Assignment 5

USERNAME: PIRANHA

Porting homework #4 to Airflow (13 pts)

- **(+2) Create tasks using @task decorator (refer to [GitHub linkLinks to an external site.])**
  - **You can use as many tasks as you want**

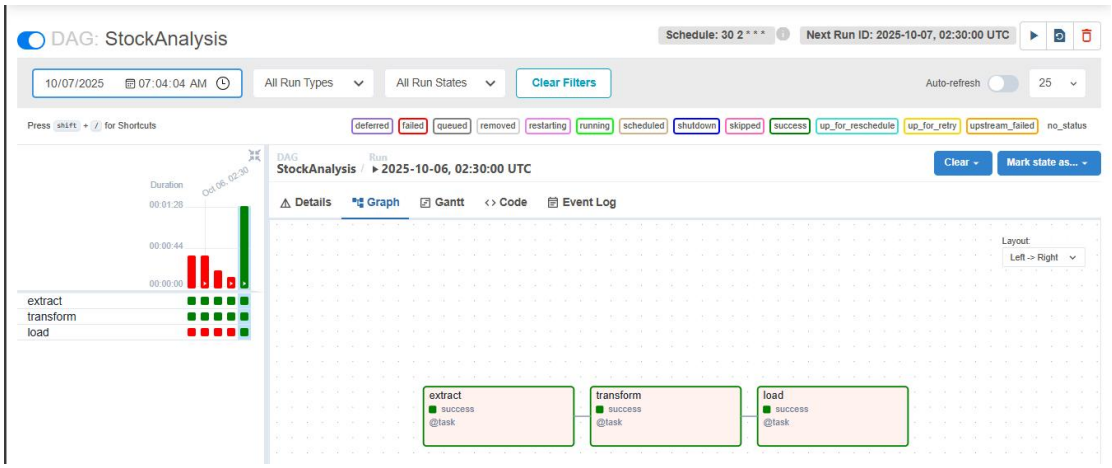
```
import requests
from airflow import DAG
from airflow.models import Variable
from airflow.decorators import task
from airflow.providers.snowflake.hooks.snowflake import SnowflakeHook
from airflow.hooks.base import BaseHook
from datetime import timedelta
from datetime import datetime
import snowflake.connector
```

```
@task
def extract(symbol):
    api_key=Variable.get("vantage_api_key")
    symbol="AAPL"
    url =
f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={
symbol}&apikey={api_key}'
    r = requests.get(url)
    data = r.json()
    results = []
    for d in data["Time Series (Daily)"]:
      stock_details = data["Time Series (Daily)"][d].copy()
      stock_details["date"] = d
      results.append(stock_details)
    return results

@task
def transform(price_list):
  price_list = sorted(price_list, key=lambda d:
d['date'],reverse=True)[:90]
  records = []
  for p in price_list:
        record = {
            "date": p["date"],
            "open": float(p["1. open"]),
```

```
            "high": float(p["2. high"]),
            "low": float(p["3. low"]),
            "close": float(p["4. close"]),
            "volume": float(p["5. volume"])}
        records.append(record)
  return records
```

- **Schedule the tasks properly (task dependency)**

```
with DAG(
    dag_id="StockAnalysis",
    start_date=datetime(2025,9,29),
    catchup=False,
    tags=["ETL"],
    schedule='30 2 * * *'
)as dag:
    #pass
    #symbol = "AAPL"
    target_table="raw.stock_datas"
    symbol = "AAPL"
    vantage_api_key=Variable.get("vantage_api_key")
    price_list = extract("AAPL")
    records = transform(price_list)
    load(records=records, symbol= "AAPL")

    extract_task >> transform_task >> load_task
```



