**Analysing COVID-19 data for India**

**Business Problem:**

In the current pandemic situation where COVID-19 has spread across the world impacting 150+ countries, India too is severely impacted: 34 states / UT affected. First Case Reported on: 30 January 2020 as of 10th June, Ministry of Health and Family Welfare reported a total of 276,804 cases, 134,843 recoveries and 7,751 deaths the infection rate of COVID-19 in India is: 1.7 (significantly lower than in the worst affected countries.

**Business Objective:**

To analyse the covid-19 India data and find key insights on patient demographics, patient clusters, state and district level spread. To be able to foresee the local transmission rate, top affected districts and predict the saturation point of the disease spread

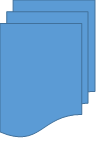
|  |  |  |
| --- | --- | --- |
| Req\_ID | Requirements | Priority |
| 1 | Data preparation: Loading into Hadoop                                   Quality data preparation                                  Identifying tool for BI | P1 |
| 2 | Identifies the 6 affected top districts in India Improves planning and strategy to provide | P1 |
| 3 | Identifies patient demographics to understand the spread and trend in daily cases treatment | P1 |
| 4 | Provides key insights for new policy formation like lockdown extension | P1 |
| 5 | Predicts the saturation point for the spread | P1 |
| 6 | Average time taken by patients to recover, average time a patient stays in hospital – hospital occupancy | P1 |
| 7 | To track load on healthcare facility and predict number of ventilators required in future | P2 |

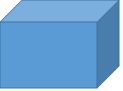
**Descriptive Analysis:**

 Visualize the data to have a view for different levels: India Level, Patient Demographics and District Level View

Predictive Analysis:

Analysis to predict saturation point for new cases at district level (top 6 affected districts).



