**MapReduce**

**Problem Statement**

**Here, we have chosen the stock market dataset on which we have performed map-reduce**

**operations. Following is the structure of the data. Kindly Find the solutions to the questions**

**below.**

**Data Structure**

**1. Exchange Name**

**2 Stock symbol**

**3. Transaction date**

**4. Opening price of the stock**

**5. Intra day high price of the stock**

**6. Intra day low price of the stock**

**7. Closing price of the stock**

**8. Total Volume of the stock on the particular day**

**9. Adjustment Closing price of the stock**

**Eclipse Code :**

import java.io.\*;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.LongWritable;

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 StockVolume

{

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

{

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

{

try

{

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

long vol = Long.parseLong(str[7]);

context.write(new Text(str[1]),new LongWritable(vol));

}

catch(Exception e)

{

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

}

}

}

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

{

private LongWritable result = new LongWritable();

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

{

long sum = 0;

for (LongWritable val : values)

{

sum += val.get();

}

result.set(sum);

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, "Volume Count");

job.setJarByClass(StockVolume.class);

job.setMapperClass(MapClass.class);

//job.setCombinerClass(ReduceClass.class);

job.setReducerClass(ReduceClass.class);

job.setNumReduceTasks(1);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(LongWritable.class);

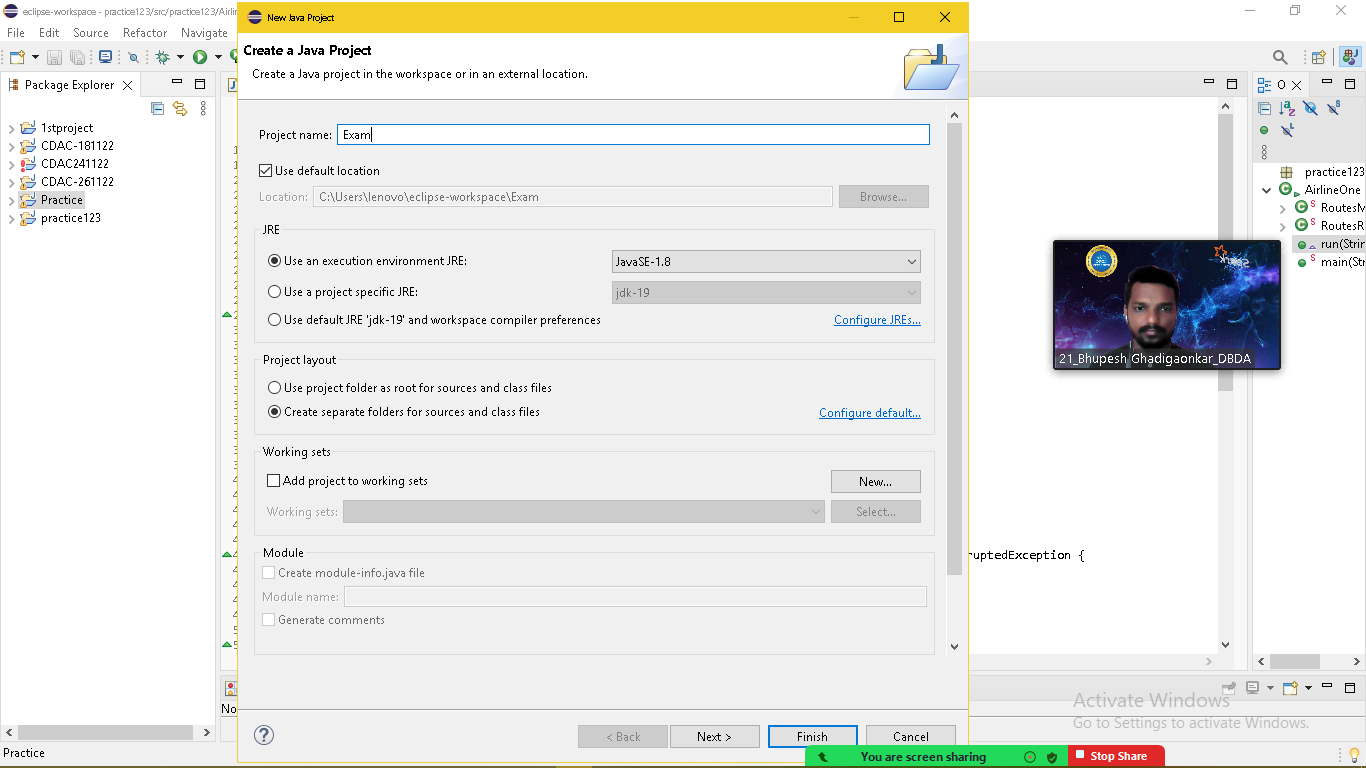
FileInputFormat.addInputPath(job, new Path(args[0]));

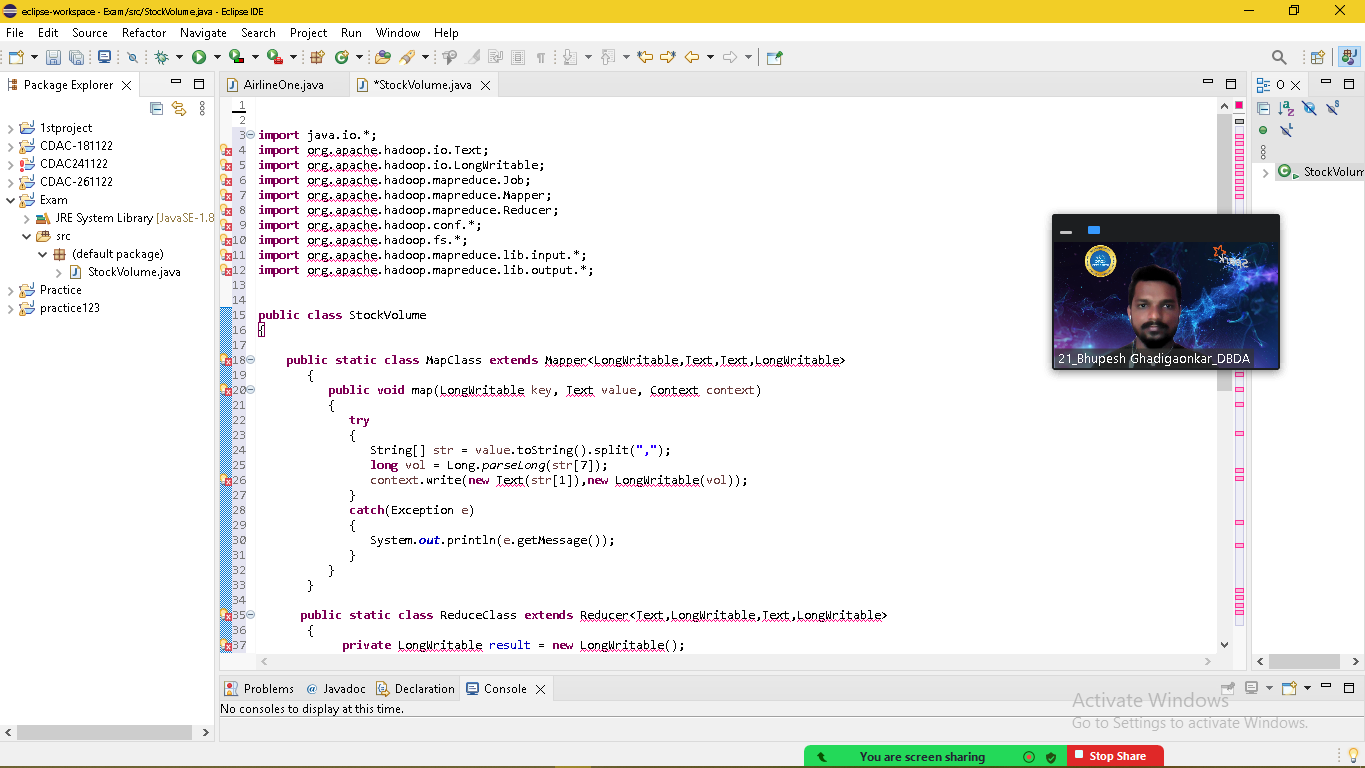
FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

}

}





$ ls

AFINN.txt counter.txt D02 **eclipse.gz** Message2.txt odi\_batting.csv uber\_data

AirLineData.csv custs(1).txt D11 Employees.txt **myjar.jar** routes.dat

airlines.csv custs\_add D12 Final\_airlines **mysql-connector-java-5.1.47-bin.jar** salary.txt

airports\_mod.dat custs.txt data1.txt mapred-hduser-historyserver-ubuntu.log nyse.csv sample.txt

