**Lab Exam**

Q1)

>

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class AllTimeHigh

{

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

{

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

{

try

{

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

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

context.write(new Text(s[1]), new DoubleWritable(high));

}

catch (Exception e)

{

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

}

}

}

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

{

private DoubleWritable res = new DoubleWritable();

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

{

double max = 0.0;

for(DoubleWritable val: values)

{

if(max < val.get())

{

max = val.get();

}

}

res.set(max);

context.write(key, res);

}

}

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

{

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Highest Stock Value");

job.setJarByClass(AllTimeHigh.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]));

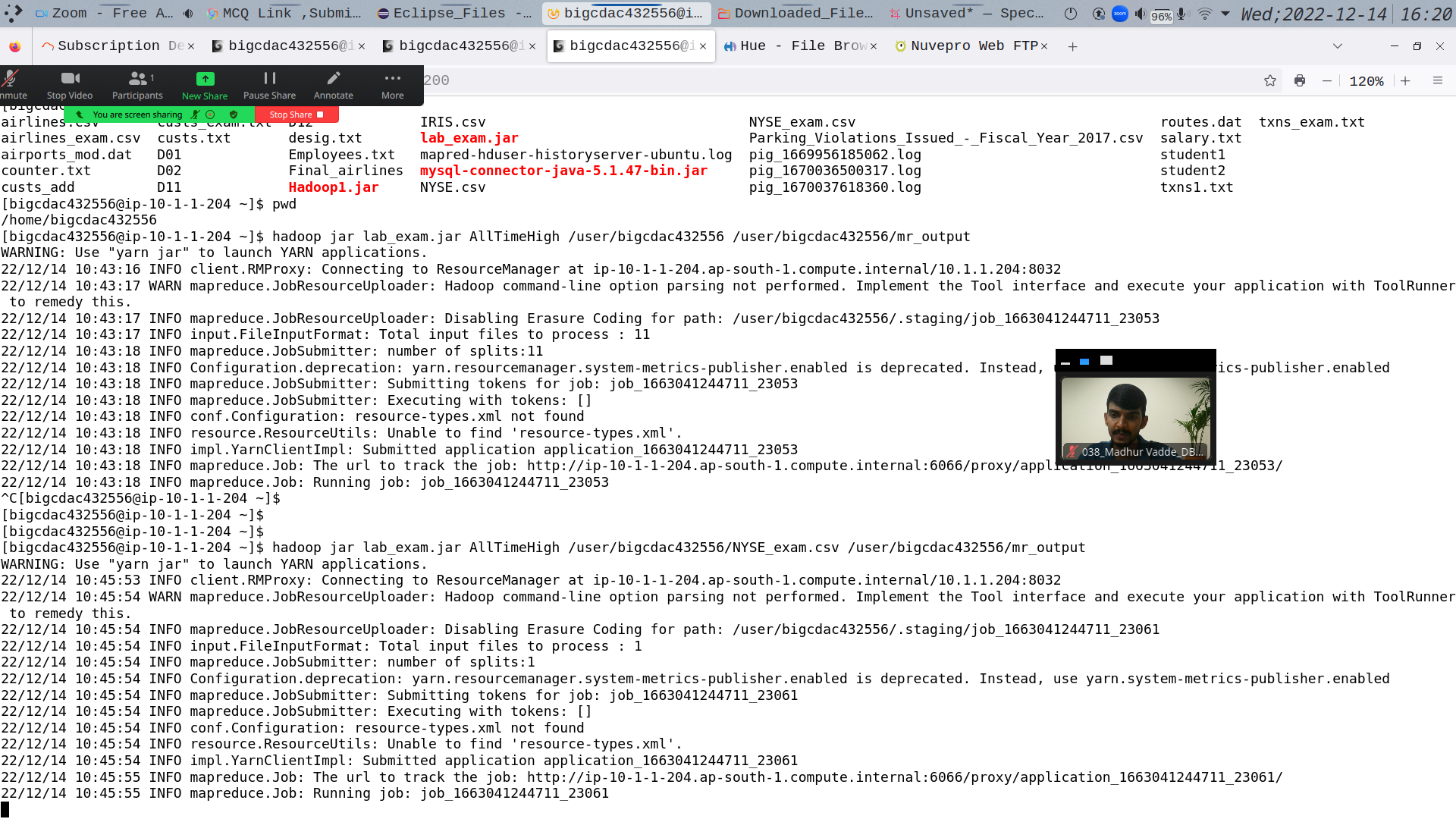
FileOutputFormat.setOutputPath(job, new Path(args[1]));

