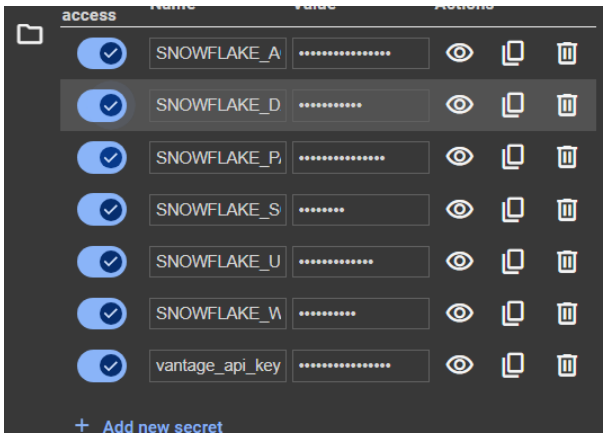
Homework 4

1. (+1) Pick up a stock symbol and get your own API key from Alpha Vantage

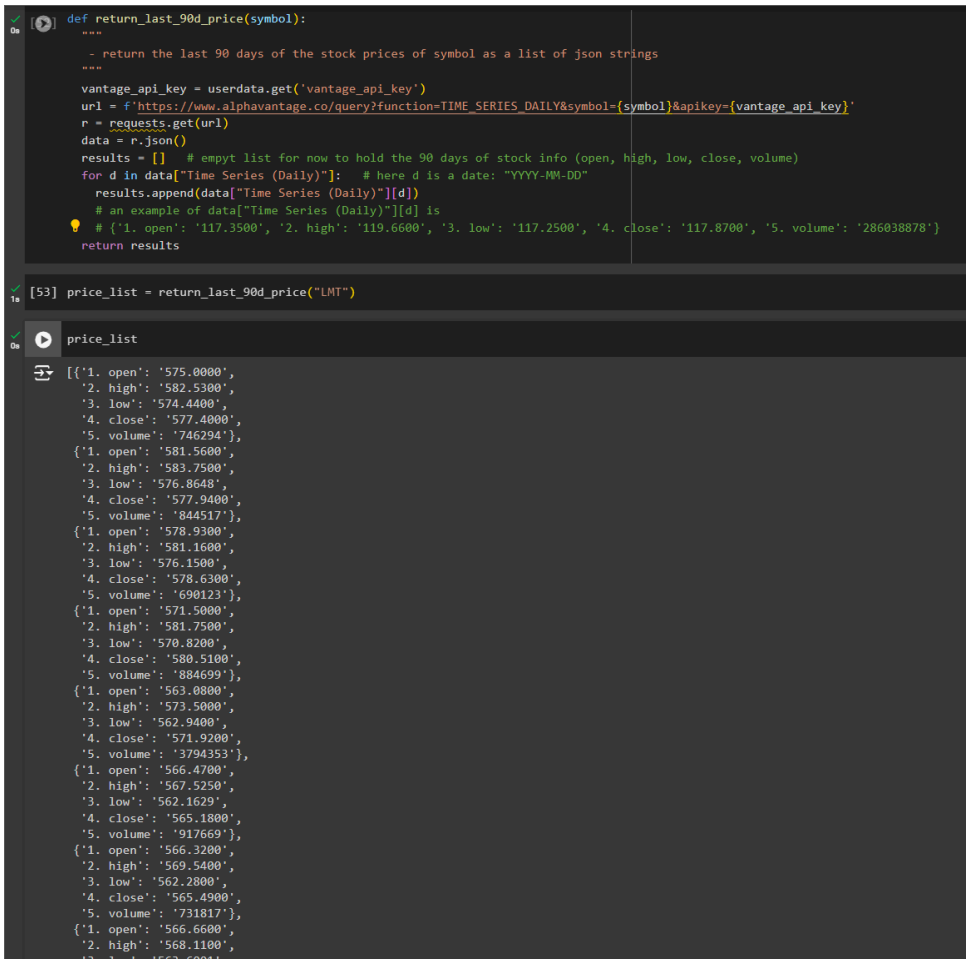
   I have chosen Lockheed Martin(LMT) stock and added my Alpha Vantage API key in secrets.

```
+ Code  + Text                                                    RAM  Disk   ✦ Gemini

[24] !pip install snowflake-connector-python
     !pip install alpha_vantage

     Show hidden output

[32] from google.colab import userdata
     vantage_api_key = userdata.get('vantage_api_key')

[33] symbol = "LMT"
     # note that Formatted String Literal is used here
     url = f"https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}"
```

2. (+1) Secure your Snowflake credentials and Alpha Vantage API key (don't expose them in the code) using the secret in Google Colab



