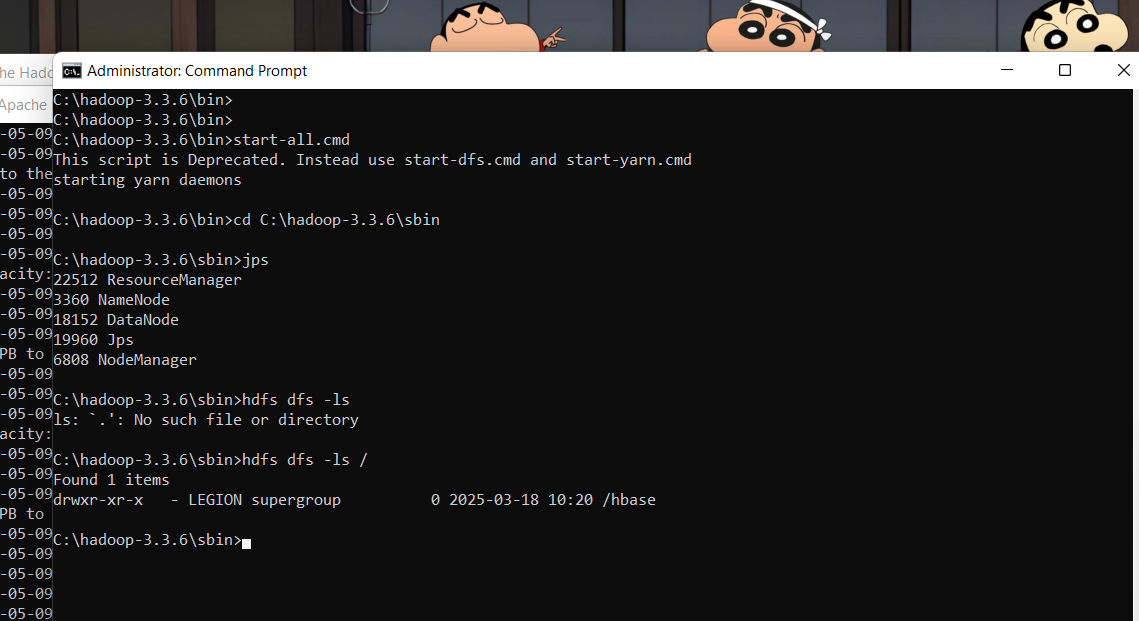
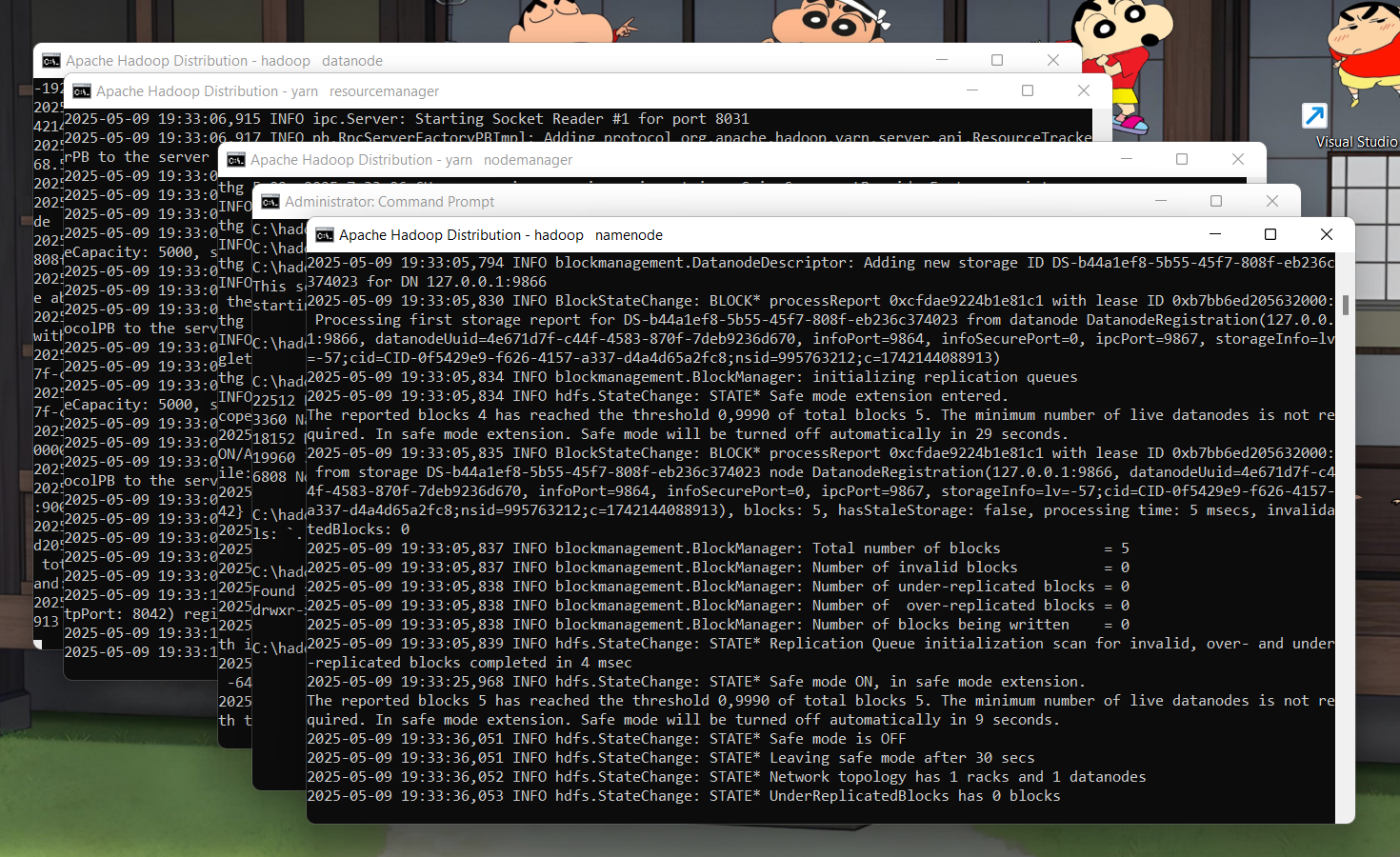
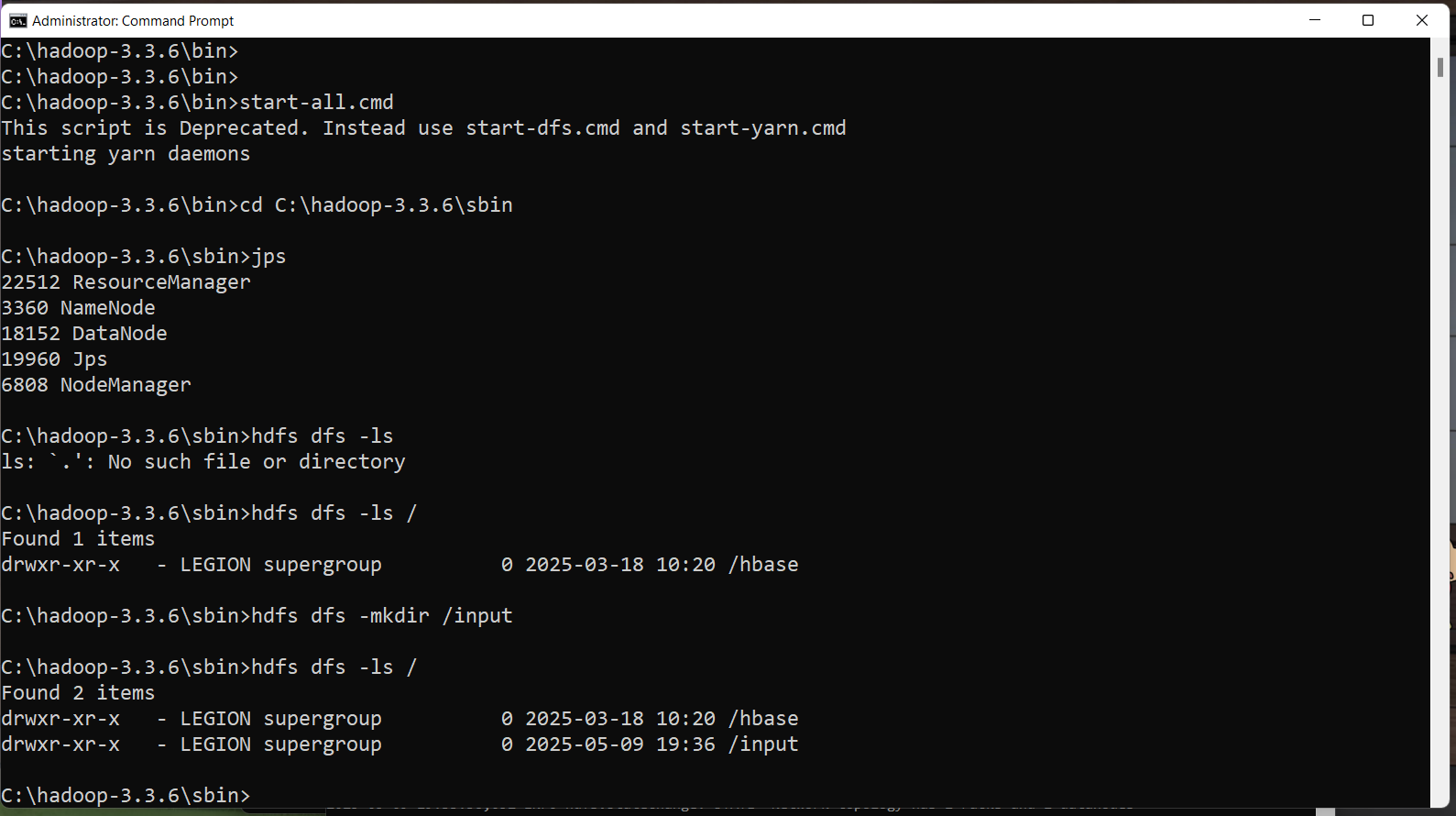
**Chương trình Mapreduce**