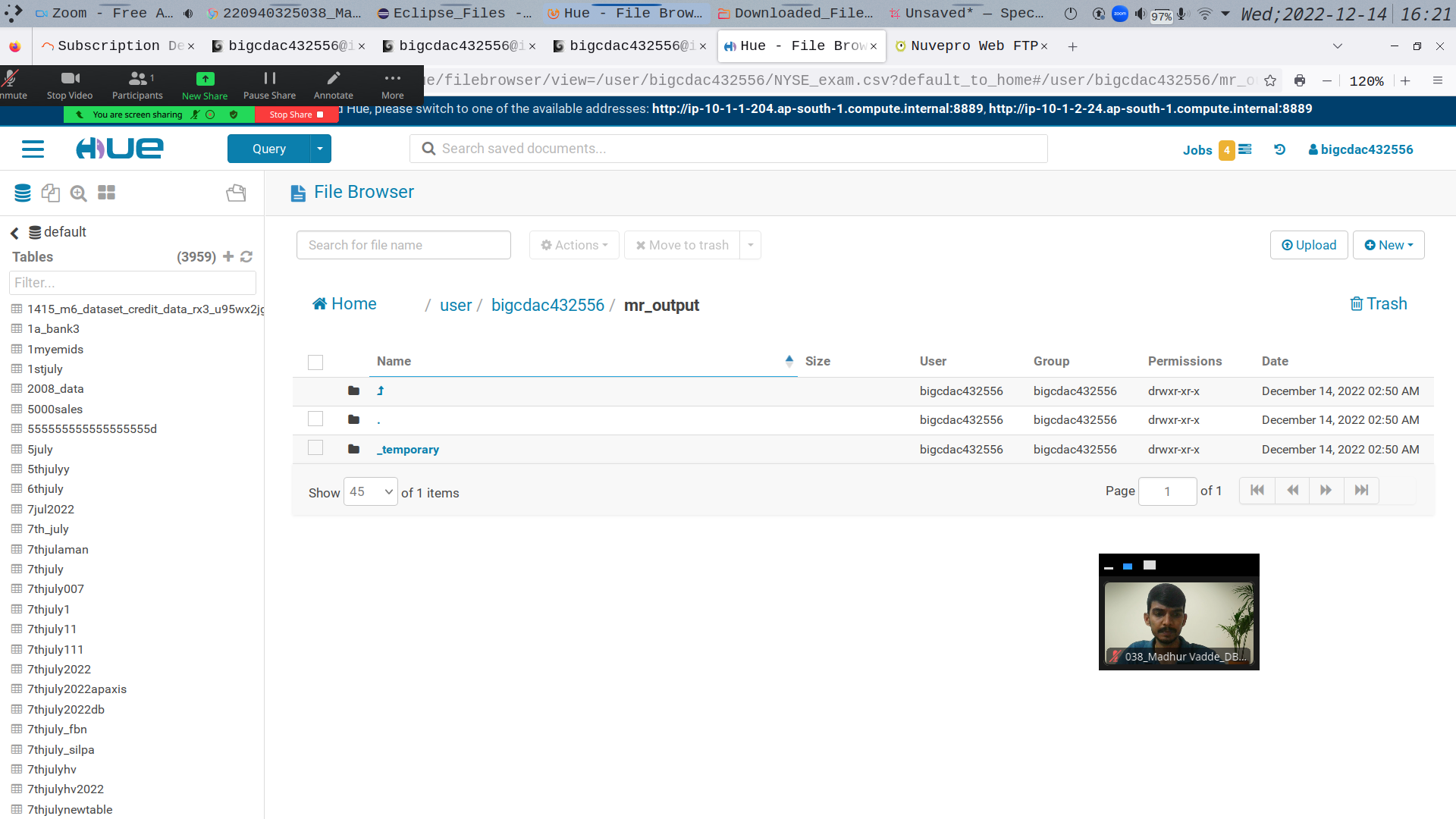
System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

hadoop jar lab\_exam.jar AllTimeHigh /user/bigcdac432556/NYSE\_exam.csv /user/bigcdac432556/mr\_output





