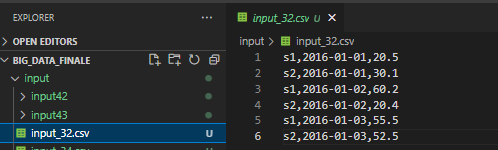
# **Exercise 32: Find Max Temperature**

**Input**: một tập hợp các tệp csv văn bản (có cấu trúc mỗi dòng là sensorId,date,PM10 value (µg/m3)) chứa giá trị hàng ngày của PM10 cho một bộ cảm biến

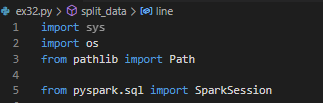


**Output**: Kết quả là giá trị nhiệt độ cao nhất từ dữ liệu, được lưu vào thư mục output/ex-32 dưới dạng tệp văn bản.

**Cần xử lý**: đọc dữ liệu, trích xuất nhiệt độ từ cảm biến, tìm giá trị nhiệt độ cao nhất, và lưu kết quả ra tệp.

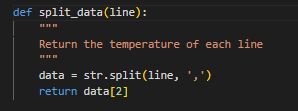
**Code thực hiện:**

*Import thư viện cần thiết*



* sys, os: Dùng để xử lý các đường dẫn tệp và các thao tác liên quan đến hệ thống.
* Path từ pathlib: Dùng để lấy đường dẫn tuyệt đối của thư mục hiện tại.
* pyspark.sql.SparkSession: Là một lớp trong PySpark giúp tạo một phiên làm việc với Spark, cho phép bạn tương tác với Spark SQL và thực hiện các tác vụ xử lý dữ liệu.

*Hàm split\_data(line)*



* data = str.split(line, ','): Tách dòng line thành một danh sách các giá trị (mỗi giá trị được phân tách bởi dấu phẩy).
* return data[2]: Trả về phần tử thứ 3 trong danh sách data (giá trị là PM10 value)

*Tạo SparkSession*



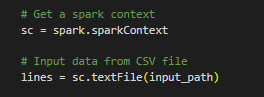
Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-32.

*Xác định đường dẫn tệp*

**

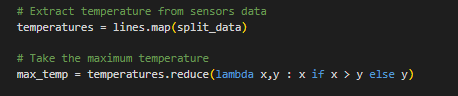
* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* input\_path: xác định đường dẫn đến tệp CSV đầu vào

*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 nhiệt độ lớn nhất*



Áp dụng hàm split\_data cho mỗi dòng trong RDD lines để tạo ra một RDD mới chứa các giá trị nhiệt độ.

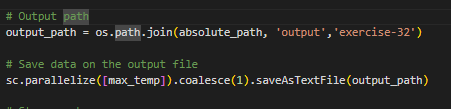
Sử dụng phương thức reduce để tính toán giá trị lớn nhất từ các phần tử trong RDD temperatures. Hàm lambda so sánh hai giá trị và giữ lại giá trị lớn hơn. Kết quả sẽ được lưu vào biến max\_temp.

*In kết quả ra màn hình*

**

In giá trị nhiệt độ lớn nhất từ biến max\_temp ra màn hình.

*Lưu kết quả vào tệp*



* Đầu tiên, một RDD mới được tạo từ giá trị max\_temp (nhiệt độ lớn nhất).
* coalesce(1): Giảm số lượng phân vùng của RDD về 1, giúp giảm thiểu số lượng tệp đầu ra.
* saveAsTextFile(output\_path): Lưu giá trị nhiệt độ lớn nhất vào tệp văn bản trong thư mục output/exercise-32.

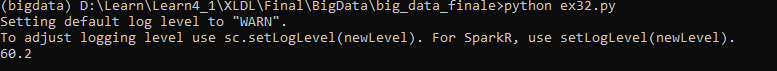
*Dừng phiên làm việc Spark*

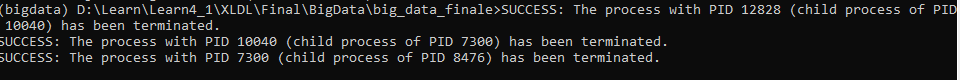


Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

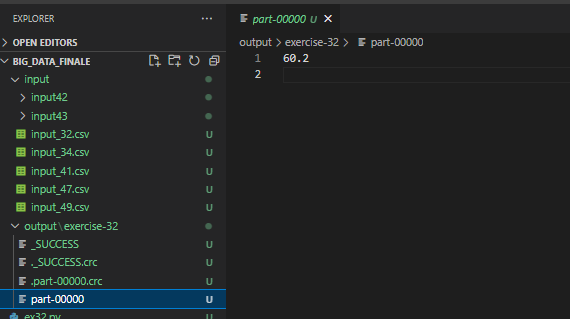
**Kết quả:**

*Chạy code*



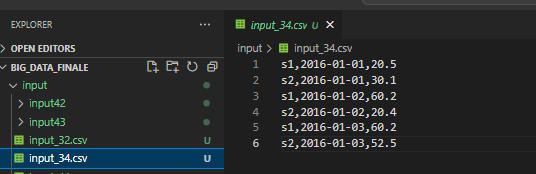


*Output được lưu vào thư mục exercise-32/part-00000*



# **Exercise 34: Filter Max Temperature**

**Input**: một tập hợp các tệp csv văn bản (có cấu trúc mỗi dòng là sensorId,date,PM10 value (µg/m3)) chứa giá trị hàng ngày của PM10 cho một bộ cảm biến

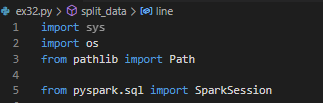


**Output**: Kết quả là (các) dòng được liên kết với mức tối đa giá trị của PM10.

**Cần xử lý**: đọc dữ liệu, tìm nhiệt độ cao nhất từ dữ liệu cảm biến và trích xuất tất cả các dòng liên quan đến nhiệt độ đó. Sau đó, lưu kết quả vào tệp đầu ra.

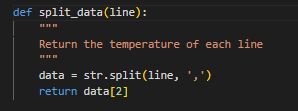
**Code thực hiện:**

*Import thư viện cần thiết*



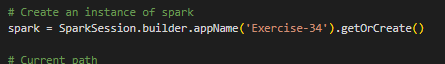
* sys, os: Dùng để xử lý các đường dẫn tệp và các thao tác liên quan đến hệ thống.
* Path từ pathlib: Dùng để lấy đường dẫn tuyệt đối của thư mục hiện tại.
* pyspark.sql.SparkSession: Là một lớp trong PySpark giúp tạo một phiên làm việc với Spark, cho phép bạn tương tác với Spark SQL và thực hiện các tác vụ xử lý dữ liệu.

*Hàm split\_data(line)*



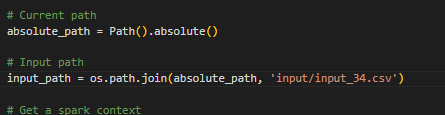
* data = str.split(line, ','): Tách dòng line thành một danh sách các giá trị (mỗi giá trị được phân tách bởi dấu phẩy).
* return data[2]: Trả về phần tử thứ 3 trong danh sách data (giá trị là PM10 value)

*Tạo SparkSession*



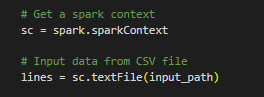
Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-34.

