Q1.

**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**);

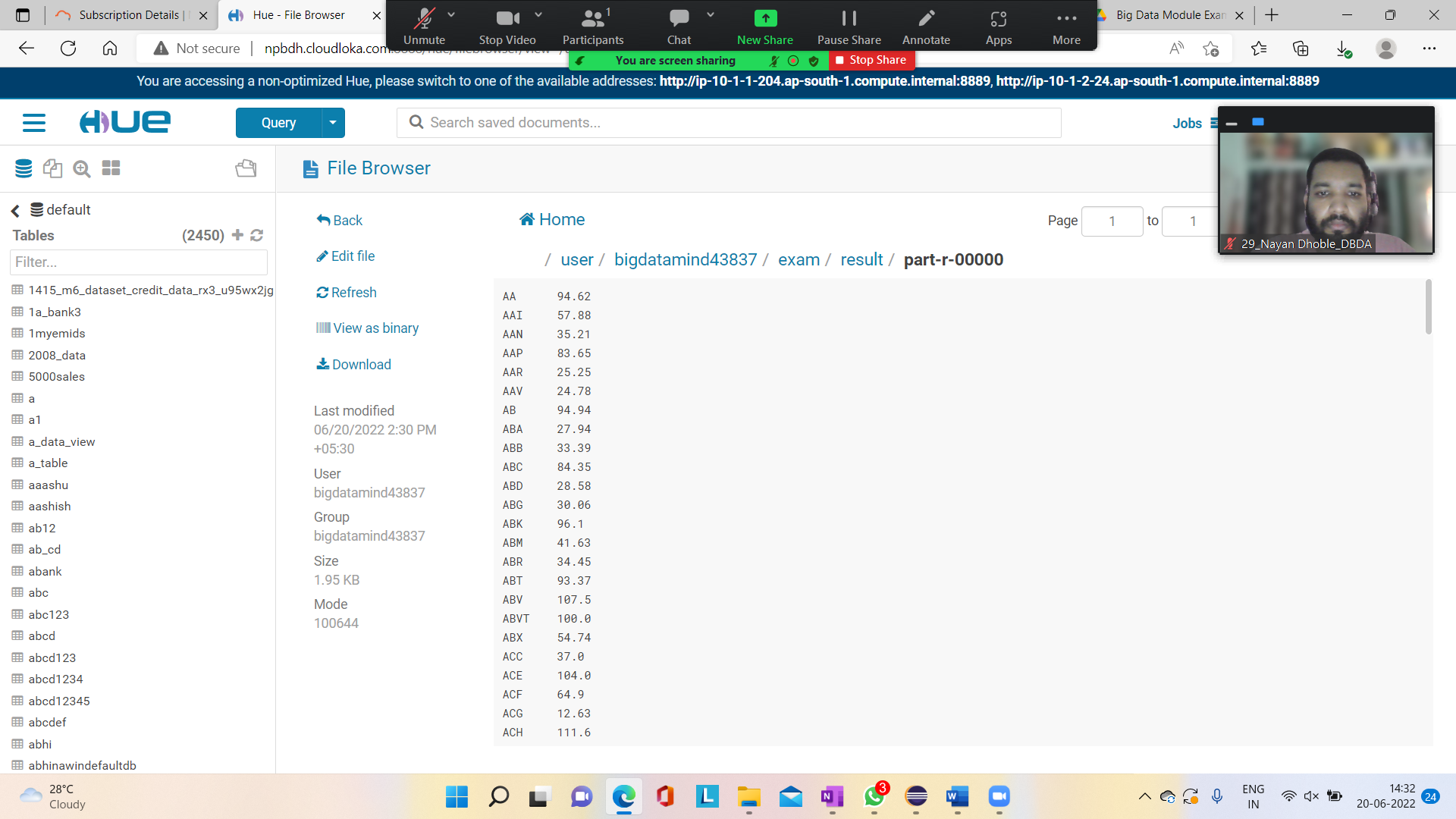
FileInputFormat.addInputPath(job, **new** Path(args[0]));

