Q1.

JAVA CODE FOR MAPREDUCE

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>

{

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

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);

//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, "Highest 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);

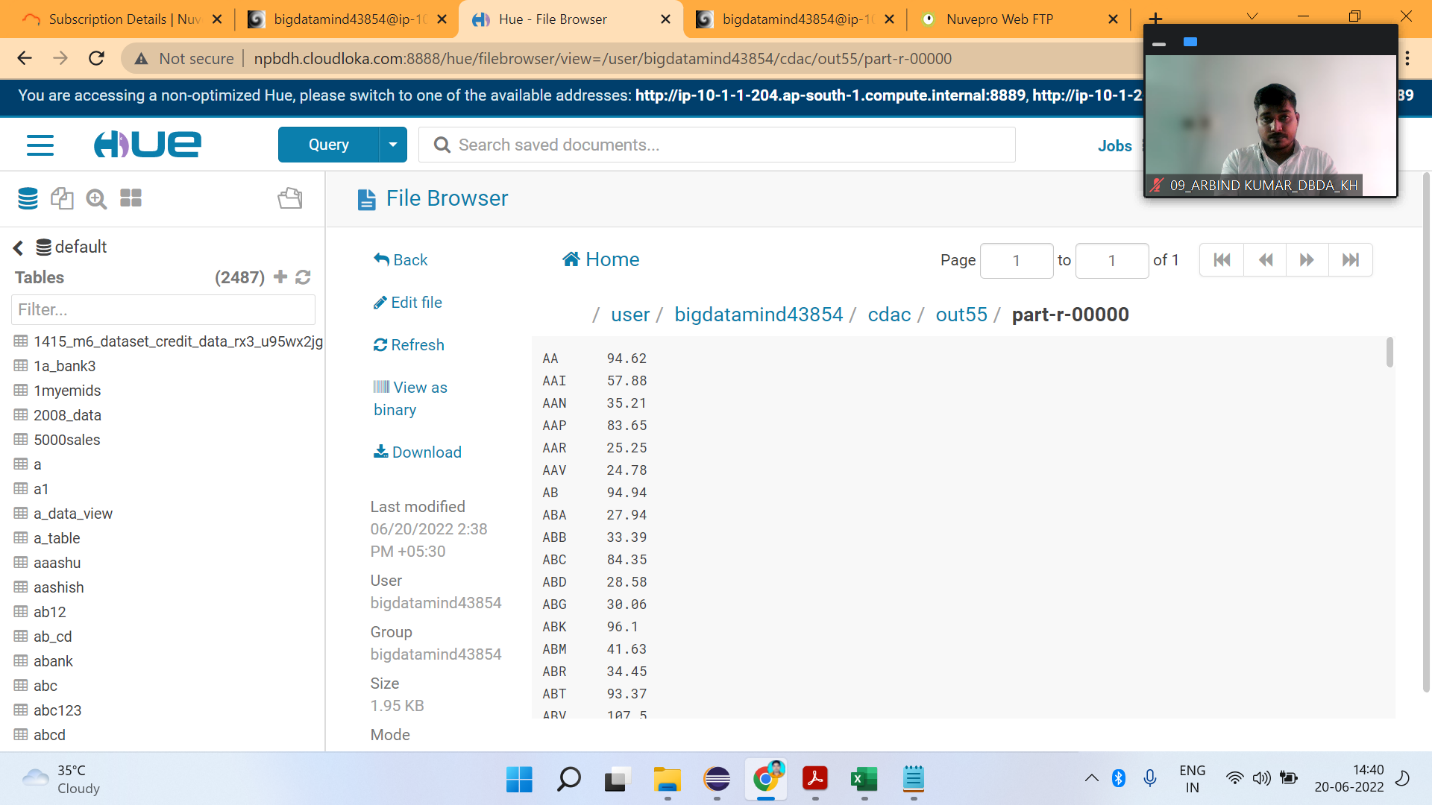
}

}

command for mapreduce

[bigdatamind43854@ip-10-1-1-204 ~]$ hadoop fs -jar myjar.jar AllTimeHigh cdac/NYSE.CSV cdac/out55

