Lab 1: Stock Price Prediction Analytics using Snowflake & Airflow

Abhinita Sanabada (018320874)

October 3, 2025

Abstract

We implemented a secure, reproducible stock analytics workflow using Airflow, `yfinance`, and Snowflake. An ETL DAG ingests OHLCV data into `RAW.STOCK_PRICES`. A second DAG trains a Snowflake-native `SNOWFLAKE.ML.FORECAST` model and writes predictions to `MODEL.FORECASTS`. A final table, `ANALYTICS.FINAL_PRICES_FORECAST`, unions actuals and forecasts for downstream visualization. All Snowflake credentials (account, user, password, role, warehouse, database) are stored only in Airflow Connections, and pipeline parameters are managed via Airflow Variables.

1 Problem Statement

Build an end-to-end analytics pipeline that:

1. Extracts daily OHLCV for selected tickers via `yfinance`.
2. Forecasts daily close prices using Snowflake's built-in ML forecasting.
3. Unifies actuals and forecasts in a single analytics table.
4. Uses Airflow for orchestration, SQL/Python transactions for correctness, and Airflow Connections/Variables for secure configuration.
5. Produces reproducible runs, screenshots, and a public code repository.

Success criteria: both DAGs succeed; RAW, MODEL, ANALYTICS populated; final table supports plotting Actual vs. Forecast; screenshots and repo links are provided.

2 System Architecture

Overview

We use three schemas inside `USER_DB_CATFISH`: **RAW** (ingest), **MODEL** (predictions), **ANALYTICS** (consumption). Two DAGs orchestrate the flow:

- **DAG #1 `yfinance_etl`**: downloads OHLCV for `stock_symbols`, MERGEs into `RAW.STOCK_PRIC`

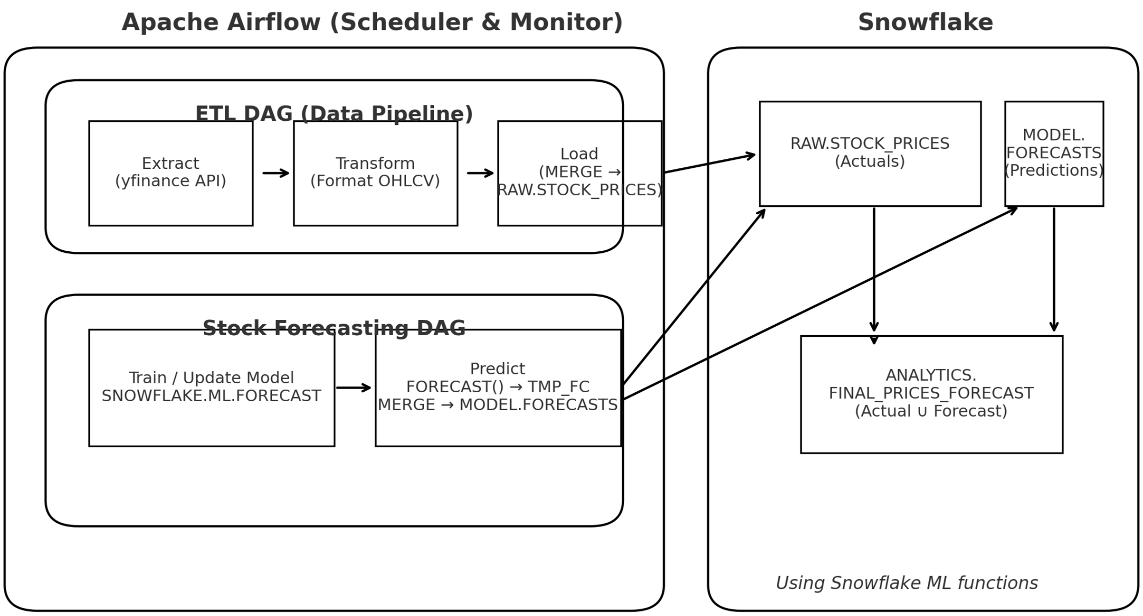

Figure 1: Archtecture diagram

- **DAG #2 `ml_forecast`**: trains/updates `SNOWFLAKE.ML.FORECAST` on multi-series history; writes predictions to `MODEL.FORECASTS`; *unions* actuals + forecasts into `ANALYTICS.FINAL_PRI`

