Data 226 Lab 2 Report

Matthew Leffler, Daniel Kim

Department of Data Analytics, San Jose State University

GitHub Link: https://github.com/matthewleffler1/DATA226_Lab2

I. Problem Statement:

Stock markets are known for being unpredictable, and that makes figuring out how prices will behave—even just within a single day—a challenge for predictive analytics. While there are many ways to look at overall market trends, it's still difficult to know exactly when prices will go up or down. That said, intentful analysis can still give us an edge, helping us better understand the market and make more confident choices.

This project focuses on building an automated ELT pipeline to uncover deeper insights from raw stock data pulled from the yfinance API. We're especially interested in two types of analysis: rolling averages and price gaps, both of which offer valuable perspectives on market behavior.

The price gap, which is the difference between a stock's opening price and the previous day's closing price, highlights what's happening outside regular trading hours. These gaps can reflect after-hours news, earnings reports, or global events that impact how investors feel before the market even opens. Tracking these gaps can give us a better sense of how sentiment and momentum build overnight.

The weekly rolling average is used to track how a stock's price moves over time by smoothing out daily ups and downs. Instead of focusing on a single day's price, we calculate the average price over a 7-day window and update it daily. This makes it easier to spot trends and patterns that might otherwise be hidden by short-term market noise. In simpler terms, it helps answer the question: *Is the stock generally going up, going down, or staying steady over the past week?* This kind of insight is useful for identifying momentum and making more confident trading decisions without getting distracted by daily price swings.

To bring this analysis to life, we'll use Apache Airflow to run a daily ELT pipeline. Stock data will be pulled automatically from the yfinance API, loaded into Snowflake, and transformed using dbt. This setup not only keeps the data fresh and accurate, but also generates the abstracted metrics we need for smarter, more informed analysis of stock behavior.

II. Solution Requirements

To enable automated stock analysis and visualization, the solution is organized into four primary components: data extraction, loading into the warehouse, transformation for analytical abstraction, and visualization using BI tools.

A. Data Extraction and Loading

Daily stock data is extracted using the yfinance API, which provides key trading metrics such as open, close, high, low, and volume. To ensure the integrity and completeness of the data for any future analysis, we store it in its raw, unmodified form directly into a Snowflake table. This approach maintains flexibility, allowing additional transformations or metrics to be derived later without needing to re-query the API.

To automate this process and eliminate the need for manual intervention, we use Apache Airflow to orchestrate and schedule the extraction tasks. Airflow triggers Python-based functions that call the API and load the resulting data into Snowflake on a daily basis. This setup ensures the data is ingested consistently and on time, and it allows for precise control over the scheduling—such as running the job at market close or during off-peak hours. Overall, this automation adds both reliability and scalability to the pipeline.

B. Data Transformation and Loading

After loading the raw stock data into Snowflake, we use dbt (data build tool) to structure and manage our SQL transformations. dbt allows us to modularize logic, define reusable models, and maintain consistent structure across calculations. In this project, dbt is responsible for transforming the raw data into more meaningful insights, such as rolling averages and price gaps, while also enabling us to apply quality checks and version control through snapshotting. These transformations are triggered by Airflow and organized into models that run automatically on a daily schedule.

We're primarily focused on generating two key metrics: the rolling average of stock prices over a week and the daily price gap between the previous close and current open. To make the process modular and efficient, each metric goes through two stages: a transformation stage to compute the necessary values, and a filtering stage to clean up the results.

For the rolling average, we use a dbt ephemeral model to calculate the average over a 7-day window. We also track the row number to later remove any incomplete data. In the filtering step, we discard any rows where the rolling window isn't fully populated, ensuring the results are accurate.

For the price gap, we bring in the previous day's closing price and subtract it from the current day's opening price. We then trim the result to just the most relevant columns: date, stock symbol, and the gap value itself.

C. Creation of Visualizations

Superset is used as the BI tool to display the two key metrics—rolling average and price gap—directly from Snowflake. Its built-in database connection allows us to visualize data in real time, without needing to manually export or reprocess anything.

The rolling average is shown using a line chart, which helps highlight longer-term price trends over the 180-day window. Since the data currently has a fixed range, we didn't add a date filter. However, as more data is added, we may need to include one to keep the chart clear and responsive.

