**Q1. MapReduce Problem Statement Here, we have chosen the stock market dataset on which we have performed map-reduce operations. Following is the structure of the data. Kindly find the solutions to the questions below.**

**1.Find all time High price for each stock**

**import** java.io.\*;

**import** org.apache.hadoop.io.Text;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.DoubleWritable;

**import** org.apache.hadoop.mapreduce.Job;

**import** org.apache.hadoop.mapreduce.Mapper;

**import** org.apache.hadoop.mapreduce.Reducer;

**import** org.apache.hadoop.conf.\*;

**import** org.apache.hadoop.fs.\*;

**import** org.apache.hadoop.mapreduce.lib.input.\*;

**import** org.apache.hadoop.mapreduce.lib.output.\*;

**public** **class** HighPrice{

**public** **static** **class** MapClass **extends** Mapper<LongWritable,Text,Text,DoubleWritable>

{

**private** Text stock\_id = **new** Text();

**private** DoubleWritable High = **new** DoubleWritable();

**public** **void** map(LongWritable key,Text value , Context context)

{

**try** {

String str[] = value.toString().split(",");

**double** high = Double.*parseDouble*(str[4]);

stock\_id.set(str[1]);

High.set(high);

context.write(stock\_id , High);

}

**catch**(Exception e)

{

System.***out***.println(e.getMessage());

}

}

}

**public** **static** **class** ReduceClass **extends** Reducer<Text,DoubleWritable,Text,DoubleWritable>

{

**private** DoubleWritable result = **new** DoubleWritable();

**public** **void** reduce(Text key,Iterable<DoubleWritable> values , Context context) **throws** IOException,InterruptedException

{

**double** max=0;

**double** temp=0;

**for**(DoubleWritable value : values)

{

temp=value.get();

**if**(temp>max) {

max = temp;

}

}

result.set(max);

context.write(key,result);

}

}

**public** **static** **void** main(String args[]) **throws** Exception{

Configuration conf = **new** Configuration();

Job job = Job.*getInstance*(conf,"Highest Price of each stock");

job.setJarByClass(HighPrice.**class**);

job.setMapperClass(MapClass.**class**);

job.setReducerClass(ReduceClass.**class**);

job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(DoubleWritable.**class**);

FileInputFormat.*addInputPath*(job , **new** Path(args[0]));

FileOutputFormat.*setOutputPath*(job , **new** Path(args[1]));

System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[bigdatamind43816@ip-10-1-1-204 ~]$ jar tvf myjar.jar

[bigdatamind43816@ip-10-1-1-204 ~]$ hadoop jar myjar.jar HighPrice cdac/NYSE.csv cdac/examoutput

