**Bài tập APACHE AIRFLOW**

Mục tiêu:

Thu thập dữ liệu giá vàng Mỹ (XAU USD) và so sánh với giá vàng Việt Nam (SJC) từ đó so sánh.

File goldDW.py chứa định nghĩa các hàm thực thi công việc lấy dữ liệu, DAG, task làm việc cho air flow

from datetime import timedelta

from airflow.models import DAG

from airflow.operators.python import PythonOperator

from airflow.utils.dates import days\_ago

import pandas as pd

import requests

import tarfile

import csv

import shutil

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from webdriver\_manager.chrome import ChromeDriverManager

from bs4 import BeautifulSoup

from lxml import etree

# Define the path for the input and output files

destination\_path = '/home/xuanquang/project/airflow'

# Function to invest xau usa data

def invest\_xauusd\_data():

options = webdriver.ChromeOptions()

options.add\_argument("--headless")

options.add\_argument("--disable-blink-features=AutomationControlled")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)

driver.get("https://uk.investing.com/currencies/xau-usd-historical-data")

html = driver.page\_source

driver.quit()

tree = etree.HTML(html)

table = tree.xpath('//table[contains(@class, "freeze-column-w-1")]')

if not table:

raise Exception("Not found table!")

table = table[0]

header\_elements = table.xpath('.//thead//tr/th//span[1]')

headers = [elem.text.strip() for elem in header\_elements if elem.text]

print("Headers:", headers)

rows = table.xpath('.//tbody//tr')

data = []

for row in rows:

cells = row.xpath('./td')

row\_data = []

for cell in cells:

texts = cell.xpath('.//text()')

cell\_text = ' '.join(t.strip() for t in texts if t.strip())

row\_data.append(cell\_text)

data.append(row\_data)

for d in data:

print(d)

columns\_to\_remove = []

for i, header in enumerate(headers):

col\_empty = True

for row in data:

if len(row) > i and row[i] != '':

col\_empty = False

break

if col\_empty:

columns\_to\_remove.append(i)

for col\_index in reversed(columns\_to\_remove):

del headers[col\_index]

for row in data:

if len(row) > col\_index:

del row[col\_index]

for d in data:

print(d)

with open("XAUUSD.csv", mode="w", newline="", encoding="utf-8") as file:

writer = csv.writer(file)

writer.writerow(headers)

writer.writerows(data)

# Function to invest usd vnd data

def invest\_usdvnd\_data():

options = webdriver.ChromeOptions()

options.add\_argument("--headless")

options.add\_argument("--disable-blink-features=AutomationControlled")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)

driver.get("https://vn.investing.com/currencies/usd-vnd-historical-data")

html = driver.page\_source

driver.quit()

tree = etree.HTML(html)

table = tree.xpath('//table[contains(@class, "freeze-column-w-1")]')

if not table:

raise Exception("Not found table!")

table = table[0]

header\_elements = table.xpath('.//thead//tr/th//span[1]')

headers = [elem.text.strip() for elem in header\_elements if elem.text]

print("Headers:", headers)

rows = table.xpath('.//tbody//tr')

data = []

for row in rows:

cells = row.xpath('./td')

row\_data = []

for cell in cells:

texts = cell.xpath('.//text()')

cell\_text = ' '.join(t.strip() for t in texts if t.strip())

row\_data.append(cell\_text)

data.append(row\_data)

for d in data:

print(d)

columns\_to\_remove = []

for i, header in enumerate(headers):

col\_empty = True

for row in data:

if len(row) > i and row[i] != '':

col\_empty = False

break

if col\_empty:

columns\_to\_remove.append(i)

for col\_index in reversed(columns\_to\_remove):

del headers[col\_index]

for row in data:

if len(row) > col\_index:

del row[col\_index]

for d in data:

print(d)

with open("USDVND.csv", mode="w", newline="", encoding="utf-8") as file:

writer = csv.writer(file)

writer.writerow(headers)

writer.writerows(data)

