**Q1.**

**MapReduce**

**: 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);

}

}

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

**hadoop jar myjar1.jar HighPrice exam/NYSE.csv out10**

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

**Hive**

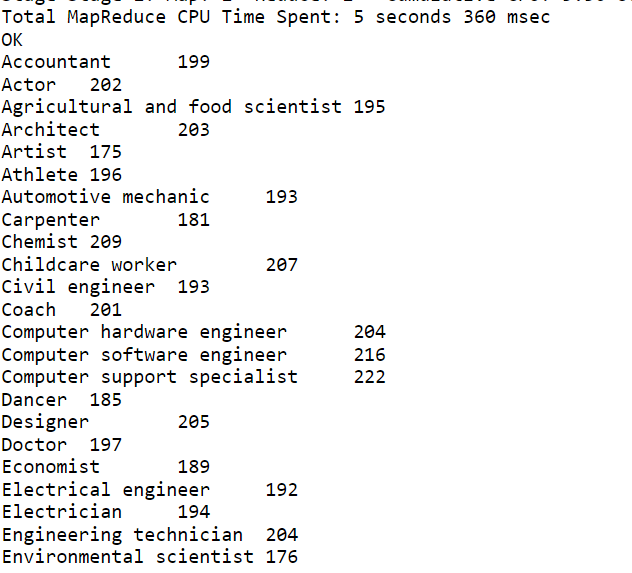
1. **Write a program to find the count of customers for each profession.**

create table custs1(cust\_id bigint, firstname string, lastname string, age int, profession string) row format delimited fields terminated by "," stored as textfile;

load data local inpath 'custs\_1.txt' overwrite into table custs1;

load data local inpath 'custs\_1.txt' overwrite into table custs1;

**>>select profession, count(cust\_id) from custs1 group by profession;**



1. **Write a program to find the top 10 products sales wise**

create table txn15

(txn\_id bigint,

txn\_date String,

cust\_id bigint,

amount float,

category String

product String,

city String,

state String,

spendby String)

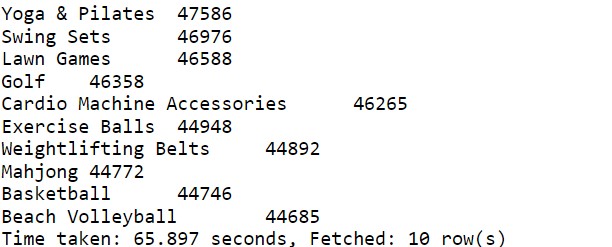
row format delimited

fields terminated by ','

stored as textfile;

load data local inpath 'txns\_1.txt' overwrite into table txn;

