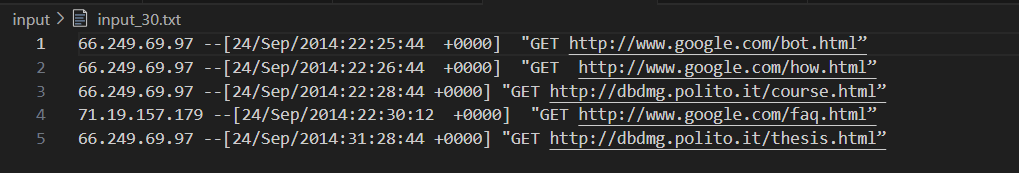
## **Exercise #30: Log filtering**

**Input**: Một nhật ký đơn giản của máy chủ web (tức là một tệp văn bản).Mỗi dòng của tệp liên quan đến một URL request.

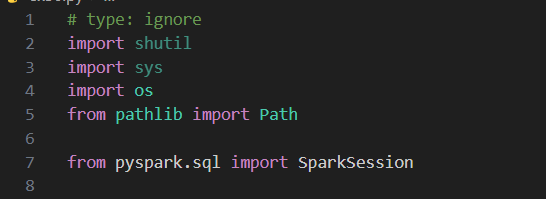


**Output**: Các dòng chứa từ “google” và lưu trữ lại

**Cần xử lý:** Lọc các dòng chứa từ “google”

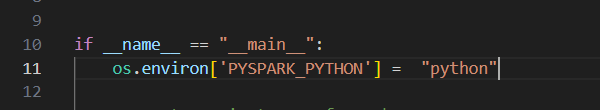
**Code thực hiện**:

*Import thư viện cần thiết*



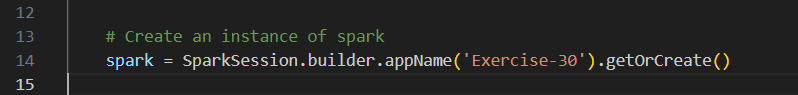
* shutil, sys, os: dùng để quản lý hệ thống tệp và đường dẫn
* Path: cung cấp cách quản lý đường dẫn dễ dàng và an toàn
* SparkSession: tạo một phiên spark để sử lý các tác vụ xử lý dữ liệu

*Thiết lập môi trường*



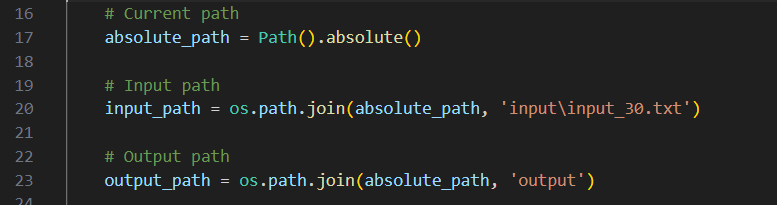
Cài đặt biến môi trường để chỉ định rằng *Python* sẽ sử dụng làm trình thông dịch *PySpark*

*Tạo một SparkSession*



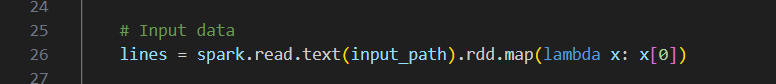
* SparkSession.builder : Khởi tạo một phiên Spark
* .appName(‘Exercise-30’): Đặt tên là ‘Exercise-30’
* .getOrCreate(): Tạo phiên Spark nếu chưa tồn tại hoặc lấy phiên Spark đang chạy

*Xác định đường dẫn tệp*



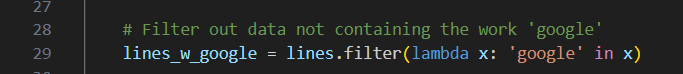
* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* input\_path: đường dẫn đầu vào chứa nhật ký web
* output\_path: đường dẫn thư mục dến nơi lưu kết quả

*Đọc và xử lý tệp đầu vào*



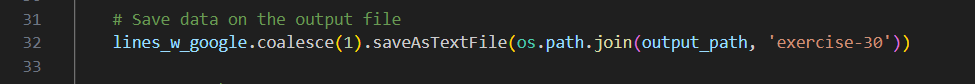
* spark.read.text(input\_path): đọc nội dung tệp đầu vào dưới dạng DataFrame
* .rdd: chuyển đổi DataFrame thành RDD (Resilient Distributed Dataset) để thực hiện xử lý dữ liệu dạng song song
* .map(lambda x: x[0]): trích xuất nội dung của từng dòng trong tệp

*Lọc các dòng chứa từ “google”*



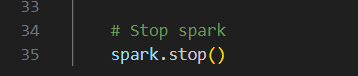
.filter(lambda x: ‘google’ in x): giữ lại các dòng có chứa từ ‘google’

*Lưu kết quả vào thư mục đầu ra*



* .coalesce(1): giảm số số lượng phân vùng RDD xuống còn 1 để ghi dữ liệu vào một tệp duy nhất
* .saveAsTextFile(): lưu kết quả vào mục ‘exercise-30’ bên trong thư mục output

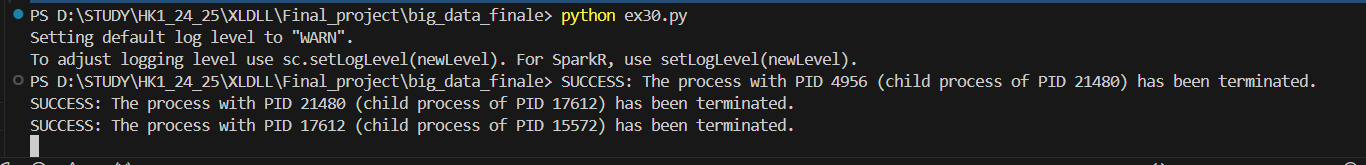
*Dừng phiên Spark*



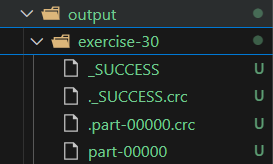
Dừng phiên Spark sau khi xử lý xong để giải phóng tài nguyên

**Kết quả:**

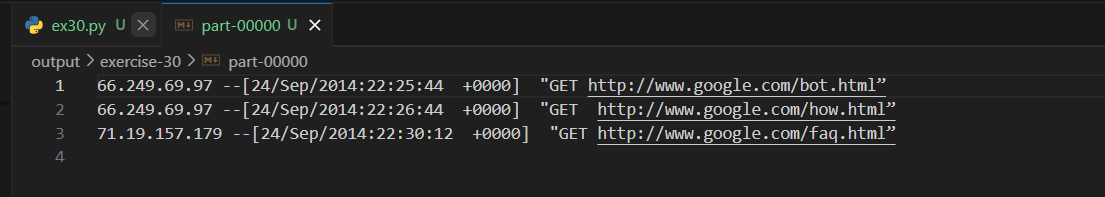
*Chạy code*



*Output được lưu vào thư mục exercise-30/part-00000*

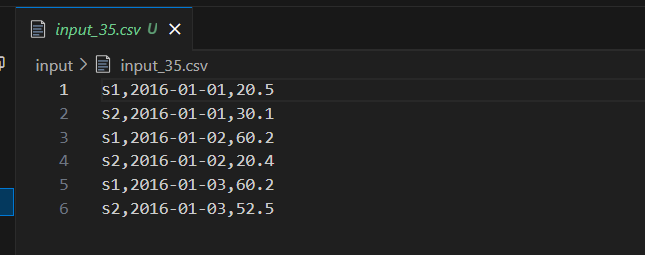


*File output*



## **Exercise #35: Dates associated with the maximum value**

**Input**: Một tập các tệp csv văn bản (có cấu trúc) chứa các giá trị hằng ngày của PM10 từ một tập hợp các cảm biến. Mỗi dòng có dạng: sensorld, date, PM10 value (µg/m3)

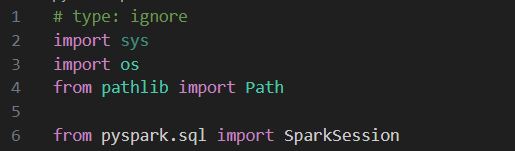


**Output**: Các ngày liên quan đến giá trị lớn nhất của PM10 và được lưu lại

**Cần xử lý:** Tìm ra giá trị PM10 lớn nhất trong dữ liệu và lọc ra ngày có giá trị PM10 bằng với giá trị lớn nhất

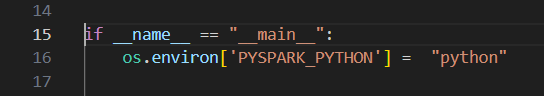
**Code thực hiện**:

