Stock Price Prediction Analytics using Snowflake & Airflow

Moksh Aggarwal, Shivam Mahendru
Department of Applied Data Science
San Jose State University
Email: moksh.aggarwal@sjsu.edu, shivam.mahendru@sjsu.edu

Abstract: This project focuses on forecasting stock prices using historical data and technical indicators, with an emphasis on building a scalable and automated data pipeline. The primary goal is to predict short-term and long-term stock price movements through machine learning models. In addition to stock price forecasting, the project involves the creation of a robust pipeline for automating data extraction, transformation, and loading (ETL) using Apache Airflow for workflow orchestration and Docker for containerization. This approach ensures efficient model deployment and reproducibility, providing a streamlined framework for integrating predictive analytics in real-world financial applications.

Index Terms

Stock Price Prediction, Machine Learning, Apache Airflow, Snowflake, Docker, ETL Pipeline

I. Introduction

Stock price prediction is a key challenge in the field of financial analysis, with significant implications for investors and traders. This project, titled "Building a Stock Price Prediction Analytics using Snowflake & Airflow," is a group effort conducted as part of the Data226 course at San Jose State University. It leverages the yfinance API to access comprehensive stock market data, including open, high, low, close, and volume information. The dataset enables various financial analyses, such as stock price prediction, trend analysis, technical indicator analysis, portfolio optimization, volatility analysis, volume analysis, and sentiment analysis. For this lab, we focus on implementing a stock price prediction system to forecast the stock prices of selected companies for the next 7 days using the last 180 days of historical data.

II. Problem Statement

The stock price prediction process involves multiple complex steps, from data collection and preprocessing to model deployment and analysis. These processes can be time-consuming, error-prone, and difficult to scale when performed manually. The challenge is to create a **scalable and maintainable automated pipeline** that can efficiently handle the end-to-end workflow of stock price prediction. A database and data pipelines are essential to store and manage large volumes of historical stock data, automate data updates, and integrate machine learning models for forecasting. This project addresses the following key problems:

- **Automation of ETL Processes**: Manual extraction, transformation, and loading (ETL) of stock data leads to inefficiencies. Automation is needed to streamline the workflow.
- **Scalability**: The pipeline must handle large volumes of stock data and continuously update forecasts without performance degradation.
- **Reproducibility and Consistency**: The process, from data collection to model deployment, must be reproducible and consistent.
- **Efficient Workflow Orchestration**: Apache Airflow is required to orchestrate tasks, enabling scheduling, monitoring, and dependency management.
- **Containerization for Deployment**: Docker ensures portability, isolation, and scalability of the pipeline across different environments.

By addressing these issues, this project provides an efficient, reliable, and automated framework for predicting stock prices, reducing the time and effort required for financial analysis.

III. Solution Requirements

The design of the stock price prediction system hinges on a set of essential components and tools, each tailored to meet the project's objectives. The following outlines the key requirements for building this automated analytics framework:

- **Data Storage with Snowflake**: The system employs Snowflake as its central repository to house historical stock data sourced from external APIs and to manage machine learning models. This database supports data persistence, model training, and retrieval of predictive outputs.

- **Workflow Automation via Airflow**: Apache Airflow orchestrates the entire pipeline by scheduling and managing tasks such as data ingestion, transformation, and forecasting. It ensures that stock data is refreshed in Snowflake on a daily basis through automated workflows.
- **Stock Data Acquisition**: The system integrates popular Python library that is YFinance API, to fetch critical market data, including stock prices, trading volumes, and related metrics, forming the foundation for analysis.
- **Python Ecosystem**: Several Python libraries power the system's functionality:
  - `y-finance api`: Python module to fetch financial and market data.
  - `airflow.providers.snowflake`: Enables seamless connectivity with Snowflake.
  - `airflow.operators.trigger_dagrun`: TriggerDagRunOperator in Apache Airflow is used to trigger the execution of another DAG in sequence after completion of current DAG.
- **Supporting Dependencies**: The implementation relies on external packages, including:
  - `apache-airflow`: Core framework for task orchestration.
  - `snowflake-connector-python`: Provides Python-Snowflake integration.
  - `apache-airflow-providers-snowflake`: Enhances Airflow's Snowflake compatibility.