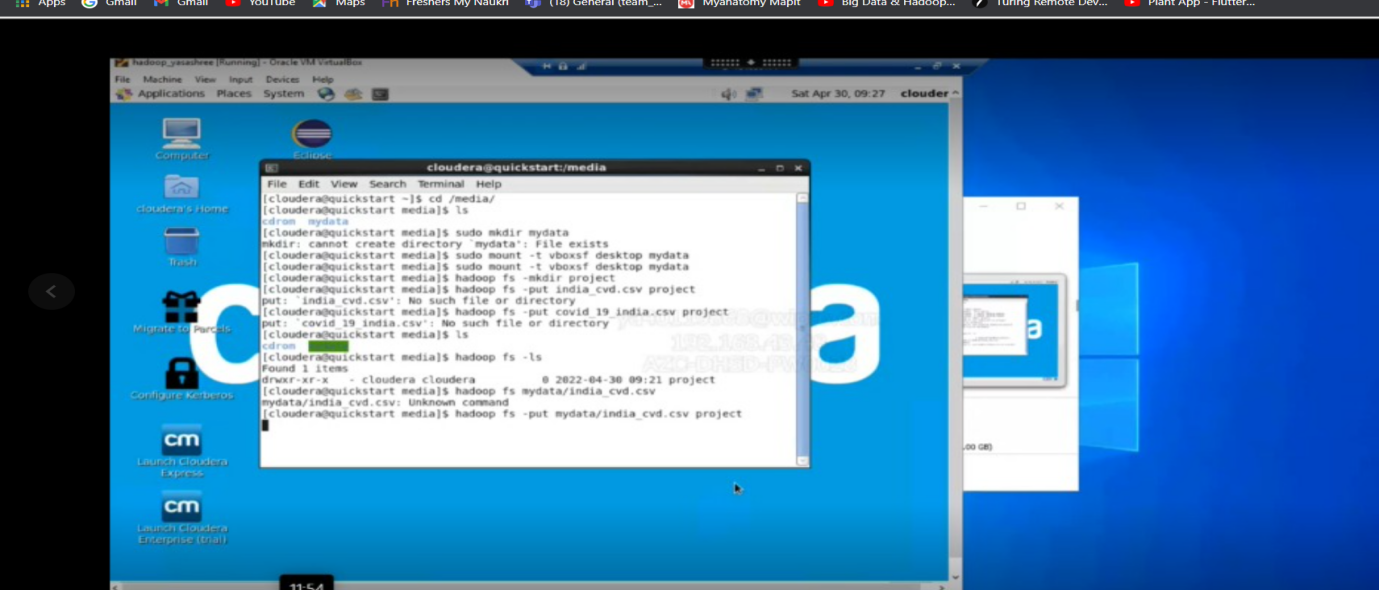




Hadoop – Map Reduce - Hive



**DATA PREPARATION:**

* Loading all data i.e.csv files into Hadoop in Remote desktop is executed.
* 

**IDENTIFY TOOL FOR BI:**

. Here we have used the Tableau as BI Tool for the quality data preparation.

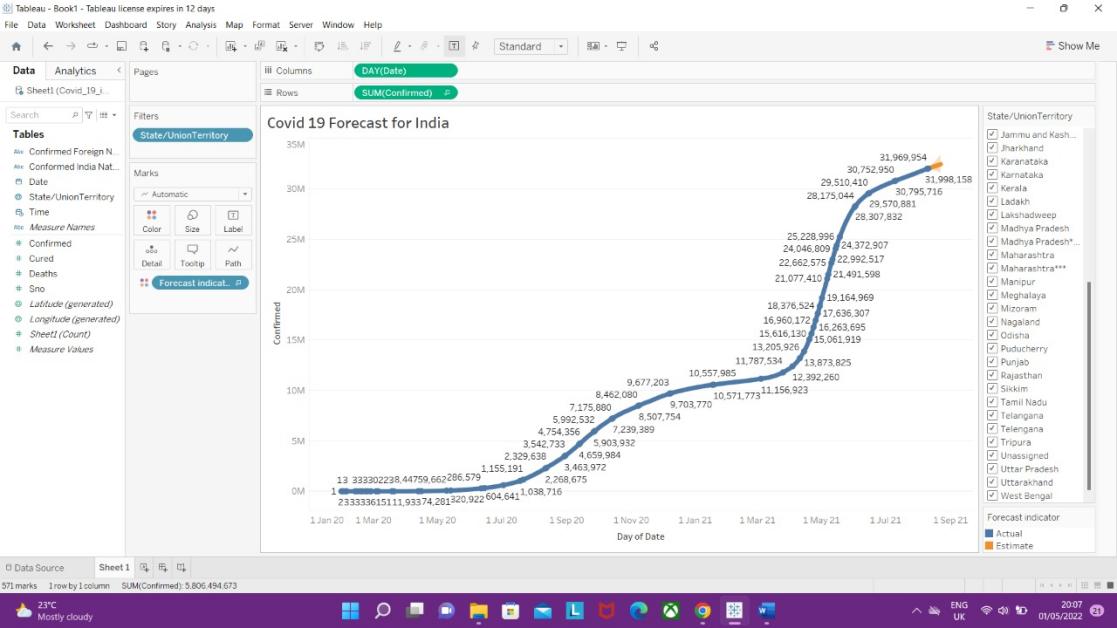
* And we have got some results of graphs after data preparation done.