3. (+2) Read the last 90 days of the price info via the API



```
def return_last_90d_price(symbol):
    """
     - return the last 90 days of the stock prices of symbol as a list of json strings
    """
    vantage_api_key = userdata.get('vantage_api_key')
    url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}'
    r = requests.get(url)
    data = r.json()
    results = []   # empyt list for now to hold the 90 days of stock info (open, high, low, close, volume)
    for d in data["Time Series (Daily)"]:   # here d is a date: "YYYY-MM-DD"
      results.append(data["Time Series (Daily)"][d])
      # an example of data["Time Series (Daily)"][d] is
      # {'1. open': '117.3500', '2. high': '119.6600', '3. low': '117.2500', '4. close': '117.8700', '5. volume': '286038878'}
    return results

[53] price_list = return_last_90d_price("LMT")

price_list

[{'1. open': '575.0000',
  '2. high': '582.5300',
  '3. low': '574.4400',
  '4. close': '577.4000',
  '5. volume': '746294'},
 {'1. open': '581.5600',
  '2. high': '583.7500',
  '3. low': '576.8648',
  '4. close': '577.9400',
  '5. volume': '844517'},
 {'1. open': '578.9300',
  '2. high': '581.1600',
  '3. low': '576.1500',
  '4. close': '578.6300',
  '5. volume': '690123'},
 {'1. open': '571.5000',
  '2. high': '581.7500',
  '3. low': '570.8200',
  '4. close': '580.5100',
  '5. volume': '884699'},
 {'1. open': '563.0800',
  '2. high': '573.5000',
  '3. low': '562.9400',
  '4. close': '571.9200',
  '5. volume': '3794353'},
 {'1. open': '566.4700',
  '2. high': '567.5250',
  '3. low': '562.1629',
  '4. close': '565.1800',
  '5. volume': '917669'},
 {'1. open': '566.3200',
  '2. high': '569.5400',
  '3. low': '562.2800',
  '4. close': '565.4900',
  '5. volume': '731817'},
 {'1. open': '566.6600',
  '2. high': '568.1100',
```

4. (+1) Create or replace a table with a primary key under raw_data schema to capture the info from the API.

```python
import snowflake.connector
from google.colab import userdata

# Fetch Snowflake credentials from Colab secrets
user = userdata.get('SNOWFLAKE_USER')
password = userdata.get('SNOWFLAKE_PASSWORD')
account = userdata.get('SNOWFLAKE_ACCOUNT')
warehouse = userdata.get('SNOWFLAKE_WAREHOUSE')
database = userdata.get('SNOWFLAKE_DATABASE')
schema = userdata.get('SNOWFLAKE_SCHEMA')

# Set up your Snowflake connection parameters
conn = snowflake.connector.connect(
    user=user,
    password=password,
    account=account,
    warehouse=warehouse,
    database=database,
    schema=schema
)

# Create a cursor object
cur = conn.cursor()

# SQL command to create the table with a primary key
create_table_query = """
CREATE OR REPLACE TABLE stock_prices (
    date DATE NOT NULL,
    open FLOAT,
    high FLOAT,
    low FLOAT,
    close FLOAT,
    volume INTEGER,
    symbol STRING,
    PRIMARY KEY (date, symbol)
);
"""

# Execute the command
cur.execute(create_table_query)

# Close the cursor and connection
cur.close()
conn.close()
```

As you can see I have added a date field and made it not null, inferring that date will be the Primary key going ahead.

5. (+2) Populate the table with the records from step 3 using INSERT SQL

```python
import snowflake.connector
from google.colab import userdata
import requests

def return_last_90d_price(symbol):
    """
    - return the last 90 days of the stock prices of symbol as a list of json strings
    """
    vantage_api_key = userdata.get('vantage_api_key')
    url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}'
    r = requests.get(url)
    data = r.json()
    results = []  # empty list to hold the 90 days of stock info (open, high, low, close, volume)
    for d in data["Time Series (Daily)"]:  # d is a date: "YYYY-MM-DD"
        daily_data = data["Time Series (Daily)"][d]
        # Create a record to insert into the database
        record = (
            d,  # date
            float(daily_data['1. open']),
            float(daily_data['2. high']),
            float(daily_data['3. low']),
            float(daily_data['4. close']),
            int(daily_data['5. volume']),
            symbol
        )
        results.append(record)
    return results

# Use the function to get the last 90 days of stock prices for a specific symbol
symbol = "LMT"
prices = return_last_90d_price(symbol)

# Set up your Snowflake connection parameters using secrets
conn = snowflake.connector.connect(
    user=userdata.get('SNOWFLAKE_USER'),
    password=userdata.get('SNOWFLAKE_PASSWORD'),
    account=userdata.get('SNOWFLAKE_ACCOUNT'),
    warehouse=userdata.get('SNOWFLAKE_WAREHOUSE'),
    database=userdata.get('SNOWFLAKE_DATABASE'),
    schema=userdata.get('SNOWFLAKE_SCHEMA')
)

# Create a cursor object
cur = conn.cursor()

# Insert records into the table
insert_query = """
INSERT INTO stock_prices (date, open, high, low, close, volume, symbol)
VALUES (%s, %s, %s, %s, %s, %s, %s);
"""

# Execute the insert statement for each price record
for price in prices:
    cur.execute(insert_query, price)

# Commit the transaction
conn.commit()

# Close the cursor and connection
cur.close()
conn.close()
```