*Import thư viện cần thiết*



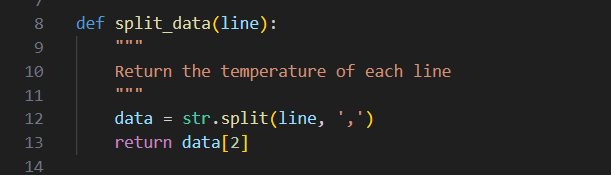
* os và Path: dùng để quản lý đường dẫn tệp
* SparkSession: tạo phiên làm việc cho PySpark

*Thiết lập môi trường*



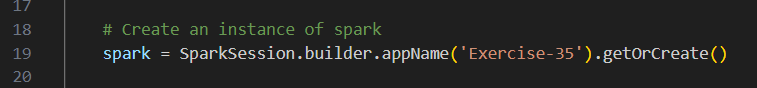
Cài đặt biến môi trường để chỉ định rằng *Python* sẽ sử dụng làm trình thông dịch *PySpark*

*Hàm split\_data*



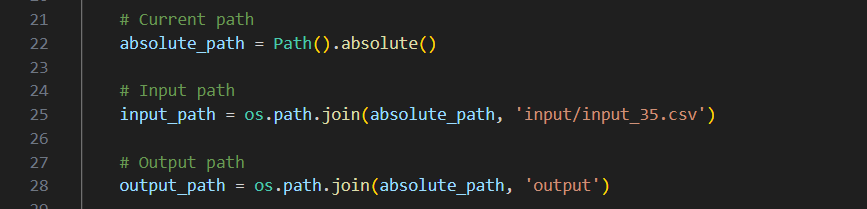
* Hàm nhận vào một dòng dữ liệu từ tệp csv (là chuỗi ký tự)
* Phân tách các dòng theo dấu phẩy và trả về cột thứ 3 (giá trị PM10)

*Tạo SparkSession*



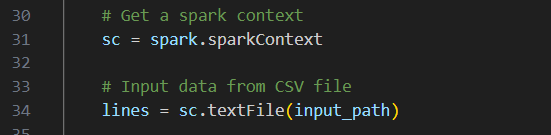
Khởi tạo phiên làm việc của PySpark với tên là Exercise-35

*Xác định đường dẫn tệp*



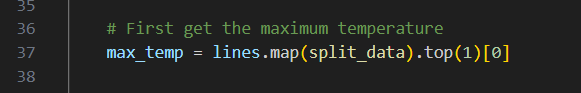
* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* input\_path: xác định đường dẫn đến tệp CSV đầu vào
* output\_path: xác định đường dẫn để lưu kết quả đầu ra

*Đọc dữ liệu từ tệp CSV*



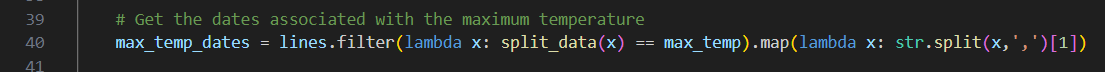
sc.textFile(input\_path): Đọc dữ liệu tệp CSV và trả về một RDD, trong đó mỗi dòng dữ liệu là một phần tử

*Tìm giá trị PM10 lớn nhất*



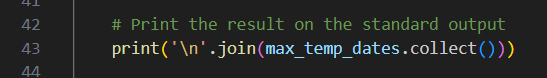
* lines.map(split\_data): lấy giá trị PM10 từ các dòng bằng cách sử dụng hàm split\_data
* .top(1)[0]: lấy giá trị lớn nhất trong các giá trị PM10

*Lọc các dòng có giá trị PM10 lớn nhất và lấy ngày tương ứng*



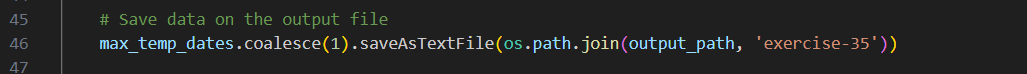
* lines.filter(lambda x: split\_data(x) == max\_temp): giữ lại các dòng có giá trị PM10 lớn nhất
* .map(lambda x: str.split(x, ‘,’)[1]): lấy cột thứ 2 (cột ngày) từ các dòng có giá trị PM10 lớn nhất

*In kết quả ra màn hình*



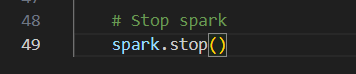
max\_temp\_dates.collect(): Thu thập ngày có giá trị PM10 lớn nhất thành một danh sách

*Lưu kết quả vào tệp*



* .coalesce(1): giảm số số lượng phân vùng RDD xuống còn 1 để ghi dữ liệu vào một tệp duy nhất
* .saveAsTextFile(): lưu kết quả vào mục ‘exercise-35’ bên trong thư mục output

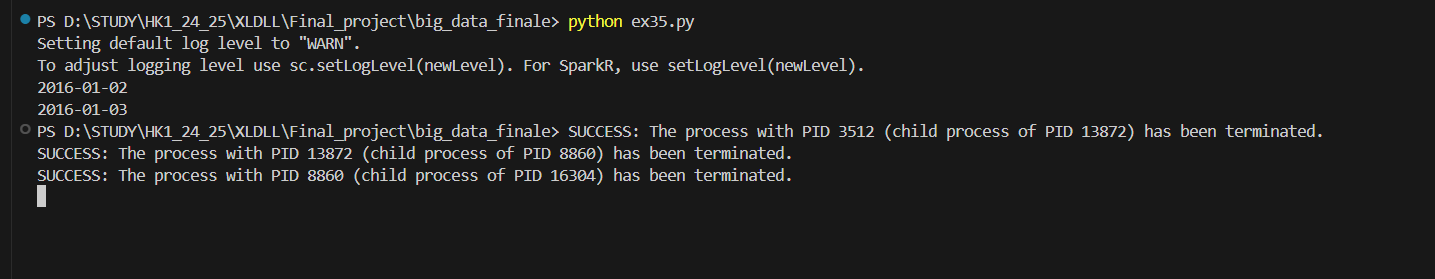
*Dừng Spark*



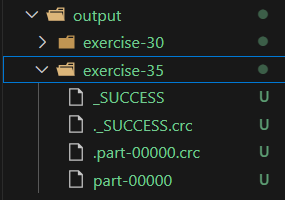
Dừng phiên Spark sau khi xử lý xong để giải phóng tài nguyên

**Kết quả:**

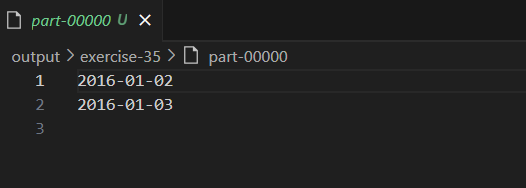
*Chạy code*



*Output được lưu vào thư mục exercise-35/part-00000*

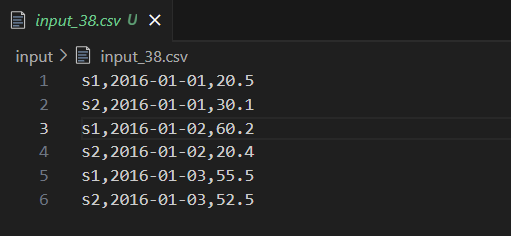


*File output*



## **Exercise #38: Pullution analysis**

**Input**: Một file CSV văn bản chứa giá trị PM10 hằng ngày từ một tập hợp các cảm biến. Mỗi dòng có định dạng: sensorId,date,PM10 value (µg/m³)

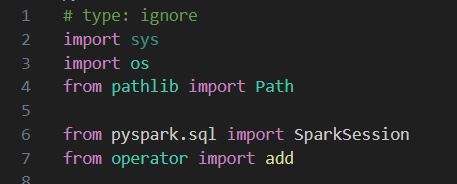


**Output**: Các cảm biến có ít nhất 2 lần ghi nhận giá trị PM10 vượt ngưỡng 50. Lưu trữ bao gồm sensorId và số lần mà cảm biến có giá trị PM10 vượt ngưỡng