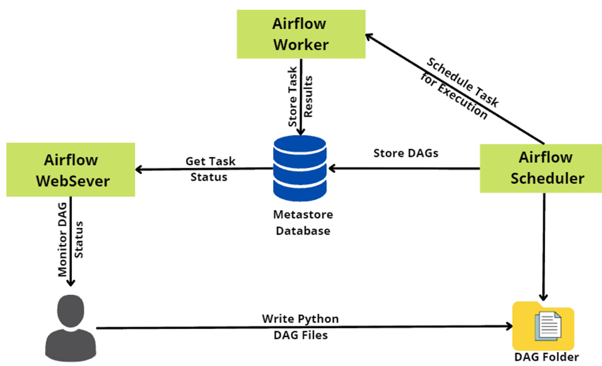


Fig. 1. Airflow Workflow Diagram

A. System Constraints

The system, while robust, operates within certain boundaries:

- **Unpredictable Market Dynamics**: External factors like geopolitical events or shifts in investor behavior can affect stock prices in ways that models based on historical data may not anticipate, potentially reducing prediction accuracy.
- **Data Retrieval Delays**: API constraints or Airflow scheduling intervals may introduce latency, preventing real-time data updates.
- **Model Assumptions**: The predictive models assume continuity in historical patterns, an assumption that may falter during extreme market volatility or novel conditions.

IV. Functional Analysis

A. Data Pipeline Components

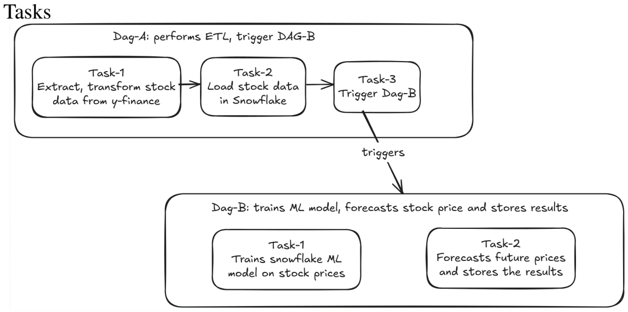
- **Data Ingestion**: Extract stock data using the yfinance API and store raw data in **Snowflake**.
- **Data Processing & Transformation**: Clean and format stock data, aggregating necessary fields for analysis.
- **Machine Learning Forecasting**: Train an ML model in Snowflake on historical stock prices to predict prices for the next **7 days or more**.
- **Data Storage & Querying**: Store raw and predicted data in **Snowflake tables** and use SQL queries for insights.
- **Automation with Airflow**: An **ETL pipeline** runs daily to update stock data, and a **forecasting pipeline** updates predictions using recent data, both managed via Airflow DAGs.

V. IMPLEMENTATION

Field	Type	Description
Stock Symbol	VARCHAR	Stock ticker symbol (AAPL and GOOG)
Date	DATE	Date of the stock data
Open	FLOAT	Opening price
Close	FLOAT	Closing price
Min	FLOAT	Minimum price
Max	FLOAT	Maximum price
Volume	INT	Volume of stocks traded

TABLE I
TABLE STRUCTURE IN SNOWFLAKE

Fig. 2. System Design of the DAGs and Tasks



VI. PYTHON CODE

The Python code defines the Airflow DAGs and tasks for ETL and forecasting, integrating with the python's y-finance module as a data source. Below are snippets of the DAG definitions:

Listing 1. DAG-1: Responsible for extracting stocks data from y-finance, transforming, loading it in Snowflake. This DAG further triggers a DAG for training ML model and making predictions on the loaded data.

```
File: https://github.com/mahendruShivam29/Airflow/blob/main/dags/extract_transform_load_stock_data.py

import yfinance as yf
from airflow import DAG
from datetime import datetime
from airflow.decorators import task
from airflow.providers.snowflake.hooks.snowflake import SnowflakeHook
from airflow.operators.trigger_dagrun import TriggerDagRunOperator

def return_snowflake_conn():
    hook = SnowflakeHook(snowflake_conn_id='snowflake_conn')
    conn = hook.get_conn()
    return conn.cursor()

@task
def extract_and_transform_last_180d_data_from_yfinance(cursor, symbols):
    """
        - Extract the last 180 days of Stock price data from Y-finance.
        - Perform Transformation, here itself. TimeStamp Object to Date. Round decimal to 4 places.
    """
    result = []
    for symb in symbols:
        stock = yf.Ticker(symb)
        df = stock.history(period='180d', interval='1d')

        for row in df.itertuples(index=True, name="Row"):
            record = {}
            # Transformation.
            record["date"] = row.Index.strftime('%Y-%m-%d')
            record["symbol"] = symb
            record["open"] = "{:.4f}".format(row.Open)
            record["high"] = "{:.4f}".format(row.High)
            record["low"] = "{:.4f}".format(row.Low)
            record["close"] = "{:.4f}".format(row.Close)
            record["volume"] = str(row.Volume)
            result.append(record)

    for r in result:
```

```
        print(r)
    return result

@task
def load_stock_data(cursor, stock_data, stock_data_table):
    """
        - Loads Stock Data in Snowflake using full refresh, try-except, commit and rollback.
    """
    try:
        cursor.execute("BEGIN;")
        create_stock_data_table_sql = f"""CREATE TABLE IF NOT EXISTS {stock_data_table} (
            symbol VARCHAR(10),
            date DATE,
            open FLOAT,
            high FLOAT,
            low FLOAT,
            close FLOAT,
            volume FLOAT,
            PRIMARY KEY (symbol, date)
        );"""
        cursor.execute(create_stock_data_table_sql)
        cursor.execute(f"DELETE FROM {stock_data_table}")

        for record in stock_data:
            symbol = record["symbol"]
            date = record["date"]
            open = record["open"]
            high = record["high"]
            low = record["low"]
            close = record["close"]
            volume = record["volume"]

            insert_record_statement = f"INSERT INTO {stock_data_table} (symbol, date, open, high, low, close, volume) VALUES (%s, %s, %s, %s, %s, %s, %s)"
            cursor.execute(insert_record_statement, (symbol, date, open, high, low, close, volume))
            print(f"INSERT INTO {stock_data_table} ({symbol}, {date}, {open}, {high}, {low}, {close}, {volume})")

        cursor.execute("COMMIT;")
    except Exception as e:
        cursor.execute("ROLLBACK;")
        print(e)
        raise e

with DAG(
    dag_id = 'extract_transform_load_stock_data',
    start_date = datetime(2025, 3, 3),
    schedule_interval = '@daily',
    catchup = False,
    tags = ['ETL']
) as dag:
    cursor = return_snowflake_conn()
    symbols = ['AAPL', 'GOOG']
    stock_data_table = "DEV.LAB.STOCK_DATA"
    stock_data = extract_and_transform_last_180d_data_from_yfinance(cursor, symbols)
    load_data = load_stock_data(cursor, stock_data, stock_data_table)
    trigger_train_predict = TriggerDagRunOperator(
        task_id = 'trigger_train_predict',
        trigger_dag_id = 'train_predict_stock_data',
        execution_date = '{{ ds }}',
        reset_dag_run = True
    )

    stock_data >> load_data >> trigger_train_predict
```

Listing 2. DAG-2: Responsible for training model in Snowflake, making stock price prediction for next 7 days and merging that with historical database.

```
File: https://github.com/mahendruShivam29/Airflow/blob/main/dags/train_predict_stock_data.py
from airflow import DAG
from airflow.decorators import task
from airflow.providers.snowflake.hooks.snowflake import SnowflakeHook
from datetime import datetime


def return_snowflake_conn():
    hook = SnowflakeHook(snowflake_conn_id='snowflake_conn')
    conn = hook.get_conn()
```

```
    return conn.cursor()

@task
def train(cursor, 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
        DATE, CLOSE, SYMBOL
    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:
        cursor.execute(create_view_sql)
        cursor.execute(create_model_sql)
        cursor.execute(f"CALL {forecast_function_name}!SHOW_EVALUATION_METRICS();")
    except Exception as e:
        print(e)
        raise e

@task
def predict(cursor, forecast_function_name, train_input_table, forecast_table, final_table):
    """
        - Generate predictions and store the results to a table named forecast table.
        - Union the predictions with historical data, then make a final table.
    """
    make_prediction_sql = f"""BEGIN
        CALL {forecast_function_name}!FORECAST(
            FORECASTING_PERIODS => 7,
            CONFIG_OBJECT => {{’prediction_interval’: 0.95}}
        );
        LET x := SQLID;
        CREATE OR REPLACE TABLE {forecast_table} AS SELECT * FROM TABLE(RESULT_SCAN(:x));
    END;"""

    create_final_table_sql = f"""CREATE OR REPLACE TABLE {final_table} AS
        SELECT SYMBOL, DATE, CLOSE AS actual, NULL as forecast, NULL as lower_bound, NULL as upper_bound
        FROM {train_input_table}
        UNION ALL
        SELECT replace(series, ’"’, ’’) as SYMBOL, ts as DATE, NULL as actual, forecast, lower_bound,
            upper_bound
        FROM {forecast_table};"""

    try:
        cursor.execute(make_prediction_sql)
        cursor.execute(create_final_table_sql)
    except Exception as e:
        print(e)
        raise e

with DAG(
    dag_id = ’train_predict_stock_data’,
    start_date = datetime(2025, 3, 5),
    catchup = False,
    tags = [’ML’, ’ELT’]
) as dag:
    cursor = return_snowflake_conn()

    train_input_table = "DEV.LAB.STOCK_DATA"
    train_view = "DEV.LAB.STOCK_DATA_VIEW"
    forecast_table = "DEV.LAB.STOCK_DATA_FORECAST"
    forecast_function_name = "DEV.LAB.PREDICT_"
    final_table = "DEV.LAB.MARKET_DATA"

    train(cursor, train_input_table, train_view, forecast_function_name) >> predict(cursor,
        forecast_function_name, train_input_table, forecast_table, final_table)
```

