LAB RECORD

ON

BIG DATA ANALYTICS (16ITC35)

Blessy Kotrika

160117737068

VII Sem-IT H-2

DEPARTMENT OF INFORMATION TECHNOLOGY
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)
(Affiliated to Osmania University; Accredited by NBA (AICTE) and NAAC (UGC), ISO Certified 9001:2015), Kokapet (V), GANDIPET(M), HYDERABAD– 500 075

Website: www.cbit.ac.in

CERTIFICATE

This is to certify that this is a bonafide record of the work done by Ms. Blessy Kotrika bearing roll number 1601-17-737-068 of BE- VII Sem in the Big Data Analytics lab during the academic year 2020-21.

Staff Member In Charge

Ms. Kratika Sharma

Asst. Professor, Dept. of IT,

CBIT, Hyderabad.

List of Experiments

PROGRAM 1

AIM: Understanding and using basic HDFS commands

Hadoop HDFS is a distributed file system which provides redundant storage space for files having huge sizes. It is used for storing files which are in the range of terabytes to petabytes. Command Line is one of the simplest interface to HDFS. Below are the basic HDFS File System Commands which are similar to Linux file system commands. Once the hadoop daemons are started running, HDFS file system is ready and file system operations like creating directories, moving files, deleting files, reading files and listing directories.

1. **mkdir:** To create a directory in hadoop file system.
   Syntax: hdfs fs -mkdir <folder-name>
   Example: hdfs fs -mkdir /hello
2. **ls:** This command is used to list all the files and directories.
   Syntax: hdfs fs -ls <path>
   Example: hdfs dfs -ls /
3. **put:** To copy files/folders from local file system to hdfs store.
   Syntax: hdfs fs -put <local file path> <dest present on hdfs>
   Example: hdfs fs -put ../Desktop/file.txt /hello
4. **get:** To copy files/folders from hdfs store to local file system.
   Syntax: hdfs fs -get <srcfile on hdfs> <local file dest>
   Example: hdfs fs -get /hello/file.txt ..Desktop/folder
5. **cat:** To print file contents.
   Syntax: hdfs fs -put <path>
   Example: hdfs fs -cat /hello/file.txt
6. **cp:** This command is used to copy files within hdfs.
   Syntax: hdfs fs -cp <src on hdfs> <dest on hdfs>
   Example: hdfs -cp /hello /hello1
7. **mv:** This command is used to move files within hdfs.
   Syntax: hdfs fs -mv <src on hdfs> <dest on hdfs>
   Example: hdfs -mv /hello/myfile.txt /hello1
8. **rm**: This command removes the file or empty directory identified by path.
   Syntax: hdfs fs -rm <path>
   Example: hdfs fs -rm /hello/file.txt
9. **getmerge:** This command retrieves all files that match the path src in HDFS, and copies them to a single, merged file in the local file system identified by localDest.

Syntax: hdfs fs -getmerge <src> <localDest>

Example:

10. **setrep**: This command is used to change the replication factor of a file/directory in HDFS. By default it is 3 for anything which is stored in HDFS (as set in hdfs core-site.xml).

    Syntax: hadoop fs -setrep <rep_factor> <path>

    Example: hadoop fs -setrep 6 hello/sample

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

    Syntax: hdfs fs -touchz <file-path>

    Example: hdfs fs -touchz /hello/myfile.txt

12. **test:** Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.

Options	Description
-d	Check whether the path given by the user is a directory or not, return 0 if it is a directory.
-e	Check whether the path given by the user exists or not, return 0 if the path exists.
-f	Check whether the path given by the user is a file or not, return 0 if it is a file.
-s	Check if the path is not empty, return 0 if a path is not empty.
-r	return 0 if the path exists and read permission is granted
-w	return 0 if the path exists and write permission is granted
-z	Checks whether the file size is 0 byte or not, return 0 if the file is of 0 bytes.

    Syntax: hdfs fs -test -[ezd] <path>

    Example: hdfs fs -test -d /sample

13. **appendToFile:** The HDFS fs shell command appendToFile appends the content of single or multiple local files specified in the localsrc to the provided destination file on the HDFS.

    Syntax: hadoop fs -appendToFile <localsrc> <dest>

    Example: hadoop fs -appendToFile /hello/file /hello1/sample

14. **df:** The Hadoop fs shell command df shows the capacity, size, and free space available on the HDFS file system.The -h option formats the file size in the human-readable format.

    Syntax: hadoop fs -df [-h] <path>

    Example: hadoop fs -df -h

15. **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 fs -du <dirName>

Example: hdfs fs -du /hello

16. **count:** The Hadoop fs shell command count counts the number of files, directories, and bytes under the paths that matches 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 [options] <path>

    Example: hadoop fs -count -v /

17. **chgrp:** The Hadoop fs shell command chgrp changes the group of the file specified in the path. The user must be the owner of the file or superuser. Sets group recursively if -R is specified.

    Syntax: hdfs fs -chgrp [-R] group <path>...

    Example: hdfs fs -chgrp newgroup /hello

18. **chmod:** Changes the file permissions associated with one or more objects identified by path. Performs changes recursively with -R. Mode is a 3-digit octal mode, or {augo}+/-{rwxX}. Assumes if no scope is specified and does not apply an umask.

    Syntax: hdfs fs -chmod [-R] mode,mode,.. <path>

    Example: hdfs fs -chmod 744 /sample/file.txt

19. **chown**: Sets the owning user and/or group for files or directories identified by path.... Sets owner recursively if -R is specified.

    Syntax: hdfs fs -chown [-R] [owner][:[group]] <path>

    Example: hadoop fs -chown newsample /hello

PROGRAM 2

AIM: Word count application using Mapper-Reducer on single node cluster.

In Hadoop, MapReduce is a computation that decomposes large manipulation jobs into individual tasks that can be executed in parallel cross a cluster of servers. The results of tasks can be joined together to compute final results.MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer.

MapReduce is the core component of **Hadoop** that process huge amount of data in parallel by dividing the work into a set of independent tasks. In MapReduce data flow in step by step from Mapper to Reducer.

Input Files: The data for a MapReduce task is stored in input files, and input files typically lives in HDFS. The format of these files is arbitrary, while line-based log files and binary format can also be used.

InputFormat InputFormat defines how these input files are split and read. It selects the files or other objects that are used for input. InputFormat creates InputSplit. Learn MapReduce InputFormat in detail.

InputSplits: It is created by InputFormat, logically represent the data which will be processed by an individual Mapper (We will understand mapper below). One map task is created for each split; thus the number of map tasks will be equal to the number of InputSplits. The split is divided into records and each record will be processed by the mapper. Learn MapReduce InputSplit in detail.

RecordReader: It communicates with the InputSplit in Hadoop MapReduce and converts the data into key-value pairs suitable for reading by the mapper. By default, it uses TextInputFormat for converting data into a key-value pair. RecordReader communicates with the InputSplit until the file reading is not completed. It assigns byte offset (unique number) to each line present in the file. Further, these key-value pairs are sent to the mapper for further processing.

Mapper:

It processes each input record (from RecordReader) and generates new key-value pair, and this key-value pair generated by Mapper is completely different from the input pair. The output of Mapper is also known as intermediate output which is written to the local disk. The output of the Mapper is not stored on HDFS as this is temporary data and writing on HDFS will create unnecessary copies (also HDFS is a high latency system). Mappers output is passed to the combiner for further process

MAP REDUCE FLOW CHART

Combiner

The combiner is also known as ‘Mini-reducer’. Hadoop MapReduce Combiner performs local aggregation on the mappers’ output, which helps to minimize the data transfer between mapper and reducer (we will see reducer below). Once the combiner functionality is executed, the output is then passed to the partitioner for further work. Learn MapReduce Combiner in detail.

Partitioner

Hadoop MapReduce, Partitioner comes into the picture if we are working on more than one reducer (for one reducer partitioner is not used).Partitioner takes the output from combiners and performs partitioning. Partitioning of output takes place on the basis of the key and then sorted. By hash function, key (or a subset of the key) is used to derive the partition. According to the key value in MapReduce, each combiner output is partitioned, and a record having the same key value goes into the same partition, and then each partition is sent to a reducer. Partitioning allows even distribution of the map output over the reducer.

Shuffling and Sorting

Now, the output is Shuffled to the reduce node (which is a normal slave node but reduce phase will run here hence called as reducer node). The shuffling is the physical movement of the data which is done over the network. Once all the mappers are finished and their output is shuffled on the reducer nodes, then this intermediate output is merged and sorted, which is then provided as input to reduce phase.

Reducer

It takes the set of intermediate key-value pairs produced by the mappers as the input and then runs a reducer function on each of them to generate the output. The output of the reducer is the final output, which is stored in HDFS. Follow this link to learn about Reducer in detail.

SHUFFLE PHASE

RecordWriter

It writes these output key-value pair from the Reducer phase to the output files.

OutputFormat

The way these output key-value pairs are written in output files by RecordWriter is determined by the OutputFormat. OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk. Thus the final output of reducer is written on HDFS by OutputFormat instances.Hence, in this manner, a Hadoop MapReduce works over the cluster.

Steps for execution in python:

Step 1: Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /wordcount

$hadoop fs -mkdir /wordput/inp

Move the text file onto the hadoop store.

$ hadoop fs -put /home /Desktop/Lab/word.txt

Step 2: Type in the mapper code and reducer code.

```
hduser@blessy-VirtualBox:~$ cat /home/blessy/Desktop/Lab/word.txt
hai hellow hai hey who what who hai when where what when where hai hellow
hduser@blessy-VirtualBox:~$
```

```
hduser@blessy68:/$ hadoop dfs -cat   /wordcount/inp/word.txt
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

20/11/18 20:28:02 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable


hai hellow hai hey who what who hai when where what when where hai hellow
hduser@blessy68:/$
```

my_mapper.py

```
#!/usr/bin/python
import sys;
```

```
for line in sys.stdin:
   line = line.strip()
   words = line.split()
   for word in words:
      print ('%s-%s' %(word,1))
```

my_reducer.py

```
#!/usr/bin/python
import sys;
from operator import itemgetter;
prev_word = None
prev_count = 0
word = None
for line in sys.stdin:
      line = line.strip()
      word,count = line.split('-',1)
      count=int(count)
      if word == prev_word:
            prev_count+=1;
      else:
            if prev_word:
                  print('%s-%s' %(prev_word,prev_count))
            prev_word = word;
            prev_count = count;
print('%s-%s' %(prev_word,prev_count))
```

Step 3: Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.9.0.jar **-file** /home/Desktop/Lab/word.txt **-mapper** "python3 mapper.py" **-file** / home /Desktop/Lab/reducer.py **-reducer** "python3 reducer.py**" -input** /wordcount/inp **-output** /wordcount/out

Step 4: Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /wordcount/out

Output on the terminal

```
hduser@blessy68:/$ hadoop dfs -cat /wordcount/out/part-00000
DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.

20/11/18 20:24:52 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
hai     4
hellow  2
hey     1
what    2
when    2
where   2
who     2
hduser@blessy68:/$
```

File contents

```
hai  4
hellow    2
hey  1
what      2
when      2
where     2
who 2
```

PROGRAM 3

AIM: Analysis of weather dataset using Mapper-Reducer on single node cluster.

MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages

Data Format

The data is from the National Climatic Data Center (NCDC, http://www .ncdc.noaa.gov/). The data is stored using a line-oriented ASCII format, in which each line is a record. The format supports a rich set of meteorological elements, many of which are optional or with variable data lengths. For simplicity, we shall focus on the basic elements, such as temperature, which are always present and are of fixed width.

Example:

0057

332130 # USAF weather station identifier

99999 # WBAN weather station identifier

19500101 # observation date

0300 # observation time

Since there are tens of thousands of weather stations, the whole dataset is made up of a large number of relatively small files. It's generally easier and more efficient to process a smaller number of relatively large files, so the data was preprocessed so that each year's readings were concatenated into a single file.The script loops through the compressed year files, first printing the year, and then processing each file using awk. The awk script extracts two fields from the data: the air temperature and the quality code.

Steps for execution in python:

Step 1: Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /weather_data

$hadoop fs -mkdir /weather_data/input

Move the text file onto the hadoop store.

$ hadoop fs -put /home /Desktop/Lab/weather/* /weather/inp

Step 2: Type in the mapper code and reducer code.

my_mapper.py

```
#!/usr/bin/python
import sys
l=list()
for line in sys.stdin:
        line = line.strip()
        year=int(line[15:19])
        temp=int(line[87:92])
        l.append([year,temp])
for record in l:
        print('%s        %s' %(record[0],record[1]))
```

my_reducer.py

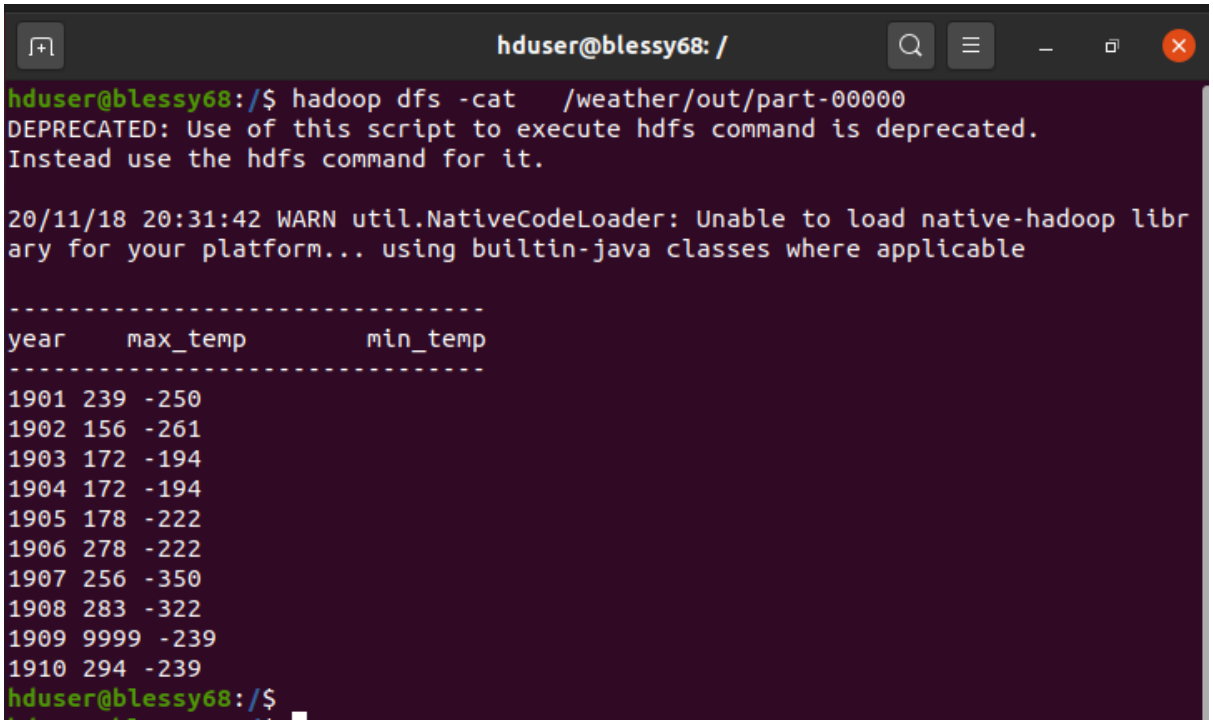
```
#!/usr/bin/python
import sys
dmax=dict()
y_list=list()
dmin=dict()
for line in sys.stdin:
        line = line.strip()
        year,temp = map(int,line.split())
        if year not in y_list:
                y_list.append(year)
                dmax[year]=temp
                dmin[year]=temp
        else:
                if dmax[year]<temp:
                        dmax[year]=temp
                if dmin[year]>temp:
                        dmin[year]=temp
print('---------------------------------')
print('year      max_temp      min_temp')
print('---------------------------------')
for i in dmax.keys():
        print('%s----------%s-----------%s' %(i,dmax[i],dmin[i]))
```

Step 3: Run the following command.

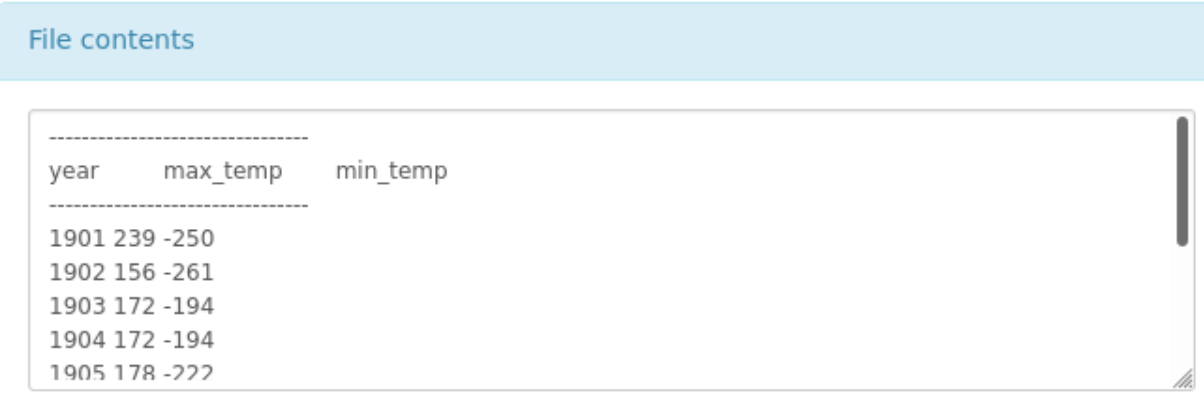
$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.9.0.jar **-file** mapperpath **-mapper** “python3 mapper.py” **-reducer path -reducer** “python3 reducer.py” **-input** /weather /inp/* **-output** /weather/out

Step 4: Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /weather_data/OUT



Output on the terminal



Result from the Hadoop web interface

PROGRAM 4

AIM: Web Log Analysis using Mapper-Reducer on single node cluster.

Web logs are data that is generated by web servers for requests they receive. There are various web servers such as Apache, Nginx, Tomcat, and so on. Each web server logs data in a specific

format. The server access log records all requests processed by the server. Of course, storing the information in the access log is only the start of log management. The next step is to analyze this information to produce useful statistics. We can store log data in many different format but here we are going to use data from the Apache Web Server, which is in combined access logs.

Combined Log Format

A typical configuration for the access log might look as follows.

LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{Useragent}i\"" combined
CustomLog log/acces_log combined

Example: 127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"

127.0.0.1 (%h)

This is the IP address of the client (remote host) which made the request to the server.

- (%l)

The "hyphen" in the output indicates that the requested piece of information is not available

frank (%u)

This is the userid of the person requesting the document as determined by HTTP authentication.

[10/Oct/2000:13:55:36 -0700] (%t)

The time that the server finished processing the request.

The format is:

[day/month/year:hour:minute:second zone]

day = 2*digit

month = 3*letter

year = 4*digit

hour = 2*digit

minute = 2*digit

second = 2*digit

zone = (`+' | `-') 4*digit

"GET /apache_pb.gif HTTP/1.0" (\"%r\")

The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is GET. Second, the client requested the resource /apache_pb.gif, and third, the client used the protocol HTTP/1.0.

200 (%>s)

This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2),

a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5).

2326 (%b)

The last entry indicates the size of the object returned to the client (in bytes), not including the response headers.

"http://www.example.com/start.html" (\"%{Referer}i\")

The "Referer" (sic) HTTP request header. This gives the site that the client reports having been referred from. (This should be the page that links to or includes /apache_pb.gif). This is the page that is linked to this URL.

"Mozilla/4.08 [en] (Win98; I ;Nav)" (\"%{User-agent}i\")

This is the browser identification string.

Write a map reduce program that reads a weblog file to give URL address and their counts. (URL, Total Count)

Steps for execution in python:

Step 1: Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /web_log

$hadoop fs -mkdir /web_log/inputdata

Move the text file onto the hadoop store.

