Q1 MAP REduce

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 AllTimeHigh {

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(new Text(str[1]),new LongWritable(vol));

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 maxValue=0;

double temp\_val=0;

for (DoubleWritable value : values) {

temp\_val = value.get();

if (temp\_val > maxValue) {

maxValue = temp\_val;

}

}

result.set(maxValue);

context.write(key, result);

//context.write(key, new LongWritable(sum));

}

}

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

Configuration conf = new Configuration();

//conf.set("name", "value")

//conf.set("mapreduce.input.fileinputformat.split.minsize", "134217728");

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

job.setJarByClass(AllTimeHigh.class);

job.setMapperClass(MapClass.class);

//job.setCombinerClass(ReduceClass.class);

job.setReducerClass(ReduceClass.class);

job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(DoubleWritable.class);

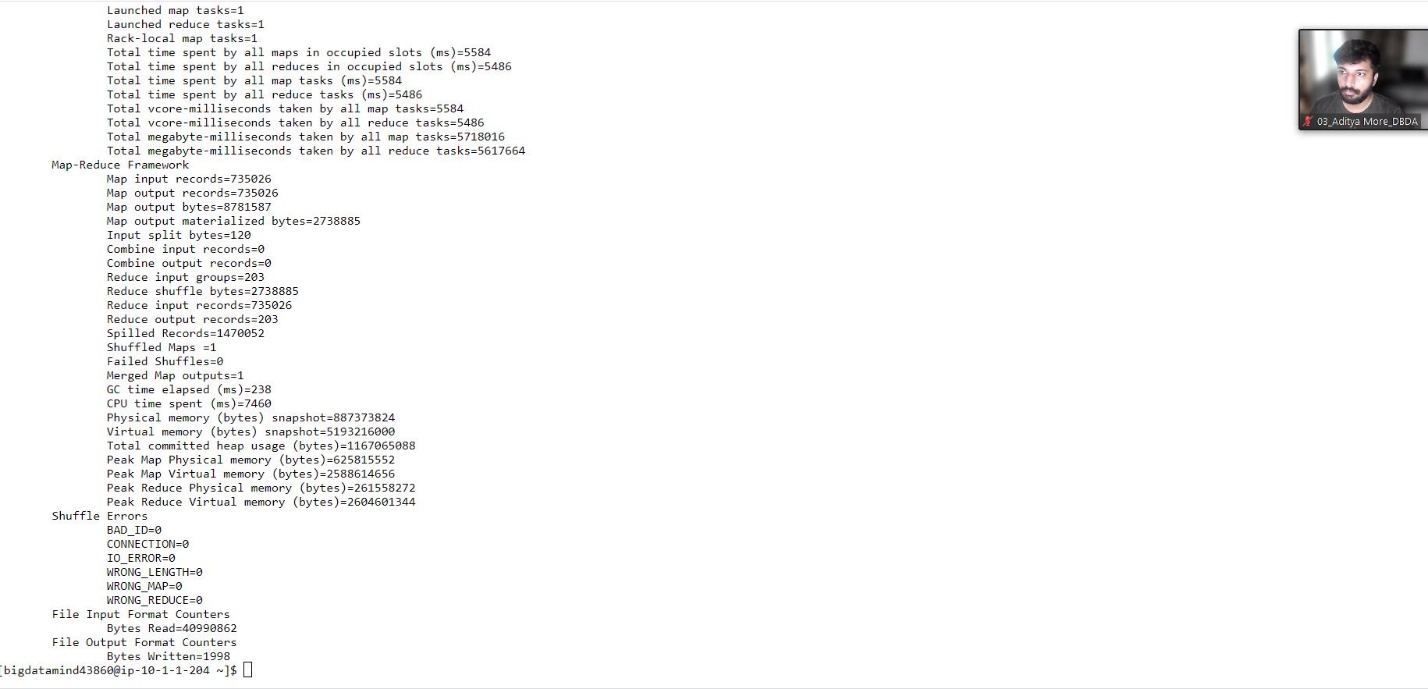
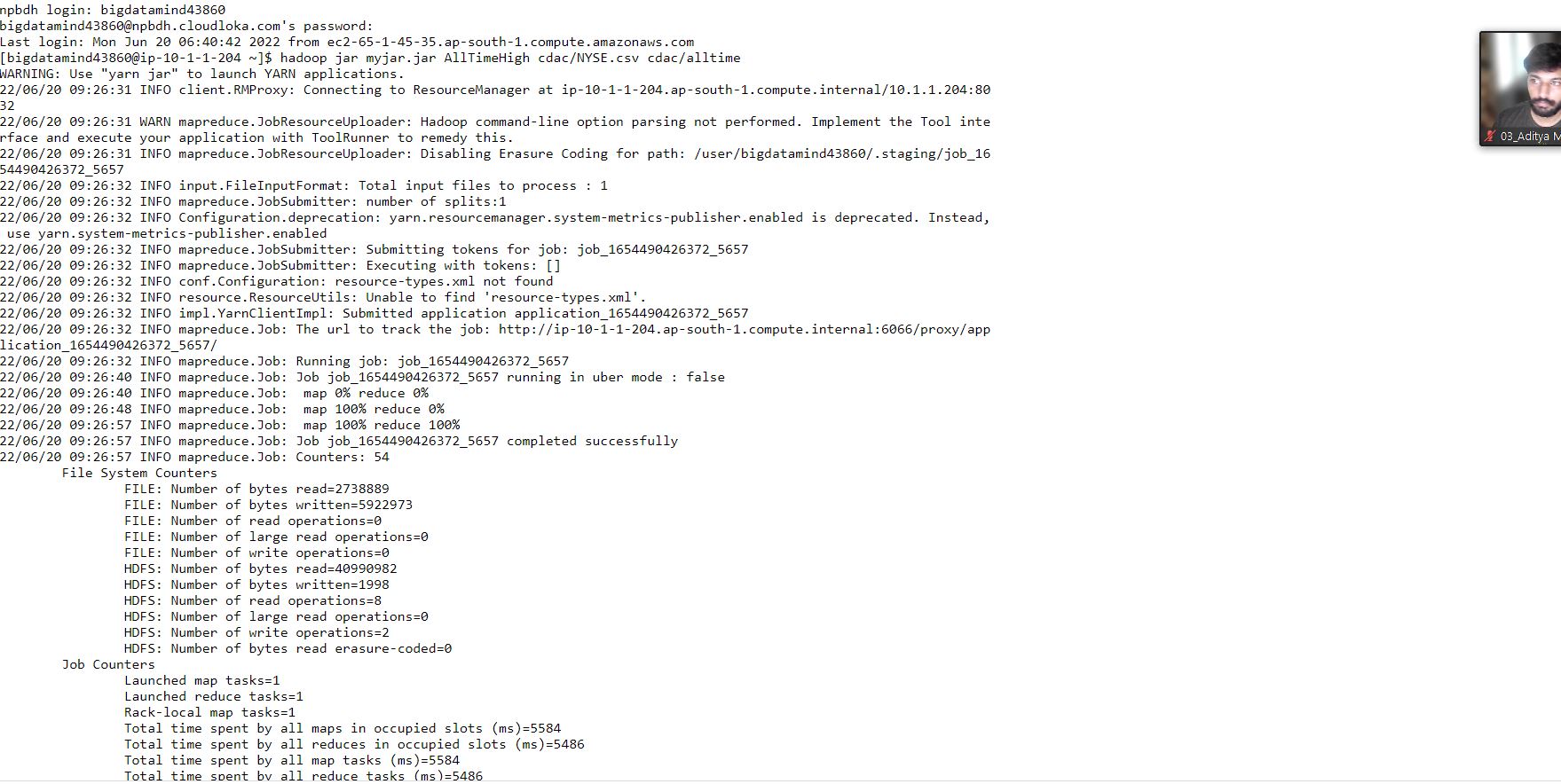
FileInputFormat.addInputPath(job, new Path(args[0]));

