Practical No. 1

Aim: Install, configure and run Hadoop and HDFS ad explore HDFS.

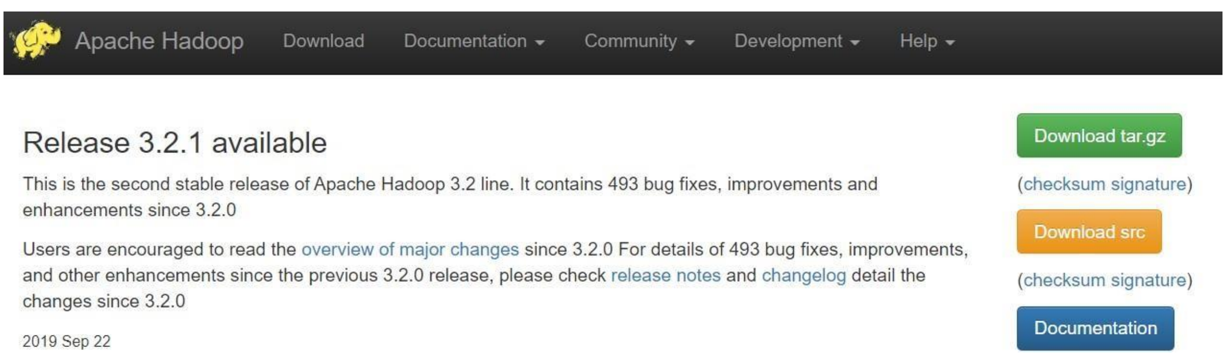
1. Prerequisites

First, we need to make sure that the following prerequisites are installed:

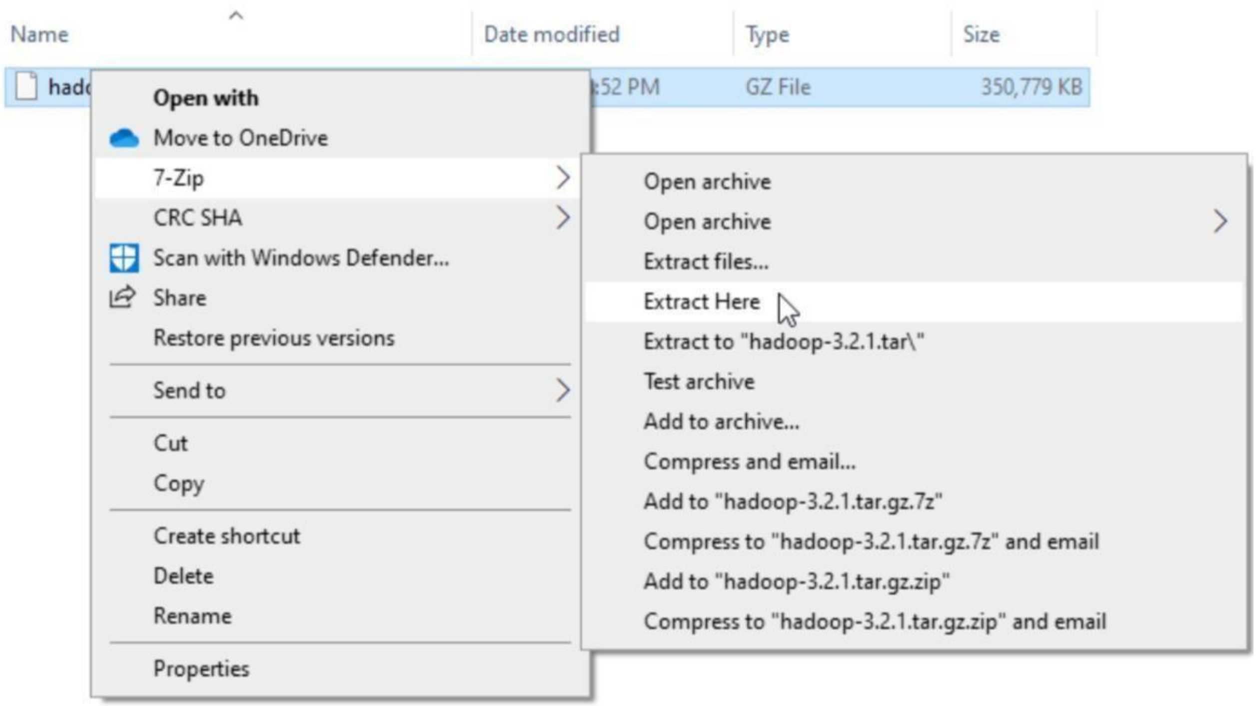
1. Java 8 runtime environment (JRE): Hadoop 3 requires a Java 8 installation. I prefer using the offline installer.
2. Java 8 development Kit (JDK)
3. To unzip downloaded Hadoop binaries, we should install 7zip.

2. Download Hadoop binaries

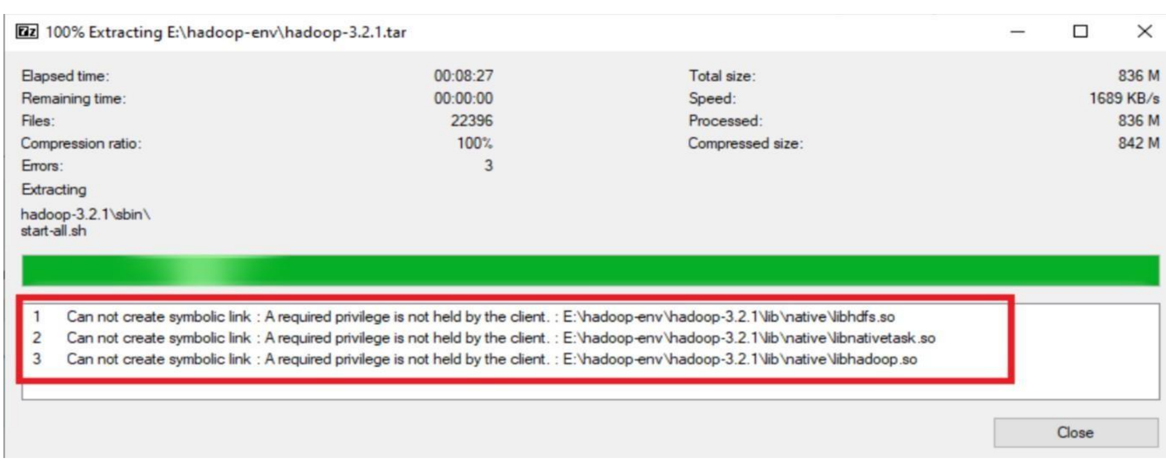
The first step is to download Hadoop binaries from the official website. The binary package size is about 342 MB.



After finishing the file download, we should unpack the package using 7zip int two steps. First, we should extract the hadoop-3.2.1.tar.gz library, and then, we should unpack the extracted tar file:



The tar file extraction may take some minutes to finish. In the end, you may see some warnings about symbolic link creation. Just ignore these warnings since they are not related to windows.



After unpacking the package, Since we are installing Hadoop 3.2.1, we should download the files located in https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.1/bin and copy them into the “hadoop-3.2.1\bin” directory.

3. Setting up environment variables

After installing Hadoop and its prerequisites, we should configure the environment variables to define Hadoop and Java default paths.

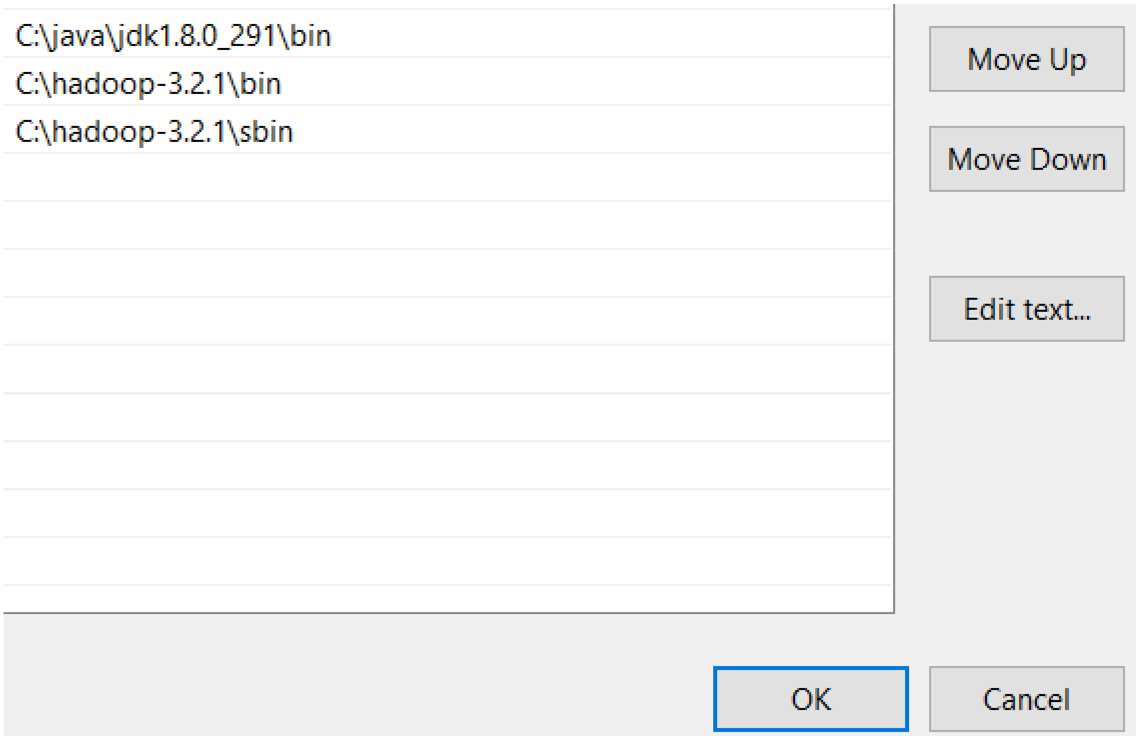
To edit environment variables, go to Control Panel > System and Security > System (or right-click > properties on My Computer icon) and click on the “Advanced system settings” link.

