Module 3 Assignment | Analysis with MapReduce/Pig/Hive

Trang Tran

CPS, Northeastern University

ALY6110 | Data Management and Big Data

Professor Andrew Kinley

Aug 08, 2023

Set up the directory and subfolders in HDFS.

hadoop fs -mkdir lab

hadoop fs -mkdir lab/input

hadoop fs -mkdir lab/output

#transfer files into HDFS

hadoop fs -copyFromLocal orders.csv lab/input

hadoop fs -copyFromLocal inventory.csv lab/input

nano inventory.csv #then delete the header and save it back into the file

cat inventory.csv

Pig # launches the Pig shell

1. orders = LOAD 'lab/input/orders.csv' USING PigStorage(',') AS (table_id: chararray, order_id: chararray, product_id: int, date: chararray);

Load the 'orders.csv' file using PigStorage with comma as the delimiter, also assign aliases to the columns, specifying the data types for each column: table_id as a chararray, order_id as a chararray, product_id as an integer, and date as a chararray.

2. inventory = LOAD 'lab/input/inventory.csv' USING PigStorage(',') AS (table_id: chararray, product_id: int, name: chararray, category: chararray, inventory: int);

Load the 'inventory.csv' file using PigStorage with comma as the delimiter, also assign aliases to the columns, specifying the data types for each column: table_id as a chararray, product_id as an integer, name as a chararray, category as a chararray, and inventory as an integer.

3. grouped_orders = GROUP orders BY product_id;

Group the ‘orders’ table by the ‘product_id’ column, and store the result in the new table named ‘grouped_orders’.

4. orders_count = FOREACH grouped_orders GENERATE group AS product_id, COUNT(orders) AS order_count;

For each group in ‘grouped_orders’, calculate the count of orders for each ‘product_id, then generate a relation named ‘orders_count’ with columns product_id and order_count.

5. grouped_inventory = GROUP inventory BY product_id;

Group the ‘inventory’ relation by the ‘product_id’ column, and store the result in the new table named ‘grouped_inventory’.

6. inventory_sum = FOREACH grouped_inventory GENERATE group AS product_id, SUM(inventory.inventory) AS total_inventory;

For each group in ‘grouped_inventory’, calculate the sum of ‘inventory’ values for each product_id, then generate a relation ‘inventory_sum’ with columns product_id and total_inventory.

7. joined_data = JOIN orders_count BY product_id LEFT OUTER, inventory_sum BY product_id;

Perform a left join between orders_count and inventory_sum based on the ‘product_id’ column, then store a relation ‘joined_data’ with columns from both orders_count and inventory_sum.

8. joined_data = FOREACH joined_data GENERATE orders_count::product_id AS product_id, orders_count::order_count AS order_count, inventory_sum::total_inventory AS total_inventory, (order_count - total_inventory) AS difference;

For each row in ‘joined_data’, calculate the difference between the order_count and total_inventory columns, and generate a new column named ‘difference’.

9. insufficient_stock = FILTER joined_data BY difference > 0;

Filter the rows in ‘joined_data’ to retain only those where the difference column is greater than 0 (where order_count > total_inventory), indicating insufficient stock for orders.

10. result = FOREACH insufficient_stock GENERATE product_id, difference AS additional_items;

For each row in ‘insufficient_stock’, generate the product_id and additional_items columns, where additional_items is the value of the difference column, then store it in the ‘result’ table (shown below).

HIVE

hadoop fs -mkdir HV

hadoop fs -mkdir HV/input

#transfer files into the directory

hadoop fs -copyFromLocal orders.csv HV/input

hadoop fs -copyFromLocal inventory.csv HV/input

Hive

1. CREATE TABLE orders (table_id STRING, order_id STRING, product_id INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

Create a new blank table named ‘orders’ with three columns: table_id of type STRING, order_id of type STRING, and product_id of type INT. The data in this table will be delimited by commas, and it will be stored in text file format.

2. LOAD DATA INPATH 'hdfs:///user/hadoop/HV/input/orders.csv' OVERWRITE INTO TABLE orders;

Load data from the specified file ‘orders.csv’ from the HDFS HV/input directory into the ‘orders’ table.