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

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

}

}



Q2.

create table customer(custid INT, first\_name STRING, last\_name STRING, age INT, profession STRING)row format delimited fields terminated by ',' stored

as textfile;

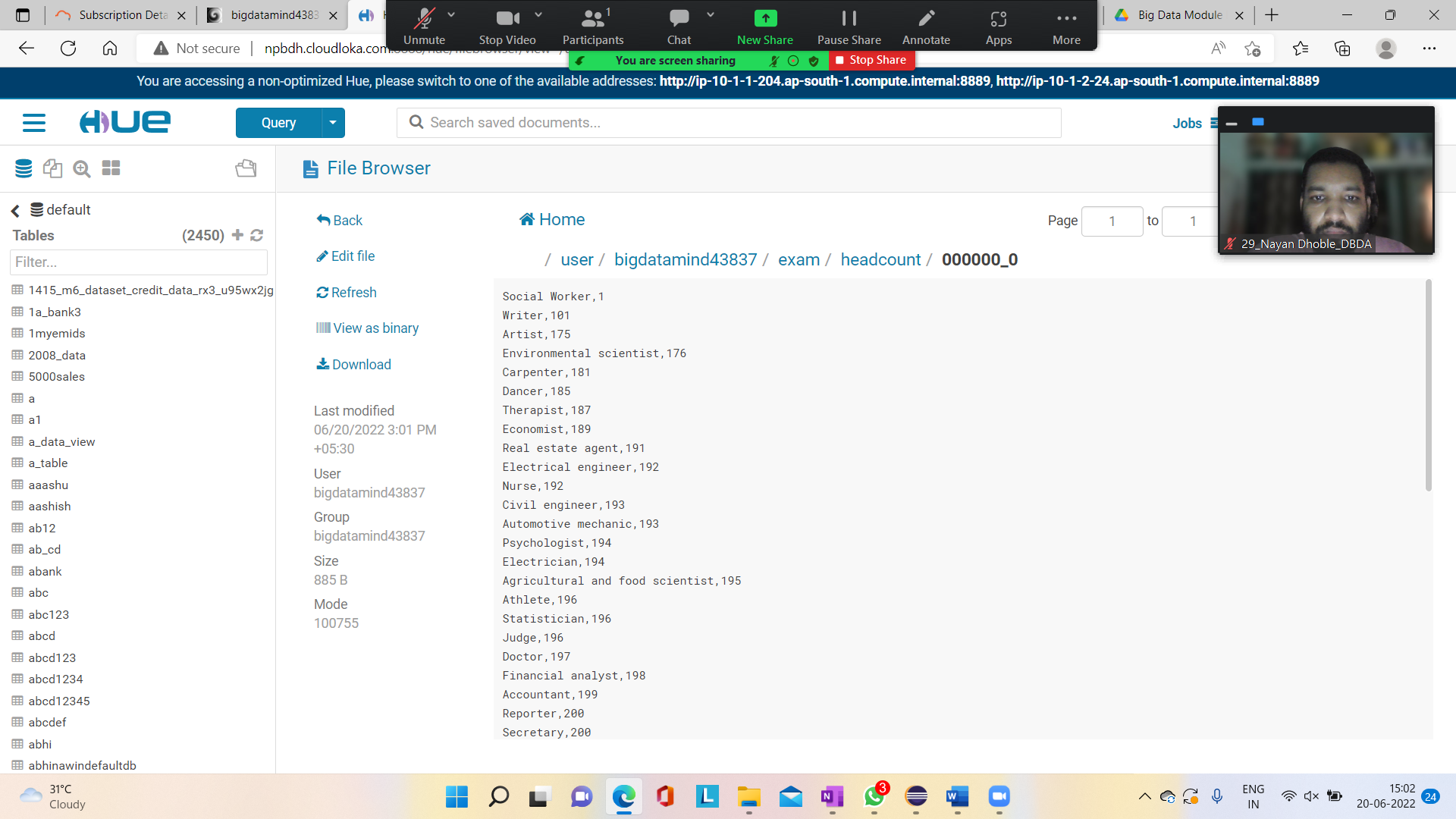
LOAD DATA LOCAL INPATH 'custs.txt' OVERWRITE INTO TABLE customer;