Book1.csv D01 desig.txt Message1.txt NYSE.csv

$ jar tvf myjar.jar

25 Wed Dec 14 15:57:04 UTC 2022 META-INF/MANIFEST.MF

380 Wed Dec 14 15:53:42 UTC 2022 .project

2408 Wed Dec 14 15:56:44 UTC 2022 StockVolume$MapClass.class

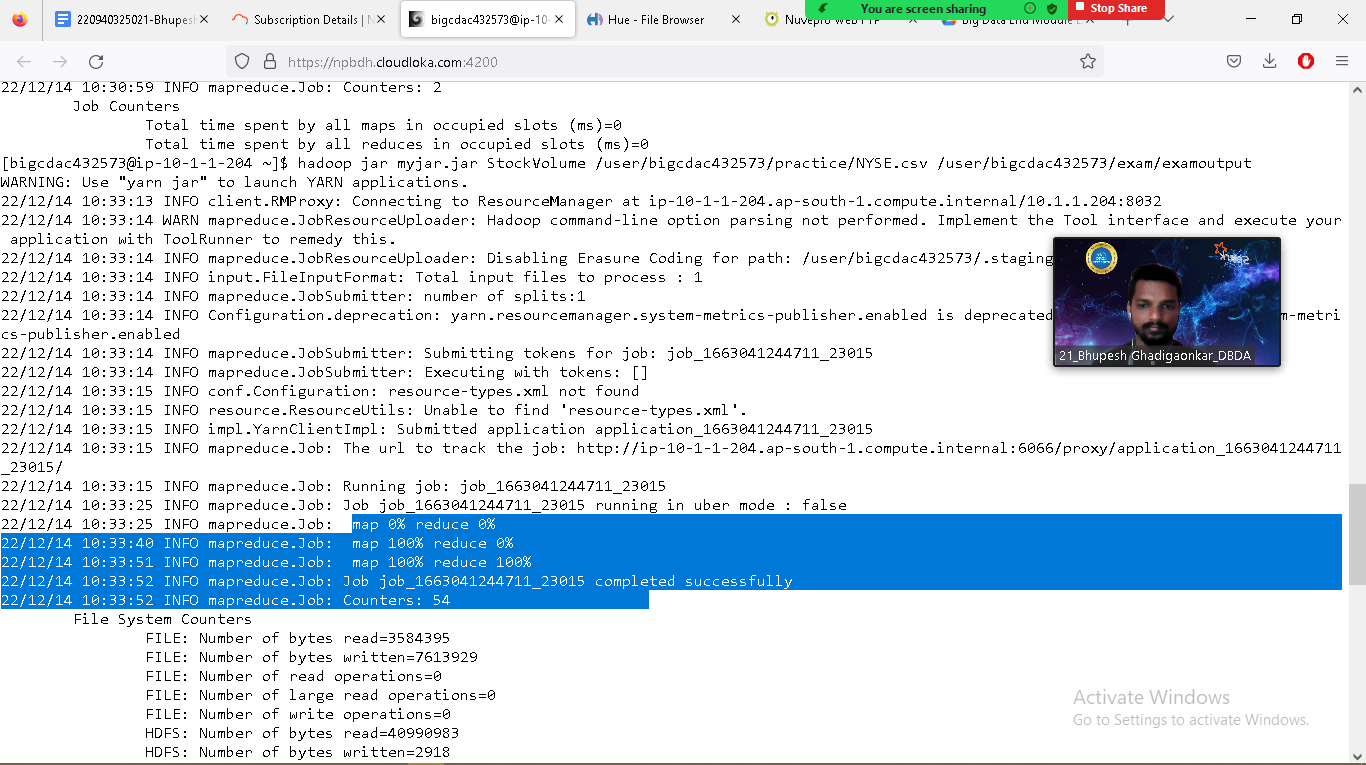
2349 Wed Dec 14 15:56:44 UTC 2022 StockVolume$ReduceClass.class

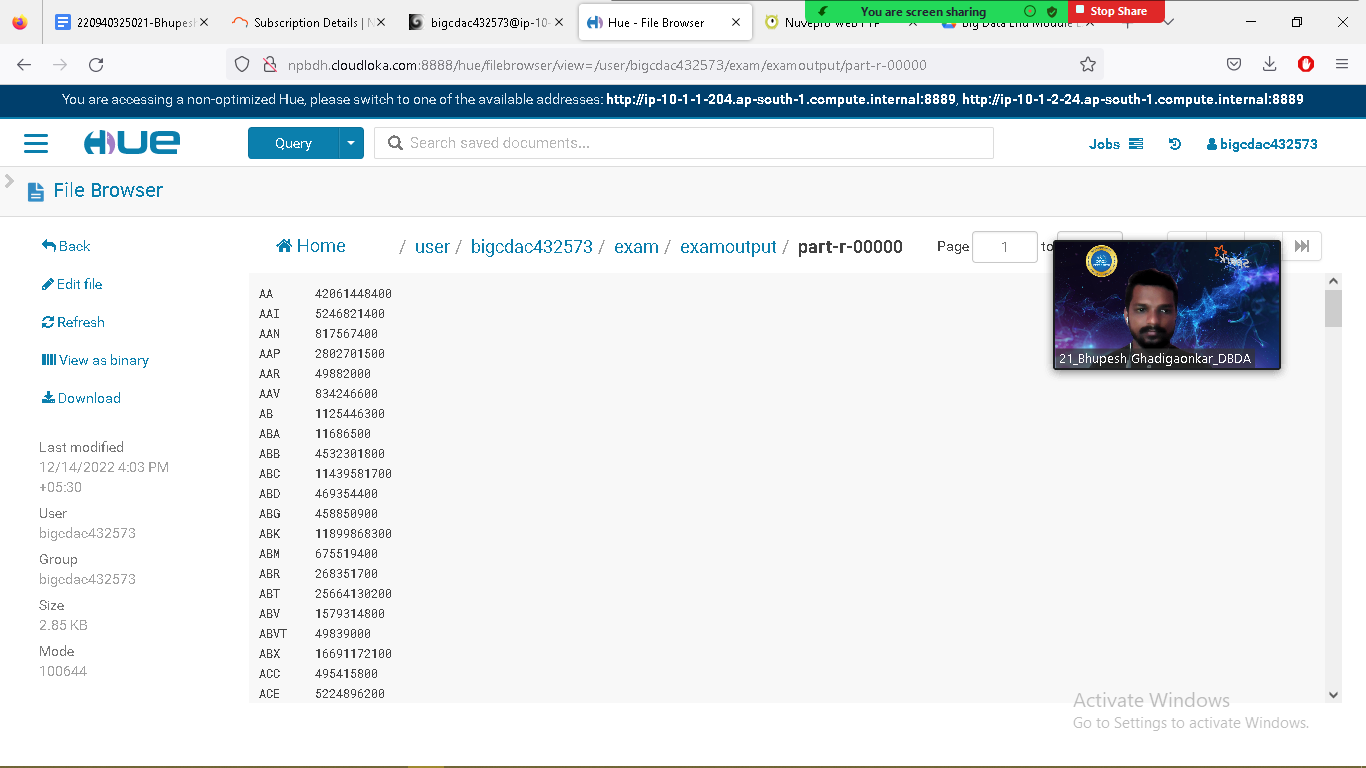
1697 Wed Dec 14 15:56:44 UTC 2022 StockVolume.class

2345 Wed Dec 14 15:56:44 UTC 2022 StockVolume.java

646 Wed Dec 14 15:56:04 UTC 2022 .classpath

$ hadoop jar myjar.jar StockVolume /user/bigcdac432573/practice/NYSE.csv /user/bigcdac432573/exam/exmoutput





**Question 2 : Find all time High price for each stock**

**[**

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 DoubleWritable(max));

}

}

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.minsize", "134217728");

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);