VII. SCREENSHOTS

Fig. 3. Airflow UI with the DAGs

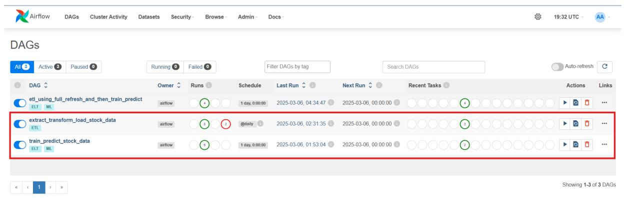


Fig. 4. DAG-1 Details, runs

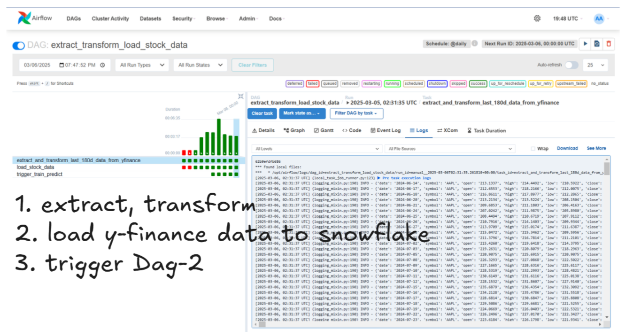


Fig. 5. DAG-1 Task dependency

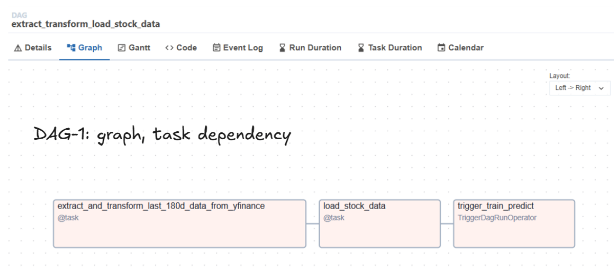


Fig. 6. DAG-2 Details, runs

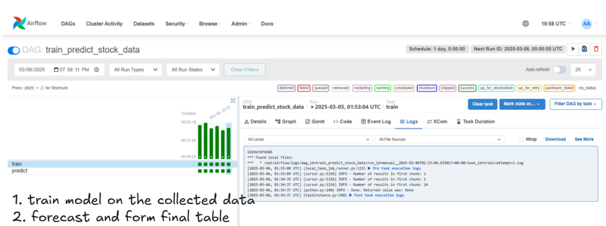


Fig. 7. DAG-2 Task dependency

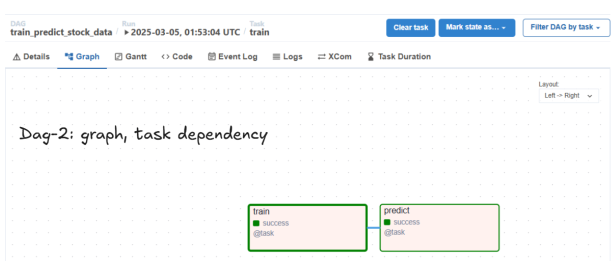
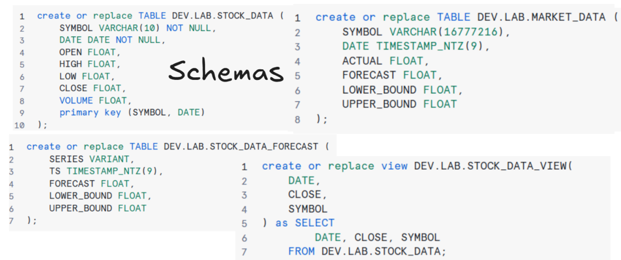