*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

*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

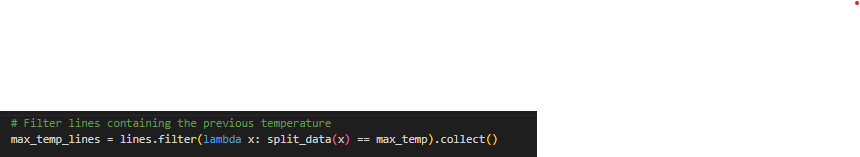
*Tìm giá trị nhiệt độ lớn nhất*



Áp dụng hàm split\_data để trích xuất nhiệt độ từ mỗi dòng.

Sử dụng phương thức top(1) để tìm nhiệt độ cao nhất. Kết quả trả về là một danh sách, lấy giá trị đầu tiên bằng [0].

*Lọc các dòng chứa nhiệt độ trước đó*



Sử dụng filter để giữ lại các dòng có nhiệt độ bằng với max\_temp.

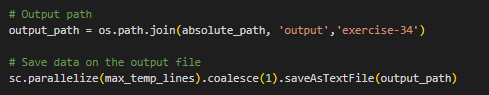
Chuyển kết quả về dạng danh sách Python bằng phương thức collect().

*In kết quả*



In các dòng dữ liệu chứa nhiệt độ cao nhất.

*Lưu kết quả vào tệp*



* Đầu tiên, một RDD mới được tạo từ giá trị max\_temp (nhiệt độ lớn nhất).
* coalesce(1): Giảm số lượng phân vùng của RDD về 1, giúp giảm thiểu số lượng tệp đầu ra.
* saveAsTextFile(output\_path): Lưu giá trị nhiệt độ lớn nhất vào tệp văn bản trong thư mục output/exercise-34.

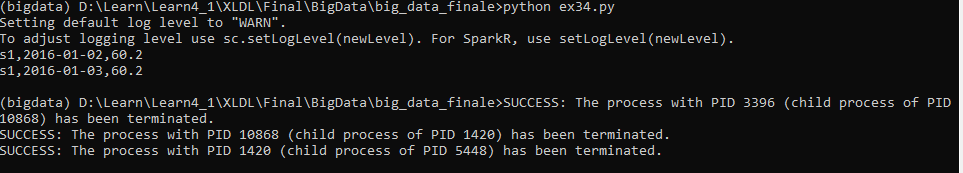
*Dừng phiên làm việc Spark*



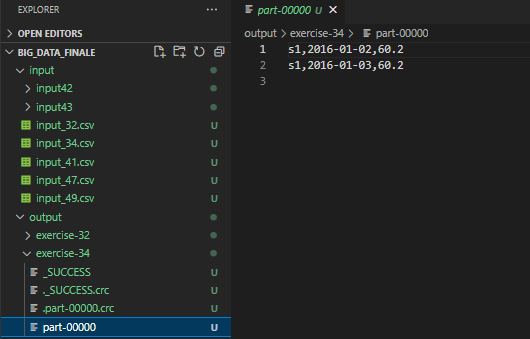
Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

**Kết quả:**

*Chạy code*

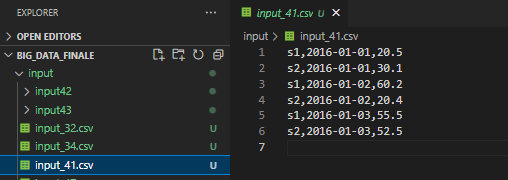


*Output được lưu vào thư mục exercise-34/part-00000*



# **Exercise 41: Filter and Count Sensors Exceeding Temperature Threshold**

**Input**: một tập hợp các tệp csv văn bản csv (có cấu trúc mỗi dòng là sensorId,date,PM10 value (µg/m3)) chứa giá trị hàng ngày của PM10 cho một bộ cảm biến, giá trị K là số cảm biến

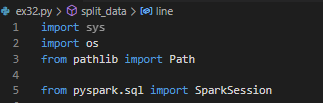


**Output**: Kết quả là in ra màn hình số lần mỗi cảm biến ghi nhận nhiệt độ trên 50°C và lấy K cảm biến có số lần ghi nhận lớn nhất. Lưu kết quả ra tệp.

**Cần xử lý**: đọc dữ liệu, đếm số lần mỗi cảm biến ghi nhận nhiệt độ trên 50°C,lấy K cảmbiến có số lần ghi nhận lớn nhất..

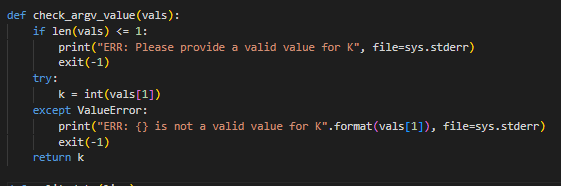
**Code thực hiện:**

*Import thư viện cần thiết*



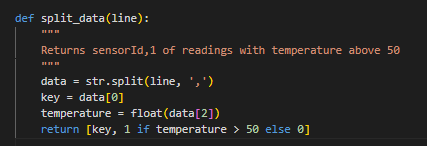
* sys, os: Dùng để xử lý các đường dẫn tệp và các thao tác liên quan đến hệ thống.
* Path từ pathlib: Dùng để lấy đường dẫn tuyệt đối của thư mục hiện tại.
* pyspark.sql.SparkSession: Là một lớp trong PySpark giúp tạo một phiên làm việc với Spark, cho phép bạn tương tác với Spark SQL và thực hiện các tác vụ xử lý dữ liệu.

*Hàm check\_argv\_value*



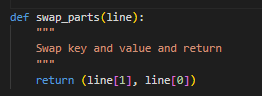
* **Mục đích:**
  + Kiểm tra giá trị K truyền vào từ dòng lệnh.
  + Đảm bảo K là một số nguyên hợp lệ.
* **Cách hoạt động:**
* Nếu không có tham số K, hoặc K không phải số nguyên, chương trình dừng và in lỗi.
* Trả về giá trị K sau khi kiểm tra hợp lệ.

*Hàm split\_data*



* data = str.split(line, ','): Tách dòng line thành một danh sách các giá trị (mỗi giá trị được phân tách bởi dấu phẩy).
* data[0]: Trả về phần tử thứ 1 trong danh sách data (giá trị là sensorId)
* data[2]: Trả về phần tử thứ 3 trong danh sách data (giá trị là PM10 value)
* Hàm trả về cặp (sensorId, 1 hoặc 0)

*Hàm swap\_parts*



**Mục đích:** Đảo ngược thứ tự cặp (sensorId, count) thành (count, sensorId) để dễ dàng sắp xếp theo số lần đếm.

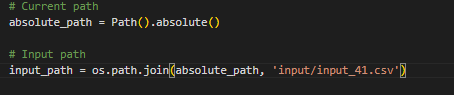
*Tạo SparkSession*



Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-41.

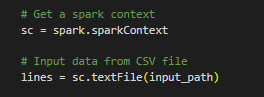
*Xác định đường dẫn tệp*

**

**

* Giá trị K là số cảm biến có số lần ghi nhận cao nhất cần lấy.
* 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

*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

*Lọc tất cả các giá trị đọc dưới ngưỡng 50*



Áp dụng split\_data, trích xuất cặp (sensorId, 1 hoặc 0) từ mỗi dòng và lọc các dòng ghi nhận nhiệt độ lớn hơn 50°C.

*Đếm số lần ghi nhận cho mỗi cảm biến  
*

* reduceByKey(add): Tính tổng giá trị 1 cho từng sensorId (số lần ghi nhận nhiệt độ trên 50°C).
* map(swap\_parts): Đảo ngược thành (count, sensorId) để sắp xếp dễ dàng.
* sortByKey(ascending=False): Sắp xếp theo số lần ghi nhận giảm dần.