For the price gap, we use a time-series bar chart to show both the direction and size of each day's price movement. This makes it easy to see whether a stock opens higher or lower than the day before—and by how much—so we can quickly understand what's happening in the market.

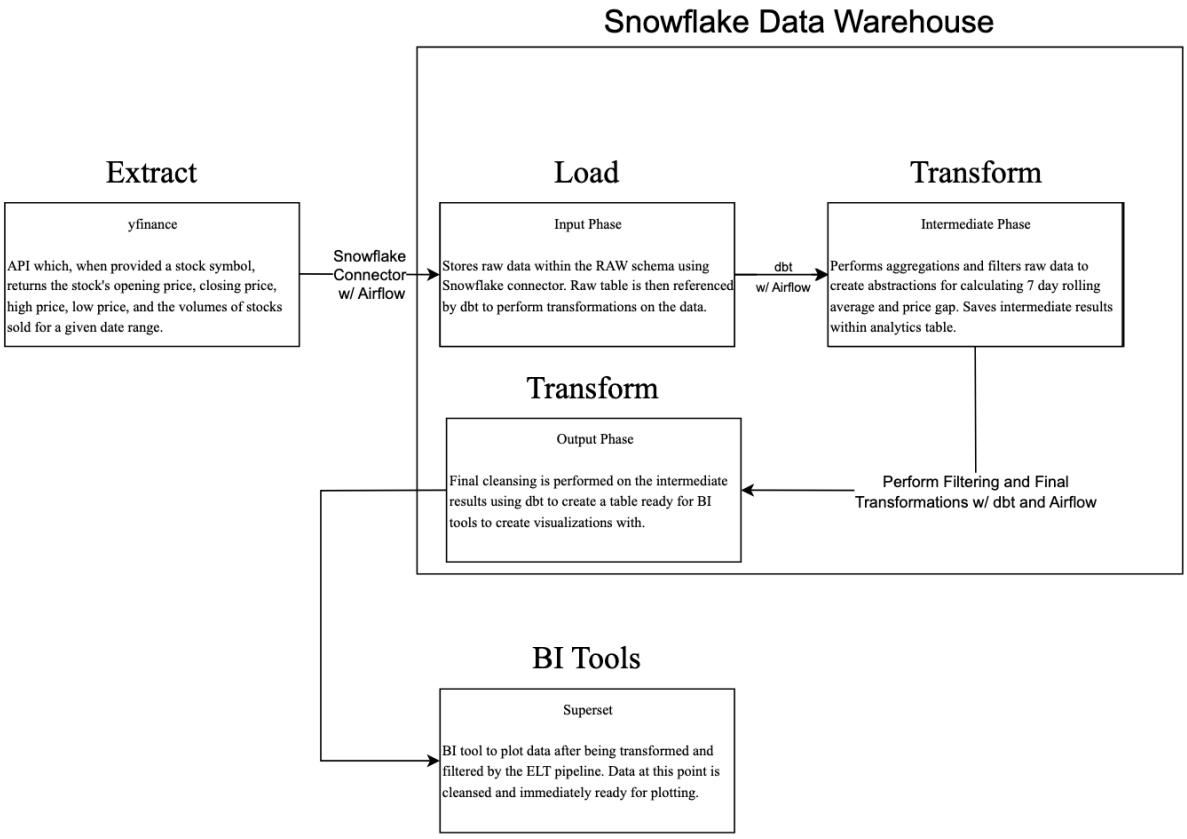

Fig. 1. System Design Diagram

III. Functional Analysis

A. Connecting to Snowflake

The first step in building the automated ELT pipeline is setting up a secure connection to Snowflake. We use Apache Airflow's Variables and its Snowflake provider to store credentials and ensure seamless, hands-free authentication during DAG execution.

```python
def return_snowflake_conn():

    # Initialize the SnowflakeHook
    hook = SnowflakeHook(snowflake_conn_id='snowflake_conn')

    # Execute the query and fetch results
    conn = hook.get_conn()
    return conn.cursor()
```

Fig. 2. Using SnowflakeHook to Connect to Snowflake

B. Obtaining and Loading Raw Data

With the connection to Snowflake established, we can now load raw stock data directly into the data warehouse from our Airflow DAGs. This data is sourced from the yfinance API and includes key stock metrics such as opening price, closing price, high, low, volume, and date. Rather than applying transformations before loading, we store the dataset in its original form to preserve all available information for downstream processing. To ensure each record is unique and easy to track, we set the combination of symbol and date as the primary key for the raw table in Snowflake. This helps maintain data integrity and prevents duplicate entries for the same stock on the same day.