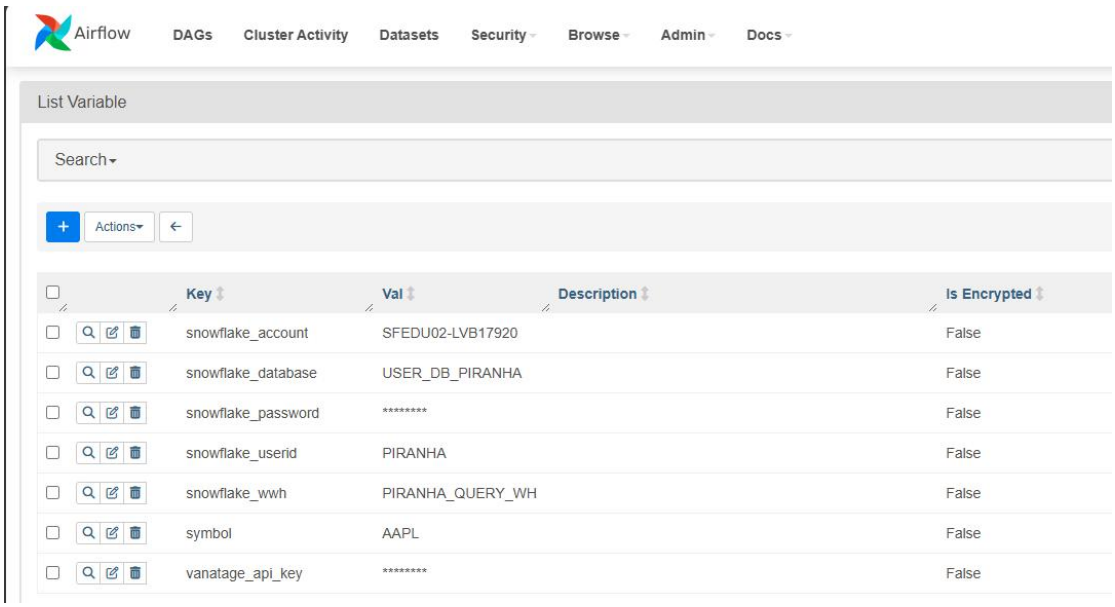
- **(+1) Set up a variable for Alpha Vantage API key**
  - **Use the variable in your code (Variable.get)**

```
Variable.get = staticmethod(lambda key: "your_secret_key" if key ==
"vantage_api_key" else None)
```

```
vantage_api_key = Variable.get("vantage_api_key")
```

```
@task
def extract(symbol):
    api_key=Variable.get("vantage_api_key")
    symbol="AAPL"
    url =
f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={
symbol}&apikey={api_key}'
    r = requests.get(url)
    data = r.json()
    results = []
    for d in data["Time Series (Daily)"]:
      stock_details = data["Time Series (Daily)"][d].copy()
      stock_details["date"] = d
      results.append(stock_details)
    return results
```

Capture the Admin -> Variables screenshot (an example will be provided ①)