# Function to invest sjc data

def invest\_sjc\_data():

options = webdriver.ChromeOptions()

options.add\_argument("--headless")

options.add\_argument("--disable-blink-features=AutomationControlled")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)

driver.get("https://uk.investing.com/equities/phu-nhuan-jewelry-jsc-historical-data")

html = driver.page\_source

driver.quit()

tree = etree.HTML(html)

table = tree.xpath('//table[contains(@class, "freeze-column-w-1")]')

if not table:

raise Exception("Not found table!")

table = table[0]

header\_elements = table.xpath('.//thead//tr/th//span[1]')

headers = [elem.text.strip() for elem in header\_elements if elem.text]

print("Headers:", headers)

rows = table.xpath('.//tbody//tr')

data = []

for row in rows:

cells = row.xpath('./td')

row\_data = []

for cell in cells:

texts = cell.xpath('.//text()')

cell\_text = ' '.join(t.strip() for t in texts if t.strip())

row\_data.append(cell\_text)

data.append(row\_data)

for d in data:

print(d)

columns\_to\_remove = []

for i, header in enumerate(headers):

col\_empty = True

for row in data:

if len(row) > i and row[i] != '':

col\_empty = False

break

if col\_empty:

columns\_to\_remove.append(i)

for col\_index in reversed(columns\_to\_remove):

del headers[col\_index]

for row in data:

if len(row) > col\_index:

del row[col\_index]

for d in data:

print(d)

with open("SJC.csv", mode="w", newline="", encoding="utf-8") as file:

writer = csv.writer(file)

writer.writerow(headers)

writer.writerows(data)

# Function to extract sjc data from CSV

def extract\_sjc\_data\_from\_csv():

input\_file = f"{destination\_path}/SJC.csv"

output\_file = f"{destination\_path}/sjcdata.csv"

with open(input\_file, 'r', encoding='utf-8-sig', newline='') as infile:

reader = csv.reader(infile)

header = next(reader, None)

with open(output\_file, 'w', newline='', encoding='utf-8') as outfile:

writer = csv.writer(outfile)

writer.writerow(['Date', 'Price'])

for row in reader:

writer.writerow([row[0], row[1]])

# Function to extract xauusd data from CSV

def extract\_xauusd\_data\_from\_csv():

input\_file = f"{destination\_path}/XAUUSD.csv"

output\_file = f"{destination\_path}/xauusddata.csv"

with open(input\_file, 'r', encoding='utf-8-sig', newline='') as infile:

reader = csv.reader(infile)

header = next(reader, None)

with open(output\_file, 'w', newline='', encoding='utf-8') as outfile:

writer = csv.writer(outfile)

writer.writerow(['Date', 'Price'])

for row in reader:

writer.writerow([row[0], row[1]])

# Function to extract usdvnd data from CSV

def extract\_usdvnd\_data\_from\_csv():

input\_file = f"{destination\_path}/USDVND.csv"

output\_file = f"{destination\_path}/usdvnddata.csv"

with open(input\_file, 'r', encoding='utf-8-sig', newline='') as infile:

reader = csv.reader(infile)

header = next(reader, None)

with open(output\_file, 'w', newline='', encoding='utf-8') as outfile:

writer = csv.writer(outfile)

writer.writerow(['Date', 'Price'])

for row in reader:

writer.writerow([row[0], row[1]])

