Automated Stock Price Forecasting and Visualization with Snowflake, Apache Airflow, dbt, yfinance, and Superset

Alisha Kartik
Department of Applied Data Science
San Jose State University
Student ID: 017518813

Rajarajeswari Premanand
Department of Applied Data Science
San Jose State University
Student ID: 017776993

***Abstract*—This project describes the development of a predictive analytics framework aimed at estimating and visualizing seven-day stock price trends for three financial entities: Wells Fargo & Co. (WFC), Bank of America Corp. (BAC), and JPMorgan Chase &Co. (JPM). With the help of the yfinance API, stock price data from the last 180 days is systematically collected and stored in a Snowflake database. To keep the records current, an automated data pipeline powered by Apache Airflow ensures the regular retrieval and processing of newly released data. Following this, an ELT pipeline is built within Snowflake, working alongside dbt and managed through Airflow, to transform raw data and create feature sets that are appropriate for predictive analysis. In the end, integrated machine learning models generate accurate, real-time predictions of seven-day stock price patterns, which are then displayed in Superset for interactive engagement and ongoing monitoring.**

I. Introduction

This initiative provides a comprehensive, end-to-end data warehousing and analytics solution for JPMorgan Chase & Co. (JPM), Bank of America Corp. (BAC), and Wells Fargo & Co. (WFC) utilizing Snowflake, Apache Airflow, dbt, and Superset.

We begin by using the yfinance API to gather daily price data from the past 180 days, including open, high, low, close, and volume for each stock ticker. This data is ingested into Snowflake's "raw" schema through Airflow's SnowflakeOperator, with credentials securely managed by Airflow Variables and Connections. A dedicated Snowflake virtual warehouse provides scalable computational resources for both data ingestion and transformation processes. Next, daily scheduled Airflow DAGs activate the ETL pipeline; raw tables are moved to "staging" views, after which dbt models within the "analytics" schema implement thorough transformations and enforce version control.

The dbt models calculate essential metrics—daily returns, 5- and 20-day simple moving averages, and Relative Strength Index (RSI)—and incorporate data quality tests such as non-null constraints, unique primary keys, and range checks on value outputs. The transformed data is stored in Snowflake tables partitioned by trade_date to enhance query performance. Finally, Superset establishes a direct relationship with the "analytics" schema, enabling the creation of interactive bar and line charts, time-series sliders, and filterable dashboards that allow users to examine volatility among tickers, investigate unusual RSI signals, and download CSV snapshots.

By integrating Snowflake's high-performance storage features, Airflow's automated orchestration, dbt's rigorous transformations, and Superset's business intelligence functionalities, this solution provides investors with reliable, real-time insights that facilitate more informed, data-driven trading decisions.

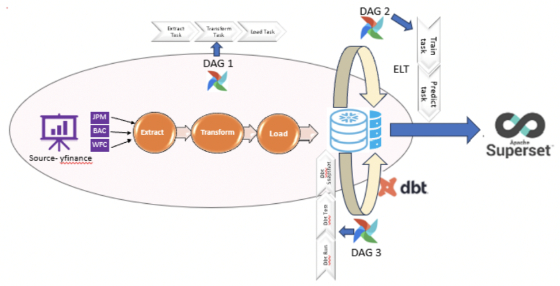


Fig. 1. System Design

II. Requirements

1) **Data Sources and APIs**
   - yfinance API: Retrieves historical stock price data.
   - Snowflake: Stores and processes stock price data.
2) **Infrastructure and Tools**
   - Apache Airflow: Automates the ETL pipeline.
   - Docker: Containerizes Airflow.
3) **Machine Learning Framework**
   - Snowflake ML Functions for stock price forecasting.
   - SNOWFLAKE.ML.FORECAST aids in model execution.

4) **Storage and Processing**
   - Raw schema for stock prices.
   - Analytics schema for models, processed data, and predictions.
5) **Superset**
   - Connects to analytics schema for interactive dashboards.
6) **Security and Access Control**
   - Airflow Connections/Variables for credentials.
   - Role-based access in Snowflake.

III. Data Description

Data for JPM, BAC, and WFC is collected via yfinance. and stored in Snowflake with the following schema:

A. Variable Descriptions

- **Symbol (STRING)**:
  Description: The STRING data type is used for stock symbols (e.g., JPM, BAC, WFC). It supports varying alphanumeric lengths, making querying and filtering efficient.This field is part of the PRIMARY KEY when combined with Date
- **Date (TIMESTAMP)**:
  Description: The TIMESTAMP data type records each stock price entry's precise date and time. It is marked as a component of the primary key to ensure data consistency and prevent duplication.
- **Open (NUMBER)**:
  Description: The NUMBER data type is used to represent the stock's opening price. This allows for decimal values,ensuring accurate representation of stock prices down to cents.
- **High(Number):**
  Description: The NUMBER data type is used to store the highest price of the stock during a trading session. This ensures precise tracking of the maximum price fluctuations in a day.
- **Close (NUMBER)**
  Description: The NUMBER data type represents the closing stock price at the end of the trading session. This value is critical for historical price analysis and forecasting.
- **Low(NUMBER)**
  Description: The NUMBER data type is also used for the lowest price, allowing for the representation of decimal values. This is important for accurately capturing the lowest price point during the trading period, which can be crucial for financial analysis.
- **Volume (NUMBER)**:
  Description: The INTEGER data type is used to store the total number of shares traded during the session.