There are two variables to define:

1. JAVA_HOME: JDK installation folder path

2. HADOOP_HOME: Hadoop installation folder path

Variable	Value
HADOOP_HOME	C:\hadoop-3.2.1\bin
JAVA_HOME	C:\java\jdk1.8.0_291\bin



4. Configuring Hadoop cluster

There are four files we should alter to configure Hadoop cluster:

1. %HADOOP_HOME%\etc\hadoop\hdfs-site.xml
2. %HADOOP_HOME%\etc\hadoop\core-site.xml
3. %HADOOP_HOME%\etc\hadoop\mapred-site.xml
4. %HADOOP_HOME%\etc\hadoop\yarn-site.xml

4.1. HDFS site configuration

As we know, Hadoop is built using a master-slave paradigm. Before altering the HDFS configuration file, we should create a directory to store all master node (name node) data and another one to store data (data node). In this example, we created the following directories:

- E:\hadoop-env\hadoop-3.2.1\data\dfs\namenode
- E:\hadoop-env\hadoop-3.2.1\data\dfs\datanode

Now, let's open "hdfs-site.xml" file located in "%HADOOP_HOME%\etc\hadoop" directory, and we should add the following properties within the

```
<configuration>
<property>
<name>dfs.replication</name>
 <value>1</value>
 </property>
 <property>
 <name>dfs.namenode.name.dir</name>
 <value>C:\hadoop-3.2.1\data\namenode</value>
 </property>
 <property>
 <name>dfs.datanode.data.dir</name>
 <value>C:\hadoop-3.2.1\data\datanode</value>
 </property>
```

```
</configuration>
```

Note that we have set the replication factor to 1 since we are creating a single node cluster.

4.2. Core site configuration

Now, we should configure the name node URL adding the following XML code into the

```
<configuration>
 <property>
 <name>fs.defaultFS</name>
 <value>hdfs://localhost:9000</value>
 </property>
</configuration>
```

4.3. Map Reduce site configuration

Now, we should add the following XML code into the <configuration></configuration> element within “mapred-site.xml”:

```
<configuration>
<property>
 <name>mapreduce.framework.name</name>
 <value>yarn</value>
 </property>
</configuration>
```

4.4. Yarn site configuration

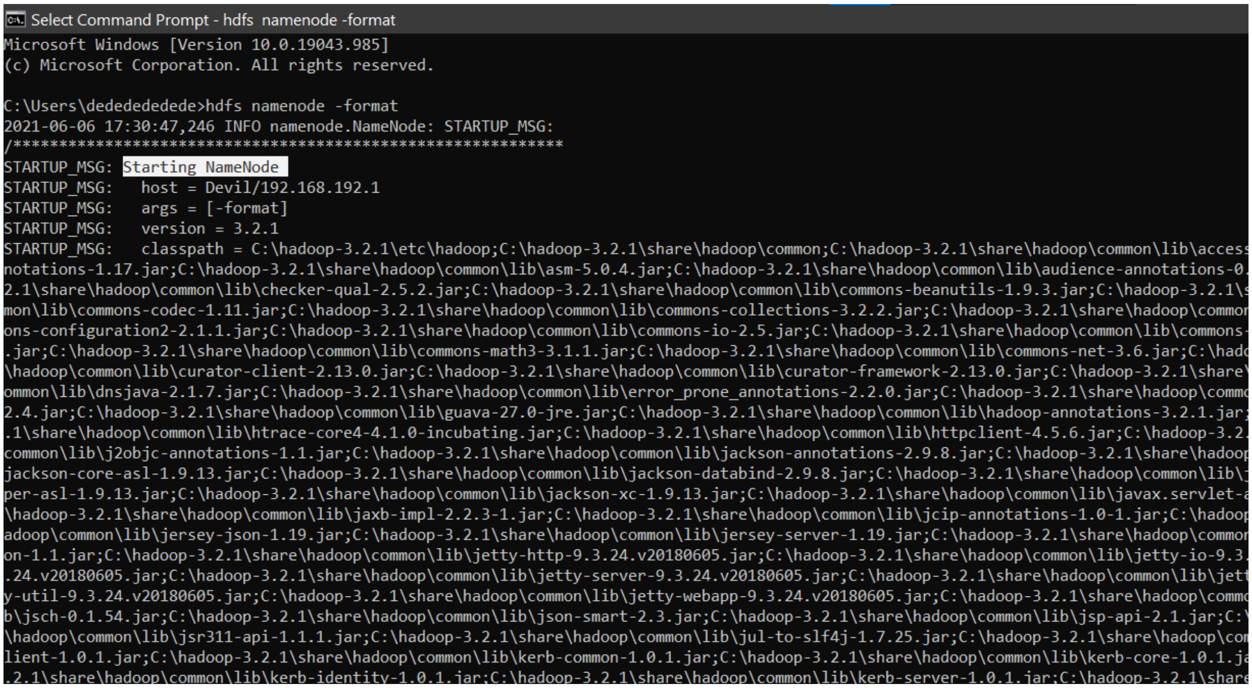
Now, we should add the following XML code into the <configuration></configuration> element within “yarn-site.xml”:

```
<configuration>

<property>

 <name>yarn.nodemanager.aux-services</name>

 <value>mapreduce_shuffle</value>

 </property>

 <property>

 <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

 </property>

</configuration>
```

5. Formatting Name node

After finishing the configuration, let’s try to format the name node using the following command:

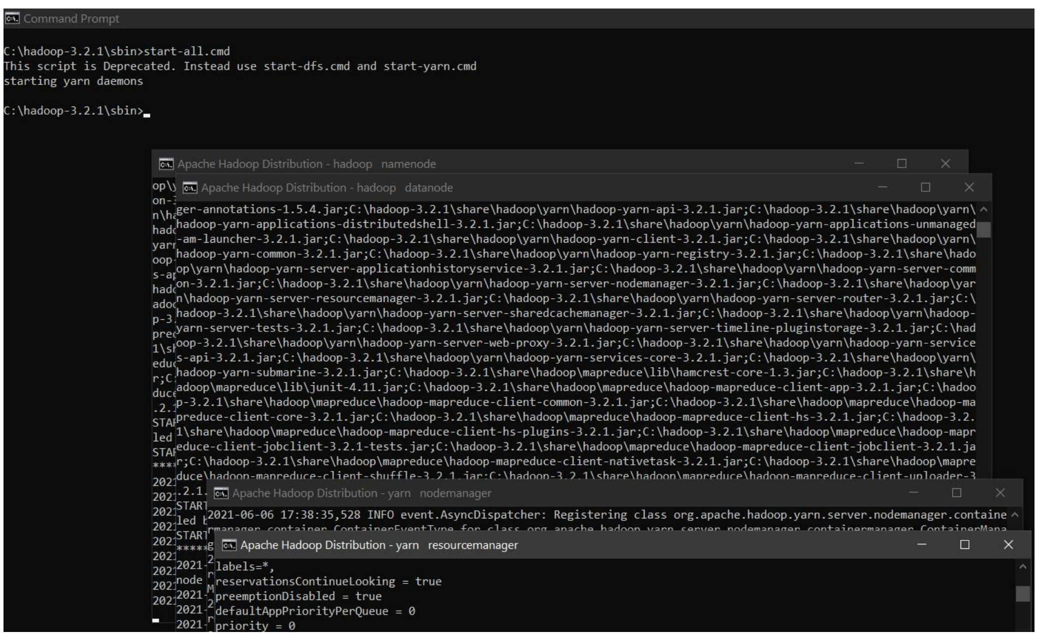
```
hdfs namenode -format
```



6. Starting Hadoop services

Now, we will open PowerShell, and navigate to “%HADOOP_HOME%\sbin” directory. Then we will run the following command to start the Hadoop nodes:

Start-all.cmd



To make sure that all services started successfully, we can run the following command:

Jps

It should display the following services:

```
C:\hadoop-3.2.1\sbin>jps
18184 DataNode
1880 ResourceManager
16428 Jps

C:\hadoop-3.2.1\sbin>
```

Practical No. 2

Aim: Implement word count / frequency programs using MapReduce.

Theory: MapReduce is a software framework for processing (large1) data sets in a distributed fashion over a several machines. The core idea behind MapReduce is mapping your data set into a collection of <key, value> pairs, and then reducing over all pairs with the same key. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*. The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

Steps for performing this practical:

Step 1: Start Hadoop
ssh localhost
/usr/local/hadoop/sbin/start-all.sh

Step2: To remove existing file from HDFS
hdfs dfs -rm /bda.txt

Step 3: Clear Output of previous run at default HDFS location
hdfs dfs -rm -r /output

Step 4: Create a text file with some words at local file system (try to include same and repeated words)
sudo nano bda.txt
(Press Ctrl+S and then Ctrl+X)

Step 5: Move bda.txt file to HDFS
hdfs dfs -put /home/hduser/bda.txt /

Step 6: Running MapReduce for wordcount file bda.txt
hadoop jar /usr/local/hadoop/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.0.jar wordcount /bda.txt /output

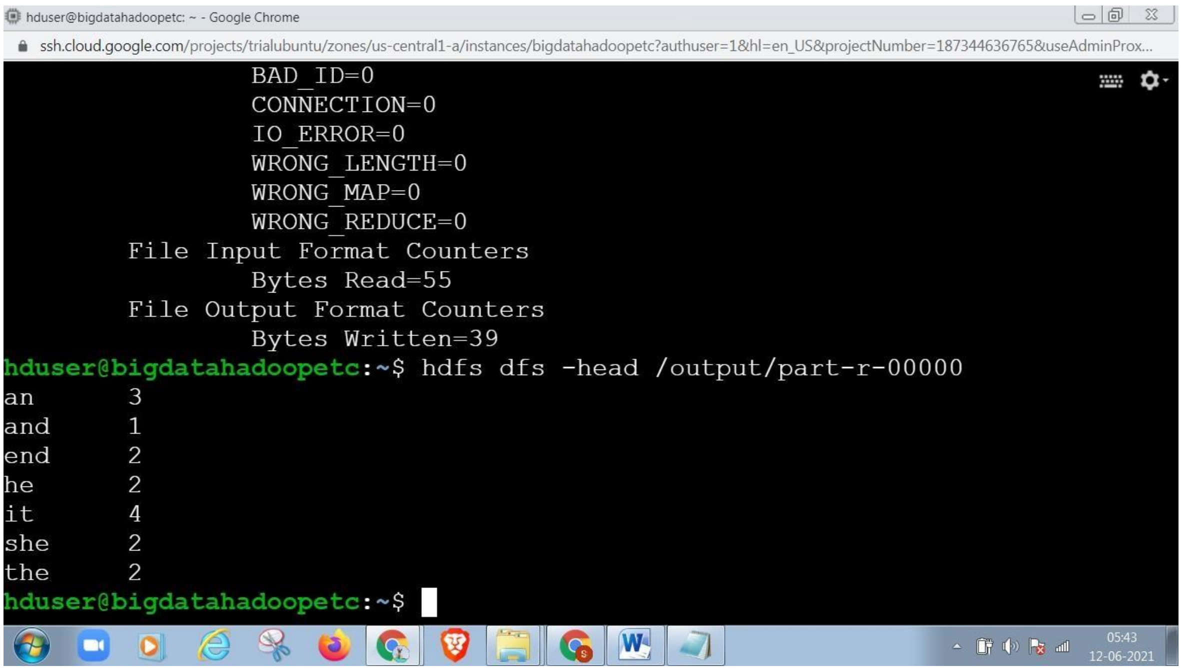
Step 7: Check/display output at default output location hdfs dfs -head /output/part-r-00000

Step 8: To get output in a .txt file in HDFS(optional)
hdfs dfs -mv /output/part-r-00000 /output/op.txt

Step 9: To get output in a .txt file in default file location(optional)
hdfs dfs -get /output/op.txt /home/hduser

Step 10: To view content of HDFS location, use following command
hdfs dfs -ls /

Output:



Practical No. 3

Aim: Implement a MapReduce program that processes a weather dataset.

