**MapReduce**

**Question: Find all time high price for each stock**

**Solution:**

**Code:**

**import** java.io.\*;

//import org.apache.hadoop.\*;

**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.mapreduce.Reducer.Context;

**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>

{

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

{

**double** max=0.0;

**for**(DoubleWritable val:value)

{

**if**(val.get()>max)

{

max=val.get();

}

}

context.write(key, **new** DoubleWritable(max));

}

}

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

{

Configuration conf = **new** Configuration();

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

job.setJarByClass(AllTimeHigh.**class**);

job.setMapperClass(MapClass.**class**);

job.setReducerClass(ReduceClass.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(DoubleWritable.**class**);

job.setNumReduceTasks(1);

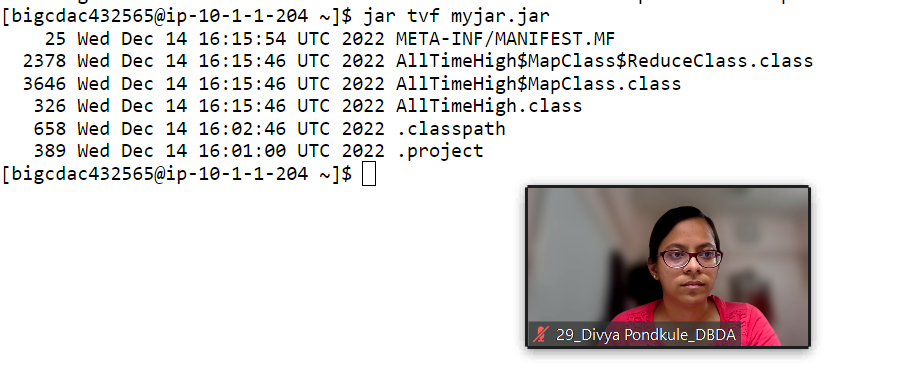
FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

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

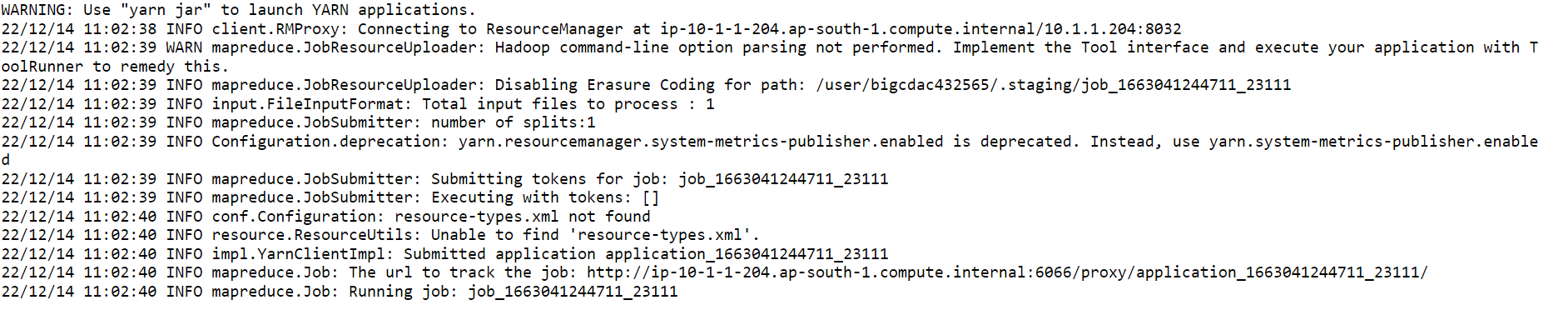
System.*exit*(job.waitForCompletion(**true**)?0:1);;

}

}

****

hadoop jar myjar.jar AllTimeHigh /user/bigcdac432565/training/NYSE.csv /user/bigcdac432565/training/mout1



**Hive**

Please find the customer data set.