* Khởi chạy Hadoop HDFS

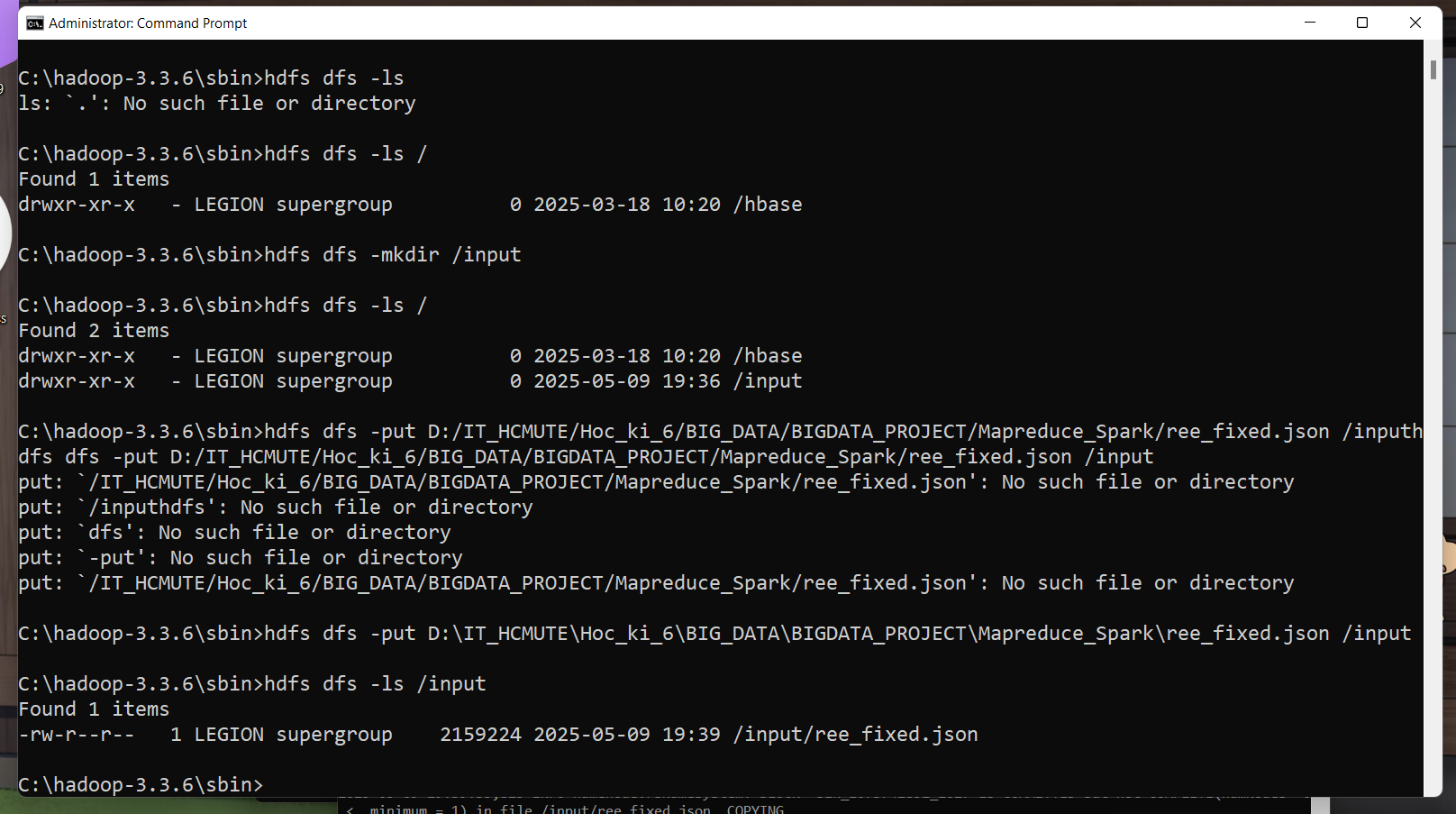


* Tạo thư mục input trên HDFS



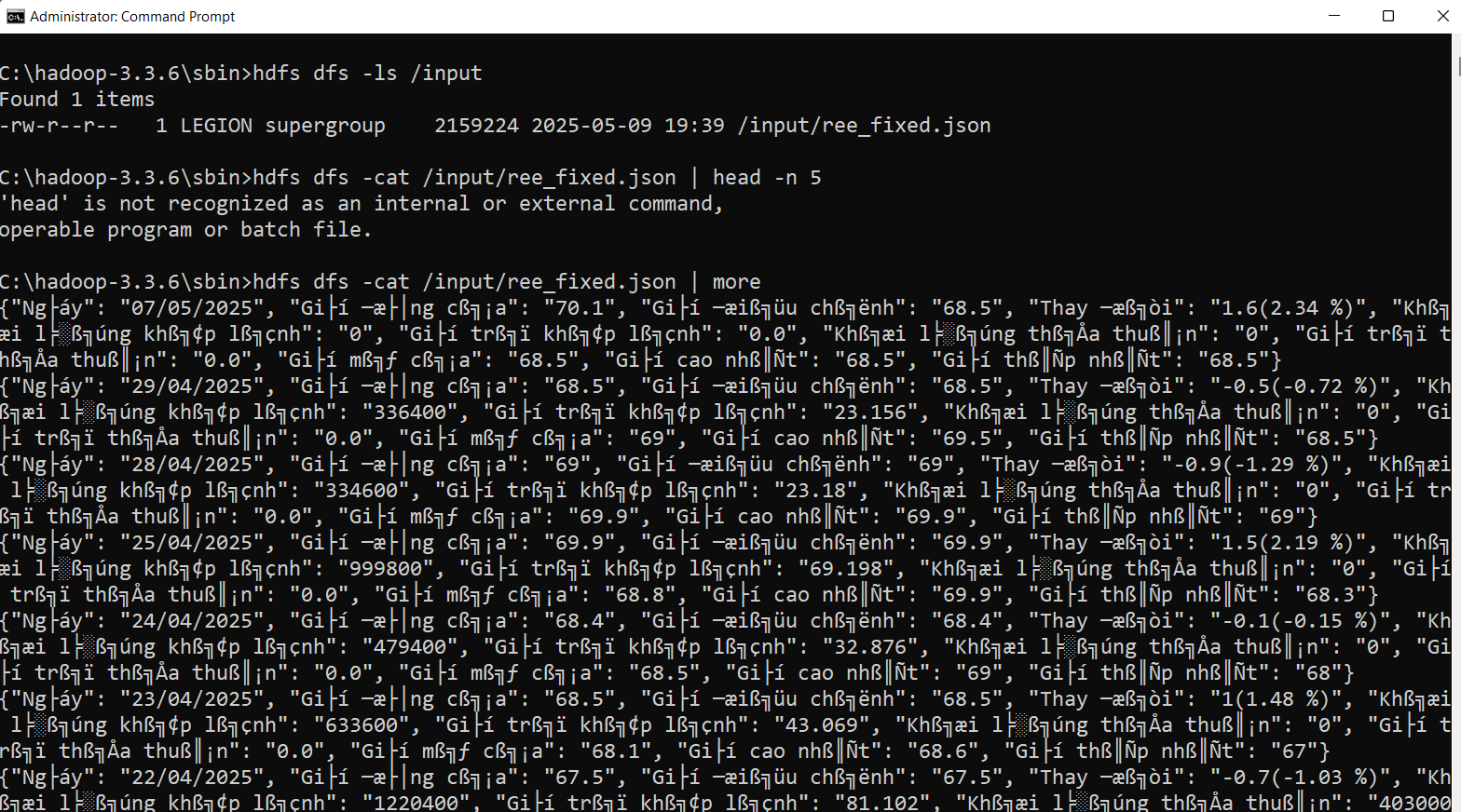
* Tải file đầu vào ree\_fixed.json từ local lên HDFS

$ hdfs dfs -put D:\IT\_HCMUTE\Hoc\_ki\_6\BIG\_DATA\BIGDATA\_PROJECT\Mapreduce\_Spark\ree\_fixed.json /input



* Kiểm tra dữ liệu đã được đưa lên HDFS – Xem 5 dòng đầu tiên của file

$ hdfs dfs -cat /input/ree\_fixed.json | more



* File đưa lên thành công
* Viết chương trình Mapreduce bằng HDFS + Spark (PySpark)

1. **Tính trung bình giá theo từng tháng và gía cao nhất – thấp nhất theo năm (Spark RDD)**

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, StringType

from datetime import datetime

spark = SparkSession.builder \

    .appName("PhanTichDuLieuChungKhoan\_RDD") \

    .master("local[\*]") \

    .config("spark.driver.memory", "4g") \

    .config("spark.executor.memory", "4g") \

    .getOrCreate()

sc = spark.sparkContext

schema = StructType([

    StructField("Ngày", StringType(), True),

    StructField("Giá đóng cửa", StringType(), True),

    StructField("Giá điều chỉnh", StringType(), True),

    StructField("Thay đổi", StringType(), True),

    StructField("Khối lượng khớp lệnh", StringType(), True),

    StructField("Giá trị khớp lệnh", StringType(), True),

    StructField("Khối lượng thỏa thuận", StringType(), True),

    StructField("Giá trị thỏa thuận", StringType(), True),

    StructField("Giá mở cửa", StringType(), True),

    StructField("Giá cao nhất", StringType(), True),

    StructField("Giá thấp nhất", StringType(), True)

])

try:

    df = spark.read.schema(schema).json("hdfs://localhost:9000/input/ree\_fixed.json")

    rdd = df.rdd

except Exception as e:

**print**(f"Error reading JSON file: {e}")

    spark.stop()

    exit(1)

def **parse\_record**(record):

    try:

        if record["Ngày"] is None or record["Giá đóng cửa"] is None:

            return None

        date = datetime.**strptime**(record["Ngày"], "%d/%m/%Y")

        year = date.year

        month = date.month

        close\_price = float(record["Giá đóng cửa"])

        return (year, month, close\_price)

    except (ValueError, TypeError) as e:

**print**(f"Error parsing record: {record}, error: {e}")

        return None

parsed\_rdd = rdd.map(**parse\_record**).filter(lambda x: x is not None).cache()

# Trung bình theo tháng

monthly\_avg\_rdd = parsed\_rdd.map(lambda x: ((x[0], x[1]), (x[2], 1))) \

    .reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1])) \

    .map(lambda x: (x[0][0], x[0][1], **round**(x[1][0] / x[1][1], 2))) \

    .sortBy(lambda x: (x[0], x[1]))

# Cao nhất và thấp nhất theo năm

min\_max\_price\_yearly = parsed\_rdd.map(lambda x: (x[0], x[2])) \

    .groupByKey() \

    .map(lambda x: (x[0], (**round**(**max**(x[1]), 2), **round**(**min**(x[1]), 2)))) \

    .sortBy(lambda x: x[0])

monthly\_avg\_df = spark.createDataFrame(

    monthly\_avg\_rdd,

    ["Năm", "Tháng", "Giá\_Đóng\_Cửa\_Trung\_Bình"]

)

yearly\_extremes\_df = spark.createDataFrame(

    min\_max\_price\_yearly.map(lambda x: (x[0], x[1][0], x[1][1])),

    ["Năm", "Giá\_Đóng\_Cửa\_Cao\_Nhất", "Giá\_Đóng\_Cửa\_Thấp\_Nhất"]

)

output\_dir = "hdfs://localhost:9000/output/ree\_1"

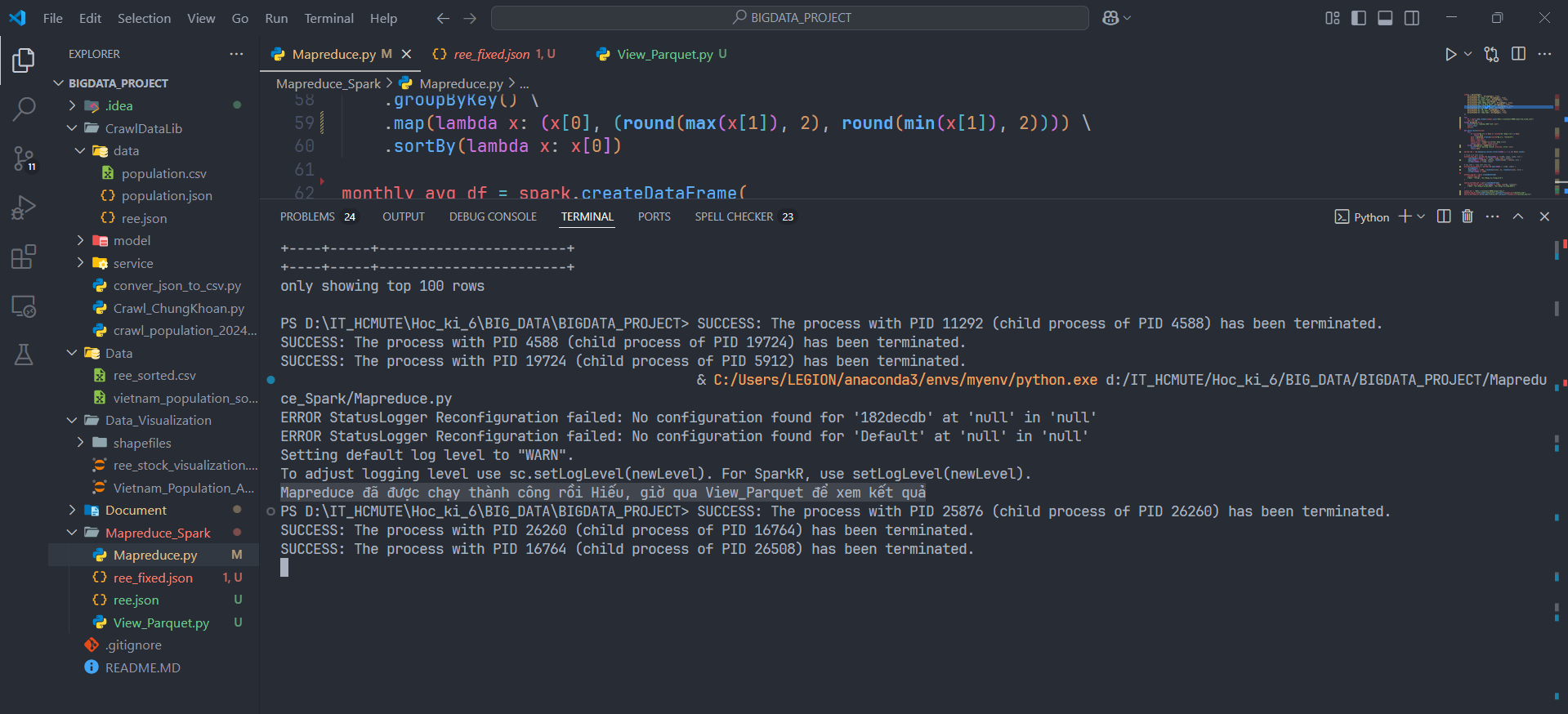
monthly\_avg\_df.write.mode("overwrite").parquet(f"{output\_dir}/monthly\_avg")

yearly\_extremes\_df.write.mode("overwrite").parquet(f"{output\_dir}/yearly\_extremes")