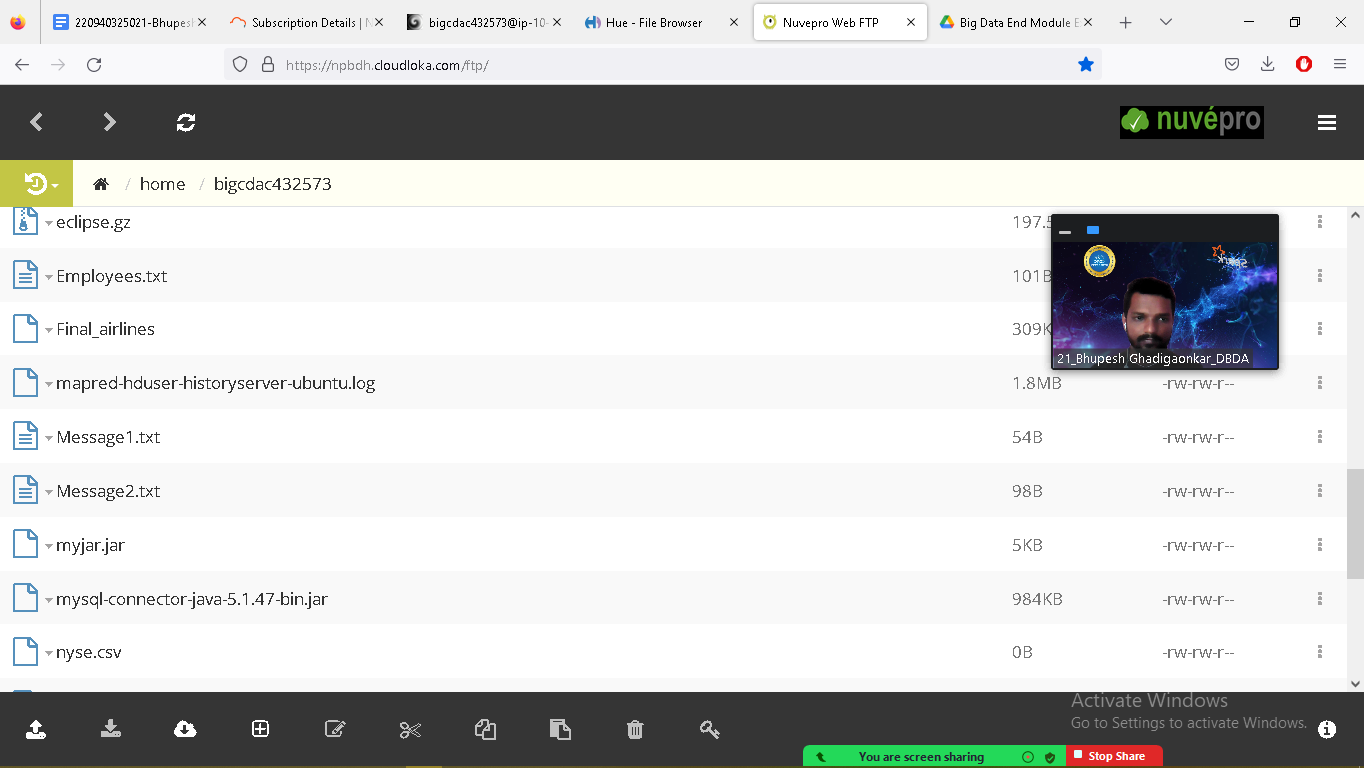
}

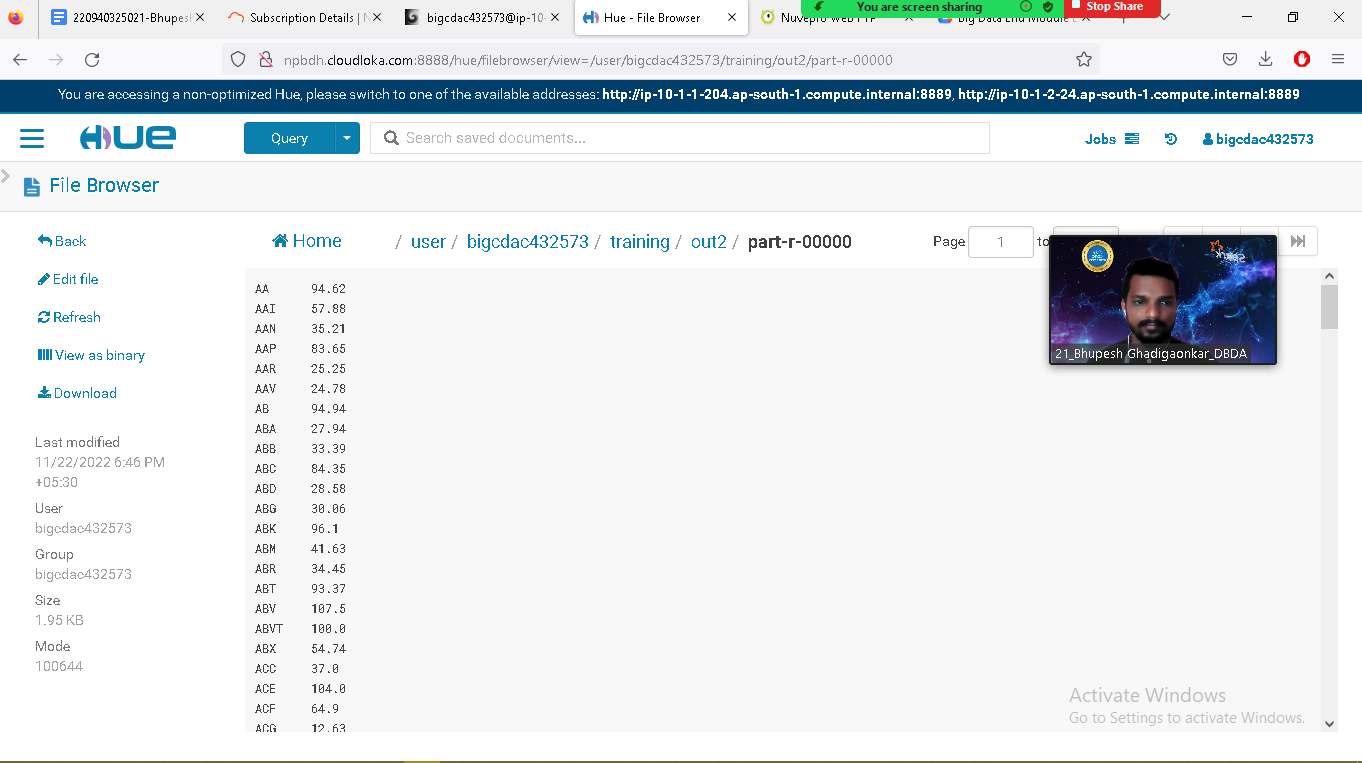
}

$ ls

$ jar tvf myjar.jar

$ hadoop jar myjar.jar AllTimeHigh /user/bigcdac432573/practice/NYSE.csv /user/bigcdac432573/training/out2





**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.**

hadoop fs -put custs.txt exam

hive> use practice432573;

OK

Time taken: 0.229 seconds

hive> show tables;

OK

hive> create table customer1(custno INT, firstname STRING, lastname STRING,age INT, profession STRING) row format delimited fields terminated by ','

stored as textfile;

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

Loading data to table practice432573.customer1

hive> select \* from customer1 limit 5;

OK

4000001 Kristina Chung 55 Pilot

4000002 Paige Chen 74 Teacher

4000003 Sherri Melton 34 Firefighter

4000004 Gretchen Hill 66 Computer hardware engineer

4000005 Karen Puckett 74 Lawyer

Time taken: 0.312 seconds, Fetched: 5 row(s)

hive> select profession, count(\*) from customer1 group by profession;

