\*\*\*\*\*\* Q.1 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>

{

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

{

try{

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

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

context.write(new Text(str[1]),new DoubleWritable(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 = 0.00;

for (DoubleWritable val : values)

{

if (val.get() > max) {

max = val.get();

}

}

result.set(max);

context.write(key, result);

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

}

}

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

Configuration conf = new Configuration();

conf.set("mapreduce.output.textoutputformat.separator",",");

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

conf.set("mapreduce.input.fileinputformat.split.maxsize", "28311552");

Job job = Job.getInstance(conf, "All Time High 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);

}

}

\*\*\*\*\*\* Q.2 hive:\*\*\*\*\*\*\*

hive

use training\_432562;

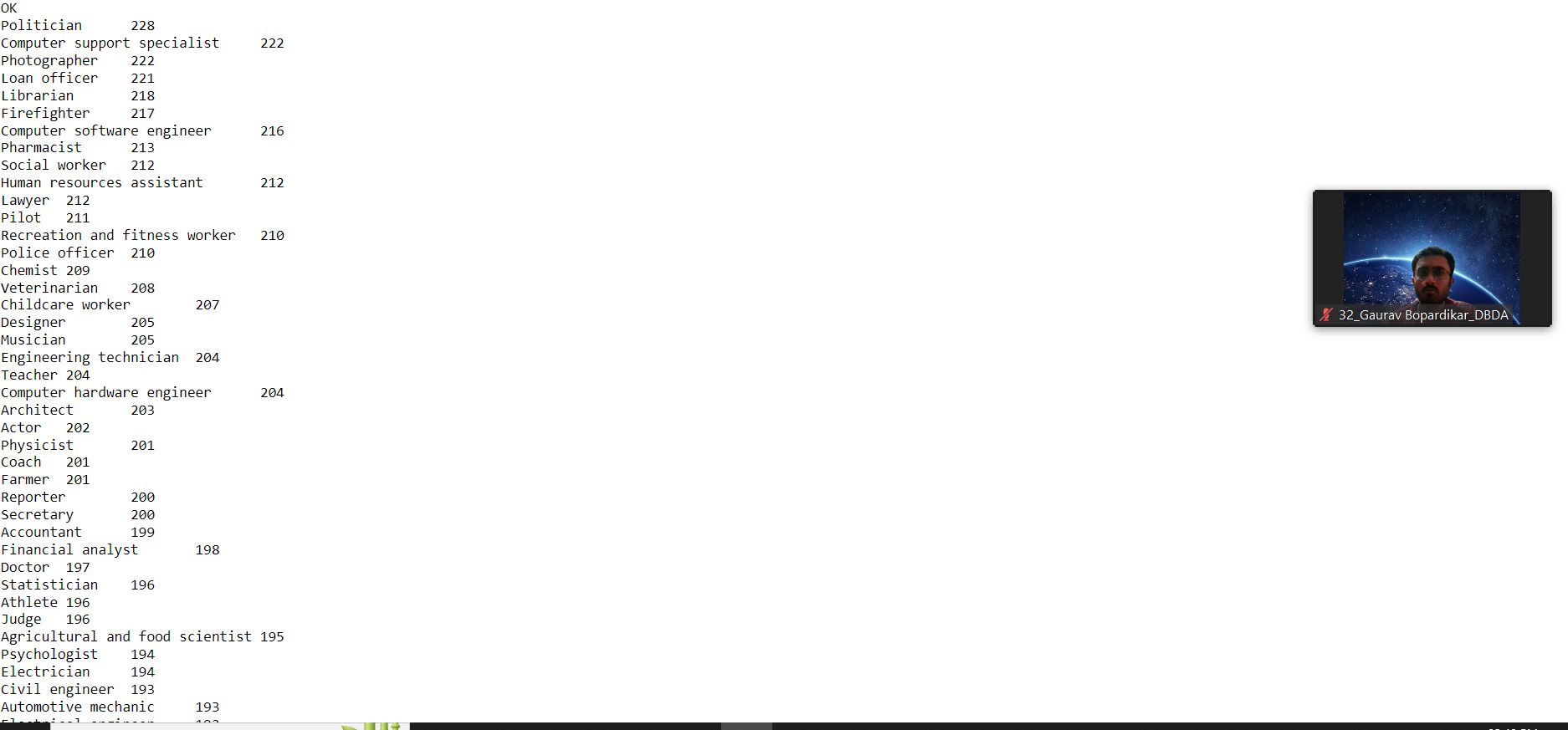
create table txn(txnno int,txndate string,custno int,amount double,category string,product string,city string,state string,spendby string) row format delimited fields terminated by ',' stored as textfile;

load data local inpath 'txns1.txt' overwrite into table txnrecords;