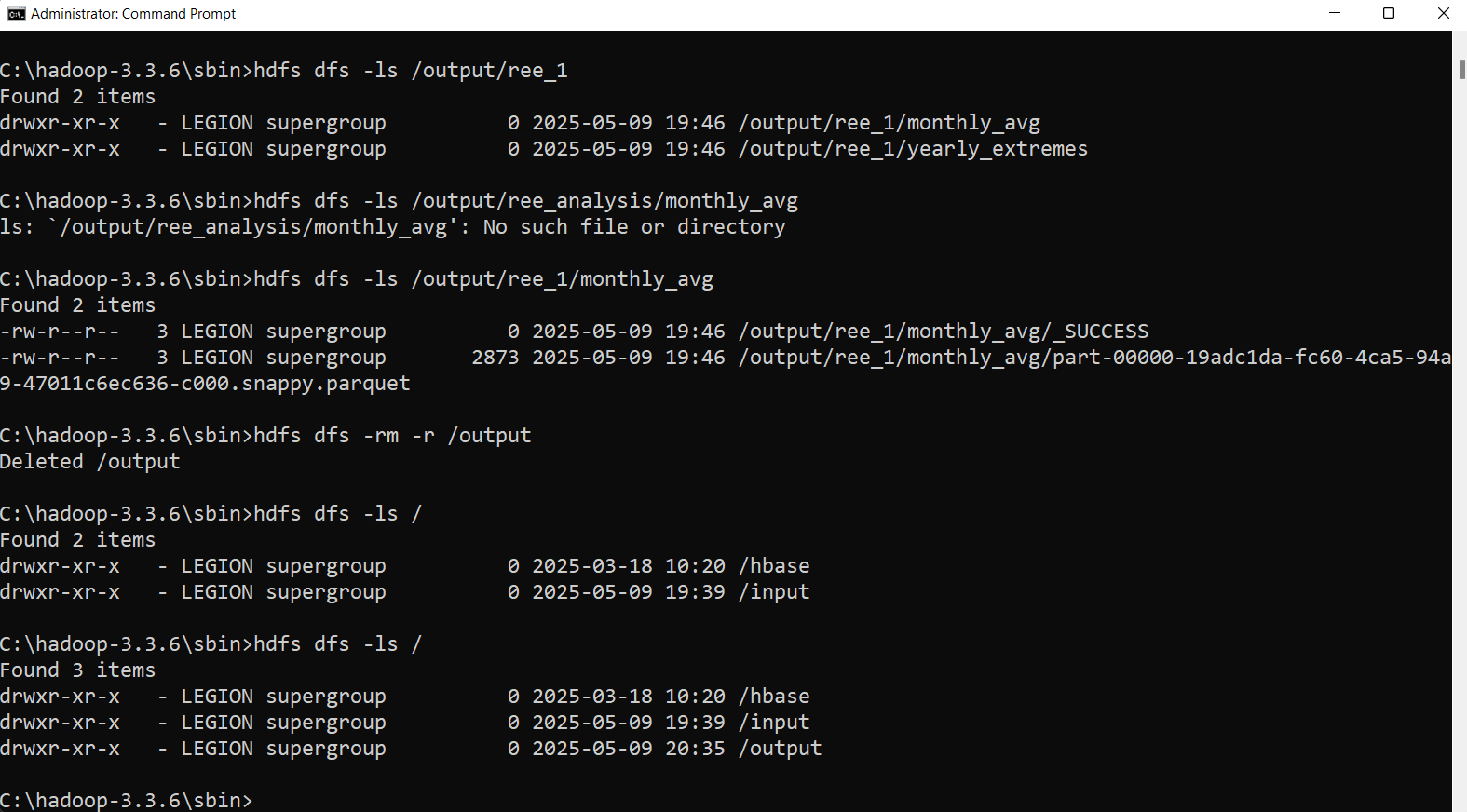
**print**("Mapreduce đã được chạy thành công rồi Hiếu, giờ qua View\_Parquet để xem kết quả")

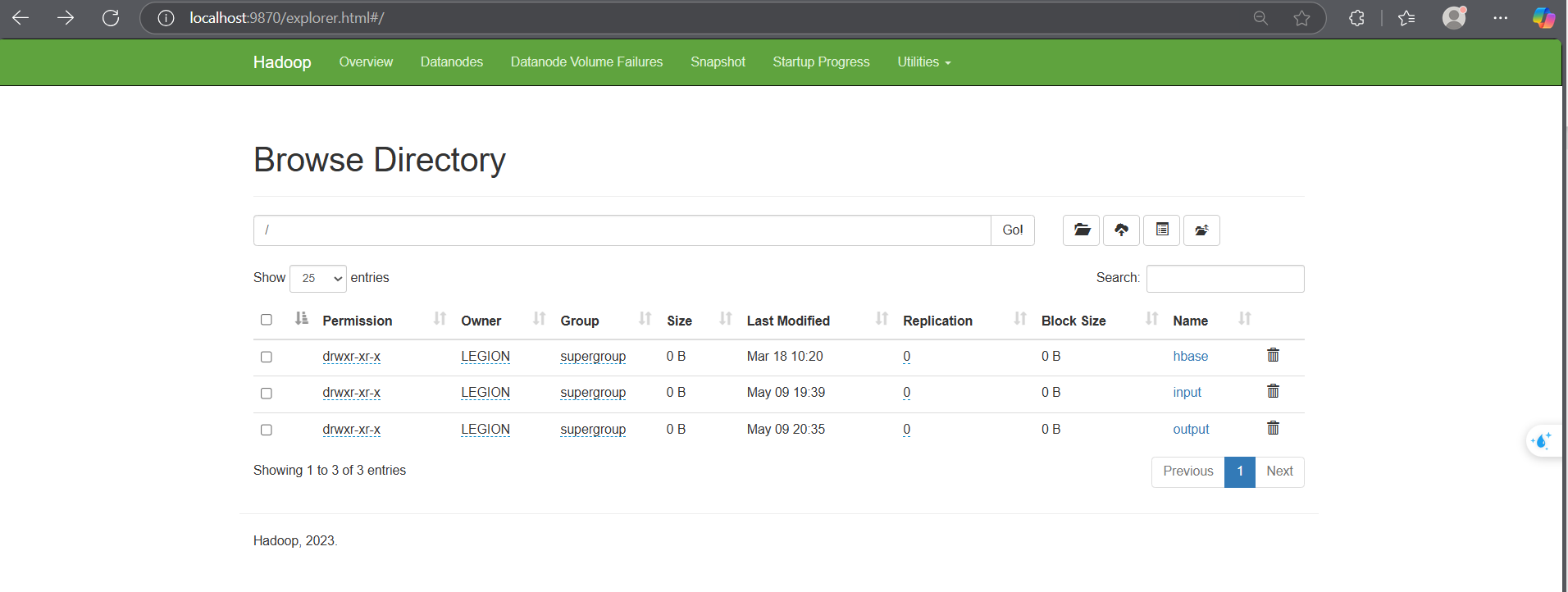
spark.stop()

* Sau khi viết chương trình Mapreduce và đưa file lên HDFS thành công, tiến hành chạy file để thực thi

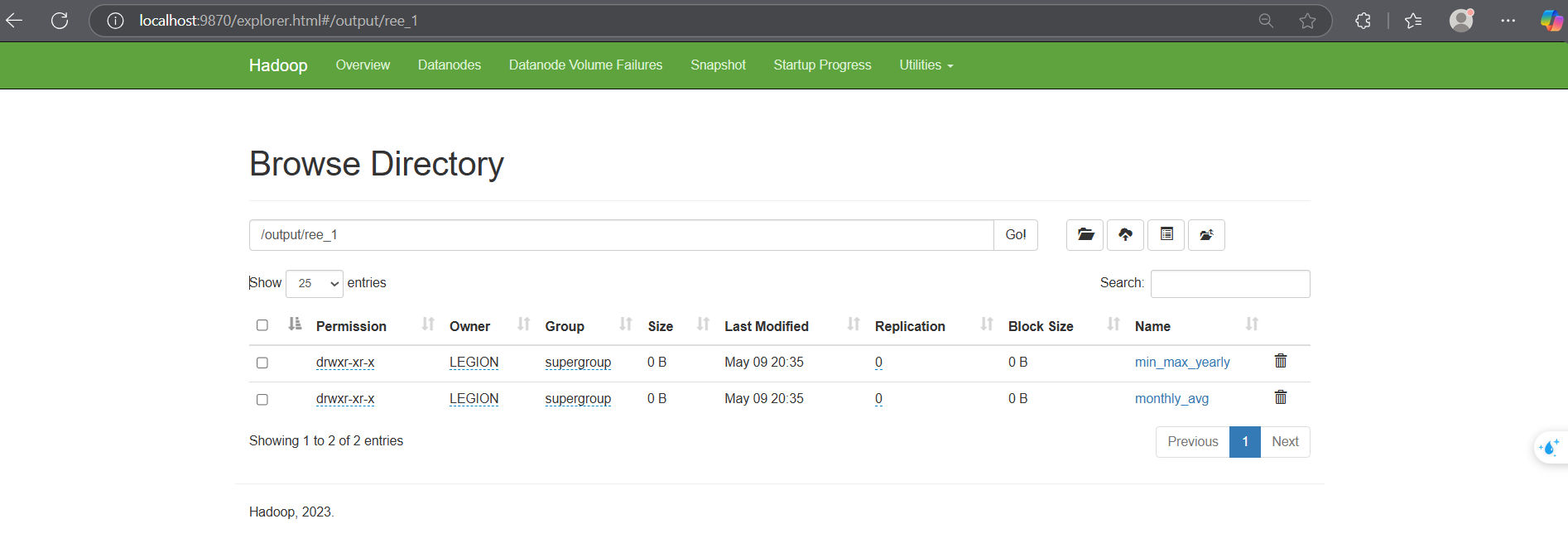


* Kiểm tra thư mục output trên HDFS





* Thư mục output đã được tạo 🡪 Chương trình Mapreduce chạy thành công



* Trong thư mục ree\_1 có hai thư mục đầu ra của 2 nội dung Mapreduce

**Lưu ý:** Vì file đầu ra của chương trình là file .parquet

**Parquet** là một định dạng file lưu trữ dữ liệu dạng cột (columnar storage format) được tối ưu hóa cho các hệ thống xử lý dữ liệu lớn như Apache Hadoop và Apache Spark. Khác với định dạng lưu trữ theo dòng (row-based) như CSV hay JSON, định dạng cột của Parquet giúp cải thiện hiệu năng truy vấn dữ liệu và giảm kích thước lưu trữ nhờ khả năng nén tốt hơn. Parquet hỗ trợ schema rõ ràng, khả năng nén từng cột riêng biệt, và đọc chọn lọc từng cột mà không cần tải toàn bộ dữ liệu vào bộ nhớ. Điều này giúp tăng tốc độ xử lý dữ liệu đáng kể, đặc biệt trong các bài toán phân tích dữ liệu lớn

Vì file Parquet là định dạng nhị phân, nên ta không thể coi trực tiếp bằng lệnh -cat thông thường như trong Linux mà sẽ đọc file Parquet bằng Python