B. Table Descriptions

Stock

```
CREATE OR REPLACE TABLE USER_DB_BLUEJAY.RAW.
    STOCK (
  SYMBOL VARCHAR(16777216) NOT NULL,
  DATE TIMESTAMP_NTZ(9) NOT NULL,
  OPEN NUMBER(38,4),
  HIGH NUMBER(38,4),
  LOW NUMBER(38,4),
  CLOSE NUMBER(38,4),
  VOLUME NUMBER(38,0),
  PRIMARY KEY (SYMBOL, DATE)
);
```

Stock_Forecast

```
CREATE OR REPLACE TABLE USER_DB_BLUEJAY.RAW.
    STOCK_FORECAST (
  SERIES VARIANT,
  TS TIMESTAMP_NTZ(9),
  FORECAST FLOAT,
  LOWER_BOUND FLOAT,
  UPPER_BOUND FLOAT
);
```

Final_Data

```
CREATE OR REPLACE TABLE USER_DB_BLUEJAY.RAW.
    FINAL_DATA (
  SYMBOL VARCHAR(16777216),
  DATE TIMESTAMP_NTZ(9),
  ACTUAL NUMBER(38,4),
  FORECAST FLOAT,
  LOWER_BOUND FLOAT,
  UPPER_BOUND FLOAT
);
```

RSI

```
CREATE OR REPLACE TRANSIENT TABLE
    USER_DB_BLUEJAY.ANALYTICS.RSI (
  SYMBOL VARCHAR(16777216),
  TRADE_DATE DATE,
  CLOSE FLOAT,
  AVG_GAIN_14D FLOAT,
  AVG_LOSS_14D FLOAT,
  RSI_14D FLOAT,
  STOCK_ID VARCHAR(16777216)
);
```

Moving Averages

```
CREATE OR REPLACE TRANSIENT TABLE
    USER_DB_BLUEJAY.ANALYTICS.MOVING_AVERAGES (
  SYMBOL VARCHAR(16777216),
  TRADE_DATE DATE,
  CLOSE FLOAT,
  CLOSE_PRICE_7D_MA FLOAT,
  CLOSE_PRICE_14D_MA FLOAT,
  CLOSE_PRICE_30D_MA FLOAT,
  STOCK_ID VARCHAR(16777216)
);
```

Volume Summary

```
CREATE OR REPLACE TRANSIENT TABLE
    USER_DB_BLUEJAY.ANALYTICS.VOLUME_SUMMARY (
  SYMBOL VARCHAR(16777216),
  TRADE_DATE DATE,
  WEEKLY_VOLUME NUMBER(38,0),
  STOCK_ID VARCHAR(16777216)
);
```

RSI Snapshot

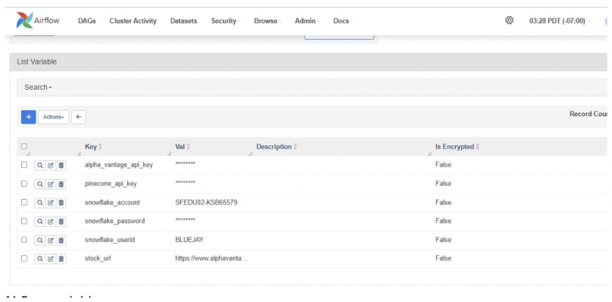
```
CREATE OR REPLACE TRANSIENT TABLE
    USER_DB_BLUEJAY.SNAPSHOTS.RSI_SNAPSHOT (
  SYMBOL VARCHAR(16777216),
  TRADE_DATE DATE,
  CLOSE FLOAT,
  AVG_GAIN_14D FLOAT,
  AVG_LOSS_14D FLOAT,
  RSI_14D FLOAT,
  STOCK_ID VARCHAR(16777216),
  DBT_SCD_ID VARCHAR(32),
  DBT_UPDATED_AT TIMESTAMP_NTZ(9),
  DBT_VALID_FROM TIMESTAMP_NTZ(9),
  DBT_VALID_TO TIMESTAMP_NTZ(9)
);
```

Moving Averages Snapshot

```
CREATE OR REPLACE TRANSIENT TABLE
    USER_DB_BLUEJAY.SNAPSHOTS.
    MOVING_AVERAGES_SNAPSHOT (
  SYMBOL VARCHAR(16777216),
  TRADE_DATE DATE,
  CLOSE FLOAT,
  CLOSE_PRICE_7D_MA FLOAT,
  CLOSE_PRICE_14D_MA FLOAT,
  CLOSE_PRICE_30D_MA FLOAT,
  STOCK_ID VARCHAR(16777216),
  DBT_SCD_ID VARCHAR(32),
  DBT_UPDATED_AT TIMESTAMP_NTZ(9),
  DBT_VALID_FROM TIMESTAMP_NTZ(9),
  DBT_VALID_TO TIMESTAMP_NTZ(9)
);
```

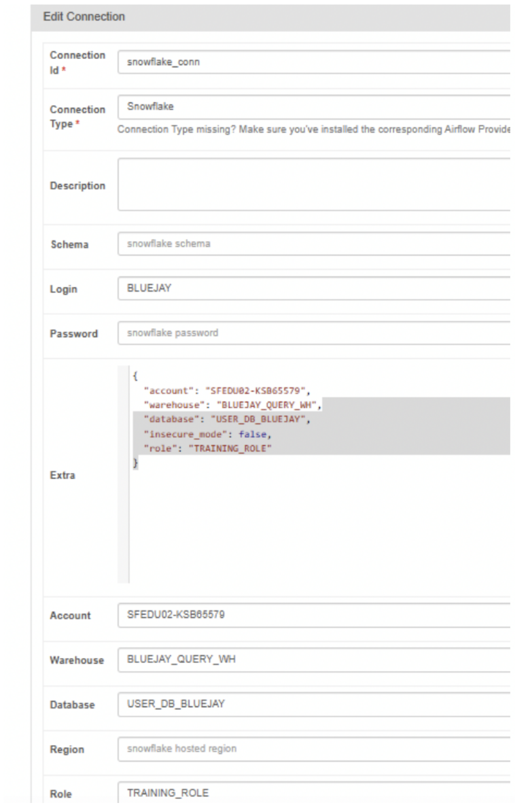
Stock Data Snapshot

```
CREATE OR REPLACE TRANSIENT TABLE
    USER_DB_BLUEJAY.SNAPSHOTS.
    STOCK_DATA_SNAPSHOT (
  SYMBOL VARCHAR(16777216),
  TRADE_DATE DATE,
  CLOSE FLOAT,
  VOLUME NUMBER(38,0),
  DAILY_CHANGE FLOAT,
  MOVING_AVG_7 NUMBER(38,2),
  STOCK_ID VARCHAR(16777216),
  DBT_SCD_ID VARCHAR(32),
  DBT_UPDATED_AT TIMESTAMP_NTZ(9),
  DBT_VALID_FROM TIMESTAMP_NTZ(9),
  DBT_VALID_TO TIMESTAMP_NTZ(9)
);
```



Key	Val	Description	Is Encrypted
alpha_vantage_api_key	********		False
pinecone_api_key	********		False
snowflake_account	SFEDU02-KSB65579		False
snowflake_password	********		False
snowflake_userid	BLUEJAY		False
stock_url	https://www.alphavanta...		False