**Cần xử lý:** Tìm ra các cảm biến có giá trị PM10 lớn hơn 50, đếm số lượng vượt ngưỡng của từng cảm biến và lọc các cảm biến có ít nhất 2 lần vượt ngưỡng

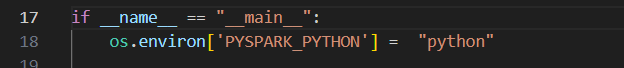
**Code thực hiện**:

*Import thư viện cần thiết*



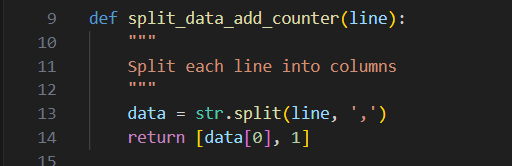
* os, sys và Path: dùng để quản lý đường dẫn tệp
* SparkSession: tạo phiên làm việc cho PySpark
* add từ operator: để thực hiện phép cộng khi giảm dữ liệu (reduceByKey)

*Thiết lập môi trường*



Cài đặt biến môi trường để chỉ định rằng *Python* sẽ sử dụng làm trình thông dịch *PySpark*

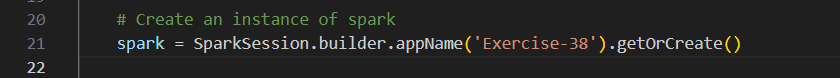
*Định nghĩa hàm split\_data\_add\_counter*



Tách các dòng dữ liệu CSV thành các cột và ánh xạ sensorId với giá trị 1

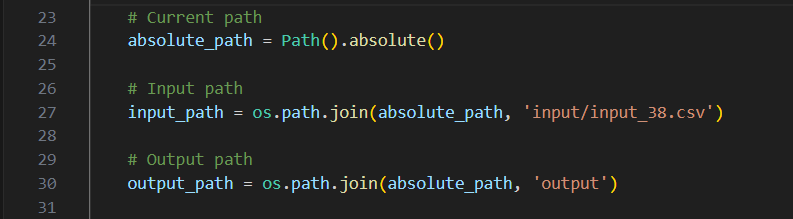
* Tách theo dấu phẩy
* Trả về giá trị [sensorId, 1] để sử dụng cho việc đếm số lần xuất hiện

*Tạo SparkSession*



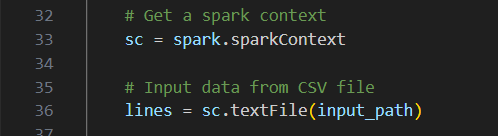
Khởi tạo phiên làm việc của PySpark với tên là Exercise-38

*Xác định đường dẫn tệp*



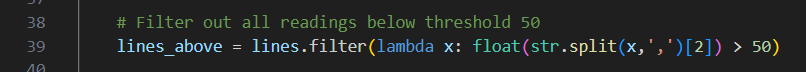
* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* input\_path: xác định đường dẫn đến tệp CSV đầu vào
* output\_path: xác định đường dẫn để lưu kết quả đầu ra

*Đọc dữ liệu từ tệp*



sc.textFile(input\_path): Đọc dữ liệu tệp CSV và trả về một RDD, trong đó mỗi dòng dữ liệu là một phần tử

*Lọc giá trị PM10 lớn hơn 50*



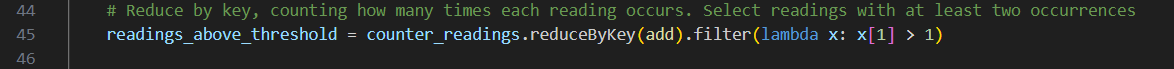
* float(str.split(x, ‘,’)[2]) > 50 : tách giá trị PM10 ở cột thứ 3 và so sánh với 50
* .filter(): lọc các dòng thoả điều kiện (giá trị PM10 lớn hơn 50)

*Ánh xạ mỗi cảm biến với bộ đếm*



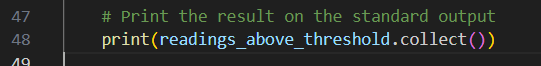
Sử dụng hàm *split\_data\_add\_counter* để gắn mỗi dòng đã lọc với [sensorId, 1]

*Tính số lần vượt ngưỡng trong cảm biến*



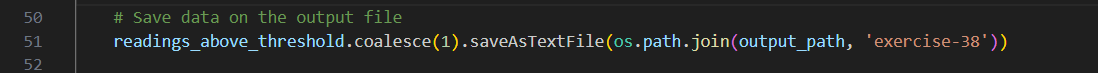
* .reduceByKey(add): Tổng hợp số lần mỗi cảm biến (sensorId) vượt ngưỡng 50
* .filter(lambda x: x[1] > 1): Chỉ giữ lại các cảm biến có ít nhất 2 lần vượt ngưỡng

*Thu thập kết quả*



.collect(): chuyển kết quả từ RDD sang danh sách Python để in ra

*Lưu kết quả vào tệp*



* .coalesce(1): giảm số số lượng phân vùng RDD xuống còn 1 để ghi dữ liệu vào một tệp duy nhất
* .saveAsTextFile(): lưu kết quả vào mục ‘exercise-38’ bên trong thư mục output

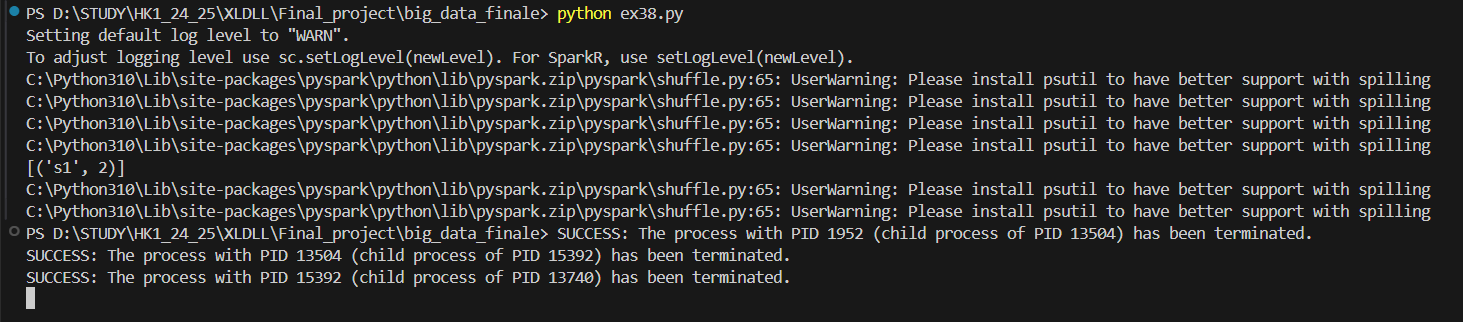
*Dừng Spark*



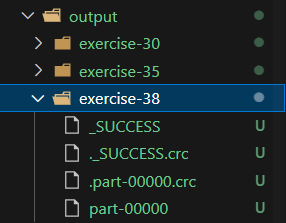
Dừng phiên Spark sau khi xử lý xong để giải phóng tài nguyên

**Kết quả:**

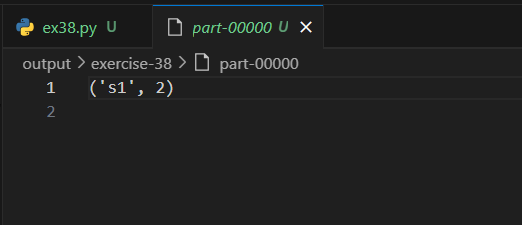
*Chạy code*



*Output được lưu vào thư mục exercise-38/part-00000*

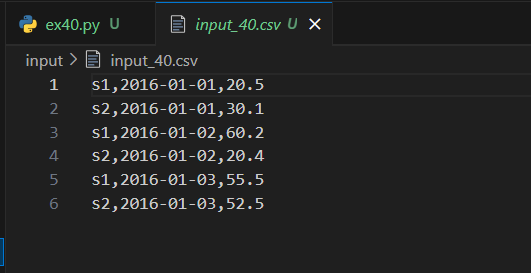


*File output*



## **Exercise #40: Order sensors by number of critical days**

**Input**: Một file CSV văn bản chứa giá trị PM10 hằng ngày từ một tập hợp các cảm biến. Mỗi dòng có định dạng: sensorId,date,PM10 value (µg/m³)

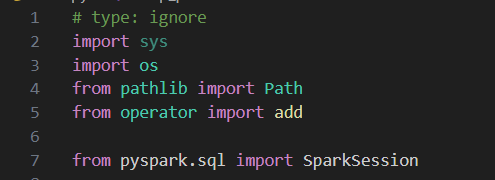


**Output**:Danh sách các cảm biến được sắp xếp theo số ngày vượt ngưỡng. Lưu trữ bao gồm sensorId và số ngày mà cảm biến có giá trị PM10 vượt ngưỡng

**Cần xử lý:** Xác định số ngày mà giá trị PM10 vượt ngưỡng 50 cho từng cảm biến. Sắp xếp các cảm biến theo số ngày vượt ngưỡng giảm dần.