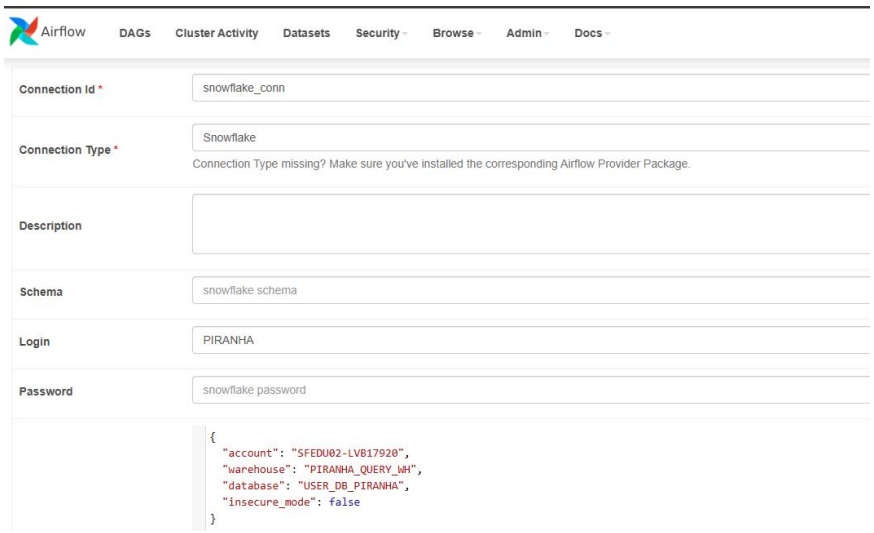


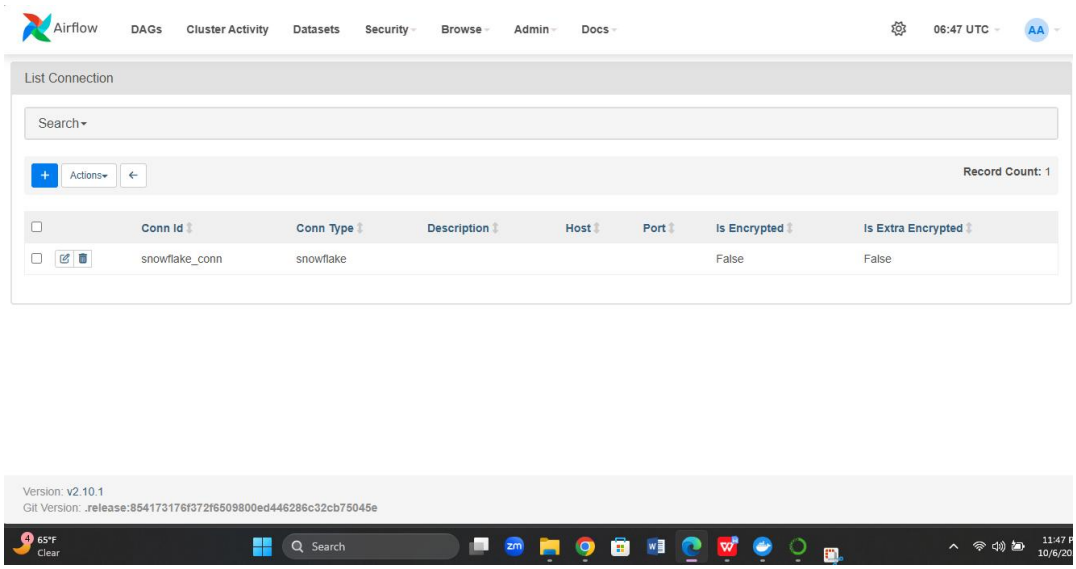
(+2) Set up Snowflake Connection (refer to [GitHub linkLinks to an external site.](#))

- **Use the connection in your code**

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

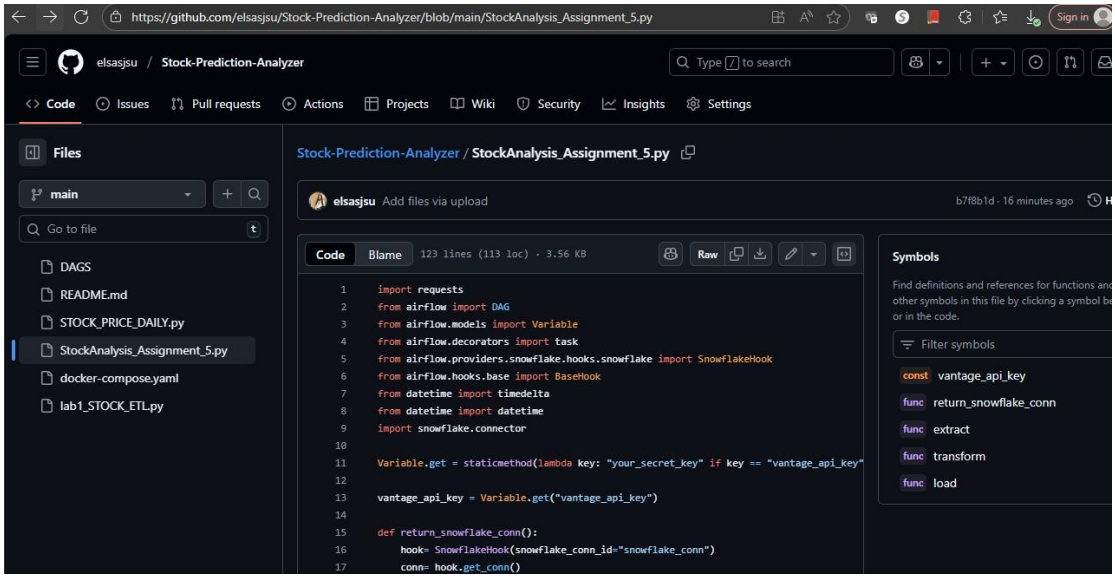
- **Capture the Connection detail page screenshot (an example will be provided ②)**