* Viết file View\_Parquet để tiến hành đọc file parquet đầu ra

from pyspark.sql import SparkSession

spark = SparkSession.builder \

    .appName("ViewParquetResults") \

    .master("local[\*]") \

    .config("spark.driver.memory", "4g") \

    .config("spark.executor.memory", "4g") \

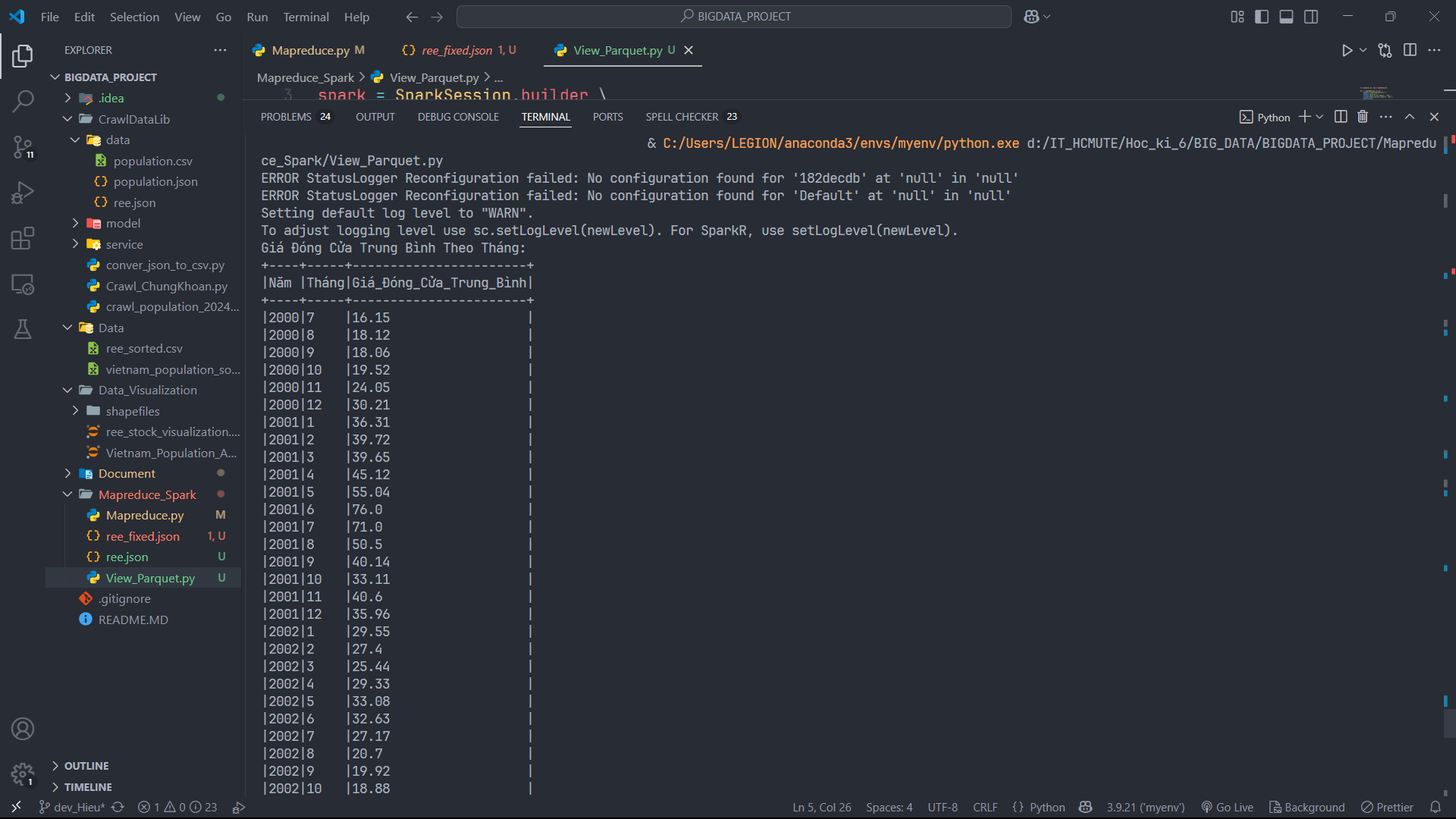
    .getOrCreate()

monthly\_avg\_df = spark.read.parquet("hdfs://localhost:9000/output/ree\_1/monthly\_avg")

**print**("Giá Đóng Cửa Trung Bình Theo Tháng:")

monthly\_avg\_df.show(100, truncate=False)

spark.stop()



* Xem output thành công

from pyspark.sql import SparkSession

spark = SparkSession.builder \

    .appName("ViewParquetResults") \

    .master("local[\*]") \

    .config("spark.driver.memory", "4g") \

    .config("spark.executor.memory", "4g") \

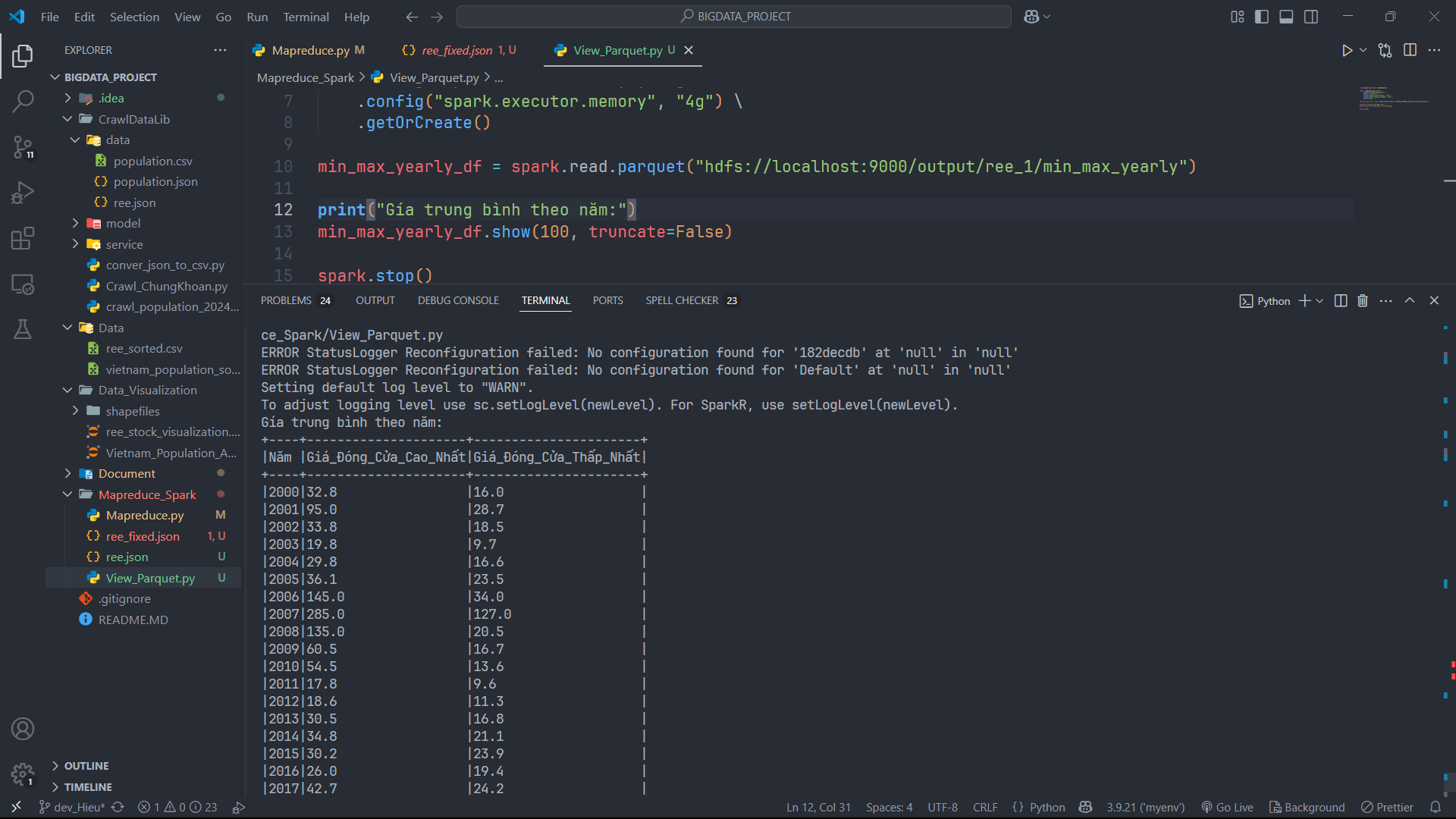
    .getOrCreate()

min\_max\_yearly\_df = spark.read.parquet("hdfs://localhost:9000/output/ree\_1/min\_max\_yearly")

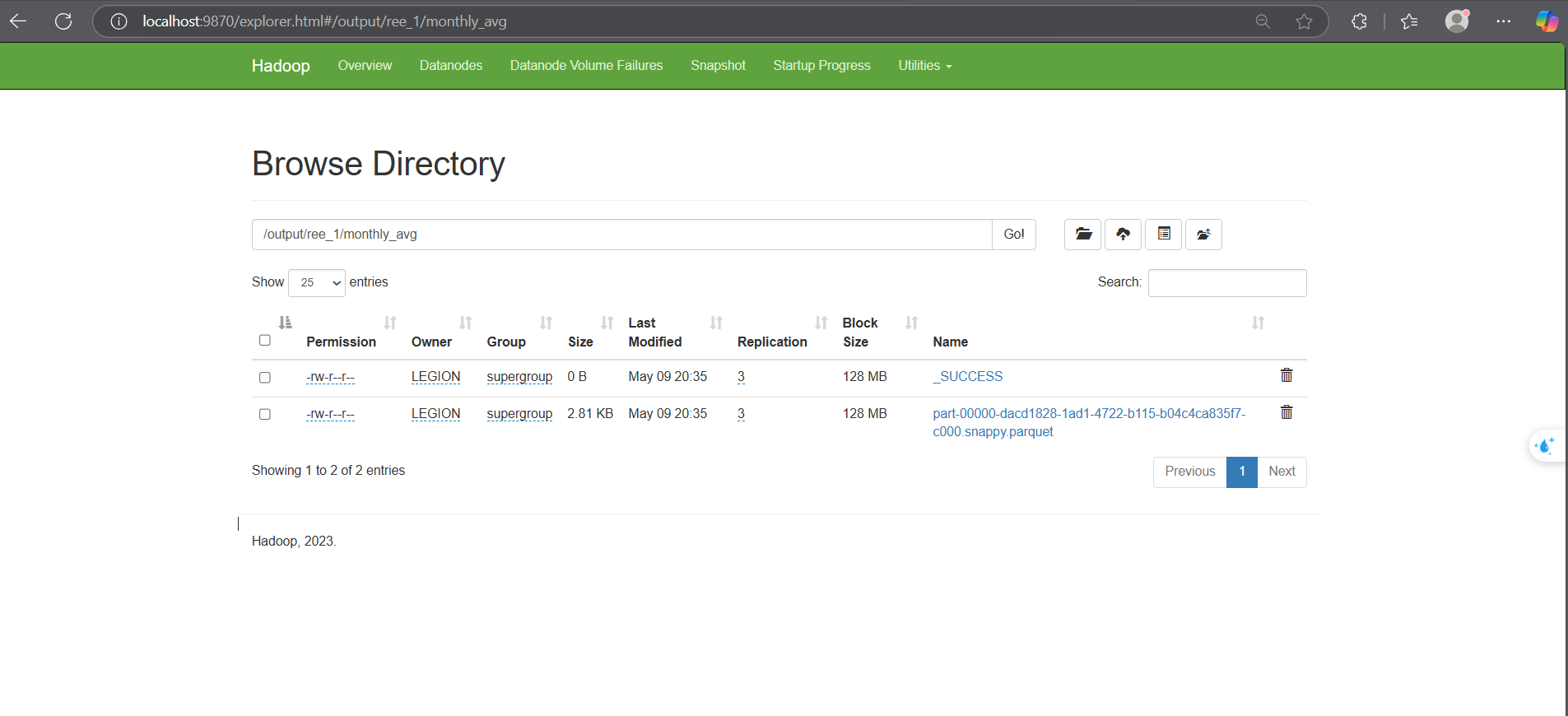
**print**("Gía trung bình theo năm:")

min\_max\_yearly\_df.show(100, truncate=False)

spark.stop()



* Xem thành công file Mapreduce của Gía trung bình theo tháng (giá cao nhất và giá thấp nhất)



1. **Tính phần trăm ngày tăng giá / giảm giá / không đổi (Spark RDD)**

from Spark\_config import **get\_spark\_session**

from pyspark.sql.types import StructType, StructField, StringType

import re

spark = **get\_spark\_session**(app\_name="BienDongGia")

sc = spark.sparkContext

schema = StructType([

    StructField("Ngày", StringType(), True),

    StructField("Giá đóng cửa", StringType(), True),

    StructField("Giá điều chỉnh", StringType(), True),

    StructField("Thay đổi", StringType(), True),

    StructField("Khối lượng khớp lệnh", StringType(), True),

    StructField("Giá trị khớp lệnh", StringType(), True),

    StructField("Khối lượng thỏa thuận", StringType(), True),

    StructField("Giá trị thỏa thuận", StringType(), True),

    StructField("Giá mở cửa", StringType(), True),

    StructField("Giá cao nhất", StringType(), True),

    StructField("Giá thấp nhất", StringType(), True)

])

try:

    df = spark.read.schema(schema).json("hdfs://localhost:9000/input/ree\_fixed.json")

    rdd = df.rdd

except Exception as e:

**print**(f"Error reading JSON file: {e}")

    spark.stop()

    exit(1)

# Xử lý bản ghi của file

def **parse\_movement**(record):

    try:

        change = record["Thay đổi"]

        if change is None:

            return None

        match = re.**match**(r"([-]?\d\*\.?\d+)", change)

        if not match:

            return None

        value = float(match.**group**(1))

        if value > 0:

            return ("Tăng", 1)

        elif value < 0:

            return ("Giảm", 1)

        else:

            return ("Không đổi", 1)

    except (ValueError, TypeError) as e:

**print**(f"Error parsing record: {record}, error: {e}")

        return None

movement\_rdd = rdd.map(**parse\_movement**) \

    .filter(lambda x: x is not None) \

    .reduceByKey(lambda x, y: x + y) \

    .cache()

tong\_ngay\_rdd = movement\_rdd.map(lambda x: x[1]).reduce(lambda x, y: x + y)

phan\_tram\_rdd = movement\_rdd.map(lambda x: (x[0], **round**((x[1] / tong\_ngay\_rdd) \* 100, 2)))

percent\_df = spark.createDataFrame(

    phan\_tram\_rdd,

    ["Loại biến động", "Phần trăm (%)"]

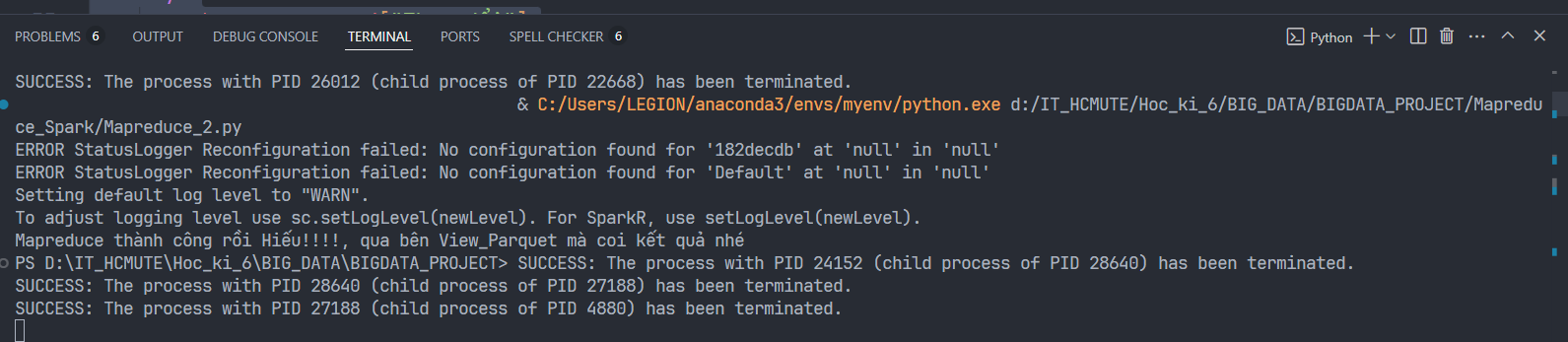
)

output\_dir = "hdfs://localhost:9000/output/ree\_2"

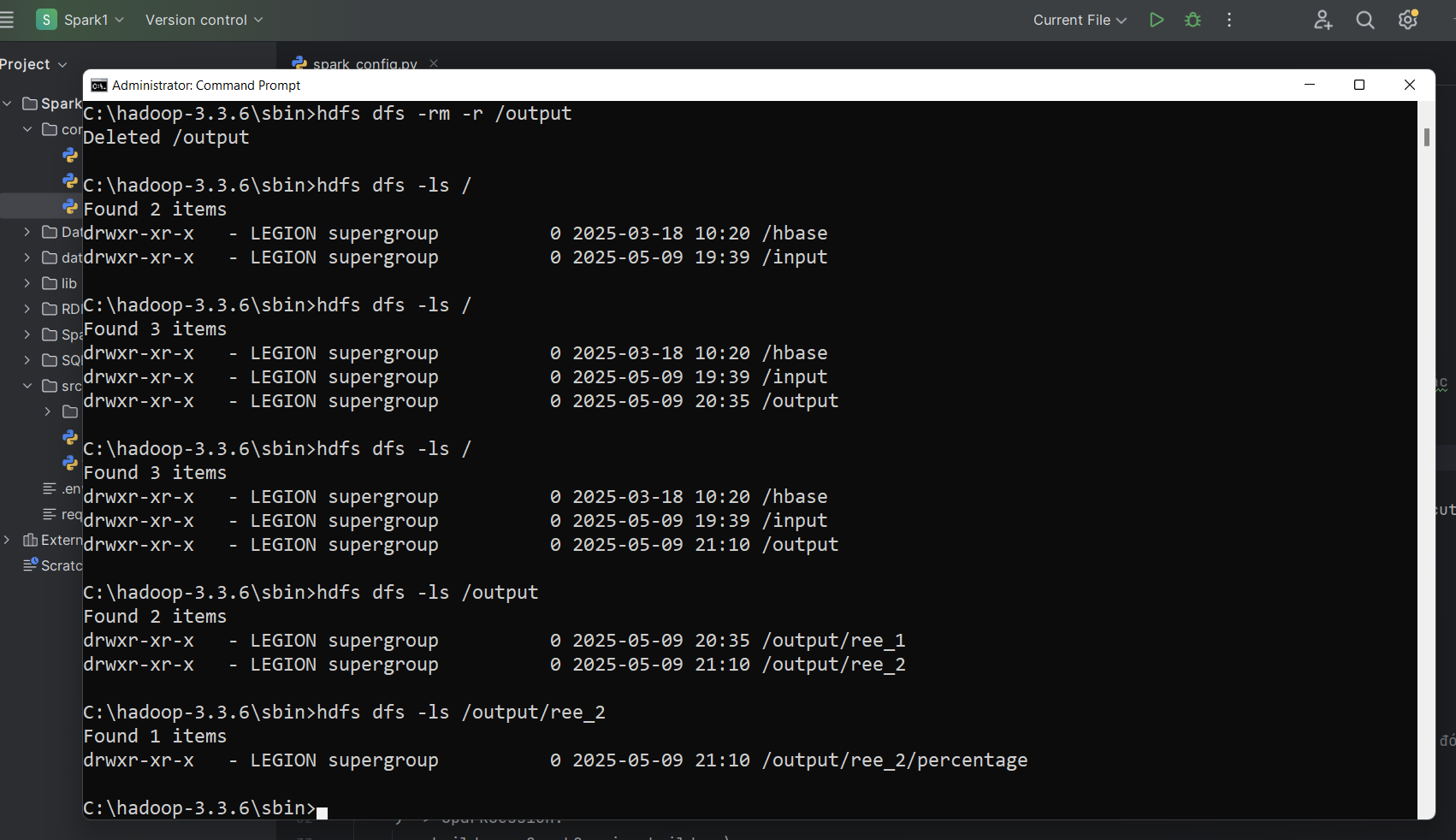
percent\_df.write.mode("overwrite").parquet(f"{output\_dir}/percentage")

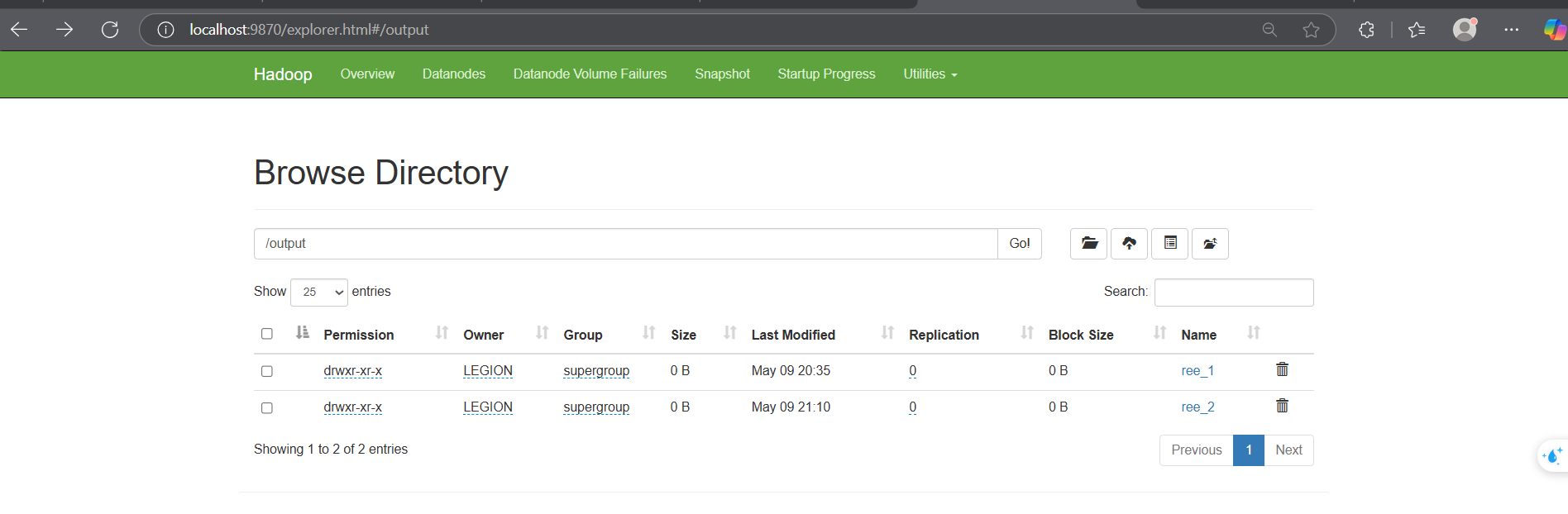
**print**("Mapreduce thành công rồi Hiếu!!!!, qua bên View\_Parquet mà coi kết quả nhé")

spark.stop()