*Lấy top K cảm biến*



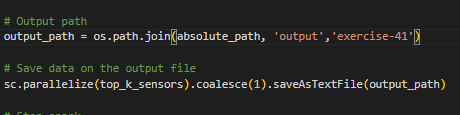
Lấy K cảm biến có số lần ghi nhận nhiệt độ trên 50°C cao nhất.

*In kết quả*



In danh sách top K cảm biến, mỗi phần tử là (count, sensorId).

*Lưu kết quả*



* Đầu tiên, một RDD mới được tạo từ giá trị max\_temp (nhiệt độ lớn nhất).
* coalesce(1): Giảm số lượng phân vùng của RDD về 1, giúp giảm thiểu số lượng tệp đầu ra.
* saveAsTextFile(output\_path): Lưu giá trị nhiệt độ lớn nhất vào tệp văn bản trong thư mục output/exercise-41.

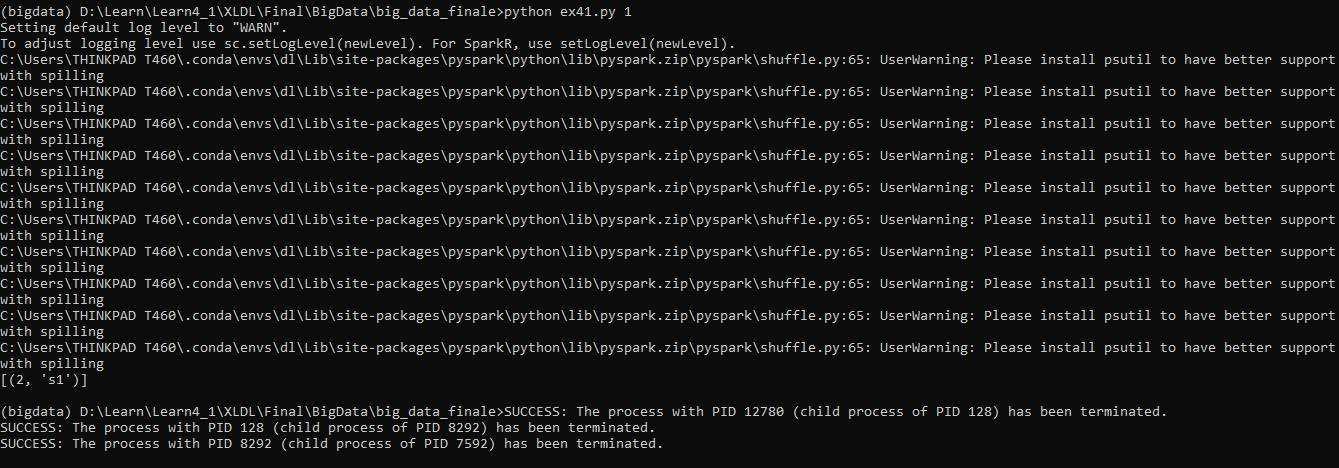
*Dừng phiên làm việc Spark*



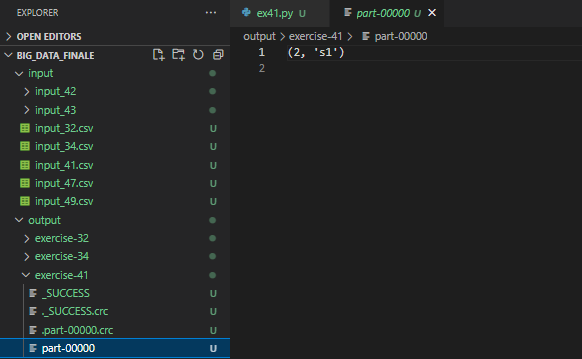
Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

**Kết quả:**

*Chạy code*



*Output được lưu vào thư mục exercise-41/part-00000*



# **Exercise 42: Mapping Question-Answer(s)**

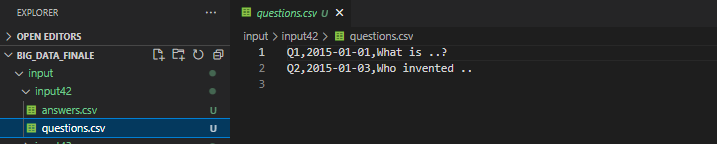
**Input:**

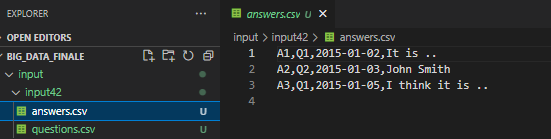
Một tệp văn bản lớn chứa một bộ câu hỏi

* Mỗi dòng có một câu hỏi
* Mỗi dòng có định dạng:QuestionId,Timestamp,TextOfTheQuestion

Một tệp văn bản lớn chứa tập hợp các câu trả lời

* Mỗi dòng có một câu trả lời
* Mỗi dòng có định dạng:AnswerId,QuestionId,Timestamp,TextOfTheAnswer



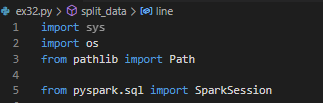


**Output**: Kết quả là một file chứa một dòng cho mỗi câu hỏi, mỗi dòng có một câu hỏi và danh sách các câu trả lời cho câu hỏi đó. Mỗi dòng có định dạng:QuestionId, TextOfTheQuestion, list of Answers

**Cần xử lý**: đọc dữ liệu, thực hiện ghép nối dựa trên QuestionId.

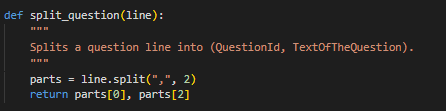
**Code thực hiện:**

*Import thư viện cần thiết*



* sys, os: Dùng để xử lý các đường dẫn tệp và các thao tác liên quan đến hệ thống.
* Path từ pathlib: Dùng để lấy đường dẫn tuyệt đối của thư mục hiện tại.
* pyspark.sql.SparkSession: Là một lớp trong PySpark giúp tạo một phiên làm việc với Spark, cho phép bạn tương tác với Spark SQL và thực hiện các tác vụ xử lý dữ liệu.

*Hàm split\_question*

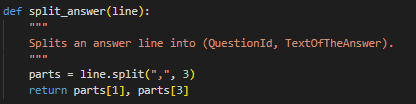


Dùng split với dấu phẩy (",") để chia dòng dữ liệu thành 3 phần

Hàm trả về:

* parts[0]: **QuestionId**.
* parts[2]: **TextOfTheQuestion**.

*Hàm split\_answer*

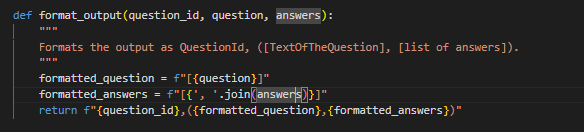


Dùng split với dấu phẩy (",") để chia dòng dữ liệu thành 4 phần

Hàm trả về:

* parts[1]: **QuestionId**.
* parts[3]: **TextOfTheAnswer**.

*Hàm format\_output*



Định dạng kết quả đầu ra thành chuỗi:

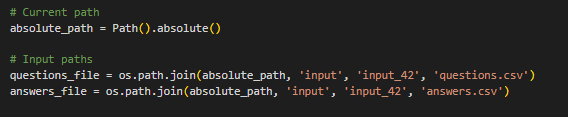
QuestionId, ([TextOfTheQuestion], [list of answers]).

