Building Building a Stock Price Prediction Analytics using Snowflake & Airflow

(Members: Sharan Patil, Mansi Deshmukh)

I. Problem Statement

The objective of this project was to design and develop an end-to-end financial data analytics system capable of automating the extraction, transformation, and loading (ETL) of real-world stock market data for analytical and forecasting purposes. The system aims to help analysts and investors monitor stock performance, trends, and future projections. The project integrates Python, Apache Airflow, and Snowflake to achieve a seamless workflow from data ingestion to forecasting.

Yahoo Finance (via the yfinance API) was selected as the data source to collect financial market data for selected companies. This system collects historical stock prices for companies such as Microsoft (MSFT) and Oracle (ORCL), covering the last 180 days. Snowflake was chosen as the central data warehouse to store and manage structured financial datasets efficiently. A database and automated data pipelines were essential for ensuring real-time data availability, accuracy, and automation.

II. Solution Requirements

The solution required an automated and scalable mechanism to collect financial data daily, process it, and store it in a structured format for further analytics. The main requirements included data reliability, scalability, and seamless integration between Python and Snowflake, orchestrated through Apache Airflow. The system needed to perform scheduled executions, ensure data consistency, and handle errors gracefully.

Key system requirements included:

- Python environment for data extraction and processing using yfinance and pandas.
- Docker-based Airflow setup for task scheduling and pipeline orchestration.
- Snowflake database integration for data warehousing and analytics.
- Daily automation of ETL processes.
- Forecasting capability using Snowflake ML (or SQL-based forecasting fallback).

While the system achieved its primary objectives, it faced limitations in using the Snowflake ML forecasting feature due to account restrictions. However, an alternative SQL-based forecasting logic was implemented successfully to produce forecast results.

III. Functional Analysis

The proposed system comprises four major components: Data Extraction, Data Storage, Pipeline Orchestration, and Forecasting. Each component contributes to the overall system workflow, from data retrieval to analysis and prediction. Stock symbols used here are MSFT (Microsoft) and ORCL (Oracle).

1. Data Extraction:

The data extraction component retrieves historical stock data from Yahoo Finance using the yfinance API. It captures attributes such as open, high, low, close prices, and trading volume. Data cleaning and validation are handled using pandas to ensure accuracy and uniformity.

2. Data Storage (Snowflake):

The processed data is stored in Snowflake in a table named STOCK_YF under the RAW schema. The table schema includes columns for SYMBOL, DATE, OPEN, HIGH, LOW, CLOSE, and VOLUME, with SYMBOL and DATE serving as primary keys. A transactional load ensures that data is refreshed atomically.

3. Data Pipeline Orchestration (Apache Airflow):

Apache Airflow, running within Docker containers, orchestrates the workflow. The Directed Acyclic Graph (DAG) defines four sequential tasks — extract, ensure_table, load_full_refresh, and forecast_task. These tasks ensure a linear flow, starting from data collection to forecast generation.