While it's common in some pipelines to remove unnecessary columns early on, we chose to retain all columns in the raw dataset. This decision supports multiple types of analysis—including rolling averages and price gap calculations—which rely on different subsets of the data. Keeping everything ensures we don't accidentally exclude important information. It also provides long-term flexibility: if we later decide to build regression models or perform deeper statistical analysis, the full dataset will be available. Finally, it improves data lineage and debugging, allowing us to trace issues back through the pipeline and re-run transformations on the original data if needed.

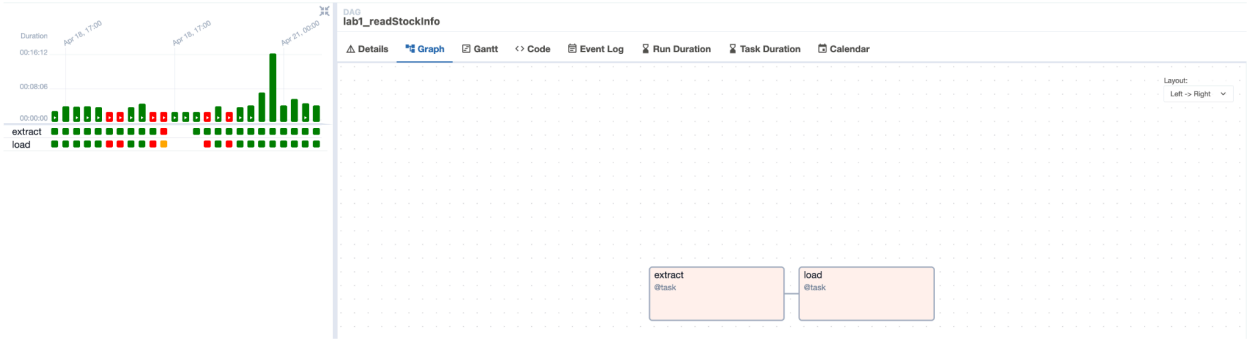

Fig. 3. Apache Airflow Web UI for Data Extraction & Loading DAG

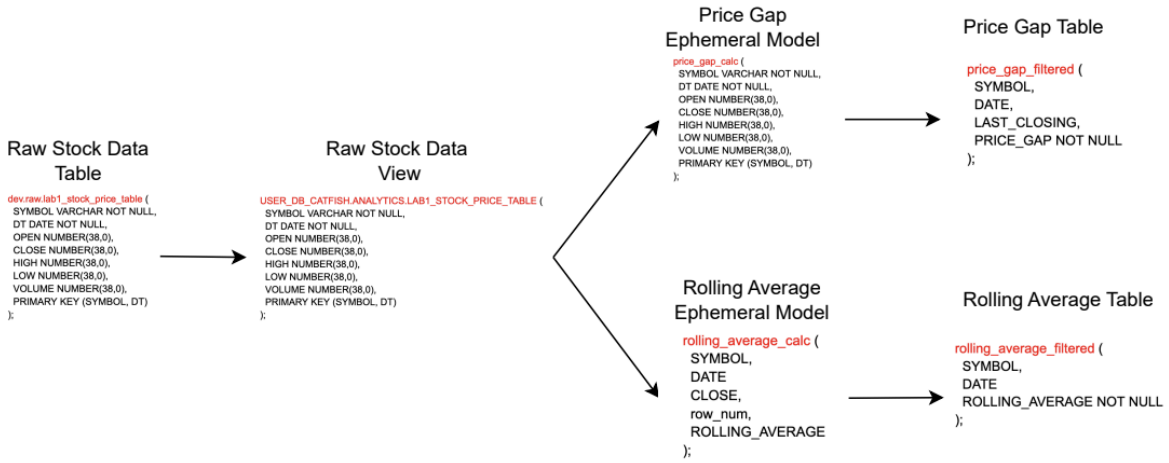

Fig. 4. Schema of ELT Pipeline

C. Transforming Raw Data with dbt

To perform SQL transformations on the raw data stored in the warehouse, we use dbt in combination with Apache Airflow to automate and orchestrate the process. With dbt, we not only apply SQL logic to add abstracted rows and filters, but we also implement data quality checks to ensure the reliability of the pipeline.

