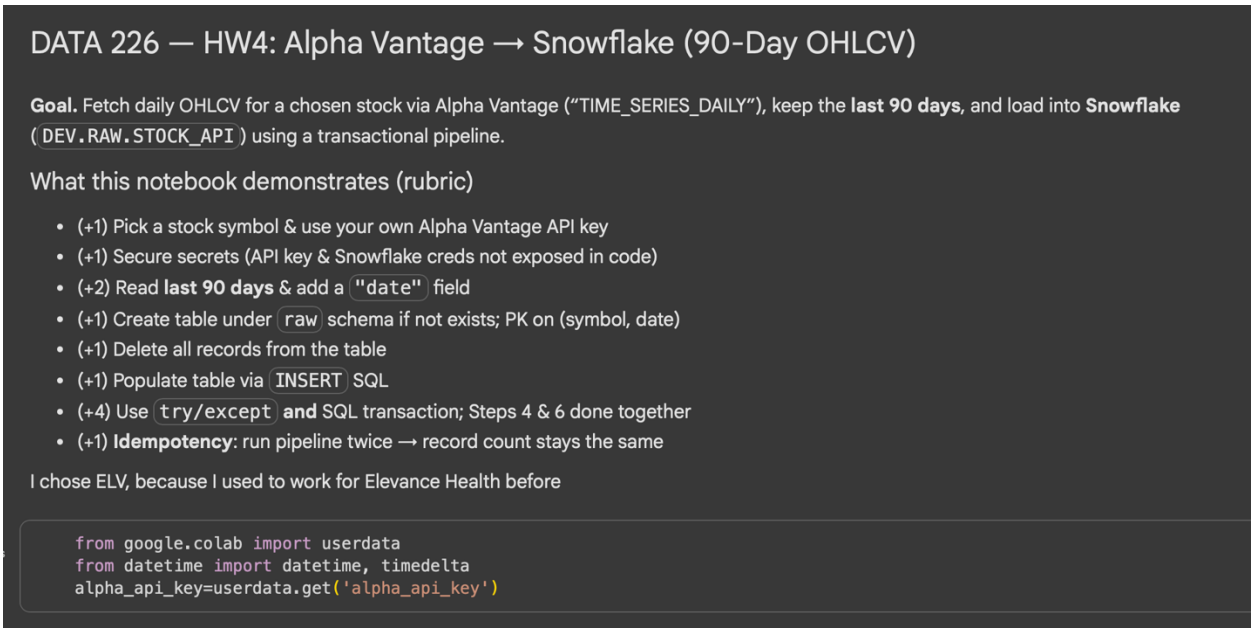
1. (+1) Pick up a stock symbol and get your own API key from Alpha Vantage

```
from google.colab import userdata
from datetime import datetime, timedelta
alpha_api_key=userdata.get('alpha_api_key')
```



DATA 226 — HW4: Alpha Vantage → Snowflake (90-Day OHLCV)

Goal. Fetch daily OHLCV for a chosen stock via Alpha Vantage ("TIME_SERIES_DAILY"), keep the **last 90 days**, and load into **Snowflake** (`DEV.RAW.STOCK_API`) using a transactional pipeline.

What this notebook demonstrates (rubric)

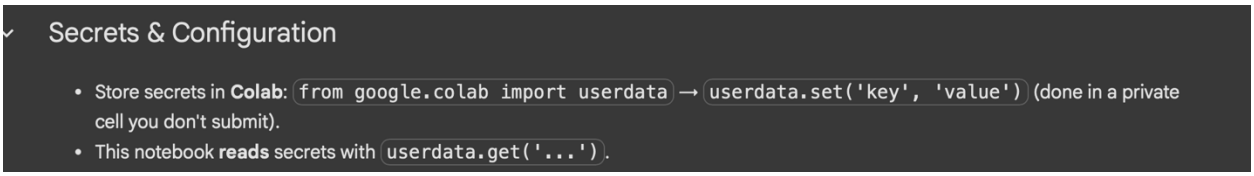
- (+1) Pick a stock symbol & use your own Alpha Vantage API key
- (+1) Secure secrets (API key & Snowflake creds not exposed in code)
- (+2) Read **last 90 days** & add a `"date"` field
- (+1) Create table under `raw` schema if not exists; PK on (symbol, date)
- (+1) Delete all records from the table
- (+1) Populate table via `INSERT` SQL
- (+4) Use `try/except` **and** SQL transaction; Steps 4 & 6 done together
- (+1) **Idempotency**: run pipeline twice → record count stays the same

I chose ELV, because I used to work for Elevance Health before

```
from google.colab import userdata
from datetime import datetime, timedelta
alpha_api_key=userdata.get('alpha_api_key')
```

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

```
user_id = userdata.get('snowflake_userid')
password = userdata.get('snowflake_password')
account = userdata.get('snowflake_account')
database = userdata.get('snowflake_database')
vwh = userdata.get('snowflake_vwh')
```