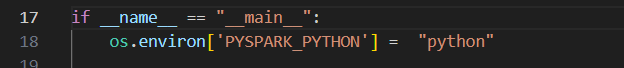
**Code thực hiện**:

*Import thư viện cần thiết*



* os, sys và Path: dùng để quản lý đường dẫn tệp
* SparkSession: tạo phiên làm việc cho PySpark
* add từ operator: để thực hiện phép cộng khi giảm dữ liệu (reduceByKey)

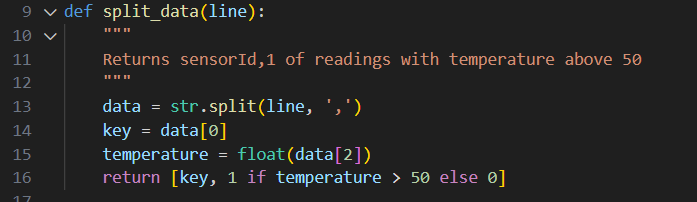
*Thiết lập môi trường*



Cài đặt biến môi trường để chỉ định rằng *Python* sẽ sử dụng làm trình thông dịch *PySpark*

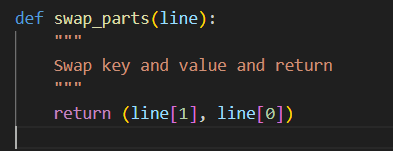
*Định nghĩa hàm hỗ trợ*

Hàm *split\_data*



Phân tích từng dòng dữ liệu và kiểm tra giá trị PM10 có vượt ngưỡng 50 hay là không

Hàm *swap\_parts*



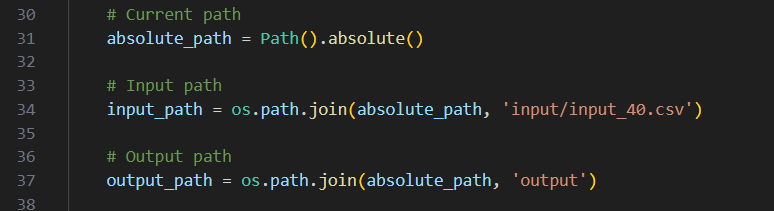
Đổi chỗ để tiện cho việc sắp xếp theo số ngày vượt ngưỡng

*Tạo SparkSession*



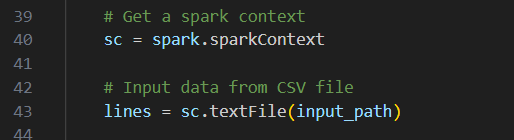
Khởi tạo phiên làm việc của PySpark với tên là Exercise-40

*Xác định đường dẫn tệp*



* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* input\_path: xác định đường dẫn đến tệp CSV đầu vào
* output\_path: xác định đường dẫn để lưu kết quả đầu ra

*Đọc dữ liệu từ tệp*



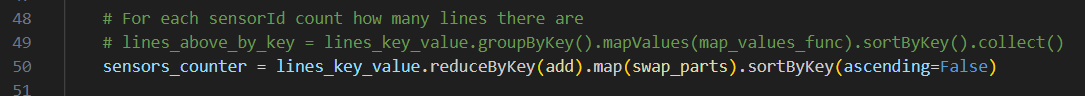
sc.textFile(input\_path): Đọc dữ liệu tệp CSV và trả về một RDD, trong đó mỗi dòng dữ liệu là một phần tử

*Chuyển đổi dữ liệu*



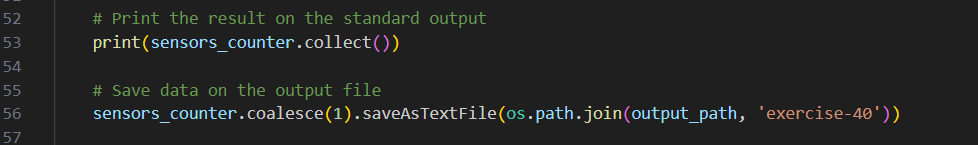
.map(split\_data): áp dụng hàm split\_data cho từng dòng, trả về dạng danh sách cặp (sensorId, 1)

*Tính tổng số ngày vượt ngưỡng của mỗi sensor*



* .reduceByKey(add): Tổng cộng số vượt ngưỡng của từng sensor
* .map(swap\_part): Đổi vị trí key-value thành (số ngày, sensorId)
* .sortByKey(ascending=False): Sắp xếp theo số ngày giảm dần

*In kết quả và lưu trữ*



* .coalesce(1): giảm số số lượng phân vùng RDD xuống còn 1 để ghi dữ liệu vào một tệp duy nhất
* .saveAsTextFile(): lưu kết quả vào mục ‘exercise-40’ bên trong thư mục output