Architecture Diagram

Screenshots (to be included)

- **Figure 2**: Airflow DAGs list showing both DAGs present.
- **Figure 3**: `yfinance_etl` Grid/Graph view with successful run.
- **Figure 4**: `ml_forecast` Grid/Graph view with successful run.

3 Data Model

All objects live in database `USER_DB_CATFISH` (warehouse: `CATFISH_QUERY_WH`, both configured via Airflow Connection). Schemas: `RAW`, `MODEL`, `ANALYTICS`.

RAW.STOCK_PRICES

`SYMBOL STRING NOT NULL; TS TIMESTAMP_NTZ NOT NULL; OPEN FLOAT; HIGH FLOAT; LOW FLOAT; CLOSE FLOAT; ADJ_CLOSE FLOAT; VOLUME NUMBER(38,0); LOAD_TS TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP();`
Primary key: (`SYMBOL`, `TS`).

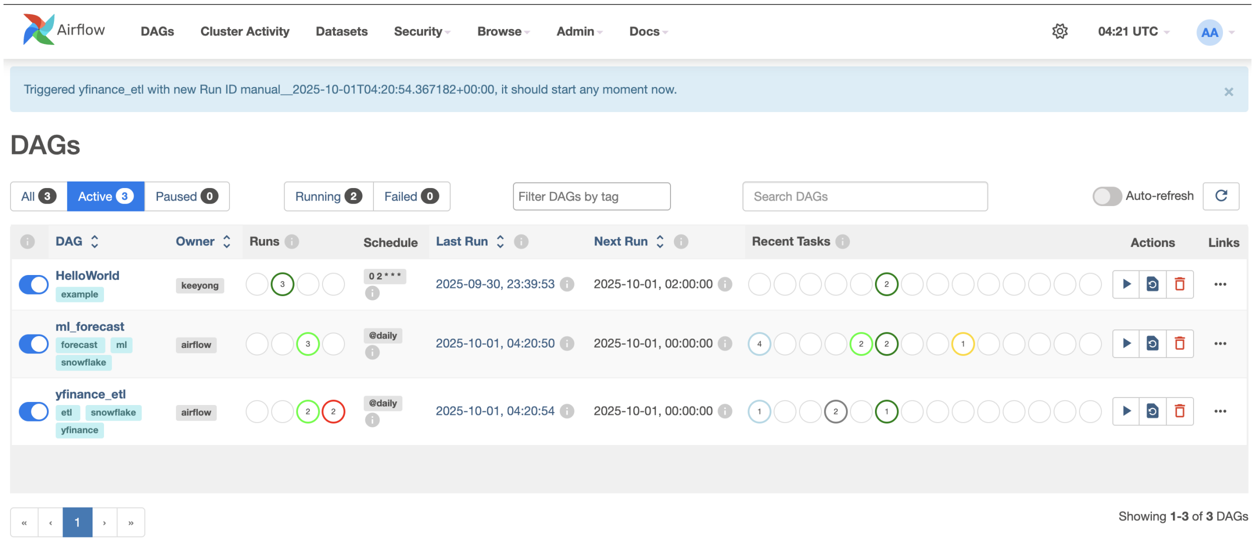

Figure 2: Airflow dags screen shot

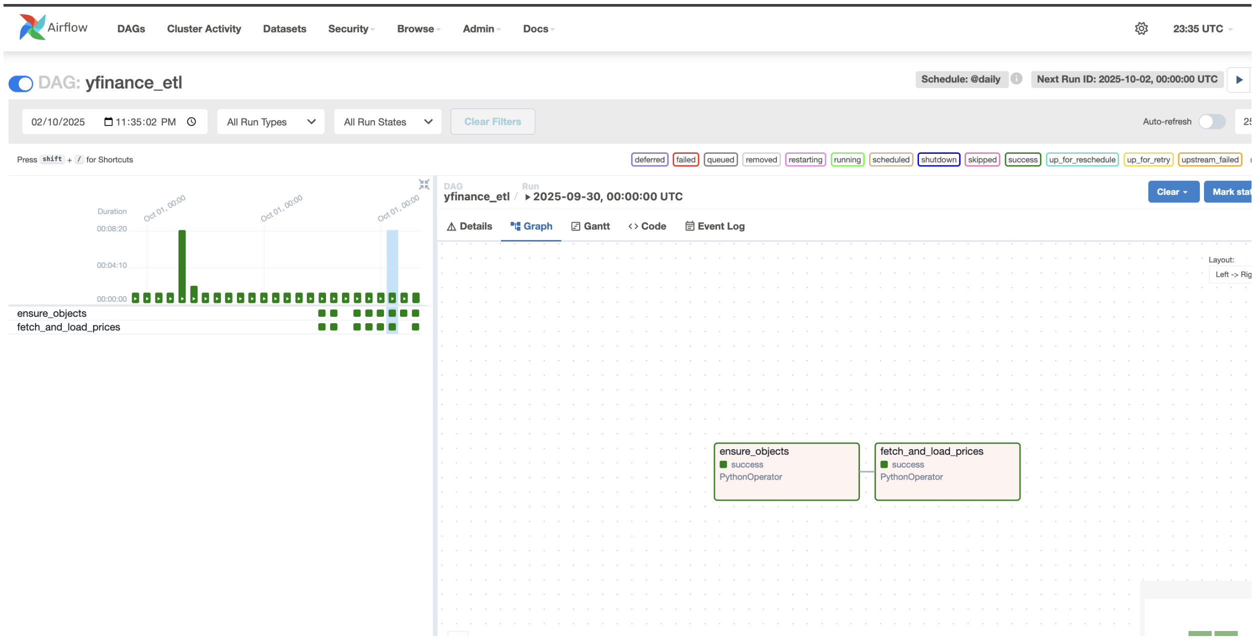

Figure 3: yfinance etl Grid/Graph

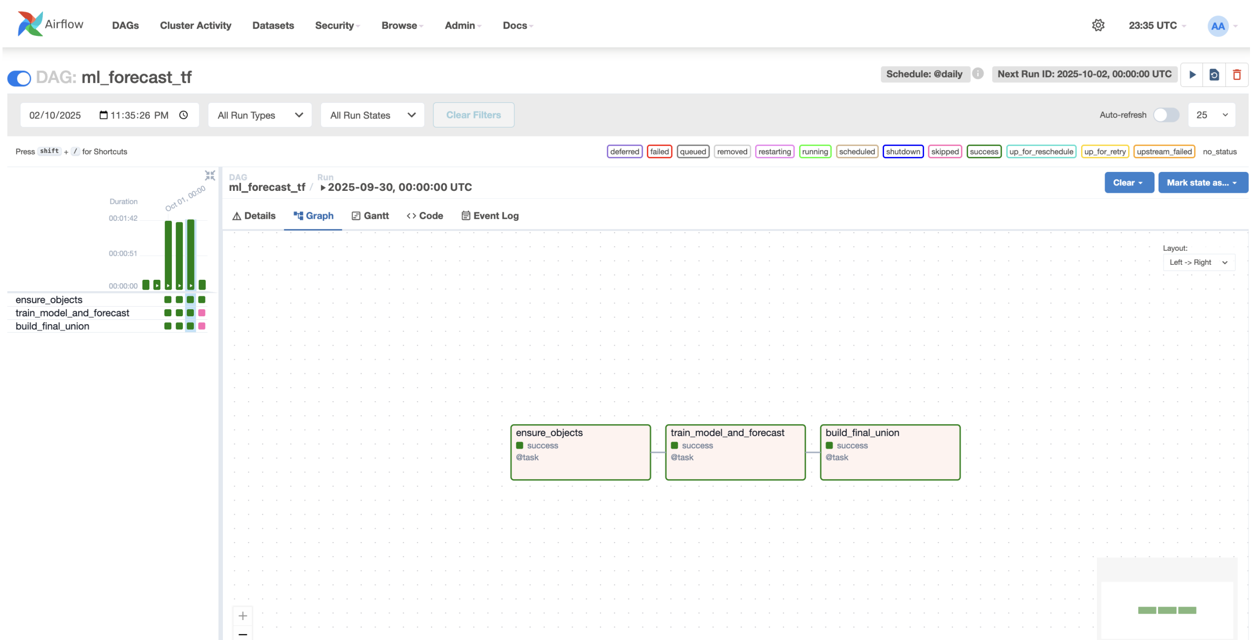