$ hadoop fs -put /home/Desktop/lab/web_log/web_log.txt /web_log/inputdata

Step 2: Type in the mapper code and reducer code.

my_mapper.py

```
#!/usr/bin/python
import sys;
import re;
for line in sys.stdin:
   line = line.strip()
   words=line.split()
   x=re.findall("http://",words[10])
   if (x):
        print('%s        %s' %(words[10],1))
```

my_reducer.py

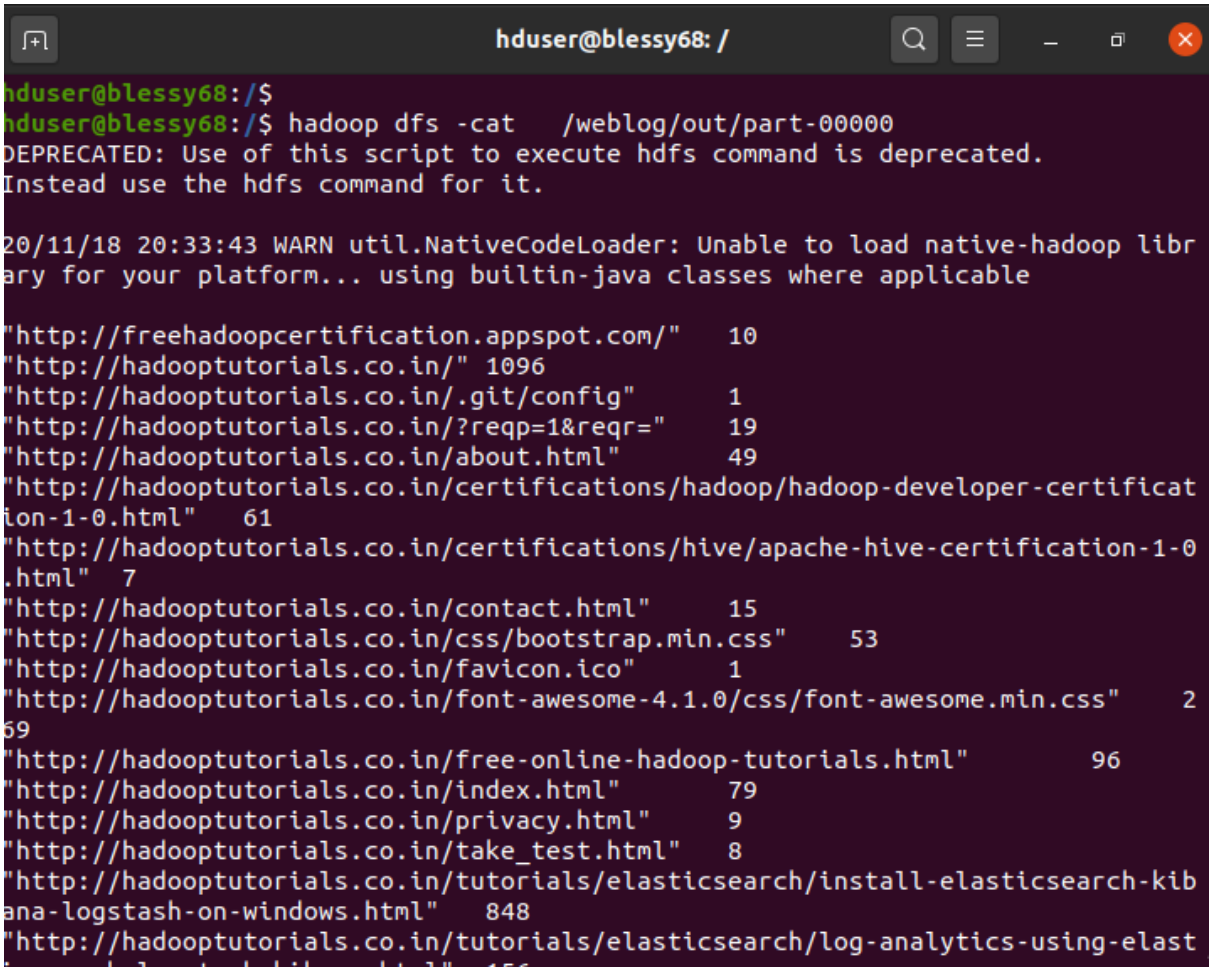
```
#!/usr/bin/python
import sys;
from operator import itemgetter;
prev_word = None
prev_count = 0
word = None
for line in sys.stdin:
        line = line.strip()
        word,count = line.split('\t',1)
        count=int(count)
        if word == prev_word:
                prev_count+=1;
        else:
                if prev_word:
                        print('%s        %s' %(prev_word,prev_count))
                prev_word = word;
                prev_count = count;
print('%s        %s' %(prev_word,prev_count))
```

Step 3: Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.9.0.jar **-file** /home /Desktop/Lab/mapper.py **-mapper** "python3 mapper.py" **-file** / home/ /Desktop/Lab/reducer.py **-reducer** "python3 reducer.py" **-input** /weblog/inp **-output** /weblog/out

Step 4: Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /web_log/out



Output

web interface

QUESTION 2

Write a map reduce program that reads a weblog file to output the most number of referral sites.

Steps for execution in python:

Step 1: Open terminal and create a text file and move it to the hadoop store using the following command.

Make a directory

$ hadoop fs -mkdir /web_log

$hadoop fs -mkdir /web_log/inputdata

Move the text file onto the hadoop store.

$ hadoop fs -put /home/blessy/Desktop/Lab/weblog/data.txt /weblog/inp

Step 2: Type in the mapper code and reducer code.

my_mapper.py

```
#!/usr/bin/python
import sys;
import re;
for line in sys.stdin:
    line = line.strip()
    words=line.split()
    x=re.findall("http://",words[10])
    if (x):
        print('%s %s' %(words[10],1))
```

most_referred_reducer.py

```
#!/usr/bin/python
import sys;
from operator import itemgetter;
prev_word = None
prev_count = 0
word = None
m=0
```

```
for line in sys.stdin:
        line = line.strip()
        word,count = line.split('\t',1)
        count=int(count)
        if word == prev_word:
                prev_count+=1;
        else:
                if prev_word and m<prev_count:
                        most=prev_word
                        m=prev_count
                prev_word = word;
                prev_count = count;
if m<prev_count:
        most=prev_word
        m=prev_count
print('%s        %s' %(most,m))
```

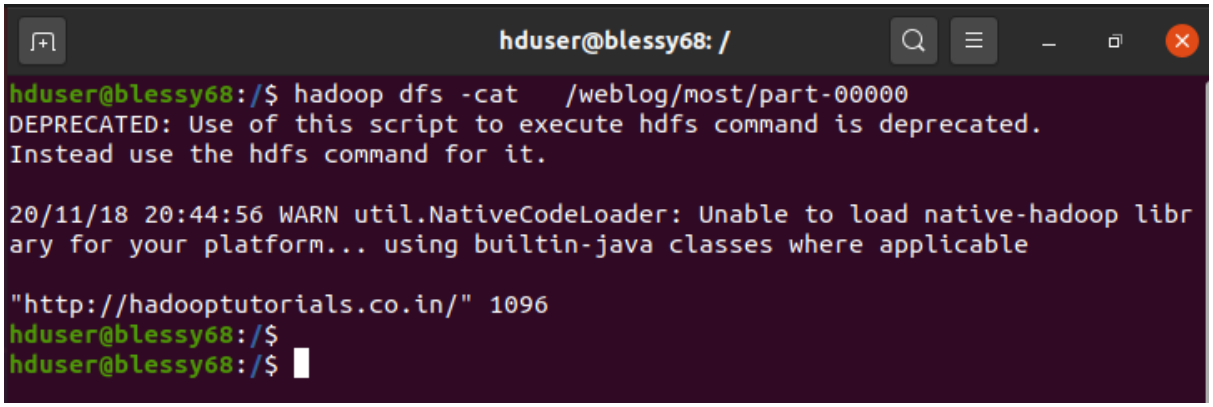
Step 3: Run the following command.

$ **hadoop jar** /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar **-file** /home/blessy/Desktop/Lab/web_log/my_mapper.py **-mapper** "python my_mapper.py" **-file** /home/blessy/Desktop/Lab/web_log/most_referred_reducer.py **-reducer** "python most_referred_reducer.py" **-input** /web_log/inputdata **-output** /web_log/ most_referred__url

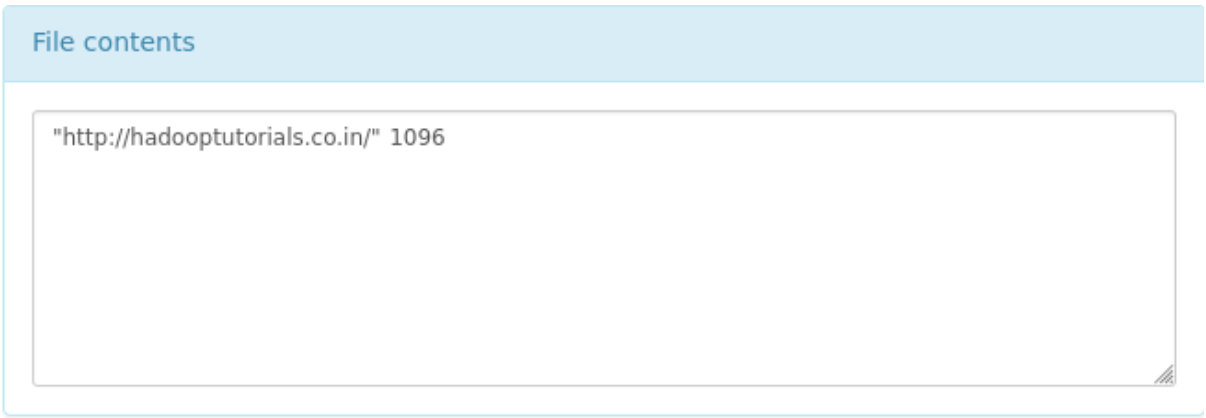
Step 4: Open result on the terminal Or you can see your result in Hadoop Web Interface http://localhost:50070/

Goto utilities> Browse file System> /weblog/ most

Output on the terminal

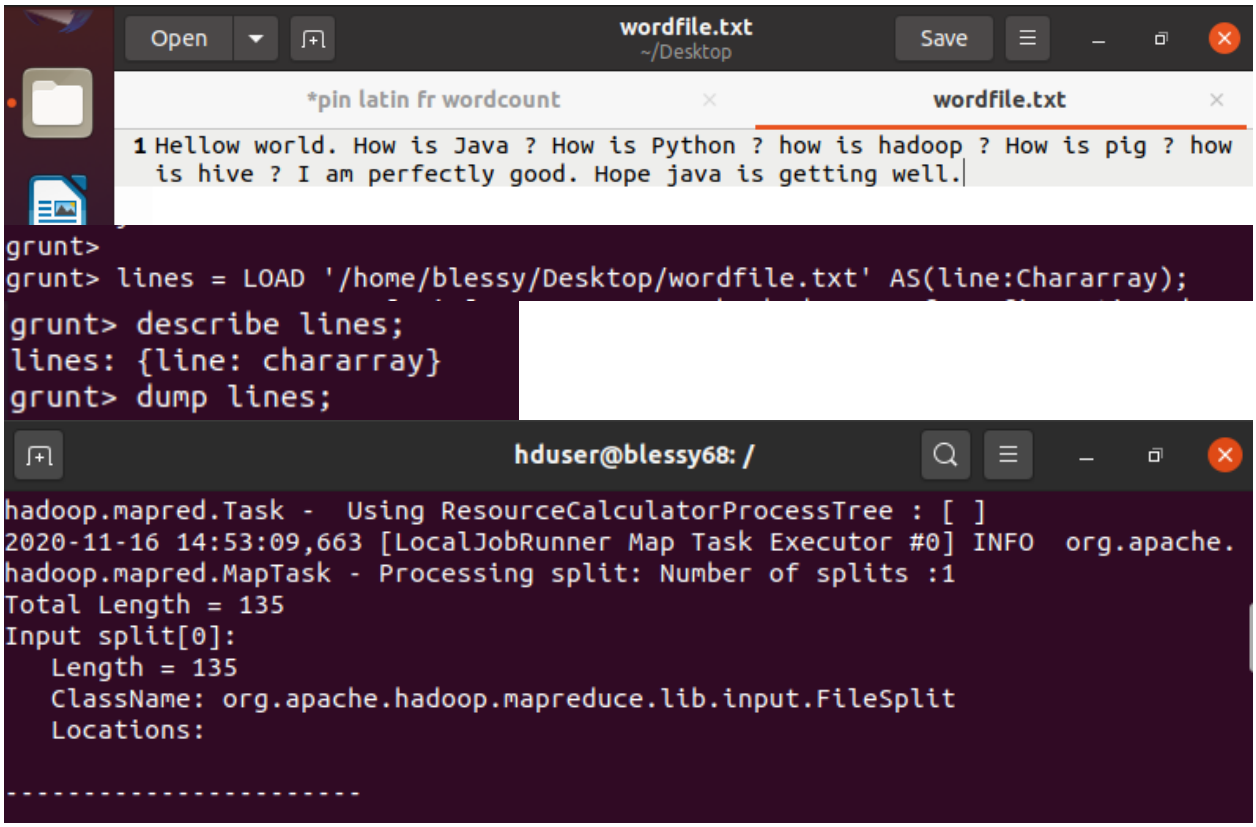


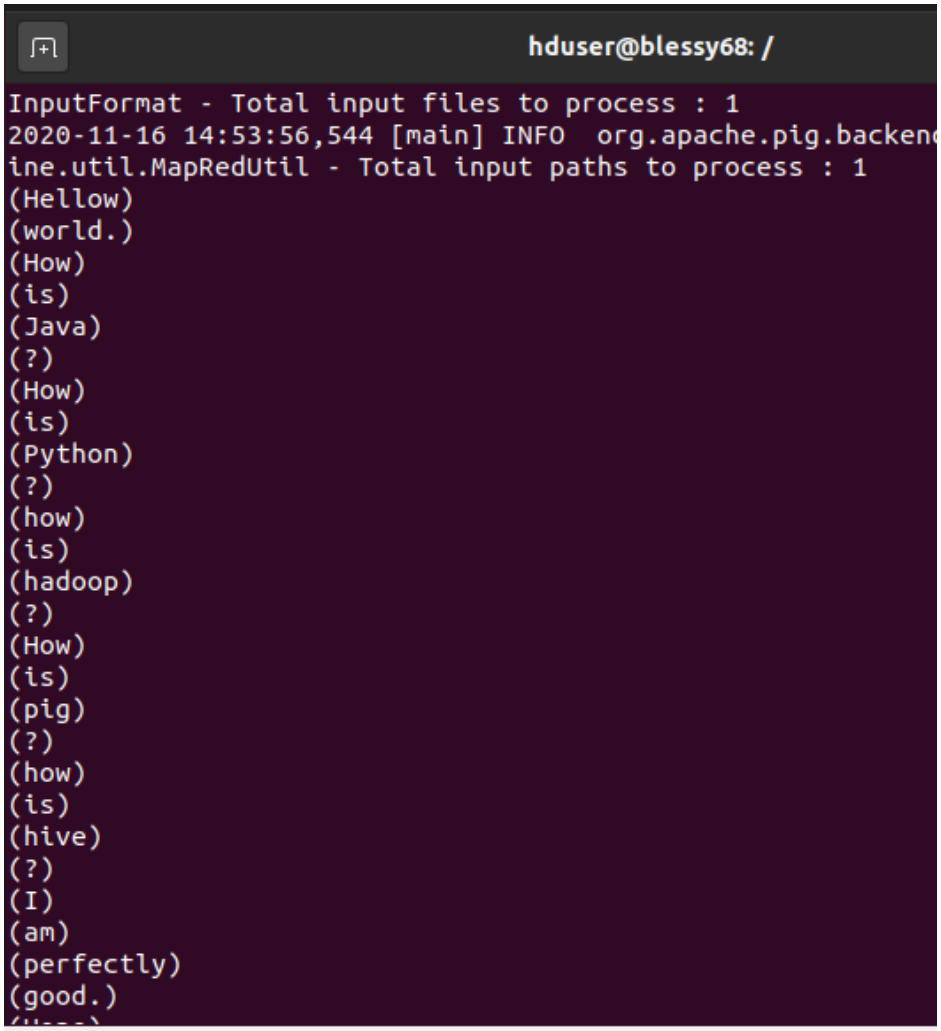
Result from the Hadoop web interface

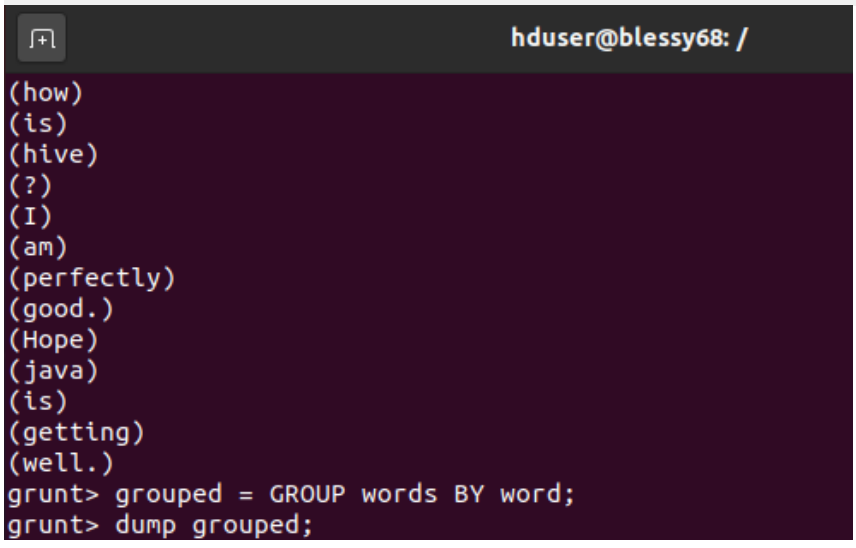


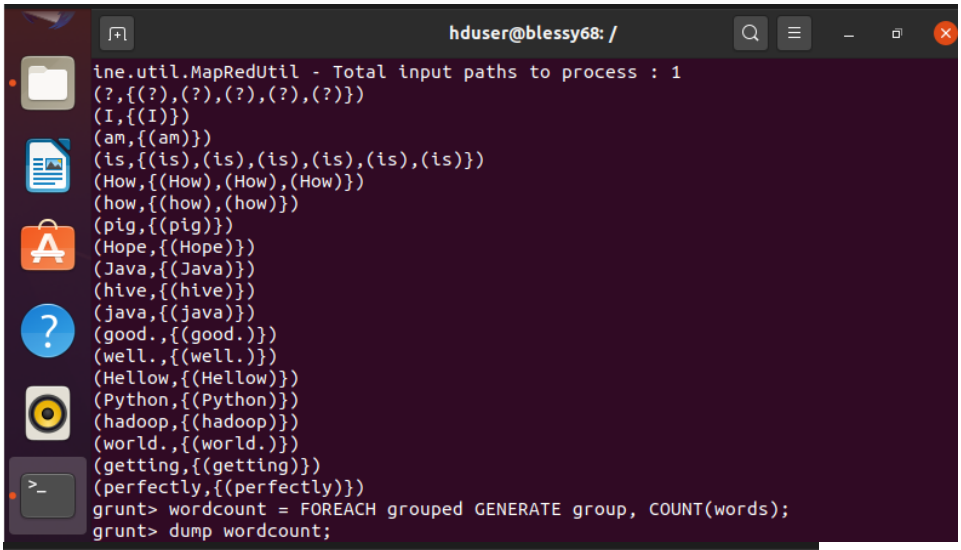
Program No 1: Write Pig Latin script to perform word count.

```
1 pig -x local
2
3 lines = LOAD '/home/blessy/Desktop/wordfile.txt' AS(line:Chararray);
4 words = FOREACH lines GENERATE FLATTEN(TOKENIZE(line)) as word;
5 grouped = GROUP words BY word;
6 wordcount = FOREACH grouped GENERATE group, COUNT(words);
7  DUMP wordcount;
8
```




hduser@blessy68: /
InputFormat - Total input files to process : 1
2020-11-16 14:53:56,544 [main] INFO org.apache.pig.backend
ine.util.MapRedUtil - Total input paths to process : 1
(Hellow)
(world.)
(How)
(is)
(Java)
(?)
(How)
(is)
(Python)
(?)
(how)
(is)
(hadoop)
(?)
(How)
(is)
(pig)
(?)
(how)
(is)
(hive)
(?)
(I)
(am)
(perfectly)
(good.)


hduser@blessy68: /
(how)
(is)
(hive)
(?)
(I)
(am)
(perfectly)
(good.)
(Hope)
(java)
(is)
(getting)
(well.)
grunt> grouped = GROUP words BY word;
grunt> dump grouped;



Program No 2: The Daily Show Data Analysis

We have a historical data of The Daily Show guests from 1999 to 2004. (attached file)
Dataset Description:
YEAR – The year the episode aired.
Occupation -Their occupation or office, according to Google's Knowledge Graph. On the other hand, if they are not in there, how Stewart introduced them on the program.

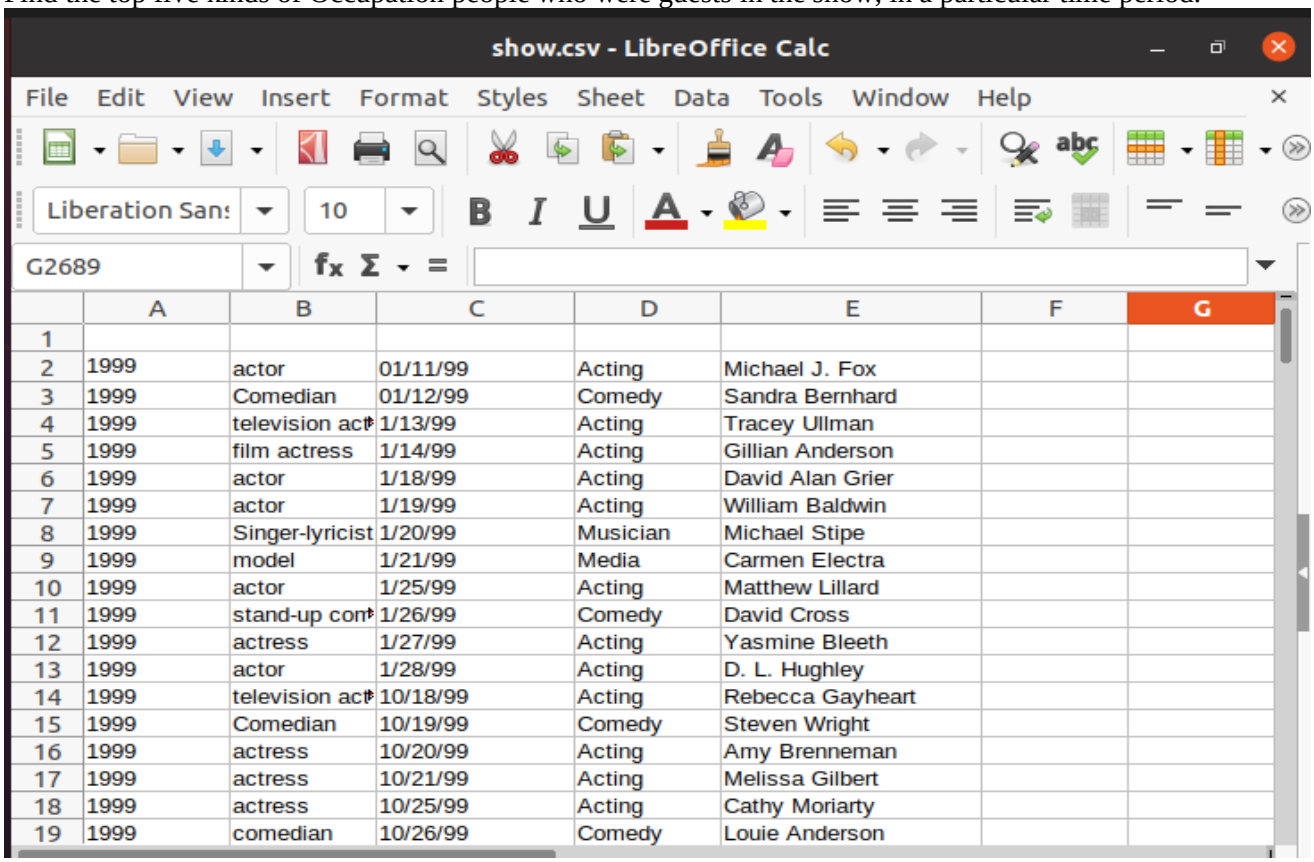
Show – Air date of the episode. Not unique, as some shows had more than one guest

Group – A larger group designation for the occupation. For instance, U.S senators, U.S presidents, and former presidents are all under "politicians"

Guest_List – The person or list of people who appeared on the show, according to Wikipedia. The

Occupation only refers to one of them in a given row.