AA 94.62

AAI 57.88

AAN 35.21

AAP 83.65

AAR 25.25

AAV 24.78

AB 94.94

ABA 27.94

ABB 33.39

ABC 84.35

ABD 28.58

ABG 30.06

ABK 96.1

ABM 41.63

ABR 34.45

ABT 93.37

ABV 107.5

ABVT 100.0

ABX 54.74

ACC 37.0

ACE 104.0

ACF 64.9

ACG 12.63

ACH 111.6

ACI 112.89

ACL 178.56

ACM 38.25

ACN 44.03

ACO 42.7

ACS 109.55

ACV 65.32

ADC 37.7

ADI 185.5

ADM 48.95

ADP 84.31

ADS 80.79

ADX 40.56

ADY 44.0

AEA 23.94

AEB 26.5

AEC 17.6

AED 26.12

AEE 56.77

AEF 27.0

AEG 148.32

AEH 26.64

AEL 14.6

AEM 83.45

AEO 88.13

AEP 53.31

AER 32.82

AES 92.5

AET 154.67

AEV 26.78

AF 63.09

AFB 17.03

AFC 25.15

AFE 26.7

AFF 25.15

AFG 54.65

AFL 74.94

AFN 11.99

AGC 20.2

AGCO 71.95

AGD 25.5

AGL 44.67

AGM 80.0

AGN 125.0

AGO 31.99

AGP 80.89

AGU 113.88

AHC 16.35

AHD 47.12

AHL 30.8

AHS 37.4

AHT 13.48

AI 28.7

AIB 125.0

AIG 157.19

AIN 43.62

AIQ 15.4

AIR 46.58

AIT 59.0

AIV 65.79

AIZ 71.31

AJG 68.5

AKF 26.42

AKP 17.45

AKR 29.0

AKS 73.07

AKT 26.25

ALB 86.52

ALC 27.86

ALD 33.35

ALE 51.7

ALEX 44.52

ALF 26.75

ALG 29.23

ALJ 47.1

ALK 62.56

ALL 100.25

ALM 27.79

ALQ 28.5

ALU 86.25

ALV 65.09

ALX 471.0

ALY 28.1

ALZ 26.25