# Function to consolidate data

def consolidate\_and\_compare\_data():

input\_file1 = f"{destination\_path}/sjcdata.csv"

input\_file2 = f"{destination\_path}/xauusddata.csv"

input\_file3 = f"{destination\_path}/usdvnddata.csv"

df\_sjc = pd.read\_csv(input\_file1, usecols=["Date", "Price"], encoding='utf-8-sig')

df\_xau = pd.read\_csv(input\_file2, usecols=["Date", "Price"], encoding='utf-8-sig')

df\_usd = pd.read\_csv(input\_file3, usecols=["Date", "Price"], encoding='utf-8-sig')

df\_sjc.columns = ["Date", "SJC\_Price"]

df\_xau.columns = ["Date", "XAU\_Price"]

df\_usd.columns = ["Date", "USD\_Price"]

df\_merged = df\_sjc.merge(df\_xau, on="Date", how="outer")

df\_merged = df\_merged.merge(df\_usd, on="Date", how="outer")

df\_merged["XAU\_Price"] = df\_merged["XAU\_Price"].str.replace(',', '', regex=False)

df\_merged["USD\_Price"] = df\_merged["USD\_Price"].str.replace(',', '', regex=False)

df\_merged["SJC\_Price"] = df\_merged["SJC\_Price"].str.replace(',', '', regex=False)

df\_merged["XAU\_Price"] = pd.to\_numeric(df\_merged["XAU\_Price"], errors='coerce')

df\_merged["USD\_Price"] = pd.to\_numeric(df\_merged["USD\_Price"], errors='coerce')

df\_merged["SJC\_Price"] = pd.to\_numeric(df\_merged["SJC\_Price"], errors='coerce')

df\_merged["XAUUSD\_Price"] = df\_merged["XAU\_Price"] \* df\_merged["USD\_Price"]

df\_merged["Compare"] = (df\_merged["XAUUSD\_Price"] / df\_merged["SJC\_Price"]\*1000)\*100

output\_file = f"{destination\_path}/golddata.csv"

df\_merged.to\_csv(output\_file, index=False, encoding='utf-8-sig')

# Default arguments for the DAG

default\_args = {

'owner': 'Your name',

'start\_date': days\_ago(0),

'email': ['your email'],

'retries': 1,

'retry\_delay': timedelta(minutes=5),

}

# Define the DAG

dag = DAG(

'Gold\_price\_data',

default\_args=default\_args,

description='Apache Airflow Final Assignment',

schedule\_interval=timedelta(days=1),

)

# Define the tasks

invest\_sjc\_task = PythonOperator(

task\_id='invest\_sjc\_data',

python\_callable=invest\_sjc\_data,

dag=dag,

)

invest\_xauusd\_task = PythonOperator(

task\_id='invest\_xauusd\_data',

python\_callable=invest\_xauusd\_data,

dag=dag,

)

invest\_usdvnd\_task = PythonOperator(

task\_id='invest\_usdvnd\_data',

python\_callable=invest\_usdvnd\_data,

dag=dag,

)

extract\_sjc\_task = PythonOperator(

task\_id='extract\_sjc\_data\_from\_csv',

python\_callable=extract\_sjc\_data\_from\_csv,

dag=dag,

)

extract\_xauusd\_task = PythonOperator(

task\_id='extract\_xauusd\_data\_from\_csv',

python\_callable=extract\_xauusd\_data\_from\_csv,

dag=dag,

)

extract\_usdvnd\_task = PythonOperator(

task\_id='extract\_usdvnd\_data\_from\_csv',

python\_callable=extract\_usdvnd\_data\_from\_csv,

dag=dag,

)

consolidate\_and\_compare\_task = PythonOperator(

task\_id='consolidate\_and\_compare\_data',

python\_callable=consolidate\_and\_compare\_data,

dag=dag,

)

# Set the task dependencies

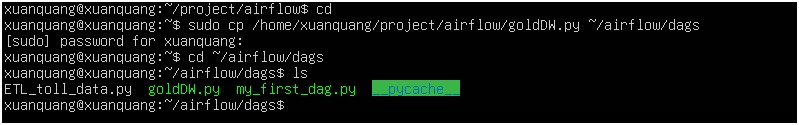
[invest\_sjc\_task, invest\_xauusd\_task, invest\_usdvnd\_task] >> [extract\_sjc\_task, extract\_xauusd\_task, extract\_usdvnd\_task] >> consolidate\_and\_compare\_task