These checks are defined in schema.yml files and are applied to our dbt models. In this project, we validate that all fields in the raw data are non-null, as the yfinance API only returns values on active trading days. This ensures that if data is present, it is complete and meaningful for downstream analysis.

In addition to these validations, dbt handles the transformations required to calculate both the price gap and rolling average. These transformations follow a three-phase structure: input/raw, intermediate, and output.

```yaml
name: 'stock_elt_project'
version: '1.0'
profile: 'dbt'

# dbt now uses 'model-paths' instead of 'source-paths'
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

# Directories to clean with `dbt clean`
clean-targets:
  - "target"
  - "dbt_packages"

# Configuring model materializations
models:
  stock_elt_project:
    +materialized: view

    intermediate:
      +materialized: ephemeral

    output:
      +materialized: table
```

Fig. 5. dbt Materialization

1) *Input Phase:* In this phase, we define a centralized view of the raw stock data within Snowflake. Instead of directly loading the raw data into each transformation model, the view acts as a single source of truth that can be referenced across multiple dbt models. This approach promotes reusability, reduces code duplication, and simplifies maintenance. If the raw data structure changes in the future, updates only need to be made in one place.

Since views are non-materialized, they offer a lightweight way to access raw data without consuming additional storage. This makes them ideal for the input phase, where the goal is not to transform the data but to make it readily accessible for downstream processing. The consistent structure provided by the view also improves pipeline reliability, ensuring that all transformations begin with the same validated dataset.

```sql
SELECT *
FROM {{ source('raw', 'raw_stock_data') }}
```

Fig. 6. Creating View from Raw Data

2) *Intermediate Phase*: As for the intermediate phase, ephemeral models will be used in order to add transformed columns to calculate later abstractions. In this case, we will be creating two files which perform SQL queries on the data. The first will be in charge of adding the previous row's closing data to the current row in order to calculate the price gap, and the second will be in charge of calculating the rolling average of the stock's closing price over the past six days.

a) Price Gap Calculation: In order to calculate the price gap, for each date, the first step will be to filter out only the necessary rows from the raw data. This ensures that the final result will only have relevant information to the price gap, and the final step will only be the difference between the opening price and the previous day's closing price. Therefore, we will filter the results of the raw data to only return the date, stock symbol, opening price, then use the lag function to obtain the previous day's closing price.

```
SELECT SYMBOL,
       DT AS DATE,
       OPEN,
       LAG(CLOSE, 1) OVER (PARTITION BY SYMBOL ORDER BY SYMBOL, DT) as LAST_CLOSING
FROM {{ ref('lab1_stock_price_table') }}
ORDER BY SYMBOL, DATE
```

Fig. 7. SQL Query to Prepare Data for Price Gap Calculation

b) Rolling Average Calculation: For rolling averages, we compute the average of the stock's closing price over a 7-day window (current day plus six prior days). We also assign a row number to each entry. This row number helps us filter out incomplete windows—rows with fewer than 7 prior entries—ensuring that our averages are based on a full week of data. This step preserves data accuracy by avoiding averages distorted by shorter timeframes.

```
SELECT SYMBOL,
       DT AS "DATE",
       CLOSE,
       ROW_NUMBER() OVER (PARTITION BY SYMBOL ORDER BY SYMBOL, DATE) AS row_num,
       ROUND(AVG(CLOSE) OVER (PARTITION BY SYMBOL ORDER BY SYMBOL, DATE ROWS BETWEEN 6 PRECEDING AND CURRENT ROW), 2) AS ROLLING_AVERAGE
FROM {{ ref('lab1_stock_price_table') }}
```

Fig. 8. SQL Query to Calculate 7 Day Rolling Average

3) *Output Phase:* For the output phase, we will be creating filters for the rolling average, and performing a final calculation based on the results of the intermediate models in order to gain the proper insights from the data.

