**End Module Exam**

**Big Data Analytics**

**Map Reduce**

**Find all time High price for each stock**

**package** kh;

**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** AllTimeHigh1 {

**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(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);

}

}

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

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

Job job = Job.*getInstance*(conf, "High price for each stock");

job.setJarByClass(AllTimeHigh1.**class**);

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

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

job.setNumReduceTasks(1);

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

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

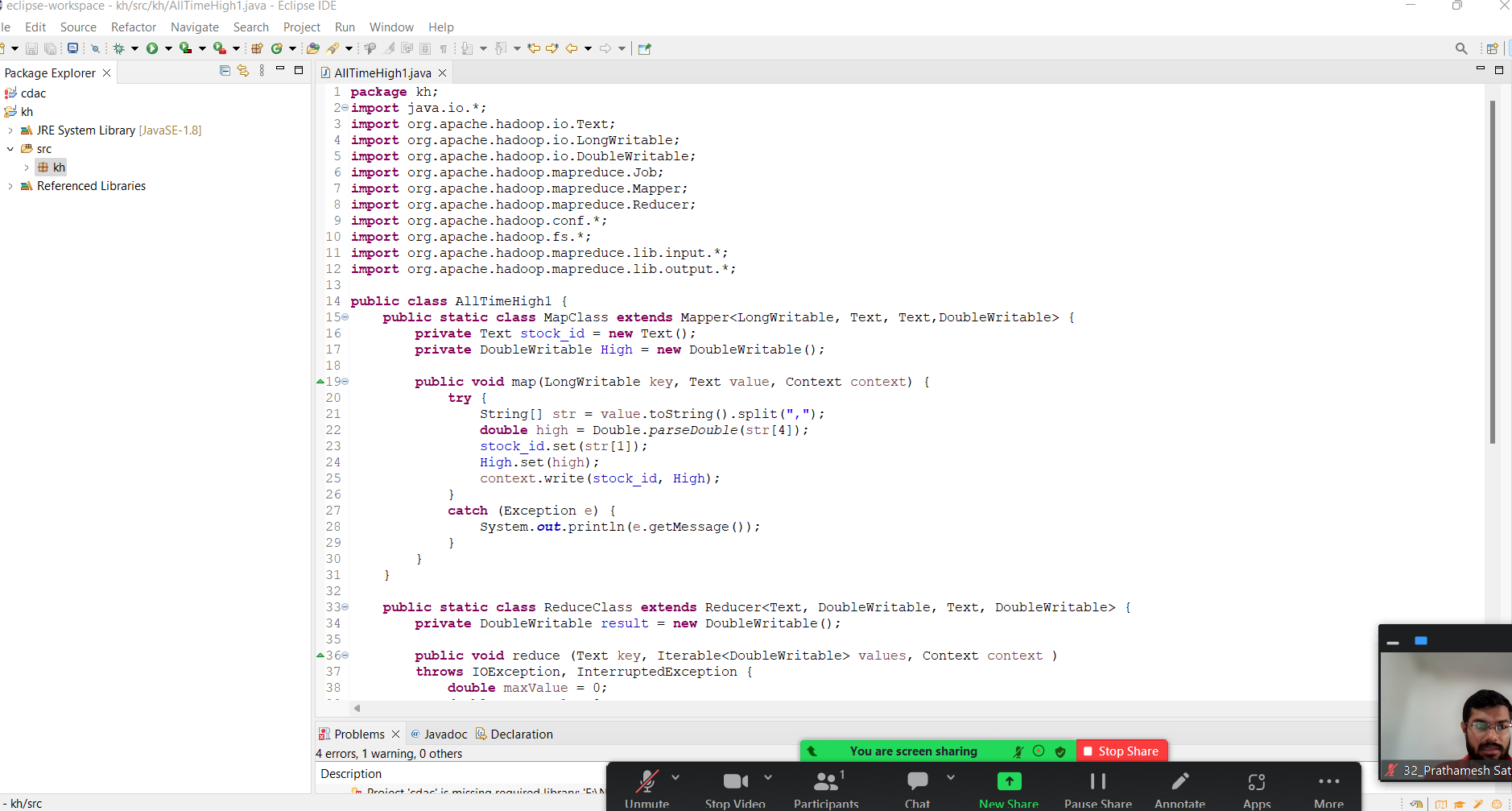
FileInputFormat.*addInputPath*(job, **new** Path(args[0]));