Q2)

>

Create customers table:

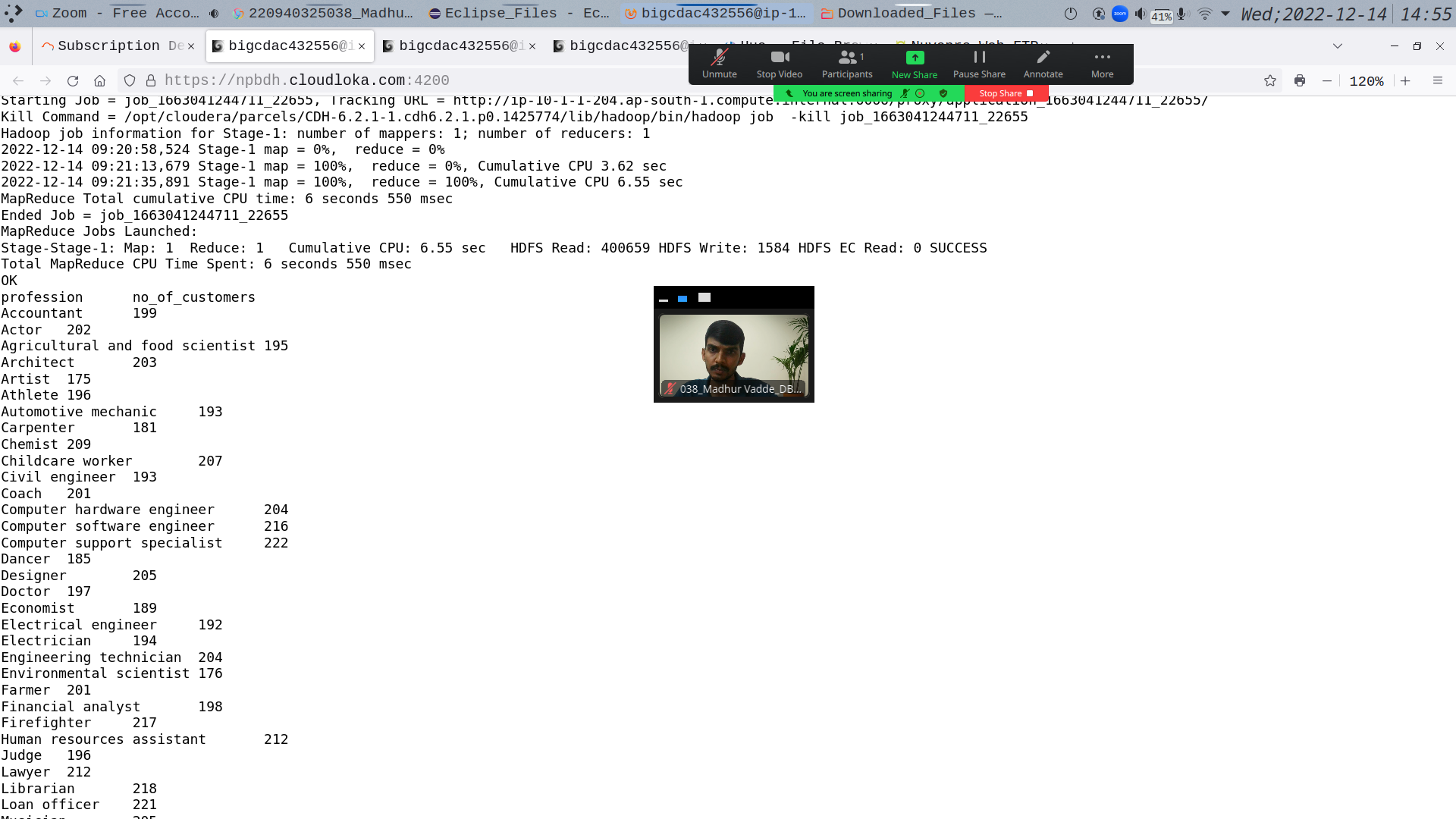
create table customer\_exam(cust\_id int, firstname string, lastname string, age int

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

load data local inpath ‘custs\_exam.txt’ overwrite into table customers\_table;

**Query 1:**

select profession, count(cust\_id) no\_of\_customers from customer\_exam group by profession;