**OUTPUT :**

OK

Accountant 199

Actor 202

Agricultural and food scientist 195

Architect 203

Artist 175

Athlete 196

Automotive mechanic 193

Carpenter 181

Chemist 209

Childcare worker 207

Civil engineer 193

Coach 201

Computer hardware engineer 204

Computer software engineer 216

Computer support specialist 222

Dancer 185

Designer 205

Doctor 197

Economist 189

Electrical engineer 192

Electrician 194

Engineering technician 204

Environmental scientist 176

Farmer 201

Financial analyst 198

Firefighter 217

Human resources assistant 212

Judge 196

Lawyer 212

Librarian 218

Loan officer 221

Musician 205

Nurse 192

Pharmacist 213

Photographer 222

Physicist 201

Pilot 211

Police officer 210

Politician 228

Psychologist 194

Real estate agent 191

Recreation and fitness worker 210

Reporter 200

Secretary 200

Social Worker 1

Social worker 212

Statistician 196

Teacher 204

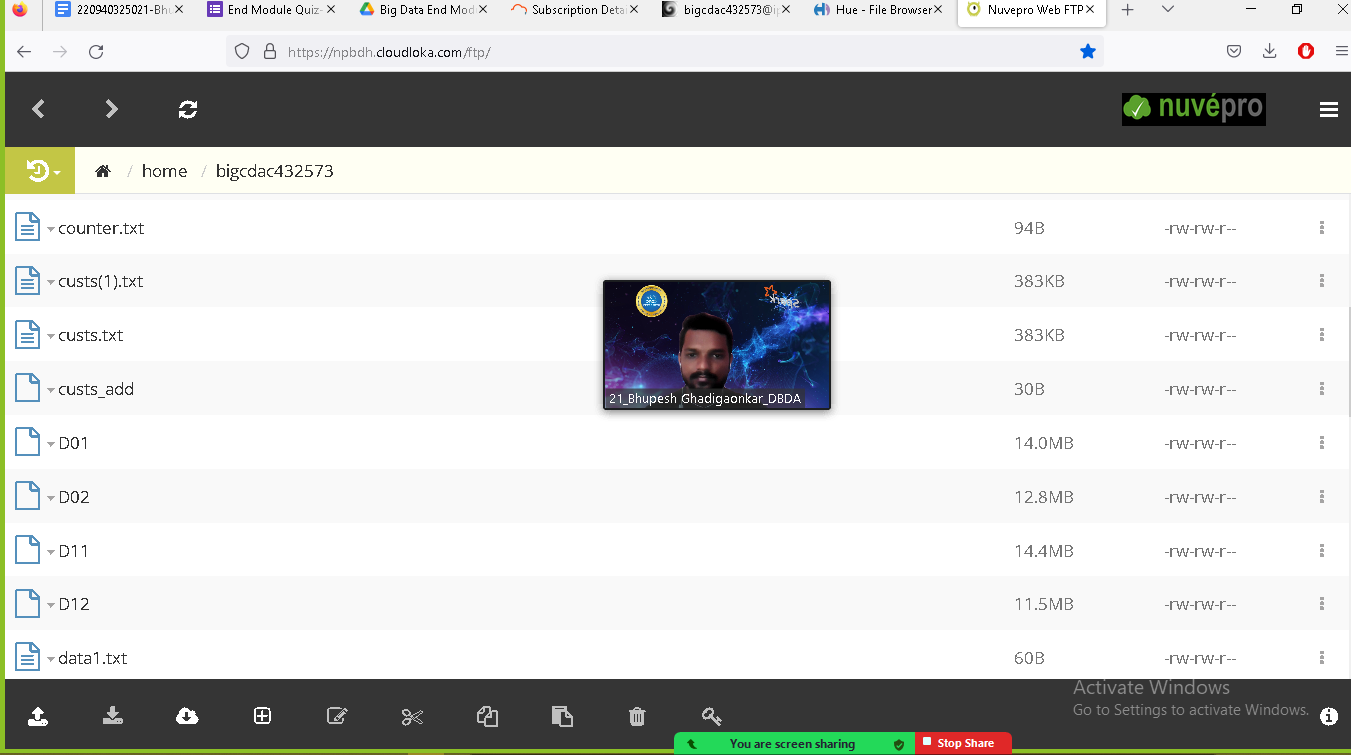
Therapist 187

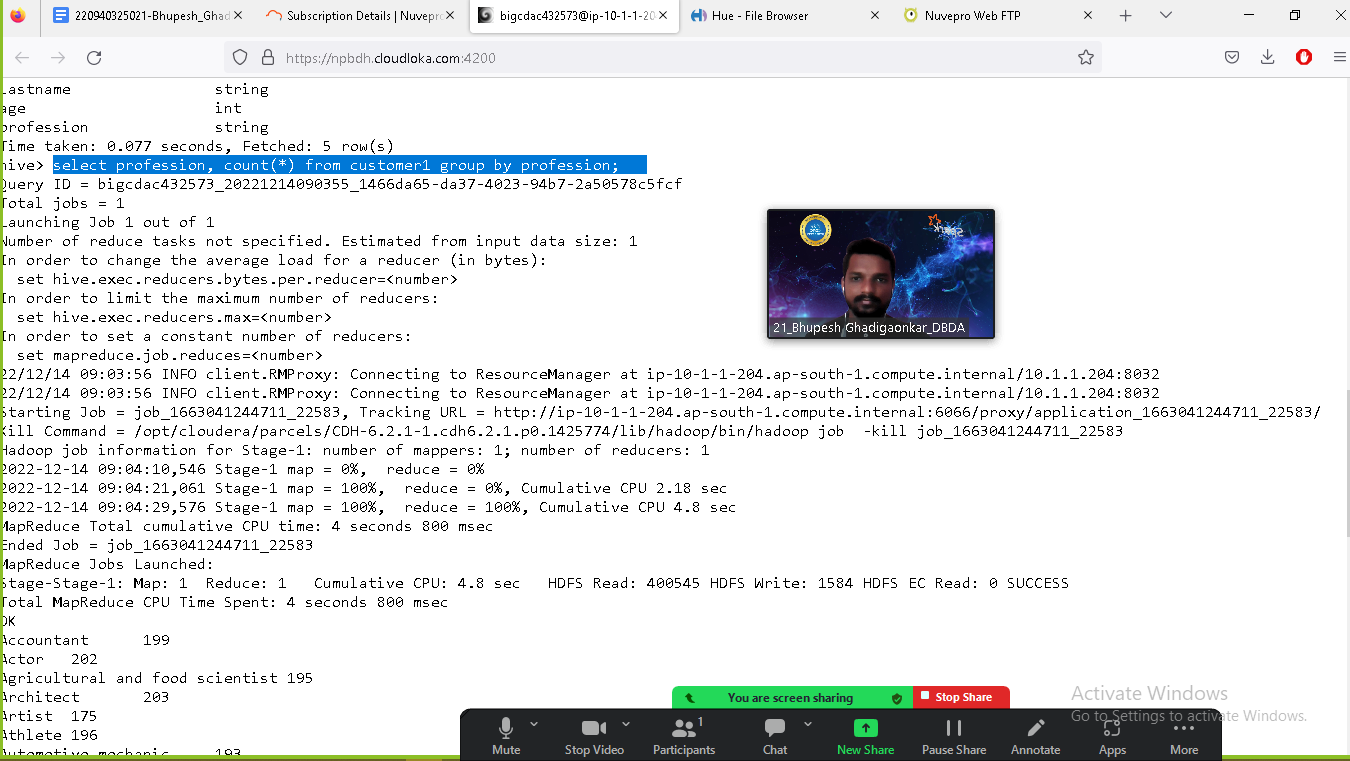
Veterinarian 208

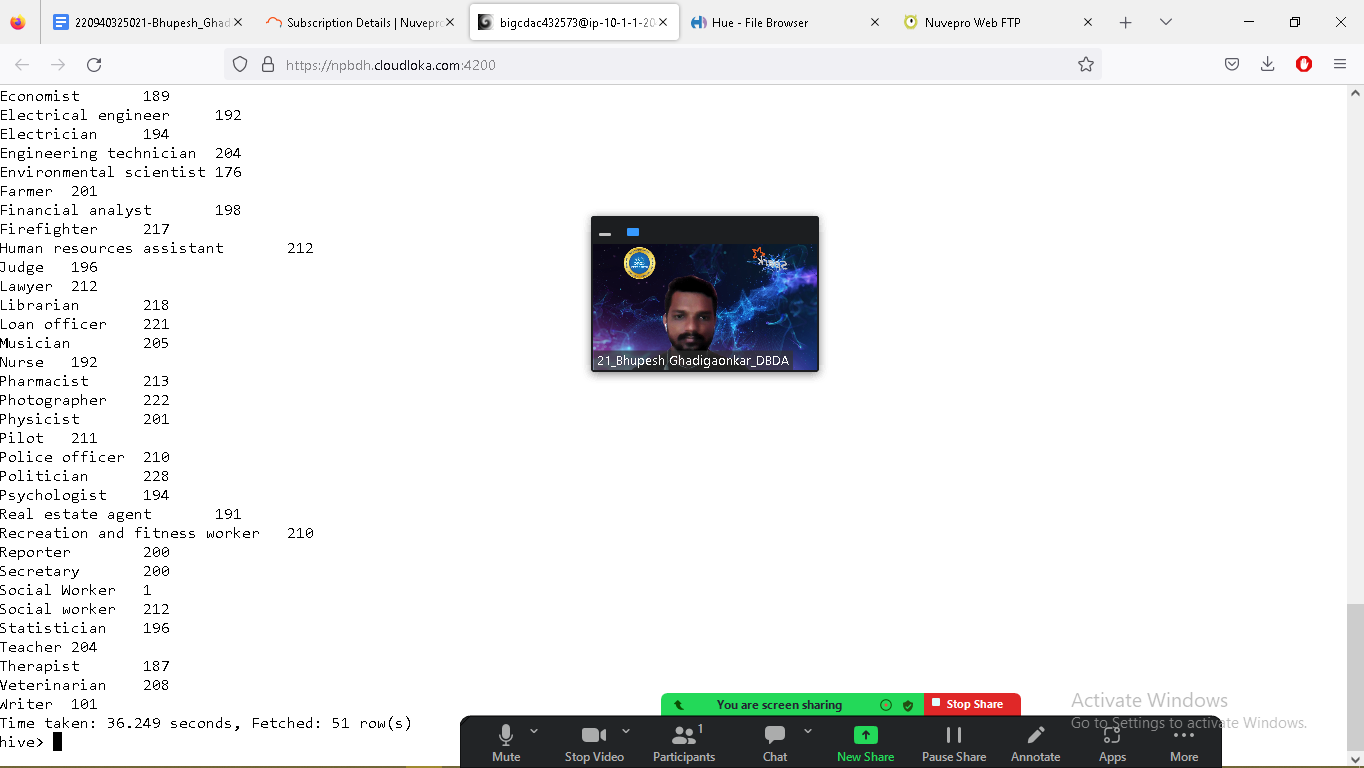
Writer 101

Time taken: 36.249 seconds, Fetched: 51 row(s)

hive>







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**

]$ hadoop fs -put txns1.txt exam

$ hive

hive> use practice432573;

hive> create table txnrecords1(txnno INT, txndate STRING, custno INT, amount DOUBLE, category STRING, product STRING, city STRING, state STRING, spe

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

hive> load data local inpath 'txns1.txt' overwrite into table txnrecords1;

hive> desc txnrecords1;

hive> select product, sum(spendby) as total from txnrecords1 group by product order by total desc limit 10;