**Copy file code vào $AIRFLOW\_HOME/dags**

$ sudo cp /home/xuanquang/project/airflow/ETL\_toll\_data.py $AIRFLOW\_HOME/dags

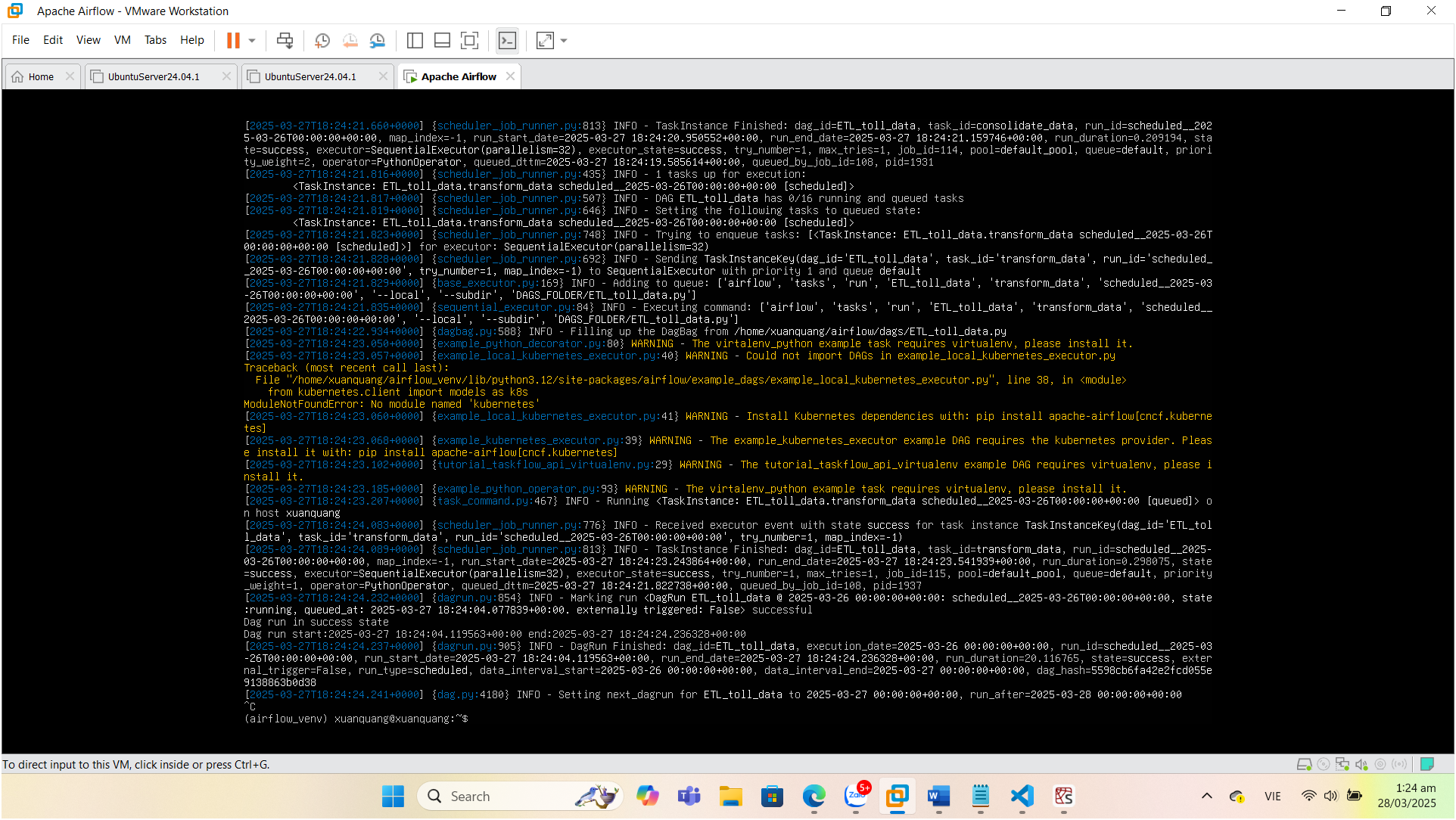
sudo cp /home/xuanquang/project/airflow/goldDW.py $AIRFLOW\_HOME/dags