Theory: MapReduce is a software framework for processing (large1) data sets in a distributed fashion over a several machines. The core idea behind MapReduce is mapping your data set into a collection of <key, value> pairs, and then reducing over all pairs with the same key. A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks. The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply *map* and *reduce* functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the *job configuration*. The Hadoop *job client* then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

Steps for performing this practical:

Step 1: Start Hadoop

ssh localhost
/usr/local/hadoop/sbin/start-all.sh

(Change your directory to the folder where you have downloaded the dataset and the jar file)

Step 2: load input dataset onto HDFS

hadoop dfs -put CRND0103-2017-AK_Fairbanks_11_NE.txt /user/hduser/

Step 3: Run MapReduce

hadoop jar temperature.jar MyMaxMin /user/hduser/CRND0103-2017-AK_Fairbanks_11_NE.txt /user/hduser/Weather-Output

Step 4: Print Output

hadoop dfs -cat /user/hduser/Weather-Output/part-r-00000

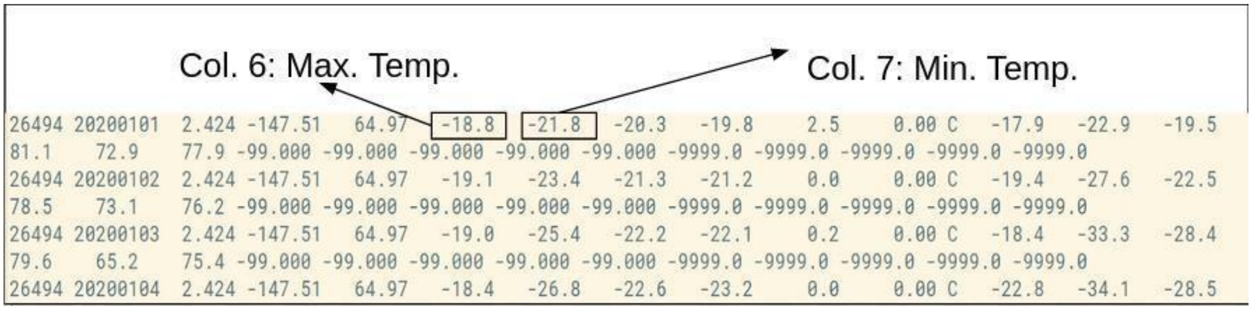
Step 5: The next time when you run this, you need to remove existing files before starting the execution

hadoop dfs -rm -f /user/hduser/CRND0103-2017-AK_Fairbanks_11_NE.txt hadoop dfs -rm -r /user/hduser/Weather-Output

Step 6: Stop Hadoop

/usr/local/hadoop/sbin/stop-all.sh

Output:




```
The Day is Cold Day :20200101   -21.8
The Day is Cold Day :20200102   -23.4
The Day is Cold Day :20200103   -25.4
The Day is Cold Day :20200104   -26.8
The Day is Cold Day :20200105   -28.8
The Day is Cold Day :20200106   -30.0
The Day is Cold Day :20200107   -31.4
The Day is Cold Day :20200108   -33.6
The Day is Cold Day :20200109   -26.6
The Day is Cold Day :20200110   -24.3
```

In the above image, you can see the top 10 results showing the cold days. The second column is a day in yyyy/mm/dd format. For Example, 20200101 means

year = 2020

month = 01

Date = 01

Practical No. 4

Aim: Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python.

Theory: HBase is a column-oriented non-relational database management system that runs on top of Hadoop Distributed File System (HDFS). HBase provides a fault-tolerant way of storing sparse data sets, which are common in many big data use cases. It is well suited for real-time data processing or random read/write access to large volumes of data.

Unlike relational database systems, HBase does not support a structured query language like SQL; in fact, HBase isn't a relational data store at all. HBase applications are written in Java™ much like a typical Apache MapReduce application. HBase does support writing applications in Apache Avro, REST and Thrift.

An HBase system is designed to scale linearly. It comprises a set of standard tables with rowsand columns, much like a traditional database. Each table must have an element defined as a primary key, and all access attempts to HBase tables must use this primary key.

Avro, as a component, supports a rich set of primitive data types including: numeric, binary data and strings; and a number of complex types including arrays, maps, enumerations and records. A sort order can also be defined for the data.

HBase relies on ZooKeeper for high-performance coordination. ZooKeeper is built into HBase, but if you're running a production cluster, it's suggested that you have a dedicated ZooKeeper cluster that's integrated with your HBase cluster.

HBase works well with Hive, a query engine for batch processing of big data, to enable fault-tolerant big data applications.

Steps for performing this practical:

Step 1: Start Hadoop
ssh localhost
/usr/local/hadoop/sbin/start-all.sh
su hduser

Step 2: Installation Part:
cd /usr/local
sudo wget https://downloads.apache.org/hbase/2.4.2/hbase-2.4.2-bin.tar.gz
sudo tar xzvf hbase-2.4.2-bin.tar.gz
sudo mv hbase-2.4.2 hbase
cd hbase/conf

Step 3: Add the following line to hbase-env.sh:
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64

Step 4: Add the following lines between <configuration> and </configuration> of hbase-site.xml:

```
<property>
        <name>hbase.rootdir</name>
        <value>file:///usr/local/hbase</value>
</property>
<property>
        <name>hbase.zookeeper.property.dataDir</name>
        <value>/usr/local/hbase/zookeeper</value>
</property>
```

Step 5: cd /usr/local

sudo chmod 777 hbase

cd /usr/local/hbase/bin

Step 6: Now to start HBase and insert data:

./start-hbase.sh

./hbase shell

Inside the hbase shell:

```
create 'test', 'cf'

put 'test', 'row1', 'cf:a', 'value1'
put 'test', 'row2', 'cf:b', 'value2'
put 'test', 'row3', 'cf:c', 'value3'
scan 'test'
exit
```

./stop-hbase.sh

Step 7: Now we shall access this data using Python:

pip3 install happybase

Step 8: Python data manipulation part:

cd /usr/local/hbase/bin

Step 9: (make sure that you start the thrift server first and then the HBase, and while closing you stop HBase first and then thrift server)

```
./hbase-daemon.sh start thrift
./start-hbase.sh
python3
import happybase as hb
conn=hb.Connection('127.0.0.1', 9090)
conn.table('test').row('row1')
conn.table('test').row('row2')
conn.table('test').row('row3')
exit()
./stop-hbase.sh
./hbase-daemon.sh stop thrift
```

Practical No. 5

Aim: Implement the program in practical 4 using Pig.

Theory: Pig is a high-level scripting language that is used with Apache Hadoop. Pig enables data workers to write complex data transformations without knowing Java. Pig's simple SQL-like scripting language is called Pig Latin, and appeals to developers already familiar with scripting languages and SQL. Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject).

Steps for performing this practical:

Step 1:
su hduser
cd /usr/local
sudo wget https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz
sudo tar -xvzf pig-0.17.0.tar.gz
sudo mv pig-0.17.0 pig
cd /home/hduser

Step 2: In the .bashrc file, add these lines very carefully:
export PATH=$PATH:$HADOOP_HOME/bin
export PATH=$PATH:$HADOOP_HOME/sbin
export PATH=$PATH:/usr/local/pig/bin
export PIG_HOME=/usr/local/pig
export PIG_CLASSPATH=/usr/local/hadoop/etc/hadoop

Step 3: Then run command:
source .bashrc
Pig is ready to install

Step 4: Create a file called customers.txt and save it in any directory

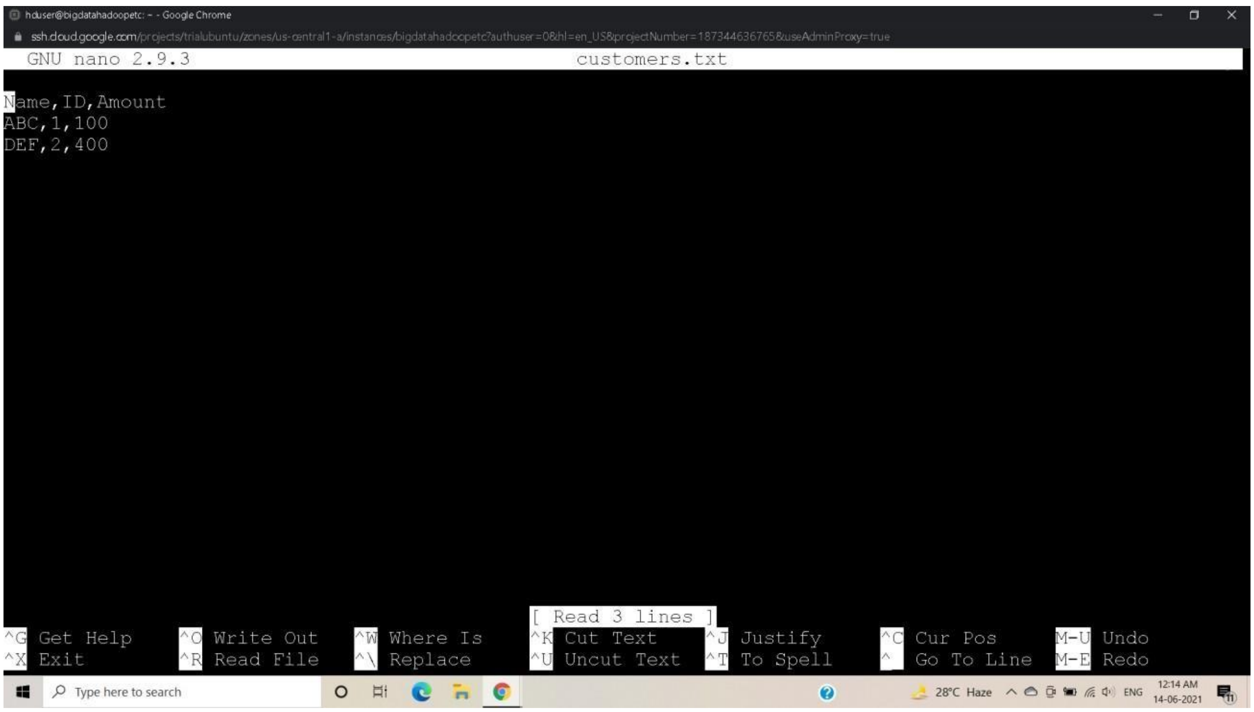
1. Running in local mode (pig can access data present only in local file system, eg the customer file)
   Start pig using following command:
   pig -x local
   customers = LOAD 'customers.txt' USING PigStorage(',');
   dump customers;
   quit;