**OUTPUT :**

**Yoga & Pilates 47804.93999999993**

**Swing Sets 47204.13999999999**

**Lawn Games 46828.44**

**Golf 46577.67999999999**

**Cardio Machine Accessories 46485.540000000045**

**Exercise Balls 45143.84**

**Weightlifting Belts 45111.67999999996**

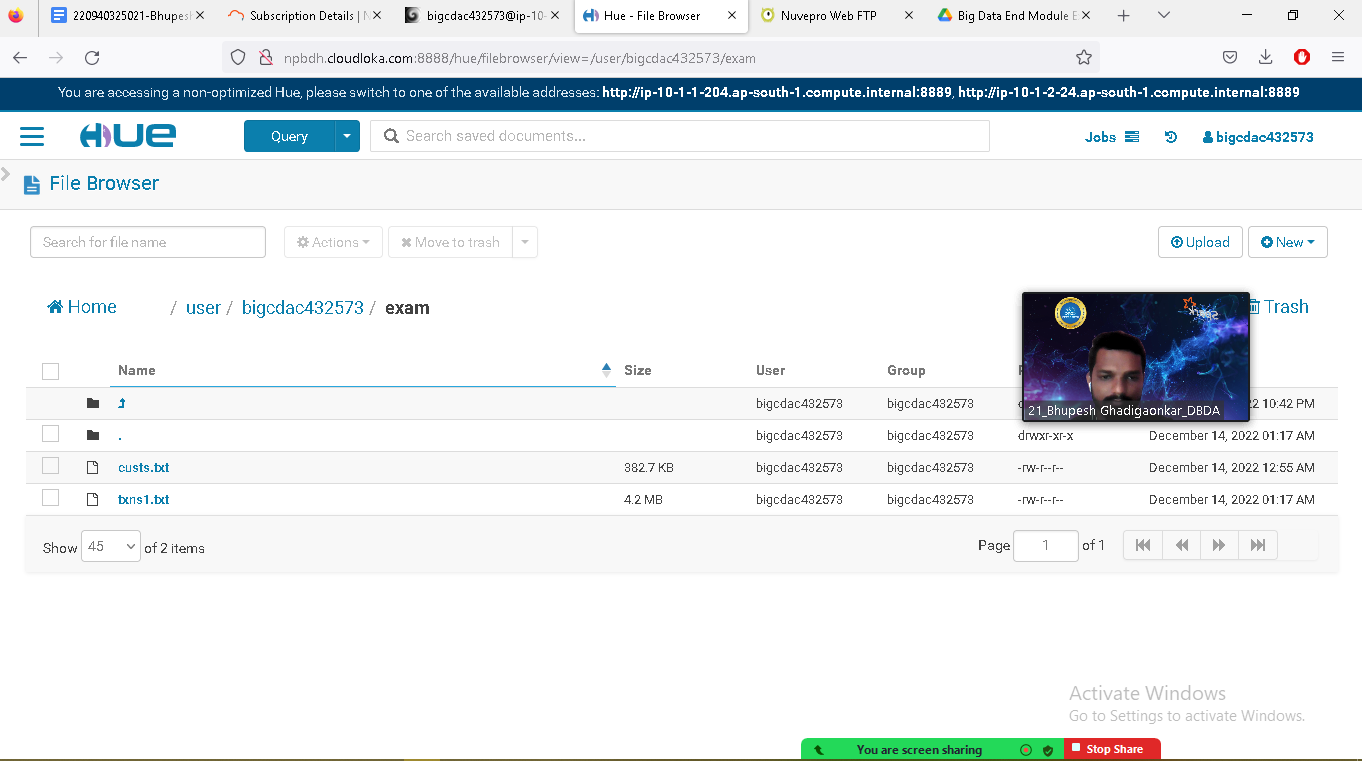