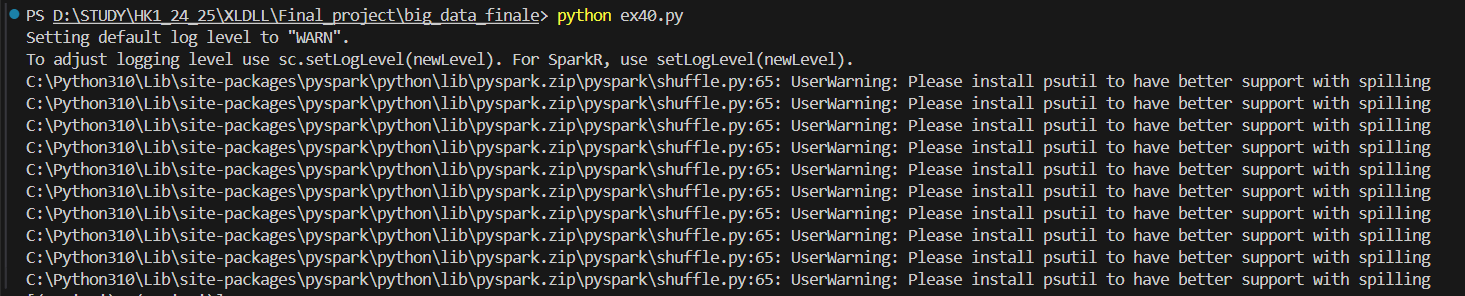
*Dừng Spark*

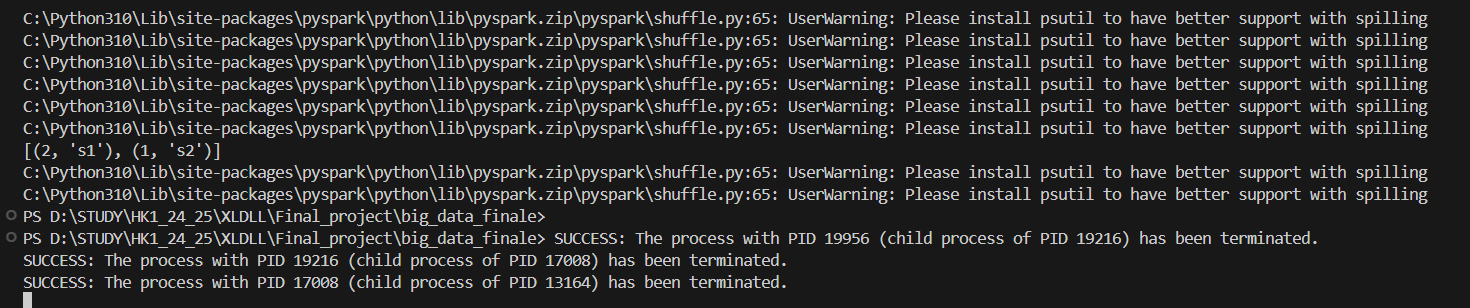


Dừng phiên Spark sau khi xử lý xong để giải phóng tài nguyên

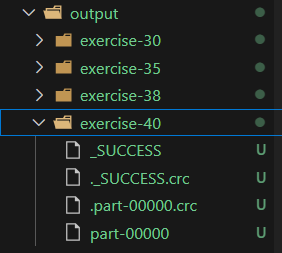
**Kết quả:**

*Chạy code*

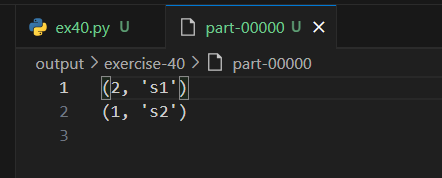




*Output được lưu vào thư mục exercise-40/part-00000*



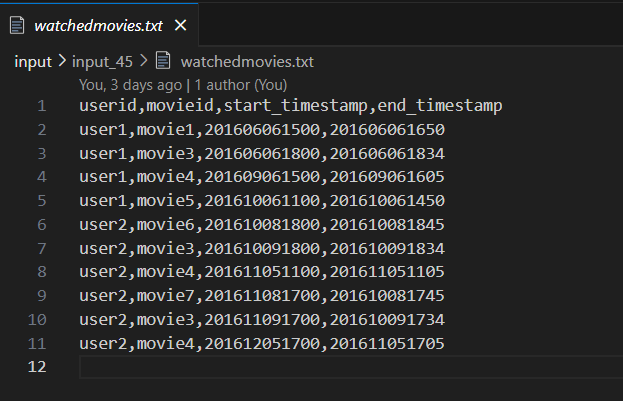
*File output*



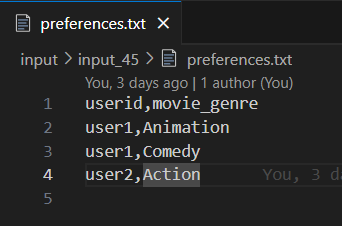
## **Exercise #45: Profile update**

**Input**:

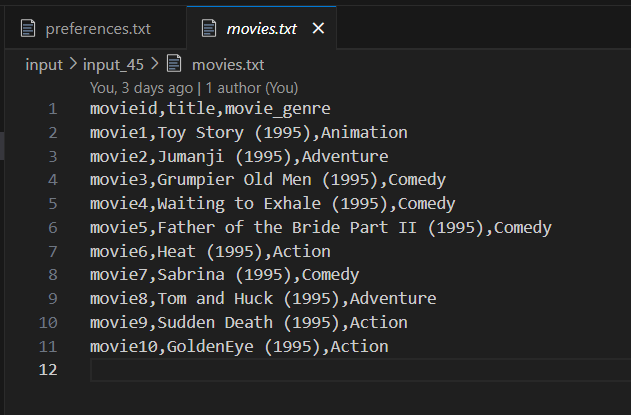
* Một file văn bản chứa danh sách bộ phim người dùng đã xem. Mỗi dòng có định dạng: userid,movieid,start-timestamp,end-timestamp



* Một file văn bản chứa danh sách sở thích của người dùng. Mỗi dòng có định dạng: userid,movie-genre



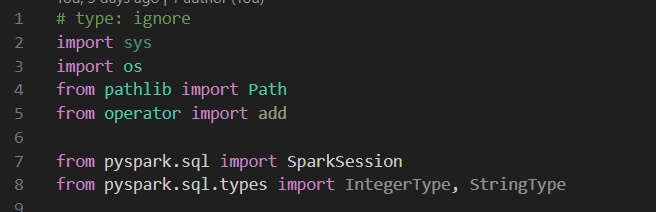
* Một file chứa danh sách bộ phim với thông tin liên quan. Mỗi dòng có định dạng: movieid,title,movie-genre



**Output**:Chọn danh sách các thể loại phim không thuộc vào thể loại phim yêu thích của người dùng (thể loại gây hiểu nhầm) nhưng lại liên quan ít nhất 5 bộ phim của người dùng đã xem

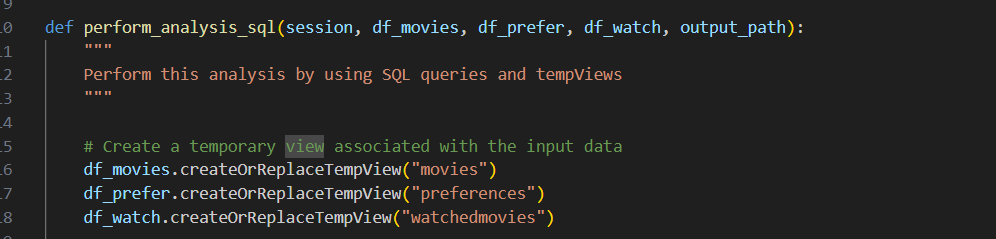
**Code thực hiện**:

*Import thư viện cần thiết*

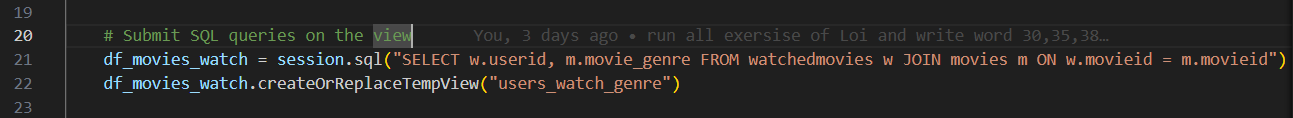


* os, sys và Path: dùng để quản lý đường dẫn tệp
* SparkSession: tạo phiên làm việc cho PySpark
* add từ operator: để thực hiện phép cộng khi giảm dữ liệu (reduceByKey)
* IntergerType, StringType: Xác định kiểu dữ liệu trong Spark DataFrame

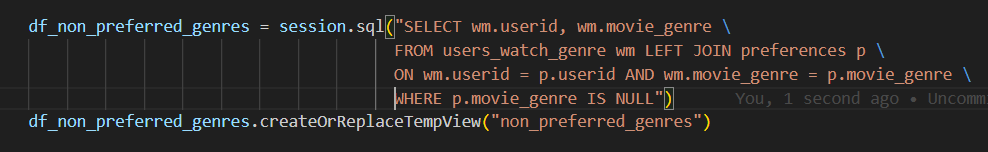
*Hàm perform\_analysis\_sql*



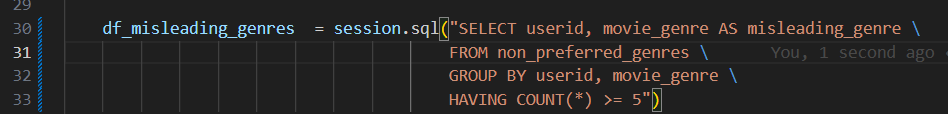
* Hàm nhận vào 4 tham số:
  + session: Phiên của Spark
  + df\_movies, df\_prefer, df\_watch: Các DataFrame chứa dữ liệu đầu vào
  + output\_path: Đường dẫn đầu ra để lưu kết quả
* .createOrReplaceTempView : Tạo các tempView từ DataFrame để có thể sử dụng truy vấn SQL từ Spark SQL
* *Bước 1*: Kết hợp thông tin người dùng với thể loại phim đã xem



* *Bước 2*: Tìm thể loại phim người dùng đã xem nhưng không thuộc sở thích



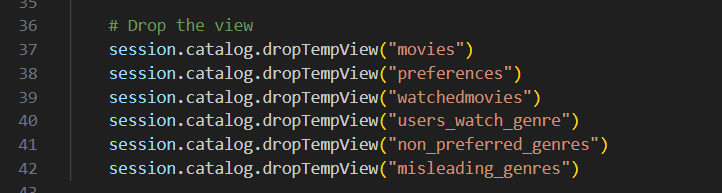
* *Bước 3*: Lọc ra các thể loại gây hiểu nhằm