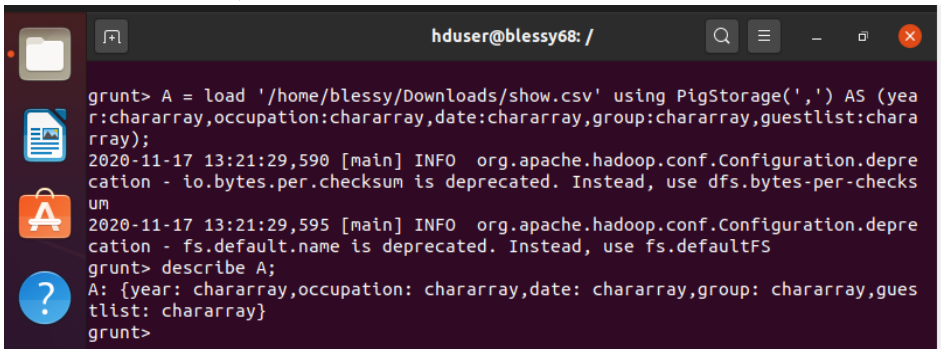
Problem Statement 1:
Find the top five kinds of Occupation people who were guests in the show, in a particular time period.

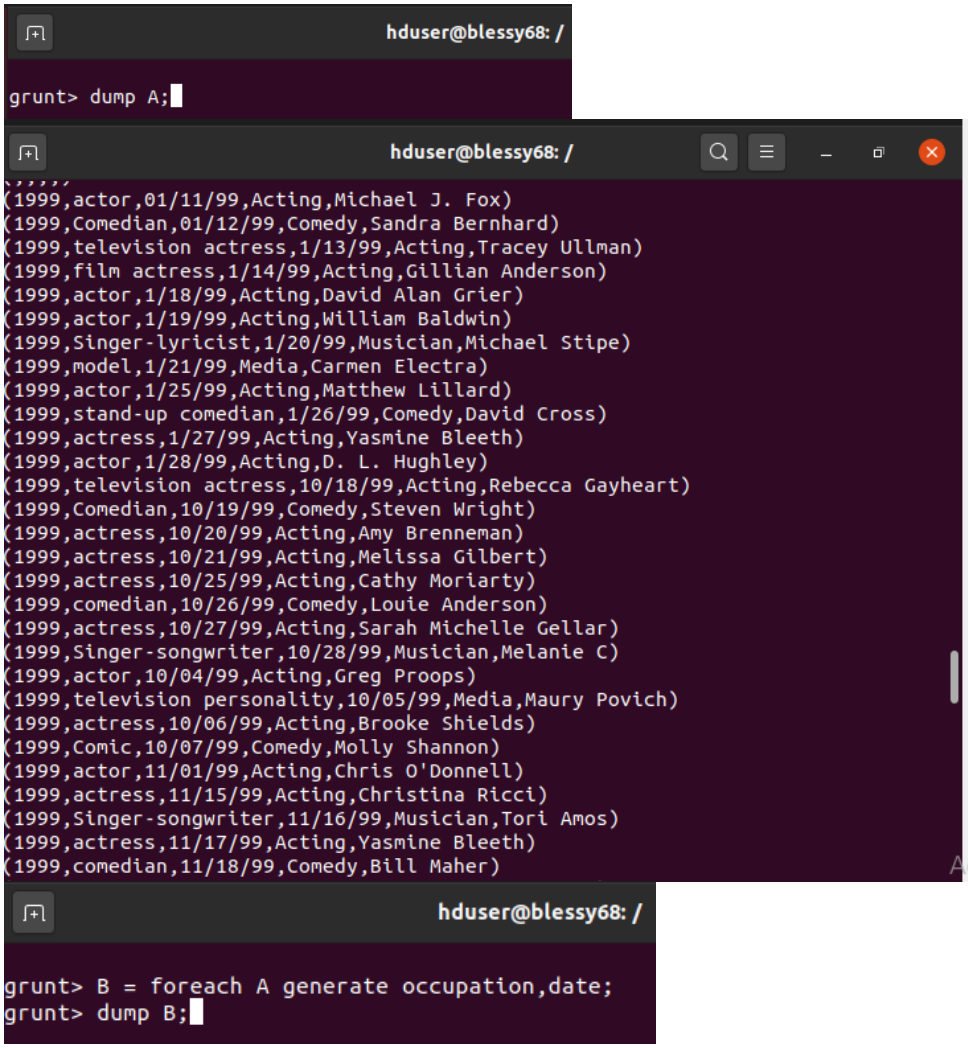


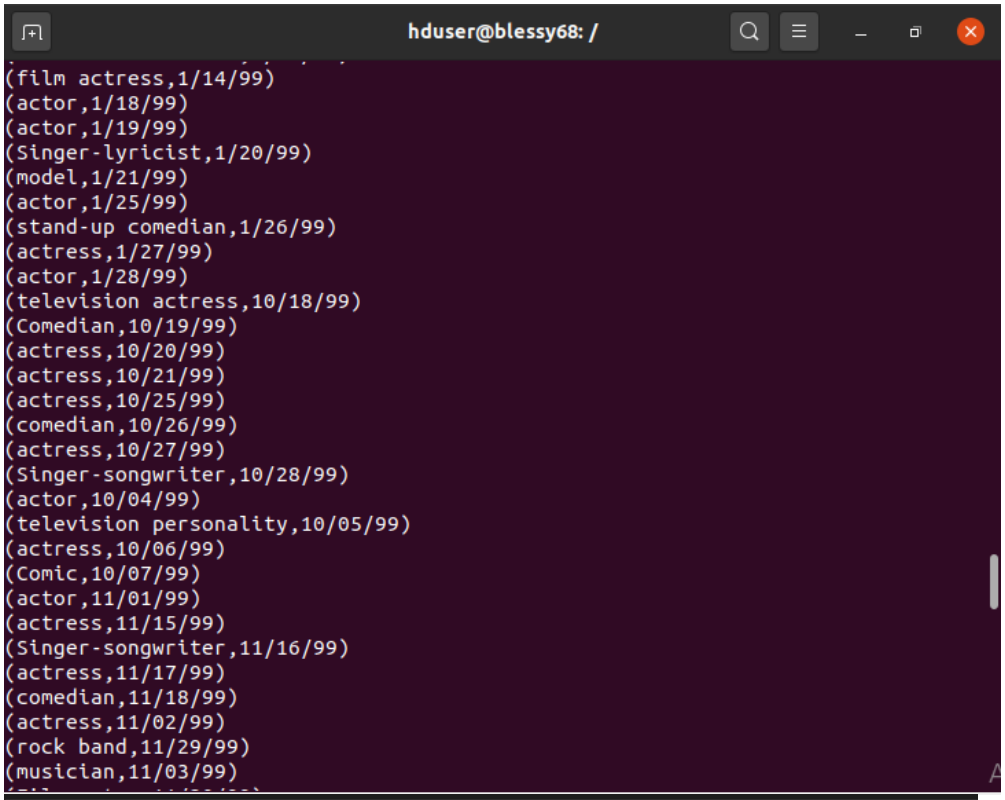
	A	B	C	D	E	F	G
1							
2	1999	actor	01/11/99	Acting	Michael J. Fox		
3	1999	Comedian	01/12/99	Comedy	Sandra Bernhard		
4	1999	television act	1/13/99	Acting	Tracey Ullman		
5	1999	film actress	1/14/99	Acting	Gillian Anderson		
6	1999	actor	1/18/99	Acting	David Alan Grier		
7	1999	actor	1/19/99	Acting	William Baldwin		
8	1999	Singer-lyricist	1/20/99	Musician	Michael Stipe		
9	1999	model	1/21/99	Media	Carmen Electra		
10	1999	actor	1/25/99	Acting	Matthew Lillard		
11	1999	stand-up com	1/26/99	Comedy	David Cross		
12	1999	actress	1/27/99	Acting	Yasmine Bleeth		
13	1999	actor	1/28/99	Acting	D. L. Hughley		
14	1999	television act	10/18/99	Acting	Rebecca Gayheart		
15	1999	Comedian	10/19/99	Comedy	Steven Wright		
16	1999	actress	10/20/99	Acting	Amy Brenneman		
17	1999	actress	10/21/99	Acting	Melissa Gilbert		
18	1999	actress	10/25/99	Acting	Cathy Moriarty		
19	1999	comedian	10/26/99	Comedy	Louie Anderson		

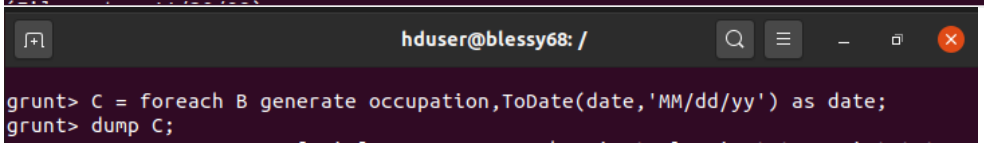
SCRIPT:

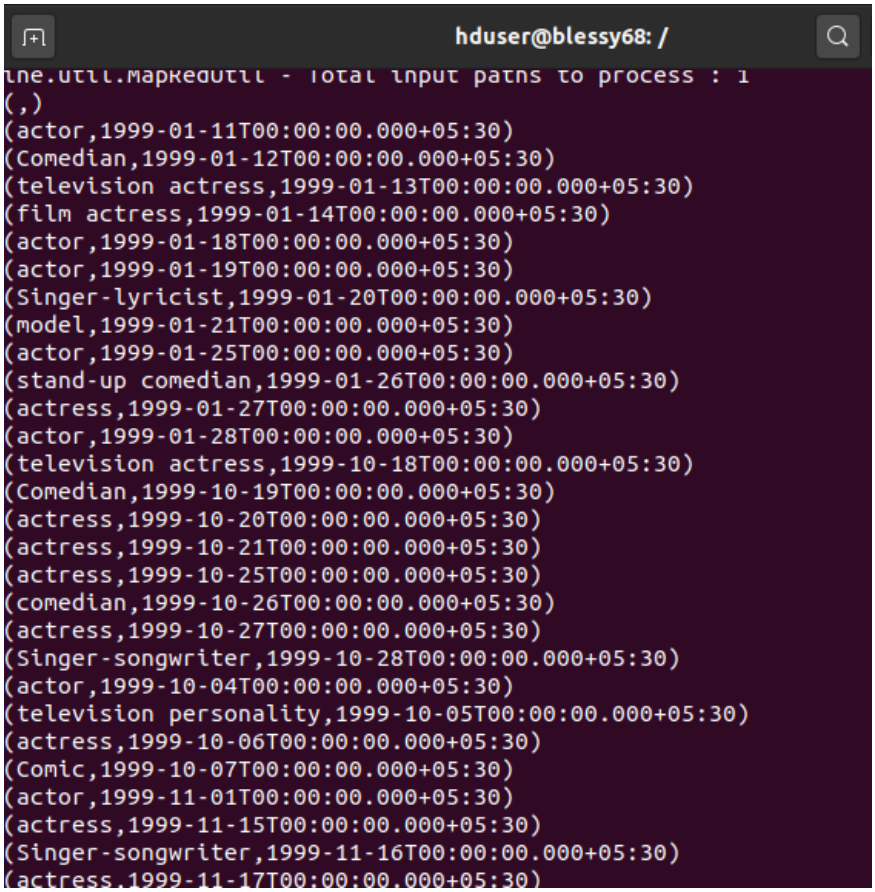
```
A = load '/home/blessy/Downloads/show.csv' using PigStorage(',') AS
(year:chararray,occupation:chararray,date:chararray,group:chararray,
gusetlist:chararray);
B = foreach A generate occupation,date;
C = foreach B generate occupation,ToDate(date,'MM/dd/yy') as date;
D = filter C by ((date> ToDate('1/11/99','MM/dd/yy')) AND
(date<ToDate('6/11/99','MM/dd/yy')));
#Date range can be modified by the user
E = group D by occupation;
F = foreach E generate group, COUNT(D) as cnt;
G = order F by cnt desc;
H = limit G 5;
```

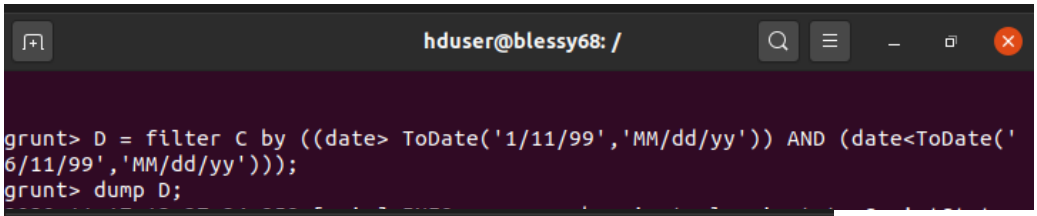


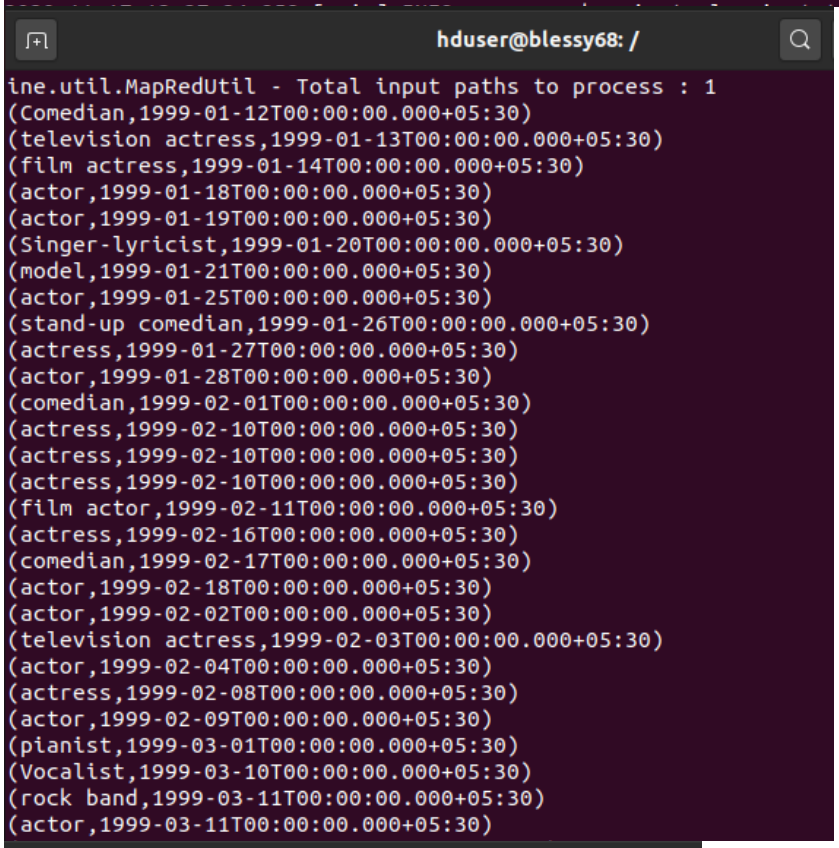

hduser@blessy68: /
grunt> dump A;
hduser@blessy68: /
(1999,actor,01/11/99,Acting,Michael J. Fox)
(1999,Comedian,01/12/99,Comedy,Sandra Bernhard)
(1999,television actress,1/13/99,Acting,Tracey Ullman)
(1999,film actress,1/14/99,Acting,Gillian Anderson)
(1999,actor,1/18/99,Acting,David Alan Grier)
(1999,actor,1/19/99,Acting,William Baldwin)
(1999,Singer-lyricist,1/20/99,Musician,Michael Stipe)
(1999,model,1/21/99,Media,Carmen Electra)
(1999,actor,1/25/99,Acting,Matthew Lillard)
(1999,stand-up comedian,1/26/99,Comedy,David Cross)
(1999,actress,1/27/99,Acting,Yasmine Bleeth)
(1999,actor,1/28/99,Acting,D. L. Hughley)
(1999,television actress,10/18/99,Acting,Rebecca Gayheart)
(1999,Comedian,10/19/99,Comedy,Steven Wright)
(1999,actress,10/20/99,Acting,Amy Brenneman)
(1999,actress,10/21/99,Acting,Melissa Gilbert)
(1999,actress,10/25/99,Acting,Cathy Moriarty)
(1999,comedian,10/26/99,Comedy,Louie Anderson)
(1999,actress,10/27/99,Acting,Sarah Michelle Gellar)
(1999,Singer-songwriter,10/28/99,Musician,Melanie C)
(1999,actor,10/04/99,Acting,Greg Proops)
(1999,television personality,10/05/99,Media,Maury Povich)
(1999,actress,10/06/99,Acting,Brooke Shields)
(1999,Comic,10/07/99,Comedy,Molly Shannon)
(1999,actor,11/01/99,Acting,Chris O'Donnell)
(1999,actress,11/15/99,Acting,Christina Ricci)
(1999,Singer-songwriter,11/16/99,Musician,Tori Amos)
(1999,actress,11/17/99,Acting,Yasmine Bleeth)
(1999,comedian,11/18/99,Comedy,Bill Maher)
hduser@blessy68: /
grunt> B = foreach A generate occupation,date;
grunt> dump B;

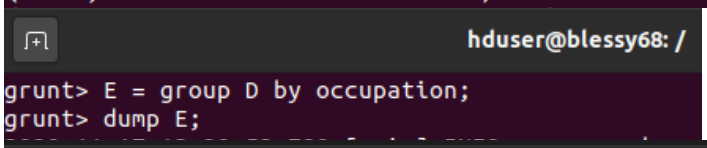

hduser@blessy68: /
(film actress,1/14/99)
(actor,1/18/99)
(actor,1/19/99)
(Singer-lyricist,1/20/99)
(model,1/21/99)
(actor,1/25/99)
(stand-up comedian,1/26/99)
(actress,1/27/99)
(actor,1/28/99)
(television actress,10/18/99)
(Comedian,10/19/99)
(actress,10/20/99)
(actress,10/21/99)
(actress,10/25/99)
(comedian,10/26/99)
(actress,10/27/99)
(Singer-songwriter,10/28/99)
(actor,10/04/99)
(television personality,10/05/99)
(actress,10/06/99)
(Comic,10/07/99)
(actor,11/01/99)
(actress,11/15/99)
(Singer-songwriter,11/16/99)
(actress,11/17/99)
(comedian,11/18/99)
(actress,11/02/99)
(rock band,11/29/99)
(musician,11/03/99)

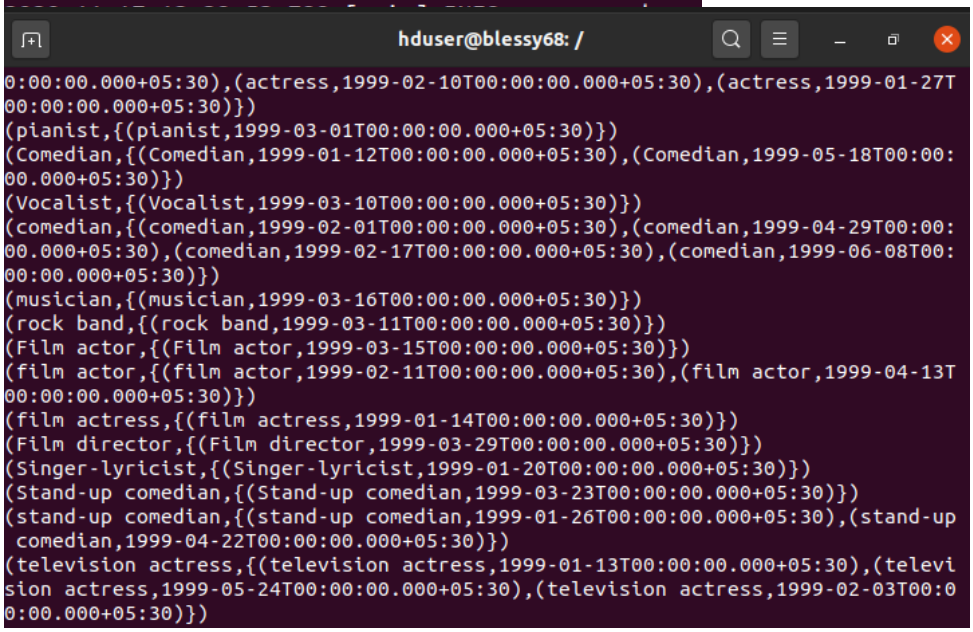

hduser@blessy68: /
grunt> C = foreach B generate occupation,ToDate(date,'MM/dd/yy') as date;
grunt> dump C;

