Data 226 Lab 1 Report

Matthew Leffler, Daniel Kim

Department of Data Analytics, San Jose State University

GitHub Link: https://github.com/matthewleffler1/DATA226_Lab1

I. Problem Statement:

The world of stock trading is an inherently volatile space where market conditions fluctuate rapidly. This unpredictability presents a challenge to investors who strive to forecast daily market movements to optimize their investment strategies. Leveraging machine learning for forecasting can significantly enhance predictability by analyzing historical stock data to project future closing prices.

The development of a data pipeline for this purpose addresses a critical need for continuous and automatic data acquisition. Given that stock prices are constantly changing, a data pipeline enables the systematic collection of stock information without manual intervention. This automation not only ensures that data used for forecasting is current, but also increases the efficiency and accuracy of these predictions.

Therefore, the purpose of this project is to implement a data pipeline that supports real-time data updates and facilitates more reliable stock market forecasts. This approach not only aids investors in making informed decisions but also contributes to a more structured method of navigating stock market volatility.

II. Solution Requirements

To build a robust data pipeline for stock prediction, our workflow is divided into three main stages, each requiring different tools.

A. Data Extraction and Transformation

The first stage involves retrieving and transforming stock market data. For this, we will utilize the yfinance API to collect daily stock information. Python will be used to extract this data in the form of a Pandas DataFrame and perform any necessary transformations before storing it. This approach ensures that the data is clean and structured before being stored in a database.

B. Data Storage and Machine Learning Implementation

The processed data will be stored in Snowflake, which is well-suited for structured, readily accessible datasets. Unlike traditional databases, Snowflake efficiently handles analytical workloads and does not require storing large raw data files, making it ideal for our use case. Additionally, to securely manage database credentials and API keys, we will be using Apache Airflow to store environment variables. This allows us to protect our account credentials when calling the Snowflake connector and ensures we can use the variables safely throughout the program.

C. Automation of Data Pipeline

Running the data pipeline manually daily is inefficient. To automate this process, we will leverage Apache Airflow, which will schedule and manage the data ingestion workflow. Airflow will execute Python scripts to extract data, transform it, and load it into Snowflake via the Snowflake connector. SQL commands will be executed on Snowflake to ensure that the database is updated daily, providing the latest stock data for prediction models.

By integrating Python, Snowflake, and Airflow, this solution ensures a scalable, secure, and automated data pipeline for stock prediction.

III. Functional Analysis:

A. Connecting to Snowflake

To establish a functional data pipeline, a secure and automated connection to Snowflake is required. This is achieved using Apache Airflow's Variables and the snowflake-connector library, which securely store authentication credentials within Airflow's metadata database and retrieve them dynamically during execution. This approach enhances security by avoiding hardcoded credentials while ensuring efficiency by automating database authentication. The connection serves as the foundation for the pipeline, enabling seamless data extraction, transformation, storage, and automation as part of the overall data ingestion and processing workflow.

```python
def return_snowflake_conn():

    user_id = Variable.get('snowflake_userid')
    password = Variable.get('snowflake_password')
    account = Variable.get('snowflake_account')

    # Establish a connection to Snowflake
    conn = snowflake.connector.connect(
        user=user_id,
        password=password,
        account=account,  # Example: 'xyz12345.us-east-1'
        warehouse='DHK_WH',
        database='dev'
    )
    # Create a cursor object
    return conn.cursor()
```

Fig. 1. Using Snowflake Connector to Connect to Snowflake

B. Obtaining and Transforming Data

With the Snowflake connector infrastructure in place, we can now proceed with the extraction and transformation phases of our ETL pipeline. To obtain stock data, we will utilize the yfinance API to retrieve historical stock data for the past 180 days. The yfinance library is particularly advantageous for data analysis, as it seamlessly integrates with Pandas, returning stock data in a DataFrame format. This is beneficial since SQL operates most efficiently with table-like structures. Additionally, yfinance offers flexible parameters that allow for easy

customization of the data retrieval period. By setting the period parameter to 180 days, we can efficiently extract and analyze stock performance over the desired timeframe.

```
yf.download("FIVE", period='180d')
```

```
[**********100%**********]  1 of 1 completed
```

Price	Close	High	Low	Open	Volume
Ticker	FIVE	FIVE	FIVE	FIVE	FIVE
Date					
2024-06-11	120.320000	121.139999	113.779999	115.370003	1761400
2024-06-12	116.570000	125.080002	116.349998	122.400002	1629300
2024-06-13	113.750000	117.169998	112.459999	117.169998	1274000
2024-06-14	112.879997	114.160004	111.089996	112.910004	2400300
2024-06-17	114.309998	116.279999	112.029999	112.739998	1135700
...	...	...	...	...	...
2025-02-24	87.529999	89.029999	84.449997	86.669998	1309700
2025-02-25	89.730003	91.419998	87.809998	87.809998	1682200
2025-02-26	89.220001	90.930000	88.769997	90.110001	1055200
2025-02-27	87.410004	89.250000	84.699997	87.389999	1475700
2025-02-28	86.889999	87.480003	84.800003	86.370003	1284200