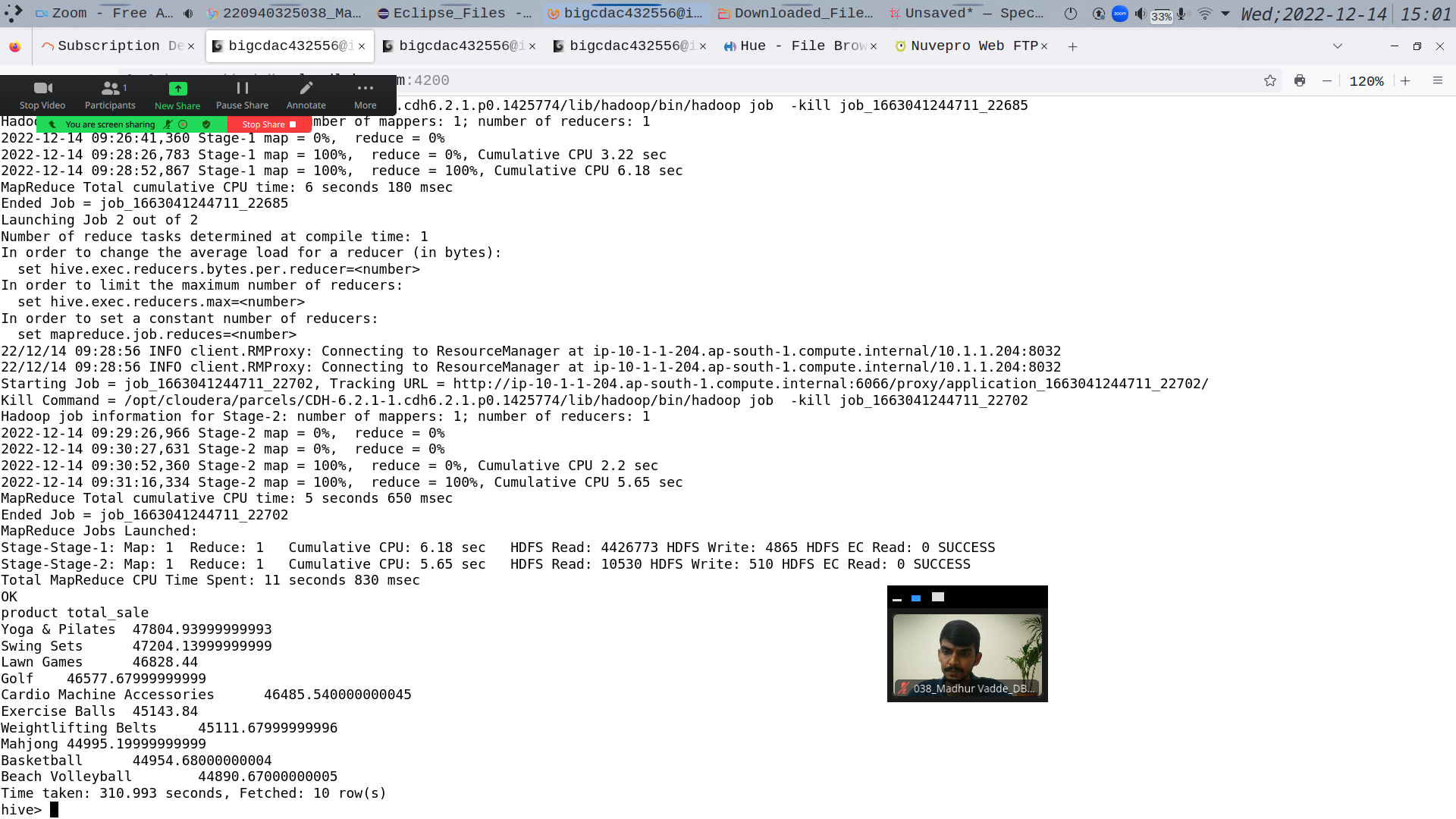


Create transactions table:

create table transactions\_exam(txn\_id int, txn\_date string, cust\_id int, amount double, category string, product string, city string, state string, spendby string) row format delimited fields terminated by ‘,’ stored as textfile