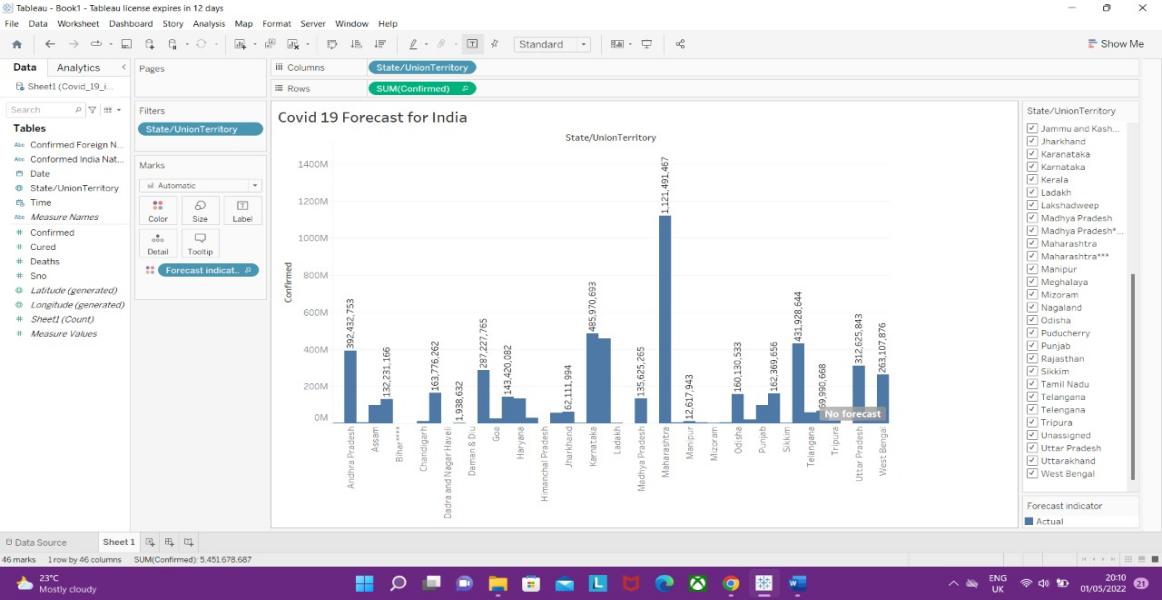
MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.

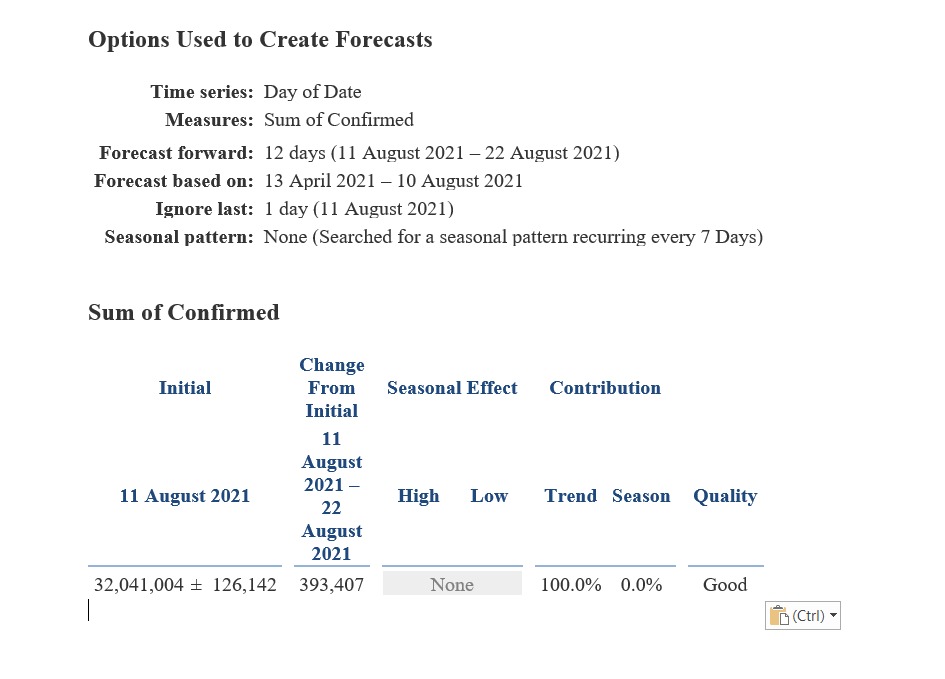
* We have created new text file in the Hadoop system.
* We are creating separate file for all the data to get stored. And we are creating a separate table for every particular aspect we need for example if we need top 6 affected districts then respective csv file and columns will be targeted.
* And then all the data will be loaded into table created by providing specific path.
* We can acquire the respective data by using hive commands.
* And all the hive commands in the java domain.