Figure 4: ml forecast Grid/Graph

MODEL.FORECASTS

SYMBOL STRING NOT NULL; TS DATE NOT NULL; PREDICTED_CLOSE FLOAT NOT NULL; MODEL_NAME STRING NOT NULL; TRAINED_AT TIMESTAMP_NTZ NOT NULL; HORIZON_D NUMBER(5,0) NOT NULL; LOAD_TS TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTA
Primary key: (SYMBOL, TS, MODEL_NAME).

ANALYTICS.FINAL_PRICES_FORECAST

SYMBOL STRING NOT NULL; TS DATE NOT NULL; CLOSE FLOAT; SOURCE STRING NOT NULL; MODEL_NAME STRING; LOAD_TS TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(
Primary key: (SYMBOL, TS, SOURCE); SOURCE $\in$ {ACTUAL, FORECAST}.

One-time Bootstrap DDL

Listing 1: Bootstrap DDL (run once).

```
BEGIN;
CREATE SCHEMA IF NOT EXISTS RAW;
CREATE SCHEMA IF NOT EXISTS MODEL;
CREATE SCHEMA IF NOT EXISTS ANALYTICS;

CREATE TABLE IF NOT EXISTS RAW.STOCK_PRICES (
  SYMBOL STRING NOT NULL, TS TIMESTAMP_NTZ NOT NULL,