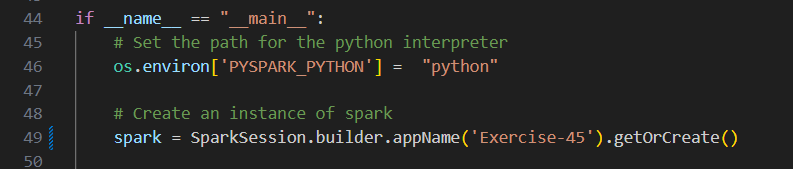
* *Bước 4*: Lưu kết quả



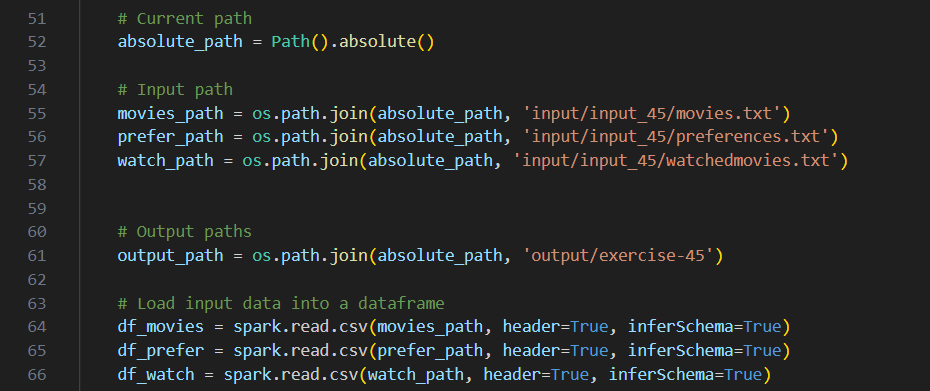
* + Ghi kết quả vào đường dẫn output\_path ở dạng csv
  + .repartition(1): đảm bảo rằng kết quả lưu vào 1 tệp duy nhất
  + mode= ‘overwrite’ : ghi đè nếu tệp có tồn tại
* *Bước 5*: Xoá các TempView

  
*Chương trình chạy chính*

* Thiết lập Spark



* Đường dẫn và đọc dữ lệu



* Gọi hàm *perform\_analysis\_sql*

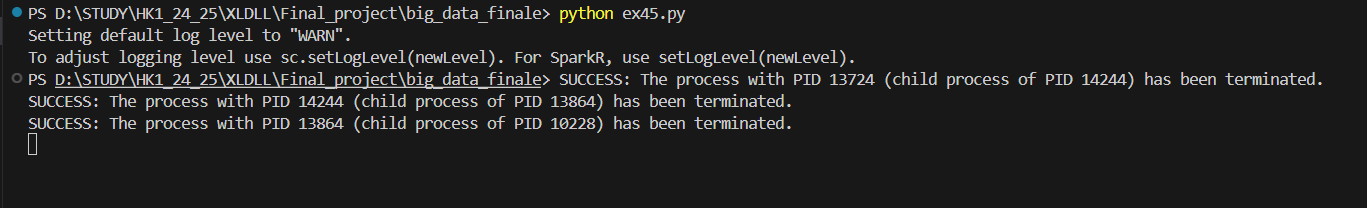


* Dừng Spark

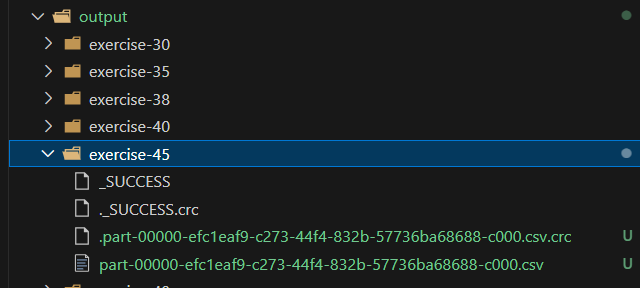


**Kết quả:**

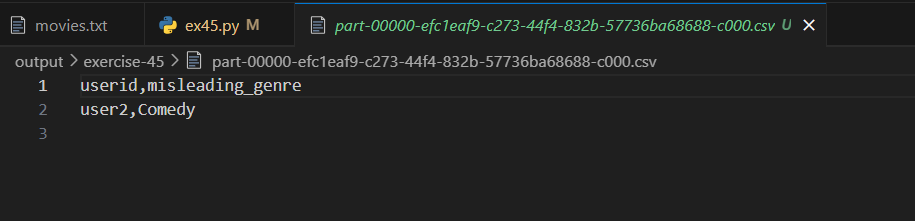
*Chạy code*



*Output được lưu vào thư mục exercise-45/part-00000-\*.csv*

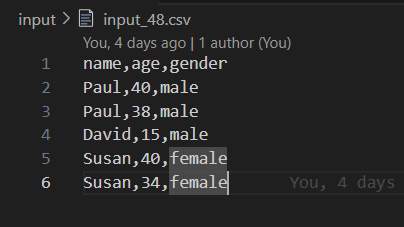


*File output*



## **Exercise #48: Average Age for Recurring Names**

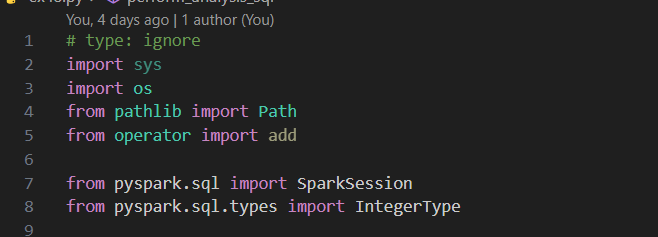
**Input**: Một tập các tệp csv văn bản (có cấu trúc) chứa các thông tin profile. Mỗi dòng có dạng: name, age, gender



**Output**: File CSV chứa tên người dùng (name) và tuổi trung bình đã được làm tròn xuống kiểu Integer

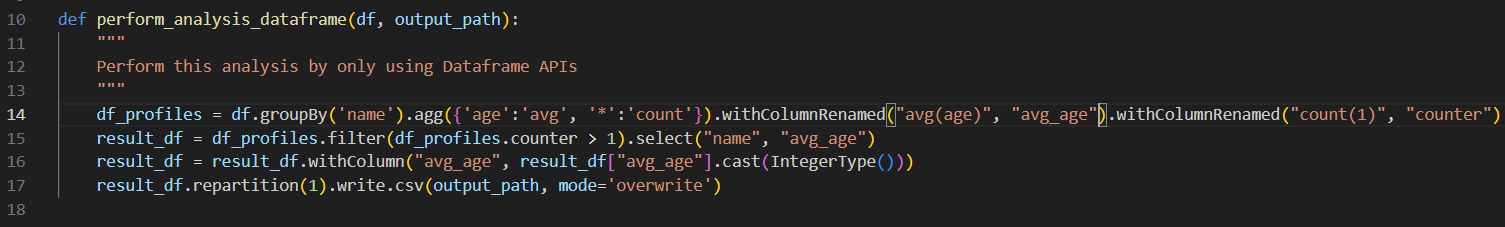
**Code thực hiện:**

*Import thư viện cần thiết*



* os, sys và Path: dùng để quản lý đường dẫn tệp
* SparkSession: tạo phiên làm việc cho PySpark
* add từ operator: để thực hiện phép cộng khi giảm dữ liệu (reduceByKey)
* IntergerType: Xác định kiểu dữ liệu trong Spark DataFrame

