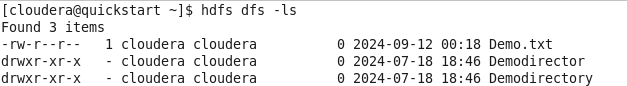
## PRACTICAL NO:01

### Aim: List of Commands (mkdir, touchz, copy from local / put, copy to local

**/ Get move from local, cp, rmr, du, dus, stat)**

1. **Hadoop fs**



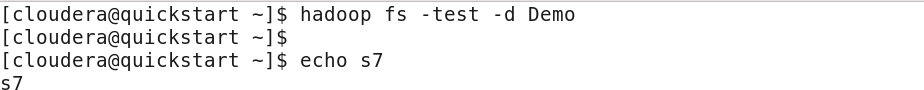
1. **touchz: It creates an empty file.**



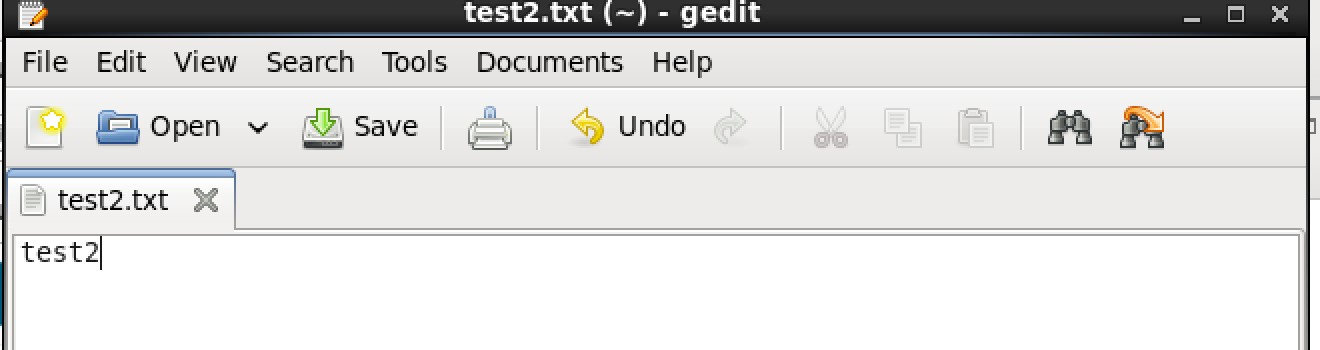
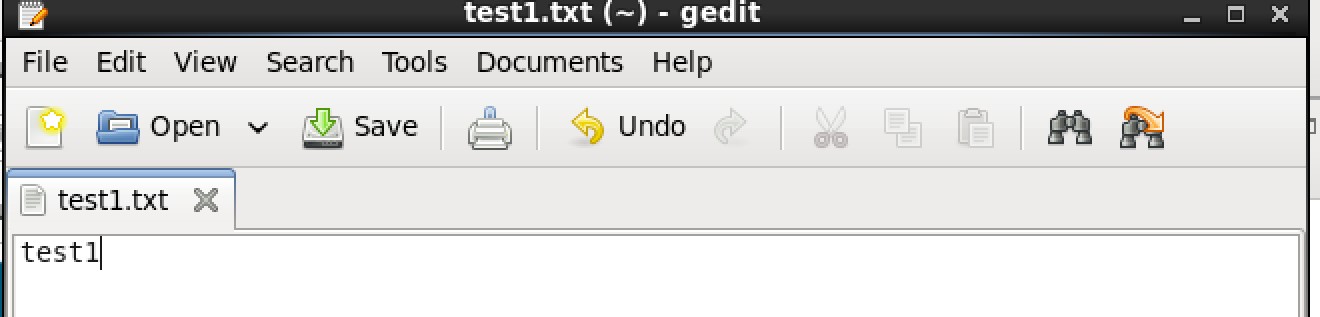
1. **copyFromLocal (or) put:**



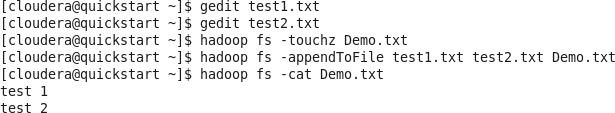
1. **test**



1. **mkdir**



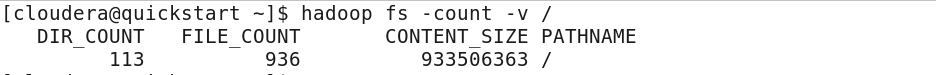
1. **appendToFile**



1. **usage**



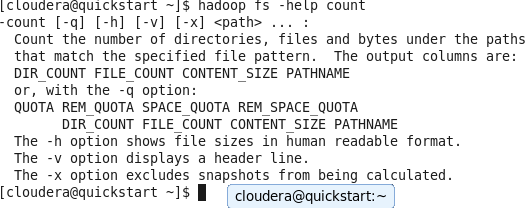
1. **Count**



1. **find**



1. **help**



## PRACTICAL NO: 2

### Aim: Write a program to Map Reduce for WordCount operation.

**WordCountDriver.java**

import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.Text; import [org.apache.hadoop.io](http://org.apache.hadoop.io/).IntWritable;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Mapper; import org.apache.hadoop.conf.Configuration; public class WordCountDriver {

public static void main(String[] args) throws Exception{ Job j1=Job.getInstance(new Configuration()); j1.setJarByClass(WordCountDriver.class); j1.setJobName("Average Word Count"); FileInputFormat.addInputPath(j1,new Path(args[0]));

FileOutputFormat.setOutputPath(j1, new Path(args[1])); j1.setMapperClass(WordCountMapper.class); j1.setReducerClass(WordCountReducer.class); j1.setOutputKeyClass(Text.class); j1.setOutputValueClass(IntWritable.class); System.exit(j1.waitForCompletion(true)? 0:1);

}

}

**WordCountMapper.java**

import java.io.IOException; import java.util.StringTokenizer;0

import org.apache.hadoop.io.Text; import [org.apache.hadoop.io](http://org.apache.hadoop.io/).IntWritable;

import [org.apache.hadoop.io](http://org.apache.hadoop.io/).LongWritable; import org.apache.hadoop.mapreduce.Mapper; import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.Reducer.Context; public class WordCountMapper extends Mapper<LongWritable,Text,Text,IntWritable> {

private final static IntWritable one=new IntWritable(1); private Text word=new Text();

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

String line=value.toString();

StringTokenizer ltr=new StringTokenizer(line); while(ltr.hasMoreTokens()){

word.set(ltr.nextToken()); context.write(word ,one);

}

}

}

**WordCountReducer.java**

import java.io.IOException;

import org.apache.hadoop.io.Text; import [org.apache.hadoop.io](http://org.apache.hadoop.io/).IntWritable;

import org.apache.hadoop.mapreduce.Reducer; public class WordCountReducer extends Reducer<Text,IntWritable,Text,IntWritable> {

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

int sum=0;

for(IntWritable value:values)

{

sum+=value.get();

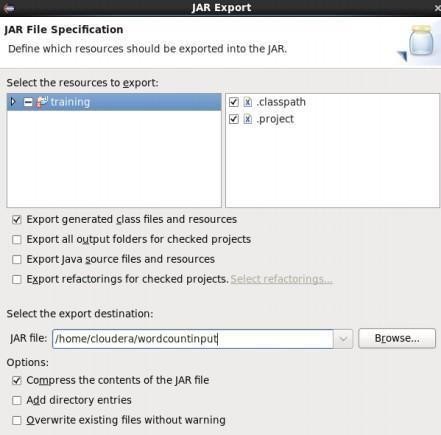
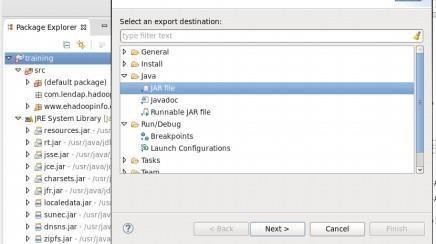
}

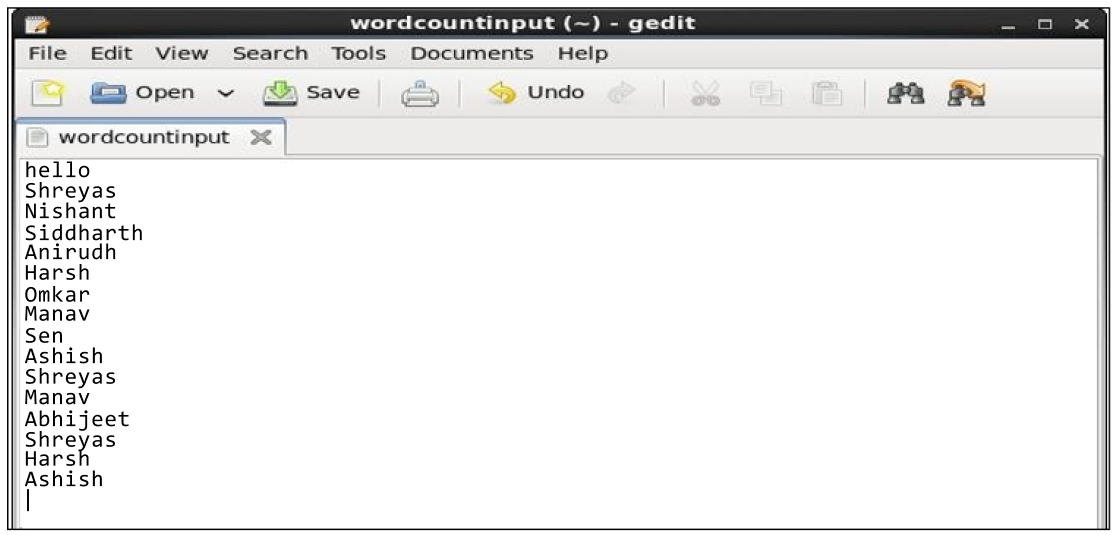
context.write(key, new IntWritable(sum));