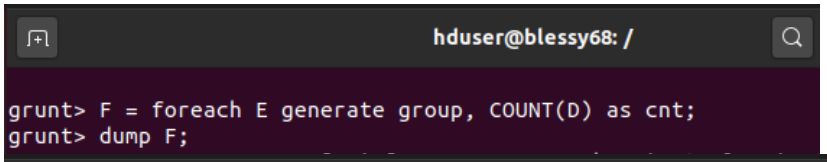

hduser@blessy68: /
(,)
(actor,1999-01-11T00:00:00.000+05:30)
(Comedian,1999-01-12T00:00:00.000+05:30)
(television actress,1999-01-13T00:00:00.000+05:30)
(film actress,1999-01-14T00:00:00.000+05:30)
(actor,1999-01-18T00:00:00.000+05:30)
(actor,1999-01-19T00:00:00.000+05:30)
(Singer-lyricist,1999-01-20T00:00:00.000+05:30)
(model,1999-01-21T00:00:00.000+05:30)
(actor,1999-01-25T00:00:00.000+05:30)
(stand-up comedian,1999-01-26T00:00:00.000+05:30)
(actress,1999-01-27T00:00:00.000+05:30)
(actor,1999-01-28T00:00:00.000+05:30)
(television actress,1999-10-18T00:00:00.000+05:30)
(Comedian,1999-10-19T00:00:00.000+05:30)
(actress,1999-10-20T00:00:00.000+05:30)
(actress,1999-10-21T00:00:00.000+05:30)
(actress,1999-10-25T00:00:00.000+05:30)
(comedian,1999-10-26T00:00:00.000+05:30)
(actress,1999-10-27T00:00:00.000+05:30)
(Singer-songwriter,1999-10-28T00:00:00.000+05:30)
(actor,1999-10-04T00:00:00.000+05:30)
(television personality,1999-10-05T00:00:00.000+05:30)
(actress,1999-10-06T00:00:00.000+05:30)
(Comic,1999-10-07T00:00:00.000+05:30)
(actor,1999-11-01T00:00:00.000+05:30)
(actress,1999-11-15T00:00:00.000+05:30)
(Singer-songwriter,1999-11-16T00:00:00.000+05:30)

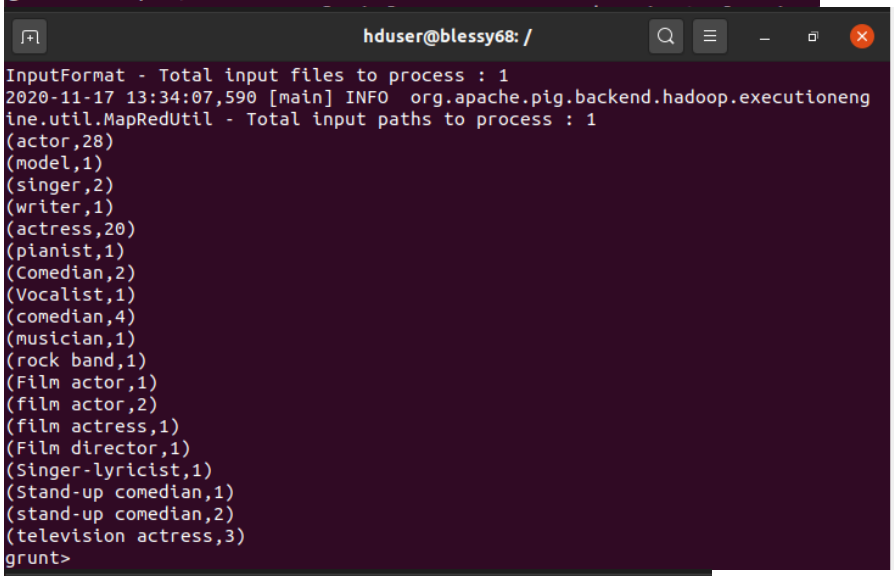

hduser@blessy68: /
grunt> D = filter C by ((date> ToDate('1/11/99','MM/dd/yy')) AND (date<ToDate('
6/11/99','MM/dd/yy')));
grunt> dump D;

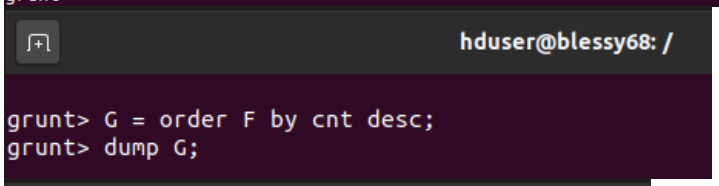

hduser@blessy68: /
ine.util.MapRedUtil - Total input paths to process : 1
(Comedian,1999-01-12T00:00:00.000+05:30)
(television actress,1999-01-13T00:00:00.000+05:30)
(film actress,1999-01-14T00:00:00.000+05:30)
(actor,1999-01-18T00:00:00.000+05:30)
(actor,1999-01-19T00:00:00.000+05:30)
(Singer-lyricist,1999-01-20T00:00:00.000+05:30)
(model,1999-01-21T00:00:00.000+05:30)
(actor,1999-01-25T00:00:00.000+05:30)
(stand-up comedian,1999-01-26T00:00:00.000+05:30)
(actress,1999-01-27T00:00:00.000+05:30)
(actor,1999-01-28T00:00:00.000+05:30)
(comedian,1999-02-01T00:00:00.000+05:30)
(actress,1999-02-10T00:00:00.000+05:30)
(actress,1999-02-10T00:00:00.000+05:30)
(actress,1999-02-10T00:00:00.000+05:30)
(film actor,1999-02-11T00:00:00.000+05:30)
(actress,1999-02-16T00:00:00.000+05:30)
(comedian,1999-02-17T00:00:00.000+05:30)
(actor,1999-02-18T00:00:00.000+05:30)
(actor,1999-02-02T00:00:00.000+05:30)
(television actress,1999-02-03T00:00:00.000+05:30)
(actor,1999-02-04T00:00:00.000+05:30)
(actress,1999-02-08T00:00:00.000+05:30)
(actor,1999-02-09T00:00:00.000+05:30)
(pianist,1999-03-01T00:00:00.000+05:30)
(Vocalist,1999-03-10T00:00:00.000+05:30)
(rock band,1999-03-11T00:00:00.000+05:30)
(actor,1999-03-11T00:00:00.000+05:30)

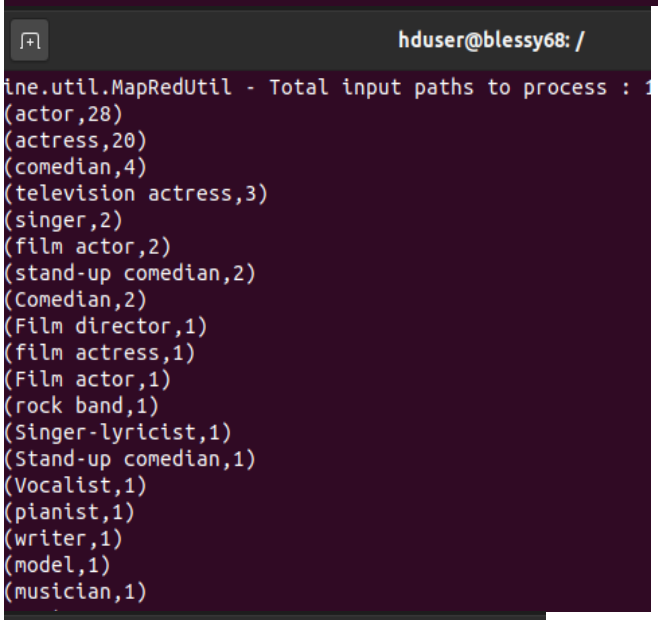

hduser@blessy68: /
grunt> E = group D by occupation;
grunt> dump E;

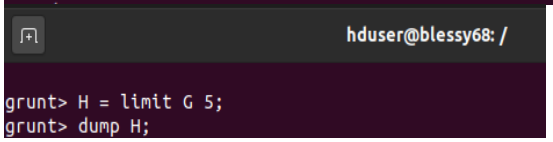

hduser@blessy68: /
0:00:00.000+05:30),(actress,1999-02-10T00:00:00.000+05:30),(actress,1999-01-27T
00:00:00.000+05:30)})
(pianist,{(pianist,1999-03-01T00:00:00.000+05:30)})
(Comedian,{(Comedian,1999-01-12T00:00:00.000+05:30),(Comedian,1999-05-18T00:00:
00.000+05:30)})
(Vocalist,{(Vocalist,1999-03-10T00:00:00.000+05:30)})
(comedian,{(comedian,1999-02-01T00:00:00.000+05:30),(comedian,1999-04-29T00:00:
00.000+05:30),(comedian,1999-02-17T00:00:00.000+05:30),(comedian,1999-06-08T00:
00:00.000+05:30)})
(musician,{(musician,1999-03-16T00:00:00.000+05:30)})
(rock band,{(rock band,1999-03-11T00:00:00.000+05:30)})
(Film actor,{(Film actor,1999-03-15T00:00:00.000+05:30)})
(film actor,{(film actor,1999-02-11T00:00:00.000+05:30),(film actor,1999-04-13T
00:00:00.000+05:30)})
(film actress,{(film actress,1999-01-14T00:00:00.000+05:30)})
(Film director,{(Film director,1999-03-29T00:00:00.000+05:30)})
(Singer-lyricist,{(Singer-lyricist,1999-01-20T00:00:00.000+05:30)})
(Stand-up comedian,{(Stand-up comedian,1999-03-23T00:00:00.000+05:30)})
(stand-up comedian,{(stand-up comedian,1999-01-26T00:00:00.000+05:30),(stand-up
comedian,1999-04-22T00:00:00.000+05:30)})
(television actress,{(television actress,1999-01-13T00:00:00.000+05:30),(televi
sion actress,1999-05-24T00:00:00.000+05:30),(television actress,1999-02-03T00:0
0:00.000+05:30)})

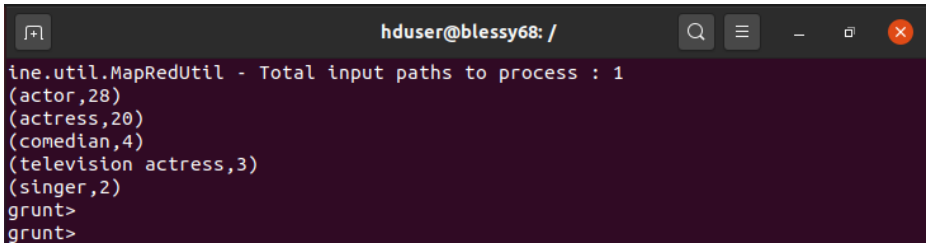

hduser@blessy68: /
grunt> F = foreach E generate group, COUNT(D) as cnt;
grunt> dump F;


hduser@blessy68: /
InputFormat - Total input files to process : 1
2020-11-17 13:34:07,590 [main] INFO org.apache.pig.backend.hadoop.executioneng
ine.util.MapRedUtil - Total input paths to process : 1
(actor,28)
(model,1)
(singer,2)
(writer,1)
(actress,20)
(pianist,1)
(Comedian,2)
(Vocalist,1)
(comedian,4)
(musician,1)
(rock band,1)
(Film actor,1)
(film actor,2)
(film actress,1)
(Film director,1)
(Singer-lyricist,1)
(Stand-up comedian,1)
(stand-up comedian,2)
(television actress,3)
grunt>


hduser@blessy68: /
grunt> G = order F by cnt desc;
grunt> dump G;


hduser@blessy68: /
ine.util.MapRedUtil - Total input paths to process : 1
(actor,28)
(actress,20)
(comedian,4)
(television actress,3)
(singer,2)
(film actor,2)
(stand-up comedian,2)
(Comedian,2)
(Film director,1)
(film actress,1)
(Film actor,1)
(rock band,1)
(Singer-lyricist,1)
(Stand-up comedian,1)
(Vocalist,1)
(pianist,1)
(writer,1)
(model,1)
(musician,1)


hduser@blessy68: /
grunt> H = limit G 5;
grunt> dump H;

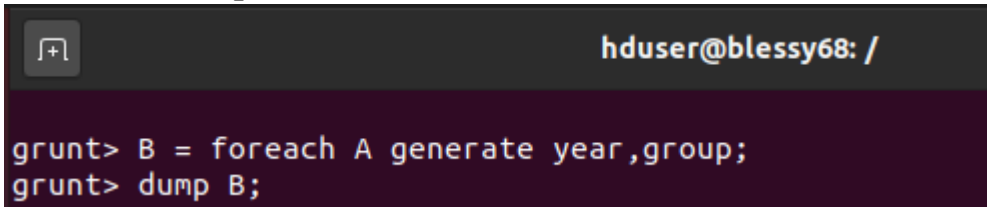


Problem Statement 2:

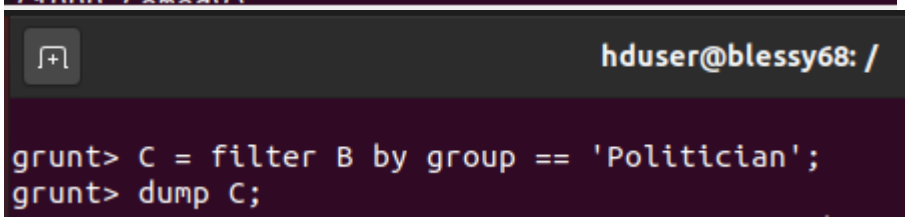
Find out the number of politicians who came each year.

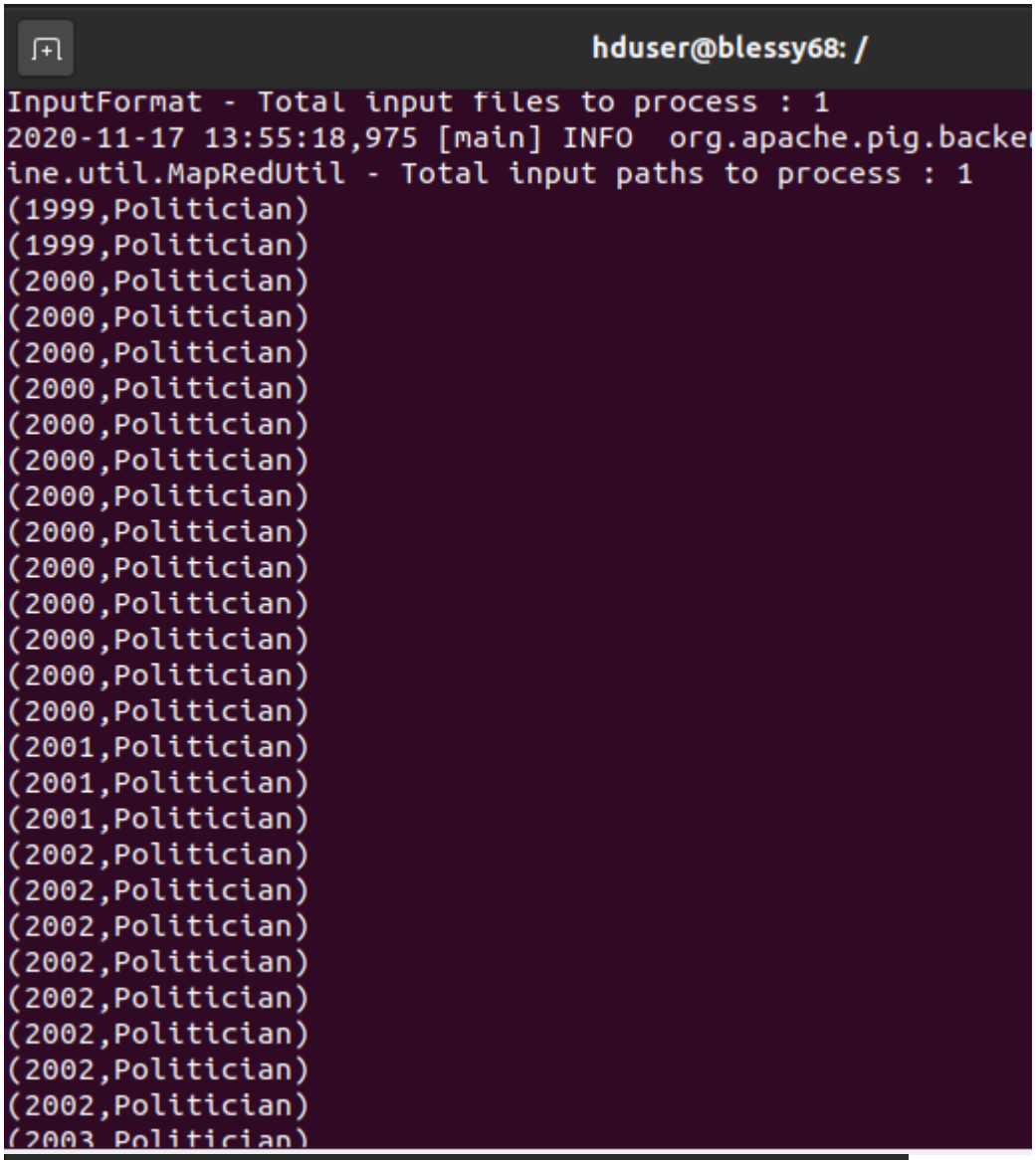
SCRIPT

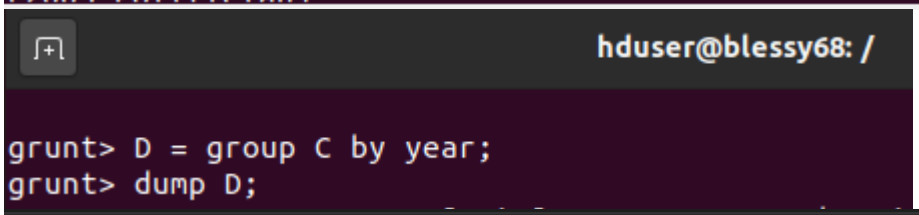
```
A = load '/home/blessy/Downloads/show.csv' using PigStorage(',') AS
(year:chararray,occupation:chararray,date:chararray,group:chararray,
gusetlist:chararray);
B = foreach A generate year,group;
C = filter B by group == 'Politician';
D = group C by year;
E = foreach D generate group, COUNT(C) as cnt;
F = order E by cnt desc;
```