*Tạo SparkSession*



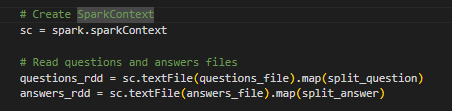
Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-42.

*Xác định đường dẫn tệp*

**

* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* questions\_file: xác định đường dẫn đến tệp questions.csv đầu vào
* answers\_file: xác định đường dẫn đến tệp answers.csv đầu vào

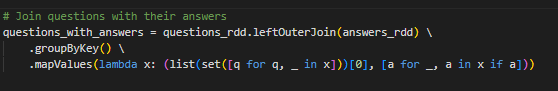
*Đọc dữ liệu đầu vào*



Đọc tệp questions.csv và answers.csv và trả về RDD.

Áp dụng split\_question và split\_answer, chuyển đổi mỗi dòng dữ liệu thành cặp (QuestionId, TextOfTheQuestion) hoặc (QuestionId, TextOfTheAnswer).

*Ghép câu hỏi với câu trả lời*



* **leftOuterJoin**: Kết hợp dữ liệu từ questions\_rdd với answers\_rdd dựa trên QuestionId.
* Nếu một câu hỏi không có câu trả lời, câu trả lời tương ứng là None.
* **groupByKey**: Nhóm các cặp (QuestionId, (TextOfTheQuestion, TextOfTheAnswer)) theo QuestionId.
* **mapValues**: Tách nội dung câu hỏi (TextOfTheQuestion) và danh sách câu trả lời (TextOfTheAnswer) tương ứng.

*Định dạng kết quả*

**

Áp dụng hàm format\_output để định dạng kết quả theo yêu cầu.

*Lưu kết quả*





* **coalesce(1):** Hợp nhất dữ liệu vào một tệp duy nhất.
* **saveAsTextFile:** Lưu kết quả dưới dạng tệp văn bản trong thư mục output/exercise-42.

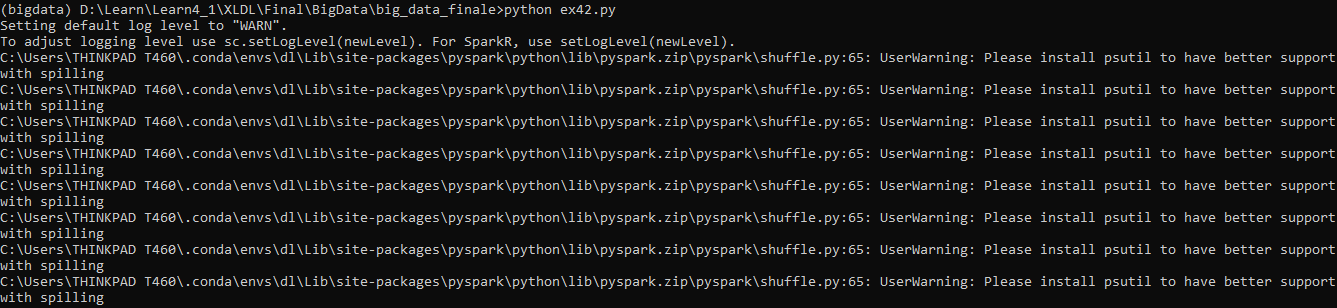
*Dừng phiên làm việc Spark*

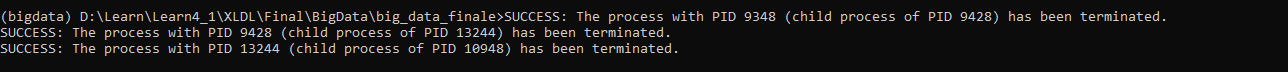


Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

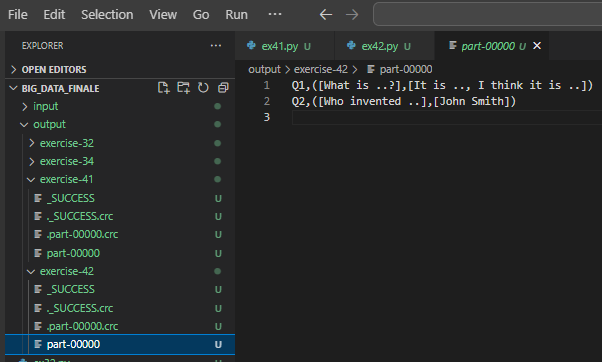
**Kết quả:**

*Chạy code*





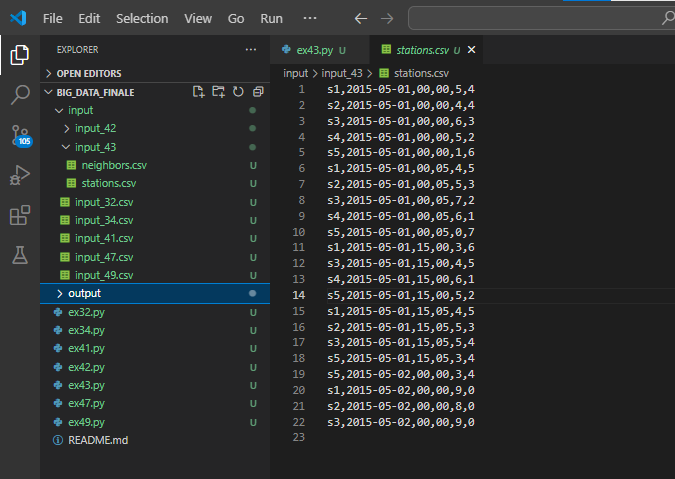
*Output được lưu vào thư mục exercise-42/part-00000*



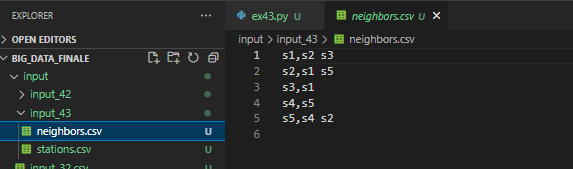
# **Exercise 43: Critical bike sharing station analysis**

**Input:**

Một tệp văn bản lớn chứa chứa thông tin về các trạm xe đạp (bikes station) tại các thời điểm khác nhau. Mỗi dòng có định dạng: stationId , date, hour, minute, num\_of\_bikes, num\_of\_free\_slots



Một tệp văn bản lớn hứa danh sách các trạm lân cận của mỗi trạm xe. Mỗi dòng có định dạng: stationIdx , list of neighbors of stationId



**Output**:

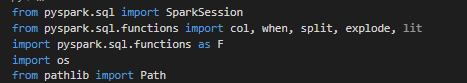
* File lưu trữ kết quả các trạm có tỷ lệ tình huống nguy cấp cao hơn 80%
* File lưu trữ các cặp (timeslot, station) với tỷ lệ phần trăm các tình huống nguy cấp cao hơn 80%
* File lưu trữ các dòng đã select và in ra tiêu chuẩn tổng số của những dòng như vậy

**Cần xử lý**: đọc dữ liệu, thực hiện các yêu cầu sau

* Tính phần trăm tình huống nguy cấp cho mỗi trạm, lưu trữ kết quả các trạm có tỷ lệ tình huống nguy cấp cao hơn 80%
* Tính tỷ lệ phần trăm các tình huống nguy kịch đối với mỗi cặp (timeslot, station), lưu trữ các cặp (timeslot, station) với tỷ lệ phần trăm các tình huống nguy cấp cao hơn 80%
* Select các tình huống trạm đầy trong đó tất cả các neighbors cũng đầy, lưu trữ các dòng đã select và in trên đầu ra tiêu chuẩn tổng số của những dòng như vậy