4. Forecasting Module:

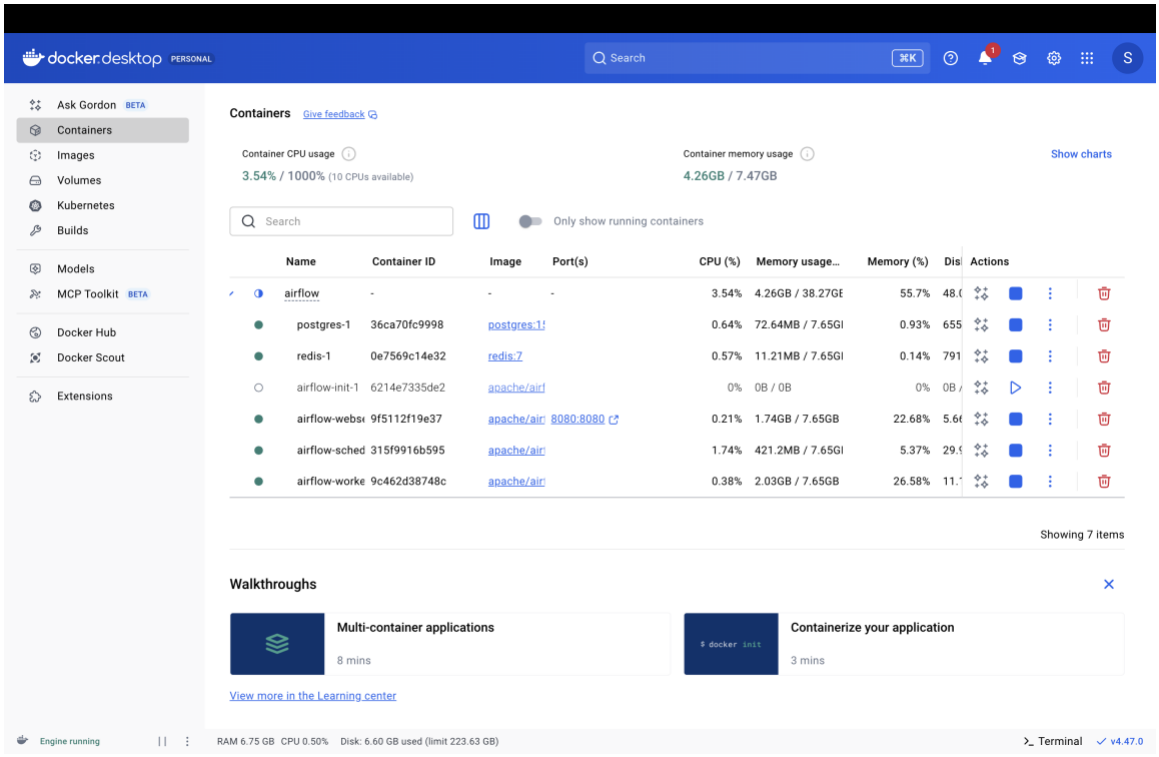
The forecasting module attempts to use Snowflake's ML. FORECAST functionality. When unavailable, a SQL-based fallback forecasting method is executed. This fallback model uses average daily changes and standard deviation to project 14 days of forecasted prices along with confidence intervals.

IV. Database and Pipeline Flow

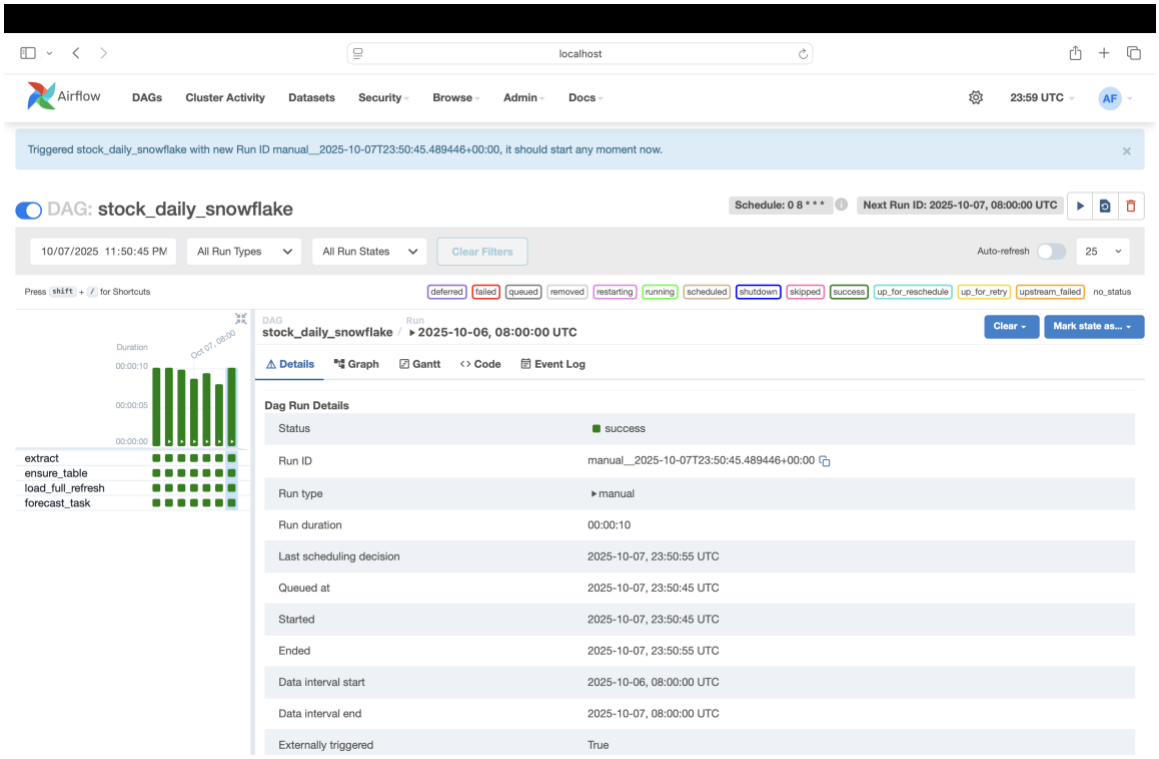
The complete system workflow is as follows:
1. Extract historical stock data from the yfinance API.
2. Transform and validate data using pandas.
3. Load cleaned data into the Snowflake STOCK_YF table.
4. Execute forecasting logic and store results in the STOCK_YF_FORECAST table.
5. Schedule automated daily executions using Apache Airflow.

