Group-16

BDA Mini Project (CA-2)

Class : D20A

Roll No Name

53 Hemant Patil

31 Vittal Kumar

69 Dhiraj Vinjar

43 Arya Nagariya

Q3) Write MapReduce/Spark Program to perform

a)Matrix Multiplication :

from pyspark.sql import SparkSession

from pyspark.sql import functions as F

spark = SparkSession.builder \

.appName("MatrixMultiplicationExample") \

.getOrCreate()

matrix\_A\_data = [(1, 2, 3),

(4, 5, 6),

(7, 8, 9)]

matrix\_B\_data = [(9, 8, 7),

(6, 5, 4),

(3, 2, 1)]

matrix\_A\_df = spark.createDataFrame(matrix\_A\_data, ["A1", "A2", "A3"]) matrix\_B\_df = spark.createDataFrame(matrix\_B\_data, ["B1", "B2", "B3"])

result\_matrix = matrix\_A\_df.crossJoin(matrix\_B\_df) \

.withColumn("result", sum(F.col("A{0}".format(i + 1)) \* F.col("B{0}".format(j + 1))

for i in range(3) for j in range(3))) \

.select("result") \

.rdd.zipWithIndex() \

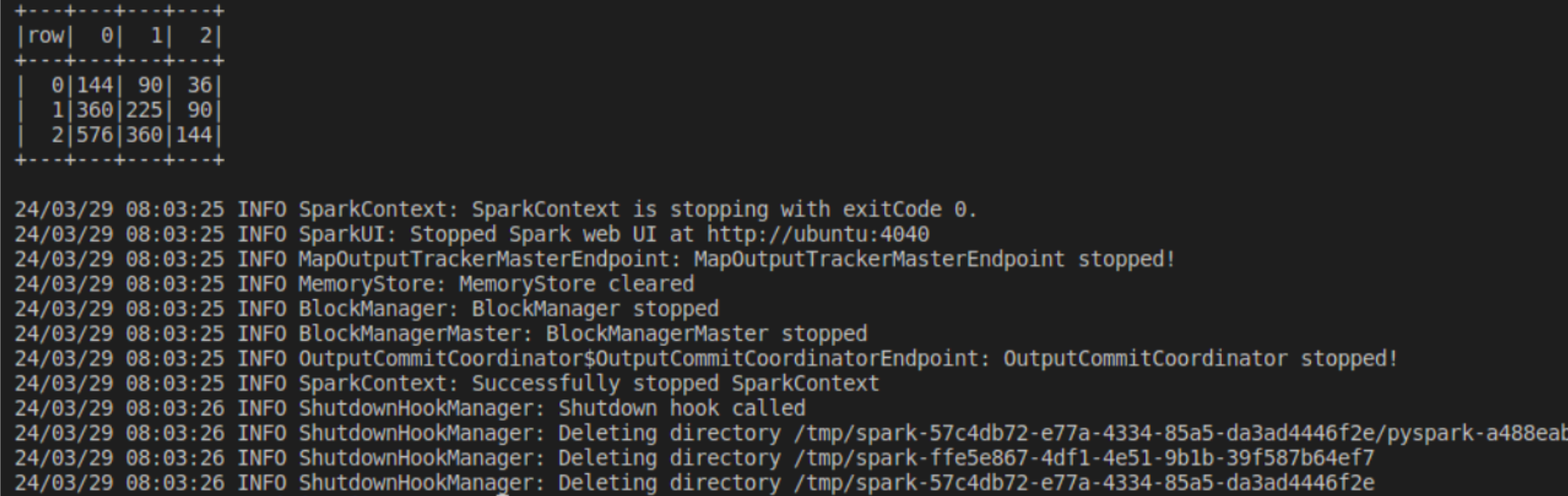
.map(lambda x: (x[1] // 3, x[1] % 3, x[0][0])) \

.toDF(["row", "col", "value"]) \

.groupBy("row").pivot("col").agg(F.first("value"))

result\_matrix.show()

spark.stop()

Output : 

b)Aggregations (Mean, Sum, Standard Deviation) :

from pyspark.sql import SparkSession

from pyspark.sql import functions as F

spark = SparkSession.builder \

.appName("Aggregations") \

.getOrCreate()