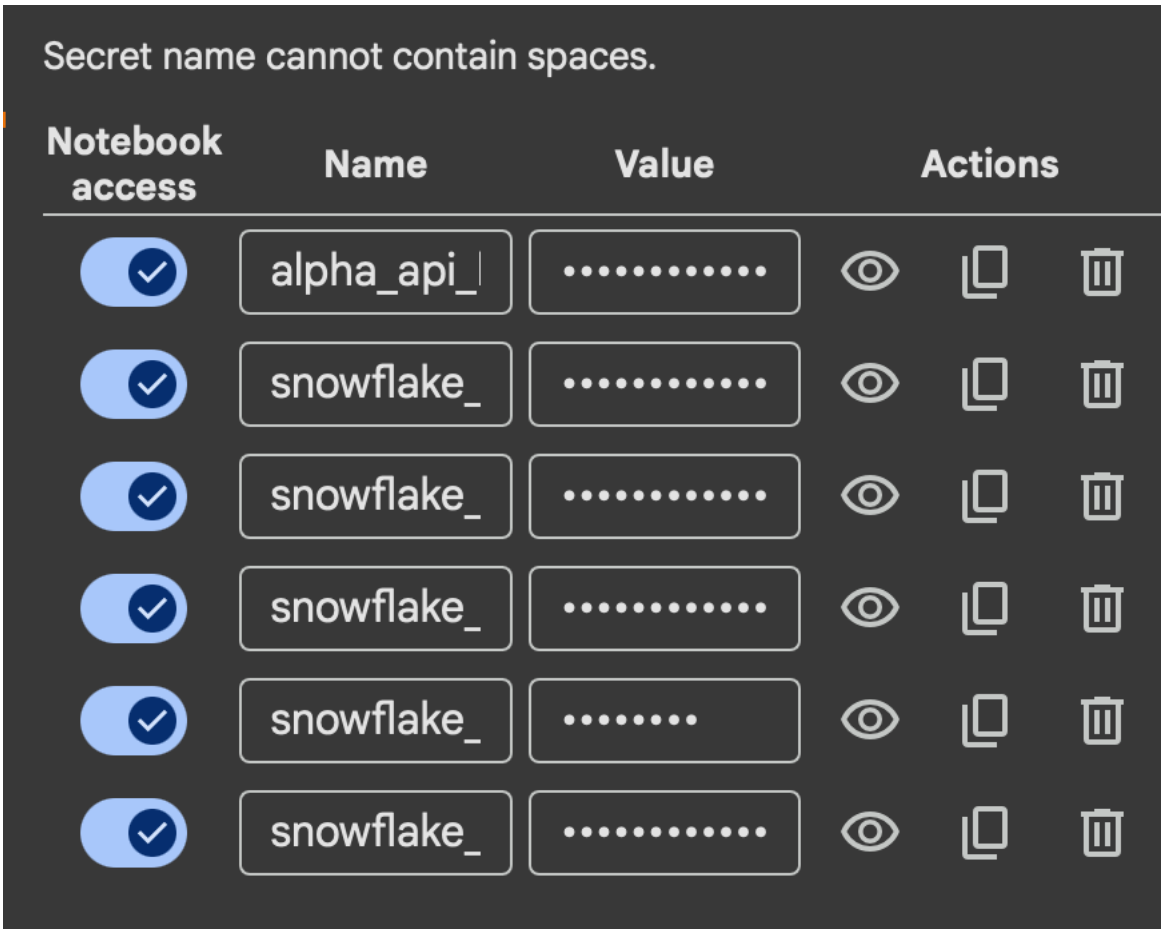


Secrets & Configuration

- Store secrets in **Colab**: `from google.colab import userdata` → `userdata.set('key', 'value')` (done in a private cell you don't submit).
- This notebook **reads** secrets with `userdata.get('...')`.

Snowflake credentials saved in secrets

```
user_id = userdata.get('snowflake_userid')
password = userdata.get('snowflake_password')
account = userdata.get('snowflake_account')
database = userdata.get('snowflake_database')
vwh = userdata.get('snowflake_vwh')
```



3. (+2) Read the last 90 days of the price info via the API (refer to [the code snippetLinks to an external site.](#)& you need to add "date")

```
def return_last_90d_price(symbol):
    vantage_api_key = userdata.get('alpha_api_key')
    url =
f'https://www.alphavantage.co/query?function=TIME_SERIES_DAILY&symbol={sym
bol}&apikey={vantage_api_key}'
    r = requests.get(url)
    data = r.json()
```

```
    cutoff = datetime.today().date() - timedelta(days=90)
    results = []

    for d in data["Time Series (Daily)"]:
        # Parse date string into a date object
        trade_date = datetime.strptime(d, "%Y-%m-%d").date()
        if trade_date >= cutoff:
            stock_info = data["Time Series (Daily)"][d]
            stock_info["date"] = d
            results.append(stock_info)

    return results
```