Screenshots proofs of the process and logs according to the flow:

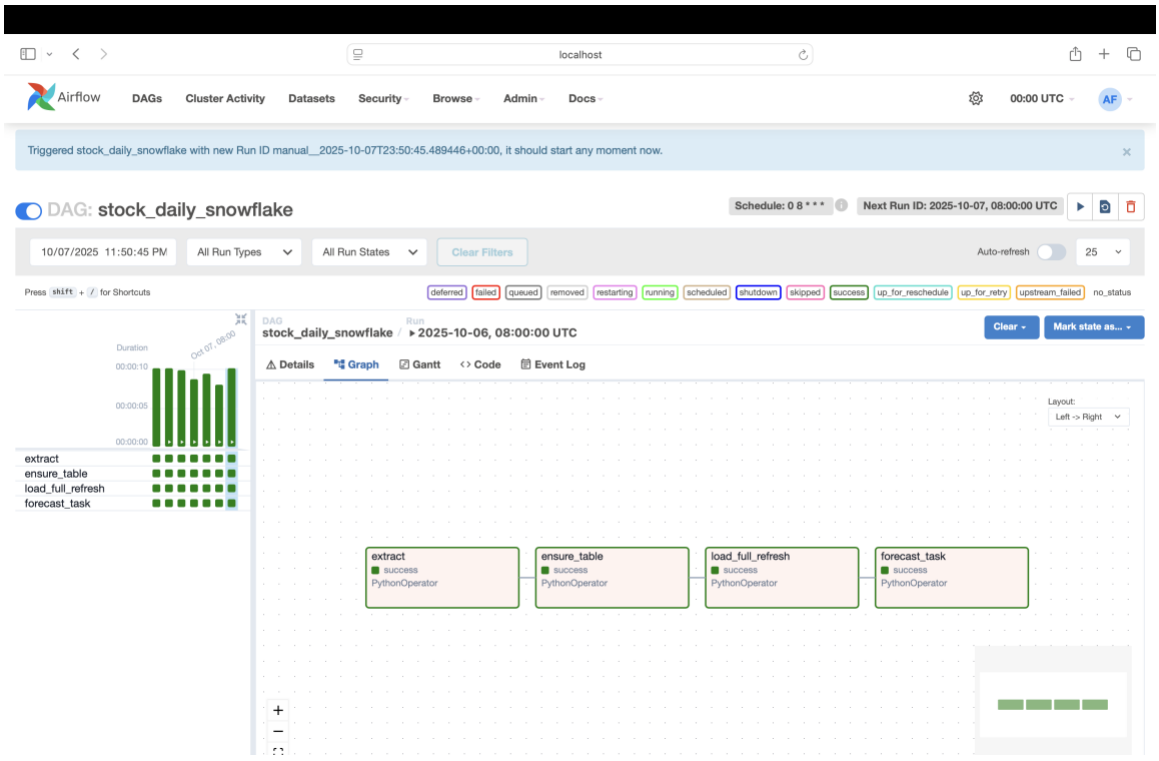
a. Docker app screen



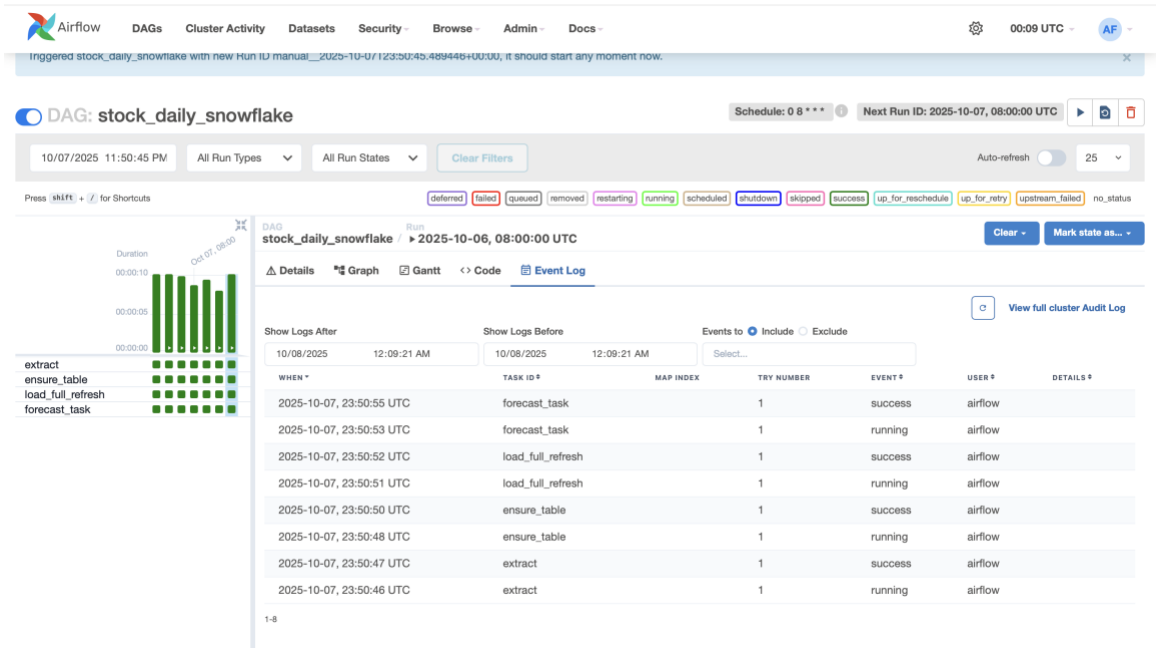
b. Airflow screenshot



c. Airflow DAG



d. Event logs



e. Extract (code)(stock_daily_snowflake.py)

1. Extract

```python
def fetch_last_180_days() -> pd.DataFrame:
    """Fetch ~180 days per ticker as a flat DataFrame with clean dtypes."""
    end = datetime.now(timezone.utc).date()
    start = end - timedelta(days=200)

    frames = []
    for t in SYMBOLS:
        # Download per symbol so columns are flat
        df = yf.download(t, start=start, end=end, progress=False).reset_index()

        # ---- Transform (light): normalize columns & dtypes ----
        df.columns = [c.lower() for c in df.columns]
        keep = ["date", "open", "high", "low", "close", "volume"]
        df = df[[c for c in keep if c in df.columns]]

        if "date" in df:
            df["date"] = pd.to_datetime(df["date"]).dt.date  # plain date for Snowflake binding

        for col in ["open", "high", "low", "close"]:
            if col in df:
                df[col] = pd.to_numeric(df[col], errors="coerce")

        if "volume" in df:
            df["volume"] = pd.to_numeric(df["volume"], errors="coerce").astype("Int64")

        df["symbol"] = t
        df = df[["symbol", "date", "open", "high", "low", "close", "volume"]]
        frames.append(df)
```

f. Transform (code) (stock_daily_snowflake.py)

2. Transform

```python
def transform_prices(df: pd.DataFrame) -> pd.DataFrame:
    # Ensure schema & dtypes exactly as Snowflake table
    cols = ["symbol","date","open","high","low","close","volume"]
    for c in cols:
        if c not in df.columns:
            df[c] = pd.NA

    df["date"] = pd.to_datetime(df["date"]).dt.date
    for col in ["open","high","low","close"]:
        df[col] = pd.to_numeric(df[col], errors="coerce")
    df["volume"] = pd.to_numeric(df["volume"], errors="coerce").astype("Int64")

    return df[cols].sort_values(["symbol","date"]).reset_index(drop=True)
```

g. Load (code) (stock_daily_snowflake.py)

