NAME: OMAR USMAN UPPAL

CMS ID : 203515

ASSIGNMENT # 3

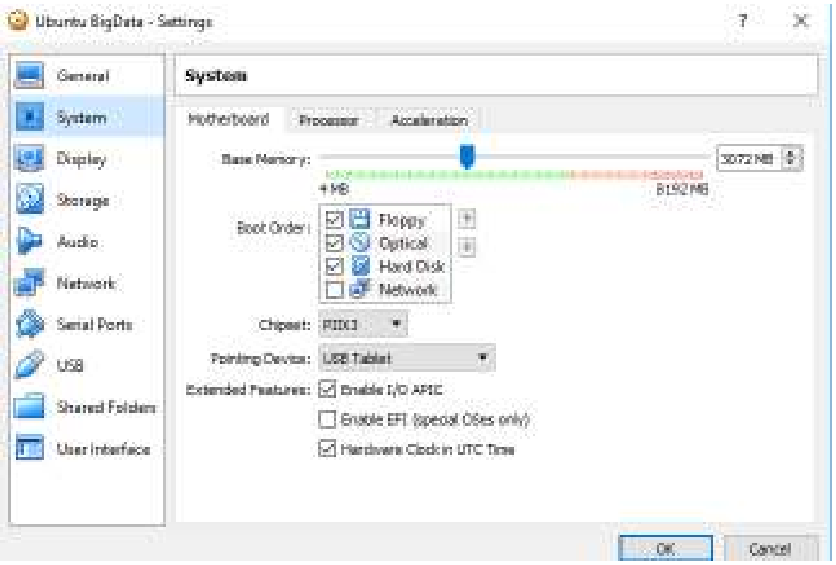
BIG DATA : HADOOP – HIVE – HBASE - SPARK

1. QUESTION 1 – HADOOP

HADOOP INTALLATION:

Following images show different steps involved in HADOOP Installation

Image 1: Hadoop 2.7.5 was downloaded and installed on a UBUNTU Virtual Machine with a RAM of 3 GB and 40 GB Hard Disk space.



Ubuntu BigData - Settings

System

Motherboard

Base Memory: 3072 MB (4 MB – 8192 MB)

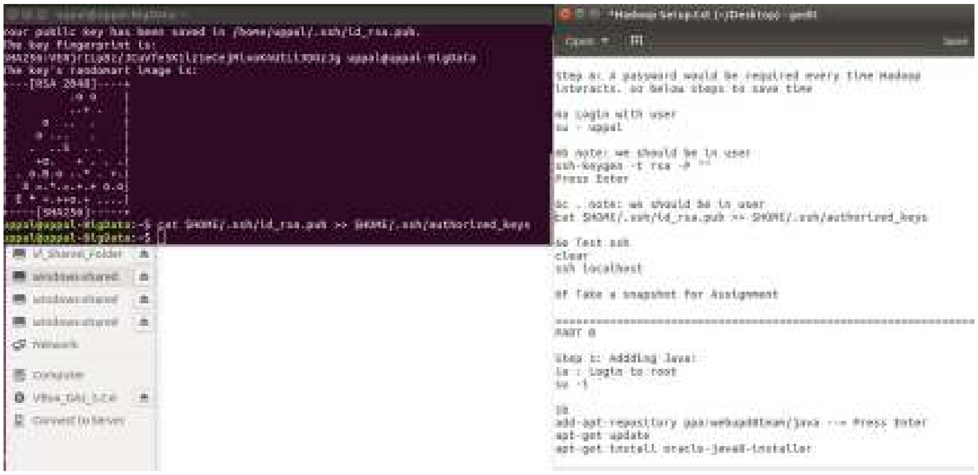
Boot Order:
☑ Floppy
☑ Optical
☑ Hard Disk
□ Network

Chipset: PIIX3

Pointing Device: USB Tablet

Extended Features:
☑ Enable I/O APIC
□ Enable EFI (special OSes only)
☑ Hardware Clock in UTC Time

Image 2



```
our public key has been saved in /home/uppal/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:[illegible] uppal@uppal-BigData
The key's randomart image is:
+---[RSA 2048]----+
[illegible]
+----[SHA256]-----+
uppal@uppal-BigData:~$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
uppal@uppal-BigData:~$
```

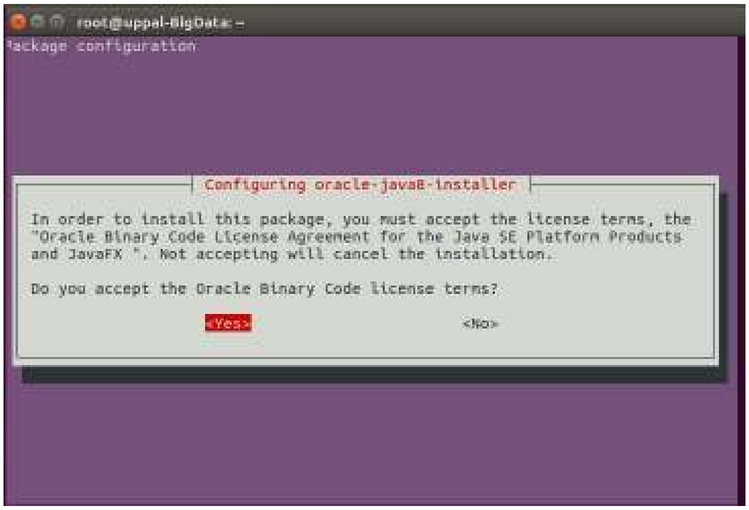
*Hadoop Setup.txt (~/Desktop) - gedit

```
Step 6: A password would be required every time Hadoop
interacts. so below steps to save time

6a Login with user
su - uppal

6b note: we should be in user
ssh-keygen -t rsa -P ""
Press Enter

6c . note: we should be in user
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

6d Test ssh
clear
ssh localhost

6f Take a snapshot for Assignment

==================================================================
PART B

Step 1: Addding Java:
1a : Login to root
su -i

1b
add-apt-repository ppa:webupd8team/java --> Press Enter
apt-get update
apt-get install oracle-java8-installer
```

Image 3

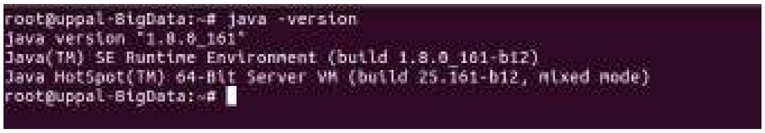
Image 4

Image 5



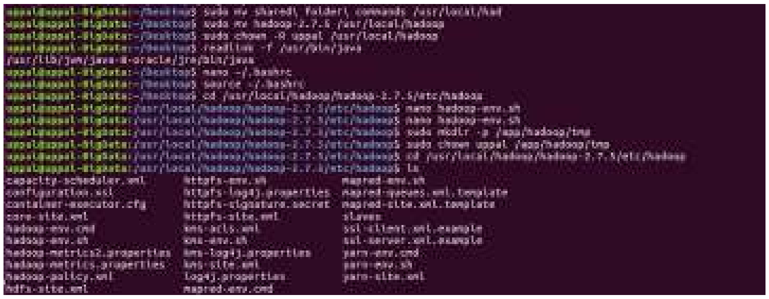
```
Package configuration

             ┤ Configuring oracle-java8-installer ├
 In order to install this package, you must accept the license terms, the
 "Oracle Binary Code License Agreement for the Java SE Platform Products
 and JavaFX ". Not accepting will cancel the installation.

 Do you accept the Oracle Binary Code license terms?

                   <Yes>                          <No>
```

Image 6



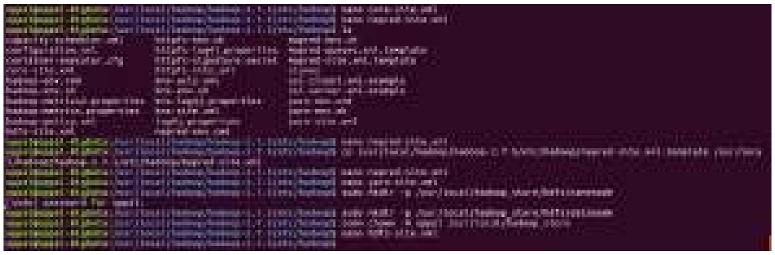
```
root@uppal-BigData:~# java -version
java version "1.8.0_161"
Java(TM) SE Runtime Environment (build 1.8.0_161-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.161-b12, mixed mode)
root@uppal-BigData:~# 
```

Image 7



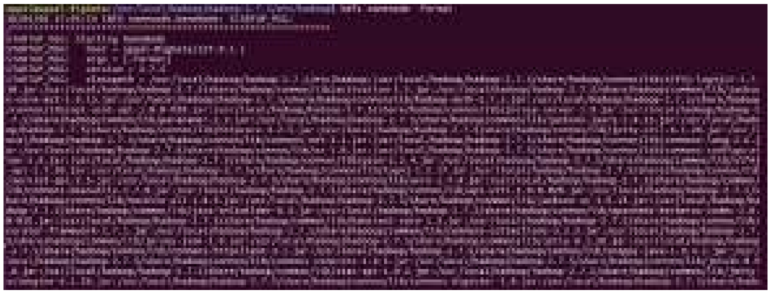
```
uppal@uppal-BigData:~/Desktop$ sudo mv shared\ folder\ commands /usr/local/had
uppal@uppal-BigData:~/Desktop$ sudo mv hadoop-2.7.5 /usr/local/hadoop
uppal@uppal-BigData:~/Desktop$ sudo chown -R uppal /usr/local/hadoop
uppal@uppal-BigData:~/Desktop$ readlink -f /usr/bin/java
/usr/lib/jvm/java-8-oracle/jre/bin/java
uppal@uppal-BigData:~/Desktop$ nano ~/.bashrc
uppal@uppal-BigData:~/Desktop$ source ~/.bashrc
uppal@uppal-BigData:~/Desktop$ cd /usr/local/hadoop/hadoop-2.7.5/etc/hadoop
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano hadoop-env.sh
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano hadoop-env.sh
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ sudo mkdir -p /app/hadoop/tmp
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ sudo chown uppal /app/hadoop/tmp
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ cd /usr/local/hadoop/hadoop-2.7.5/etc/hadoop
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ ls
capacity-scheduler.xml      httpfs-env.sh               mapred-env.sh
configuration.xsl           httpfs-log4j.properties     mapred-queues.xml.template
container-executor.cfg      httpfs-signature.secret     mapred-site.xml.template
core-site.xml               httpfs-site.xml             slaves
hadoop-env.cmd              kms-acls.xml                ssl-client.xml.example
hadoop-env.sh               kms-env.sh                  ssl-server.xml.example
hadoop-metrics2.properties  kms-log4j.properties        yarn-env.cmd
hadoop-metrics.properties   kms-site.xml                yarn-env.sh
hadoop-policy.xml           log4j.properties            yarn-site.xml
hdfs-site.xml               mapred-env.cmd
```

Image 8



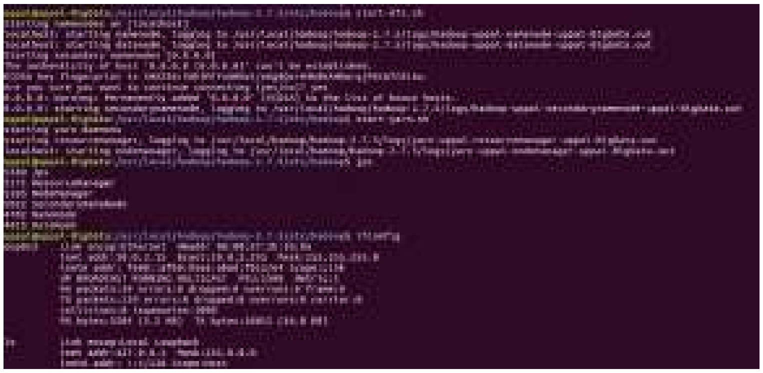
```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano core-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano mapred-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ ls
capacity-scheduler.xml      httpfs-env.sh               mapred-env.sh
configuration.xsl           httpfs-log4j.properties     mapred-queues.xml.template
container-executor.cfg      httpfs-signature.secret     mapred-site.xml.template
core-site.xml               httpfs-site.xml             slaves
hadoop-env.cmd              kms-acls.xml                ssl-client.xml.example
hadoop-env.sh               kms-env.sh                  ssl-server.xml.example
hadoop-metrics2.properties  kms-log4j.properties        yarn-env.cmd
hadoop-metrics.properties   kms-site.xml                yarn-env.sh
hadoop-policy.xml           log4j.properties            yarn-site.xml
hdfs-site.xml               mapred-env.cmd
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano mapred-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ cp /usr/local/hadoop/hadoop-2.7.5/etc/hadoop/mapred-site.xml.template /usr/local
/hadoop/hadoop-2.7.5/etc/hadoop/mapred-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano mapred-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano yarn-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ sudo mkdir -p /usr/local/hadoop_store/hdfs/namenode
[sudo] password for uppal:
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ sudo mkdir -p /usr/local/hadoop_store/hdfs/datanode
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ sudo chown -R uppal /usr/local/hadoop_store
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ nano hdfs-site.xml
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$
```