}

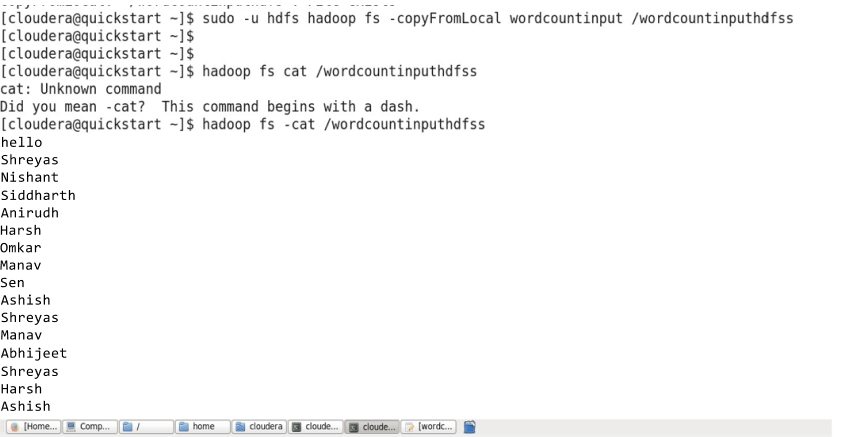
}

**Export .jar file. Right click on training and select export.**

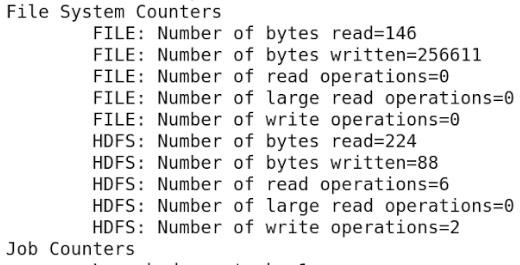




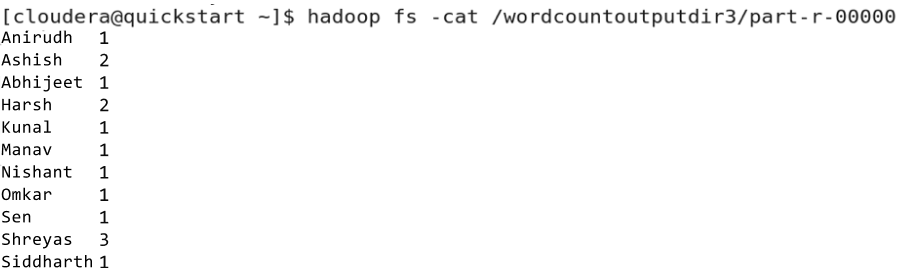
**Follow below HDFS commands**

****

**Executing the jar file using hadoop command:**



**To output the file**

****

### Aim: write a Program in Map Reduce for Matrix Multiplication.

**MatrixMultiplication.java**

import java.io.DataInput; import java.io.DataOutput; import java.io.IOException; import java.util.ArrayList;

import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.Path;

import [org.apache.hadoop.io](http://org.apache.hadoop.io/).DoubleWritable; import [org.apache.hadoop.io](http://org.apache.hadoop.io/).IntWritable; import org.apache.hadoop.io.Writable;

import [org.apache.hadoop.io](http://org.apache.hadoop.io/).WritableComparable; 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.\*;

import org.apache.hadoop.mapreduce.lib.output.\*; import org.apache.hadoop.util.ReflectionUtils; class Element implements Writable {

int tag; int index;

double value;

Element() {

tag = 0;

index = 0;

value = 0.0;

}

Element(int tag, int index, double value) { this.tag = tag;

this.index = index; this.value = value;

}

@Override

public void readFields(DataInput input) throws IOException { tag = input.readInt();

index = input.readInt(); value = input.readDouble();

}

@Override

public void write(DataOutput output) throws IOException { output.writeInt(tag);

output.writeInt(index); output.writeDouble(value);

}

}

class Pair implements WritableComparable<Pair> { int i;

int j;

Pair() {

i = 0;

j = 0;

}

Pair(int i, int j) { this.i = i; this.j = j;

}

@Override

public void readFields(DataInput input) throws IOException { i = input.readInt();

j = input.readInt();

}

@Override

public void write(DataOutput output) throws IOException {

output.writeInt(i); output.writeInt(j);

}

@Override

public int compareTo(Pair compare) { if (i > compare.i) {

return 1;

}

else if (i < compare.i) { return -1;

}

else {

if (j > compare.j) { return 1;

}

else if (j < compare.j) { return -1;

}

}

return 0;

}

public String toString() { return i + " " + j + " ";

}

}

public class MatrixMultiply {

public static class MatrixMapperM extends

Mapper<Object, Text, IntWritable, Element> { @Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

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

int index = Integer.parseInt(tokens[0]);

double elementVal = Double.parseDouble(tokens[2]); Element e = new Element(0, index, elementVal);

IntWritable keyval = new IntWritable(Integer.parseInt(tokens[1])); context.write(keyval, e);

}

}

public static class MatrixMapperN extends Mapper<Object, Text, IntWritable, Element> {

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

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

int index = Integer.parseInt(tokens[1]);

double elementVal = Double.parseDouble(tokens[2]); Element e = new Element(1, index, elementVal);

IntWritable keyval = new IntWritable(Integer.parseInt(tokens[0])); context.write(keyval, e);

}

}

public static class ReducerMN extends

Reducer<IntWritable, Element, Pair, DoubleWritable> {

@Override

public void reduce(IntWritable key, Iterable<Element> values,

Context context) throws IOException, InterruptedException { ArrayList<Element> M = new ArrayList<Element>(); ArrayList<Element> N = new ArrayList<Element>();

Configuration conf = context.getConfiguration(); for (Element element : values) {

Element temp = ReflectionUtils.newInstance(Element.class, conf); ReflectionUtils.copy(conf, element, temp);

if (temp.tag == 0)

{

M.add(temp);

}

else if (temp.tag == 1)

{

N.add(temp);

}

}

for (int i = 0; i < M.size(); i++) {

for (int j = 0; j < N.size(); j++) {

Pair p = new Pair(M.get(i).index, N.get(j).index); double mul = M.get(i).value \* N.get(j).value; context.write(p, new DoubleWritable(mul));

}

}

}

}

public static class MapMN extends

Mapper<Object, Text, Pair, DoubleWritable> { @Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String readLine = value.toString(); String[] pairValue = readLine.split(" ");

Pair p = new Pair(Integer.parseInt(pairValue[0]), Integer.parseInt(pairValue[1]));

DoubleWritable val = new DoubleWritable( Double.parseDouble(pairValue[2]));

context.write(p, val);

}

}

public static class ReduceMN extends

Reducer<Pair, DoubleWritable, Pair, DoubleWritable> { @Override

public void reduce(Pair key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {

double sum = 0.0;

for (DoubleWritable value : values) { sum += value.get();

}

context.write(key, new DoubleWritable(sum));

}

}

public static void main(String[] args) throws Exception { Path MPath = new Path("/expt4/input/M");

Path NPath = new Path("/expt4/input/N");

Path intermediatePath = new Path("/expt4/interim"); Path outputPath = new Path("/expt4/output");

Job job1 = Job.getInstance(); job1.setJobName("Map Intermediate"); job1.setJarByClass(MatrixMultiply.class);

MultipleInputs.addInputPath(job1, MPath, TextInputFormat.class,MatrixMapperM.class); MultipleInputs.addInputPath(job1, NPath, TextInputFormat.class, MatrixMapperN.class); job1.setReducerClass(ReducerMN.class);

job1.setMapOutputKeyClass(IntWritable.class); job1.setMapOutputValueClass(Element.class); job1.setOutputKeyClass(Pair.class); job1.setOutputValueClass(DoubleWritable.class); job1.setOutputFormatClass(TextOutputFormat.class); FileOutputFormat.setOutputPath(job1, intermediatePath); job1.waitForCompletion(true);

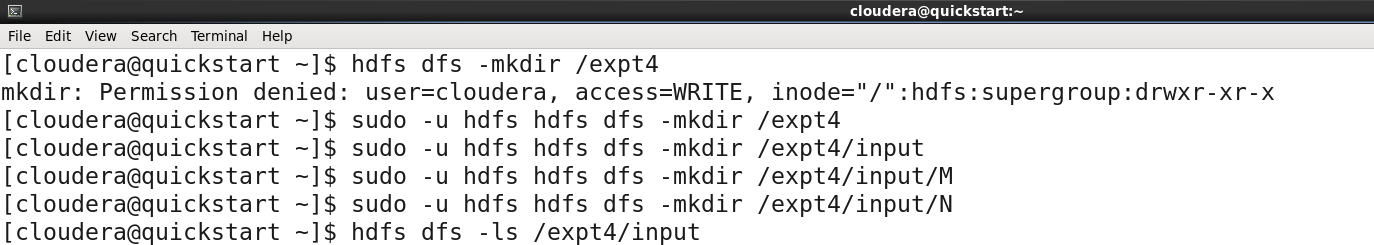
Job job2 = Job.getInstance(); job2.setJobName("Map Final Output"); job2.setJarByClass(MatrixMultiply.class); job2.setMapperClass(MapMN.class); job2.setReducerClass(ReduceMN.class); job2.setOutputKeyClass(Pair.class); job2.setOutputValueClass(DoubleWritable.class); job2.setInputFormatClass(TextInputFormat.class);

job2.setOutputFormatClass(TextOutputFormat.class); FileInputFormat.addInputPath(job2, intermediatePath); FileOutputFormat.setOutputPath(job2, outputPath); job2.waitForCompletion(true);

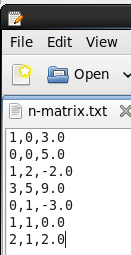
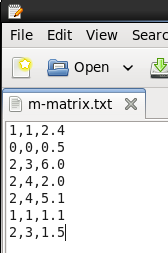
}

}

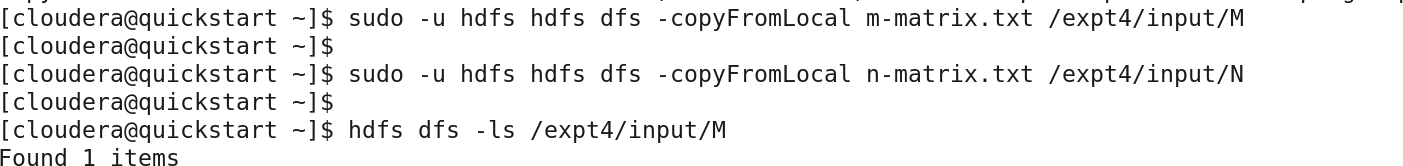
**Prerequisites create the input directories to store the input matrices M and N**



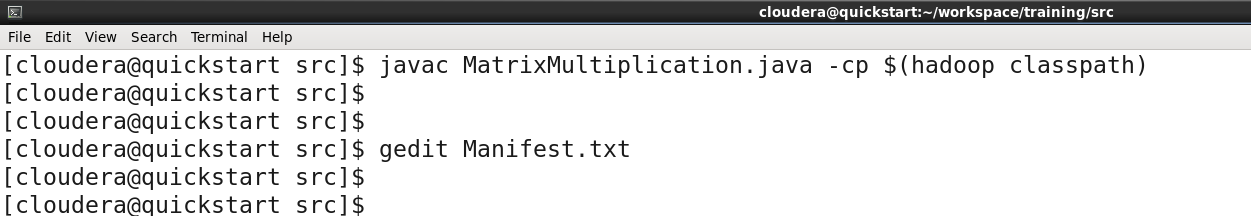
**The figure below shows the matrix data used for this implementation:**



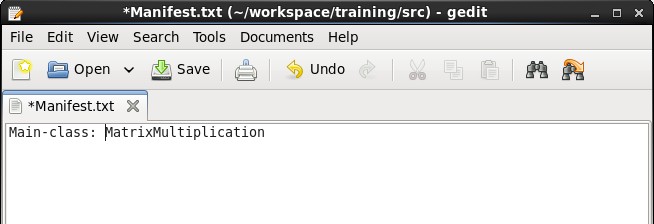
**Copy the matrix data from the local system to HDFS**



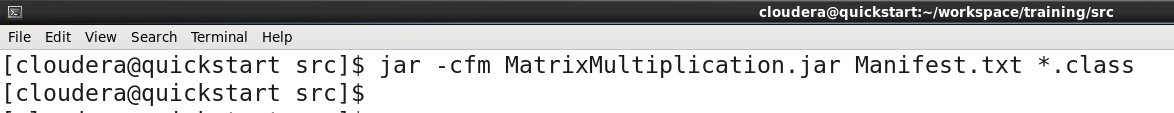
**Compile the code to create the classes**