3. Load

```python
def ensure_table():
    create_sql = f"""
    CREATE TABLE IF NOT EXISTS {TABLE_FQN} (
      SYMBOL STRING NOT NULL,
      DATE   DATE   NOT NULL,
      OPEN   NUMBER(18,4),
      HIGH   NUMBER(18,4),
      LOW    NUMBER(18,4),
      CLOSE  NUMBER(18,4),
      VOLUME NUMBER,
      CONSTRAINT PK_STOCK_YF PRIMARY KEY (SYMBOL, DATE)
    );
    """
    with sf_conn() as con, con.cursor() as cur:
        cur.execute(create_sql)

def full_refresh_load():
    """Delete-all + insert the freshly extracted/cleaned data."""
    ensure_table()
    df = fetch_last_180_days()

    if df.empty:
        print("No data fetched from yfinance; skipping insert.")
        return

    rows = [
        (
            str(r["symbol"]),
            r["date"],
            None if pd.isna(r["open"])  else float(r["open"]),
            None if pd.isna(r["high"])  else float(r["high"]),
            None if pd.isna(r["low"])   else float(r["low"]),
```

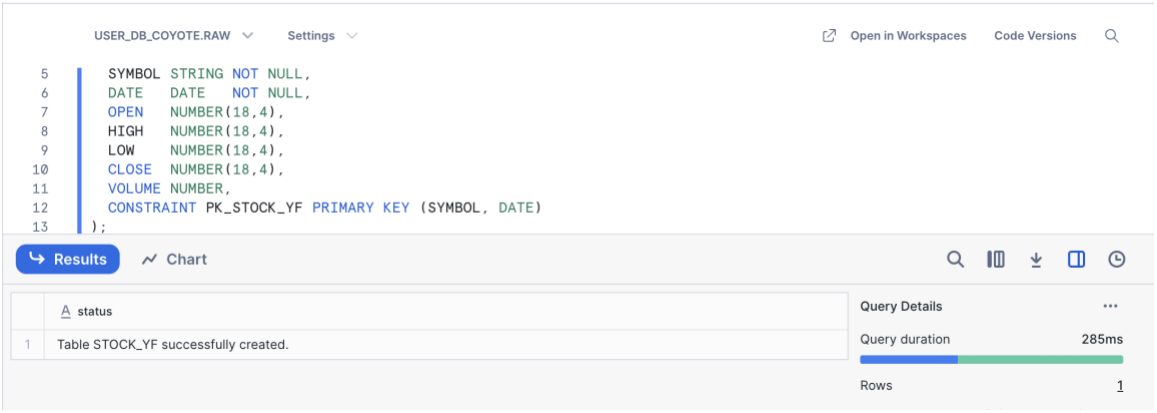
h. Forecast(code) (stock_daily_snowflake.py) in pipeline

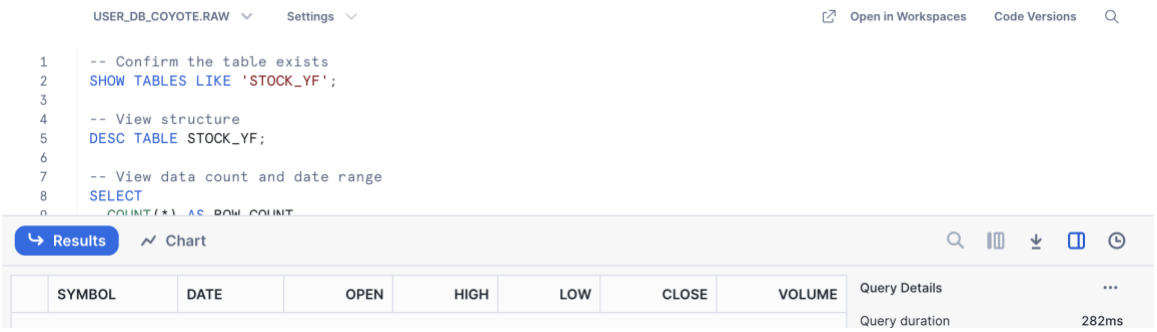
Forecast

```python
def run_forecast_or_skip():
    """
    Try SNOWFLAKE.ML.FORECAST (if the account has it). If not present,
    log and exit gracefully so your DAG still stays green.
    """
    with sf_conn() as con, con.cursor() as cur:
        try:
            cur.execute("SHOW FUNCTIONS LIKE 'FORECAST' IN SCHEMA SNOWFLAKE.ML")
            has = cur.fetchall()
        except Exception:
            has = []

        if not has:
            print("SNOWFLAKE.ML.FORECAST not available in this account. Skipping forecast step.")
            return

        cur.execute(f"""
            CREATE OR REPLACE TABLE {SF_DATABASE}.{SF_SCHEMA}.STOCK_YF_FORECAST AS
            WITH BASE AS (
                SELECT SYMBOL, DATE, CLOSE
                FROM {TABLE_FQN}
                WHERE CLOSE IS NOT NULL
            )
            SELECT
                f.SYMBOL,
                f.FORECASTED_TIME AS DATE,
                f.FORECASTED_VALUE AS CLOSE_FORECAST,
                f.LOWER_BOUND AS LCL,
                f.UPPER_BOUND AS UCL
```