Fig. 2. Airflow Variables



Connection Id *	snowflake_conn
Connection Type *	Snowflake Connection Type missing? Make sure you've installed the corresponding Airflow Provide
Description	
Schema	snowflake schema
Login	BLUEJAY
Password	snowflake password
Extra	
Account	SFEDU02-KSB65579
Warehouse	BLUEJAY_QUERY_WH
Database	USER_DB_BLUEJAY
Region	snowflake hosted region
Role	TRAINING_ROLE

```
{
  "account": "SFEDU02-KSB65579",
  "warehouse": "BLUEJAY_QUERY_WH",
  "database": "USER_DB_BLUEJAY",
  "insecure_mode": false,
  "role": "TRAINING_ROLE"
}
```

Fig. 3. Airflow Connections

IV. Functional Analysis

A. Data Collection

- We gather historical daily stock price information through the yfinance API for three tickers: JPM, BAC, and WFC.
- Each extracted variable (volume, date, close, high, low, and open) is converted into structured formats and stored accordingly.
- An automated job in Airflow is set up to fetch and keep only the latest 180 days of stock price data.

B. Airflow Variables and Connections

- Secure Airflow variables manage sensitive credentials for yfinance API keys and Snowflake login.
- Airflow UI configures connections to the Snowflake database.

C. Configure Docker

-
  - Docker was used to containerize Apache Airflow and Superset, with Snowflake connectivity provided by the `apache-airflow-providers-snowflake` and `snowflake-connector-python` packages and transformation support via `dbt-snowflake`.

- A concise Docker Compose file (`docker-compose.yaml`) orchestrates service dependencies, volume mounts, and port mappings for the analytics stack.
- The `docker-compose.yaml` file defines and links three primary services—Airflow scheduler/webserver, Superset, and any required databases—by specifying each container's build context or image, exposed ports, shared volumes (for DAGs, logs, and configuration), environment variables (e.g., Snowflake connection credentials), and inter-service dependencies.
- Under the Airflow service, we point to a custom Dockerfile that copies `requirements.txt` into the image and runs `pip install -r requirements.txt` to install Python libraries such as `snowflake-connector-python`, `apache-airflow-providers-snowflake`, `dbt-snowflake`, and `yfinance`.
- This setup guarantees that, at container startup, Airflow has all necessary Python and dbt dependencies to orchestrate the ETL/ELT pipelines, connect securely to Snowflake, and execute dbt models.

D. Data Storage

- The Snowflake database contains two schemas:
  - Raw Data: Stores raw stock data.
  - Analytics: Maintains machine learning models and processed data.
- The stock prices table in the Raw Data schema loads and maintains stock price data restricted to the last 180 days.
- To ensure data accuracy and consistency, an Apache Airflow DAG (Directed Acyclic Graph) is set up to automatically refresh the Snowflake database with the stock prices for the previous 180 days every day.

E. Data Processing

1) ETL DAG

- Extract the most recent 180 days of OHLCV data for each ticker via the Alpha Vantage (or yfinance) API.
- Transform and upsert into Snowflake's raw schema using SQL atomic transactions to guarantee idempotency.
- Load into `raw.stock` table, keyed on (symbol, date), so reruns won't duplicate or miss records.

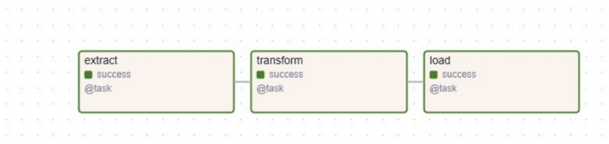
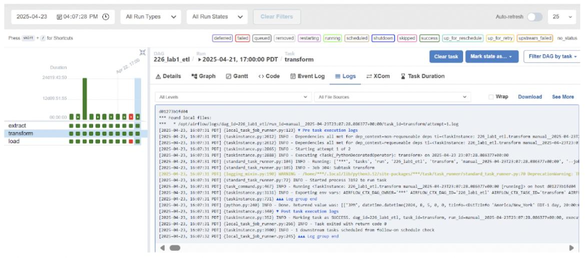