180 rows × 5 columns

Fig. 2. Pandas DataFrame Returned Using yfinance

Although the raw data structure is nearly suitable for analysis, several adjustments are required before loading it into Snowflake. As shown in Figure 2, the yfinance download function returns a multi-indexed DataFrame, which could complicate analysis when imported into Snowflake. While it is essential to retain information about which stock each row corresponds to, it is more practical to store this detail as a separate column within the DataFrame. This ensures compatibility when importing data into a single Snowflake database, where information for multiple stocks will be stored together. SQL's capabilities allow analysts to efficiently group and

analyze the data for statistical modeling and forecasting as needed. Figure 3 illustrates the transformed DataFrame after these adjustments, demonstrating that the data is now properly structured and ready for storage in Snowflake.

```
five = yf.download(symbol, period='180d')
five.droplevel(0, axis=1)
five.columns = ['Open', 'Close', 'High', 'Low', 'Volume']
five = five.reset_index()
five = five[['Date', 'Open', 'Close', 'High', 'Low', 'Volume']]
five.insert(0, 'Symbol', symbol)
```

```
[**********100%**********]  1 of 1 completed
```

```
five
```

	Symbol	Date	Open	Close	High	Low	Volume
0	FIVE	2024-06-11	120.320000	121.139999	113.779999	115.370003	1761400
1	FIVE	2024-06-12	116.570000	125.080002	116.349998	122.400002	1629300
2	FIVE	2024-06-13	113.750000	117.169998	112.459999	117.169998	1274000
3	FIVE	2024-06-14	112.879997	114.160004	111.089996	112.910004	2400300
4	FIVE	2024-06-17	114.309998	116.279999	112.029999	112.739998	1135700
...	...	...	...	...	...	...	...
175	FIVE	2025-02-24	87.529999	89.029999	84.449997	86.669998	1309700
176	FIVE	2025-02-25	89.730003	91.419998	87.809998	87.809998	1682200
177	FIVE	2025-02-26	89.220001	90.930000	88.769997	90.110001	1055200
178	FIVE	2025-02-27	87.410004	89.250000	84.699997	87.389999	1475700
179	FIVE	2025-02-28	86.889999	87.480003	84.800003	86.370003	1284200

180 rows × 7 columns

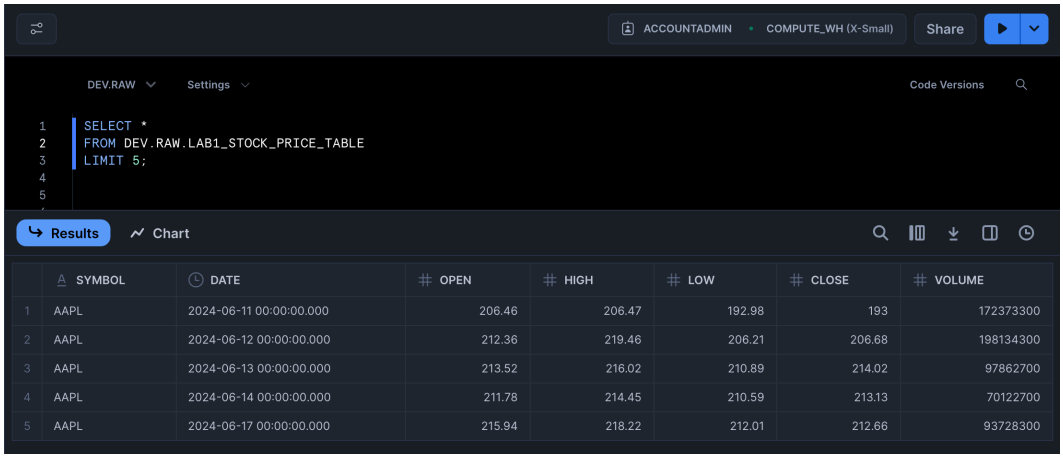
Fig. 3. Transforming DataFrame for Snowflake

C. Loading Data into Snowflake

With the tables properly formatted, the next step is to load the data into Snowflake. To establish a connection, we use either the snowflake.connector library or Apache Airflow's SnowflakeHook, enabling seamless interaction with the database. The Snowflake database will be used to store historical data, while a view will be created to hold predicted values. Once predictions are generated, the view will be merged with the historical data to ensure the dataset remains up to date. Using a view in this structure provides two key benefits: it allows for efficient filtering based on stock symbols and ensures that predictive data remains separate from historical records, maintaining the integrity of the original table. The merging process follows Snowflake's upsert (MERGE) method, ensuring data consistency by replacing existing values for matching keys while inserting new records. This approach preserves historical data while seamlessly integrating new predictions into the data warehouse.