**To indicate the main class file, create a Manifest file to point out to the main driver class.**



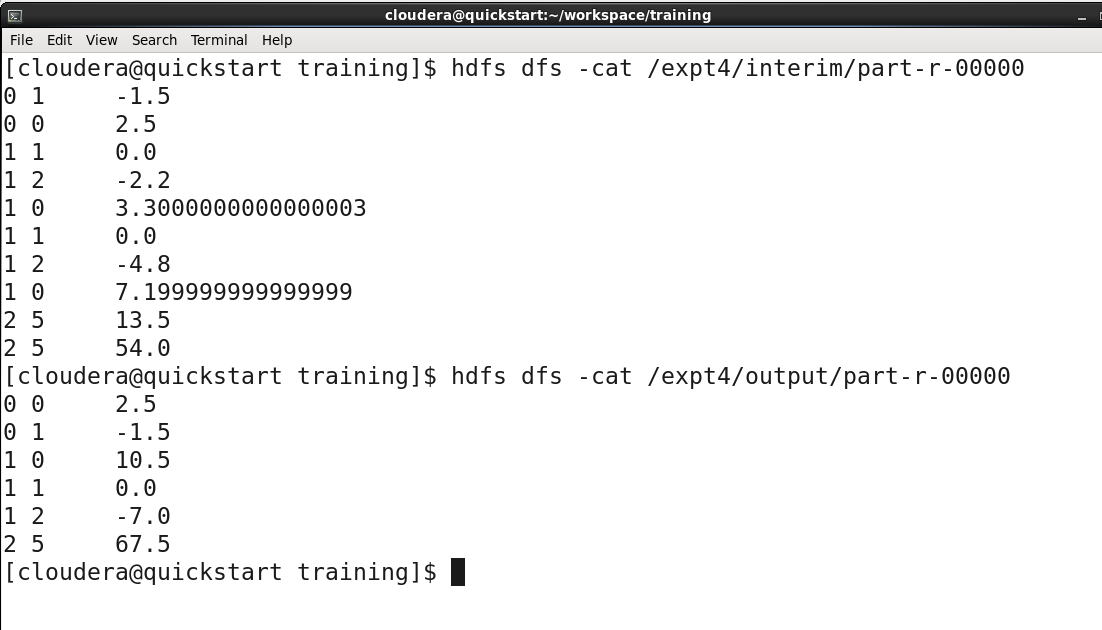
**Compile and create the Jar file required to run the MapReduce Task**



**Run the jar file on the Hadoop ecosystem to trigger all the MapReduce classes.**

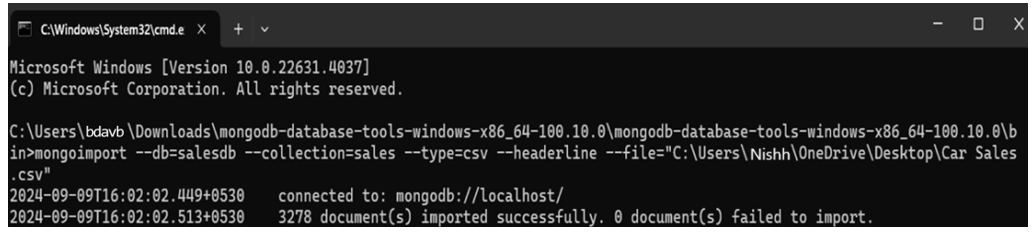
https://lh3.googleusercontent.com/kcNtpDIutpyjyALErkI3Z6_rVJDmoVAuUKYZfqWyAg6r0nT2fpEpunyuCCL1RioTAwqQmmTu-O15RD74-vcVLBqWpu4fM8UMD32_qTdp_XnE4a_LvnmD91Bb0ZqF335c_BZEVSZ0cv8VL_TvXZgKMu3leOpDlJiMX0_YPAH65LmiX8Hc3kIGda-XMi-4DA

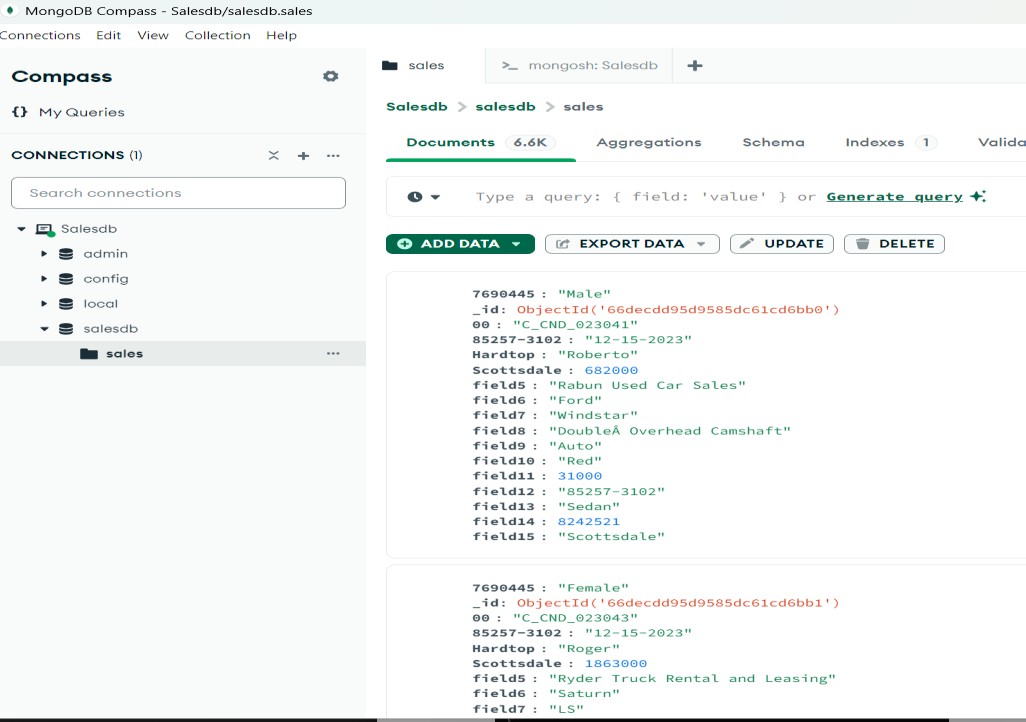
**Output of the file**



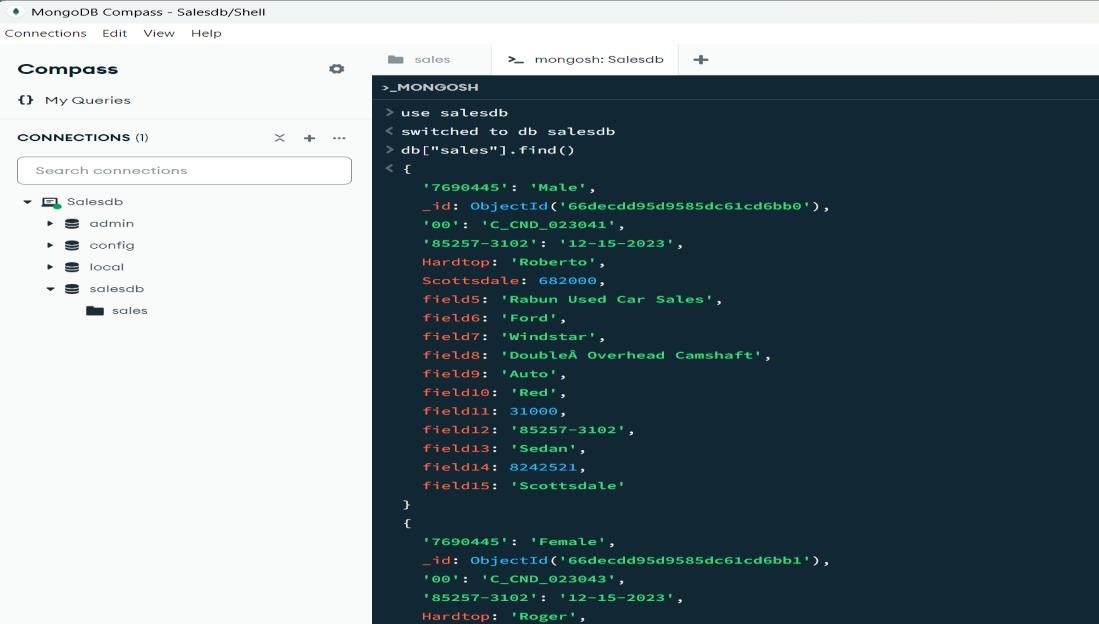
# PRACTICAL NO:3

### Aim: Query the Sample Database using MongoDB querying commands. Query: mongo import –db=sales dB --collection=sales --type=csv --headerline -

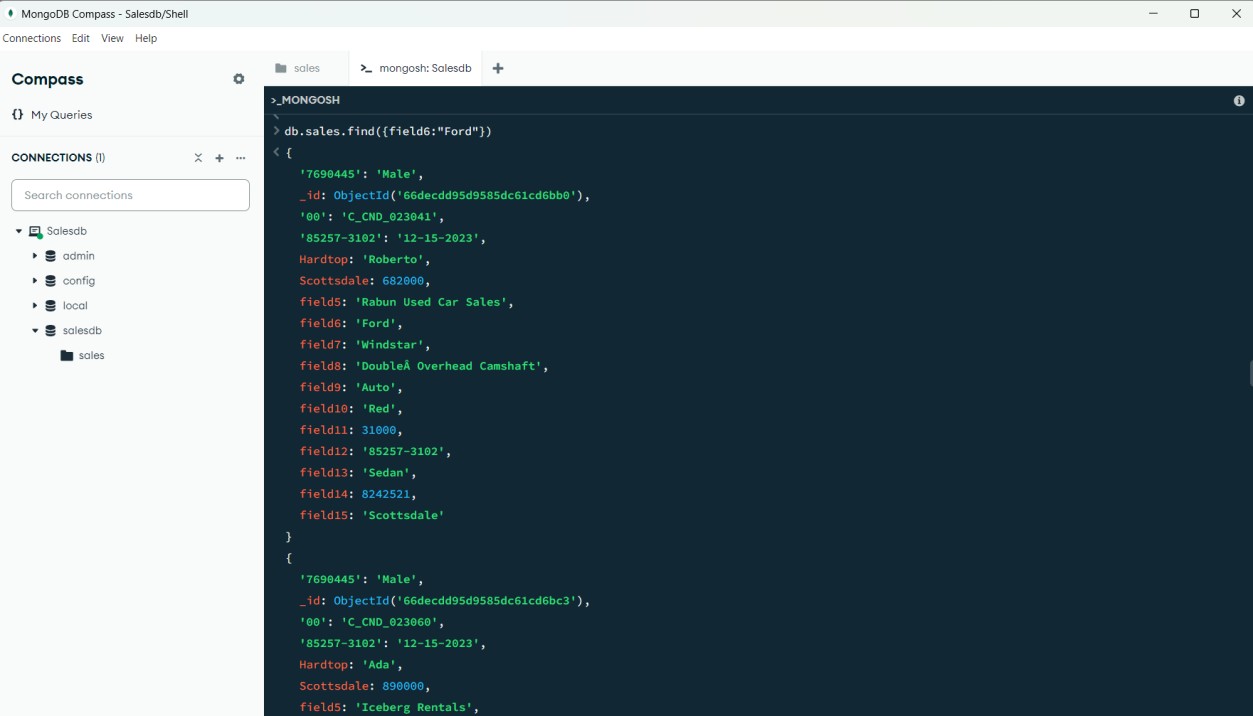
**-file=”C:\Users\Nishh\Desktop\Car Sales.csv”**