2. Running in HDFS mode (pig can access data on HDFS)

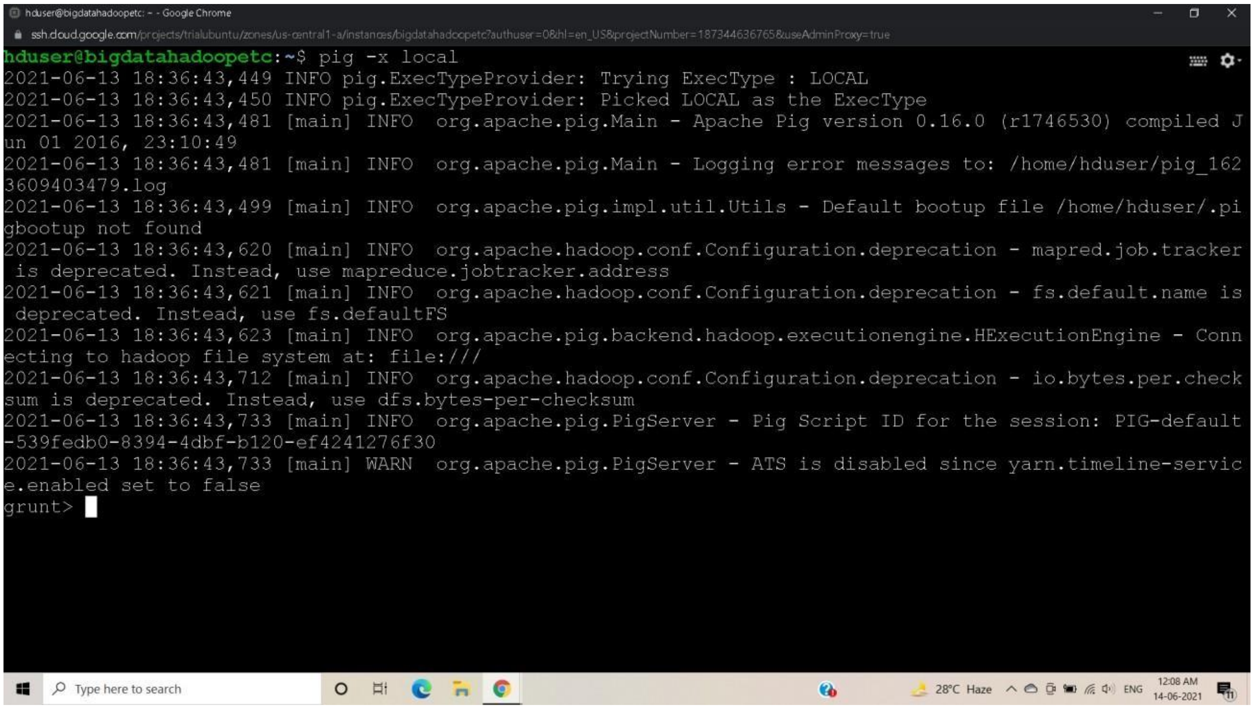
First we need to move customers.txt to HDFS
For that start hadoop,
Check via jps if you want
hdfs dfs -put ./customers.txt /user/hduser/
Start pig using pig command
customers = LOAD 'hdfs://localhost:54310/user/hduser/customers.txt' USING PigStorage(',');
dump customers;
quit;

Output:

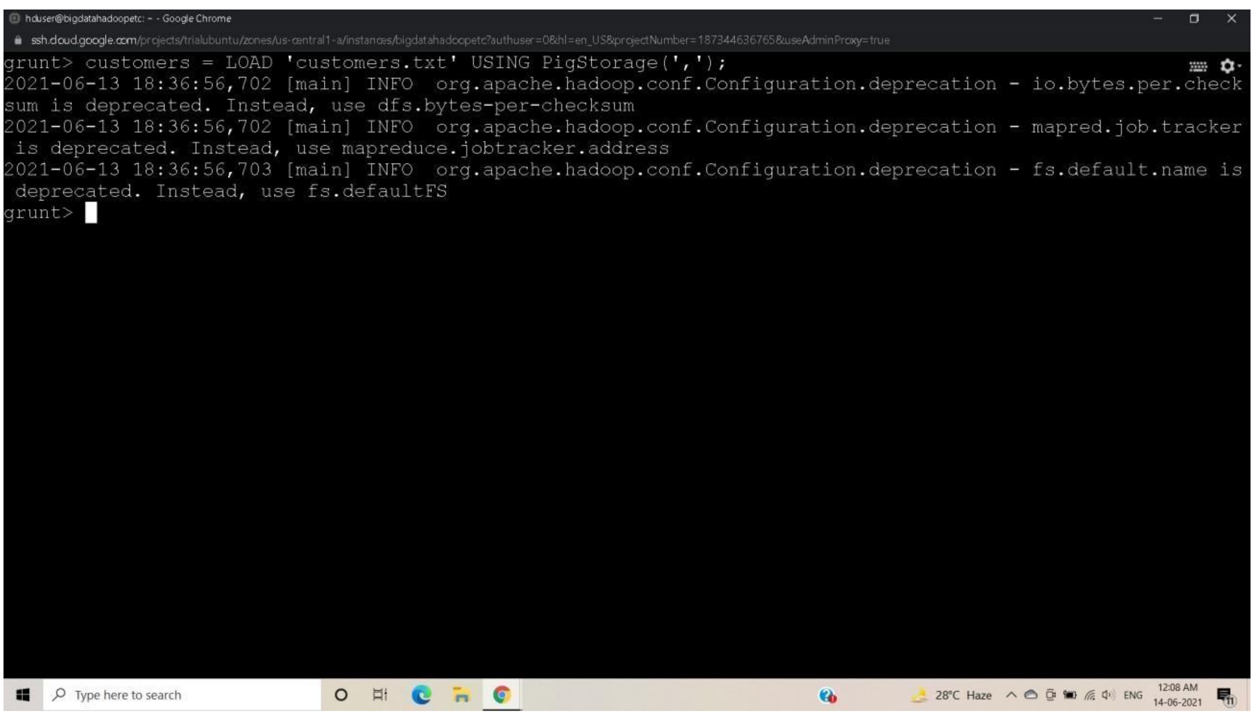
Customers.txt file



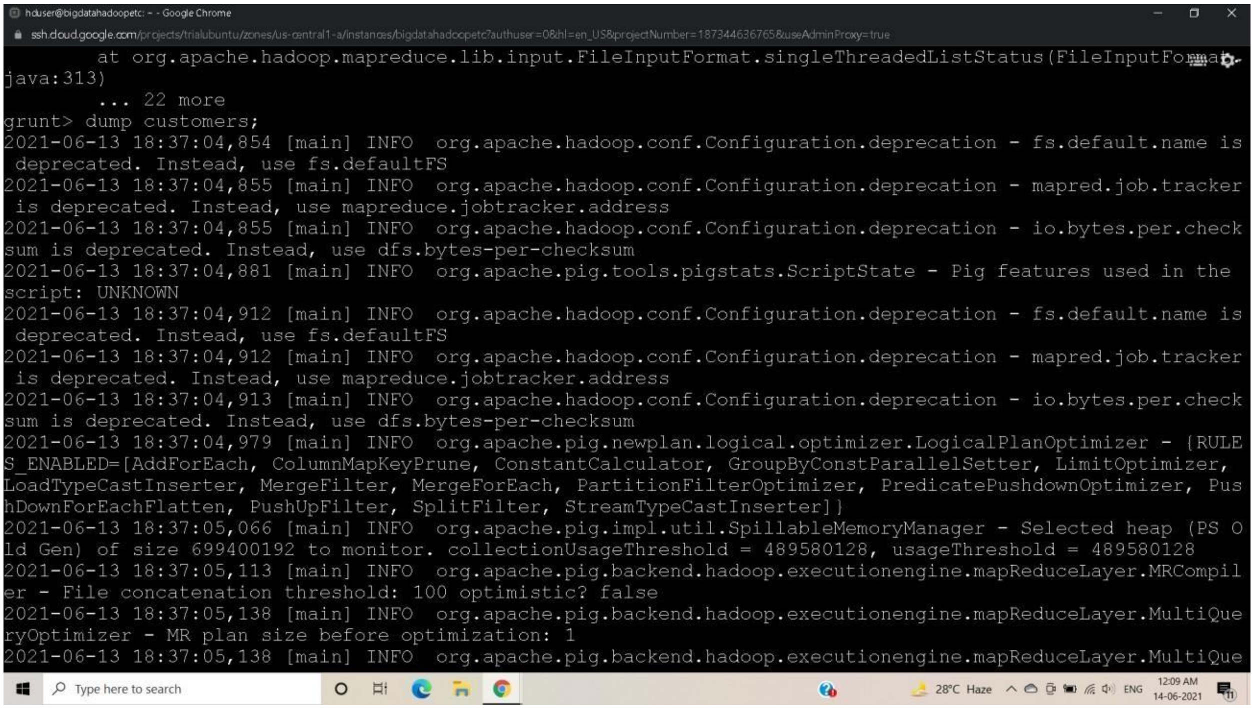
Opening pig in local mode



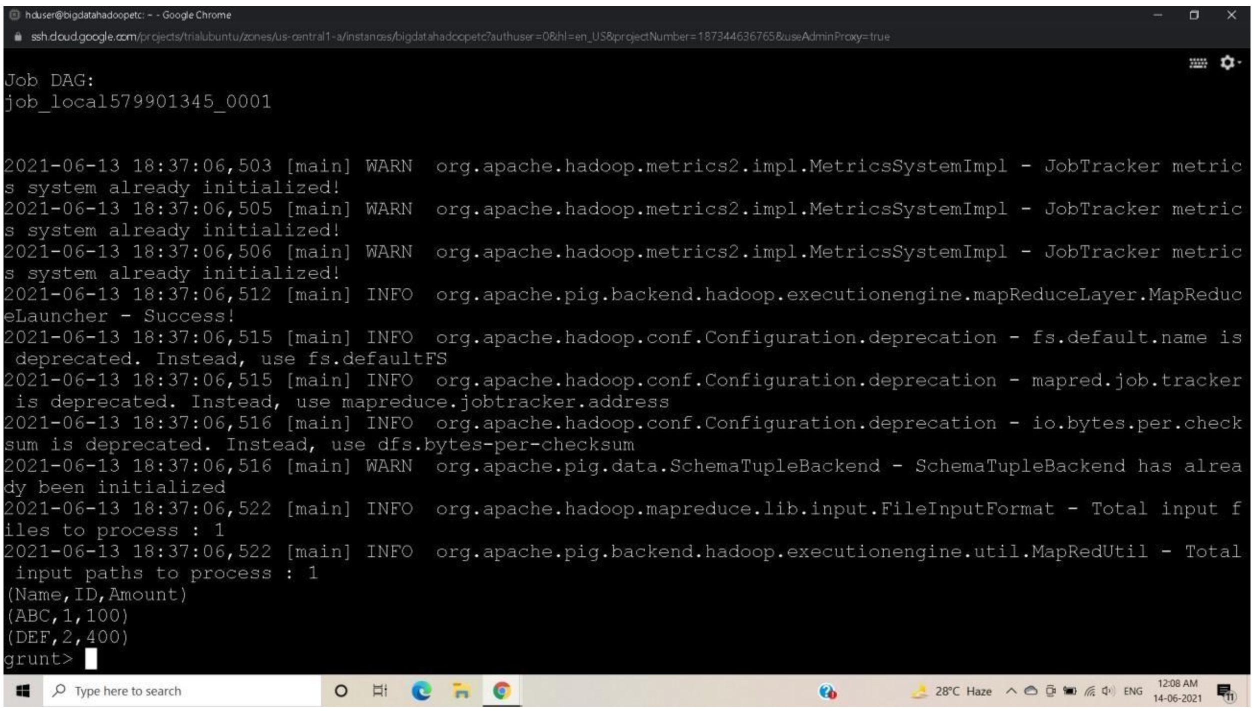
Loading customers.txt file



Dumping customers onto the screen

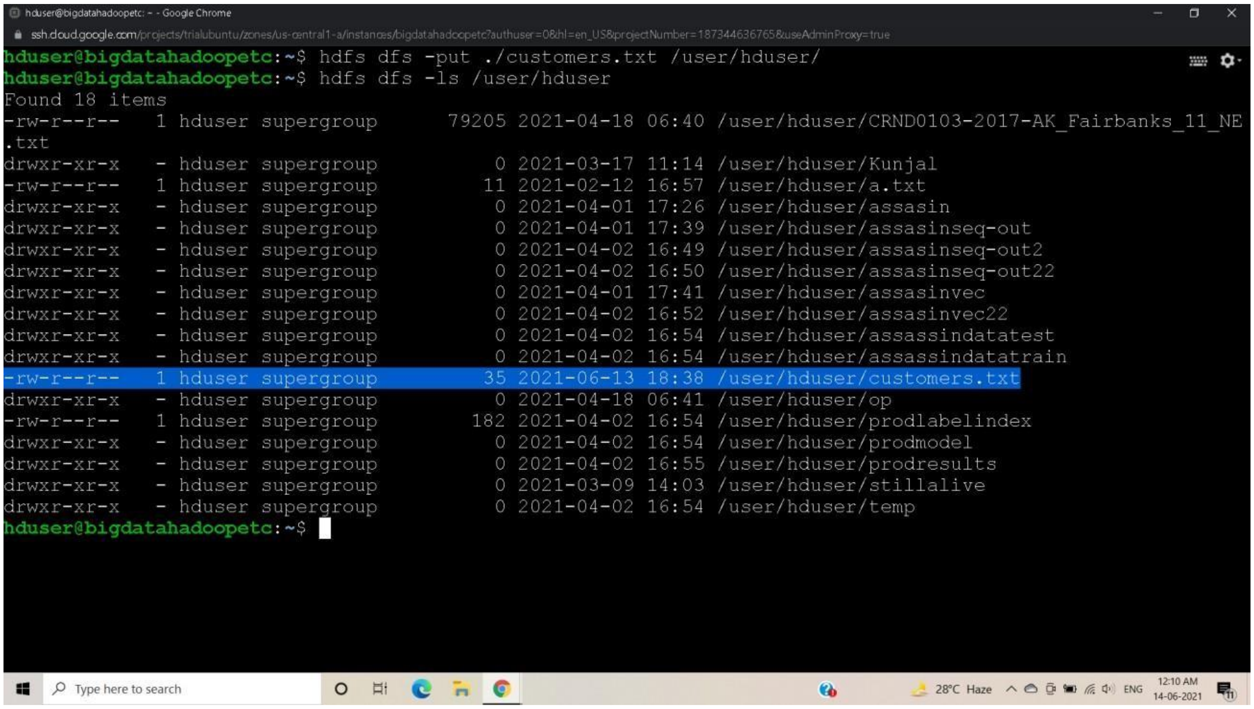


```
        at org.apache.hadoop.mapreduce.lib.input.FileInputFormat.singleThreadedListStatus(FileInputFormat.
java:313)
        ... 22 more
grunt> dump customers;
2021-06-13 18:37:04,854 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is
 deprecated. Instead, use fs.defaultFS
2021-06-13 18:37:04,855 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker
 is deprecated. Instead, use mapreduce.jobtracker.address
2021-06-13 18:37:04,855 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.check
sum is deprecated. Instead, use dfs.bytes-per-checksum
2021-06-13 18:37:04,881 [main] INFO  org.apache.pig.tools.pigstats.ScriptState - Pig features used in the
script: UNKNOWN
2021-06-13 18:37:04,912 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is
 deprecated. Instead, use fs.defaultFS
2021-06-13 18:37:04,912 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker
 is deprecated. Instead, use mapreduce.jobtracker.address
2021-06-13 18:37:04,913 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.check
sum is deprecated. Instead, use dfs.bytes-per-checksum
2021-06-13 18:37:04,979 [main] INFO  org.apache.pig.newplan.logical.optimizer.LogicalPlanOptimizer - {RULE
S_ENABLED=[AddForEach, ColumnMapKeyPrune, ConstantCalculator, GroupByConstParallelSetter, LimitOptimizer,
LoadTypeCastInserter, MergeFilter, MergeForEach, PartitionFilterOptimizer, PredicatePushdownOptimizer, Pus
hDownForEachFlatten, PushUpFilter, SplitFilter, StreamTypeCastInserter]}
2021-06-13 18:37:05,066 [main] INFO  org.apache.pig.impl.util.SpillableMemoryManager - Selected heap (PS O
ld Gen) of size 699400192 to monitor. collectionUsageThreshold = 489580128, usageThreshold = 489580128
2021-06-13 18:37:05,113 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MRCompil
er - File concatenation threshold: 100 optimistic? false
2021-06-13 18:37:05,138 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQue
ryOptimizer - MR plan size before optimization: 1
2021-06-13 18:37:05,138 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MultiQue
```

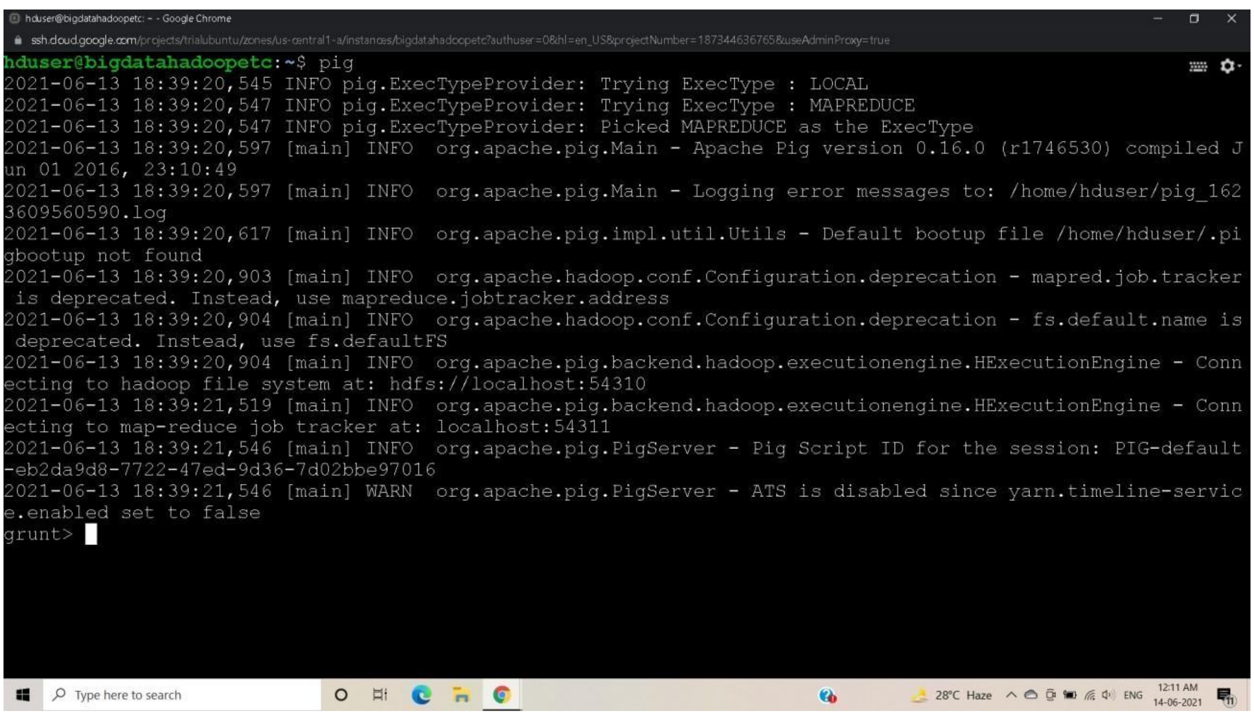


```
Job DAG:
job_local579901345_0001


2021-06-13 18:37:06,503 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metric
s system already initialized!
2021-06-13 18:37:06,505 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metric
s system already initialized!
2021-06-13 18:37:06,506 [main] WARN  org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metric
s system already initialized!
2021-06-13 18:37:06,512 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduc
eLauncher - Success!
2021-06-13 18:37:06,515 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is
 deprecated. Instead, use fs.defaultFS
2021-06-13 18:37:06,515 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker
 is deprecated. Instead, use mapreduce.jobtracker.address
2021-06-13 18:37:06,516 [main] INFO  org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.check
sum is deprecated. Instead, use dfs.bytes-per-checksum
2021-06-13 18:37:06,516 [main] WARN  org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has alrea
dy been initialized
2021-06-13 18:37:06,522 [main] INFO  org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input f
iles to process : 1
2021-06-13 18:37:06,522 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total
 input paths to process : 1
(Name,ID,Amount)
(ABC,1,100)
(DEF,2,400)
grunt>
```