AM 62.88

AMB 66.86

AMD 97.0

AME 53.12

AMG 136.51

AMN 138.65

AMP 69.25

AMR 69.01

AMT 55.5

AMX 69.15

AN 53.93

ANF 101.5

ANH 16.65

ANN 53.06

ANR 119.3

ANW 48.63

AOB 14.48

AOD 21.85

AOI 23.38

AOL 27.0

AON 75.56

AOS 58.06

AP 54.46

APA 149.23

APB 36.14

APC 113.95

APD 106.06

APF 24.53

APH 121.06

APL 56.88

APU 42.94

APX 12.38

ARB 55.63

ARD 71.08

ARE 116.5

ARG 65.45

ARI 19.2

ARJ 48.02

ARK 8.29

ARL 22.25

ARM 32.5

ARO 47.82

ARP 39.0

ART 28.1

ARW 64.12

ASA 92.6

ASF 89.12

ASG 12.56

ASH 76.25

ASI 24.21

ASP 16.87

ASR 63.54

ASX 7.49

ATE 58.02

ATI 119.7

ATK 120.9

ATO 33.47

ATR 76.98

ATT 27.14

ATU 70.17

ATV 32.33

ATW 126.92

AU 62.2

AUO 28.5

AUY 19.93

AV 15.05

AVA 67.76

AVB 149.94

AVD 51.0

AVF 27.0

AVK 29.75

AVP 90.45

AVT 81.12

AVX 100.0

AVY 78.5

AWC 32.85

AWF 15.46

AWH 53.48

AWI 57.48

AWK 23.77

AWP 20.55

AWR 48.0

AXA 80.94

AXE 88.4

AXL 42.1

AXP 169.5

AXR 149.99

AXS 43.35

AYE 65.48

AYI 66.89

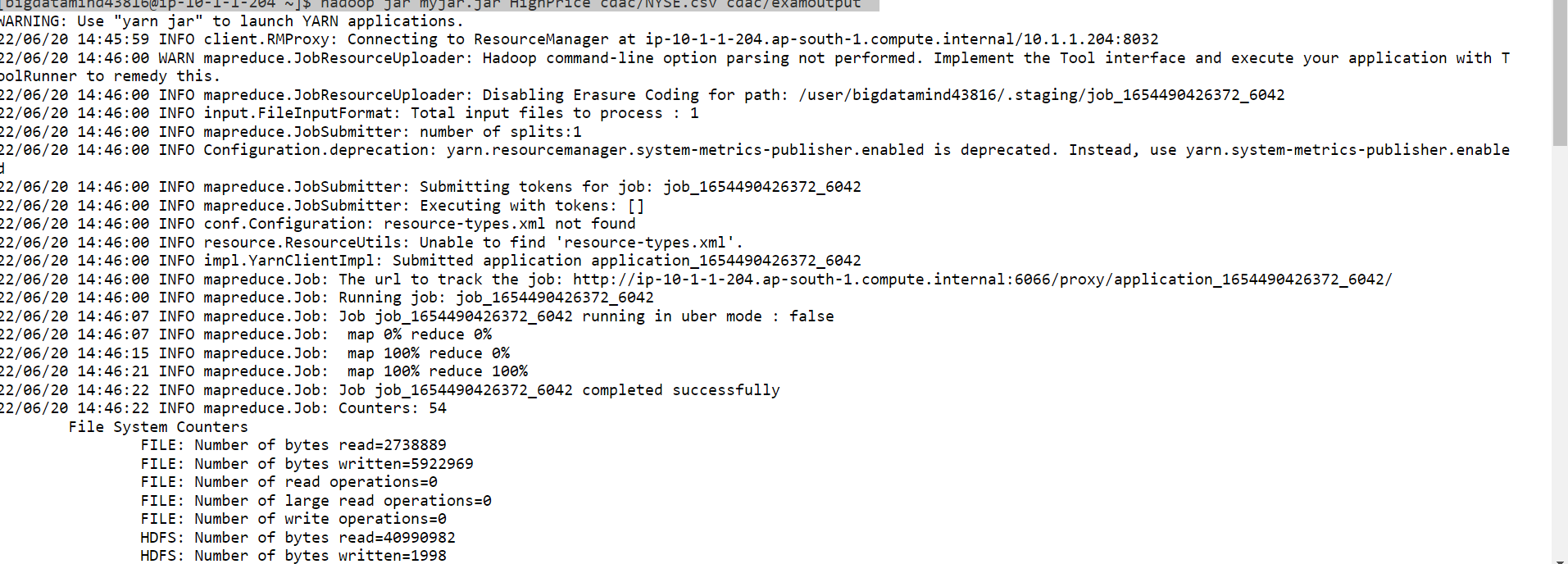
AYN 15.42

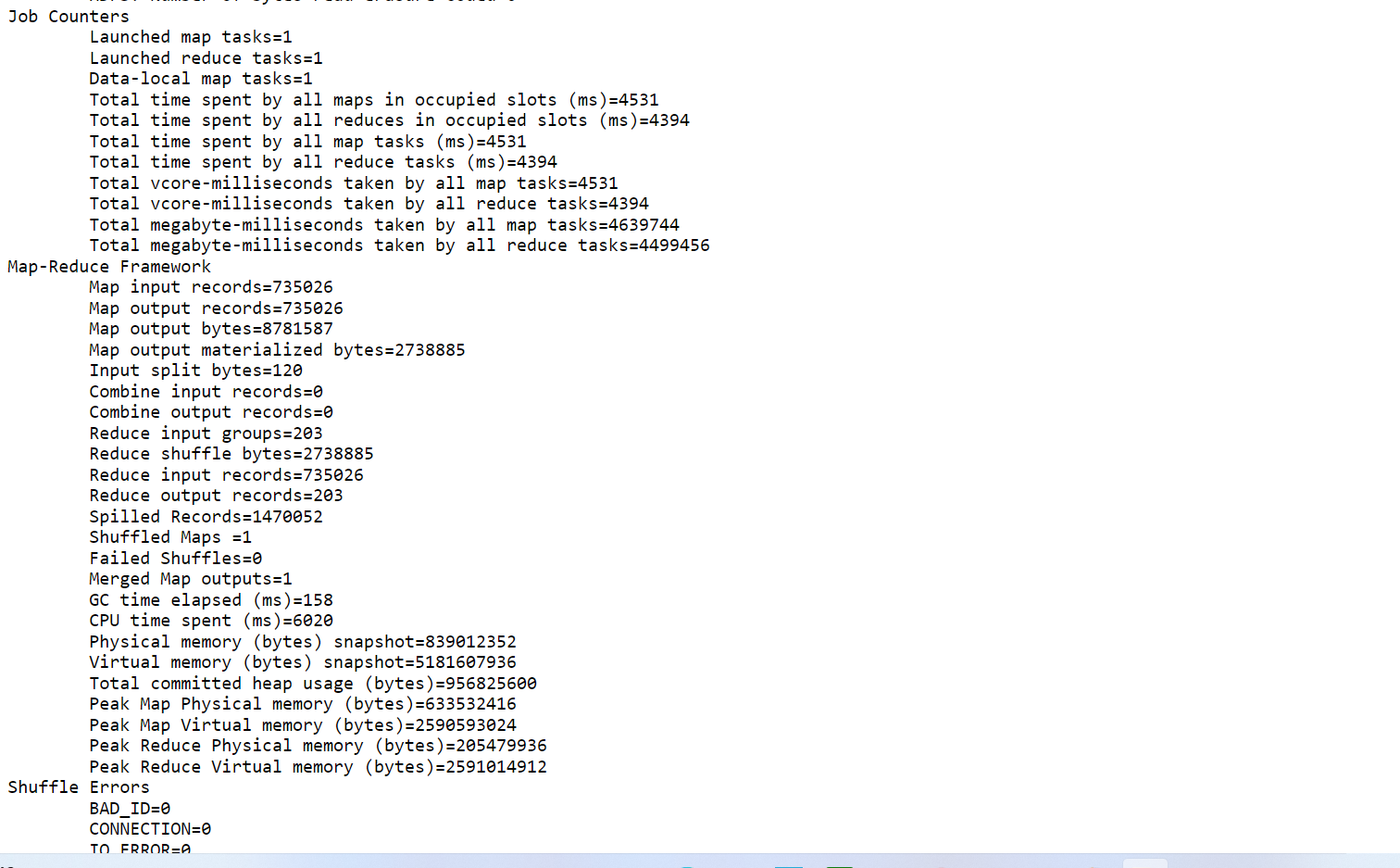
AYR 41.31

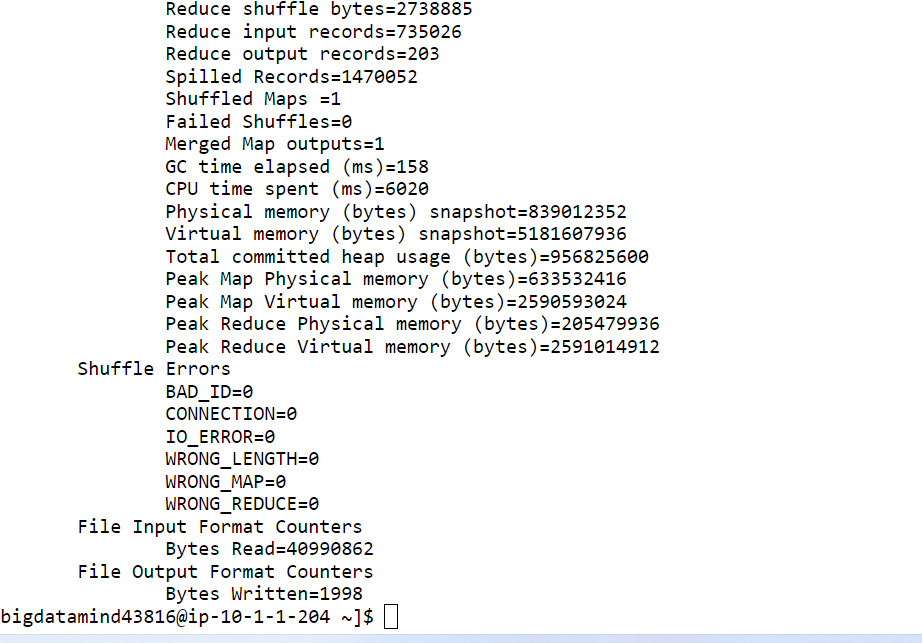
AZN 145.41

AZO 169.99

AZZ 59.2