**Mahjong 44995.19999999999**

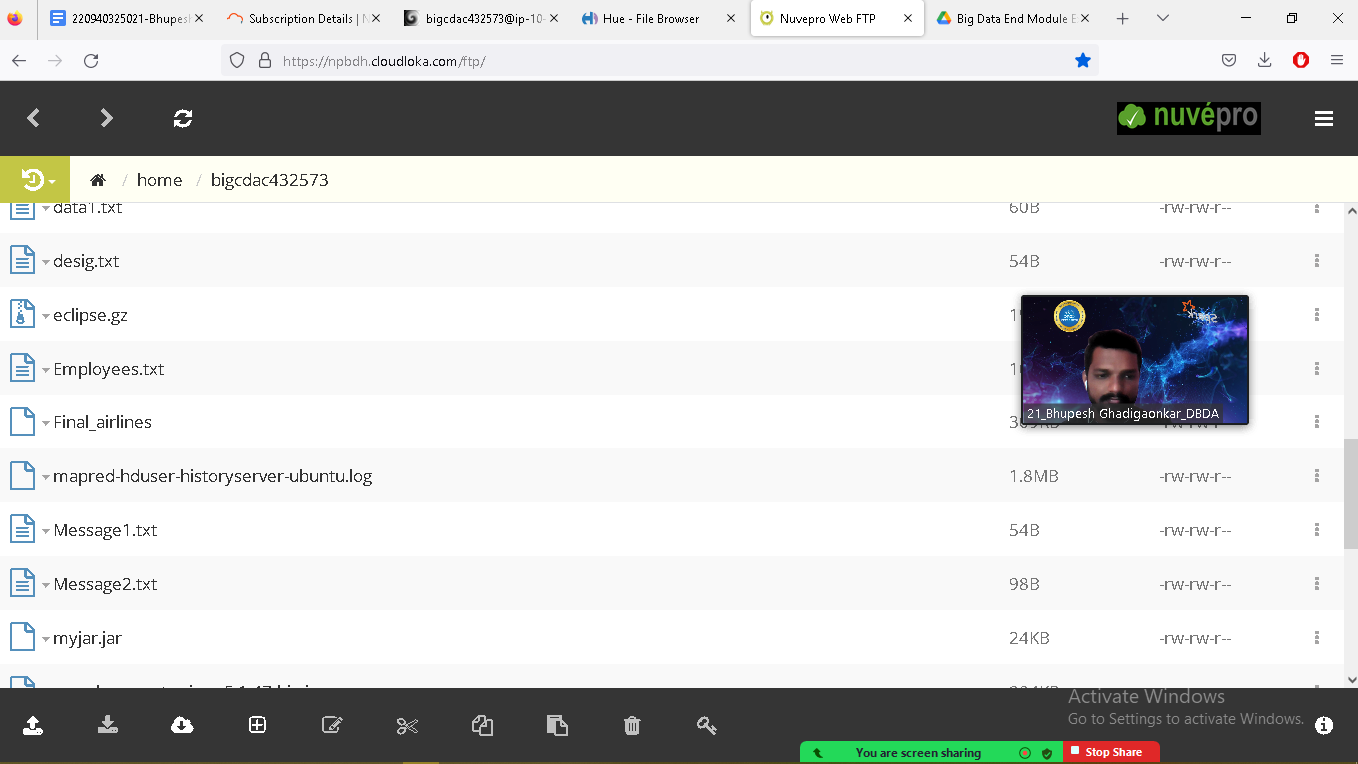
**Basketball 44954.68000000004**

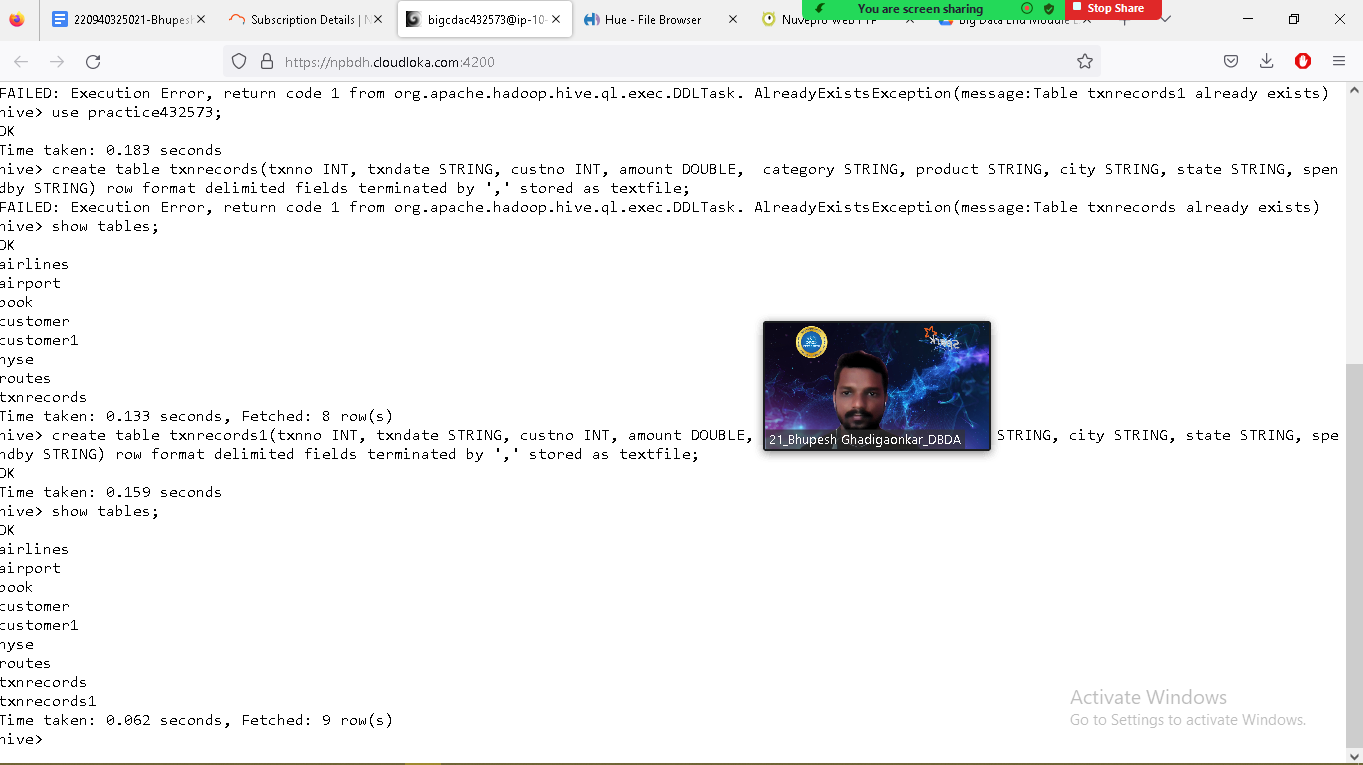
**Beach Volleyball 44890.67000000005**

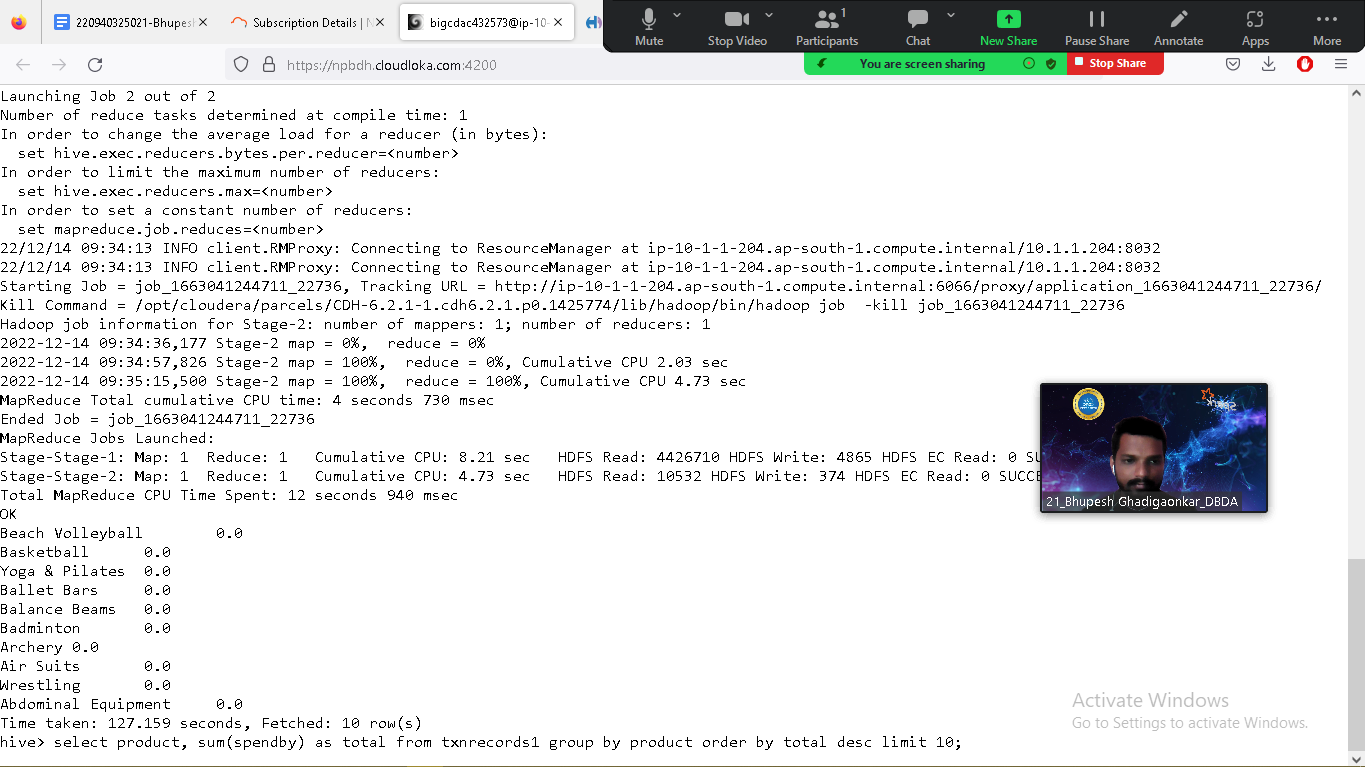
**Time taken: 170.248 seconds, Fetched: 10 row(s)**