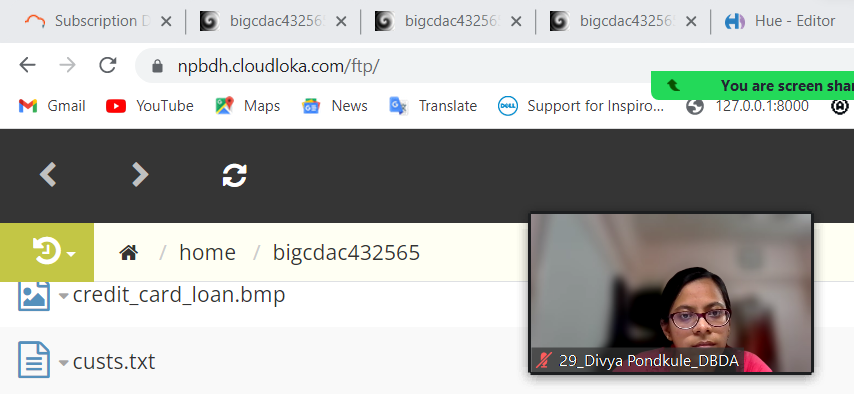
cust id

firstname

lastname

age

profession



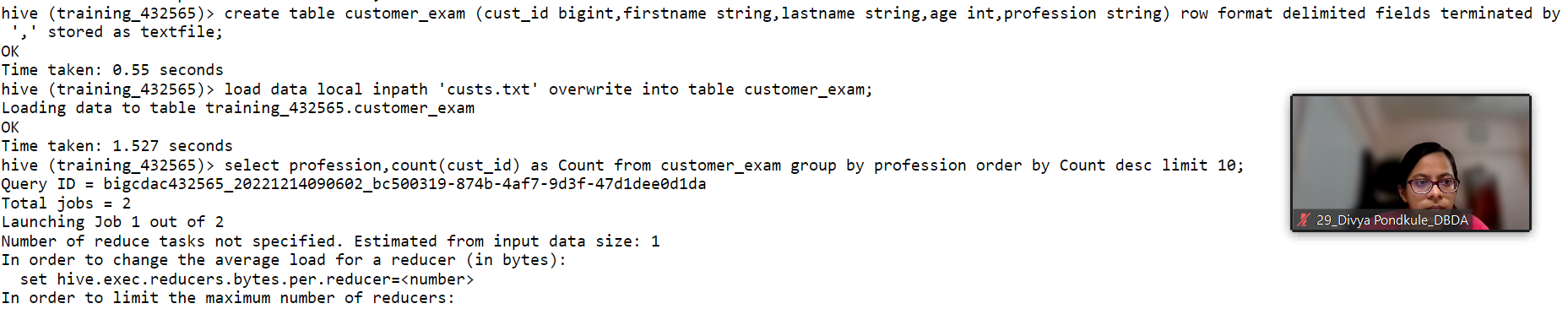
**1) Write a program to find the count of customers for each profession.**

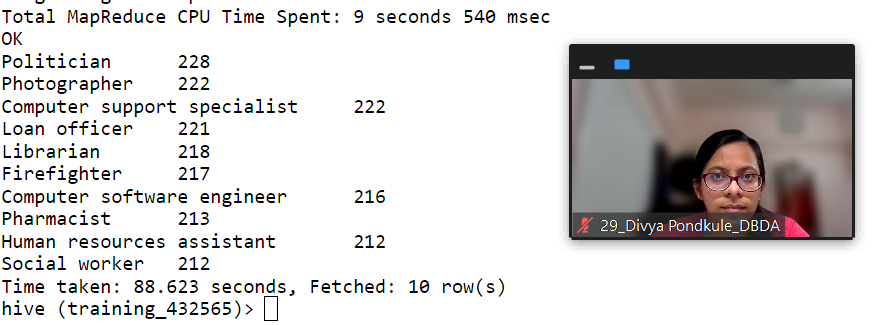
**Solution:**

create table customer\_exam (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.txt' overwrite into table customer\_exam;

select profession,count(cust\_id) as Count from customer\_exam group by profession order by Count desc limit 10;





Please find the sales data set.

