Roll No. **42**

Exam Seat No. _____

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY

Hashu Advani Memorial Complex, Collector's Colony, R. C. Marg,Chembur, Mumbai – 400074. Contact No. 02261532532

Since 1962

CERTIFICATE

Certified that Mr. **CHRISTY PHILIP [ROLL NO: 42]**of **SYMCA-2B** has satisfactorily completed a course of the necessary experiments in **MCAL31 – Big Data Analytics & Visualization Lab** under the supervision of **Dr. Meeanakshi Garg** in the Institute of Technology in the academic year **2022- 2023.**

Principal

Head of Department

Lab In-charge

Subject Teacher

V.E.S. Institute of Technology, Collector Colony, Chembur, Mumbai

Department of M.C.A

MCAL31 – BIG DATA ANALYTICS AND VISUALIZATION LAB INDEX

Sr. No	Contents	Date Of Preparation	Date Of Submission	Marks	Sign
1	Set up and Configuration Hadoop Using Cloudera Creating a HDFS System with minimum 1 Name Node and 1 Data Nodes HDFS Commands.	19/08/2022	26/08/2022	10	
2	Map Reduce Programming: Examples: Matrix Multiplication & Word Count.	26/08/2022	09/09/2022	10	
3	Mongo DB: Installation and Creation of database and Collection, CRUD Document: Insert, Query, Update and Delete Document.	09/09/2022	23/09/2022	10	
4	Hive: Introduction Creation of Database and Table, Hive Partition, Hive Built in Function and Operators, Hive View and Index.	23/09/2022	07/10/2022	10	
5	Pig: Pig Latin Basic Pig Shell, Pig Data Types, Creating a Pig Data Model, Reading and Storing Data, Pig Operations.	07/10/2022	21/10/2022	10	
6	Spark: RDD, Actions and Transformation on RDD , Ways to Create - file, data in memory, other RDD. Lazy Execution, Persisting RDD	21/10/2022	28/10/2022	10	
7	Visualization: Connect to data, Build Charts and Analyze Data, Create Dashboard, Create Stories using Tableau	28/10/2022	18/11/2022	10	
A1	Assignment I: Apache Spark	19/09/2022	26/09/2022	10	
A2	Assignment II: Apache Kafka	21/09/2022	28/09/2022	10	

PRACTICAL 1

AIM: - To execute Hadoop commands using Ubuntu.

1) version

The Hadoop fs shell command version prints the Hadoop version.

Syntax: -hadoop version

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop version
Hadoop 3.3.1
Source code repository https://github.com/apache/hadoop.git -r a3b9c37a397ad4188041dd80621bdeefc46885f2
Compiled by ubuntu on 2021-06-15T05:13Z
Compiled with protoc 3.7.1
From source with checksum 88a4ddb2299aca054416d6b7f81ca55
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.3.1.jar
vaish@vaishVirtualBox:~/CHRISTY$
```

2) mkdir

This command's unique feature is the creation of a directory in the HDFS filesystem cluster if the directory does not exist. Besides, if the specified directory is present, then the output message will show an error signifying the directory's existence.

Syntax: - hdfs dfs -mkdir christyp

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -mkdir christyp
vaish@vaishVirtualBox:~/CHRISTY$
```

3) ls

This command is used to list all the files. Use lsr for recursive approach. It is useful when we want a hierarchy of a folder.

Synatx: - hdfs dfs -ls

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -ls
Found 1 items
drwxr-xr-x   - vaish vaish       4096 2022-09-23 15:48 christyp
vaish@vaishVirtualBox:~/CHRISTY$
```

4) nano

Nano command will help in opening a new editor.

Do Ctrl+x do yes to save file then ctrl +z

Syntax: - nano text1.txt

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -ls
Found 1 items
drwxr-xr-x   - vaish vaish       4096 2022-09-23 15:48 christyp
vaish@vaishVirtualBox:~/CHRISTY$
```

```
 GNU nano 4.8                          text1.txt                          Modified
This is file of nano command
```

5) cat

The cat command reads the file in HDFS and displays the content of the file on console or stdout.

Syntax: - hdfs dfs -cat christyp.txt

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -cat christyp.txt
vaish@vaishVirtualBox:~/CHRISTY$ ls
christyp  christyp.txt  text1.txt
vaish@vaishVirtualBox:~/CHRISTY$
```

6) usage ls

The Hadoop fs shell command usage returns the help for an individual command.

Syntax: - hdfs dfs -usage ls

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -usage ls
Usage: hadoop fs [generic options] -ls [-C] [-d] [-h] [-q] [-R] [-t] [-S] [-r] [-u] [-e] [<path> ...]
vaish@vaishVirtualBox:~/CHRISTY$
```

7) usage test

The Hadoop fs shell command usage returns the help for an individual command.

Syntax: - hdfs dfs -usage test

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -usage test
Usage: hadoop fs [generic options] -test -[defswrz] <path>
vaish@vaishVirtualBox:~/CHRISTY$
```

8) count

The Hadoop fs shell command count counts the number of files, directories, and bytes under the paths that match the specified file pattern.

Options:

-q – shows quotas(quota is the hard limit on the number of names and amount of space used for individual directories)

-u – it limits output to show quotas and usage only

-h – shows sizes in a human-readable format

-v – shows header line

Syntax: - hadoop fs -count -v christyp.txt

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -count -v christyp.txt
   DIR_COUNT   FILE_COUNT       CONTENT_SIZE PATHNAME
           0            1                  0 christyp.txt
vaish@vaishVirtualBox:~/CHRISTY$
```

9) find

The Hadoop fs shell command find finds all files that match the specified expression.

If no path is specified, then it defaults to the present working directory. If an expression is not specified, then it defaults to -print.

Syntax: - hadoop fs -find -name christyp.txt

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -find -name christyp.txt
christyp.txt
vaish@vaishVirtualBox:~/CHRISTY$
```

10) help

Help command will help in getting details of a specific command.

Syntax: - hadoop fs -help cat

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -help cat
-cat [-ignoreCrc] <src> ... :
  Fetch all files that match the file pattern <src> and display their content on
  stdout.
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -help cat
```

11) expunge

This command is used to empty the trash available in an HDFS system.

Syntax: - hadoop fs -expunge

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -expunge
vaish@vaishVirtualBox:~/CHRISTY$
```

12) du

This Hadoop fs shell command du prints a summary of the amount of disk usage of all files/directories in the path.

Syntax: - hdfs dfs -du

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -du
0     0     christyp
1024  1024  .text1.txt.swp
0     0     text1.txt
0     0     christyp.txt
```

13) -du -s

This command is used to show the amount of space in bytes that have been used by the files that match the specified file pattern. Even without the –s option, this only shows the size summaries one level deep in the directory.

-s

Used to show the size of each individual file that matches the pattern, shows the total (summary) size.

Syntax: - hdfs dfs -du -s

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -du -s
1024  1024  .
vaish@vaishVirtualBox:~/CHRISTY$
```

14) stat

The Hadoop fs shell command stat prints the statistics about the file or directory in the specified format.

Syntax: - hadoop fs - stat

Syntax: - hadoop fs -stat / christyp.txt

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -stat /
2021-05-07 14:37:30
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -stat / christyp.txt
2021-05-07 14:37:30
2022-09-23 10:22:14
vaish@vaishVirtualBox:~/CHRISTY$
```

15) touchz

This command allows the user to create a new file in the HDFS cluster. The "directory" in the command refers to the directory name where the user wishes to create the new file, and the "filename" signifies the name of the new file which will be created upon the completion of the command.

Syntax: -hdfs dfs -touchz file_name

```
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -touchz cp.txt
vaish@vaishVirtualBox:~/CHRISTY$ hdfs dfs -ls
Found 5 items
-rw-rw-r--   1 vaish vaish       1024 2022-09-23 15:56 .text1.txt.swp
drwxr-xr-x   - vaish vaish       4096 2022-09-23 15:48 christyp
-rw-rw-r--   1 vaish vaish          0 2022-09-23 15:52 christyp.txt
-rw-r--r--   1 vaish vaish          0 2022-09-23 16:21 cp.txt
-rw-r--r--   1 vaish vaish          0 2022-09-23 15:59 text1.txt
vaish@vaishVirtualBox:~/CHRISTY$
```

16) copyFromLocal

Hadoop copyFromLocal command is used to copy the file from your local file system to the HDFS(Hadoop Distributed File System). copyFromLocal command has an optional switch –f which is used to replace the already existing file in the system, means it can be used to update that file. -f switch is similar to first delete a file and then copying it. If the file is already present in the folder then copy it into the same folder will automatically throw an error.

Syntax: - hadoop fs -copyFromLocal file_name dir_name

```
vaish@vaishVirtualBox:~/CHRISTY$ touch text2.txt
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -copyFromLocal text2.txt files
vaish@vaishVirtualBox:~/CHRISTY$
```

17) test

This particular command fulfills the purpose of testing the existence of a file in the HDFS cluster. The characters from "[defsz]" in the command have to be modified as needed. Here is a brief description of these characters:

d -> Checks if it is a directory or not

e -> Checks if it is a path or not

f -> Checks if it is a file or not

s -> Checks if it is an empty path or not

r -> Checks the path existence and read permission

w -> Checks the path existence and write permission

z -> Checks the file size

Syntax: -hadoop fs -test -d file_name

```
vaish@vaishVirtualBox:~/CHRISTY$ hadoop fs -test -d christyp.txt
vaish@vaishVirtualBox:~/CHRISTY$
```

CONCLUSION: -

From this practical I have learned to execute Hadoop commands using Ubuntu.

PRACTICAL 2

AIM: - Map Reduce Programming Examples Matrix Multiplication. Word Count.

THEORY: -

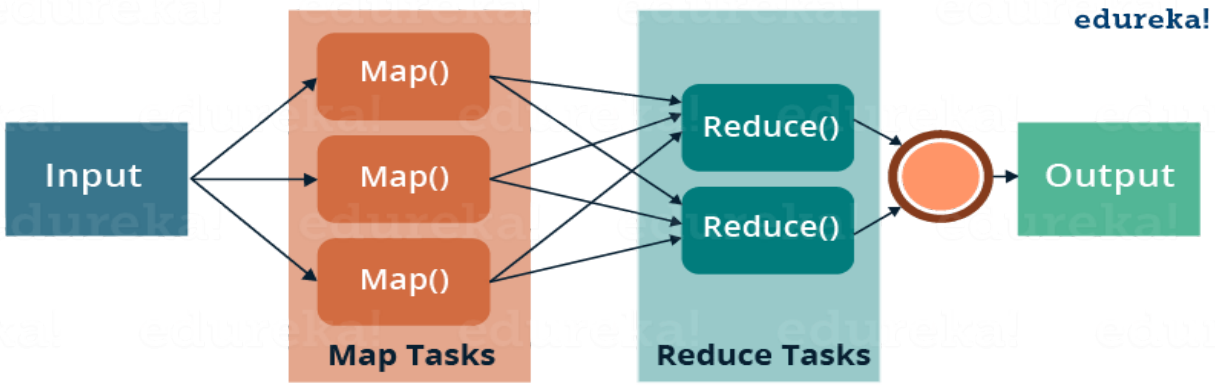
What is MapReduce?

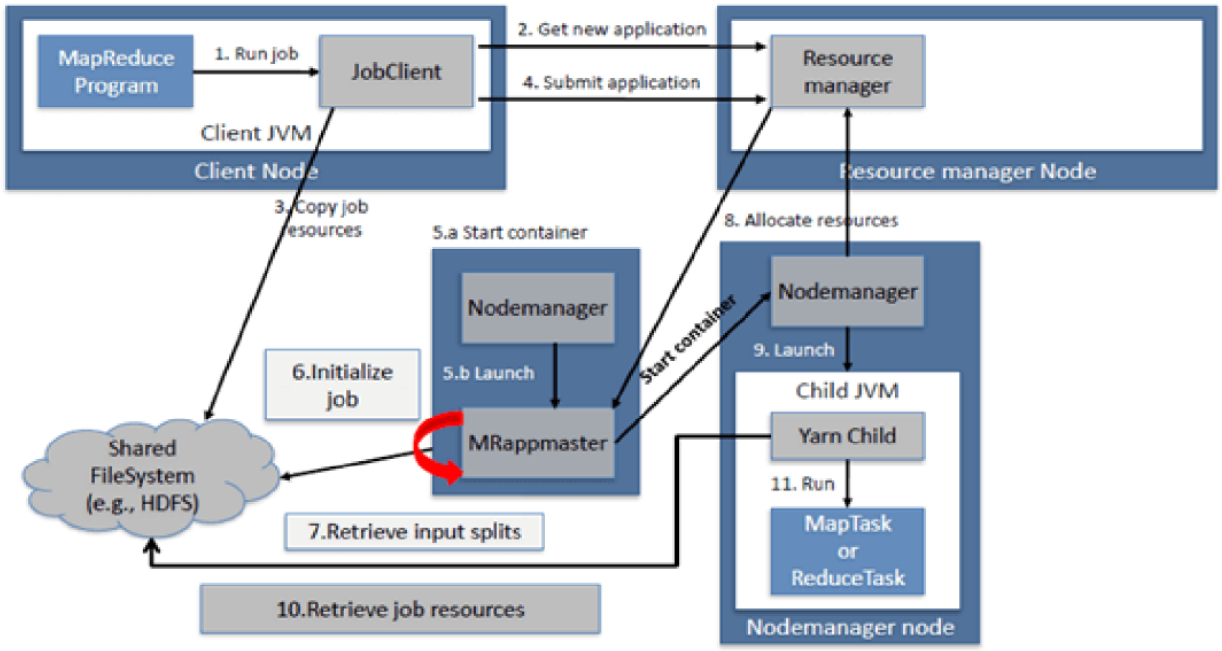
A MapReduce is a data processing tool which is used to process the data parallely in a distributed form. It was developed in 2004, on the basis of a paper titled as "MapReduce: Simplified Data Processing on Large Clusters," published by Google.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of the reducer is the final output.

Steps in Map Reduce

- The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
- An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.





Sort and Shuffle

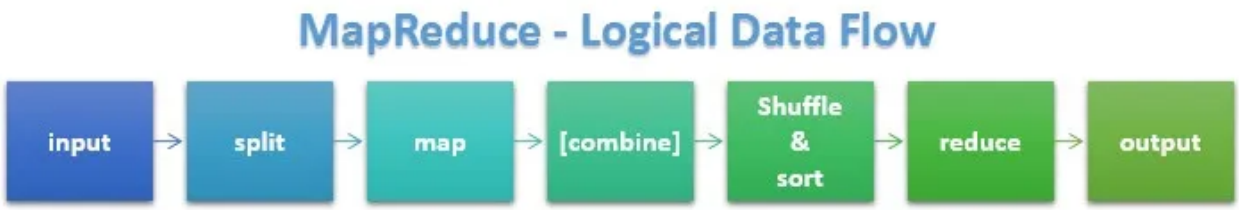
The sort and shuffle occur on the output of Mapper and before the reducer. When the Mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. Using the input from each Mapper <k2,v2>, we collect all the values for each unique key k2. This output from the shuffle phase in the form of <k2, list(v2)> is sent as input to reducer phase.

Usage of MapReduce

- It can be used in various applications like document clustering, distributed sorting, and web link-graph reversal.
- It can be used for distributed pattern-based searching.
- We can also use MapReduce in machine learning.
- It was used by Google to regenerate Google's index of the World Wide Web.
- It can be used in multiple computing environments such as multi-cluster, multi-core, and mobile environments.

Data Flow In MapReduce

MapReduce is used to compute huge amounts of data . To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.



Phases of MapReduce data flow

1) Input reader

The input reader reads the upcoming data and splits it into the data blocks of the appropriate size (64MB to 128 MB). Each data block is associated with a Map function. Once input reads the data, it generates the corresponding key-value pairs. The input files reside in HDFS.

2) Map function

The map function processes the upcoming key-value pairs and generates the corresponding output key-value pairs. The map input and output type may be different from each other.

3) Partition function

The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide this function. It returns the index of reducers.

4) Shuffling and Sorting

The data is shuffled between/within nodes so that it moves out from the map and get ready to process for reduced function. Sometimes, the shuffling of data can take much computation time. The sorting operation is performed on input data for the Reduce function. Here, the data is compared using a comparison function and arranged in a sorted form.

5) Reduce function

The Reduce function is assigned to each unique key. These keys are already arranged in sorted order. The values associated with the keys can iterate the Reduce and generates the corresponding output

6) Output writer

Once the data flow from all the above phases, Output writer executes. The role of Output writer is to write the Reduce output to the stable storage

A) WORD COUNT: -

CODE: -

WC_Mapper.java: -

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
public class WC_Mapper extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable> {
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
@Override
public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter) throws IOException {
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
word.set(tokenizer.nextToken());
output.collect(word, one);
}
}
}
```

WC_Reducer.java: -

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
```

```
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
public class WC_Reducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {
@Override
public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
output,
Reporter reporter) throws IOException {
int sum = 0;
while (values.hasNext()) {
sum += values.next().get();
}
output.collect(key, new IntWritable(sum));
}
}
```

WC_Runner.java: -

```
import java.io.IOException;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class WC_Runner {
public static void main(String[] args) throws IOException {
JobConf conf = new JobConf(WC_Runner.class);
conf.setJobName("WordCount");
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(WC_Mapper.class);
conf.setCombinerClass(WC_Reducer.class);
conf.setReducerClass(WC_Reducer.class);
conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);
```

```
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
JobClient.runJob(conf);
}
}
```

STEPS: -

1) Write above code in Netbeans, import jar file while writing code in Netbeans hadoop-core-1.1.2.jar

2) After the code is written right click project and do clean build

3) Locate the jar file file in the dist folder by going to project properties and referring to the path.

4) Upload the jar file in drive

5) Now in Ubuntu download the jar file from drive

6) Execute the commands and we will need a text file for the word count because the count is done of the file.

OUTPUT: -

Commands and Outputs

1) ./start-all.sh

```
narender@narender-VirtualBox:~/hadoop-2.7.3/sbin$ ./start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
narender@localhost's password:
localhost: starting namenode, logging to /home/narender/hadoop-2.7.3/logs/hadoop-narender-namenode-naren
der-VirtualBox.out
narender@localhost's password:
localhost: starting datanode, logging to /home/narender/hadoop-2.7.3/logs/hadoop-narender-datanode-naren
der-VirtualBox.out
Starting secondary namenodes [0.0.0.0]
narender@0.0.0.0's password:
0.0.0.0: starting secondarynamenode, logging to /home/narender/hadoop-2.7.3/logs/hadoop-narender-seconda
rynamenode-narender-VirtualBox.out
starting yarn daemons
starting resourcemanager, logging to /home/narender/hadoop-2.7.3/logs/yarn-narender-resourcemanager-nare
nder-VirtualBox.out
narender@localhost's password:
localhost: starting nodemanager, logging to /home/narender/hadoop-2.7.3/logs/yarn-narender-nodemanager-n
arender-VirtualBox.out
```

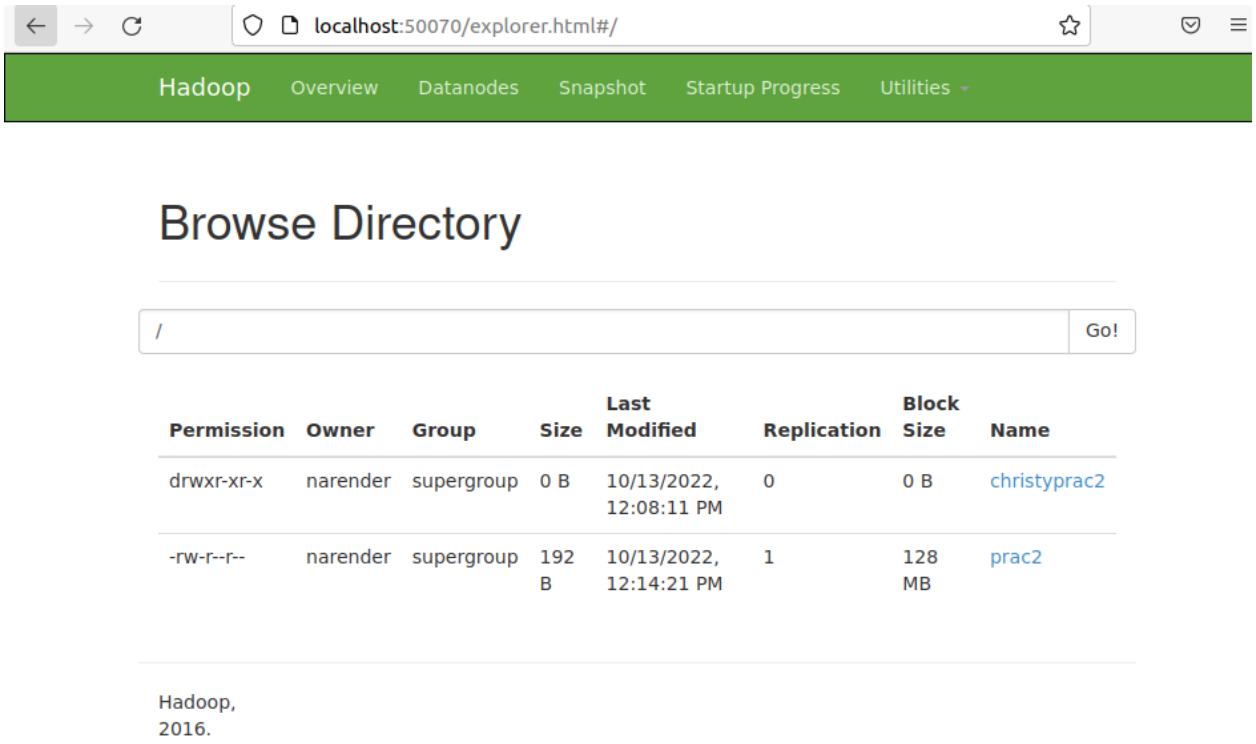
2) jps

```
narender@narender-VirtualBox:~/hadoop-2.7.3/sbin$ jps
4769 Jps
4660 NodeManager
4215 SecondaryNameNode
4343 ResourceManager
3864 NameNode
4025 DataNode
```

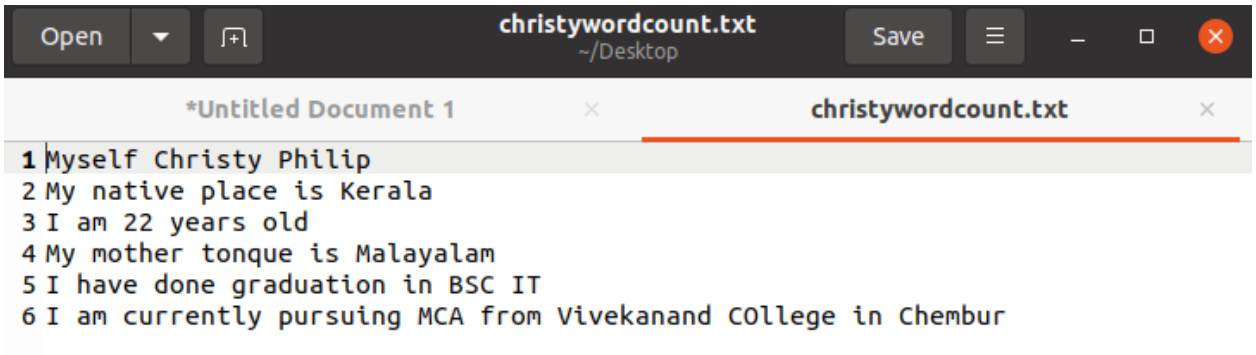
3) hadoop fs -mkdir /christyprac2

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop fs -mkdir /prac2
```

4) Now in localhost:50070 it may vary also system to system. So directory prac2 has been created



5) Now create a txt file with some contents for the word count.



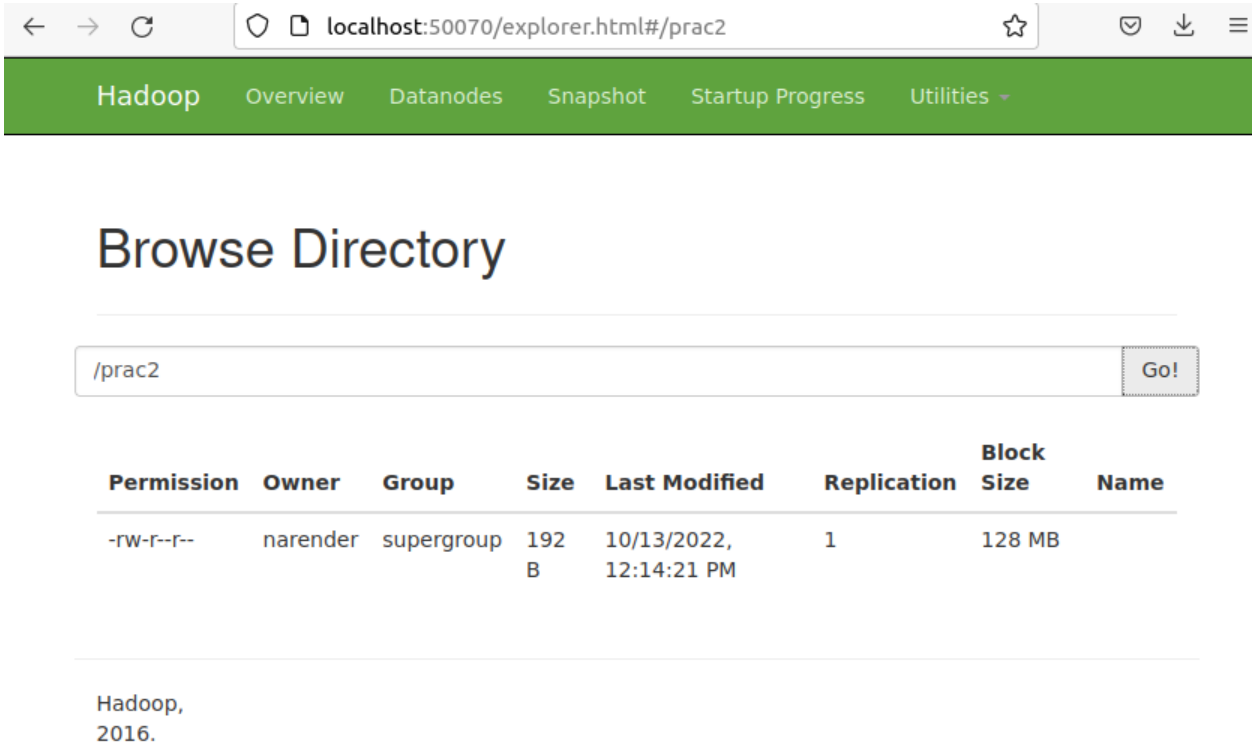
6) Now we need to move the file from local to hdfs. So we will do it using put command
hadoop dfs -put /home/narender/Desktop/christywordcount.txt

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -put /home/narender/Desktop/christywordcount.txt
 /prac2
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
```

7) To list and check the directory
hadoop dfs -ls /prac2

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -ls /prac2
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

-rw-r--r--   1 narender supergroup        192 2022-10-13 12:14 /prac2
```

8) So in web the file is shown



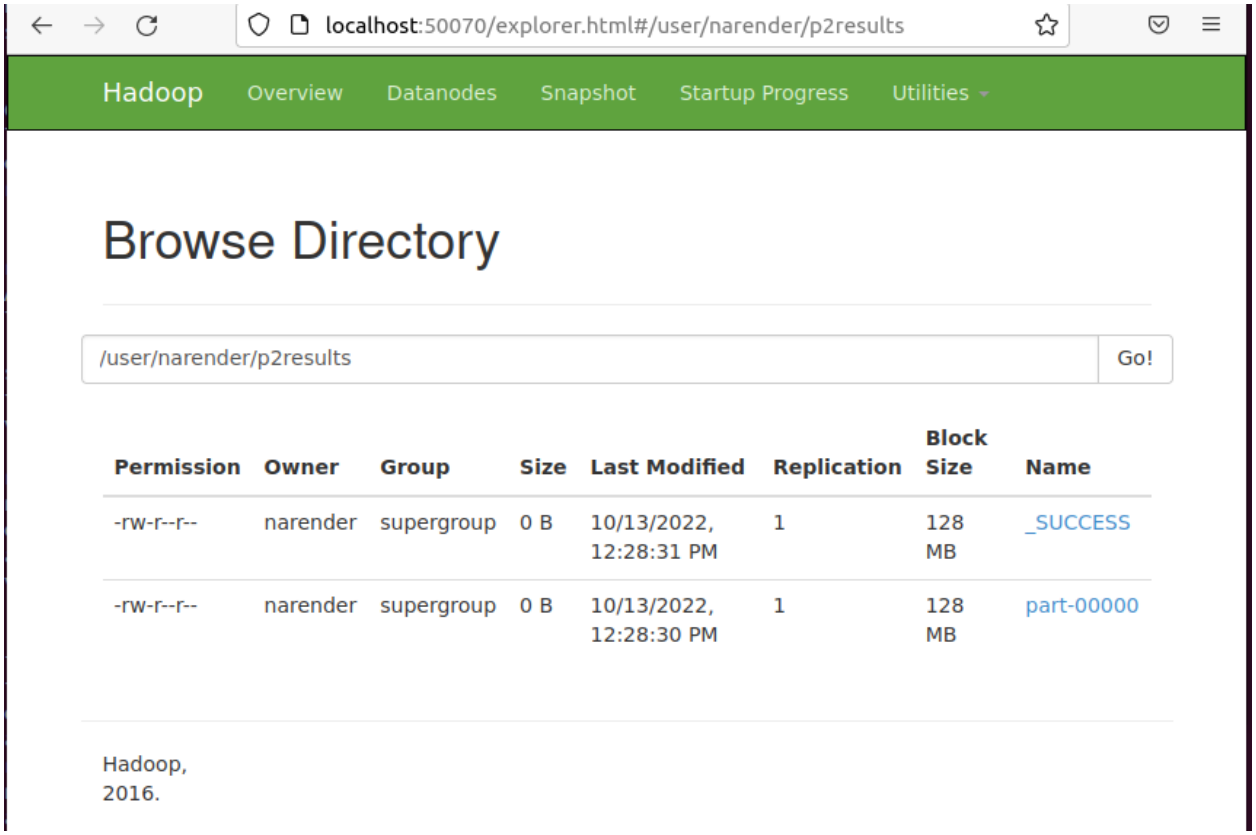
9) Now the main process of executing the jar is done.
Format: -
hadoop jar path_of_jar name_of_class /dir/output name_of_file_for_output
hadoop dfs -cat /user/narender/p2results1/part-00000

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop jar /home/narender/Downloads/WordCountExample.jar WC
_Runner /prac2 p2results1
22/10/13 13:07:04 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/10/13 13:07:04 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/10/13 13:07:05 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your application with ToolRunner to remedy this.
22/10/13 13:07:05 INFO mapred.FileInputFormat: Total input paths to process : 1
22/10/13 13:07:05 INFO mapreduce.JobSubmitter: number of splits:2
22/10/13 13:07:05 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1665669989095_0004
22/10/13 13:07:06 INFO impl.YarnClientImpl: Submitted application application_1665669989095_0004
22/10/13 13:07:06 INFO mapreduce.Job: The url to track the job: http://narender-VirtualBox:8088/proxy/ap
plication_1665669989095_0004/
22/10/13 13:07:06 INFO mapreduce.Job: Running job: job_1665669989095_0004
22/10/13 13:07:17 INFO mapreduce.Job: Job job_1665669989095_0004 running in uber mode : false
22/10/13 13:07:17 INFO mapreduce.Job:  map 0% reduce 0%
22/10/13 13:07:31 INFO mapreduce.Job:  map 100% reduce 0%
22/10/13 13:07:39 INFO mapreduce.Job:  map 100% reduce 100%
22/10/13 13:07:40 INFO mapreduce.Job: Job job_1665669989095_0004 completed successfully
22/10/13 13:07:40 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=382
                FILE: Number of bytes written=357916
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=446
                HDFS: Number of bytes written=234
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=22290
```

```
                Combine output records=32
                Reduce input groups=29
                Reduce shuffle bytes=388
                Reduce input records=32
                Reduce output records=29
                Spilled Records=64
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=441
                CPU time spent (ms)=1770
                Physical memory (bytes) snapshot=542560256
                Virtual memory (bytes) snapshot=5515702272
                Total committed heap usage (bytes)=355999744
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=288
        File Output Format Counters
                Bytes Written=234
```

10) Now we will see the final output on cmd as well as on web

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -cat /user/narender/p2results1/part-00000
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

22      1
BSC     1
COllege 1
Chembur 1
Christy 1
I       3
IT      1
Kerala  1
MCA     1
Malayalam       1
My      2
Myself  1
Philip  1
Vivekanand      1
am      2
currently       1
done    1
from    1
graduation      1
have    1
in      2
is      2
mother  1
native  1
old     1
place   1
pursuing        1
tonque  1
years   1
```



B)

CODE: -

MatrixMultiply.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.DoubleWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Writable;
import 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("/prac4/input/M");
Path NPath = new Path("/prac4/input/N");
Path intermediatePath = new Path("/prac4/interim");
Path outputPath = new Path("/prac4/output");
Job job1 = new Job(new Configuration());
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 = new Job(new Configuration());
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);
}
}
```

STEPS: -

1) Write above code in Netbeans, import jar file while writing code in Netbeans hadoop-core-1.1.2.jar

2) After the code is written right click project and do clean build

3) Locate the jar file file in the dist folder by going to project properties and referring to the path.

4) Upload the jar file in drive

5) Now in Ubuntu download the jar file from drive

6) Execute the commands and we will need two text files for the matrix multiplication because both the matrices of different files are multiplied.

OUTPUT: -

Commands and Output

1) We are creating four directories by using four commands shown below

hadoop fs -mkdir /prac4
hadoop fs -mkdir /prac4/input
hadoop fs -mkdir /prac4/input/M
hadoop fs -mkdir /prac4/input/N

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop fs -mkdir /prac4
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop fs -mkdir /prac4/input
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop fs -mkdir /prac4/input/M
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop fs -mkdir /prac4/input/N
narender@narender-VirtualBox:~/hadoop-2.7.3$
```

2) Now we will put cpm-matrix and cpn-matrix in HDFS using put command

hadoop dfs -put /home/narender/cpm-matrix.txt /prac4/input/M

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -put /home/narender/cpm-matrix.txt /prac4/input/
M
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
```

hadoop dfs -put /home/narender/cpn-matrix.txt /prac4/input/N

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -put /home/narender/cpn-matrix.txt /prac4/input/
N
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
```

3) Now we will check the input folder using ls command

hadoop dfs -ls /prac4/input

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -ls /prac4/input
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 2 items
drwxr-xr-x   - narender supergroup          0 2022-10-13 16:01 /prac4/input/M
drwxr-xr-x   - narender supergroup          0 2022-10-13 16:01 /prac4/input/N
```

4) Now we will check the M folder inside the input folder
hadoop dfs -ls /prac4/input/M

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -ls /prac4/input/M
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 1 items
-rw-r--r--   1 narender supergroup         92 2022-10-13 16:01 /prac4/input/M/cpm-matrix.txt
```

5) Now we will check the N folder inside the input folder
hadoop dfs -ls /prac4/input/N

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -ls /prac4/input/N
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 1 items
-rw-r--r--   1 narender supergroup         88 2022-10-13 16:01 /prac4/input/N/cpn-matrix.txt
```

6) Now the main process of executing the jar is done.
Format: -
hadoop jar path_of_jar name_of_class /dir/output name_of_file_for_output
Command: -
hadoop jar /home/narender/Downloads/MatrixMultiply.jar MatrixMultiply

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop jar /home/narender/Downloads/MatrixMultiply.jar Matr
ixMultiply
22/10/13 16:03:58 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/10/13 16:03:59 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed.
Implement the Tool interface and execute your application with ToolRunner to remedy this.
22/10/13 16:03:59 INFO input.FileInputFormat: Total input paths to process : 1
22/10/13 16:03:59 INFO input.FileInputFormat: Total input paths to process : 1
22/10/13 16:03:59 INFO mapreduce.JobSubmitter: number of splits:2
22/10/13 16:04:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1665669989095_0006
22/10/13 16:04:00 INFO impl.YarnClientImpl: Submitted application application_1665669989095_0006
22/10/13 16:04:00 INFO mapreduce.Job: The url to track the job: http://narender-VirtualBox:8088/proxy/ap
plication_1665669989095_0006/
22/10/13 16:04:00 INFO mapreduce.Job: Running job: job_1665669989095_0006
22/10/13 16:04:10 INFO mapreduce.Job: Job job_1665669989095_0006 running in uber mode : false
22/10/13 16:04:10 INFO mapreduce.Job:  map 0% reduce 0%
22/10/13 16:04:23 INFO mapreduce.Job:  map 100% reduce 0%
22/10/13 16:04:31 INFO mapreduce.Job:  map 100% reduce 100%
22/10/13 16:04:31 INFO mapreduce.Job: Job job_1665669989095_0006 completed successfully
22/10/13 16:04:31 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=490
                FILE: Number of bytes written=359527
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=686
                HDFS: Number of bytes written=321
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
```

```
                Total vcore-milliseconds taken by all reduce tasks=4997
                Total megabyte-milliseconds taken by all map tasks=4487168
                Total megabyte-milliseconds taken by all reduce tasks=5116928
        Map-Reduce Framework
                Map input records=33
                Map output records=33
                Map output bytes=528
                Map output materialized bytes=600
                Input split bytes=113
                Combine input records=0
                Combine output records=0
                Reduce input groups=9
                Reduce shuffle bytes=600
                Reduce input records=33
                Reduce output records=9
                Spilled Records=66
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=127
                CPU time spent (ms)=1080
                Physical memory (bytes) snapshot=329244672
                Virtual memory (bytes) snapshot=3679305728
                Total committed heap usage (bytes)=202379264
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=321
        File Output Format Counters
                Bytes Written=89
```

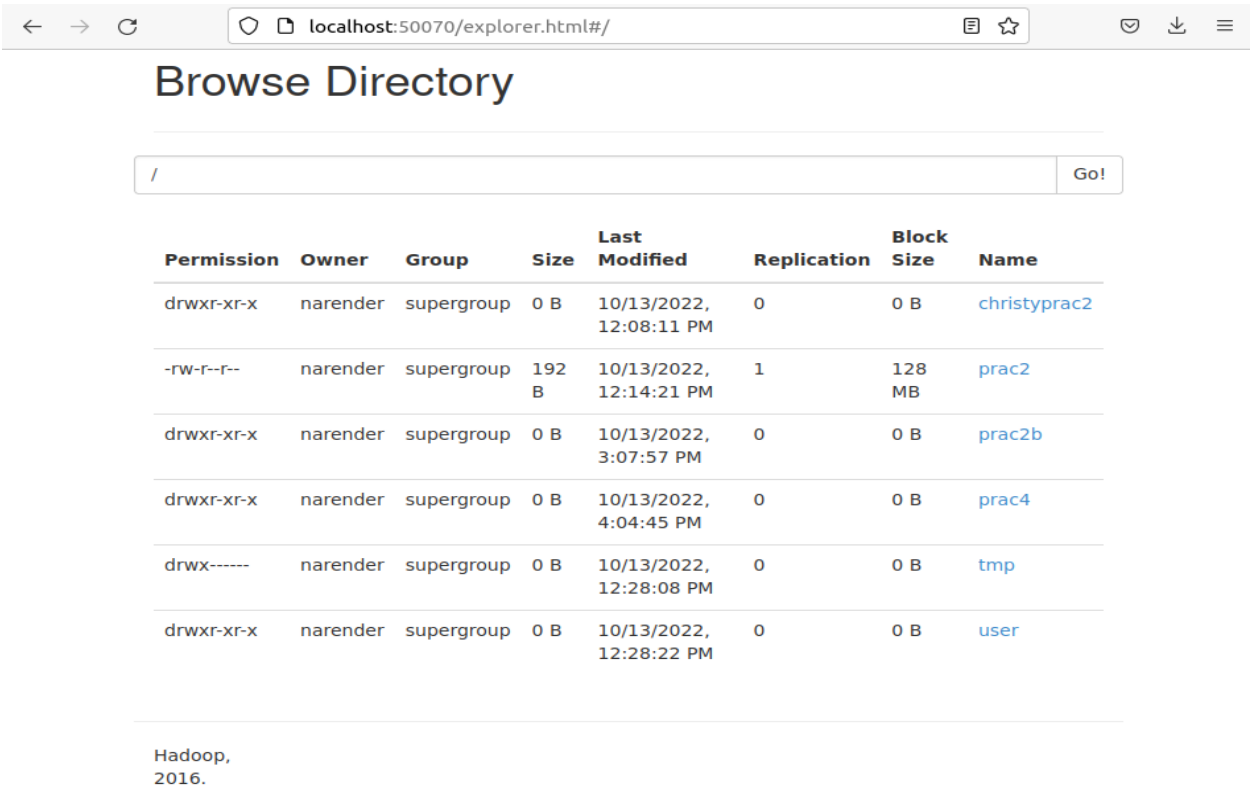
7) Now here we check the intermediate output
hadoop dfs -ls /prac4/interim

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -ls /prac4/interim
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

Found 2 items
-rw-r--r--   1 narender supergroup          0 2022-10-13 16:04 /prac4/interim/_SUCCESS
-rw-r--r--   1 narender supergroup        321 2022-10-13 16:04 /prac4/interim/part-r-00000
```

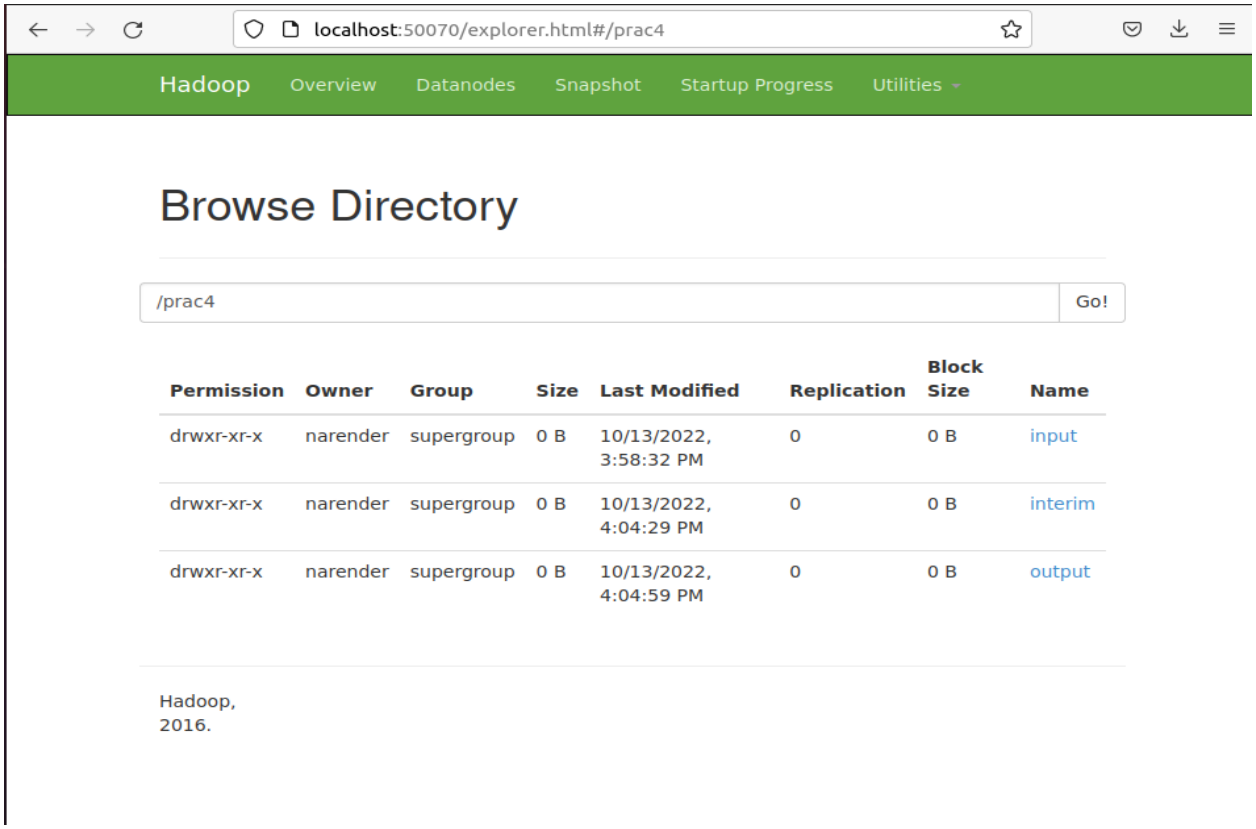
8) Now the output is finally viewed using below shown command
hadoop dfs -cat /prac4/interim/part-r-00000

```
narender@narender-VirtualBox:~/hadoop-2.7.3$ hadoop dfs -cat /prac4/interim/part-r-00000
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

1 2     -3.0
1 0     -3.0
1
1
0 2     -2.0
0 0     -2.0
0 0     -4.0
0 0     -4.0
0 2     5.0
0 0     5.0
0 0     10.0
0 0     10.0
3 3     21.0
3 2     3.0
3 2     9.0
1 3     21.0
1 2     3.0
1 2     9.0
3 3     28.0
3 2     4.0
3 2     12.0
1 3     -21.0
1 2     -3.0
1 2     -9.0
2 2     -6.0
2 1     -4.0
2 1     -6.0
2 2     6.0
2 1     4.0
2 1     6.0
2 2     18.0
2 1     12.0
2 1     18.0
narender@narender-VirtualBox:~/hadoop-2.7.3$
```

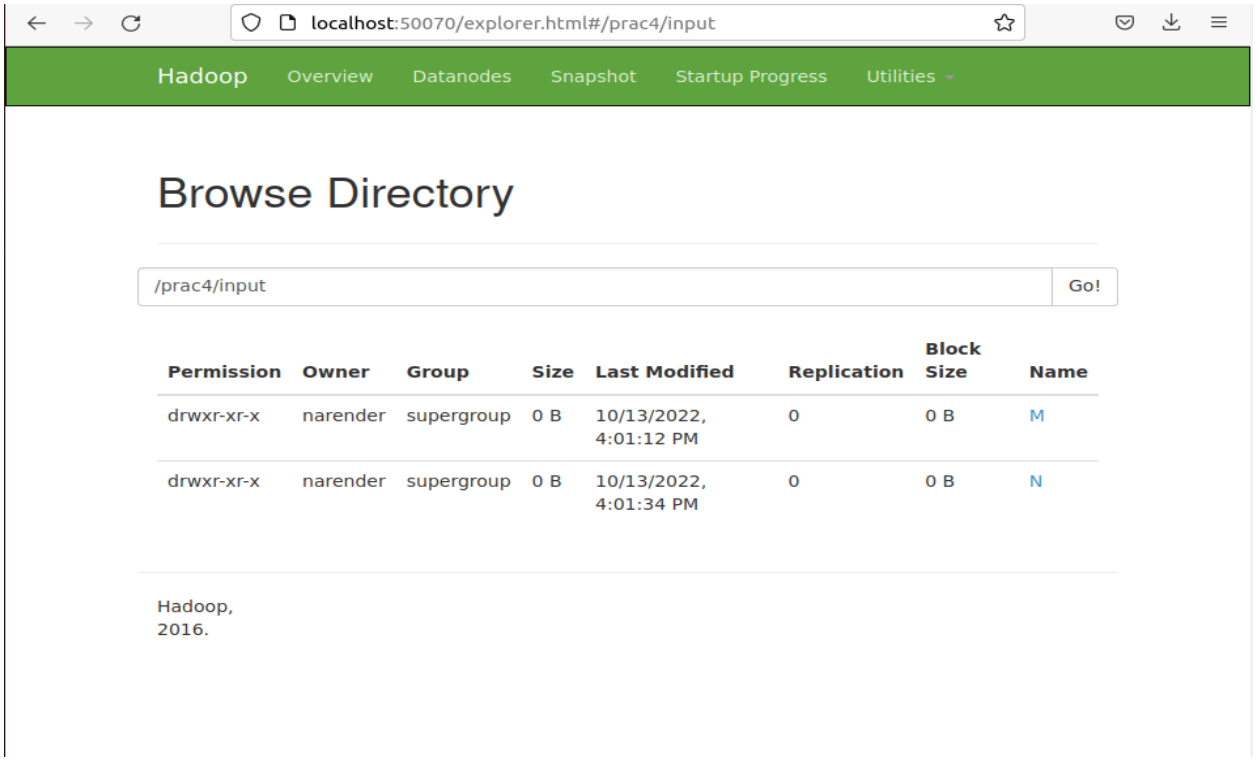
Search for localhost:50070 in the Ubuntu machine browser search
Note: -
This localhost may vary too based on different machines