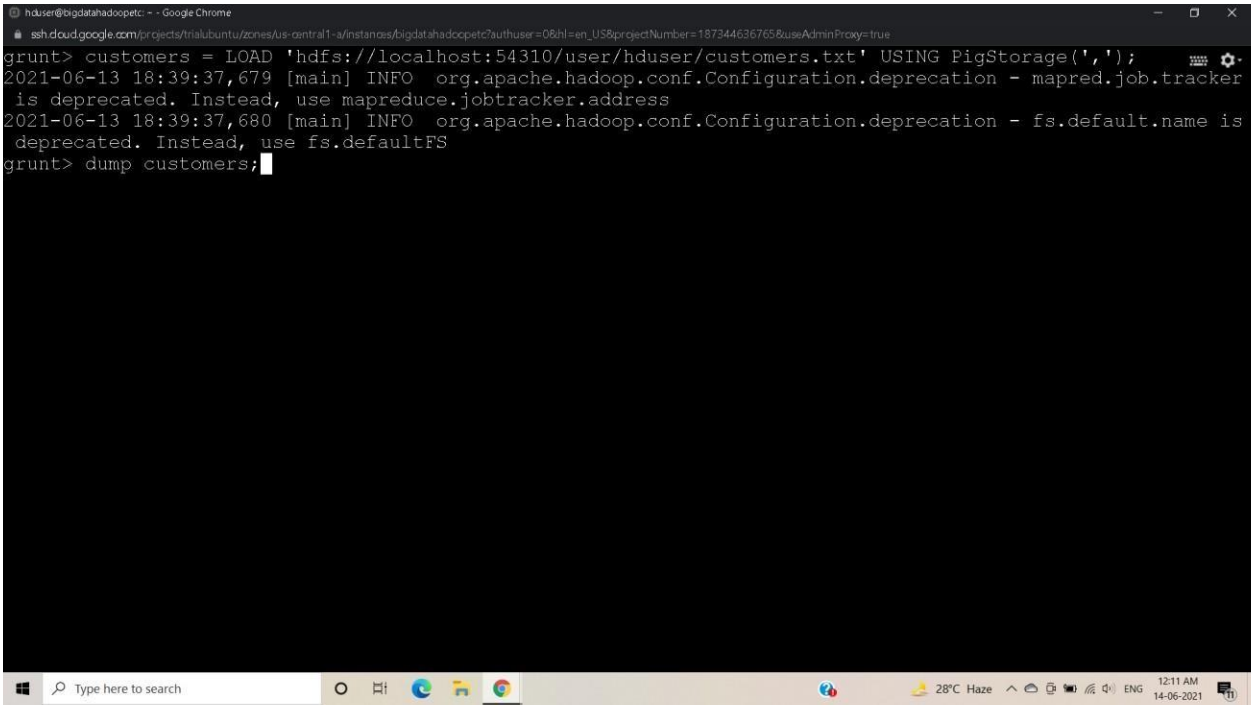
Putting customers.txt onto HDFS

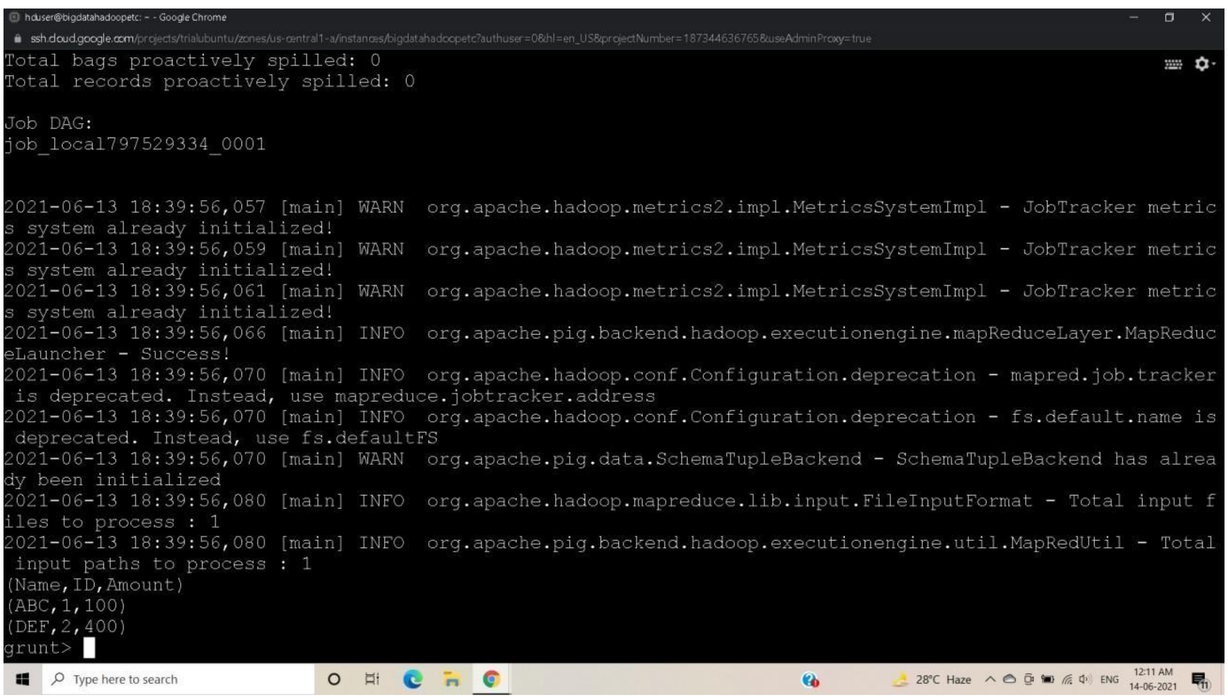


Starting pig in HDFS mode



Loading customers.txt file and Dumping customers onto the screen


grunt> customers = LOAD 'hdfs://localhost:54310/user/hduser/customers.txt' USING PigStorage(',');
2021-06-13 18:39:37,679 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2021-06-13 18:39:37,680 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
grunt> dump customers;


Total bags proactively spilled: 0
Total records proactively spilled: 0
Job DAG:
job_local797529334_0001
2021-06-13 18:39:56,057 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2021-06-13 18:39:56,059 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2021-06-13 18:39:56,061 [main] WARN org.apache.hadoop.metrics2.impl.MetricsSystemImpl - JobTracker metrics system already initialized!
2021-06-13 18:39:56,066 [main] INFO org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2021-06-13 18:39:56,070 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2021-06-13 18:39:56,070 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - fs.default.name is deprecated. Instead, use fs.defaultFS
2021-06-13 18:39:56,070 [main] WARN org.apache.pig.data.SchemaTupleBackend - SchemaTupleBackend has already been initialized
2021-06-13 18:39:56,080 [main] INFO org.apache.hadoop.mapreduce.lib.input.FileInputFormat - Total input files to process : 1
2021-06-13 18:39:56,080 [main] INFO org.apache.pig.backend.hadoop.executionengine.util.MapRedUtil - Total input paths to process : 1
(Name,ID,Amount)
(ABC,1,100)
(DEF,2,400)
grunt>

Practical No. 6

Aim: Configure the Hive and implement the application in Hive.

Theory: 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.

Steps for performing this practical:

Step 1: First we need to start Hadoop:
su hduser
ssh localhost
start-all.sh

Step 2: Now to uninstall existing version:
cd /usr/local
sudo rm -r hive
hdfs dfs -rm -r -f /tmp
hdfs dfs -rm -r /user/hive

Step 3: Now we download and setup hive:
cd /usr/local
sudo wget https://apachemirror.wuchna.com/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
sudo tar -xvzf apache-hive-3.1.2-bin.tar.gz
sudo mv apache-hive-3.1.2-bin hive
sudo chmod 777 hive
cd /home/hduser

Step 4: In the .bashrc file, add these lines very carefully:
export HIVE_HOME=/usr/local/hive
export PATH=$PATH:$HIVE_HOME/bin

Step 5: Then run command:
source .bashrc

Step 6: Then go to hive bin directory by command:
cd /usr/local/hive/bin

Step 7: Add the following line to hive-config.sh
export HADOOP_HOME=/usr/local/hadoop

Step 8: Hive is installed now, but you need to first create some directories in HDFS for Hive to store its data.
hdfs dfs -mkdir /tmp

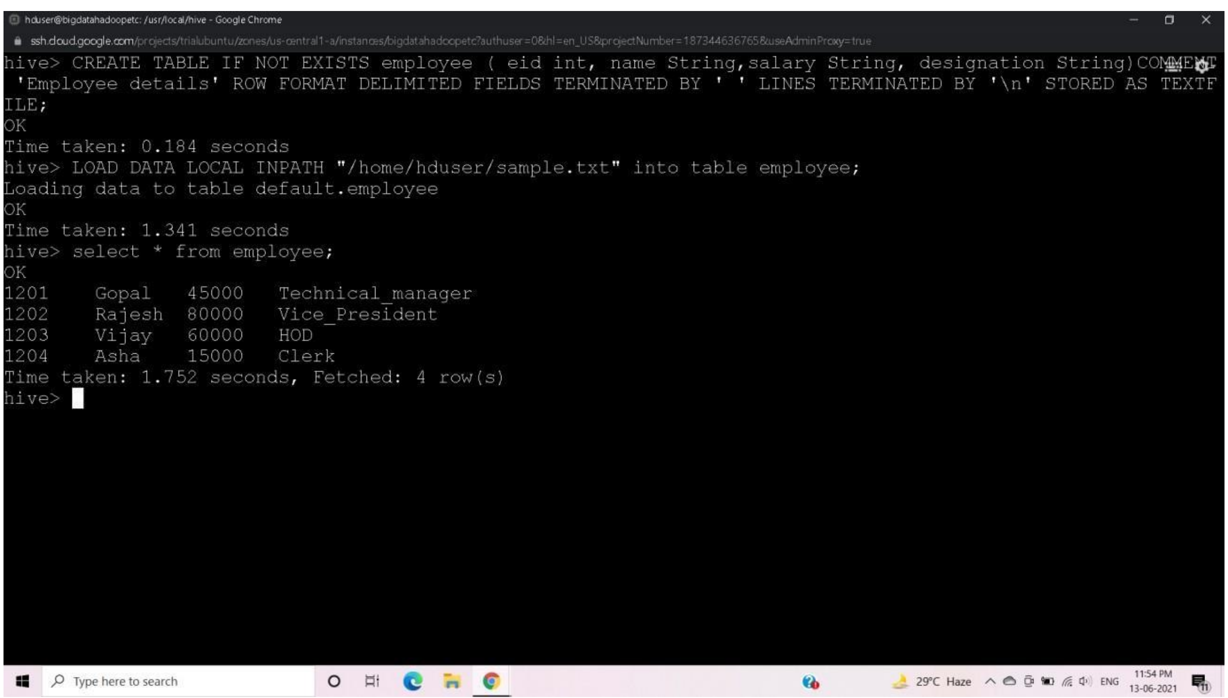
hdfs dfs -chmod g+w /tmp
hdfs dfs -mkdir -p /user/hive/warehouse
hdfs dfs -chmod g+w /user/hive/warehouse
sudo chmod 777 /usr/local/hive