0UTPUT



Q2

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

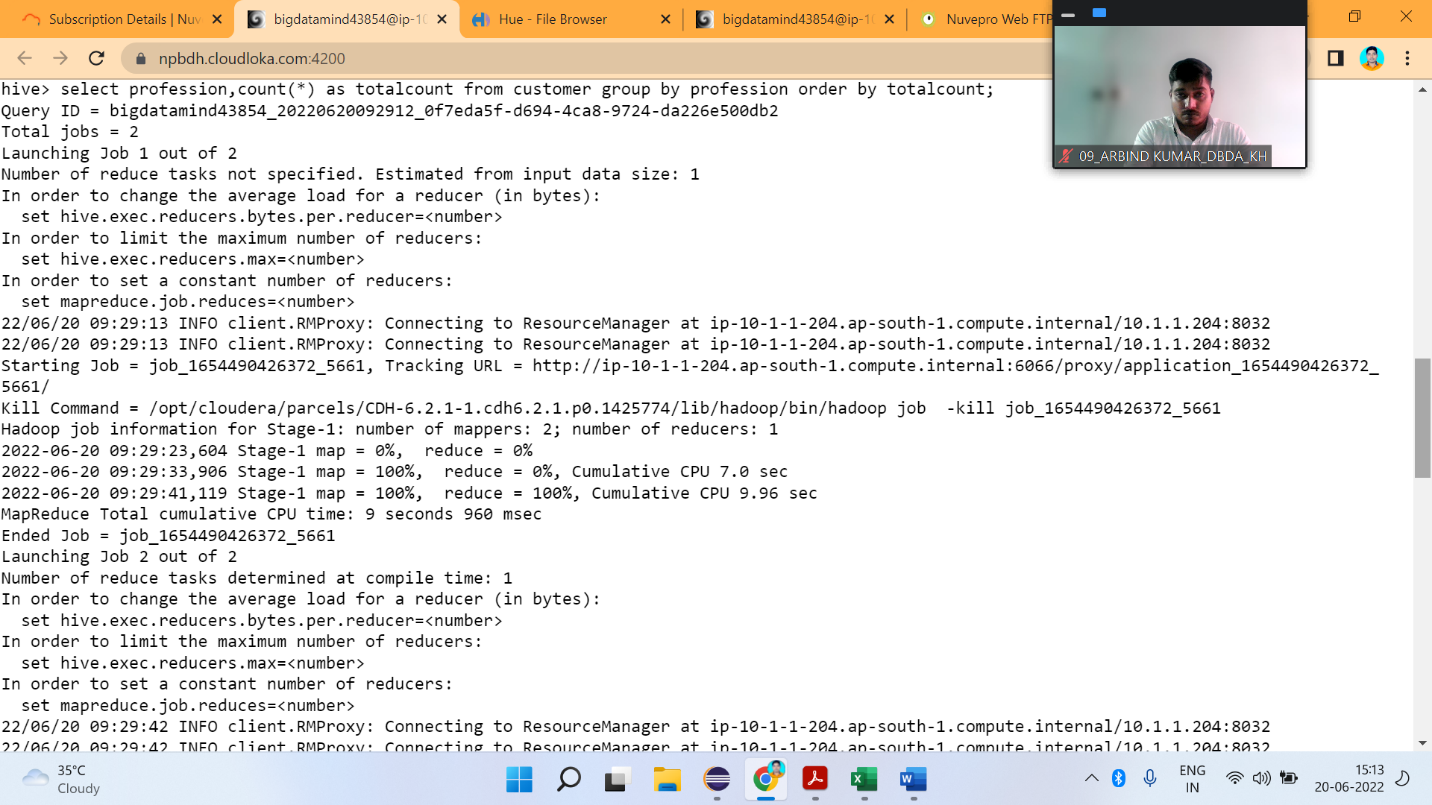
create table customer(custno INT, firstname STRING,lastname STRING,age INT, profession STRING) row format fields terminated by ‘,’ stored as textfile;