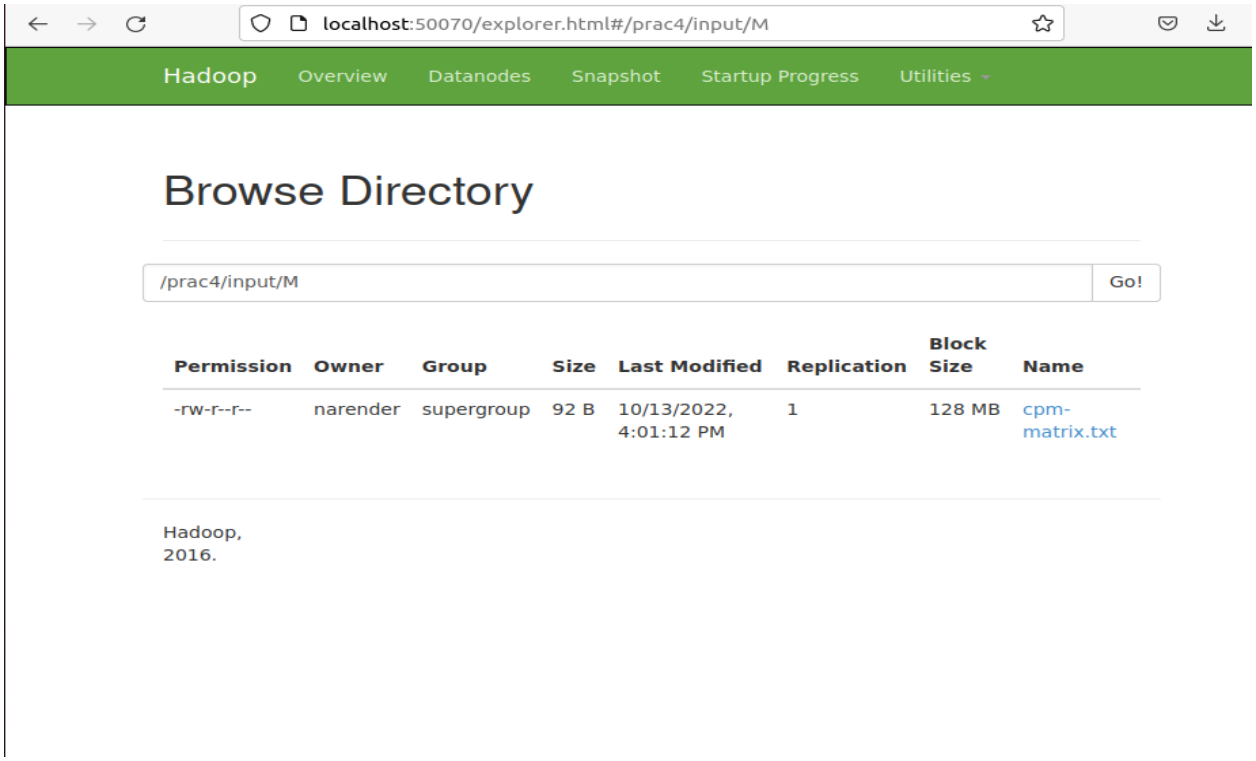
Inside /prac4 folder



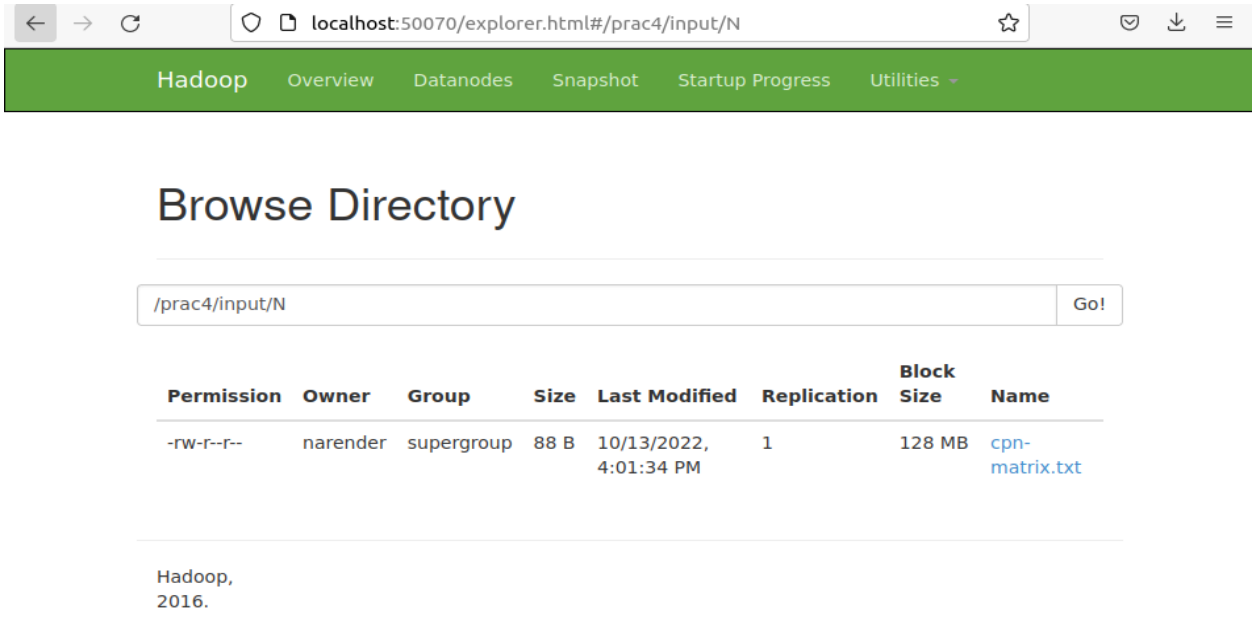
Inside /prac4/input folder



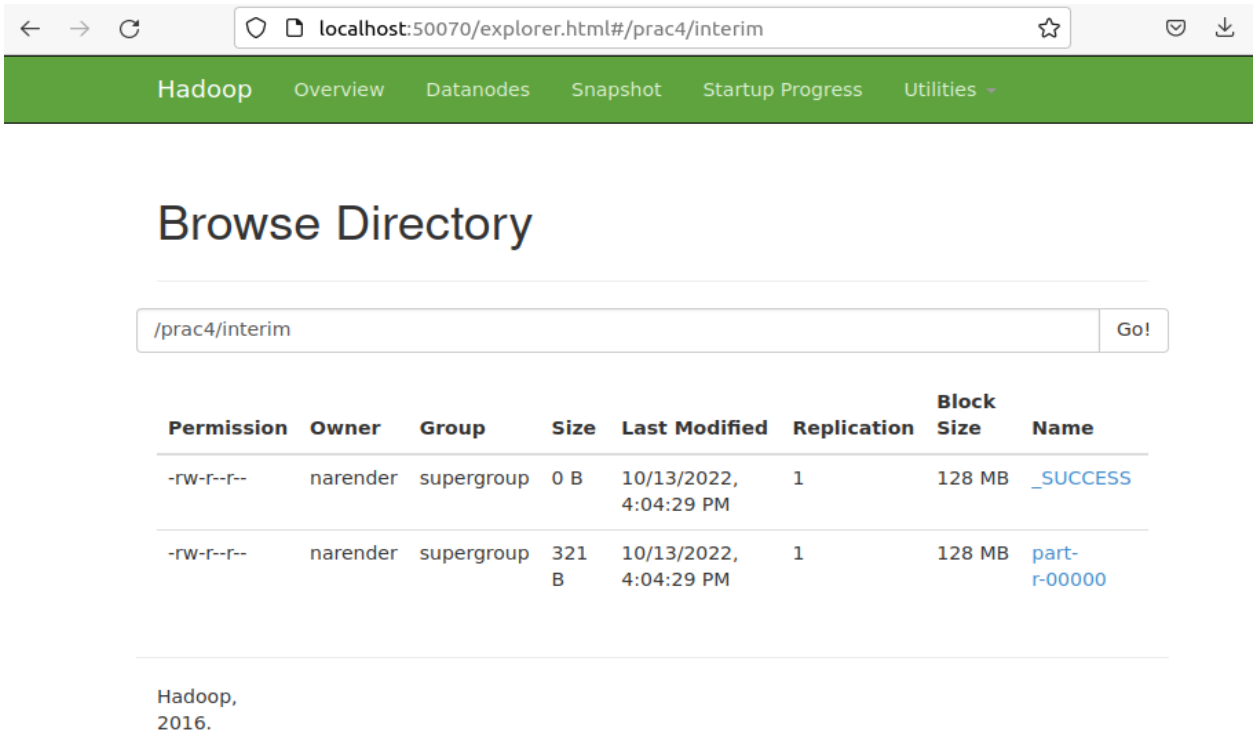
Inside /prac4/input/M folder



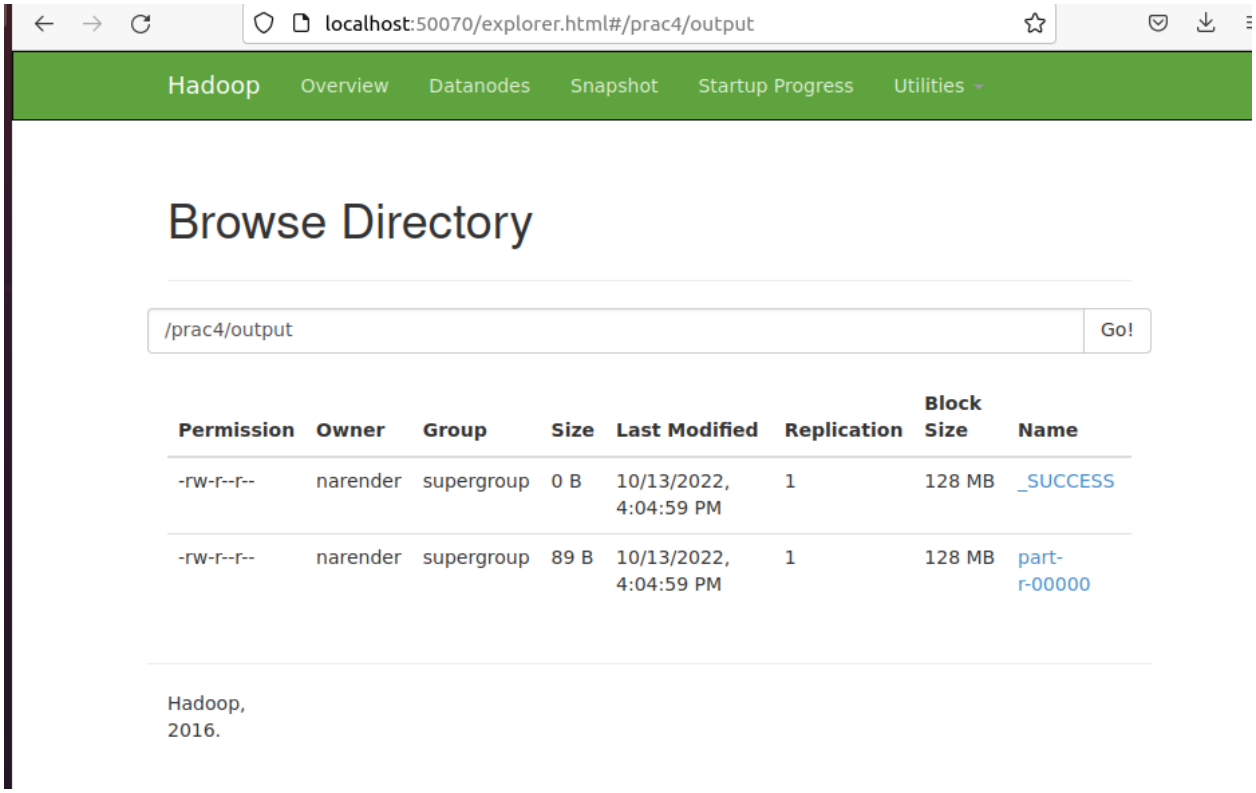
Inside/prac4/input/N folder



Inside /prac4/interim folder



Inside /prac4/output folder



Text Files of cpm-matrix.txt and cpn-matrix.txt

cpm-matrix.txt

```
 1 1,2,-3.0
 2 0,1,5.0
 3 2,3,6.0
 4 0,1,-2.0
 5 1,0,3.0
 6 3,2,4.0
 7 1,2,3.0
 8 2,3,2.0
 9 3,2,3.0
10 1,1,-3.0
11 2,3,-2.0
```

cpn-matrix.txt

```
 1 1,0,2.0
 2 2,2,3.0
 3 3,1,3.0
 4 2,2,1.0
 5 1,0,2.0
 6 3,1,2.0
 7 1,0,1.0
 8 2,3,7.0
 9 6,1,2.0
10 3,2,3.0
11 1,2,1.0
```

CONCLUSION: -

From this practical I have learned to implement Map Reduce Programming Examples Matrix Multiplication. Word Count.

PRACTICAL 3

A)

AIM: - Mongo DB: Installation and Creation of database and Collection

THEORY: -

What is MongoDB?

MongoDB is an open-source document-oriented database that is designed to store a large scale of data and also allows you to work with that data very efficiently. It is categorized under the NoSQL (Not only SQL) database because the storage and retrieval of data in the MongoDB are not in the form of tables.

The MongoDB database is developed and managed by MongoDB.Inc under SSPL(Server Side Public License) and initially released in February 2009. It also provides official driver support for all the popular languages like C, C++, C#, and .Net, Go, Java, Node.js, Perl, PHP, Python, Motor, Ruby, Scala, Swift, Mongoid. So, that you can create an application using any of these languages. Nowadays there are so many companies that used MongoDB like Facebook, Nokia, eBay, Adobe, Google, etc. to store their large amount of data.

Features of MongoDB :

Schema-less Database:

It is the great feature provided by the MongoDB. A Schema-less database means one collection can hold different types of documents in it. Or in other words, in the MongoDB database, a single collection can hold multiple documents and these documents may consist of the different numbers of fields, content, and size. It is not necessary that the one document is similar to another document like in the relational databases. Due to this cool feature, MongoDB provides great flexibility to databases.

Document Oriented:

In MongoDB, all the data stored in the documents instead of tables like in RDBMS. In these documents, the data is stored in fields(key-value pair) instead of rows and columns which make the data much more flexible in comparison to RDBMS. And each document contains its unique object id.

Indexing:
In MongoDB database, every field in the documents is indexed with primary and secondary indices this makes easier and takes less time to get or search data from the pool of the data. If the data is not indexed, then database search each document with the specified query which takes lots of time and not so efficient.

Scalability:
MongoDB provides horizontal scalability with the help of sharding. Sharding means to distribute data on multiple servers, here a large amount of data is partitioned into data chunks using the shard key, and these data chunks are evenly distributed across shards that reside across many physical servers. It will also add new machines to a running database.

Replication:
MongoDB provides high availability and redundancy with the help of replication, it creates multiple copies of the data and sends these copies to a different server so that if one server fails, then the data is retrieved from another server.

Aggregation:
It allows to perform operations on the grouped data and get a single result or computed result. It is similar to the SQL GROUPBY clause. It provides three different aggregations i.e, aggregation pipeline, map-reduce function, and singlepurpose aggregation methods

High Performance:
The performance of MongoDB is very high and data persistence as compared to another database due to its features like scalability, indexing, replication, etc.

INSTALLATION: -

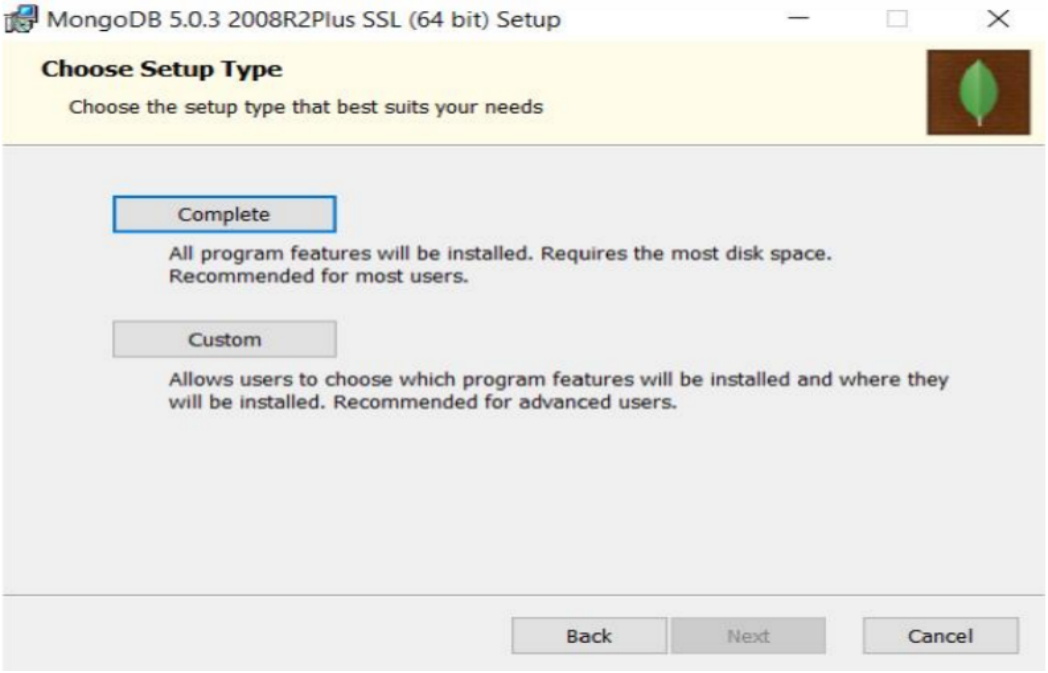
1) Download MongoDB from the MongoDB from the official website

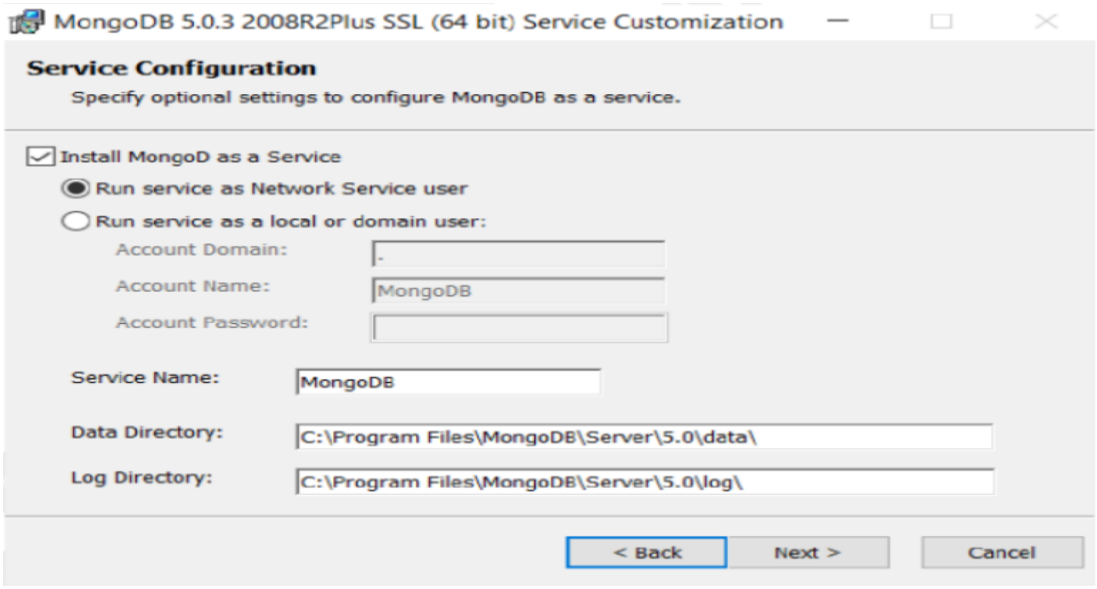
2) After the download run the file foR Installation.

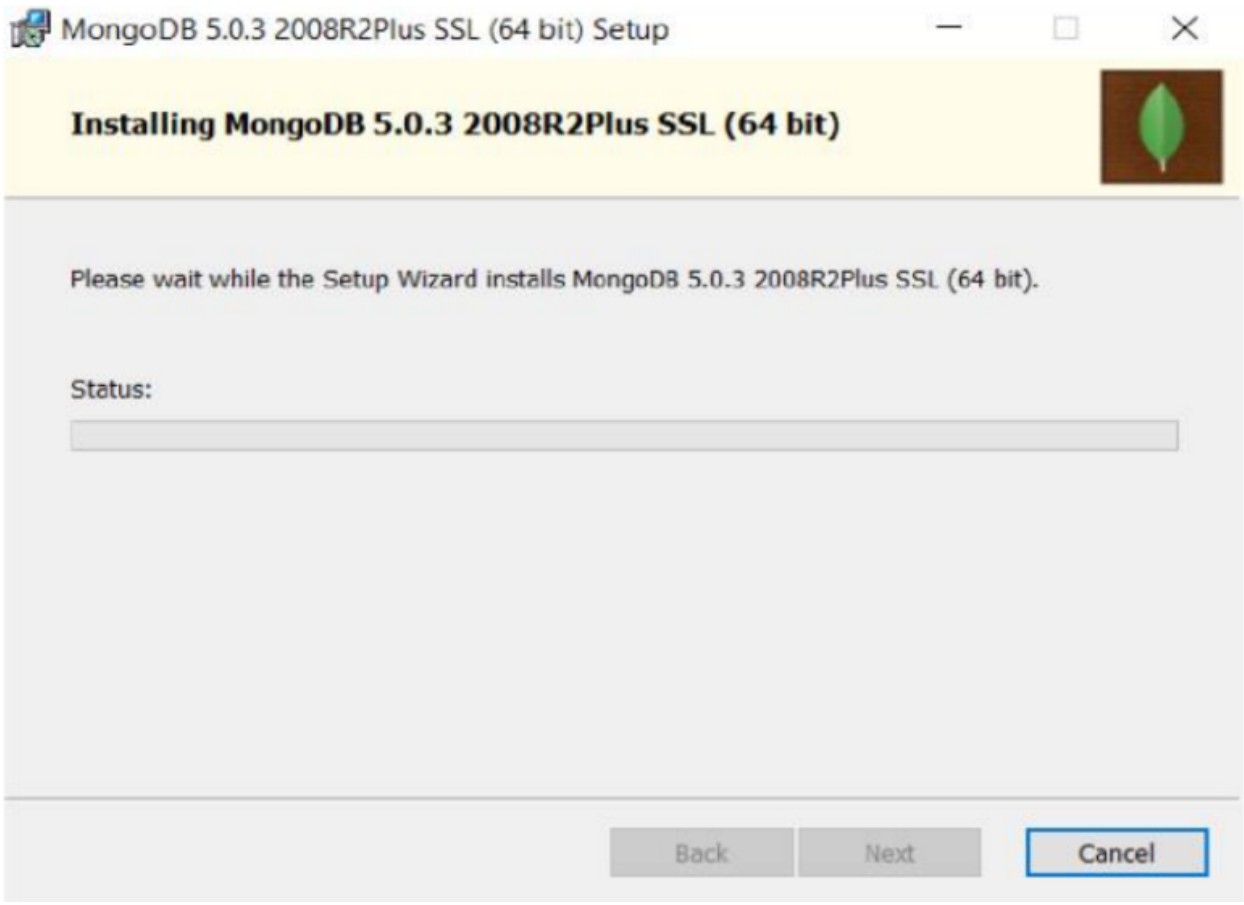
3) Do Next as bydefault option is to be chosen

Below images may provide the help for Installation

PROCESS: -







CONCLUSION: -

From this practical I have learned to install MongoDB.

B)

AIM: - CRUD Document: Insert, Query, Update and Delete Document.

THEORY: -

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

MongoDB documents are modified by connecting to a server, querying the proper documents, and then changing the setting properties before sending the data back to the database to be updated. CRUD is data-oriented, and it's standardized according to HTTP action verbs.

When it comes to the individual CRUD operations:

The Create operation is used to insert new documents in the MongoDB database.

The Read operation is used to query a document in the database.

The Update operation is used to modify existing documents in the database.

The Delete operation is used to remove documents in the database.

A) Create Operations

For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed. Create operations in MongoDB target a single collection, not multiple collections. Insert operations in MongoDB are atomic on a single document level.

MongoDB provides two different create operations that you can use to insert documents into a collection:

db.collection.insertOne()

db.collection.insertMany()

1) insertOne()

As the namesake, insertOne() allows you to insert one document into the collection. For this example, we're going to work with a collection called RecordsDB. We can insert a single entry into our collection by calling the insertOne() method on RecordsDB. We then provide the information we want to insert in the form of key-value pairs, establishing the schema.

2) insertMany()

It's possible to insert multiple items at one time by calling the insertMany() method on the desired collection. In this case, we pass multiple items into our chosen collection (RecordsDB) and separate them by commas. Within the parentheses, we use brackets to indicate that we are passing in a list of multiple entries. This is commonly referred to as a nested method.

B) Read Operations

The read operations allow you to supply special query filters and criteria that let you specify which documents you want. The MongoDB documentation contains more information on the available query filters. Query modifiers may also be used to change how many results are returned.

MongoDB has two methods of reading documents from a collection:

db.collection.find()

db.collection.findOne()

1) find()

In order to get all the documents from a collection, we can simply use the find() method on our chosen collection. Executing just the find() method with no arguments will return all records currently in the collection.

2) findOne()

In order to get one document that satisfies the search criteria, we can simply use the findOne() method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk. If no documents satisfy the search criteria, the function returns null.

C) Update Operations

Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

We should be careful when updating documents, as updates are permanent and can't be rolled back. This applies to delete operations as well.

For MongoDB CRUD, there are three different methods of updating documents:

db.collection.updateOne()

db.collection.updateMany()

db.collection.replaceOne()

1) updateOne()

We can update a currently existing record and change a single document with an update operation. To do this, we use the updateOne() method on a chosen collection, which here is "RecordsDB." To update a document, we provide the method with two arguments: an update filter and an update action.

The update filter defines which items we want to update, and the update action defines how to update those items. We first pass in the update filter. Then, we use the "$set" key and provide the fields we want to update as a value. This method will update the first record that matches the provided filter.

2) updateMany()

updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items. This update operation uses the same syntax for updating a single document.

3) replaceOne()

The replaceOne() method is used to replace a single document in the specified collection. replaceOne() replaces the entire document, meaning fields in the old document not contained in the new will be lost.

D) Delete Operations

Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document. You can provide delete operations with filters and criteria in order to specify which documents you would like to delete from a collection. The filter options rely on the same syntax that read operations utilize.

MongoDB has two different methods of deleting records from a collection:

db.collection.deleteOne()

db.collection.deleteMany()

1) deleteOne()

deleteOne() is used to remove a document from a specified collection on the MongoDB server. A filter criteria is used to specify the item to delete. It deletes the first record that matches the provided filter.

2) deleteMany()

deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation. A list is passed into the method and the individual items are defined with filter criteria as in deleteOne().

STEPS:

1) Goto the terminal type mongo command.

2) The mongo commands shall be executed when the terminal of the mongo is open and this will be done only after step 1.

3) Now goto libreoffice and choose a spreadsheet to enter data as per columns in the table.

4) Open another terminal and access the file without going to the mongo terminal that means directly on terminal.

5) The file created on the spreadsheet should be saved using .csv.

6) All files should be accessed using terminalt and not by mongo terminal.

QUERIES & OUTPUT: -

1) Enter mongodb shell to check if it installed properly:

```
vaish@vaishVirtualBox:~/Desktop$ mongo
MongoDB shell version v3.6.8
connecting to: mongodb://127.0.0.1:27017
Implicit session: session { "id" : UUID("69a3cba3-4c05-457b-98c7-522e58da5016") }
MongoDB server version: 3.6.8
Server has startup warnings:
2022-10-06T06:17:30.711+0530 I STORAGE  [initandlisten]
2022-10-06T06:17:30.711+0530 I STORAGE  [initandlisten] ** WARNING: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine
2022-10-06T06:17:30.711+0530 I STORAGE  [initandlisten] **          See http://dochub.mongodb.org/core/prodnotes-filesystem
2022-10-06T06:17:32.651+0530 I CONTROL  [initandlisten]
2022-10-06T06:17:32.651+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2022-10-06T06:17:32.651+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unrestricted.
2022-10-06T06:17:32.651+0530 I CONTROL  [initandlisten]
>
```

2) Create a database named records and switch to records

Query: -use records

```
> use records
switched to db records
>
```

3) We have collection namely result_mca. Let us populate them.
Run a query to insert into the table
db.result_mca.insertMany([{id:1,name:"Christy",result:"pass",marks:85},{id:2,name:"Chinmay",result:"pass",marks:90},{id:3,name:"Yash",result:"pass",marks:80},{id:4,name:"Devendra",result:"pass",marks:90},{id:5,name:"Dhanraj",result:"pass",marks:80},{id:6,name:"Dhanshri",result:"pass",marks:90}])

```
> db.result_mca.insertMany([{id:1,name:"Christy",result:"pass",marks:85},{id:2,name:"Chinmay",result:"pass",
marks:90},{id:3,name:"Yash",result:"pass",marks:80},{id:4,name:"Devendra",result:"pass",marks:90},{id:5,name
:"Dhanraj",result:"pass",marks:80},{id:6,name:"Dhanshri",result:"pass",marks:90}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("633e28ba555c2c8335b8e54d"),
                ObjectId("633e28ba555c2c8335b8e54e"),
                ObjectId("633e28ba555c2c8335b8e54f"),
                ObjectId("633e28ba555c2c8335b8e550"),
                ObjectId("633e28ba555c2c8335b8e551"),
                ObjectId("633e28ba555c2c8335b8e552")
        ]
}
>
```

Then we run a query to import result_mca.csv in result_mca collection.
Command to be executed in new terminal
mongoimport -d records -c result_mca --type csv --file /home/vaish/Documents/Doc.csv --headerline

```
vaish@vaishVirtualBox:~/Desktop$ mongoimport -d records -c result_mca --type csv --file /home/vaish/Documents/Doc.csv --headerline
2022-10-06T06:40:23.917+0530    connected to: localhost
2022-10-06T06:40:23.938+0530    imported 9 documents
```

The data created in the spreadsheet

	A	B	C	D
1	id	name	result	marks
2	7	Lalita	pass	80
3	8	Aditi	pass	85
4	9	Ashvita	pass	80
5	10	Jason	pass	80
6	11	Ronal	pass	90
7	12	Shardul	pass	80
8	13	Aastha	pass	80
9	14	Imran	pass	80
10	15	Aashish	pass	80

Check if data is imported in result_mca collection:

Query: - db.result_mca.find()

```
> db.result_mca.find()
{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e552"), "id" : 6, "name" : "Dhanshri", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196d"), "id" : 7, "name" : "Lalita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196e"), "id" : 8, "name" : "Aditi", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196f"), "id" : 9, "name" : "Ashvita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41970"), "id" : 10, "name" : "Jason", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41971"), "id" : 11, "name" : "Ronal", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41972"), "id" : 12, "name" : "Shardul", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41973"), "id" : 13, "name" : "Aastha", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41974"), "id" : 14, "name" : "Imran", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41975"), "id" : 15, "name" : "Aashish", "result" : "pass", "marks" : 80 }
>
```

Similarly create another collection student_mca and import data from csv file

Query: -

db.students_mca.insertMany([{id:1,div:"B"},
... ... {id:2,div:"B"},
... ... {id:3,div:"B"},
... ... {id:4,div:"B"},
... ... {id:5,div:"B"}])

```
> db.students_mca.insertMany([{id:1,div:"B"},
...  ... {id:2,div:"B"},
...  ... {id:3,div:"B"},
...  ... {id:4,div:"B"},
...  ... {id:5,div:"B"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("633e2bc9555c2c8335b8e553"),
                ObjectId("633e2bc9555c2c8335b8e554"),
                ObjectId("633e2bc9555c2c8335b8e555"),
                ObjectId("633e2bc9555c2c8335b8e556"),
                ObjectId("633e2bc9555c2c8335b8e557")
        ]
}
>
```

Importing data from csv file to be executed in simple terminal

mongoimport -d records -c students_mca --type csv --file /home/vaish/Documents/Students.csv --headerline

```
vaish@vaishVirtualBox:~/Desktop$ mongoimport -d records -c students_mca --type csv --file /home/vaish/Documents/Students.csv --headerline
2022-10-06T06:48:47.148+0530    connected to: localhost
2022-10-06T06:48:47.150+0530    imported 10 documents
```

Query: -

db.students_mca.find()

```
> db.students_mca.find()
{ "_id" : ObjectId("633e2bc9555c2c8335b8e553"), "id" : 1, "div" : "B" }
{ "_id" : ObjectId("633e2bc9555c2c8335b8e554"), "id" : 2, "div" : "B" }
{ "_id" : ObjectId("633e2bc9555c2c8335b8e555"), "id" : 3, "div" : "B" }
{ "_id" : ObjectId("633e2bc9555c2c8335b8e556"), "id" : 4, "div" : "B" }
{ "_id" : ObjectId("633e2bc9555c2c8335b8e557"), "id" : 5, "div" : "B" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c41985"), "id" : 6, "div" : "A" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c41986"), "id" : 7, "div" : "A" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c41987"), "id" : 8, "div" : "A" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c41988"), "id" : 9, "div" : "A" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c41989"), "id" : 10, "div" : "A" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c4198a"), "id" : 11, "div" : "B" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c4198b"), "id" : 12, "div" : "B" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c4198c"), "id" : 13, "div" : "B" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c4198d"), "id" : 14, "div" : "B" }
{ "_id" : ObjectId("633e2cf7daac3c9d65c4198e"), "id" : 15, "div" : "B" }
>
```

4) QUERIES: -

A) Find data where result is pass in the given database

Query: -

db.result_mca.find({result:"pass"}).limit(5)

```
> db.result_mca.find( {result:"pass"}).limit(5)
{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj", "result" : "pass", "marks" : 80 }
>
```

B) Display name and id whose marks are greater than 75 and result is pass.

Query-

db.result_mca.find({ marks:{ $gt :75} ,result :"pass"},{id:1,name:1}).limit(5)

```
> db.result_mca.find({ marks:{ $gt :75} ,result :"pass"},{id:1,name:1}).limit(5)

{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy" }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay" }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash" }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra" }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj" }
>
```

C) Update documents having name as john to jack.

Query: -

db.result_mca.updateMany({name:"Jason"},{ $set :{name:"Jason L"}})

```
> db.result_mca.updateMany({name:"Jason"},{ $set :{name:"Jason L"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

```
> db.result_mca.find()
{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e552"), "id" : 6, "name" : "Dhanshri", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196d"), "id" : 7, "name" : "Lalita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196e"), "id" : 8, "name" : "Aditi", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196f"), "id" : 9, "name" : "Ashvita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41970"), "id" : 10, "name" : "Jason L", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41971"), "id" : 11, "name" : "Ronal", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41972"), "id" : 12, "name" : "Shardul", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41973"), "id" : 13, "name" : "Aastha", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41974"), "id" : 14, "name" : "Imran", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41975"), "id" : 15, "name" : "Aashish", "result" : "pass", "marks" : 80 }
>
```

D) Delete document whose id is 12

Query: -

db.result_mca.deleteOne({id : 12})

Here first checking is done before deleting

```
> db.result_mca.find()
{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e552"), "id" : 6, "name" : "Dhanshri", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196d"), "id" : 7, "name" : "Lalita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196e"), "id" : 8, "name" : "Aditi", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196f"), "id" : 9, "name" : "Ashvita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41970"), "id" : 10, "name" : "Jason L", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41971"), "id" : 11, "name" : "Ronal", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41972"), "id" : 12, "name" : "Shardul", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41973"), "id" : 13, "name" : "Aastha", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41974"), "id" : 14, "name" : "Imran", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41975"), "id" : 15, "name" : "Aashish", "result" : "pass", "marks" : 80 }
>
```

```
> db.result_mca.deleteOne({id : 12})
{ "acknowledged" : true, "deletedCount" : 1 }
>
> db.result_mca.find()
{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e28ba555c2c8335b8e552"), "id" : 6, "name" : "Dhanshri", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196d"), "id" : 7, "name" : "Lalita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196e"), "id" : 8, "name" : "Aditi", "result" : "pass", "marks" : 85 }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196f"), "id" : 9, "name" : "Ashvita", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41970"), "id" : 10, "name" : "Jason L", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41971"), "id" : 11, "name" : "Ronal", "result" : "pass", "marks" : 90 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41973"), "id" : 13, "name" : "Aastha", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41974"), "id" : 14, "name" : "Imran", "result" : "pass", "marks" : 80 }
{ "_id" : ObjectId("633e2affdaac3c9d65c41975"), "id" : 15, "name" : "Aashish", "result" : "pass", "marks" : 80 }
>
```

E) Here we have performed join and aggregate function using db.collection.aggregate and $lookup here

So here aggregating collections results_mca into students_mca by taking localfield as id of results_mca and foreignfield as id from students_mca

Query: -

```
db.result_mca.aggregate([{
... ... $lookup:{
... ... from:"students_mca",
... ... localField:"id",
... ... foreignField:"id",
... ... as:"marks"
... ... }
... ... }])
```

```
> db.result_mca.aggregate([{
... ... $lookup:{
... ... from:"students_mca",
... ... localField:"id",
... ... foreignField:"id",
... ... as:"marks"
... ... }
... ... }])
{ "_id" : ObjectId("633e28ba555c2c8335b8e54d"), "id" : 1, "name" : "Christy", "result" : "pass", "marks"
: [ { "_id" : ObjectId("633e2bc9555c2c8335b8e553"), "id" : 1, "div" : "B" } ] }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54e"), "id" : 2, "name" : "Chinmay", "result" : "pass", "marks"
: [ { "_id" : ObjectId("633e2bc9555c2c8335b8e554"), "id" : 2, "div" : "B" } ] }
{ "_id" : ObjectId("633e28ba555c2c8335b8e54f"), "id" : 3, "name" : "Yash", "result" : "pass", "marks" : [
 { "_id" : ObjectId("633e2bc9555c2c8335b8e555"), "id" : 3, "div" : "B" } ] }
{ "_id" : ObjectId("633e28ba555c2c8335b8e550"), "id" : 4, "name" : "Devendra", "result" : "pass", "marks"
 : [ { "_id" : ObjectId("633e2bc9555c2c8335b8e556"), "id" : 4, "div" : "B" } ] }
{ "_id" : ObjectId("633e28ba555c2c8335b8e551"), "id" : 5, "name" : "Dhanraj", "result" : "pass", "marks"
: [ { "_id" : ObjectId("633e2bc9555c2c8335b8e557"), "id" : 5, "div" : "B" } ] }
{ "_id" : ObjectId("633e28ba555c2c8335b8e552"), "id" : 6, "name" : "Dhanshri", "result" : "pass", "marks"
 : [ { "_id" : ObjectId("633e2cf7daac3c9d65c41985"), "id" : 6, "div" : "A" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196d"), "id" : 7, "name" : "Lalita", "result" : "pass", "marks" :
 [ { "_id" : ObjectId("633e2cf7daac3c9d65c41986"), "id" : 7, "div" : "A" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196e"), "id" : 8, "name" : "Aditi", "result" : "pass", "marks" :
[ { "_id" : ObjectId("633e2cf7daac3c9d65c41987"), "id" : 8, "div" : "A" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c4196f"), "id" : 9, "name" : "Ashvita", "result" : "pass", "marks"
: [ { "_id" : ObjectId("633e2cf7daac3c9d65c41988"), "id" : 9, "div" : "A" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c41970"), "id" : 10, "name" : "Jason L", "result" : "pass", "marks"
 : [ { "_id" : ObjectId("633e2cf7daac3c9d65c41989"), "id" : 10, "div" : "A" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c41971"), "id" : 11, "name" : "Ronal", "result" : "pass", "marks" :
 [ { "_id" : ObjectId("633e2cf7daac3c9d65c4198a"), "id" : 11, "div" : "B" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c41973"), "id" : 13, "name" : "Aastha", "result" : "pass", "marks"
: [ { "_id" : ObjectId("633e2cf7daac3c9d65c4198c"), "id" : 13, "div" : "B" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c41974"), "id" : 14, "name" : "Imran", "result" : "pass", "marks" :
 [ { "_id" : ObjectId("633e2cf7daac3c9d65c4198d"), "id" : 14, "div" : "B" } ] }
{ "_id" : ObjectId("633e2affdaac3c9d65c41975"), "id" : 15, "name" : "Aashish", "result" : "pass", "marks"
 : [ { "_id" : ObjectId("633e2cf7daac3c9d65c4198e"), "id" : 15, "div" : "B" } ] }
> _
```

Extract the salesdb database in json format using following command in terminal

Mongo --db records --collection result_mca --out /home/vaish/Desktop/results.json

```
vaish@vaishVirtualBox:~/Desktop$ mongoexport --db records --collection result_mca --out /home/vaish/Desktop/results.json
2022-10-06T06:57:14.790+0530    connected to: localhost
2022-10-06T06:57:14.796+0530    exported 14 records
```


```
{"_id":{"$oid":"633e28ba555c2c8335b8e54d"},"id":1.0,"name":"Christy","result":"pass","marks":85.0}
{"_id":{"$oid":"633e28ba555c2c8335b8e54e"},"id":2.0,"name":"Chinmay","result":"pass","marks":90.0}
{"_id":{"$oid":"633e28ba555c2c8335b8e54f"},"id":3.0,"name":"Yash","result":"pass","marks":80.0}
{"_id":{"$oid":"633e28ba555c2c8335b8e550"},"id":4.0,"name":"Devendra","result":"pass","marks":90.0}
{"_id":{"$oid":"633e28ba555c2c8335b8e551"},"id":5.0,"name":"Dhanraj","result":"pass","marks":80.0}
{"_id":{"$oid":"633e28ba555c2c8335b8e552"},"id":6.0,"name":"Dhanshri","result":"pass","marks":90.0}
{"_id":{"$oid":"633e2affdaac3c9d65c4196d"},"id":7,"name":"Lalita","result":"pass","marks":80}
{"_id":{"$oid":"633e2affdaac3c9d65c4196e"},"id":8,"name":"Aditi","result":"pass","marks":85}
{"_id":{"$oid":"633e2affdaac3c9d65c4196f"},"id":9,"name":"Ashvita","result":"pass","marks":80}
{"_id":{"$oid":"633e2affdaac3c9d65c41970"},"id":10,"name":"Jason L","result":"pass","marks":80}
{"_id":{"$oid":"633e2affdaac3c9d65c41971"},"id":11,"name":"Ronal","result":"pass","marks":90}
{"_id":{"$oid":"633e2affdaac3c9d65c41973"},"id":13,"name":"Aastha","result":"pass","marks":80}
{"_id":{"$oid":"633e2affdaac3c9d65c41974"},"id":14,"name":"Imran","result":"pass","marks":80}
{"_id":{"$oid":"633e2affdaac3c9d65c41975"},"id":15,"name":"Aashish","result":"pass","marks":80}
```


CONCLUSION: -

From this practical I have learned to implement demonstration of a CRUD operations on document-oriented NoSQL database - MongoDB.

PRACTICAL 4

AIM: - Hive: Introduction Creation of Database and Table, Hive Partition, Hive Built in Function and Operators, Hive View and Index.

THEORY: -

What is Hive?

Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data, and makes querying and analyzing easy.

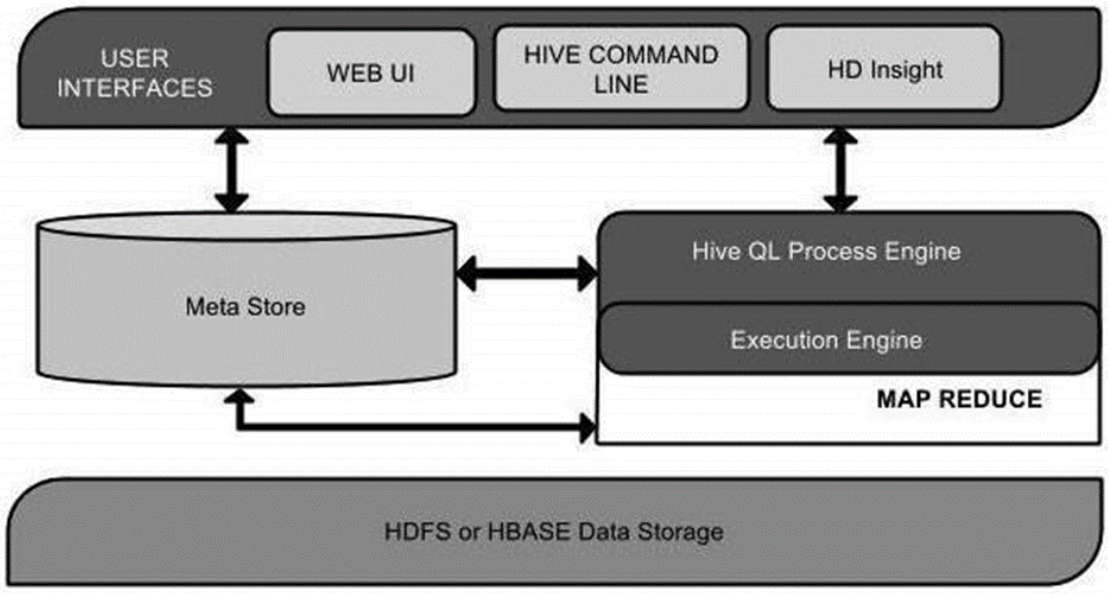
Initially Hive was developed by Facebook, later the Apache Software Foundation took it up and developed it further as an open source under the name Apache Hive. It is used by different companies. For example, Amazon uses it in Amazon Elastic MapReduce.

Features of Hive

- It stores schema in a database and processes data into HDFS.
- It is designed for OLAP.
- It provides SQL type language for querying called HiveQL or HQL.
- It is familiar, fast, scalable, and extensible.

Architecture of Hive

The following component diagram depicts the architecture of Hive:

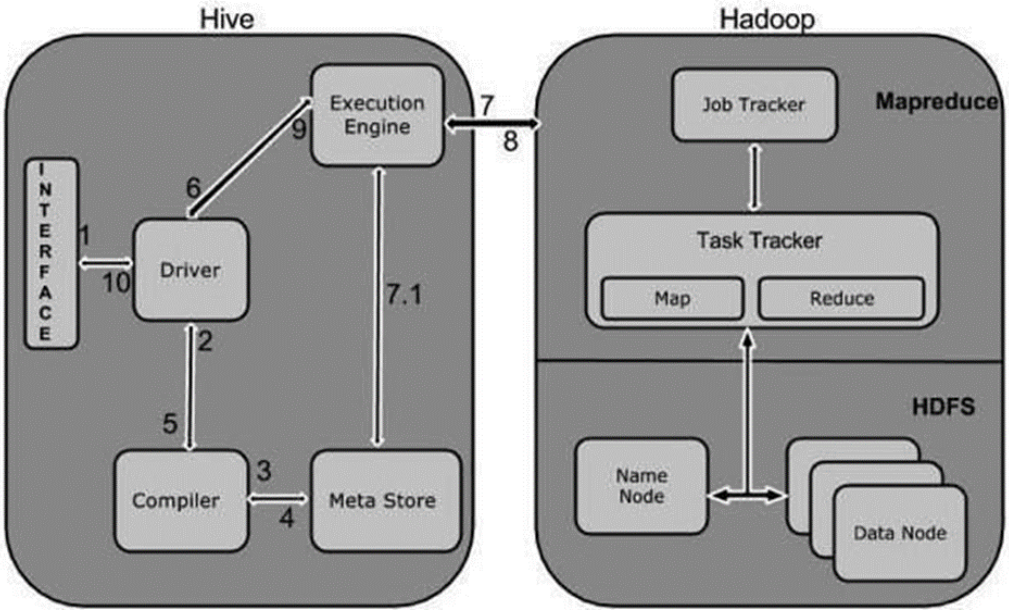


This component diagram contains different units. The following table describes each unit:

Unit Name	**Operation**
User Interface	Hive is a data warehouse infrastructure software that can create interaction between user and HDFS. The user interfaces that Hive supports are Hive Web UI, Hive command line, and Hive HD Insight (In Windows server).
Meta Store	Hive chooses respective database servers to store the schema or Metadata of tables, databases, columns in a table, their data types, and HDFS mapping.
HiveQL Process Engine	HiveQL is similar to SQL for querying on schema info on the Metastore. It is one of the replacements of traditional approach for MapReduce program. Instead of writing MapReduce program in Java, we can write a query for MapReduce job and process it.
Execution Engine	The conjunction part of HiveQL process Engine and MapReduce is Hive Execution Engine. Execution engine processes the query and generates results as same as MapReduce results. It uses the flavor of MapReduce.
HDFS or HBASE	Hadoop distributed file system or HBASE are the data storage techniques to store data into file system.

Working of Hive

The following diagram depicts the workflow between Hive and Hadoop.



The following table defines how Hive interacts with Hadoop framework:

Step No.	Operation
1	**Execute Query -** The Hive interface such as Command Line or Web UI sends query to Driver (any database driver such as JDBC, ODBC, etc.) to execute.
2	**Get Plan -** The driver takes the help of query compiler that parses the query to check the syntax and query plan or the requirement of query.
3	**Get Metadata -** The compiler sends metadata request to Metastore (any database).
4	**Send Metadata -** Metastore sends metadata as a response to the compiler.

5	**Send Plan -** The compiler checks the requirement and resends the plan to the driver. Up to here, the parsing and compiling of a query is complete.
6	**Execute Plan -** The driver sends the execute plan to the execution engine.
7	**Execute Job -** Internally, the process of execution job is a MapReduce job. The execution engine sends the job to JobTracker, which is in Name node and it assigns this job to TaskTracker, which is in Data node. Here, the query executes MapReduce job.
7.1	**Metadata Ops -** Meanwhile in execution, the execution engine can execute metadata operations with Metastore.
8	**Fetch Result -** The execution engine receives the results from Data nodes.
9	**Send Results -** The execution engine sends those resultant values to the driver.
10	**Send Results –** The driver sends the results to Hive Interfaces.

Creation of Database and Table:
Database Creation in hive -

Hive is a database technology that can define databases and tables to analyze structured data. The theme for structured data analysis is to store the data in a tabular manner, and pass queries to analyze it. This chapter explains how to create a Hive database. Hive contains a default database named **default**.

Create Database Statement – Create Database is a statement used to create a database in Hive. A database in Hive is a namespace or a collection of tables. The syntax for this statement is as follows:

CREATE DATABASE|SCHEMA [IF NOT EXISTS] <database name>

Here, IF NOT EXISTS is an optional clause, which notifies the user that a database with the same name already exists. We can use SCHEMA in place of DATABASE in this command. The following query is executed to create a database named **userdb**: hive> CREATE DATABASE [IF NOT EXISTS] userdb;

or

hive> CREATE SCHEMA userdb;

The following query is used to verify a databases list:

hive> SHOW
DATABASES; default
userdb

Table Creation in hive -

Create Table is a statement used to create a table in Hive. The syntax and example are as follows:

Syntax:

CREATE [TEMPORARY] [EXTERNAL] TABLE [IF NOT EXISTS] [db_name.]
table_name
[(col_name data_type [COMMENT col_comment], ...)]
[COMMENT table_comment]
[ROW FORMAT row_format]
[STORED AS file_format]

Hive Partition:

Hive organizes tables into partitions. It is a way of dividing a table into related parts based on the values of partitioned columns such as date, city, and department. Using partition, it is easy to query a portion of the data.

Tables or partitions are sub-divided into **buckets,** to provide extra structure to the data that may be used for more efficient querying. Bucketing works based on the value of hash function of some column of a table.

For example, a table named **Tab1** contains employee data such as id, name, dept, and yoj (i.e., year of joining). Suppose you need to retrieve the details of all employees who joined in 2012. A query searches the whole table for the required information. However, if you partition the employee data with the year and store it in a separate file, it reduces the query processing time.

Adding a Partition:
We can add partitions to a table by altering the table. Let us assume we have a table called **employee** with fields such as Id, Name, Salary, Designation, Dept, and yoj.

Syntax:

ALTER TABLE table_name ADD [IF NOT EXISTS] PARTITION partition_spec [LOCATION 'location1'] partition_spec [LOCATION 'location2'] ...; partition_spec:

: (p_column = p_col_value, p_column = p_col_value, ...)

Renaming a Partition:
The syntax of this command is as follows.

ALTER TABLE table_name PARTITION partition_spec RENAME TO PARTITION partition_spec;

Dropping a Partition:
The following syntax is used to drop a partition:

ALTER TABLE table_name DROP [IF EXISTS] PARTITION partition_spec, PARTITION partition_spec,...;

Hive Builtin Function and Operators:
Builtin Functions -

Basically, to perform several operations there are some functions available. Similarly, in Hive also there are some built-in functions available. Such as Hive Collection Functions, Hive Date Functions, Hive Mathematical Functions, Hive Conditional Functions and Hive String Functions.

Functions in Hive are categorized as below.

- **Mathematical Functions:** These functions are mainly used to perform mathematical calculations.
- **Date Functions:** These functions are used to perform operations on date data types like adding the number of days to the date etc.

- **String Functions:** These functions are used to perform operations on strings like finding the length of a string etc.
- **Conditional Functions:** These functions are used to test conditions and returns a value based on whether the test condition is true or false.
- **Collection Functions:** These functions are used to find the size of the complex types like array and map. The only collection function is SIZE. The SIZE function is used to find the number of elements in an array and map.

Builtin Operators -

Hive provides Built-in operators for Data operations to be implemented on the tables present inside Hive warehouse. These operators are used for mathematical operations on operands, and it will return specific value as per the logic applied.

Below are the main types of Built-in Operators in HiveQL:

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Operators on Complex types
- Complex type Constructors

Hive View and Index:

Hive View –

The main objective of creating a hive view is to simplify the complexities of a larger table into a more Flat structure. For example, if you have a table that has 100 columns, but you are only interested in 10 columns, you could create a View with those 10 columns.

Features:

- Hive Views are similar to tables or “copy” of the table.
- Hive Views are generated based on the user requirements.
- It is similar to SQL view.
- View Support All DML operations.
- Subsequent changes in the below table will not be reflected in the View, and if the table is dropped, the view will fail.

Create View:
In this example, we are creating a view called "emp_25000" where it will display all the row values with a salary field greater than 25000.

