**All time high**

**import** java.io.IOException;

**import** java.util.StringTokenizer;

**import** org.apache.hadoop.conf.Configuration;

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.io.IntWritable;

**import** org.apache.hadoop.io.LongWritable;

**import** org.apache.hadoop.io.DoubleWritable;

**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, org.w3c.dom.Text value, Context context)

{

**try** {

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

**double** high=**double**.parseDouble(str[4]);

context.write(**new** Text(str[1]),**new** DoubleWriteable(high));

}

**catch**(Exception e)

{

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

}

}

}

**public** **static** **class** Reduceclass **extends** Reducer<Text,DoubleWritable,Text,DoubleWritable> {

**private** DoubleWriteable result=**new** DoubleWriteable();

**public** **void** reduce(Test key, Iterable<DoubleWritable> values, Context context)InterruptedException

{

**double** max=0.00;

**for** (DoubleWritable val: values)

{

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

{

max=val.get();

}

}

result.set(max);

context.write(key,result);

}

}

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

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

Job job = Job.*getInstance*(conf, " All time high");

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

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

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

job.setNumReduceTasks(1);

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

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

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

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

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

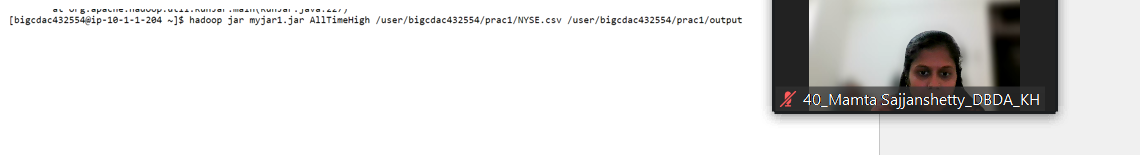
FileOutputFormat.*setOutputPath*(job, **new** Path(args[1]));

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

}

}

}



**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.**

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**

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

**ANS:**

**Table created:**

> create table customer1(cust\_id INT, first\_name String, last\_name String, age INT, profession String)

> row format delimited

> fields terminated by ','

> stored as textfile;

Loading file:

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

**Graphical user interface, text, application

Description automatically generated**

1. select profession, count(cust\_id) from customer1 group by profession;

****



**Sales table created:**

> create table sales2(txn\_id INT, txn\_date String, cust\_id INT,amount Double, category String, Product String, city String, state String, spendby String)

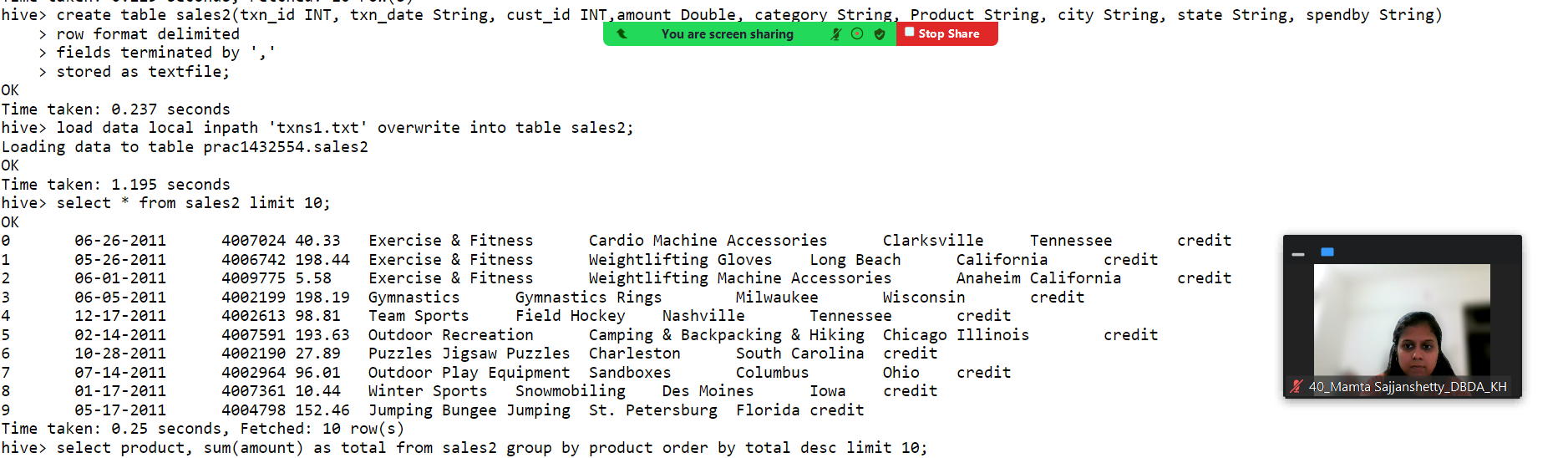
> row format delimited

> fields terminated by ','

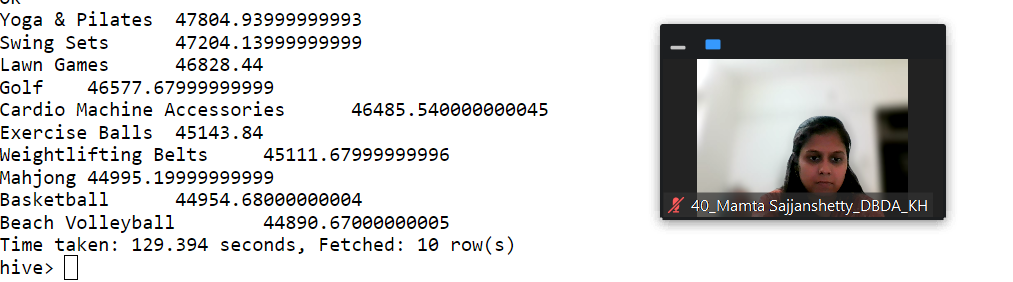
> stored as textfile;

hive> load data local inpath 'txns1.txt' overwrite into table sales2;

**Loaded file:**



hive> select product, sum(amount) as total from sales2 group by product order by total desc limit 10;



**partition table:**

create table sales3\_part(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 into sales2\_part partition(category) select txn\_id,txn\_date,cust\_id,amount,Product,city,state,spendby from sales2;

**Q 3.**

**PySpark**

Please find the AIRLINES data set

Year

Quarter

Average revenue per seat

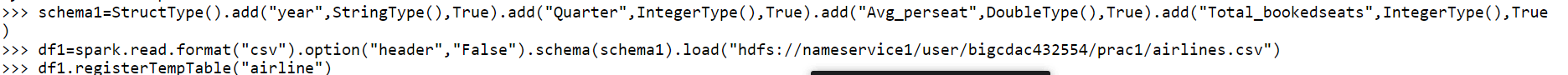
Total number of booked seats

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

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

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

**group)**

****

schema1=StructType().add("year",StringType(),True).add("Quarter",IntegerType(),True).add("Avg\_perseat",DoubleType(),True).add("Total\_bookedseats",IntegerType(),True

)

>>> df1=spark.read.format("csv").option("header","False").schema(schema1).load("hdfs://nameservice1/user/bigcdac432554/prac1/airlines.csv")

>>> df1.registerTempTable("airline")

Ans1:

Q1=spark.sql(“select year, sum(Total\_bookedseats)as highest from airline group by year order by highest desc limit 1”)

Text

Description automatically generated with low confidence

Ans2: q2=spark.sql(“select year, sum(Avg\_perseat) as high from airline group by year order by high limit 1”)