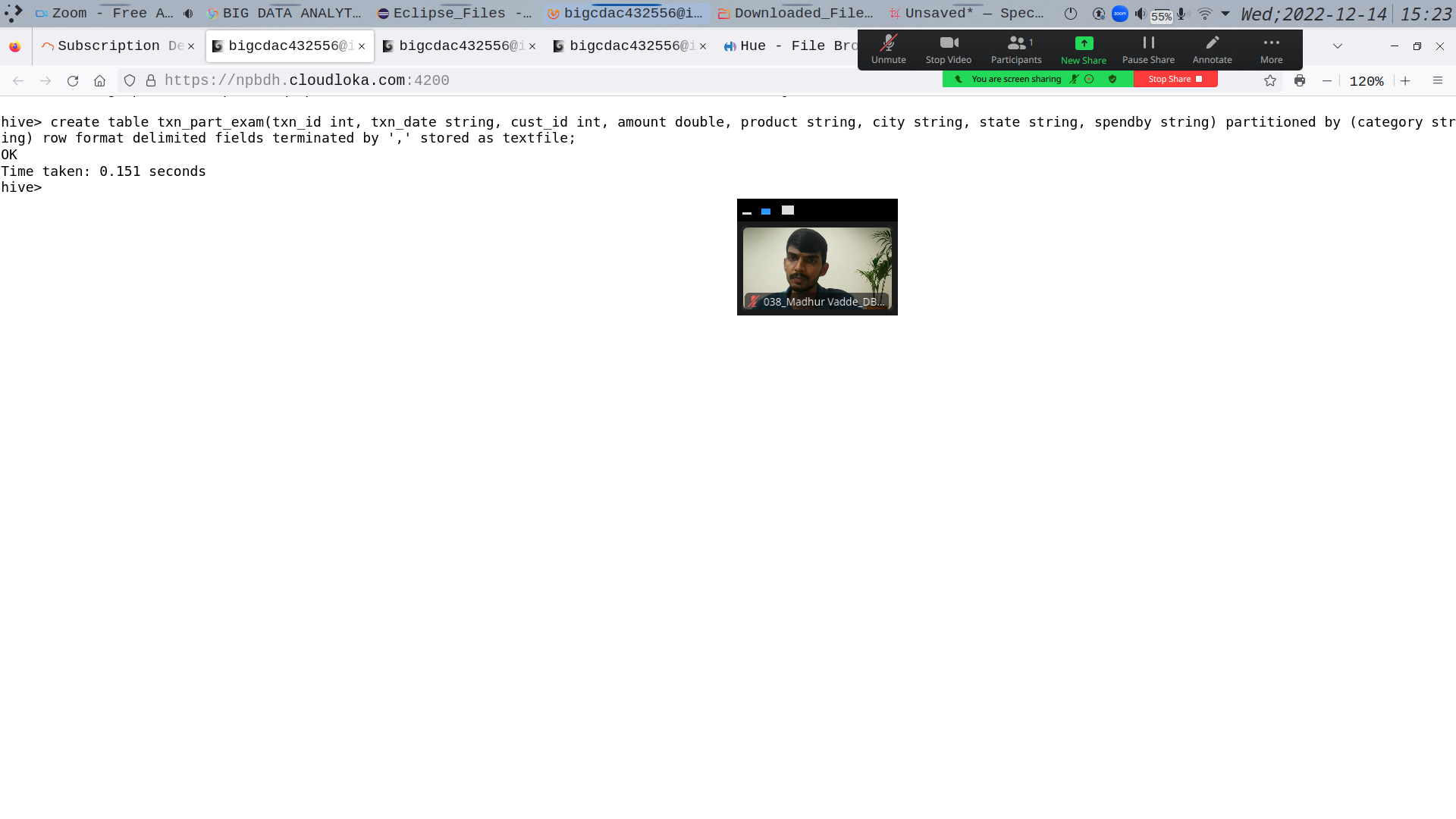
**Query 2:**

select product, sum(amount) total\_sale from transactions\_exam group by product order by total\_sale desc limit 10;



**Partitioned table on category:**

create table txn\_part\_exam(txn\_id int, txn\_date string, cust\_id 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)