****

****

****

**Q2.**

**Hive**

**Please find the customer data set.**

**cust id firstname lastname age profession**

**1)Write a program to find the count of customers for each profession.**

hive> **use training050;**

OK

Time taken: 1.743 seconds

hive> **create table cust1(cust\_id bigint,firstname string , lastname string , age int , profession string)**

**> row format delimited**

**> fields terminated by ','**

**> stored as textfile;**

OK

Time taken: 0.487 seconds

hive> **load data local inpath 'custs.txt' overwrite into table cust1;**

Loading data to table training44.cust1

OK

Time taken: 1.421 seconds

hive> **select \* from custs1 limit 10;**

OK

4000001 Kristina Chung 55 Pilot

4000002 Paige Chen 74 Teacher

4000003 Sherri Melton 34 Firefighter

4000004 Gretchen Hill 66 Computer hardware engineer

4000005 Karen Puckett 74 Lawyer

4000006 Patrick Song 42 Veterinarian

4000007 Elsie Hamilton 43 Pilot

4000008 Hazel Bender 63 Carpenter

4000009 Malcolm Wagner 39 Artist

4000010 Dolores McLaughlin 60 Writer

Time taken: 0.377 seconds, Fetched: 10 row(s)

hive> **select profession,count(cust\_id) as no\_of\_customers from cust1 group by profession;**