We are using some MapReduce codes in Hive i.e., in java domain. These MapReduce codes helps us to monitor and fetch out the required data we need and also it changes all the unstructured data.

**RESULT OF DATA PREPARATION:**

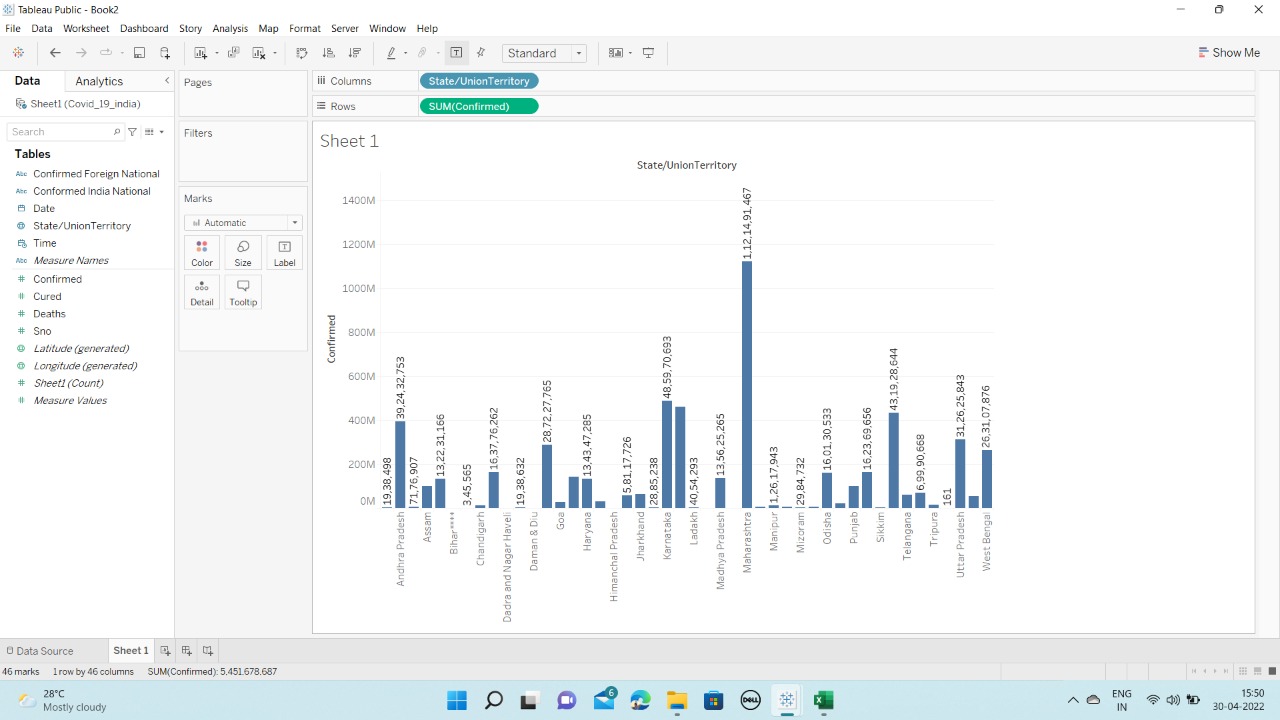