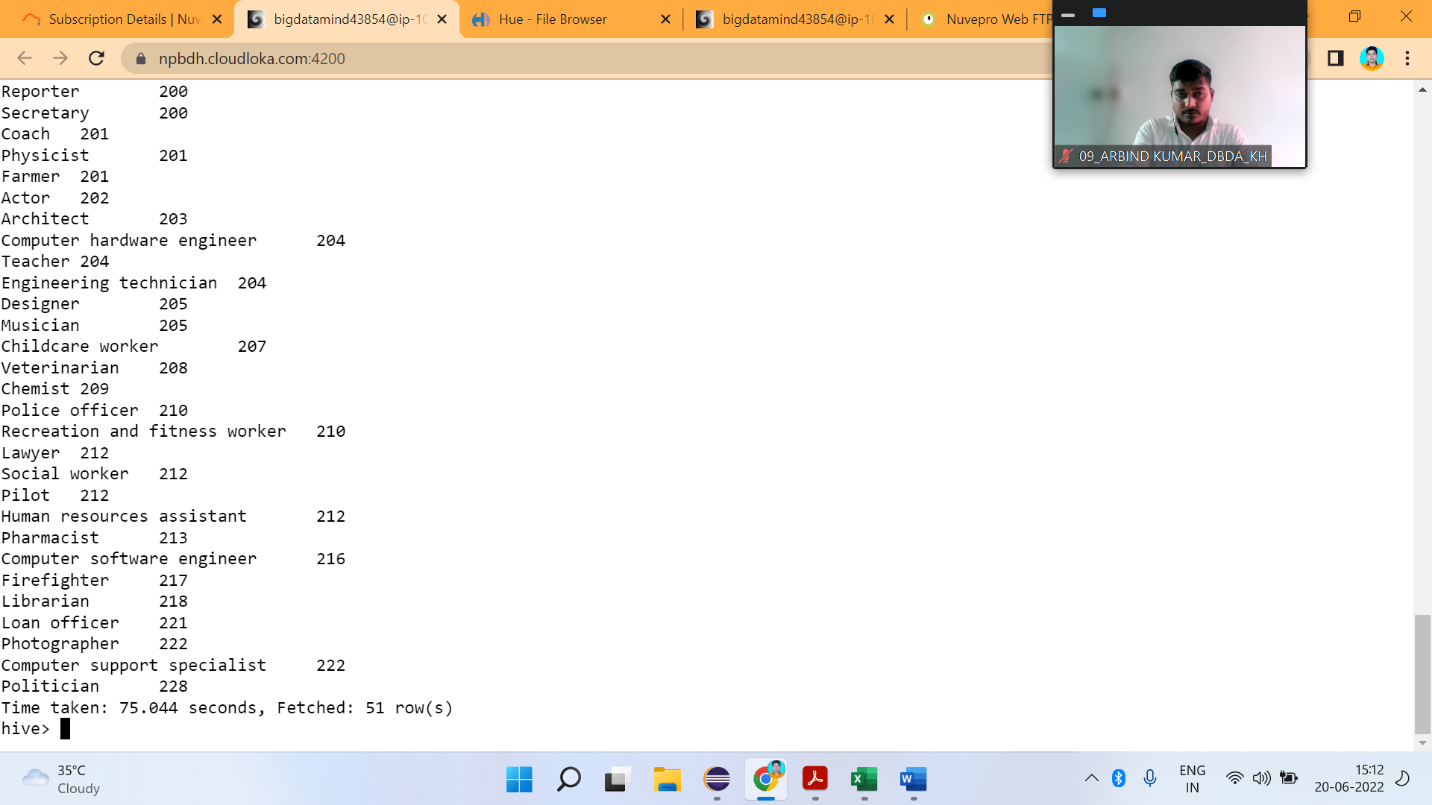
load data into customer table command

LOAD DATA LOCAL INPATH ‘cust.txt’ OVERWRITE INTO TABLE customer;

Query:

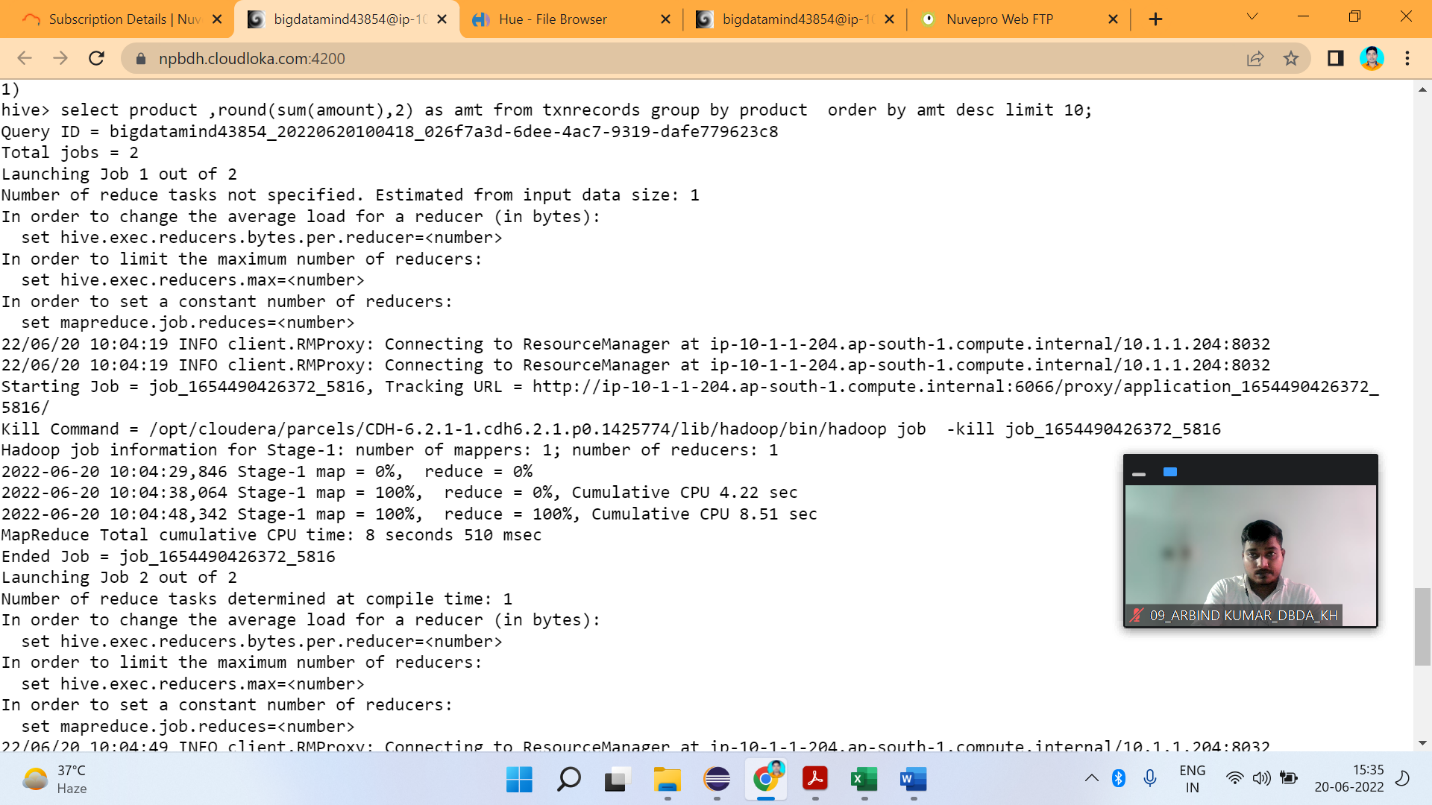
Select profession,count(\*) as totalcount from customer group by profession order by totalcount;