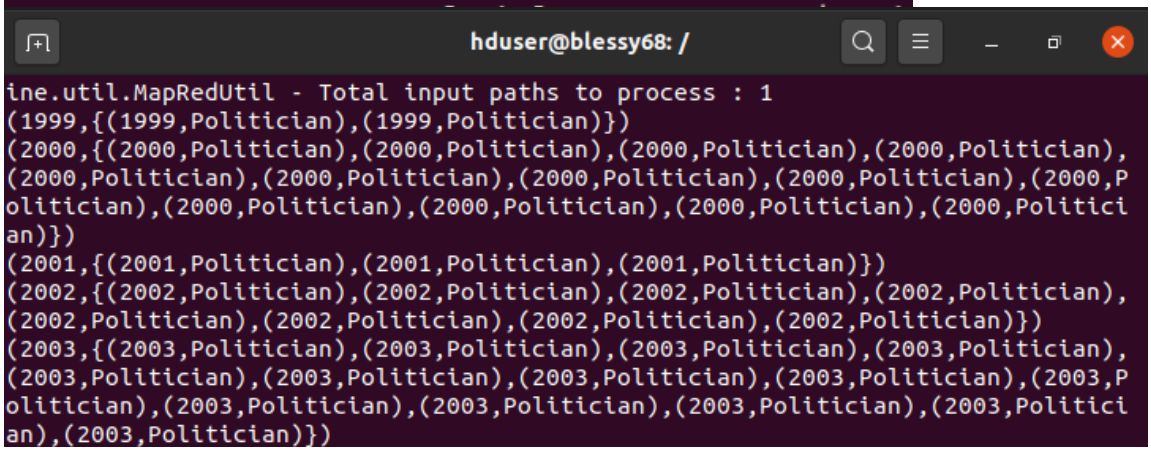


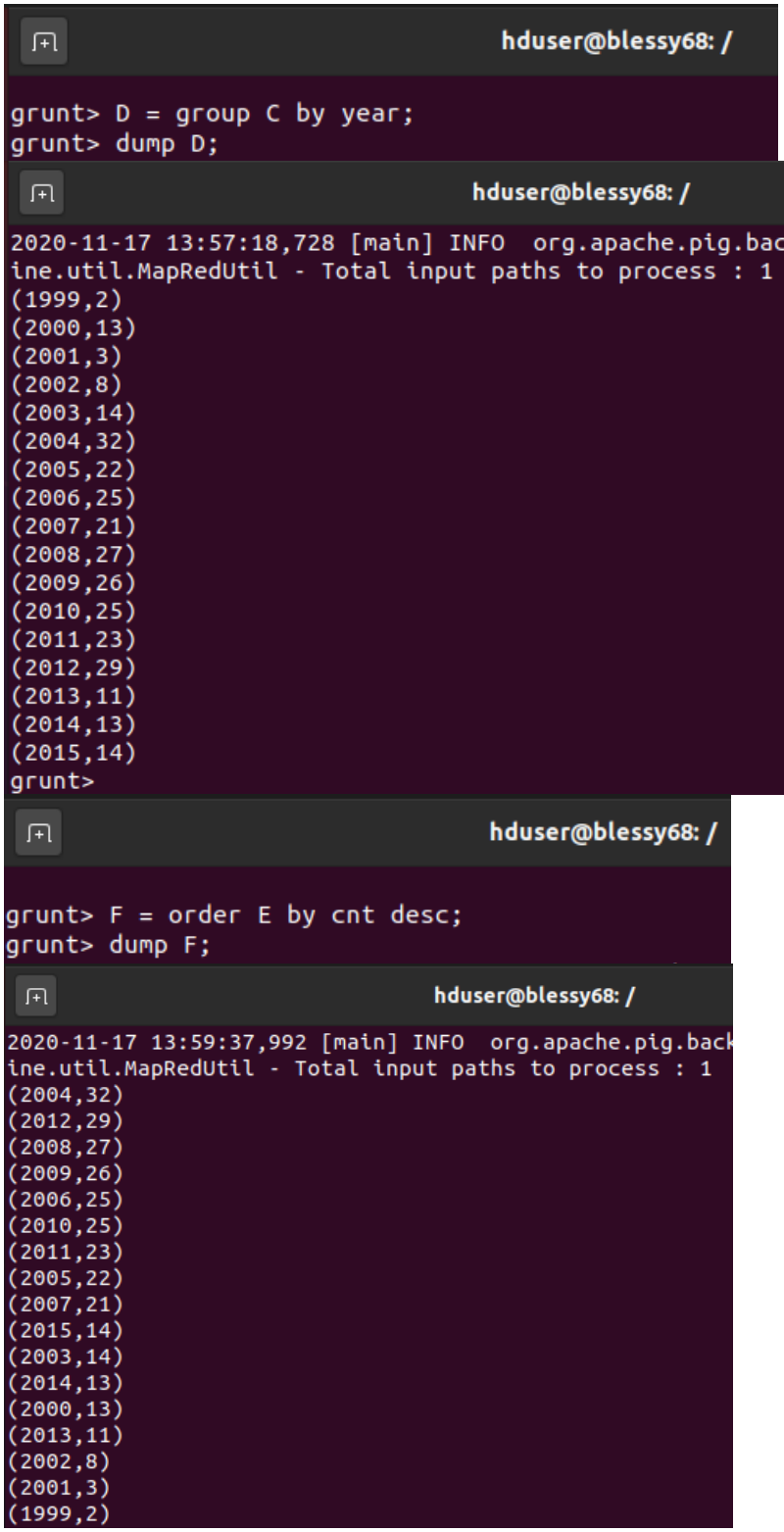

hduser@blessy68: /
(,)
(1999,Acting)
(1999,Comedy)
(1999,Acting)
(1999,Acting)
(1999,Acting)
(1999,Acting)
(1999,Musician)
(1999,Media)
(1999,Acting)
(1999,Comedy)
(1999,Acting)
(1999,Acting)
(1999,Acting)
(1999,Comedy)
(1999,Acting)
(1999,Acting)
(1999,Acting)
(1999,Comedy)
(1999,Acting)
(1999,Musician)
(1999,Acting)
(1999,Media)
(1999,Acting)
(1999,Comedy)
(1999,Acting)
(1999,Acting)
(1999,Musician)
(1999,Acting)


hduser@blessy68: /
grunt> C = filter B by group == 'Politician';
grunt> dump C;


hduser@blessy68: /
InputFormat - Total input files to process : 1
2020-11-17 13:55:18,975 [main] INFO org.apache.pig.backe
ine.util.MapRedUtil - Total input paths to process : 1
(1999,Politician)
(1999,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2000,Politician)
(2001,Politician)
(2001,Politician)
(2001,Politician)
(2002,Politician)
(2002,Politician)
(2002,Politician)
(2002,Politician)
(2002,Politician)
(2002,Politician)
(2002,Politician)
(2002,Politician)
(2003,Politician)


hduser@blessy68: /
grunt> D = group C by year;
grunt> dump D;


hduser@blessy68: /
ine.util.MapRedUtil - Total input paths to process : 1
(1999,{(1999,Politician),(1999,Politician)})
(2000,{(2000,Politician),(2000,Politician),(2000,Politician),(2000,Politician),
(2000,Politician),(2000,Politician),(2000,Politician),(2000,Politician),(2000,P
olitician),(2000,Politician),(2000,Politician),(2000,Politician),(2000,Politici
an)})
(2001,{(2001,Politician),(2001,Politician),(2001,Politician)})
(2002,{(2002,Politician),(2002,Politician),(2002,Politician),(2002,Politician),
(2002,Politician),(2002,Politician),(2002,Politician),(2002,Politician)})
(2003,{(2003,Politician),(2003,Politician),(2003,Politician),(2003,Politician),
(2003,Politician),(2003,Politician),(2003,Politician),(2003,Politician),(2003,P
olitician),(2003,Politician),(2003,Politician),(2003,Politician),(2003,Politici
an),(2003,Politician)})


hduser@blessy68: /
grunt> D = group C by year;
grunt> dump D;
hduser@blessy68: /
2020-11-17 13:57:18,728 [main] INFO org.apache.pig.back
ine.util.MapRedUtil - Total input paths to process : 1
(1999,2)
(2000,13)
(2001,3)
(2002,8)
(2003,14)
(2004,32)
(2005,22)
(2006,25)
(2007,21)
(2008,27)
(2009,26)
(2010,25)
(2011,23)
(2012,29)
(2013,11)
(2014,13)
(2015,14)
grunt>
hduser@blessy68: /
grunt> F = order E by cnt desc;
grunt> dump F;
hduser@blessy68: /
2020-11-17 13:59:37,992 [main] INFO org.apache.pig.back
ine.util.MapRedUtil - Total input paths to process : 1
(2004,32)
(2012,29)
(2008,27)
(2009,26)
(2006,25)
(2010,25)
(2011,23)
(2005,22)
(2007,21)
(2015,14)
(2003,14)
(2014,13)
(2000,13)
(2013,11)
(2002,8)
(2001,3)
(1999,2)

Aviation Data Analysis

The U.S. Department of Transportation's (DOT) Bureau of Transportation Statistics (BTS) tracks the on-time performance of domestic flights operated by large air carriers. Summary information on the number of on-time, delayed, canceled, and diverted flights appears in DOT's monthly Air Travel Consumer Report, published about 30 days after the month's end, as well as in summary tables posted on this website. Summary statistics and raw data are made available to the public at the time the Air Travel Consumer Report is released.

Delayed_Flights.csv

Airports.csv

These are 2 different datasets, i.e., Delayed_Flights.csv and Airports.csv. Let us understand one at a time.

Delayed_Flights.csv Datasets

There are 29 columns in this dataset. Some of them have been mentioned below:

- Year: 1987 – 2008
- Month: 1 – 12
- FlightNum: Flight number
- Canceled: Was the flight canceled?
- CancelleationCode: The reason for cancellation.

Airports.csv Datasets

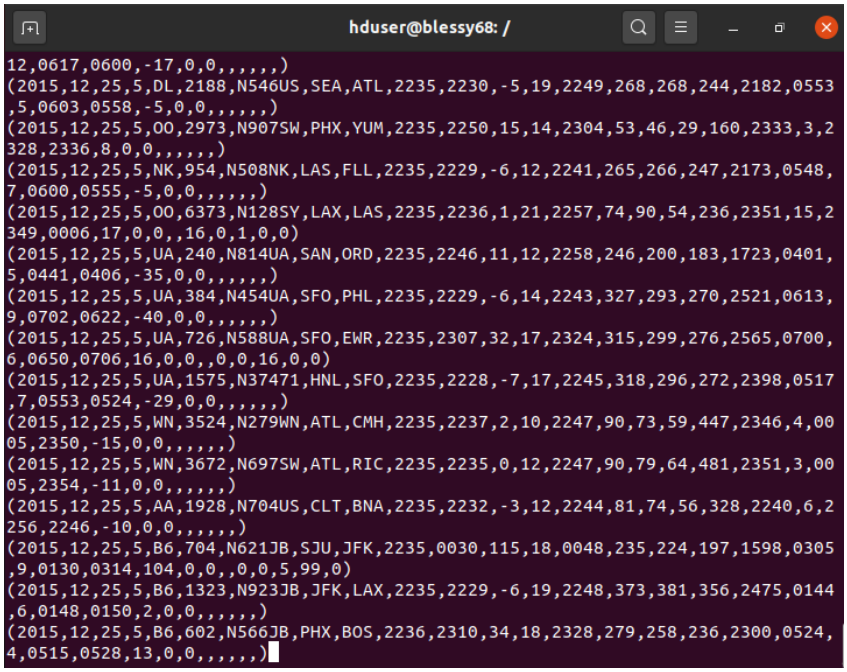
- iata: the international airport abbreviation code
- name of the airport
- city and country in which airport is located.
- lat and long: the latitude and longitude of the airport

Now, using Apache pig, we will try to gain more insights from these datasets.

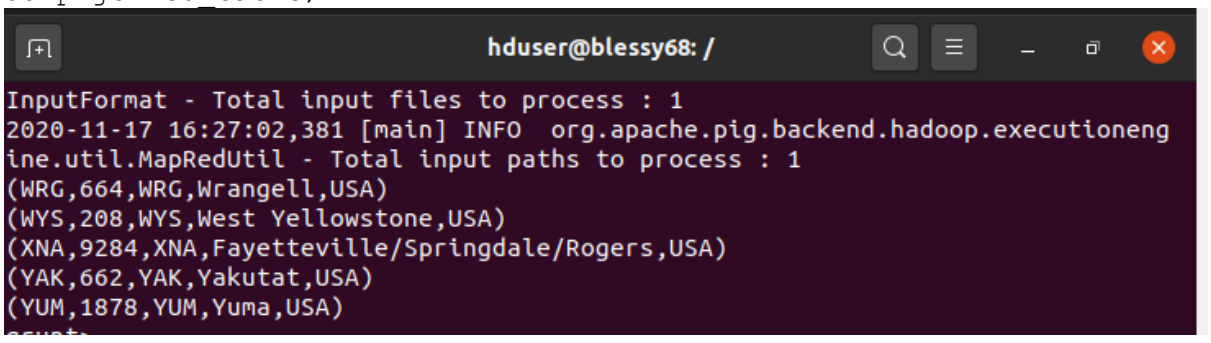
Problem Statement 1

Find out the top 5 most visited destinations.

```
REGISTER '/home/blessy/Downloads/air/piggybank.jar';
A = load '/home/airline_usecase/DelayedFlights.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO_MULTILINE',
'UNIX','SKIP_INPUT_HEADER');
```




```
B = foreach A generate (int)$1 as year, (int)$10 as flight_num,
(chararray)$17 as origin,(chararray) $18 as dest;
C = filter B by dest is not null;
D = group C by dest;
E = foreach D generate group, COUNT(C.dest);
F = order E by $1 DESC;
Result = LIMIT F 5;
A1 = load '/home/acadgild/airline_usecase/airports.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO_MULTILINE',
'UNIX','SKIP_INPUT_HEADER');
A2 = foreach A1 generate (chararray)$0 as dest, (chararray)$2 as
city, (chararray)$4 as country;
joined_table = join Result by $0, A2 by dest;
dump joined_table;
```

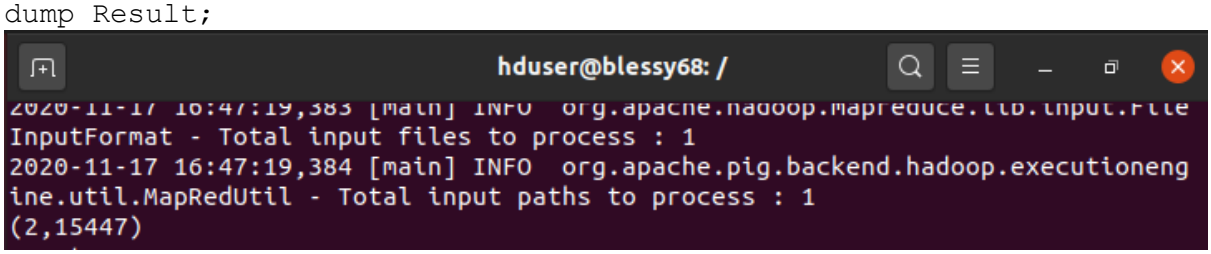


Problem Statement 2

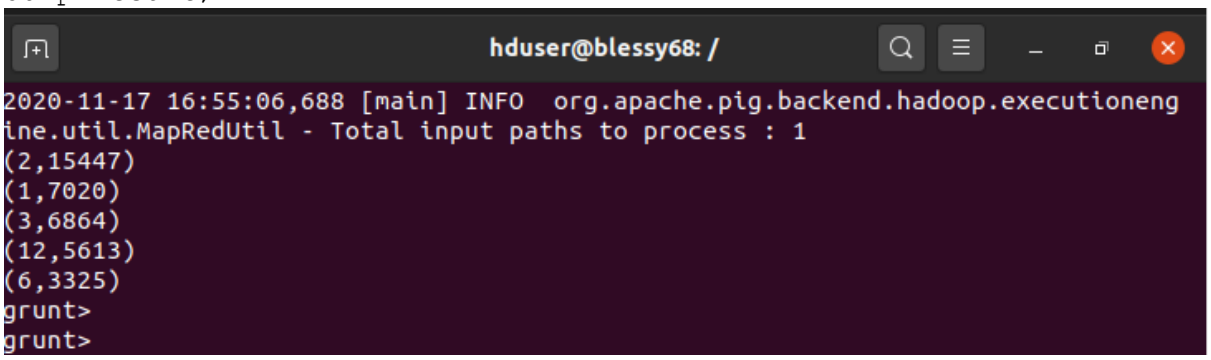
Which month has seen the most number of cancellations due to bad weather?

```
REGISTER '/home/blessy/Downloads/air/piggybank.jar';
A = load '/home/airline_usecase/DelayedFlights.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO_MULTILINE',
'UNIX','SKIP_INPUT_HEADER');
B = foreach A generate (int)$2 as month,(int)$10 as
flight_num,(int)$22 as cancelled,(chararray)$23 as cancel_code;
```

```
C = filter B by cancelled == 1 AND cancel_code =='B';
D = group C by month;
E = foreach D generate group, COUNT(C.cancelled);
F= order E by $1 DESC;
Result = limit F 1;
dump Result;
```




```
Result = limit F 5;
dump Result;
```



Problem Statement 3

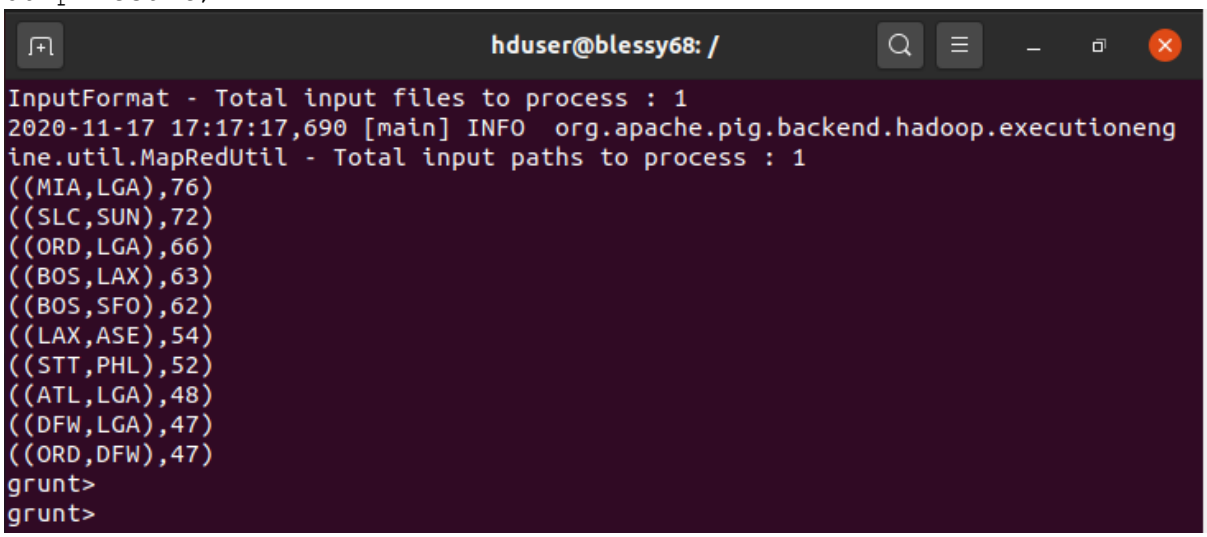
Top ten origins with the highest AVG departure delay

```
REGISTER '/home/blessy/Downloads/air/piggybank.jar';
A = load '/home/acadgild/airline_usecase/DelayedFlights.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO_MULTILINE',
'UNIX','SKIP_INPUT_HEADER');
B1 = foreach A generate (int)$16 as dep_delay, (chararray)$17 as
origin;
C1 = filter B1 by (dep_delay is not null) AND (origin is not null);
D1 = group C1 by origin;
E1 = foreach D1 generate group, AVG(C1.dep_delay);
Result = order E1 by $1 DESC;
Top_ten = limit Result 10;
Lookup = load '/home/airline_usecase/airports.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO_MULTILINE',
'UNIX','SKIP_INPUT_HEADER');
Lookup1 = foreach Lookup generate (chararray)$0 as origin,
(chararray)$2 as city, (chararray)$4 as country;
Joined = join Lookup1 by origin, Top_ten by $0;
Final = foreach Joined generate $0,$1,$2,$4;
Final_Result = ORDER Final by $3 DESC;
dump Final_Result;
```

Problem Statement 4

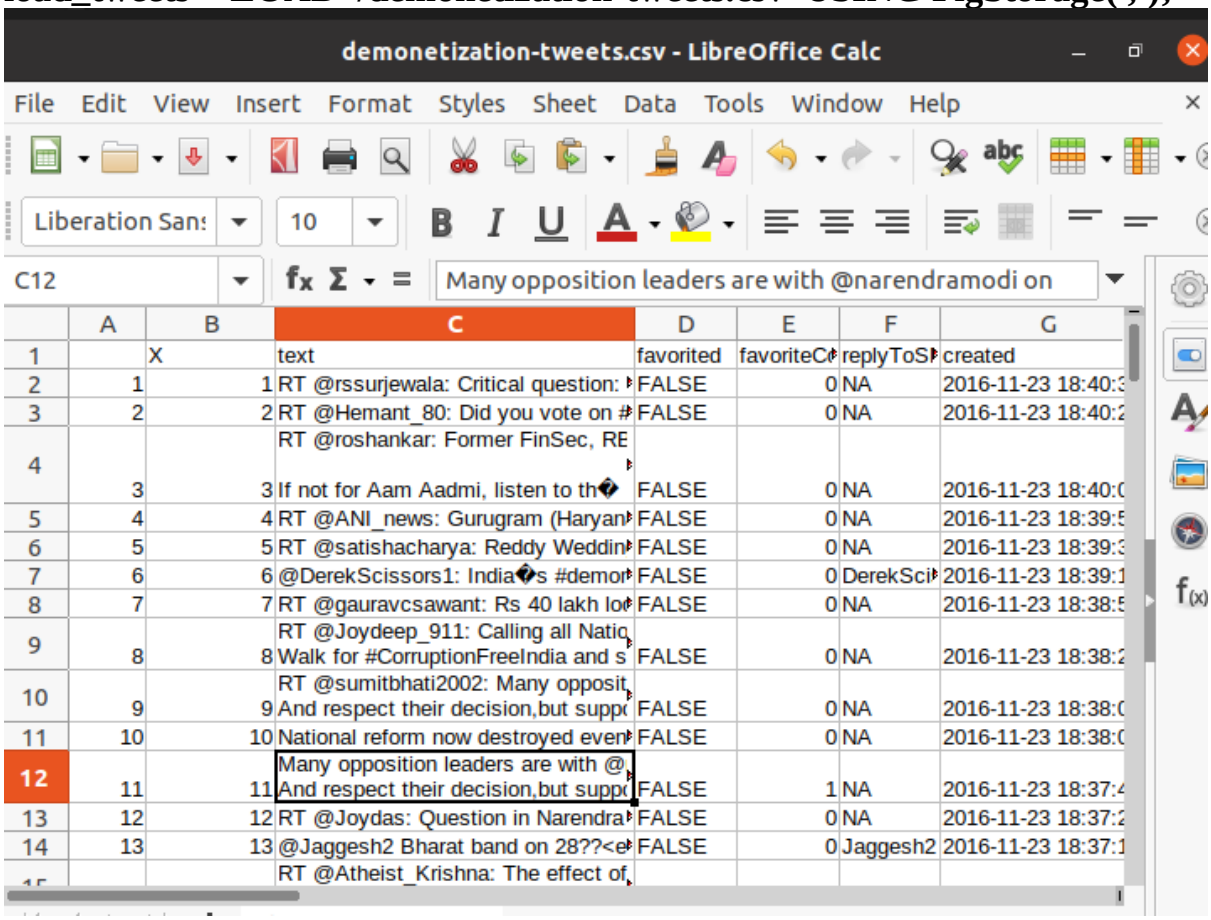
Which route (origin & destination) has seen the maximum diversion?

```
REGISTER '/home/blessy/Downloads/air/piggybank.jar';
A = load '/home/airline_usecase/DelayedFlights.csv' USING
org.apache.pig.piggybank.storage.CSVExcelStorage(',','NO_MULTILINE',
'UNIX','SKIP_INPUT_HEADER');
B = FOREACH A GENERATE (chararray)$17 as origin, (chararray)$18 as
dest, (int)$24 as diversion;
C = FILTER B BY (origin is not null) AND (dest is not null) AND
(diversion == 1);
D = GROUP C by (origin,dest);
E = FOREACH D generate group, COUNT(C.diversion);
F = ORDER E BY $1 DESC;
Result = limit F 10;
dump Result;
```