```
hive> create view emp_25000 AS
>select * from employee
>where salary>25000;
```

Display the View:

```
hive>select * from emp_25000;
```

Dropping View:
This query drops a view named called as "emp_25000"

```
hive> drop view emp_25000;
```

Hive Index –

- Hive Index is nothing but a pointer on a particular column of a table.
- The user has to manually define the index.
- Indexes maintain the reference of the records. So that it is easy to search for a record with minimum overhead.
- Indexes also speed up the searching of data.

Need of indexing in Hive:

Apache Hive is a data warehousing tool built on the top of Hadoop, it provides the SQL interface to perform queries on large data sets. Since Hive deals with Big Data, the size of files is very large and can expand up to Terabytes and Petabytes. Now if we want to perform any operation or a query on this huge amount of data it will take more time.

In a Hive table, there are many numbers of rows and columns. If we want to perform queries only on some columns without indexing, it will take more time because queries will be executed on all the columns present in the table.

Alter an index –

```
hive>alter index indx_salary on employee REBUILD;
```

Display an index -
Using below command we can check the available indexes for the table "employee"

hive> show formatted index on employee;

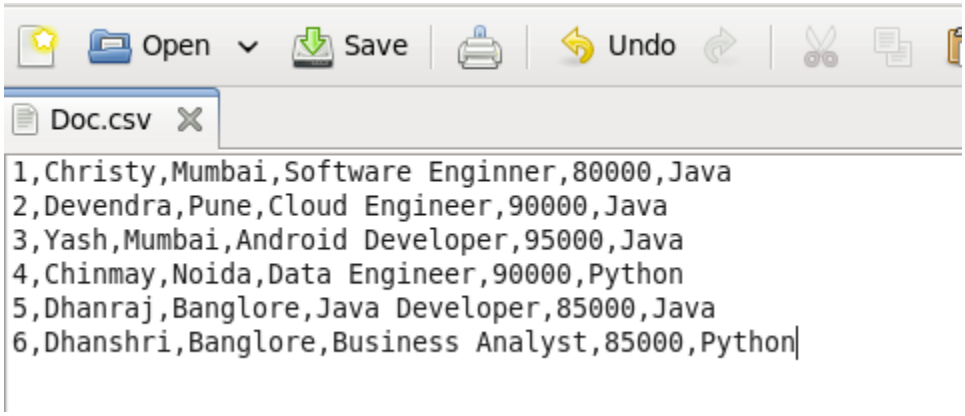
Dropping an Index -

The following query used to drops an index named “index_salary”

hive> drop index index_salary ON employee;

COMMANDS AND OUTPUTS: -

Doc.csv



1) Creating a directory /Newinput

Command: - hdfs dfs -mkdir /Newinput

```
[cloudera@quickstart Desktop]$ hdfs dfs -mkdir /Newinput
```

2) Providing the Doc.csv file to Newinut folder

Command: - hdfs dfs -put /home/cloudera/Desktop/Doc.csv Newinput

```
[cloudera@quickstart Desktop]$ hdfs dfs -put /home/cloudera/Desktop/Doc.csv Newinput
```

3) We will entering into hive terminal using below command

Command: -sudo hive

```
[cloudera@quickstart Desktop]$ sudo hive
2022-10-14 02:57:10,636 WARN  [main] mapreduce.TableMapReduceUtil: The hbase-prefix-tree module jar containing
PrefixTreeCodec is not present.  Continuing without it.

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j.properties
WARNING: Hive CLI is deprecated and migration to Beeline is recommended.
```

4) Here we will be creating a database called New;

Command: - create database New;

```
hive> create database New;
OK
Time taken: 0.961 seconds
```

5) We will list down the databases in the system

Command: - show databases;

```
hive> show databases;
OK
default
new
records
Time taken: 0.178 seconds, Fetched: 3 row(s)
```

6) We will enter into the database using below command

Command: - use new;

```
hive> use new;
OK
Time taken: 0.1 seconds
```

7) So now we are creating a table inside the database called Employees

Command: - create table Employees1(
eid int,
ename string,
ecity string,
department string,
salary int,
domain string)
row format delimited
fields terminated by ',';

```
hive> create table Employees1(
    > eid int,
    > ename string,
    > ecity string,
    > department string,
    > salary int,
    > domain string)
    > row format delimited
    > fields terminated by ',';
OK
Time taken: 0.077 seconds
```

8) We will see the datatype of each column using below command

Command: - describe Employees1;

```
hive> describe Employees1;
OK
eid                     int
ename                   string
ecity                   string
department              string
salary                  int
domain                  string
Time taken: 0.075 seconds, Fetched: 6 row(s)
```

9) This will give detailed information about the table

Command: - describe formatted Employees1;

```
OK
# col_name              data_type               comment

eid                     int
ename                   string
ecity                   string
department              string
salary                  int
domain                  string

# Detailed Table Information
Database:               new
Owner:                  root
CreateTime:             Fri Oct 14 03:02:02 PDT 2022
LastAccessTime:         UNKNOWN
Protect Mode:           None
Retention:              0
Location:               hdfs://quickstart.cloudera:8020/user/hive/warehouse/new.db/
employees1
Table Type:             MANAGED_TABLE
Table Parameters:
        transient_lastDdlTime   1665741722
```

10) Now we will be creating a Doc.csv file and load it to database using below command

Command: - LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Doc.csv' INTO TABLE Employees1;

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Doc.csv' INTO TABLE Employees1;
Loading data to table new.employees1
Table new.employees1 stats: [numFiles=1, totalSize=269]
OK
Time taken: 0.993 seconds
```

11) The below command will list the data inside the table

Command: - select * from Employees1;

```
hive> select * from Employees1;
OK
1       Christy Mumbai  Software Engineer       80000   Java
2       Devendra        Pune    Cloud Engineer  90000   Java
3       Yash    Mumbai  Android Developer       95000   Java
4       Chinmay Noida   Data Engineer   90000   Python
5       Dhanraj Banglore        Java Developer  85000   Java
6       Dhanshri        Banglore        Business Analyst        85000   Python
Time taken: 0.2 seconds, Fetched: 6 row(s)
```

12) The below command will do the sum of salary column

Command: - select sum(salary) from Employees1;

```
hive> select sum(salary) from Employees1;
Query ID = root_20221014030909_aeb92347-0311-468e-b91c-f34bf0ffa51d
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.88 sec   HDFS Read: 7125 HDFS Write: 7 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 880 msec
OK
525000
Time taken: 19.085 seconds, Fetched: 1 row(s)
```

13) The below command will give the max salary from salary column in table

Command: - select max(salary) from Employees1;

```
hive> select max(salary) from Employees1;
Query ID = root_20221014031010_2376f352-f0a6-4425-98b9-266e29b1cda1
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.31 sec   HDFS Read: 7126 HDFS Write: 6 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 310 msec
OK
95000
Time taken: 17.206 seconds, Fetched: 1 row(s)
```

14) The below command will show the count of values in table

Command: - select count(*) from Employees1;

```
hive> select count(*) from Employees1;
Query ID = root_20221014031111_16a7f88f-5015-4dce-96a5-fac7e4e9834c
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.29 sec   HDFS Read: 7014 HDFS Write: 2 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 290 msec
OK
6
Time taken: 15.886 seconds, Fetched: 1 row(s)
```

15) The below command will list the data inside the table

Command: - select * from Employees1;

```
hive> select * from Employees1;
OK
1       Christy Mumbai  Software Engineer       80000   Java
2       Devendra        Pune    Cloud Engineer  90000   Java
3       Yash    Mumbai  Android Developer       95000   Java
4       Chinmay Noida   Data Engineer   90000   Python
5       Dhanraj Banglore        Java Developer  85000   Java
6       Dhanshri        Banglore        Business Analyst        85000   Python
Time taken: 0.026 seconds, Fetched: 6 row(s)
```

16) We will display data using order by clause eid in a descending order

Command: - Select * from Employees1 order by eid desc;

```
hive> Select * from Employees1 order by eid desc;
Query ID = root_20221014031515_ebac1b92-65c2-4453-ae3b-3a2a7141aaa9
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
MapReduce Total cumulative CPU time: 1 seconds 270 msec
Ended Job = job_1665736730690_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.27 sec   HDFS Read: 7248 HDFS Write: 269 SUCCESS
Total MapReduce CPU Time Spent: 1 seconds 270 msec
OK
6       Dhanshri        Banglore        Business Analyst        85000   Python
5       Dhanraj Banglore        Java Developer  85000   Java
4       Chinmay Noida   Data Engineer   90000   Python
3       Yash    Mumbai  Android Developer       95000   Java
2       Devendra        Pune    Cloud Engineer  90000   Java
1       Christy Mumbai  Software Engineer       80000   Java
Time taken: 16.823 seconds, Fetched: 6 row(s)
```

17) We will sort the data by ename in an descending order

Command: -Select * from Employees1 sort by ename desc;

```
hive> Select * from Employees1 sort by ename desc;
Query ID = root_20221014031717_d25977c8-6bdb-4325-bfce-a01038d8e4ab
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
```

```
MapReduce Total cumulative CPU time: 1 seconds 270 msec
Ended Job = job_1665736730690_0008
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 1.27 sec   HDFS Read: 7248 HDFS Write: 269 SUCCE
Total MapReduce CPU Time Spent: 1 seconds 270 msec
OK
3       Yash    Mumbai  Android Developer       95000   Java
6       Dhanshri        Banglore        Business Analyst        85000   Python
5       Dhanraj Banglore        Java Developer  85000   Java
2       Devendra        Pune    Cloud Engineer  90000   Java
1       Christy Mumbai  Software Engineer       80000   Java
4       Chinmay Noida   Data Engineer   90000   Python
Time taken: 15.968 seconds, Fetched: 6 row(s)
```

HCATALOG

The below commands need to be executed in terminal of Ubuntu or Cloudera and not the Hive terminal

1) We are creating the table vendors

Command: - hcat -e "CREATE TABLE vendors \
(id INT, company STRING, email STRING) \
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' \
LOCATION '/dualcore/vendors' "

```
[cloudera@quickstart Desktop]$ hcat -e "CREATE TABLE vendors \
> (id INT, company STRING, email STRING) \
> ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' \
> LOCATION '/dualcore/vendors' "
OK
Time taken: 0.531 seconds
[cloudera@quickstart Desktop]$ hcat -e 'show tables'
OK
employee
employee1
employeeexternal
forestfires
product
student
student1
students
updt_emp
vendors
Time taken: 0.589 seconds
```

2) We are describing the vendors table

Command: - hcat -e 'describe vendors'

```
[cloudera@quickstart Desktop]$ hcat -e 'describe vendors'
OK
id                      int
company                 string
email                   string
Time taken: 0.469 seconds
[cloudera@quickstart Desktop]$ hcat -e 'DROP TABLE vendors'
OK
Time taken: 0.934 seconds
```

3) We are deleting the vendors table

Command: - hcat -e 'DROP TABLE vendors'

```
[cloudera@quickstart Desktop]$ hcat -e 'DROP TABLE vendors'
OK
Time taken: 0.934 seconds
```

4) We are listing the tables

Command: - hcat -e 'show tables'

```
[cloudera@quickstart Desktop]$ hcat -e 'show tables'
OK
employee
employee1
employeeexternal
forestfires
product
student
student1
students
updt_emp
Time taken: 0.482 seconds
```

5) We are creating an external table in the below command

Command: -hcat -e "create external table data_ext(Name String, Sal Int)row format delimited fields terminated by ',';"

```
[cloudera@quickstart Desktop]$ hcat -e "create external table data_ext(Name Stri
ng, Sal Int)row format delimited fields terminated by ',';"
OK
Time taken: 0.523 seconds
```

6) We are listing the tables

Command: - hcat -e 'show tables'

```
[cloudera@quickstart Desktop]$ hcat -e 'show tables'
OK
data_ext
employee
employee1
employeeexternal
forestfires
lalita_ext
product
student
student1
students
updt_emp
Time taken: 0.462 seconds
```

SCHEMA

1) We are creating a schema in the below command

Command: - create schema user;

```
hive> create schema user;
OK
Time taken: 1.081 seconds
```

2) We are entering into the schema using below command

Command: - use user;

```
hive> use user;
OK
Time taken: 0.039 seconds
```

JOINS

1) We are creating a table customers for the join

Command: - CREATE TABLE customers(
ID int,
Name string,
Age int,
Address string,
Salary int
)
COMMENT 'This is customers table stored as textfile'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

```
hive> CREATE TABLE customers(
    > ID int,
    > Name string,
    > Age int,
    > Address string,
    > Salary int
    > )
    > COMMENT 'This is customers table stored as textfile'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.345 seconds
```

Table1.txt
The txt file is used to import the data into table

2) We are loading the data from above text file into table customers
Command: - LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Table1.txt' INTO TABLE customers;

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Table1.txt' INTO TABLE custome
rs;
Loading data to table user.customers
Table user.customers stats: [numFiles=1, totalSize=234]
OK
Time taken: 0.296 seconds
```

3) We are displaying the data inside the customers table
Command: - select * from customers;

```
hive> select * from customers;
OK
1       Christy 22      Mumbai  2000
2       Devendra        22      Kurla   2000
3       Chinmay 21      Thane   2001
4       Yash    22      Thane   2001
5       Dhanraj 24      Mumbai  1998
6       Aryan   22      Thane   2000
7       Rahul   22      Kurla   2000
8       Shubham 22      Kurla   2000
9       Imran   24      Mumbai  1998
10      Jason   22      Thane   2000
Time taken: 0.447 seconds, Fetched: 10 row(s)
```

4) We are describing the table customers

Command: -describe customers;

```
hive> describe customers;
OK
id                      int
name                    string
age                     int
address                 string
salary                  int
Time taken: 0.044 seconds, Fetched: 5 row(s)
```

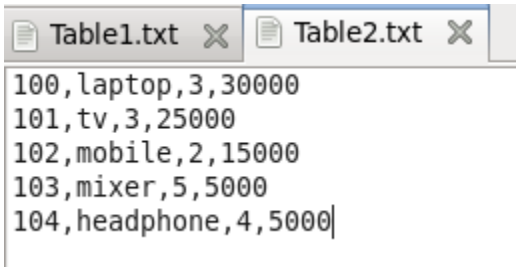
5) We are creating another table called orders for join

Command: - CREATE TABLE orders(
OID int,
order_name string,
Customer_ID int,
Amount int)
COMMENT 'This is orders table stored as textfile'
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

```
hive> CREATE TABLE orders(
    > OID int,
    > order_name string,
    > Customer_ID int,
    > Amount int)
    > COMMENT 'This is orders table stored as textfile'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.053 seconds
```

Table2.txt

This txt file will be used to import data into another table or above table



6) We are loading the txt file data into orders table

Command: - LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Table2.txt' INTO TABLE orders;

```
hive> LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/Table2.txt' INTO TABLE orders;

Loading data to table user.orders
Table user.orders stats: [numFiles=1, totalSize=91]
OK
Time taken: 0.131 seconds
```

7) We are displaying data inside the orders table

Command: -select * from orders;

```
hive> select * from orders;
OK
100     laptop  3       30000
101     tv      3       25000
102     mobile  2       15000
103     mixer   5       5000
104     headphone       4       5000
Time taken: 0.04 seconds, Fetched: 5 row(s)
```

INNER JOIN

Command: - SELECT c.ID,c.NAME,c.AGE,o.AMOUNT FROM CUSTOMERS c JOIN ORDERS o ON(c.ID=o.CUSTOMER_ID);

```
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 0.97 sec   HDFS Read: 6330 HDFS Write: 91 S
UCCESS
Total MapReduce CPU Time Spent: 970 msec
OK
2       Devendra        22      15000
3       Chinmay 21      30000
3       Chinmay 21      25000
4       Yash    22      5000
5       Dhanraj 24      5000
Time taken: 24.171 seconds, Fetched: 5 row(s)
```

LEFT OUTER JOIN

Command: - SELECT c.ID,c.NAME,o.AMOUNT,o.order_name FROM CUSTOMERS c LEFT OUTER JOIN ORDERS o ON(c.ID=o.CUSTOMER_ID);

```
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 0.65 sec   HDFS Read: 6188 HDFS Write: 198
SUCCESS
Total MapReduce CPU Time Spent: 650 msec
OK
1       Christy NULL    NULL
2       Devendra        15000   mobile
3       Chinmay 30000   laptop
3       Chinmay 25000   tv
4       Yash    5000    headphone
5       Dhanraj 5000    mixer
6       Aryan   NULL    NULL
7       Rahul   NULL    NULL
8       Shubham NULL    NULL
9       Imran   NULL    NULL
10      Jason   NULL    NULL
Time taken: 14.628 seconds, Fetched: 11 row(s)
```

RIGHT OUTER JOIN

Command: - SELECT c.ID,c.NAME,c.AGE,o.AMOUNT,o.order_name FROM CUSTOMERS c RIGHT OUTER JOIN ORDERS o ON(c.ID=o.CUSTOMER_ID);

```
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 0.66 sec   HDFS Read: 6133 HDFS Write: 124
SUCCESS
Total MapReduce CPU Time Spent: 660 msec
OK
3       Chinmay 21      30000   laptop
3       Chinmay 21      25000   tv
2       Devendra        22      15000   mobile
5       Dhanraj 24      5000    mixer
4       Yash    22      5000    headphone
Time taken: 14.742 seconds, Fetched: 5 row(s)
```

FULL OUTER JOIN

Command: - SELECT c.ID,c.NAME,c.AGE,o.AMOUNT,o.order_name FROM CUSTOMERS c FULL OUTER JOIN ORDERS o ON(c.ID=o.CUSTOMER_ID);

```
MapReduce Jobs Launched:
Stage-Stage-1: Map: 2  Reduce: 1   Cumulative CPU: 2.68 sec   HDFS Read: 12813 HDFS
 Write: 231 SUCCESS
Total MapReduce CPU Time Spent: 2 seconds 680 msec
OK
1       Christy 22      NULL    NULL
2       Devendra        22      15000   mobile
3       Chinmay 21      25000   tv
3       Chinmay 21      30000   laptop
4       Yash    22      5000    headphone
5       Dhanraj 24      5000    mixer
6       Aryan   22      NULL    NULL
7       Rahul   22      NULL    NULL
8       Shubham 22      NULL    NULL
9       Imran   24      NULL    NULL
10      Jason   22      NULL    NULL
Time taken: 24.418 seconds, Fetched: 11 row(s)
```

VIEWS

1) We are creating the view here

Command: - create view data AS select * from customers where salary>=2000;

```
hive> create view data AS select * from customers where salary>=2000;
OK
Time taken: 0.257 seconds
```

2) We are displaying the view here

select * from data;

```
hive> select * from data;
OK
1       Christy 22      Mumbai  2000
2       Devendra        22      Kurla   2000
3       Chinmay 21      Thane   2001
4       Yash    22      Thane   2001
6       Aryan   22      Thane   2000
7       Rahul   22      Kurla   2000
8       Shubham 22      Kurla   2000
10      Jason   22      Thane   2000
Time taken: 0.116 seconds, Fetched: 8 row(s)
```

3) We are dropping the view here

drop view data;

```
hive> drop view data;
OK
Time taken: 1.251 seconds
```

CREATING INDEX

1) We are creating the index here

Command: - create index index_salary on table customers(salary) As 'org.apache.hadoop.hive.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;

```
hive> create index index_salary on table customers(salary) As 'org.apache.hadoop.hi
ve.ql.index.compact.CompactIndexHandler' WITH DEFERRED REBUILD;
OK
Time taken: 0.343 seconds
```

2) We are displaying the formatted index

Command: - show formatted index on customers;

```
hive> show formatted index on customers;
OK
idx_name                tab_name                col_names               idx_tab_nam
e               idx_type                comment


index_salary            customers               salary                  user__custo
mers_index_salary__     compact
Time taken: 0.065 seconds, Fetched: 4 row(s)
```

3) We are dropping the index

Command: - drop index index_salary ON customers;

```
hive> drop index index_salary ON customers;
OK
Time taken: 0.226 seconds
hive>
```

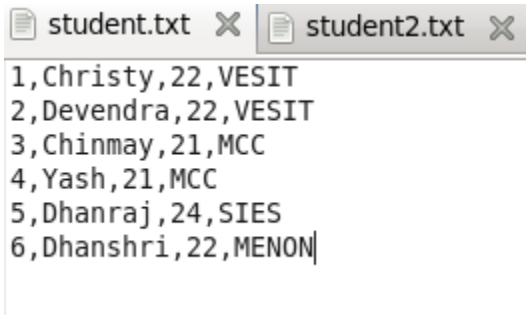
PARTITION

1) We are creating a table for the partition

Command: - create table new_student(id int, name string, age int, institute string) partitioned by(course string) row format delimited fields terminated by ',';

```
hive> create table new_student(id int, name string, age int, institute string) part
itioned by(course string) row format delimited fields terminated by ',';
OK
Time taken: 0.42 seconds
```

student.txt

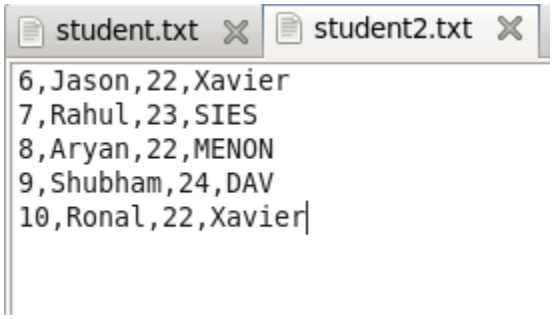


2) We are loading the data for performing the partition

Command: - load data local inpath '/home/cloudera/Desktop/student.txt' into table new_student partition(course= "java");

```
hive> load data local inpath '/home/cloudera/Desktop/student.txt' into table new_st
udent partition(course= "java");
Loading data to table default.new_student partition (course=java)
Partition default.new_student{course=java} stats: [numFiles=1, numRows=0, totalSize
=108, rawDataSize=0]
OK
Time taken: 0.64 seconds
```

student2.txt



3) We are loading the student2.txt for performing the partition

Command: - load data local inpath '/home/cloudera/Desktop/student2.txt' into table new_student partition(course= "hadoop");

```
hive> load data local inpath '/home/cloudera/Desktop/student2.txt' into table new_s
tudent partition(course= "hadoop");
Loading data to table default.new_student partition (course=hadoop)
Partition default.new_student{course=hadoop} stats: [numFiles=1, numRows=0, totalSi
ze=87, rawDataSize=0]
OK
Time taken: 0.196 seconds
```

4) We are displaying the new_student table

Command: -select * from new_student;

```
hive> select * from new_student;
OK
6       Jason   22      Xavier  hadoop
7       Rahul   23      SIES    hadoop
8       Aryan   22      MENON   hadoop
9       Shubham 24      DAV     hadoop
10      Ronal   22      Xavier  hadoop
1       Christy 22      VESIT   java
2       Devendra        22      VESIT   java
3       Chinmay 21      MCC     java
4       Yash    21      MCC     java
5       Dhanraj 24      SIES    java
6       Dhanshri        22      MENON   java
Time taken: 0.278 seconds, Fetched: 11 row(s)
```

5) Displaying the records where course=java

Command: - select * from new_student where course="java";

```
hive> select * from new_student where course="java";
OK
1       Christy 22      VESIT   java
2       Devendra        22      VESIT   java
3       Chinmay 21      MCC     java
4       Yash    21      MCC     java
5       Dhanraj 24      SIES    java
6       Dhanshri        22      MENON   java
Time taken: 0.295 seconds, Fetched: 6 row(s)
```

6) Displaying the records where course=hadoop

Command: - select * from new_student where course="hadoop";

```
hive> select * from new_student where course="hadoop";
OK
6       Jason   22      Xavier  hadoop
7       Rahul   23      SIES    hadoop
8       Aryan   22      MENON   hadoop
9       Shubham 24      DAV     hadoop
10      Ronal   22      Xavier  hadoop
Time taken: 0.073 seconds, Fetched: 5 row(s)
```

7) We are altering the table by dropping the partition where course=hadoop

Command: - ALTER TABLE new_student DROP PARTITION(course="hadoop");

```
hive> ALTER TABLE new_student DROP PARTITION(course="hadoop");
Dropped the partition course=hadoop
OK
Time taken: 0.298 seconds
```

8) We are altering table by renaming new_students to students

Command: - alter table new_student rename to students;

```
hive> alter table new_student rename to students;
OK
Time taken: 0.115 seconds
```

HADOOP

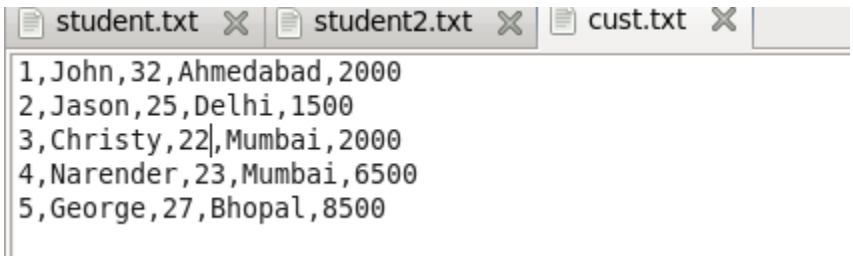
1) Creating table new_customers

Command: -

```
CREATE TABLE new_customers(
ID int,
   > Name string,
   > Age int,
   > Address string,
   > Salary int )
   > COMMENT 'This is customers table stored as textfile'
   > ROW FORMAT DELIMITED
   > FIELDS TERMINATED BY ','
   > STORED AS TEXTFILE;
```

```
hive> CREATE TABLE new_customers(
    > ID int,
    > Name string,
    > Age int,
    > Address string,
    > Salary int )
    > COMMENT 'This is customers table stored as textfile'
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > STORED AS TEXTFILE;
OK
Time taken: 0.055 seconds
```

cust.txt



2) HDFS commands for making directory,uploading and listing

Commands: -

hdfs dfs -mkdir /hivefiles

hdfs dfs -put /home/cloudera/Desktop/cust.txt /hivefiles

hdfs dfs -ls /hivefiles

```
[cloudera@quickstart Desktop]$ hdfs dfs -mkdir /hivefiles
[cloudera@quickstart Desktop]$ hdfs dfs -put /home/cloudera/Desktop/cust.txt /hi
vefiles
22/10/17 03:22:23 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DF
SOutputStream.java:862)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutpu
tStream.java:600)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStre
am.java:789)
[cloudera@quickstart Desktop]$ hdfs dfs -put /home/cloudera/Desktop/cust.txt /hi
vefiles
put: `/hivefiles/cust.txt': File exists
[cloudera@quickstart Desktop]$ hdfs dfs -ls /hivefiles
Found 1 items
-rw-r--r--   1 cloudera supergroup        124 2022-10-17 03:22 /hivefiles/cust.t
xt
[cloudera@quickstart Desktop]$
```

3) We are loading the data

Command: - load data inpath '/hivefiles/cust.txt' into table new_customers;

```
hive> load data inpath '/hivefiles/cust.txt' into table new_customers;
Loading data to table default.new_customers
Table default.new_customers stats: [numFiles=1, totalSize=124]
OK
Time taken: 0.152 seconds
```

3) We are displaying the data

Command: - select * from new_customers;

```
hive> select * from new_customers;
OK
1       John    32      Ahmedabad       2000
2       Jason   25      Delhi   1500
3       Christy 22      Mumbai  2000
4       Narender        23      Mumbai  6500
5       George  27      Bhopal  8500
NULL    NULL    NULL    NULL    NULL
NULL    NULL    NULL    NULL    NULL
Time taken: 0.037 seconds, Fetched: 7 row(s)
hive>
```

CONCLUSION: -

From this practical I have learned Creation of Database and Table, Hive Partition, Hive Built In Function and Operators, Hive View and Index.

PRACTICAL 5

AIM: - Pig: Pig Latin Basic Pig Shell, Pig Data Types, Creating a Pig Data Model, Reading and Storing Data, Pig Operations

THEORY: -

Apache Pig is an abstraction over MapReduce. It is a tool/platform which is used to analyze large sets of data representing them as data flows. Pig is generally used with Hadoop, We can perform all the data manipulation operations in Hadoop using Apache Pig.

To write data analysis programs, Pig provides a high-level language known as Pig Latin. This language provides various operators using which programmers can develop their own functions for reading, writing, and processing data.

To analyze data using Apache Pig, programmers need to write scripts using Pig Latin language. All these scripts are internally converted to Map and Reduce tasks. Apache Pig has a component known as Pig Engine that accepts the Pig Latin scripts as input and converts those scripts into MapReduce jobs.

Why Do We Need Apache Pig?

Programmers who are not so good at Java normally used to struggle working with Hadoop, especially while performing any MapReduce tasks. Apache Pig is a boon for all such programmers.

- Using Pig Latin, programmers can perform MapReduce tasks easily without having to type complex codes in Java.
- Apache Pig uses a multi-query approach, thereby reducing the length of codes. For example, an operation that would require you to type 200 lines of code (LoC) in Java can be easily done by typing as less as just 10 LoC in Apache Pig. Ultimately Apache Pig reduces the development time by almost 16 times.
- Pig Latin is SQL-like language and it is easy to learn Apache Pig when you are familiar with SQL.
- Apache Pig provides many built-in operators to support data operations like joins, filters, ordering, etc. In addition, it also provides nested data types like tuples, bags, and maps that are missing from MapReduce.

Features of Pig

Apache Pig comes with the following features −

- Rich set of operators − It provides many operators to perform operations like join, sort, filer, etc.
- Ease of programming − Pig Latin is similar to SQL and it is easy to write a Pig script if you are good at SQL.
- Optimization opportunities − The tasks in Apache Pig optimize their execution automatically, so the programmers need to focus only on semantics of the language.
- Extensibility − Using the existing operators, users can develop their own functions to read, process, and write data.
- UDF's − Pig provides the facility to create User-defined Functions in other programming languages such as Java and invoke or embed them in Pig Scripts.
- Handles all kinds of data − Apache Pig analyzes all kinds of data, both structured as well as unstructured. It stores the results in HDFS.

COMMAND AND OUTPUTS: -

1) This command will help in entering pig terminal

Command: - pig -x local

```
vaish@vaishVirtualBox:~/Desktop$ pig -x local
2022-10-17 23:25:08,764 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
2022-10-17 23:25:08,769 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
2022-10-17 23:25:09,016 [main] INFO  org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386)
compiled Jun 02 2017, 15:41:58
2022-10-17 23:25:09,017 [main] INFO  org.apache.pig.Main - Logging error messages to: /home/vais
h/Desktop/pig_1666029308997.log
2022-10-17 23:25:09,222 [main] INFO  org.apache.pig.impl.util.Utils - Default bootup file /home/
vaish/.pigbootup not found
2022-10-17 23:25:09,597 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.j
ob.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2022-10-17 23:25:09,608 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEng
ine - Connecting to hadoop file system at: file:///
2022-10-17 23:25:10,126 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-17 23:25:10,187 [main] INFO  org.apache.pig.PigServer - Pig Script ID for the session: P
IG-default-6aefef2f-9cc1-4fdd-a686-ad55fde878ce
2022-10-17 23:25:10,190 [main] WARN  org.apache.pig.PigServer - ATS is disabled since yarn.timel
ine-service.enabled set to false
```

Open different terminal of Ubuntu/Cloudera

2) Making a directory called pigfiles

Command: - hdfs dfs -mkdir /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -mkdir /home/vaish/CHRISTY/pigfiles
```

3) Listing to check whether the directory has been created or not

Command: - hdfs dfs -ls /home/vaish/CHRISTY

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls /home/vaish/CHRISTY
Found 4 items
drwxrwxr-x   - vaish vaish       4096 2022-10-18 05:58 /home/vaish/CHRISTY/PIg
drwxr-xr-x   - vaish vaish       4096 2022-10-18 06:00 /home/vaish/CHRISTY/pifiles
drwxr-xr-x   - vaish vaish       4096 2022-10-18 06:01 /home/vaish/CHRISTY/pigfiles
-rw-rw-rw-   1 vaish vaish     527958 2022-09-29 23:05 /home/vaish/CHRISTY/sales_data_sample.csv
```

record.txt

data1.txt | data2.txt | record.txt

```
1 1,Christy,22,8.52
2 2,Devendra,22,8.5
3 3,Dhanraj,24,23.5
4 4,Chinmay,22,7.7
5 5,Yash,22,23.5
```

4) Uploading the record.txt to pigfiles directory

Command: - hdfs dfs -put /home/vaish/record.txt /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/record.txt /home/vaish/CHRISTY/pigfil
es
```

In pig terminal

5) Loading the file in the below format

Command: - A= LOAD '/home/vaish/CHRISTY/pigfiles/record.txt' USING PigStorage(',') AS (id:int,name:chararray,age:int,package:double);

```
grunt> A= LOAD '/home/vaish/CHRISTY/pigfiles/record.txt' USING PigStorage(',') AS (id:int,name:c
hararray,age:int,package:double);
2022-10-18 06:07:43,387 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

Here I opened another terminal for only the below command. If hadoop localhost is not viewable. Below command will help.

Command: - hdfs dfs -cat /home/vaish/Desktop/sampleoutput/part-m-00000

```
vaish@vaishVirtualBox:~$ hdfs dfs -cat /home/vaish/Desktop/sampleoutput/part-m-
0000

1|Christy|22|8.52
2|Devendra|22|8.5
3|Dhanraj|24|23.5
4|Chinmay|22|7.7
5|Yash|22|23.5
vaish@vaishVirtualBox:~$
```

6) By this command we will check the format of A

Command: - DESCRIBE A;

```
grunt> DESCRIBE A;
A: {id: int,name: chararray,age: int,package: double}
```

7) By this command we will be separating value using (|) symbol in bracket

Command: - store A into 'sampleoutput' using PigStorage('|');

```
grunt> store A into 'sampleoutput' using PigStorage('|');
2022-10-18 06:10:50,156 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum

Success!

Job Stats (time in seconds):
JobId   Maps    Reduces MaxMapTime      MinMapTime      AvgMapTime      MedianMapTime   MaxReduc
eTime   MinReduceTime   AvgReduceTime   MedianReducetime        Alias   Feature Outputs
job_local1477158811_0001        1       0       n/a     n/a     n/a     n/a     0       0       0
0       A       MAP_ONLY        file:///home/vaish/Desktop/sampleoutput,

Input(s):
Successfully read 5 records from: "/home/vaish/CHRISTY/pigfiles/record.txt"

Output(s):
Successfully stored 5 records in: "file:///home/vaish/Desktop/sampleoutput"

Counters:
Total records written : 5
Total bytes written : 0
Spillable Memory Manager spill count : 0
Total bags proactively spilled: 0
Total records proactively spilled: 0
```

8) By this command we will locate the localhost value stored in location

Command: - hdfs dfs -cat /home/vaish/Desktop/sampleoutput/part-m-00000

```
vaish@vaishVirtualBox:~$ hdfs dfs -cat /home/vaish/Desktop/sampleoutput/part-m-
0000

1|Christy|22|8.52
2|Devendra|22|8.5
3|Dhanraj|24|23.5
4|Chinmay|22|7.7
5|Yash|22|23.5
vaish@vaishVirtualBox:~$
```

JOINS

data1.txt

data1.txt

```
1 1,Christy,22,Mumbai,20
2 2,Devendra,22,Pune,24
3 3,Dhanraj,24,Banglore,24
4 4,Chinmay,22,Noida,26
5 5,Yash,22,Hyderabad,26
```

data2.txt

data2.txt

```
1 101,2022-10-16,1,2000
2 102,2022-08-10,2,5000
3 103,2022-06-16,3,5500
4 104,2022-10-02,4,10000
5 105,2022-08-21,5,8000
```

1) We will be uploading data1.txt and data2.txt using put command into pigfiles directory and wil list it

Commands: -

hdfs dfs -put /home/vaish/Desktop/data1.txt /home/vaish/CHRISTY/pigfiles

hdfs dfs -put /home/vaish/Desktop/data2.txt /home/vaish/CHRISTY/pigfiles

hdfs dfs -ls /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/data1.txt /home/vaish/CHRISTY
/pigfiles
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/data2.txt /home/vaish/CHRISTY
/pigfiles
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls  /home/vaish/CHRISTY/pigfiles
Found 3 items
-rw-r--r--   1 vaish vaish        115 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data1.txt
-rw-r--r--   1 vaish vaish        111 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data2.txt
-rw-r--r--   1 vaish vaish         86 2022-10-18 06:05 /home/vaish/CHRISTY/pigfiles/record.txt
```

2) Loading the data into customers variable

Command: - customers = LOAD '/home/vaish/CHRISTY/pigfiles/data1.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);

```
grunt> customers = LOAD '/home/vaish/CHRISTY/pigfiles/data1.txt' USING PigStorage(',') as (id:in
t, name:chararray, age:int, address:chararray, salary:int);
2022-10-18 06:53:40,795 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

3) We are loading data into orders variable

Command: - orders = LOAD '/home/vaish/CHRISTY/pigfiles/data2.txt' USING PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);

```
grunt> orders = LOAD '/home/vaish/CHRISTY/pigfiles/data2.txt' USING PigStorage(',') as (oid:int,
 date:chararray, customer_id:int, amount:int);
2022-10-18 06:54:18,757 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

SELF JOIN

1) We are loading data into customers1 variable

Command: - customers1 = LOAD '/home/vaish/CHRISTY/pigfiles/data1.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);

```
grunt> customers1 = LOAD '/home/vaish/CHRISTY/pigfiles/data1.txt' USING PigStorage(',') as (id:i
nt, name:chararray, age:int, address:chararray, salary:int);
2022-10-18 06:56:54,474 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

2) We are loading data into customers2 variable

Command: - customers2 = LOAD '/home/vaish/CHRISTY/pigfiles/data1.txt' USING PigStorage(',') as (id:int, name:chararray, age:int, address:chararray, salary:int);

```
grunt> customers2 = LOAD '/home/vaish/CHRISTY/pigfiles/data1.txt' USING PigStorage(',') as (id:i
nt, name:chararray, age:int, address:chararray, salary:int);
2022-10-18 06:57:31,876 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

3) We are loading data into customers_self_join variable and performing join

Command: - customers_self_join = JOIN customers1 BY id, customers2 BY id;

```
grunt> customers_self_join = JOIN customers1 BY id, customers2 BY id;
```

4) By Dump command we can display the data

Command: - Dump customers_self_join;

```
grunt> Dump customers_self_join;
2022-10-18 06:59:10,325 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: HASH_JOIN
2022-10-18 06:59:10,350 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
(1,Christy,22,Mumbai,20,1,Christy,22,Mumbai,20)
(2,Devendra,22,Pune,24,2,Devendra,22,Pune,24)
(3,Dhanraj,24,Banglore,24,3,Dhanraj,24,Banglore,24)
(4,Chinmay,22,Noida,26,4,Chinmay,22,Noida,26)
(5,Yash,22,Hyderabad,26,5,Yash,22,Hyderabad,26)
```

LEFT JOIN

1) We are loading data into customers_order_left_join and performing left join

Command: - customer_orders_left_join = JOIN customers BY id LEFT OUTER, orders BY customer_id;

```
grunt> customer_orders_left_join = JOIN customers BY id LEFT OUTER, orders BY customer_id;
```

2) We are displaying the output of left join

Command: - Dump customer_orders_left_join;

```
grunt> Dump customer_orders_left_join;
2022-10-18 21:38:50,802 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: HASH_JOIN
2022-10-18 21:38:50,871 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 21:38:50,875 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
(1,Christy,22,Mumbai,20,101,2022-10-16,1,2000)
(2,Devendra,22,Pune,24,102,2022-08-10,2,5000)
(3,Dhanraj,24,Banglore,24,103,2022-06-16,3,5500)
(4,Chinmay,22,Noida,26,104,2022-10-02,4,10000)
(5,Yash,22,Hyderabad,26,105,2022-08-21,5,8000)
```

RIGHT JOIN

1) We are performing right join and storing it in customer_orders_right_join

Command: - customer_orders_right_join = JOIN customers BY id RIGHT, orders BY customer_id;

```
grunt> customer_orders_right_join = JOIN customers BY id RIGHT, orders BY customer_id;
```

2) We are displaying the output of right join

Command: - Dump customer_orders_right_join;

```
grunt> Dump customer_orders_right_join;
2022-10-18 21:45:09,168 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features u
ed in the script: HASH_JOIN
2022-10-18 21:45:09,255 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.byte
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 21:45:09,256 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBacken
 has already been initialized
(1,Christy,22,Mumbai,20,101,2022-10-16,1,2000)
(2,Devendra,22,Pune,24,102,2022-08-10,2,5000)
(3,Dhanraj,24,Banglore,24,103,2022-06-16,3,5500)
(4,Chinmay,22,Noida,26,104,2022-10-02,4,10000)
(5,Yash,22,Hyderabad,26,105,2022-08-21,5,8000)
```

FULL OUTER JOIN

1) We are performing full outer join and storing it in customer_orders_full_outer_join variable

Command: - customer_orders_full_outer_join = JOIN customers BY id FULL OUTER, orders BY customer_id;

```
grunt> customer_orders_full_outer_join = JOIN customers BY id FULL OUTER, orders BY customer_id;
```

2) We are displaying the output of full outer join

Command: - Dump customer_orders_full_outer_join;

```
grunt> Dump customer_orders_full_outer_join;
2022-10-18 21:49:51,967 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the script: HASH_JOIN
2022-10-18 21:49:52,042 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 21:49:52,050 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
(1,Christy,22,Mumbai,20,101,2022-10-16,1,2000)
(2,Devendra,22,Pune,24,102,2022-08-10,2,5000)
(3,Dhanraj,24,Banglore,24,103,2022-06-16,3,5500)
(4,Chinmay,22,Noida,26,104,2022-10-02,4,10000)
(5,Yash,22,Hyderabad,26,105,2022-08-21,5,8000)
```

EXPLAIN

1) We are explaining the customers here

Command: - explain customers;

```
grunt> explain customers;
2022-10-18 21:52:16,987 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 21:52:16,991 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2022-10-18 21:52:16,992 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULES_ENABLED=[AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupByConstParallelSetter, LimitOptimizer, LoadTypeCastInserter, MergeFilter, MergeForEach, NestedLimitOptimizer, PartitionFilterOptimizer, PredicatePushdownOptimizer, PushDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
```

```
# New Logical Plan:
#-----------------------------------------------
customers: (Name: LOStore Schema: id#708:int,name#709:chararray,age#710:int,address#711:chararra
y,salary#712:int)
|
|---customers: (Name: LOForEach Schema: id#708:int,name#709:chararray,age#710:int,address#711:ch
ararray,salary#712:int)
    |   |
    |   (Name: LOGenerate[false,false,false,false,false] Schema: id#708:int,name#709:chararray,a
ge#710:int,address#711:chararray,salary#712:int)ColumnPrune:OutputUids=[708, 709, 710, 711, 712]
ColumnPrune:InputUids=[708, 709, 710, 711, 712]
    |   |   |
    |   |   (Name: Cast Type: int Uid: 708)
    |   |   |
    |   |   |---id:(Name: Project Type: bytearray Uid: 708 Input: 0 Column: (*))
    |   |   |
    |   |   (Name: Cast Type: chararray Uid: 709)
    |   |   |
    |   |   |---name:(Name: Project Type: bytearray Uid: 709 Input: 1 Column: (*))
    |   |   |
    |   |   (Name: Cast Type: int Uid: 710)
    |   |   |
    |   |   |---age:(Name: Project Type: bytearray Uid: 710 Input: 2 Column: (*))
    |   |   |
    |   |   (Name: Cast Type: chararray Uid: 711)
    |   |   |
    |   |   |---address:(Name: Project Type: bytearray Uid: 711 Input: 3 Column: (*))
    |   |   |
    |   |   (Name: Cast Type: int Uid: 712)
    |   |   |
    |   |   |---salary:(Name: Project Type: bytearray Uid: 712 Input: 4 Column: (*))
    |   |
    |   |---(Name: LOInnerLoad[0] Schema: id#708:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[1] Schema: name#709:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[2] Schema: age#710:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[3] Schema: address#711:bytearray)
    |   |
    |   |---(Name: LOInnerLoad[4] Schema: salary#712:bytearray)
    |
    |---customers: (Name: LOLoad Schema: id#708:bytearray,name#709:bytearray,age#710:bytearray,a
ddress#711:bytearray,salary#712:bytearray)RequiredFields:null
```

```
# Physical Plan:
#-----------------------------------------------
customers: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-241
|
|---customers: New For Each(false,false,false,false,false)[bag] - scope-240
    |   |
    |   Cast[int] - scope-226
    |   |
    |   |---Project[bytearray][0] - scope-225
    |   |
    |   Cast[chararray] - scope-229
    |   |
    |   |---Project[bytearray][1] - scope-228
    |   |
    |   Cast[int] - scope-232
    |   |
    |   |---Project[bytearray][2] - scope-231
    |   |
    |   Cast[chararray] - scope-235
    |   |
    |   |---Project[bytearray][3] - scope-234
    |   |
    |   Cast[int] - scope-238
    |   |
    |   |---Project[bytearray][4] - scope-237
    |
    |---customers: Load(/home/vaish/CHRISTY/pigfiles/data1.txt:PigStorage(',')) - scope-224

2022-10-18 21:52:17,015 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLaye
r.MRCompiler - File concatenation threshold: 100 optimistic? false
2022-10-18 21:52:17,016 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLaye
r.MultiQueryOptimizer - MR plan size before optimization: 1
2022-10-18 21:52:17,023 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLaye
r.MultiQueryOptimizer - MR plan size after optimization: 1
#-----------------------------------------------
# Map Reduce Plan
#-----------------------------------------------
MapReduce node scope-242
Map Plan
customers: Store(fakefile:org.apache.pig.builtin.PigStorage) - scope-241
|
|---customers: New For Each(false,false,false,false,false)[bag] - scope-240
    |   |
    |   Cast[int] - scope-226
    |   |
    |   |---Project[bytearray][0] - scope-225
    |   |
    |   Cast[chararray] - scope-229
    |   |
    |   |---Project[bytearray][1] - scope-228
    |   |
    |   Cast[int] - scope-232
    |   |
    |   |---Project[bytearray][2] - scope-231
    |   |
    |   Cast[chararray] - scope-235
    |   |
    |   |---Project[bytearray][3] - scope-234
    |   |
    |   Cast[int] - scope-238
    |   |
    |   |---Project[bytearray][4] - scope-237
    |
    |---customers: Load(/home/vaish/CHRISTY/pigfiles/data1.txt:PigStorage(',')) - scope-224-----
---
Global sort: false
----------------
```