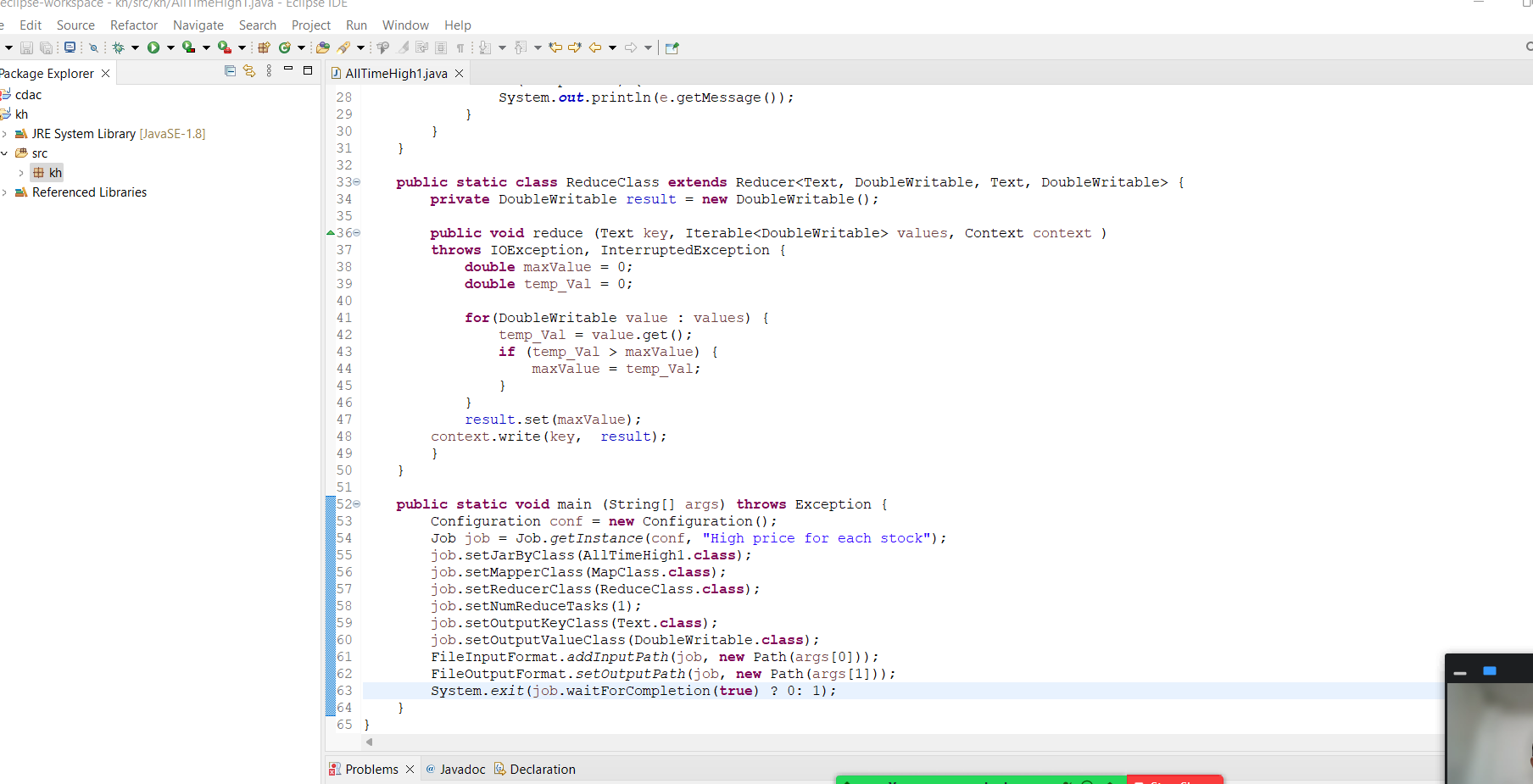
FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