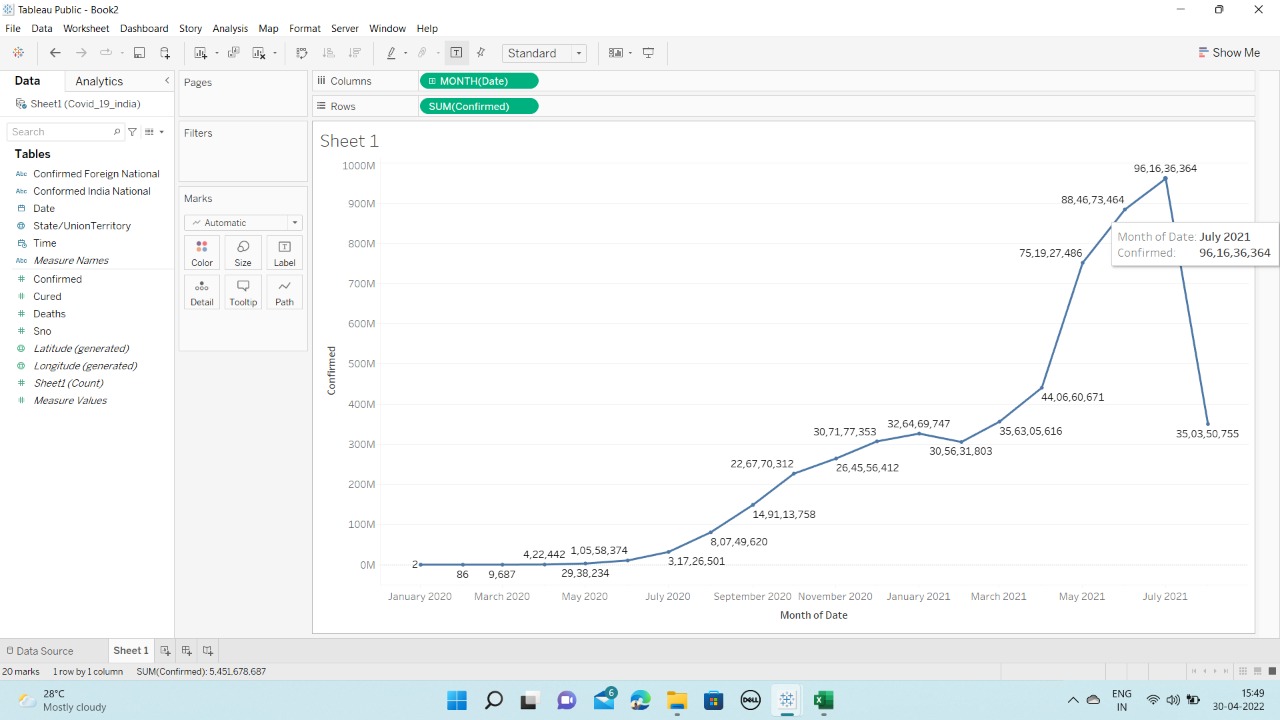




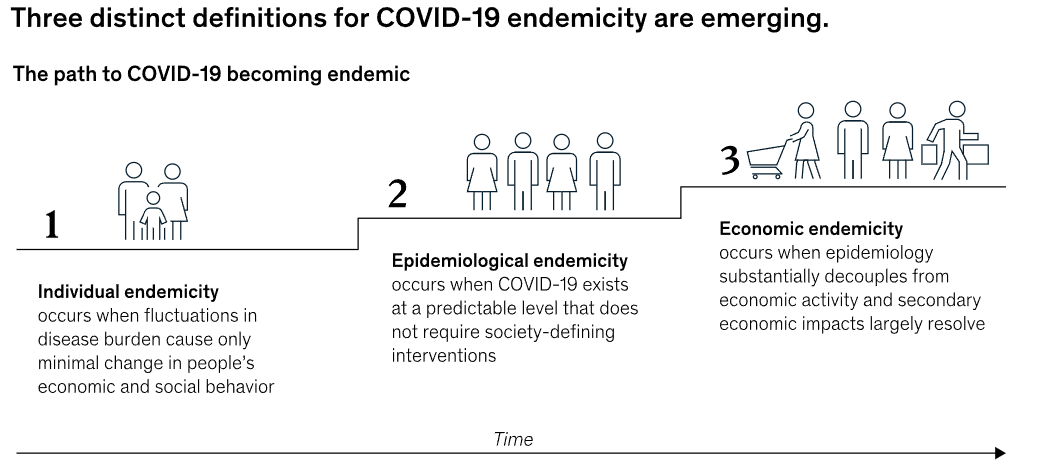
**PATIENT DEMOGRAPHIC ANALYSIS:**

we have clearly analyzed the patient data and done a quality data preparation using Tableau. And by this data preparation we can get to know about the trend in daily cases treatment.