**Find data query Query : db.sales.find()**

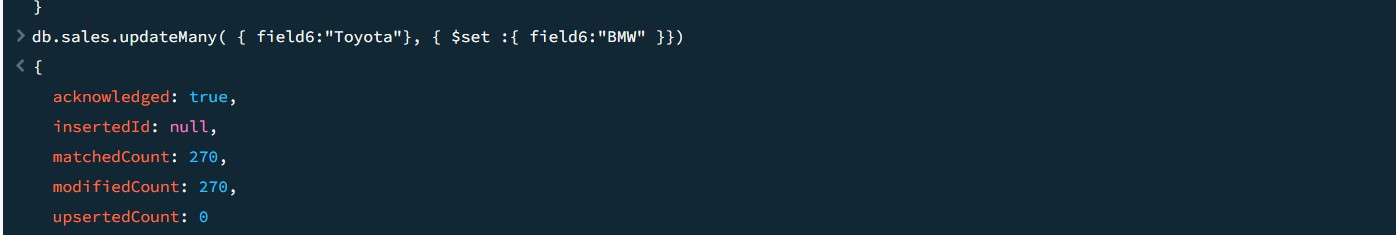


**Db.sales.find({field6:”Ford”})**

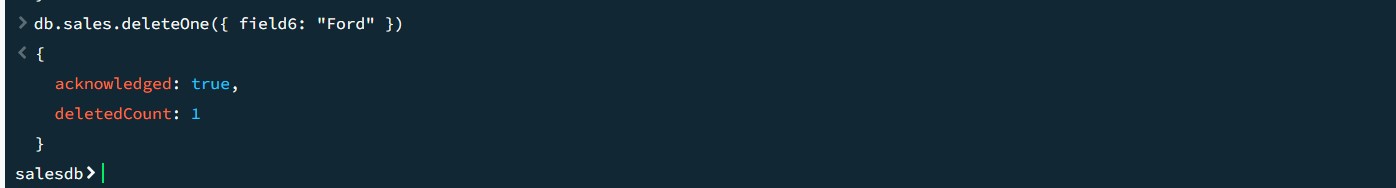


**Update document**

**db. sales.updateMany( { field6 : “Toyota”}, { $set :{ field: “BMW” }})**

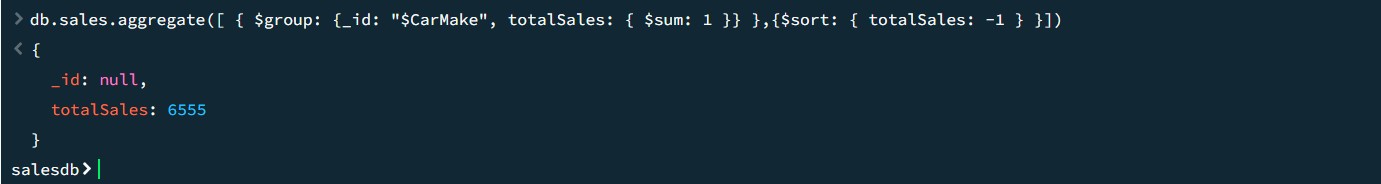


**Delete document db.sales.deleteOne({field6:”Ford” })**



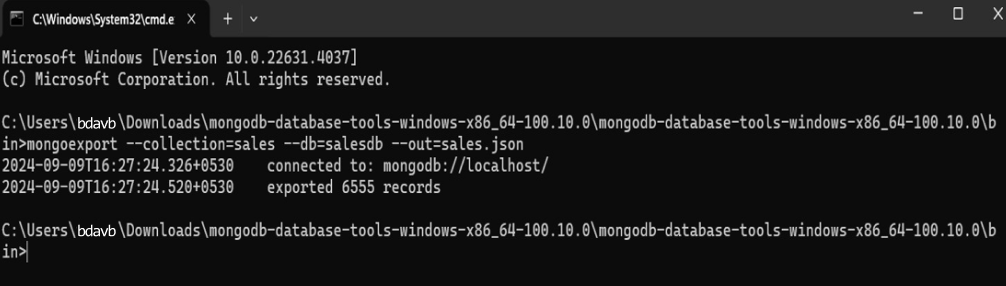
**Aggregate function**

**db. sales.aggregate ([ {$group: {\_id: "$CarMake", totalSales: {$sum: 1}}, {$sort: { totalSales: 1} }])**



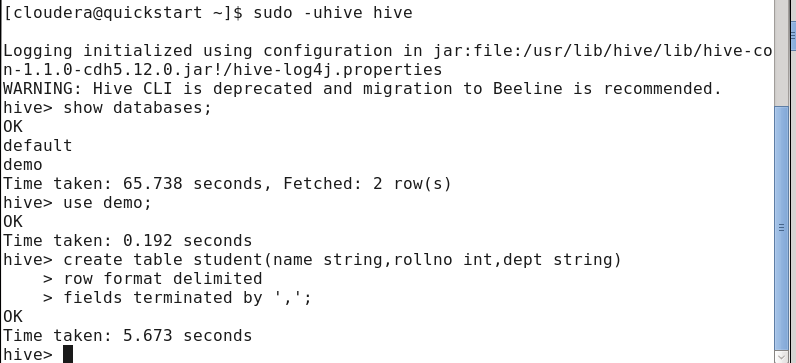
**Extract the salesdb database in json**

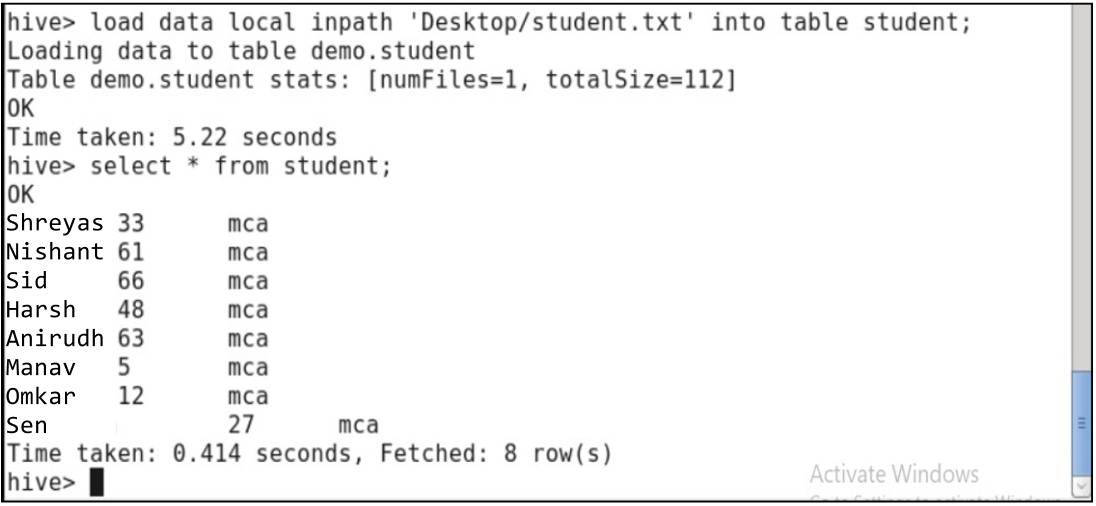
**mongoexport --collection=sales --db=salesdb --out=sales.json**

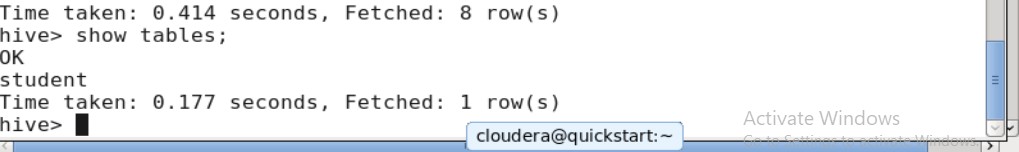
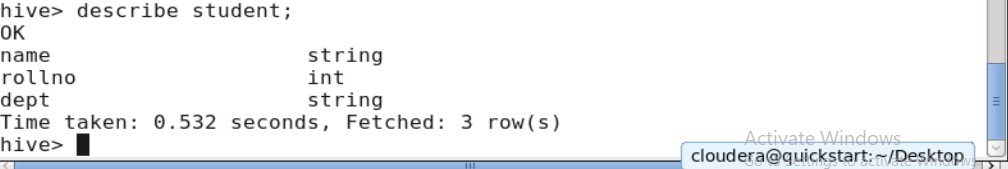


**Aim: Basic Hive Commands**

**PRACTICAL NO:4**



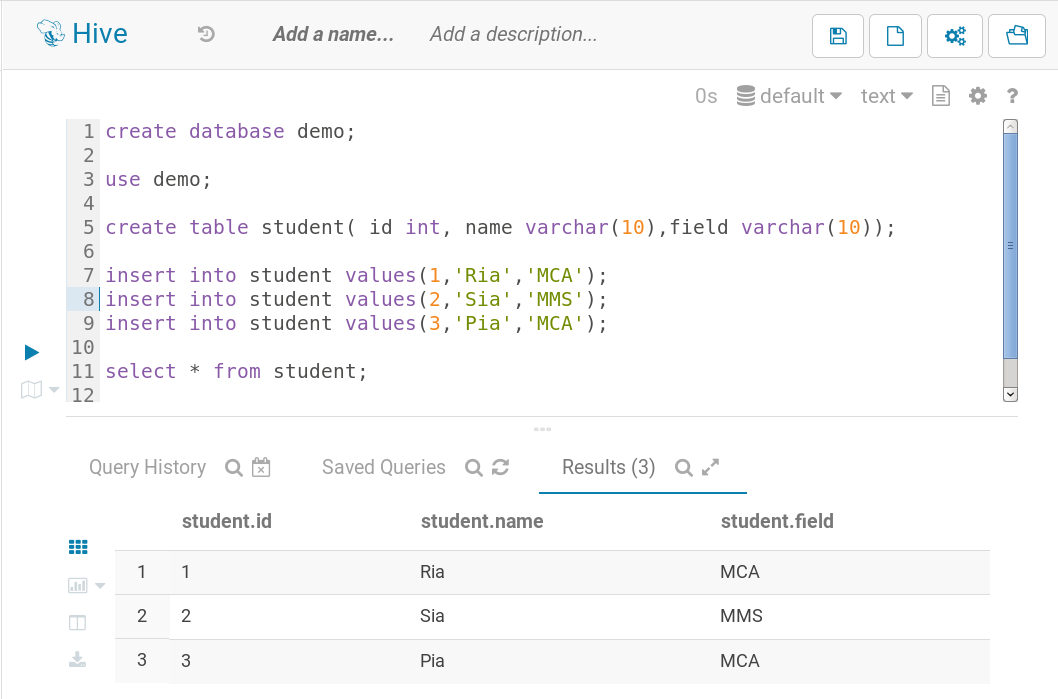




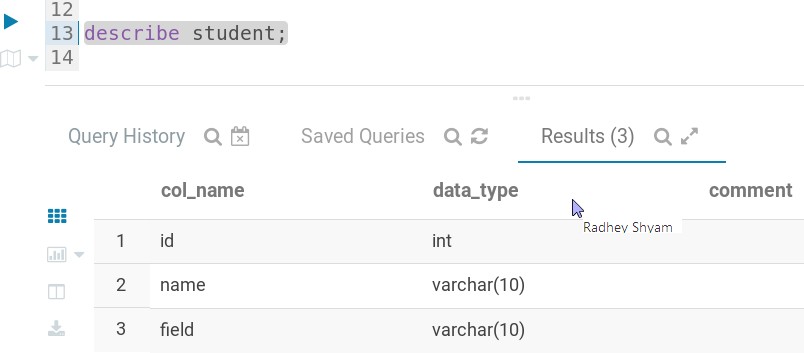
create database demo; use demo;

create table student( id int, name varchar(10),field varchar(10)); insert into student values(1,'Ria','MCA');