Fig. 4. ETL DAG Flow

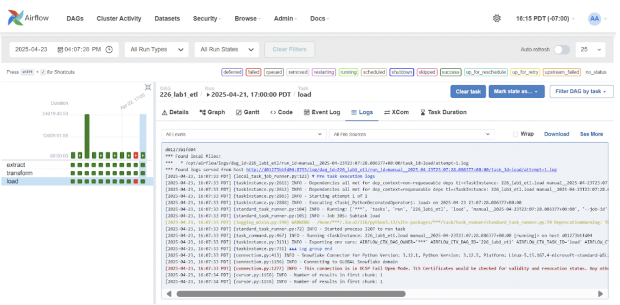
Fig. 5. ETL Extract Task



DAG 226_lab1_etl / Run 2025-04-21, 17:00:00 PDT / Task transform

```
d01273b1fd04
*** Found local files:
***   * /opt/airflow/logs/dag_id=226_lab1_etl/run_id=manual__2025-04-23T23:07:28.086377+00:00/task_id=transform/attempt=1.log
[2025-04-23, 16:07:31 PDT] {local_task_job_runner.py:123} ▼ Pre task execution logs
[2025-04-23, 16:07:31 PDT] {taskinstance.py:2612} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: 226_lab1_etl.transform manual__2025-04-23T
[2025-04-23, 16:07:31 PDT] {taskinstance.py:2612} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: 226_lab1_etl.transform manual__2025-04-23T23:0
[2025-04-23, 16:07:31 PDT] {taskinstance.py:2865} INFO - Starting attempt 1 of 2
[2025-04-23, 16:07:31 PDT] {taskinstance.py:2888} INFO - Executing <Task(_PythonDecoratedOperator): transform> on 2025-04-23 23:07:28.086377+00:00
[2025-04-23, 16:07:31 PDT] {standard_task_runner.py:104} INFO - Running: ['***', 'tasks', 'run', '226_lab1_etl', 'transform', 'manual__2025-04-23T23:07:28.086377+00:00', '--jo
[2025-04-23, 16:07:31 PDT] {standard_task_runner.py:105} INFO - Job 304: Subtask transform
[2025-04-23, 16:07:31 PDT] {logging_mixin.py:190} WARNING - /home/***/.local/lib/python3.12/site-packages/***/task/task_runner/standard_task_runner.py:70 DeprecationWarning: T
[2025-04-23, 16:07:31 PDT] {standard_task_runner.py:72} INFO - Started process 3192 to run task
[2025-04-23, 16:07:31 PDT] {task_command.py:467} INFO - Running <TaskInstance: 226_lab1_etl.transform manual__2025-04-23T23:07:28.086377+00:00 [running]> on host d01273b1fd04
[2025-04-23, 16:07:31 PDT] {taskinstance.py:3131} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='***' AIRFLOW_CTX_DAG_ID='226_lab1_etl' AIRFLOW_CTX_TASK_ID='transform' AIRF
[2025-04-23, 16:07:31 PDT] {taskinstance.py:731} ▲▲▲ Log group end
[2025-04-23, 16:07:31 PDT] {python.py:240} INFO - Done. Returned value was: [['JPM', datetime.datetime(2024, 8, 5, 0, 0, tzinfo=<DstTzInfo 'America/New_York' EDT-1 day, 20:00:
[2025-04-23, 16:07:31 PDT] {taskinstance.py:340} ▼ Post task execution logs
[2025-04-23, 16:07:31 PDT] {taskinstance.py:352} INFO - Marking task as SUCCESS. dag_id=226_lab1_etl, task_id=transform, run_id=manual__2025-04-23T23:07:28.086377+00:00, execu
[2025-04-23, 16:07:31 PDT] {local_task_job_runner.py:266} INFO - Task exited with return code 0
[2025-04-23, 16:07:32 PDT] {taskinstance.py:3900} INFO - 1 downstream tasks scheduled from follow-on schedule check
[2025-04-23, 16:07:32 PDT] {local_task_job_runner.py:245} ▲▲▲ Log group end
```

Fig. 6. ETL Transform Task



DAG 226_lab1_etl / Run 2025-04-21, 17:00:00 PDT / Task load

```
d01273b1fd04
*** Found local files:
***   * /opt/airflow/logs/dag_id=226_lab1_etl/run_id=manual__2025-04-23T23:07:28.086377+00:00/task_id=load/attempt=1.log
*** Found logs served from host http://d01273b1fd04:8793/log/dag_id=226_lab1_etl/run_id=manual__2025-04-23T23:07:28.086377+00:00/task_id=load/attempt=1.log
[2025-04-23, 16:07:33 PDT] {local_task_job_runner.py:123} ▼ Pre task execution logs
[2025-04-23, 16:07:33 PDT] {taskinstance.py:2612} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: 226_lab1_etl.load manual__2025-04-23T23:07
[2025-04-23, 16:07:33 PDT] {taskinstance.py:2612} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: 226_lab1_etl.load manual__2025-04-23T23:07:28.
[2025-04-23, 16:07:33 PDT] {taskinstance.py:2865} INFO - Starting attempt 1 of 2
[2025-04-23, 16:07:33 PDT] {taskinstance.py:2888} INFO - Executing <Task(_PythonDecoratedOperator): load> on 2025-04-23 23:07:28.086377+00:00
[2025-04-23, 16:07:33 PDT] {standard_task_runner.py:104} INFO - Running: ['***', 'tasks', 'run', '226_lab1_etl', 'load', 'manual__2025-04-23T23:07:28.086377+00:00', '--job-id'
[2025-04-23, 16:07:33 PDT] {standard_task_runner.py:105} INFO - Job 305: Subtask load
[2025-04-23, 16:07:33 PDT] {logging_mixin.py:190} WARNING - /home/***/.local/lib/python3.12/site-packages/***/task/task_runner/standard_task_runner.py:70 DeprecationWarning: T
[2025-04-23, 16:07:33 PDT] {standard_task_runner.py:72} INFO - Started process 3207 to run task
[2025-04-23, 16:07:33 PDT] {task_command.py:467} INFO - Running <TaskInstance: 226_lab1_etl.load manual__2025-04-23T23:07:28.086377+00:00 [running]> on host d01273b1fd04
[2025-04-23, 16:07:33 PDT] {taskinstance.py:3131} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='***' AIRFLOW_CTX_DAG_ID='226_lab1_etl' AIRFLOW_CTX_TASK_ID='load' AIRFLOW_C
[2025-04-23, 16:07:33 PDT] {taskinstance.py:731} ▲▲▲ Log group end
[2025-04-23, 16:07:33 PDT] {connection.py:413} INFO - Snowflake Connector for Python Version: 3.12.1, Python Version: 3.12.5, Platform: Linux-5.15.167.4-microsoft-standard-WSL
[2025-04-23, 16:07:33 PDT] {connection.py:1196} INFO - Connecting to GLOBAL Snowflake domain
[2025-04-23, 16:07:33 PDT] {connection.py:1277} INFO - This connection is in OCSP Fail Open Mode. TLS Certificates would be checked for validity and revocation status. Any oth
[2025-04-23, 16:07:34 PDT] {cursor.py:1156} INFO - Number of results in first chunk: 1
[2025-04-23, 16:07:34 PDT] {cursor.py:1156} INFO - Number of results in first chunk: 1
```

Fig. 7. ETL Load Task

2) Predict DAG

- Create or replace a Snowflake view on `raw.stock` that casts dates and selects close prices.
- Define (or replace) a forecasting stored procedure/UDF in Snowflake.
- Call that procedure to generate seven-day forecasts and write results into `raw.stock_forecast`.

- Merge actuals and forecasts into `raw.final_data`, tagging rows as “actual” or “forecast.”