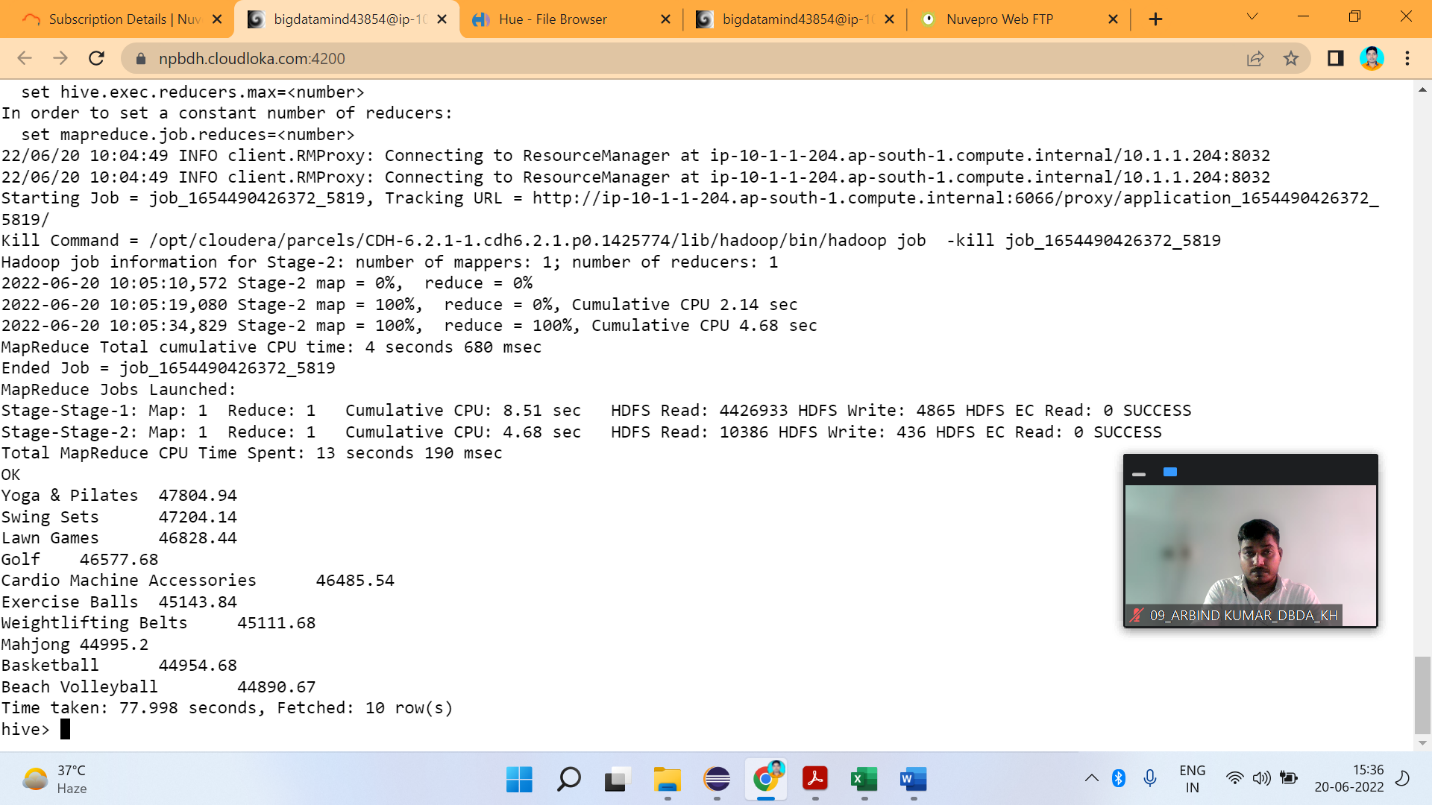




**2) Write a program to find the top 10 products sales wise**

select product ,round(sum(amount),2) as amt from txnrecords group by product order by amt desc limit 10;