Image 9



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ hdfs namenode -format
[illegible] INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = uppal-BigData/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 2.7.5
STARTUP_MSG:   classpath = /usr/local/hadoop/hadoop-2.7.5/etc/hadoop:/usr/local/hadoop/hadoop-2.7.5/share/hadoop/common/lib/[illegible]
[illegible]
```

Image 10

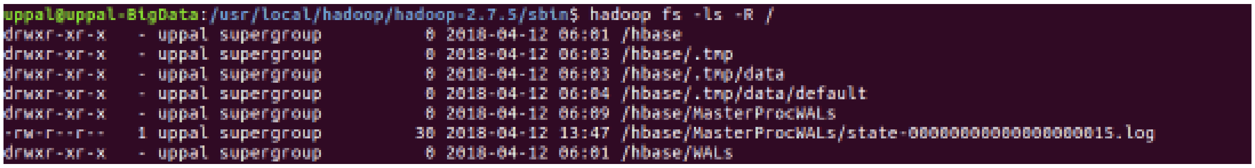


```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop/hadoop-2.7.5/logs/hadoop-uppal-namenode-uppal-BigData.out
localhost: starting datanode, logging to /usr/local/hadoop/hadoop-2.7.5/logs/hadoop-uppal-datanode-uppal-BigData.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is SHA256:[illegible]
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop/hadoop-2.7.5/logs/hadoop-uppal-secondarynamenode-uppal-BigData.out
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ start-yarn.sh
starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop/hadoop-2.7.5/logs/yarn-uppal-resourcemanager-uppal-BigData.out
localhost: starting nodemanager, logging to /usr/local/hadoop/hadoop-2.7.5/logs/yarn-uppal-nodemanager-uppal-BigData.out
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ jps
[illegible] Jps
[illegible] ResourceManager
[illegible] NodeManager
[illegible] SecondaryNameNode
[illegible] NameNode
[illegible] DataNode
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/etc/hadoop$ ifconfig
[illegible]
```

HADOOP COMMANDS:

Command 1: $ hadoop fs –ls –R /

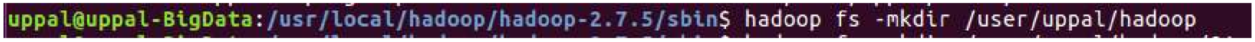
This command shows the complete directory system of the Hadoop file system including the files inside the folder.



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:01 /hbase
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:03 /hbase/.tmp
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:03 /hbase/.tmp/data
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:04 /hbase/.tmp/data/default
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:09 /hbase/MasterProcWALs
-rw-r--r--   1 uppal supergroup         30 2018-04-12 13:47 /hbase/MasterProcWALs/state-00000000000000000015.log
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:01 /hbase/WALs
```

Command 2: $ hadoop fs -mkdir /user/uppal/Hadoop

"hadoop fs –mkdir" is used to create a folder in the Hadoop file system. The datasets downloaded are inside Ubuntu file system and they need to be accessed by Hadoop file system so Hadoop commands could be executed on those files. Therefore, a new folder is created inside another Hadoop accessed directory.



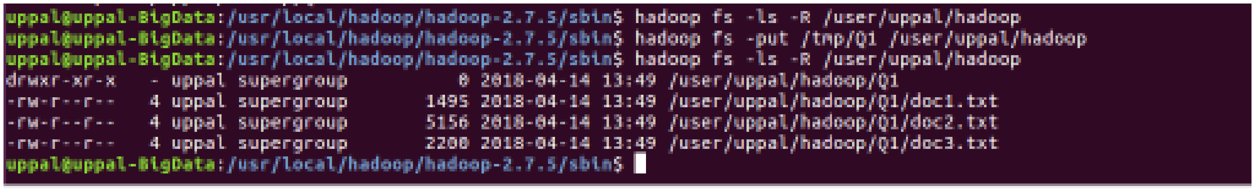
```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -mkdir /user/uppal/hadoop
```

Command 3: $ hadoop fs –put /tmp/Q1 /user/uppal/hadoop

3 datasets were downaloaded from website: https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/ and renamed as "doc1", "doc2" and "doc3". Using "put" command datasets were moved to Hadoop folder "/user/uppal/hadoop/Q1".

Command 4: $ hadoop fs –ls –R /user/uppal/Hadoop

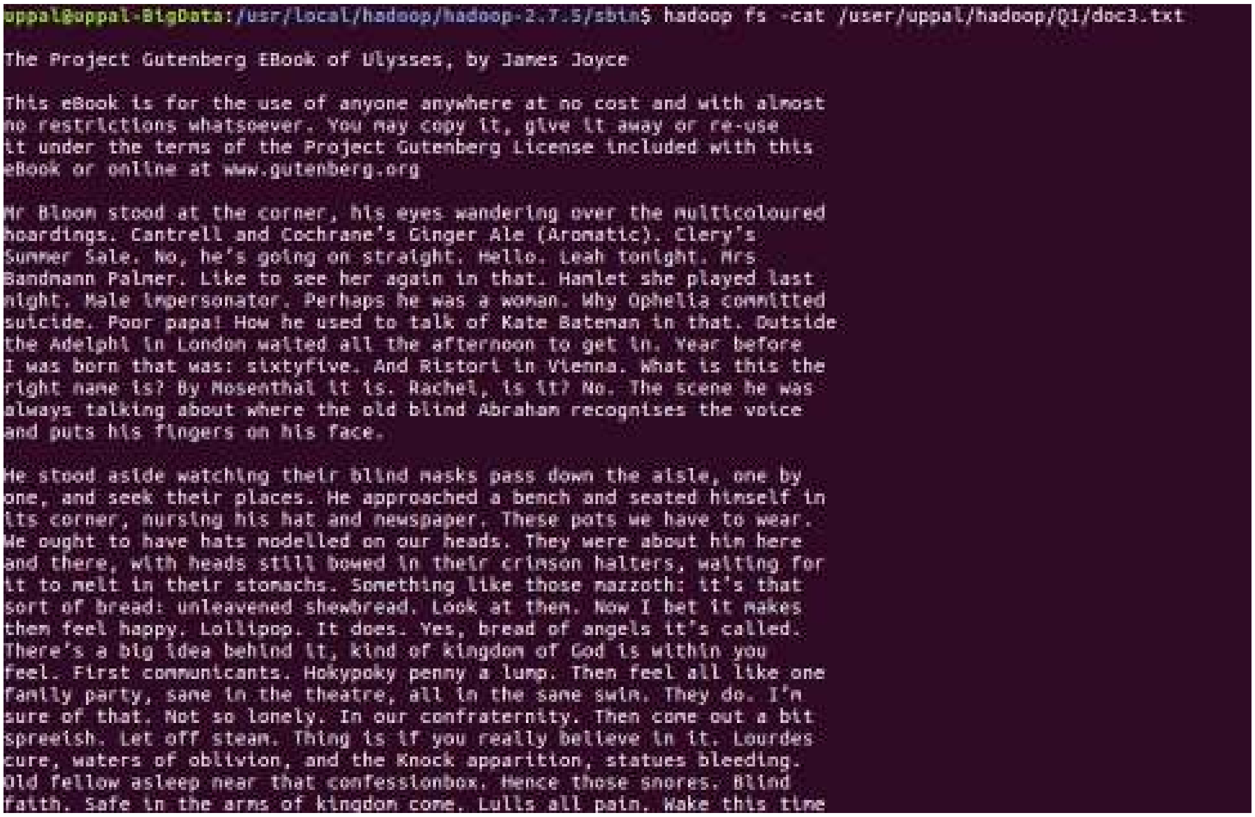
Datasets were stored in **../Q1** folder and the folder was moved to Hadoop folder **../Hadoop**. It can be seen in the below snapshots.



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /user/uppal/hadoop
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -put /tmp/Q1 /user/uppal/hadoop
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /user/uppal/hadoop
drwxr-xr-x   - uppal supergroup          0 2018-04-14 13:49 /user/uppal/hadoop/Q1
-rw-r--r--   4 uppal supergroup       1495 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc1.txt
-rw-r--r--   4 uppal supergroup       5156 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc2.txt
-rw-r--r--   4 uppal supergroup       2200 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc3.txt
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ 
```

Command 5: $ hadoop fs -cat /user/uppal/hadoop/doc3.txt

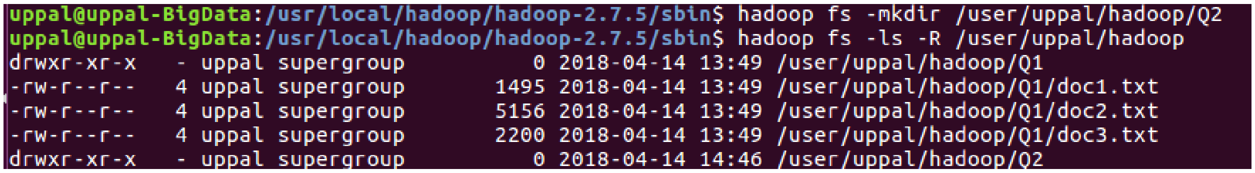
"CAT" command was used to print the content of the dataset on the console.



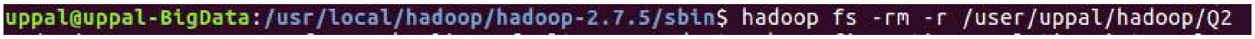
```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -cat /user/uppal/hadoop/Q1/doc3.txt

The Project Gutenberg EBook of Ulysses, by James Joyce

This eBook is for the use of anyone anywhere at no cost and with almost
no restrictions whatsoever. You may copy it, give it away or re-use
it under the terms of the Project Gutenberg License included with this
eBook or online at www.gutenberg.org

Mr Bloom stood at the corner, his eyes wandering over the multicoloured
hoardings. Cantrell and Cochrane's Ginger Ale (Aromatic). Clery's
Summer Sale. No, he's going on straight. Hello. Leah tonight. Mrs
Bandmann Palmer. Like to see her again in that. Hamlet she played last
night. Male impersonator. Perhaps he was a woman. Why Ophelia committed
suicide. Poor papa! How he used to talk of Kate Bateman in that. Outside
the Adelphi in London waited all the afternoon to get in. Year before
I was born that was: sixtyfive. And Ristori in Vienna. What is this the
right name is? By Mosenthal it is. Rachel, is it? No. The scene he was
always talking about where the old blind Abraham recognises the voice
and puts his fingers on his face.

He stood aside watching their blind masks pass down the aisle, one by
one, and seek their places. He approached a bench and seated himself in
its corner, nursing his hat and newspaper. These pots we have to wear.
We ought to have hats modelled on our heads. They were about him here
and there, with heads still bowed in their crimson halters, waiting for
it to melt in their stomachs. Something like those mazzoth: it's that
sort of bread: unleavened shewbread. Look at them. Now I bet it makes
them feel happy. Lollipop. It does. Yes, bread of angels it's called.
There's a big idea behind it, kind of kingdom of God is within you
feel. First communicants. Hokypoky penny a lump. Then feel all like one
family party, same in the theatre, all in the same swim. They do. I'm
sure of that. Not so lonely. In our confraternity. Then come out a bit
spreeish. Let off steam. Thing is if you really believe in it. Lourdes
cure, waters of oblivion, and the Knock apparition, statues bleeding.
Old fellow asleep near that confessionbox. Hence those snores. Blind
faith. Safe in the arms of kingdom come. Lulls all pain. Wake this time
```

Command 6: $ hadoop fs –rm –r /user/uppal/hadoop/Q2

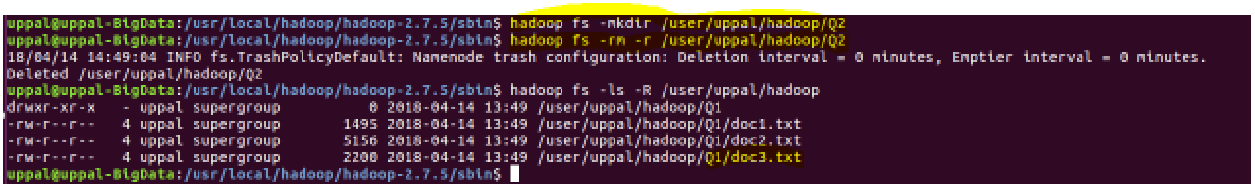
Q2 folder was creared and than using “-rm –r” command the folder as removed from the Hadoop file system as it can be seen in below set of snapshots.



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -mkdir /user/uppal/hadoop/Q2
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /user/uppal/hadoop
drwxr-xr-x   - uppal supergroup          0 2018-04-14 13:49 /user/uppal/hadoop/Q1
-rw-r--r--   4 uppal supergroup       1495 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc1.txt
-rw-r--r--   4 uppal supergroup       5156 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc2.txt
-rw-r--r--   4 uppal supergroup       2200 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc3.txt
drwxr-xr-x   - uppal supergroup          0 2018-04-14 14:46 /user/uppal/hadoop/Q2
```