Create table customer(custno int,firstname string,lastname string,age int,profession string) row format delimited fields terminated by ',' stored as textfile;

load data local inpath 'custs.txt' overwrite into table customer;

Q.1- select profession,count(profession) as cnt from customer group by profession order by cnt desc;





Q.2- select product,count(product) as top from txn group by product order by top desc limit 10;

Q.3- create table txnrecsByCat(txnno int,txndate string,custno int,amount double,product string,city string,state string,spendby string);

set

hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.dynamic.partition=true;

insert overwrite table txnrecsByCat 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;

\*\*\*\*\*\*\* Q.3 Pyspark:\*\*\*\*\*\*\*

pyspark

from pyspark.sql.types import StructType,StringType,IntegerType,DoubleType,LongType

schema1=StructType().add("Year",IntegerType(),True).add("Quarter",IntegerType(),True).add("Average\_revenue\_per\_seat",DoubleType(),True).add("total\_booked\_seats",IntegerType(),T

rue)

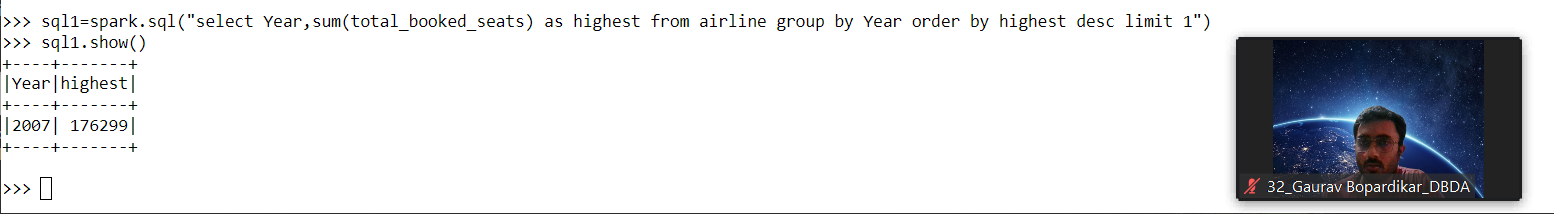
airRDD=spark.read.format("csv").option("header","True").schema(schema1).load("hdfs://nameservice1/user/bigcdac432562/training/airlines.csv")

airRDD.show()

airRDD.registerTempTable("airline")

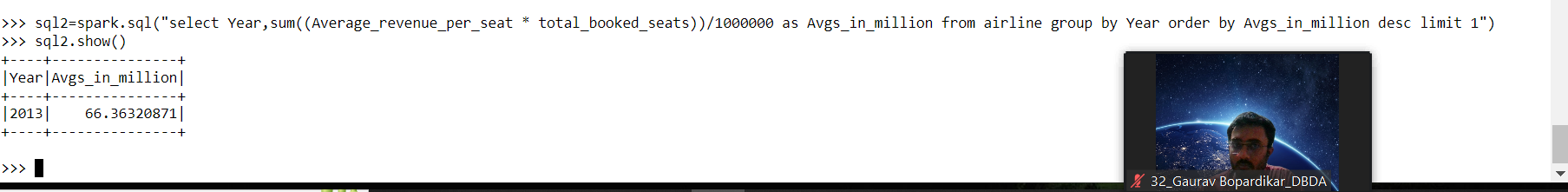
Q.1: sql1=spark.sql("select Year,sum(total\_booked\_seats) as highest from airline group by Year order by highest desc limit 1")

sql1.show()



Q.2: sql2=spark.sql("select Year,sum((Average\_revenue\_per\_seat \* total\_booked\_seats))/1000000 as Avgs\_in\_million from airline group by Year order by Avgs\_in\_million desc limit 1")

sql2.show()



Q.3: sql3=spark.sql("select Year,Quarter,(Average\_revenue\_per\_seat \* total\_booked\_seats)/1000000 as Avgs\_in\_million from airline order by Avgs\_in\_million desc")

sql3.show()