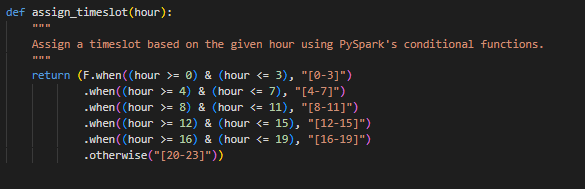
**Code thực hiện:**

*Import thư viện cần thiết*



* sys, os: Dùng để xử lý các đường dẫn tệp và các thao tác liên quan đến hệ thống.
* Path từ pathlib: Dùng để lấy đường dẫn tuyệt đối của thư mục hiện tại.
* pyspark.sql.SparkSession: Là một lớp trong PySpark giúp tạo một phiên làm việc với Spark, cho phép bạn tương tác với Spark SQL và thực hiện các tác vụ xử lý dữ liệu.
* col: Để tham chiếu tới cột trong DataFrame.
* when: Hàm điều kiện tương tự if-else.
* split: Tách chuỗi thành mảng.
* explode: Mở rộng mảng thành các dòng riêng lẻ.
* lit: Chèn hằng số vào cột.

*Hàm gán timeslot*



Nhóm giờ (hour) vào các khoảng thời gian (timeslot).

**Hàm F.when**: Gán nhãn dựa trên điều kiện:

* hour từ 0-3 → [0-3].
* hour từ 4-7 → [4-7].
* Tương tự cho các khoảng khác.

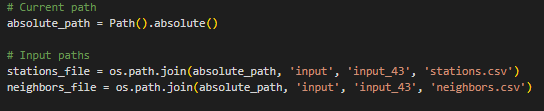
**otherwise**: Giá trị mặc định nếu không thoả điều kiện trên.

*Tạo SparkSession*



Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-43.

*Xác định đường dẫn tệp*

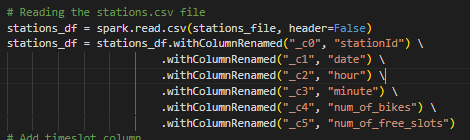
**

* absolute\_path: lấy đường dẫn tuyệt đối của thư mục đang làm việc hiện tại
* stations: xác định đường dẫn đến tệp stations.csv đầu vào
* neighbors: xác định đường dẫn đến tệp neighbors.csv đầu vào



* **output\_path**: Đường dẫn lưu kết quả đầu ra.

Đọc stations.csv



**spark.read.csv**: Đọc tệp CSV thành DataFrame.

**header=False**: Tệp không có dòng tiêu đề.

**withColumnRenamed**: Đổi tên các cột từ \_c0, \_c1... thành các tên dễ hiểu:

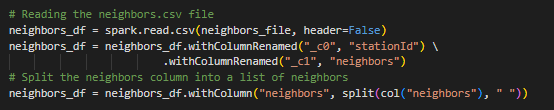
* stationId
* date, hour, minute.
* num\_of\_bikes
* num\_of\_free\_slots

*Thêm cột timeslot*



* **withColumn**: Thêm cột mới timeslot vào DataFrame.
* **col("hour").cast("int")**: Ép kiểu cột hour thành int.
* **assign\_timeslot**: Gọi hàm để gán timeslot dựa trên giờ.

*Đọc neighbors.csv*

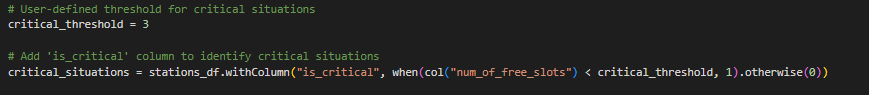


**neighbors.csv** chứa danh sách các neighbors:

* \_c0: stationId.
* \_c1: neighbors.

**split**: Chuyển chuỗi neighbors thành mảng.

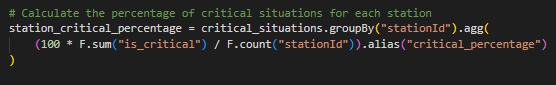
**Tính phần trăm tình huống nguy cấp cho mỗi trạm**



**Mục đích:** Xác định tình trạng nguy cấp (is\_critical), thêm cột is\_critical vào stations\_df.

**Ngưỡng critical\_threshold**:

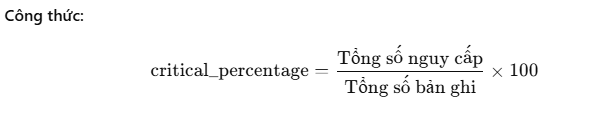
* Nếu num\_of\_free\_slots < 3 → is\_critical = 1.
* Ngược lại → is\_critical = 0.

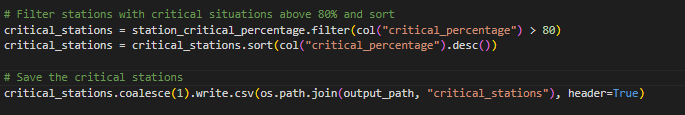


**groupBy("stationId")**: Nhóm theo stationsId.

**F.sum("is\_critical")**: Tổng số lần xảy ra nguy cấp.

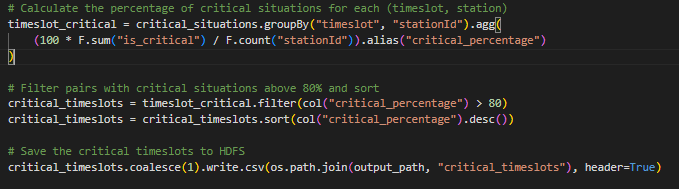
**F.count("stationId")**: Tổng số bản ghi.





* **critical\_stations**: DataFrame chứa danh sách các trạm có tỷ lệ tình trạng nguy cấp.
* **filter**: Giữ các trạm có critical\_percentage > 80.
* **sort(col("critical\_percentage").desc()):** Sắp xếp các hàng trong DataFrame dựa trên giá trị của cột critical\_percentage, theo thứ tự từ cao xuống thấp.
* **coalesce(1)**: Kết hợp dữ liệu thành một tệp duy nhất.
* **write.csv**: Lưu kết quả ra thư mục đầu ra.

**Tính tỷ lệ phần trăm các tình huống nguy kịch đối với mỗi cặp (timeslot, station)**

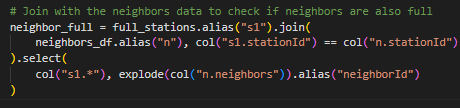


* Tương tự phần trên nhưng nhóm (groupBy) theo timeslot và stationId.

**Select các tình huống trạm đầy trong đó tất cả các neighbors cũng đầy**

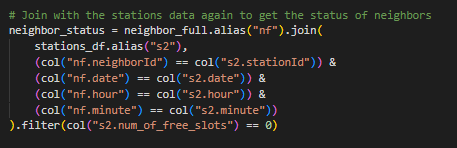


Lọc các trạm có số khe trống (num\_of\_free\_slots) bằng 0 từ DataFrame stations\_df. Đây là những trạm đầy xe.



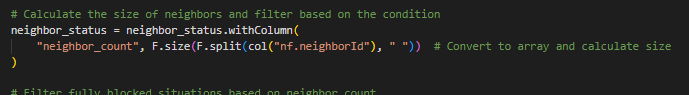
* **join:** Kết hợp DataFrame full\_stations với DataFrame neighbors\_df thông qua cột stationId. Điều này thêm danh sách lân cận của mỗi trạm đầy.
* **explode:** Tách danh sách các lân cận (định dạng chuỗi) thành từng hàng riêng biệt.
* **select:** Giữ lại thông tin của trạm đầy (từ full\_stations) và ID của từng lân cận (được đặt là neighborId).