**>>select product, sum(amount) as sales from txn15 group by product order by sales desc limit 10;**

****

1. **Write a program to create partiioned table on category**

**create table txnPartition**

**(txn\_id bigint,**

**txn\_date String,**

**cust\_id bigint,**

**amount float,**

**product String,**

**city String,**

**state String,**

**spendby String)**

**partitioned by (category string)**

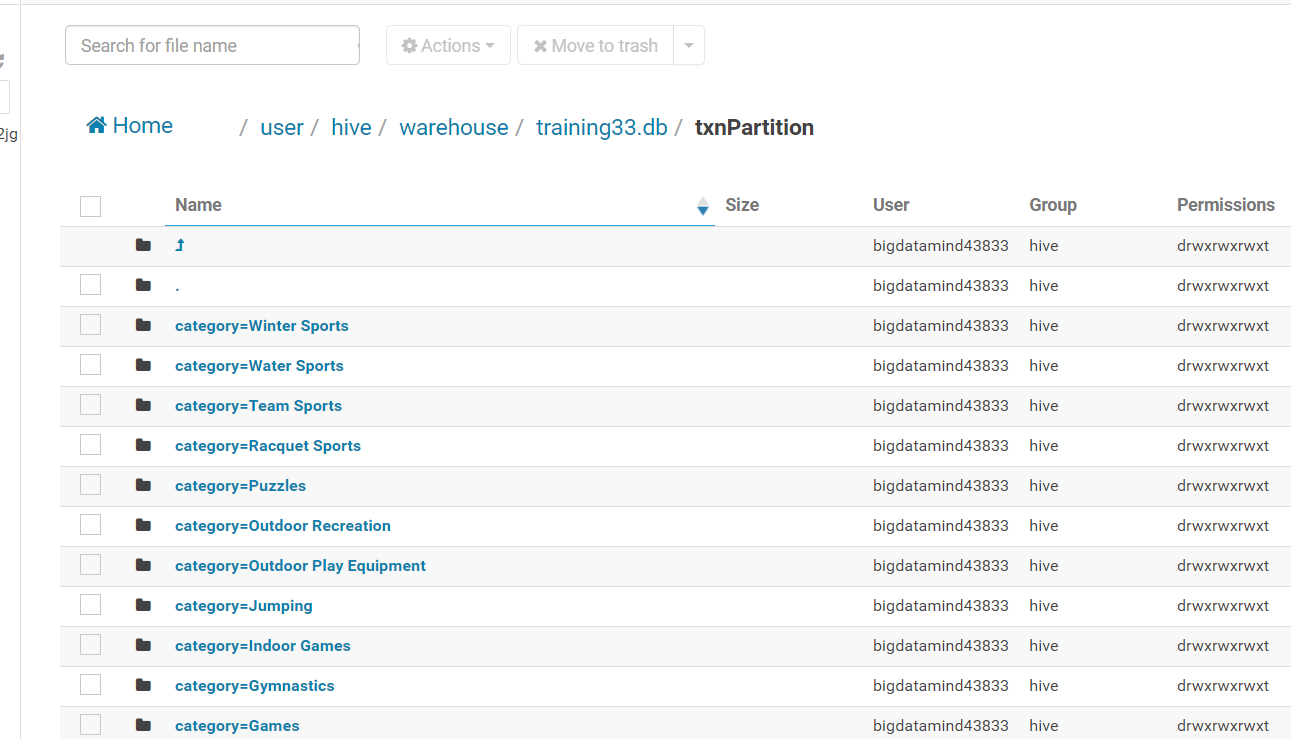
**row format delimited**

**fields terminated by ',’**

**stored as textfile;**

**insert overwrite table txnPartition 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 txn t**

**distribute by category;**



**QUESTION 3---Pyspark**

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

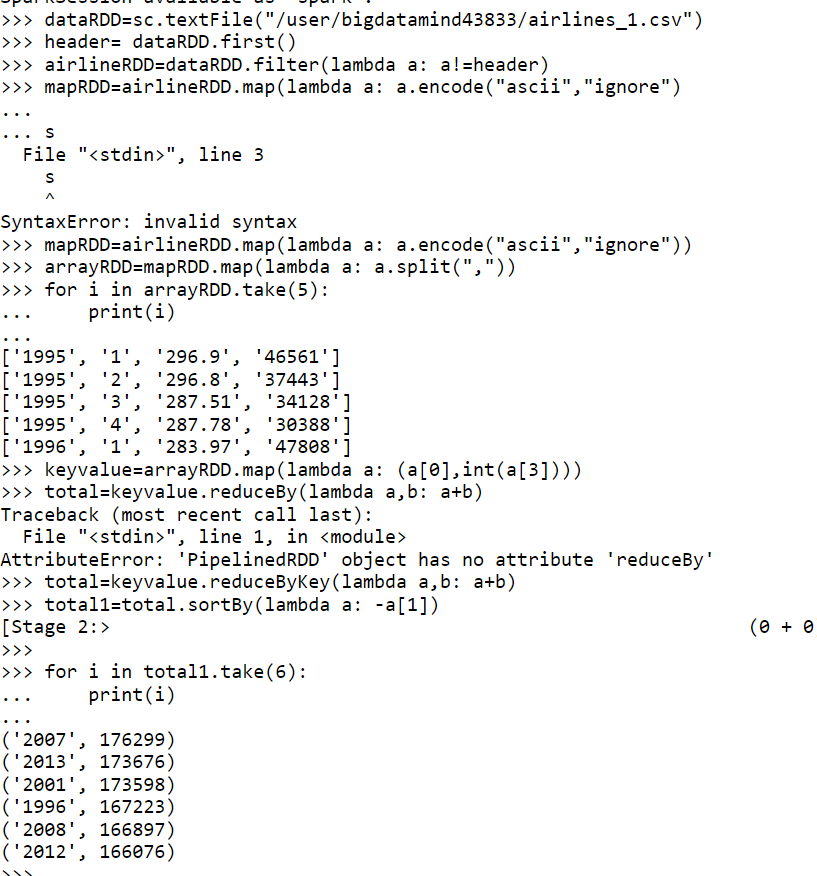
>>> key = array.map(lambda a : (a[0],int(a[3])))

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

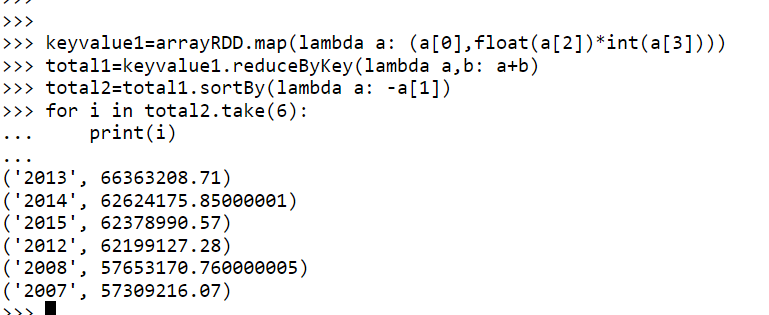
>>> total1 = total.sortBy(lambda a: -a[1])

>>> total1.first()

**('2007', 176299)**

****

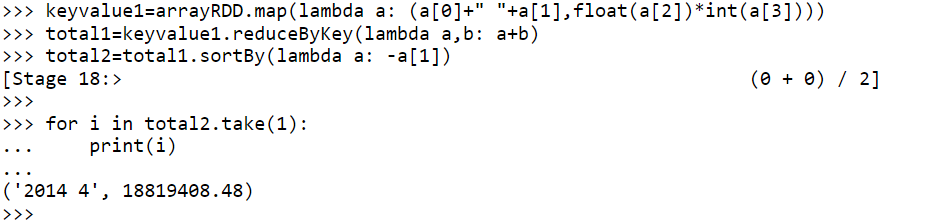
**2]Identifying the highest revenue generation for which year**

****total1.first()

**('2013', 66363208.71)**

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

**group)**

****

>>> total1.first()

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