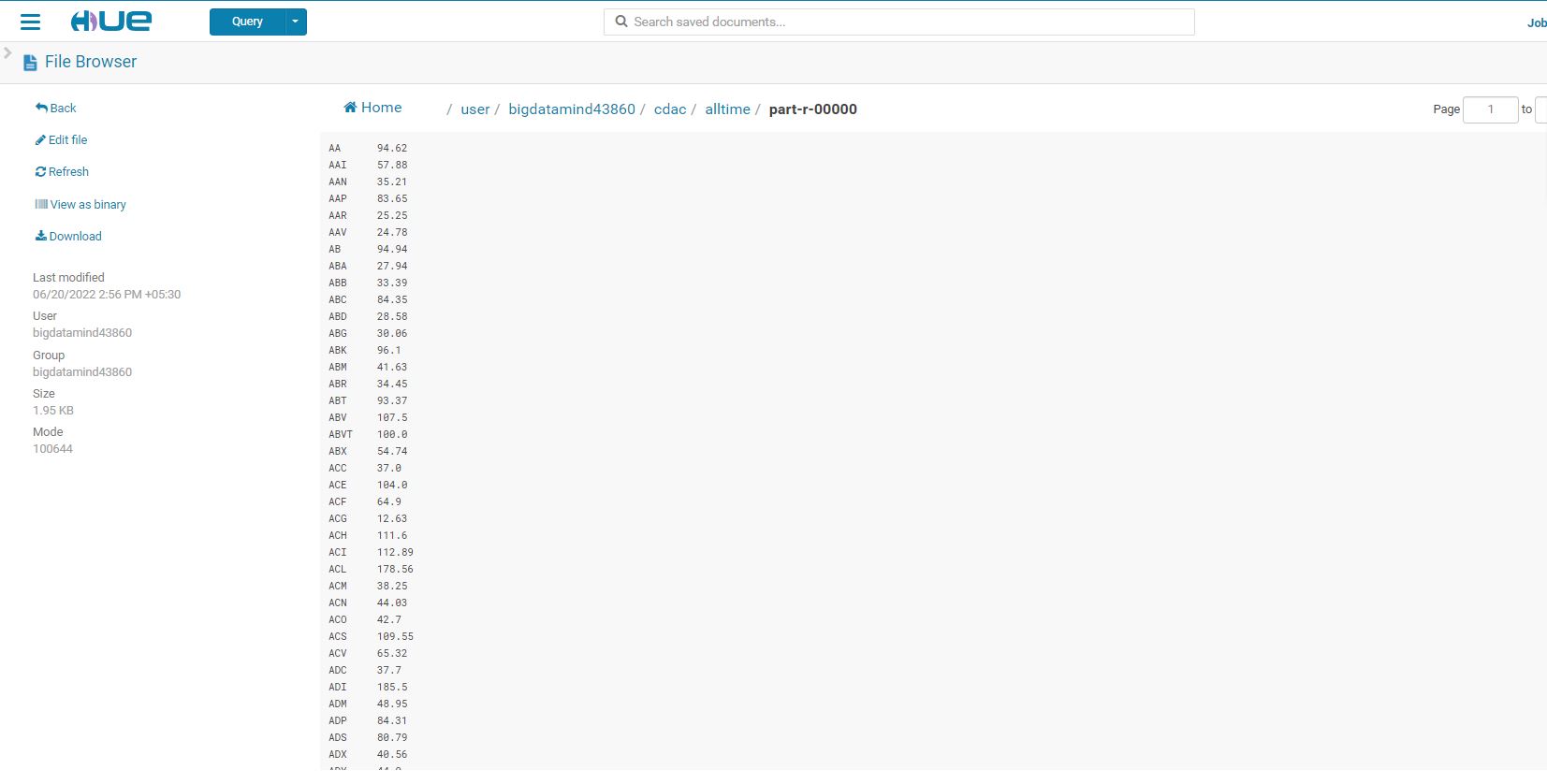
FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