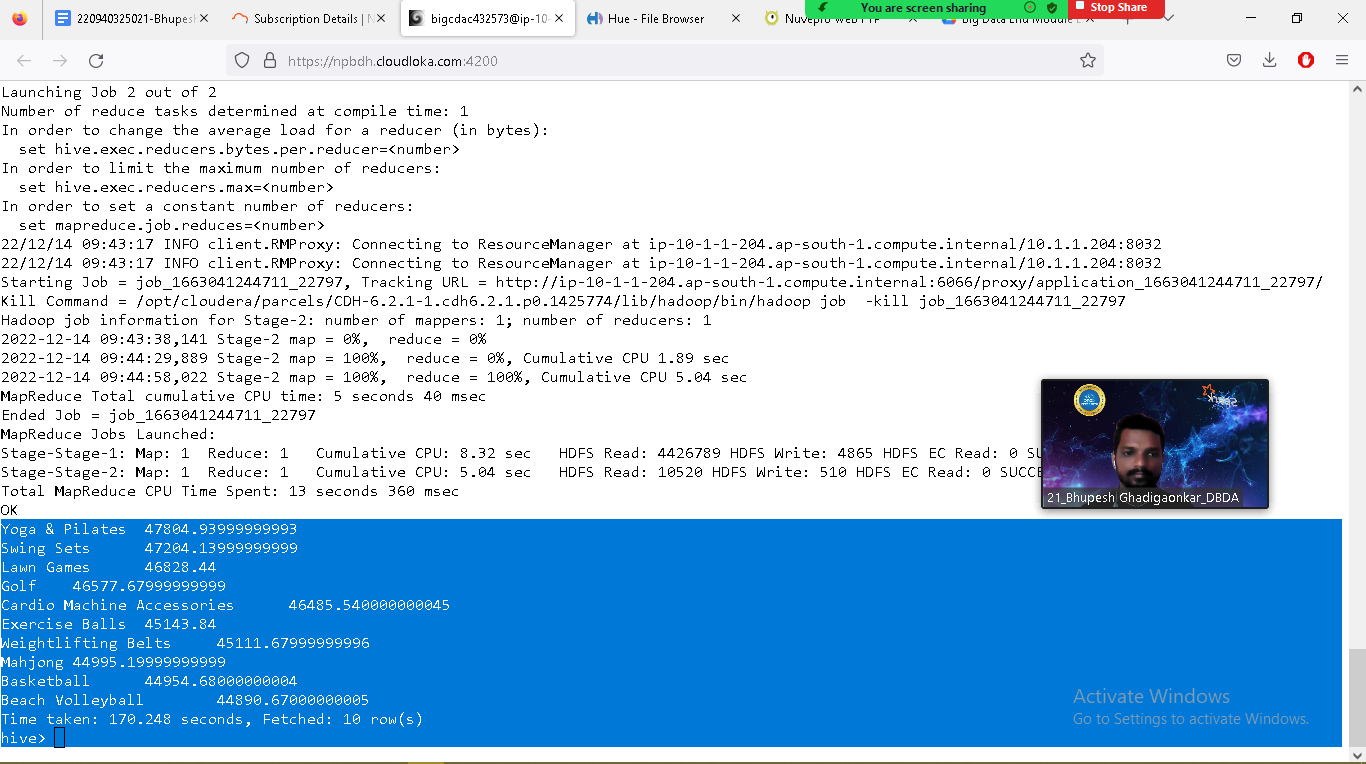
**hive>**



****

****

****

****

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

**create table txnrecsByCat(txnno INT, txndate STRING, custno INT, amount DOUBLE,**

**product STRING, city STRING, state STRING, spendby STRING)**

**partitioned by (category STRING)**

**row format delimited**

**fields terminated by ','**

**stored as textfile;**

****

**PySpark**

**Please find the AIRLINES data set**

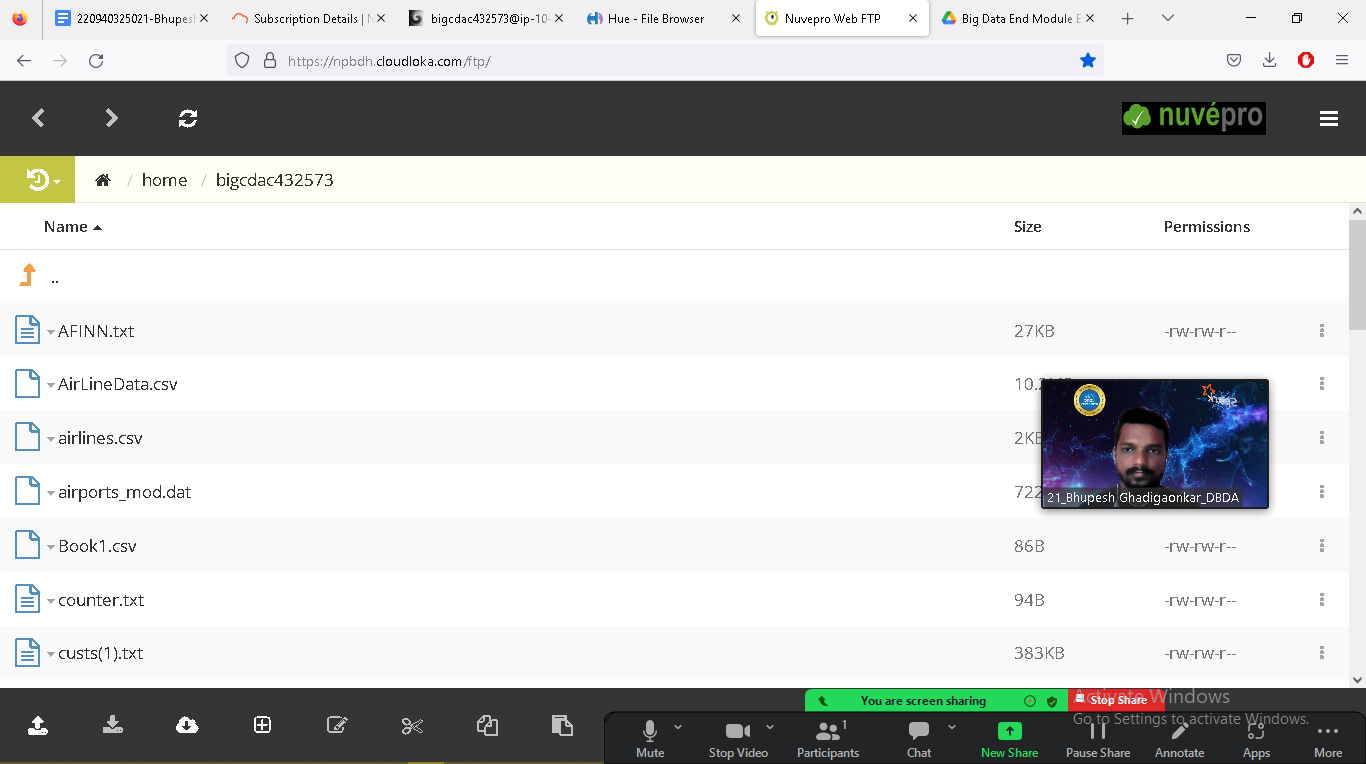
**Year**

**Quarter**

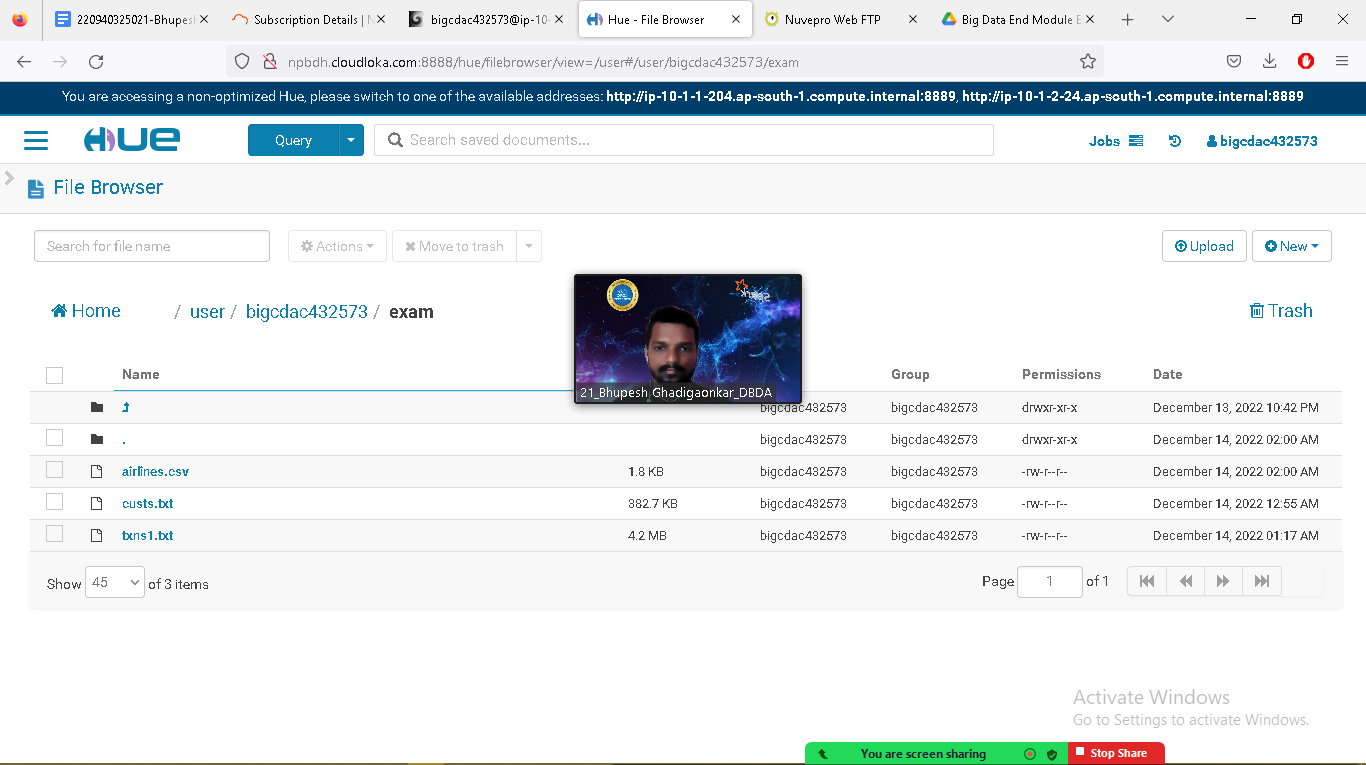
**Average revenue per seat**

**Total number of booked seats**

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



$ hadoop fs -put airlines.csv exam



$ pyspark

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

schema123 = StructType().add("Year",StringType(),True).add("Quarter",StringType(),True).add("ARPS",DoubleType(),True).add("Booked\_seats",IntegerType(),True)

df\_with\_schema2 = spark.read.format("csv").option("header", "True").schema(schema123).load("hdfs://nameservice1/user/bigcdac432573/exam/airlines.csv")

df\_with\_schema2.registerTempTable("airlines")

>>> df\_with\_schema2.printSchema()

root

|-- Year: string (nullable = true)

|-- Quarter: string (nullable = true)

|-- ARPS: double (nullable = true)

|-- Booked\_seats: integer (nullable = true)

>>> df\_with\_schema2.show()

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

|Year|Quarter| ARPS|Booked\_seats|

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

|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

YearWisePs= spark.sql("select year, sum(booked\_seats) as total\_psx from airlines group by year order by total\_psx desc")

YearWisePs.show()

>>> YearWisePs.show()