Fig. 8. DAGs Dependency

Fig. 9. Table Schema and Structure




```
create or replace TABLE DEV.LAB.STOCK_DATA (
    SYMBOL VARCHAR(10) NOT NULL,
    DATE DATE NOT NULL,
    OPEN FLOAT,
    HIGH FLOAT,
    LOW FLOAT,
    CLOSE FLOAT,
    VOLUME FLOAT,
    primary key (SYMBOL, DATE)
);
```

```
create or replace TABLE DEV.LAB.MARKET_DATA (
    SYMBOL VARCHAR(16777216),
    DATE TIMESTAMP_NTZ(9),
    ACTUAL FLOAT,
    FORECAST FLOAT,
    LOWER_BOUND FLOAT,
    UPPER_BOUND FLOAT
);
```

```
create or replace TABLE DEV.LAB.STOCK_DATA_FORECAST (
    SERIES VARIANT,
    TS TIMESTAMP_NTZ(9),
    FORECAST FLOAT,
    LOWER_BOUND FLOAT,
    UPPER_BOUND FLOAT
);
```

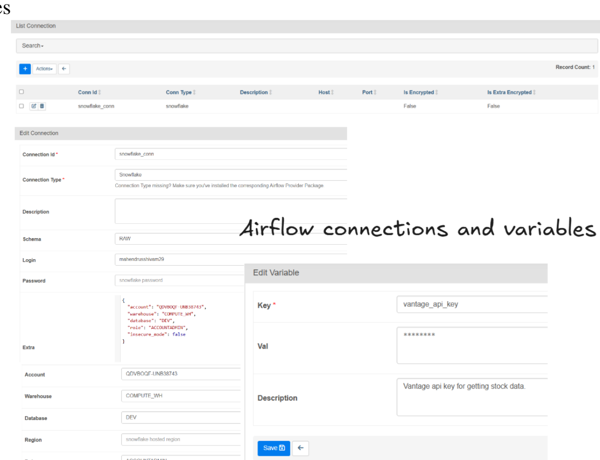
```
create or replace view DEV.LAB.STOCK_DATA_VIEW(
    DATE,
    CLOSE,
    SYMBOL
) as SELECT
        DATE, CLOSE, SYMBOL
    FROM DEV.LAB.STOCK_DATA;
```

Fig. 10. Table showing combined results

Combined Table

	SYMBOL	DATE	ACTUAL	FORECAST	LOWER_BOUND	UPPER_BOUND
1	AAPL	2025-03-14 00:00:00.000	null	235.322277348	219.009299471	252.104183735
2	AAPL	2025-03-13 00:00:00.000	null	235.373258934	219.281726649	250.927019976
3	AAPL	2025-03-12 00:00:00.000	null	235.468905614	220.889434243	250.730269899
4	AAPL	2025-03-11 00:00:00.000	null	235.578997961	222.301150113	249.252257028
5	AAPL	2025-03-10 00:00:00.000	null	235.400693799	222.990799435	247.691974633
6	AAPL	2025-03-07 00:00:00.000	null	235.518340448	224.699369437	246.412152783
7	AAPL	2025-03-06 00:00:00.000	null	235.603656932	226.136092389	244.731998972
8	AAPL	2025-03-05 00:00:00.000	235.74	null	null	null
9	AAPL	2025-03-04 00:00:00.000	235.93	null	null	null
10	AAPL	2025-03-03 00:00:00.000	238.03	null	null	null
11	AAPL	2025-02-28 00:00:00.000	241.84	null	null	null

Fig. 11. Airflow connections and variables



Link to the codes.
DAG-1: ETL, trigger DAG-2.
DAG-2: Train ML model and forecast future prices.

VIII. Conclusion

This project successfully developed an automated data pipeline for stock price prediction using Apache Airflow and Snowflake. By leveraging historical stock data and training machine learning model, we built a scalable solution to forecast stock prices for the next 7 days which could be extended based on the requirement. Docker ensured reproducibility and scalability, while Airflow provided robust workflow orchestration. The system supports financial decision-making by delivering actionable insights.

References

[1] yfinance API, https://pypi.org/project/yfinance/.
[2] Airflow, https://airflow.apache.org/docs/.