DataFrame neighbor\_full chứa thông tin về mỗi trạm đầy và từng trạm lân cận của nó.

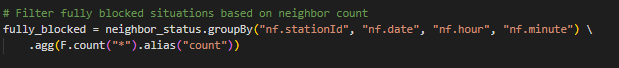


* **join:** Kết hợp neighbor\_full với stations\_df để kiểm tra trạng thái của các trạm lân cận tại cùng thời điểm (dựa trên stationId, date, hour, minute).
* **filter:** Giữ lại chỉ những lân cận có số khe trống bằng 0 (các trạm lân cận đầy).

DataFrame neighbor\_status chứa danh sách các trạm đầy cùng các lân cận đầy của chúng tại các thời điểm tương ứng.

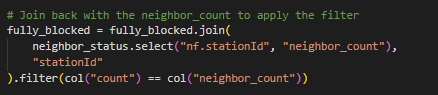


Tính số lượng lân cận của mỗi trạm, thêm neighbor\_count vào neighbor\_status.



Nhóm dữ liệu theo stationId, date, hour, minute.

Đếm số lượng lân cận đầy (số hàng trong nhóm).



Thêm thông tin về tổng số lân cận (neighbor\_count) từ neighbor\_status.

Lọc ra các trạm đầy (count) mà tất cả lân cận cũng đầy (count == neighbor\_count).



**coalesce(1):** Hợp nhất dữ liệu thành một file duy nhất.

**write.csv:** Ghi kết quả ra file CSV với tên fully\_blocked trong thư mục output.

**count:** Đếm tổng số tình huống fully blocked

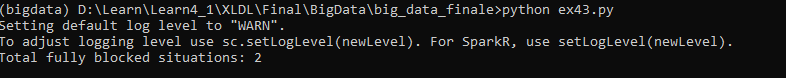
*Dừng phiên làm việc Spark*

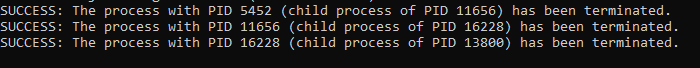


Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

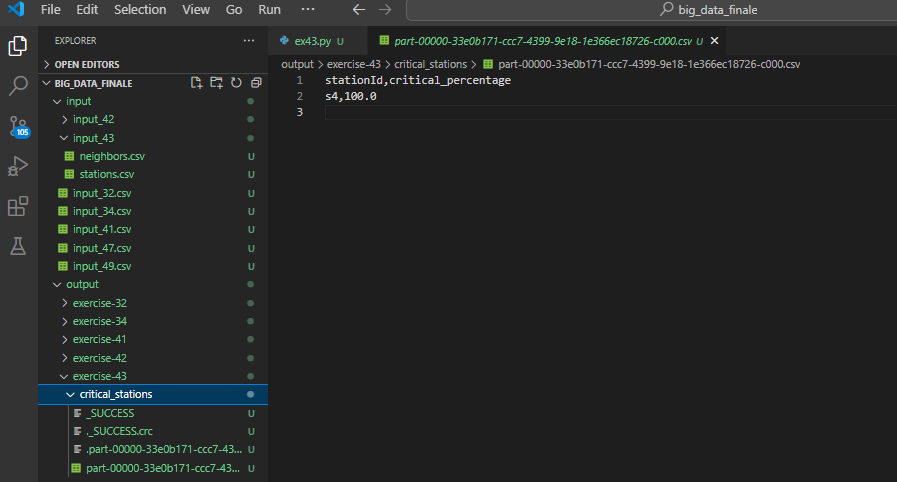
**Kết quả:**

*Chạy code*

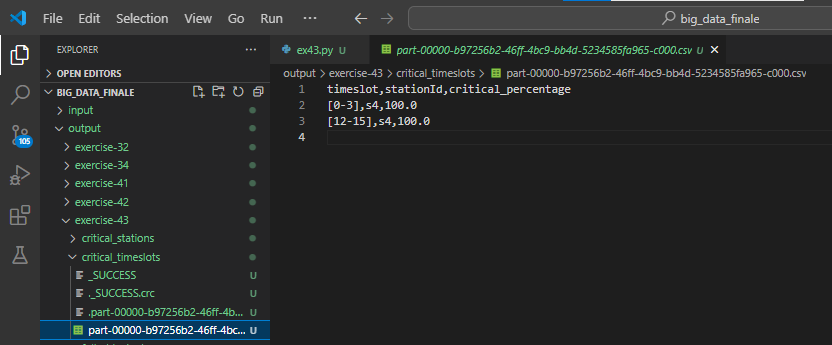




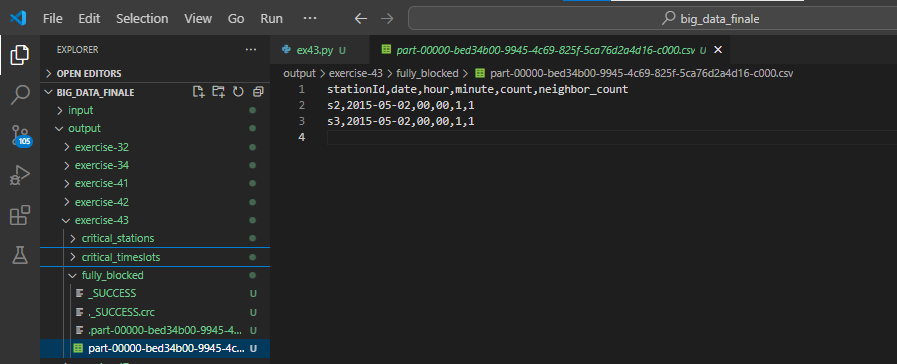
*Output được lưu vào thư mục exercise-43/* *critical\_stations/* *part-00000-33e0b171-ccc7-4399-9e18-1e366ec18726-c000.csv*



*Output được lưu vào thư mục exercise-43/* *part-00000-b97256b2-46ff-4bc9-bb4d-5234585fa965-c000.csv*

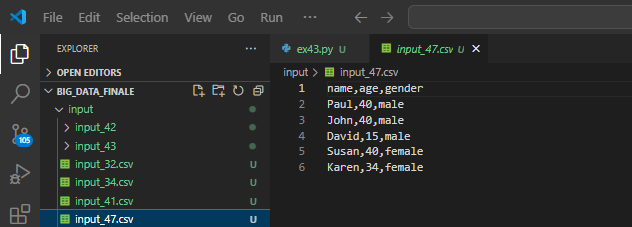


*Output được lưu vào thư mục exercise-43/* *fully\_blocked/* *part-00000-bed34b00-9945-4c69-825f-5ca76d2a4d16-c000.csv*



# **Exercise 47: Select Male User,** **increase by one their age, sorted by decreasing age and ascending name (if the age value is the same).**

**Input:** Tệp CSV chứa danh sách hồ sơ người dùng, Mỗi dòng có định dạng: name, age, gender. Mỗi dòng file chứa thông tin về một người

****

**Output**: file chứa danh sánh người giới tính nam, tăng thêm một tuổi này được sắp xếp theo độ tuổi giảm dần và tên tăng dần (nếu giá trị độ tuổi giống nhau).

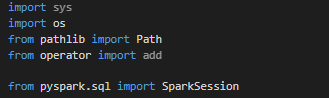
**Cần xử lý**: đọc dữ liệu, chọn người giới nam (giới tính="nam"), tăng thêm một tuổi và lưu vào thư mục đầu ra.