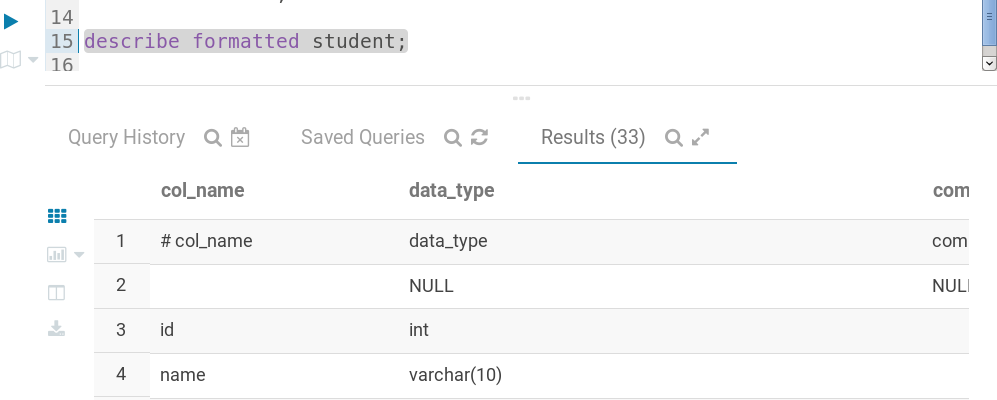
insert into student values(2,'Sia','MMS'); insert into student values(3,'Pia','MCA'); select \* from student;



**describe student;**



**describe formatted student;**



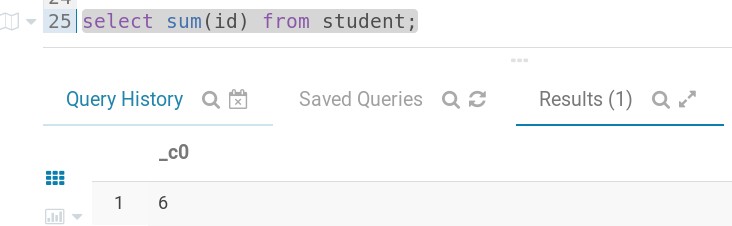
**Select \* from student order by id desc;**