Actornta202 199

Agricultural and food scientist 195

Architect 203

Artist 175

Athlete 196

Automotive mechanic 193

Carpenter 181

Chemist 209

Childcare worker 207

Civil engineer 193

Coach 201

Computer hardware engineer 204

Computer software engineer 216

Computer support specialist 222

Dancer 185

Designer 205

Doctor 197

Economist 189

Electrical engineer 192

Electrician 194

Engineering technician 204

Environmental scientist 176

Farmer 201

Financial analyst 198

Firefighter 217

Human resources assistant 212

Judge 196

Lawyer 212

Librarian 218

Loan officer 221

Musician 205

Nurse 192

Pharmacist 213

Photographer 222

Physicist 201

Pilot 211

Police officer 210

Politician 228

Psychologist 194

Real estate agent 191

Recreation and fitness worker 210

Reporter 200

Secretary 200

Social Worker 1

Social worker 212

Statistician 196

Teacher 204

Therapist 187

Veterinarian 208

Writer 101

Time taken: 29.499 seconds, Fetched: 51 row(s)

**Please find the sales data set.**

txn id

txn date

cust id

amount

category

product

city

state

spendby

hive> **create table txnsales(txn\_id bigint , txn\_date string , cust\_id bigint , amount double,category string,product string,city string ,state string ,spendby string)**

**> row format delimited**

**> fields terminated by ','**

**> stored as textfile;**

OK

Time taken: 0.095 seconds

hive> **load data local inpath 'txns1.txt' into table txnsales;**

Loading data to table training44.txnsales

OK

Time taken: 0.701 seconds

hive> **select \* from txnsales limit 10;**

OK

0 06-26-2011 4007024 40.33 Exercise & Fitness Cardio Machine Accessories Clarksville Tennessee credit

1 05-26-2011 4006742 198.44 Exercise & Fitness Weightlifting Gloves Long Beach California credit

2 06-01-2011 4009775 5.58 Exercise & Fitness Weightlifting Machine Accessories Anaheim California credit

3 06-05-2011 4002199 198.19 Gymnastics Gymnastics Rings Milwaukee Wisconsin credit

4 12-17-2011 4002613 98.81 Team Sports Field Hockey Nashville Tennessee credit

5 02-14-2011 4007591 193.63 Outdoor Recreation Camping & Backpacking & Hiking Chicago Illinois credit

6 10-28-2011 4002190 27.89 Puzzles Jigsaw Puzzles Charleston South Carolina credit

7 07-14-2011 4002964 96.01 Outdoor Play Equipment Sandboxes Columbus Ohio credit

8 01-17-2011 4007361 10.44 Winter Sports Snowmobiling Des Moines Iowa credit

9 05-17-2011 4004798 152.46 Jumping Bungee Jumping St. Petersburg Florida credit

10 05-29-2011 4004646 180.28 Outdoor Recreation Archery Reno Nevada credit

**2) Write a program to find the top 10 products sales wise**

hive> **select product,sum(amount) as max from txnsales group by product order by max desc limit 10;**