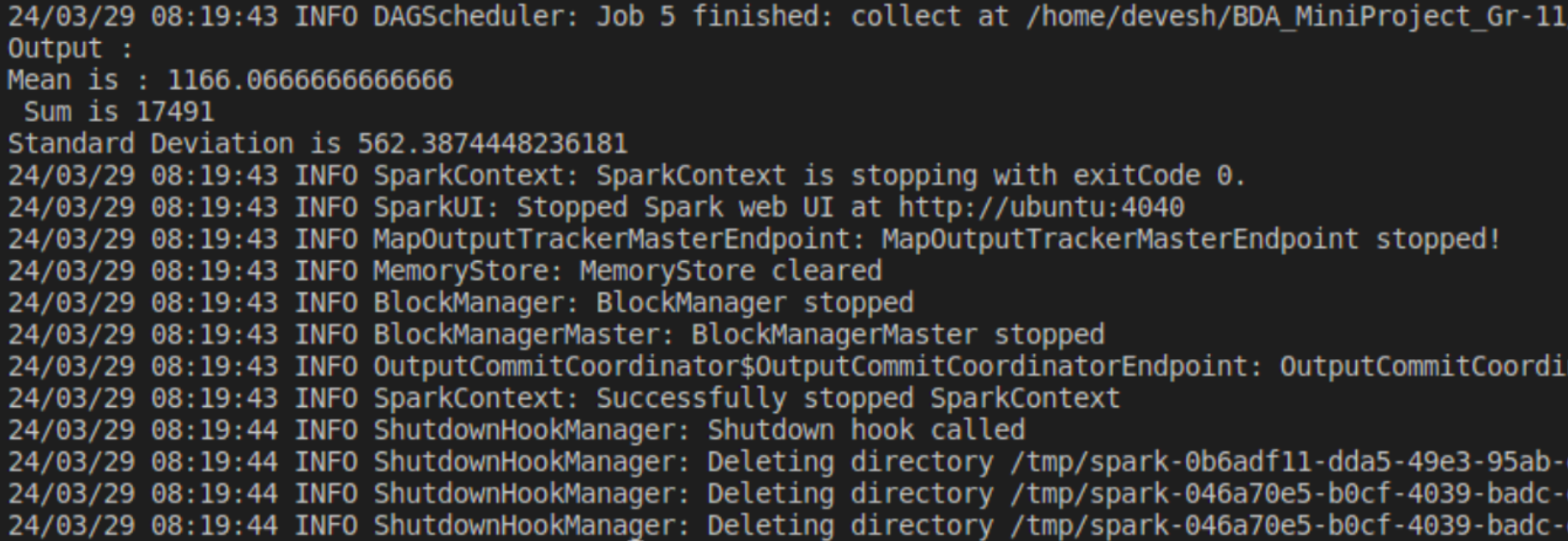
data = [(225,), (346,), (518,), (687,), (823,), (944,), (1056,), (1223,), (1375,), (1442,), (1565,), (1678,), (1790,), (1876,), (1943,)]

df = spark.createDataFrame(data, ["measurement"])

mean = df.agg(F.mean("measurement")).collect()[0][0] sum\_val = df.agg(F.sum("measurement")).collect()[0][0] std\_dev = df.agg(F.stddev("measurement")).collect()[0][0] print("Output : ")

print(f"Mean is : {mean}\n Sum is {sum\_val} \nStandard Deviation is {std\_dev}" )

spark.stop()

Output : 

c)Sorting :

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("SortingExample") \

.getOrCreate()

data = [

(101, "John", 50000),

(102, "Alice", 60000),

(103, "Bob", 45000),

(104, "Emily", 70000),

(105, "Michael", 55000),

(106, "Emma", 62000),

(107, "David", 48000)

]

df = spark.createDataFrame(data, ["emp\_id", "emp\_name", "salary"])

print("Before Sorting:")

df.show()

sorted\_df = df.orderBy("salary")

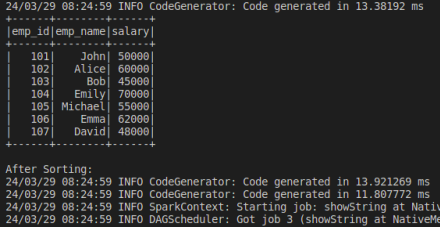
print("After Sorting:")

sorted\_df.show()

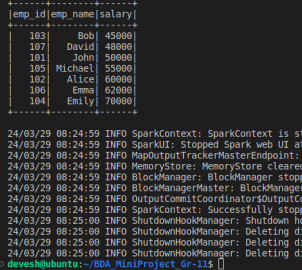
spark.stop()

Output :

Before Sorting :



After Sorting :



d)Searching a data Element :

from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Search") \

.getOrCreate()

data = [("Apple", "iPhone 13"), ("Samsung", "Galaxy S21"), ("Google", "Pixel 6"),

("Huawei", "Mate 40"), ("Xiaomi", "Mi 11"), ("OnePlus", "9 Pro")] df = spark.createDataFrame(data, ["brand", "model"])

search\_result = df.filter(df.brand == "Samsung").collect()

if search\_result:

print("Found:", search\_result[0])

else:

print("Not Found")

search\_result = df.filter(df.brand == "Sony").collect()

if search\_result:

print("Found:", search\_result[0])

else:

print("Not Found")

spark.stop()

Output :



e) Joins - Map side and Reduce side :

from pyspark import SparkContext

sc = SparkContext("local", "Joins")

left\_data = sc.parallelize([(101, "John"), (102, "Alice"), (103, "Bob"), (104, "Emily")])

right\_data = sc.parallelize([(101, 25), (102, 30), (105, 28)]) map\_join = left\_data.join(right\_data)

reduce\_join = left\_data.union(right\_data).reduceByKey(lambda x, y: (x, y))

print("Map Side Join:")

for result in map\_join.collect():

print(result)

print("\nReduce Side Join:")

for result in reduce\_join.collect():

print(result)

sc.stop()

Output :