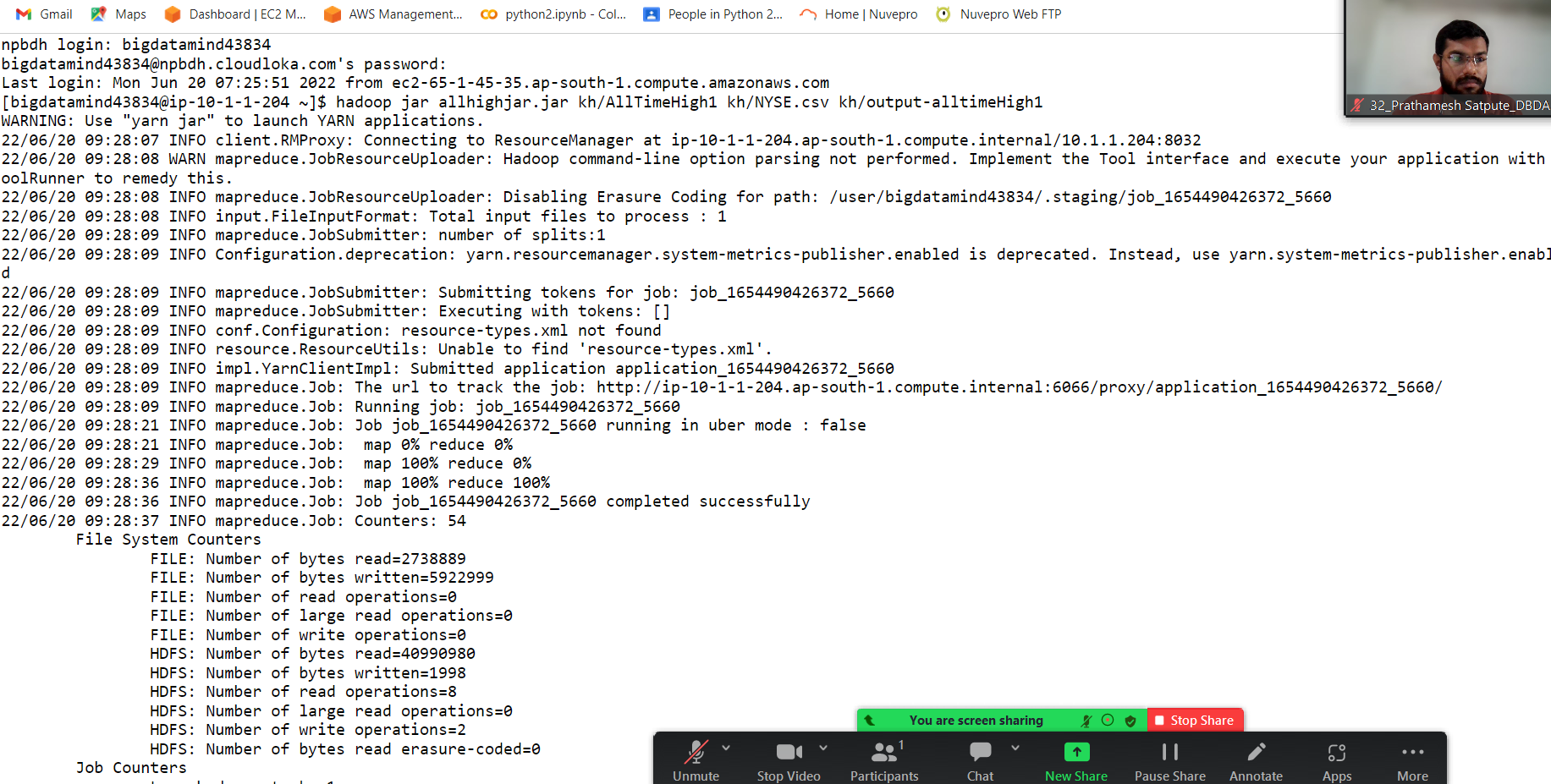
System.*exit*(job.waitForCompletion(**true**) ? 0: 1);