ILLUSTRATE

1) We are using Illustrate command on customers

Command: - Illustrate customers;

```
grunt> Illustrate customers;
2022-10-18 21:55:20,610 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 21:55:20,618 [main] INFO  org.apache.pig.backend.hadoop.executionengine.HExecutionEng
ine - Connecting to hadoop file system at: file:///
2022-10-18 21:55:20,744 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 21:55:20,745 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimiz
er - {RULES_ENABLED=[ConstantCalculator, LoadTypeCastInserter, PredicatePushdownOptimizer, Strea
---------------------------------------------------------------------------------------------
----------
| customers     | id:int     | name:chararray     | age:int     | address:chararray     | salary
:int     |
---------------------------------------------------------------------------------------------
----------
|               | 4          | Chinmay            | 22          | Noida                 | 26
         |
---------------------------------------------------------------------------------------------
----------
```

GROUP BY

1) We are grouping the data using age attribute

Command: - group_by = GROUP customers by age;

```
grunt> group_by = GROUP customers by age;
```

2) We are displaying the output of grouping data by age

Command: - Dump group_by;

```
grunt> Dump group_by;
2022-10-18 22:00:33,537 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-18 22:00:33,601 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 22:00:33,612 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
(22,{(5,Yash,22,Hyderabad,26),(4,Chinmay,22,Noida,26),(2,Devendra,22,Pune,24),(1,Christy,22,Mumbai,20)})
(24,{(3,Dhanraj,24,Banglore,24)})
grunt>
```

GROUP ALL

1) We are using the group all command

Command: - group_all = GROUP customers all;

```
grunt> group_all = GROUP customers all;
```

2) We are displaying the output of group all

Command: - Dump group_all;

```
grunt> Dump group_all;
2022-10-18 22:05:59,096 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-18 22:05:59,178 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 22:05:59,182 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
(all,{(5,Yash,22,Hyderabad,26),(4,Chinmay,22,Noida,26),(3,Dhanraj,24,Banglore,24),(2,Devendra,22,Pune,24),(1,Christy,22,Mumbai,20)})
```

ORDER BY

1) We are storing data using ORDER by clause and in a descending order using DESC

Command: - order_by_age_data = ORDER customers BY age DESC;

```
grunt> order_by_age_data = ORDER customers BY age DESC;
```

2) We are displaying the output of ORDER by clause

Command: - Dump order_by_age_data;

```
grunt> Dump order_by_age_data;
2022-10-18 22:09:31,036 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: ORDER_BY
2022-10-18 22:09:31,094 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-18 22:09:31,098 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
(3,Dhanraj,24,Banglore,24)
(5,Yash,22,Hyderabad,26)
(4,Chinmay,22,Noida,26)
(2,Devendra,22,Pune,24)
(1,Christy,22,Mumbai,20)
```

LIMIT

1) We are limiting the record and storing it in limit_data

Command: - limit_data = LIMIT customers 2;

```
grunt> limit_data = LIMIT customers 2;
```

2) We are displaying the output of limiting the data

Command: - Dump limit_data;

```
grunt> Dump limit_data;
2022-10-19 06:08:31,786 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: LIMIT
2022-10-19 06:08:31,855 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:08:31,857 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
```

```
(1,Christy,22,Mumbai,20)
(2,Devendra,22,Pune,24)
```

FILTER

1) We are filtering the data where the records are having address Mumbai

Command: - filter_data = FILTER customers BY address == 'Mumbai';

```
grunt> filter_data = FILTER customers BY address == 'Mumbai';
```

2) We are displaying the output of filtering

Command: - Dump filter_data;

```
grunt> Dump filter_data;
2022-10-19 06:11:32,326 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: FILTER
2022-10-19 06:11:32,394 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:11:32,394 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
```

```
(1,Christy,22,Mumbai,20)
```

FOR EACH

1) We are using FOREACH clause here

Command: - foreach_data = FOREACH customers GENERATE id,name,age;

```
grunt> foreach_data = FOREACH customers GENERATE id,name,age;
```

2) We are displaying the data of FOREACH clause

Command: - Dump foreach_data;

```
grunt> Dump foreach_data;
2022-10-19 06:16:10,831 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: UNKNOWN
2022-10-19 06:16:10,902 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:16:10,902 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
```

```
(1,Christy,22)
(2,Devendra,22)
(3,Dhanraj,24)
(4,Chinmay,22)
(5,Yash,22)
```

SPLIT

1) We are storing the records by splitting based on different parameters

Command: - SPLIT customers into customers3 if age<=24, customers4 if (22<age and age>28);

```
grunt> SPLIT customers into customers3 if age<=24, customers4 if (22>=age and age<=24);
```

2) We are displaying the data splitted in customer3

Command: - Dump customers3;

```
grunt> Dump customers3;
2022-10-19 06:26:54,480 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: UNKNOWN
2022-10-19 06:26:54,572 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:26:54,573 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
(1,Christy,22,Mumbai,20)
(2,Devendra,22,Pune,24)
(3,Dhanraj,24,Banglore,24)
(4,Chinmay,22,Noida,26)
(5,Yash,22,Hyderabad,26)
```

3) We are displaying the splitting in customers4

Command: - Dump customers4;

```
grunt> Dump customers4;
2022-10-19 06:28:08,696 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: UNKNOWN
2022-10-19 06:28:08,744 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:28:08,746 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend
 has already been initialized
(1,Christy,22,Mumbai,20)
(2,Devendra,22,Pune,24)
(4,Chinmay,22,Noida,26)
(5,Yash,22,Hyderabad,26)
```

SUM

1) We are grouping the customers

Command: - customers_group = Group customers all;

```
grunt> customers_group = Group customers all;
```

2) Here summing of records is done

Command: - customers_salary_sum = foreach customers_group Generate(customers.name,customers.salary),SUM(customers.salary);

```
grunt> customers_salary_sum = foreach customers_group
>> Generate(customers.name,customers.salary),SUM(customers.salary);
```

3) Here displaying the sum is done

Command: - Dump customers_salary_sum;

```
grunt> Dump customers_salary_sum;
2022-10-19 06:31:56,722 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-19 06:31:56,794 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:31:56,795 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
```

```
(({(Yash),(Chinmay),(Dhanraj),(Devendra),(Christy)},{(26),(26),(24),(24),(20)}),120)
```

MAX

1) We are finding the max salary in customers_group

Command: - customers_salary_max = foreach customers_group Generate(customers.name,customers.salary),MAX(customers.salary);

```
grunt> customers_salary_max = foreach customers_group
>> Generate(customers.name,customers.salary),MAX(customers.salary);
```

2) We are displaying the maximum salary

Command: - Dump customers_salary_max;

```
grunt> Dump customers_salary_max;
2022-10-19 06:35:38,850 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-19 06:35:38,904 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:35:38,906 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
```

```
(({(Yash),(Chinmay),(Dhanraj),(Devendra),(Christy)},{(26),(26),(24),(24),(20)}),26)
```

MIN

1) We are finding the minimum salary in customers_group

Command: - customers_salary_min = foreach customers_group Generate(customers.name,customers.salary),MIN(customers.salary);

```
grunt> customers_salary_min = foreach customers_group
>> Generate(customers.name,customers.salary),MIN(customers.salary);
```

2) We are displaying the minimum salary

Command: - Dump customers_salary_min;

```
grunt> Dump customers_salary_min;
2022-10-19 06:42:12,321 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-19 06:42:12,363 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:42:12,364 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
```

```
(({(Yash),(Chinmay),(Dhanraj),(Devendra),(Christy)},{(26),(26),(24),(24),(20)}),20)
```

AVG

1) We are finding the average salary in customers_group

Command: - customers_salary_avg = foreach customers_group Generate(customers.name,customers.salary),AVG(customers.salary);

```
grunt> customers_salary_avg = foreach customers_group Generate(customers.name,customers.salary),
AVG(customers.salary);
```

2) We are displaying the avg salary

Command: - Dump customers_salary_avg;

```
grunt> Dump customers_salary_avg;
2022-10-19 06:48:30,603 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-19 06:48:30,677 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:48:30,682 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
(({(Yash),(Chinmay),(Dhanraj),(Devendra),(Christy)},{(26),(26),(24),(24),(20)}),24.0)
```

COUNT

1) We are displaying the records by counting based on salary

Command: - customers_salary_count = foreach customers_group Generate(customers.name,customers.salary),COUNT(customers.salary);

```
grunt> customers_salary_count = foreach customers_group Generate(customers.name,customers.salary
),COUNT(customers.salary);
```

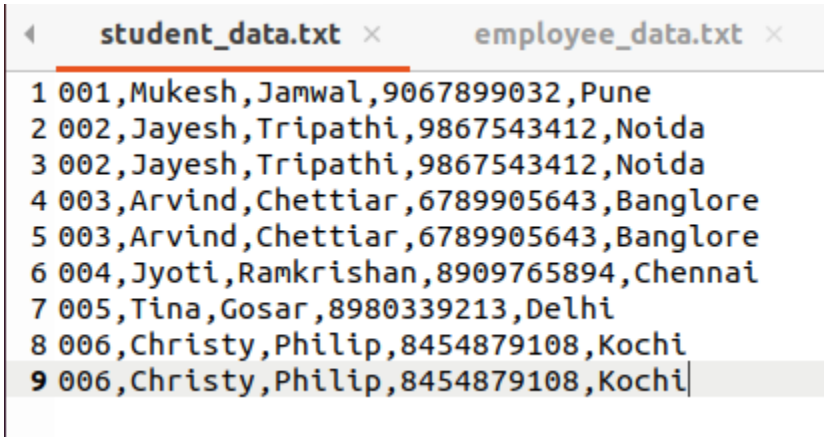
2) We are displaying the count of salaried employees

Command: - Dump customers_salary_count;

```
grunt> Dump customers_salary_count;
2022-10-19 06:52:40,423 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: GROUP_BY
2022-10-19 06:52:40,491 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 06:52:40,498 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
(({(Yash),(Chinmay),(Dhanraj),(Devendra),(Christy)},{(26),(26),(24),(24),(20)}),5)
```

DISTINCT

student_data.txt



1) We are uploading student_data.txt to pigfiles directory using put in the command

Command: - hdfs dfs -put /home/vaish/Desktop/student_data.txt /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/student_data.txt /home/vaish/
CHRISTY/pigfiles
```

2) We are listing pigfiles directory for checking

Command: - hdfs dfs -ls /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls  /home/vaish/CHRISTY/pigfiles
Found 4 items
-rw-r--r--   1 vaish vaish        115 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data1.txt
-rw-r--r--   1 vaish vaish        111 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data2.txt
-rw-r--r--   1 vaish vaish         86 2022-10-18 06:05 /home/vaish/CHRISTY/pigfiles/record.txt
-rw-r--r--   1 vaish vaish        332 2022-10-19 22:31 /home/vaish/CHRISTY/pigfiles/student_data
.txt
```

In Pig Terminal

3) We are loading the data into student_data

Command: - student_data = LOAD '/home/vaish/CHRISTY/pigfiles/student_data.txt' USING PigStorage(',')
as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);

```
grunt> student_data = LOAD '/home/vaish/CHRISTY/pigfiles/student_data.txt' USING PigStorage(',')

>> as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
2022-10-19 23:27:33,710 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

4) We are finding the distinct data

Command: - distinct_data = DISTINCT student_data;

```
grunt> distinct_data = DISTINCT student_data;
```

5) We are displaying the distinct data

Command: - Dump distinct_data;

```
grunt> Dump distinct_data;
2022-10-19 23:27:54,490 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: DISTINCT
2022-10-19 23:27:54,518 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 23:27:54,529 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
(1,Mukesh,Jamwal,9067899032,Pune)
(2,Jayesh,Tripathi,9867543412,Noida)
(3,Arvind,Chettiar,6789905643,Banglore)
(4,Jyoti,Ramkrishan,8909765894,Chennai)
(5,Tina,Gosar,8980339213,Delhi)
(6,Christy,Philip,8454879108,Kochi)
```

UNION

student_data2.txt

student_data2.txt × student_data.txt ×

```
1 007,Dhanshri,Shinde,9848022337,Ahemdabad
2 008,Jason,Lobo,9848022338,Goa
3 009,Rahul,Verma,9848022338,Lucknow
4 010,Dhairya,Savla,7852641235,Surat
```

1) We are uploading the student_data2.txt into pigfiles directory

Command: - hdfs dfs -put /home/vaish/Desktop/student_data2.txt /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/student_data2.txt /home/vaish
/CHRISTY/pigfiles
```

2) We are listing the directory for checking

Command: - hdfs dfs -ls /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls  /home/vaish/CHRISTY/pigfiles
Found 5 items
-rw-r--r--   1 vaish vaish        115 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data1.txt
-rw-r--r--   1 vaish vaish        111 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data2.txt
-rw-r--r--   1 vaish vaish         86 2022-10-18 06:05 /home/vaish/CHRISTY/pigfiles/record.txt
-rw-r--r--   1 vaish vaish        332 2022-10-19 22:31 /home/vaish/CHRISTY/pigfiles/student_data
.txt
-rw-r--r--   1 vaish vaish        144 2022-10-19 23:35 /home/vaish/CHRISTY/pigfiles/student_data
2.txt
```

In Pig terminal

3) We are loading the data into student_data2

Command: - student_data2 = LOAD '/home/vaish/CHRISTY/pigfiles/student_data2.txt' USING PigStorage(',')
as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);

```
grunt> student_data2 = LOAD '/home/vaish/CHRISTY/pigfiles/student_data2.txt' USING PigStorage(',
')
>> as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
2022-10-19 23:41:01,809 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

4) We are performing the Union operation and storing

Command: - student_union = UNION student_data, student_data2;

```
grunt> student_union = UNION student_data, student_data2;
```

5) We are displaying the result of Union operation

Command: - Dump student_union;

```
grunt> Dump student_union;
2022-10-19 23:41:49,337 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: UNION
2022-10-19 23:41:49,371 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 23:41:49,371 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
```

```
(1,Mukesh,Jamwal,9067899032,Pune)
(2,Jayesh,Tripathi,9867543412,Noida)
(2,Jayesh,Tripathi,9867543412,Noida)
(3,Arvind,Chettiar,6789905643,Banglore)
(3,Arvind,Chettiar,6789905643,Banglore)
(4,Jyoti,Ramkrishan,8909765894,Chennai)
(5,Tina,Gosar,8980339213,Delhi)
(6,Christy,Philip,8454879108,Kochi)
(6,Christy,Philip,8454879108,Kochi)
(7,Dhanshri,Shinde,9848022337,Ahemdabad)
(8,Jason,Lobo,9848022338,Goa )
(9,Rahul,Verma,9848022338,Lucknow )
(10,Dhairya,Savla,7852641235,Surat )
```

COGROUP
employee_data.txt

```
employee_data.txt
1 001,Christy,22,Kochi
2 002,Devendra,22,Noida
3 003,Dhanraj,24,Banglore
4 004,Chinmay,22,Pune
5 005,Yash,22,Hyderabad
6 006,Jason,22,Chennai
```

1) We are uploading the employee_data.txt into pigfiles directory
Command: - hdfs dfs -put /home/vaish/Desktop/employee_data.txt /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/employee_data.txt /home/vaish
/CHRISTY/pigfiles
```

2) We are listing the directory for checking
Command: - hdfs dfs -ls /home/vaish/CHRISTY/pigfiles

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls  /home/vaish/CHRISTY/pigfiles
Found 6 items
-rw-r--r--   1 vaish vaish        115 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data1.txt
-rw-r--r--   1 vaish vaish        111 2022-10-18 06:49 /home/vaish/CHRISTY/pigfiles/data2.txt
-rw-r--r--   1 vaish vaish        134 2022-10-19 23:49 /home/vaish/CHRISTY/pigfiles/employee_dat
a.txt
-rw-r--r--   1 vaish vaish         86 2022-10-18 06:05 /home/vaish/CHRISTY/pigfiles/record.txt
-rw-r--r--   1 vaish vaish        332 2022-10-19 22:31 /home/vaish/CHRISTY/pigfiles/student_data
.txt
-rw-r--r--   1 vaish vaish        144 2022-10-19 23:35 /home/vaish/CHRISTY/pigfiles/student_data
2.txt
```

In Pig Terminal
3) We are loading the data into employee_data
employee_data = LOAD '/home/vaish/CHRISTY/pigfiles/employee_data.txt' USING PigStorage(',')
as (id:int, name:chararray, age:int, city:chararray);

```
grunt> employee_data = LOAD '/home/vaish/CHRISTY/pigfiles/employee_data.txt' USING PigStorage(',
')
>> as (id:int, name:chararray, age:int, city:chararray);
2022-10-19 23:51:37,182 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
```

4) We are performing the COGROUP operation and storing

Command: - cogroup_data = COGROUP customers by age, employee_data by age;

```
grunt> cogroup_data = COGROUP customers by age, employee_data by age;
grunt>
```

5) We are displaying the data of COGROUP operation

Command: - Dump cogroup_data;

```
grunt> Dump cogroup_data;
2022-10-19 23:52:45,620 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: COGROUP
2022-10-19 23:52:45,638 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 23:52:45,638 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
(22,{(5,Yash,22,Hyderabad,26),(4,Chinmay,22,Noida,26),(2,Devendra,22,Pune,24),(1,Christy,22,Mumb
ai,20)},{(6,Jason,22,Chennai),(5,Yash,22,Hyderabad ),(4,Chinmay,22,Pune ),(2,Devendra,22,Noida )
,(1,Christy,22,Kochi )})
(24,{(3,Dhanraj,24,Banglore,24)},{(3,Dhanraj,24,Banglore)})
```

CROSS JOIN

1) We are storing the cross join operation

Command: - cross_data = CROSS customers, orders;

```
grunt> cross_data = CROSS customers, orders;
```

2) We are displaying the cross join operation

Command: - Dump cross_data;

```
grunt> Dump cross_data;
2022-10-19 23:53:10,874 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features us
ed in the script: CROSS
2022-10-19 23:53:10,889 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes
.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2022-10-19 23:53:10,890 [main] INFO  org.apache.pig.data.SchemaTupleBackend - Key [pig.schematup
le] was not set... will not generate code.
```

```
(5,Yash,22,Hyderabad,26,105,2022-08-21,5,8000)
(5,Yash,22,Hyderabad,26,104,2022-10-02,4,10000)
(5,Yash,22,Hyderabad,26,103,2022-06-16,3,5500)
(5,Yash,22,Hyderabad,26,102,2022-08-10,2,5000)
(5,Yash,22,Hyderabad,26,101,2022-10-16,1,2000)
(4,Chinmay,22,Noida,26,105,2022-08-21,5,8000)
(4,Chinmay,22,Noida,26,104,2022-10-02,4,10000)
(4,Chinmay,22,Noida,26,103,2022-06-16,3,5500)
(4,Chinmay,22,Noida,26,102,2022-08-10,2,5000)
(4,Chinmay,22,Noida,26,101,2022-10-16,1,2000)
(3,Dhanraj,24,Banglore,24,105,2022-08-21,5,8000)
(3,Dhanraj,24,Banglore,24,104,2022-10-02,4,10000)
(3,Dhanraj,24,Banglore,24,103,2022-06-16,3,5500)
(3,Dhanraj,24,Banglore,24,102,2022-08-10,2,5000)
(3,Dhanraj,24,Banglore,24,101,2022-10-16,1,2000)
(2,Devendra,22,Pune,24,105,2022-08-21,5,8000)
(2,Devendra,22,Pune,24,104,2022-10-02,4,10000)
(2,Devendra,22,Pune,24,103,2022-06-16,3,5500)
(2,Devendra,22,Pune,24,102,2022-08-10,2,5000)
(2,Devendra,22,Pune,24,101,2022-10-16,1,2000)
(1,Christy,22,Mumbai,20,105,2022-08-21,5,8000)
(1,Christy,22,Mumbai,20,104,2022-10-02,4,10000)
(1,Christy,22,Mumbai,20,103,2022-06-16,3,5500)
(1,Christy,22,Mumbai,20,102,2022-08-10,2,5000)
(1,Christy,22,Mumbai,20,101,2022-10-16,1,2000)
```

CONCLUSION: -

I have learned creating a Pig Data Model, Reading and StoringData, Pig Operation.

PRACTICAL 6

AIM: - Spark: RDD, Actions and Transformation on RDD , Ways to Create -file, data in memory, other RDD. Lazy Execution, Persisting RDD

THEORY: -

Apache Spark is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since.

Apache Spark has its architectural foundation in the resilient distributed dataset (RDD), a readonly multiset of data items distributed over a cluster of machines, that is maintained in a fault tolerant way. The Dataframe API was released as an abstraction on top of the RDD, followed by the Dataset API. In Spark 1.x, the RDD was the primary application programming interface (API), but as of Spark 2.x use of the Dataset API is encouraged even though the RDD API is not deprecated. The RDD technology still underlies the Dataset API. Spark and its RDDs were developed in 2012 in response to limitations in the MapReduce cluster computing paradigm, which forces a particular linear dataflow structure on distributed programs, MapReduce programs read input data from disk, map a function across the data, reduce the results of the map, and store reduction results on disk. Spark's RDDs function as a working set for distributed programs that offers a (deliberately) restricted form of distributed shared memory.

Inside Apache Spark the workflow is managed as a directed acyclic graph (DAG). Nodes represent RDDs while edges represent the operations on the RDDs.

Spark facilitates the implementation of both iterative algorithms, which visit their data set multiple times in a loop, and interactive/exploratory data analysis, i.e., the repeated database-style querying of data. The latency of such applications may be reduced by several orders of magnitude compared to Apache Hadoop MapReduce implementation. Among the class of iterative algorithms are the training algorithms for machine learning systems, which formed the initial impetus for developing Apache Spark.

Apache Spark requires a cluster manager and a distributed storage system. For cluster management, Spark supports standalone (native Spark cluster, where you can launch a cluster

either manually or use the launch scripts provided by the install package. It is also possible to run these daemons on a single machine for testing), Hadoop YARN, Apache Mesos or Kubernetes. For distributed storage, Spark can interface with a wide variety, including
Alluxio, Hadoop Distributed File System (HDFS), MapR File System
(MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu, Lustre file system, or a custom solution can be implemented. Spark also supports a pseudo-distributed local mode, usually used only for development or testing purposes, where distributed storage is not required and the local file system can be used instead; in such a scenario, Spark is run on a single machine with one executor per CPU core.

COMMAND AND OUTPUTS: -

For editing the document

To open a document

Command: - gedit doc.txt

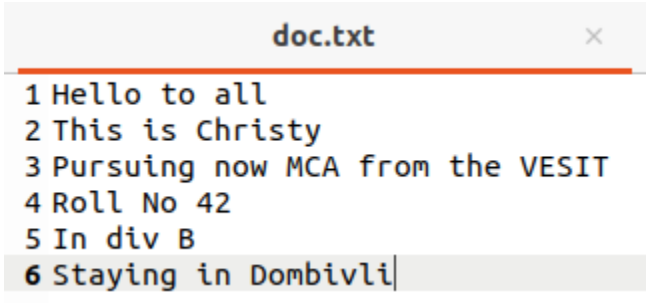
```
vaish@vaishVirtualBox:~/Desktop$ gedit doc.txt
```

1) We will enter into spark shell by command given below

Command: - spark-shell

```
vaish@vaishVirtualBox:~$ spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
22/10/19 10:56:31 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Spark context Web UI available at http://vaishVirtualBox:4040
Spark context available as 'sc' (master = local[*], app id = local-1666157193091
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Scala version 2.12.15 (OpenJDK 64-Bit Server VM, Java 11.0.16)
Type in expressions to have them evaluated.
Type :help for more information.
```

doc.txt



Open new terminal window of Clouder/Ubuntu

2) We are making a Directory called the spark

Command: - hdfs dfs -mkdir /home/vaish/Desktop/spark

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -mkdir /home/vaish/Desktop/spark
```

3) We are uploading the doc.txt into spark directory using put command

Command: - hdfs dfs -put /home/vaish/Desktop/doc.txt /home/vaish/Desktop/spark

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/doc.txt /home/vaish/Desktop/spark
```

4) We are listing the directory for checking

Command: - dfs dfs -ls /home/vaish/Desktop/spark

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls /home/vaish/Desktop/spark
Found 1 items
-rw-r--r--   1 vaish vaish        101 2022-10-19 11:07 /home/vaish/Desktop/spark/doc.txt
```

In Pig Terminal

5) This is used to check the application used

Command: - sc.appName

```
scala> sc.appName
res0: String = Spark shell
```

6) We are loading the text file into mydata variable

Command: - val mydata=sc.textFile("/home/vaish/Desktop/spark/doc.txt")

```
scala> val mydata=sc.textFile("/home/vaish/Desktop/spark/doc.txt")
mydata: org.apache.spark.rdd.RDD[String] = /home/vaish/Desktop/spark/doc.txt MapPartitionsRDD[5] at textFile at <console>:23
```

7) We are counting the data in mydata variable of doc.txt

Command: - mydata.count()

```
scala> mydata.count()
res2: Long = 6
```

8) This command will print first to line of code

Command: - for(line<- mydata.take(2))
println(line)

```
scala> for(line<- mydata.take(2))
     | println(line)
Hello to all
This is Christy
```

9) This command will convert the letters to Uppercase letters

Command: - val newdata = mydata.map(line=> line.toUpperCase)

```
scala> val newdata = mydata.map(line=> line.toUpperCase)
newdata: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[6] at map at <console>:23
```

10) The output of above command

Command: - for(line <- newdata.take(2)) println(line)

```
scala> for(line <- newdata.take(2)) println(line)
HELLO TO ALL
THIS IS CHRISTY
```

11) This command will print the line ending with l

Command: - val filt_data = mydata.filter(line => line.endsWith("l"))

```
scala> val filt_data = mydata.filter(line => line.endsWith("l"))
filt_data: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at filter at <console>:23
```

12) The printing of above command

Command: - for(line <- filt_data.take(2)) println(line)

```
scala> for(line <- filt_data.take(2)) println(line)
Hello to all
```

13) This command will print the line ending with T

Command: - val filt_data = mydata.filter(line =>line.endsWith("T"))

```
scala> val filt_data = mydata.filter(line =>line.endsWith("T"))
filt_data: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[8] at filter at <console>:23
```

14) The output of above command

Command: - for(line <- filt_data.take(2)) println(line)

```
scala> for(line <- filt_data.take(2)) println(line)
Pursuing now MCA from the VESIT
```

15) The command is storing the values as list

Command: - val datanum = sc.parallelize(List(10, 20, 30))

```
scala> val datanum = sc.parallelize(List(10, 20, 30))
datanum: org.apache.spark.rdd.RDD[Int] = ParallelCollectionRDD[9] at parallelize at <console>:23
```

16) The output of the data in datanum variable

Command: - datanum.collect()

```
scala> datanum.collect()
res7: Array[Int] = Array(10, 20, 30)
```

17) The value is incremented by 10 in the above list

Command: - val mapfunc = datanum.map(x=> x+10)

```
scala> val mapfunc = datanum.map(x=> x+10)
mapfunc: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[10] at map at <console>:23
```

18) The output of the incremented value

Command: - mapfunc.collect()

```
scala> mapfunc.collect()
res9: Array[Int] = Array(20, 30, 40)
```

19) In this value is being stored

Command: - val name= Seq("Christy","Philip")

```
scala> val name= Seq("Christy","Philip")
name: Seq[String] = List(Christy, Philip)
```

20) The value in above name variable is converted to lowercase

Command: - val result1= name.map(_.toLowerCase)

```
scala> val result1= name.map(_.toLowerCase)
result1: Seq[String] = List(christy, philip)
```

21) This will separate each letter in a name varaiable

Command: - name.flatMap(_.toLowerCase)

```
scala> name.flatMap(_.toLowerCase)
res10: Seq[Char] = List(c, h, r, i, s, t, y, p, h, i, l, i, p)
```

data.txt

data.txt

```
 1 Christy Philip
 2 Christy Philip
 3 Christy Philip
 4 Devendra Yadav
 5 Devendra Yadav
 6 Dhanraj Shetty
 7 Dhanraj Shetty
 8 Chinmay Vyapari
 9 Chinmay Vyapari
10 Chinmay Vyapari
11 Yash Sawant
12 Yash Sawant
```

22) The above data.txt file is uploaded in spark directory using put command

Command: - hdfs dfs -put /home/vaish/Desktop/data.txt /home/vaish/Desktop/spark

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -put /home/vaish/Desktop/data.txt /home/vaish/Desktop/spark
```

23) We are listing the directory for checking

Command: - hdfs dfs -ls /home/vaish/Desktop/spark

```
vaish@vaishVirtualBox:~/Desktop$ hdfs dfs -ls /home/vaish/Desktop/spark
Found 2 items
-rw-r--r--   1 vaish vaish        177 2022-10-19 11:34 /home/vaish/Desktop/spark/data.txt
-rw-r--r--   1 vaish vaish        101 2022-10-19 11:07 /home/vaish/Desktop/spark/doc.txt
```

24) The doc.txt file is stored in a1 variable

Command: - val a1 = sc.textFile("/home/vaish/Desktop/spark/data.txt")

```
scala> val a1 = sc.textFile("/home/vaish/Desktop/spark/data.txt")
a1: org.apache.spark.rdd.RDD[String] = /home/vaish/Desktop/spark/data.txt MapPartitionsRDD[12] at textFile at <console>:23
```

25) This will print all the content in data.txt which is stored in a1 variable

Command: - a1.collect()

```
scala> a1.collect()
res11: Array[String] = Array(Christy Philip, Christy Philip, Christy Philip, Devendra Yadav, Deven
dra Yadav, Dhanraj Shetty, Dhanraj Shetty, Chinmay Vyapari, Chinmay Vyapari, Chinmay Vyapari, Yash
 Sawant, Yash Sawant)
```

26) This will print number of contents in data.txt file which is in a1 variable

Command: - a1.count()

```
scala> a1.count()
res12: Long = 12
```

27) The data is being split using split function

Command: - val splitdata=a1.flatMap(line => line.split(" "));

```
scala> val splitdata=a1.flatMap(line => line.split(" "));
splitdata: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[13] at flatMap at <console>:23
```

28) The splitted data is printed

Command: - splitdata.collect()

```
scala> splitdata.collect()
res13: Array[String] = Array(Christy, Philip, Christy, Philip, Christy, Philip, Devendra, Yadav, D
evendra, Yadav, Dhanraj, Shetty, Dhanraj, Shetty, Chinmay, Vyapari, Chinmay, Vyapari, Chinmay, Vya
pari, Yash, Sawant, Yash, Sawant)
```

29) Here in mapdata variable the each word is assigned with number 1

Command: - val mapdata = splitdata.map(word =>(word,1))

```
scala> val mapdata = splitdata.map(word =>(word,1))
mapdata: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[14] at map at <console>:23
```

30) The mapdata variable is printed

Command: - mapdata.collect()

```
scala> mapdata.collect()
res14: Array[(String, Int)] = Array((Christy,1), (Philip,1), (Christy,1), (Philip,1), (Christy,1),
 (Philip,1), (Devendra,1), (Yadav,1), (Devendra,1), (Yadav,1), (Dhanraj,1), (Shetty,1), (Dhanraj,1
), (Shetty,1), (Chinmay,1), (Vyapari,1), (Chinmay,1), (Vyapari,1), (Chinmay,1), (Vyapari,1), (Yash
,1), (Sawant,1), (Yash,1), (Sawant,1))
```

31) In reducedata each word count is recorded

Command: - val reducedata = mapdata.reduceByKey(_+_);

```
scala> val reducedata = mapdata.reduceByKey(_+_);
reducedata: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[15] at reduceByKey at <console>:
23
```

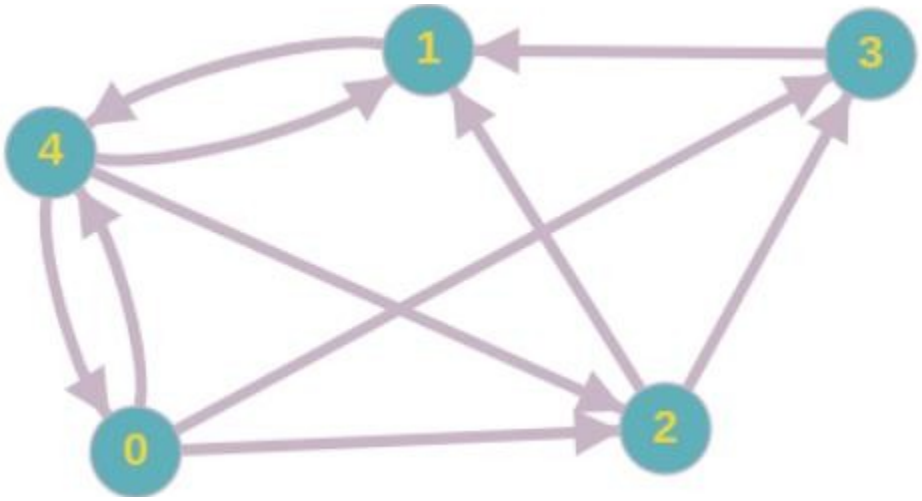
32) The result of variable reducedata is being printed

Command: - reducedata.collect()

```
scala> reducedata.collect()
res15: Array[(String, Int)] = Array((Vyapari,3), (Shetty,2), (Devendra,2), (Chinmay,3), (Christy,3
), (Yash,2), (Dhanraj,2), (Philip,3), (Yadav,2), (Sawant,2))
```

PAGERANK

The PageRank algorithm or Google algorithm was introduced by Larry Page, one of the founders of Google. It was first used to rank web pages in the Google search engine. Nowadays, it is more and more used in many different fields, for example in ranking users in social media etc… What is fascinating with the PageRank algorithm is how to start from a complex problem and end up with a very simple solution. You just need to have some basics in algebra and Markov Chains. Here, we will use ranking web pages as a use case to illustrate the PageRank algorithm.



The web can be represented like a directed graph where nodes represent the web pages and edges form links between them. Typically, if a node (web page) i is linked to a node j, it means that i refers to j.

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

We have to define what is the importance of a web page. As a first approach, we could say that it is the total number of web pages that refer to it. If we stop to this criteria, the importance of these web pages that refer to it is not taken into account. In other words, an important web page and a less important one has the same weight. Another approach is to assume that a web page spreads its importance equally to all web pages it links to. By doing that, we can then define the score of a node j as follows:

where r_i is the score of the node i and d_i its out-degree. PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents, such as the World Wide Web, with the purpose of "measuring" its relative importance within the set. The algorithm may be applied to any collection of entities with reciprocal quotations and references. The numerical weight that it assigns to any given element E is referred to as the PageRank of E and denoted by PR(E).

A PageRank results from a mathematical algorithm based on the webgraph, created by all World Wide Web pages as nodes and hyperlinks as edges, taking into consideration authority hubs such as cnn.com or mayoclinic.org. The rank value indicates an importance of a particular page. A hyperlink to a page counts as a vote of support. The PageRank of a page is defined recursively and depends on the number and PageRank metric of all pages that link to it ("incoming links"). A page that is linked to by many pages with high PageRank receives a high rank itself.

Numerous academic papers concerning PageRank have been published since Page and Brin's original paper. In practice, the PageRank concept may be vulnerable to manipulation. Research has been conducted into identifying falsely influenced PageRank rankings. The goal is to find an effective means of ignoring links from documents with falsely influenced PageRank.

Algorithm:

The PageRank algorithm outputs a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. PageRank can be calculated for collections of documents of any size. It is assumed in several research papers that the distribution is evenly divided among all documents in the collection at the beginning of the computational process. The PageRank computations require several passes, called "iterations", through the collection to adjust approximate PageRank values to more closely reflect the theoretical true value.

A probability is expressed as a numeric value between 0 and 1. A 0.5 probability is commonly expressed as a "50% chance" of something happening. Hence, a document with a PageRank of 0.5 means there is a 50% chance that a person clicking on a random link will be directed to said document.

Data set used:

Kaggle dataset link: https://www.kaggle.com/pappukrjha/google-web-graph

STEPS: -

1) Download the dataset from the above link.

2) Extract it

3) Keep it at a proper location

web-Google.txt

web-Google.txt

```
# Directed graph (each unordered pair of nodes is saved once): web-
Google.txt
# Webgraph from the Google programming contest, 2002
# Nodes: 875713 Edges: 5105039
# FromNodeId    ToNodeId
0       11342
0       824020
0       867923
0       891835
11342   0
11342   27469
11342   38716
11342   309564
11342   322178
11342   387543
11342   427436
11342   538214
11342   638706
11342   645018
11342   835220
11342   856657
11342   867923
```

1) Make a directory called pgrank

Command: - hdfs dfs -mkdir /pgrank

```
[cloudera@quickstart Desktop]$ hdfs dfs -mkdir /pgrank
```

2) Uploading web-Google.txt into pgrank directory using put command

Command: - hdfs dfs -put /home/cloudera/Desktop/web-Google.txt /pgrank

```
[cloudera@quickstart Desktop]$ hdfs dfs -put /home/cloudera/Desktop/web-Googl
e.txt /pgrank
```

3) Type pyspark in command line of cloudera/Ubuntu to enter pyspark

Command: - pyspark

```
[cloudera@quickstart Desktop]$ pyspark
Python 2.6.6 (r266:84292, Jul 23 2015, 15:22:56)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-11)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
22/10/19 00:39:46 INFO spark.SparkContext: Running Spark version 1.6.0
```

4) Code1

Command: -

def computeContribs(neighbors,rank):

for neighbor in neighbors: yield(neighbor, rank/len(neighbors))

```
>>> def computeContribs(neighbors,rank):
...     for neighbor in neighbors: yield(neighbor, rank/len(neighbors))
...
```

5) Code2

Command: -

links = sc.textFile('/pgrank/web-Google.txt')\
.map(lambda line: line.split())\
.map(lambda pages: (pages[0],pages[1]))\
.distinct()\
.groupByKey()\
.persist()

```
>>> links = sc.textFile('/pgrank/web-Google.txt')\
...     .map(lambda line: line.split())\
...     .map(lambda pages: (pages[0],pages[1]))\
...     .distinct()\
...     .groupByKey()\
...     .persist()
22/10/19 00:51:35 INFO storage.MemoryStore: Block broadcast_2 stored as value
s in memory (estimated size 191.1 KB, free 450.0 KB)
22/10/19 00:51:35 INFO storage.MemoryStore: Block broadcast_2_piece0 stored a
s bytes in memory (estimated size 22.1 KB, free 472.1 KB)
22/10/19 00:51:35 INFO storage.BlockManagerInfo: Added broadcast_2_piece0 in
memory on localhost:42813 (size: 22.1 KB, free: 534.5 MB)
22/10/19 00:51:35 INFO spark.SparkContext: Created broadcast 2 from textFile
at NativeMethodAccessorImpl.java:-2
22/10/19 00:51:35 INFO mapred.FileInputFormat: Total input paths to process :
 1
```

6) Code3

Command: -

ranks=links.map(lambda (page, neighbors): (page, 1.0))

```
>>> ranks=links.map(lambda (page, neighbors): (page, 1.0))
>>> for x in xrange(10):
```

7) Code4

Command: -

```
for x in xrange(10):
    contribs=links\
    .join(ranks)\
    .flatMap(lambda(page,(neighbors, rank)):
 computeContribs(neighbors,rank))
    ranks=contribs\
    .reduceByKey(lambda v1,v2: v1+v2)\
    .map(lambda (page, contrib):(page, contrib*0.85+0.15))
```

```
>>> for x in xrange(10):
...     contribs=links\
...     .join(ranks)\
...     .flatMap(lambda(page,(neighbors, rank)):
... computeContribs(neighbors,rank))
...     ranks=contribs\
...     .reduceByKey(lambda v1,v2: v1+v2)\
...     .map(lambda (page, contrib):(page, contrib*0.85+0.15))
...
```

8) Command: -

for rank in ranks.collect(): print rank

```
>>> for rank in ranks.collect(): print rank
...
22/10/19 01:11:44 INFO spark.SparkContext: Starting job: collect at <stdin>:1
22/10/19 01:11:44 INFO scheduler.DAGScheduler: Registering RDD 7 (distinct at <stdin>:3)
22/10/19 01:11:44 INFO scheduler.DAGScheduler: Registering RDD 11 (groupByKey at <stdin>:3)
22/10/19 01:11:44 INFO scheduler.DAGScheduler: Registering RDD 19 (join at <stdin>:3)
22/10/19 01:11:44 INFO scheduler.DAGScheduler: Registering RDD 23 (reduceByKey at <stdin>:7)
```

```
(u'878116', 0.16532200752168372)
(u'193065', 0.21335268939024027)
(u'207776', 0.4784688435924187)
(u'520006', 1.4939375405067166)
(u'47512', 0.95522978862440777)
(u'414854', 0.17968807568616471)
(u'312954', 0.1999441128444194)
(u'482296', 0.33699591121104439)
(u'880182', 4.4520706620457107)
(u'855612', 0.59704848951968537)
(u'79707', 1.0201882285040331)
(u'485322', 2.5817713769920418)
(u'335595', 0.27021269289160449)
(u'827542', 1.6544414686855093)
(u'495379', 0.47581635802320144)
(u'76215', 0.75559977686470647)
(u'841795', 0.61377566996716659)
(u'582993', 0.18157405762524642)
(u'73661', 0.20139829821729688)
(u'792921', 0.4573366976544686)
(u'459241', 0.23088789435816665)
(u'482951', 0.15932065764399067)
(u'163385', 0.22840123193163842)
(u'457326', 0.16599964554669411)
(u'120852', 0.17739287525079489)
(u'713032', 0.40471270576244489)
(u'905074', 0.20680606166936394)
(u'493124', 0.22760856034351734)
(u'368333', 0.28078601804338288)
(u'911210', 0.25828958283707476)
```

9) Command: - ranks.take(5)

```
>>> ranks.take(5)
22/10/19 01:54:05 INFO spark.SparkContext: Starting job: runJob at PythonRDD.scala:393
22/10/19 01:54:05 INFO spark.MapOutputTrackerMaster: Size of output statuses for shuffle 21 is 143 bytes
22/10/19 01:54:05 INFO spark.MapOutputTrackerMaster: Size of output statuses for shuffle 20 is 143 bytes
22/10/19 01:54:05 INFO spark.MapOutputTrackerMaster: Size of output statuses for shuffle 19 is 155 bytes
22/10/19 01:54:05 INFO spark.MapOutputTrackerMaster: Size of output statuses for shuffle 18 is 152 bytes


[(u'322044', 0.31368969580732142), (u'681601', 0.20267241923465629), (u'40135', 0.17065510181207566), (u'
880578', 0.94074180913713767), (u'110557', 0.21491360381831726)]
```

10) Commands: -

ranks.saveAsTextFile('page_ranks_output')

hdfs dfs -cat /user/cloudera/page_ranks_output/part-00000

```
>>> ranks.saveAsTextFile('page_ranks_output')
22/10/19 01:56:13 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 1
22/10/19 01:56:13 INFO spark.SparkContext: Starting job: saveAsTextFile at NativeMethodAccessorImpl.ja
22/10/19 01:56:13 INFO scheduler.DAGScheduler: Got job 2 (saveAsTextFile at NativeMethodAccessorImpl.j
22/10/19 01:56:13 INFO scheduler.DAGScheduler: Final stage: ResultStage 68 (saveAsTextFile at NativeMe
22/10/19 01:56:13 INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 67)
22/10/19 01:56:13 INFO scheduler.DAGScheduler: Missing parents: List()
22/10/19 01:56:13 INFO scheduler.DAGScheduler: Submitting ResultStage 68 (MapPartitionsRDD[129] at sav
sing parents
```

This is opened on localhost:8080 in Browser

Hadoop Overview Datanodes Snapshot Startup Progress Utilities

Browse Directory

/user/cloudera Go!

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
drwxr-xr-x	cloudera	cloudera	0 B	Wed Oct 19 01:56:18 -0700 2022	0	0 B	page_ranks_output

Hadoop, 2016.

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	cloudera	cloudera	0 B	Wed Oct 19 01:56:18 -0700 2022	1	128 MB	_SUCCESS
-rw-r--r--	cloudera	cloudera	1.86 MB	Wed Oct 19 01:56:13 -0700 2022	1	128 MB	part-00000
-rw-r--r--	cloudera	cloudera	1.86 MB	Wed Oct 19 01:56:14 -0700 2022	1	128 MB	part-00001
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:15 -0700 2022	1	128 MB	part-00002
-rw-r--r--	cloudera	cloudera	1.84 MB	Wed Oct 19 01:56:15 -0700 2022	1	128 MB	part-00003
-rw-r--r--	cloudera	cloudera	1.84 MB	Wed Oct 19 01:56:15 -0700 2022	1	128 MB	part-00004
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:16 -0700 2022	1	128 MB	part-00005
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:16 -0700 2022	1	128 MB	part-00006
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:16 -0700 2022	1	128 MB	part-00007
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:17 -0700 2022	1	128 MB	part-00008
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:17 -0700 2022	1	128 MB	part-00009
-rw-r--r--	cloudera	cloudera	1.85 MB	Wed Oct 19 01:56:17 -0700 2022	1	128 MB	part-00010

Command is typed on Ubuntu/Cloudera terminal

```
[cloudera@quickstart Desktop]$ hdfs dfs -cat /user/cloudera/page_ranks_output/part-00000
```

```
(u'492052', 0.21168161068005553)
(u'651490', 0.66318705282791124)
(u'213550', 0.96882091541661453)
(u'306548', 0.21202108953420662)
(u'402085', 0.9900401629872162)
(u'253445', 0.35876979715144203)
(u'332381', 2.5081539351684983)
(u'431585', 0.21763427875582245)
(u'585564', 0.16347770246730145)
(u'644851', 0.40519734489958781)
(u'120457', 0.34061227087104884)
(u'309254', 0.18479206881737004)
(u'580938', 3.5445059783343749)
(u'585450', 0.38550849440624901)
(u'357414', 1.3334679226289725)
(u'266067', 0.3019415477221194)
(u'758218', 0.46269823635865093)
(u'915208', 0.32135540383169331)
(u'540958', 0.25320613933405212)
(u'460667', 0.19652997422137206)
(u'420606', 0.3604566647701376)
(u'893311', 0.21131078743369167)
(u'151932', 0.23335128722170823)
(u'758215', 0.33140638704592368)
(u'259396', 0.16447488489783649)
(u'331036', 0.29913811640760046)
(u'126248', 0.21635831526818533)
(u'586793', 0.5951795324883431)
(u'107946', 2.3826261432985563)
(u'194749', 0.29908518858453853)
(u'738828', 0.17388151934548934)
(u'635571', 0.18914443110828827)
(u'776736', 0.27629488205270702)
```

Page rank can be used to rank pages according to the importance, by assigning the numerical weighting to each element of a hyperlinked set of documents. The algorithm can be easily implemented in hadoop cluster in spark.

CONCLUSION: -

From this practical I have learned to create all mentioned operations Spark: RDD, Actions and Transformation on RDD , Ways to Create -file, data in memory, other RDD. Lazy Execution, Persisting RDD and PageRank.

PRACTICAL 7

AIM: - Visualization: Connect to data, Build Charts and Analyze Data, Create Dashboard, Create Stories using Tableau

THEORY: -

Tableau is a Business Intelligence tool for visually analyzing the data. Users can create and distribute an interactive and shareable dashboard, which depict the trends, variations, and density of the data in the form of graphs and charts. Tableau can connect to files, relational and Big Data sources to acquire and process data. The software allows data blending and real-time collaboration, which makes it very unique. It is used by businesses, academic researchers, and many government organizations for visual data analysis. It is also positioned as a leader Business Intelligence and Analytics Platform in Gartner Magic Quadrant.