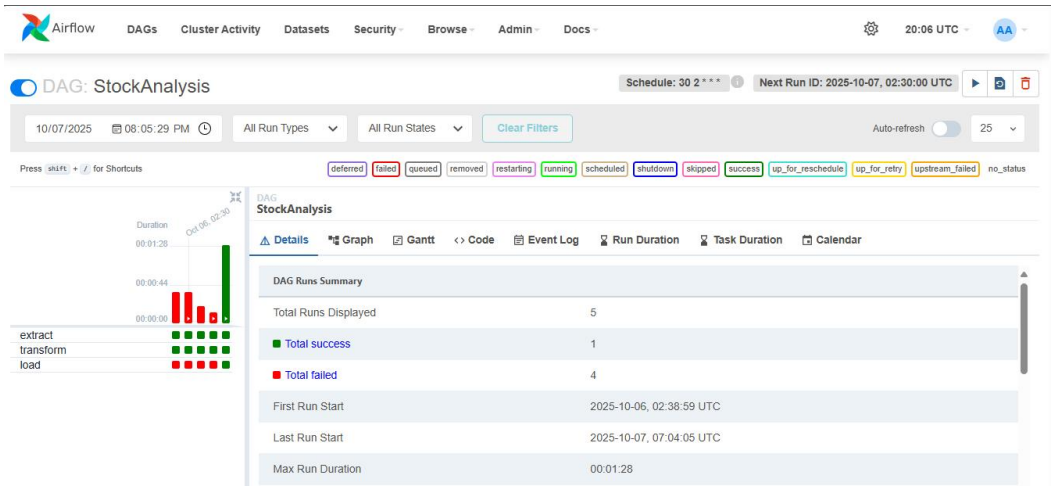


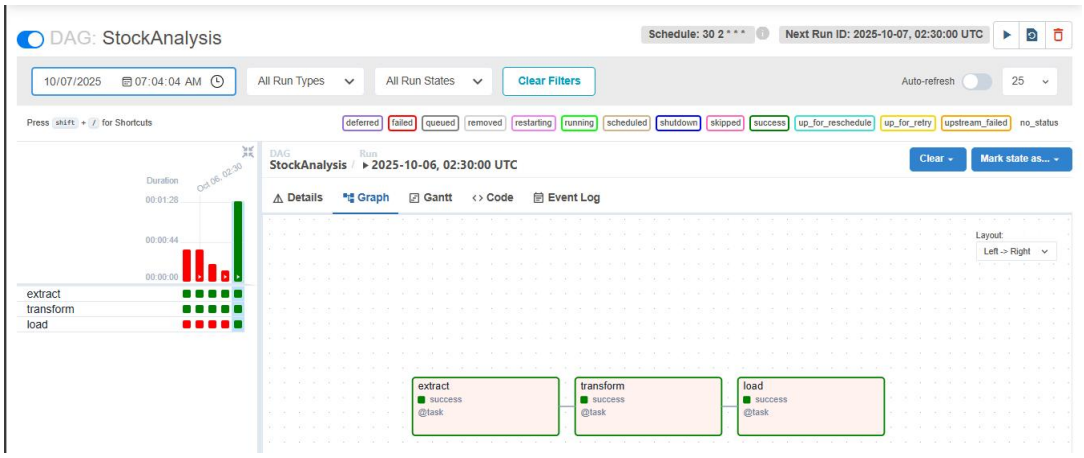
(+5) Ensure the overall DAG is implemented properly and runs successfully

- **A github link with the entire code needs to be submitted (2 pts)**



Git Hub Link: [Stock-Prediction-Analyzer/StockAnalysis_Assignment_5.py at main · elsasjsu/Stock-Prediction-Analyzer]