sudo cp /home/xuanquang/project/airflow/goldDW.py ~/airflow/dags



* **Khởi động Airflow Scheduler và Web server nếu chưa khởi động**

$ airflow scheduler



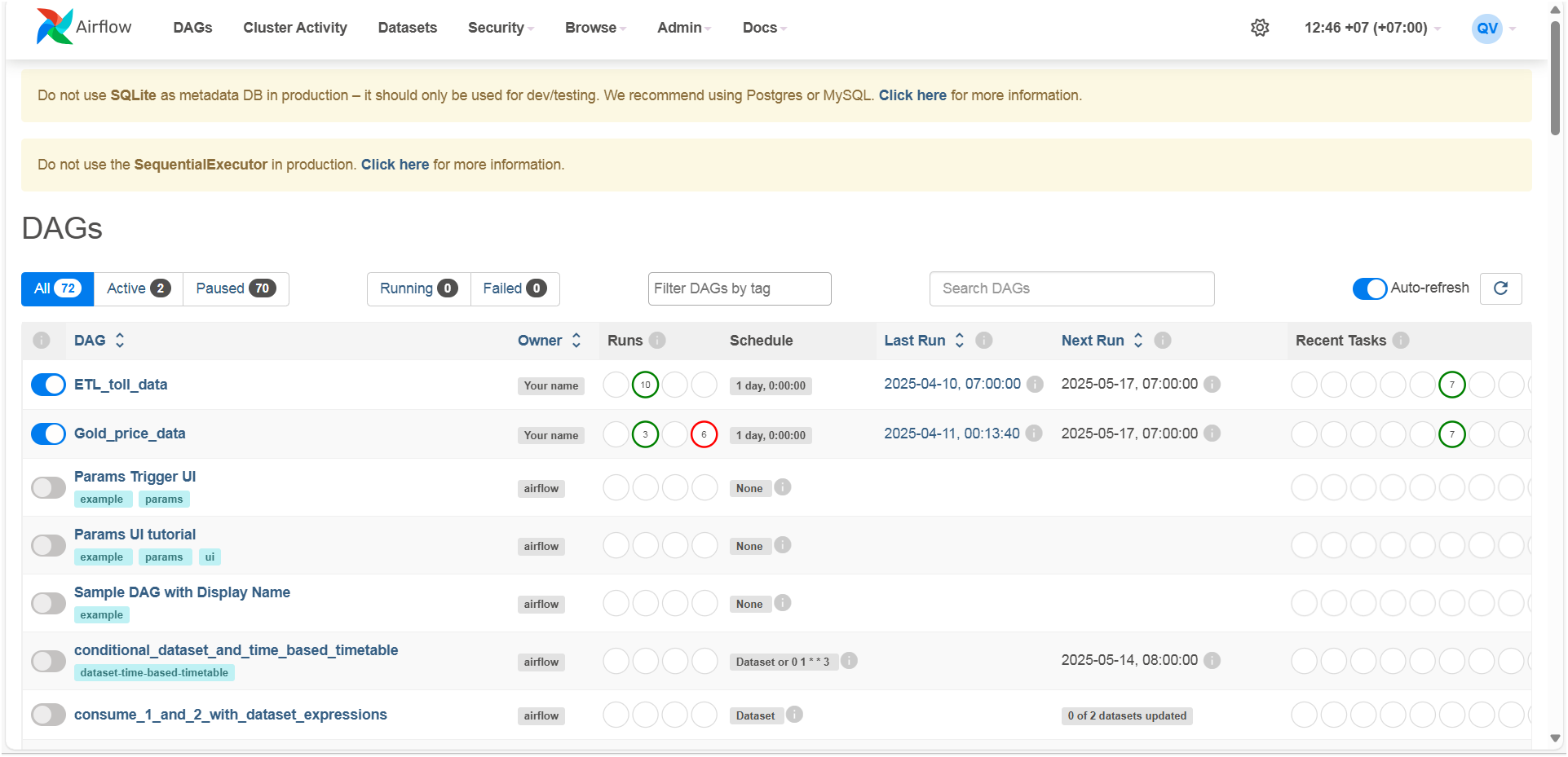