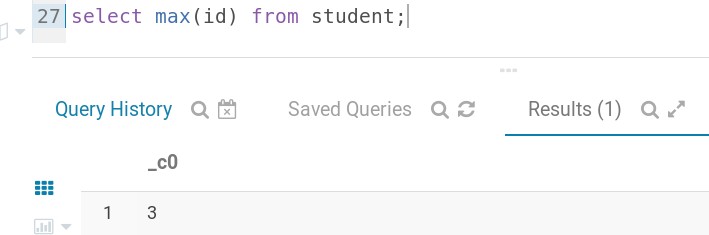
**select \* from student where id=2;**



**select sum(id) from student;**



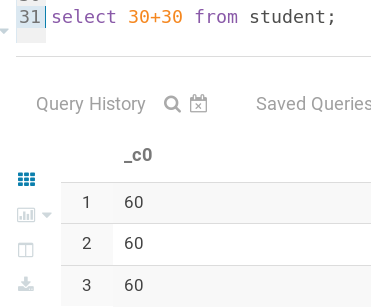
**select max(id) from student;**



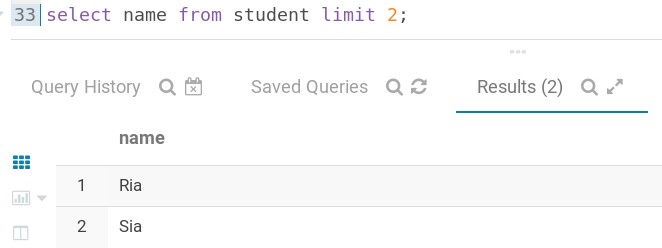
**select count(\*) from student;**



**select 30+30 from student;**



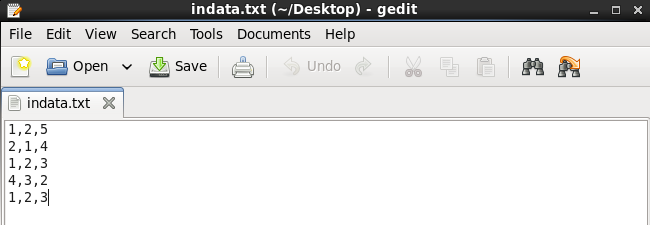
**select name from student limit 2;**



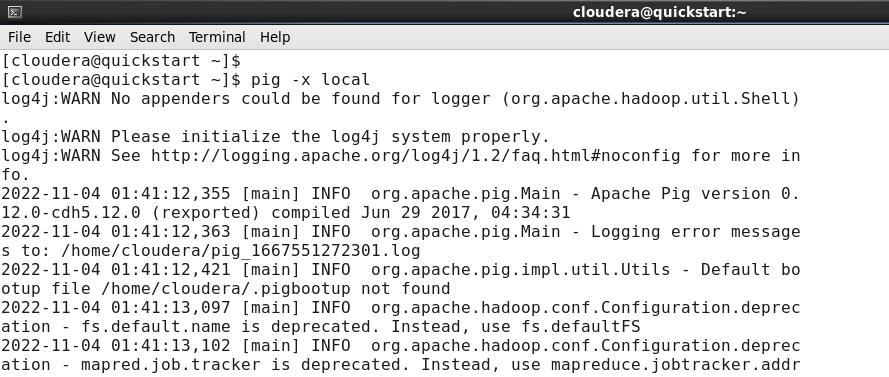
## PRACTICAL NO:5

### Aim: PIG List of Commands

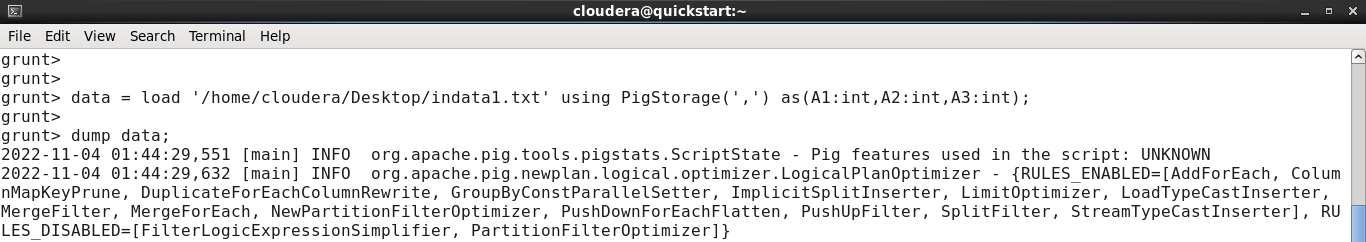
**Indata1.txt**

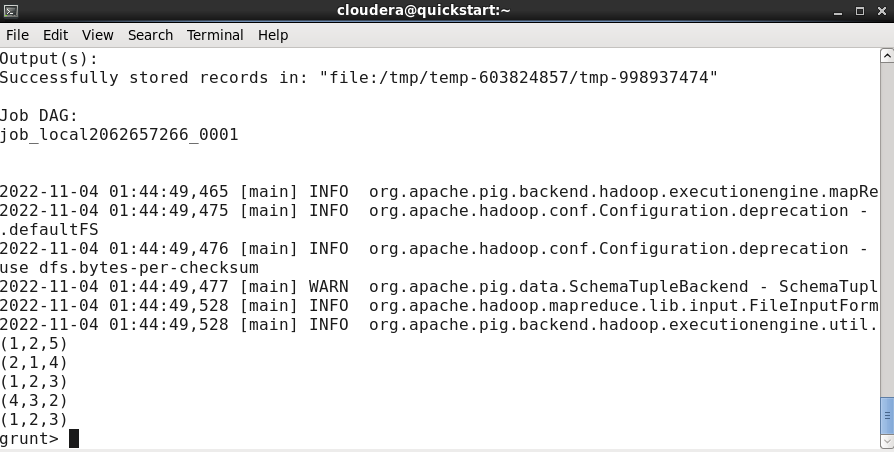


**Pig –x local**

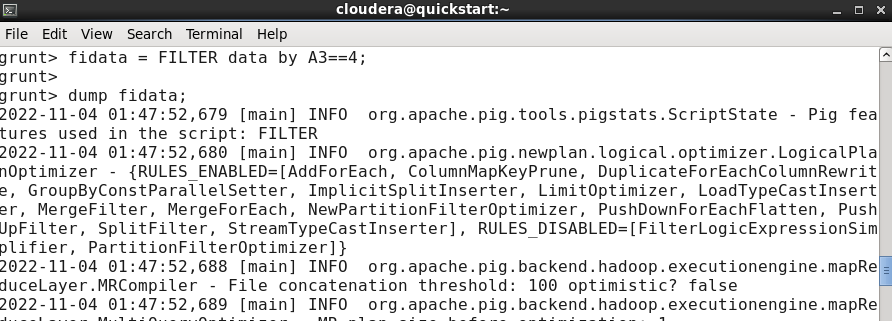


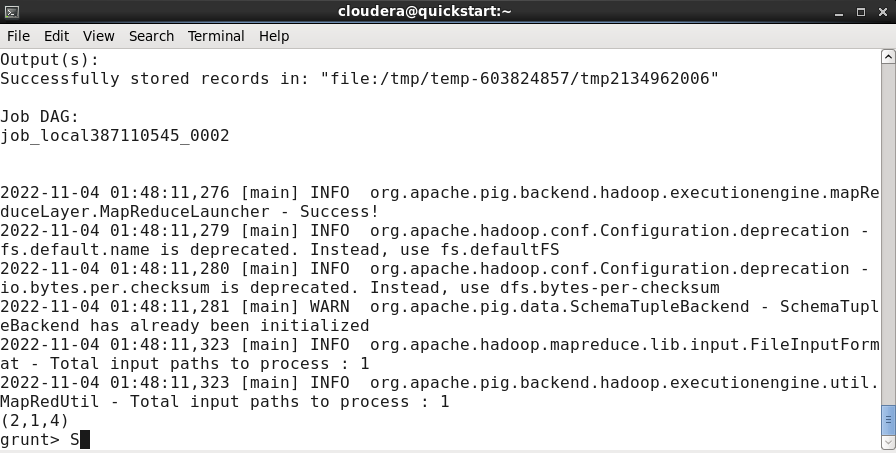
**Load data and dump data**



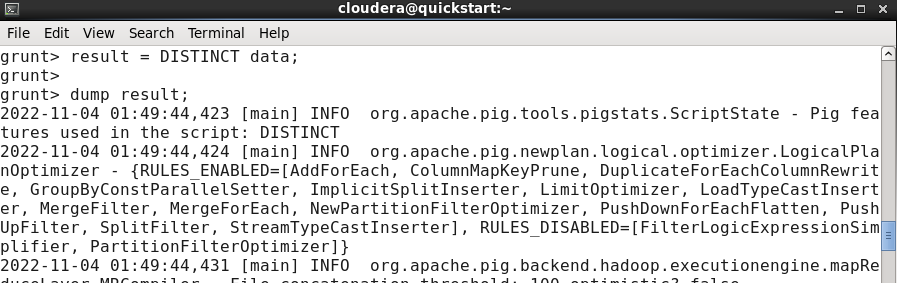


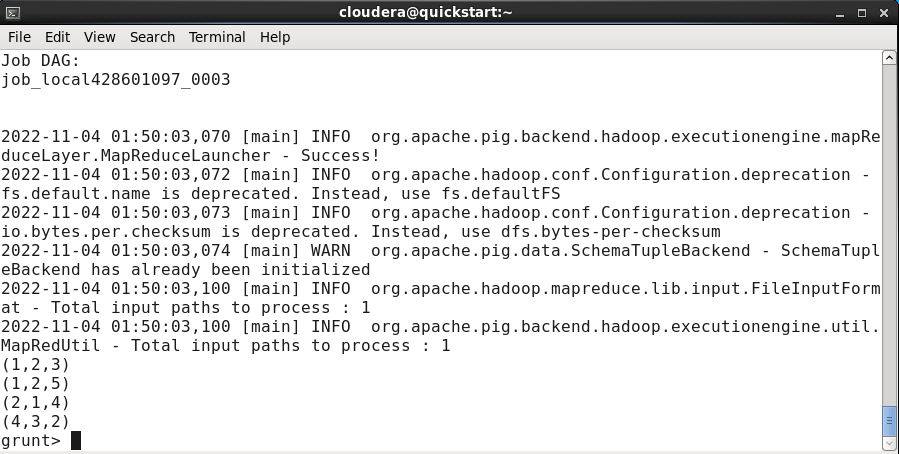
**Filter**



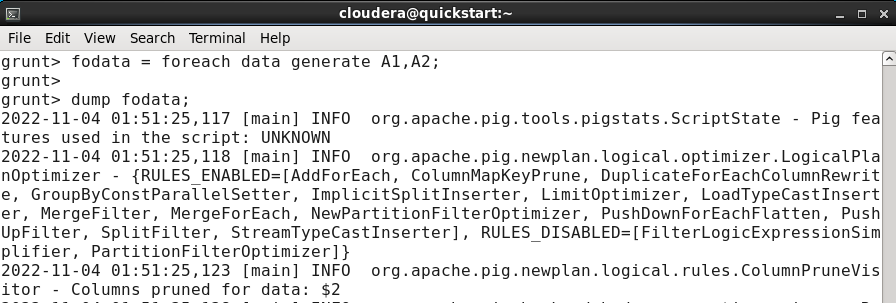


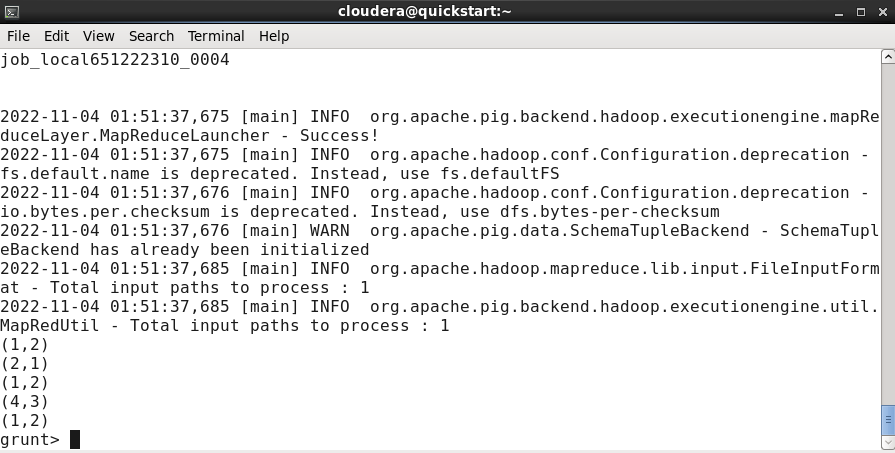
**Distinct :**



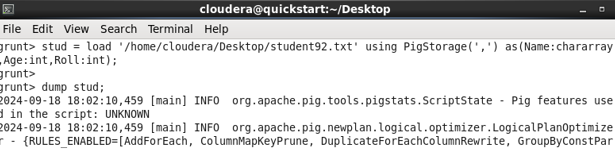


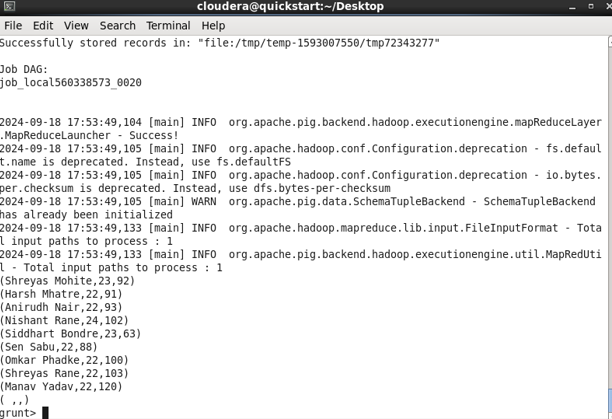
**Foreach:**



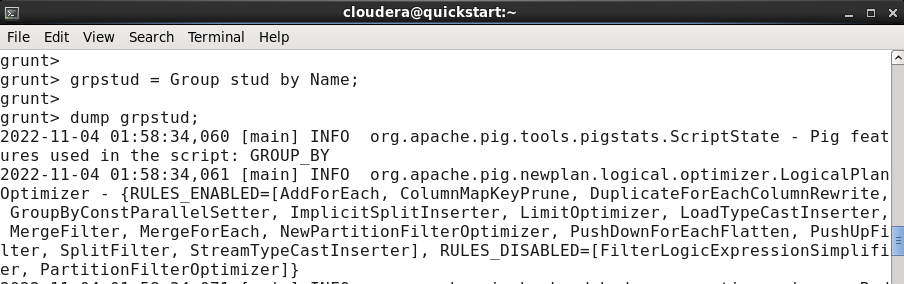


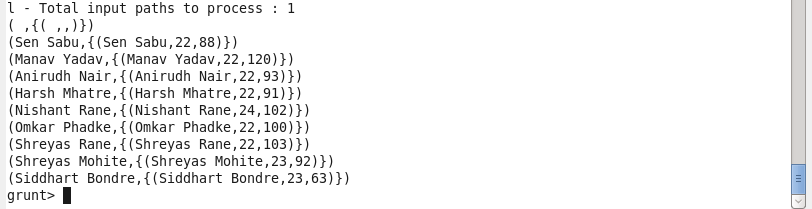
**Load:**

****

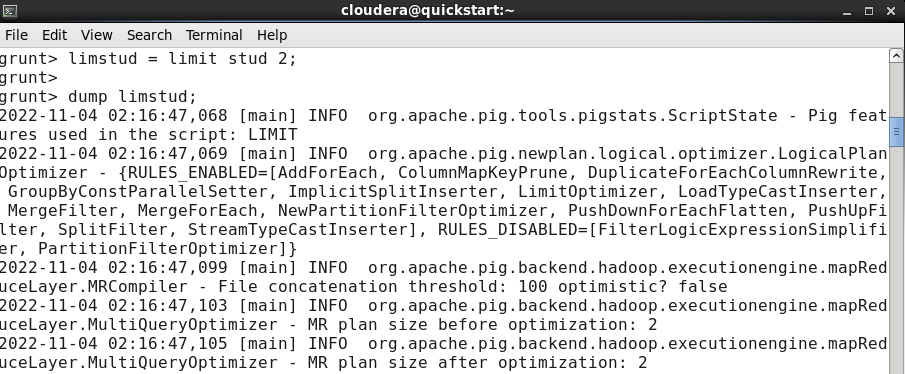