txn id

txn date

cust id

amount

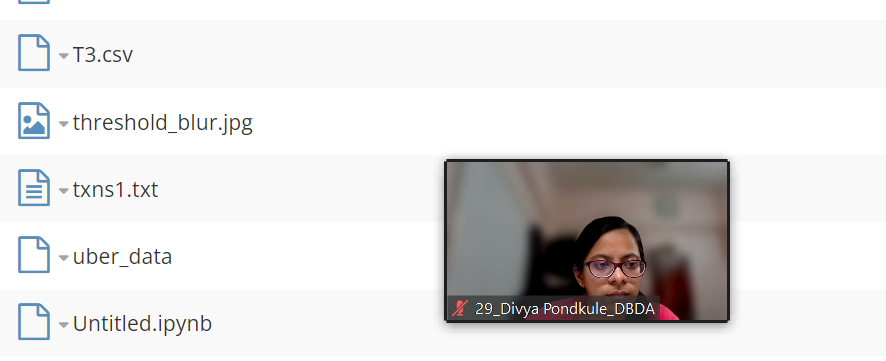
category

product

city

state

spendby

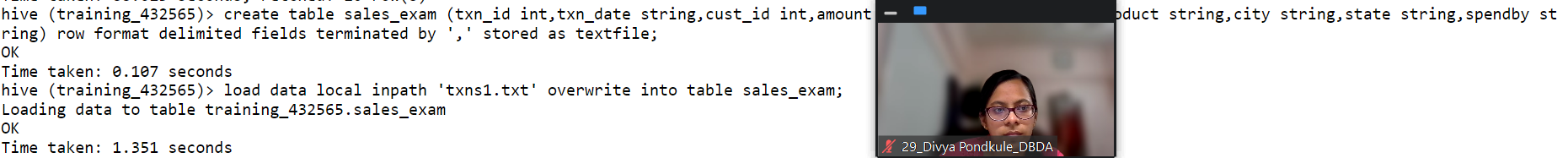


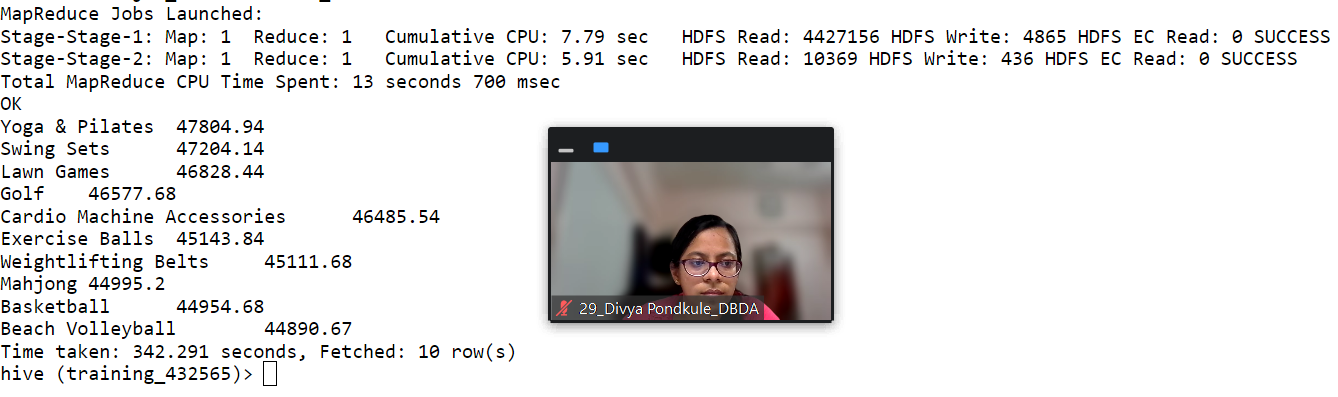
**2) Write a program to find the top 10 products sales wise**

create table sales\_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;

load data local inpath 'txns1.txt' overwrite into table sales\_exam;

select product,round(sum(amount),2) as Sales from sales\_exam group by product order by Sales desc limit 10;