Yoga & Pilates 47804.93999999993

Swing Sets 47204.13999999999

Lawn Games 46828.44

Golf 46577.67999999999

Cardio Machine Accessories 46485.540000000045

Exercise Balls 45143.84

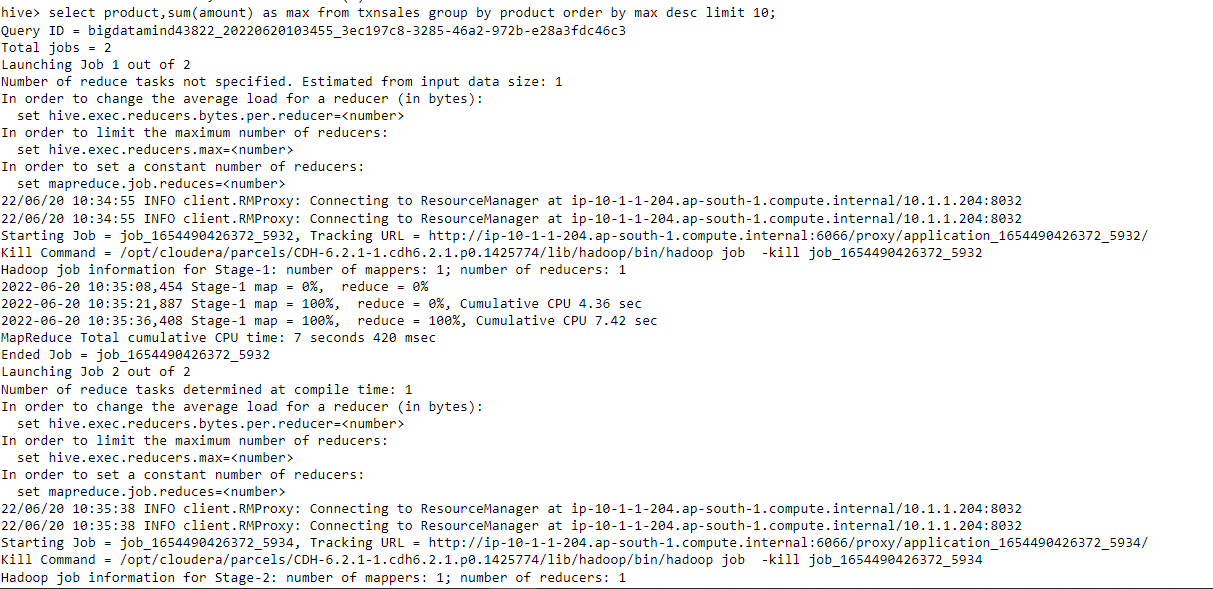
Weightlifting Belts 45111.67999999996

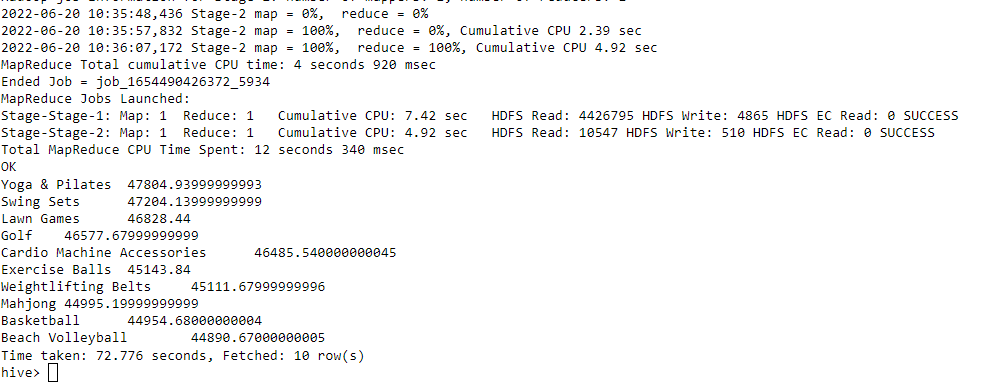
Mahjong 44995.19999999999

Basketball 44954.68000000004

Beach Volleyball 44890.67000000005

Time taken: 72.776 seconds, Fetched: 10 row(s)





**3) Write a program to create partiioned table on category**

hive>**set hive.exec.dynamic.partition.mode=nonstrict;**

hive>**set hive.exec.dynamic.partition=true;**

hive> **create table txnsales1(txn\_id bigint,txn\_date string,cust\_id bigint,amount double,product string,city string ,state string ,spendby string)**