****

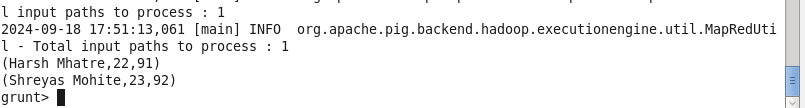
**Group:**



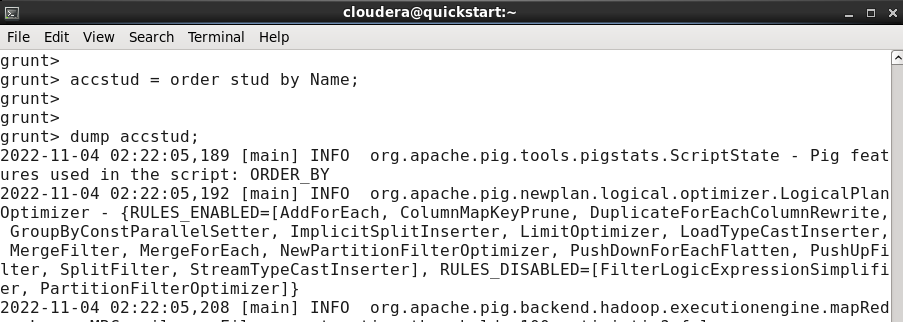
****

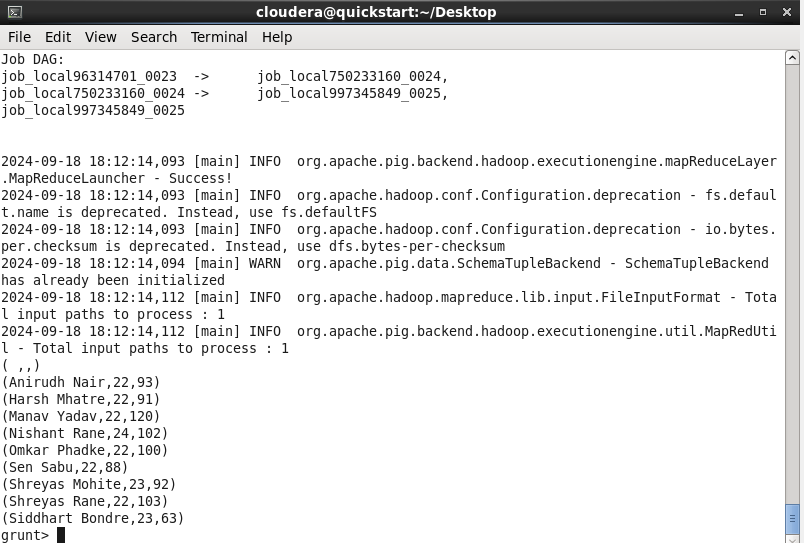
**Limit:**



****

**Orderby:**

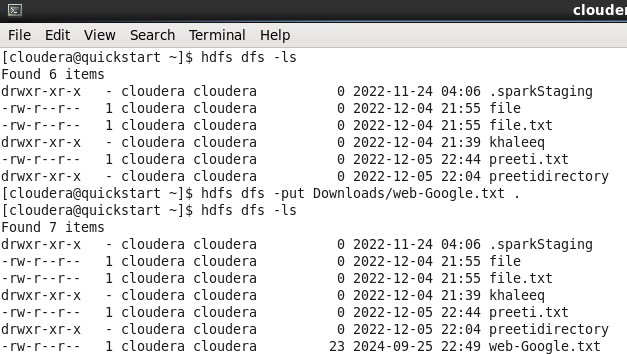


****

# PRACTICAL NO:6

## Aim: Basic Spark Commands

**Move the data into hadoop file system**



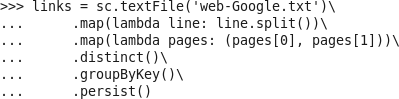
**Start pyspark in terminal**



**Writing compute contrib function**



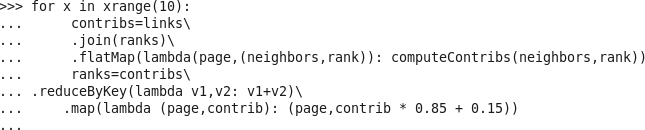
**Create a RDD named links with following command**



**Create a ranks rdd storing the ranks data**



**Create a loop in order to calculate contribs and ranks**



**Code to collect all ranks**

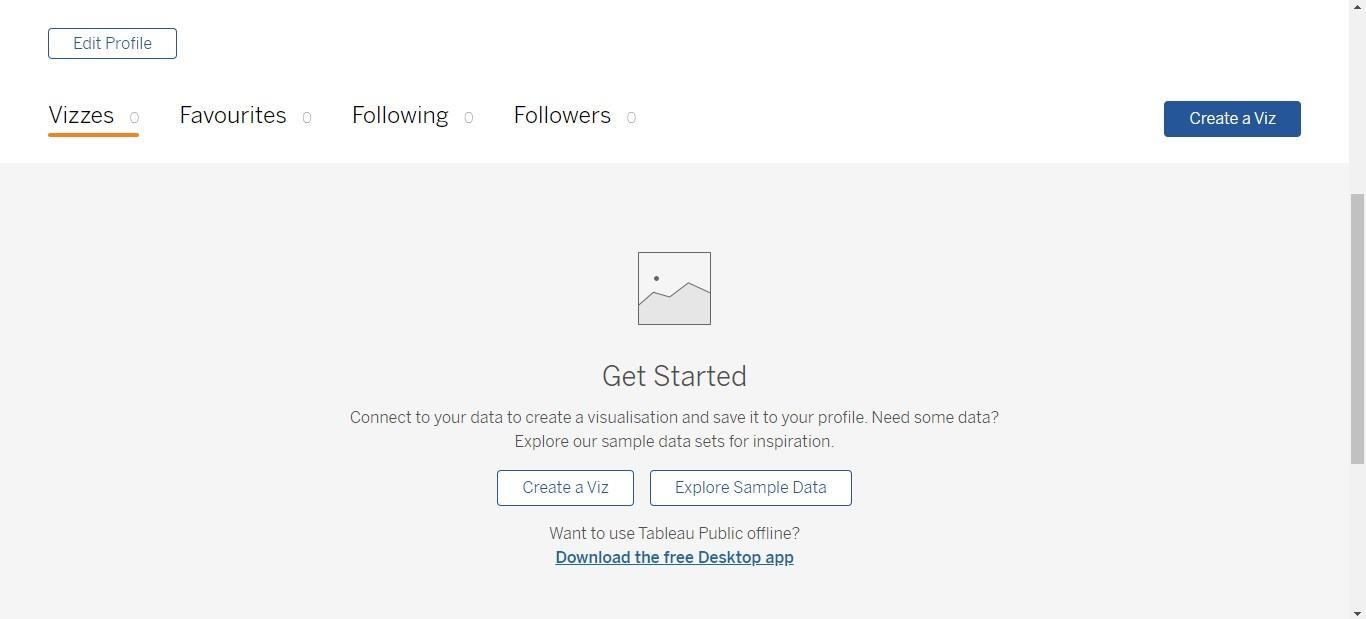


# PRACTICAL NO:7

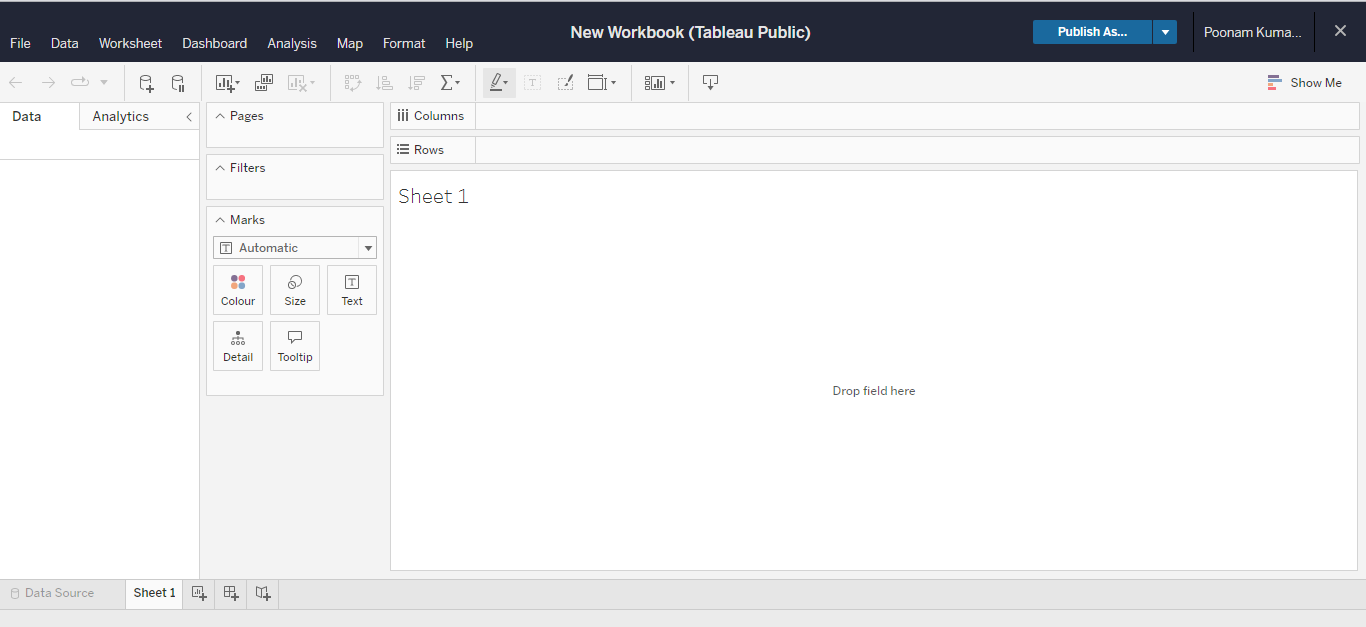
## Aim: To Demonstrate Visualization using Tableau

Create an Account on [Tableau Public](https://public.tableau.com/app/discover)

**Create vizzes**



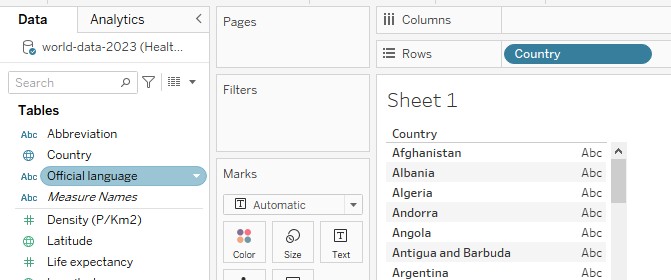
**Importing Data**



**Click on data Now click on file you want to use.**

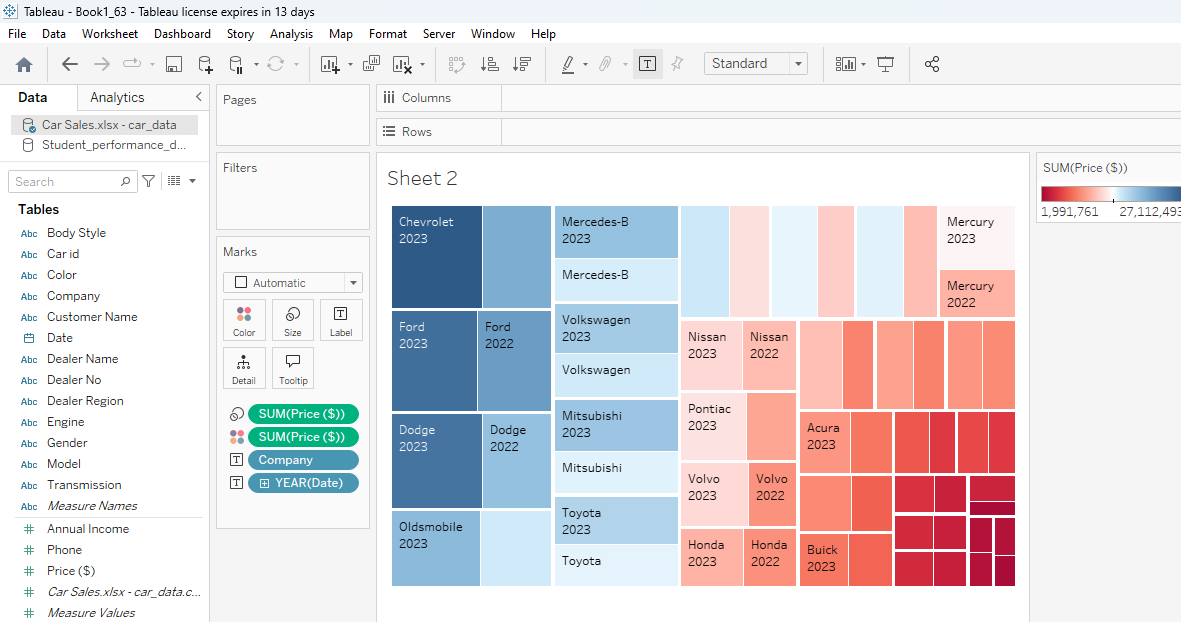
Now wait for it to load completely it may take some time if file is too large.