Sentiment Analysis on Demonetization

Let us find out the views of different people on the demonetization by analysing the tweets from twitter.
Now we will load the data into pig using PigStorage as follows:
load_tweets = LOAD '/demonetization-tweets.csv' USING PigStorage(',');



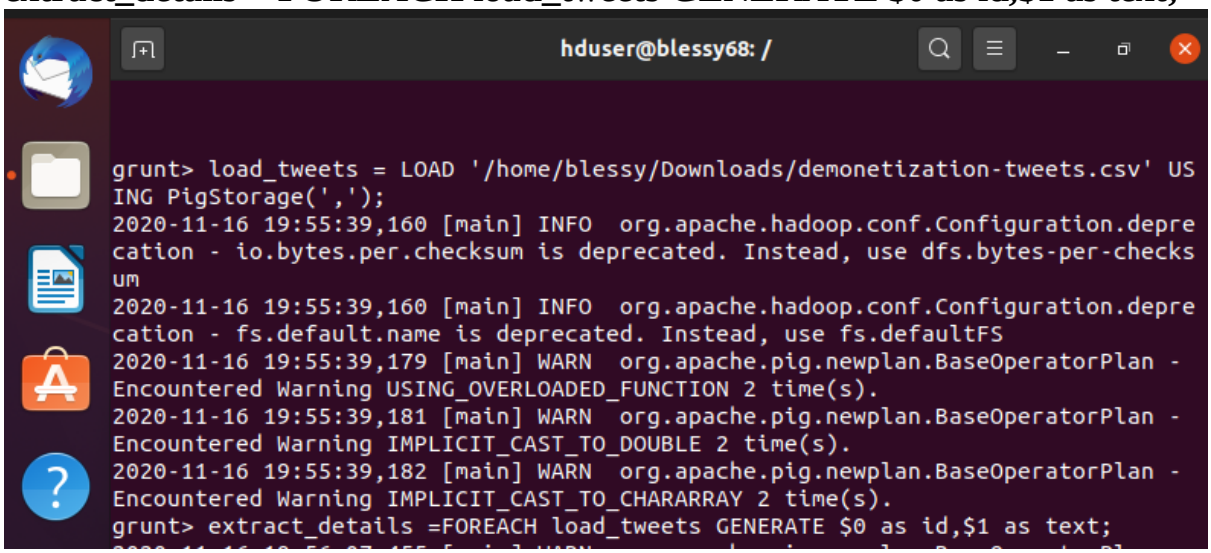
	A	B	C	D	E	F	G
1		X	text	favorited	favoriteCo	replyToSI	created
2	1	1	RT @rssurjewala: Critical question:	FALSE	0	NA	2016-11-23 18:40:3
3	2	2	RT @Hemant_80: Did you vote on #	FALSE	0	NA	2016-11-23 18:40:2
4	3	3	RT @roshankar: Former FinSec, RE If not for Aam Aadmi, listen to th	FALSE	0	NA	2016-11-23 18:40:0
5	4	4	RT @ANI_news: Gurugram (Haryan	FALSE	0	NA	2016-11-23 18:39:5
6	5	5	RT @satishacharya: Reddy Weddin	FALSE	0	NA	2016-11-23 18:39:3
7	6	6	@DerekScissors1: India s #demor	FALSE	0	DerekSci	2016-11-23 18:39:1
8	7	7	RT @gauravcsawant: Rs 40 lakh lo	FALSE	0	NA	2016-11-23 18:38:5
9	8	8	RT @Joydeep_911: Calling all Natio Walk for #CorruptionFreeIndia and s	FALSE	0	NA	2016-11-23 18:38:2
10	9	9	RT @sumitbhati2002: Many opposit And respect their decision,but suppo	FALSE	0	NA	2016-11-23 18:38:0
11	10	10	National reform now destroyed even	FALSE	0	NA	2016-11-23 18:38:0
12	11	11	Many opposition leaders are with @ And respect their decision,but suppo	FALSE	1	NA	2016-11-23 18:37:4
13	12	12	RT @Joydas: Question in Narendra	FALSE	0	NA	2016-11-23 18:37:2
14	13	13	@Jaggesh2 Bharat band on 28??<e	FALSE	0	Jaggesh2	2016-11-23 18:37:1
			RT @Atheist_Krishna: The effect of				

Now after loading successfully, you can see the tweets loaded successfully into pig by using the **dump** command.
Here is the sample tweet
Metadata of the tweets are as follows:

- id
- Text (Tweets)
- favorited
- favoriteCount
- replyToSN
- created
- truncated
- replyToSID
- id

- replyToUID
- statusSource
- screenName
- retweetCount
- isRetweet
- retweeted

Now from this columns, we will extract the **id** and the **tweet_text** as follows
extract_details = FOREACH load_tweets GENERATE $0 as id,$1 as text;



Now if you dump the extracted columns, you will get the id and the tweet_text
Now we will divide the tweet_text into words to calculate the sentiment of the whole tweet.

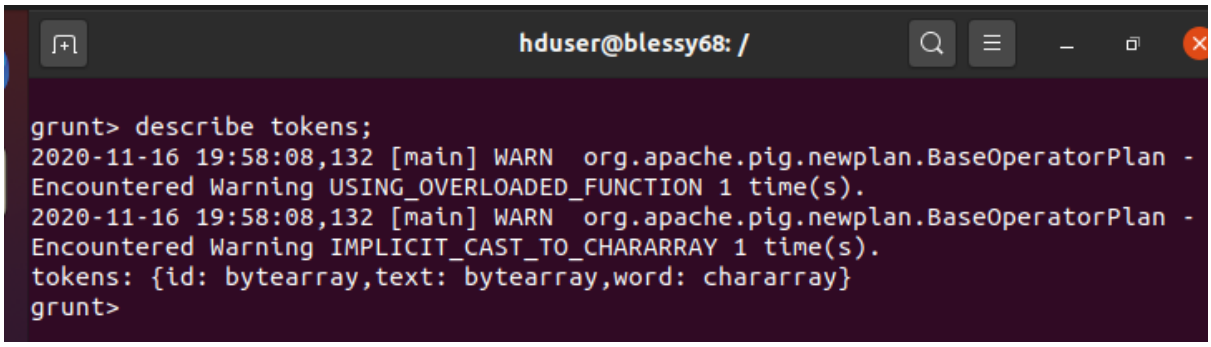
tokens = foreach extract_details generate id,text,
FLATTEN(TOKENIZE(text)) As word;

For every word in the tweet_text, each word will be taken and created as a new row
You can use the dump command to check the same. Here is the sample.

("1","RT @rssurjewala: Critical question: Was PayTM informed about #Demonetization edict by PM? It's clearly fishy and requires full disclosure &

and is as follows:

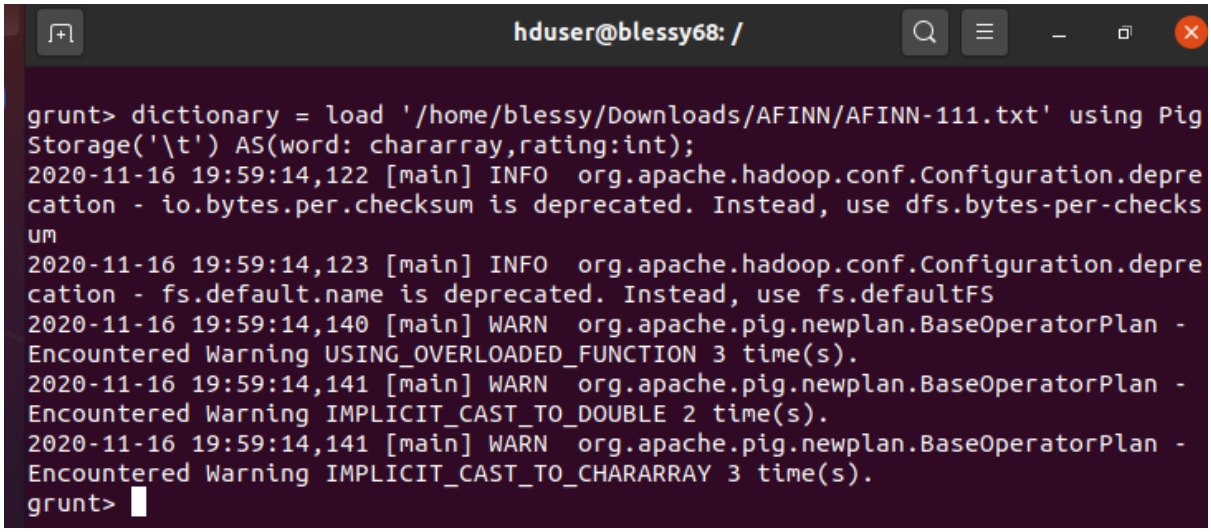
tokens: {id: bytearray,text: bytearray,word: chararray}



Now, we have to analyse the Sentiment for the tweet by using the words in the text. We will rate the word as per its meaning from +5 to -5 using the dictionary AFINN. The AFINN is a dictionary which consists of 2500 words which are rated from +5 to -5 depending on their meaning.

We will load the dictionary into pig by using the below statement:

dictionary = load '/AFINN.txt' using PigStorage('\t') AS(word:chararray,rating:int);



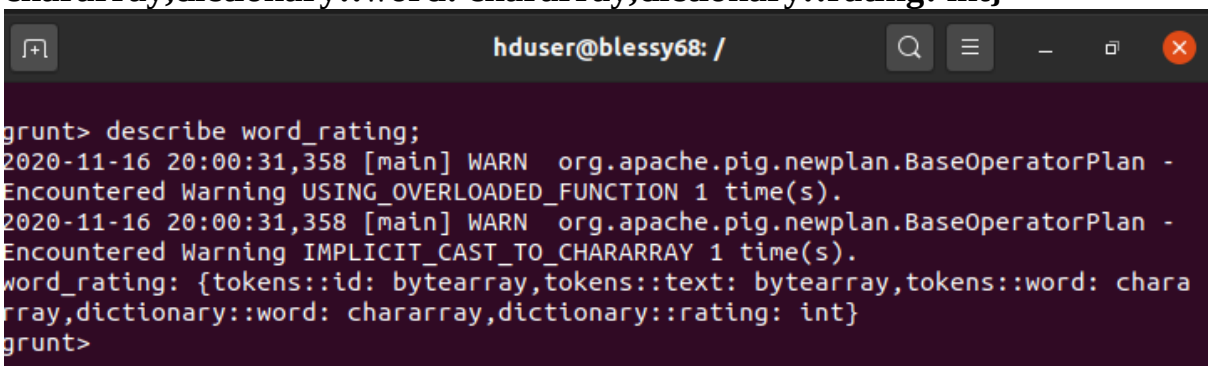
Now, let's perform a map side join by joining the **tokens** statement and the dictionary contents using this relation:

word_rating = join tokens by word left outer, dictionary by word using 'replicated';

We can see the schema of the statement after performing join operation by using the below command:

describe word_rating;

word_rating: {tokens::id: bytearray,tokens::text: bytearray,tokens::word: chararray,dictionary::word: chararray,dictionary::rating: int}



In the above statement **describe word_rating**, we can see that the word_rating has joined the tokens (consists of id, tweet text, word) statement and the dictionary(consists of word, rating).

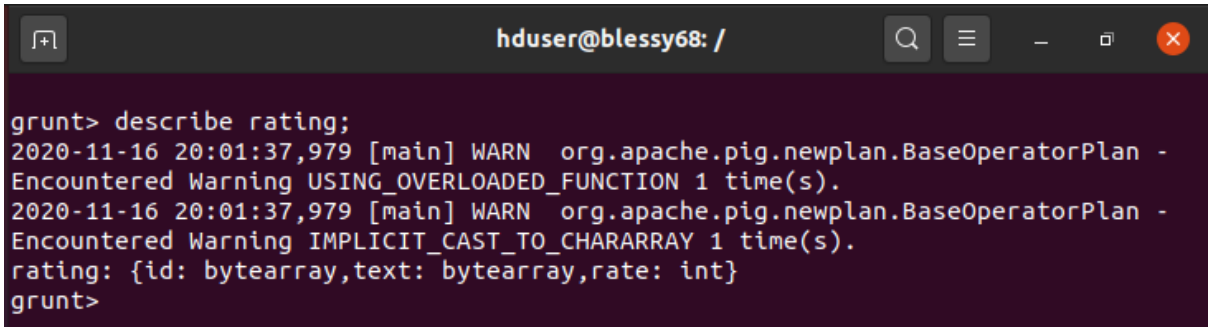
Now we will extract the **id,tweet text** and **word rating(**from the dictionary**)** by using the below relation.

rating = foreach word_rating generate tokens::id as id,tokens::text as text, dictionary::rating as rate;

```
grunt> rating = foreach word_rating generate tokens::id as id, tokens::text as
text,dictionary::rating as rate;
```

We can now see the schema of the relation rating by using the command **describe rating**.

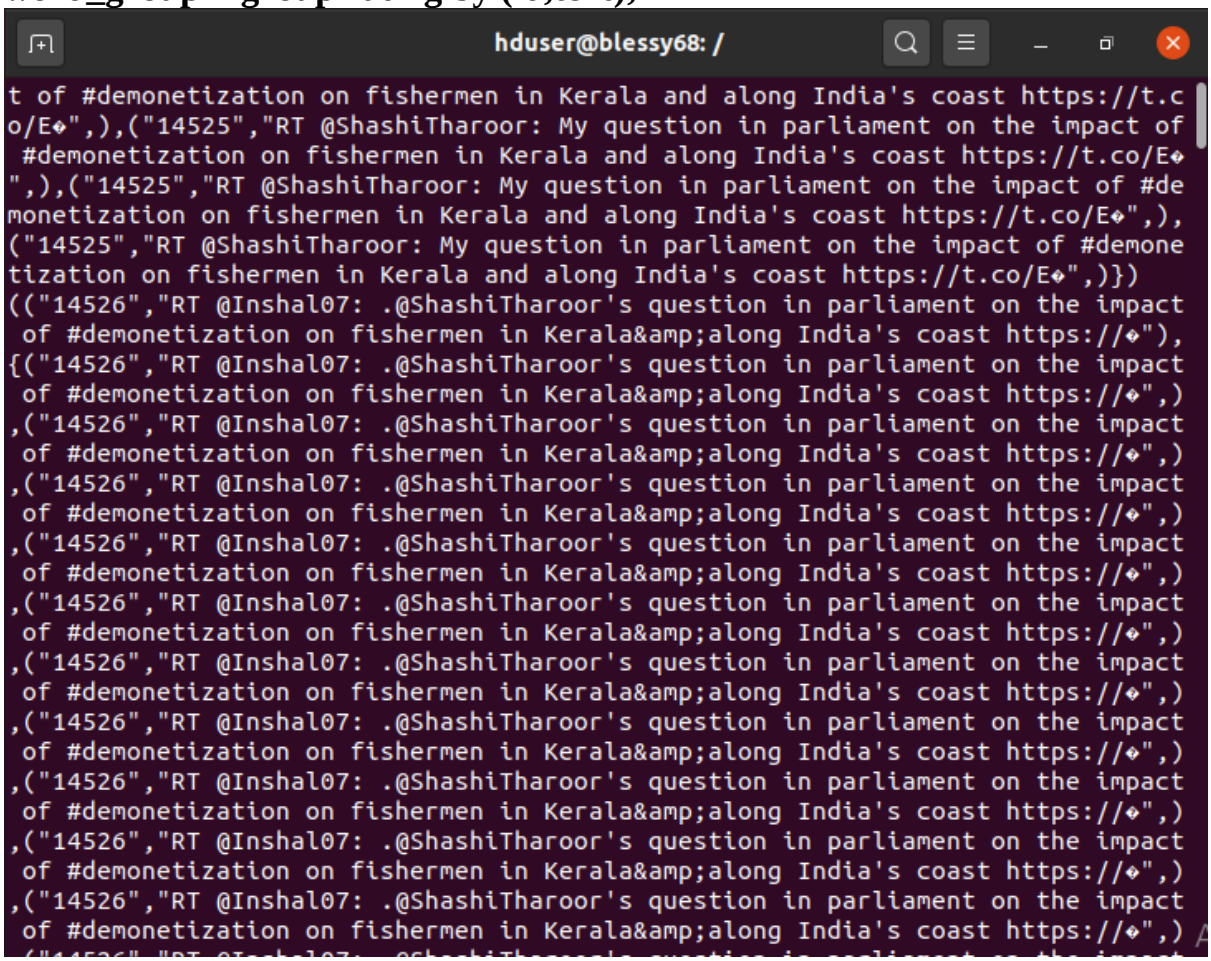
rating: {id: bytearray,text: bytearray,rate: int}



In the above statement **describe rating** we can see that our relation now consists of **id,tweet text** and **rate(**for each word**).**

Now, we will group the **rating of all the words in a tweet** by using the below relation:

word_group = group rating by (id,text);



Here we have grouped by two constraints, **id** and **tweet text**.

Now, let's perform the **Average** operation on the **rating of the words per each tweet**.

avg_rate = foreach word_group generate group, AVG(rating.rate) as tweet_rating;

Now we have calculated the Average rating of the tweet using the rating of each word.

From the above relation, we will get all the tweets i.e., both positive and negative. Here, we can classify the positive tweets by taking the rating of the tweet which can be from **0-5.** We can classify the negative tweets by taking the rating of the tweet from **-5 to -1.**

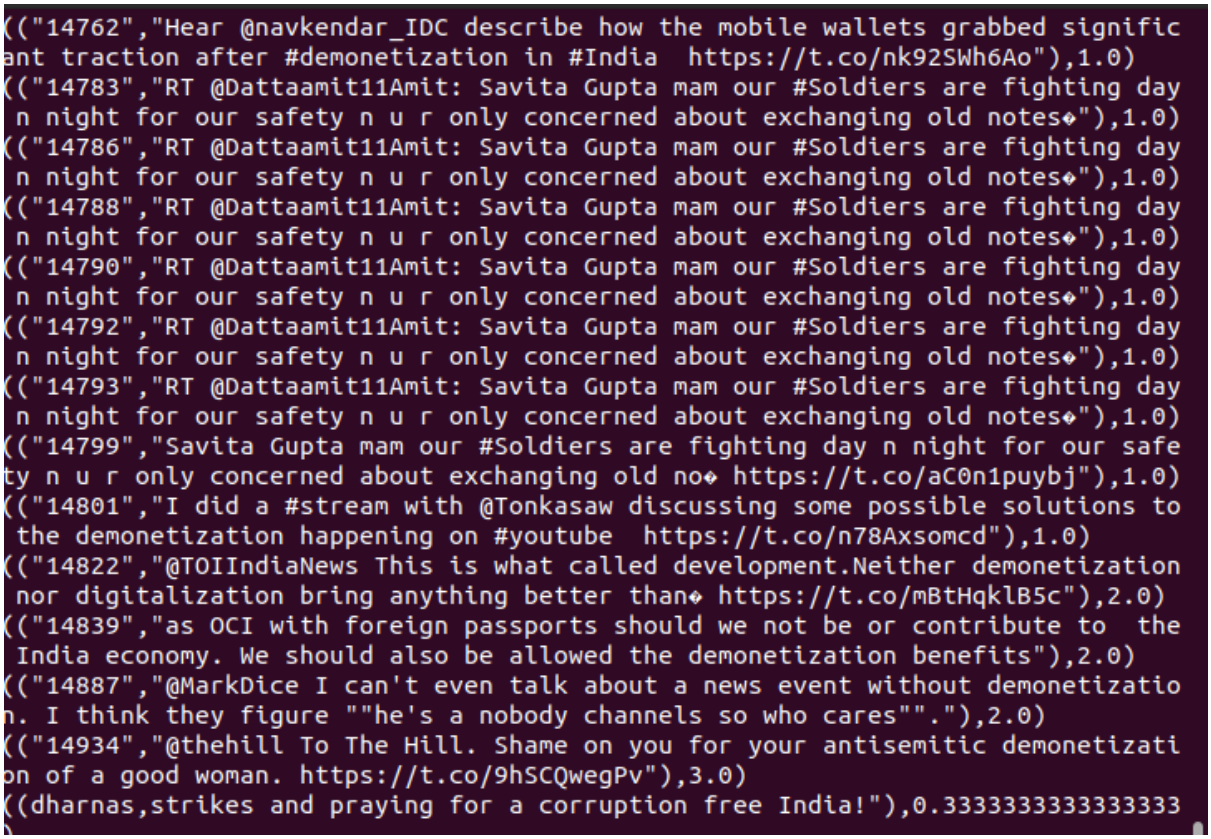
We have now successfully performed the Sentiment Analysis on Twitter data using Pig. We now have the tweets and its rating, so let's perform an operation to filter out the positive tweets.