  OPEN FLOAT, HIGH FLOAT, LOW FLOAT, CLOSE FLOAT, ADJ_CLOSE FLOAT,
     VOLUME NUMBER(38,0),
```

```
  LOAD_TS TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
  CONSTRAINT PK_STOCK_PRICES PRIMARY KEY (SYMBOL, TS)
);

CREATE TABLE IF NOT EXISTS MODEL.FORECASTS (
  SYMBOL STRING NOT NULL, TS DATE NOT NULL, PREDICTED_CLOSE FLOAT
     NOT NULL,
  MODEL_NAME STRING NOT NULL, TRAINED_AT TIMESTAMP_NTZ NOT NULL,
     HORIZON_D NUMBER(5,0) NOT NULL,
  LOAD_TS TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(),
  CONSTRAINT PK_FORECASTS PRIMARY KEY (SYMBOL, TS, MODEL_NAME)
);

CREATE TABLE IF NOT EXISTS ANALYTICS.FINAL_PRICES_FORECAST (
  SYMBOL STRING NOT NULL, TS DATE NOT NULL, CLOSE FLOAT,
  SOURCE STRING NOT NULL, MODEL_NAME STRING, LOAD_TS TIMESTAMP_NTZ
     DEFAULT CURRENT_TIMESTAMP(),
  CONSTRAINT PK_FINAL PRIMARY KEY (SYMBOL, TS, SOURCE)
);
COMMIT;
```