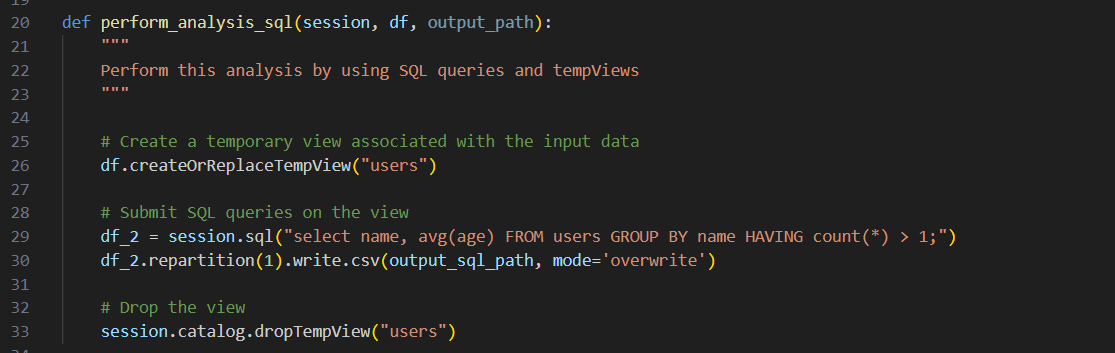
*Hàm perform\_analysis\_dataframe*



Thực hiện phân tích bằng cách chỉ sử dụng các API của DataFrame

* .groupBy(‘name’) : gom nhóm dữ liệu theo cột name
* .agg({‘age’: ‘avg’, ‘\*’: ‘count’}): tính tuổi trung bình (avg(age)) và số lần xuất hiện của mỗi tên (count(\*))
* .withColumnRename : Đổi avg(age) thành avg\_age và đổi count(1) thành counter.
* .filter : Chỉ giữ các tên xuất hiện nhiều hơn một lần (counter > 1)
* .select : Giữ lại hai cột: name và avg\_age.
* .cast(IntergerType()): Ép avg\_age thành kiểu số nguyên (IntegerType).
* repartition(1): đảm bảo rằng kết quả lưu vào 1 tệp duy nhất

*Hàm perform\_analysis\_sql*

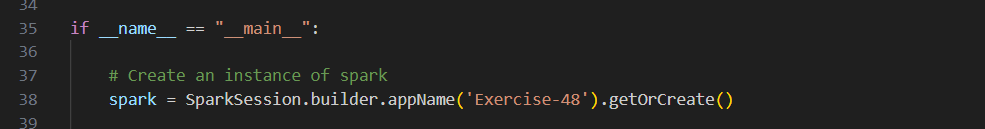


Thực hiện phân tích bằng cách sử dụng các truy vấn SQL

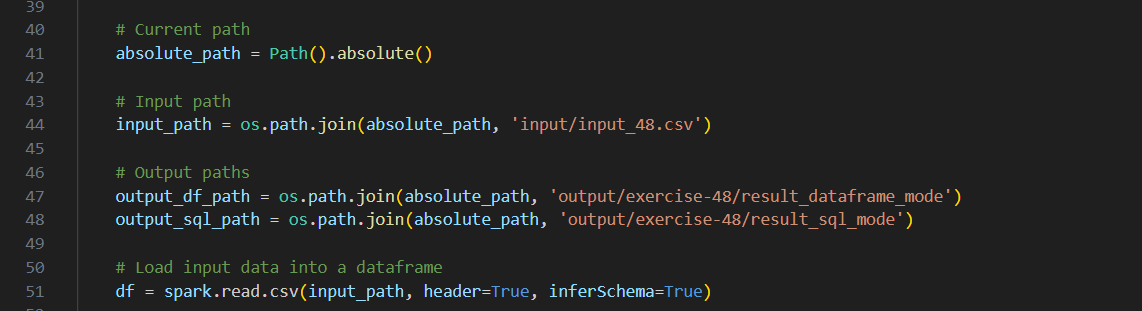
* .createOrReplaceTempView: Tạo View tạm thời. Gắn DataFrame với view tạm thời tên users
* session.sql : truy vấn sql
  + Tính trung bình tuổi (avg(age)) và nhóm theo name
  + Lọc các tên có số lần xuất hiện lớn hơn một (HAVING count(\*) > 1).
* repartition(1).write.csv: Ghi kết quả ra tệp CSV, tương tự như trong hàm DataFrame
* dropTempView : Xoá View để giải phóng tài nguyên

*Chương trình chạy chính*

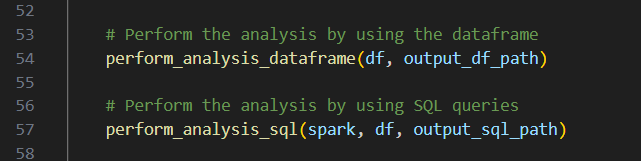
* Thiết lập Spark



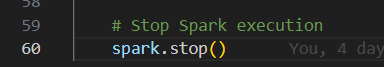
* Đường dẫn và đọc dữ lệu



* Gọi hàm *perform\_analysis\_dataframe* và *perform\_analysis\_sql*

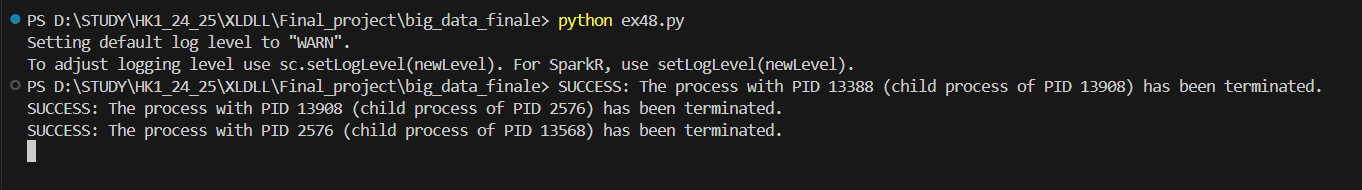


* Dừng Spark

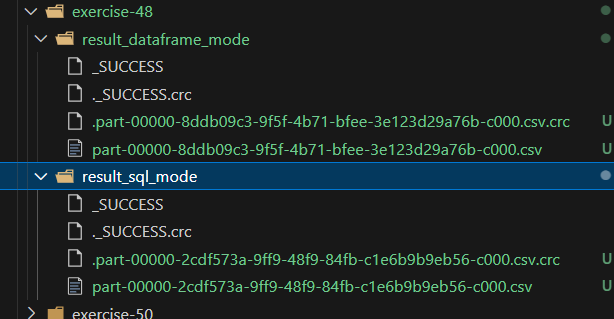


**Kết quả:**

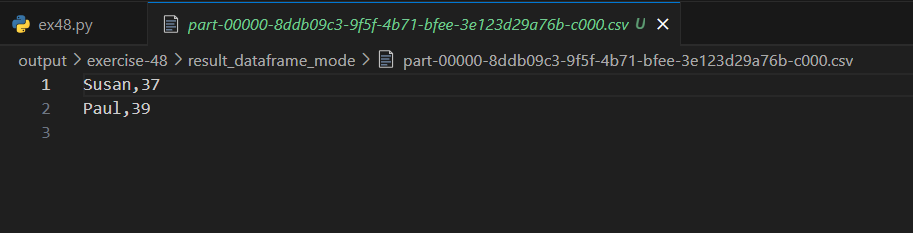
*Chạy code*

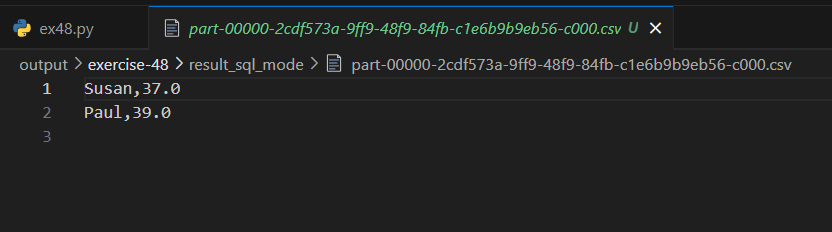


*Output được lưu vào thư mục exercise-48*



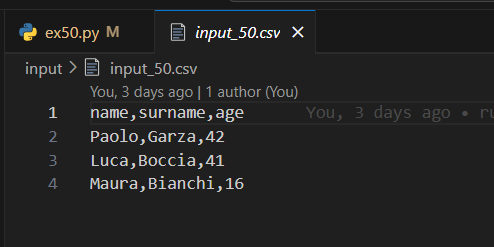
*File output*





## **Exercise #50: Merge Columns**

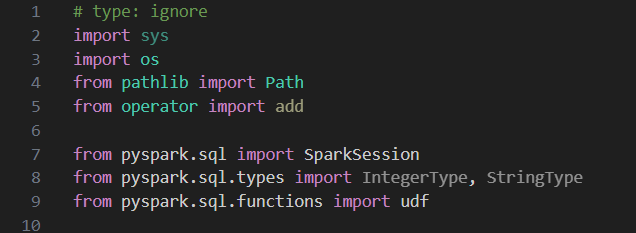
**Input**: Một file CSV văn bản chứa các thông tin profile. Mỗi dòng có định dạng: name, surname, age