We have predicted the saturation point for the spread by the whole data study and various analysis.



* The World Health Organization previously cautioned the world against assuming the COVID-19 pandemic is over since there are still cases spiking across the world.
* Most recently, Hong Kong has felt the brunt of an omicron wave.
* And researchers are still searching for new COVID-19 variants, which could turn the tide of the pandemic as quickly as the delta and omicron variants did.

**KEY INSIGHTS FOR NEW POLICY FORMATION:**

* Mobilize all sectors and communities to ensure that every sector of government and society takes ownership of and participates in the response and in preventing cases through hand hygiene, respiratory etiquette and individual-level physical distancing.
* Every country must put in place comprehensive public health measures to maintain a sustainable steady state of low-level or no transmission and have the surge capacity to rapidly control sporadic cases and clusters of cases to prevent community transmission from occurring.
* Develop safe and effective vaccines and therapeutics that can be delivered at scale and that are accessible based on need.

Control sporadic cases and clusters and prevent community transmission by rapidly finding and isolating all cases, providing them with appropriate care, and tracing, quarantining, and supporting all contacts.

MAPREDUCE CODES:

1.

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.BooleanWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Covid19\_1 {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

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

private Text row = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

Configuration conf = context.getConfiguration();

String w = conf.get("world\_data");

boolean world = Boolean.parseBoolean(w);

if(value.toString().contains("new\_cases"))

return;

String[] covid\_row = value.toString().split(",");

String loc\_val = covid\_row[1];

String[] date\_val = covid\_row[0].split("-");

int year = Integer.parseInt(date\_val[0]);

if((!world) && (loc\_val.equals("World") || loc\_val.equals("International")))

return;

else{

if(year != 2019){

IntWritable num\_cases = new IntWritable(Integer.parseInt(covid\_row[2]));

Text loc = new Text(covid\_row[1].toString());

context.write(loc , num\_cases);

}

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

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

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

String check = args[1].toString();

conf.set("world\_data", check);

Job job = Job.getInstance(conf, "covid 19\_1");

job.setJarByClass(Covid19\_1.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

2.

import java.io.IOException;

import java.util.StringTokenizer;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.HashMap;

import java.net.URI;

import java.io.BufferedReader;

import java.io.FileReader;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.BooleanWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Covid19\_3 {

public static class TokenizerMapper

extends Mapper<Object, Text, Text, FloatWritable>{

private Text row = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

if(value.toString().contains("new\_cases"))

return;

String[] covid\_row = value.toString().split(",");

FloatWritable num\_cases = new FloatWritable(Integer.parseInt(covid\_row[2]));

Text loc = new Text(covid\_row[1]);

context.write(loc , num\_cases);

}

}

public static class IntSumReducer

extends Reducer<Text,FloatWritable,Text,FloatWritable> {

HashMap<String, Integer> pop\_map = new HashMap<String, Integer>();

public void setup(Context context) throws IOException, InterruptedException{

URI [] cfile = context.getCacheFiles();

if (cfile != null && cfile.length > 0){

BufferedReader reader;

try{

reader = new BufferedReader(new FileReader("pop\_file"));

int i =0;

String line = "";

while((line = reader.readLine()) != null){

String[] pop\_data = line.split(",");

if(i!=0 && pop\_data.length == 5){

int j = Integer.parseInt(pop\_data[4]);

pop\_map.put(pop\_data[0].toString(), j);