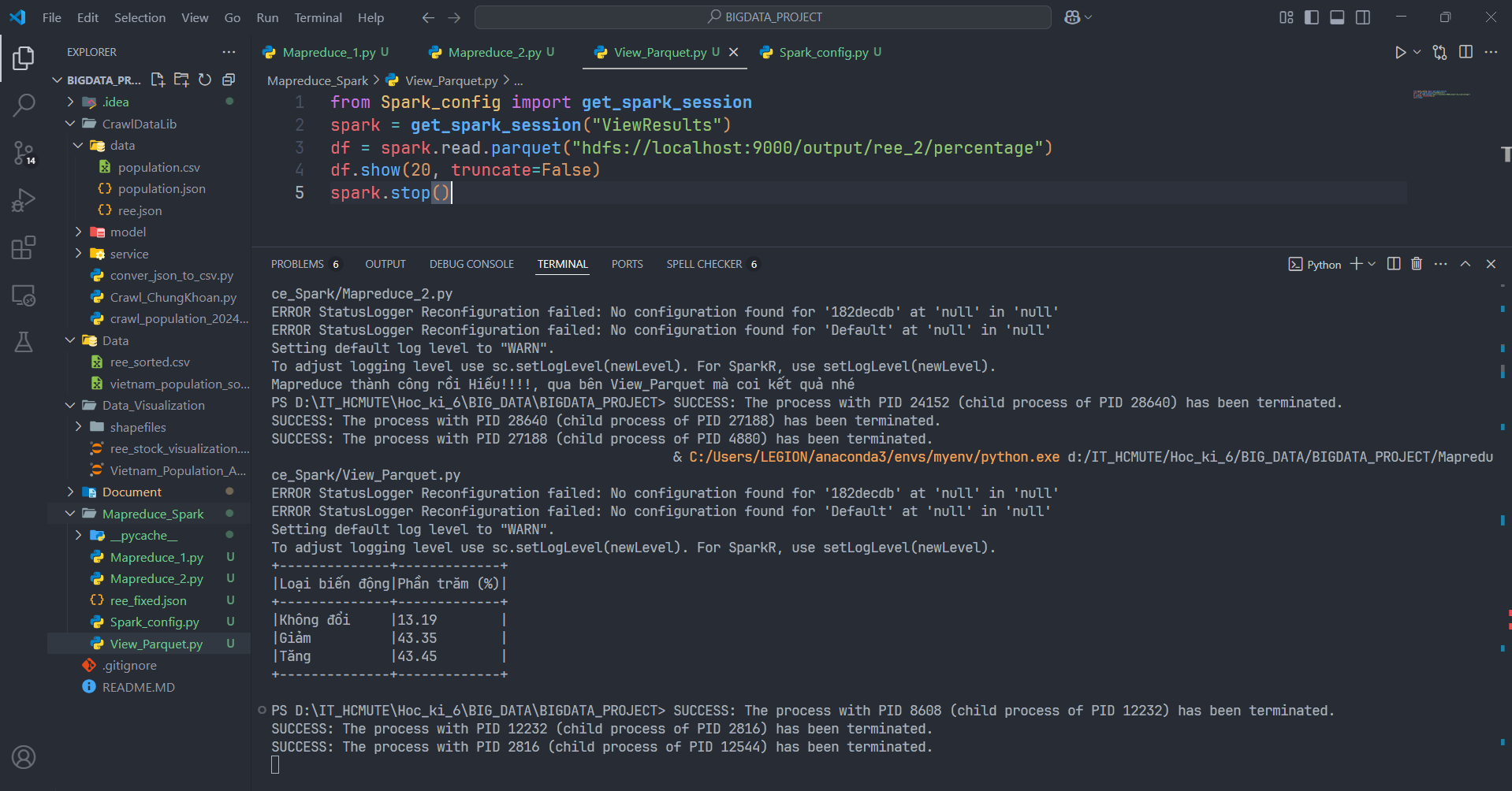


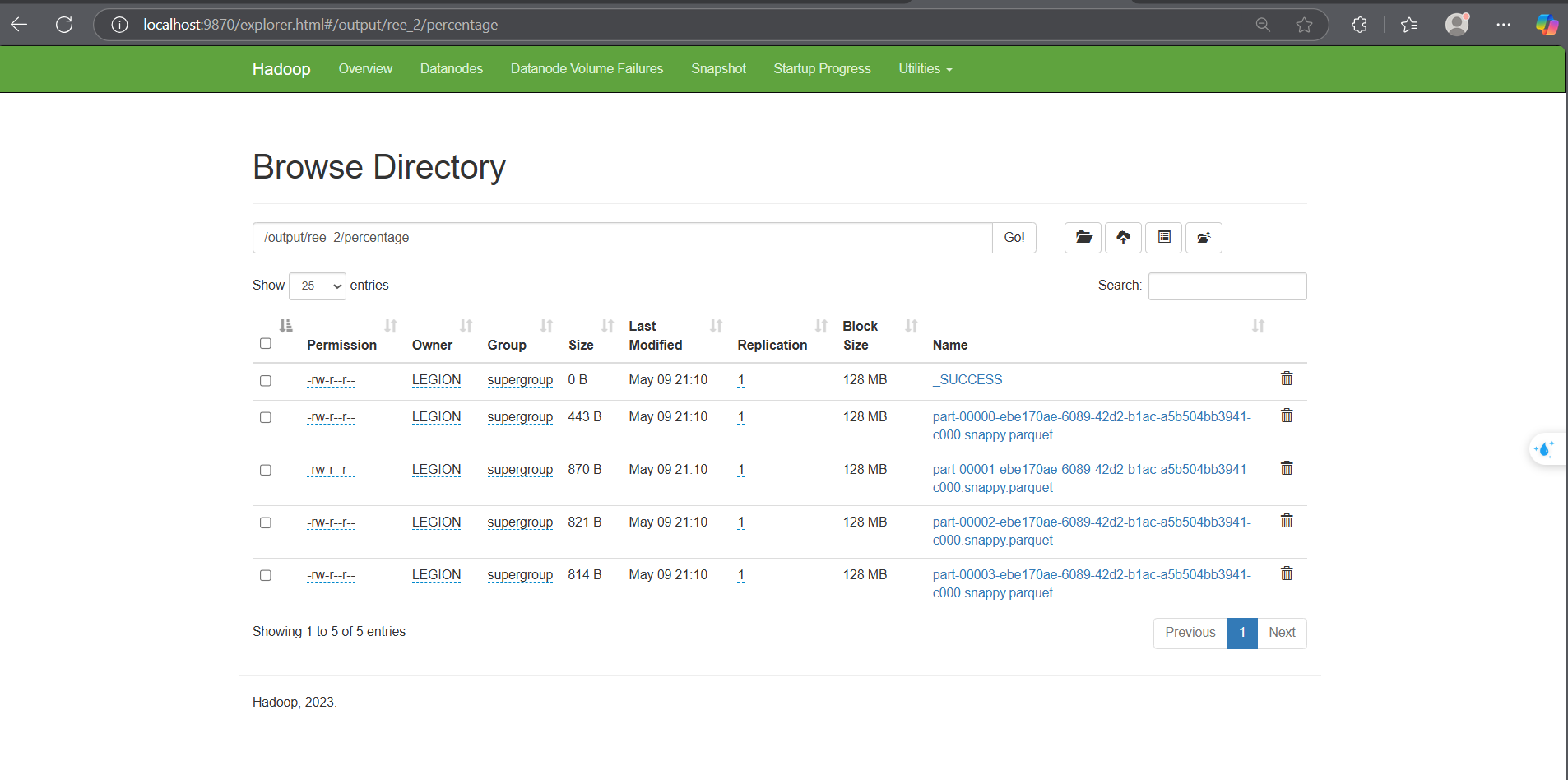
* Kiểm tra thư mục trên HDFS





* Xem kết quả trên View\_Parquet





* Trong bài này đã sử dụng Spark\_config và kết quả đầu ra cho ra nhiều file Parquet như trên là do đâu??

Vì trong cấu hình Spark, có sự phân vùng dữ liệu thành nhiều phân vùng (partitions) để xử lý song song trên các executor, mỗi phân vùng nó sẽ được lưu thành một file Parquet riêng biệt

Trong Spark\_config.py ta đã cấu hình: .config("spark.sql.shuffle.partitions", "8"), điều này cho ta thấy Spark sẽ tạo tối đa 8 phân vùng khi thực hiện các thao tác shuffle, nhưng vì kết quả đầu ra có kích thước nhỏ chỉ chứa 3 trường Tăng, Giảm, Không đổi, nên nó chỉ chia 4 phân vùng

1. **Tính tổng khối lượng giao dịch (khớp lệnh + thỏa thuận) theo từng tuần (Spark DataFrame)**

from Spark\_config import **get\_spark\_session**

from pyspark.sql.types import StructType, StructField, StringType

from pyspark.sql.functions import **col**, **to\_date**, **year**, **weekofyear**

spark = **get\_spark\_session**(app\_name="WeeklyVolumeAnalysisDataFrame")

schema = StructType([

    StructField("Ngày", StringType(), True),

    StructField("Giá đóng cửa", StringType(), True),

    StructField("Giá điều chỉnh", StringType(), True),

    StructField("Thay đổi", StringType(), True),

    StructField("Khối lượng khớp lệnh", StringType(), True),

    StructField("Giá trị khớp lệnh", StringType(), True),

    StructField("Khối lượng thỏa thuận", StringType(), True),

    StructField("Giá trị thỏa thuận", StringType(), True),

    StructField("Giá mở cửa", StringType(), True),

    StructField("Giá cao nhất", StringType(), True),

    StructField("Giá thấp nhất", StringType(), True)

])

try:

    df = spark.read.schema(schema).json("hdfs://localhost:9000/input/ree\_fixed.json")

except Exception as e:

**print**(f"Error reading JSON file: {e}")

    spark.stop()

    exit(1)

weekly\_volume\_df = df \

    .filter(**col**("Ngày").isNotNull() &

**col**("Khối lượng khớp lệnh").isNotNull() &

**col**("Khối lượng thỏa thuận").isNotNull()) \

    .withColumn("Date", **to\_date**(**col**("Ngày"), "dd/MM/yyyy")) \

    .withColumn("Năm", **year**(**col**("Date"))) \

    .withColumn("Tuần", **weekofyear**(**col**("Date"))) \

    .withColumn("Khối lượng khớp lệnh", **col**("Khối lượng khớp lệnh").**cast**("float")) \

    .withColumn("Khối lượng thỏa thuận", **col**("Khối lượng thỏa thuận").**cast**("float")) \

    .withColumn("Tổng Khối Lượng", **col**("Khối lượng khớp lệnh") + **col**("Khối lượng thỏa thuận")) \

    .groupBy("Năm", "Tuần") \

    .sum("Tổng Khối Lượng") \

    .withColumnRenamed("sum(Tổng Khối Lượng)", "Tổng Khối Lượng Giao Dịch") \

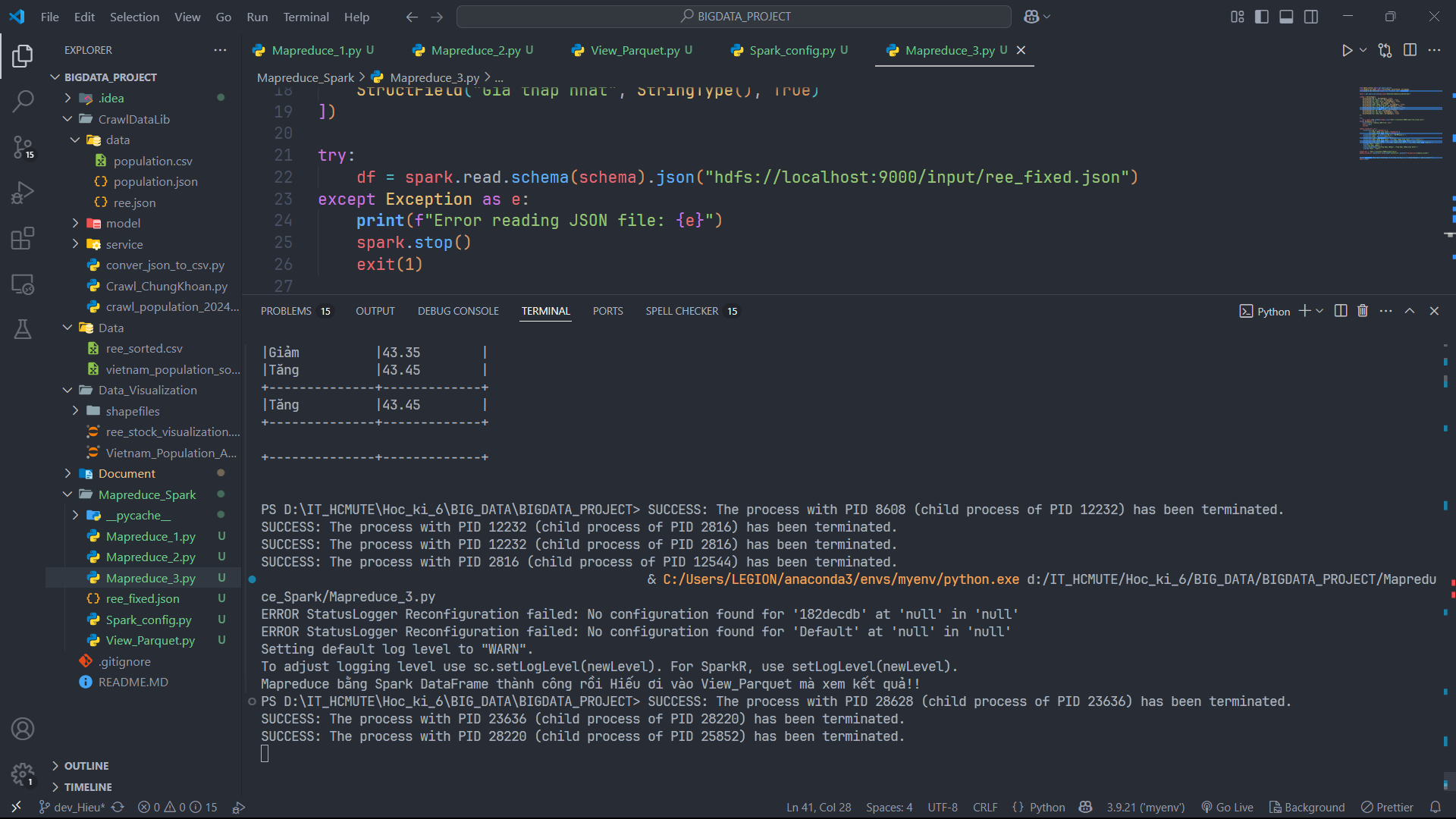
    .orderBy("Năm", "Tuần")

output\_dir = "hdfs://localhost:9000/output/ree\_3"