a) Price Gap Filter: Since the filtering had already occurred in the intermediate phase, the only step for finalizing the data is to calculate the difference between the opening price and the previous day's closing price. In order to do this, we simply subtract the opening price by the last day's closing price. The order of this step is important, as we want a negative price gap to show a decrease in price from day to day.

```
SELECT *,
       (OPEN - LAST_CLOSING) AS PRICE_GAP
FROM {{ ref('price_gap_calc') }}
WHERE LAST_CLOSING IS NOT NULL
```

Fig. 9. SQL Query for Calculating Price Gap

b) Rolling Average Calculation: The rolling average has already been calculated, so this phase only involves filtering out the date, stock symbol, and the 7-day rolling average. No further computations are needed—only relevant values are retained for downstream use.

```
SELECT SYMBOL,
       DATE,
       ROLLING_AVERAGE
FROM {{ ref('rolling_average_calc') }}
WHERE row_num >= 7
ORDER BY SYMBOL, DATE, row_num
```

Fig. 10. SQL Query for Filtering 7 Day Rolling Average

As with the extraction step, Apache Airflow handles running our SQL transformations automatically. This automation ensures that all abstracted data models stay synchronized with the latest raw data in the warehouse. As a result, the visualizations built on top of these models remain accurate, up to date, and reliable for ongoing analysis.

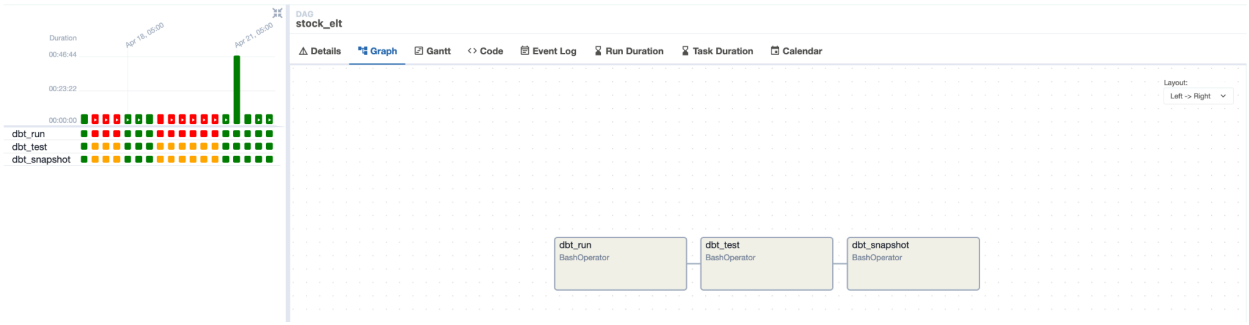

Fig. 11. Airflow Web UI Showing ELT DAG

D. Running dbt commands

In order for tracking changes to the source data (in case the API changes), we are able to run three different dbt commands to ensure data integrity is withheld. Those three commands are dbt run, dbt test, and dbt snapshot.

1) dbt run: The dbt run command triggers the execution of all SQL models defined within the project. In our stock pipeline, this includes building the ephemeral models for intermediate transformations and the final output tables used in Superset dashboards. When new raw data is added to the warehouse, dbt run recalculates metrics like the 7-day rolling average and daily price gap. This ensures the transformed tables reflect the most recent trading activity, keeping downstream reports reliable and up to date.

```
myvenvmatthewleffler@Matthews-MacBook-Air dbt % dbt run
08:34:19  Running with dbt=1.9.4
08:34:20  Registered adapter: snowflake=1.8.0
08:34:20  Found 5 models, 1 snapshot, 9 data tests, 2 sources, 462 macros
08:34:20
08:34:20  Concurrency: 1 threads (target='dev')
08:34:20
08:34:22  1 of 3 START sql view model analytics.lab1_stock_price_table ..... [RUN]
08:34:23  1 of 3 OK created sql view model analytics.lab1_stock_price_table ..... [SUCCESS 1 in 0.98s]
08:34:23  2 of 3 START sql table model analytics.price_gap_filtered ..... [RUN]
08:34:25  2 of 3 OK created sql table model analytics.price_gap_filtered ..... [SUCCESS 1 in 1.49s]
08:34:25  3 of 3 START sql table model analytics.rolling_avg_filtered ..... [RUN]
08:34:26  3 of 3 OK created sql table model analytics.rolling_avg_filtered ..... [SUCCESS 1 in 1.41s]
08:34:26
08:34:26  Finished running 2 table models, 1 view model in 0 hours 0 minutes and 6.10 seconds (6.10s).
08:34:26
08:34:26  Completed successfully
08:34:26
08:34:26  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
```

