Name : Mrunali Katta

Student ID: 017516785

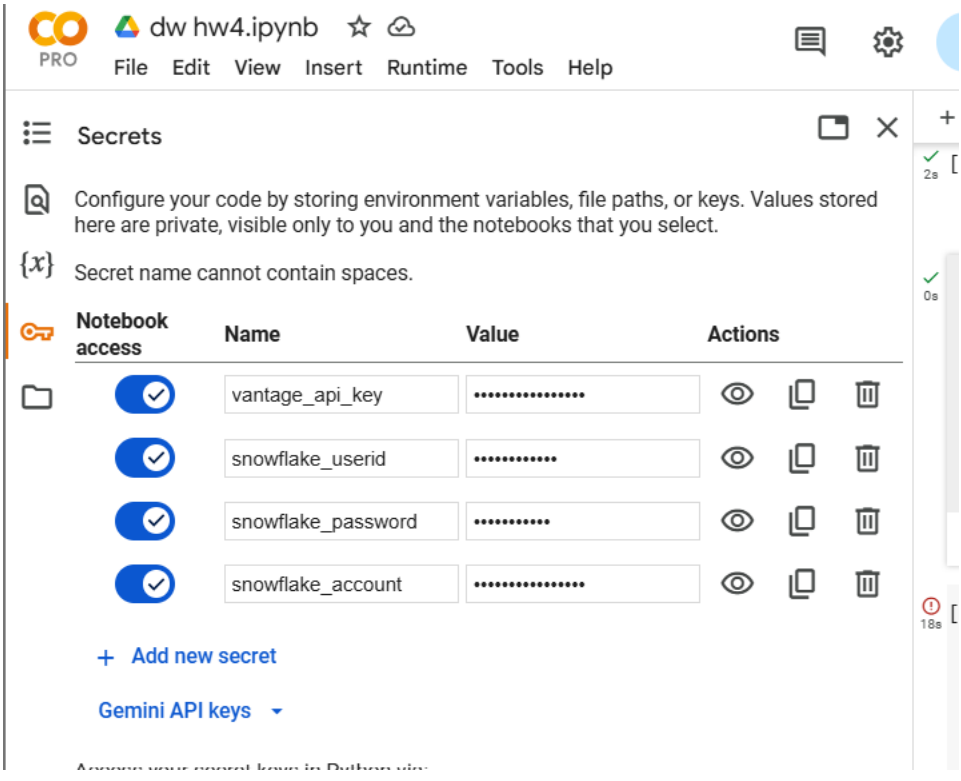
DATA 226 - Homework 04

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

I registered on the Alpha Vantage website and received an API key . Then chose 'IBM' as the stock symbol for retrieving daily stock price data using the Alpha Vantage API.

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

I have secured my Alpha Vantage API key and Snowflake credentials using environment variables in Google Colab, ensuring they are not exposed in my source code or any public repositories




```
from google.colab import userdata

# to fetch the stored API key that is in Secrets
vantage_api_key = userdata.get('vantage_api_key')

print("API Key is stored securely")
```

```
import snowflake.connector

# fetch Snowflake credentials
```

```
sf_user = userdata.get('snowflake_userid')
sf_password = userdata.get('snowflake_password')
sf_account = userdata.get('snowflake_account')

# establish connection to Snowflake
conn = snowflake.connector.connect(
    user=sf_user,
    password=sf_password,
    account=sf_account
)

print("Successfully connected to Snowflake!")
```

```
[110] from google.colab import userdata

      # to fetch the stored API key that is in Secrets
      vantage_api_key = userdata.get('vantage_api_key')

      print("API Key is stored securely")
```

```
API Key is stored securely
```

```
[112] import snowflake.connector

      # fetch Snowflake credentials
      sf_user = userdata.get('snowflake_userid')
      sf_password = userdata.get('snowflake_password')
      sf_account = userdata.get('snowflake_account')

      # establish connection to Snowflake
      conn = snowflake.connector.connect(
          user=sf_user,
          password=sf_password,
          account=sf_account
      )

      print("Successfully connected to Snowflake!")
```

```
Successfully connected to Snowflake!
```