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -rm -r /user/uppal/hadoop/Q2
```



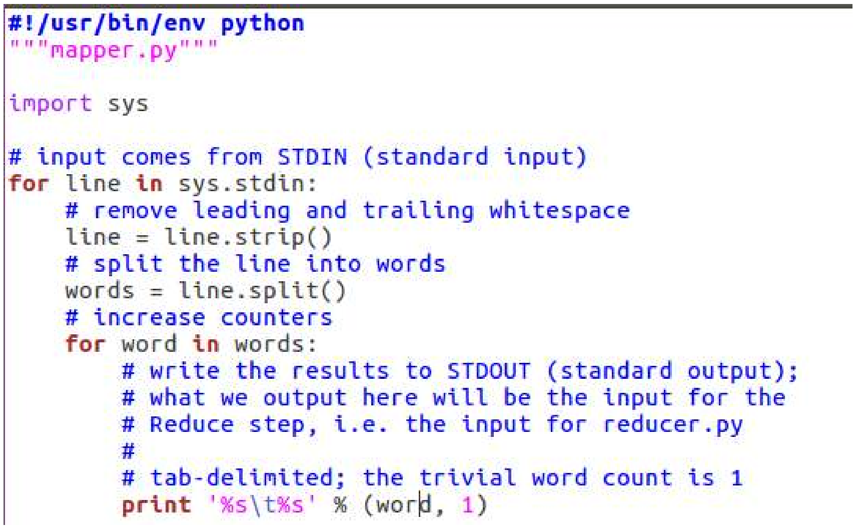
```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -mkdir /user/uppal/hadoop/Q2
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -rm -r /user/uppal/hadoop/Q2
18/04/14 14:49:04 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /user/uppal/hadoop/Q2
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /user/uppal/hadoop
drwxr-xr-x   - uppal supergroup          0 2018-04-14 13:49 /user/uppal/hadoop/Q1
-rw-r--r--   4 uppal supergroup       1495 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc1.txt
-rw-r--r--   4 uppal supergroup       5156 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc2.txt
-rw-r--r--   4 uppal supergroup       2200 2018-04-14 13:49 /user/uppal/hadoop/Q1/doc3.txt
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$
```

MAP-REDUCE function using Hadoop Streaming:

Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. The mapper and reducer codes are written in python as below:

MAPPER CODE:

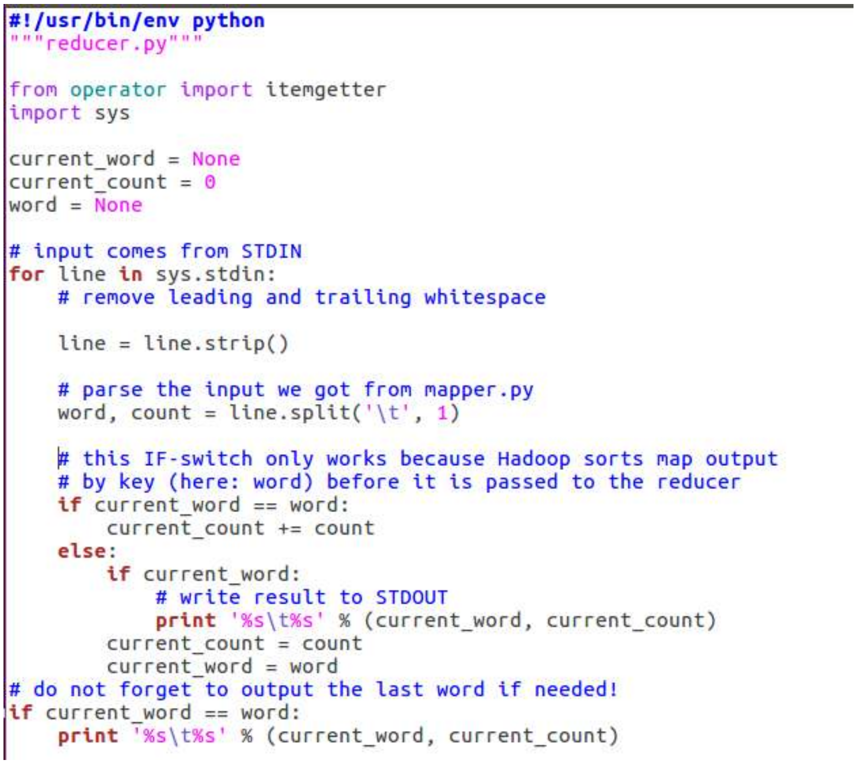
Mapper code will read the data from STDIN and splits into key-value pairs (word, count=1). The output is list of lines of the pairs to STDOUT.



```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

REDUCER CODE:

Reducer code will read the results of mapper function from STDIN and sums the occurrence of each word to final count. The output is list of lines of the pairs (words, sum) to STDOUT.



```
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace

    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word
# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Command 7: $ chmod +x /usr/local/hadoop/mapper.py

The command is run for getting the execution permission of the mapper.py file else it will run into problems. Same permission is give reducer.py.

STRUCTURE OF HADOOP STREAMING COMMAND:

mapred streaming --> streaming command *.jar

-input myInputDirs --> directory containing input files

-output myOutputDir --> output directory will be created. It should not exist else it will give error

-mapper PythonMap.py --> mapper script

-reducer PythonReduce.py --> reducer script

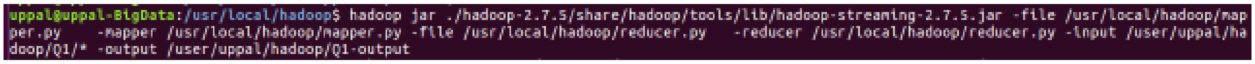
Command 8:

**hadoop jar ./hadoop-2.7.5/share/hadoop/tools/lib/hadoop-streaming-2.7.5.jar **

**-file /usr/local/hadoop/mapper.py -mapper /usr/local/hadoop/mapper.py **

**-file /usr/local/hadoop/reducer.py -reducer /usr/local/hadoop/reducer.py **

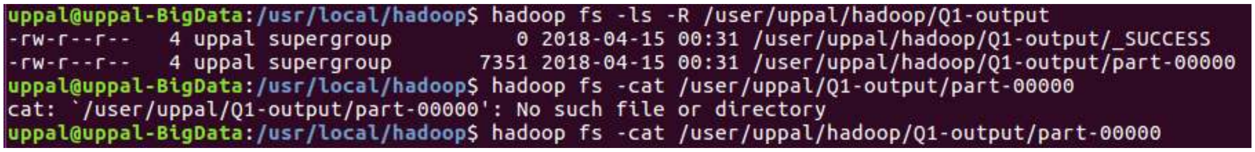
-input /user/uppal/hadoop/Q1/* -output /user/uppal/hadoop/Q1-output



```
uppal@uppal-BigData:/usr/local/hadoop$ hadoop jar ./hadoop-2.7.5/share/hadoop/tools/lib/hadoop-streaming-2.7.5.jar -file /usr/local/hadoop/map
per.py    -mapper /usr/local/hadoop/mapper.py -file /usr/local/hadoop/reducer.py    -reducer /usr/local/hadoop/reducer.py -input /user/uppal/ha
doop/Q1/* -output /user/uppal/hadoop/Q1-output
```

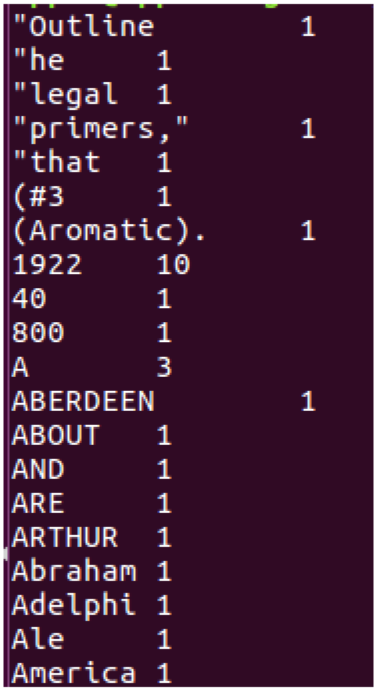
OUTPUT:

Output of the command was stored in a newly formed output directory as give in the image. The name of the output file of reducer is “part-00000”



```
uppal@uppal-BigData:/usr/local/hadoop$ hadoop fs -ls -R /user/uppal/hadoop/Q1-output
-rw-r--r--   4 uppal supergroup          0 2018-04-15 00:31 /user/uppal/hadoop/Q1-output/_SUCCESS
-rw-r--r--   4 uppal supergroup       7351 2018-04-15 00:31 /user/uppal/hadoop/Q1-output/part-00000
uppal@uppal-BigData:/usr/local/hadoop$ hadoop fs -cat /user/uppal/Q1-output/part-00000
cat: `/user/uppal/Q1-output/part-00000': No such file or directory
uppal@uppal-BigData:/usr/local/hadoop$ hadoop fs -cat /user/uppal/hadoop/Q1-output/part-00000
```

Executing the “cat” command displays a long list of key-value pairs. A small part is shown in the image below:



```
"Outline        1
"he     1
"legal  1
"primers,"      1
"that   1
(#3     1
(Aromatic).     1
1922    10
40      1
800     1
A       3
ABERDEEN        1
ABOUT   1
AND     1
ARE     1
ARTHUR  1
Abraham 1
Adelphi 1
Ale     1
America 1
```

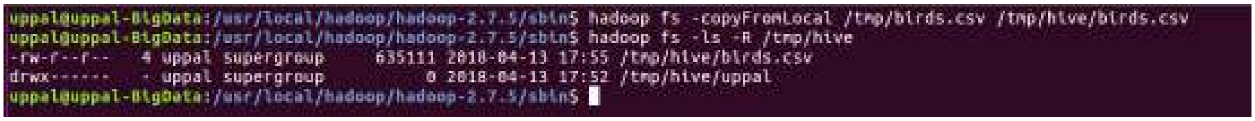
2. QUESTION 2 - HIVE

Download Dataset

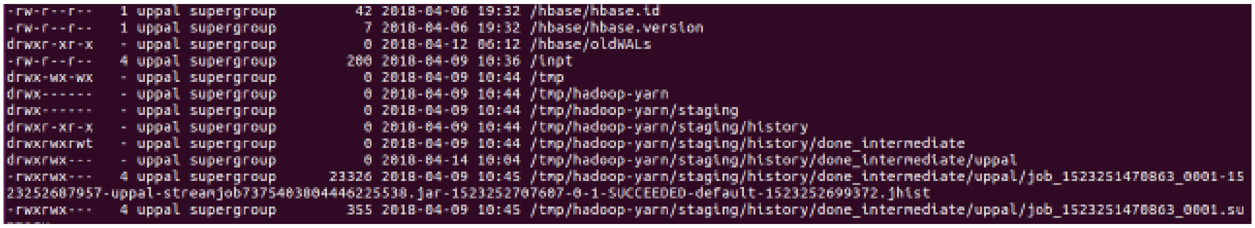
Bird.csv dataset is downloaded from the link:

http://stat-computing.org/dataexpo/2011/resources/data/birds.csv

Copy the file to Hadoop folder



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -copyFromLocal /tmp/birds.csv /tmp/hive/birds.csv
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /tmp/hive
-rw-r--r--   4 uppal supergroup     635111 2018-04-13 17:55 /tmp/hive/birds.csv
drwx------   - uppal supergroup          0 2018-04-13 17:52 /tmp/hive/uppal
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ 
```



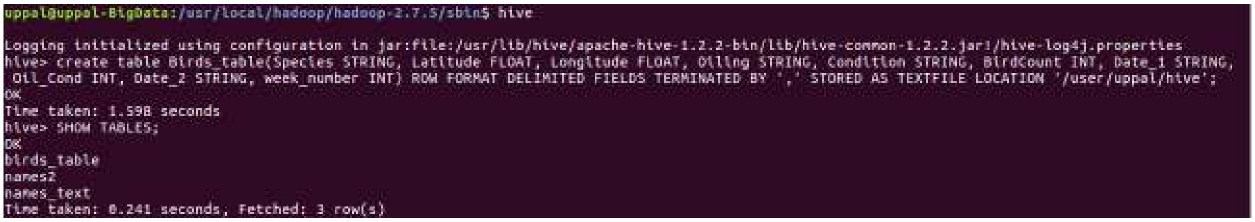
```
-rw-r--r--   1 uppal supergroup         42 2018-04-06 19:32 /hbase/hbase.id
-rw-r--r--   1 uppal supergroup          7 2018-04-06 19:32 /hbase/hbase.version
drwxr-xr-x   - uppal supergroup          0 2018-04-12 06:12 /hbase/oldWALs
-rw-r--r--   4 uppal supergroup        200 2018-04-09 10:36 /inpt
drwx-wx-wx   - uppal supergroup          0 2018-04-09 10:44 /tmp
drwx------   - uppal supergroup          0 2018-04-09 10:44 /tmp/hadoop-yarn
drwx------   - uppal supergroup          0 2018-04-09 10:44 /tmp/hadoop-yarn/staging
drwxr-xr-x   - uppal supergroup          0 2018-04-09 10:44 /tmp/hadoop-yarn/staging/history
drwxrwxrwt   - uppal supergroup          0 2018-04-09 10:44 /tmp/hadoop-yarn/staging/history/done_intermediate
drwxrwx---   - uppal supergroup          0 2018-04-14 10:04 /tmp/hadoop-yarn/staging/history/done_intermediate/uppal
-rwxrwx---   4 uppal supergroup      23326 2018-04-09 10:45 /tmp/hadoop-yarn/staging/history/done_intermediate/uppal/job_1523251470863_0001-15
23252607957-uppal-streamjob7375403804446225538.jar-1523252707607-0-1-SUCCEEDED-default-1523252699372.jhist
-rwxrwx---   4 uppal supergroup        355 2018-04-09 10:45 /tmp/hadoop-yarn/staging/history/done_intermediate/uppal/job_1523251470863_0001.su
```