Fig. 12. Running dbt run command

2) dbt test: dbt test serves as a quality control mechanism in our stock analytics pipeline. By applying not_null tests to key columns in both raw and derived tables, we can detect missing data early in the process. This is especially important for calculated metrics like PRICE_GAP and ROLLING_AVERAGE, where missing values would break trend analyses or mislead visualizations. If any test fails, it acts as a signal that either the upstream data source has changed or the transformation logic requires adjustment.

```
myvenvmatthewleffler@Matthews-MacBook-Air dbt % dbt test
08:37:45  Running with dbt=1.9.4
08:37:45  Registered adapter: snowflake=1.8.0
08:37:45  Found 5 models, 1 snapshot, 9 data tests, 2 sources, 462 macros
08:37:45
08:37:45  Concurrency: 1 threads (target='dev')
08:37:45
08:37:47  1 of 9 START test not_null_lab1_stock_price_table_CLOSE ..... [RUN]
08:37:47  1 of 9 PASS not_null_lab1_stock_price_table_CLOSE ..... [PASS in 0.76s]
08:37:47  2 of 9 START test not_null_lab1_stock_price_table_DT ..... [RUN]
08:37:48  2 of 9 PASS not_null_lab1_stock_price_table_DT ..... [PASS in 0.69s]
08:37:48  3 of 9 START test not_null_lab1_stock_price_table_HIGH ..... [RUN]
08:37:49  3 of 9 PASS not_null_lab1_stock_price_table_HIGH ..... [PASS in 0.68s]
08:37:49  4 of 9 START test not_null_lab1_stock_price_table_LOW ..... [RUN]
08:37:49  4 of 9 PASS not_null_lab1_stock_price_table_LOW ..... [PASS in 0.65s]
08:37:49  5 of 9 START test not_null_lab1_stock_price_table_OPEN ..... [RUN]
08:37:50  5 of 9 PASS not_null_lab1_stock_price_table_OPEN ..... [PASS in 0.59s]
08:37:50  6 of 9 START test not_null_lab1_stock_price_table_SYMBOL ..... [RUN]
08:37:51  6 of 9 PASS not_null_lab1_stock_price_table_SYMBOL ..... [PASS in 0.86s]
08:37:51  7 of 9 START test not_null_lab1_stock_price_table_VOLUME ..... [RUN]
08:37:52  7 of 9 PASS not_null_lab1_stock_price_table_VOLUME ..... [PASS in 0.63s]
08:37:52  8 of 9 START test not_null_price_gap_filtered_PRICE_GAP ..... [RUN]
08:37:52  8 of 9 PASS not_null_price_gap_filtered_PRICE_GAP ..... [PASS in 0.86s]
08:37:52  9 of 9 START test not_null_rolling_avg_filtered_ROLLING_AVERAGE ..... [RUN]
08:37:53  9 of 9 PASS not_null_rolling_avg_filtered_ROLLING_AVERAGE ..... [PASS in 0.81s]
08:37:53
08:37:53  Finished running 9 data tests in 0 hours 0 minutes and 8.05 seconds (8.05s).
08:37:53
08:37:53  Completed successfully
08:37:53
08:37:53  Done. PASS=9 WARN=0 ERROR=0 SKIP=0 TOTAL=9
```

Fig. 13. Running dbt test command

3) dbt snapshot: In this pipeline, dbt snapshot creates a historical audit trail of the raw stock data by recording changes to any column in lab1_stock_price_table. With the check strategy and check_cols='all', dbt detects changes across every field—such as shifts in closing prices or volume. This is particularly useful for financial datasets, where values might be adjusted retroactively. By snapshotting against the SYMBOL and DT composite key, we ensure that we maintain a reliable timeline of how stock data evolved each day.

```
myvenvmatthewleffler@Matthews-MacBook-Air dbt % dbt snapshot
08:45:08  Running with dbt=1.9.4
08:45:08  Registered adapter: snowflake=1.8.0
08:45:08  Found 5 models, 1 snapshot, 9 data tests, 2 sources, 462 macros
08:45:08
08:45:08  Concurrency: 1 threads (target='dev')
08:45:08
08:45:10  1 of 1 START snapshot snapshot.snapshot_raw_stock_price ..... [RUN]
08:45:14  1 of 1 OK snapshotted snapshot.snapshot_raw_stock_price ..... [SUCCESS 0 in 4.00s]
08:45:14
08:45:14  Finished running 1 snapshot in 0 hours 0 minutes and 5.95 seconds (5.95s).
08:45:14
08:45:14  Completed successfully
08:45:14
08:45:14  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
```