**> partitioned by (category string)**

**> row format delimited**

**> fields terminated by ','**

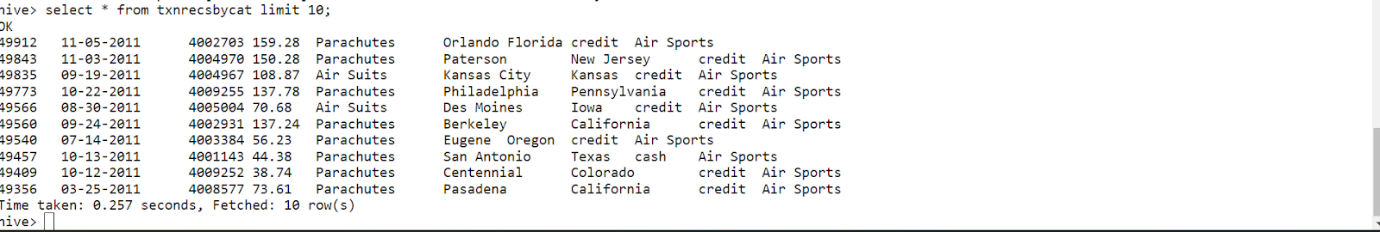
**> stored as textfile;**

OK

Time taken: 0.088 seconds

hive> **insert overwrite table txnsales1 partition(category) select t.txn\_id,t.txn\_date,t.cust\_id,t.amount,t.product,t.city,t.state,t.spendby,t.category from txnsales t**

**distribute by category;**



**QUESTION 3**

**PySpark**

**Please find the AIRLINES data set**

Year

Quarter

Average

revenue per seat

Total number of booked seats

>>> rdd=sc.textFile("/user/bigdatamind43816/airlines.csv")

>>> rdd.count()

85

>>> rdd1 = rdd.map(lambda a : a.encode("ascii","ignore"))

>>> header = rdd1.first()

>>> rdd2 = rdd1.filter(lambda a : a!=header)

>>> rdd2.count()

84

>>> array = rdd2.map(lambda a : a.split(","))

>>> for i in array.take(4):

... print(i)

...

['1995', '1', '296.9', '46561']

['1995', '2', '296.8', '37443']

['1995', '3', '287.51', '34128']

['1995', '4', '287.78', '30388']

**1) What was the highest number of people travelled in which year?**

>>> key = array.map(lambda a : (a[0],int(a[3])))

>>> total = key.reduceByKey(lambda a,b : a+b)

>>> total1 = total.sortBy(lambda a: -a[1])

>>> total1.first()

**('2007', 176299)**

>>> for i in total1.take(5):

... print(i)

...

('2007', 176299)

('2013', 173676)

('2001', 173598)

('1996', 167223)

('2008', 166897)

**2) Identifying the highest revenue generation for which year**

>>> array = rdd2.map(lambda a : a.split(","))

>>> key = array.map(lambda a: (a[0],float(a[2])\*int(a[3])))

>>> total = key.reduceByKey(lambda a,b: a+b)

>>> total1 = total.sortBy(lambda a: -a[1])

>>> total1.first()

**('2013', 66363208.71)**

>>> for i in total1.take(5):

... print(i)

...

**('2013', 66363208.71)**

('2014', 62624175.85000001)

('2015', 62378990.57)

('2012', 62199127.28)

('2008', 57653170.760000005)

**3) Identifying the highest revenue generation for which year and quarter (Common group)**

>>> array = rdd2.map(lambda a : a.split(","))

>>> key = array.map(lambda a: (a[0]+" "+a[1],float(a[2])\*int(a[3])))

>>> total = key.reduceByKey(lambda a,b: a+b)

>>> total1 = total.sortBy(lambda a: -a[1])

>>> total1.first()

**('2014 4', 18819408.48)**

>>> for i in total1.take(5):

... print(i)

...