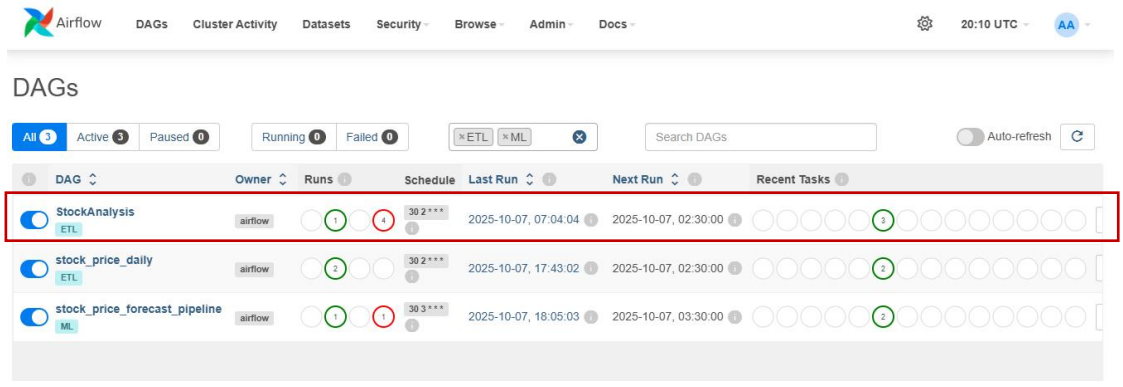
- **Implement the same full refresh using SQL transaction (3 pts)**

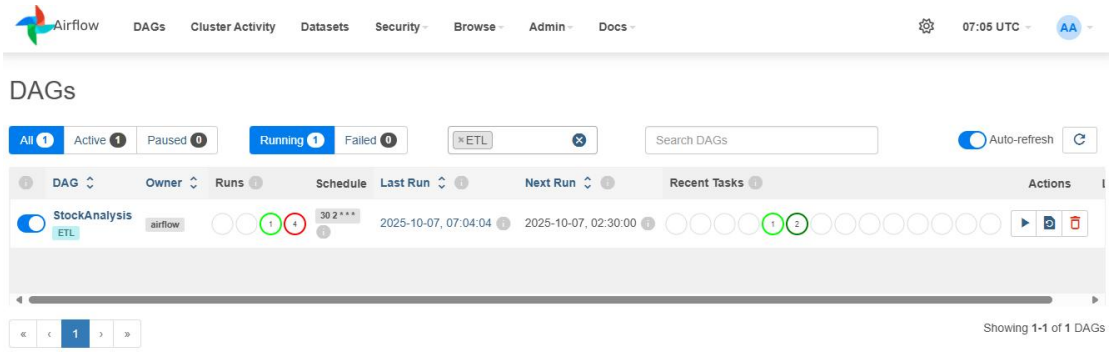
```
@task
def load(records, symbol):
    target = "RAW.STOCK_DATAS"
    conn = return_snowflake_conn()
    cur = conn.cursor()
    try:
        cur.execute("BEGIN")
        extras =
(BaseHook.get_connection("snowflake_conn").extra_dejson or {})
        wh = extras.get("warehouse")
        db = extras.get("database")
        if wh:
            cur.execute(f"USE WAREHOUSE {wh}")
        db = extras.get("database")
        if db:
            cur.execute(f"USE DATABASE {db}")
        cur.execute(f"CREATE SCHEMA IF NOT EXISTS RAW")
        cur.execute("USE SCHEMA RAW")

        cur.execute(f"""
            CREATE TABLE IF NOT EXISTS {target} (
                symbol VARCHAR NOT NULL,
                date DATE NOT NULL,
                open FLOAT,
                high FLOAT,
                low FLOAT,
                volume NUMBER,
                PRIMARY KEY (symbol, date)
            );
        """)
        sql = f"""
            INSERT INTO {target} (symbol, date, open, high, low, volume)
            VALUES (%s, %s, %s, %s, %s, %s)
        """
        cur.execute(f"DELETE FROM {target};")
        for row in records:
            cur.execute(
                sql,
                (
                    symbol,
                    str(row["date"]),
                    float(row["open"]),
                    float(row["high"]),
                    float(row["low"]),
                    float(row["volume"]),
                ),
            )
```

```
        cur.execute("COMMIT;")
    except Exception as e:
        cur.execute("ROLLBACK;")
        print(e)
        raise
    finally:
        cur.close()
        conn.close()
```

```
# Usage example outside the function:
#cur = return_snowflake_conn()
#load_v(cur, database=database, df=df, symbol="AAPL")
```