After the file is loaded click on “Sheet 1” at bottom left of window. Then it will Extract allfile data

**Now drag and drop any field you want to use on row.**

**Analyzing using charts.**

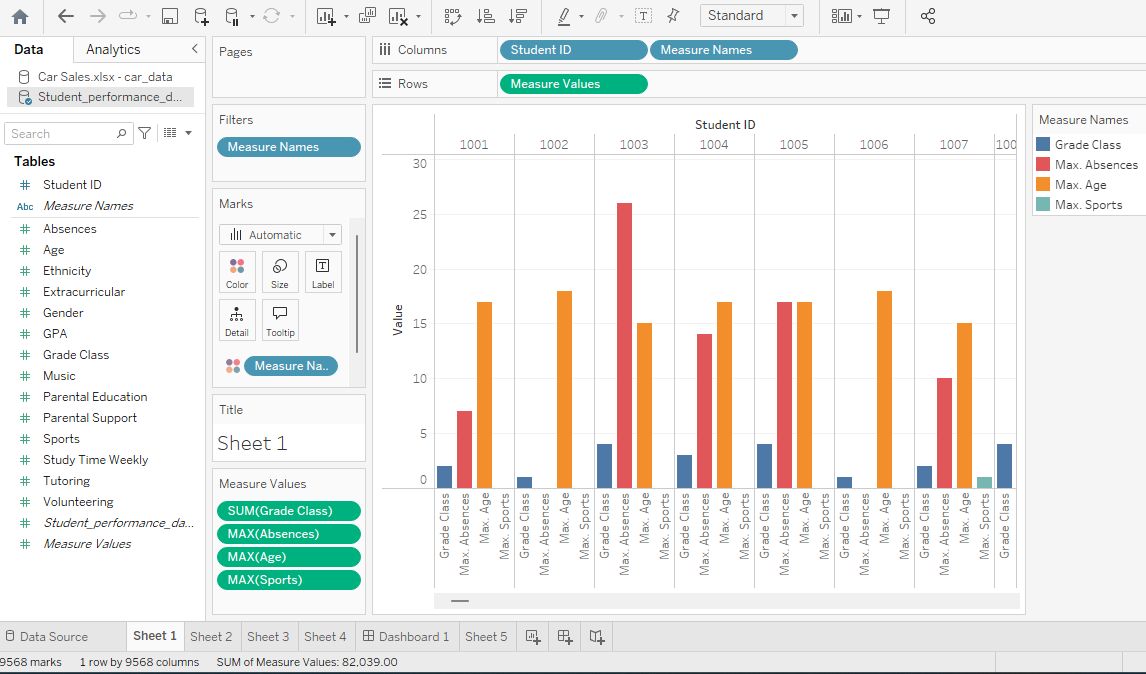
**Treemaps**



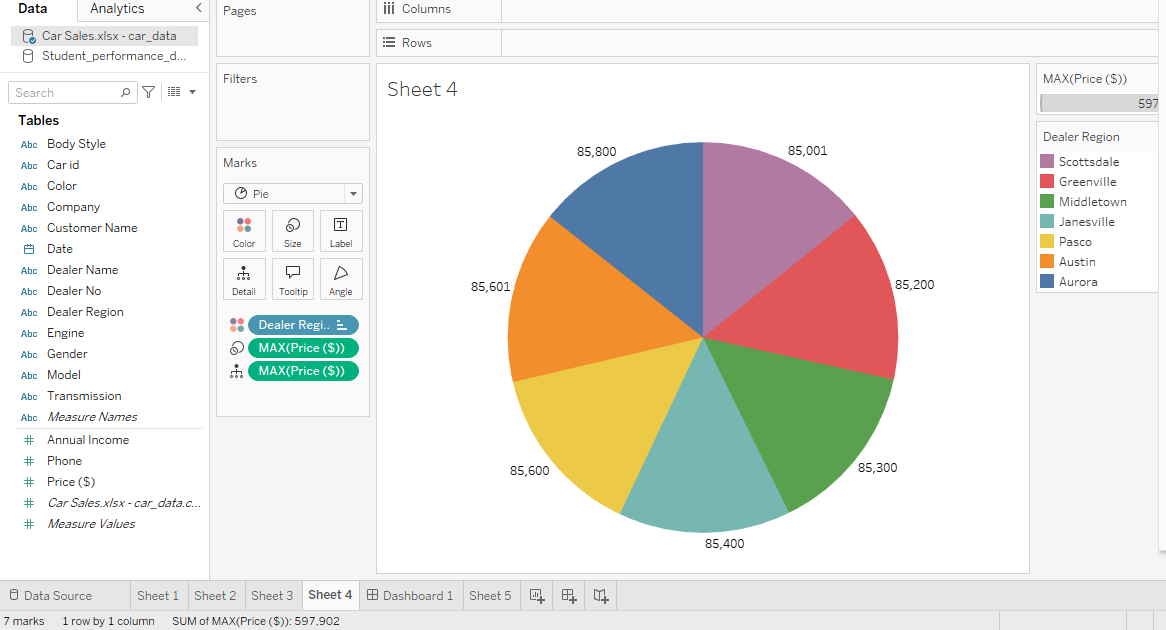
**packed bubbles**

****

**side-by-side bars**

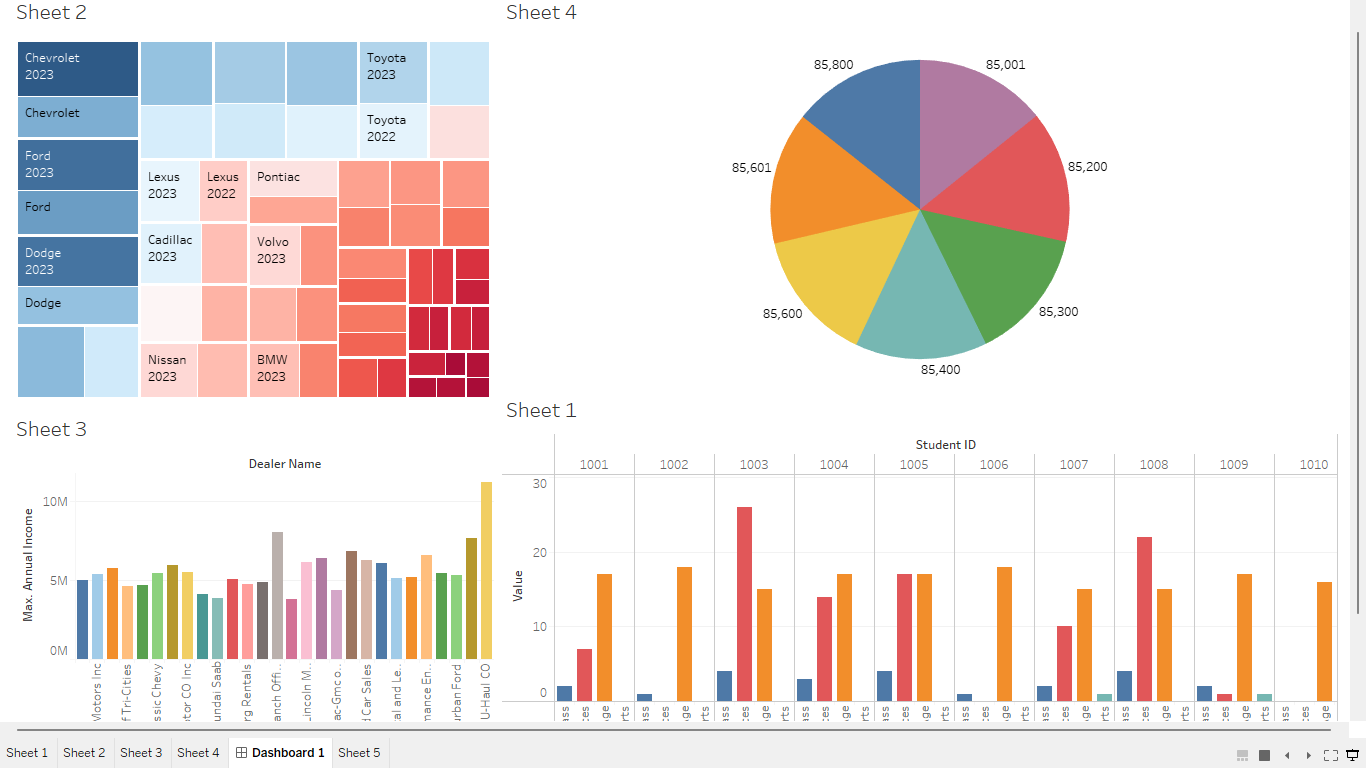


**Pie charts**



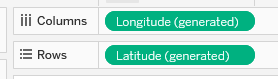
### Creating Dashboards

First drag and drop your sheet from Sheets at left side to Main working space, Add other sheets and arrange them in similar way and your dashboard is ready.

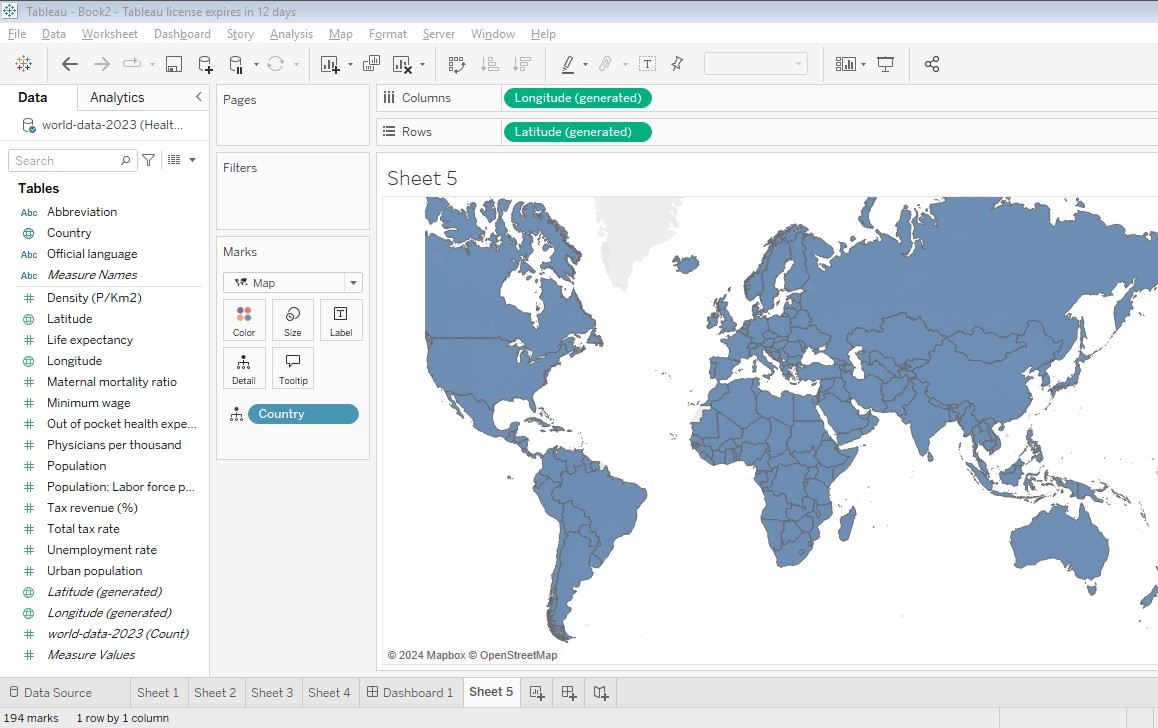


* **Working with maps**

1. Import data and add data as shown below.



1. In this dataset i am using “Country” attribute to be display displayed on given Map.



* + **Telling Stories with Tableau**