Drive Link

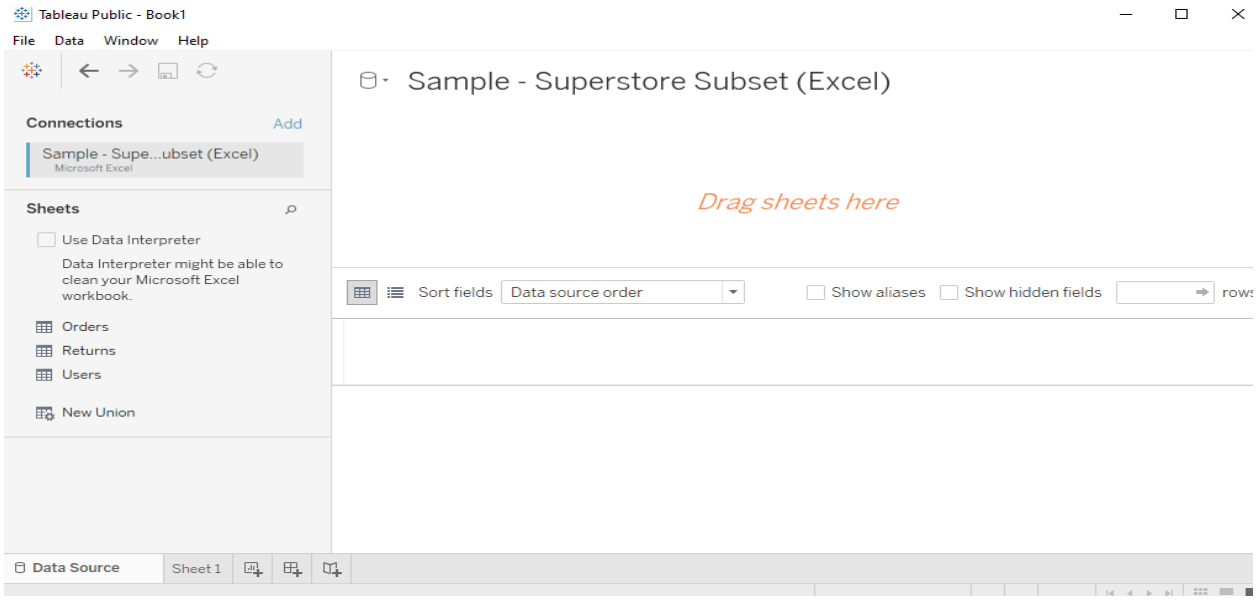
Download Sales.txt file and save it as json file

https://drive.google.com/drive/folders/1v6T9_Xq6tVrYy1tl_f_sb_P8GoP5C-m6?usp=sharing

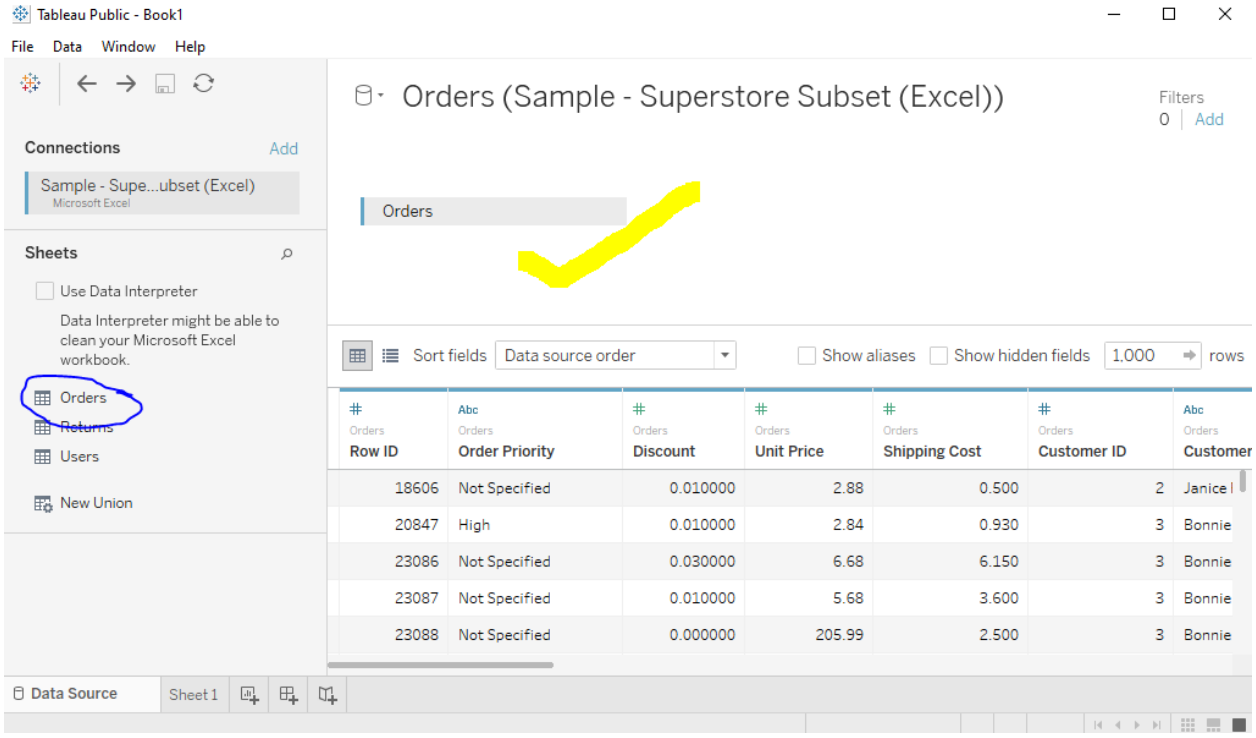
ASSIGNMENT 1

Q 1: Find the customer with the highest overall profit. What is his/her profit ratio?

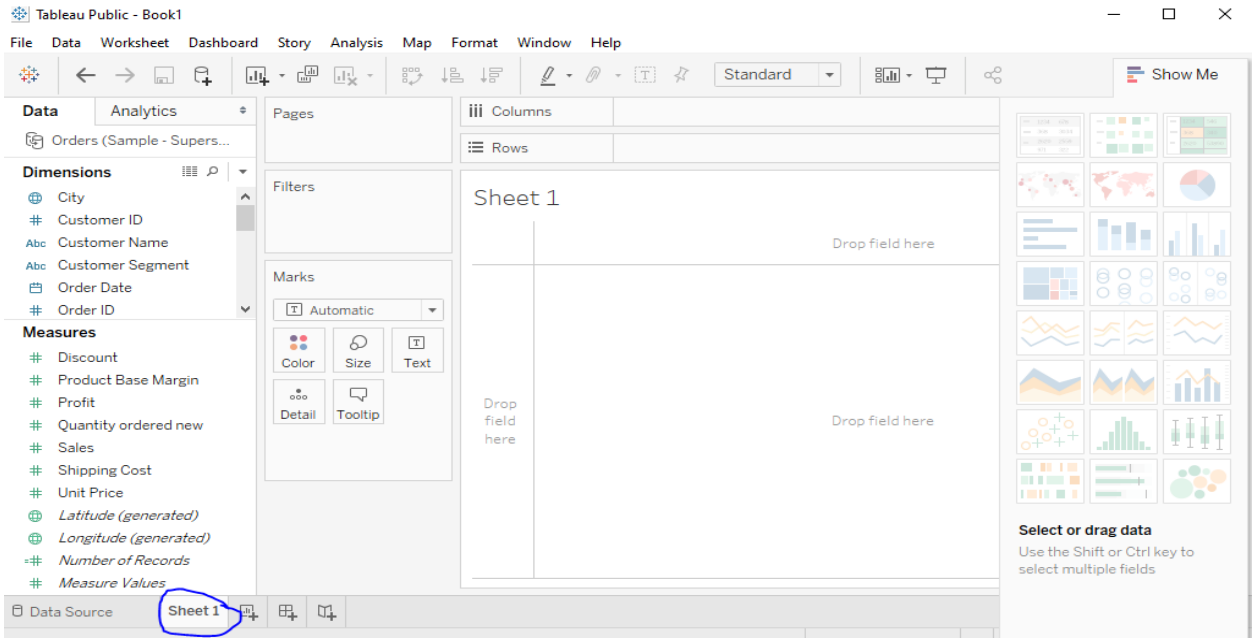
Step 1: Open the superstoreus2015 excel data set by browsing from where you have downloaded and kept



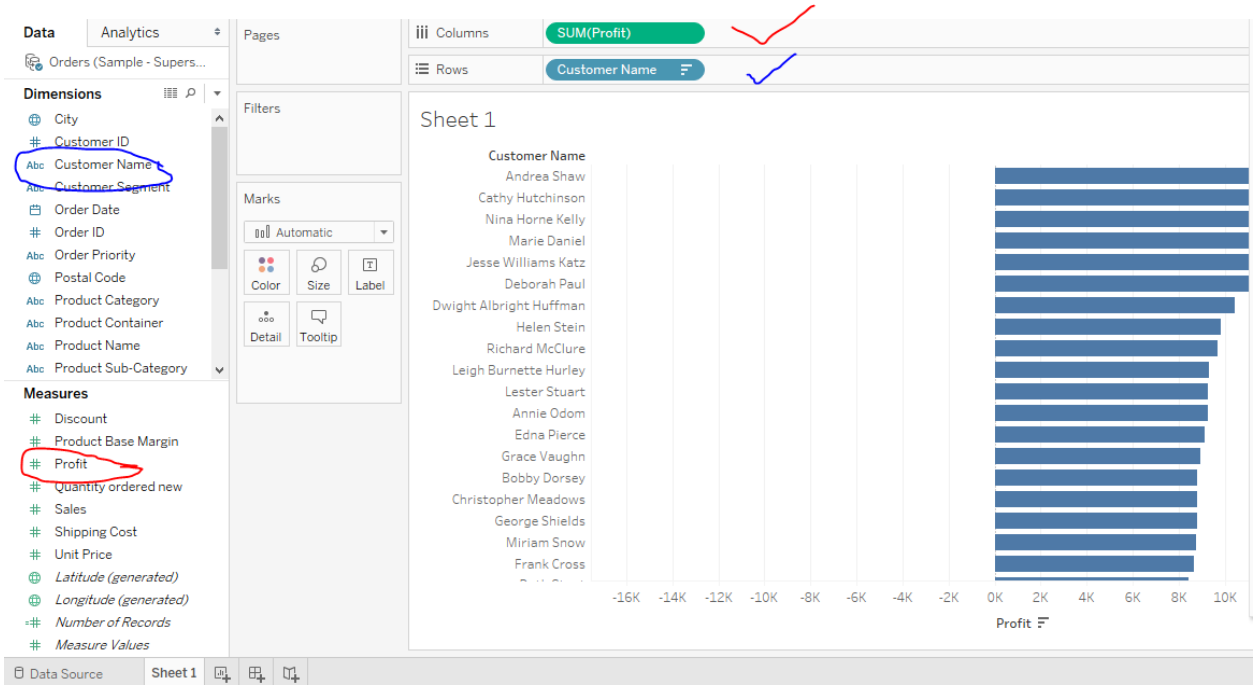
Step 2: Drag Orders sheet to sheet area



Step 3: Go to sheet 1 and add Customer name as rows and profit as column



Step 4: Sort the data by clicking on Profit label on bottom

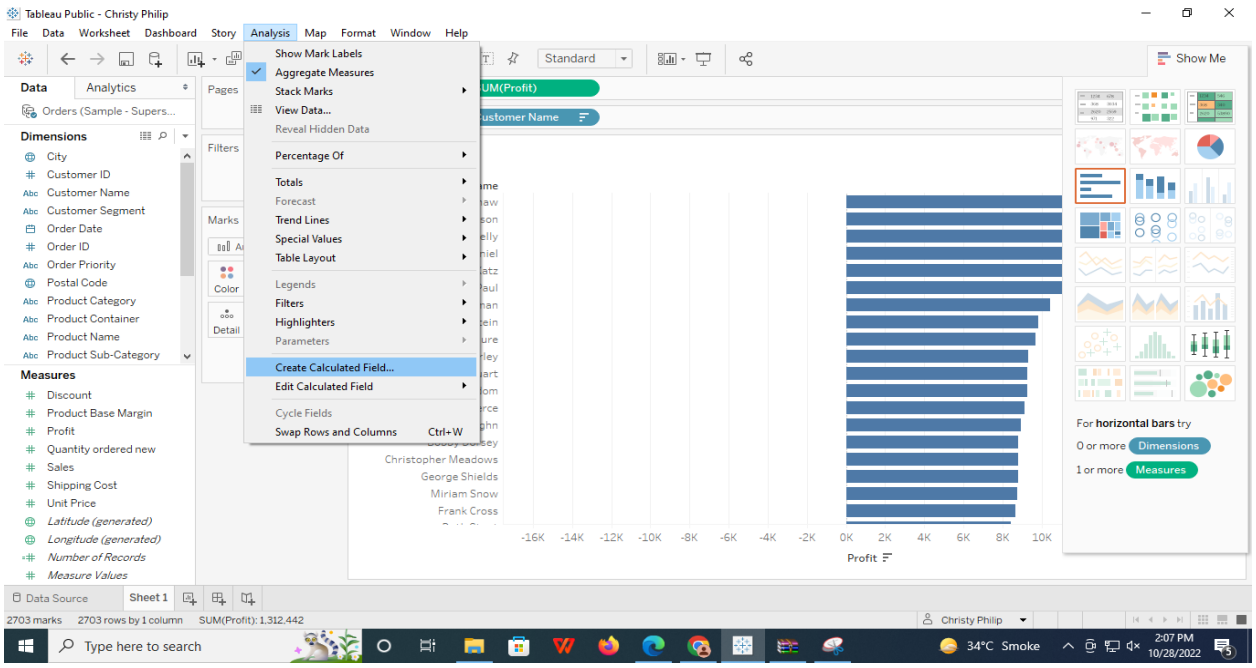


Step 5: To calculate profit Ratio

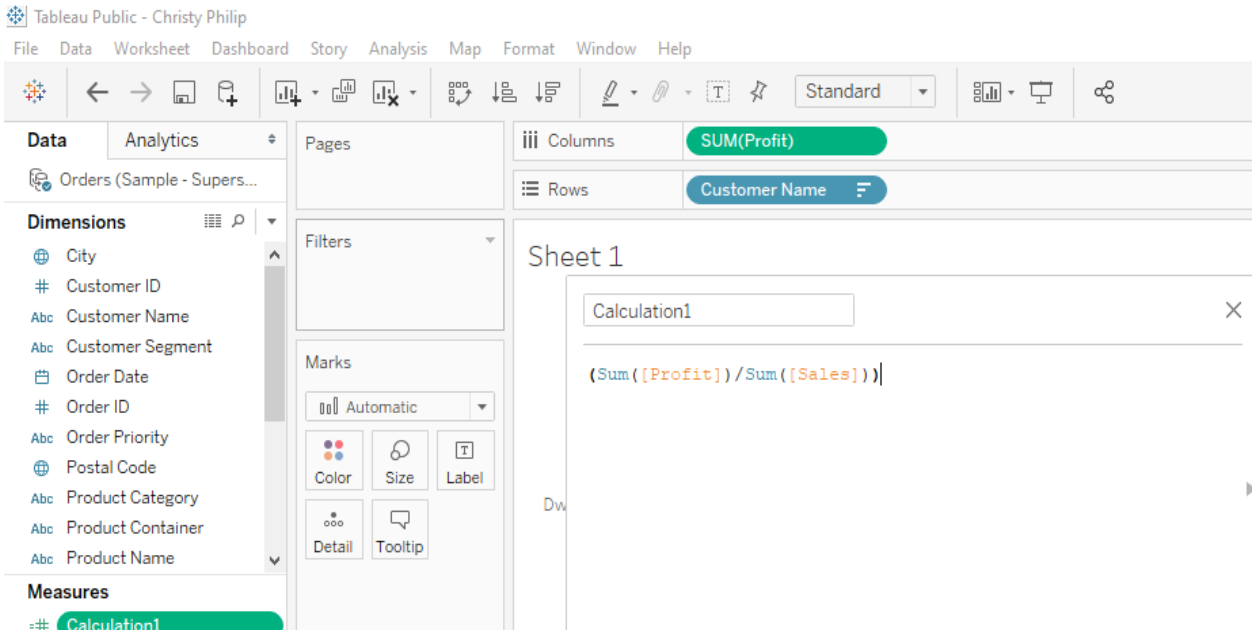
Profit Ratio= (Sum([Profit])/Sum([Sales]))

This formula needs to be entered as tooltip or label

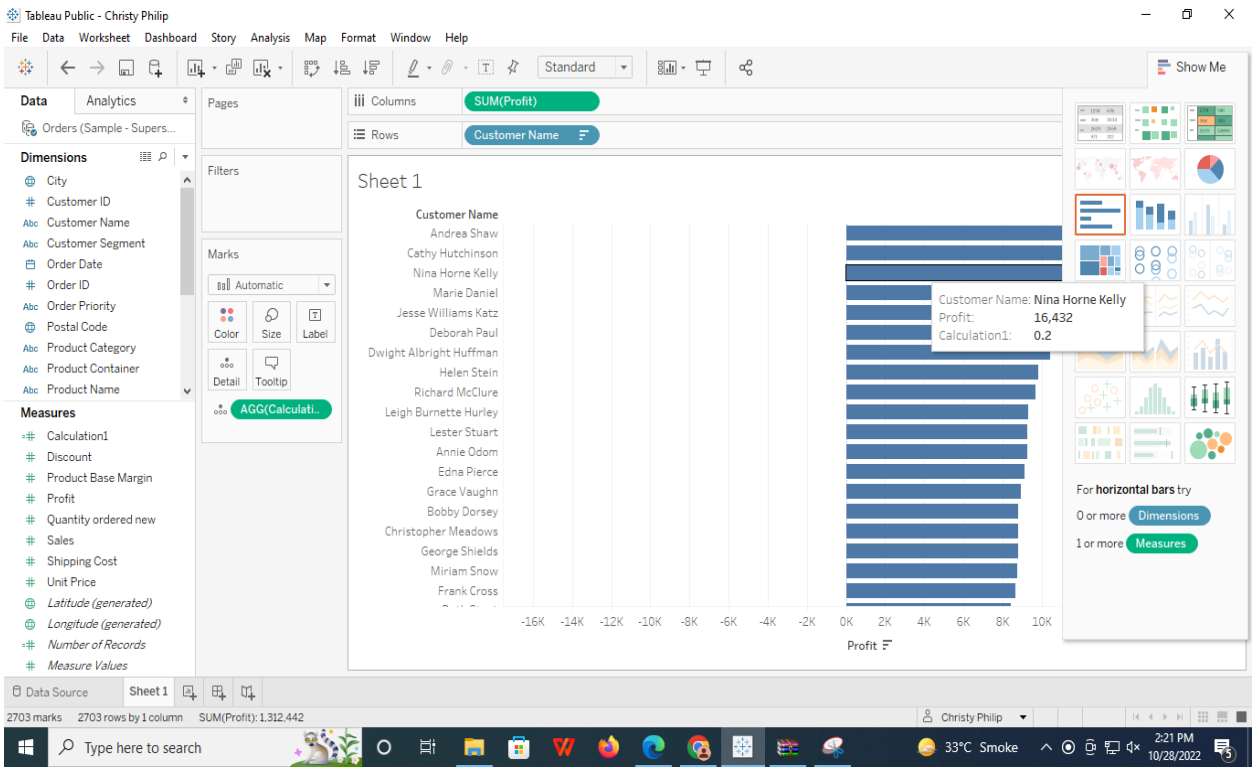
Click on Analysis>Create Calculated Field and enter the formula



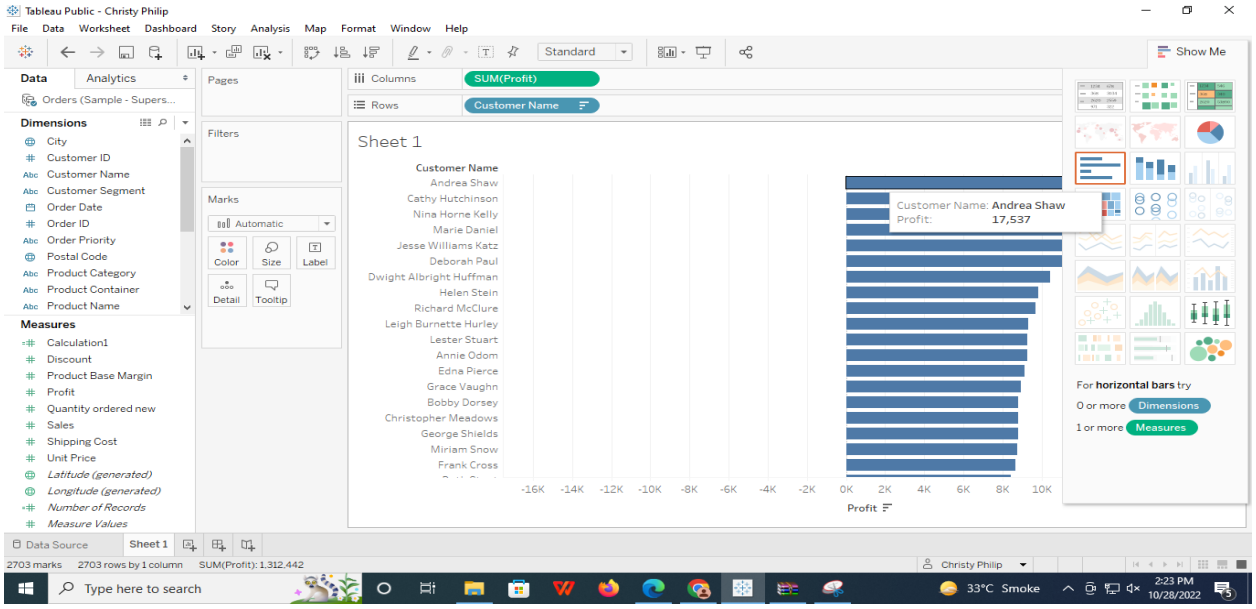
Step 6: In Create Calculated → Field Enter the below formula



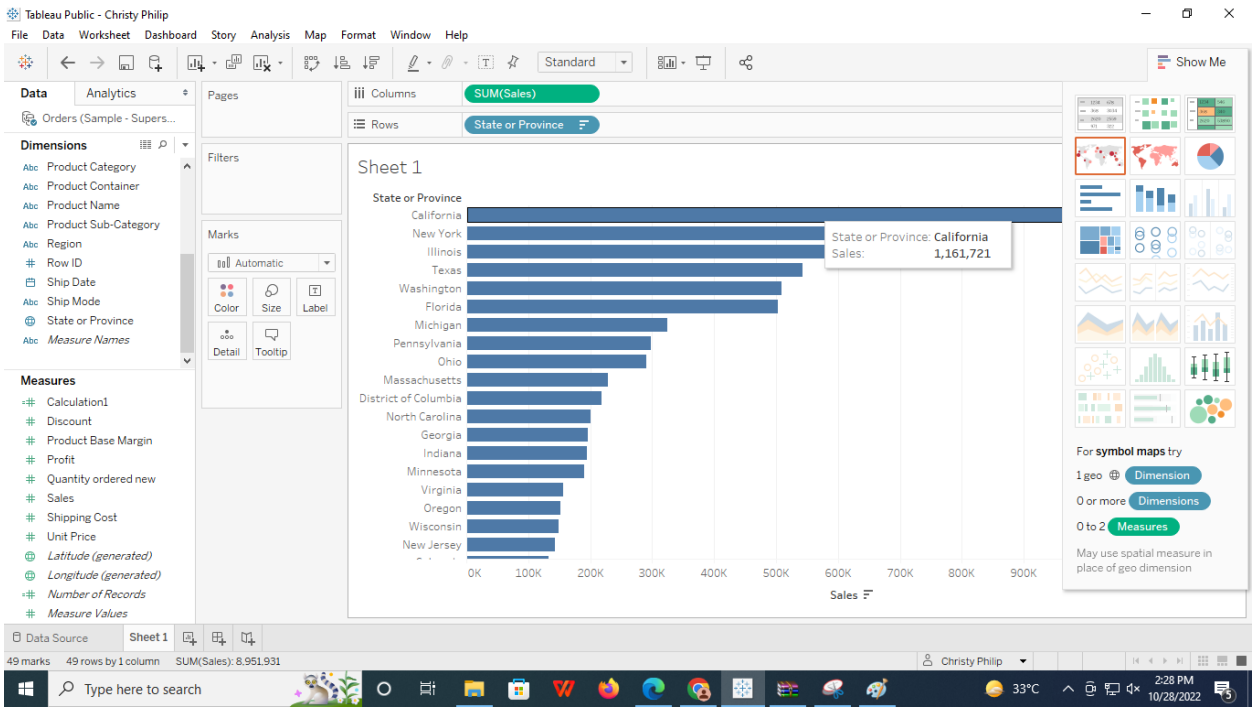
Step 7: You can see Calculation1 in measures. Drag is to the Marks area.



Step 8: As you hover on graph a Calculation parameter is shown

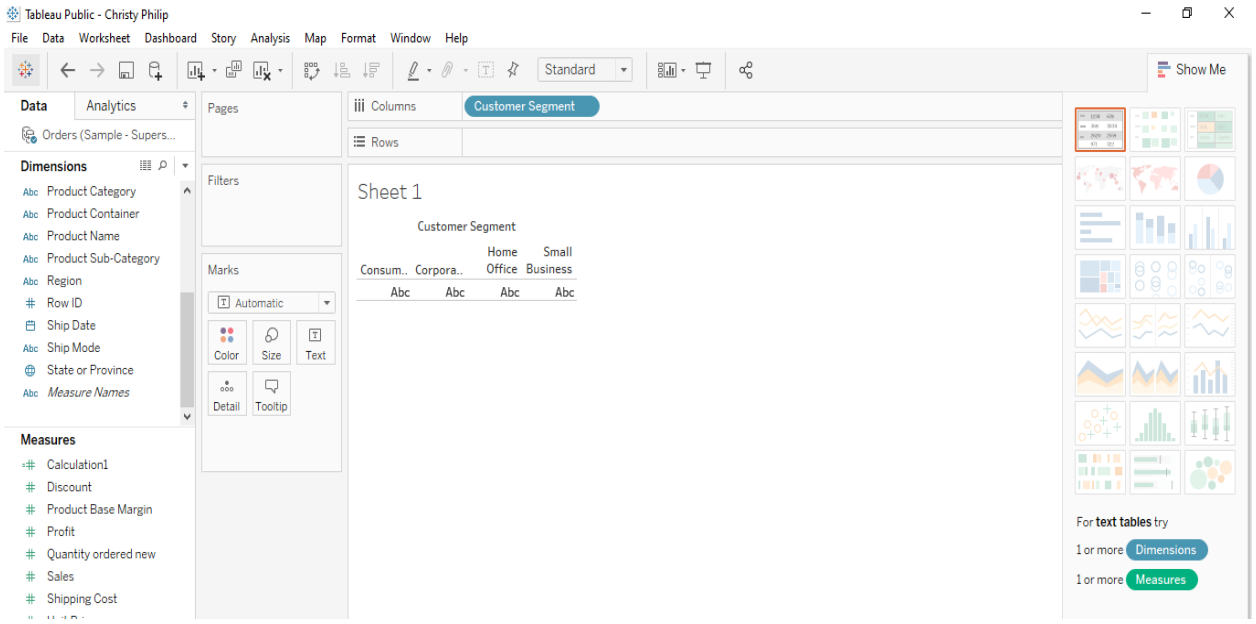


Q. 2) Which state has the highest Sales (Sum)? What is the total Sales for that state?
Step 9: Now drag and drop Sales in Columns from Measures and State or province in Rows from Dimensions



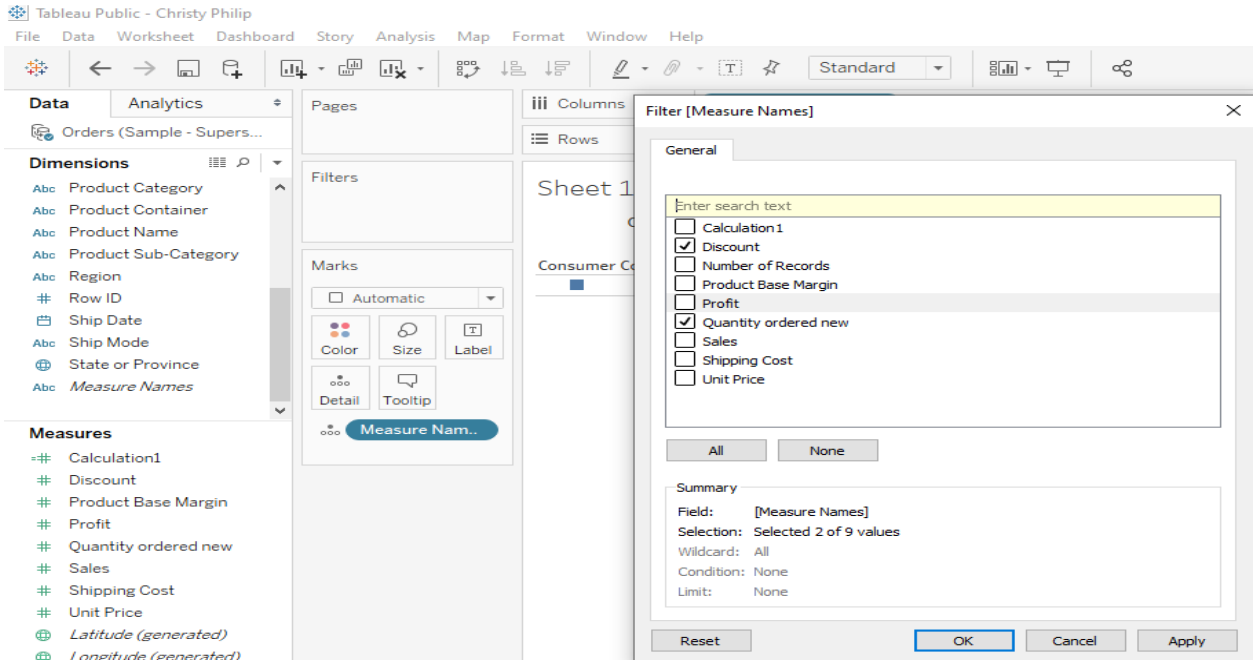
Q.3 Which customer segment has both the highest order quantity and average discount rate? What is the order quantity and average discount rate for that state?

Step 10: Drag and drop Customer Segment from Dimensions in Columns

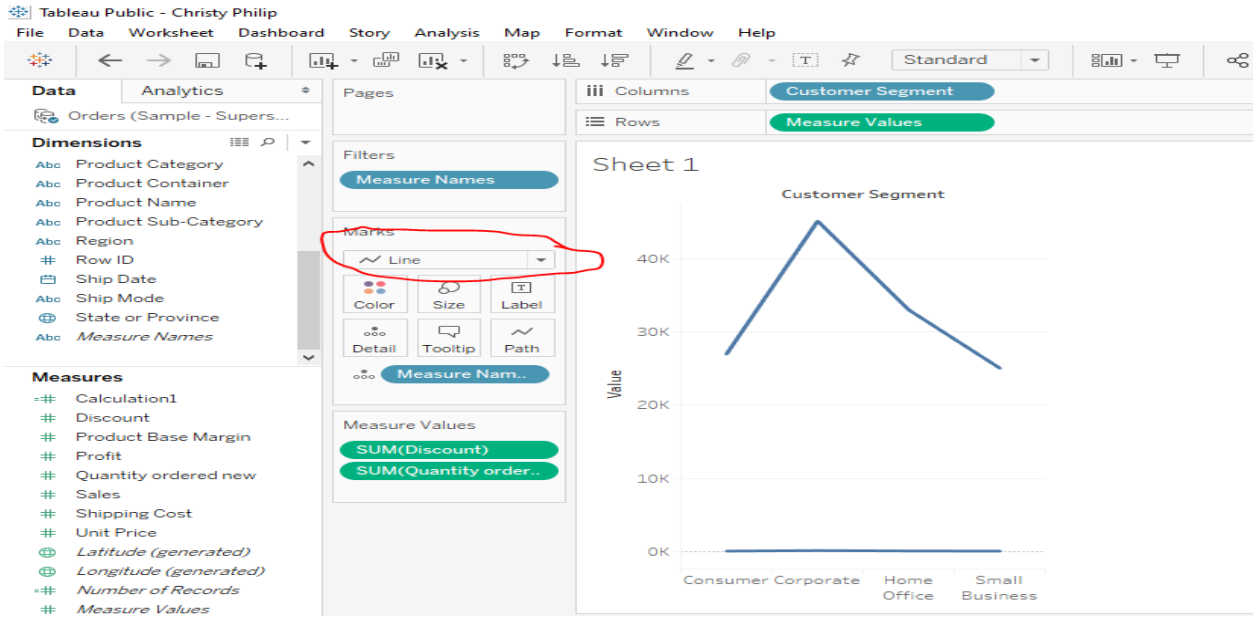


Step 11: Drag and drop measure names into Filter

Step 12: Double click on the dragged Measure Names from Dimensions select Discount and Quantity ordered new



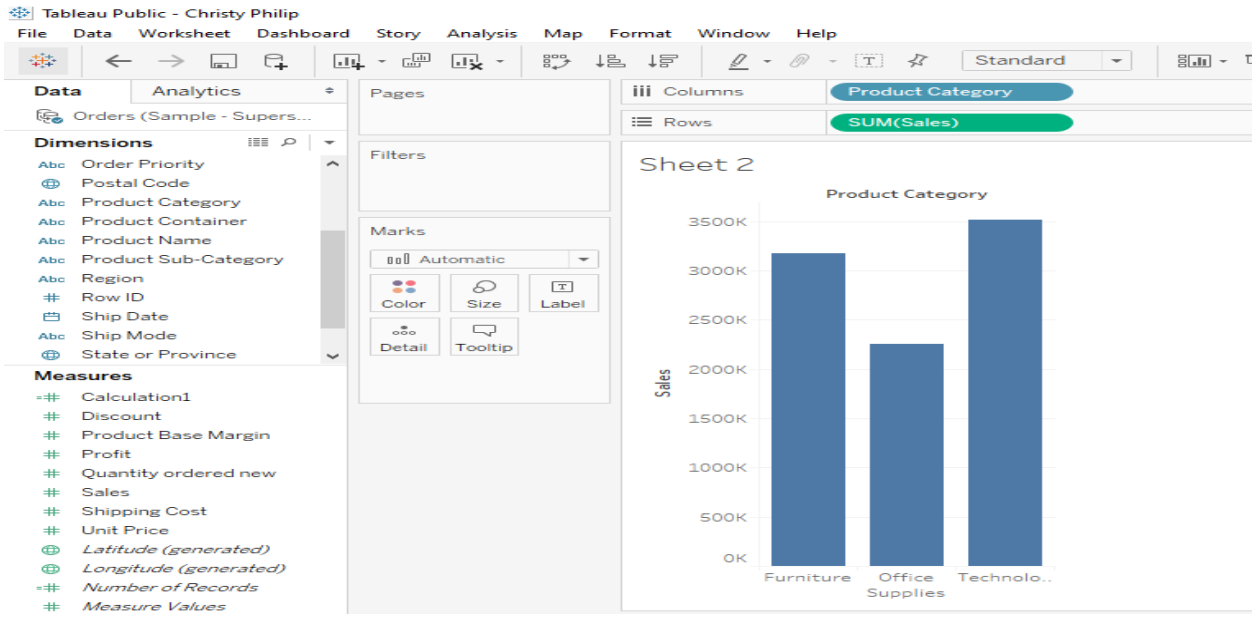
Step 13: In circled area choose Line and in Rows drag and drop Measure Values from Measures in Rows



Q 4: Which Product Category has the highest total Sales? Which Product Category has the worst Profit? Name the Product Category and $ amount for each.

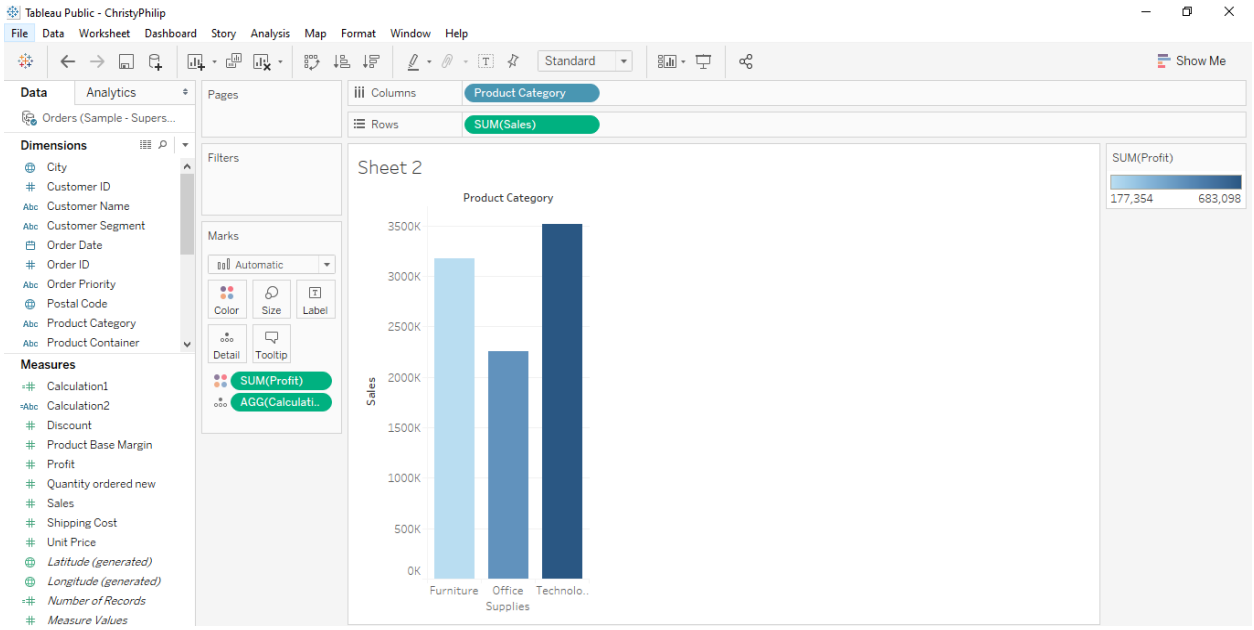
a. Bar Chart displaying total Sales for each Product Category

Step 14: Drag and Drop Product Category in Columns from Dimensions and Sales from Measures in Rows



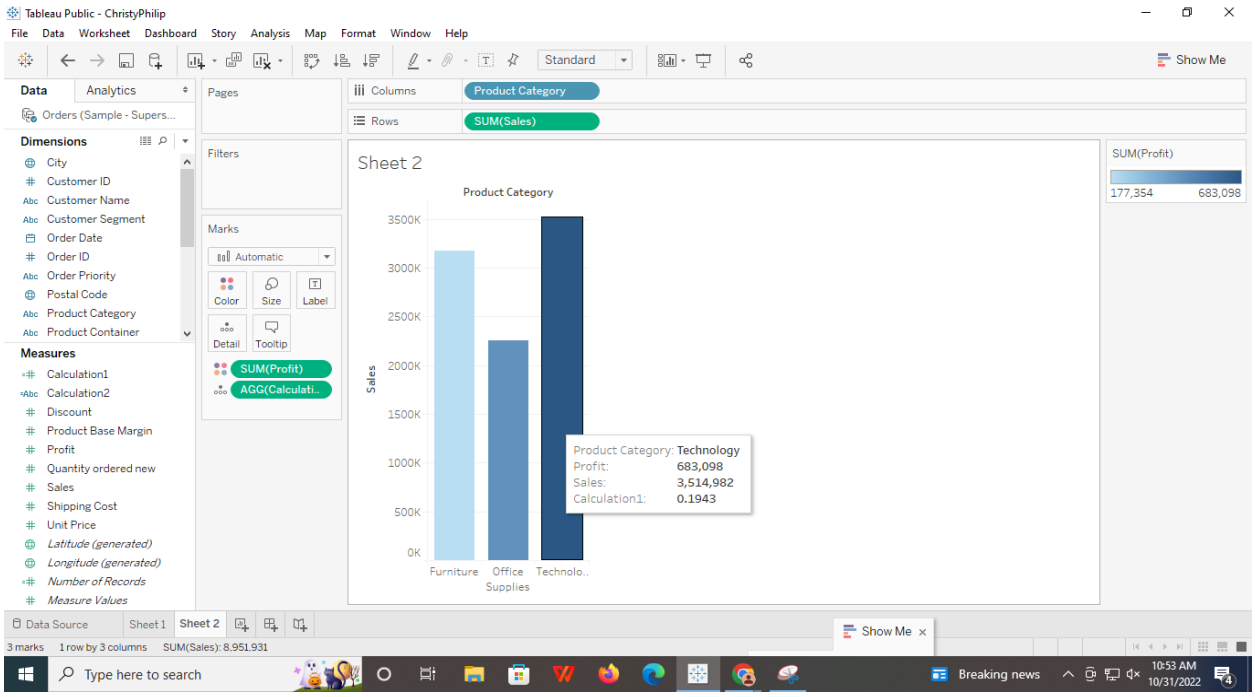
b. Add a color scale indicating Profit

Step 15: Drag Sum(Profit) and AGG(Calculation) in marks and set SUM(PROFIT) color by clicking SUM(Profit)



c. Each Product Category labeled with total Sales and Each Product Category labeled with Profit

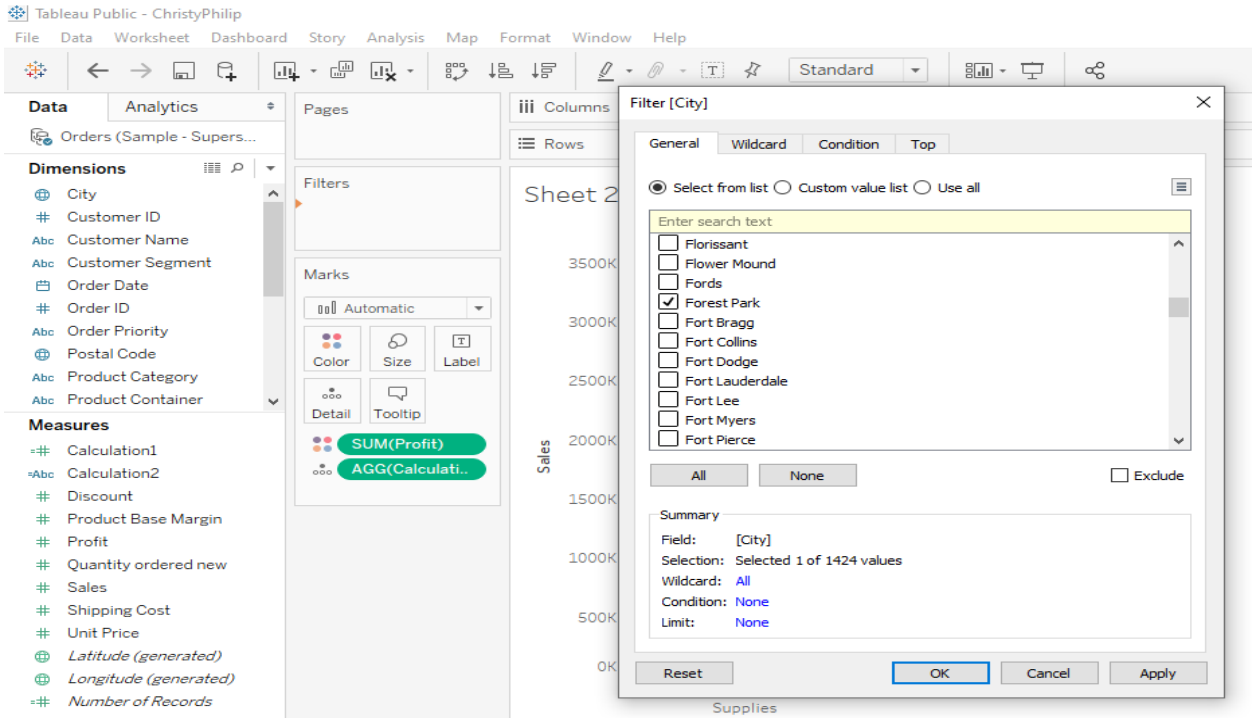
Step 16: Each Product Category labeled with total Sales and Each Product Category labeled with Profit



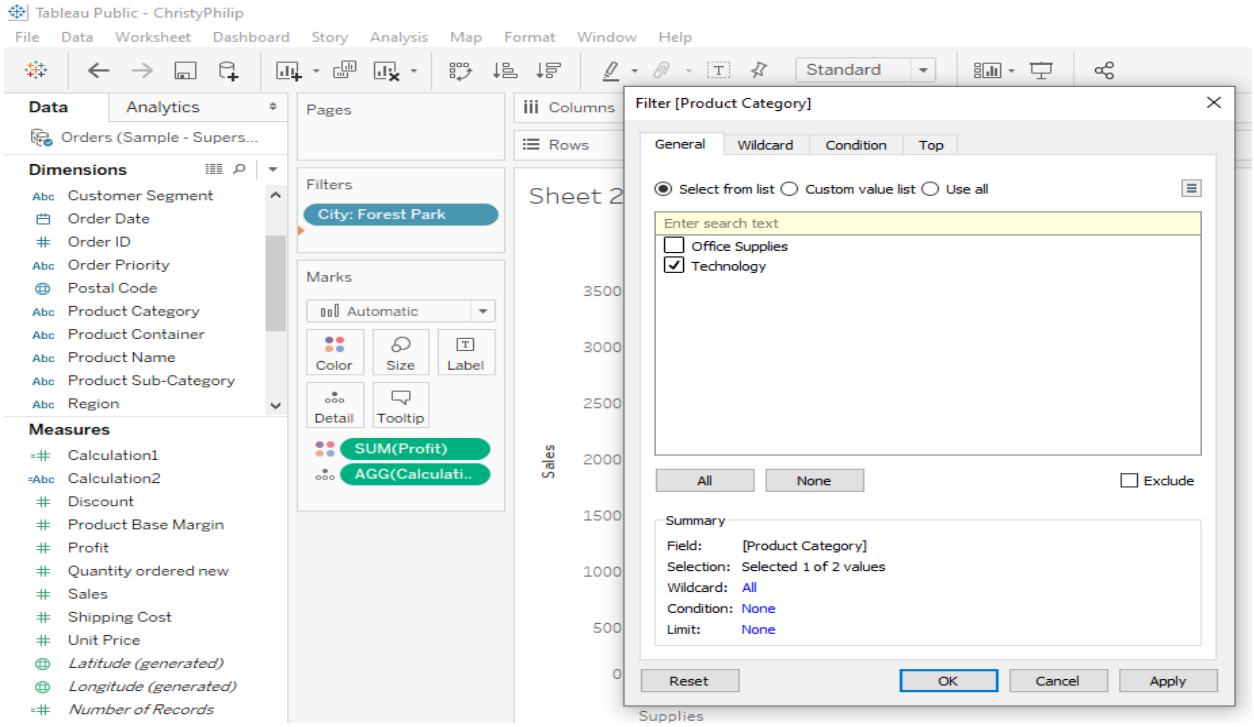
Q 5: Use the same visualization created for Question #4.What was the Profit on Technology (Product Category) in Boca Raton (City) ?

Step 17: Drag and drop City in Filter

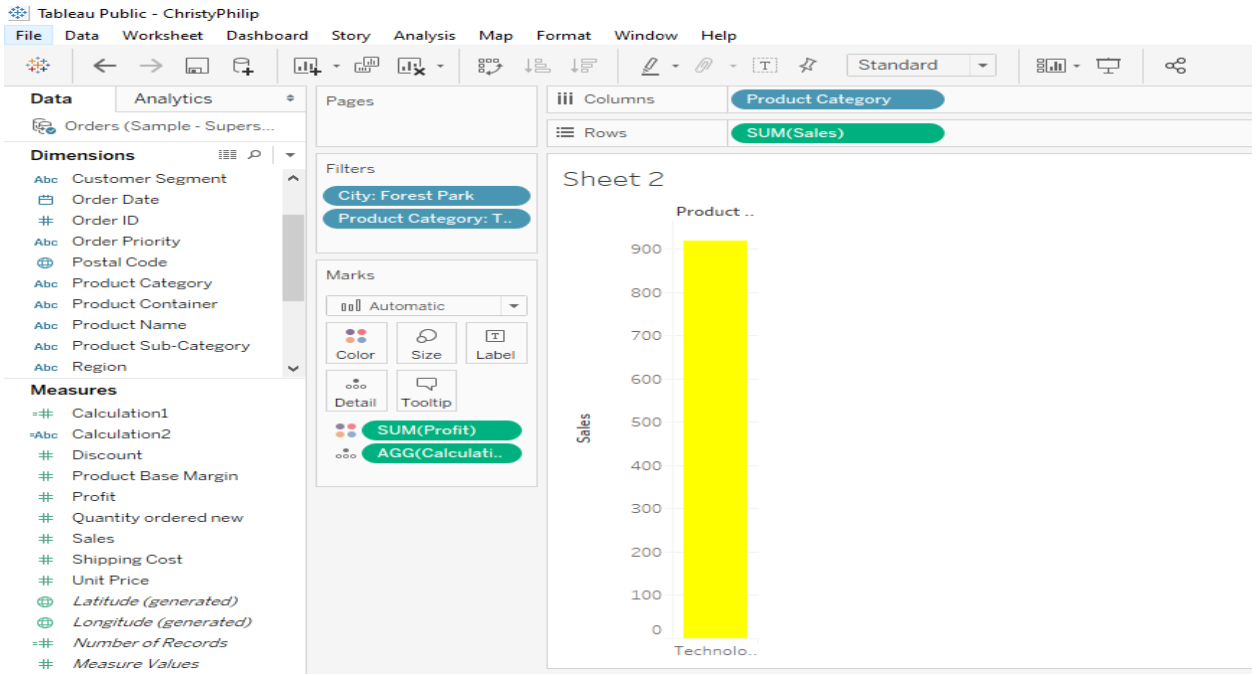
Select the city to show a single bar for the city



Step 18: Drag and drop Product Category in Filter and choose Technology



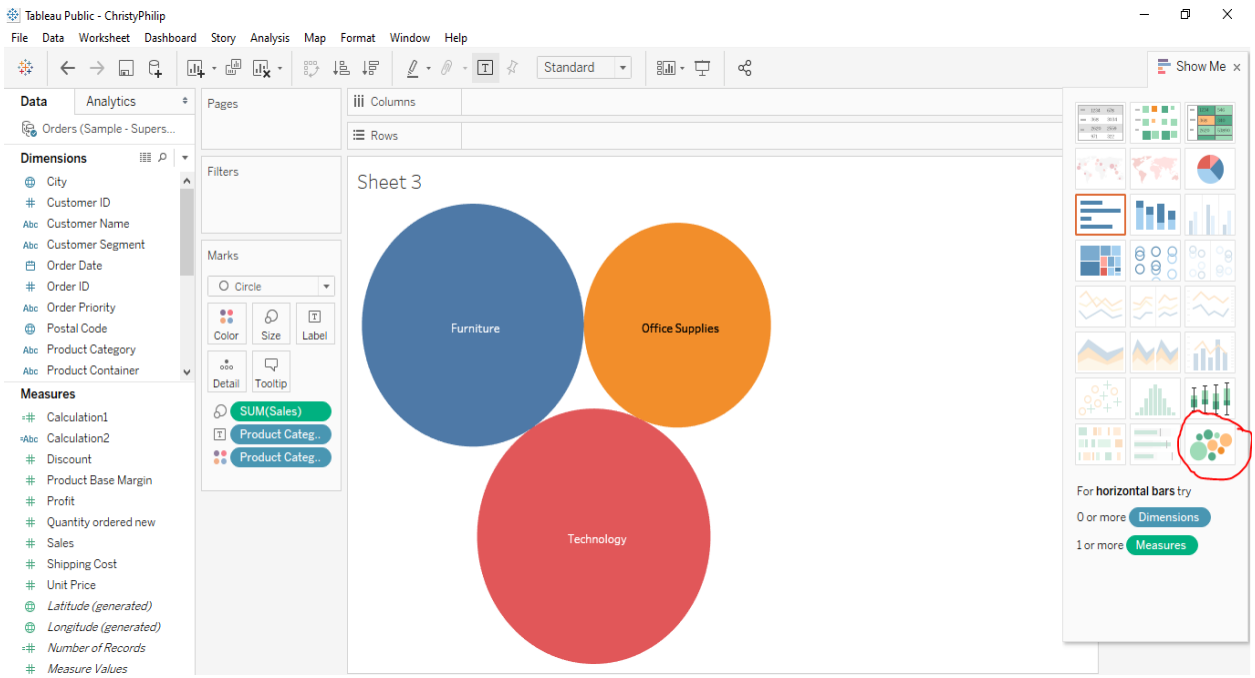
Step 19: The SUM(Profit) in marks set a color for it.