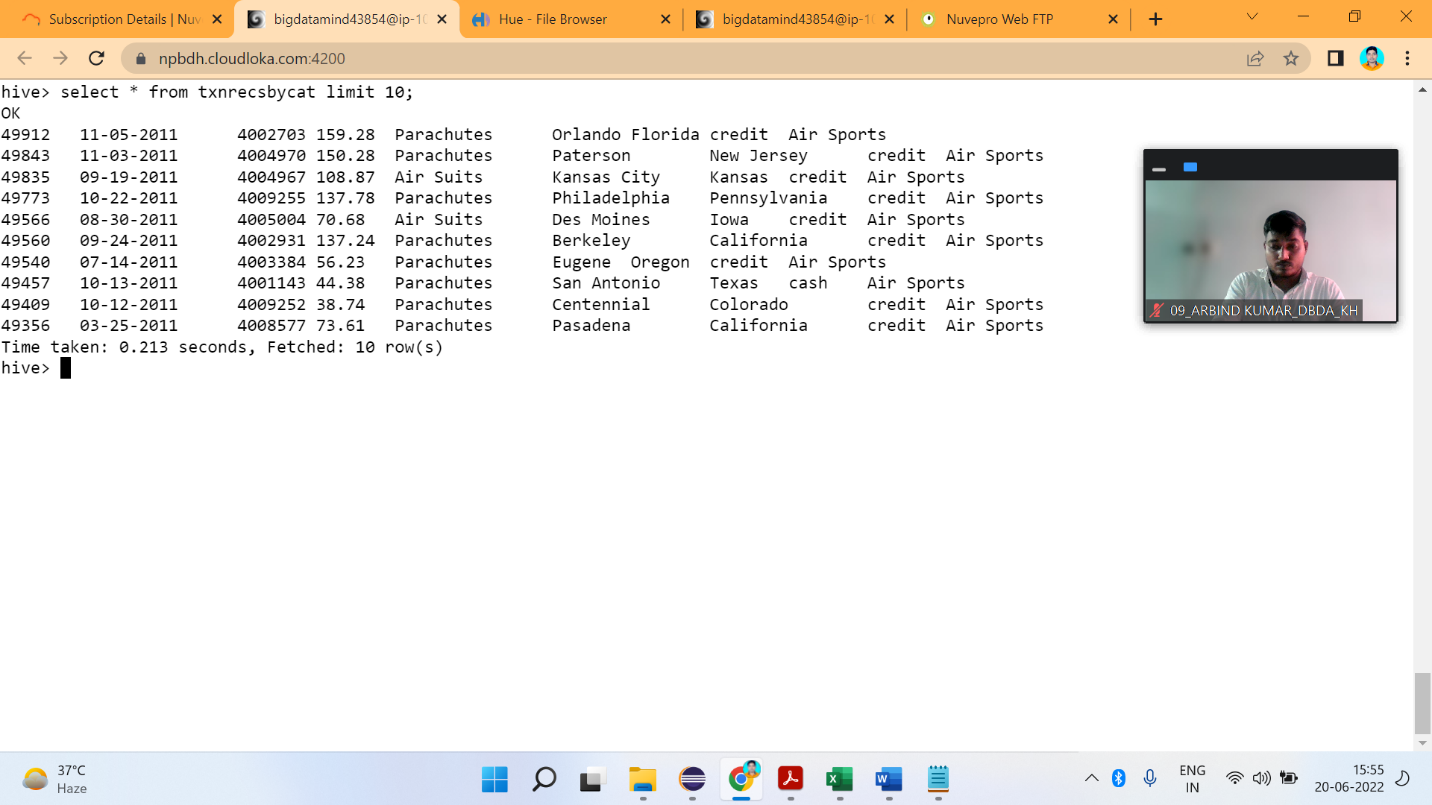
**3) Write a program to create partiioned table on category**

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

set hive.exec.dynamic.partition=true;

**create table 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;**

**INSERT OVERWRITE TABLE txnrecsbycat PARTITION(category) select txn.txnno,txn.txndate,txn.custno,txn.amount,txn.product,txn.city,txn.state,txn.spendby,txn.category from txnrecords txn DISTRIBUTED By category ;**

****

**Q3**

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

rdd = sc.textFile("/user/bigdatamind43854/airlines.csv") header = rdd.first()

rdd2 = rdd.filter(lambda a : a != header)

rdd3 = rdd2.map(lambda a : a.encode("ascii","ignore"))

arrayrdd = rdd3.map(lambda a : a.split(","))

kvrdd = arrayrdd.map(lambda a : (a[0],int(a[3])))