4 Implementation

4.1 Airflow Connections & Variables

We created a Snowflake Connection **snowflake_catfish** with account, user, password, role, default warehouse CATFISH_QUERY_WH, and database USER_DB_CATFISH. **No secrets in code.** Variables:

- stock_symbols: JSON list, e.g., ["AAPL","MSFT","TSLA"].
- lookback_days: e.g., 365.
- forecast_horizon_days: e.g., 14.
- target_schema_raw=RAW, target_schema_model=MODEL, target_schema_analytics=ANA

4.2 DAG #1: yfinance_etl (ETL)

Python downloads OHLCV for stock_symbols over the last lookback_days and MERGEs into RAW.STOCK_PRICES. Credentials/DB/WH/role come from the Airflow Connection at runtime (via snowflake.connector). We use a transactional pattern (commit/rollback).

4.3 Create Snowflake Connection in Airflow UI

Step 1: Open the Airflow web UI (e.g., http://localhost:8080).

Step 2: Navigate to **Admin** → **Connections**.

Step 3: Click **+** (*Add a new record*).

Step 4: Fill the form:

- **Conn Id**: `snowflake_catfish`
- **Conn Type**: `Snowflake`
- **Login**: your Snowflake *username*
- **Password**: your Snowflake *password*
- **Extra** (JSON): paste the following and adjust values:

```
{
  "account":   "abcd-xy123",
  "warehouse": "COMPUTE_WH",
  "database":  "YOUR_DB",
  "schema":    "RAW",
  "role":      "SYSADMIN"
}
```

Step 5: Click **Test** (top-right). If successful, click **Save**.

Notes.

- The DAGs access this via `BaseHook.get_connection("snowflake_catfish")` and read fields from `c.extra_dejson`.
- If your ML pipeline expects a different default schema (e.g., `MODEL`), either change it here or pass a schema argument in code.

4.4 Create Airflow Variables

Step 1: Go to **Admin** → **Variables**.

Step 2: Click **+** to add each key/value below (use exact keys):

Runtime DML (executed by the DAG after staging rows):

SQL

Listing 2: ETL MERGE (executed each run).

```

BEGIN;
CREATE TEMP TABLE TMP_LOAD (
  SYMBOL STRING, TS TIMESTAMP_NTZ, OPEN FLOAT, HIGH FLOAT, LOW FLOAT
    ,
  CLOSE FLOAT, ADJ_CLOSE FLOAT, VOLUME NUMBER(38,0)
```

```sql
);
-- Python inserts many rows into TMP_LOAD via executemany(...)

MERGE INTO RAW.STOCK_PRICES AS t
USING TMP_LOAD AS s
  ON  t.SYMBOL = s.SYMBOL
  AND t.TS     = s.TS
WHEN MATCHED THEN UPDATE SET
  OPEN=s.OPEN, HIGH=s.HIGH, LOW=s.LOW, CLOSE=s.CLOSE,
  ADJ_CLOSE=s.ADJ_CLOSE, VOLUME=s.VOLUME, LOAD_TS=CURRENT_TIMESTAMP()
WHEN NOT MATCHED THEN INSERT (
  SYMBOL, TS, OPEN, HIGH, LOW, CLOSE, ADJ_CLOSE, VOLUME
) VALUES (
  s.SYMBOL, s.TS, s.OPEN, s.HIGH, s.LOW, s.CLOSE, s.ADJ_CLOSE, s.VOLUME
);
COMMIT;
```

4.5 DAG #2: `ml_forecast` (Snowflake ML) + Final Union

This DAG uses only `SnowflakeOperator`. It trains/updates a multi-series model via `SNOWFLAKE.ML.FORE` stages horizon-wide predictions, MERGEs them into `MODEL.FORECASTS`, and rebuilds `ANALYTICS.FINAL_F` by unioning ACTUAL and FORECAST rows.