Fig. 14. Running dbt snapshot command

E. Creating Visualizations

With the Snowflake tables containing the Rolling Average and Price Gap metrics prepared, we proceed to visualize these key indicators using Superset. A database connection is first established in Superset using Snowflake credentials, allowing seamless access to the warehouse. The two metric tables are then added as datasets within Superset. From these datasets, we create two separate charts—one for the rolling average and one for the daily price gap. Finally, both charts are placed on a shared dashboard, enabling side-by-side comparison and easy interpretation of market trends.

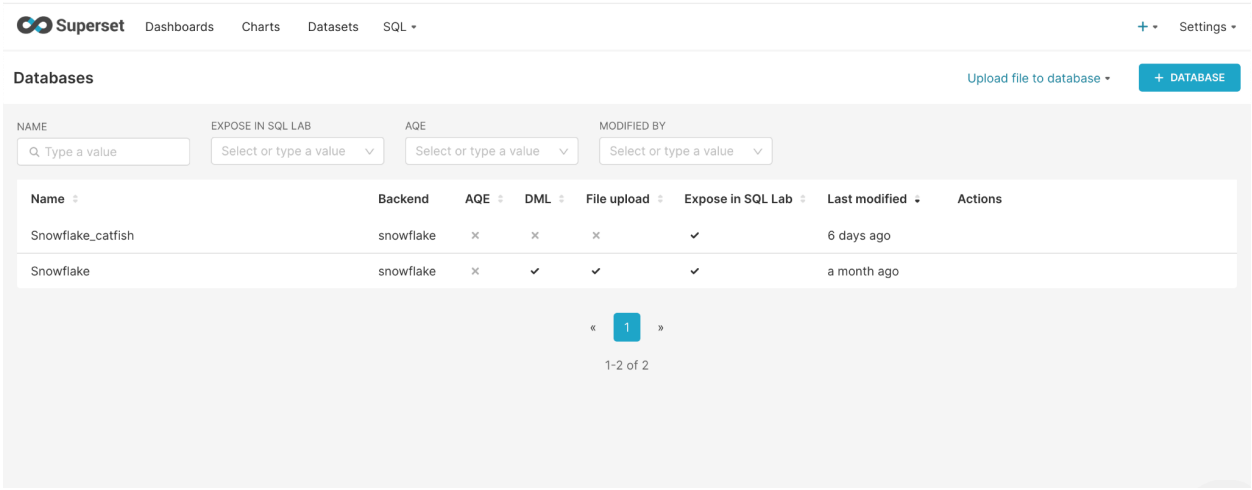

Fig. 15. Superset Dataset Connection to Snowflake

4) Rolling Average: The Snowflake table containing the rolling average metric includes three key fields: date, rolling average, and symbol. In Superset, we visualize this data using a line chart, with date on the x-axis, rolling average on the y-axis, and symbol set as the dimension. This configuration generates a separate line for each stock symbol—such as AAPL and FIVE—clearly distinguished by color. Since the dataset currently spans a 180-day period, we do not apply a date range filter. However, as the data continues to grow, it may become necessary to add a filter either directly within the Snowflake table or within the Superset chart to maintain readability and performance.