Operation 1: Starts hive shell → $ hive

Operation 2: Create Table → $ create table Birds_table(Species STRING, Latitude FLOAT, Longitude FLOAT, Oiling STRING, Condition STRING, BirdCount INT, Date_1 STRING, Oil_Cond INT, Date_2 STRING, week_number INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/uppal/hive';

The command will create a table "Birds_table" with column names give in the brakcets with their data type. For Example, "Species" is a column name with "STRING" as data type. Since it is a *.csv file so delimitation criteria is give and then table is stored at a specific location

Operation 3: Check table created? → SHOW TABLES;

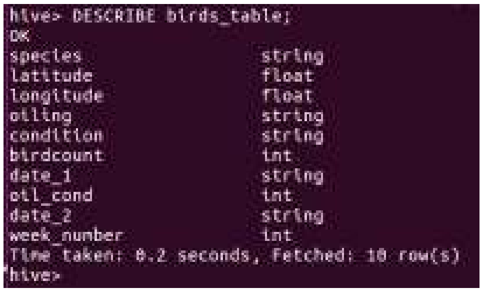


```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hive

Logging initialized using configuration in jar:file:/usr/lib/hive/apache-hive-1.2.2-bin/lib/hive-common-1.2.2.jar!/hive-log4j.properties
hive> create table Birds_table(Species STRING, Latitude FLOAT, Longitude FLOAT, Oiling STRING, Condition STRING, BirdCount INT, Date_1 STRING,
 Oil_Cond INT, Date_2 STRING, week_number INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE LOCATION '/user/uppal/hive';
OK
Time taken: 1.598 seconds
hive> SHOW TABLES;
OK
birds_table
names2
names_text
Time taken: 0.241 seconds, Fetched: 3 row(s)
```

The command shows the tables available in the HIVE.

Operation 4: Check table information: → DESCRIBE "Birds_table"

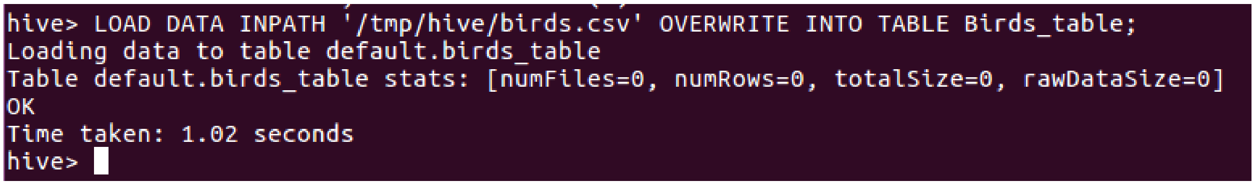
We can see that this command shows the column names and their data types of the table. It also shows the count that there are 10 columns and the time it took to provide the information.



```
hive> DESCRIBE birds_table;
OK
species                 string
latitude                float
longitude               float
oiling                  string
condition               string
birdcount               int
date_1                  string
oil_cond                int
date_2                  string
week_number             int
Time taken: 0.2 seconds, Fetched: 10 row(s)
hive>
```

Operation 5: LOAD Data: LOAD DATA INPATH '/tmp/hive/birds.csv' OVERWRITE INTO TABLE Birds_table;

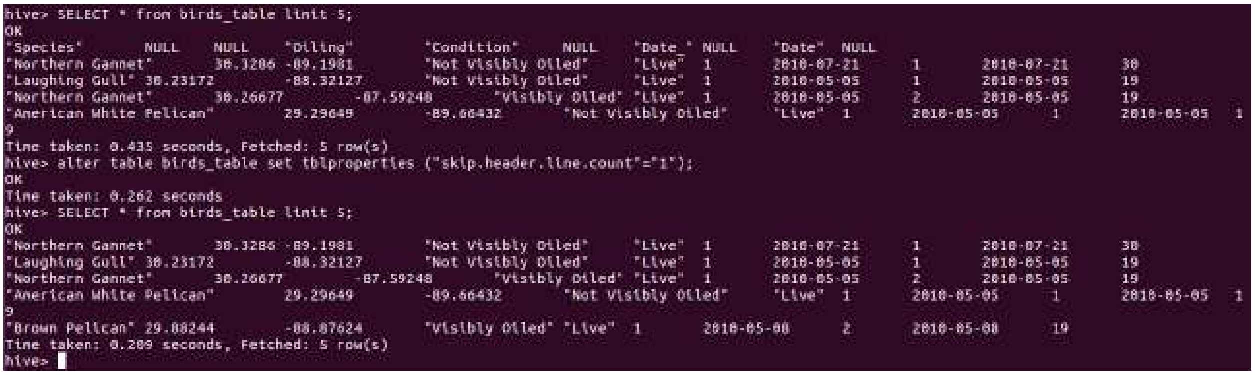
The data from the dataset "birds.csv" will be uploaded in the table. Once done we can execute query commands on the table.



```
hive> LOAD DATA INPATH '/tmp/hive/birds.csv' OVERWRITE INTO TABLE Birds_table;
Loading data to table default.birds_table
Table default.birds_table stats: [numFiles=0, numRows=0, totalSize=0, rawDataSize=0]
OK
Time taken: 1.02 seconds
hive> 
```

Query 1: SELECT * from birds_table limit 5;

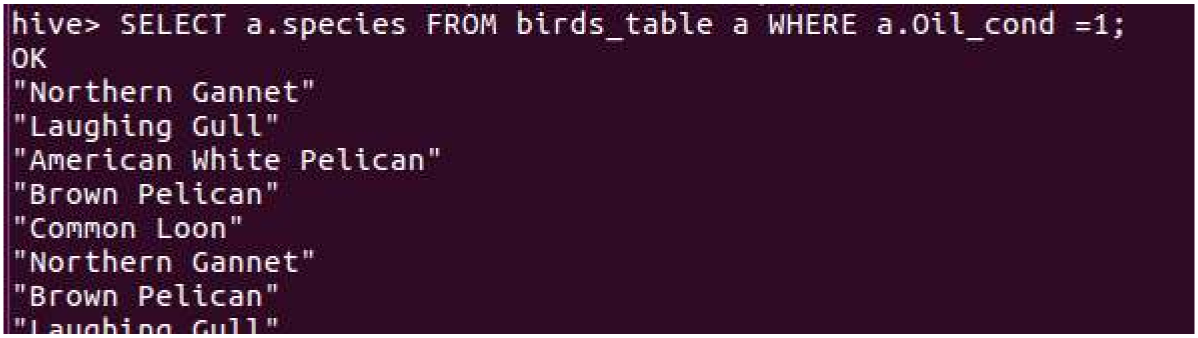
The query returns 5 arbitrary birds data.



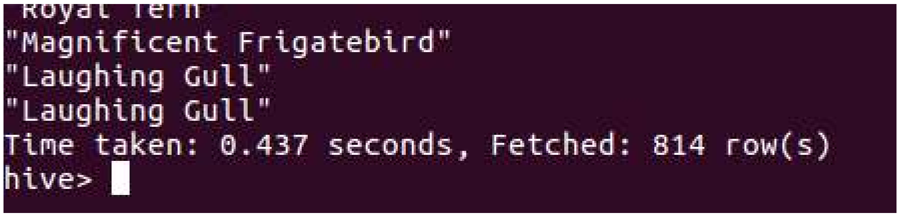
```
hive> SELECT * from birds_table limit 5;
OK
"Species"       NULL    NULL    "Oiling"        "Condition"     NULL    "Date_" NULL    "Date"  NULL
"Northern Gannet"       30.3286 -89.1981        "Not Visibly Oiled"     "Live"  1       2010-07-21      1       2010-07-21      30
"Laughing Gull" 30.23172        -88.32127       "Not Visibly Oiled"     "Live"  1       2010-05-05      1       2010-05-05      19
"Northern Gannet"       30.26677        -87.59248       "Visibly Oiled" "Live"  1       2010-05-05      2       2010-05-05      19
"American White Pelican"        29.29649        -89.66432       "Not Visibly Oiled"     "Live"  1       2010-05-05      1       2010-05-05      1
9
Time taken: 0.435 seconds, Fetched: 5 row(s)
hive> alter table birds_table set tblproperties ("skip.header.line.count"="1");
OK
Time taken: 0.262 seconds
hive> SELECT * from birds_table limit 5;
OK
"Northern Gannet"       30.3286 -89.1981        "Not Visibly Oiled"     "Live"  1       2010-07-21      1       2010-07-21      30
"Laughing Gull" 30.23172        -88.32127       "Not Visibly Oiled"     "Live"  1       2010-05-05      1       2010-05-05      19
"Northern Gannet"       30.26677        -87.59248       "Visibly Oiled" "Live"  1       2010-05-05      2       2010-05-05      19
"American White Pelican"        29.29649        -89.66432       "Not Visibly Oiled"     "Live"  1       2010-05-05      1       2010-05-05      1
9
"Brown Pelican" 29.08244        -88.87624       "Visibly Oiled" "Live"  1       2010-05-08      2       2010-05-08      19
Time taken: 0.209 seconds, Fetched: 5 row(s)
hive> 
```

Query 2: SELECT species FROM birds_table WHERE Oil_cond =1;

The query returns list of Species where oiling condition is 1. There are 814 rows given at the end of the result while the small subset of output format is shown in the figure.



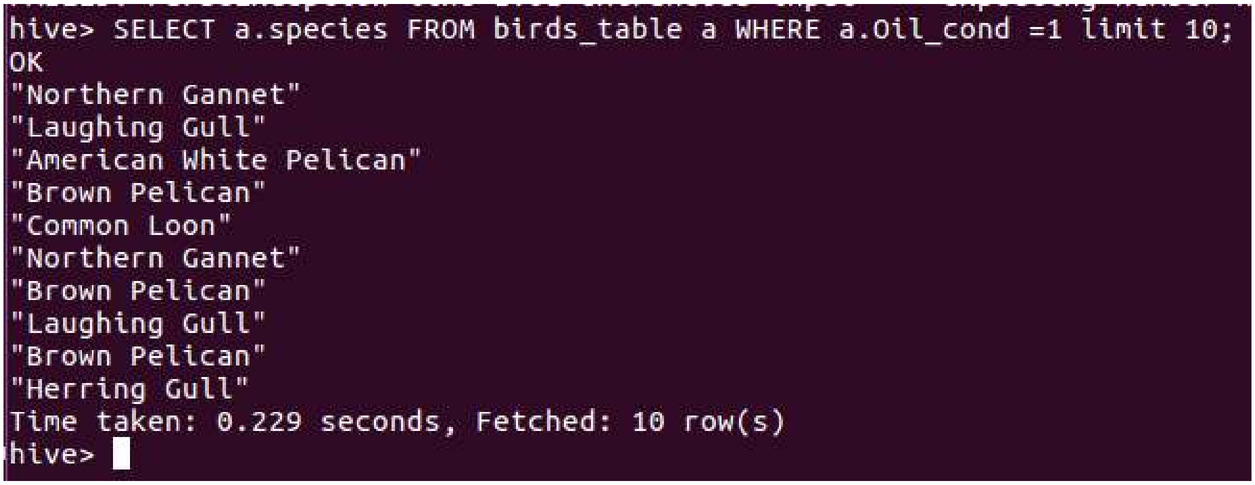
```
hive> SELECT a.species FROM birds_table a WHERE a.Oil_cond =1;
OK
"Northern Gannet"
"Laughing Gull"
"American White Pelican"
"Brown Pelican"
"Common Loon"
"Northern Gannet"
"Brown Pelican"
"Laughing Gull"
```



```
 Royal Tern
"Magnificent Frigatebird"
"Laughing Gull"
"Laughing Gull"
Time taken: 0.437 seconds, Fetched: 814 row(s)
hive> 
```

Query 3: SELECT a.species FROM birds_table a WHERE a.Oil_cond =1 limit 10;

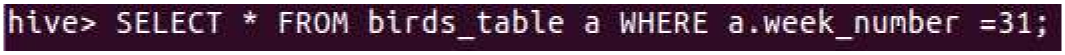
The query returns 10 arbitrary name of species that meet the criteria of Oil_cond=1;



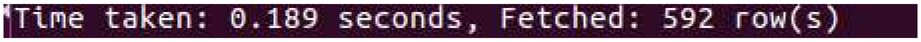
```
hive> SELECT a.species FROM birds_table a WHERE a.Oil_cond =1 limit 10;
OK
"Northern Gannet"
"Laughing Gull"
"American White Pelican"
"Brown Pelican"
"Common Loon"
"Northern Gannet"
"Brown Pelican"
"Laughing Gull"
"Brown Pelican"
"Herring Gull"
Time taken: 0.229 seconds, Fetched: 10 row(s)
hive> 
```

Query 4: SELECT * FROM birds_table a WHERE a.week_number =31;

The query returns all the data of week.number 31. It shows there are 592 rows of such type.