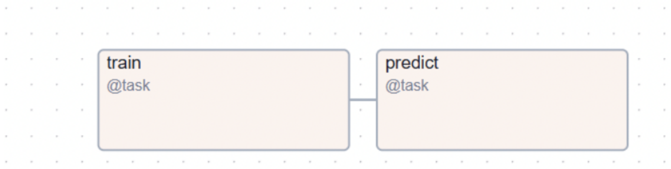
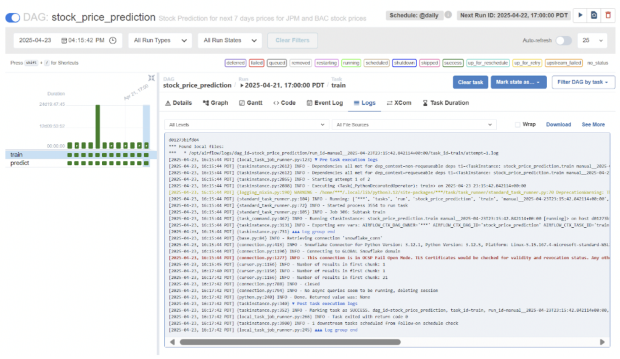


Fig. 8. Train–Predict DAG



```
d01273b1fd04
*** Found local files:
***   * /opt/airflow/logs/dag_id=stock_price_prediction/run_id=manual__2025-04-23T23:15:42.842114+00:00/task_id=train/attempt=1.log
[2025-04-23, 16:15:44 PDT] {local_task_job_runner.py:123} ▼ Pre task execution logs
[2025-04-23, 16:15:44 PDT] {taskinstance.py:2612} INFO - Dependencies all met for dep_context=non-requeueable deps ti=<TaskInstance: stock_price_prediction.train manual__2025-
[2025-04-23, 16:15:44 PDT] {taskinstance.py:2612} INFO - Dependencies all met for dep_context=requeueable deps ti=<TaskInstance: stock_price_prediction.train manual__2025-04-2
[2025-04-23, 16:15:44 PDT] {taskinstance.py:2865} INFO - Starting attempt 1 of 2
[2025-04-23, 16:15:44 PDT] {taskinstance.py:2888} INFO - Executing <Task(_PythonDecoratedOperator): train> on 2025-04-23 23:15:42.842114+00:00
[2025-04-23, 16:15:44 PDT] {logging_mixin.py:190} WARNING - /home/***/.local/lib/python3.12/site-packages/***/task/task_runner/standard_task_runner.py:70 DeprecationWarning: T
[2025-04-23, 16:15:44 PDT] {standard_task_runner.py:104} INFO - Running: ['***', 'tasks', 'run', 'stock_price_prediction', 'train', 'manual__2025-04-23T23:15:42.842114+00:00',
[2025-04-23, 16:15:44 PDT] {standard_task_runner.py:72} INFO - Started process 3554 to run task
[2025-04-23, 16:15:44 PDT] {standard_task_runner.py:105} INFO - Job 306: Subtask train
[2025-04-23, 16:15:44 PDT] {task_command.py:467} INFO - Running <TaskInstance: stock_price_prediction.train manual__2025-04-23T23:15:42.842114+00:00 [running]> on host d01273b
[2025-04-23, 16:15:44 PDT] {taskinstance.py:3131} INFO - Exporting env vars: AIRFLOW_CTX_DAG_OWNER='***' AIRFLOW_CTX_DAG_ID='stock_price_prediction' AIRFLOW_CTX_TASK_ID='train
[2025-04-23, 16:15:44 PDT] {taskinstance.py:731} ▲▲▲ Log group end
[2025-04-23, 16:15:44 PDT] {base.py:84} INFO - Retrieving connection 'snowflake_conn'
[2025-04-23, 16:15:44 PDT] {connection.py:413} INFO - Snowflake Connector for Python Version: 3.12.1, Python Version: 3.12.5, Platform: Linux-5.15.167.4-microsoft-standard-WSL
[2025-04-23, 16:15:44 PDT] {connection.py:1196} INFO - Connecting to GLOBAL Snowflake domain
[2025-04-23, 16:15:44 PDT] {connection.py:1277} INFO - This connection is in OCSP Fail Open Mode. TLS Certificates would be checked for validity and revocation status. Any oth
[2025-04-23, 16:15:45 PDT] {cursor.py:1156} INFO - Number of results in first chunk: 1
[2025-04-23, 16:17:40 PDT] {cursor.py:1156} INFO - Number of results in first chunk: 1
[2025-04-23, 16:17:42 PDT] {cursor.py:1156} INFO - Number of results in first chunk: 21
[2025-04-23, 16:17:42 PDT] {connection.py:788} INFO - closed
[2025-04-23, 16:17:42 PDT] {connection.py:794} INFO - No async queries seem to be running, deleting session
[2025-04-23, 16:17:42 PDT] {python.py:240} INFO - Done. Returned value was: None
[2025-04-23, 16:17:42 PDT] {taskinstance.py:340} ▼ Post task execution logs
[2025-04-23, 16:17:42 PDT] {taskinstance.py:352} INFO - Marking task as SUCCESS. dag_id=stock_price_prediction, task_id=train, run_id=manual__2025-04-23T23:15:42.842114+00:00,
[2025-04-23, 16:17:42 PDT] {local_task_job_runner.py:266} INFO - Task exited with return code 0
[2025-04-23, 16:17:42 PDT] {taskinstance.py:3900} INFO - 1 downstream tasks scheduled from follow-on schedule check
[2025-04-23, 16:17:42 PDT] {local_task_job_runner.py:245} ▲▲▲ Log group end
```

Fig. 9. Train Task

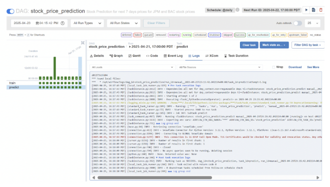


Fig. 10. Predict Task

3) **ELT DAG**

- Invoke `dbt run` to build analytics models (MAs, RSI, stock_indicators) in the `analytics` schema.
- Run `dbt test` to enforce schema and data-quality checks.
- Run `dbt snapshot` to capture historical versions of your analytics tables for auditing.