Q 6: Which Product Department has the highest Shipping Costs? Name the Department and cost.

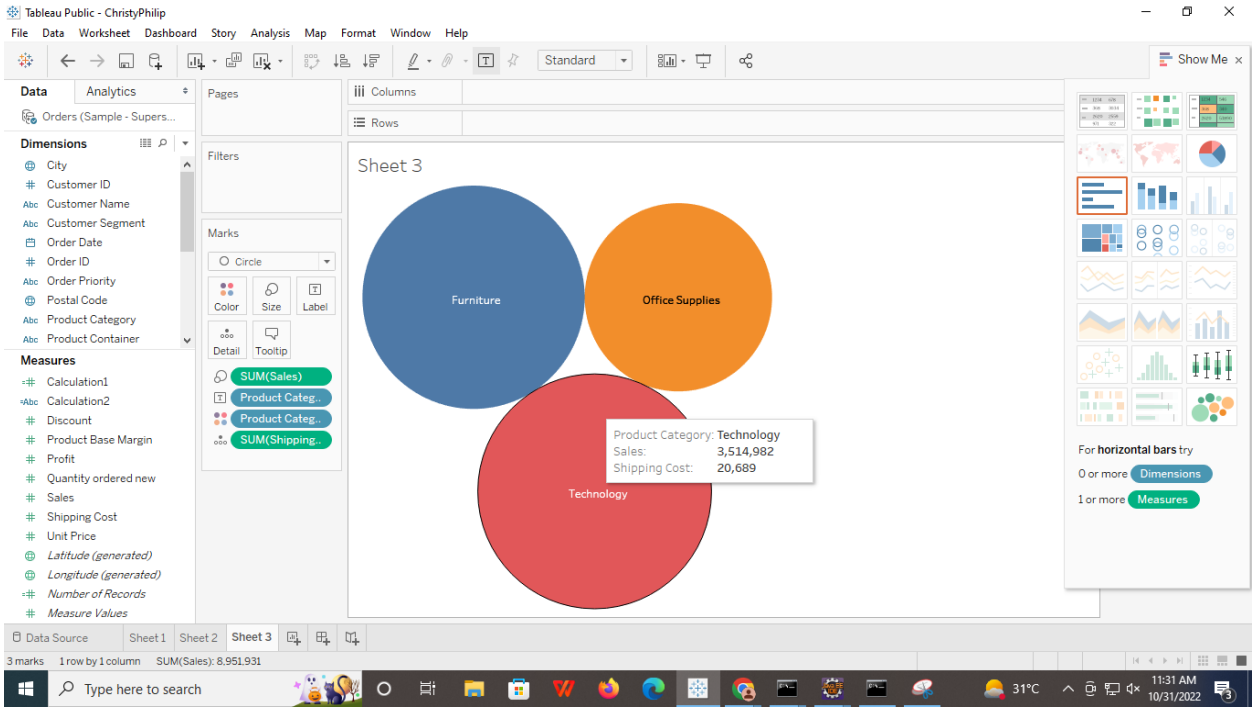
a. Packed bubble chart showing each Product Department as a colored bubble

Step 20: In the Marks section Sales, Product Category and once again Product Category should be dragged from Dimensions and Measures and then in show me the circled one should be selected



b. Use Shipping Cost to display the size of each bubble

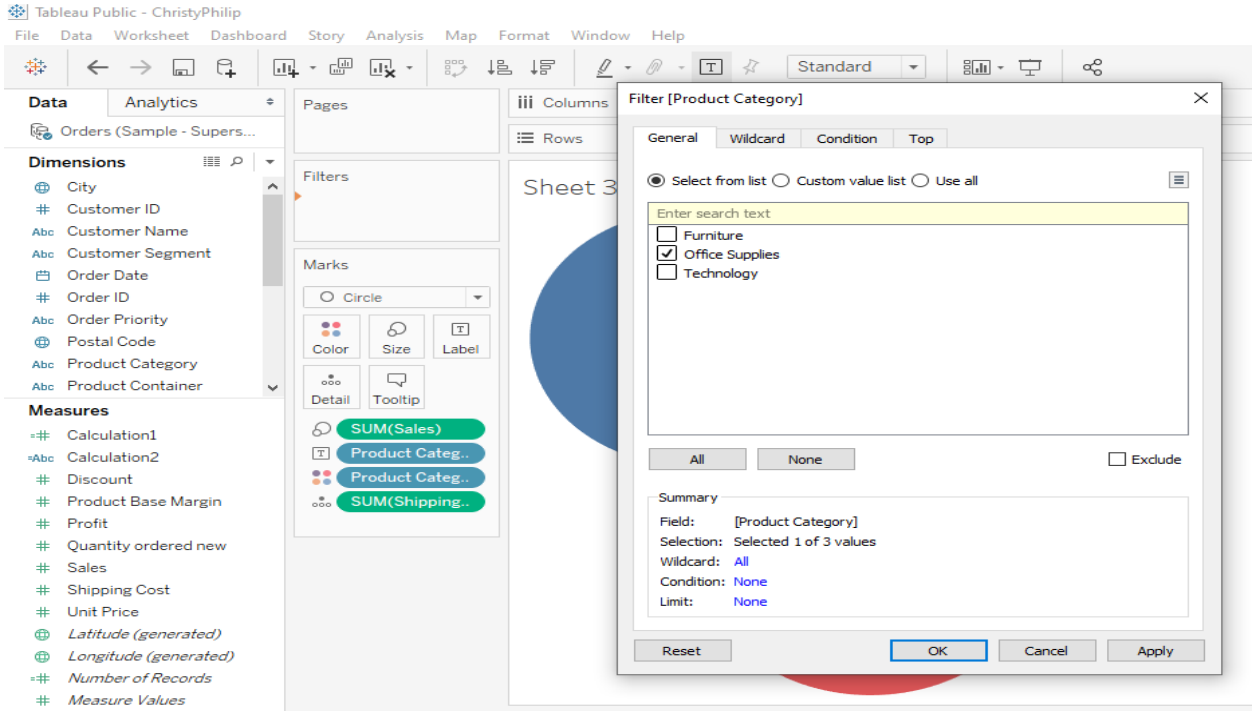
Step 21: Drag and drop shipping Cost in marks on hovering the graph details would be shown

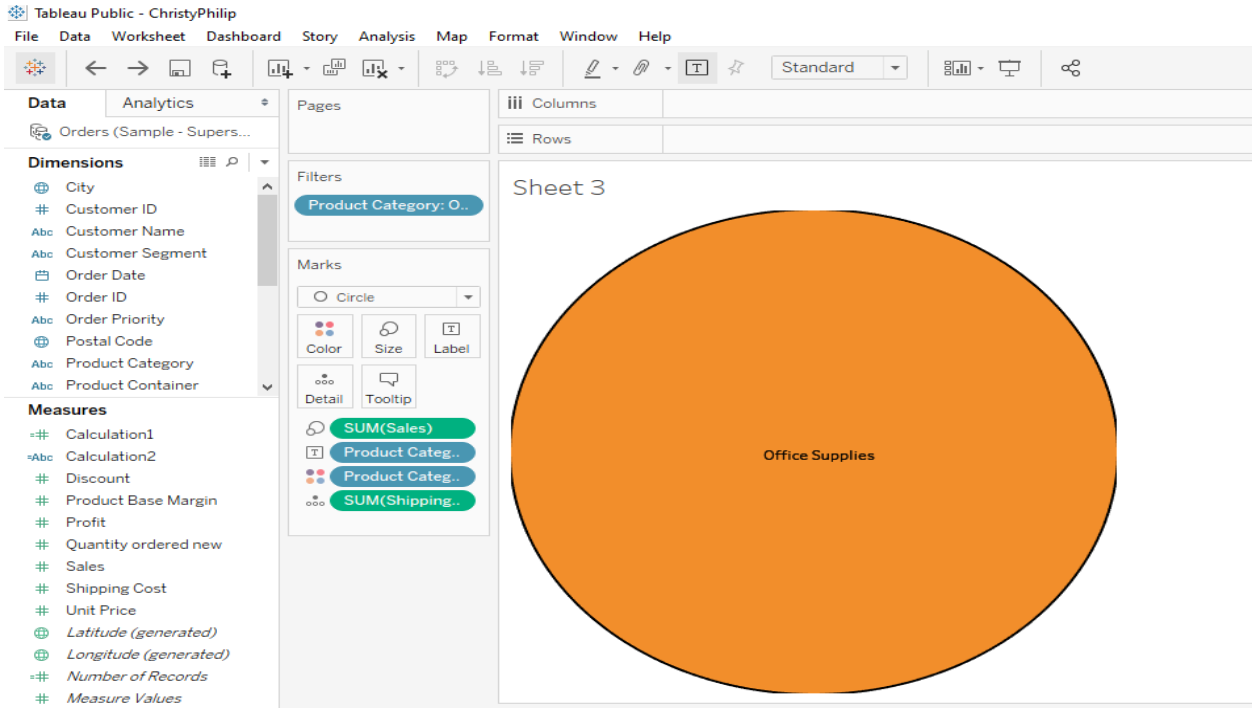


Hovered Result:

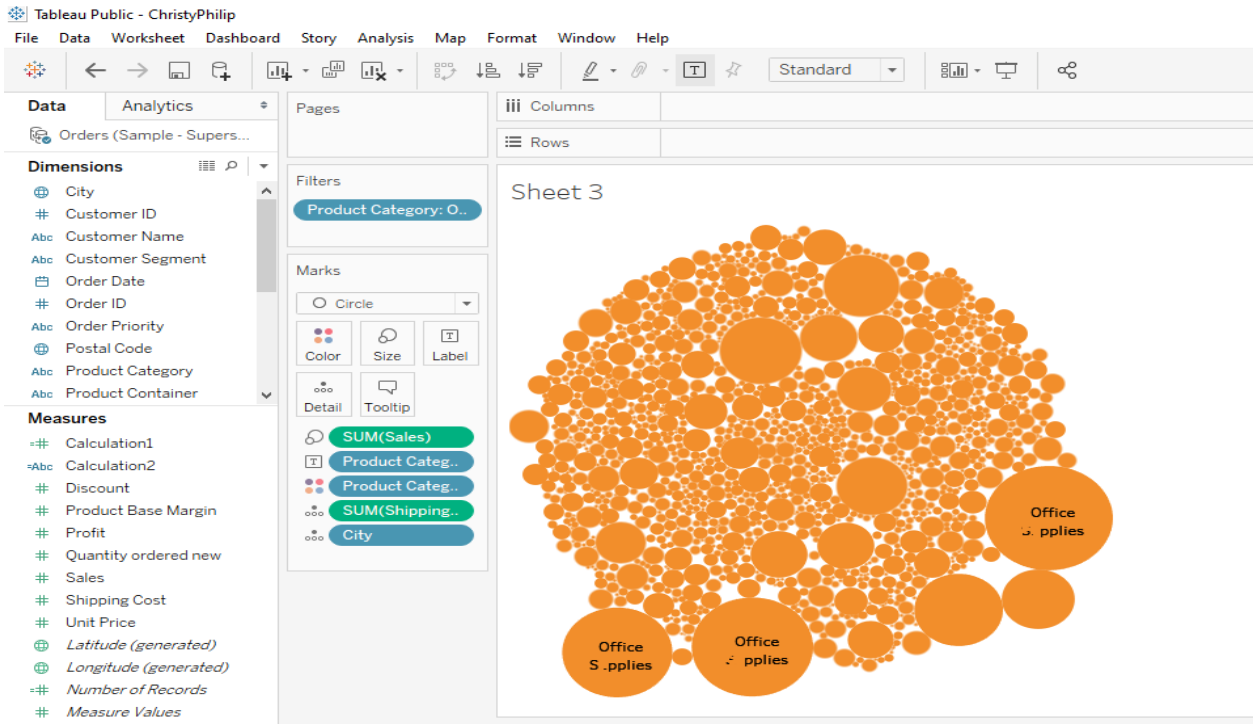
Q 7: Use the same visualization created for Question #6. What was the shipping cost of Office Supplies for Xerox 1905 in the Home Customer Segment in Cambridge? Ans: Apply filters as per requirement

Step 22: In Filters section drag and drop Product Category and select Office Supplies

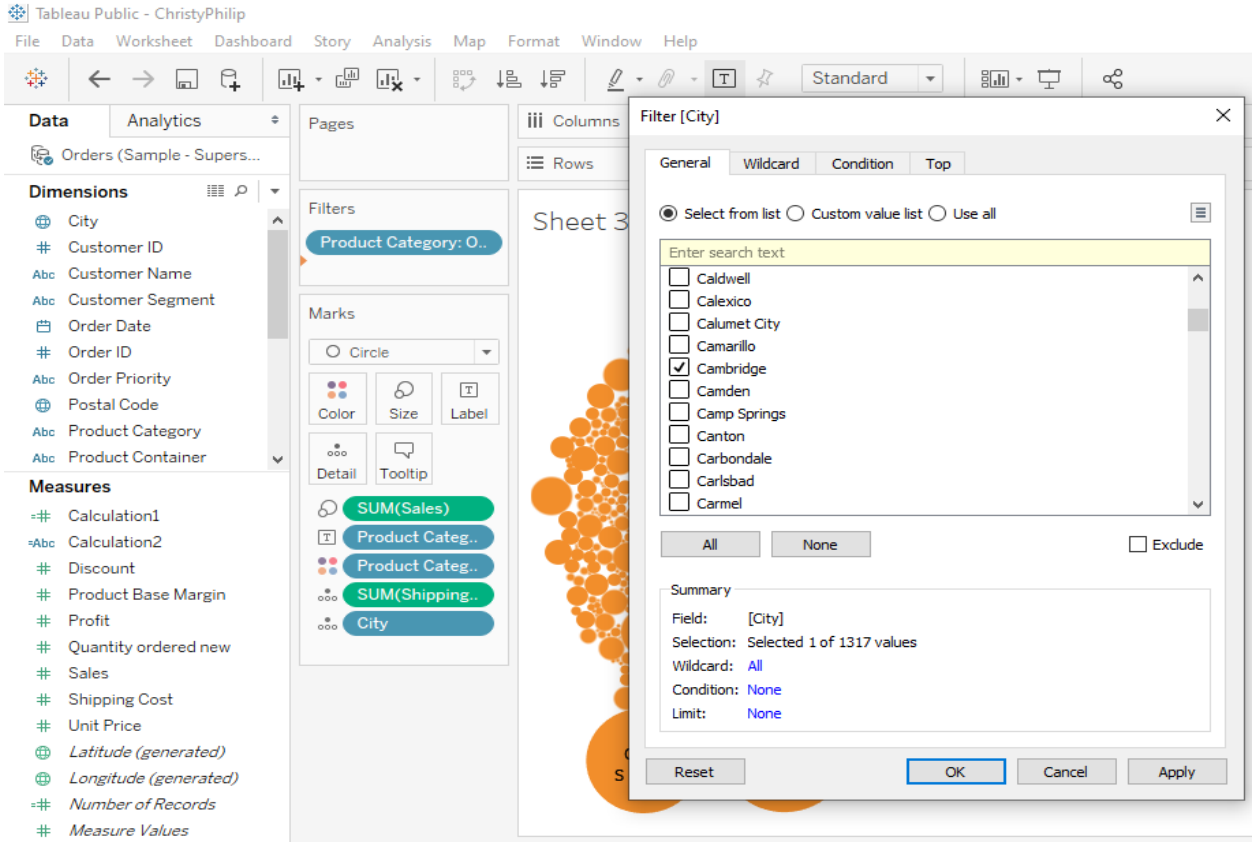




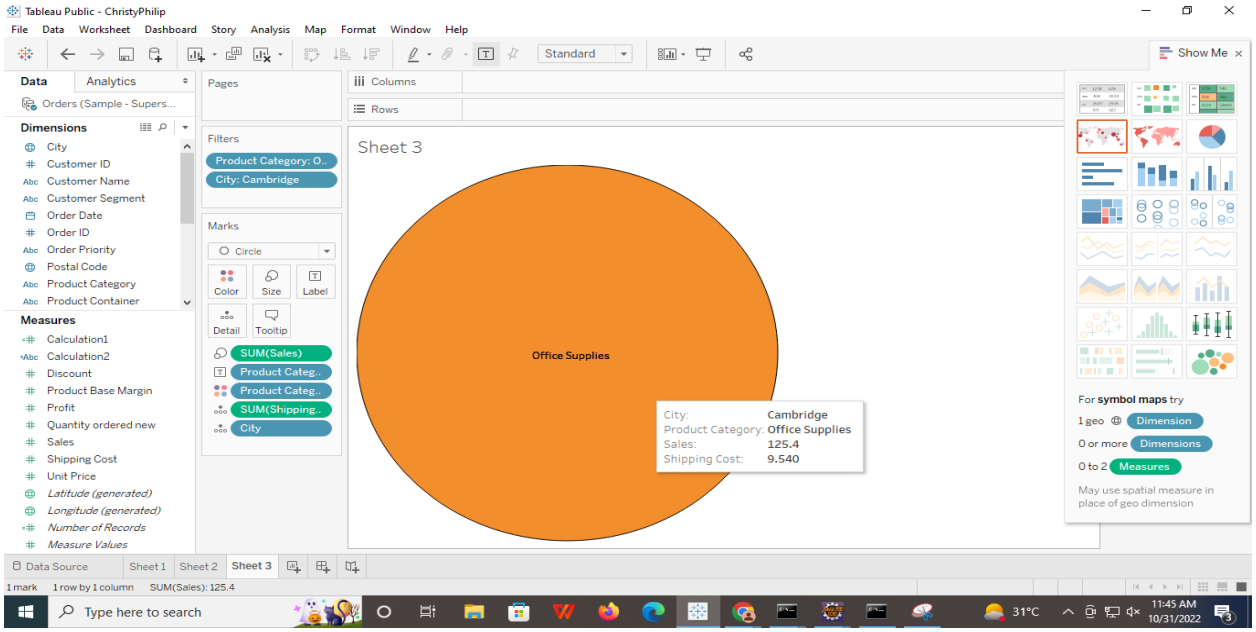
Step 23: Drag and drop city in Marks section



Step 24: Drag and drop City in Filters and choose Cambridge

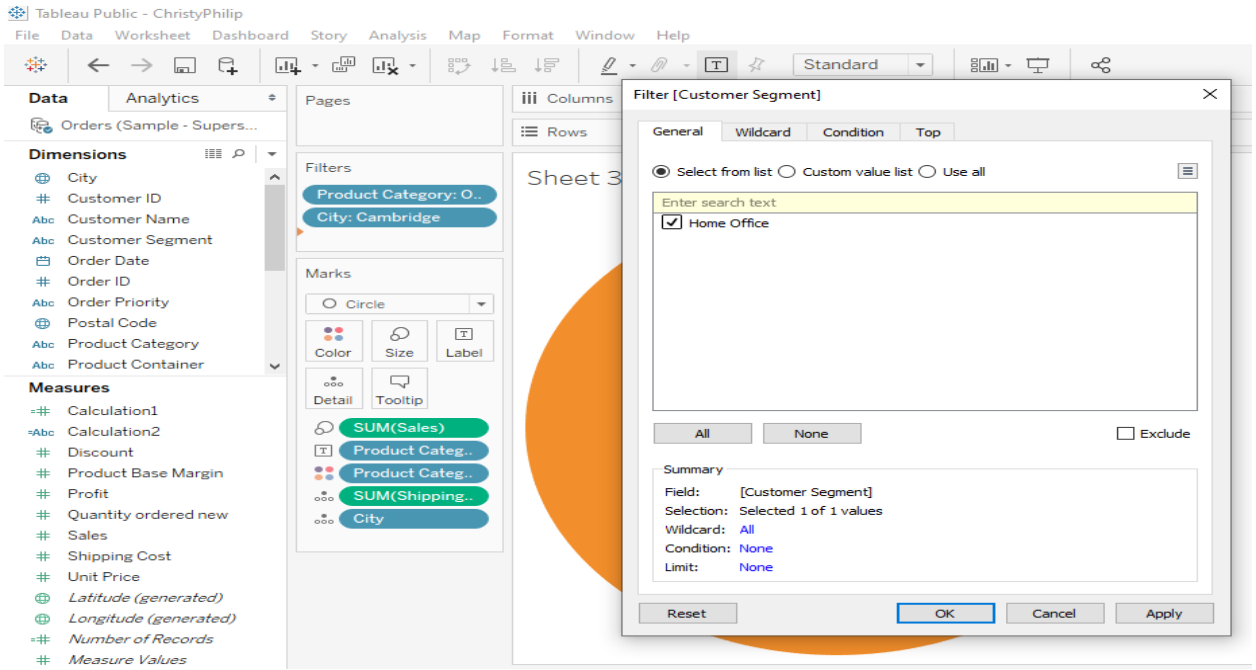


Step 25: Hover on graph to see Result

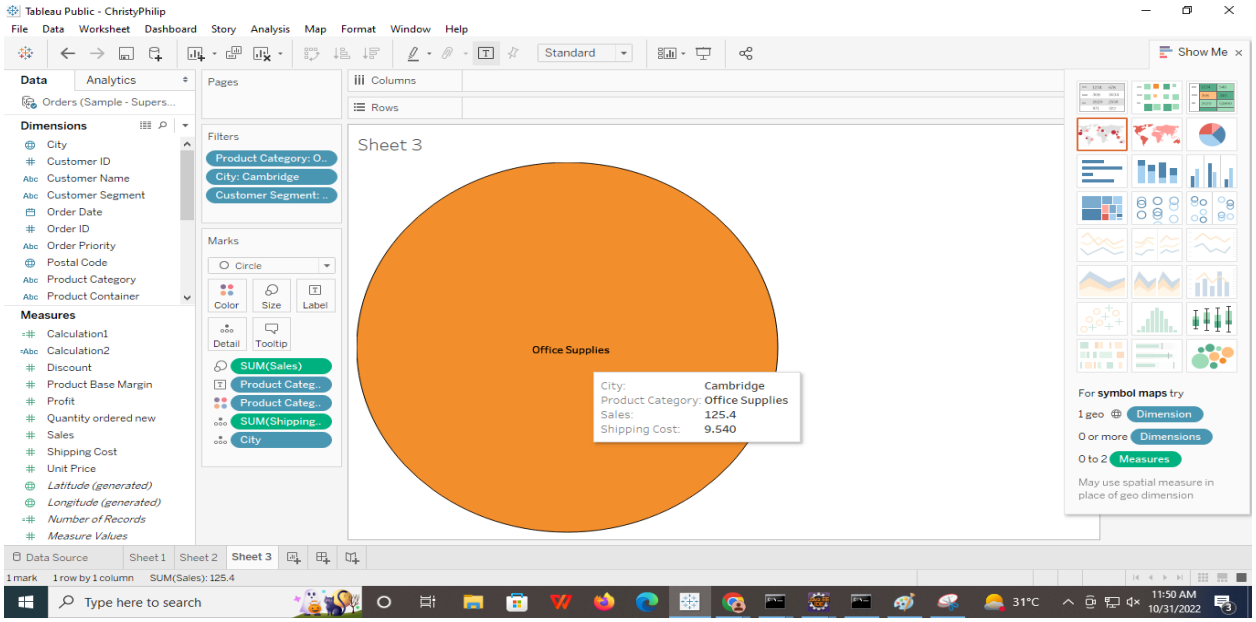


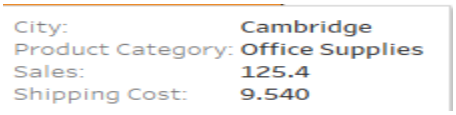
Result:

Step 26: Drag and drop Customer Segment in Filter and choose Home office and hover to see result



Result:



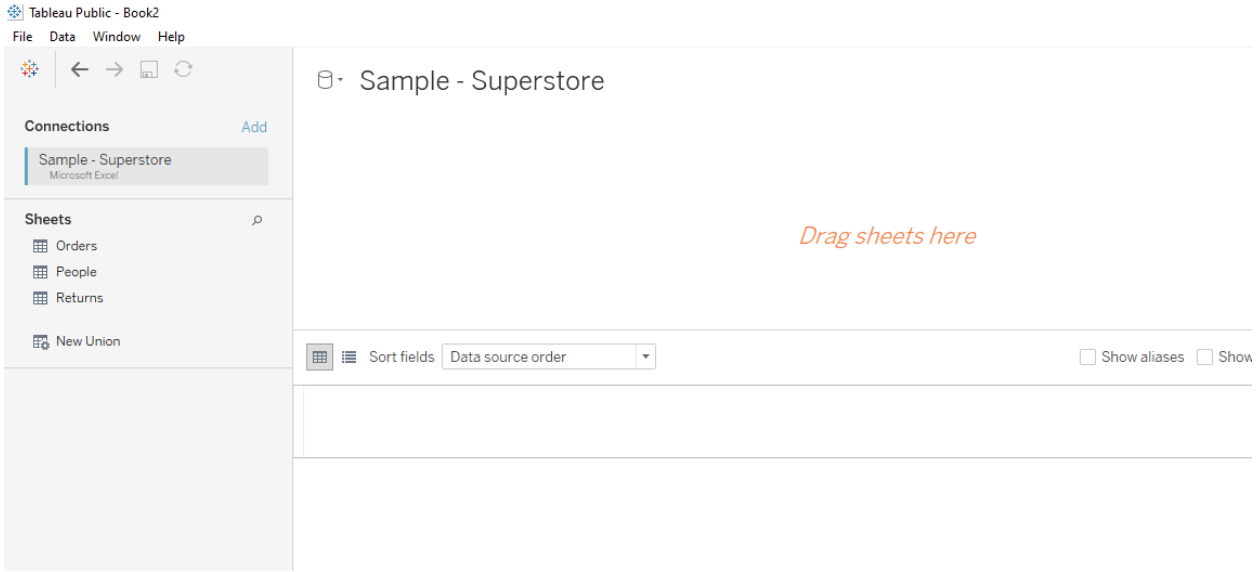


ASSIGNMENT 2

Preparing Maps

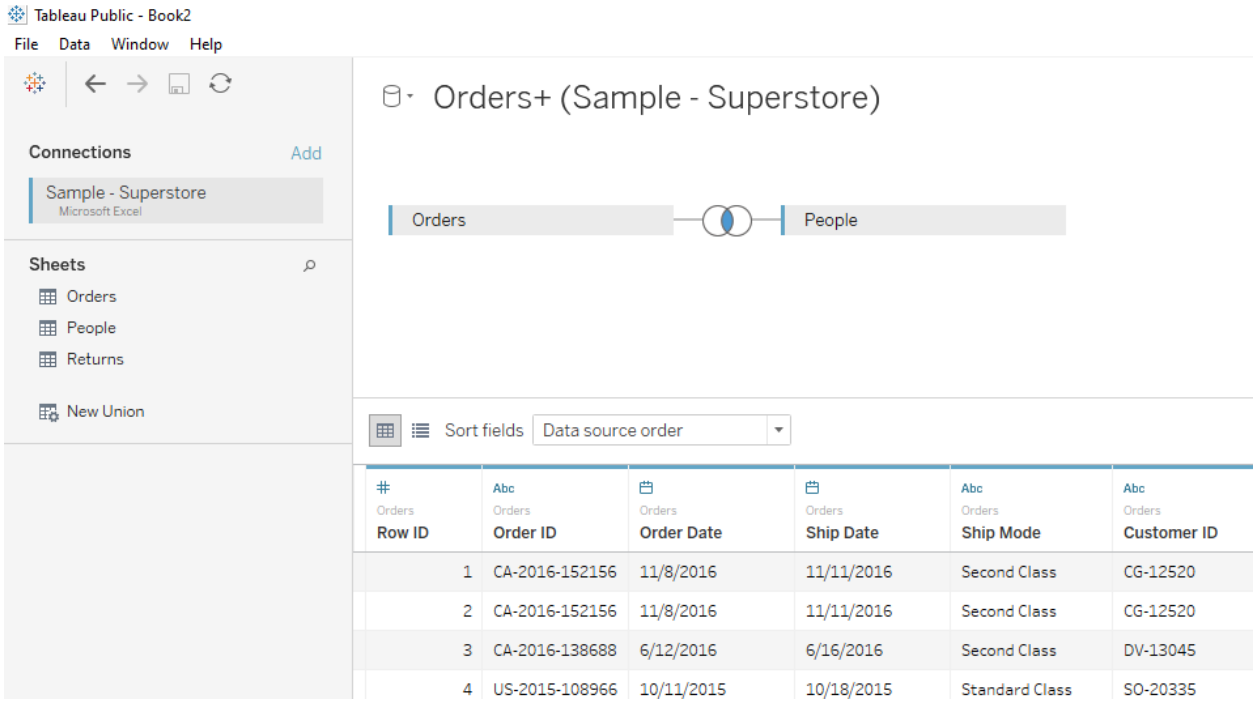
Q 1: Prepare a Geographic map to show sales in each state.

Step 1: Connect to dataset

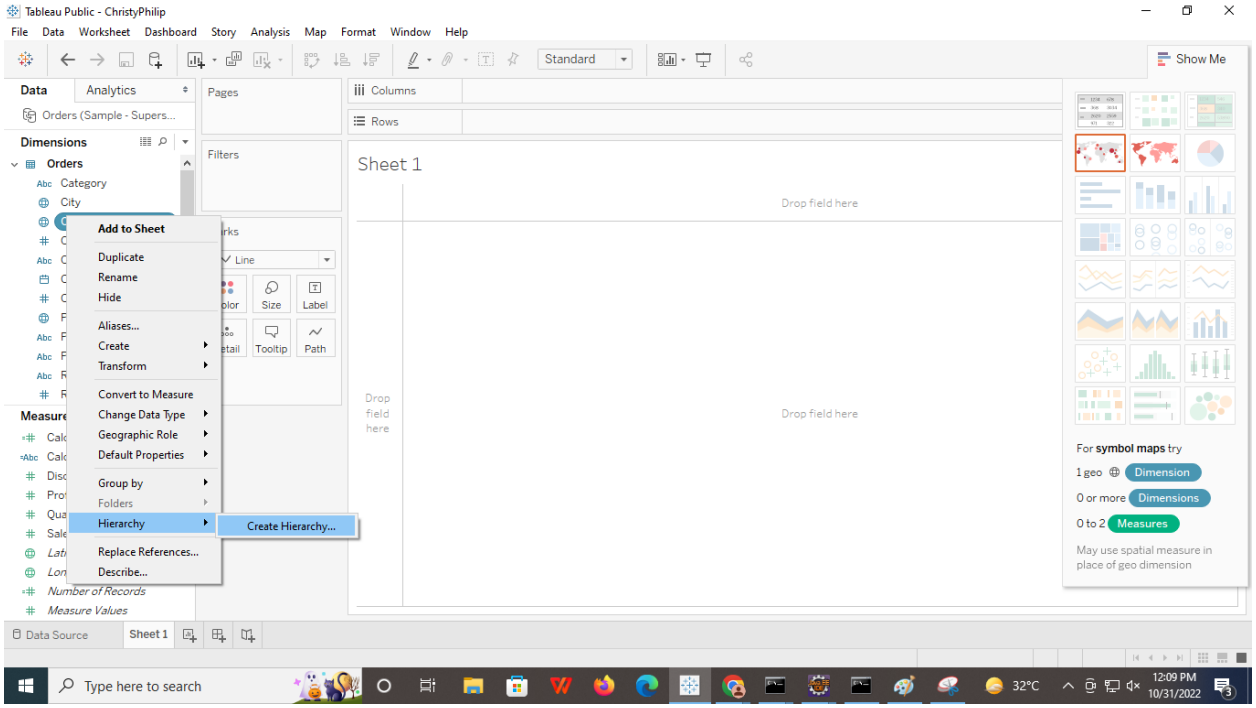


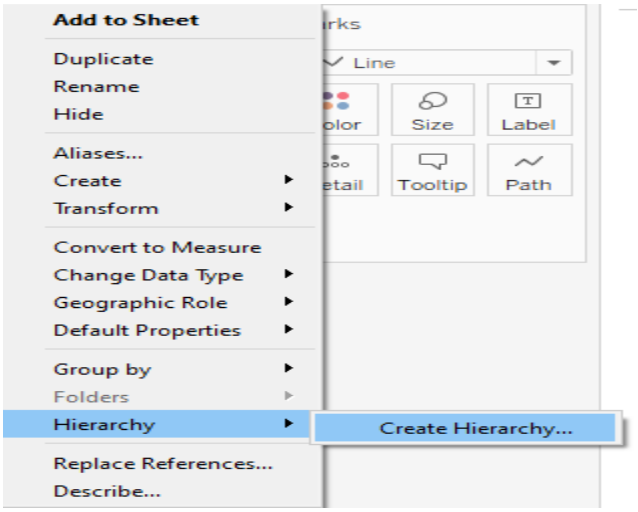
b) Join Sheets

Step 2: Drag Orders then Drag people to perform join



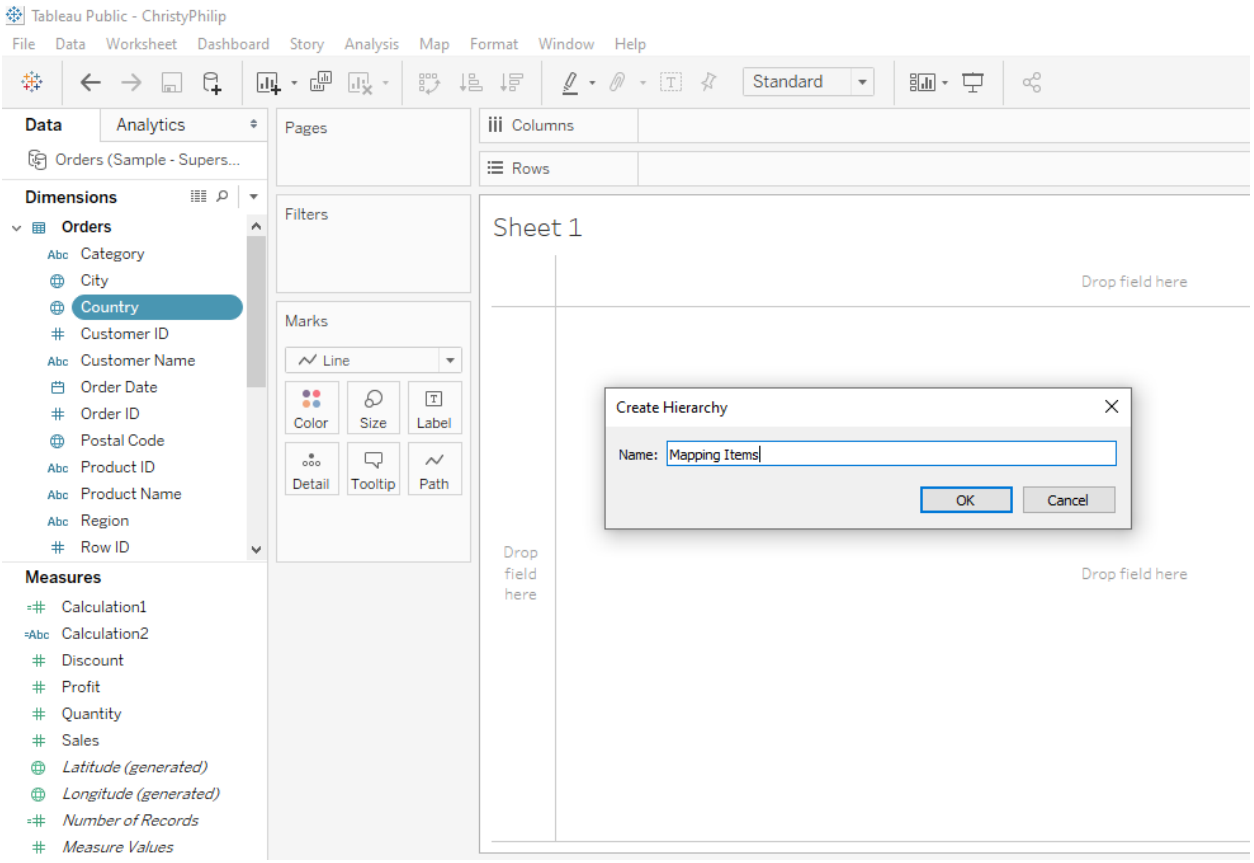
Step 3: Choose Hierarchy by right clicking Country in Dimensions





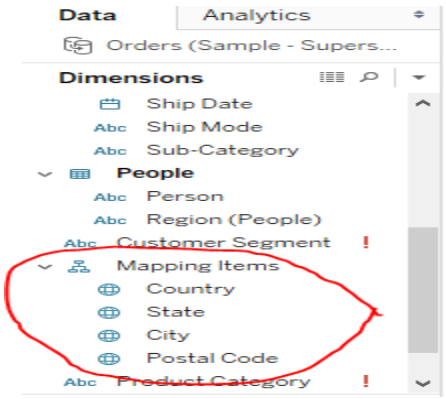
c. Create a Geographic Hierarchy

Step 4: In the Create Hierarchy dialog box that opens, give the hierarchy a name, such as Mapping Items, and then click OK.

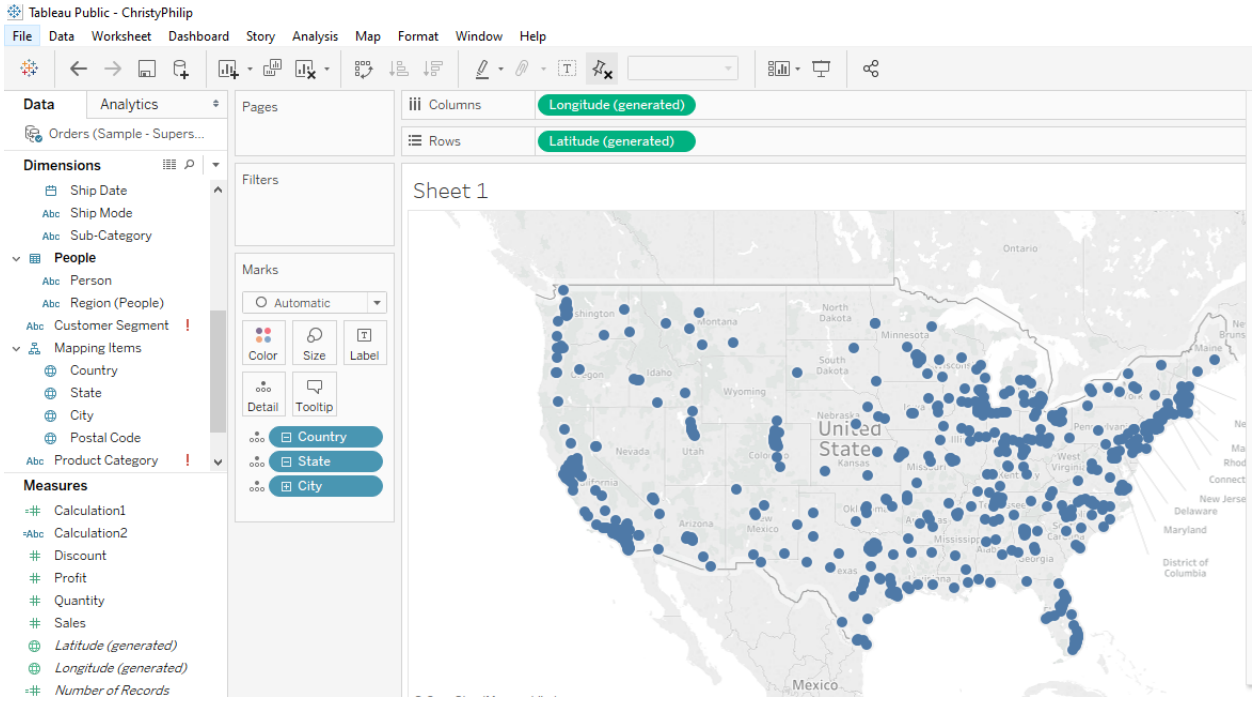


Step 5: In the Data pane, drag the State field to the hierarchy and place it below the Country field.
Step 6: Repeat step 3 for the City and Postal Code fields.

In the below way all items should be there

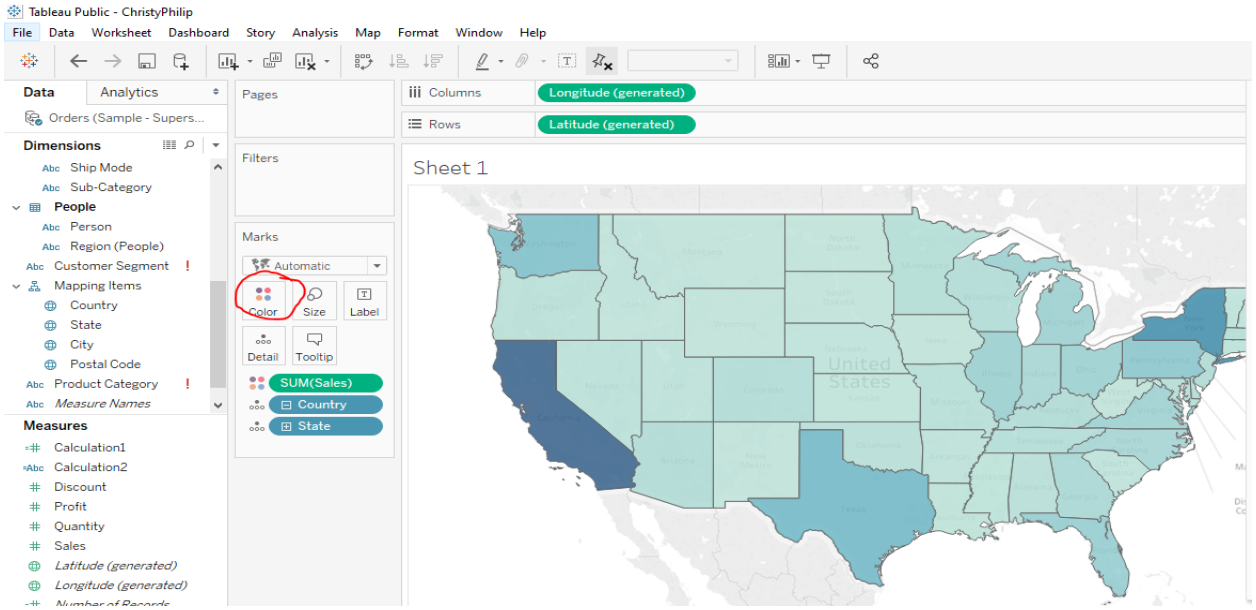


d. Build a basic map
Step 7: In the Data pane, double-click Country.
Step 8: On the Marks card, click the + icon on the Country field.



e. Add visual detail

Step 9: So after Dragging Sales from Measures to circled part i.e. Color we may see below output. Also check in Marks instead of automatic maps is set or not



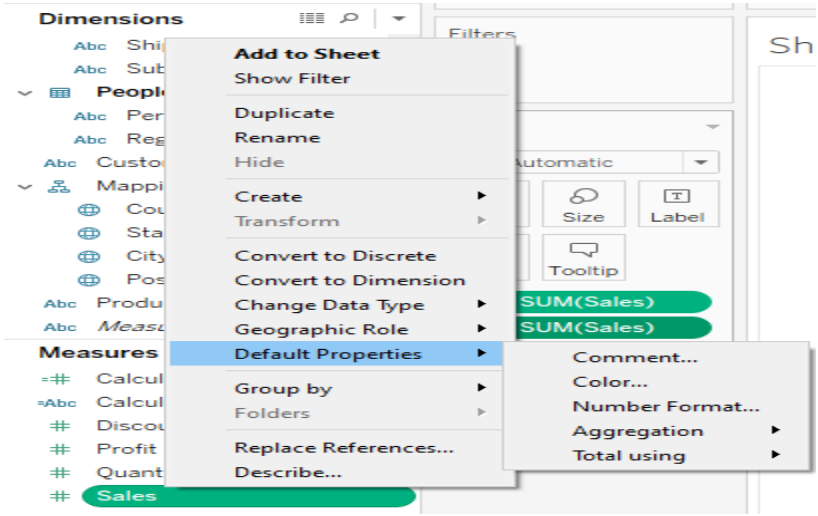
Step 10: Add labels

1. From Measures, drag Sales to Label on the Marks card. Each state is labeled with sum of sales. The numbers need a little bit of formatting, however.

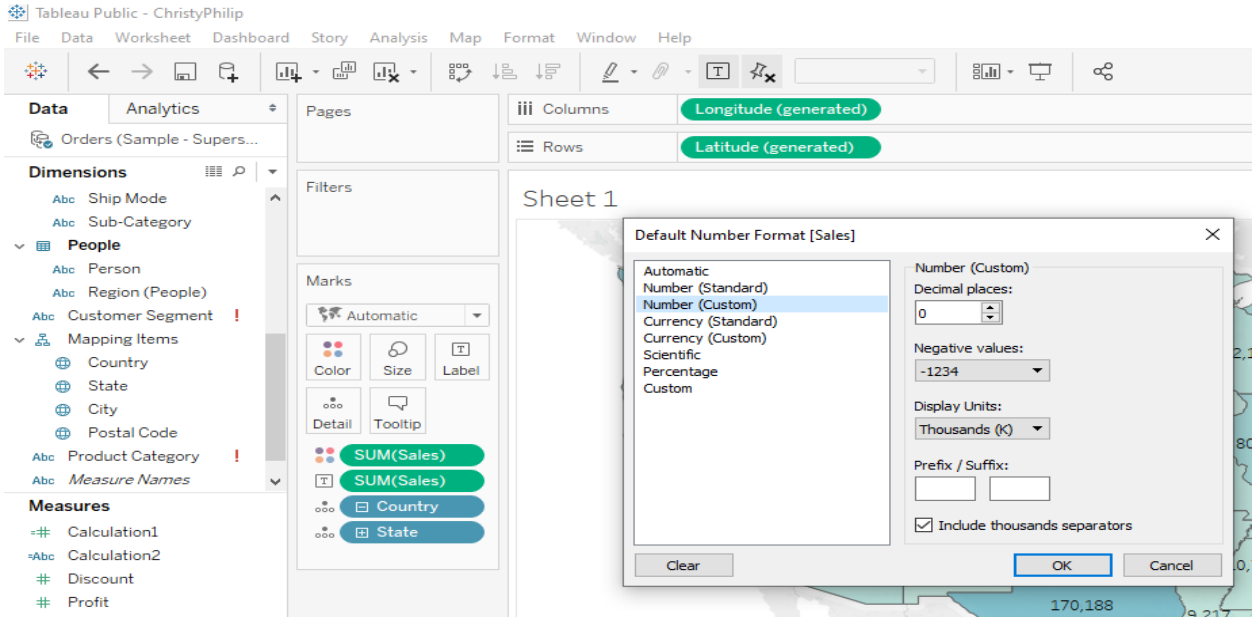
2. In the Data pane, right-click Sales and select Default Properties > Number Format.