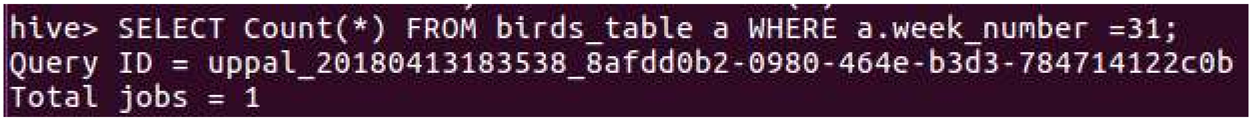
```
hive> SELECT * FROM birds_table a WHERE a.week_number =31;
```



```
Time taken: 0.189 seconds, Fetched: 592 row(s)
```

Query 5: SELECT Count(*) FROM birds_table a WHERE a.week_number =31;

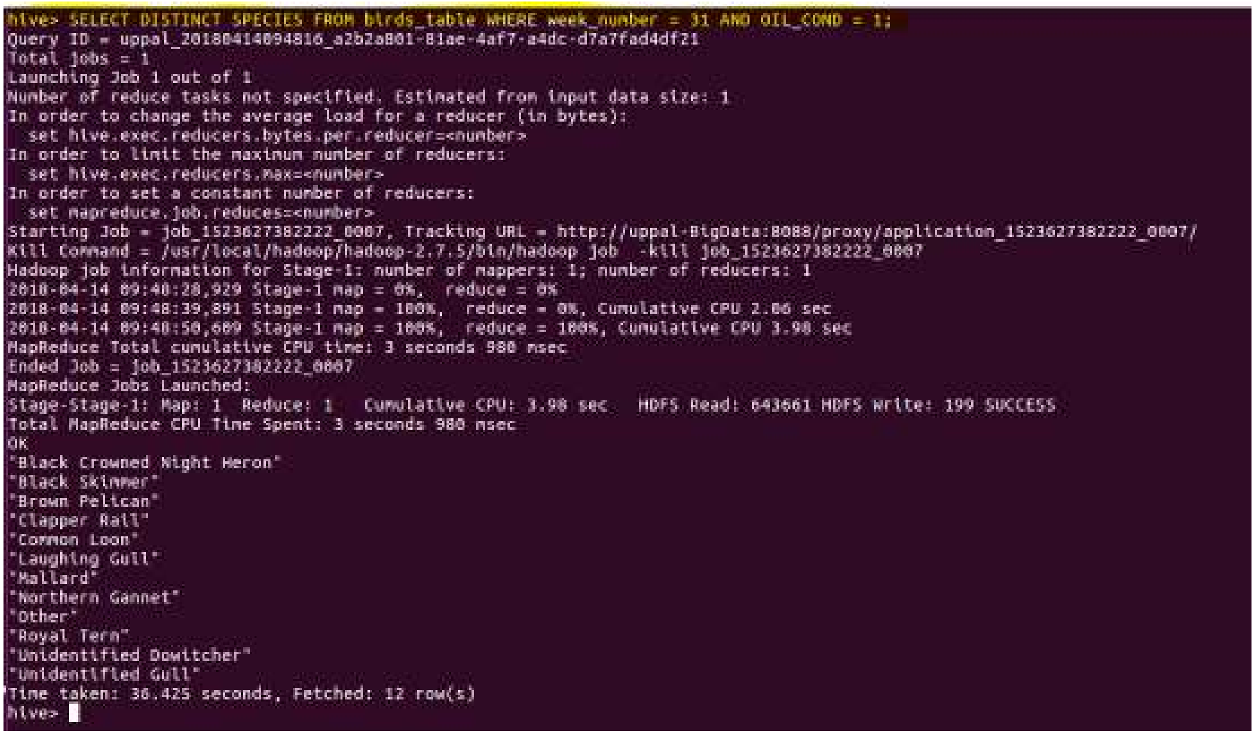
There are 592 samples from Week 31. This can be checked using Count(*).



```
hive> SELECT Count(*) FROM birds_table a WHERE a.week_number =31;
Query ID = uppal_20180413183538_8afdd0b2-0980-464e-b3d3-784714122c0b
Total jobs = 1
```

Query 6: SELECT DISTINCT SPECIES FROM birds_table WHERE week_number = 31 AND OIL_COND = 1;

Using the “DISTINCT” keyword we can retrieve the list of species without duplicate that were affected by oil_cond=1 in week.number 31.



```
hive> SELECT DISTINCT SPECIES FROM birds_table WHERE week_number = 31 AND OIL_COND = 1;
Query ID = uppal_20180414094016_a2b2a801-81ae-4af7-a4dc-d7a7fad4df21
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1523627382222_0007, Tracking URL = http://uppal-BigData:8088/proxy/application_1523627382222_0007/
Kill Command = /usr/local/hadoop/hadoop-2.7.5/bin/hadoop job  -kill job_1523627382222_0007
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2018-04-14 09:40:28,929 Stage-1 map = 0%,  reduce = 0%
2018-04-14 09:40:39,891 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 2.06 sec
2018-04-14 09:40:50,609 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 3.98 sec
MapReduce Total cumulative CPU time: 3 seconds 980 msec
Ended Job = job_1523627382222_0007
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 3.98 sec   HDFS Read: 643661 HDFS Write: 199 SUCCESS
Total MapReduce CPU Time Spent: 3 seconds 980 msec
OK
"Black Crowned Night Heron"
"Black Skimmer"
"Brown Pelican"
"Clapper Rail"
"Common Loon"
"Laughing Gull"
"Mallard"
"Northern Gannet"
"Other"
"Royal Tern"
"Unidentified Dowitcher"
"Unidentified Gull"
Time taken: 36.425 seconds, Fetched: 12 row(s)
hive> 
```

3. QUESTION 3 - HBASE

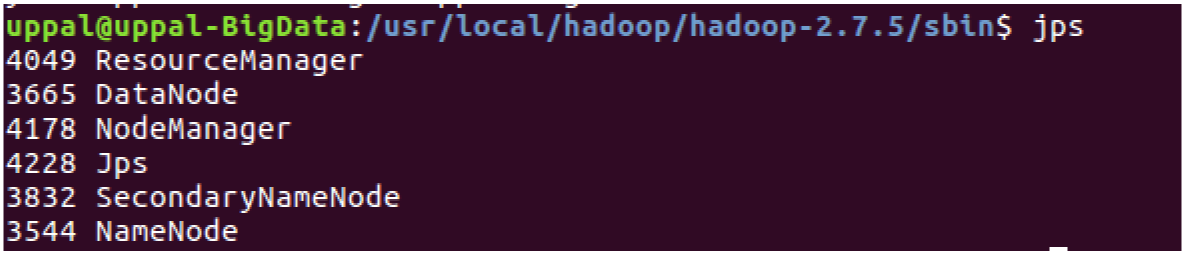
Download Dataset

turtles.csv dataset is downloaded from the link:

http://stat-computing.org/dataexpo/2011/resources/data/turtles.csv

Operation 1: $ jps

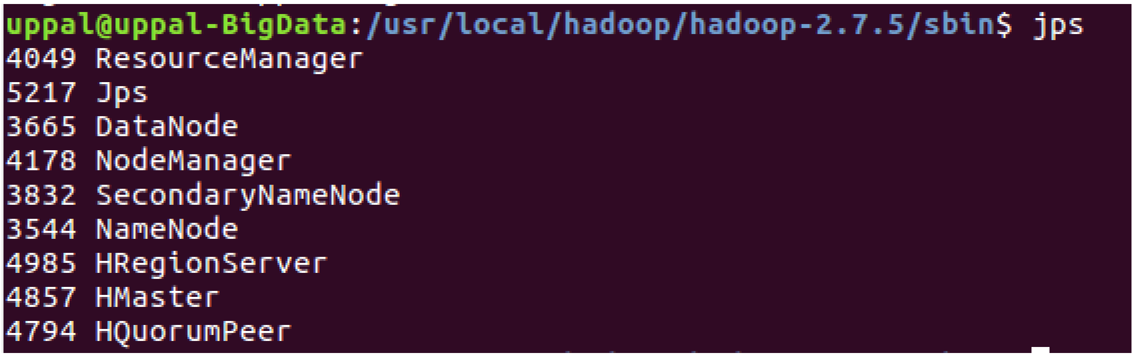
“jps” displays different daemons running in the background. We can see that “hbase” is not active.



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ jps
4049 ResourceManager
3665 DataNode
4178 NodeManager
4228 Jps
3832 SecondaryNameNode
3544 NameNode
```

Operation 2: $ start-hbase.sh

This command will start the hbase and we can see the result using “jps” command.



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ jps
4049 ResourceManager
5217 Jps
3665 DataNode
4178 NodeManager
3832 SecondaryNameNode
3544 NameNode
4985 HRegionServer
4857 HMaster
4794 HQuorumPeer
```

Operation 3: $ hbase shell

We will enter the hbase shell and with that we can create tables in the shell and start our query operations.

Operation 4: $ version

The command will result the information about the “hbase”. It includes the version of Hbase and its release etc. This command will be executed inside the shell.

Operation 5: $ whoami

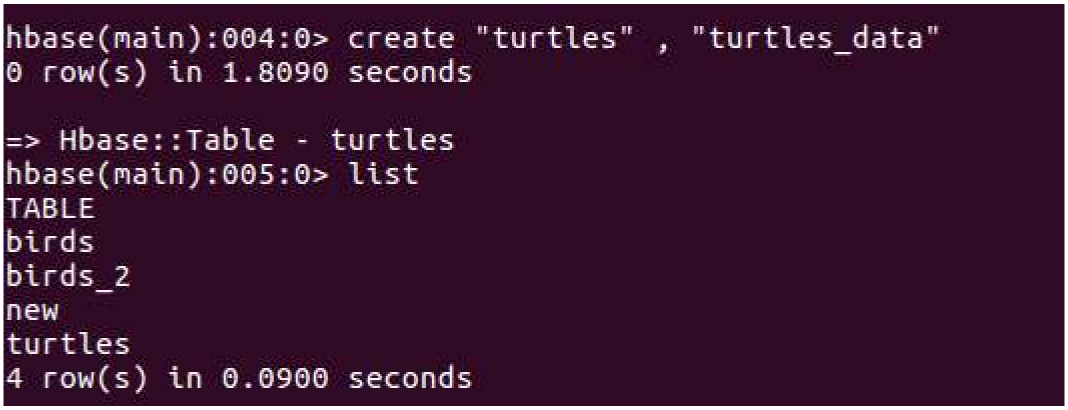
It will give the information of the user currently active and its permission status regarding “Hbase”. This command will be executed inside the shell.

Operation 6 - Create table: create "turtles", "turtles_data"

The command will be executed inside hbase shell and table “turtles” will be created. “turtles_data” is used for columns family name.

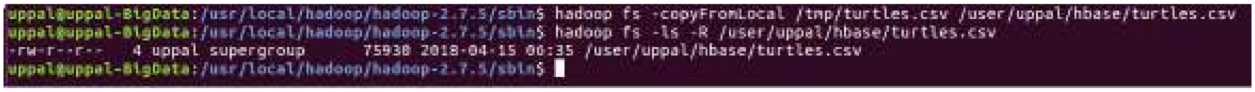
Operation 7 - $ list

It will display the available tables in the hbase system. Table is created and now will exit the hbase shell and execute “hbase” command to load the data in the table.



```
hbase(main):004:0> create "turtles" , "turtles_data"
0 row(s) in 1.8090 seconds

=> Hbase::Table - turtles
hbase(main):005:0> list
TABLE
birds
birds_2
new
turtles
4 row(s) in 0.0900 seconds
```

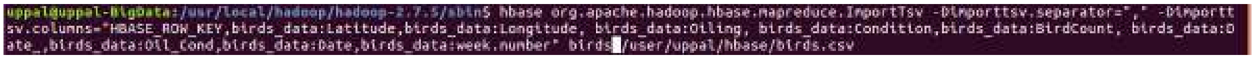
Operation 8 - Copy the file to Hadoop folder: $ hadoop fs -copyFromLocal /tmp/turtles.csv /user/uppal/hbase/turtles.csv



```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -copyFromLocal /tmp/turtles.csv /user/uppal/hbase/turtles.csv
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hadoop fs -ls -R /user/uppal/hbase/turtles.csv
-rw-r--r--   4 uppal supergroup      75938 2018-04-15 06:35 /user/uppal/hbase/turtles.csv
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ 
```

Operation 9: Load the Data: $ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator="," -Dimporttsv.columns="HBASE_ROW_KEY,turtles_data:obsData, turtles_data:State,turtles_data:Latitude,turtles_data:Longitude, turtles_data:New,turtles_data:Confirmed, turtles_data:Alive,turtles_data:Date, turtles_data:week.number" turtles /user/uppal/hbase/turtles.csv

This command will load the data in the “turtles” table. Column name “HBASE_ROW_KEY” is used to designate that this column should be used as the row key for each imported record. In our case it is the “species” column of “turtles.csv”, which will now be the “row key”.