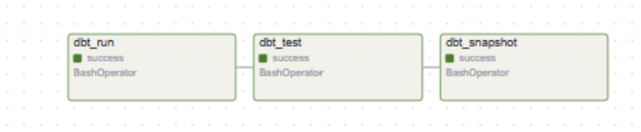
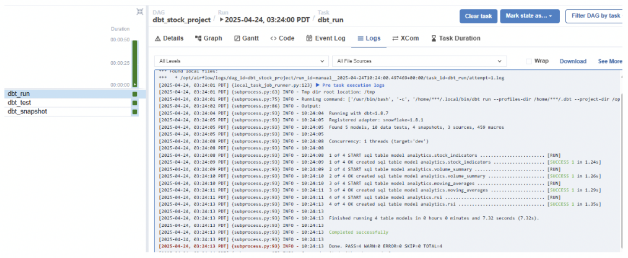


Fig. 11. ELT DAG



```
*** Found local files:
***   * /opt/airflow/logs/dag_id=dbt_stock_project/run_id=manual__2025-04-24T10:24:00.497469+00:00/task_id=dbt_run/attempt=1.log
[2025-04-24, 03:24:01 PDT] {local_task_job_runner.py:123} ▶ Pre task execution logs
[2025-04-24, 03:24:01 PDT] {subprocess.py:63} INFO - Tmp dir root location: /tmp
[2025-04-24, 03:24:01 PDT] {subprocess.py:75} INFO - Running command: ['/usr/bin/bash', '-c', '/home/***/.local/bin/dbt run --profiles-dir /home/***/.dbt --project-dir /op
[2025-04-24, 03:24:02 PDT] {subprocess.py:86} INFO - Output:
[2025-04-24, 03:24:04 PDT] {subprocess.py:93} INFO - 10:24:04  Running with dbt=1.8.7
[2025-04-24, 03:24:05 PDT] {subprocess.py:93} INFO - 10:24:05  Registered adapter: snowflake=1.8.1
[2025-04-24, 03:24:05 PDT] {subprocess.py:93} INFO - 10:24:05  Found 5 models, 10 data tests, 4 snapshots, 3 sources, 459 macros
[2025-04-24, 03:24:05 PDT] {subprocess.py:93} INFO - 10:24:05
[2025-04-24, 03:24:08 PDT] {subprocess.py:93} INFO - 10:24:08  Concurrency: 1 threads (target='dev')
[2025-04-24, 03:24:08 PDT] {subprocess.py:93} INFO - 10:24:08
[2025-04-24, 03:24:08 PDT] {subprocess.py:93} INFO - 10:24:08  1 of 4 START sql table model analytics.stock_indicators ......................... [RUN]
[2025-04-24, 03:24:09 PDT] {subprocess.py:93} INFO - 10:24:09  1 of 4 OK created sql table model analytics.stock_indicators .................... [SUCCESS 1 in 1.24s]
[2025-04-24, 03:24:09 PDT] {subprocess.py:93} INFO - 10:24:09  2 of 4 START sql table model analytics.volume_summary ........................... [RUN]
[2025-04-24, 03:24:10 PDT] {subprocess.py:93} INFO - 10:24:10  2 of 4 OK created sql table model analytics.volume_summary ...................... [SUCCESS 1 in 1.26s]
[2025-04-24, 03:24:10 PDT] {subprocess.py:93} INFO - 10:24:10  3 of 4 START sql table model analytics.moving_averages .......................... [RUN]
[2025-04-24, 03:24:11 PDT] {subprocess.py:93} INFO - 10:24:11  3 of 4 OK created sql table model analytics.moving_averages ..................... [SUCCESS 1 in 1.29s]
[2025-04-24, 03:24:11 PDT] {subprocess.py:93} INFO - 10:24:11  4 of 4 START sql table model analytics.rsi ...................................... [RUN]
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13  4 of 4 OK created sql table model analytics.rsi ................................. [SUCCESS 1 in 1.35s]
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13  Finished running 4 table models in 0 hours 0 minutes and 7.32 seconds (7.32s).
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13  Completed successfully
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13
[2025-04-24, 03:24:13 PDT] {subprocess.py:93} INFO - 10:24:13  Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
```

Fig. 12. dbt Run Task

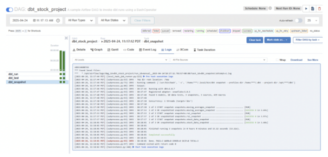


Fig. 13. dbt Snapshot Task

4) **DBT Tests**

- **Not Null**: Ensures that each entry in a designated column has a value—no empty fields or NULL values are permitted. This guarantees that no critical data (such as each date, symbol, or closing price) is missing and will raise an error if any rows breach this rule.
- **Unique**: Confirms that values in a column (or a combination of columns) are always distinct. For instance, applying uniqueness on (symbol, date) prevents the loading of duplicate prices for the same day, maintaining a one-to-one relationship between each symbol and each trading date.
- **Relationship**: Verifies referential integrity between two tables by ensuring that every value in a ”child” column corresponds to a value in a related ”parent” column. In our process, a relationship test on analytics.stock_indicators.symbol $\rightarrow$ raw.stock.symbol verifies that every symbol in our analytics models was indeed sourced from the raw data.

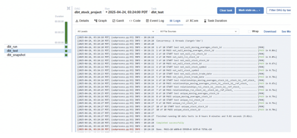


Fig. 14. dbt Test Results

G. Dashboard Visualization

The Superset dashboard connects directly to Snowflake’s `analytics` schema, queries time-series data, and renders interactive charts that surface trend patterns and key metrics at a glance.

- **RSI (Relative Strength Index):**

- The RSI is a momentum oscillator that assesses the rate and change of price fluctuations on a scale from 0 to 100, calculated by comparing recent gains to recent losses over a defined look-back period.
- *Insights:*
  * In early September, all three stocks (JPM, WFC, BAC) momentarily surpassed an RSI of 70, suggesting overbought conditions that led to a pullback in mid-September.
  * WFC and BAC fell below an RSI of 30 in December, indicating oversold conditions; both rebounded significantly in January with higher trading volume.
  * During this timeframe, BAC's RSI mostly fluctuated between 40 and 60, indicating reduced momentum swings and more range-bound trading.

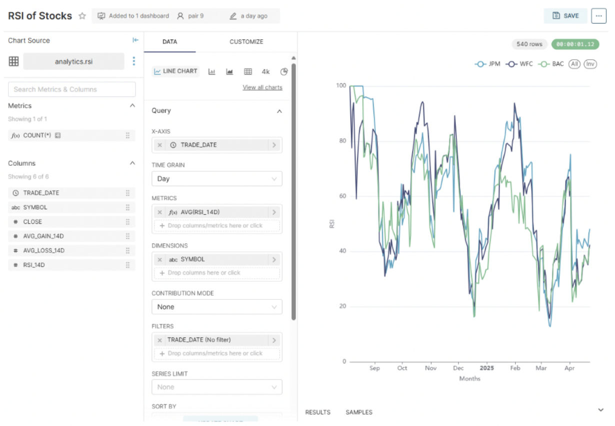

RSI of Stocks · Added to 1 dashboard · pair 9 · a day ago · SAVE

Chart Source: analytics.rsi

Metrics (Showing 1 of 1): COUNT(*)