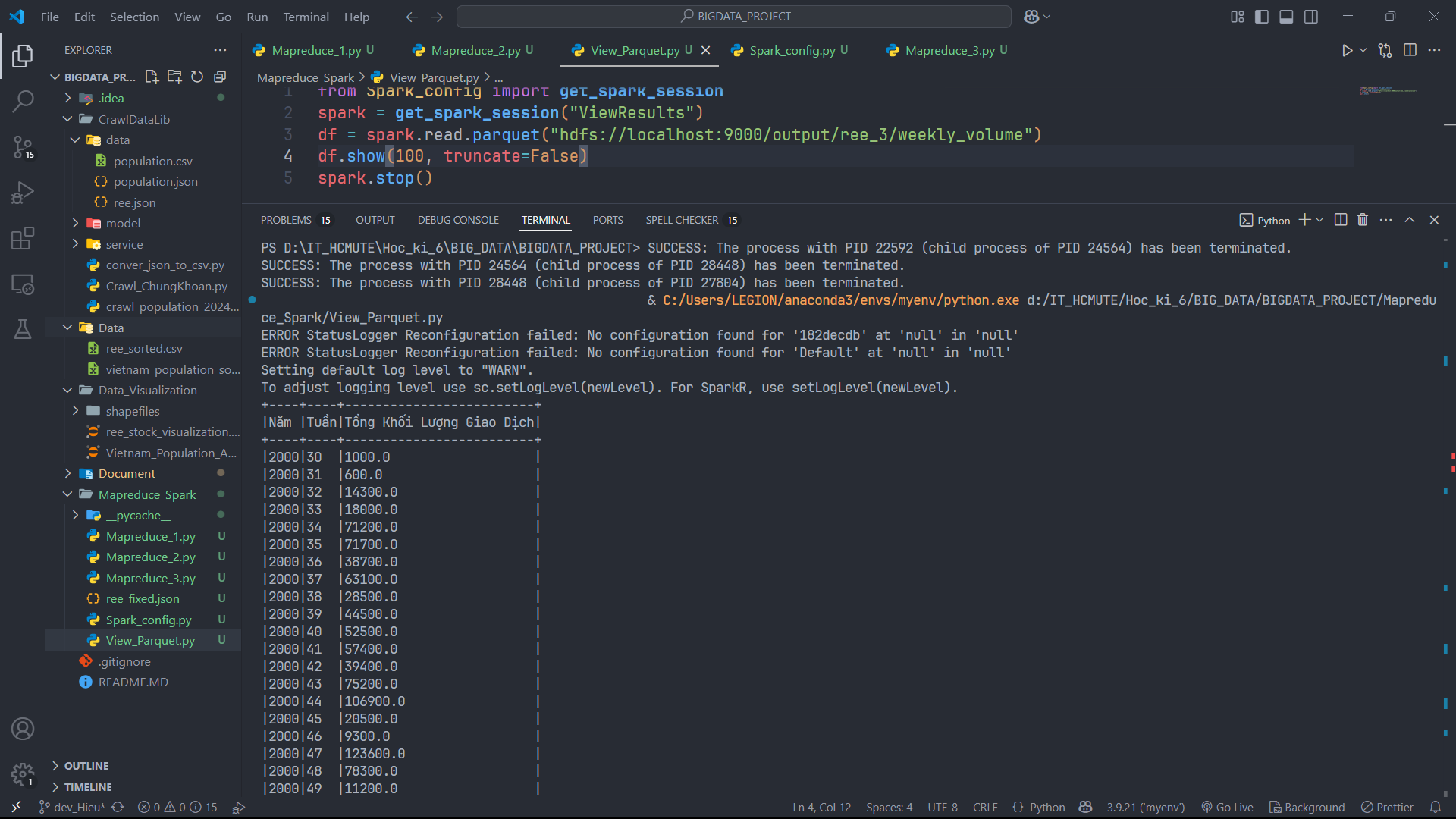
weekly\_volume\_df.coalesce(1).write.mode("overwrite").parquet(f"{output\_dir}/weekly\_volume")

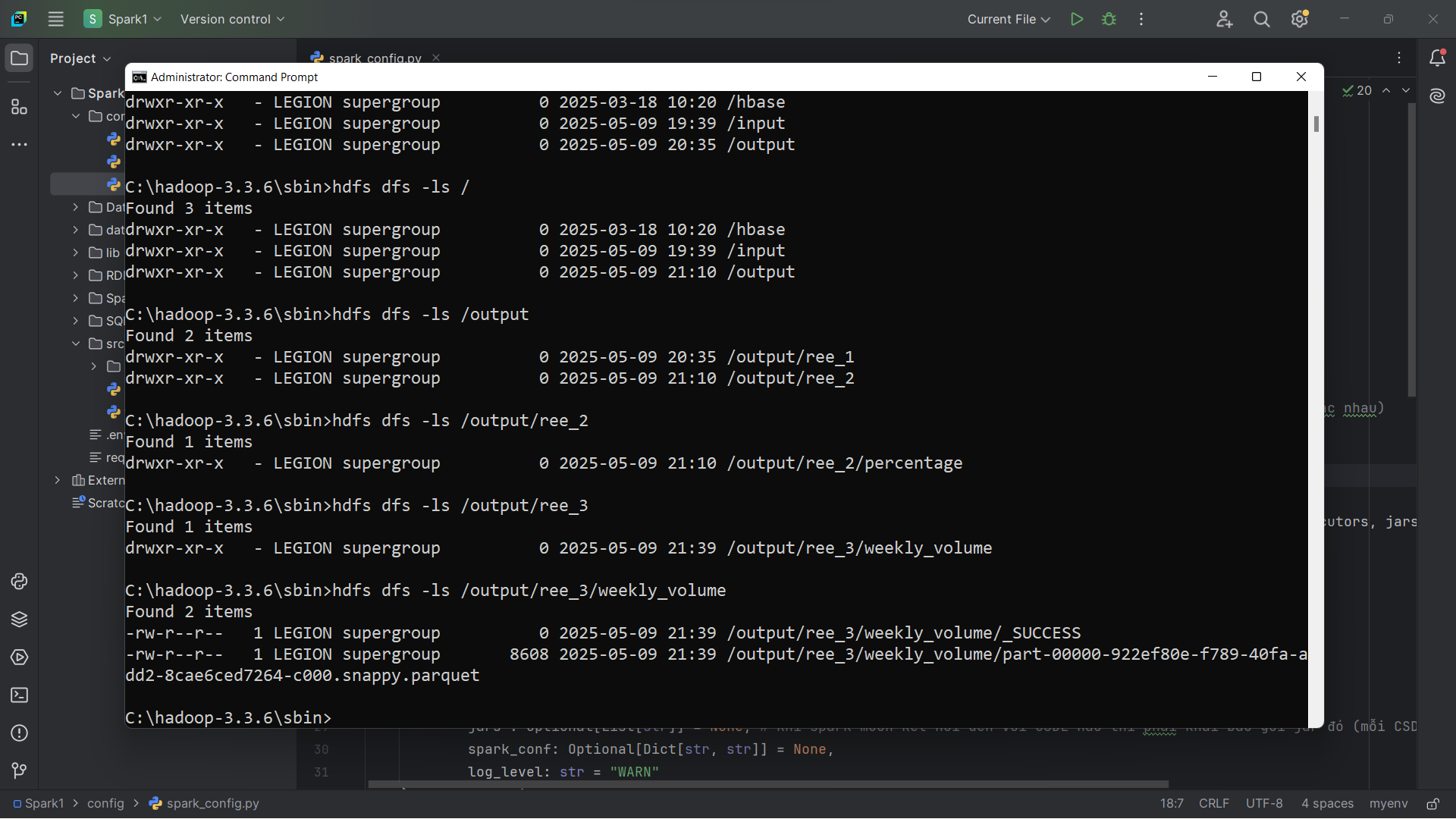
**print**("Mapreduce bằng Spark DataFrame thành công rồi Hiếu ơi vào View\_Parquet mà xem kết quả!!")

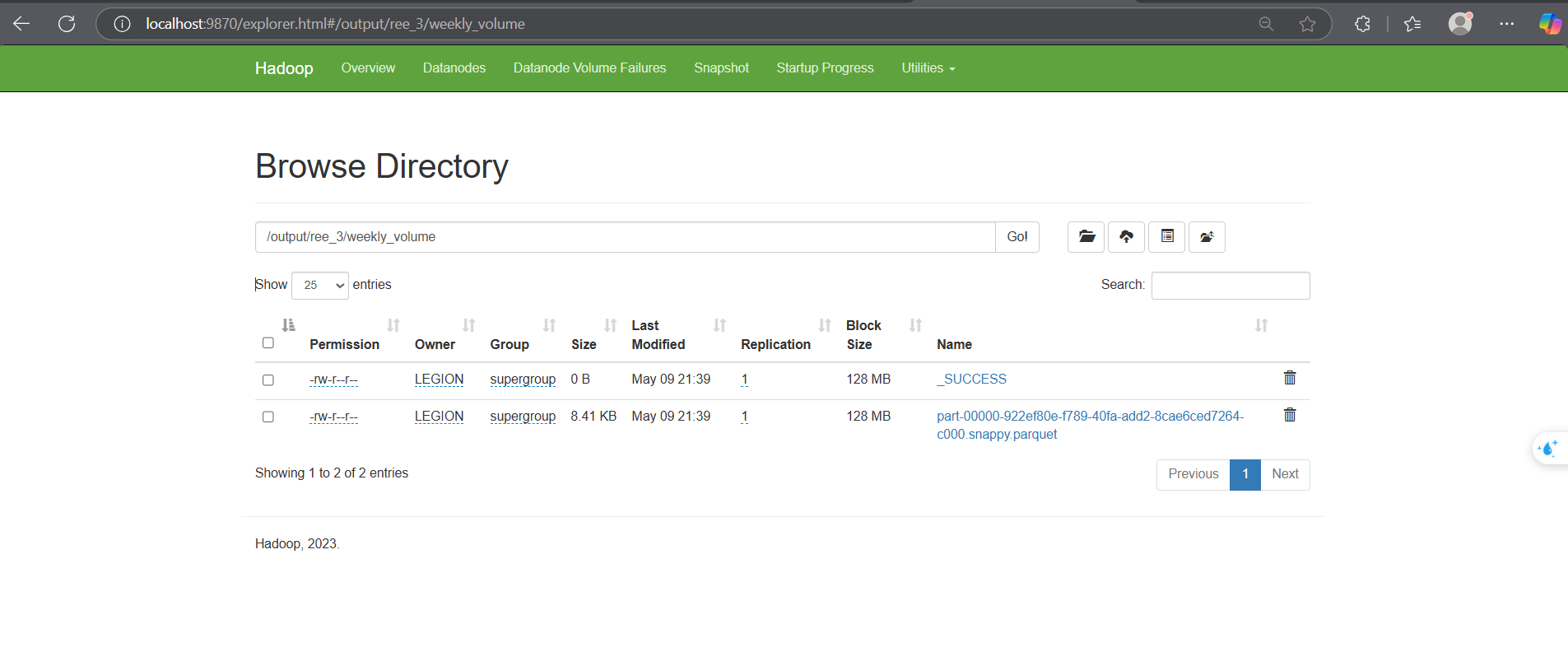
spark.stop()

****

* **Xem kết quả**

****

****



1. **Tìm ngày có giá đóng cửa cao nhất và thấp nhất mỗi năm (Spark DataFrame)**

from Spark\_config import **get\_spark\_session**

from pyspark.sql.types import StructType, StructField, StringType

from pyspark.sql.functions import **col**, **to\_date**, **year**, **max**, **min**

spark = **get\_spark\_session**(app\_name="YearlyExtremesDatesDataFrame")

schema = StructType([

    StructField("Ngày", StringType(), True),

    StructField("Giá đóng cửa", StringType(), True),

    StructField("Giá điều chỉnh", StringType(), True),

    StructField("Thay đổi", StringType(), True),

    StructField("Khối lượng khớp lệnh", StringType(), True),

    StructField("Giá trị khớp lệnh", StringType(), True),

    StructField("Khối lượng thỏa thuận", StringType(), True),

    StructField("Giá trị thỏa thuận", StringType(), True),

    StructField("Giá mở cửa", StringType(), True),

    StructField("Giá cao nhất", StringType(), True),

    StructField("Giá thấp nhất", StringType(), True)

])

try:

    df = spark.read.schema(schema).json("hdfs://localhost:9000/input/ree\_fixed.json")

except Exception as e:

**print**(f"Error reading JSON file: {e}")

    spark.stop()

    exit(1)

df\_min\_max\_price\_date = df \

    .filter(**col**("Ngày").isNotNull() & **col**("Giá đóng cửa").isNotNull()) \

    .withColumn("Date", **to\_date**(**col**("Ngày"), "dd/MM/yyyy")) \

    .withColumn("Năm", **year**(**col**("Date"))) \

    .withColumn("Giá đóng cửa", **col**("Giá đóng cửa").**cast**("float"))

yearly\_extremes = df\_min\_max\_price\_date \

    .groupBy("Năm") \

    .agg(

**max**("Giá đóng cửa").**alias**("Giá Cao Nhất"),

**min**("Giá đóng cửa").**alias**("Giá Thấp Nhất")

    )

df\_alias = df\_min\_max\_price\_date.alias("df")

yearly\_extremes\_alias = yearly\_extremes.alias("extremes")

max\_dates = df\_alias.join(

    yearly\_extremes\_alias,

    (**col**("df.Năm") == **col**("extremes.Năm")) &

    (**col**("df.Giá đóng cửa") == **col**("extremes.Giá Cao Nhất")),

    "inner"

).select(

**col**("df.Năm"),

**col**("df.Ngày").**alias**("Ngày Cao Nhất"),

**col**("df.Giá đóng cửa").**alias**("Giá Cao Nhất")

).distinct()

min\_dates = df\_alias.join(

    yearly\_extremes\_alias,

    (**col**("df.Năm") == **col**("extremes.Năm")) &

    (**col**("df.Giá đóng cửa") == **col**("extremes.Giá Thấp Nhất")),

    "inner"