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")
1. With regard to adding "date", please look at the next slide

```
def return_last_90d_price(symbol):

    """ Returns the last 90 days of the stock prices for the given symbol
        Includes 'date', 'open', 'high', 'low', 'close', 'volume'   """
    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}&outputsize=compact'
    r = requests.get(url)
```

```
    data = r.json()

    # Extract time series data
    time_series = data.get("Time Series (Daily)", {})
    # Calculate 90 days ago from today
    ninety_days_ago = (datetime.now() - timedelta(days=90)).strftime('%Y-%m-%d')
    results = [
        {"date": d, "open": data["Time Series (Daily)"][d]["1. open"],
         "high": data["Time Series (Daily)"][d]["2. high"],
         "low": data["Time Series (Daily)"][d]["3. low"],
         "close": data["Time Series (Daily)"][d]["4. close"],
         "volume": data["Time Series (Daily)"][d]["5. volume"]}
        for d in time_series if d >= ninety_days_ago
    ]
    return results
#for IBM stock
price_list = return_last_90d_price("IBM")
# Print first 10 entries
for entry in price_list[:10]:
    print(entry)
```

```
[143] def return_last_90d_price(symbol):
        """ Returns the last 90 days of the stock prices for the given symbol
            Includes 'date', 'open', 'high', 'low', 'close', 'volume'   """
        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}&outputsize=compact'
        r = requests.get(url)
        data = r.json()

        # Extract time series data
        time_series = data.get("Time Series (Daily)", {})
        # Calculate 90 days ago from today
        ninety_days_ago = (datetime.now() - timedelta(days=90)).strftime('%Y-%m-%d')
        results = [
            {"date": d, "open": data["Time Series (Daily)"][d]["1. open"],
             "high": data["Time Series (Daily)"][d]["2. high"],
             "low": data["Time Series (Daily)"][d]["3. low"],
             "close": data["Time Series (Daily)"][d]["4. close"],
             "volume": data["Time Series (Daily)"][d]["5. volume"]}
            for d in time_series if d >= ninety_days_ago
        ]
        return results
```

```
#for IBM stock
price_list = return_last_90d_price("IBM")
# Print first 10 entries
for entry in price_list[:10]:
    print(entry)
```

```
{'date': '2025-02-26', 'open': '258.1000', 'high': '258.3250', 'low': '254.4104', 'close': '255.8400', 'volume': '3460124'}
{'date': '2025-02-25', 'open': '261.0800', 'high': '263.4800', 'low': '256.7700', 'close': '257.7500', 'volume': '6292487'}
{'date': '2025-02-24', 'open': '261.5000', 'high': '263.8450', 'low': '259.5800', 'close': '261.8700', 'volume': '4398107'}
{'date': '2025-02-21', 'open': '263.8450', 'high': '264.8300', 'low': '261.1000', 'close': '261.4800', 'volume': '5667874'}
{'date': '2025-02-20', 'open': '263.6500', 'high': '265.0900', 'low': '262.1500', 'close': '264.7400', 'volume': '4884805'}
{'date': '2025-02-19', 'open': '262.0000', 'high': '264.3600', 'low': '260.0900', 'close': '264.3200', 'volume': '3718678'}
{'date': '2025-02-18', 'open': '261.9300', 'high': '263.9650', 'low': '259.8300', 'close': '263.0700', 'volume': '4262812'}
{'date': '2025-02-14', 'open': '259.0000', 'high': '261.9400', 'low': '257.9100', 'close': '261.2800', 'volume': '3925277'}
{'date': '2025-02-13', 'open': '255.6600', 'high': '259.2800', 'low': '254.4100', 'close': '259.1900', 'volume': '4531538'}
{'date': '2025-02-12', 'open': '252.7200', 'high': '256.4000', 'low': '252.0200', 'close': '255.8100', 'volume': '3075308'}
```

4. (+1) Create a table under “raw” schema if it doesn’t exist to capture the info from the API
1. symbol, date, open, close, high, low, volume: symbol and date should be **primary keys**

```
# to create : "dev" database , "raw" schema
cur.execute("CREATE DATABASE dev;")
print("Database 'dev' created.")
```

```
cur.execute("USE DATABASE dev;")

# Create the "raw" schema if it doesn't exist
cur.execute("CREATE SCHEMA raw;")
print("Schema 'raw' created.")
cur.execute("USE SCHEMA raw;")

# Create table if not exists
cur.execute("""
CREATE TABLE IF NOT EXISTS stock_price (
    symbol STRING,
    date DATE,
    open FLOAT,
    high FLOAT,
    low FLOAT,
    close FLOAT,
    volume INT,
    PRIMARY KEY (symbol, date)
);
""")

print("Table 'stock_price' created in Snowflake!")
```