>

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

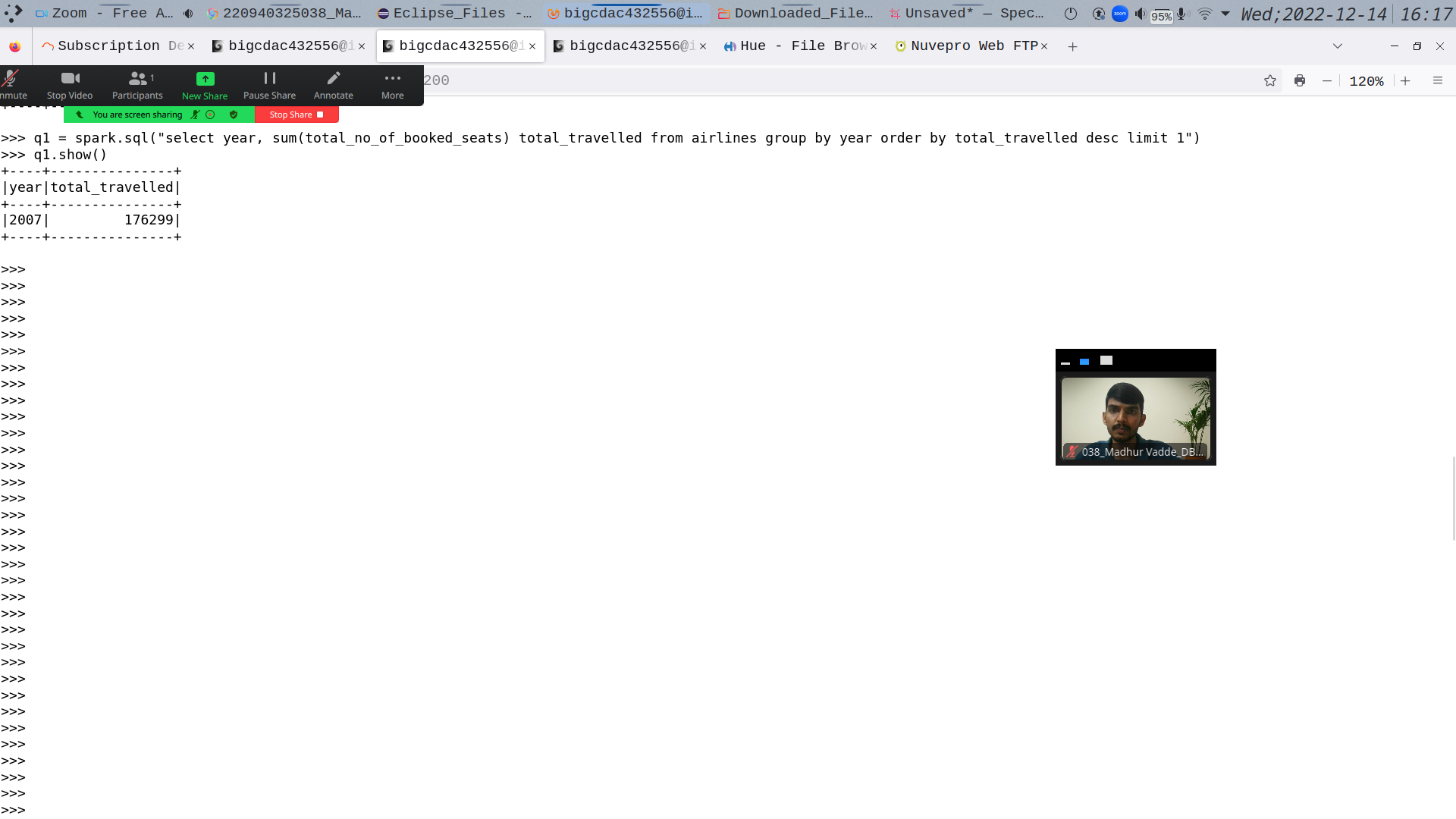
air\_schema = StructType().add("year", StringType(), True).add("quarter", StringType(), True).add("avg\_revenue\_per\_seat", DoubleType(), True).add("total\_no\_of\_booked\_seats", LongType(), True)

air\_df = spark.read.format("csv").option("header","True").schema(air\_schema).load("hdfs://nameservice1/user/bigcdac432556/airlines\_exam.csv")

air\_df.registerTempTable("airlines")