Now we will filter the positive tweets using the below statement:

positive_tweets = filter avg_rate by tweet_rating>=0;

Output of the tweets with weights >=0;

```
aparty BTW USA's other EVM rigged thug is doing the same thing in Venezuela. Aa
dhaar,),("14893","RT @Stupidosaur: @localteaparty BTW USA's other EVM rigged th
ug is doing the same thing in Venezuela. Aadhaar,),("14893","RT @Stupidosaur: @
localteaparty BTW USA's other EVM rigged thug is doing the same thing in Venezu
ela. Aadhaar,),("14893","RT @Stupidosaur: @localteaparty BTW USA's other EVM ri
gged thug is doing the same thing in Venezuela. Aadhaar,),("14893","RT @Stupido
saur: @localteaparty BTW USA's other EVM rigged thug is doing the same thing in
 Venezuela. Aadhaar,),("14893","RT @Stupidosaur: @localteaparty BTW USA's other
 EVM rigged thug is doing the same thing in Venezuela. Aadhaar,),("14893","RT @
Stupidosaur: @localteaparty BTW USA's other EVM rigged thug is doing the same t
hing in Venezuela. Aadhaar,),("14893","RT @Stupidosaur: @localteaparty BTW USA'
s other EVM rigged thug is doing the same thing in Venezuela. Aadhaar,),("14893
","RT @Stupidosaur: @localteaparty BTW USA's other EVM rigged thug is doing the
 same thing in Venezuela. Aadhaar,),("14893","RT @Stupidosaur: @localteaparty B
TW USA's other EVM rigged thug is doing the same thing in Venezuela. Aadhaar,),
("14893","RT @Stupidosaur: @localteaparty BTW USA's other EVM rigged thug is do
ing the same thing in Venezuela. Aadhaar,),("14893","RT @Stupidosaur: @localtea
party BTW USA's other EVM rigged thug is doing the same thing in Venezuela. Aad
haar,),("14893","RT @Stupidosaur: @localteaparty BTW USA's other EVM rigged thu
g is doing the same thing in Venezuela. Aadhaar,),("14893","RT @Stupidosaur: @l
ocalteaparty BTW USA's other EVM rigged thug is doing the same thing in Venezue
la. Aadhaar,),("14893","RT @Stupidosaur: @localteaparty BTW USA's other EVM rig
ged thug is doing the same thing in Venezuela. Aadhaar,),("14893","RT @Stupidos
aur: @localteaparty BTW USA's other EVM rigged thug is doing the same thing in
Venezuela. Aadhaar,-1),("14893","RT @Stupidosaur: @localteaparty BTW USA's othe
r EVM rigged thug is doing the same thing in Venezuela. Aadhaar,)})
(("14894","RT @rahulroushan: Prohibition is Demonetization 2.0: You kept quiet
when poor men's tobacco-gutkha was banned),{("14894","RT @rahulroushan: Prohibi
tion is Demonetization 2.0: You kept quiet when poor men's tobacco-gutkha was b
```



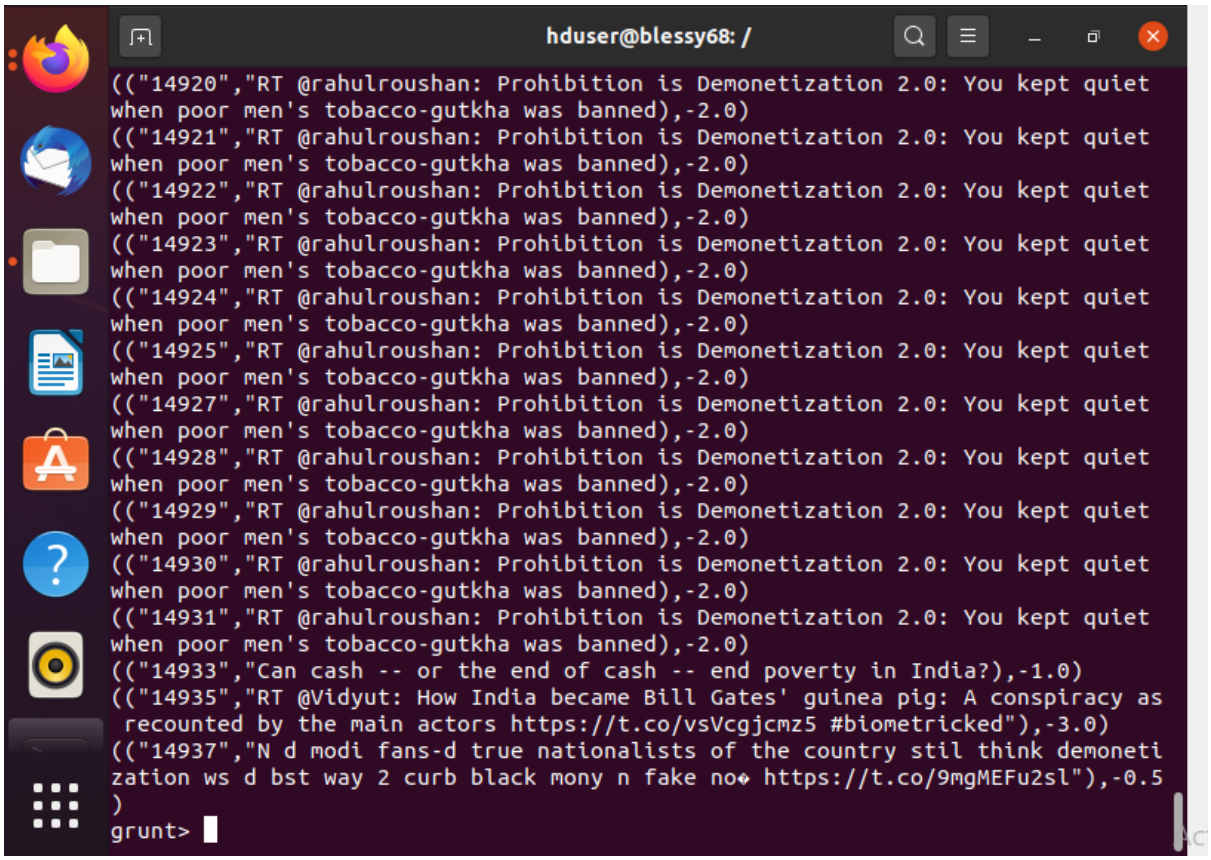
Here are the sample tweets with positive ratings.

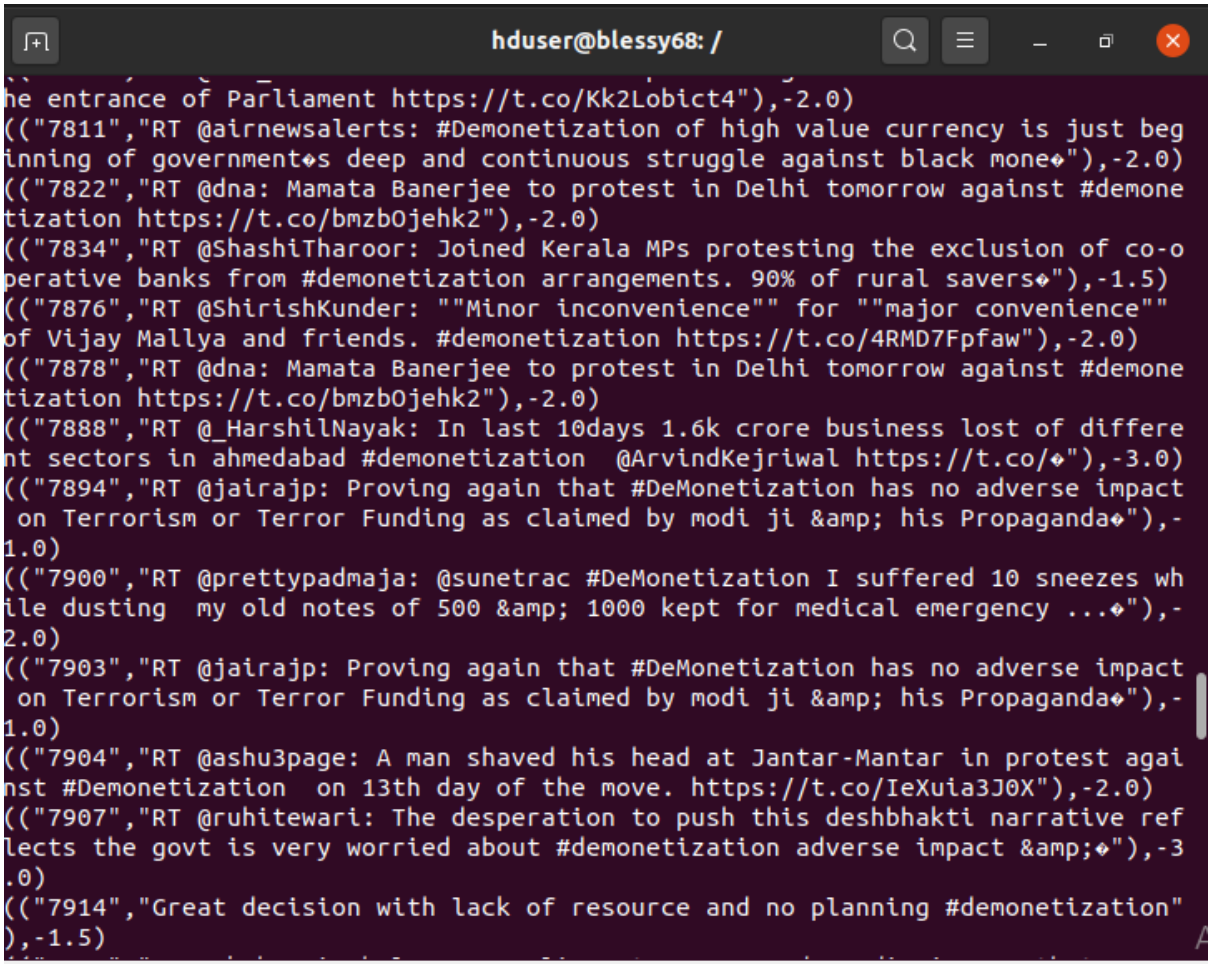
Like this we will also filter the negative tweets as follows:

negative_tweets = filter avg_rate by tweet_rating<0;

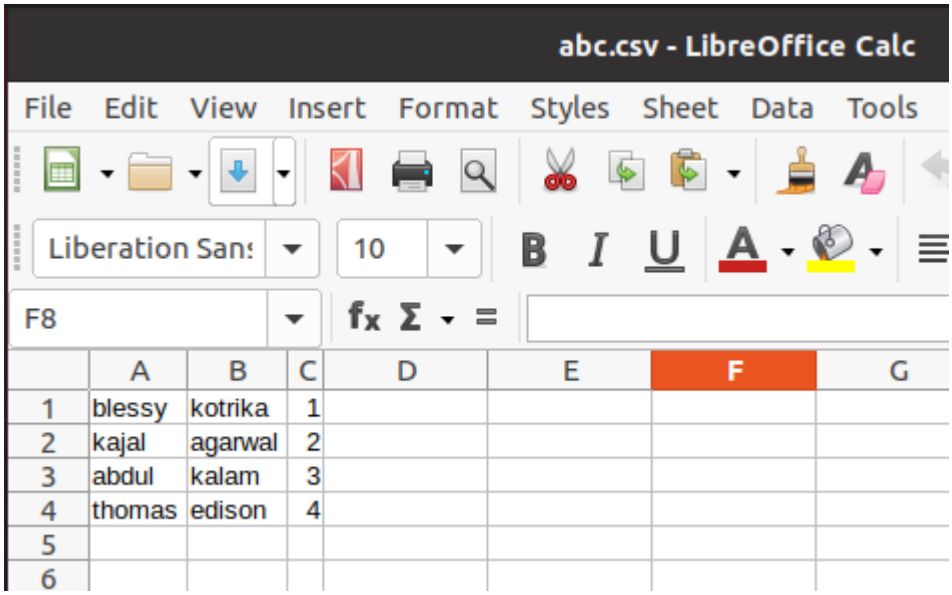
Like this, you can perform sentiment analysis using Pig.

Output of the tweets with weights <0;




hduser@blessy68: /
he entrance of Parliament https://t.co/Kk2Lobict4"),-2.0)
(("7811","RT @airnewsalerts: #Demonetization of high value currency is just beg
inning of government◆s deep and continuous struggle against black mone◆"),-2.0)
(("7822","RT @dna: Mamata Banerjee to protest in Delhi tomorrow against #demone
tization https://t.co/bmzbOjehk2"),-2.0)
(("7834","RT @ShashiTharoor: Joined Kerala MPs protesting the exclusion of co-o
perative banks from #demonetization arrangements. 90% of rural savers◆"),-1.5)
(("7876","RT @ShirishKunder: ""Minor inconvenience"" for ""major convenience""
of Vijay Mallya and friends. #demonetization https://t.co/4RMD7Fpfaw"),-2.0)
(("7878","RT @dna: Mamata Banerjee to protest in Delhi tomorrow against #demone
tization https://t.co/bmzbOjehk2"),-2.0)
(("7888","RT @_HarshilNayak: In last 10days 1.6k crore business lost of differe
nt sectors in ahmedabad #demonetization @ArvindKejriwal https://t.co/◆"),-3.0)
(("7894","RT @jairajp: Proving again that #DeMonetization has no adverse impact
 on Terrorism or Terror Funding as claimed by modi ji & his Propaganda◆"),-
1.0)
(("7900","RT @prettypadmaja: @sunetrac #DeMonetization I suffered 10 sneezes wh
ile dusting my old notes of 500 & 1000 kept for medical emergency ...◆"),-
2.0)
(("7903","RT @jairajp: Proving again that #DeMonetization has no adverse impact
 on Terrorism or Terror Funding as claimed by modi ji & his Propaganda◆"),-
1.0)
(("7904","RT @ashu3page: A man shaved his head at Jantar-Mantar in protest agai
nst #Demonetization on 13th day of the move. https://t.co/IeXuia3J0X"),-2.0)
(("7907","RT @ruhitewari: The desperation to push this deshbhakti narrative ref
lects the govt is very worried about #demonetization adverse impact &◆"),-3
.0)
(("7914","Great decision with lack of resource and no planning #demonetization"
),-1.5)

PIG- UDF



pig -x local
register '/home/blessy/Desktop/test1.py' using org.apache.pig.scripting.jython.JythonScriptEngine as blessy;

a = load '/home/blessy/Desktop/abc.csv' using PigStorage(',') as (fn:chararray, ln:chararray,num:int);
b = foreach a generate blessy.concat(fn,ln);
dump b;
b = foreach a generate blessy.square(num);

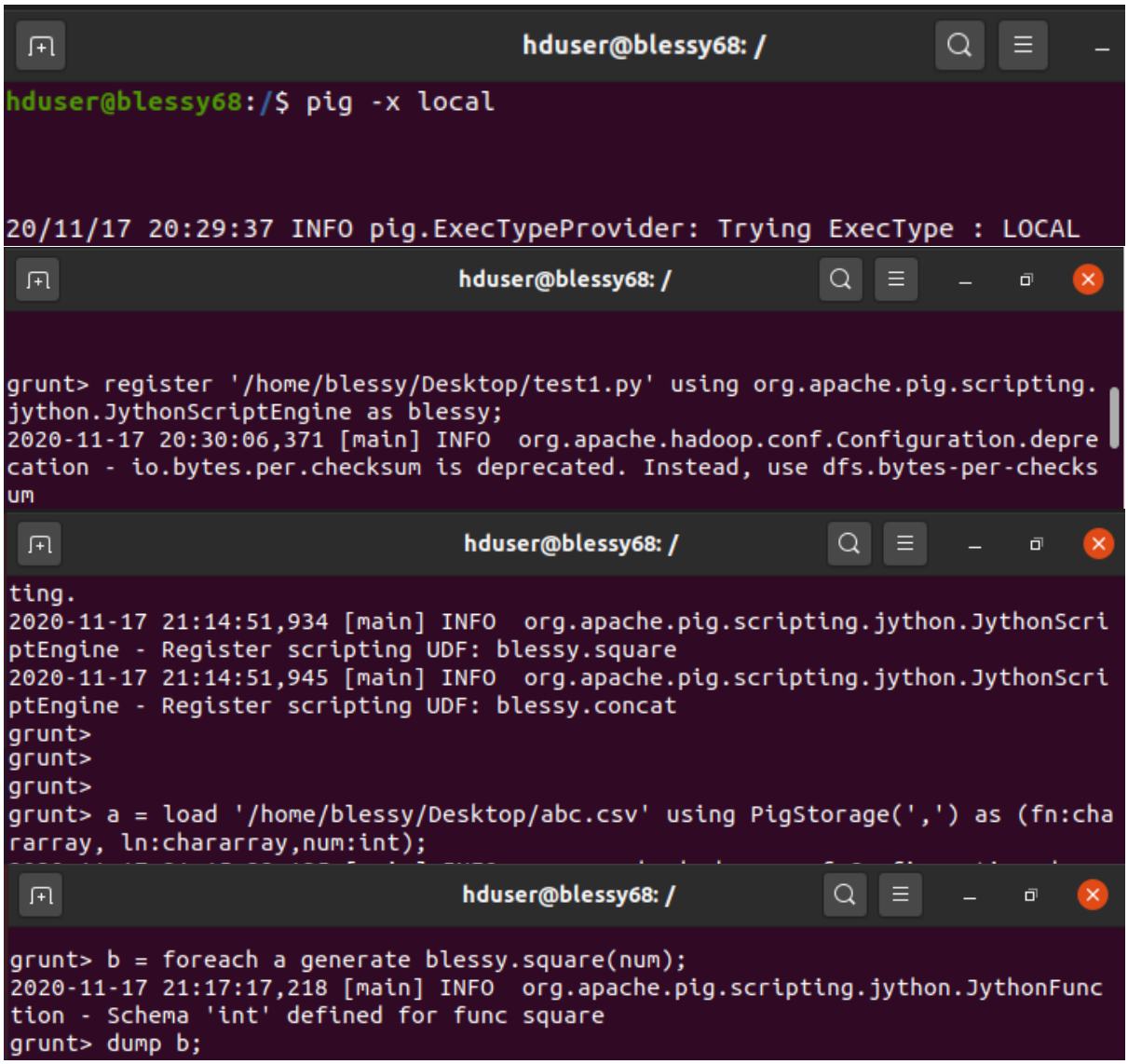
Open test1.py ~/Desktop

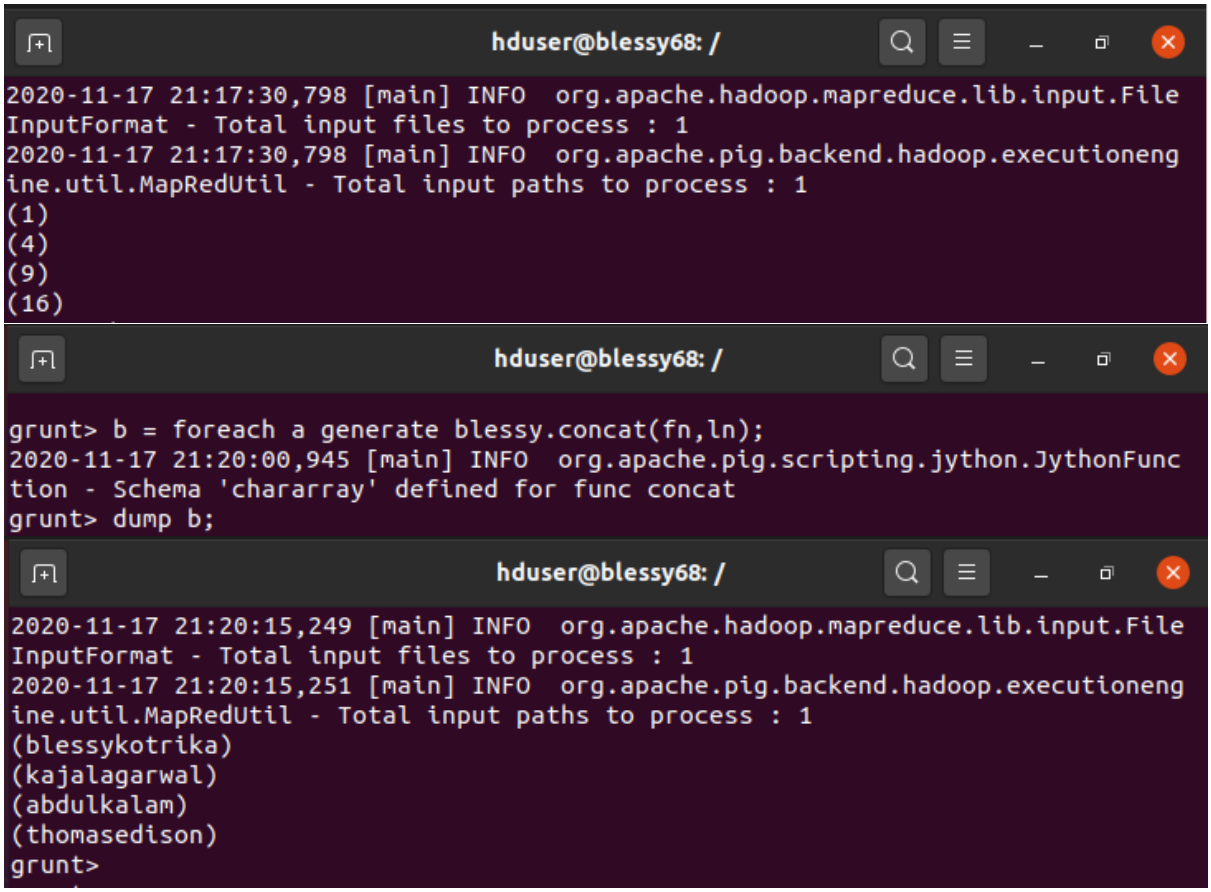
```
1 #from pigutil import outputSchema
2
3 @outputSchema("int")
4 def square(num):
5     if num == None:
6         return None
7     return ((num) * (num))
8
9 @outputSchema("chararray")
10 def concat(a,b):
11     return a +b
```

dump b;