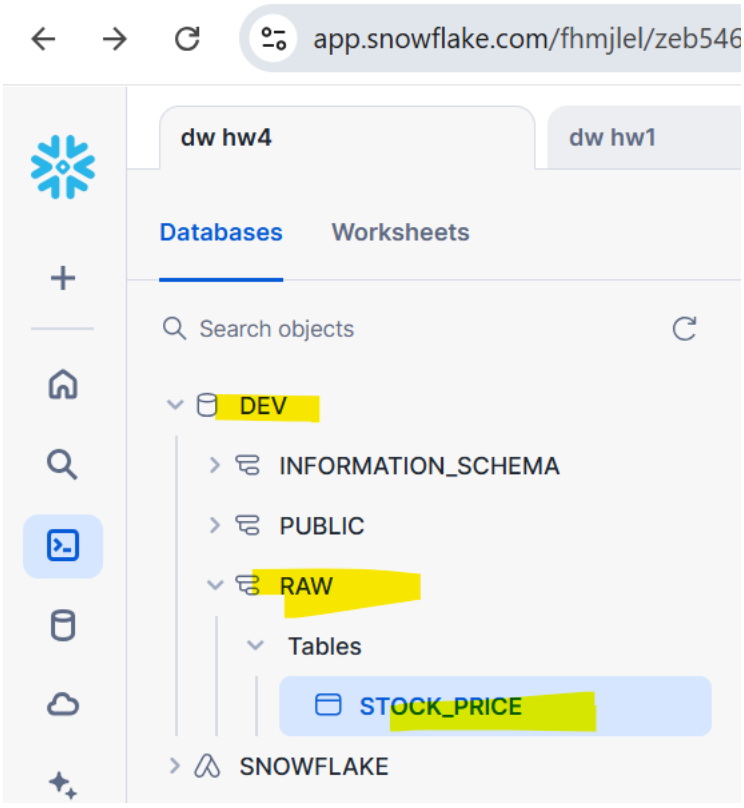
0s [117] Start coding or generate with AI.

0s [145]
```
# to create : "dev" database , "raw" schema
cur.execute("CREATE DATABASE dev;")
print("Database 'dev' created.")
cur.execute("USE DATABASE dev;")

# Create the "raw" schema if it doesn't exist
cur.execute("CREATE SCHEMA raw;")
print("Schema 'raw' created.")
cur.execute("USE SCHEMA raw;")
```

```
Database 'dev' created.
Schema 'raw' created.
<snowflake.connector.cursor.SnowflakeCursor at 0x7d52d20
```

[150]
```
# Create table if not exists
cur.execute("""
CREATE TABLE IF NOT EXISTS stock_price (
    symbol STRING,
    date DATE,
    open FLOAT,
    high FLOAT,
    low FLOAT,
    close FLOAT,
    volume INT,
    PRIMARY KEY (symbol, date)
);
""")

print("Table 'stock_price' created in Snowflake!")
```

```
Table 'stock_price' created in Snowflake!
```



5. (+1) Delete all records from the table

```
cur.execute("DELETE FROM dev.raw.stock_price;")
print("All previous records deleted from 'stock_price' table.")
```

```
cur.execute("DELETE FROM dev.raw.stock_price;")
print("All previous records deleted from 'stock_price' table.")
```