}

}





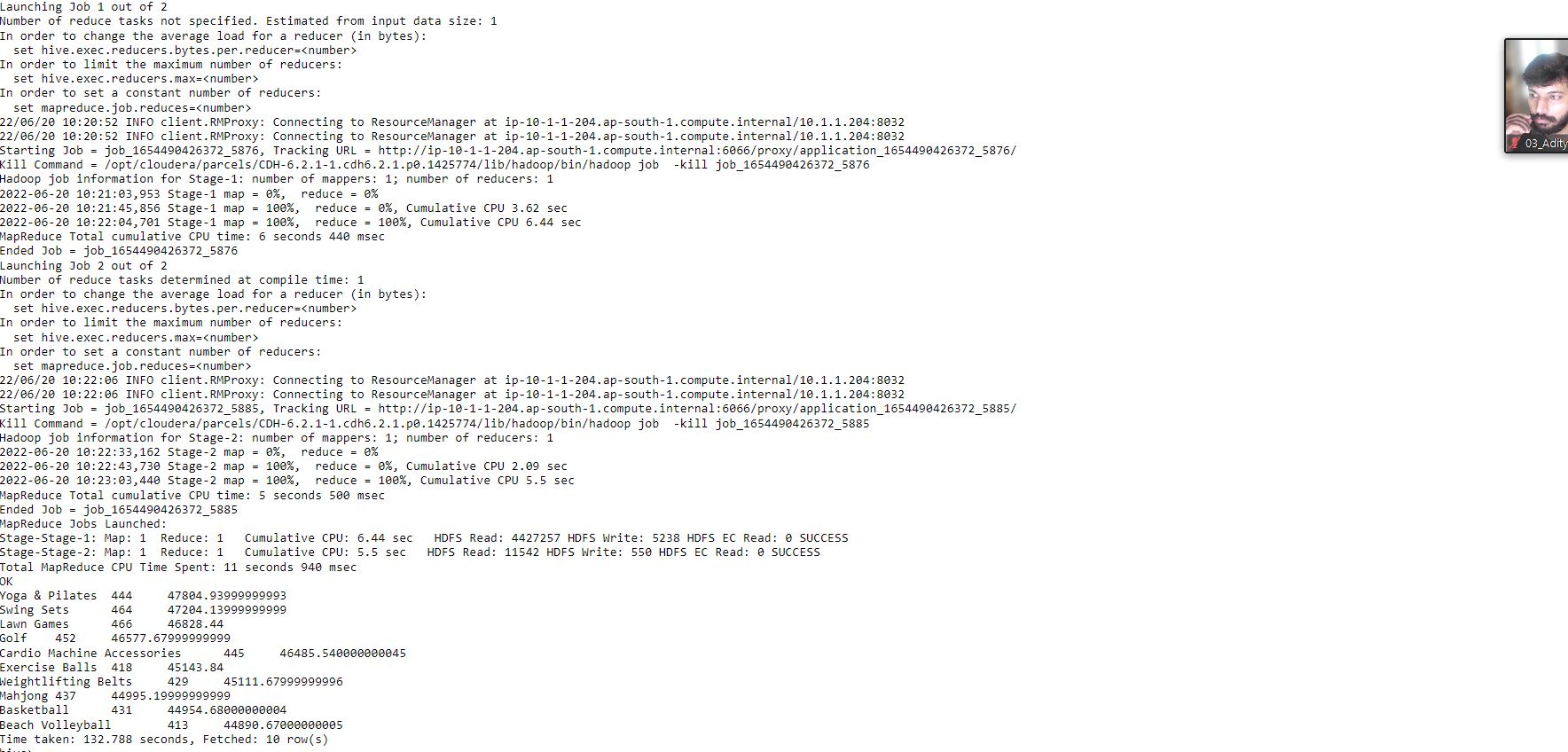
Q2 HIve

1.