Tên và tuổi của những người này được sắp xếp theo độ tuổi giảm dần và tên tăng dần (nếu giá trị độ tuổi giống nhau).

- Đầu ra không chứa dòng tiêu đề.

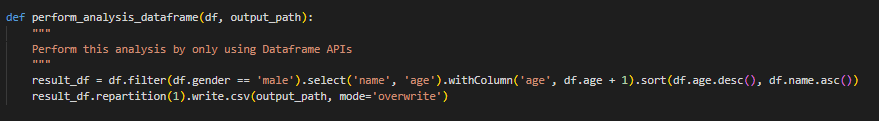
**Code thực hiện:**

*Import thư viện cần thiết*



* **sys**: Hỗ trợ các thao tác liên quan đến hệ thống, như đường dẫn hoặc các tham số dòng lệnh.
* **os**: Cung cấp các hàm tương tác với hệ thống file và môi trường hệ điều hành.
* **pathlib.Path**: Thao tác với đường dẫn file một cách tiện lợi hơn so với sử dụng chuỗi trực tiếp.
* **add** từ **operator**: Hàm cộng hai giá trị (dư thừa, không được sử dụng trong mã này).

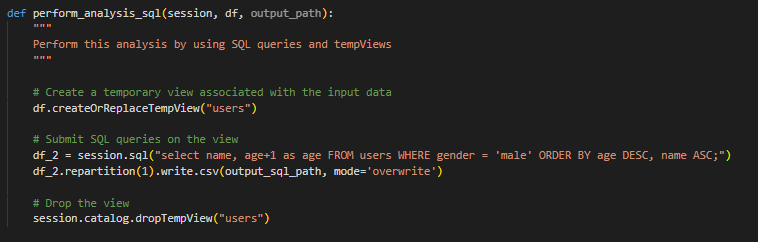
*Hàm perform\_analysis\_dataframe*



Hàm này thực hiện phân tích dữ liệu bằng cách sử dụng API DataFrame của PySpark. Kết quả được lưu vào đường dẫn output\_path.

* **df.filter(df.gender == 'male')**: Lọc các hàng có cột gender bằng 'male'.
* **.select('name', 'age')**: Chỉ giữ lại hai cột name và age.
* **.withColumn('age', df.age + 1)**: Tạo một cột mới age bằng giá trị hiện tại của age cộng thêm 1.
* **.sort(df.age.desc(), df.name.asc())**: Sắp xếp theo cột age giảm dần (desc) và cột name tăng dần (asc).
* **repartition(1)**: Đảm bảo chỉ tạo một file CSV trong thư mục đầu ra.
* **.write.csv(output\_path, mode='overwrite')**: Ghi DataFrame vào thư mục output\_path dưới định dạng CSV. Nếu thư mục đã tồn tại, nó sẽ bị ghi đè (overwrite).

*Hàm perform\_analysis\_sql*



Hàm này thực hiện phân tích dữ liệu bằng cách sử dụng các câu truy vấn SQL trên DataFrame. Kết quả được lưu vào đường dẫn output\_path.

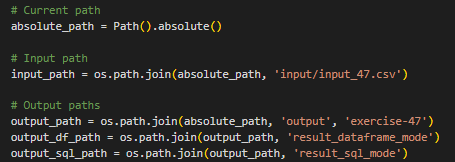
* **createOrReplaceTempView("users")**: Tạo hoặc thay thế một View tạm thời có tên là users, cho phép truy vấn DataFrame qua SQL.
* **session.sql**: Gửi câu lệnh SQL đến Spark để xử lý.
* **SQL**: Chọn các hàng có gender = 'male', cộng 1 vào cột age, sắp xếp theo age giảm dần và name tăng dần.
* Ghi kết quả của truy vấn SQL vào thư mục output\_sql\_path dưới định dạng CSV.
* **dropTempView("users")**: Xóa View tạm thời users sau khi hoàn thành.

*Tạo SparkSession*



Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-47.

*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: Đường dẫn lưu kết quả đầu ra cho 2 chế độ dưới
* output\_df\_path: Đường dẫn lưu kết quả đầu ra chế độ DataFrame
* output\_sql\_path: Đường dẫn lưu kết quả đầu ra chế độ SQL

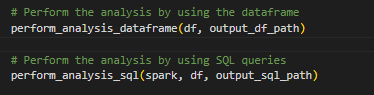
*Đọc file CSV*



**header=True**: Dòng đầu tiên trong file là tên cột.

**inferSchema=True**: Tự động suy luận kiểu dữ liệu của các cột.

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



* Gọi hàm phân tích bằng API DataFrame và lưu kết quả vào output\_df\_path.
* Gọi hàm phân tích bằng SQL và lưu kết quả vào output\_sql\_path.

*Dừng phiên làm việc Spark*

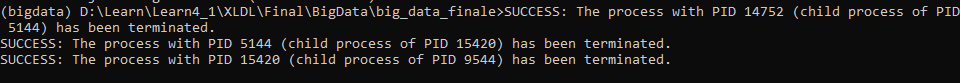


Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

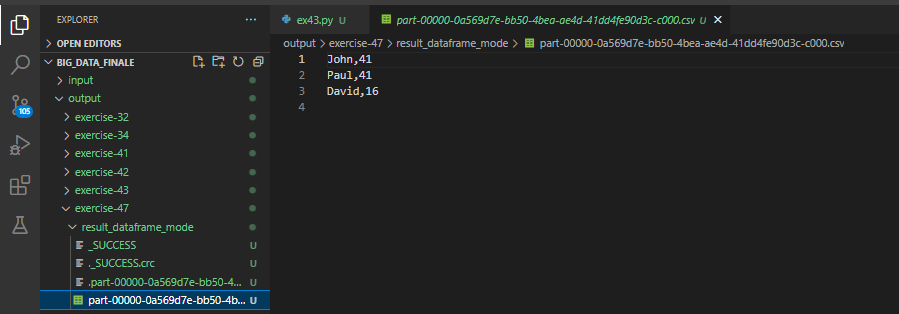
**Kết quả:**

*Chạy code*

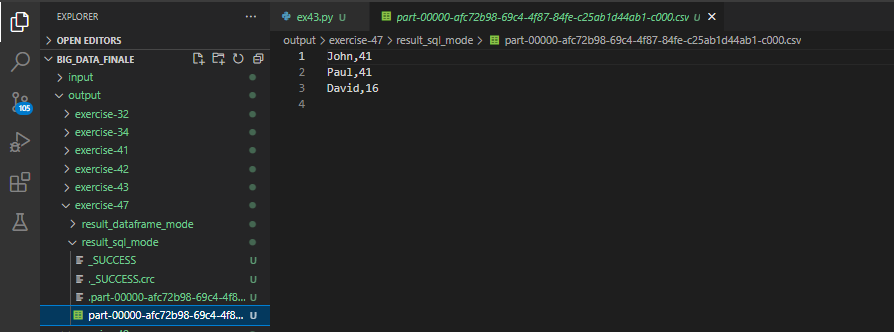




*Output được lưu vào thư mục exercise-47/result\_dataframe\_mode/part-00000-0a569d7e-bb50-4bea-ae4d-41dd4fe90d3c-c000.csv*

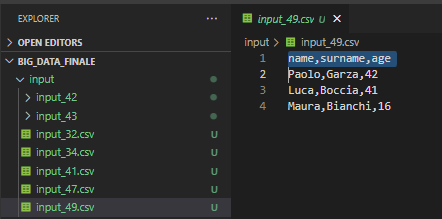
**

*Output được lưu vào thư mục exercise-47/result\_sql\_mode/part-00000-afc72b98-69c4-4f87-84fe-c25ab1d44ab1-c000.csv*



# **Exercise 49: The original age attribute is replaced by a new attribute called rangeage**

**Input:** Tệp CSV chứa danh sách hồ sơ người dùng, Mỗi dòng có định dạng: name, surname, age. Mỗi dòng file chứa thông tin về một người

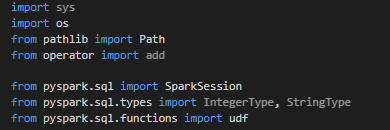
****

**Output**: Tệp CSV chứa một dòng cho mỗi hồ sơ. Thuộc tính tuổi ban đầu được thay thế bằng thuộc tính mới có tên là rangeage.