Columns (Showing 6 of 6): TRADE_DATE, SYMBOL, CLOSE, AVG_GAIN_14D, AVG_LOSS_14D, RSI_14D

DATA | CUSTOMIZE — LINE CHART

Query

X-AXIS: TRADE_DATE

TIME GRAIN: Day

METRICS: AVG(RSI_14D)

DIMENSIONS: SYMBOL

CONTRIBUTION MODE: None

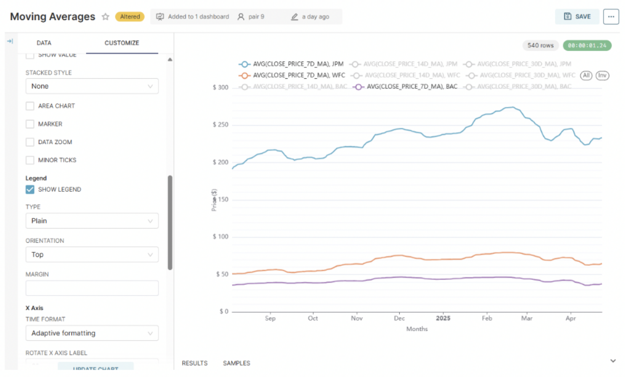
FILTERS: TRADE_DATE (No filter)

SERIES LIMIT: None

540 rows · 00:00:01.12

Fig. 15. RSI of Stocks

- **Moving Average:**
  - A moving average smooths out price data by calculating the arithmetic mean of closing prices over a rolling window.
  - *Insights:*
    * 7-Day MA: Reacts promptly to recent price fluctuations—crossing above the 14- or 30-day MA (e.g., JPM in late January) often marked multi-week surges; crossing below indicated short-term downturns.
    * 14-Day MA: Acts as a dependable mid-term trend indicator—prices often rebounded from it during October–November, and its crossover above the 30-day MA in early November confirmed the shift from consolidation into an uptrend.
    * 30-Day MA: Reduces noise to highlight the long-term direction—JPM's 30-day MA rose steadily from \$190 to \$260 (Aug–Feb), while BAC and WFC moved in narrower bands (\$40–50 and \$65–80), reflecting more subdued trends.


Moving Averages · Altered · Added to 1 dashboard · pair 9 · a day ago · SAVE

DATA | CUSTOMIZE

STACKED STYLE: None

☐ AREA CHART ☐ MARKER ☐ DATA ZOOM ☐ MINOR TICKS

Legend: ☑ SHOW LEGEND

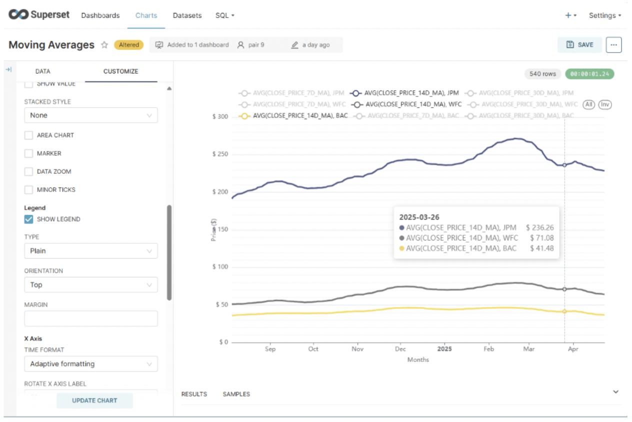
TYPE: Plain

ORIENTATION: Top

X Axis — TIME FORMAT: Adaptive formatting

540 rows · 00:00:01.24
SHOW VALUE
MARGIN
ROTATE X AXIS LABEL
UPDATE CHART
AVG(CLOSE_PRICE_7D_MA), JPM
AVG(CLOSE_PRICE_14D_MA), JPM
AVG(CLOSE_PRICE_30D_MA), JPM
AVG(CLOSE_PRICE_7D_MA), WFC
AVG(CLOSE_PRICE_14D_MA), WFC
AVG(CLOSE_PRICE_30D_MA), WFC
AVG(CLOSE_PRICE_14D_MA), BAC
AVG(CLOSE_PRICE_7D_MA), BAC
AVG(CLOSE_PRICE_30D_MA), BAC
All
Inv
$ 300
$ 250
$ 200
$ 150
$ 100
$ 50
$ 0
Price ($)
Sep
Oct
Nov
Dec
2025
Feb
Mar
Apr
Months
RESULTS
SAMPLES

Fig. 16. 7-Day Moving Average


Superset · Dashboards · Charts · Datasets · SQL · Settings

Moving Averages · Altered · Added to 1 dashboard · pair 9 · a day ago · SAVE

DATA | CUSTOMIZE

STACKED STYLE: None

☐ AREA CHART ☐ MARKER ☐ DATA ZOOM ☐ MINOR TICKS

Legend: ☑ SHOW LEGEND

TYPE: Plain

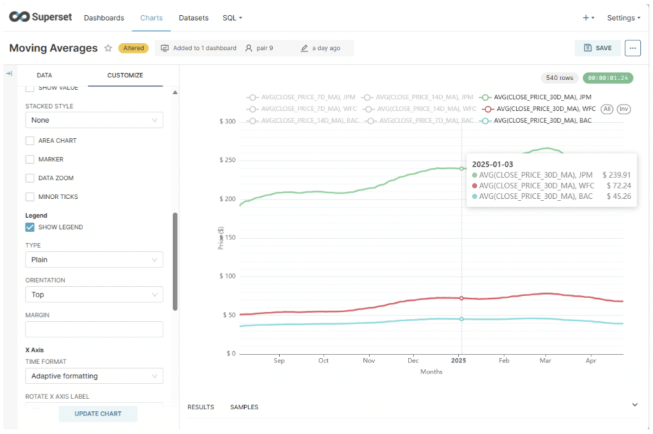
ORIENTATION: Top

X Axis — TIME FORMAT: Adaptive formatting

540 rows · 00:00:01.24

2025-03-26
● AVG(CLOSE_PRICE_14D_MA), JPM $ 236.26
● AVG(CLOSE_PRICE_14D_MA), WFC $ 71.08
● AVG(CLOSE_PRICE_14D_MA), BAC $ 41.48
SHOW VALUE
MARGIN
ROTATE X AXIS LABEL
UPDATE CHART
AVG(CLOSE_PRICE_7D_MA), JPM
AVG(CLOSE_PRICE_14D_MA), JPM
AVG(CLOSE_PRICE_30D_MA), JPM
AVG(CLOSE_PRICE_7D_MA), WFC
AVG(CLOSE_PRICE_14D_MA), WFC
AVG(CLOSE_PRICE_30D_MA), WFC
AVG(CLOSE_PRICE_14D_MA), BAC
AVG(CLOSE_PRICE_7D_MA), BAC
AVG(CLOSE_PRICE_30D_MA), BAC
All
Inv
$ 300
$ 250
$ 200
$ 150
$ 100
$ 50
$ 0
Price ($)
Sep
Oct
Nov
Dec
2025
Feb
Mar
Apr
Months
RESULTS
SAMPLES