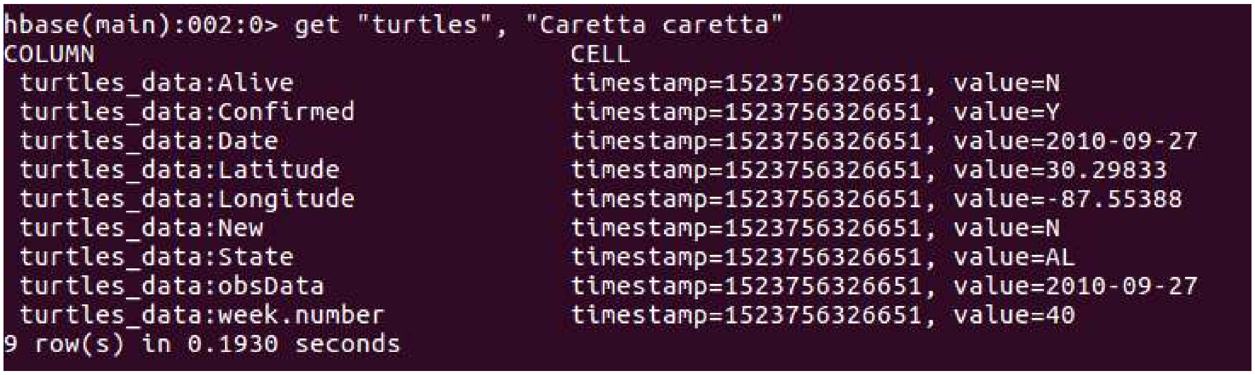
```
uppal@uppal-BigData:/usr/local/hadoop/hadoop-2.7.5/sbin$ hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator="," -Dimportt
sv.columns="HBASE_ROW_KEY,birds_data:Latitude,birds_data:Longitude, birds_data:Oiling, birds_data:Condition,birds_data:BirdCount, birds_data:D
ate_,birds_data:Oil_Cond,birds_data:Date,birds_data:week.number" birds /user/uppal/hbase/birds.csv
```

Operation 10- Hbase Shell again: $ Hbase shell

Table is loaded with data and we will execute our queries inside the hbase shell.

Query 1: get "turtles", "Caretta caretta"

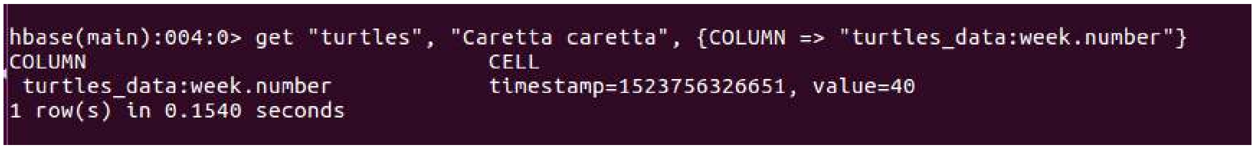
The query returns the values of all columns against row id of “Caretta caretta”.



```
hbase(main):002:0> get "turtles", "Caretta caretta"
COLUMN                                CELL
 turtles_data:Alive                   timestamp=1523756326651, value=N
 turtles_data:Confirmed               timestamp=1523756326651, value=Y
 turtles_data:Date                    timestamp=1523756326651, value=2010-09-27
 turtles_data:Latitude                timestamp=1523756326651, value=30.29833
 turtles_data:Longitude               timestamp=1523756326651, value=-87.55388
 turtles_data:New                     timestamp=1523756326651, value=N
 turtles_data:State                   timestamp=1523756326651, value=AL
 turtles_data:obsData                 timestamp=1523756326651, value=2010-09-27
 turtles_data:week.number             timestamp=1523756326651, value=40
9 row(s) in 0.1930 seconds
```

Query 2: get "turtles", "Caretta caretta", {COLUMN => "turtles_data:week.number"}

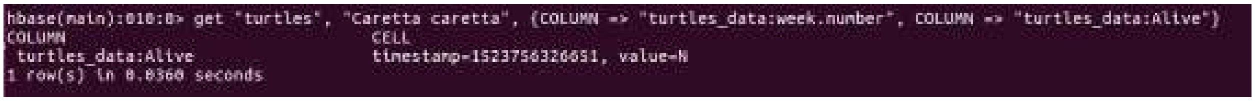
The query returns the values of all column “week.number” which is 40 against row id of “Caretta caretta”.



```
hbase(main):004:0> get "turtles", "Caretta caretta", {COLUMN => "turtles_data:week.number"}
COLUMN                                CELL
 turtles_data:week.number             timestamp=1523756326651, value=40
1 row(s) in 0.1540 seconds
```

Query 3: get "turtles", "Caretta caretta", {COLUMN => "turtles_data:week.number", COLUMN => "turtles_data:Alive"}

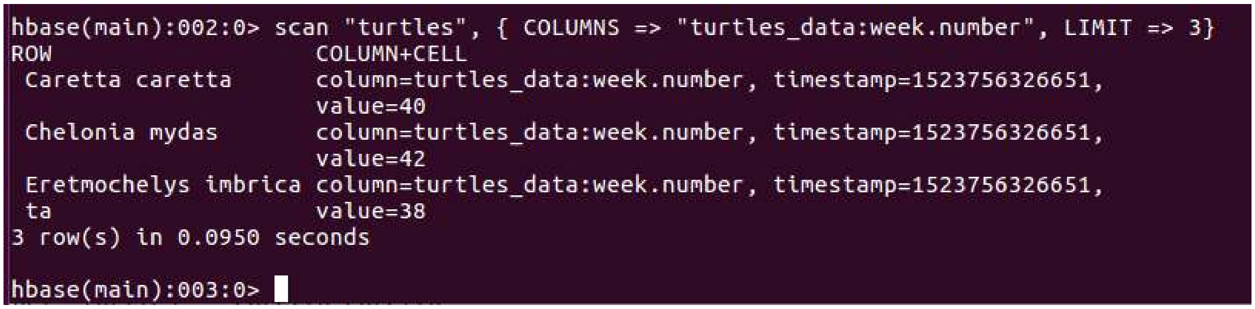
The query returns the values of week.number and if it is alive or not against row value of specie “Caretta caretta”.



```
hbase(main):010:0> get "turtles", "Caretta caretta", {COLUMN => "turtles_data:week.number", COLUMN => "turtles_data:Alive"}
COLUMN                                CELL
 turtles_data:Alive                   timestamp=1523756326651, value=N
1 row(s) in 0.0360 seconds
```

Query 4: scan "turtles", { COLUMNS => "turtles_data:week.number", LIMIT => 3}

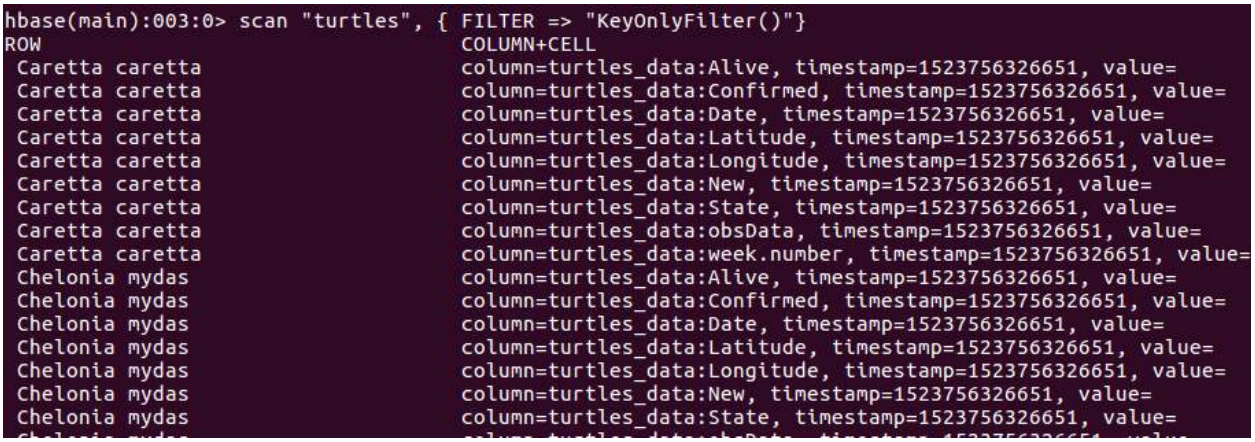
This query scans the table “turtles” and returns the three rows. The value shown is of 1 column only which is “week.number”.



```
hbase(main):002:0> scan "turtles", { COLUMNS => "turtles_data:week.number", LIMIT => 3}
ROW                   COLUMN+CELL
 Caretta caretta      column=turtles_data:week.number, timestamp=1523756326651,
                      value=40
 Chelonia mydas       column=turtles_data:week.number, timestamp=1523756326651,
                      value=42
 Eretmochelys imbrica column=turtles_data:week.number, timestamp=1523756326651,
 ta                   value=38
3 row(s) in 0.0950 seconds

hbase(main):003:0>
```

Query 5: scan "turtles", { FILTER => "KeyOnlyFilter()"}

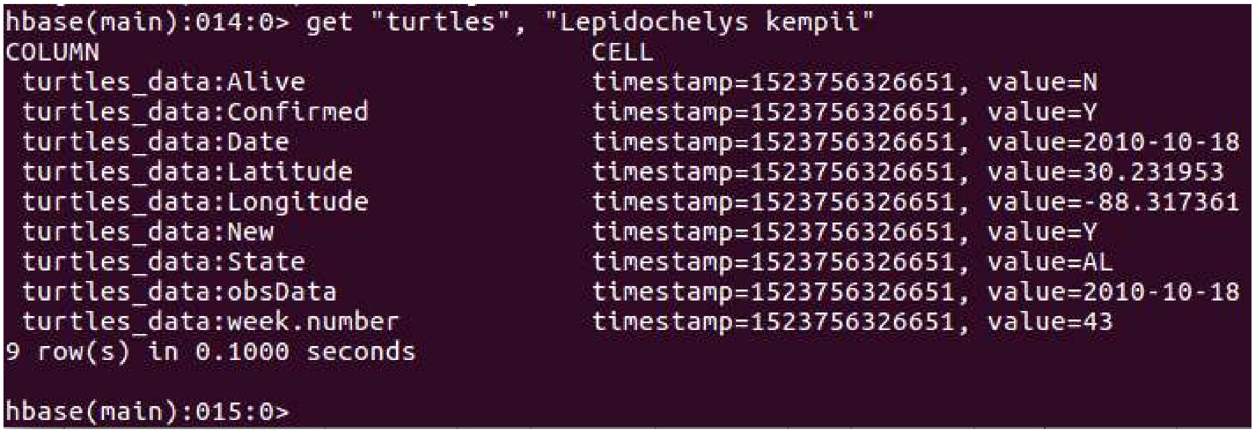
This filter does not take any arguments. It returns only the key component of each k.



```
hbase(main):003:0> scan "turtles", { FILTER => "KeyOnlyFilter()"}
ROW                                  COLUMN+CELL
 Caretta caretta                     column=turtles_data:Alive, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:Confirmed, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:Date, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:Latitude, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:Longitude, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:New, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:State, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:obsData, timestamp=1523756326651, value=
 Caretta caretta                     column=turtles_data:week.number, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:Alive, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:Confirmed, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:Date, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:Latitude, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:Longitude, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:New, timestamp=1523756326651, value=
 Chelonia mydas                      column=turtles_data:State, timestamp=1523756326651, value=
```

Query 6: get "turtles", "Lepidochelys kempii"

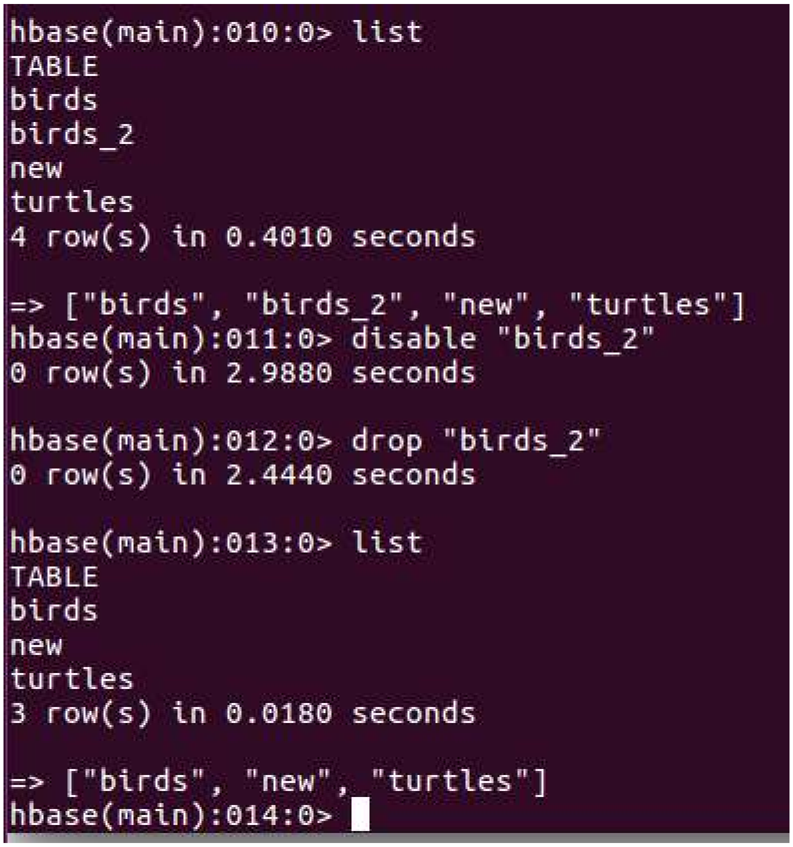
The query returns the values of all columns against row id of "Lepidochelys kempii".



```
hbase(main):014:0> get "turtles", "Lepidochelys kempii"
COLUMN                               CELL
 turtles_data:Alive                  timestamp=1523756326651, value=N
 turtles_data:Confirmed              timestamp=1523756326651, value=Y
 turtles_data:Date                   timestamp=1523756326651, value=2010-10-18
 turtles_data:Latitude               timestamp=1523756326651, value=30.231953
 turtles_data:Longitude              timestamp=1523756326651, value=-88.317361
 turtles_data:New                    timestamp=1523756326651, value=Y
 turtles_data:State                  timestamp=1523756326651, value=AL
 turtles_data:obsData                timestamp=1523756326651, value=2010-10-18
 turtles_data:week.number            timestamp=1523756326651, value=43
9 row(s) in 0.1000 seconds

hbase(main):015:0>
```