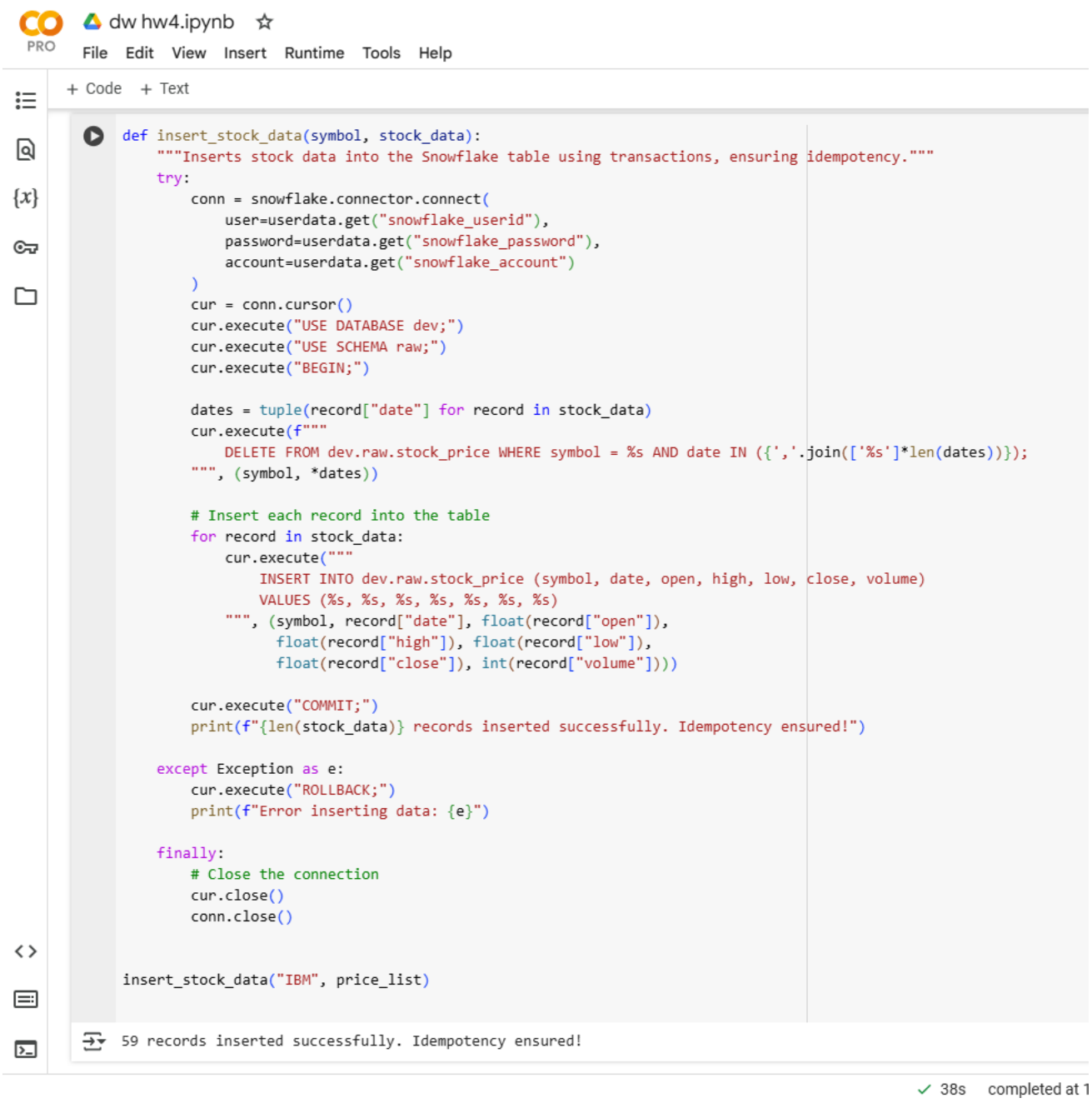
```
All previous records deleted from 'stock_price' table.
```

```
[196] Start coding or generate with AI.
```

6. (+1) Populate the table with the records from step 2 using INSERT SQL (refer to [the relevant code snippetLinks to an external site.](#) as a starting point)

```
def insert_stock_data(symbol, stock_data):
    """Inserts stock data into the Snowflake table using transactions, ensuring idempotency."""
    try:
        conn = snowflake.connector.connect(
            user=userdata.get("snowflake_userid"),
            password=userdata.get("snowflake_password"),
            account=userdata.get("snowflake_account")
        )
        cur = conn.cursor()
        cur.execute("USE DATABASE dev;")
        cur.execute("USE SCHEMA raw;")
        cur.execute("BEGIN;")
```