Step 9: Now we need to initialize derby database.
cd $HIVE_HOME
$HIVE_HOME/bin/schematool -initSchema -dbType derby

Step 10: Start hive shell by typing ‘hive’:
You may face error like ‘NoSuchMethodFound’
To solve it:
sudo cp $HADOOP_HOME/share/hadoop/common/lib/guava-27.0-jre.jar /usr/local/hive/lib/
sudo rm /usr/local/hive/lib/guava-19.0.jar
cd $HIVE_HOME
$HIVE_HOME/bin/schematool -initSchema -dbType derby
Start hive shell by typing ‘hive’.

Step 11:
CREATE TABLE IF NOT EXISTS employee (eid int, name String,salary String, designation String)COMMENT 'Employee details' ROW FORMAT DELIMITED FIELDS
TERMINATED BY ' ' LINES TERMINATED BY '\n' STORED AS TEXTFILE;
LOAD DATA LOCAL INPATH "/home/hduser/sample.txt" into table employee;
select * from employee;
Then stop hadoop and close.

Output:



Practical No. 7

Aim: Write a program to illustrate the working Jaql.

Theory:

- JAQL, pronounced as "jackal", is a functional, declarative programming language usedfor querying large volumes of structured, semi-structured and unstructured data.
- JAQL is one of the languages that help to abstract complexities of MapReduce programming framework within Hadoop.
- JAQL's data model is based on JSON Query Language. It can handle deeply nested semi-structured and heterogeneous data.

JAQL PROPERTIES:

Name	**Type**	**Required**	**Description**
datasource	String/object	Yes	States the name of datasource which has to be queried.
metadata	object[]	Yes	Contains an array of JAQL elements (dimension/measure)
format	string	No	States the expected return type; CSV or JSON
count	number	No	Cuts the query result by setting the row offset and row count

DIMENSION PROPERTIES

Name	Type	Required	Description
dim	string	Yes	The dimension name
level	string	No	States the level of data in dim
filter	object	No	Defines element's filter

AGGREGATIONS:

Name	**Type**	**Required**	**Description**
dim	string	Yes	The dimension name
level	string	No	States the date level in a Date dimension
agg	string	Yes	Defines the measure aggregation over the dimension defined in the dim property

filter	object	No	Defines the element's filter

REQUIREMENTS:

Software requirements: Sisense

Hardware requirements: Minimum 8GB RAM.

Dataset: Sample Healthcare

URL: http://localhost:8081/app/jaqleditor

Data: http://localhost:8081/app/data/

WORKING:

Query 1: To find the names of all Doctors

Code:

```
{
	"datasource": "Sample Healthcare",
	"metadata": [
		{
			"dim":"[Doctors.Name]"

		}
	]
}
```

Explanation: In Data source field the name of the dataset has to be mentioned, in our case, the data set is “Sample Healthcare”.

Metadata is the extra information about the data. In dim section we enter the column name followed by the property name.

The above jaql query is just like the below SQL query :

Select name from Sample Healthcare

where entity name=”Doctors”

On Execute, following results were displayed,

The system had extracted the name of all Doctors in the Sample Healthcare dataset.

Can be cross-check from the path –C:\Program Files\Sisense\Samples\Sources\Sample Healthcare\Doctors

Output:

```
{
    "headers": [
        "Name"
    ],
    "metadata": [
        {
            "dim": "[Doctors.Name]"
        }
    ],
    "datasource": {
        "fullname": "LocalHost/Sample Healthcare",
        "revisionId": "854f754c-9f35-4548-aaf0-b839940bcd3e"
    },
    "values": [
        [
            {
                "data": "Aline",
                "text": "Aline"
            }
        ],
        [
            {
                "data": "Cassady",
                "text": "Cassady"
            }
        ],
        [
            {
                "data": "Donna",
                "text": "Donna"
            }
        ],
        [
            {
                "data": "Eleanor",
                "text": "Eleanor"
            }
        ],
        [
            {
                "data": "Imogene",
```


Query 2: To find the count of specialty offered by the doctors using aggregation "*agg*".

Code:

```
{
        "datasource": "Sample Healthcare",
        "metadata": [
                {
                        "dim":"[Doctors.Specialty]",
                        "agg":"count"
                }
        ]
}
```

Explanation: This query returns the count of the specialties present in the dataset. The output returned-6. When cross-checked with dataset, there also 6 specialties are present, namely-Pediatrics, Oncology, Cardiology, Surgeon, Emergency Room, Neurology.

Output:

Query 3: Find the name of doctors with a specialties using filter.
Code:

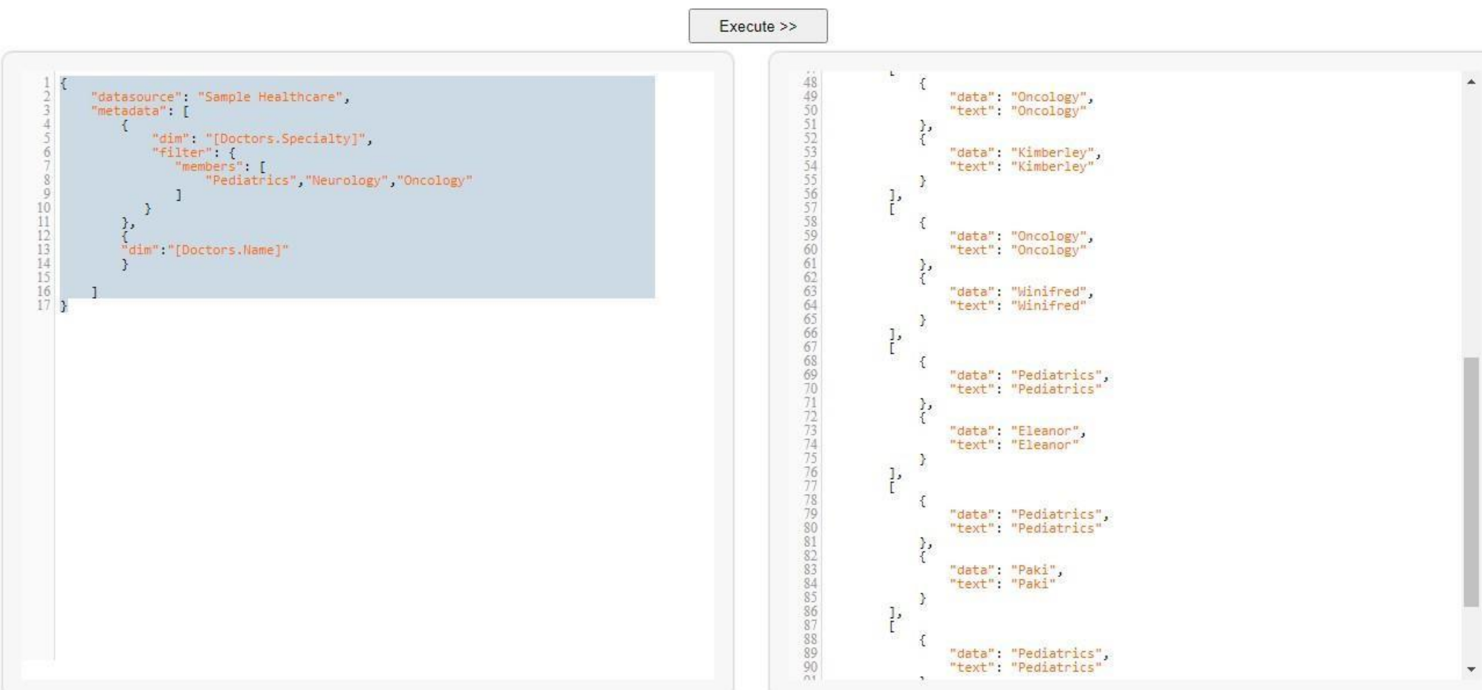
```
{
	"datasource": "Sample Healthcare",
	"metadata": [
		{
			"dim": "[Doctors.Specialty]",
			"filter": {
   "members": [
      "Pediatrics","Neurology","Oncology"
   ]
 }
		},
		{
		"dim":"[Doctors.Name]"
		}

	]
}
```

Explanation: This query searches only for specific values "Pediatrics","Neurology","Oncology" mentioned in filter. The output will be the names of the doctors if having any of the specialties above. Basically, filter works like the where clause in SQL, thereby helping to search for specific values.

In output notice “Kimberley” with data above-“Oncology”. When checked in Sample Healthcare dataset, same is the speciality.

ID	Name	Surname	Specialty	Division_ID	
1	Eleanor	Freeman	Pediatrics	2	1
2	Kimberley	Cortez	Oncology	1	2
3	Uma	Conley	Pediatrics	2	3
4	Imogene	Fletcher	Cardiolog	3	4
5	Winifred	Sharp	Oncology	1	5
6	Porter	Ware	Surgeon	5	6
7	Janna	Penningtc	Cardiolog	3	7
8	Paki	Zimmerm	Pediatrics	2	8
9	Jermaine	Vaughn	Emergenc	4	9

Output:



Practical No. 8

PART-A

Aim: Decision Tree classification.