Query 7: Delete Table:

To delete a table, first we have to disable it and then we can drop the table. It is shown in the image below:



```
hbase(main):010:0> list
TABLE
birds
birds_2
new
turtles
4 row(s) in 0.4010 seconds

=> ["birds", "birds_2", "new", "turtles"]
hbase(main):011:0> disable "birds_2"
0 row(s) in 2.9880 seconds

hbase(main):012:0> drop "birds_2"
0 row(s) in 2.4440 seconds

hbase(main):013:0> list
TABLE
birds
new
turtles
3 row(s) in 0.0180 seconds

=> ["birds", "new", "turtles"]
hbase(main):014:0> 
```

4. SPARK BASIC OPERATIONS

Basic Operations:

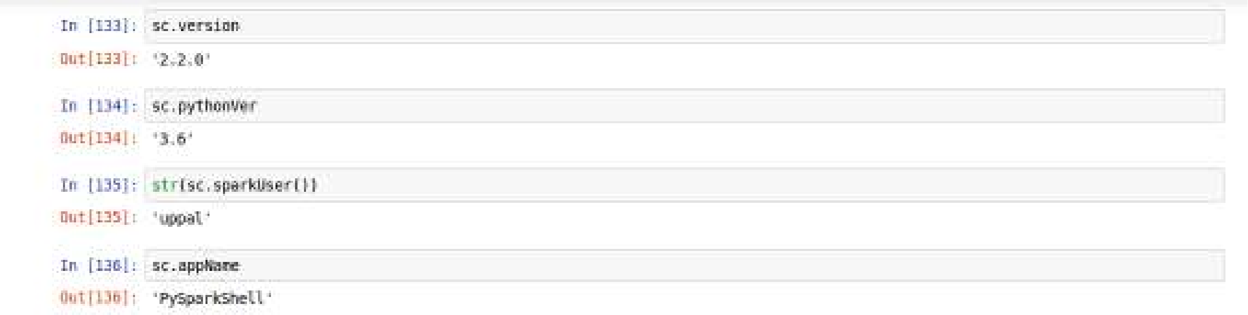
Some of the basic Operation of SPARK are below mentioned in the screenshots with their results.

Operation 1: sc.version () returns the version of PySpark.

Operation 2: sc.pythonVer returns the verison of python.

Operation 3: str(sc.sparkUser()) returns the information about the current Spark user.

Operation 4: sc.appName retruns the name of the app.



```
In [133]: sc.version
Out[133]: '2.2.0'

In [134]: sc.pythonVer
Out[134]: '3.6'

In [135]: str(sc.sparkUser())
Out[135]: 'uppal'

In [136]: sc.appName
Out[136]: 'PySparkShell'
```

Data Operations:

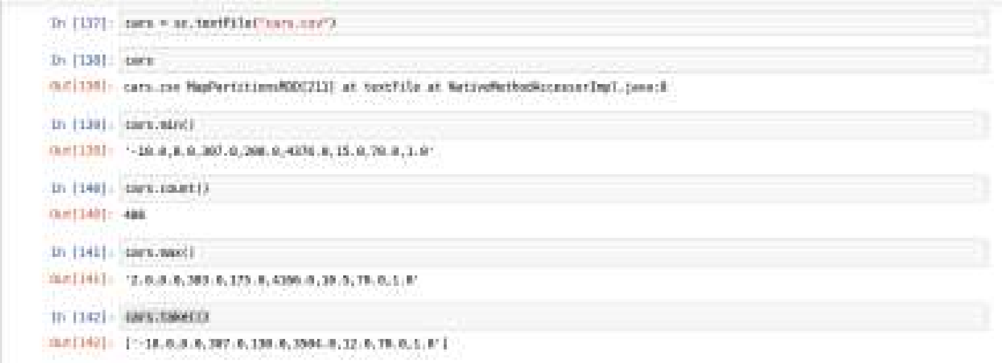
File can be read using Spark Context. To check the basic commands of Spark the dataset we are using is "cars.csv" downloaded from link " http://lib.stat.cmu.edu/datasets/cars.data ". Preprocessing of the data is not done but other than that we can apply some basic operations like: min, count and max etc.

Operation 5: Min() function returns the minimum value in each column.

Operation 6: Count() function returns the total number of rows or data samples in the dataset which is 406.

Operation 7: Max() function returns the maximum value in each column.

Operation 8: cars.take(1) returns the first sample and cars.take(2) returns 2 samples.



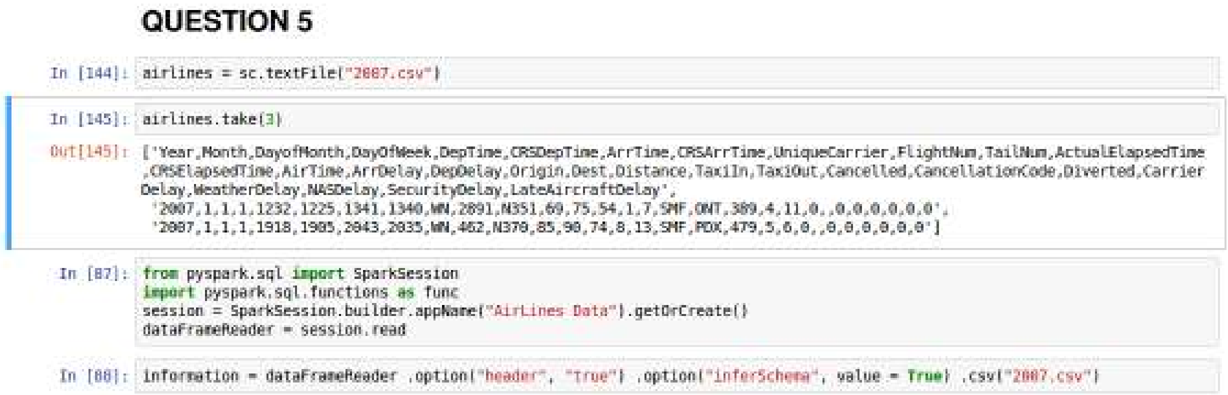
```
In [137]: cars = sc.textFile("cars.csv")

In [138]: cars
Out[138]: cars.csv MapPartitionsRDD[211] at textFile at NativeMethodAccessorImpl.java:0

In [139]: cars.min()
Out[139]: '-10.0,8.0,307.0,200.0,4376.0,15.0,70.0,1.0'

In [140]: cars.count()
Out[140]: 406

In [141]: cars.max()
Out[141]: '2.0,8.0,383.0,175.0,4390.0,10.5,70.0,1.0'

In [142]: cars.take(1)
Out[142]: ['18.0,8.0,307.0,130.0,3504.0,12.0,70.0,1.0']
```

5. QUESTION 5 –AIRLINE Data:

Airline Data of 2005 from link http://stat-computing.org/dataexpo/2009/2007.csv.bz2 is downloaded and used for this example.

Operation 1: Read Data

We used 2 ways to read the data. The image shows below how we can import the data inside SPARK using **SPARK CONTEXT** and **SPARK SESSION**.

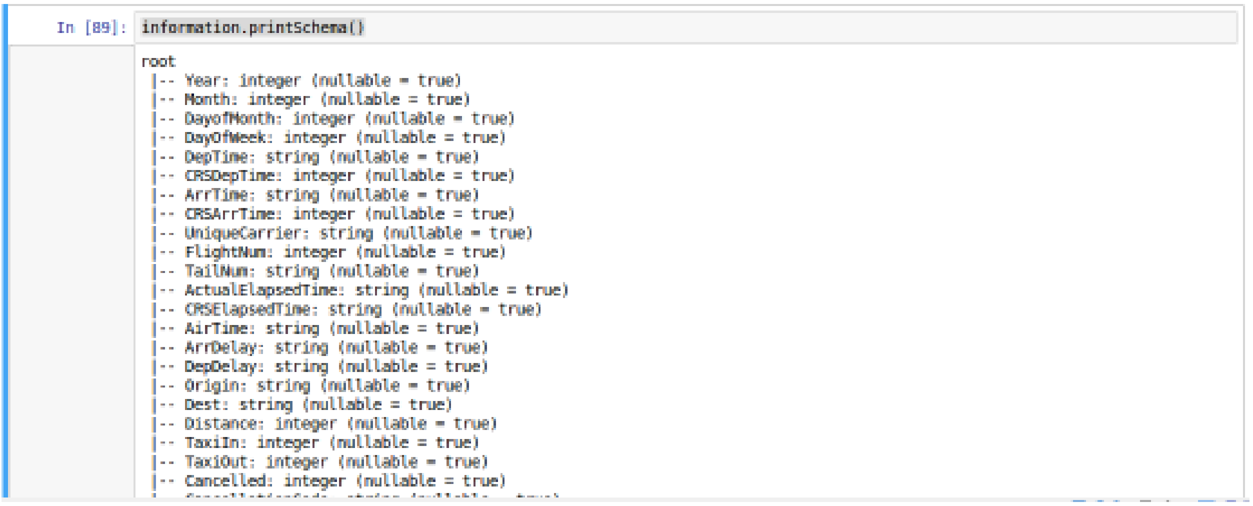


QUESTION 5

```
In [144]: airlines = sc.textFile("2007.csv")

In [145]: airlines.take(3)

Out[145]: ['Year,Month,DayofMonth,DayOfWeek,DepTime,CRSDepTime,ArrTime,CRSArrTime,UniqueCarrier,FlightNum,TailNum,ActualElapsedTime
          ,CRSElapsedTime,AirTime,ArrDelay,DepDelay,Origin,Dest,Distance,TaxiIn,TaxiOut,Cancelled,CancellationCode,Diverted,Carrier
          Delay,WeatherDelay,NASDelay,SecurityDelay,LateAircraftDelay',
           '2007,1,1,1,1232,1225,1341,1340,WN,2891,N351,69,75,54,1,7,SMF,ONT,389,4,11,0,,0,0,0,0,0,0',
           '2007,1,1,1,1918,1905,2043,2035,WN,462,N370,85,90,74,8,13,SMF,PDX,479,5,6,0,,0,0,0,0,0,0']

In [87]: from pyspark.sql import SparkSession
         import pyspark.sql.functions as func
         session = SparkSession.builder.appName("AirLines Data").getOrCreate()
         dataFrameReader = session.read

In [88]: information = dataFrameReader .option("header", "true") .option("inferSchema", value = True) .csv("2007.csv")
```

Operation 2: Print Header : $ information.printSchema()

".printSchema()" returns the name of the header/column names of the dataset.