# then we do the same thing with the ‘inventory’ table in lines 3 and 4 below

3. CREATE TABLE inventory (table_id STRING, product_id INT, name STRING, category STRING, inventory INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

4. LOAD DATA INPATH 'hdfs:///user/hadoop/HV/input/inventory.csv' OVERWRITE INTO TABLE inventory;

5. CREATE TABLE orders_count AS SELECT product_id, COUNT(order_id) AS order_count FROM orders GROUP BY product_id;

Create a new table named ‘orders_count’ by selecting the ‘product_id’ column and calculate the count of orders for each product_id and assign the result to the ‘order_count’column.

6. CREATE TABLE inventory_sum AS SELECT product_id, SUM(inventory) AS total_inventory FROM inventory GROUP BY product_id;

Create a new table named ‘inventory_sum’ by calculating the sum of inventory values for each ‘product_id’ and assign the result to the ‘total_inventory’ column.

7. CREATE TABLE joined_data AS SELECT o.product_id, o.order_count, i.total_inventory, (o.order_count - i.total_inventory) AS difference FROM orders_count o LEFT JOIN inventory_sum i ON o.product_id = i.product_id;

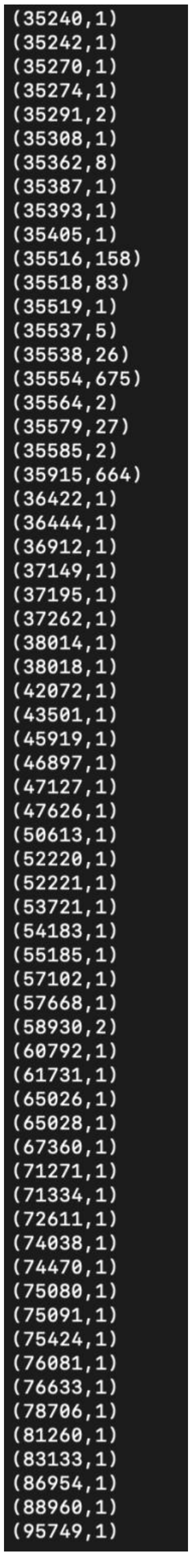
Create a new table named ‘joined_data’ by performing a LEFT JOIN between the ‘orders_count’ and ‘inventory_sum’ tables by the ‘product_id’ column.

8. CREATE TABLE insufficient_stock AS SELECT product_id, difference AS additional_items FROM joined_data WHERE difference > 0;

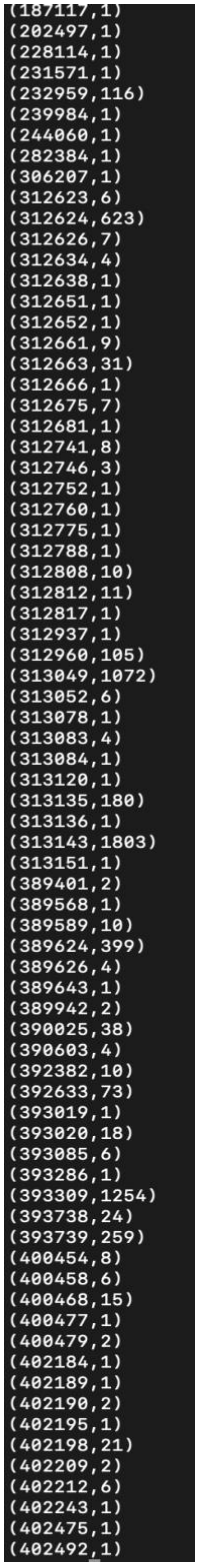
Create a new table named ‘insufficient_stock’ by filtering and selecting rows where the difference column is greater than 0 and rename the difference column as ‘additional_items’.