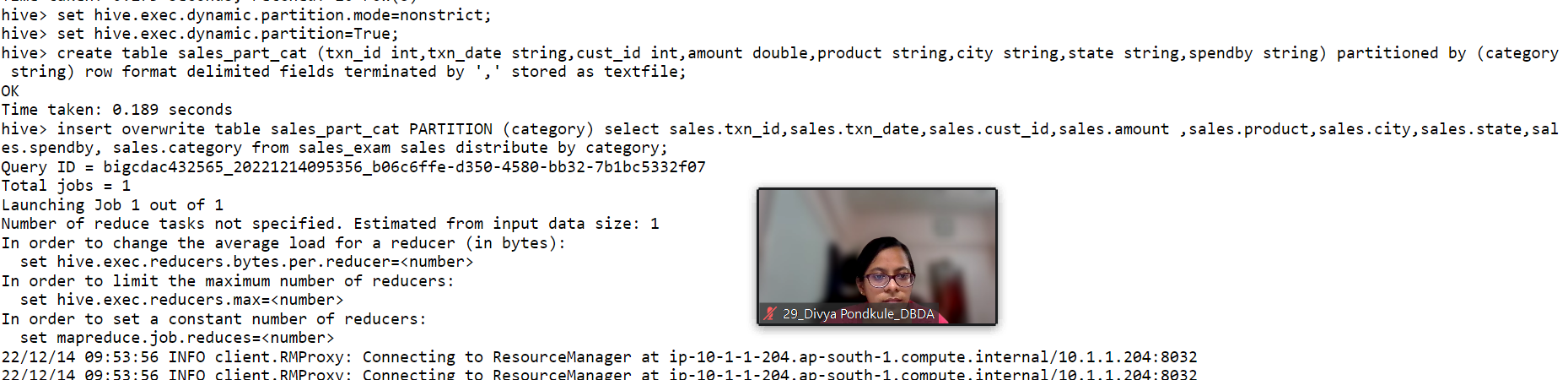
**3) Write a program to create partiioned table on category**

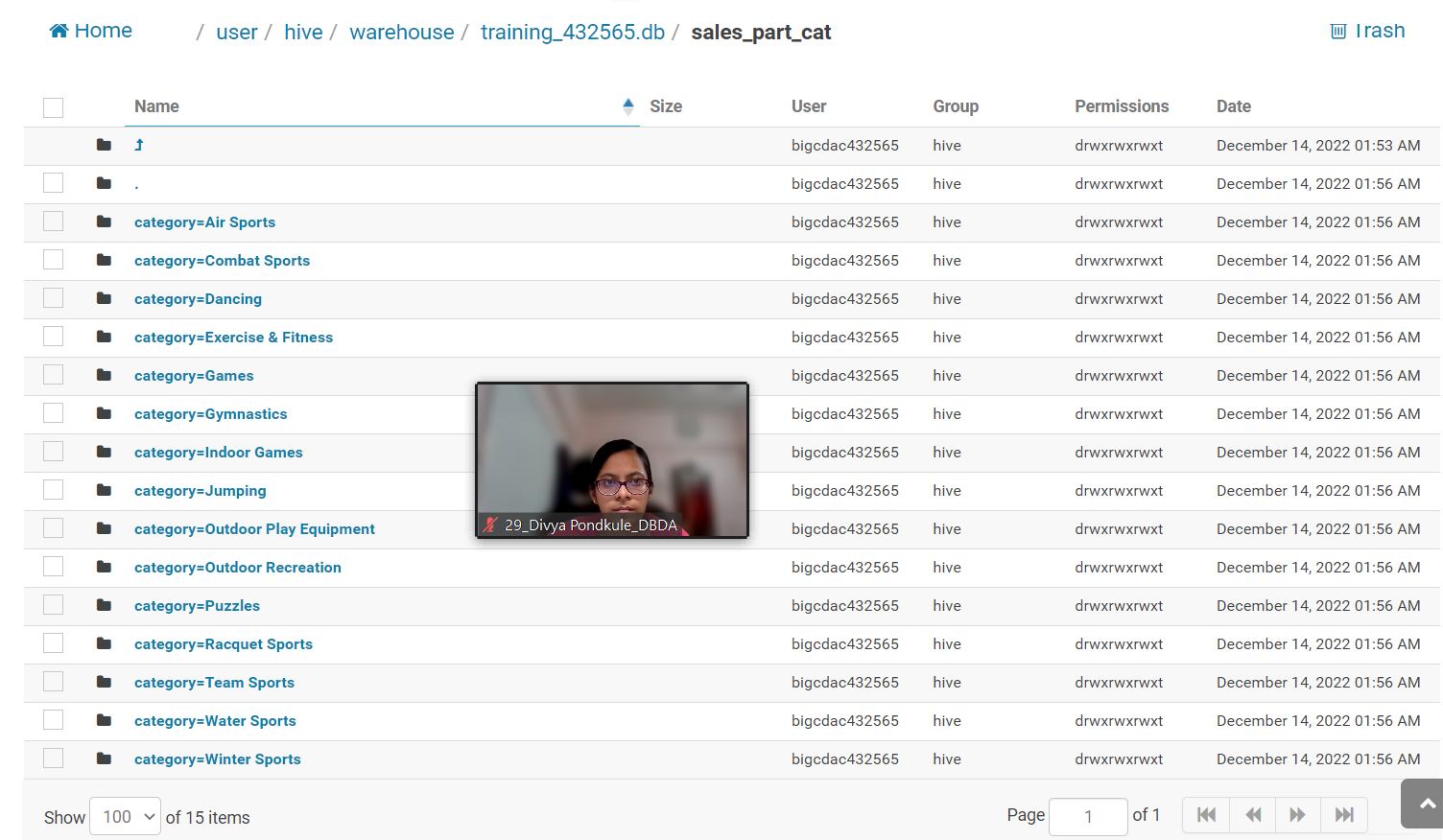
set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.dynamic.partition=True;