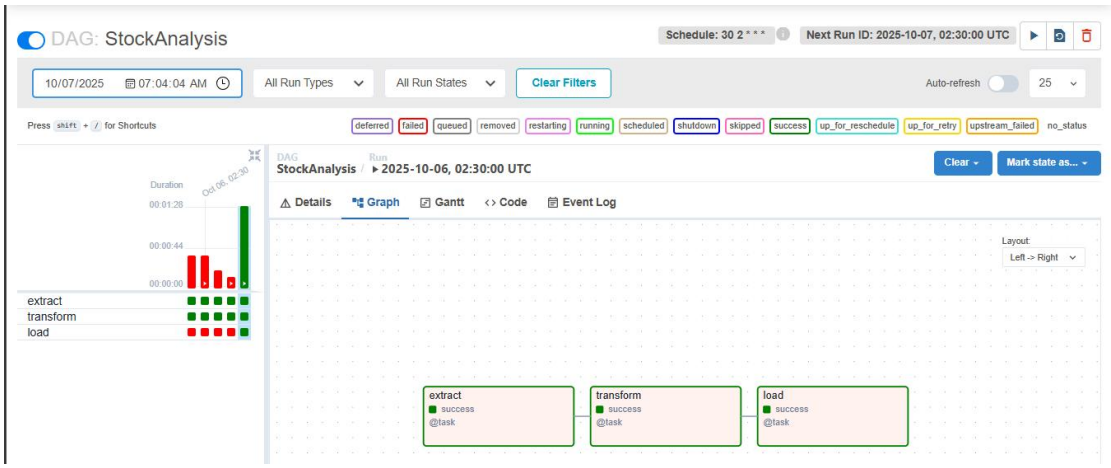
- **(+2) Capture two screenshot of your Airflow Web UI (examples to follow**
    - **One with the Airlow homepage showing the DAG (③)**



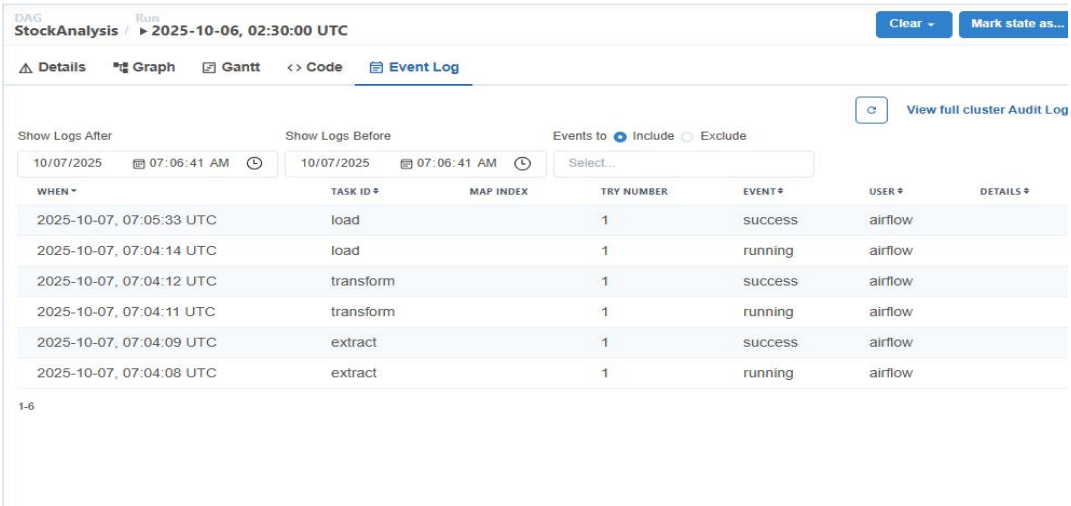


- **The other with the log screen of the DAG (④)**

GRAPH VIEW



LOG SCREEN



- (+1) Overall formatting