1.

select profession, count(\*) as headcount from customer group by profession order by headcount;

insert overwrite directory '/user/bigdatamind43837/exam/headcount' row format delimited fields terminated by ',' select profession, count(\*) as headcou

nt from customer group by profession order by headcount;



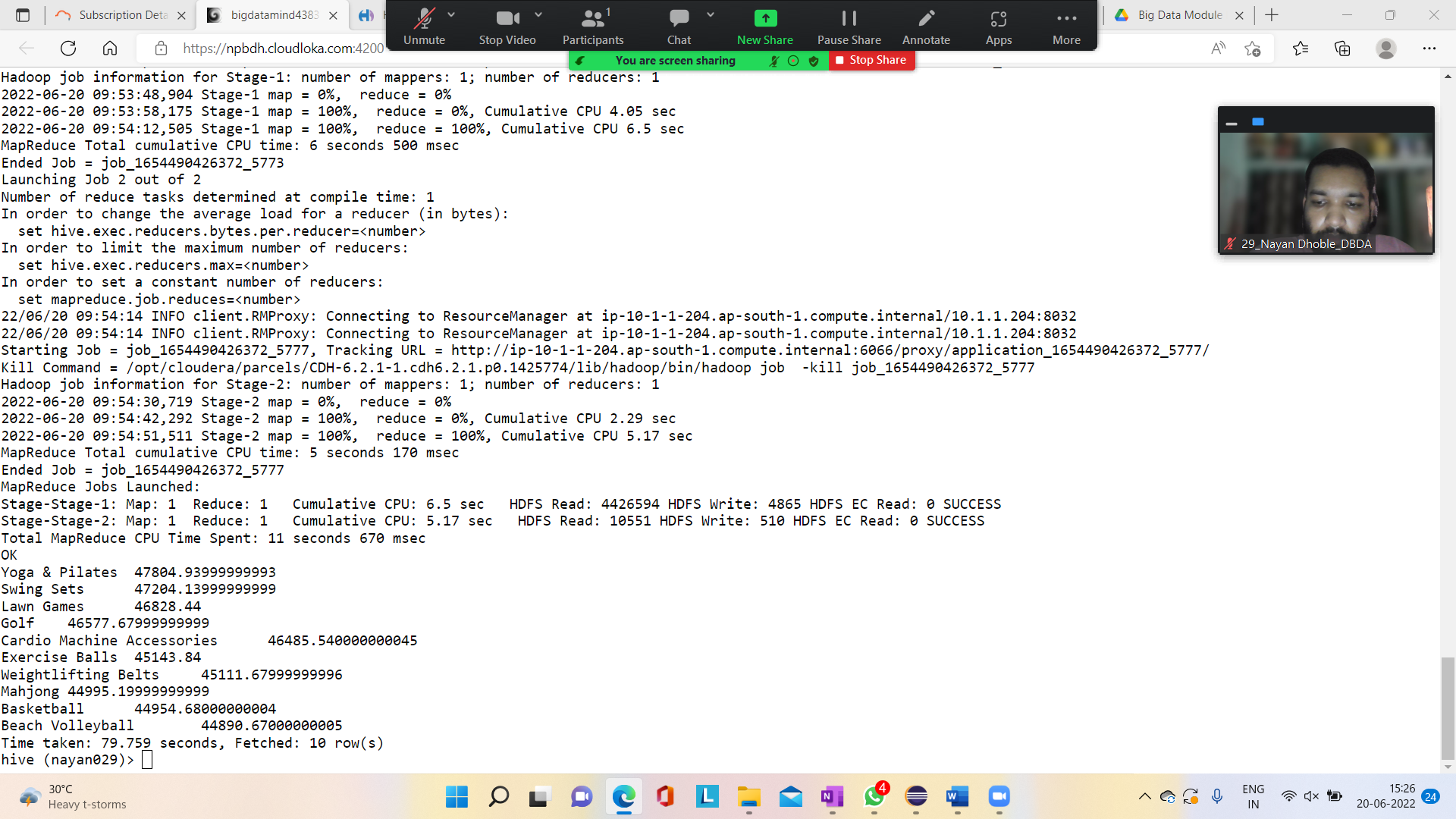
2.