9. SELECT * FROM insufficient_stock;

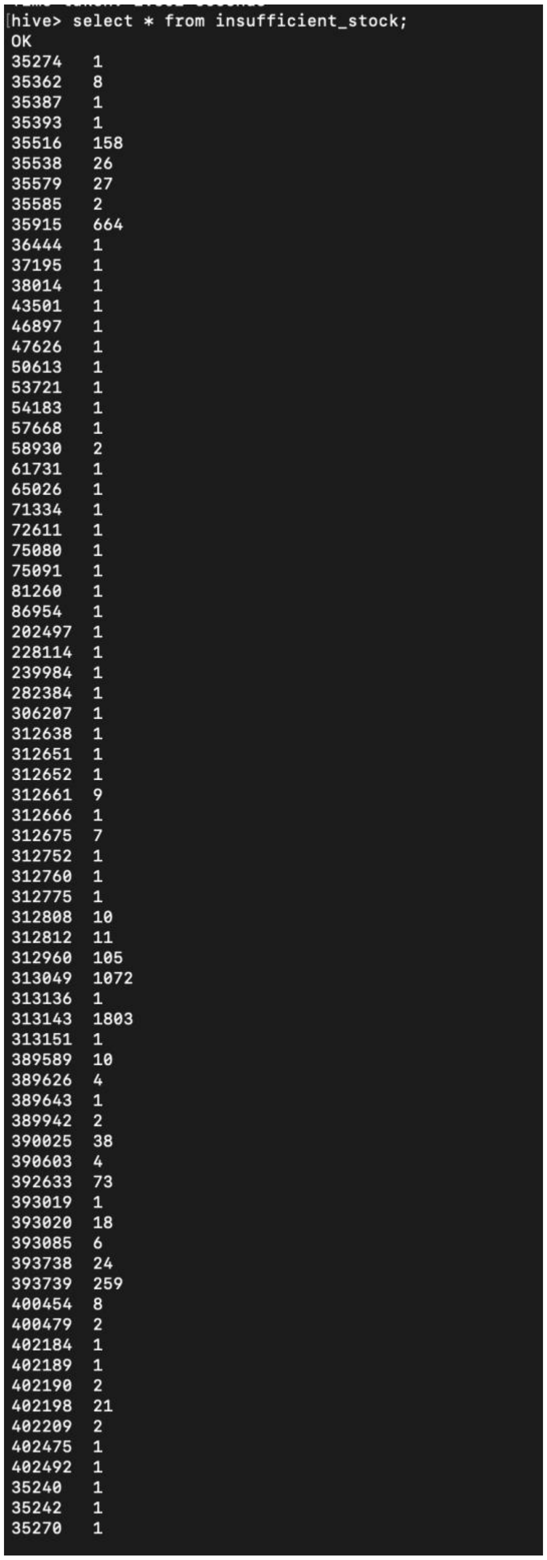
View all content in the 'insufficient_stock' table (shown below).



```
(35240,1)
(35242,1)
(35270,1)
(35274,1)
(35291,2)
(35308,1)
(35362,8)
(35387,1)
(35393,1)
(35405,1)
(35516,158)
(35518,83)
(35519,1)
(35537,5)
(35538,26)
(35554,675)
(35564,2)
(35579,27)
(35585,2)
(35915,664)
(36422,1)
(36444,1)
(36912,1)
(37149,1)
(37195,1)
(37262,1)
(38014,1)
(38018,1)
(42072,1)
(43501,1)
(45919,1)
(46897,1)
(47127,1)
(47626,1)
(50613,1)
(52220,1)
(52221,1)
(53721,1)
(54183,1)
(55185,1)
(57102,1)
(57668,1)
(58930,2)
(60792,1)
(61731,1)
(65026,1)
(65028,1)
(67360,1)
(71271,1)
(71334,1)
(72611,1)
(74038,1)
(74470,1)
(75080,1)
(75091,1)
(75424,1)
(76081,1)
(76633,1)
(78706,1)
(81260,1)
(83133,1)
(86954,1)
(88960,1)
(95749,1)
```



```
(187117,1)
(202497,1)
(228114,1)
(231571,1)
(232959,116)
(239984,1)
(244060,1)
(282384,1)
(306207,1)
(312623,6)
(312624,623)
(312626,7)
(312634,4)
(312638,1)
(312651,1)
(312652,1)
(312661,9)
(312663,31)
(312666,1)
(312675,7)
(312681,1)
(312741,8)
(312746,3)
(312752,1)
(312760,1)
(312775,1)
(312788,1)
(312808,10)
(312812,11)
(312817,1)
(312937,1)
(312960,105)
(313049,1072)
(313052,6)
(313078,1)
(313083,4)
(313084,1)
(313120,1)