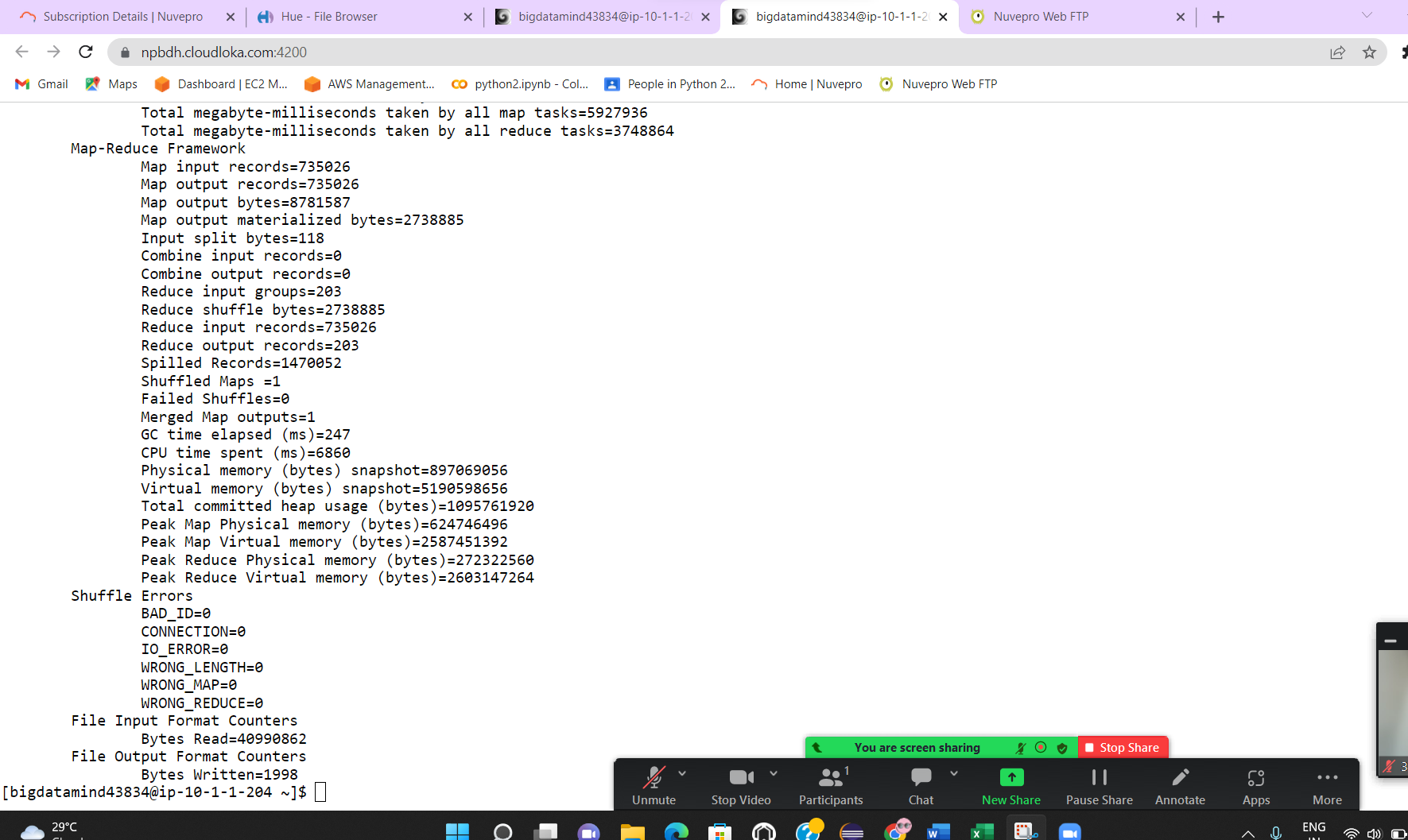
}

}









**Find all time High price for each stock**

AA 94.62

AAI 57.88

AAN 35.21

AAP 83.65

AAR 25.25

AAV 24.78

AB 94.94

ABA 27.94

ABB 33.39

ABC 84.35

ABD 28.58

ABG 30.06

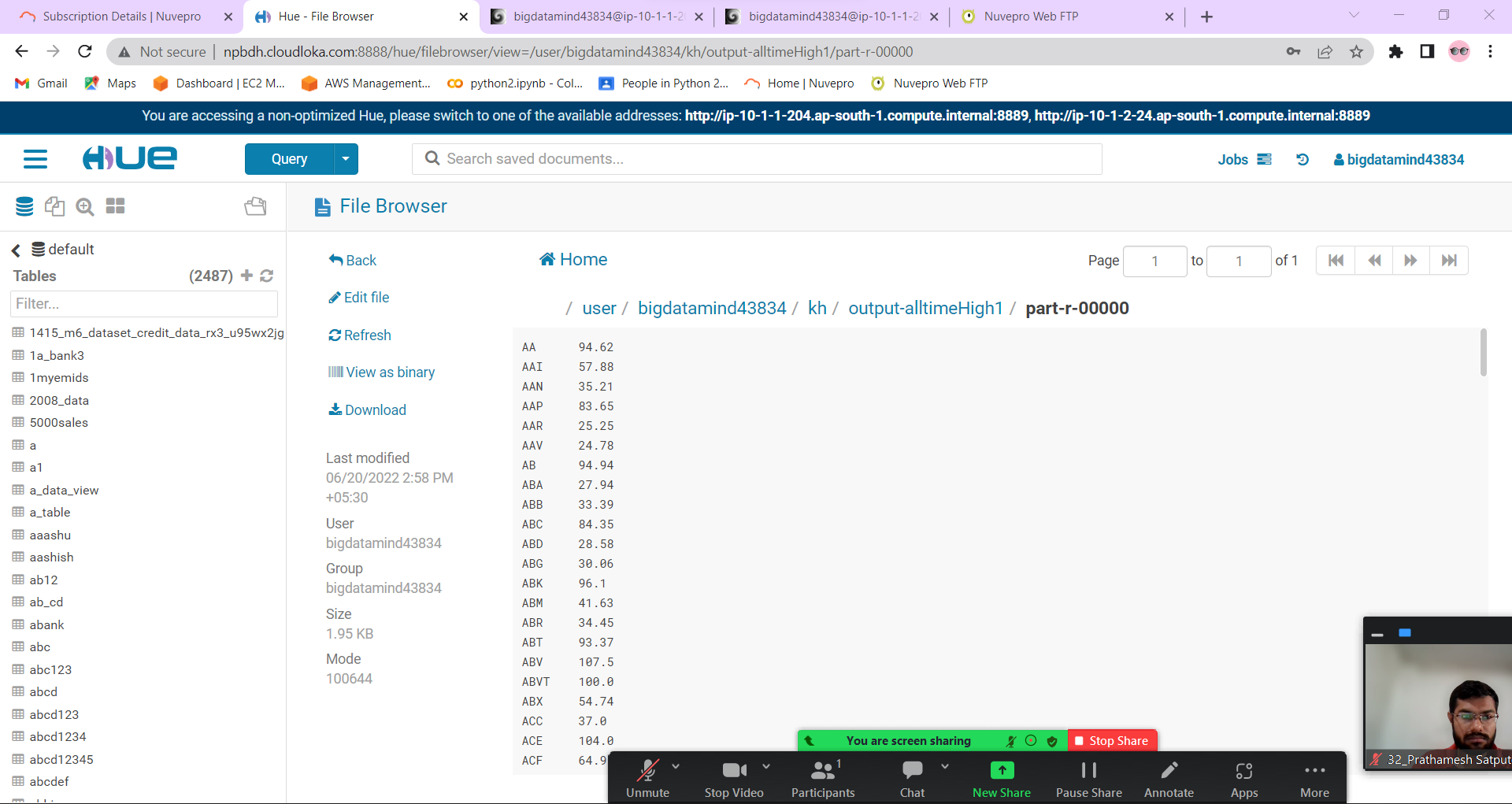
ABK 96.1

ABM 41.63

ABR 34.45

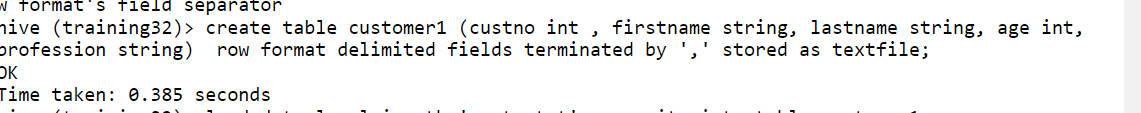
ABT 93.37

ABV 107.5



**Hive**

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