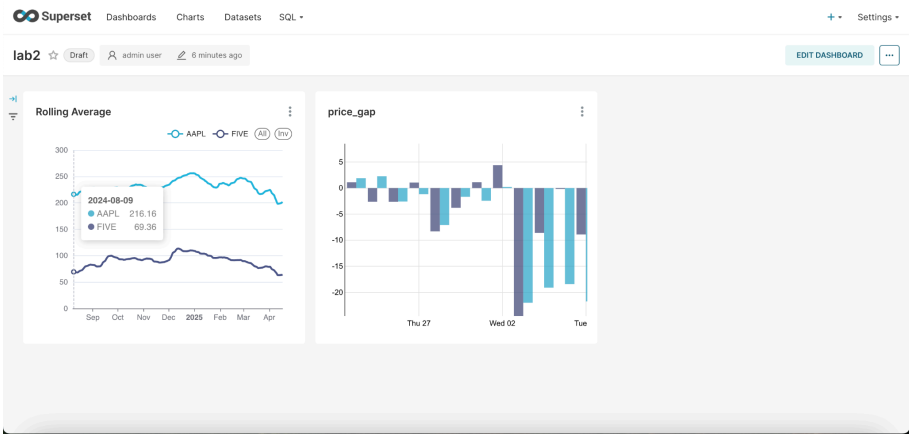

Fig. 16. Line Chart for Rolling Average in Dashboard

5) Price Gap: The Snowflake table for the price gap metric includes the fields date, price gap, and symbol. This data is visualized in Superset using a time-series bar chart, with date on the x-axis, price gap on the y-axis, and symbol as the dimension. The bar chart format was chosen because it effectively conveys both the direction (positive or negative) and magnitude of daily price changes, offering a quick snapshot of market sentiment and momentum for each stock. To maintain clarity, we apply a date filter set to display only the past week, as the bar chart is less effective for larger date ranges. However, this filter can be adjusted to show broader periods—such as the past month, quarter, or year—depending on the analytical needs.

Fig. 17. Time Series Bar Chart for Price Gap in Dashboard (Last Week)

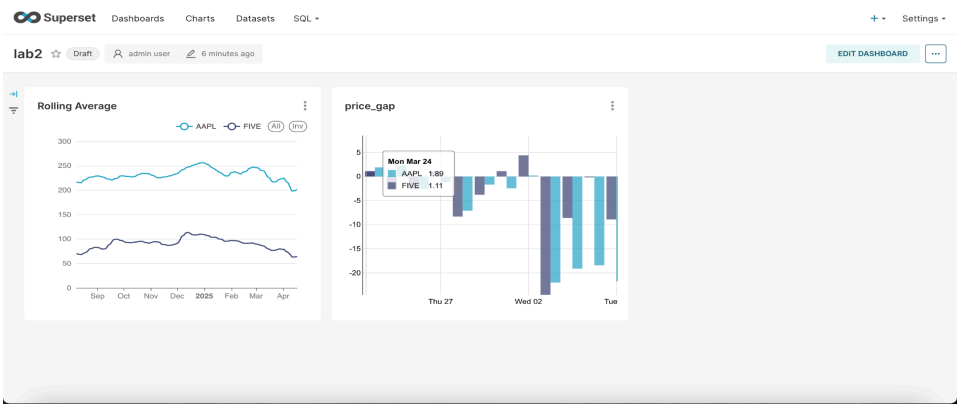

Fig. 18. Time Series Bar Chart for Price Gap in Dashboard (Last Month)

IV. Conclusion

Through this project, we successfully designed and implemented an automated ELT pipeline for stock data analysis using modern data tools such as Apache Airflow, Snowflake, dbt, and Superset. By focusing on rolling averages and price gaps—two metrics that provide both trend-based and sentiment-based insights—we demonstrated how raw financial data can be transformed into meaningful visualizations that support more informed trading decisions. Each tool contributed towards the whole data pipeline process. Snowflake provided a single source of truth with high scalability and integration with other tools like dbt. dbt transformations ensured transparency and data quality. Visualization with Superset brought clarity to the trends and market sentiments reflected in the data, enabling rapid interpretation and action. Finally airflow was able to automate this whole data pipeline process with logs for better recovery and backfill functionality. With this data pipeline, data-driven decision-making in financial markets can be made easier and future improvements for it can be modularized since each tool handles different parts of the whole data pipeline.

By analyzing the visualizations created from the Snowflake tables in Superset, we were able to identify both stock-specific trends and broader market patterns. While a sharp drop in a single stock—such as AAPL's recent decline—may appear as an isolated event, consistent downward movements across the rolling average and price gap charts suggest the presence of larger market forces, such as tariffs or economic shifts. The ELT pipeline not only

streamlines the delivery of data to the analytics team but also simplifies the creation of visualizations for business users, effectively bridging the gap between technical processes and strategic decision-making.