Model Training, Forecasting & Upsert

Listing 3: Snowflake ML model + forecast + upsert.

```sql
BEGIN;
USE SCHEMA MODEL;

WITH symbols AS (
  SELECT value::string AS symbol
  FROM TABLE(FLATTEN(input => PARSE_JSON('{{ var.value.stock_symbols }}')))
),
training_data AS (
  SELECT
    TO_VARIANT(sp.SYMBOL) AS SERIES,
    sp.TS,
    sp.CLOSE
  FROM RAW.STOCK_PRICES sp
  JOIN symbols s ON s.symbol = sp.SYMBOL
```

```
  WHERE sp.TS >= DATEADD('day', -{{ var.value.lookback_days | default('365', true) }}, CURRENT_TIMESTAMP())
)

CREATE OR REPLACE SNOWFLAKE.ML.FORECAST PRICE_FORECASTER (
  INPUT_DATA         => SYSTEM$QUERY_REFERENCE($$ SELECT SERIES, TS, CLOSE FROM training_data $$),
  SERIES_COLNAME     => 'SERIES',
  TIMESTAMP_COLNAME => 'TS',
  TARGET_COLNAME     => 'CLOSE',
  CONFIG_OBJECT      => {{ '{{' }} 'method':'fast','on_error':'skip' {{ '}}' }}
);

CREATE OR REPLACE TEMP TABLE TMP_FC AS
SELECT
  SERIES::STRING                    AS SYMBOL,
  CAST(TS AS DATE)                  AS TS,
  FORECAST                          AS PREDICTED_CLOSE,
  'SNOWFLAKE_ML'                    AS MODEL_NAME,
  CURRENT_TIMESTAMP()               AS TRAINED_AT,
  {{ var.value.forecast_horizon_days | default('14', true) }}::NUMBER AS HORIZON_D
FROM TABLE(PRICE_FORECASTER!FORECAST(
  FORECASTING_PERIODS => {{ var.value.forecast_horizon_days | default('14', true) }}
));

MERGE INTO MODEL.FORECASTS AS t
USING TMP_FC AS s
  ON  t.SYMBOL     = s.SYMBOL
  AND t.TS         = s.TS
  AND t.MODEL_NAME = s.MODEL_NAME
WHEN MATCHED THEN UPDATE SET
  PREDICTED_CLOSE = s.PREDICTED_CLOSE,
  TRAINED_AT      = s.TRAINED_AT,
  HORIZON_D       = s.HORIZON_D,
  LOAD_TS         = CURRENT_TIMESTAMP()
WHEN NOT MATCHED THEN INSERT (
  SYMBOL, TS, PREDICTED_CLOSE, MODEL_NAME, TRAINED_AT, HORIZON_D
) VALUES (
  s.SYMBOL, s.TS, s.PREDICTED_CLOSE, s.MODEL_NAME, s.TRAINED_AT, s.HORIZON_D
);
COMMIT;
```

Final Union Build (ACTUAL FORECAST)

Listing 4: Rebuild ANALYTICS final table.

```
BEGIN;
USE SCHEMA ANALYTICS;

WITH symbols AS (
  SELECT value::string AS symbol
  FROM TABLE(FLATTEN(input => PARSE_JSON('{{ var.value.stock_symbols
      }}')))
)

TRUNCATE TABLE ANALYTICS.FINAL_PRICES_FORECAST;

-- ACTUALS from RAW
INSERT INTO ANALYTICS.FINAL_PRICES_FORECAST (SYMBOL, TS, CLOSE,
    SOURCE, MODEL_NAME)
SELECT
  sp.SYMBOL,
  CAST(sp.TS AS DATE) AS TS,
  sp.CLOSE,
  'ACTUAL' AS SOURCE,
  NULL      AS MODEL_NAME
FROM RAW.STOCK_PRICES sp
JOIN symbols s ON s.symbol = sp.SYMBOL;

-- FORECASTS from MODEL
INSERT INTO ANALYTICS.FINAL_PRICES_FORECAST (SYMBOL, TS, CLOSE,
    SOURCE, MODEL_NAME)
SELECT
  f.SYMBOL,
  f.TS,
  f.PREDICTED_CLOSE AS CLOSE,
  'FORECAST'         AS SOURCE,
  f.MODEL_NAME
FROM MODEL.FORECASTS f
JOIN symbols s ON s.symbol = f.SYMBOL;

COMMIT;
```