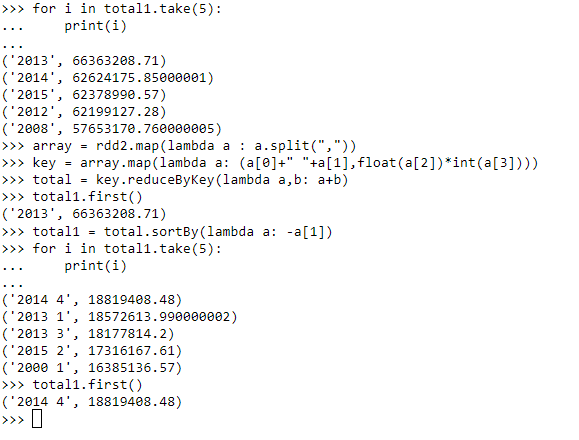
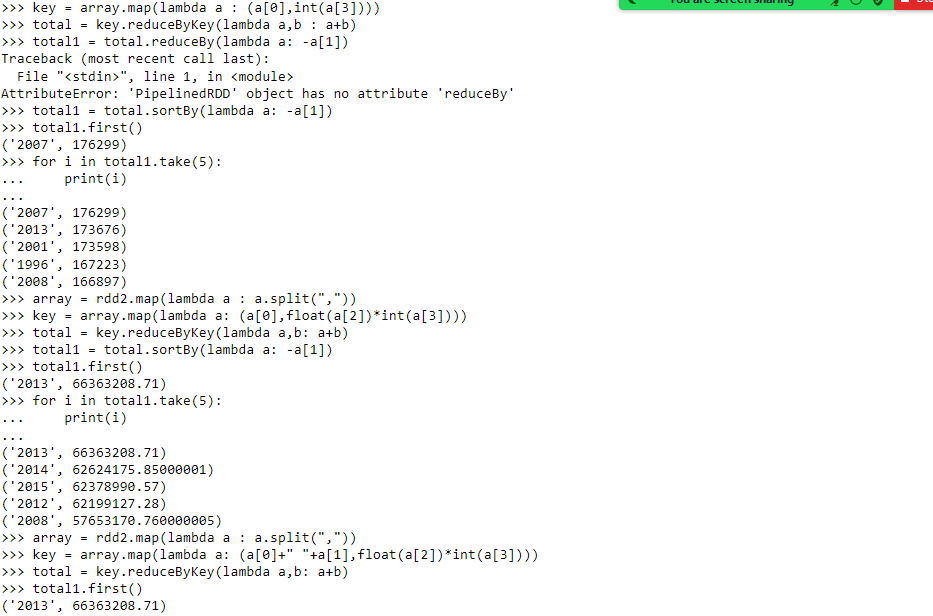
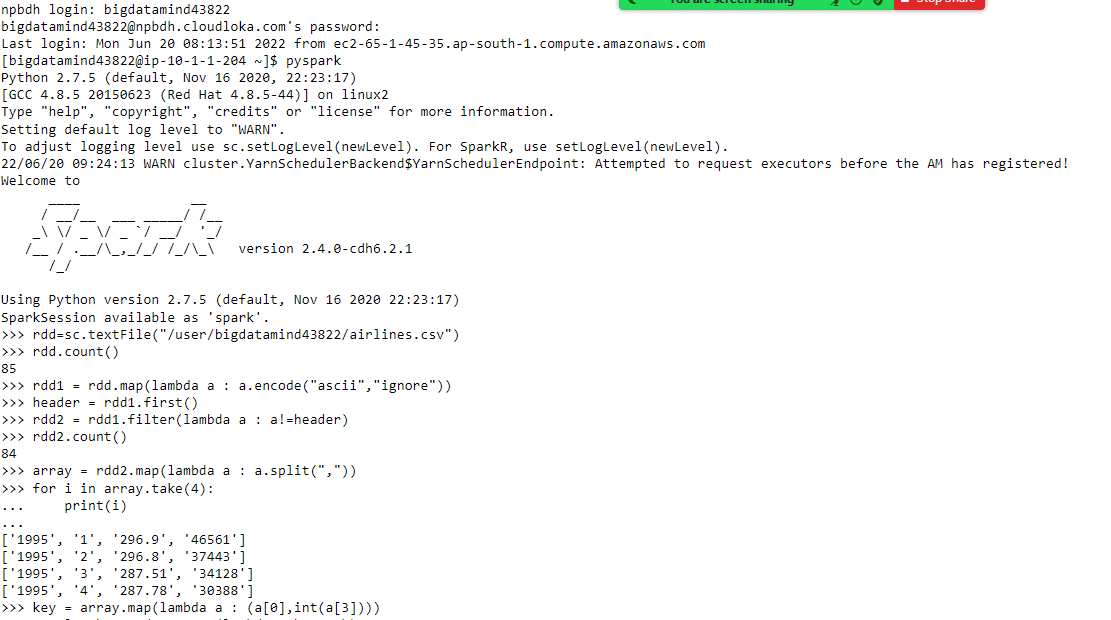
**('2014 4', 18819408.48)**

('2013 1', 18572613.990000002)

('2013 3', 18177814.2)

('2015 2', 17316167.61)

('2000 1', 16385136.57)

****