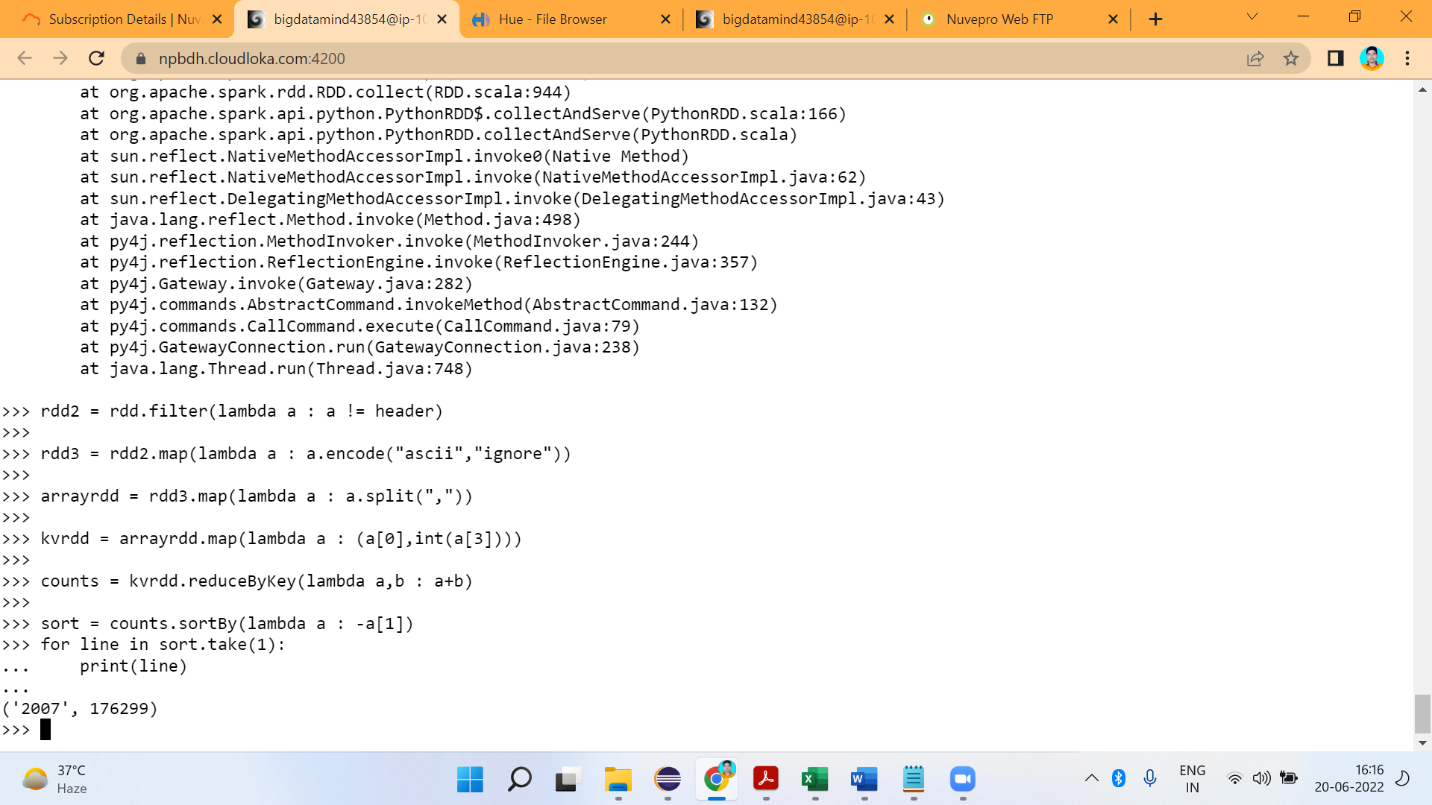
counts = kvrdd.reduceByKey(lambda a,b : a+b)

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

for line in sort.take(1):

print(line)

(‘2007’, 176299)



identifying the highest revenue generation for which year

>>> key\_value = arrayRDD.map(lambda a : (a[0] ,float(a[2]) \* float(a[3])))

>>> add\_total = key\_value.reduceByKey(lambda a,b : a+b)

>>> sortbyval = add\_total.sortBy(lambda a : -a[1])

>>> for i in sortbyval.collect():

... print(i)

('2013', 66363208.71)