Fig. 17. 14-Day Moving Average


Superset · Dashboards · Charts · Datasets · SQL · Settings

Moving Averages · Altered · Added to 1 dashboard · pair 9 · a day ago · SAVE

DATA | CUSTOMIZE

STACKED STYLE: None

☐ AREA CHART ☐ MARKER ☐ DATA ZOOM ☐ MINOR TICKS

Legend: ☑ SHOW LEGEND

TYPE: Plain

ORIENTATION: Top

X Axis — TIME FORMAT: Adaptive formatting

540 rows · 00:00:01.24

2025-01-03
● AVG(CLOSE_PRICE_30D_MA), JPM $ 239.91
● AVG(CLOSE_PRICE_30D_MA), WFC $ 72.24
● AVG(CLOSE_PRICE_30D_MA), BAC $ 45.26
SHOW VALUE
MARGIN
ROTATE X AXIS LABEL
UPDATE CHART
AVG(CLOSE_PRICE_7D_MA), JPM
AVG(CLOSE_PRICE_14D_MA), JPM
AVG(CLOSE_PRICE_30D_MA), JPM
AVG(CLOSE_PRICE_7D_MA), WFC
AVG(CLOSE_PRICE_14D_MA), WFC
AVG(CLOSE_PRICE_30D_MA), WFC
AVG(CLOSE_PRICE_14D_MA), BAC
AVG(CLOSE_PRICE_7D_MA), BAC
AVG(CLOSE_PRICE_30D_MA), BAC
All
Inv
$ 300
$ 250
$ 200
$ 150
$ 100
$ 50
$ 0
Price ($)
Sep
Oct
Nov
Dec
2025
Feb
Mar
Apr
Months
RESULTS
SAMPLES

Fig. 18. 30-Day Moving Average

- **Average Volume:**
  - The average volume is the mean number of shares (or contracts) traded per day over a specified look-back window.
  - *Insights:*
    - Average monthly trading volume peaked in September and April; the April surge—largely driven by BAC—confirmed significant price breakouts.
    - Oversold RSI readings in December were followed by higher-than-average volume in January, demonstrating RSI's efficacy as a contrarian buy signal.
    - High volume on JPM's up-days in late January and February reinforced the strength of its rally, while lower volume during WFC's moves indicated reduced conviction.

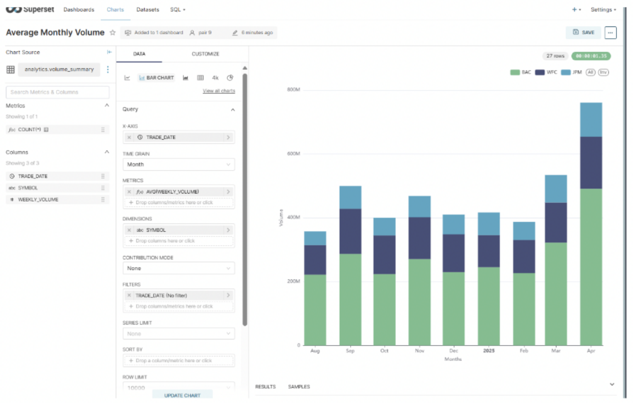

Average Monthly Volume

Chart Source: analytics.volume_summary

Metrics (Showing 1 of 1): COUNT(*)

Columns (Showing 3 of 3): TRADE_DATE, SYMBOL, WEEKLY_VOLUME

Query

X-AXIS: TRADE_DATE

TIME GRAIN: Month

METRICS: AVG(WEEKLY_VOLUME)

DIMENSIONS: SYMBOL

CONTRIBUTION MODE: None

FILTERS: TRADE_DATE (No filter)

SERIES LIMIT: None

ROW LIMIT: 10000

27 rows 00:00:01.35

Fig. 19. Average Monthly Volume

V. CODE AND RESULTS

The entire code and corresponding files are available at https://github.com/al-rv/226_Lab2_Pair9.

VI. CONCLUSION

To summarize, this project has showcased a comprehensive automated framework that encompasses the ingestion, forecasting, and visualization of stock market data. By utilizing containerized implementations of Apache Airflow and Superset, along with Snowflake's scalable storage capabilities, we created a daily ETL pipeline that consistently loads the latest 180 days of OHLCV data through the yfinance API. A specialized forecasting DAG subsequently trains and executes a Snowflake-native prediction function to produce price forecasts for seven days, which are seamlessly integrated with the historical data. Ultimately, dbt-driven ELT workflows generate crucial analytics—such as daily returns, various moving averages, trend classifications, and RSI—validated through not-null, uniqueness, and relationship testing, and displayed in interactive Superset dashboards.

The combination of effective orchestration, thorough transformation, and meaningful visualization enables stakeholders to detect momentum shifts, crossover trends, and volume-confirmed breakouts in JPM, BAC, and WFC. In the future, broadening the model library with ensemble techniques, adding sentiment or macroeconomic indicators, and facilitating real-time alerts would significantly improve the platform's predictive capabilities and operational efficiency.

REFERENCES

[1] K. Lee, *SJSU Data-226 SP25 - https://github.com/keeyong/sjsu-data226-SP25/*

[2] Apache Airflow Documentation. [Online]. Available: https://airflow.apache.org/docs/apache-airflow/stable/index.html

[3] Apache Superset Documentation. [Online]. Available: https://superset.apache.org/docs/intro/

[4] yfinance Documentation. [Online]. Available: https://pypi.org/project/yfinance/