**Cần xử lý**: đọc dữ liệu, thuộc tính tuổi ban đầu được thay thế bằng thuộc tính mới có tên là rangeage., dữ liệu trong thuộc tính mới được tính:

rangeage = "[" + (age/10)\*10 + "-" + (age/10)\*10 +9"]"

**Code thực hiện:**



* **sys, os, Path**: Quản lý đường dẫn tệp và các thông tin liên quan đến môi trường hệ thống.
* **add** từ **operator**: Hàm cộng hai giá trị (dư thừa, không được sử dụng trong mã này).
* **SparkSession**: Giao diện chính để tương tác với Spark.
* **IntegerType**, **StringType**: Các kiểu dữ liệu Spark.
* **udf**: Dùng để định nghĩa User Defined Functions (UDF), là các hàm tùy chỉnh áp dụng lên các cột trong DataFrame.

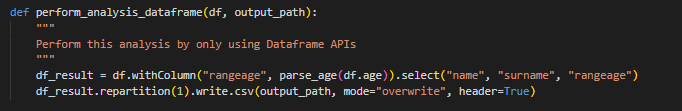
*Định nghĩa UDF parse\_age*



Đây là một UDF để phân loại độ tuổi (age) vào các khoảng như [20-29], [30-39],...

* + **int(age/10)\*10**: Tính giá trị thấp nhất trong khoảng.
  + **int(age/10)\*10+9**: Tính giá trị cao nhất trong khoảng.

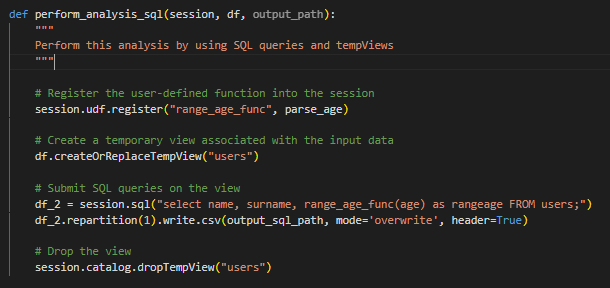
*Hàm perform\_analysis\_dataframe*



Hàm này thực hiện phân tích dữ liệu bằng cách sử dụng API DataFrame của PySpark. Kết quả được lưu vào đường dẫn output\_path.

* **withColumn("rangeage", parse\_age(df.age))**: Tạo thêm cột rangeage chứa khoảng tuổi, sử dụng UDF parse\_age.
* **select("name", "surname", "rangeage")**: Chọn các cột cần thiết là name, surname, và rangeage.
* **repartition(1)**: Đảm bảo dữ liệu được ghi ra chỉ vào một file CSV duy nhất.
* **.write.csv(output\_path, mode="overwrite", header=True)**: Lưu kết quả vào output\_path với cột tiêu đề (header=True).

*Hàm perform\_analysis\_sql*



Hàm này thực hiện phân tích dữ liệu bằng cách sử dụng các câu truy vấn SQL trên DataFrame. Kết quả được lưu vào đường dẫn output\_path.

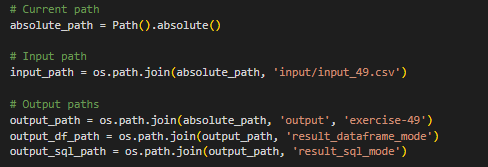
* **session.udf.register("range\_age\_func", parse\_age)**: Đăng ký UDF parse\_age để sử dụng trong câu lệnh SQL dưới tên range\_age\_func.
* **df.createOrReplaceTempView("users")**: Tạo một View tạm thời có tên users từ DataFrame df, cho phép truy vấn bằng SQL.
* **session.sql(...)**: Chạy truy vấn SQL:
* Chọn các cột name, surname và thêm cột rangeage từ hàm UDF.
* **.write.csv(...)**: Lưu kết quả dưới dạng CSV vào output\_path.
* **session.catalog.dropTempView("users")**: Xóa View tạm users để giải phóng tài nguyên.

*Tạo SparkSession*



Dòng này khởi tạo một phiên làm việc Spark với tên ứng dụng là Exercise-49.

*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: Đường dẫn lưu kết quả đầu ra cho 2 chế độ dưới
* output\_df\_path: Đường dẫn lưu kết quả đầu ra chế độ DataFrame
* output\_sql\_path: Đường dẫn lưu kết quả đầu ra chế độ SQL

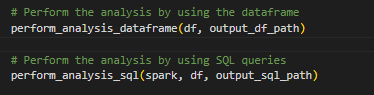
*Đọc file CSV*



**header=True**: Dòng đầu tiên trong file là tên cột.

**inferSchema=True**: Tự động suy luận kiểu dữ liệu của các cột.

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



* Gọi hàm phân tích bằng API DataFrame và lưu kết quả vào output\_df\_path.
* Gọi hàm phân tích bằng SQL và lưu kết quả vào output\_sql\_path.

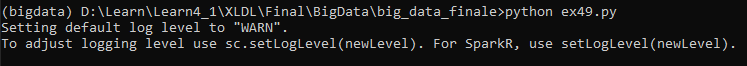
*Dừng phiên làm việc Spark*



Dừng phiên làm việc Spark sau khi hoàn thành các tác vụ.

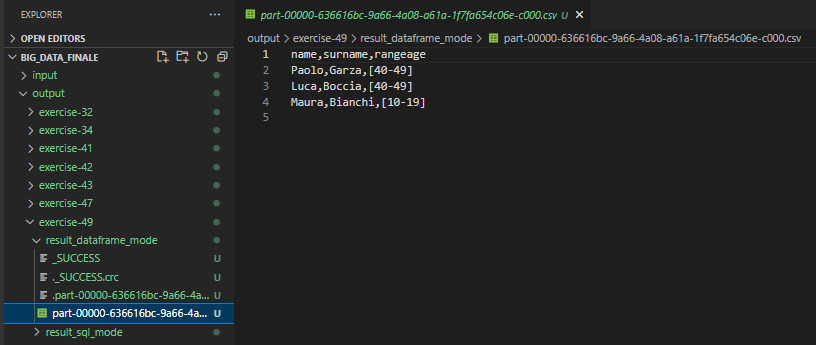
**Kết quả:**

*Chạy code*





*Output được lưu vào thư mục exercise-49/result\_dataframe\_mode/part-00000-636616bc-9a66-4a08-a61a-1f7fa654c06e-c000.csv*

**

*Output được lưu vào thư mục exercise-49/result\_sql\_mode/part-00000-bf202aa9-9fba-47d6-8d69-ed581c8f05d3-c000.csv*