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

|year|total\_psx|

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

|2007| 176299|

|2013| 173676|

|2001| 173598|

|1996| 167223|

|2008| 166897|

|2012| 166076|

|2015| 165438|

|2004| 164800|

|2010| 163741|

|2014| 159823|

|1997| 157972|

|2003| 156153|

|2000| 154376|

|2006| 153789|

|2002| 152195|

|2005| 150610|

|2009| 150308|

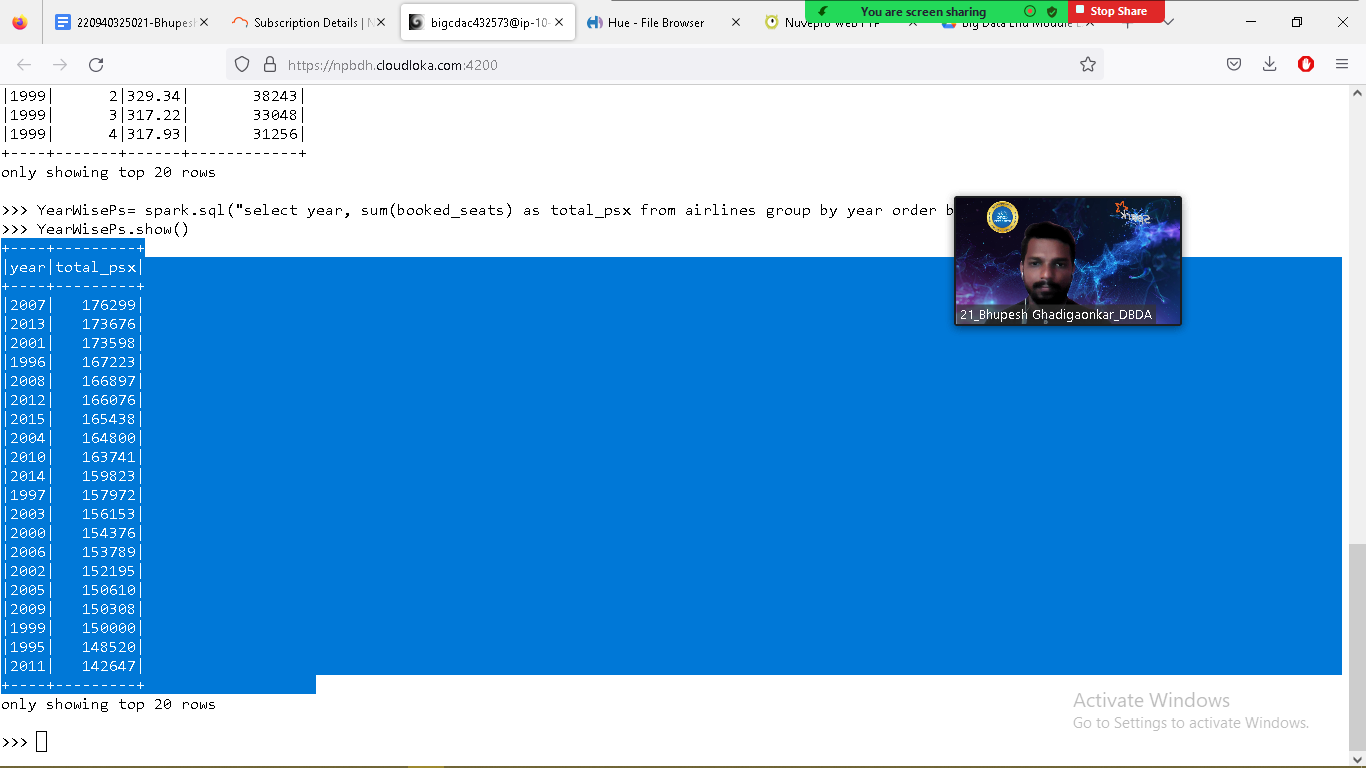
|1999| 150000|

|1995| 148520|

|2011| 142647|

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

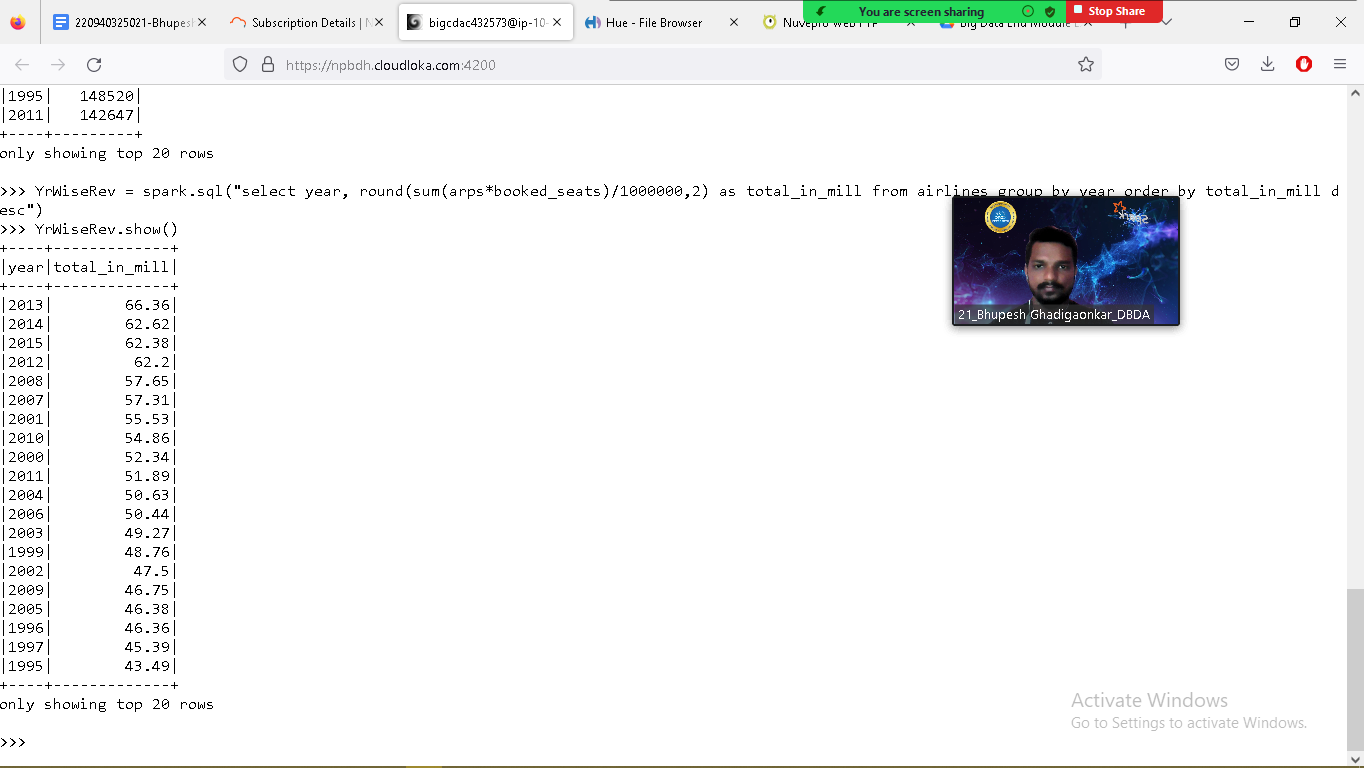
only showing top 20 rows



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

**>>>YrWiseRev = spark.sql("select year, round(sum(arps\*booked\_seats)/1000000,2) as total\_in\_mill from airlines group by year order by total\_in\_mill desc")**

**>>> YrWiseRev.show()**

****

**+----+-------------+**

**|year|total\_in\_mill|**

**+----+-------------+**

**|2013| 66.36|**

**|2014| 62.62|**

**|2015| 62.38|**

**|2012| 62.2|**

**|2008| 57.65|**

**|2007| 57.31|**

**|2001| 55.53|**

**|2010| 54.86|**

**|2000| 52.34|**

**|2011| 51.89|**

**|2004| 50.63|**

**|2006| 50.44|**

**|2003| 49.27|**

**|1999| 48.76|**

**|2002| 47.5|**

**|2009| 46.75|**

**|2005| 46.38|**

**|1996| 46.36|**

**|1997| 45.39|**

**|1995| 43.49|**

**+----+-------------+**

**>>> YrWiseRev.show(1)**

**>>> YrWiseRev.show(1)**

**+----+-------------+**

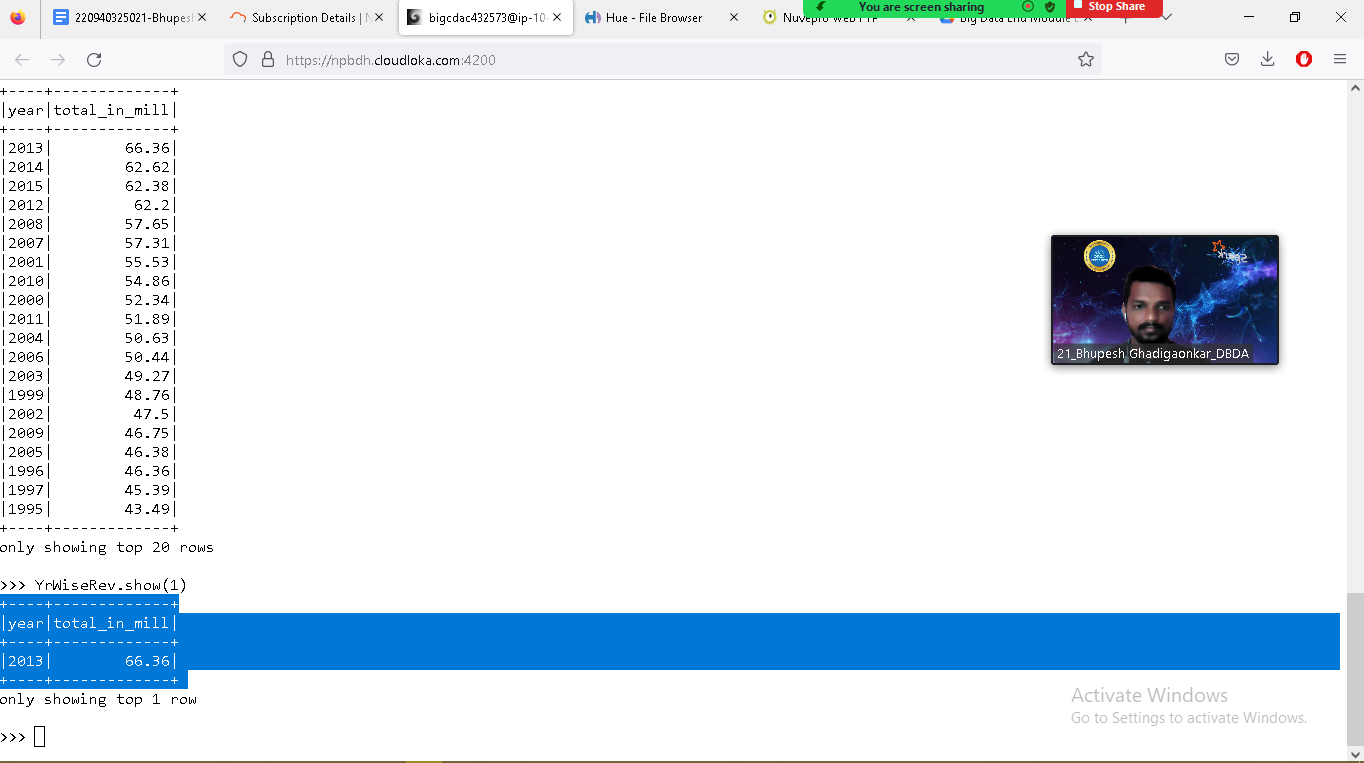
**|year|total\_in\_mill|**

**+----+-------------+**

**|2013| 66.36|**

**+----+-------------+**

**only showing top 1 row**

****