(313135,180)
(313136,1)
(313143,1803)
(313151,1)
(389401,2)
(389568,1)
(389589,10)
(389624,399)
(389626,4)
(389643,1)
(389942,2)
(390025,38)
(390603,4)
(392382,10)
(392633,73)
(393019,1)
(393020,18)
(393085,6)
(393286,1)
(393309,1254)
(393738,24)
(393739,259)
(400454,8)
(400458,6)
(400468,15)
(400477,1)
(400479,2)
(402184,1)
(402189,1)
(402190,2)
(402195,1)
(402198,21)
(402209,2)
(402212,6)
(402243,1)
(402475,1)
(402492,1)
```



```
[hive> select * from insufficient_stock;
OK
35274   1
35362   8
35387   1
35393   1
35516   158
35538   26
35579   27
35585   2
35915   664
36444   1
37195   1
38014   1
43501   1
46897   1
47626   1
50613   1
53721   1
54183   1
57668   1
58930   2
61731   1
65026   1
71334   1
72611   1
75080   1
75091   1
81260   1
86954   1
202497  1
228114  1
239984  1
282384  1
306207  1
312638  1
312651  1
312652  1
312661  9
312666  1
312675  7
312752  1
312760  1
312775  1
312808  10
312812  11
312960  105
313049  1072
313136  1
313143  1803
313151  1
389589  10
389626  4
389643  1
389942  2
390025  38
390603  4
392633  73
393019  1
393020  18
393085  6
393738  24
393739  259
400454  8
400479  2
402184  1
402189  1
402190  2
402198  21
402209  2
402475  1
402492  1
35240   1
35242   1
35270   1
```