$ airflow webserver --port 8080

airflow webserver -p 8080 &

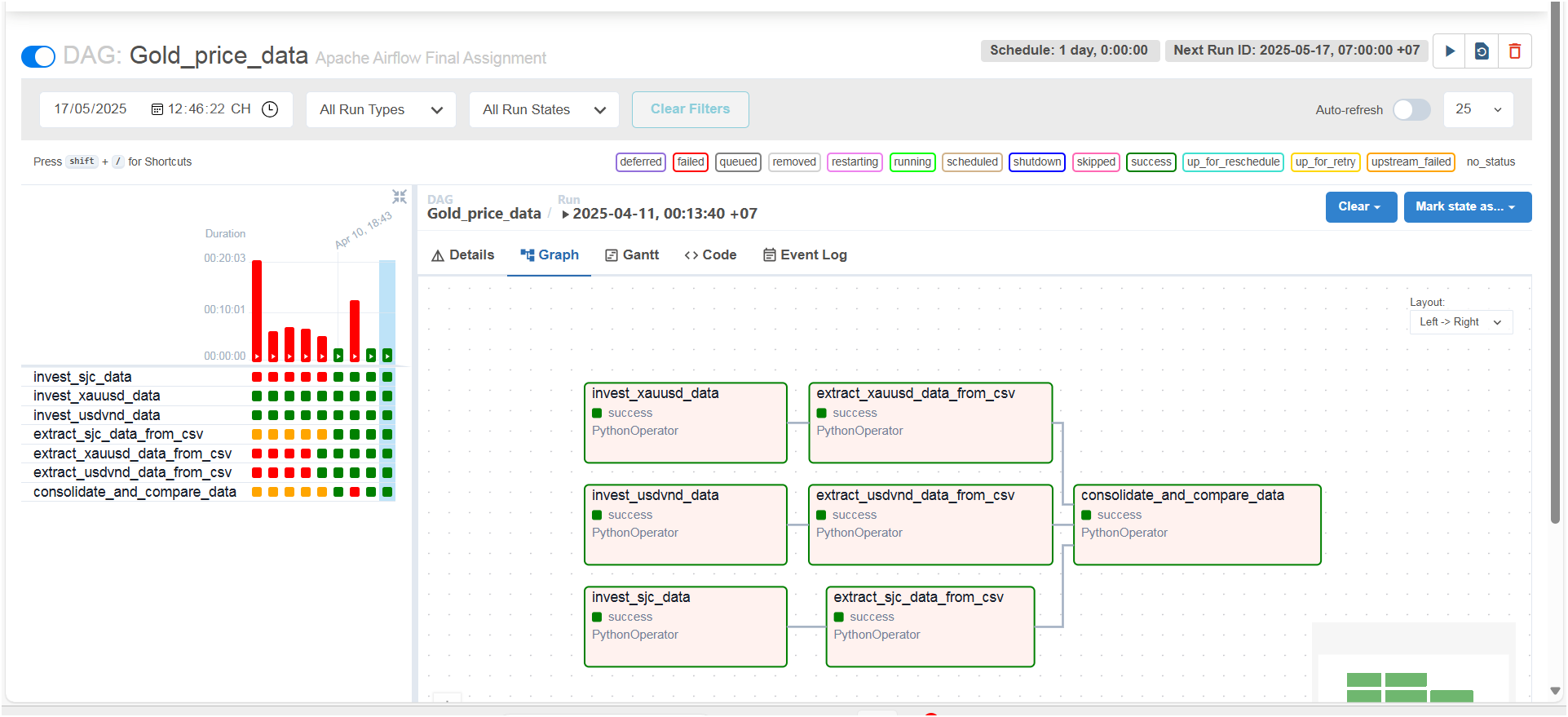
**Nếu chưa khởi động môi trường ảo sẽ bị lỗi khi chạy 2 lệnh trên:**