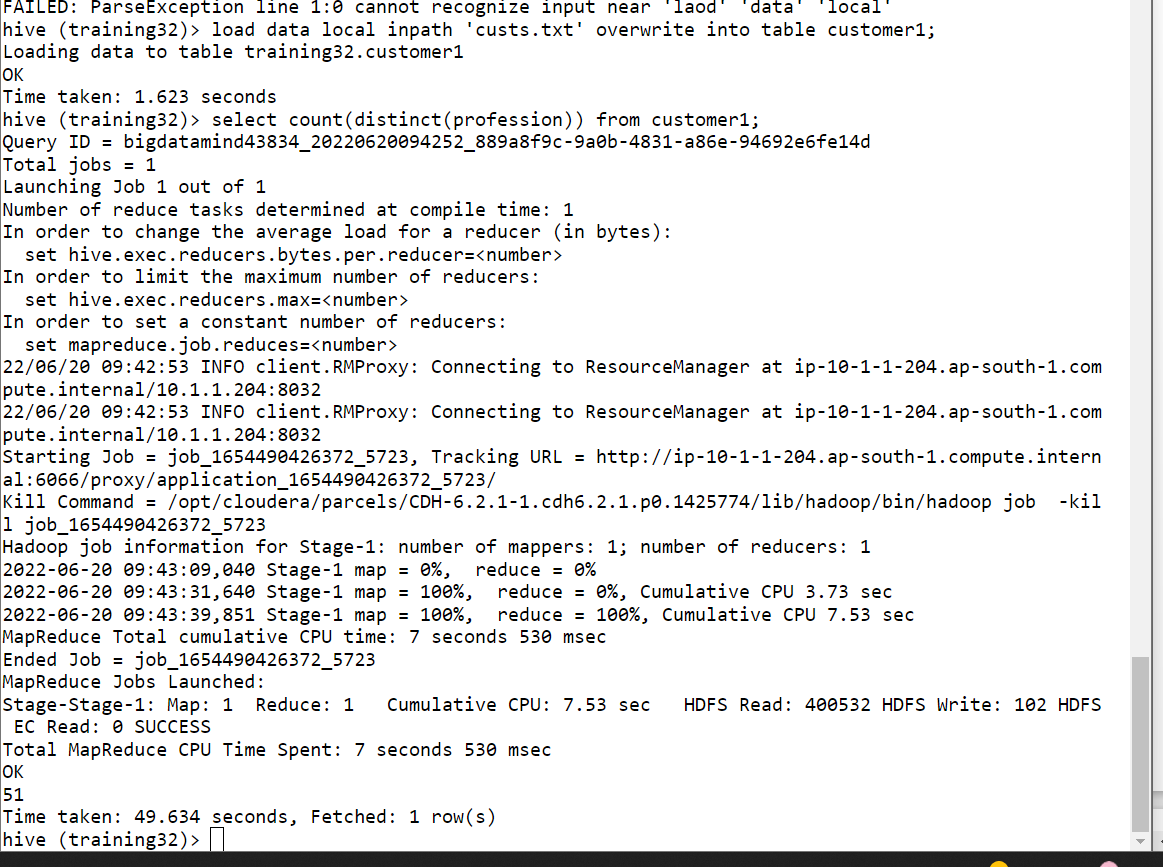


hive (training32)> create table customer1 (custno int , firstname string, lastname string, age int,

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

OK

Time taken: 0.385 seconds



hive (training32)> load data local inpath 'custs.txt' overwrite into table customer1;

Loading data to table training32.customer1

OK

Time taken: 1.623 seconds

hive (training32)> select count(distinct(profession)) from customer1;