3. In the Default Number Format dialog box that opens, select Number (Custom), and then do the following:

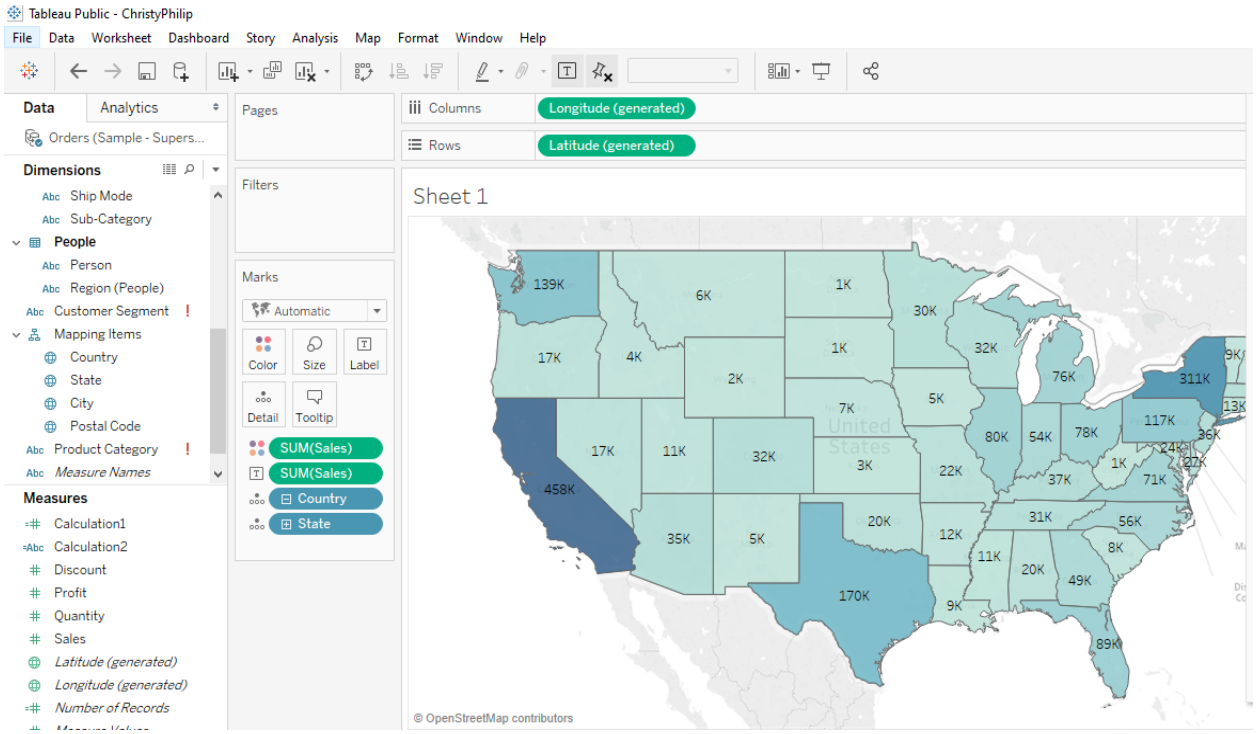
- **For Decimal Places, enter 0.**
- **For Units, select Thousands (K).**
- **Click OK.**



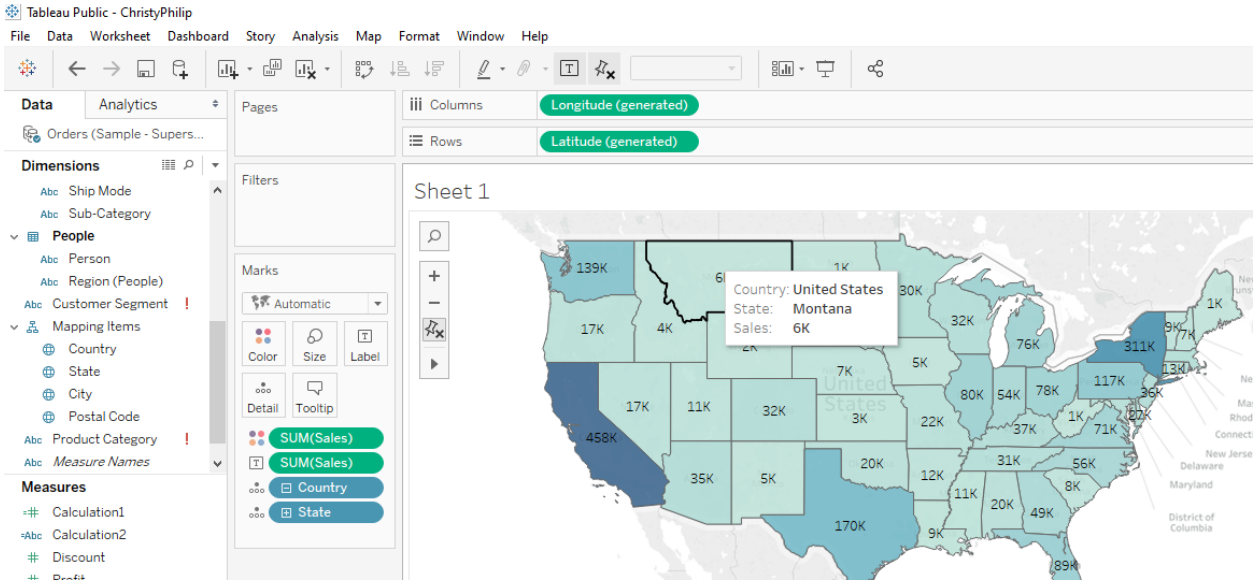
Step 11: Choose Number Formatting→Do below setting and click Ok.



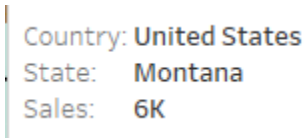
Q 2: Show Profit Ratio of each state as tooltip on map



Step 12:Hover on map to see result



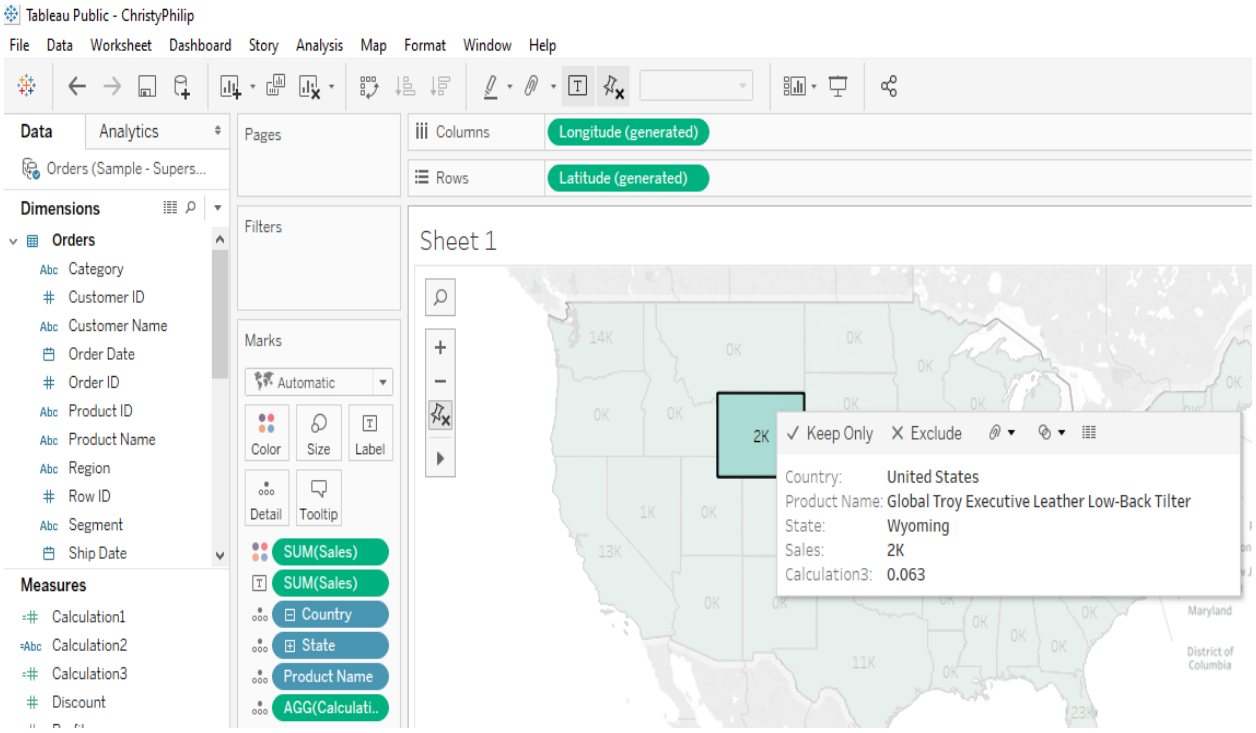
Result:



Q3: Show Profit ratio for Grip Envelop products Profit Ratio= (Sum([Profit])/Sum([Sales]))

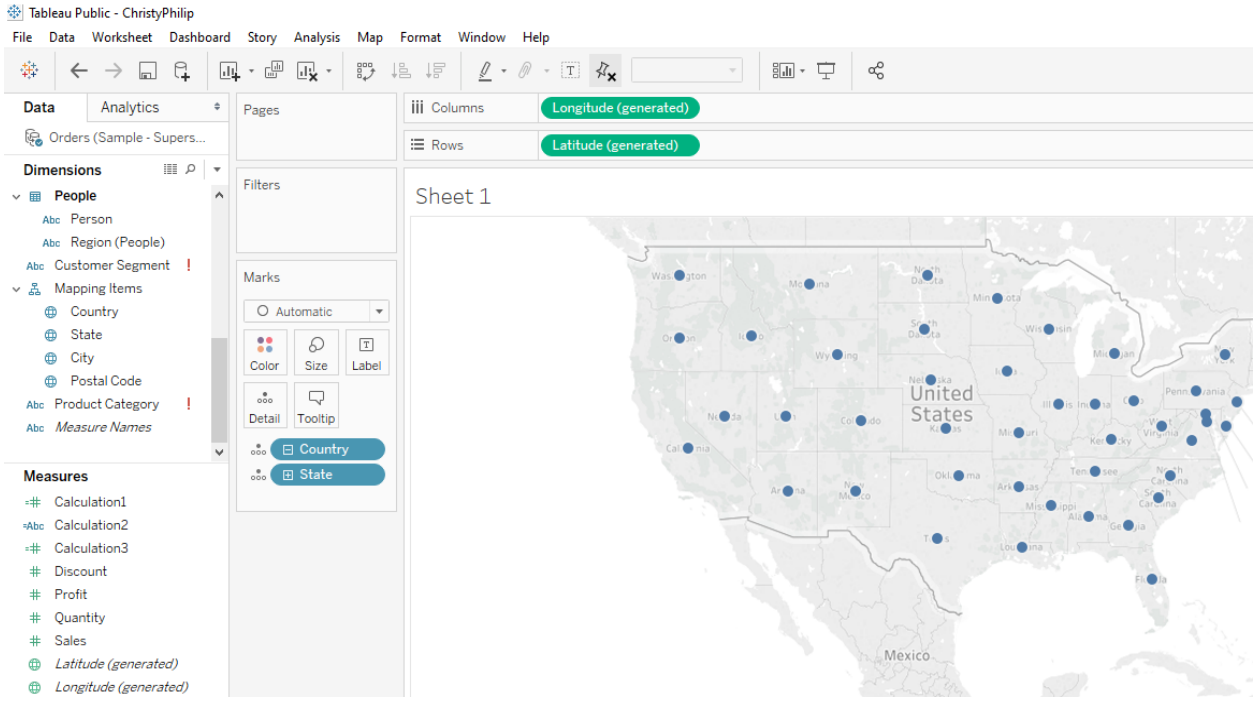
Step 13: In analysis field→Create Calculated Field→Enter Below Formula then Drag and drop from measures pane

Step 14: Hover to see the result

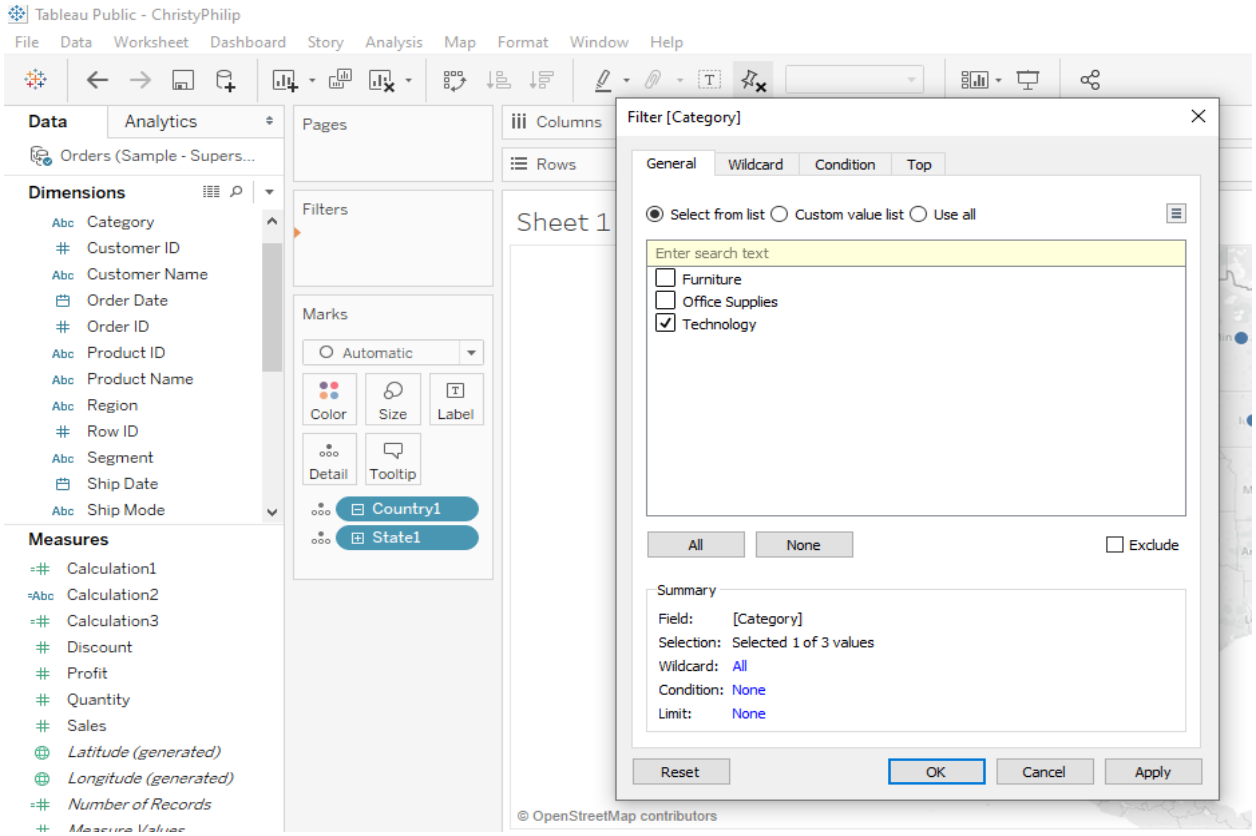


Q 4: In the technology product category which unprofitable state is surrounded by Only profitable states.

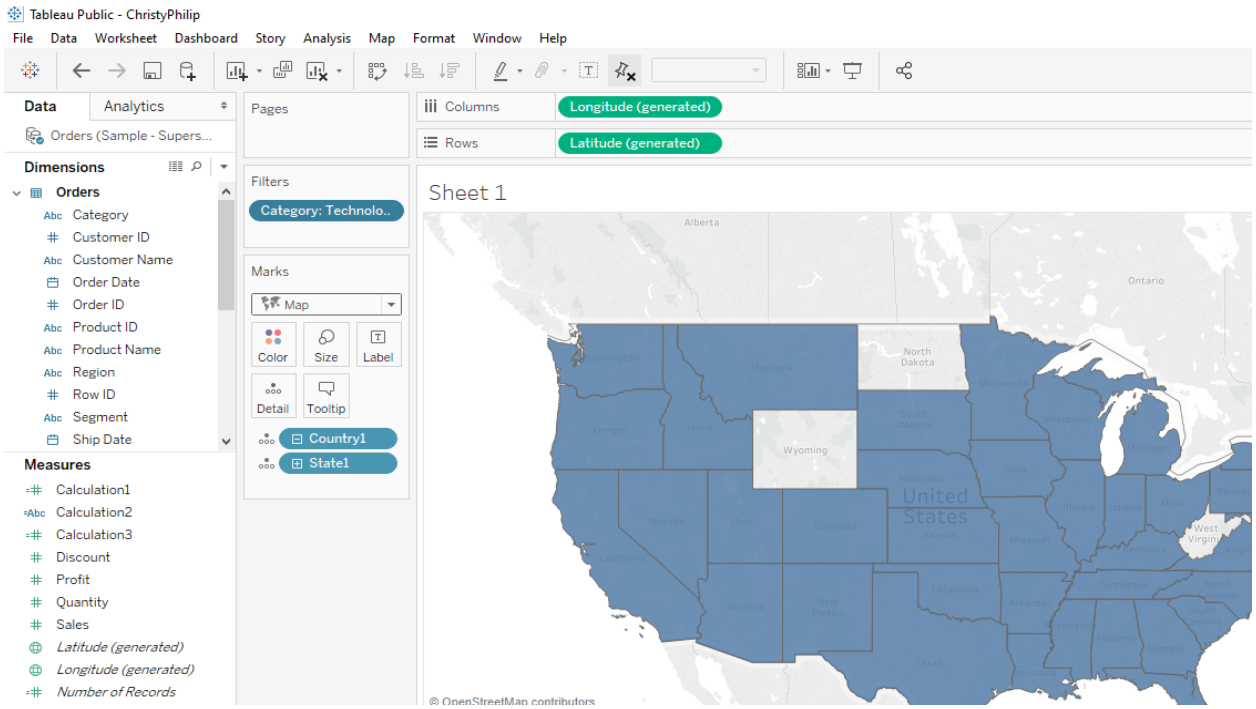
Step 15: In Marks area drag and drop Country and State



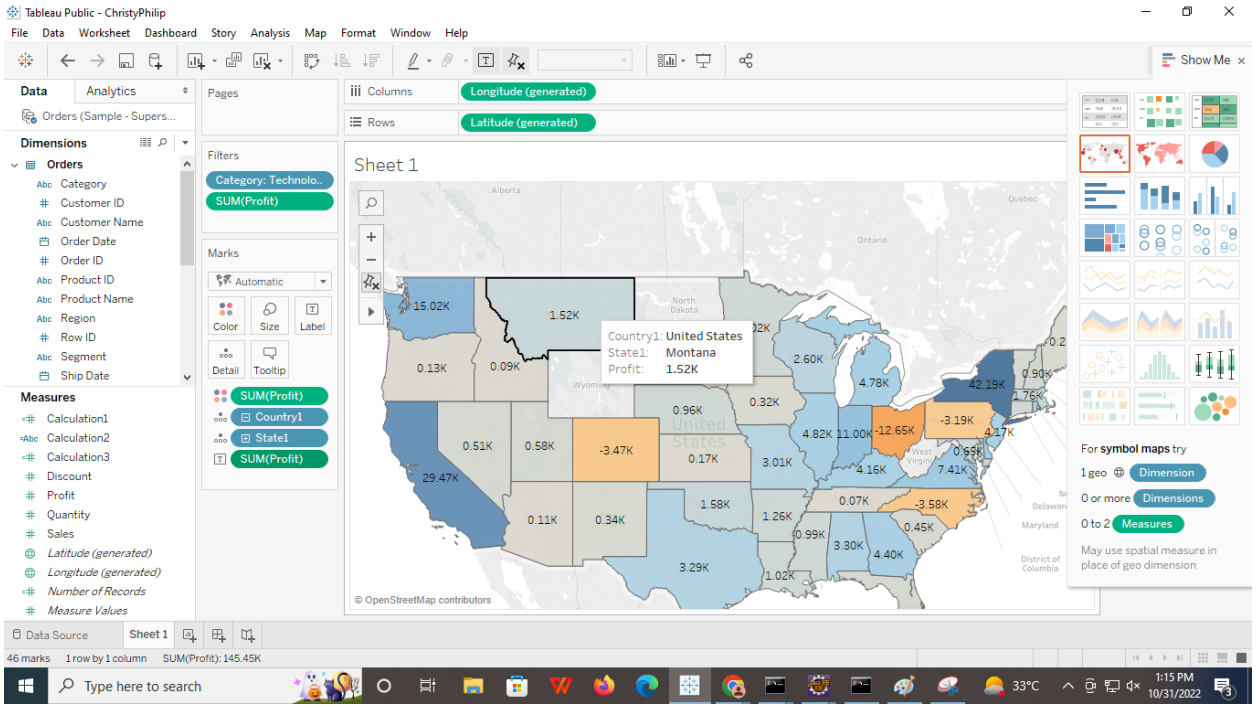
Step 16: Drag and drop the category into filters and choose Technology



Result:



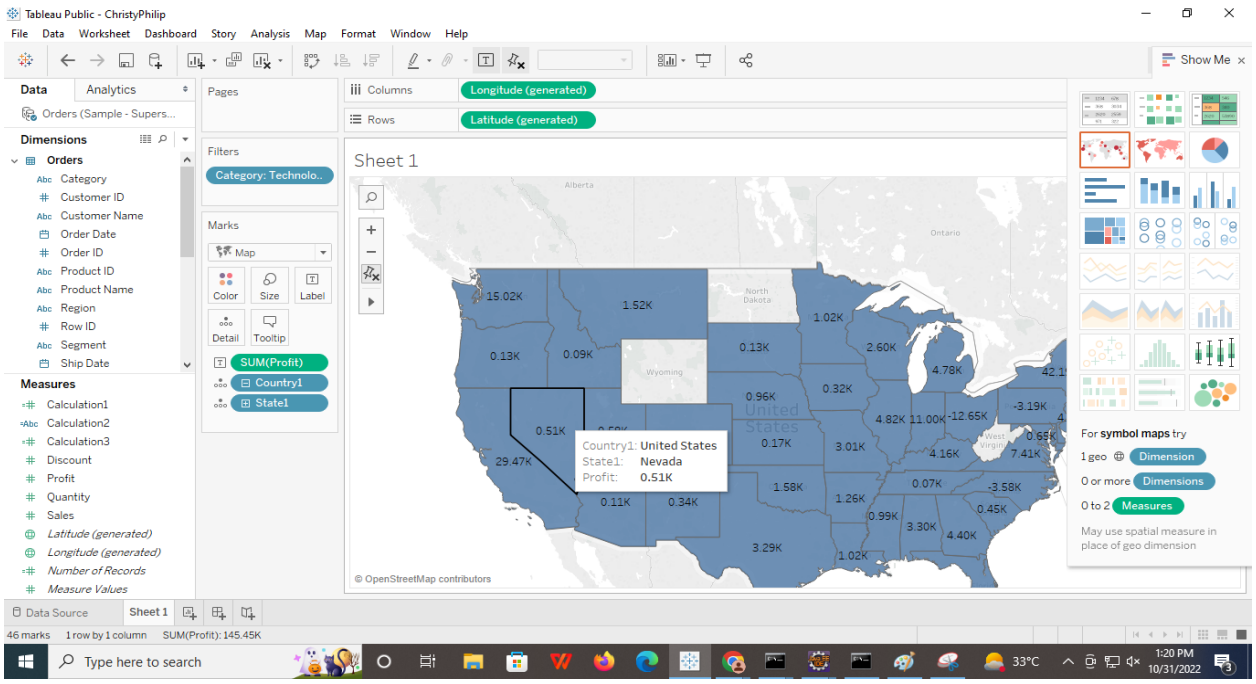
Step 17: Now Drag the profit measure to color mark and hover on state to see result



Result:

Country1: United States
State1: Montana
Profit: 1.52K

Step 18: In filters remove SUM(Profit) and from Marks remove color SUM(Profit) and hover to see result

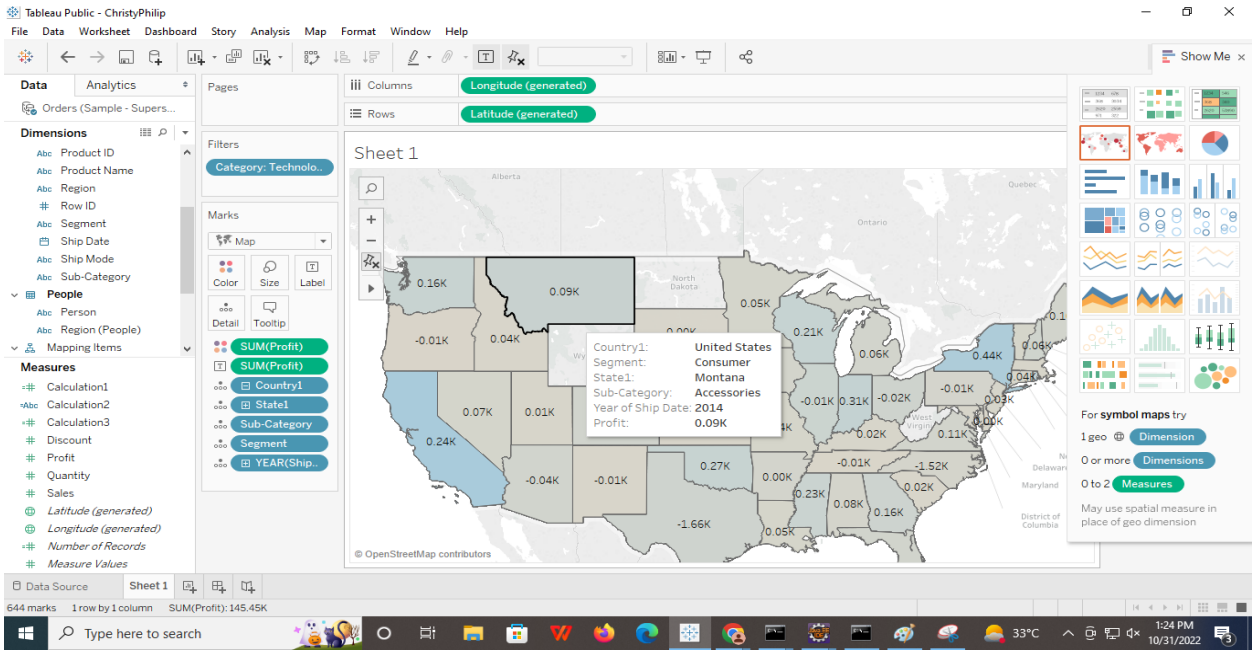


Result:

Country1: United States
State1: Nevada
Profit: 0.51K

Q 5: Which state has the worst Gross Profit Ratio on Envelopes in the Corporate Customer Segment that were Shipped in 2015?

Step 19: Add Sum(Profit), Sub Category, Segment and Year in Marks section set SUM(Profit) to color



Result:

Country1: United States
Segment: Consumer
State1: Montana
Sub-Category: Accessories
Year of Ship Date: 2014
Profit: 0.09K

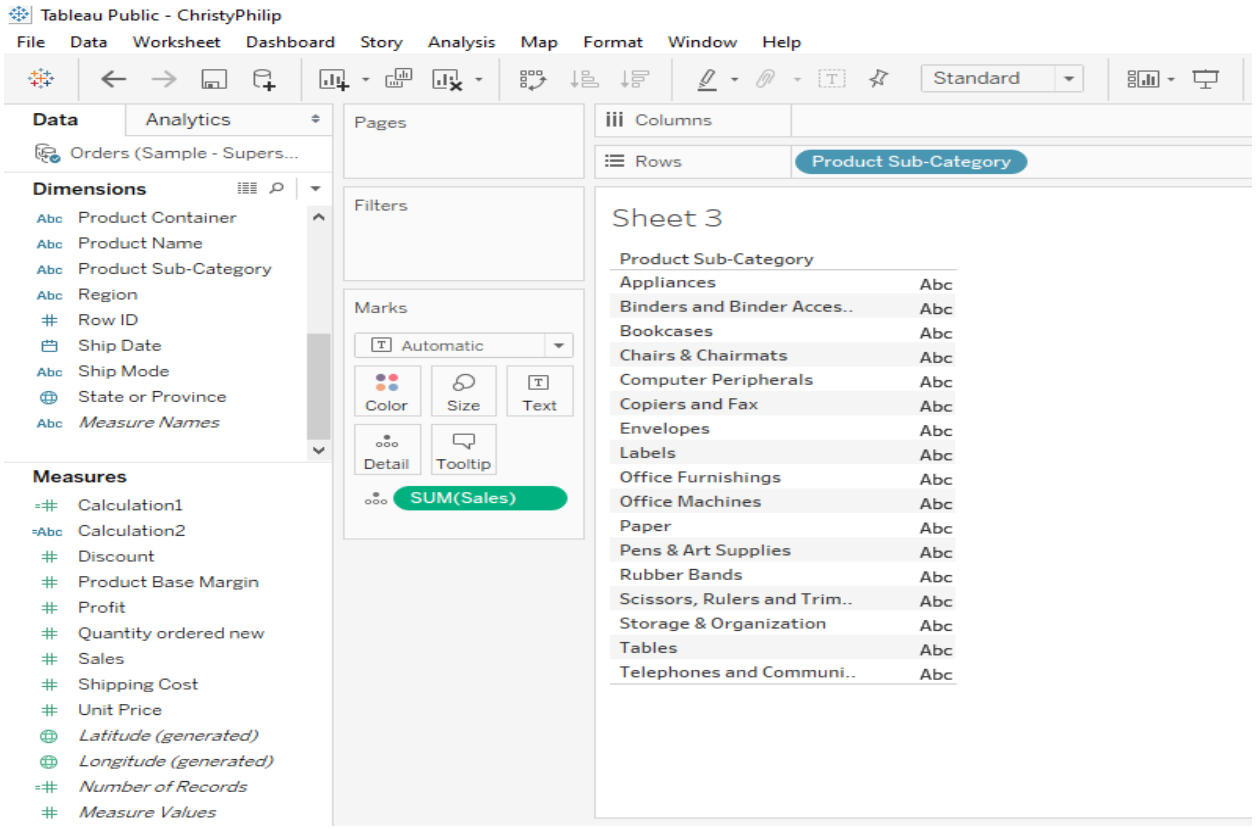
ASSIGNMENT 3

PREPARING REPORTS

Data Set: Super store

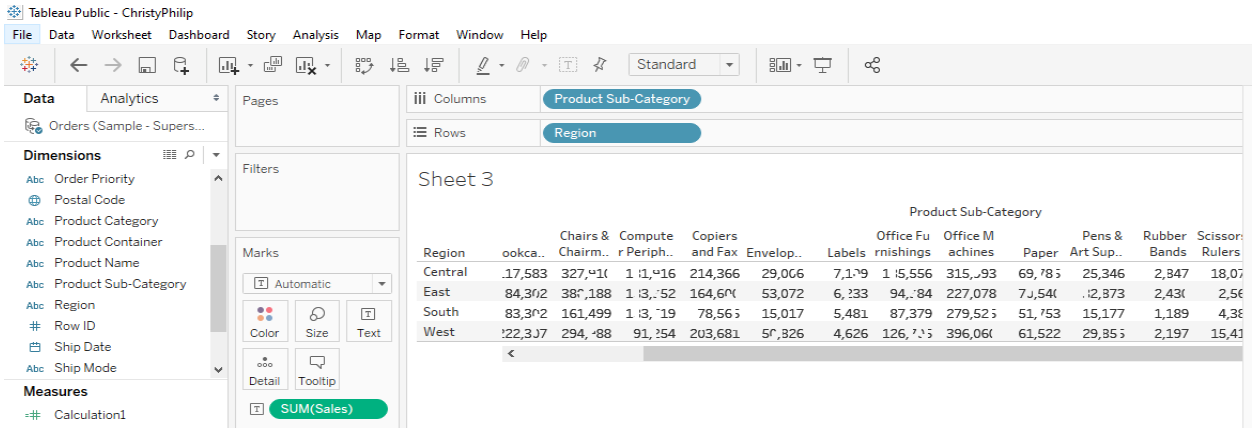
1) Prepare a report showing product category wise sales

Step 1: Drag and drop Product Sub-Category in Rows and SUM(Sales) in Marks



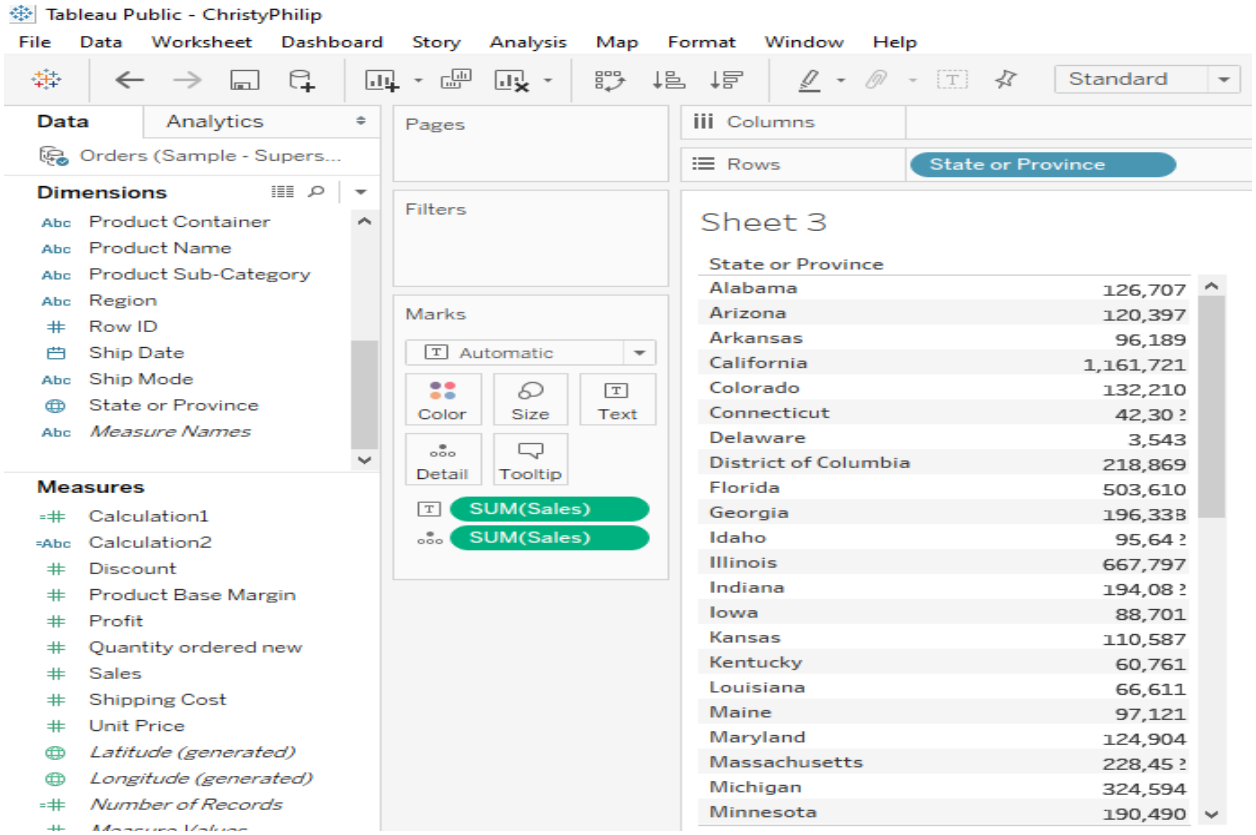
2) Report showing region wise product wise sales

Step 2: In Columns drag and drop Product Sub-Category and in rows drag and drop Region

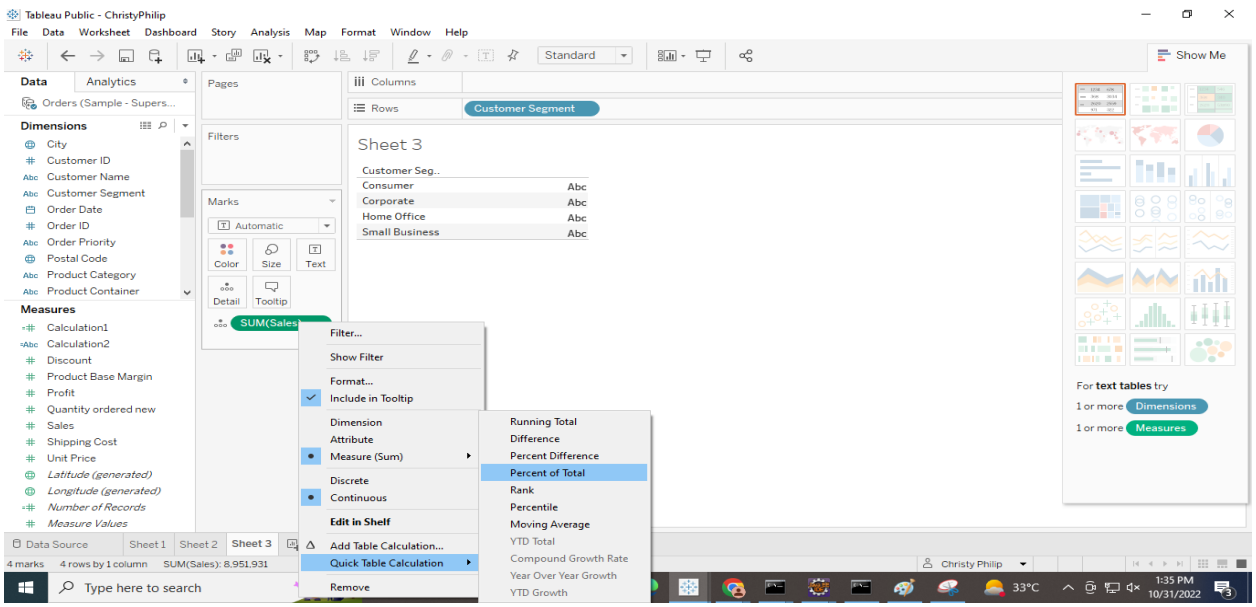


3) Report showing state wise sales

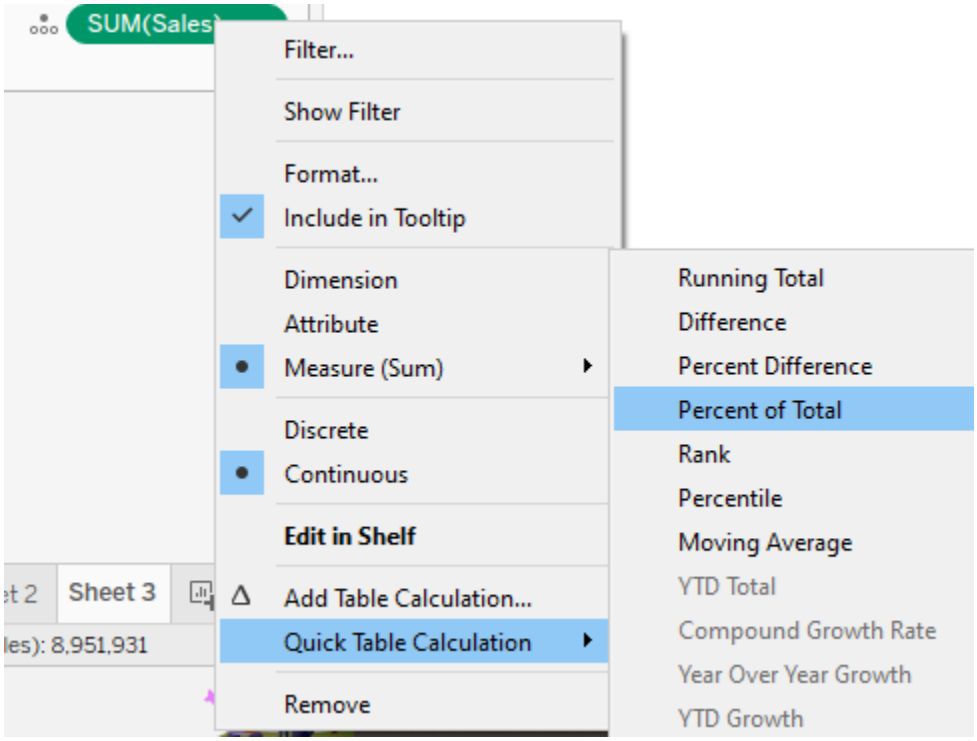
Step 4: Drag and drop State or Province in Rows



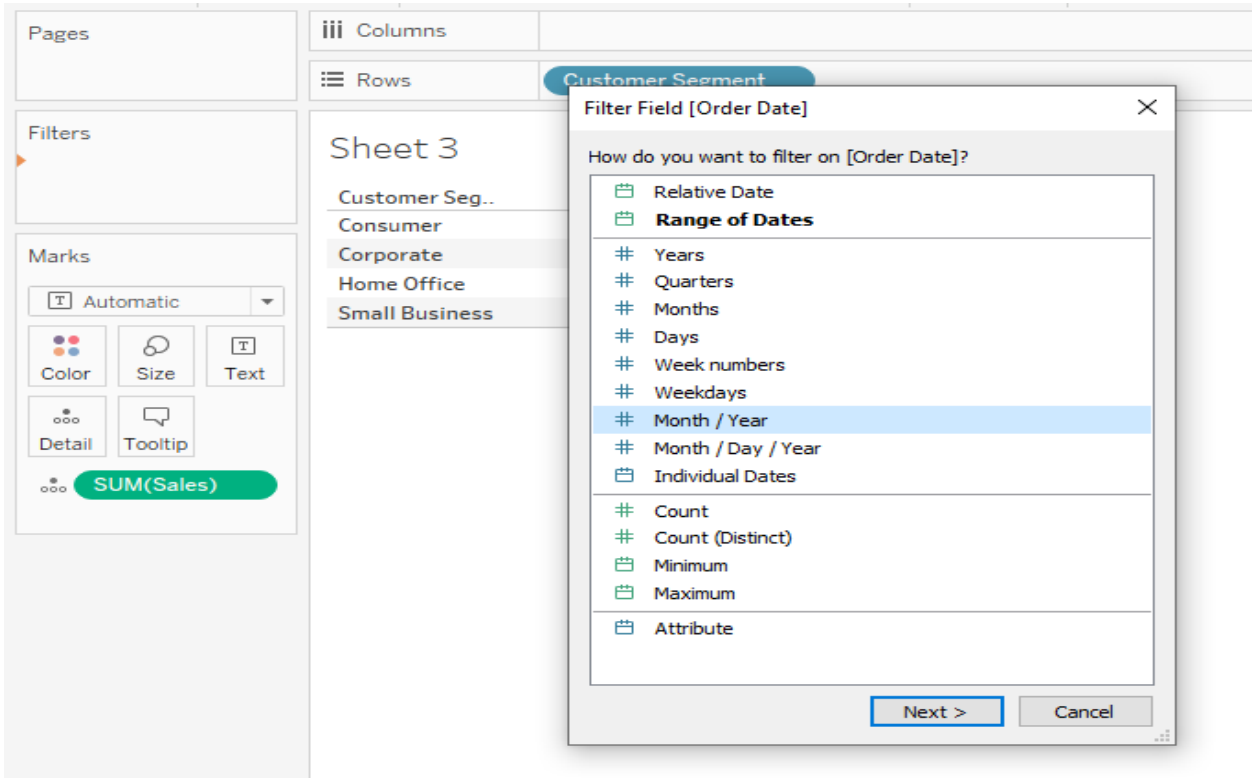
4) What is the percent of total Sales for the 'Home Office' Customer Segment in July of 2014?



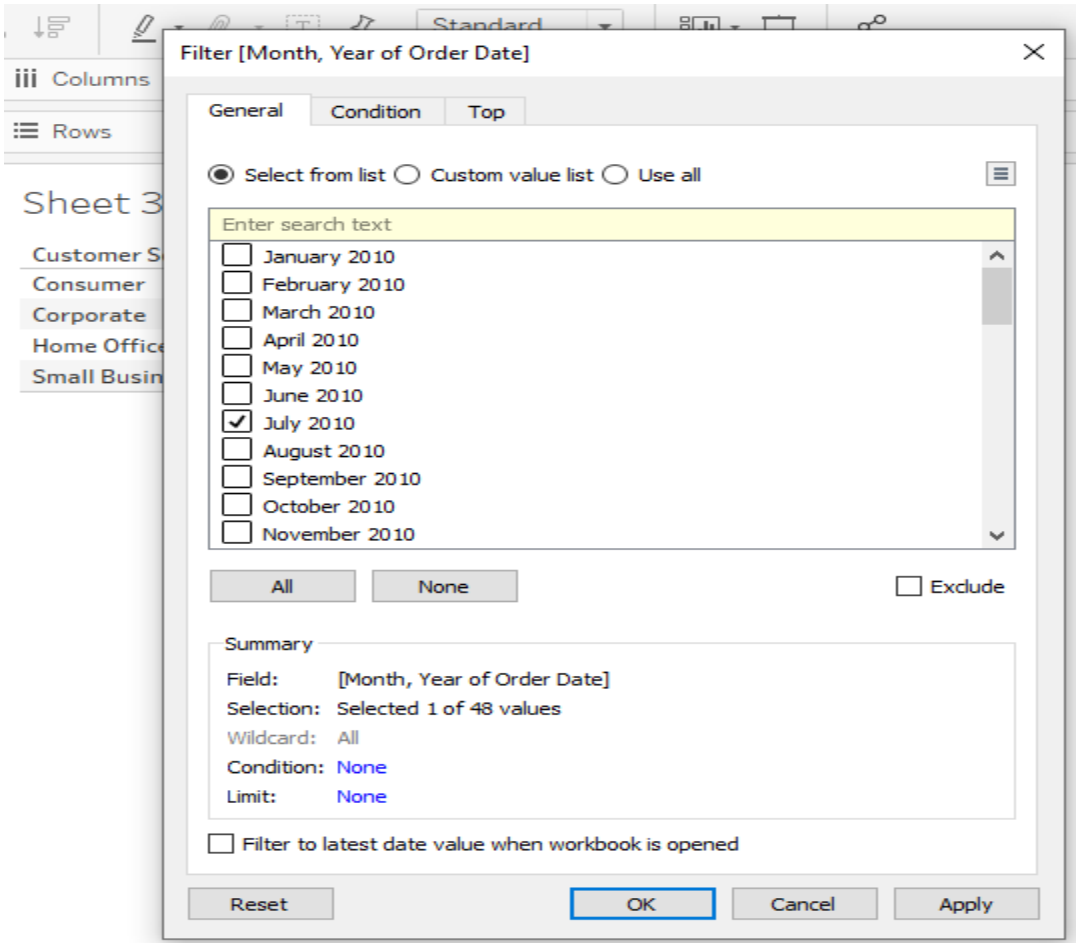
Step 5: Right click on sales which was dragged in Marks section and go to→Quick Table Calculation→Percent of Total



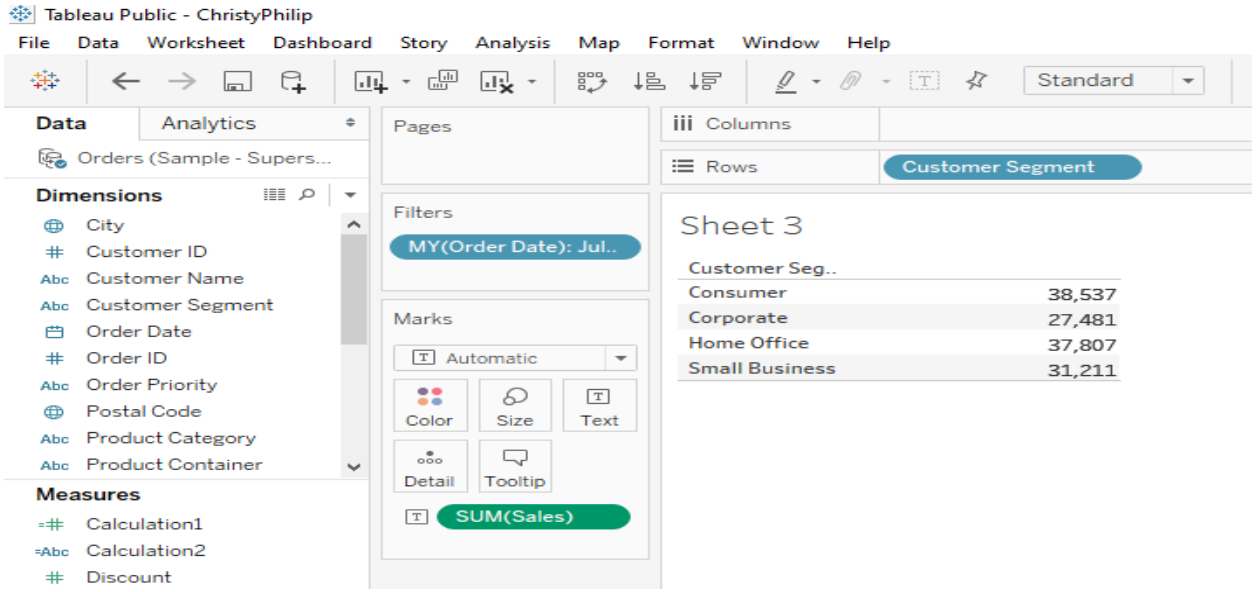
Step 6: Drag and drop Order Date in Filters section and choose Month/Year



Step 7: Choose July 2010



Step 8: Set SUM(Sales) to Label in Marks field

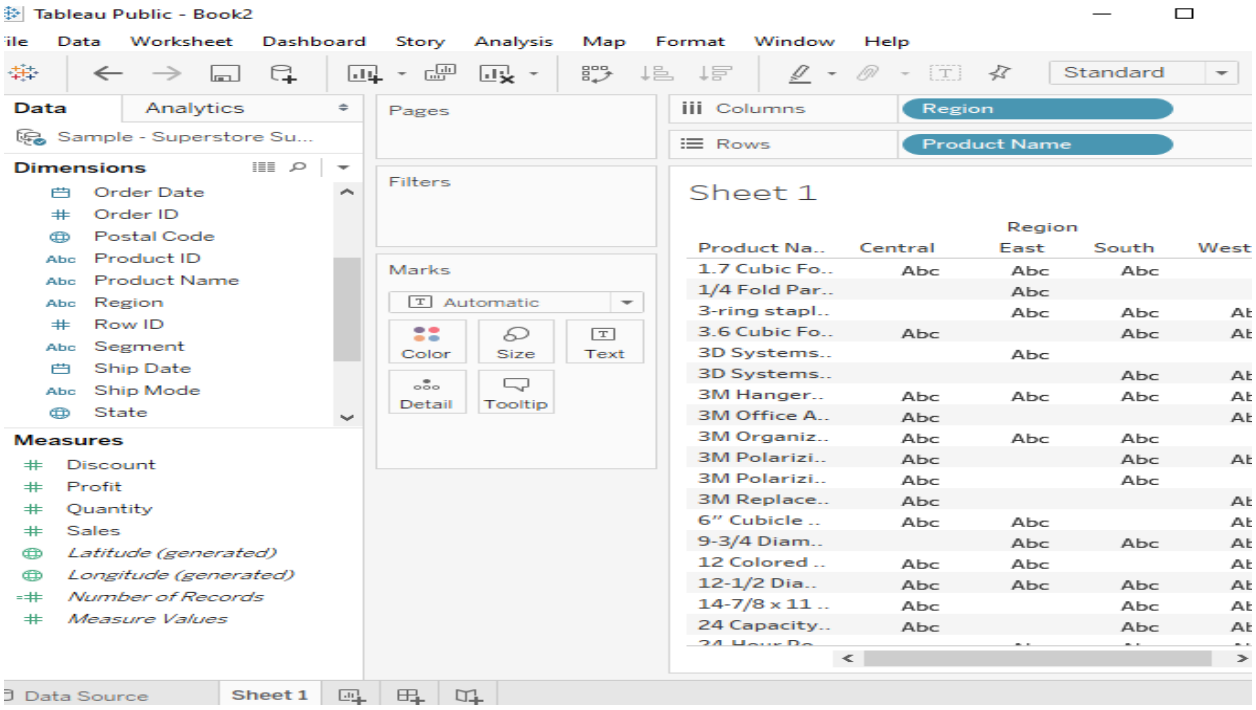


5) Find the top 10 Product Names by Sales within each region. Which product is ranked #2 in both the Central & West regions in 2015?