6. (+4) Steps 4 and 5 need to be done together

```python
def return_last_90d_price(symbol):
    """
    - return the last 90 days of the stock prices of symbol as a list of tuples
    """
    vantage_api_key = userdata.get('vantage_api_key')
    url = f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={symbol}&apikey={vantage_api_key}'
    r = requests.get(url)
    data = r.json()
    results = [] # empty list to hold the 90 days of stock info (open, high, low, close, volume)
    for d in data["Time Series (Daily)"]:  # d is a date: "YYYY-MM-DD"
        daily_data = data["Time Series (Daily)"][d]
        # Create a record to insert into the database
        record = (
            d,  # date
            float(daily_data['1. open']),
            float(daily_data['2. high']),
            float(daily_data['3. low']),
            float(daily_data['4. close']),
            int(daily_data['5. volume']),
            symbol
        )
        results.append(record)
    return results
```

```python
def get_record_count(cur):
    """
    Get the count of records in the stock_prices table
    """
    cur.execute("SELECT COUNT(*) FROM stock_prices;")
    return cur.fetchone()[0]

# Use the function to get the last 90 days of stock prices for a specific symbol
symbol = "LMT"
prices = return_last_90d_price(symbol)

# Set up your Snowflake connection parameters using secrets
conn = snowflake.connector.connect(
    user=userdata.get('SNOWFLAKE_USER'),
    password=userdata.get('SNOWFLAKE_PASSWORD'),
    account=userdata.get('SNOWFLAKE_ACCOUNT'),
    warehouse=userdata.get('SNOWFLAKE_WAREHOUSE'),
    database=userdata.get('SNOWFLAKE_DATABASE'),
    schema=userdata.get('SNOWFLAKE_SCHEMA')
)
```

```python
cur = conn.cursor()

initial_count = get_record_count(cur)
print(f"Initial record count: {initial_count}")

merge_query = """
MERGE INTO stock_prices AS target
USING (SELECT %s AS date, %s AS open, %s AS high, %s AS low, %s AS close, %s AS volume, %s AS symbol) AS source
ON target.date = source.date AND target.symbol = source.symbol
WHEN NOT MATCHED THEN
    INSERT (date, open, high, low, close, volume, symbol)
    VALUES (source.date, source.open, source.high, source.low, source.close, source.volume, source.symbol);
"""

for price in prices:
    cur.execute(merge_query, price)

conn.commit()

first_insertion_count = get_record_count(cur)
print(f"Record count after first insertion: {first_insertion_count}")

for price in prices:
    cur.execute(merge_query, price)

conn.commit()

second_insertion_count = get_record_count(cur)
print(f"Record count after second insertion: {second_insertion_count}")

cur.close()
conn.close()
```

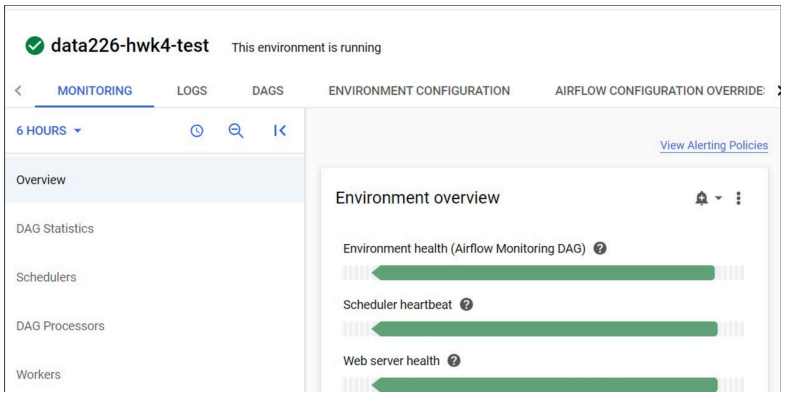
7. (+1) Demonstrate your work ensures Idempotency by running it twice in a row and checking the number of records

```python
# Execute the merge statement for each price record
for price in prices:
    cur.execute(merge_query, price)

# Commit the transaction
conn.commit()

# Check record count after first insertion
first_insertion_count = get_record_count(cur)
print(f"Record count after first insertion: {first_insertion_count}")

# Run the insertion again to demonstrate idempotency
for price in prices:
    cur.execute(merge_query, price)

# Commit the transaction again
conn.commit()

# Check record count after second insertion
second_insertion_count = get_record_count(cur)
print(f"Record count after second insertion: {second_insertion_count}")

# Close the cursor and connection
cur.close()
conn.close()
```

```
Initial record count: 100
Record count after first insertion: 100
Record count after second insertion: 100
```

As we can see after 2 insert operations, the state of the table does not change ensuring idempotency is maintained

8. (+1) Follow today's demo (you can find relevant slides from today's lecture notes too) and capture your Cloud Composer Environment screen


data226-hwk4-test
This environment is running
MONITORING
LOGS
DAGS
ENVIRONMENT CONFIGURATION
AIRFLOW CONFIGURATION OVERRIDE
6 HOURS
Overview
DAG Statistics
Schedulers
DAG Processors
Workers
View Alerting Policies
Environment overview
Environment health (Airflow Monitoring DAG)
Scheduler heartbeat
Web server health