To maintain data integrity, we will wrap the data loading process within SQL transactions, ensuring that all commands execute successfully before committing changes. If an error occurs at any point in the pipeline, the transaction will roll back the tables to their previous state, preventing corrupted or incomplete data from being inserted. This safeguards the final table from skewed results caused by programming or API failures. Additionally,

when defining the table schema, we must ensure that the Date column is cast as a TIMESTAMP_NTZ type, as Snowflake's forecasting functions require datetime-formatted data for proper analysis.



	SYMBOL	DATE	OPEN	HIGH	LOW	CLOSE	VOLUME
1	AAPL	2024-06-11 00:00:00.000	206.46	206.47	192.98	193	172373300
2	AAPL	2024-06-12 00:00:00.000	212.36	219.46	206.21	206.68	198134300
3	AAPL	2024-06-13 00:00:00.000	213.52	216.02	210.89	214.02	97862700
4	AAPL	2024-06-14 00:00:00.000	211.78	214.45	210.59	213.13	70122700
5	AAPL	2024-06-17 00:00:00.000	215.94	218.22	212.01	212.66	93728300

Fig. 4. Data Loaded into Snowflake

D. Forecasting Stock Prices with Machine Learning

With the data successfully loaded into Snowflake, the next step is to create a view to structure the dataset for forecasting. In Snowflake, we can leverage the SERIES_COLNAME parameter to automatically separate data based on the stock symbol, allowing us to handle multiple stocks within the same dataset efficiently. Given this capability, our view only needs to isolate key columns—SYMBOL, DATE, and CLOSE—which are essential for performing time-series forecasting.

Once the view is created, we can apply Snowflake's built-in forecasting function to generate stock price predictions. By specifying the target column (CLOSE) and timestamp (DATE), Snowflake automatically detects the data intervals within the dataset, ensuring accurate time-series analysis. Furthermore, Snowflake applies multiple machine learning models to the data and automatically selects the most effective one based on performance metrics.

```python
@task
def train(cur, train_input_table, train_view, forecast_function_name):
    """
     - Create a view with training related columns
     - Create a model with the view above
    """

    create_view_sql = f"""CREATE OR REPLACE VIEW {train_view} AS SELECT
        symbol,date, close
        FROM {train_input_table};"""

    create_model_sql = f"""CREATE OR REPLACE SNOWFLAKE.ML.FORECAST {forecast_function_name} (
        INPUT_DATA => SYSTEM$REFERENCE('VIEW', '{train_view}'),
        SERIES_COLNAME => 'symbol',
        TIMESTAMP_COLNAME => 'date',
        TARGET_COLNAME => 'close',
        CONFIG_OBJECT => {{ 'ON_ERROR': 'SKIP' }}
    );"""

    try:
        cur.execute(create_view_sql)
        cur.execute(create_model_sql)
        # Inspect the accuracy metrics of your model.
        cur.execute(f"CALL {forecast_function_name}!SHOW_EVALUATION_METRICS();")

    except Exception as e:
        print(e)
        raise
```

Fig. 5. Forecasting Function using Connector

E. Automating Pipeline and Predictions

Now that we have the code and programs setup to make predictions on historical stock prices, we need to use Apache Airflow to automate the process of loading data into the table. In order to do so, we need to port our files into DAGS for Apache to run automatically, so we don't need to manually run the program on a daily basis. In order to do so, we simply convert our functions into tasks by adding a decorator above the function declaration, and declare a DAG in which we are able to specify the order of operations in which the program executes. In our case, we will have two separate files. The first will be in charge of the ETL proess, and the second DAG will be in charge of creating predictions, and merging the predictions into the stock price database.

1) *ETL DAG*

In the ETL DAG, the program is scheduled to run once per day, as the API updates the stock's closing price on a daily basis. Since the process involves retrieving data via an API call, transforming the data, and then storing the results in Snowflake, the tasks must be executed sequentially in this order for each stock.

2) *Stock Prediction DAG*

For the closing price prediction program, a separate DAG will be created to store the predicted values from the model, and merge those into the historical data table. The workflow consists of a task that creates a view to ensure the original table remains unchanged during predictions while allowing for filtering of the stock being analyzed, followed by a task that performs the prediction based on the closing stock price from the selected stock in the appropriate view. Since these operations must be executed sequentially, Apache Airflow's bitshift operator (>>) will be used to ensure that model training runs first, followed by the prediction task, which depends on the model's results.

After Apache Airflow's Web UI performs a refresh, we are able to see the currently running DAGs on Airflow to ensure the DAGs loaded successfully into Airflow, and are ready for automation.