PYTHON SCRIPT

```
#from pigutil import outputSchema

@outputSchema("int")
def square(num):
    if num == None:
        return None
    return ((num) * (num))

@outputSchema("chararray")
def concat(a,b):
    return a +b
```



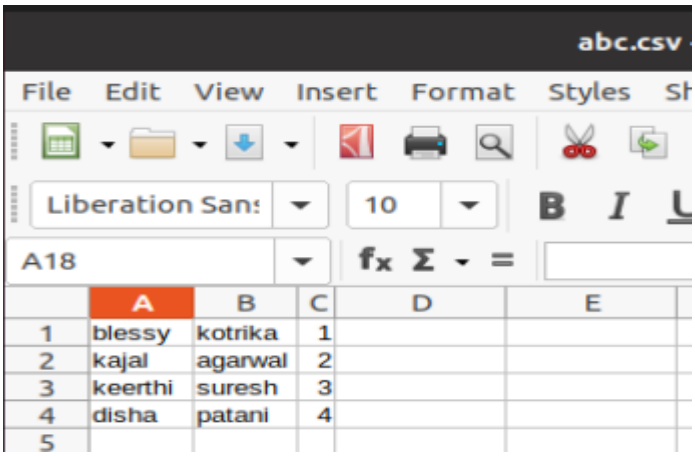

hduser@blessy68: /
2020-11-17 21:17:30,798 [main] INFO org.apache.hadoop.mapreduce.lib.input.File
InputFormat - Total input files to process : 1
2020-11-17 21:17:30,798 [main] INFO org.apache.pig.backend.hadoop.executioneng
ine.util.MapRedUtil - Total input paths to process : 1
(1)
(4)
(9)
(16)
hduser@blessy68: /
grunt> b = foreach a generate blessy.concat(fn,ln);
2020-11-17 21:20:00,945 [main] INFO org.apache.pig.scripting.jython.JythonFunc
tion - Schema 'chararray' defined for func concat
grunt> dump b;
hduser@blessy68: /
2020-11-17 21:20:15,249 [main] INFO org.apache.hadoop.mapreduce.lib.input.File
InputFormat - Total input files to process : 1
2020-11-17 21:20:15,251 [main] INFO org.apache.pig.backend.hadoop.executioneng
ine.util.MapRedUtil - Total input paths to process : 1
(blessykotrika)
(kajalagarwal)
(abdulkalam)
(thomasedison)
grunt>

Hive UDF

Code:

```
CREATE table bda(fn STRING, ln STRING, ids int)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/home/blessy/Downloads/abc.csv' OVERWRITE
INTO TABLE bda;
add FILE /home/blessy/Downloads/test.py;

SELECT TRANSFORM(fn,ln,ids) USING 'python3
/home/blessy/Downloads/test.py' AS (fn,ln,ids) FROM bda;
```

abc.csv



test.py

```
1 import sys
2 import datetime
3 for line in sys.stdin:
4         fn,ln,ids = line.strip().split()
5         fn = fn.upper()
6         ln = ln.upper()
7         ids = int(ids)
8         print(str(fn)+"\t"+str(ln)+"\t"+str(ids*ids)+"\t")
```

```
hive> CREATE table bda(fn STRING,ln STRING, ids int)
    > ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
OK
Time taken: 3.067 seconds
hive> LOAD DATA LOCAL INPATH '/home/blessy/Downloads/abc.txt' OVERWRITE INTO TA
BLE bda;
Loading data to table default.bda
OK
Time taken: 2.779 seconds
```

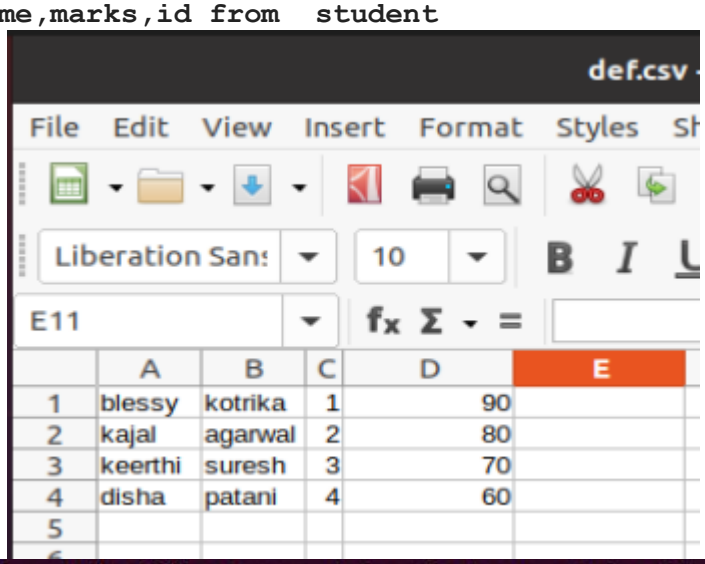
```
hive> add FILE /home/blessy/Downloads/test.py;
Added resources: [/home/blessy/Downloads/test.py]
hive> SELECT TRANSFORM(fn,ln,ids) USING 'python3 /home/blessy/Downloads/test.py
' AS (fn,ln,ids) FROM bda;
```

```
hive> SELECT TRANSFORM(fn,ln,ids) USING 'python3 /home/blessy/Downloads/test.py
' AS (fn,ln,ids) FROM bda;
Query ID = hduser_20201121123519_0d7eb0d2-532a-4183-b703-6cf7fe5f77dc
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Job running in-process (local Hadoop)
2020-11-21 12:35:22,015 Stage-1 map = 100%,  reduce = 0%
Ended Job = job_local193232085_0013
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 848 HDFS Write: 133 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
BLESSY  KOTRIKA 1
KAJAL   AGARWAL 4
KEERTHI SURESH  9
DISHA   PATANI  16
Time taken: 2.369 seconds, Fetched: 4 row(s)
hive>
```

Hive partitioning

Code:

```
create table student(firstname string,lastname string,id int, marks
int)
row format delimited fields terminated by ',';

Load data local inpath '/home/blessy/Downloads/def.csv' into table
student;
create table stud_part(firstname string,marks string) PARTITIONED
BY(id int);

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

INSERT OVERWRITE TABLE stud_part PARTITION(id)
SELECT firstname,marks,id from  student
```




```
hive> create table student(firstname string,lastname string,id int, marks int)
    > row format delimited fields terminated by ',';
OK
Time taken: 9.505 seconds
```

```
    > Load data local inpath '/home/blessy/Downloads/def.csv' into table studen
t;
Loading data to table default.student
OK
Time taken: 1.039 seconds
hive>
```

```
hive> set hive.exec.dynamic.partition.mode=nonstrict
    >
```

```
    > create table stud_part(firstname string,marks string) PARTITIONED BY(id i
nt);
```

hduser@blessy68: /

```
    >
    > INSERT OVERWRITE TABLE stud_part PARTITION(id)
    > SELECT firstname,marks,id from  student;
Query ID = hduser_20201121125741_543fb90a-e353-4a79-aabe-a14443ca59de
Total jobs = 3
Launching Job 1 out of 3
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>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-11-21 12:58:03,824 Stage-1 map = 0%,  reduce = 0%
2020-11-21 12:58:16,054 Stage-1 map = 100%,  reduce = 0%
2020-11-21 12:58:18,161 Stage-1 map = 100%,  reduce = 100%
Ended Job = job_local1787775788_0001
Stage-4 is selected by condition resolver.
Stage-3 is filtered out by condition resolver.
Stage-5 is filtered out by condition resolver.
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/stud_part/.h
ive-staging_hive_2020-11-21_12-57-41_176_8094420143561531135-1/-ext-10000
Loading data to table default.stud_part partition (id=null)


        Time taken to load dynamic partitions: 8.271 seconds
        Time taken for adding to write entity : 0.035 seconds
MapReduce Jobs Launched:
```

```
Moving data to directory hdfs://localhost:9000/user/hive/warehouse/stud_part/.h
ive-staging_hive_2020-11-21_12-57-41_176_8094420143561531135-1/-ext-10000
Loading data to table default.stud_part partition (id=null)


        Time taken to load dynamic partitions: 8.271 seconds
        Time taken for adding to write entity : 0.035 seconds
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 308 HDFS Write: 866 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 50.414 seconds
hive>
    >
    > hduser@blessy68:/$
```

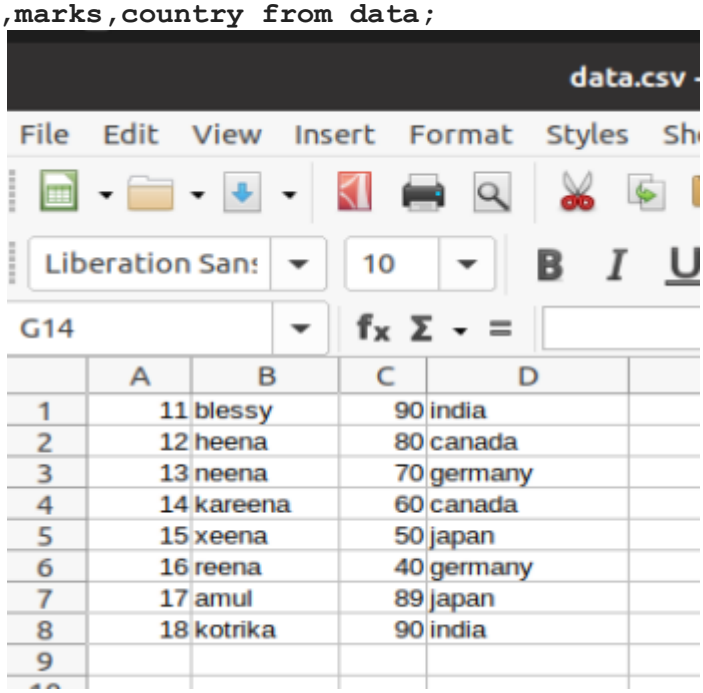
```
hduser@blessy68:/$ hadoop fs -ls /user/hive/warehouse/stud-part
20/11/21 13:00:25 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
ls: `/user/hive/warehouse/stud-part': No such file or directory
hduser@blessy68:/$ hadoop fs -ls /user/hive/warehouse/stud_part
20/11/21 13:00:41 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
Found 4 items
drwxr-xr-x   - hduser supergroup          0 2020-11-21 12:58 /user/hive/warehou
se/stud_part/id=1
drwxr-xr-x   - hduser supergroup          0 2020-11-21 12:58 /user/hive/warehou
se/stud_part/id=2
drwxr-xr-x   - hduser supergroup          0 2020-11-21 12:58 /user/hive/warehou
se/stud_part/id=3
drwxr-xr-x   - hduser supergroup          0 2020-11-21 12:58 /user/hive/warehou
se/stud_part/id=4
hduser@blessy68:/$
```

Four partitions are created based on four IDs

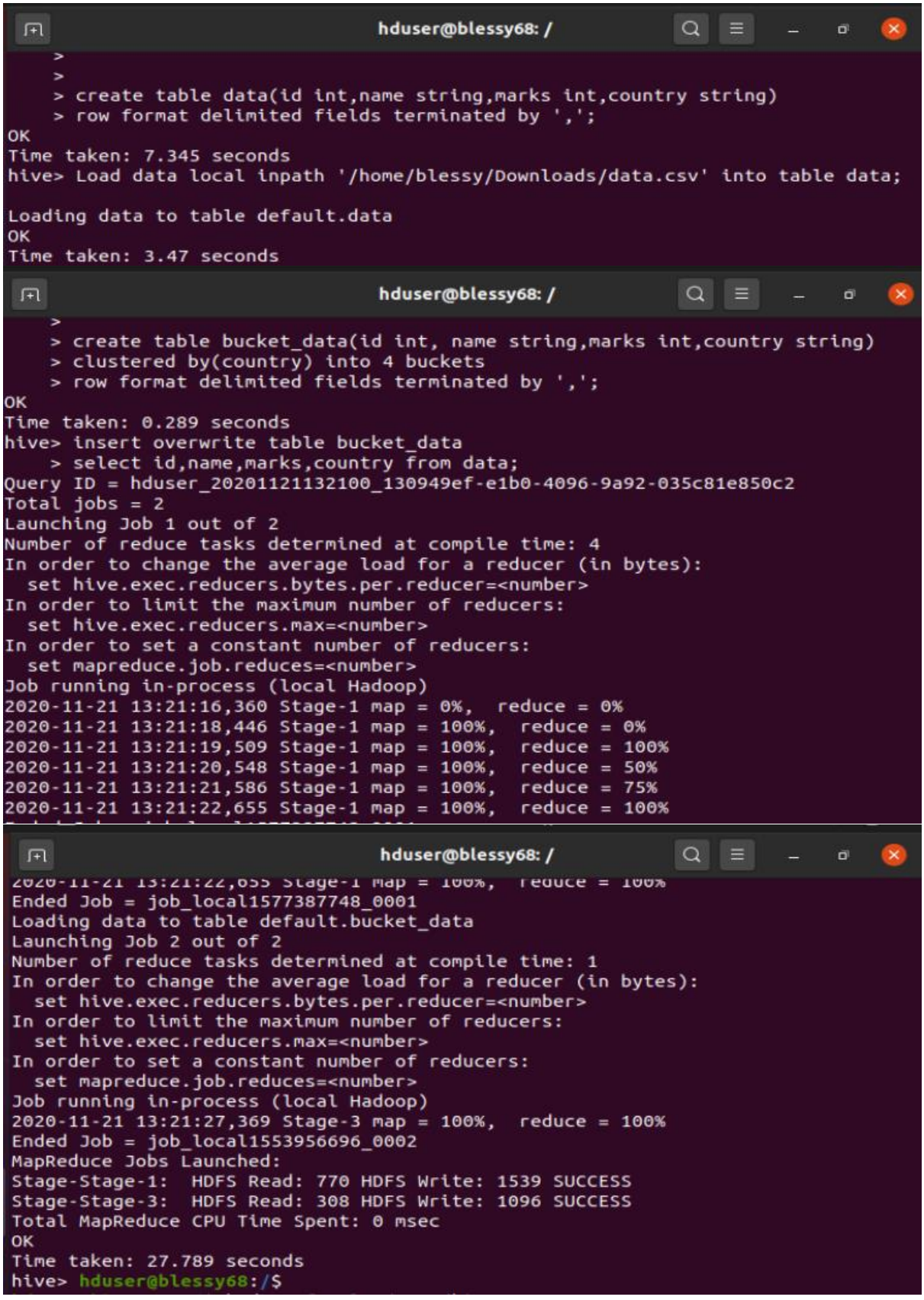
Hive Bucketing

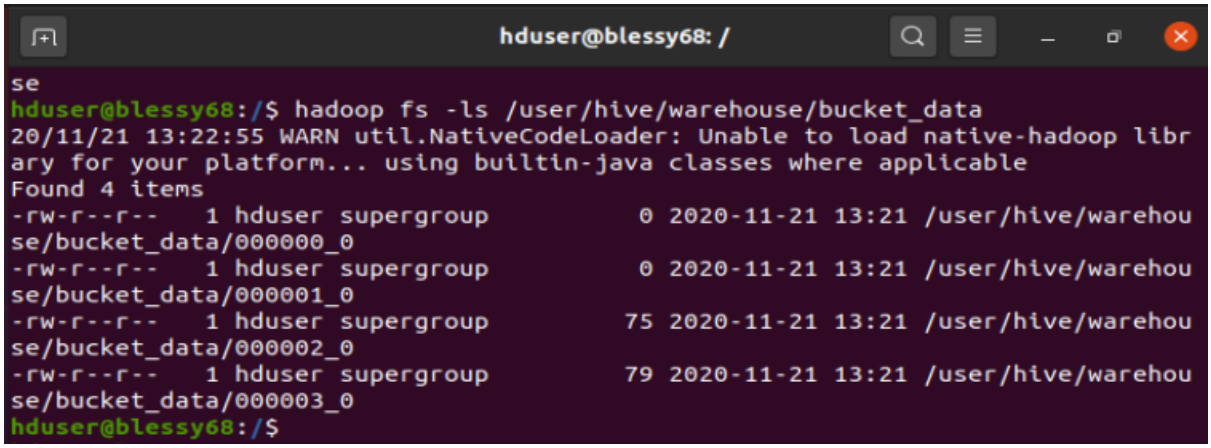
Code:

```
create table data(id int,name string,marks int,country string)
row format delimited fields terminated by ',';

Load data local inpath '/home/blessy/Downloads/data.csv' into table
data;

create table bucket_data(id int, name string,marks int,country
string)
clustered by(country) into 4 buckets
row format delimited fields terminated by ',';


insert overwrite table bucket_data
select id,name,marks,country from data;
```



	A	B	C	D
1	11	blessy	90	india
2	12	heena	80	canada
3	13	neena	70	germany
4	14	kareena	60	canada
5	15	xeena	50	japan
6	16	reena	40	germany
7	17	amul	89	japan
8	18	kotrika	90	india
9				


hduser@blessy68: /
>
>
> create table data(id int,name string,marks int,country string)
> row format delimited fields terminated by ',';
OK
Time taken: 7.345 seconds
hive> Load data local inpath '/home/blessy/Downloads/data.csv' into table data;
Loading data to table default.data
OK
Time taken: 3.47 seconds
hduser@blessy68: /
>
> create table bucket_data(id int, name string,marks int,country string)
> clustered by(country) into 4 buckets
> row format delimited fields terminated by ',';
OK
Time taken: 0.289 seconds
hive> insert overwrite table bucket_data
> select id,name,marks,country from data;
Query ID = hduser_20201121132100_130949ef-e1b0-4096-9a92-035c81e850c2
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks determined at compile time: 4
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>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-11-21 13:21:16,360 Stage-1 map = 0%, reduce = 0%
2020-11-21 13:21:18,446 Stage-1 map = 100%, reduce = 0%
2020-11-21 13:21:19,509 Stage-1 map = 100%, reduce = 100%
2020-11-21 13:21:20,548 Stage-1 map = 100%, reduce = 50%
2020-11-21 13:21:21,586 Stage-1 map = 100%, reduce = 75%
2020-11-21 13:21:22,655 Stage-1 map = 100%, reduce = 100%
hduser@blessy68: /
2020-11-21 13:21:22,655 Stage-1 map = 100%, reduce = 100%
Ended Job = job_local1577387748_0001
Loading data to table default.bucket_data
Launching Job 2 out of 2
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>
In order to limit the maximum number of reducers:
set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2020-11-21 13:21:27,369 Stage-3 map = 100%, reduce = 100%
Ended Job = job_local1553956696_0002
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 770 HDFS Write: 1539 SUCCESS
Stage-Stage-3: HDFS Read: 308 HDFS Write: 1096 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Time taken: 27.789 seconds
hive> hduser@blessy68:/$



4 buckets are created under bucket_data table

Spark Scala Installation Steps:

Before downloading and setting up Spark, you need to install necessary dependencies. This step includes installing the following packages:

- JDK
- Scala
- Git

Open a terminal window and run the following command to install all three packages at once:

```
sudo apt install default-jdk scala git -y
```

verify the installed dependencies by running these commands:

```
java -version; javac -version; scala -version; git –version
```

We **need to download the version of Spark we want** form their website or using the command

```
wget https://downloads.apache.org/spark/spark-3.0.1/spark-3.0.1-bin-hadoop2.7.tgz
```

extract the downloaded folder using

```
tar xvf spark-*
```

Let the process complete. The output shows the files that are being unpacked from the archive.Finally, move the unpacked directory spark-3.0.1-bin-hadoop2.7 to the opt/spark directory.Use the mv command to do so:

```
sudo mv spark-3.0.1-bin-hadoop2.7 /opt/spark
```

The terminal returns no response if it successfully moves the directory.

Before starting a master server, you need to configure environment variables. There are a few Spark home paths you need to add to the user profile.

Use the echo command to add these three lines to .profile:

```
echo "export SPARK_HOME=/opt/spark" >> ~/.profile

echo "export
PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin" >>
~/.profile

echo "export PYSPARK_PYTHON=/usr/bin/python3" >>
~/.profile
```

When you finish adding the paths, load the .profile file in the command line by typing:

```
source ~/.profile
```

set the same on .bashrc file

In the terminal, type:

```
start-master.sh
```

To view the Spark Web user interface, open a web browser and enter the localhost IP address on port 8080.

http://127.0.0.1:8080/

to start slave:

```
start-slave.sh spark://blessy:7077
```

to start the scala shell

use: **spark**

and :**q[Enter]** to quit

SPARK WORDCOUNT

```
>>> words = sc.textFile("/spark/input/word.txt").flatMap(lambda line: line.spli
t(" "))
>>>
>>> wordCounts = words.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a +b)
>>> wordCounts.saveAsTextFile("/spark/output/")
[Stage 0:>                                                          (0 + 0) / 1
[Stage 0:>                                                          (0 + 1) / 1
[Stage 1:>                                                          (0 + 1) / 1

>>>
```

```
hduser@blessy68:/$ hadoop fs -cat /spark/output/part-00000
20/11/21 17:18:51 WARN util.NativeCodeLoader: Unable to load native-hadoop libr
ary for your platform... using builtin-java classes where applicable
('Hellow', 1)
('Blessy', 1)
('how', 2)
('are', 2)
('you', 2)
('hellow', 2)
('blessy', 5)
('kotrika', 2)
('hi', 2)
('child', 1)
('doll', 3)
('cat', 2)
('calm', 1)
hduser@blessy68:/$
```