create table txnrecords(txnno INT, txndate STRING, custid INT, amount DOUBLE, category STRING, product STRING, CITY STRING, state STRING, spendby STRIN

G)row format delimited fields terminated by ',' stored as textfile;

LOAD DATA LOCAL INPATH 'txns1.txt' OVERWRITE INTO TABLE txnrecords;

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



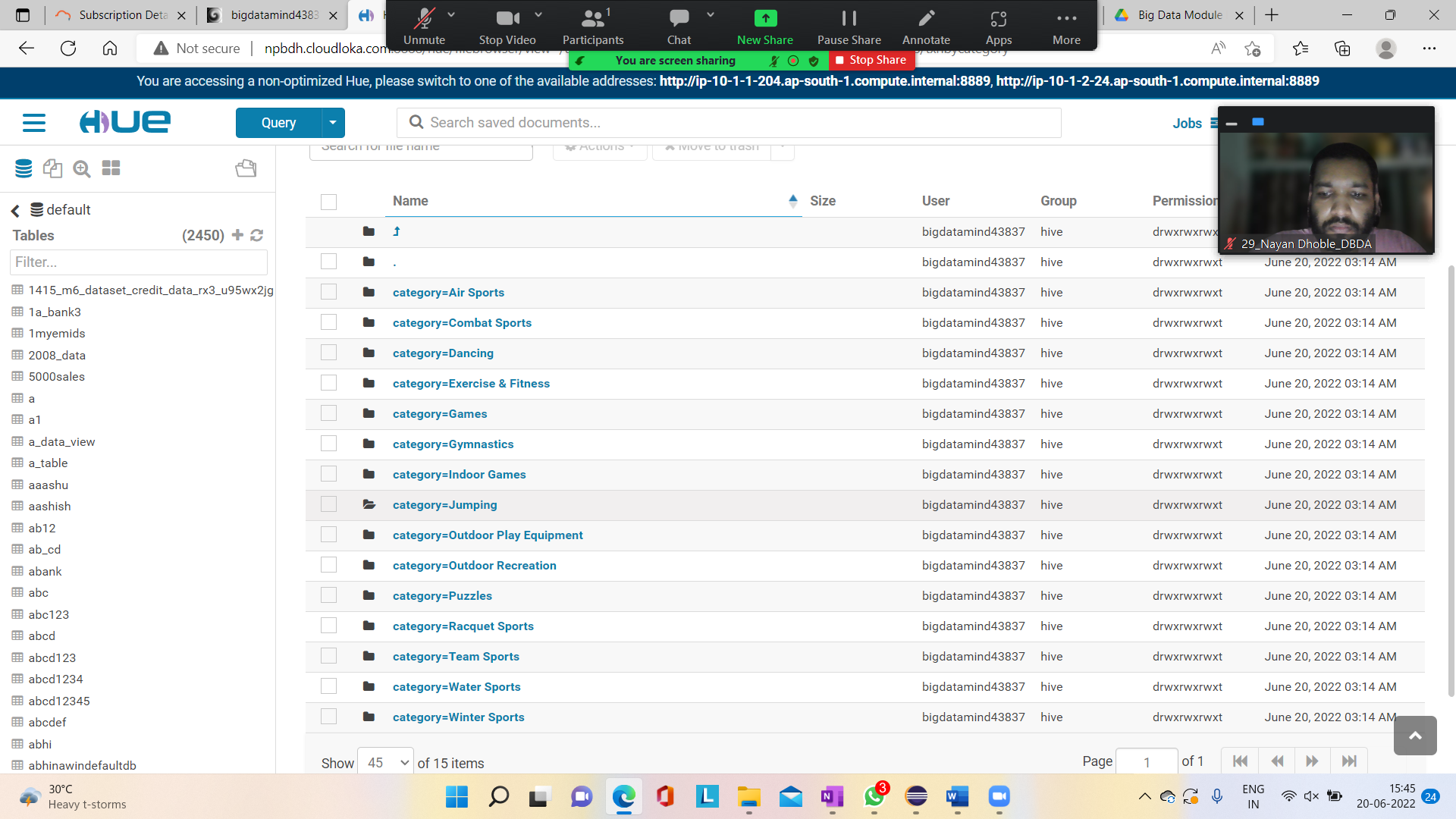
3.