Theory: A Decision Tree is a simple representation for classifying examples. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter. Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

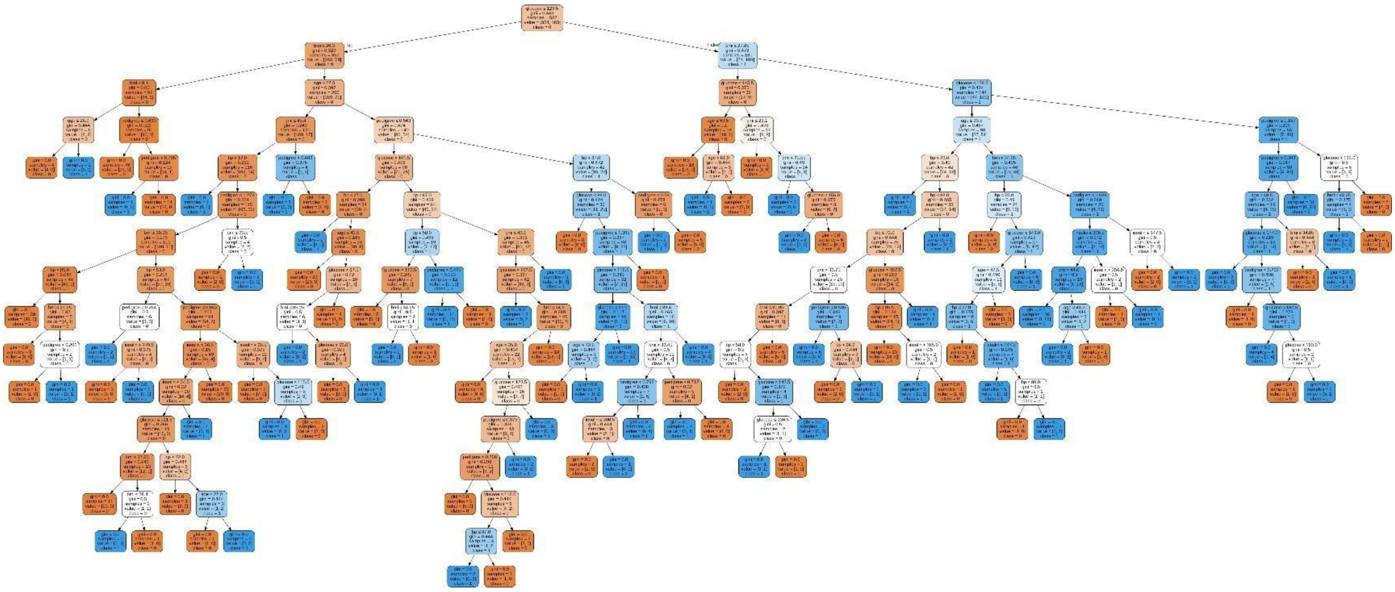
Code:

```
!pip install graphviz
!pip install pydotplus

import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

# load dataset
pima = pd.read_csv("diabetes.csv")
col_names = pima.columns

#split dataset in features and target variable
feature_cols = ['insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70%
training and 30% test

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.658008658008658

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from  IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes1.png')
Image(graph.create_png())
```

Output:



PART-B.

Aim: Implement SVM classification techniques.

Theory: Support Vector Machine (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well. Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).

Code:

```
# !pip install graphviz

# !pip install pydotplus

import pandas as pd
from sklearn.svm import SVC # Import SVM Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

# load dataset
pima = pd.read_csv("diabetes.csv")
col_names = pima.columns

pima.head()
```

Output:

	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
2	183	64	0	0	23.3	0.672	32	1
3	89	66	23	94	28.1	0.167	21	0
4	137	40	35	168	43.1	2.288	33	1

```
#split dataset in features and target variable
feature_cols = ['insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=3) # 70%
training and 30% test

clf = SVC()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
Accuracy: 0.696969696969697

clf = SVC(kernel="linear")

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:

Accuracy: 0.7316017316017316

Practical No. 9

PART-A.

Aim: REGRESSION MODEL Import a data from web storage. Name the dataset and now do Logistic Regression to find out relation between variables that are affecting the admission of a student in an institute based on his or her GRE score, GPA obtained and rank of the student. Also check the model is fit or not. require (foreign), require (MASS).

Theory: Regression analysis is a form of predictive modelling technique which investigates the relationship between a **dependent** (target) and **independent variable (s)** (predictor). This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression. Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized.

Code:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression # Import LogisticRegression
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation

gre = pd.read_csv("binary.csv")
col_names = gre.columns

gre.head()
```

Output:

	admit	gre	gpa	rank
0	0	380	3.61	3
1	1	660	3.67	3
2	1	800	4.00	1
3	1	640	3.19	4
4	0	520	2.93	4

```
#split dataset in features and target variable
feature_cols = ['gre', 'gpa', 'rank']
X = gre[feature_cols]
y=gre.admit
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70%
training and 30% test

clf = LogisticRegression()

# fit the model with data
clf.fit(X_train,y_train)

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Output:
Accuracy: 0.7583333333333333

PART-B.

Aim: MULTIPLE REGRESSION MODEL Apply multiple regressions, if data have a continuous independent variable. Apply on above dataset.

Theory: Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable. In essence, multiple regression is the extension of ordinary least-squares (OLS) regression because it involves more than one explanatory variable. Simple linear regression is a function that allows an analyst or statistician to make predictions about one variable based on the information that is known about another variable. Linear regression can only be used when one has two continuous variables—an independent variable and a dependent variable. The independent variable is the parameter that is used to calculate the dependent variable or outcome. A multiple regression model extends to several explanatory variables.

Code:

```
import pandas
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
from sklearn import linear_model
regr = linear_model.LinearRegression()
regr.fit(X, y)
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])
import pandas
from sklearn import linear_model
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
regr.fit(X, y)
print(regr.coef_)
```

Output:

```
[0.00755095 0.00780526]
```

```
import pandas
from sklearn import linear_model
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
regr.fit(X, y)
predictedCO2 = regr.predict([[3300, 1300]])
print(predictedCO2)
```

Output:

```
[114.75968007]
```

Practical No. 10

PART -A

Aim: CLASSIFICATION MODEL

a. Install relevant package for classification.
b. Choose classifier for classification problem.
c. Evaluate the performance of classifier.

Theory: A classification model attempts to draw some conclusion from observed values. Given one or more inputs a classification model will try to predict the value of one or more outcomes. Outcomes are labels that can be applied to a dataset. For example, when filtering emails "spam" or "not spam" (also known as "ham", < seriously, look it up if you don't believe me), when looking at transaction data, "fraudulent", or "authorized" There are two approaches to machine learning: supervised and unsupervised. In a supervised model, a training dataset is fed into the classification algorithm. That lets the model know what is, for example, "authorized" transactions. Then the test data sample is compared with that to determine if there is a "fraudulent" transaction. This type of learning falls under "Classification". Unsupervised models on the other hand, are fed a dataset that is not labelled and looks for clusters of data points. It can be used to search data for similarities, detect patterns, or identify outliers within a dataset. A typical use case would be finding similar images. Unsupervised models can also be used to find "fraudulent" transactions by looking for anomalies within a dataset.

Code:

```
import pandas as pd
fruits = pd.read_table('fruits_data.txt')
fruits.head()
```

Output:

	fruit_label	fruit_name	fruit_subtype	mass	width	height	color_score
0	1	apple	granny_smith	192	8.4	7.3	0.55
1	1	apple	granny_smith	180	8.0	6.8	0.59
2	1	apple	granny_smith	176	7.4	7.2	0.60
3	2	mandarin	mandarin	86	6.2	4.7	0.80
4	2	mandarin	mandarin	84	6.0	4.6	0.79

```
#Training and testing datasets

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
feature_names = ['mass', 'width', 'height', 'color_score']
X = fruits[feature_names]
y = fruits['fruit_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#K-nearest neighbor classifier

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'
    .format(knn.score(X_train, y_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
    .format(knn.score(X_test, y_test)))
```

Output:

Accuracy of K-NN classifier on training set: 0.95

Accuracy of K-NN classifier on test set: 1.00

PART-B.

Aim: CLUSTERING MODEL

a. Clustering algorithms for unsupervised classification.
b. Plot the cluster data using R visualization.

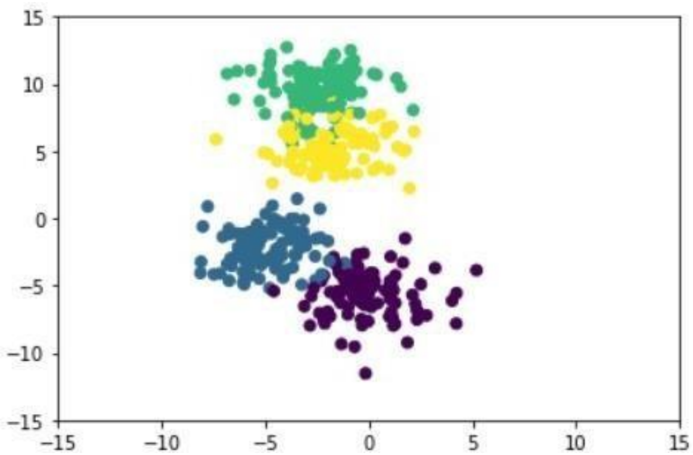
Theory: Clustering is a Machine Learning technique that involves the grouping of data points. Given a set of data points, we can use a clustering algorithm to classify each data point into a specific group. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features. Clustering is a method of unsupervised learning and is a common technique for statistical data analysis used in many fields. clustering analysis to gain some valuable insights from our data by seeing what groups the data points fall into when we apply a clustering algorithm.

Code:

```
# import statements
from sklearn.datasets import make_blobs
import numpy as np
import matplotlib.pyplot as plt
# create blobs
data = make_blobs(n_samples=400, n_features=2, centers=4, cluster_std=1.6, random_state=
50)
# create np array for data points
points = data[0]
# create scatter plot
plt.scatter(data[0][:,0], data[0][:,1], c=data[1], cmap='viridis')
plt.xlim(-15,15)
plt.ylim(-15,15)
```

Output:

(-15.0, 15.0)



```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)

kmeans.fit(points)
print(kmeans.cluster_centers_)
y_km = kmeans.fit_predict(points)
plt.scatter(points[y_km ==0,0], points[y_km == 0,1], s=100, c='red')
plt.scatter(points[y_km ==1,0], points[y_km == 1,1], s=100, c='black')
plt.scatter(points[y_km ==2,0], points[y_km == 2,1], s=100, c='blue')
plt.scatter(points[y_km ==3,0], points[y_km == 3,1], s=100, c='cyan')
```

Output:

```
[[-0.17419501 -5.53888403]
 [-2.58575308 9.9704047 ]
 [-1.74809641  5.54583068]
 [-5.01621736 -2.11522242]]
<matplotlib.collections.PathCollection at 0x7f5e7f6f2d50>
```