select profession,count(custno)from customer group by profession;



2. select product,count(product),sum(amount)as total from txnrecords group by product order by total desc limit 10;



3.

create table txnrecsByCatp(txnno INT, txndate STRING, custno INT, amount DOUBLE,

product STRING, city STRING, state STRING, spendby STRING)

partitioned by (category STRING)

row format delimited

fields terminated by ','

stored as textfile;

set hive.exec.dynamic.partition=true;

set hive.mapred.supports.subdirectories=true;

set mapred.input.dir.recursive=true;

INSERT OVERWRITE TABLE txnrecsByCatp PARTITION(category) select txn.txnno, txn.txndate,txn.custno, txn.amount,txn.product,txn.city,txn.state, txn.spendby, txn.category from txnrecords txn DISTRIBUTE By category;

Q3 Pyspark

1

rdd=sc.textFile("/user/bigdatamind43860/airlines.csv")

rdd2=rdd.map(lambda a: a.encode("ascii","ignore"))

header=rdd2.first()

rdd3=rdd2.filter(lambda a:a!= header)

rdd4=rdd3.map(lambda a : a.split(","))

rdd=sc.textFile("/user/bigdatamind43860/airlines.csv")

rdd2=rdd.map(lambda a: a.encode("ascii","ignore"))

header=rdd2.first()

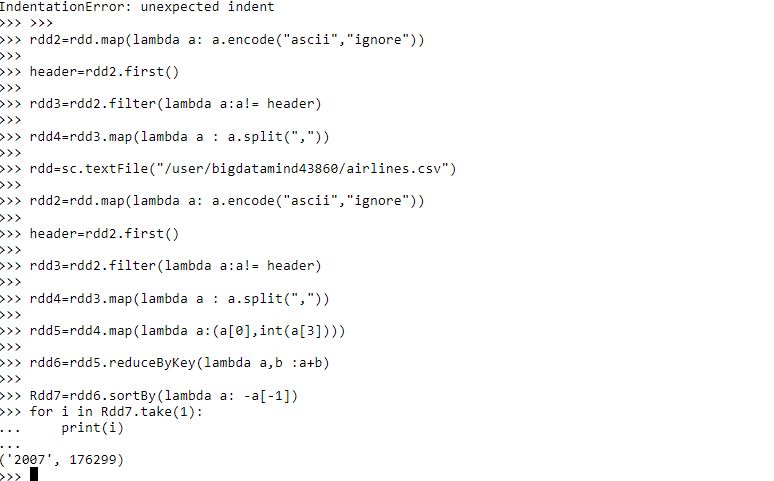
rdd3=rdd2.filter(lambda a:a!= header)

rdd4=rdd3.map(lambda a : a.split(","))

rdd5=rdd4.map(lambda a:(a[0],int(a[3])))

rdd6=rdd5.reduceByKey(lambda a,b :a+b)

Rdd7=rdd6.sortBy(lambda a: -a[-1])



2.

rdd=sc.textFile("/user/bigdatamind43860/airlines.csv")

rdd2=rdd.map(lambda a: a.encode("ascii","ignore"))

header=rdd2.first()

rdd3=rdd2.filter(lambda a:a!= header)

rdd4=rdd3.map(lambda a : a.split(","))

rdd=sc.textFile("/user/bigdatamind43860/airlines.csv")

rdd2=rdd.map(lambda a: a.encode("ascii","ignore"))

header=rdd2.first()

rdd3=rdd2.filter(lambda a:a!= header)

rdd4=rdd3.map(lambda a : a.split(","))

rdd5 = rdd4(lambda a : (a[0] ,float(a[2]) \* float(a[3])))

rdd6 = rdd5.reduceByKey(lambda a,b : a+b)

rdd7 = rdd6.sortBy(lambda a : -a[1])

for i in rdd7.collect():

... print(i)