Total MapReduce CPU Time Spent: 7 seconds 530 msec

OK

51

Time taken: 49.634 seconds, Fetched: 1 row(s)

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

hive (training32)> create table txnrecords(txnno int,txndate string, custno int, amount double, cate

gory string, product string, city string, state string, spendby string) row format delimited fields

terminated by ',' stored as textfile;

OK

Time taken: 0.102 seconds

hive (training32)> load data local inpath 'txns1.txt' overwrite into table txnrecords;

Loading data to table training32.txnrecords

OK

Time taken: 0.821 seconds

hive (training32)> select product, sum(amount) as sales from txnrecords group by product order by sa

les limit 10;

Air Suits 19371.570000000007

Mechanical Puzzles 20381.669999999984

Disc Golf 33944.02000000002

Surfing 35069.079999999965

Tetherball 35611.92999999999

Downhill Skiing 36870.35999999996

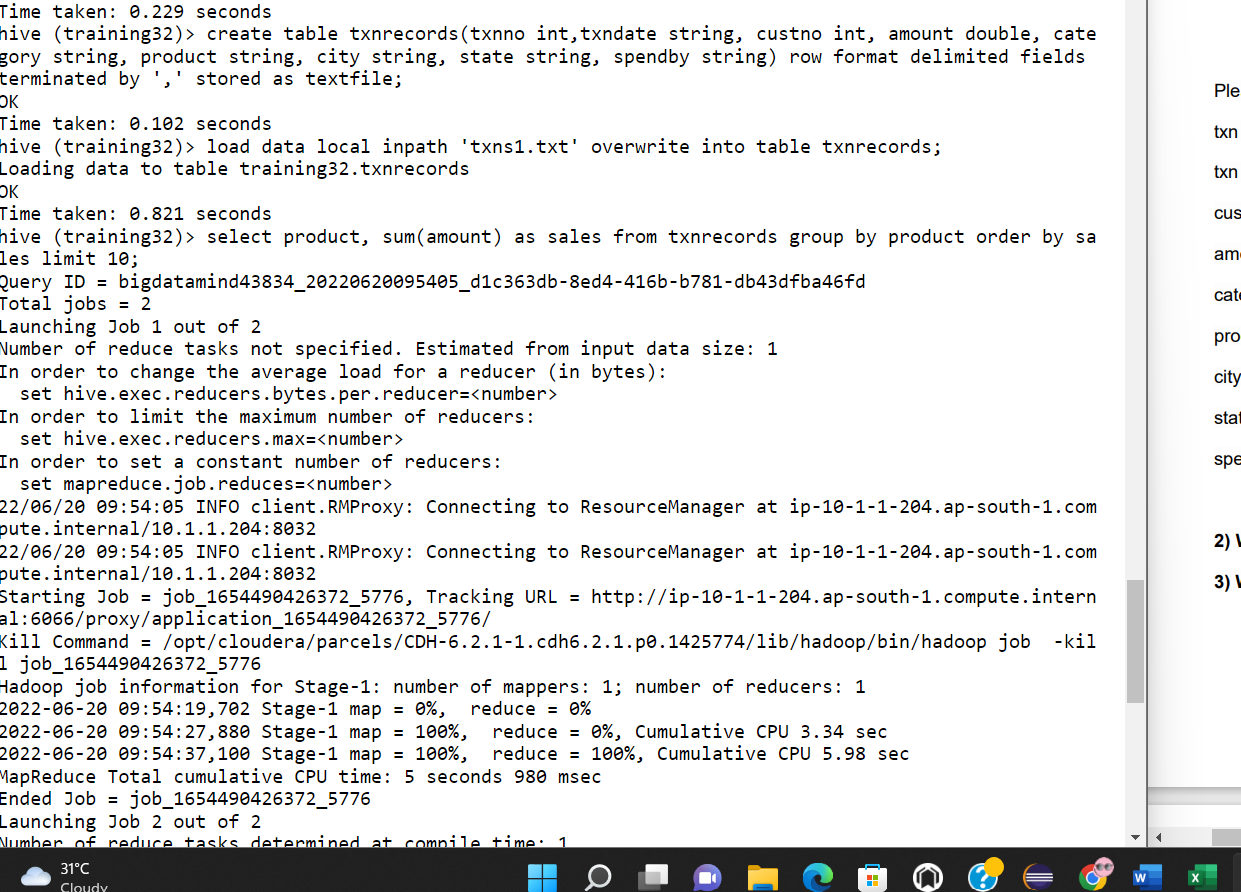
Cricket 37061.58000000001

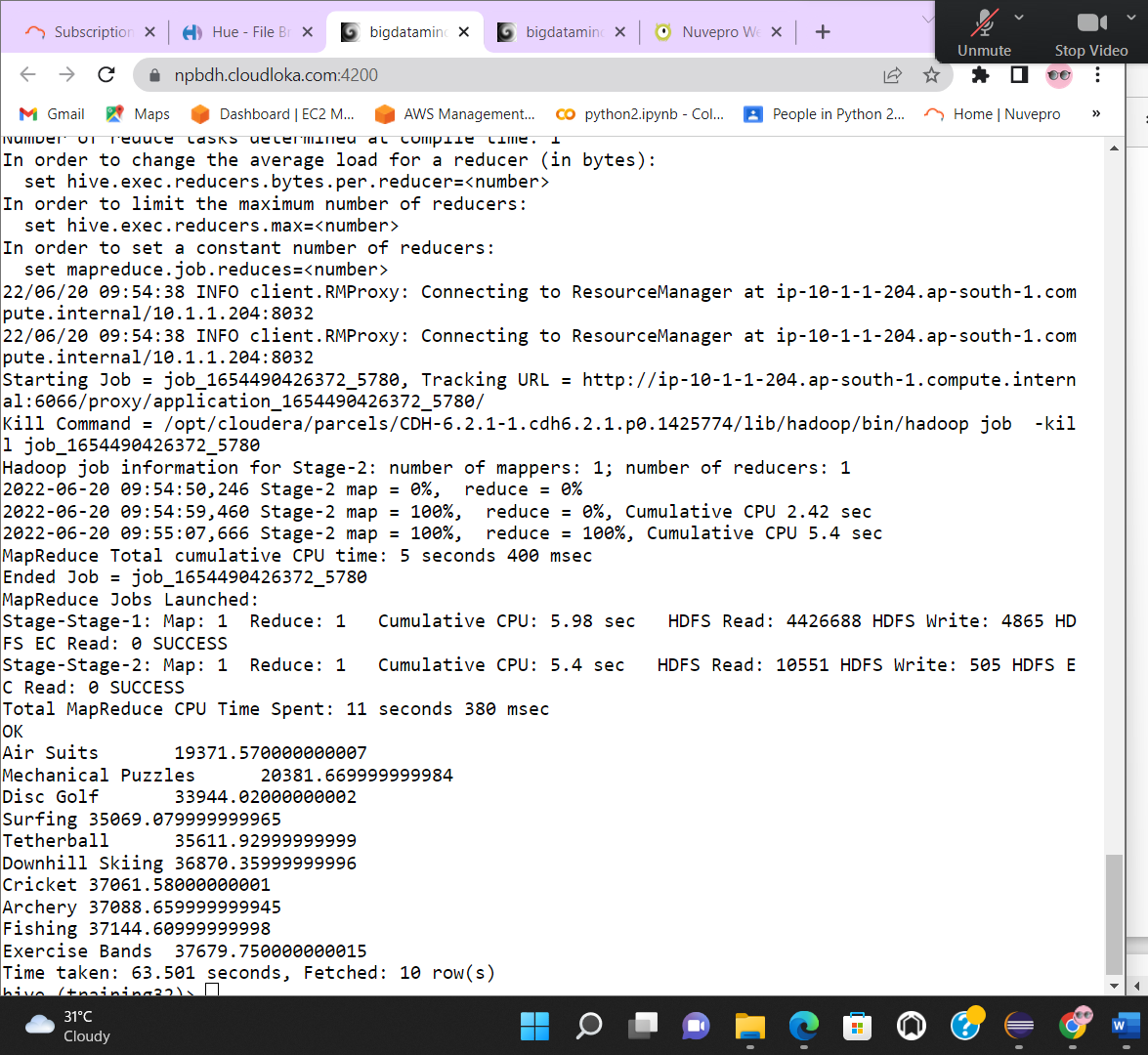
Archery 37088.659999999945

Fishing 37144.60999999998

Exercise Bands 37679.750000000015

Time taken: 63.501 seconds, Fetched: 10 row(s)





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

hive> create table txnrecordsByCategoryPart(txnno int,txndate string, custno int, amount double, pro

duct string, city string, state string, spendby string) partitioned by (category string) row format

delimited fields terminated by ',' stored as textfile;

OK

Time taken: 2.025 seconds

**Spark**

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

>>> airrdd = sc.textFile("/user/bigdatamind43834/cdac/airlines.csv")

>>> airrdd1 = airrdd.map(lambda a : a.encode("ascii","ignore"))

>>> airrdd1.count()

85

>>> header = airrdd1.first()

>>> new = airrdd1.filter(lambda a : a != header)

>>> for i in new.take(2):

... print(i)

...

1995,1,296.9,46561

1995,2,296.8,37443

>>>

>>> new.count()

84

>>> new111 = new.map(lambda a : a.split(","))

>>> for i in new111.take(2):

... print(i)

...