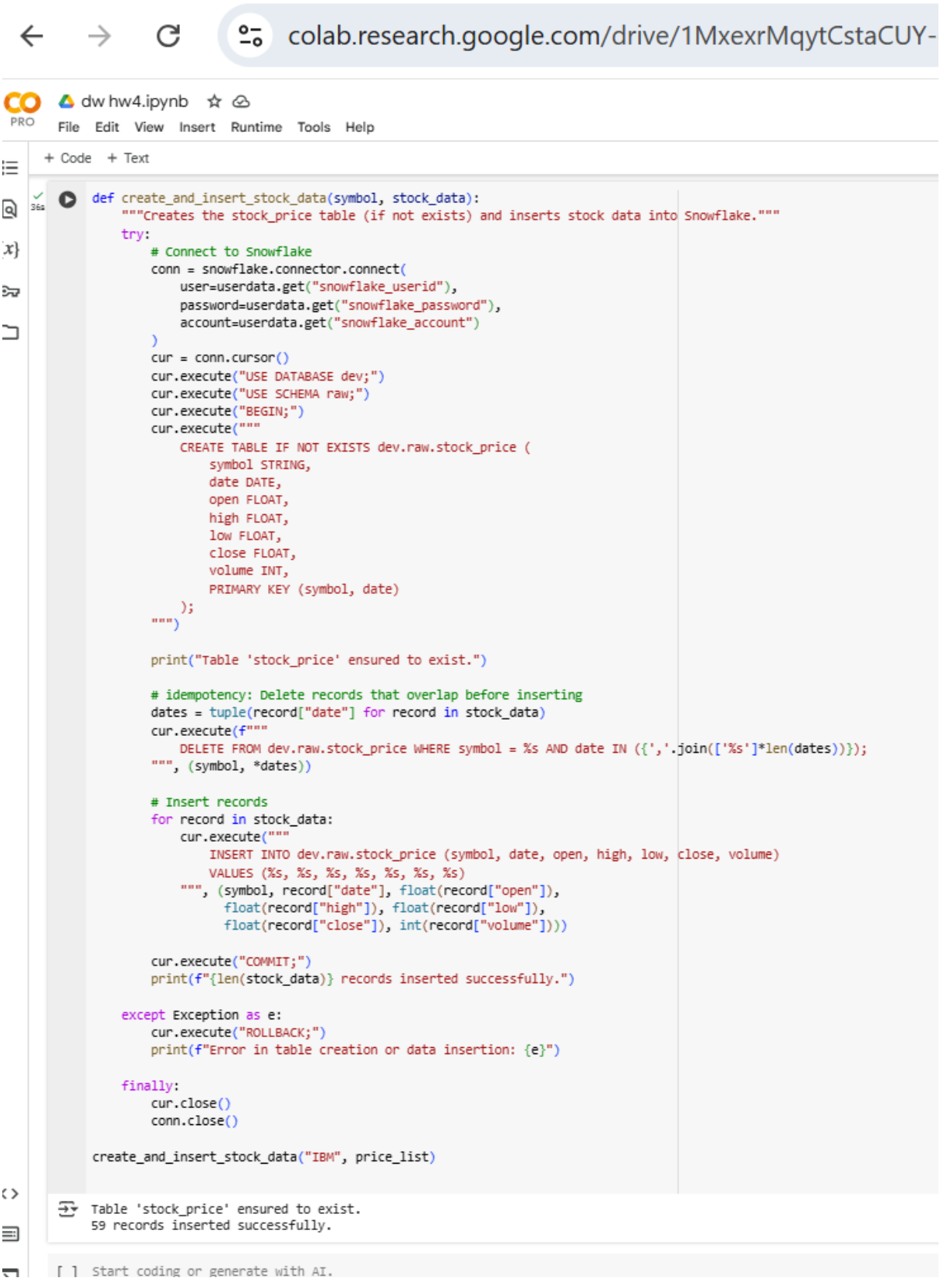
```
        dates = tuple(record["date"] for record in stock_data)
        cur.execute(f"""
            DELETE FROM dev.raw.stock_price WHERE symbol = %s AND date IN
({','.join(['%s']*len(dates))});
        """, (symbol, *dates))

        # Insert each record into the table
        for record in stock_data:
            cur.execute("""
                INSERT INTO dev.raw.stock_price (symbol, date, open, high, low,
close, volume)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, (symbol, record["date"], float(record["open"]),
                  float(record["high"]), float(record["low"]),
                  float(record["close"]), int(record["volume"])))

        cur.execute("COMMIT;")
        print(f"{len(stock_data)} records inserted successfully. Idempotency
ensured!")

    except Exception as e:
        cur.execute("ROLLBACK;")
        print(f"Error inserting data: {e}")

    finally:
        # Close the connection
        cur.close()
        conn.close()
insert_stock_data("IBM", price_list)
```