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

set hive.exec.dynamic.partition=true;

create table txnbycategory(txnno INT, txndate STRING, custid INT, amount DOUBLE, product STRING, city STRING, state STRING, spendby STRING)partitioned

by (category STRING)row format delimited fields terminated by ',' stored as textfile;



Q3.

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

rdd = sc.textFile("/user/bigdatamind43837/airlines.csv")

>>> header = rdd.first()

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

>>> rdd3 = rdd2.map(lambda a : a.encode("ascii","ignore"))

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

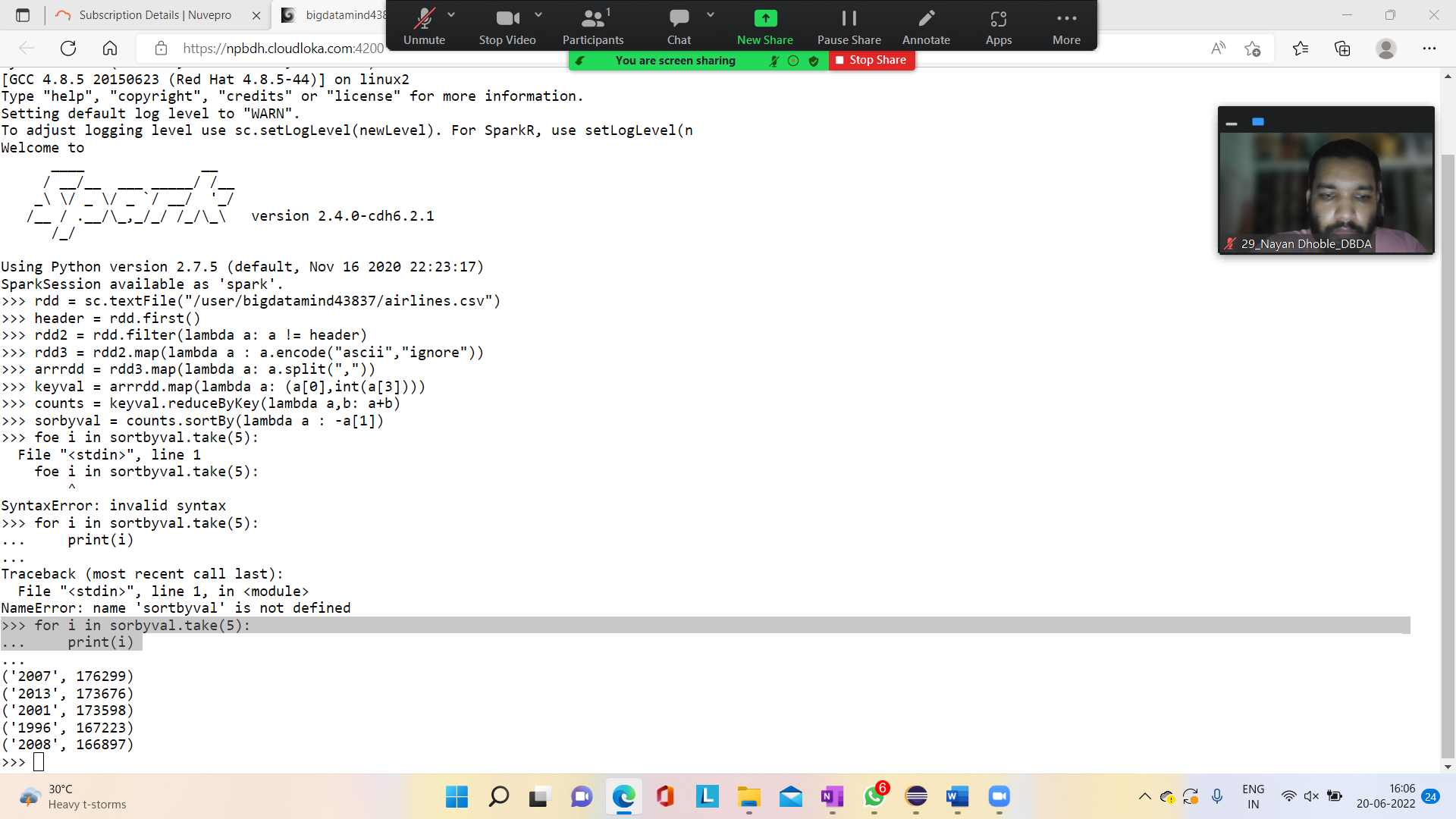
>>> keyval = arrrdd.map(lambda a: (a[0],int(a[3])))