['1995', '1', '296.9', '46561']

['1995', '2', '296.8', '37443']

>>>

>>> max = new111.map(lambda a : (a[0], int(a[3])))

>>> maxpass = max.reduceByKey(lambda a,b : a+b)

>>> maxpass.count()

21

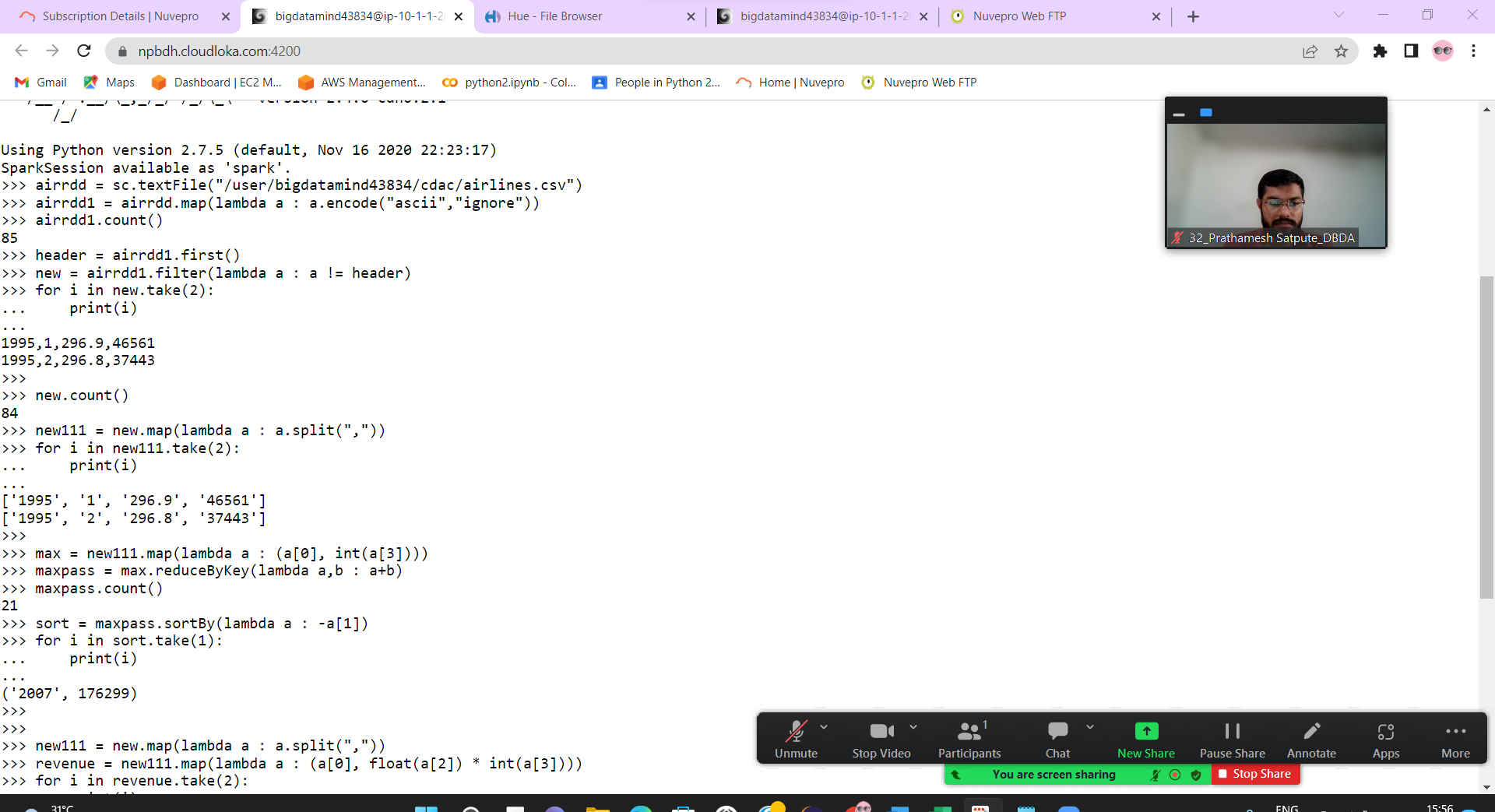
>>> sort = maxpass.sortBy(lambda a : -a[1])

>>> for i in sort.take(1):

... print(i)

...

('2007', 176299)



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

>>> new111 = new.map(lambda a : a.split(","))

>>> revenue = new111.map(lambda a : (a[0], float(a[2]) \* int(a[3])))

>>> for i in revenue.take(2):

... print(i)

...

('1995', 13823960.899999999)

('1995', 11113082.4)

>>>

>>> reduce = revenue.reduceByKey(lambda a,b : a+b)

>>> sort11 = reduce.sortBy(lambda a : -a[1])

>>> for i in sort11.take(1):

... print(i)

...

('2013', 66363208.71)

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

>>> new111 = new.map(lambda a : a.split(","))

>>> qtr = new111.map(lambda a : (a[0] + " "+a[1], float(a[2])\* int(a[3])))

>>> aaa = qtr.reduceByKey(lambda a,b : a+b)

>>> sortqtr = aaa.sortBy(lambda a : -a[1])

>>> for i in sortqtr.take(1):

... print(i)

...

('2014 4', 18819408.48)