create table sales\_part\_cat (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;

insert overwrite table sales\_part\_cat PARTITION (category) select sales.txn\_id,sales.txn\_date,sales.cust\_id,sales.amount ,sales.product,sales.city,sales.state,sales.spendby, sales.category from sales\_exam sales distribute by category;





**QUESTION 3 [15 marks]**

**PySpark**

Please find the AIRLINES data set

Year

Quarter

Average revenue per seat

Total number of booked seats

**Solution**

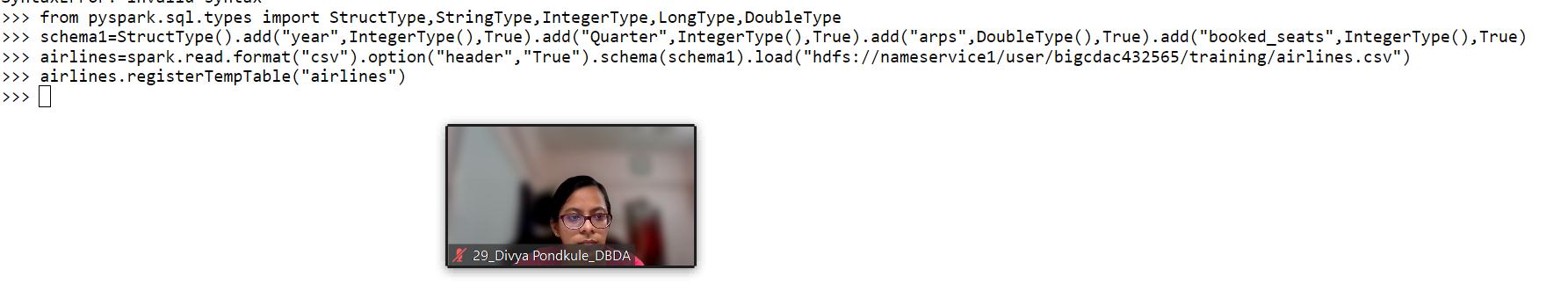
from pyspark.sql.types import StructType,StringType,IntegerType,LongType,DoubleType