**Query 1:**

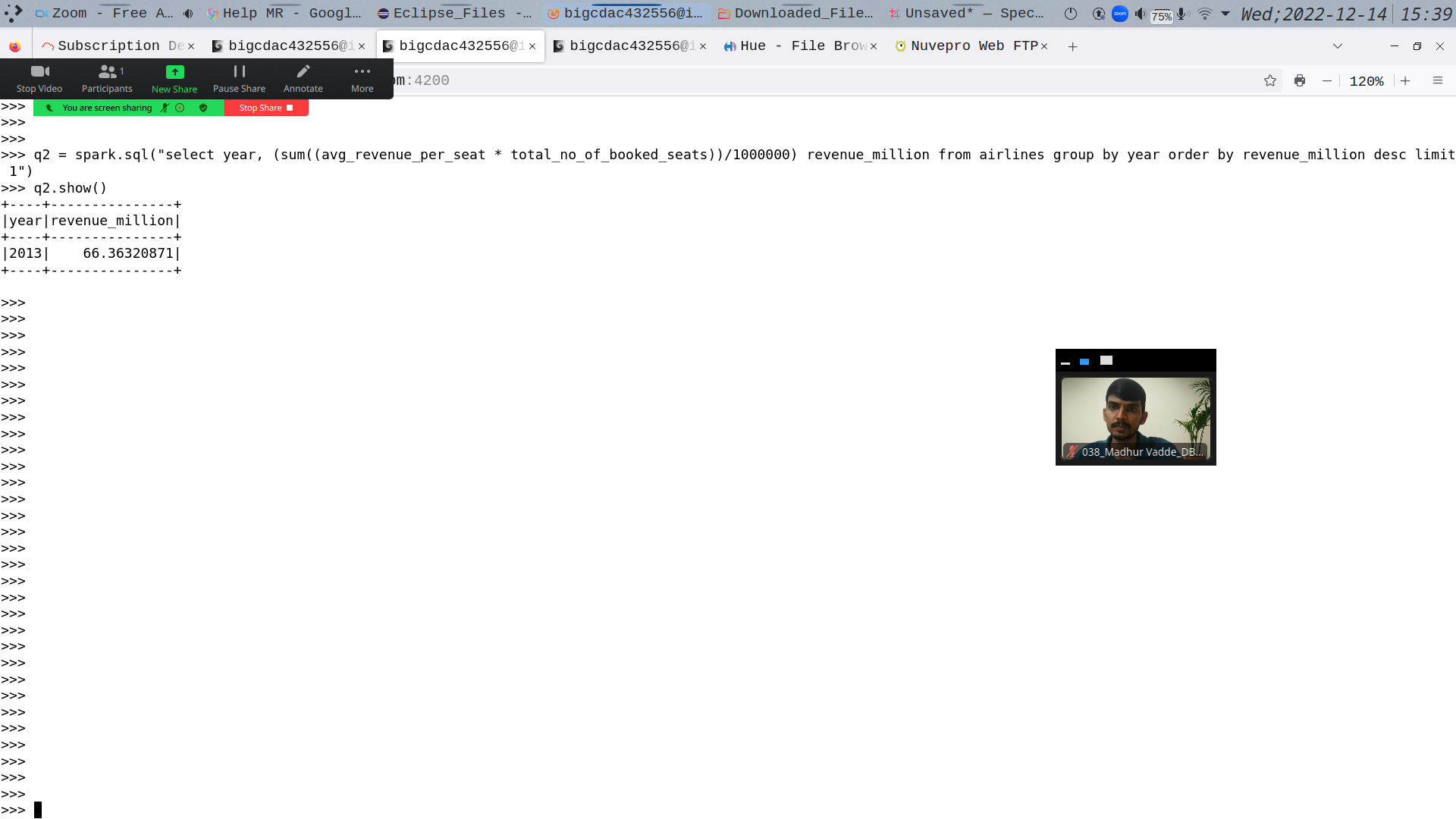
q1 = spark.sql("select year, sum(total\_no\_of\_booked\_seats) total\_travelled from airlines group by year order by total\_travelled desc limit 1")



**Query 2:**

q2 = spark.sql("select year, (sum((avg\_revenue\_per\_seat \* total\_no\_of\_booked\_seats))/1000000) revenue\_million from airlines group by year order by revenue\_million desc limit

1")



**Query 3:**

q3 = spark.sql("select year, quarter, ((avg\_revenue\_per\_seat \* total\_no\_of\_booked\_seats)/1000000) highest\_revenue\_million from airlines order by highest\_revenue\_million desc limit 1")