MapReduce

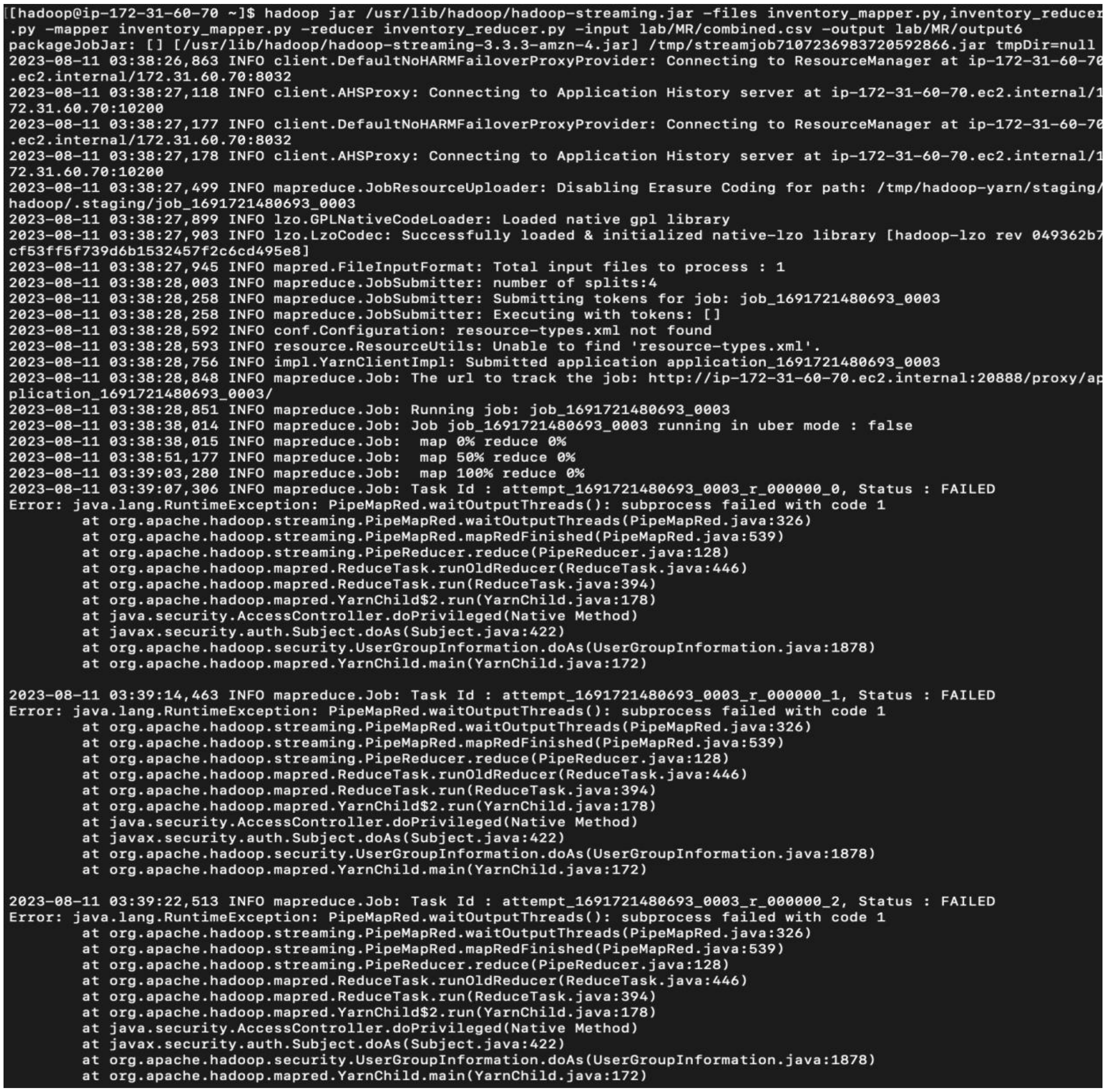
```
hadoop fs -mkdir lab/MR

hadoop fs -copyFromLocal combined.csv lab/MR

hadoop fs -copyFromLocal inventory_mapper.py lab/MR

hadoop fs -copyFromLocal inventory_reducer.py lab/MR

chmod +x inventory_mapper.py

chmod +x inventory_reducer.py

hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -files

inventory_mapper.py,inventory_reducer.py -mapper inventory_mapper.py -reducer

inventory_reducer.py -input lab/MR/combined.csv -output lab/MR/output
```

Mapper

```
# import the module for reading and writing data

import sys

# input is read by STDIN (standard input) and do the following for each input line

for line in sys.stdin:

    # remove leading and trailing whitespace

    line = line.strip()

    # split the line by comma separator, a list is produced

    line = line.split(",")

    # assign values of the list to the variable names

    table_name = line[0]

    product_id = line[1]

    inventory_value = line[-1]  # Assuming inventory value is the last column
    if table_name == 'order_id':
        print('%s\t%s' % (product_id, 'order'))
    elif table_name == 'inventory_id':
        print('%s\t%s' % (product_id, inventory_value))
```