4. (+1) Create a table under “raw” schema if it doesn’t exist to capture the info from the API
    a. symbol, date, open, close, high, low, volume: symbol and date should be **primary keys**
5. (+1) Delete all records from the table
6. (+1) Populate the table with the records from step 2 using INSERT SQL (refer tothe relevant code snippetLinks to an external site. as a starting point)
7. (+4) Steps 4 and 6 need to be done together
    a. Use try/except along with SQL transaction. (use the code hereLinks to an external site. as reference)

```
ef load(con, records,symbol):
    target_table = "user_db_hedgehog.raw.stock_api"
    try:
        con.execute("BEGIN;")
        con.execute(f"""CREATE TABLE IF NOT EXISTS {target_table} (
          symbol VARCHAR NOT NULL,
          trade_date DATE NOT NULL,
          open  NUMBER(18,4),
          close NUMBER(18,4),
          high  NUMBER(18,4),
          low   NUMBER(18,4),
          volume NUMBER(38,0),
          CONSTRAINT pk_symbol_date PRIMARY KEY (symbol, trade_date) NOT
ENFORCED);;""")
        con.execute(f"""DELETE FROM {target_table}""")
        for r in records:
          trade_date = r["date"]
          open = r["1. open"]
          high = r["2. high"]
          low = r["3. low"]
```

```
            close = r["4. close"]
            volume = r["5. volume"]
            insert_sql = f"INSERT INTO {target_table} (symbol, trade_date,
open, close, high,low, volume) VALUES
('{symbol}',TO_DATE('{trade_date}','YYYY-MM-DD'),{open}, {close},
{high},{low}, {volume})"
            con.execute(insert_sql)
        con.execute("COMMIT;")
    except Exception as e:
        con.execute("ROLLBACK;")
        print(e)
        raise e
```

8. (+1) Demonstrate your work ensures Idempotency by running your pipeline (from extract to load) twice in a row and checking the number of records (the number needs to remain the same)

```
def check_table_stats(table,con):
    result = con.execute(f"SELECT * FROM {table} order by symbol,
trade_date")
    df = con.fetch_pandas_all()
    print(df.head())
    print(df.tail())
    print("The count is ",len(df))
def stock_api(symbol):
    con = return_snowflake_conn()
    data = return_last_90d_price(symbol)
    load(con, data,symbol)
    check_table_stats("user_db_hedgehog.raw.stock_api",con)
    con.close()
def check_idempotency(symbol):
    con = return_snowflake_conn()
    data = return_last_90d_price(symbol)
    load(con, data,symbol)
    check_table_stats("user_db_hedgehog.raw.stock_api",con)
    con.close()
```

Idempotency Demo

We:

1. Count rows after first run
2. Run the pipeline **again**
3. Count rows again → the number should remain the same

Running for the first time

Running the second time

[45] ✓ 43s

```
check_idempotency("ELV")
```

```
  SYMBOL  TRADE_DATE    OPEN   CLOSE    HIGH      LOW   VOLUME
0    ELV  2025-07-02  367.20  350.25  374.58  349.615  4805803
1    ELV  2025-07-03  355.01  347.84  357.24  347.210  2006213
2    ELV  2025-07-07  348.00  346.88  350.50  343.720  2256047
3    ELV  2025-07-08  345.75  349.46  350.68  344.900  1619591
4    ELV  2025-07-09  347.00  346.36  348.00  342.630  1822210
   SYMBOL  TRADE_DATE     OPEN   CLOSE      HIGH       LOW   VOLUME
57    ELV  2025-09-23  317.275  323.73  324.9100  317.2750  2118644
58    ELV  2025-09-24  324.610  326.38  329.0500  319.8500  2096779
59    ELV  2025-09-25  326.000  316.70  328.0400  315.5200  1860727
60    ELV  2025-09-26  319.350  318.61  320.1724  316.2801  1482009
61    ELV  2025-09-29  318.310  319.69  321.6300  316.0100  1139336
The count is  62
```

[44] ✓ 45s

```
stock_api("ELV")
```

```
  SYMBOL  TRADE_DATE    OPEN   CLOSE    HIGH      LOW   VOLUME
0    ELV  2025-07-02  367.20  350.25  374.58  349.615  4805803
1    ELV  2025-07-03  355.01  347.84  357.24  347.210  2006213
2    ELV  2025-07-07  348.00  346.88  350.50  343.720  2256047
3    ELV  2025-07-08  345.75  349.46  350.68  344.900  1619591
4    ELV  2025-07-09  347.00  346.36  348.00  342.630  1822210
   SYMBOL  TRADE_DATE     OPEN   CLOSE      HIGH       LOW   VOLUME
57    ELV  2025-09-23  317.275  323.73  324.9100  317.2750  2118644
58    ELV  2025-09-24  324.610  326.38  329.0500  319.8500  2096779
59    ELV  2025-09-25  326.000  316.70  328.0400  315.5200  1860727
60    ELV  2025-09-26  319.350  318.61  320.1724  316.2801  1482009
61    ELV  2025-09-29  318.310  319.69  321.6300  316.0100  1139336
The count is  62
```

9. (+2) Follow today's demo and capture Docker Desktop screen showing Airflow