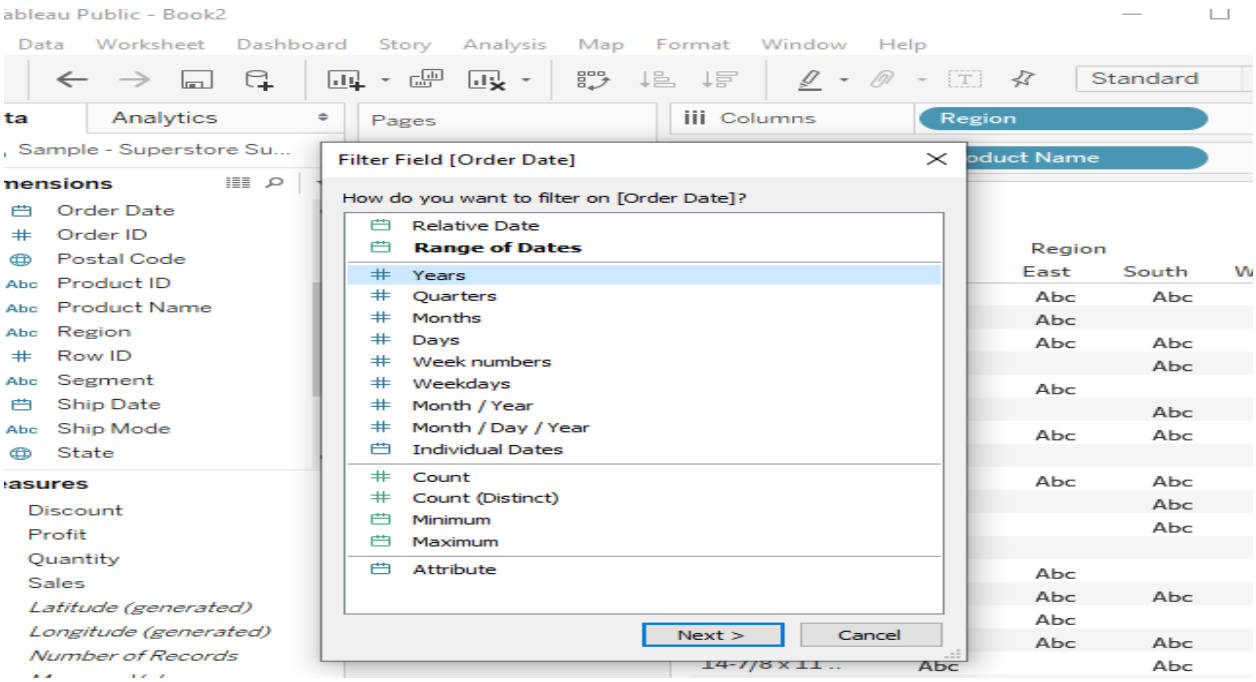
Step 9: Drag Region to Columns

Drag Product name to Rows do Add all members

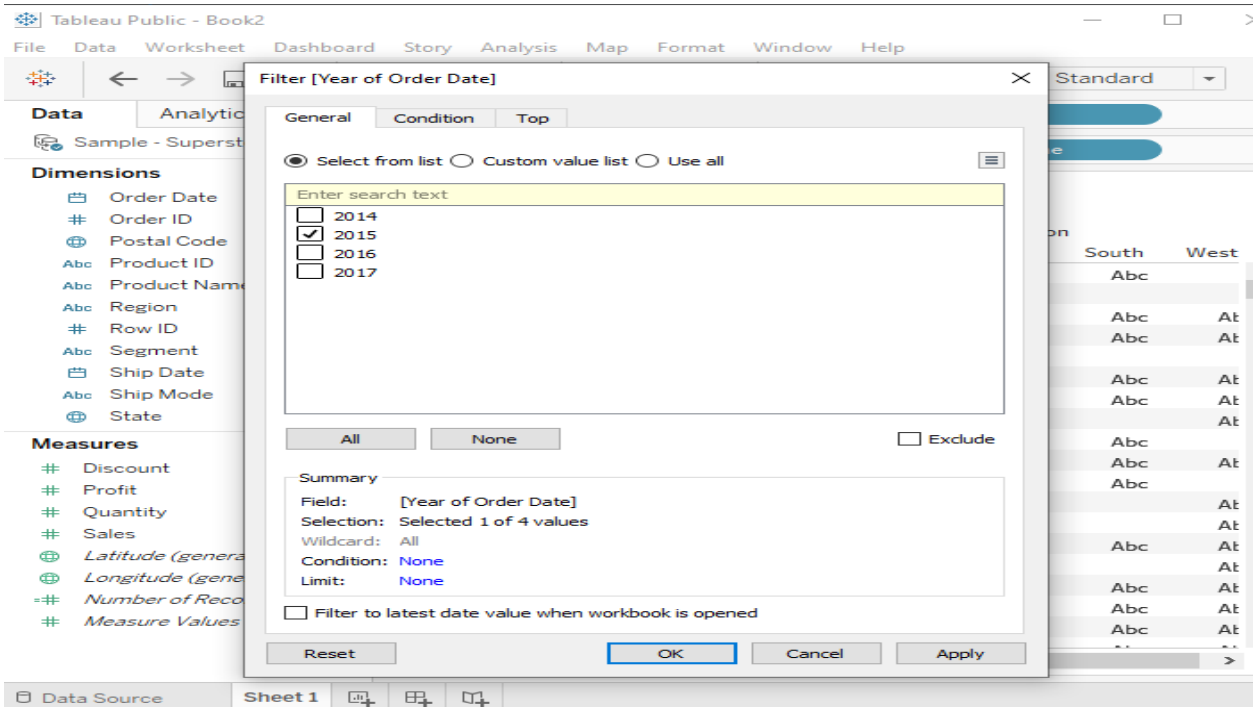
Drag Order date on filters



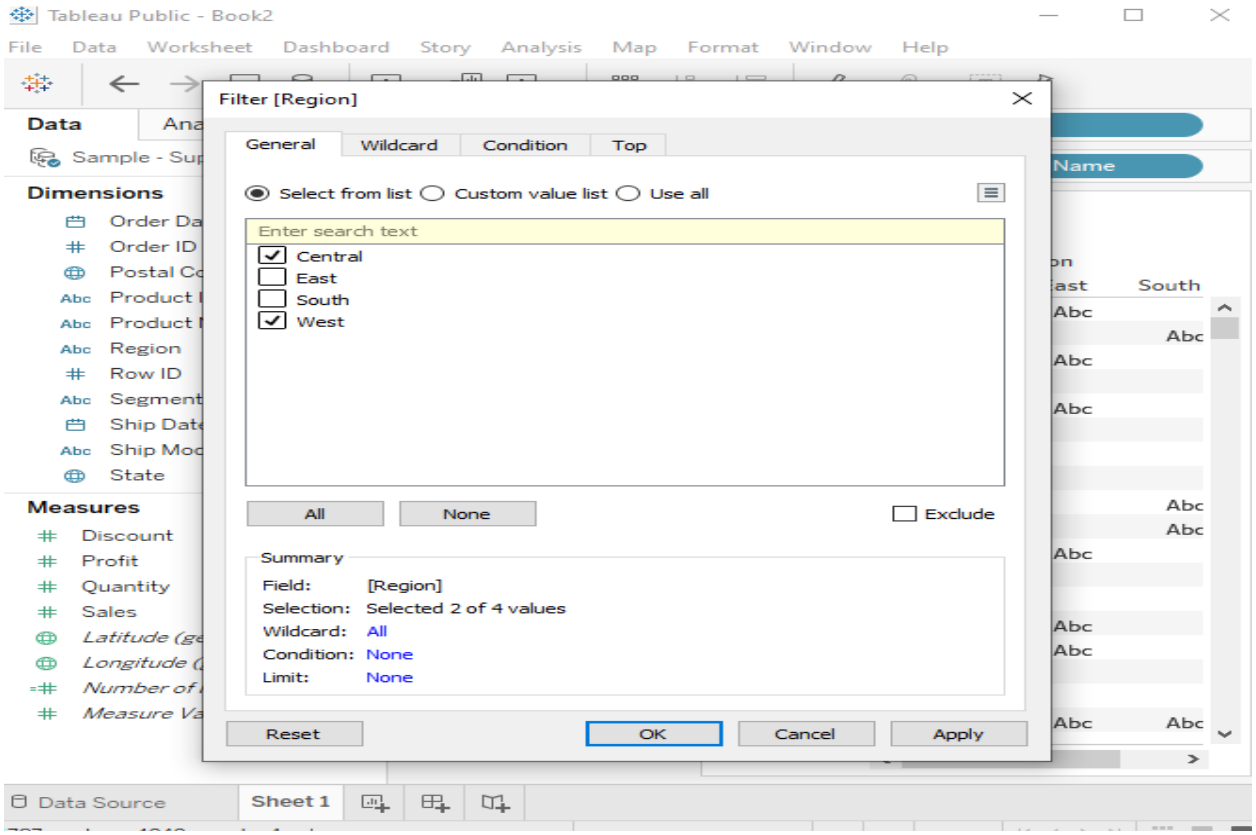
Step 10: Choose Years



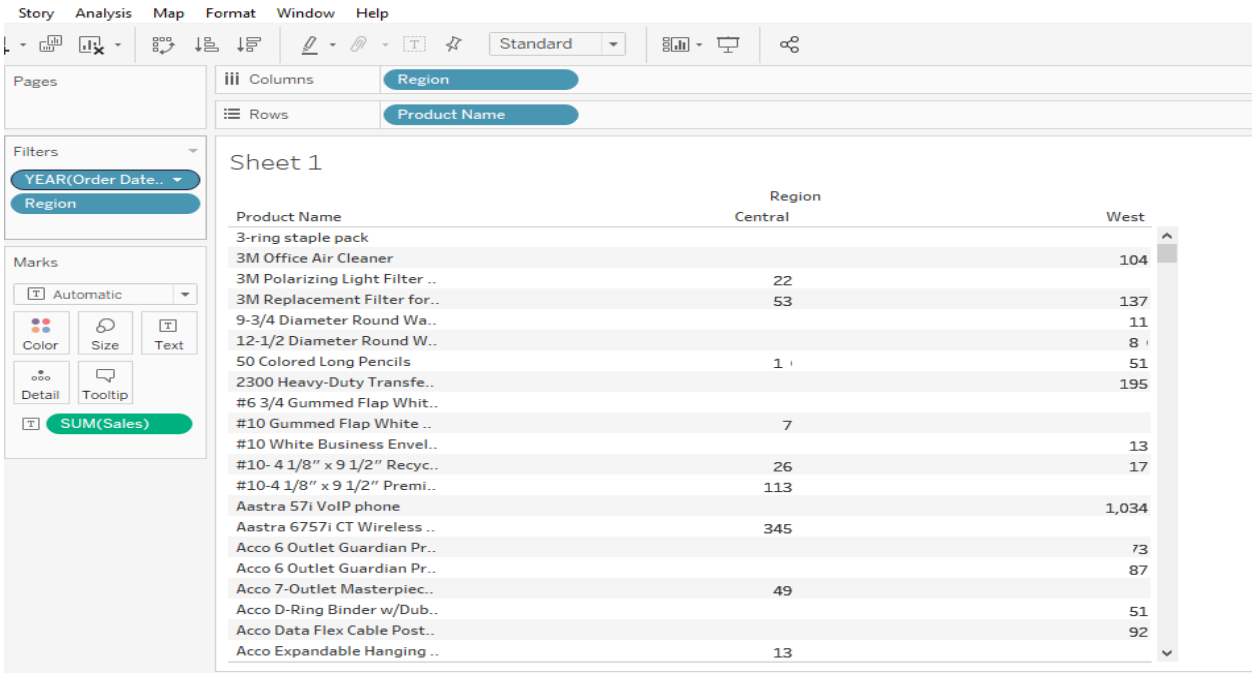
Step 11: Choose 2015



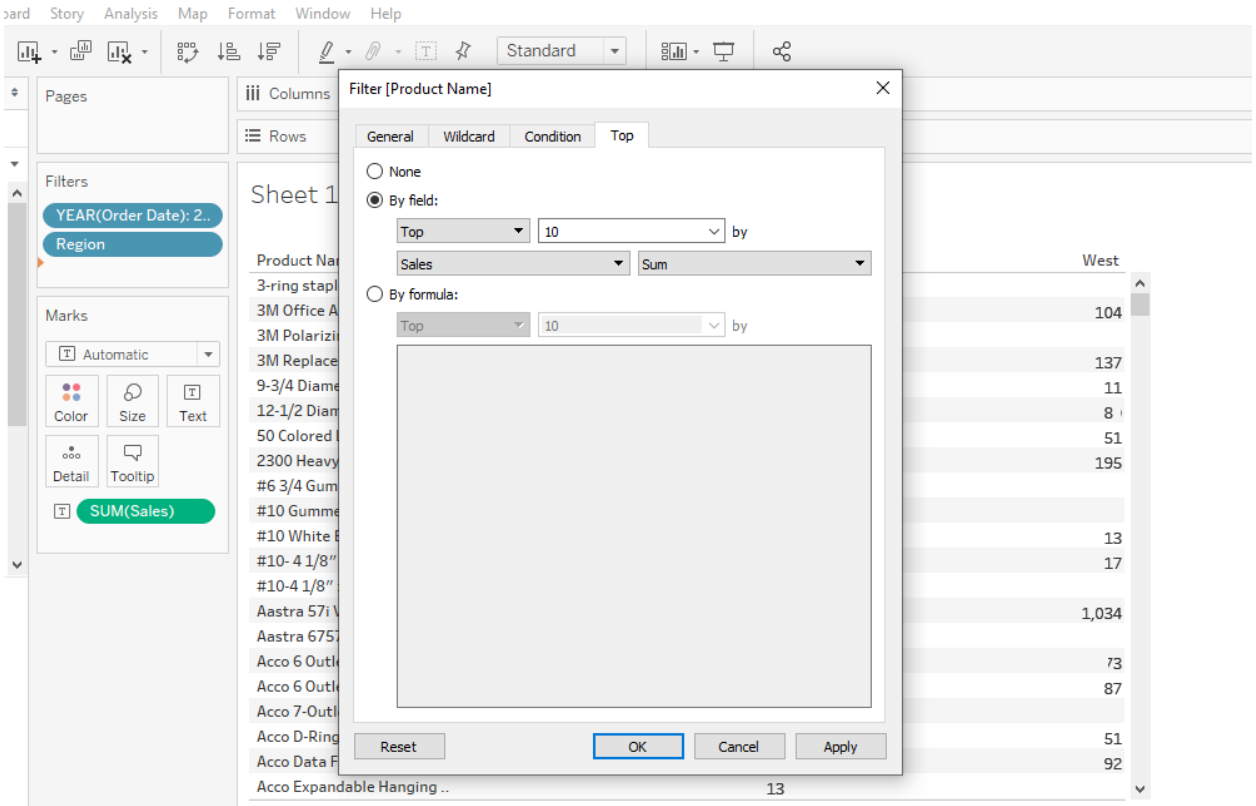
Step 12: Drag Region to Filter shelf and choose Central and West



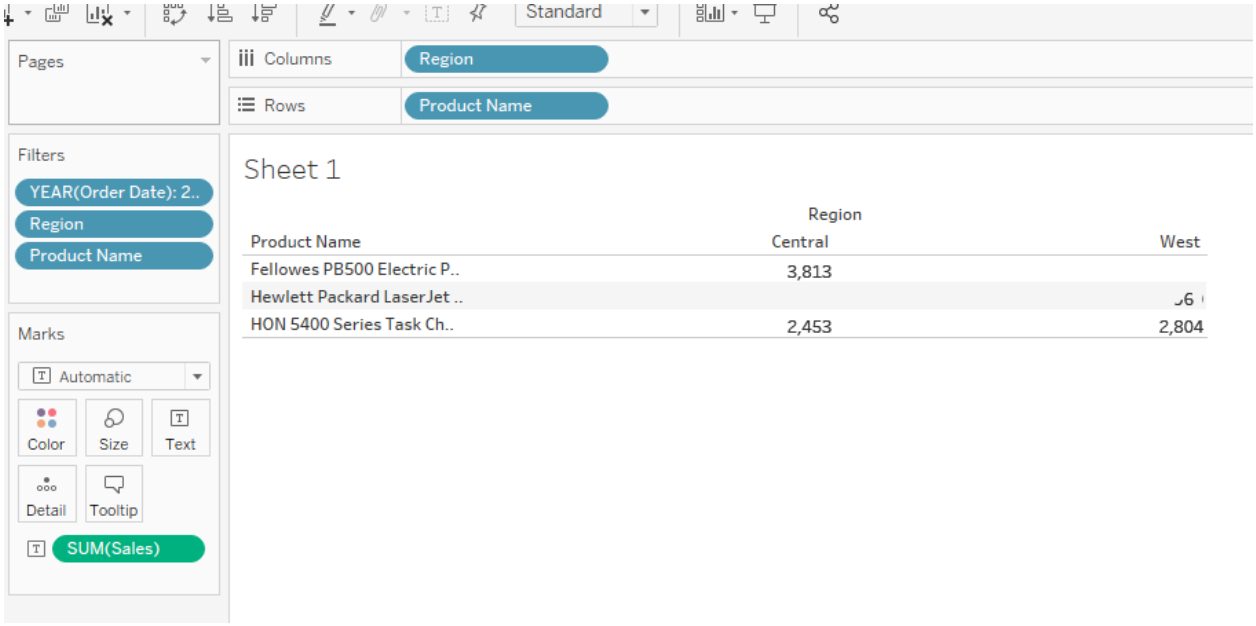
Step 13: Drag Sales from Orders to Marks field and set Sales to Text



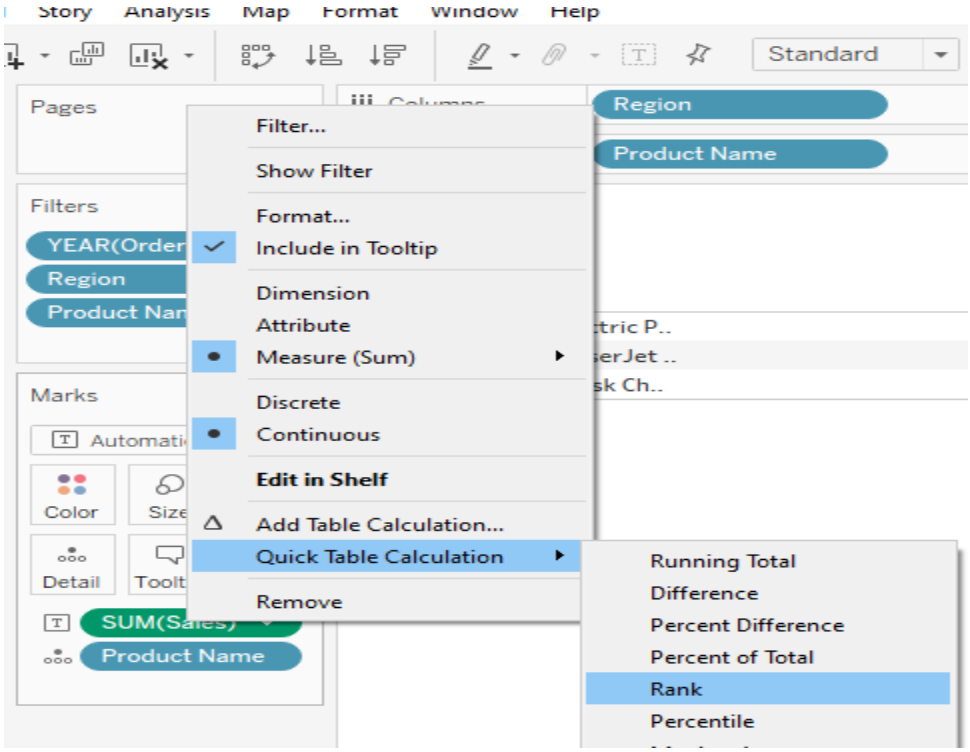
Step 14: Add the "Product name" on the Filter shelf. Once the Filter Pop up is open,Select "TOP" tab choose By Field it should be Top 10 by Sum (Sales).



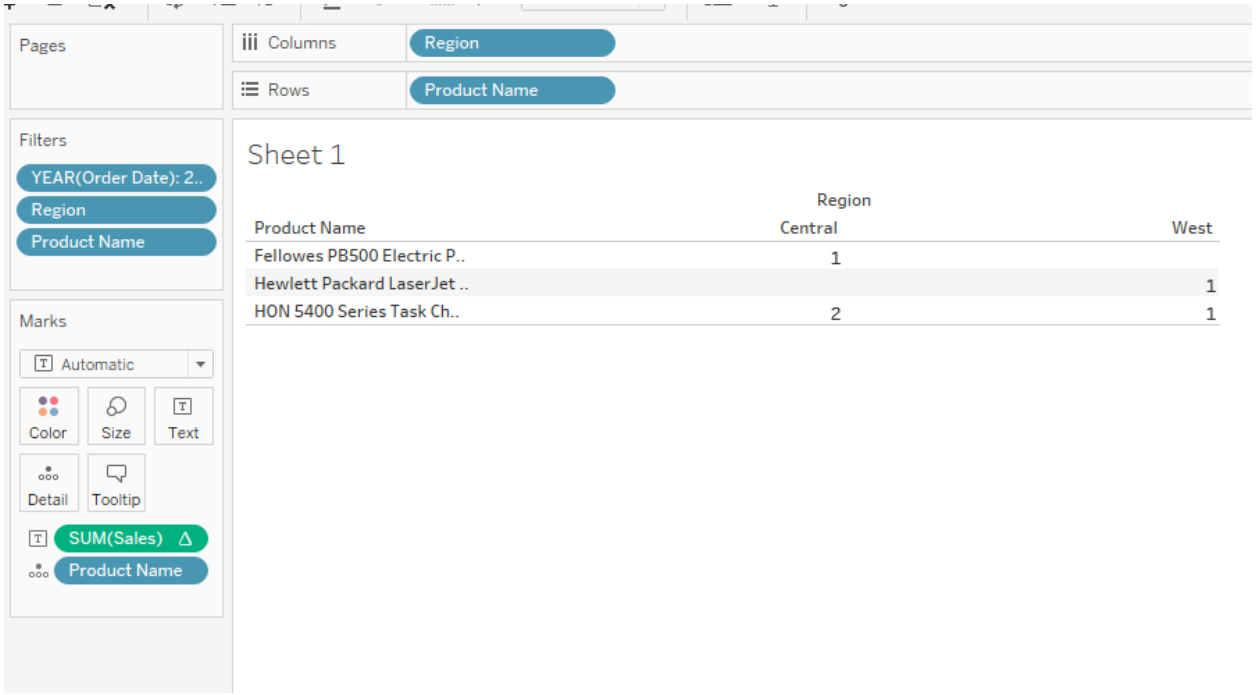
Step 15: After clicking Ok the result view



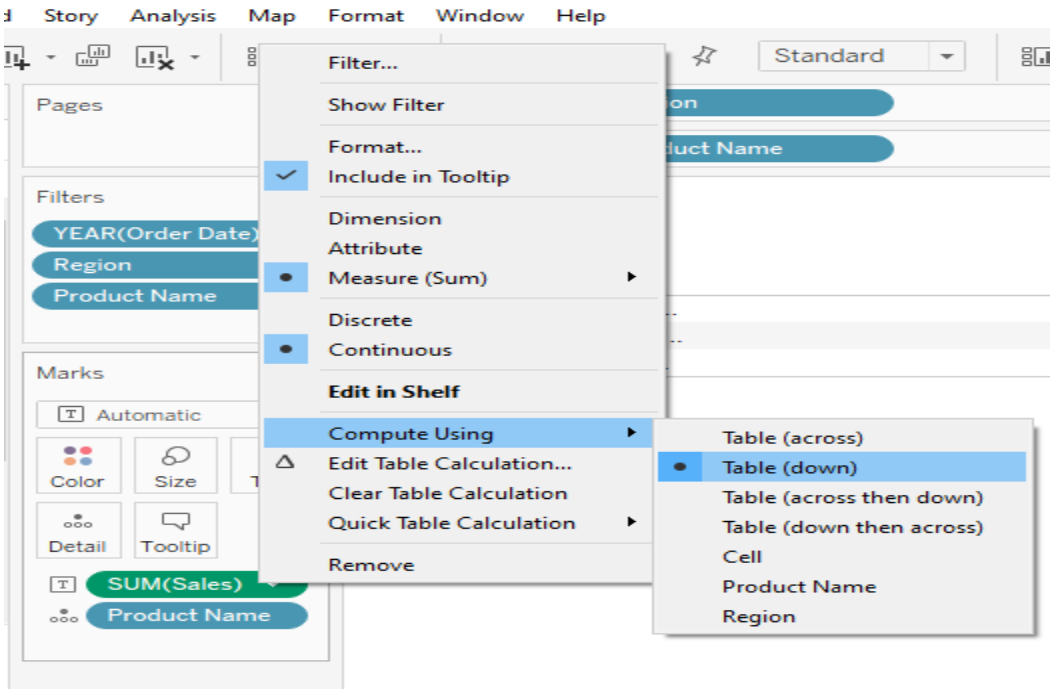
Step 16: Right click on the SUM(Sales) measure in Marks field and then select Quick Table Calculation and click on arrow sign > Rank.



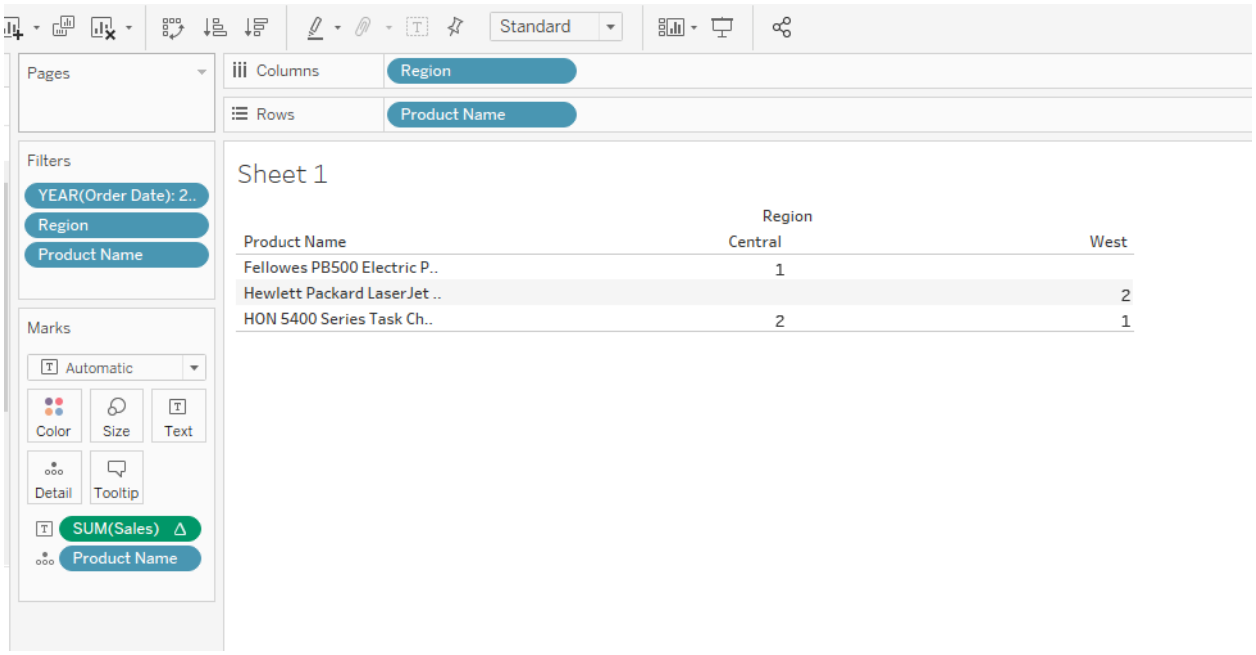
Step 17: Click Ok



Step 18: As the default addressing is Table across, please change it into Table Down (Compute using →Table Down).
Right click on sales in Marks field→Compute using →Table Down



Result:



MONGODB [JSON] & TABLEAU :

Goto Ubuntu

Command: - use Tableau

```
> use Tableau
switched to db Tableau
```

Command: - db.createCollection('sales');

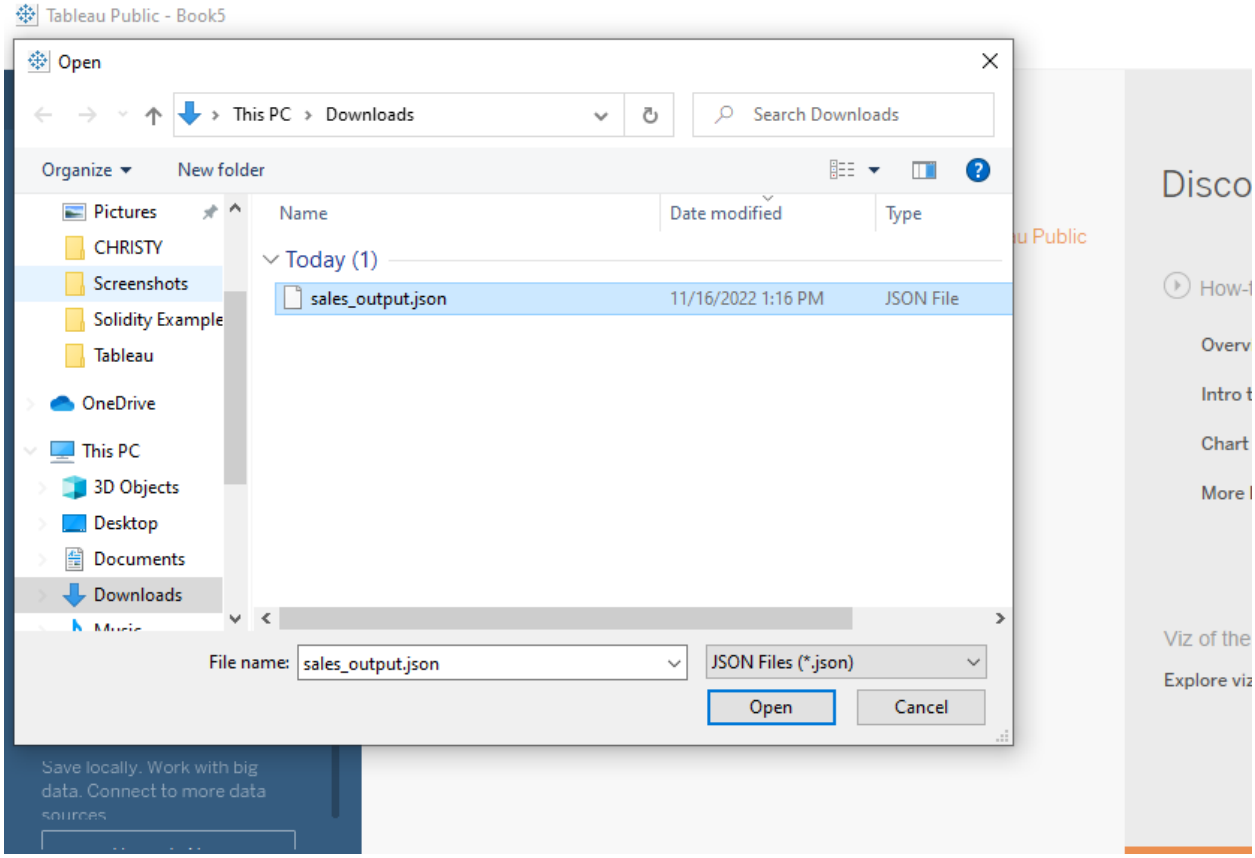
```
> db.createCollection('sales');
{ "ok" : 1 }
```

Command: - mongoexport --db Tableau --collection sales --out "/home/vaish/sales_output.json"

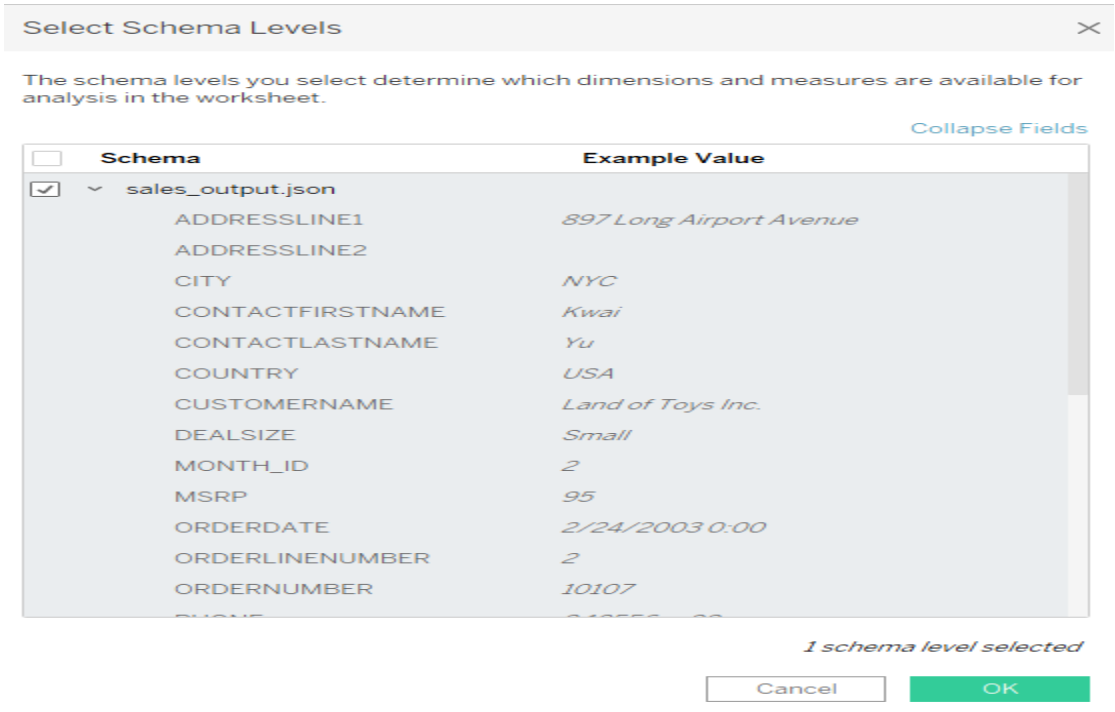
```
vaish@vaishVirtualBox:~$ mongoexport --db Tableau --collection sales --out "/home/vaish/sales_output.json"
2022-11-16T13:07:13.825+0530    connected to: localhost
2022-11-16T13:07:13.829+0530    exported 1 record
vaish@vaishVirtualBox:~$
```

New window of the Tableau

Connect to Data Source: In Data Source choose JSON File



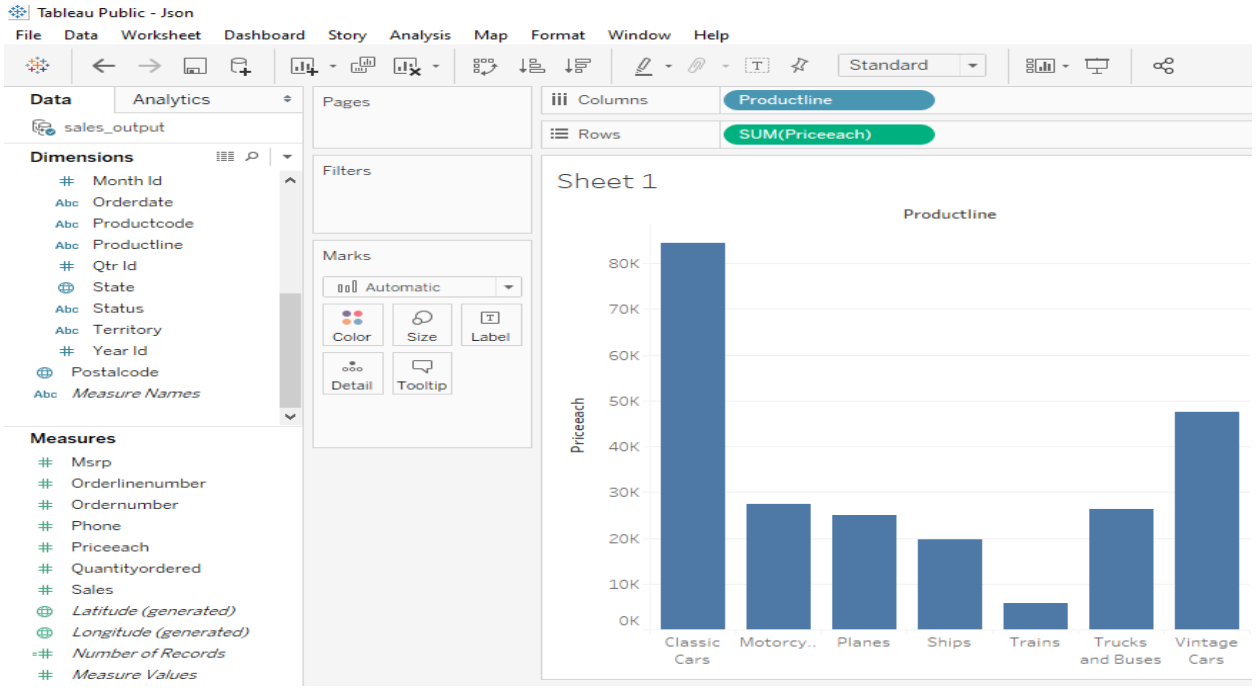
Choose OK



HISTOGRAM: [PRODUCTLINE VS SUM(PRICEEACH]

Step 1: Drag Productline into the columns field

Step 2: Drag Priceeach from Measures to Rows field

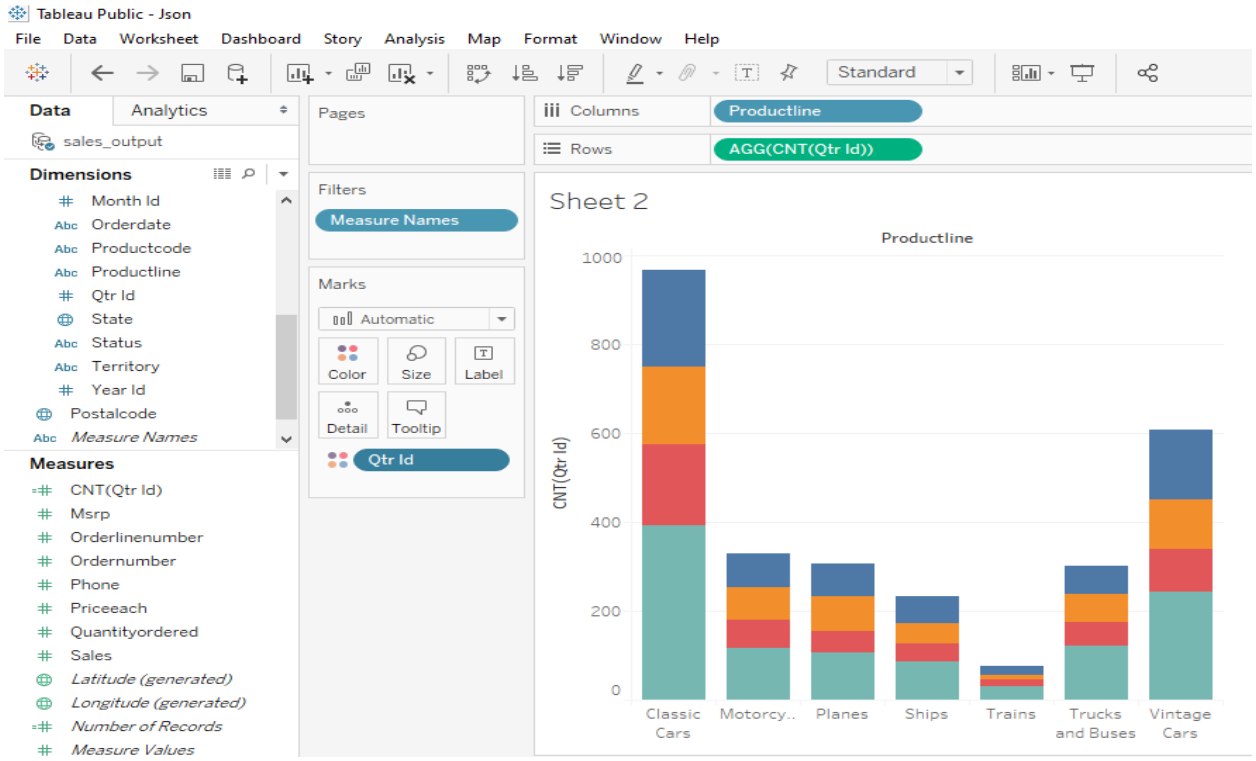


STACKED BAR GRAPH:

Step 3: Create Calculated filed

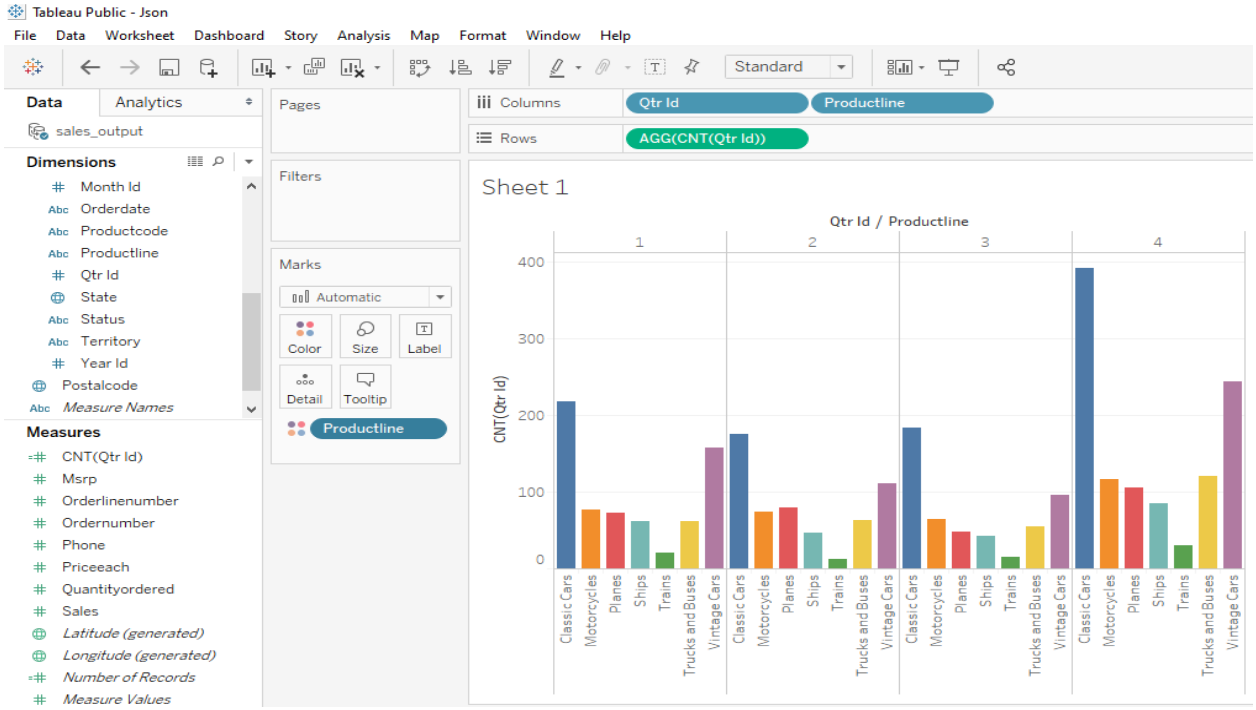
Formula: - COUNT(Qtr Id)

Step 4: Drag and drop Productline to Columns and the Formula created in Measures to Rows. Drag Qtr Id from Dimesnsions to Marks area and set the color



GROUPED STACKED BAR GRAPH: [QTR ID, PRODUCTLINE VS CNT(QTR ID)]

Step 5: Drag Qtr Id, Productline to Columns field and CNT(Qtr Id) from Measures



HORIZONTAL GROUPED BAR GRAPH:

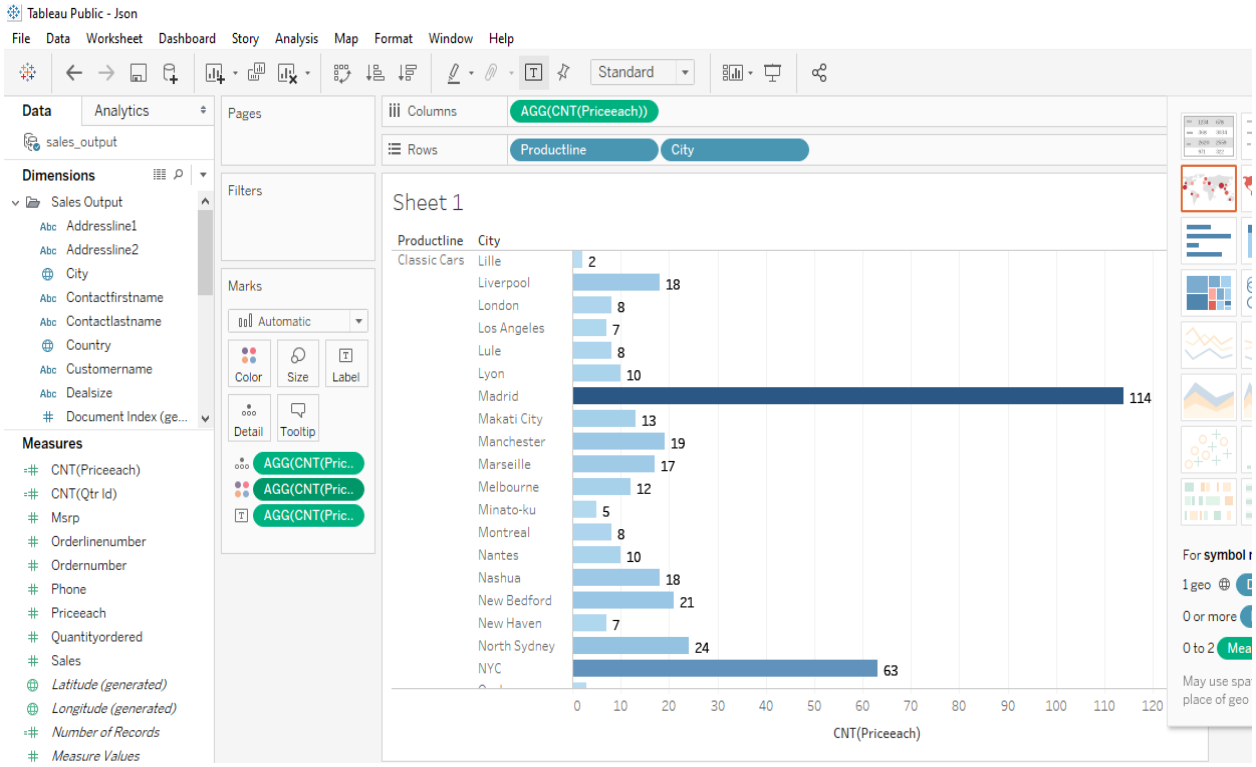
Step 6: Create Calculated Field use below formula

Step 7: Click OK

Step 8: Drag and drop formula created in Step 6 and 7 into Columns(You can see it in Measures section).

In Rows drag and drop Productline and City from Dimensions

In Marks area drag nad drop formula created in Step 6 and 7 three times set 1 to color, another to Label and 1 should be normal



CONCLUSION: -

From this practical I have learned to Connect to data, Build Charts and Analyze Data, Create Dashboard, Create Stories using Tableau.