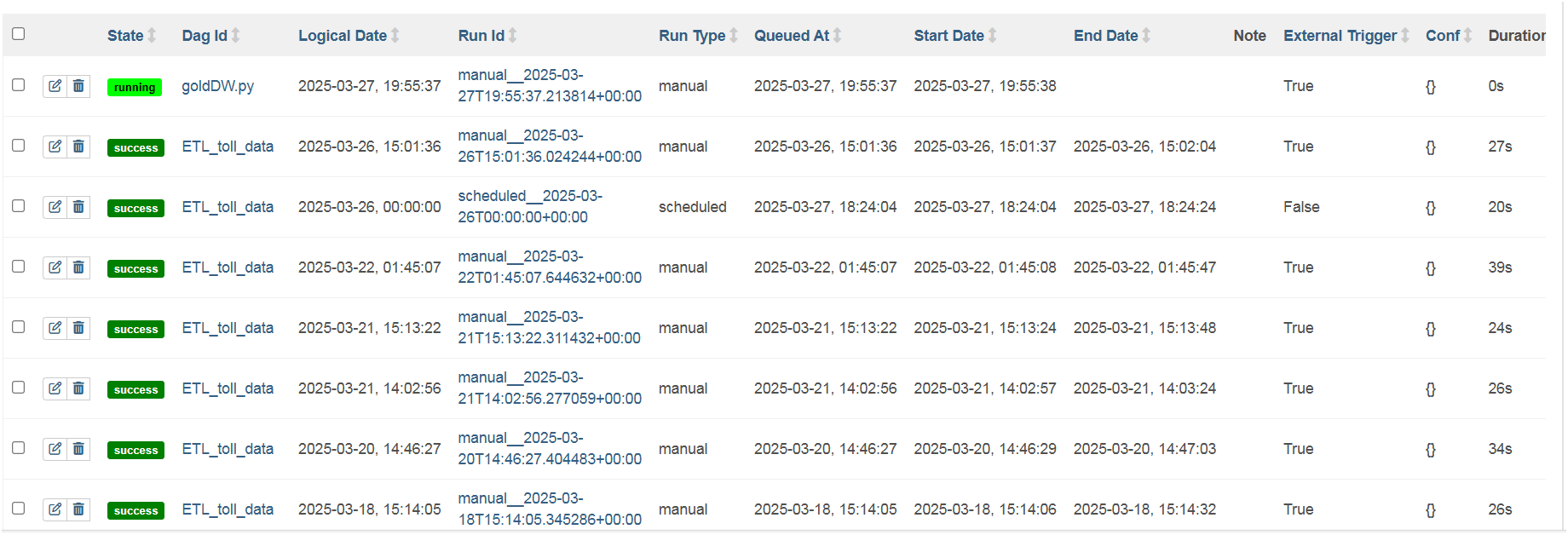
$ source ./airflow\_venv/bin/activate

• **Kiểm tra trên Web server hoặc liệt kê DAGs LIST để xem ETL vừa tạo có tồn tại không**

****

**Khởi động và theo dõi ETL hoạt động**

****

****

**Các task success hết -> thành công !**