**Output**: File CSV chứa một cột name\_surname kiểu dữ liệu String

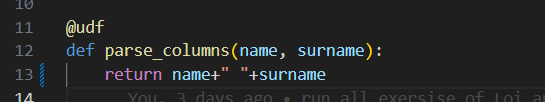
**Code thực hiện:**

*Import thư viện cần thiết*



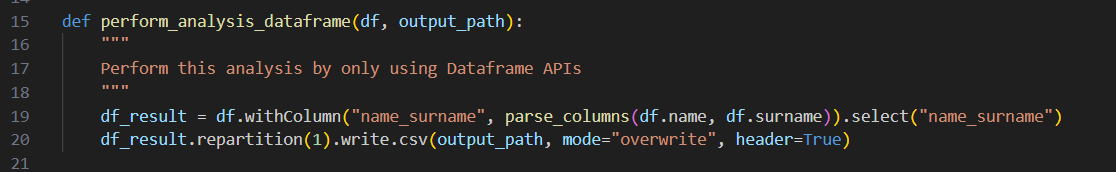
* os, sys và Path: dùng để quản lý đường dẫn tệp
* SparkSession: tạo phiên làm việc cho PySpark
* add từ operator: để thực hiện phép cộng khi giảm dữ liệu (reduceByKey)
* IntergerType, StringType: Xác định kiểu dữ liệu trong Spark DataFrame
* udf: Tạo User Defined Function trong Spark

*Định nghĩa udf*



* Hàm này là 1 udf trong Spark
* Nối cột name và surname thành một chuỗi mới cách nhau bởi “ ”
* Hàm này sử dụng được cả trong DataFrame API và SQL API

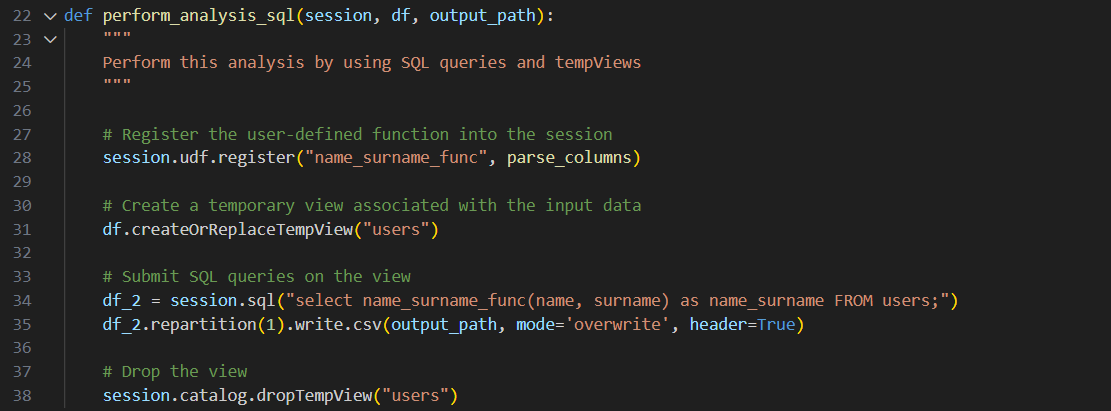
*Hàm perform\_analysis\_dataframe*



Phân tích dữ liệu và lưu kết quả đầu ra bằng cách sử dụng DataFrame API

* df.withColumn: Thêm một cột mới tên là name\_surname bằng cách áp dụng udf *parse\_columns* lên 2 cột *name* và *surname*
* .select(“name\_surname”): Truy vấn cột name\_surname
* .repartition(1): đảm bảo rằng kết quả lưu vào 1 tệp duy nhất
* mode= ‘overwrite’ : ghi đè nếu tệp có tồn tại

*Hàm perform\_analysis\_sql*

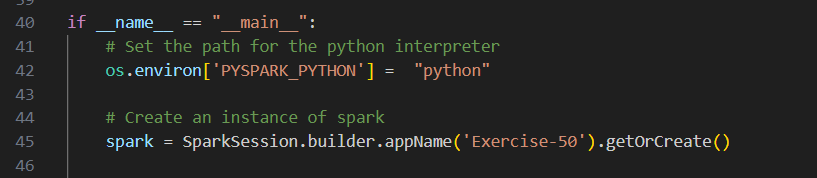


Phân tích dữ liệu và lưu kết quả đầu ra bằng cách sử dụng SQL API trong Spark

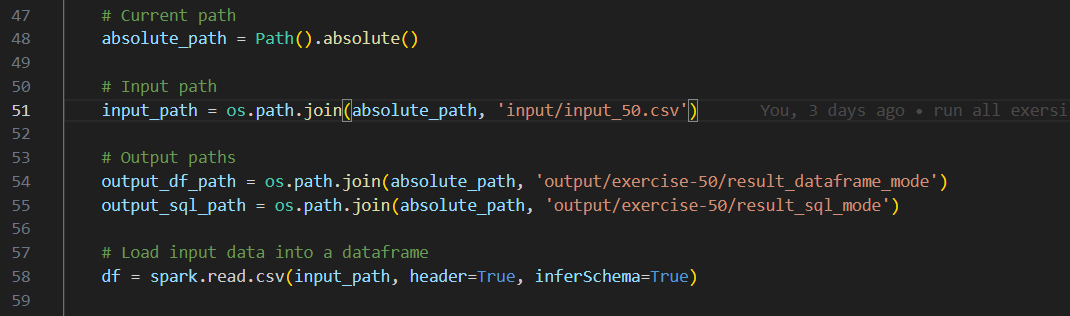
* session.udf.register : Đăng ký udf *parse\_columns* trong Spark SQL với tên là *name\_surname\_func*
* .createOrReplaceTempView(“users”): tạo một tempView từ DataFrame, cho phép truy vần SQL
* session.sql : Chạy truy vấn SQL áp dụng lên udf *name\_surname\_func* lên cột name và surname
* repartition(1).write.csv: Ghi kết quả ra tệp CSV, tương tự như trong hàm DataFrame.
* .dropTempView : Xoá View để giải phóng tài nguyên

*Chương trình chạy chính*

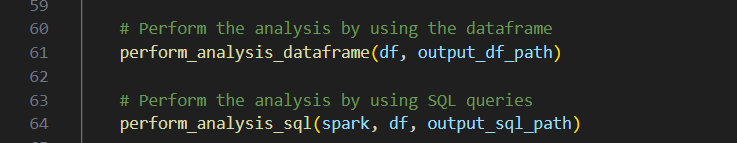
* Thiết lập Spark



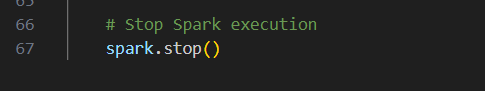
* Đường dẫn và đọc dữ lệu



* Gọi hàm *perform\_analysis\_dataframe* và *perform\_analysis\_sql*

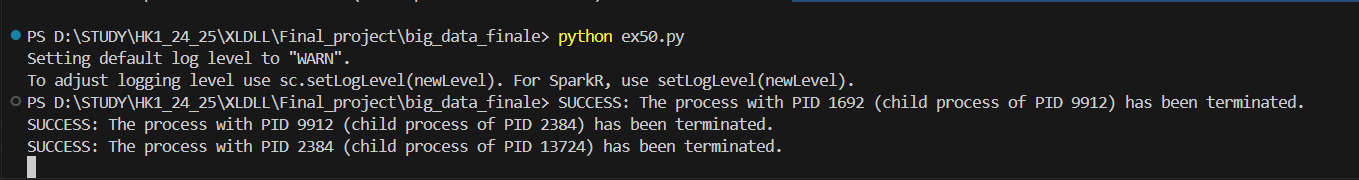


* Dừng Spark

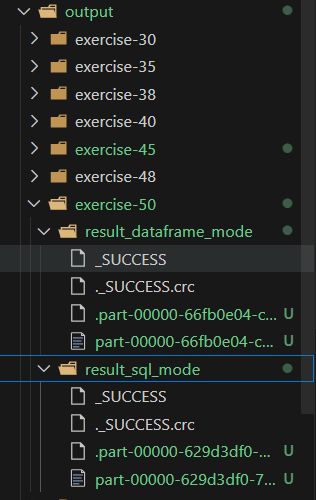


**Kết quả:**

*Chạy code*



*Output được lưu vào thư mục exercise-50*



*File output*