>>> counts = keyval.reduceByKey(lambda a,b: a+b)

>>> sorbyval = counts.sortBy(lambda a : -a[1])

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

... print(i) s



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

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

schema1 = StructType().add("year", StringType(), True).add("qtr", StringType(), True).add("arps", DoubleType(),True).add("booked\_seat", LongType(),True)

airlinesDF =spark.read.format("csv").option("header", "TRUE").schema(schema1).load("/user/bigdatamind43824/airlines.csv")

airlinesDF.show()

+----+---+------+-----------+

|year|qtr| arps|booked\_seat|

+----+---+------+-----------+

|1995| 1| 296.9| 46561|

|1995| 2| 296.8| 37443|

|1995| 3|287.51| 34128|

|1995| 4|287.78| 30388|

|1996| 1|283.97| 47808|

|1996| 2|275.78| 43020|

|1996| 3|269.49| 38952|

|1996| 4|278.33| 37443|

|1997| 1| 283.4| 35067|

|1997| 2|289.44| 46565|

|1997| 3|282.27| 38886|

|1997| 4|293.51| 37454|

|1998| 1|304.74| 31315|

|1998| 2|300.97| 30852|

|1998| 3|315.25| 38118|

|1998| 4|316.18| 35393|

|1999| 1|331.74| 47453|

|1999| 2|329.34| 38243|

|1999| 3|317.22| 33048|

|1999| 4|317.93| 31256|

+----+---+------+-----------+

only showing top 20 rows

airlinesDF.registerTempTable("airline")

>>> high\_revenue = spark.sql("select year, round(sum(arps \* booked\_seat)/1000000) as in\_mill from airline group by year order by in\_mill desc")

>>> high\_revenue.show()