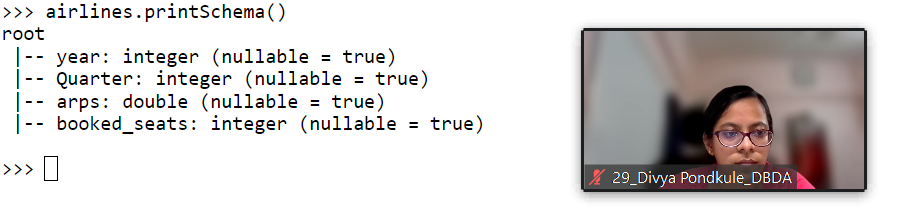
schema1=StructType().add("year",IntegerType(),True).add("Quarter",IntegerType(),True).add("arps",DoubleType(),True).add("booked\_seats",IntegerType(),True)

airlines=spark.read.format("csv").option("header","True").schema(schema1).load("hdfs://nameservice1/user/bigcdac432565/training/airlines.csv")

airlines.registerTempTable("airlines")

airlines.printSchema()



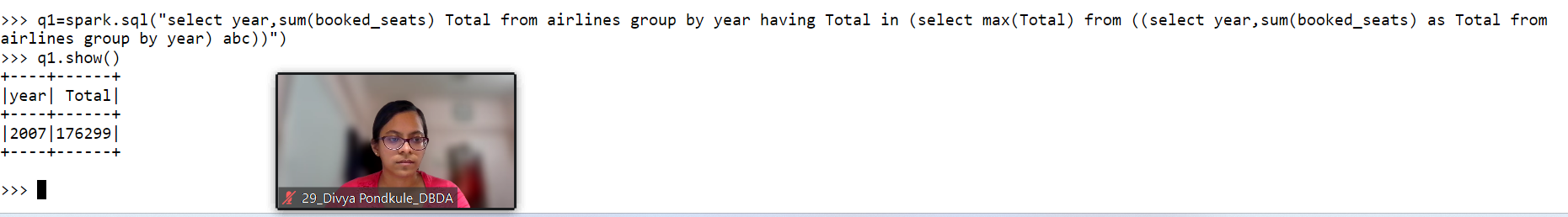


1. **What was the highest number of people travelled in which year?**

q1=spark.sql("select year,sum(booked\_seats) Total from airlines group by year having Total in (select max(Total) from ((select year,sum(booked\_seats) as Total from

airlines group by year) abc))")

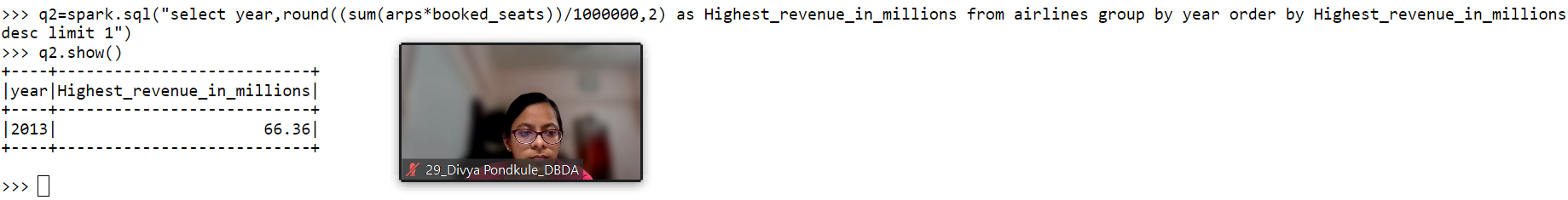
q1.show()

****

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

q2=spark.sql("select year,round((sum(arps\*booked\_seats))/1000000,2) as Highest\_revenue\_in\_millions from airlines group by year order by Highest\_revenue\_in\_millions desc limit 1")

q2.show()



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

q3=spark.sql("select year,quarter,round((sum(arps\*booked\_seats))/1000000,2) as Highest\_revenue\_in\_millions from airlines group by year,quarter order by Highest\_revenue\_in\_millions desc limit 1")

q3.show()