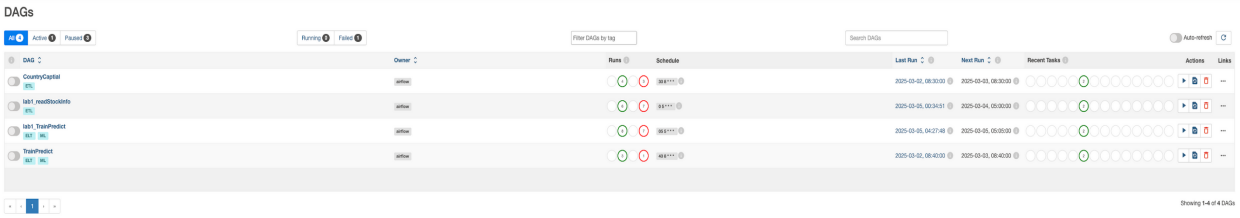


Fig. 6. Apache Airflow's Web UI

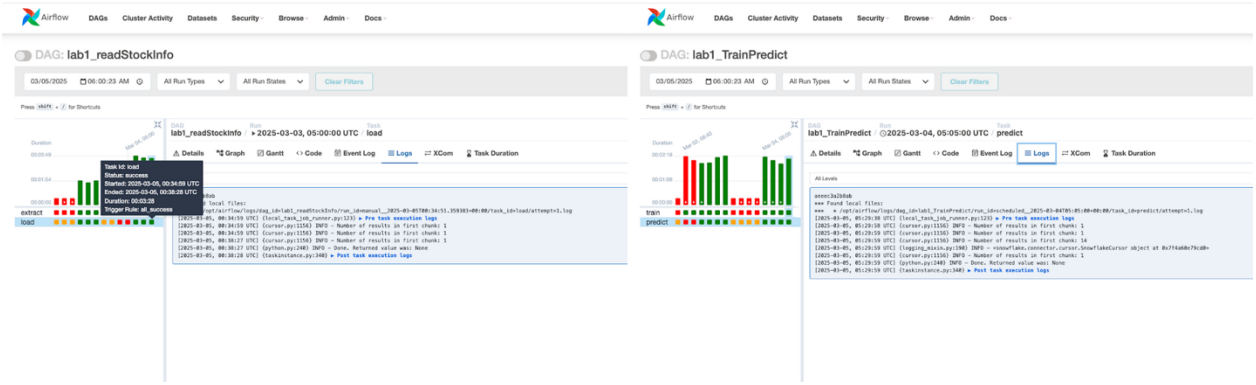


Fig. 7. Apache DAG Logs

Now that the tasks have been loaded into Apache Airflow, the system will automatically execute them once the specified cron condition is met. If the process completes successfully, the resulting table can be viewed. After reviewing the logs, we confirmed a successful run and were able to observe the program's execution results.

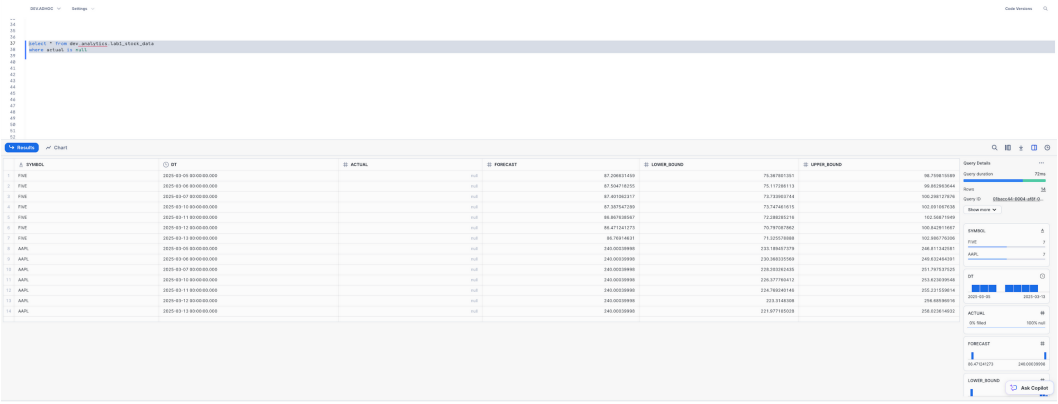


Fig. 8. Table Showing Forecasted Values

Based on the SQL execution results, the query has successfully generated predictions for the closing stock price over the upcoming days and seamlessly merged them into the target table alongside historical values. This analysis allows for quick assessments of potential buying opportunities. For instance, as illustrated in Figure 7, the model forecasts a slight increase in Five Below's stock price on the second predicted day, followed by a decline until March 13, when the price is expected to recover. Additionally, the model anticipates no significant movement in Apple's stock price over the forecasted period. However, the widening gap between the lower and upper bounds—where the lower bound decreases while the upper bound rises—indicates increased price volatility.