).select(

**col**("df.Năm"),

**col**("df.Ngày").**alias**("Ngày Thấp Nhất"),

**col**("df.Giá đóng cửa").**alias**("Giá Thấp Nhất")

).distinct()

max\_dates\_alias = max\_dates.alias("max\_dates")

min\_dates\_alias = min\_dates.alias("min\_dates")

result\_df = max\_dates\_alias.join(

    min\_dates\_alias,

**col**("max\_dates.Năm") == **col**("min\_dates.Năm"),

    "outer"

).select(

**col**("max\_dates.Năm"),

**col**("max\_dates.Ngày Cao Nhất"),

**col**("max\_dates.Giá Cao Nhất"),

**col**("min\_dates.Ngày Thấp Nhất"),

**col**("min\_dates.Giá Thấp Nhất")

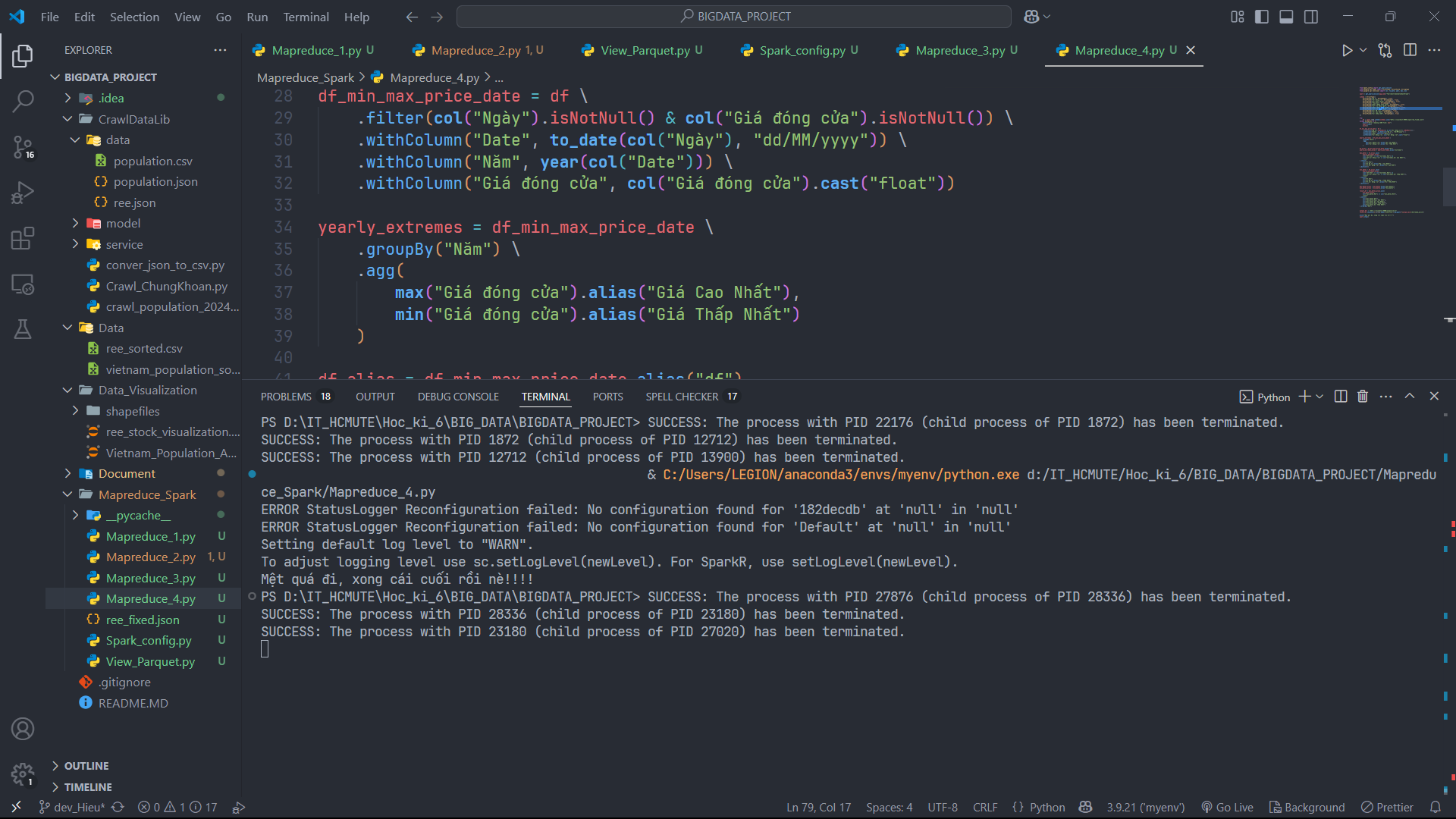
).orderBy("Năm")

output\_dir = "hdfs://localhost:9000/output/ree\_4"

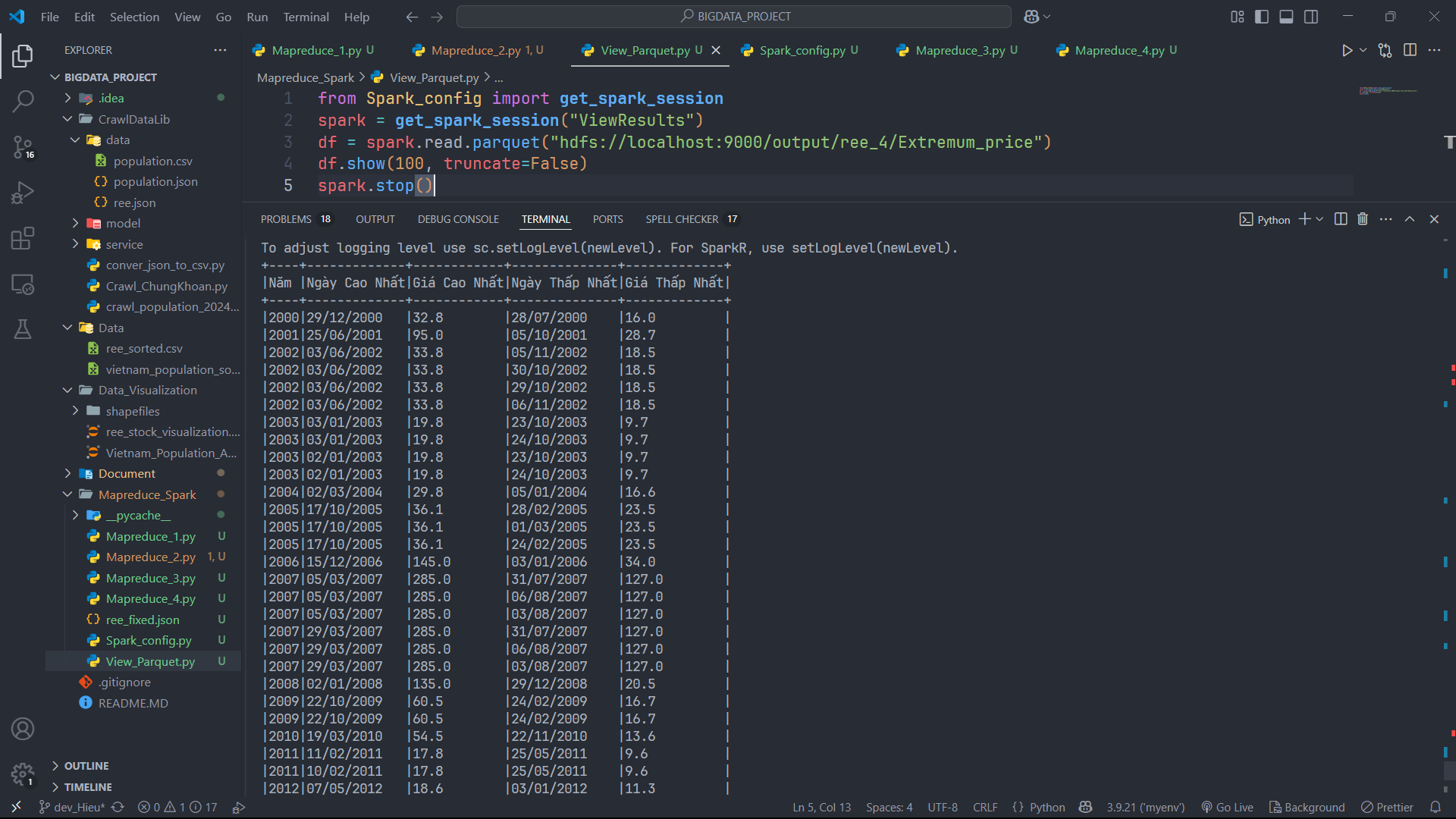
result\_df.coalesce(1).write.mode("overwrite").parquet(f"{output\_dir}/Extremum\_price")

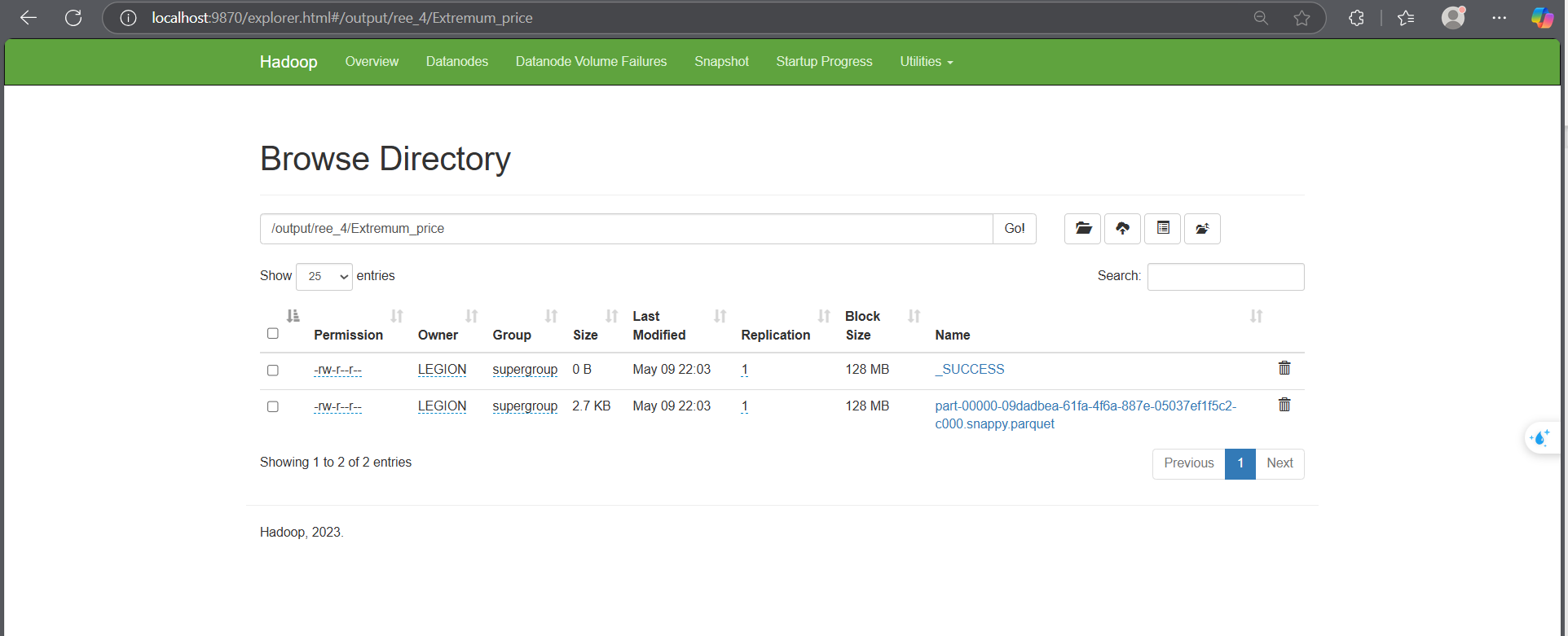
**print**("Mệt quá đi, xong cái cuối rồi nè!!!!")

spark.stop()



* Mapreduce thành công và xem kết quả





Ta thấy được chạy Mapreduce trên HDFS Apache Spark tốc độ xử lý hơn rất nhiều so với Hadoop Mapreduce. Bởi vì Apache Spark xử lý trên In memory (CPU + RAM) nên tốc độ nhanh hơn rất nhiều so với Hadoop Mapreduce (Cơ chế Disk-based ghi đọc liên tục lên HDFS giữa các bước). Nếu nói về tốc độ thì Apache Spark rất là nhanh đặc biết với các job phức tạp và nhiều bước, còn Hadoop Mapreduce chậm hơn do phải ghi đĩa sau mỗi bước Map/Reduce

Bên Apache Spark sử dụng API linh hoạt hơn (nhờ Spark SQL, DataFrame, Streaming,..) còn bên Hadoop Mapreduce là một mô hình cứng nhắc Map -> Shuffle -> Reduce