('2014', 62624175.85000001)

('2015', 62378990.57)

('2012', 62199127.28)

('2008', 57653170.760000005)

('2007', 57309216.07)

('2001', 55533779.99999999)

('2010', 54861521.29)

('2000', 52342926.550000004)

('2011', 51888286.22)

('2004', 50631364.949999996)

('2006', 50437898.419999994)

('2003', 49273210.83)

('1999', 48757714.48)

('2002', 47499146.5)

('2009', 46746446.59)

('2005', 46376786.24)

('1996', 46358778.03)

('1997', 45385236.16)

('1995', 43494243.22)

('1998', 42035717.78)

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

group)

CODE

----------

>>> key\_value = arrayRDD.map(lambda a : ((a[0]+" "+a[1]),float(a[2]) \* float(a[3])))

>>> add\_total = key\_value.reduceByKey(lambda a,b : a+b)

>>> sortbyval = add\_total.sortBy(lambda a : -a[1])

>>> for i in sortbyval.collect():

... print(i)

SOLUTION

------------

('2014 4', 18819408.48)

('2013 1', 18572613.990000002)

('2013 3', 18177814.2)

('2015 2', 17316167.61)

('2000 1', 16385136.57)

('2010 1', 16300345.36)

('2012 4', 16087025.010000002)

('2014 3', 15956667.09)

('2012 3', 15947048.32)

('1999 1', 15742058.22)

('2004 1', 15698315.06)

('2008 1', 15626301.65)

('2007 2', 15539975.62)

('2015 4', 15486025.28)

('2008 4', 15459962.940000001)

('2012 2', 15447962.530000001)

('2006 3', 15339355.92)

('2001 1', 15247249.57)

('2010 4', 15157499.28)

('2011 3', 15124745.98)

('2005 4', 14985094.08)

('2015 1', 14899061.76)

('2013 2', 14861070.0)

('2013 4', 14751710.520000001)

('2012 1', 14717091.42)

('2015 3', 14677735.920000002)

('2007 4', 14177801.61)

('2001 2', 14148586.16)

('2007 1', 14082536.879999999)

('2014 1', 13995861.6)

('2002 3', 13988802.6)

('2009 1', 13866450.52)

('2014 2', 13852238.68)

('1995 1', 13823960.899999999)

('2001 3', 13717715.399999999)

('2004 2', 13665002.549999999)

('2008 3', 13652979.34)

('1996 1', 13576037.760000002)

('2007 3', 13508901.96)

('1997 2', 13477773.6)

('2003 1', 13409491.09)

('2011 2', 13331400.16)

('2006 4', 13120918.4)

('2000 2', 12999663.64)

('2008 2', 12913926.83)

('2000 3', 12720698.100000001)

('2003 3', 12626803.799999999)

('2003 4', 12598591.459999999)

('1999 2', 12594949.62)

('2011 4', 12560625.44)

('2001 4', 12420228.87)

('2002 1', 12372293.219999999)

('2010 2', 12159615.360000001)

('2004 4', 12111781.76)

('1998 3', 12016699.5)

('1996 2', 11864055.6)

('2009 4', 11716105.5)

('2006 2', 11429949.959999999)

('2009 3', 11357456.95)

('2010 3', 11244061.29)

('1998 4', 11190558.74)

('2002 2', 11129457.58)

('1995 2', 11113082.4)

('1997 4', 10993123.54)

('2005 3', 10991040.39)

('1997 3', 10976351.219999999)

('2011 1', 10871514.64)

('2005 2', 10755267.6)

('2003 2', 10638324.479999999)

('2006 1', 10547674.139999999)

('1996 3', 10497174.48)

('1999 3', 10483486.56)

('1996 4', 10421510.19)

('2000 4', 10237428.24)

('2002 4', 10008593.100000001)

('1997 1', 9937987.799999999)

('1999 4', 9937220.08)

('1995 3', 9812141.28)

('2009 2', 9806433.62)

('2005 1', 9645384.17)

('1998 1', 9542933.1)

('1998 2', 9285526.440000001)

('2004 3', 9156265.58)

('1995 4', 8745058.639999999)