+----+-------+

|year|in\_mill|

+----+-------+

|2013| 66.0|

|2014| 63.0|

|2015| 62.0|

|2012| 62.0|

|2008| 58.0|

|2007| 57.0|

|2001| 56.0|

|2010| 55.0|

|2011| 52.0|

|2000| 52.0|

|2004| 51.0|

|2006| 50.0|

|2003| 49.0|

|1999| 49.0|

|2002| 47.0|

|2009| 47.0|

|2005| 46.0|

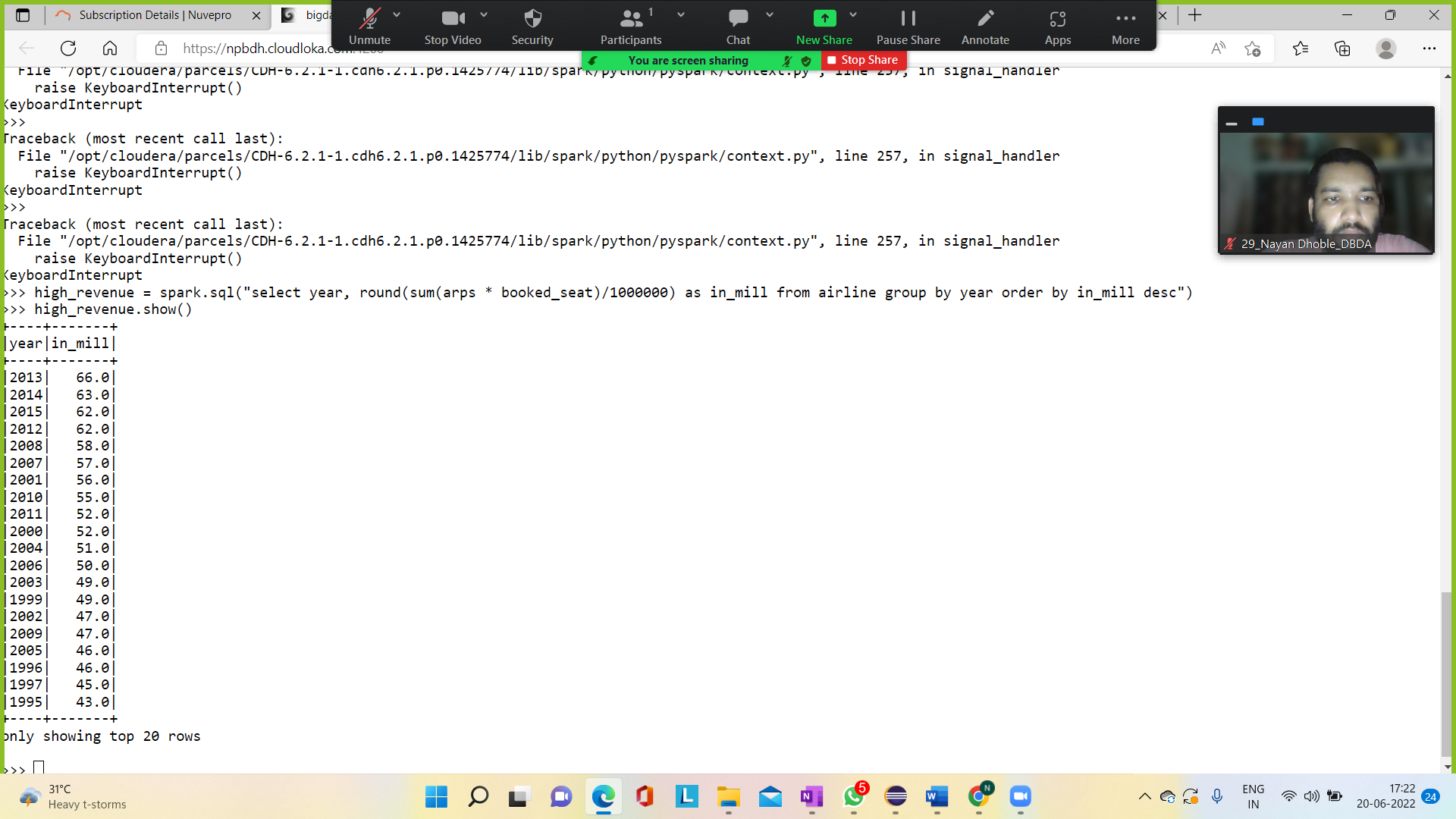
|1996| 46.0|

|1997| 45.0|

|1995| 43.0|

+----+-------+

only showing top 20 rows



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

highrevenue = spark.sql("select (year,qtr) as period, sum(arps\*booked\_seat)/1000000 as rev\_in\_million from airline group by period order by rev\_in\_million desc limit 1

")

Output:-

>>> highrevenue.show()

+---------+--------------+

|period|rev\_in\_million|

+---------+--------------+

|[2014, 4]| 18.81940848|

+---------+--------------+