7. (+4) Steps 4 and 6 need to be done together
1. Use try/except along with SQL transaction. (use [the code hereLinks to an external site.]() as reference)

```
def create_and_insert_stock_data(symbol, stock_data):
    """Creates the stock_price table (if not exists) and inserts stock data into
Snowflake."""
    try:
        # Connect to Snowflake
        conn = snowflake.connector.connect(
            user=userdata.get("snowflake_userid"),
            password=userdata.get("snowflake_password"),
            account=userdata.get("snowflake_account")
        )
        cur = conn.cursor()
```

```
        cur.execute("USE DATABASE dev;")
        cur.execute("USE SCHEMA raw;")
        cur.execute("BEGIN;")
        cur.execute("""
            CREATE TABLE IF NOT EXISTS dev.raw.stock_price (
                symbol STRING,
                date DATE,
                open FLOAT,
                high FLOAT,
                low FLOAT,
                close FLOAT,
                volume INT,
                PRIMARY KEY (symbol, date)
            );
        """)

        print("Table 'stock_price' ensured to exist.")

        # idempotency: Delete records that overlap before inserting
        dates = tuple(record["date"] for record in stock_data)
        cur.execute(f"""
            DELETE FROM dev.raw.stock_price WHERE symbol = %s AND date IN
({','.join(['%s']*len(dates))});
        """, (symbol, *dates))

        # Insert records
        for record in stock_data:
            cur.execute("""
                INSERT INTO dev.raw.stock_price (symbol, date, open, high, low,
close, volume)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, (symbol, record["date"], float(record["open"]),
                  float(record["high"]), float(record["low"]),
                  float(record["close"]), int(record["volume"])))

        cur.execute("COMMIT;")
        print(f"{len(stock_data)} records inserted successfully.")

    except Exception as e:
        cur.execute("ROLLBACK;")
        print(f"Error in table creation or data insertion: {e}")

    finally:
        cur.close()
        conn.close()

create_and_insert_stock_data("IBM", price_list)
```



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)

```python
def check_record_count():
    """Returns the total count of records in the stock_price table."""
    try:
        conn = snowflake.connector.connect(
            user=userdata.get("snowflake_userid"),
            password=userdata.get("snowflake_password"),
            account=userdata.get("snowflake_account")
        )
        cur = conn.cursor()

        cur.execute("USE DATABASE dev;")
        cur.execute("USE SCHEMA raw;")

        cur.execute("SELECT COUNT(*) FROM dev.raw.stock_price;")
        count = cur.fetchone()[0]

        return count

    except Exception as e:
        print(f"Error fetching record count: {e}")
        return None

    finally:
        cur.close()
        conn.close()

# running  the pipeline twice
print("Running first insert...")
create_and_insert_stock_data("IBM", price_list)
count_before = check_record_count()

print("Running second insert to test idempotency...")
create_and_insert_stock_data("IBM", price_list)
count_after = check_record_count()

print(f"Records before second run: {count_before}")
print(f"Records after second run: {count_after}")

# check the count remains the same
assert count_before == count_after, "Idempotency test failed!"
print("Idempotency test passed! No duplicate records inserted.")
```

```python
def check_record_count():
    """Returns the total count of records in the stock_price table."""
    try:
        conn = snowflake.connector.connect(
            user=userdata.get("snowflake_userid"),
            password=userdata.get("snowflake_password"),
            account=userdata.get("snowflake_account")
        )
        cur = conn.cursor()

        cur.execute("USE DATABASE dev;")
        cur.execute("USE SCHEMA raw;")

        cur.execute("SELECT COUNT(*) FROM dev.raw.stock_price;")
        count = cur.fetchone()[0]

        return count

    except Exception as e:
        print(f"Error fetching record count: {e}")
        return None

    finally:
        cur.close()
        conn.close()

# running  the pipeline twice
print("Running first insert...")
create_and_insert_stock_data("IBM", price_list)
count_before = check_record_count()

print("Running second insert to test idempotency...")
create_and_insert_stock_data("IBM", price_list)
count_after = check_record_count()

print(f"Records before second run: {count_before}")
print(f"Records after second run: {count_after}")

# check the count remains the same
assert count_before == count_after, "Idempotency test failed!"
print("Idempotency test passed! No duplicate records inserted.")
```

```
Running first insert...
Table 'stock_price' ensured to exist.
59 records inserted successfully.
Running second insert to test idempotency...
Table 'stock_price' ensured to exist.
59 records inserted successfully.
Records before second run: 59
Records after second run: 59
Idempotency test passed! No duplicate records inserted.
```

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