Reducer

```
# import the module for reading and writing data
import sys
# Initialize variables: lastKey as None and maxValue as 0
(lastKey, maxValue) = (None, 0)
# Read input from STDIN (standard input) line by line
for line in sys.stdin:
    # Remove leading and trailing whitespace
    line = line.strip()
    # Split the line into key and value based on tab separation
    (key, value) = line.split('\t', 1)
    # Convert the value to an integer
    value = int(value)
    # Check if lastKey is defined and not equal to the current key
    if lastKey and lastKey != key:
        # Output the product_id and the difference between inventory and orders
        print('%s\t%s' % (lastKey, maxValue - value))
        (lastKey, maxValue) = (key, value)
    else:
        # Update maxValue if the current value is larger
        maxValue = max(maxValue, value)
        lastKey = key
# Output the result for the last product_id
if lastKey:
    print('%s\t%s' % (lastKey, maxValue))
```



```
[hadoop@ip-172-31-60-70 ~]$ hadoop jar /usr/lib/hadoop/hadoop-streaming.jar -files inventory_mapper.py,inventory_reducer
.py -mapper inventory_mapper.py -reducer inventory_reducer.py -input lab/MR/combined.csv -output lab/MR/output6
packageJobJar: [] [/usr/lib/hadoop/hadoop-streaming-3.3.3-amzn-4.jar] /tmp/streamjob7107236983720592866.jar tmpDir=null
2023-08-11 03:38:26,863 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at ip-172-31-60-70
.ec2.internal/172.31.60.70:8032
2023-08-11 03:38:27,118 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-60-70.ec2.internal/1
72.31.60.70:10200
2023-08-11 03:38:27,177 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at ip-172-31-60-70
.ec2.internal/172.31.60.70:8032
2023-08-11 03:38:27,178 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-60-70.ec2.internal/1
72.31.60.70:10200
2023-08-11 03:38:27,499 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/
hadoop/.staging/job_1691721480693_0003
2023-08-11 03:38:27,899 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
2023-08-11 03:38:27,903 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library [hadoop-lzo rev 049362b7
cf53ff5f739d6b1532457f2c6cd495e8]
2023-08-11 03:38:27,945 INFO mapred.FileInputFormat: Total input files to process : 1
2023-08-11 03:38:28,003 INFO mapreduce.JobSubmitter: number of splits:4
2023-08-11 03:38:28,258 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1691721480693_0003
2023-08-11 03:38:28,258 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-08-11 03:38:28,592 INFO conf.Configuration: resource-types.xml not found
2023-08-11 03:38:28,593 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-08-11 03:38:28,756 INFO impl.YarnClientImpl: Submitted application application_1691721480693_0003
2023-08-11 03:38:28,848 INFO mapreduce.Job: The url to track the job: http://ip-172-31-60-70.ec2.internal:20888/proxy/ap
plication_1691721480693_0003/
2023-08-11 03:38:28,851 INFO mapreduce.Job: Running job: job_1691721480693_0003
2023-08-11 03:38:38,014 INFO mapreduce.Job: Job job_1691721480693_0003 running in uber mode : false
2023-08-11 03:38:38,015 INFO mapreduce.Job:  map 0% reduce 0%
2023-08-11 03:38:51,177 INFO mapreduce.Job:  map 50% reduce 0%
2023-08-11 03:39:03,280 INFO mapreduce.Job:  map 100% reduce 0%
2023-08-11 03:39:07,306 INFO mapreduce.Job: Task Id : attempt_1691721480693_0003_r_000000_0, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.reduce(PipeReducer.java:128)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:446)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:394)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:178)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1878)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:172)

2023-08-11 03:39:14,463 INFO mapreduce.Job: Task Id : attempt_1691721480693_0003_r_000000_1, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.reduce(PipeReducer.java:128)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:446)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:394)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:178)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1878)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:172)

2023-08-11 03:39:22,513 INFO mapreduce.Job: Task Id : attempt_1691721480693_0003_r_000000_2, Status : FAILED
Error: java.lang.RuntimeException: PipeMapRed.waitOutputThreads(): subprocess failed with code 1
        at org.apache.hadoop.streaming.PipeMapRed.waitOutputThreads(PipeMapRed.java:326)
        at org.apache.hadoop.streaming.PipeMapRed.mapRedFinished(PipeMapRed.java:539)
        at org.apache.hadoop.streaming.PipeReducer.reduce(PipeReducer.java:128)
        at org.apache.hadoop.mapred.ReduceTask.runOldReducer(ReduceTask.java:446)
        at org.apache.hadoop.mapred.ReduceTask.run(ReduceTask.java:394)
        at org.apache.hadoop.mapred.YarnChild$2.run(YarnChild.java:178)
        at java.security.AccessController.doPrivileged(Native Method)
        at javax.security.auth.Subject.doAs(Subject.java:422)
        at org.apache.hadoop.security.UserGroupInformation.doAs(UserGroupInformation.java:1878)
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:172)
```

****I followed all the steps above and used 2 Python files as stated for 'Mapper' and 'Reducer', but I still couldn't go through the final streaming result. I tried editing the script several times and run it again and again but still cannot figure out the errors in this case.*