4.6 Transactions & Error Handling

- **DAG #1**: `snowflake.connector` uses `conn.autocommit(False)` and wraps DDL/DML in `try/except` with `commit()` on success and `rollback()` on failure.
- **DAG #2**: Each `SnowflakeOperator` task uses `BEGIN; ...COMMIT;` so the entire step is atomic.

5 Results

After triggering yfinance_etl and then ml_forecast, we validated:

- RAW.STOCK_PRICES contains daily OHLCV for each ticker.
- MODEL.FORECASTS contains forecast_horizon_days predictions per ticker with MODEL_NAME='SNOWFLAKE_ML'.
- ANALYTICS.FINAL_PRICES_FORECAST holds both ACTUAL and FORECAST rows.

Validation Queries

```
SELECT 'RAW' AS t, COUNT(*) c FROM RAW.STOCK_PRICES
UNION ALL SELECT 'MODEL', COUNT(*) FROM MODEL.FORECASTS
UNION ALL SELECT 'ANALYTICS', COUNT(*) FROM ANALYTICS.
    FINAL_PRICES_FORECAST;

SELECT SYMBOL, MIN(TS) AS min_ts, MAX(TS) AS max_ts, COUNT(*) AS n
FROM ANALYTICS.FINAL_PRICES_FORECAST
GROUP BY SYMBOL
ORDER BY SYMBOL, min_ts;
```

6 Discussion

Pros. Snowflake-side training avoids heavy Python deps and simplifies daily orchestration. Airflow manages schedule, retries, and secret handling via Connections/Variables. Transactions ensure atomic loads and clear failure modes.

create the Airflow Connection and Variables

SYMBOL STRING NOT NULL; TS DATE NOT NULL; CLOSE FLOAT; SOURCE STRING NOT NULL; MODEL_NAME STRING; LOAD_TS TIMESTAMP_NTZ DEFAULT CURRENT_TIMESTAMP(
Primary key: (SYMBOL, TS, SOURCE); SOURCE $\in$ {ACTUAL, FORECAST}.
Airflow Setup: Snowflake Connection & Variables
Required

- stock_symbols (JSON):

  ["AAPL", "MSFT", "TSLA"]

- lookback_days: 365
- target_schema_raw: RAW

Optional (recommended)

- `forecast_horizon_days`: `14`
- `target_schema_model`: `MODEL`
- `target_schema_analytics`: `ANALYTICS`

Tips.

- Ensure JSON variables are valid (double quotes, no trailing commas).
- Access in code with `Variable.get("stock_symbols")`; parse JSON if needed.
- UI changes to Connections/Variables take effect immediately; new DAG code may need a short scheduler refresh window.

7 Quick Checklist

- `snowflake_catfish` exists and **Test** passes.
- `stock_symbols`, `lookback_days`, and `target_schema_raw` are present.
- DAG code references the same *Conn Id* and variable names.

8 Common Issues & Fixes

- **"The conn_id isn't defined"**: the Connection ID in UI doesn't match the DAG.
- **Auth failures**: wrong `account/role/warehouse` in `Extra`.
- **JSON errors in Variables**: re-save with valid JSON (use the code blocks above).

9 Conclusion

The pipeline meets the lab goals: ETL → Snowflake ML Forecast → union table; clean Airflow orchestration with transactions; secure configuration using Connections/Variables.

References

- yfinance (PyPI): `https://pypi.org/project/yfinance/`
- Apache Airflow: `https://airflow.apache.org/`
- Snowflake Documentation: `https://docs.snowflake.com/`