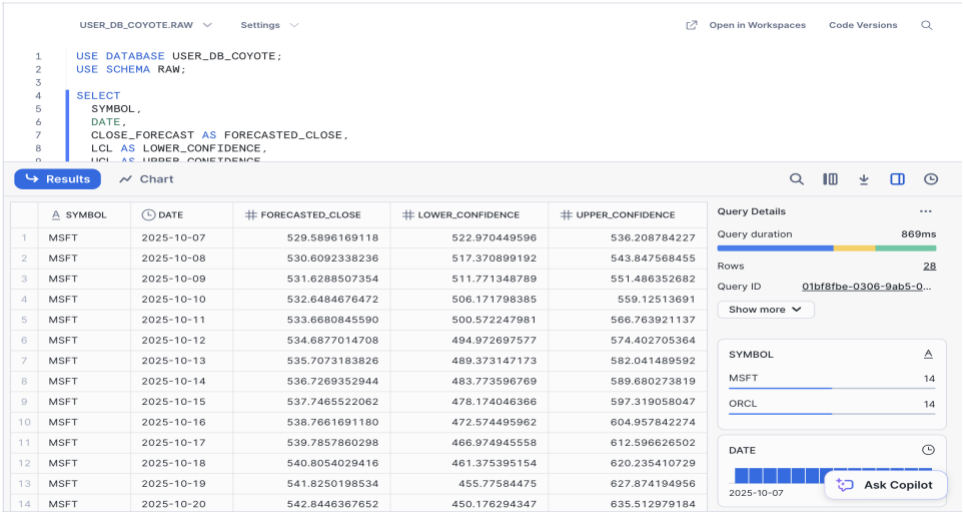
i. Snowflake forecast table SQL

j. Table after pulling data from Yfinance. (STOCK_YF)





k. ML forecast table () for MSFT, we can change it to the other symbol according to the need



V. Table Structures

Two main tables were created in Snowflake as part of the project:

Table 1: STOCK_YF (Raw and Processed Data)
- SYMBOL – String (Primary Key)
- DATE – Date (Primary Key)
- OPEN – Number(18,4)
- HIGH – Number(18,4)
- LOW – Number(18,4)
- CLOSE – Number(18,4)
- VOLUME – Number

Table 2: STOCK_YF_FORECAST (Forecasted Data)
- SYMBOL – String
- DATE – Date
- CLOSE_FORECAST – Number(18,4)
- LCL – Lower Confidence Limit
- UCL – Upper Confidence Limit

VI. Results and Discussion

The pipeline executed successfully through Apache Airflow, performing all ETL steps and generating a forecast table in Snowflake. The DAG ran smoothly with all tasks marked successful, demonstrating the correctness and stability of the workflow. The SQL-based forecast logic generated reliable trend projections when the Snowflake ML forecasting function was unavailable.

The system achieved daily data automation, accurate record insertion, and fault tolerance. With modular design, additional analytical tasks such as sentiment analysis or portfolio optimization can be easily integrated in future upgrades.

VII. Conclusion

This project effectively demonstrates an automated financial data pipeline using modern cloud-based data engineering tools. The integration of Python, Snowflake, and Apache Airflow enabled efficient data handling, automation, and forecasting. The fallback SQL-based forecast ensured continuous functionality despite Snowflake ML feature limitations. The outcome validates a scalable, practical approach to financial analytics automation suitable for enterprise use.

GITHUB LINK:

https://github.com/sharan9219790/DATAWAREHOUSE-LAB1

References

[1] Apache Airflow Documentation: https://airflow.apache.org/

[2] Snowflake Documentation: https://docs.snowflake.com/

[3] yfinance Library: https://pypi.org/project/yfinance/

[4] Pandas Documentation: https://pandas.pydata.org/