}

i=1;

}

} catch(Exception e){ System.out.println(e);}

}

}

public void reduce(Text key, Iterable<FloatWritable> values,

Context context

) throws IOException, InterruptedException {

if(pop\_map.containsKey(key.toString())){

int pop = pop\_map.get(key.toString());

float sum =0;

for (FloatWritable val : values) {

sum += val.get();

}

float m = sum/pop\* 1000000;

FloatWritable result = new FloatWritable(m);

context.write(key, result);

}

else

return;

}

}

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

Configuration conf = new Configuration();

String cpath = args[1].toString();

Job job = Job.getInstance(conf, "covid 19\_3");

job.setJarByClass(Covid19\_3.class);

job.setMapperClass(TokenizerMapper.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(FloatWritable.class);

job.addCacheFile(new URI(cpath + "#pop\_file"));

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

FileOutputFormat.setOutputPath(job, new Path(args[2]));

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

}

}

3.

import java.io.IOException;

import java.util.StringTokenizer;

import java.text.ParseException;

import java.lang.RuntimeException;

import java.text.SimpleDateFormat;

import java.util.Date;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.BooleanWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Covid19\_2 {

public static class MyException

extends RuntimeException{

public MyException(String e){

super(e);

}

}

public static class TokenizerMapper

extends Mapper<Object, Text, Text, IntWritable>{

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

private Text row = new Text();

public void map(Object key, Text value, Context context

) throws IOException, InterruptedException {

Configuration conf = context.getConfiguration();

SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");

format.setLenient(false);

String[] covid\_row = value.toString().split(",");

String fd, td, d\_val;

Date fdate = new Date();

Date tdate= new Date();

Date date\_val = new Date();

Date first\_date = new Date();

Date last\_date = new Date();

try{

fd = conf.get("from\_date");

fdate = format.parse(fd);

td = conf.get("to\_date");

tdate = format.parse(td);

d\_val = covid\_row[0];

date\_val = format.parse(d\_val);

}catch(ParseException e){System.out.println(e); }

if(value.toString().contains("new\_deaths"))

return;

int start = date\_val.compareTo(fdate);

int end = date\_val.compareTo(tdate);

if(start>=0 && end<=0){

IntWritable num\_cases = new IntWritable(Integer.parseInt(covid\_row[3]));

Text loc = new Text(covid\_row[1]);

context.write(loc , num\_cases);

}

}

}

public static class IntSumReducer

extends Reducer<Text,IntWritable,Text,IntWritable> {

private IntWritable result = new IntWritable();

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

Context context

) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

result.set(sum);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

String f = args[1].toString();

String t = args[2].toString();

String fc = "2019-12-01";

String lc = "2020-04-30";

SimpleDateFormat sdformat = new SimpleDateFormat("yyyy-MM-dd");

sdformat.setLenient(false);

try {

Date dt\_f = sdformat.parse(f);

Date dt\_t = sdformat.parse(t);

Date dt\_fc = sdformat.parse(fc);

Date dt\_lc = sdformat.parse(lc);

if (dt\_f.before(dt\_fc) || dt\_t.after(dt\_lc))

throw new MyException("Invalid Date Format: Dates not in the range 2019-12-01 to 2020-04-30");

}

catch (ParseException e) { System.out.println(e);

System.exit(0);}

catch (MyException e) { System.out.println(e);

System.exit(0);}

conf.set("from\_date", f);

conf.set("to\_date", t);

Job job = Job.getInstance(conf, "covid 19\_2");

job.setJarByClass(Covid19\_2.class);

job.setMapperClass(TokenizerMapper.class);

job.setCombinerClass(IntSumReducer.class);

job.setReducerClass(IntSumReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

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

FileOutputFormat.setOutputPath(job, new Path(args[3]));

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

}

}

**CONCLUSION:**

Hence the Covid 19 Data analysis is done using Tableau [BI Tool], Hive and map reduce.