGH	FLOAT	—	Highest price
LOW	FLOAT	—	Lowest price
CLOSE	FLOAT	—	Closing price
VOLUME	NUMBER(38,0)	—	Trading volume
SOURCE	VARCHAR(8)	—	Indicates data source (RAW/FORECAST)
CLOSE_FORECAST	FLOAT	—	Predicted closing price
OPEN_BOUND_FORECAST	FLOAT	—	Upper prediction bound
LOWERBOUND_FORECAST	FLOAT	—	Lower prediction bound
CREATED_AT	TIMESTAMP_LTZ(9)	—	Record creation timestamp

Fig. 10. Final combined table integrating historical and predicted data for analysis.

VI. OUTPUT AND IMPLEMENTATION REPOSITORY

The final output of the proposed system includes automated ETL ingestion, model training, and forecasting workflows, all orchestrated via Apache Airflow and executed within Snowflake.

Upon successful pipeline execution:

- The **RAW schema** is populated with per-symbol daily OHLCV records (e.g., RAW.AAPL, RAW.GS).
- The **ANALYTICS schema** generates:
 - Forecast tables such as AAPL_FORECAST containing predicted prices and confidence intervals.
 - Training views (e.g., V_TRAIN_AAPL) derived from recent historical data for model retraining.
 - Final combined tables (e.g., AAPL_FINAL) that merge historical and forecasted records for visualization and downstream analytics.
- The **Airflow UI** displays DAG execution status and task-level mappings, confirming concurrent training and forecasting across multiple tickers (as shown in Fig. 3).

The complete implementation, including the Airflow DAGs (yfinance_etl and snowflake_train_and_forecast_parallel), helper modules, and SQL scripts for Snowflake schema management, is available in the project's public GitHub repository:

https://github.com/pateldhruv1672/DATA_226_LAB1

REFERENCES

- [1] yfinance, <https://pypi.org/project/yfinance/>
- [2] Apache Airflow, <https://airflow.apache.org/>
- [3] Snowflake Machine Learning, <https://docs.snowflake.com/en/user-guide/ml>
- [4] pandas, <https://pandas.pydata.org/>