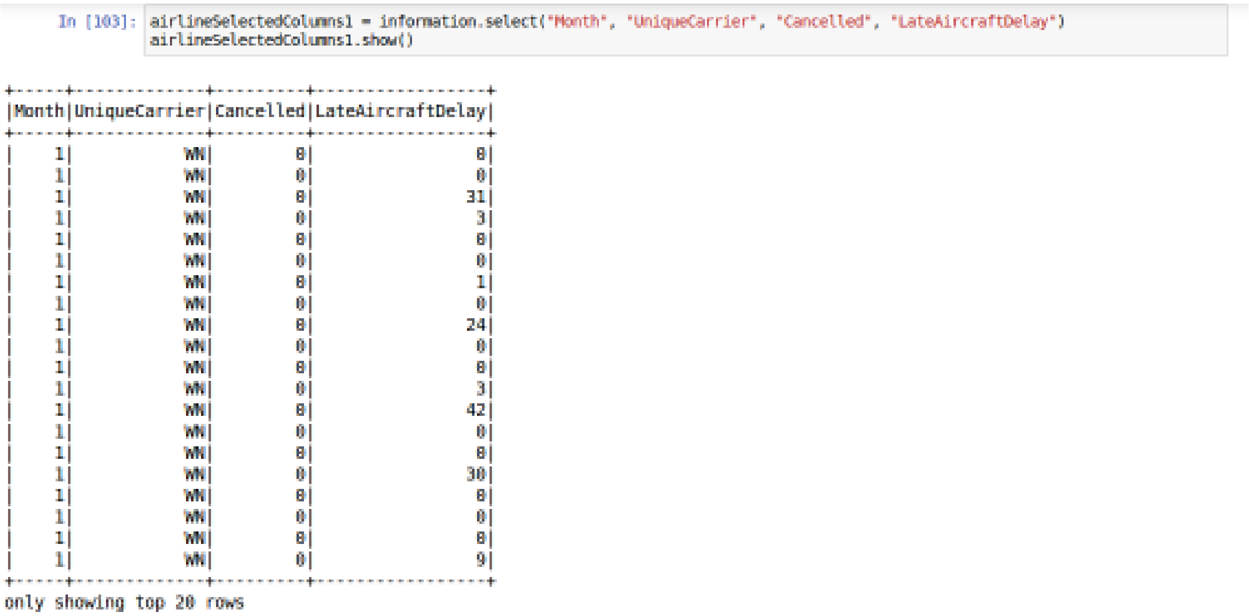
```
In [89]: information.printSchema()

         root
          |-- Year: integer (nullable = true)
          |-- Month: integer (nullable = true)
          |-- DayofMonth: integer (nullable = true)
          |-- DayOfWeek: integer (nullable = true)
          |-- DepTime: string (nullable = true)
          |-- CRSDepTime: integer (nullable = true)
          |-- ArrTime: string (nullable = true)
          |-- CRSArrTime: integer (nullable = true)
          |-- UniqueCarrier: string (nullable = true)
          |-- FlightNum: integer (nullable = true)
          |-- TailNum: string (nullable = true)
          |-- ActualElapsedTime: string (nullable = true)
          |-- CRSElapsedTime: string (nullable = true)
          |-- AirTime: string (nullable = true)
          |-- ArrDelay: string (nullable = true)
          |-- DepDelay: string (nullable = true)
          |-- Origin: string (nullable = true)
          |-- Dest: string (nullable = true)
          |-- Distance: integer (nullable = true)
          |-- TaxiIn: integer (nullable = true)
          |-- TaxiOut: integer (nullable = true)
          |-- Cancelled: integer (nullable = true)
```

Operation 3: Create Subset : $

airlineSelectedColumns1 = information**.select**("Month", "UniqueCarrier", "Cancelled", "LateAircraftDelay") **.show()**

Above command creates a subset of dataset with selected columns using "**.select()**" and showed on console using "**.show()**". Column names are mentioned as string inside the brackets.

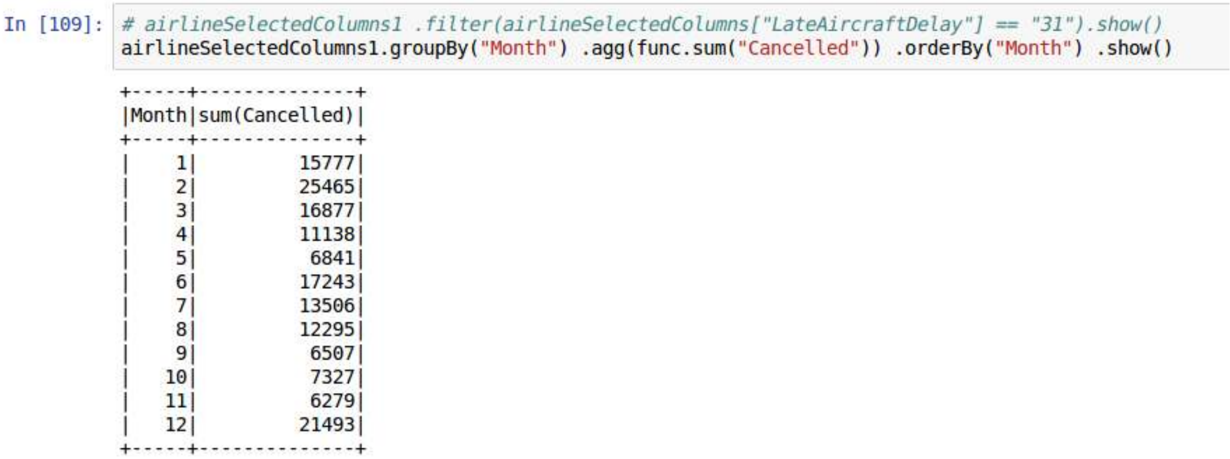
Top 20 results are displayed



```
In [103]: airlineSelectedColumns1 = information.select("Month", "UniqueCarrier", "Cancelled", "LateAircraftDelay")
          airlineSelectedColumns1.show()

+-----+-------------+---------+-----------------+
|Month|UniqueCarrier|Cancelled|LateAircraftDelay|
+-----+-------------+---------+-----------------+
|    1|           WN|        0|                0|
|    1|           WN|        0|                0|
|    1|           WN|        0|               31|
|    1|           WN|        0|                3|
|    1|           WN|        0|                0|
|    1|           WN|        0|                0|
|    1|           WN|        0|                1|
|    1|           WN|        0|                0|
|    1|           WN|        0|               24|
|    1|           WN|        0|                0|
|    1|           WN|        0|                0|
|    1|           WN|        0|                3|
|    1|           WN|        0|               42|
|    1|           WN|        0|                0|
|    1|           WN|        0|                0|
|    1|           WN|        0|               30|
|    1|           WN|        0|                0|
|    1|           WN|        0|                0|
|    1|           WN|        0|                0|
|    1|           WN|        0|                9|
+-----+-------------+---------+-----------------+
only showing top 20 rows
```

Query 1: Apply query on Subset : $ airlineSelectedColumns.groupBy("Month").agg(func.sum("Cancelled")) .orderBy("Month") .show()

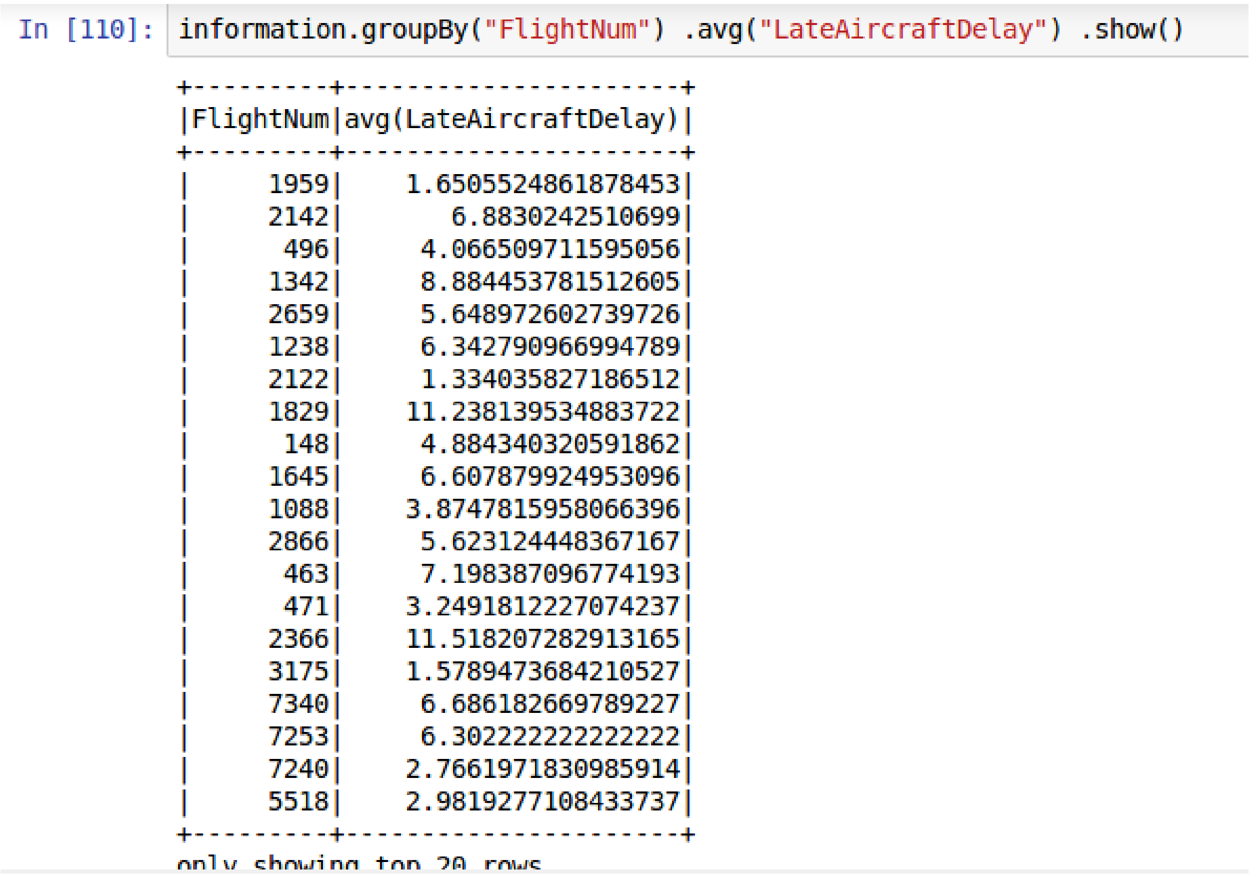
This commands shows results on selected criteria which in this case is total flights cancelled in each month of year 2007. The data is for year 2007 and the data is print in order of months.



```
In [109]: # airlineSelectedColumns1 .filter(airlineSelectedColumns["LateAircraftDelay"] == "31").show()
          airlineSelectedColumns1.groupBy("Month") .agg(func.sum("Cancelled")) .orderBy("Month") .show()

          +-----+--------------+
          |Month|sum(Cancelled)|
          +-----+--------------+
          |    1|         15777|
          |    2|         25465|
          |    3|         16877|
          |    4|         11138|
          |    5|          6841|
          |    6|         17243|
          |    7|         13506|
          |    8|         12295|
          |    9|          6507|
          |   10|          7327|
          |   11|          6279|
          |   12|         21493|
          +-----+--------------+
```

Query 2: Average Delay of Each flight : $ information.groupBy("FlightNum") .avg("LateAircraftDelay") .show()

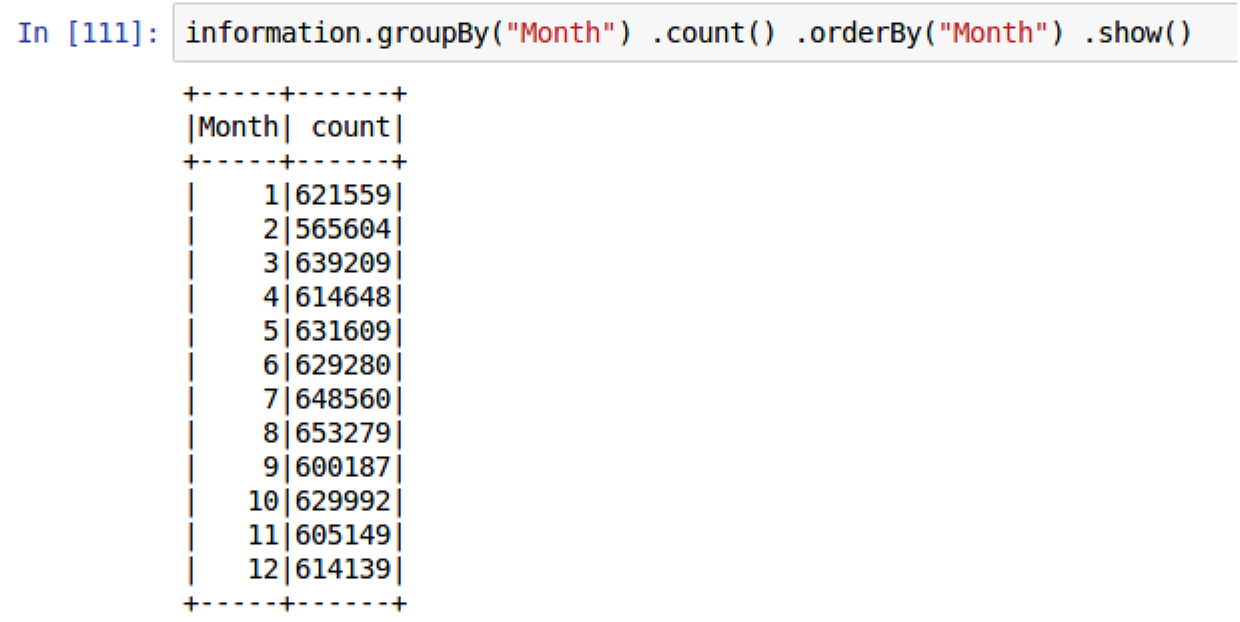
.avg() is used to calculate the average. Here we have printed average delay of each aircraft. The show is not ordered as “.orderBy()” is not used in this example.



```
In [110]: information.groupBy("FlightNum") .avg("LateAircraftDelay") .show()

+---------+----------------------+
|FlightNum|avg(LateAircraftDelay)|
+---------+----------------------+
|     1959|    1.6505524861878453|
|     2142|       6.8830242510699|
|      496|     4.066509711595056|
|     1342|     8.884453781512605|
|     2659|     5.648972602739726|
|     1238|     6.342790966994789|
|     2122|     1.334035827186512|
|     1829|    11.238139534883722|
|      148|     4.884340320591862|
|     1645|     6.607879924953096|
|     1088|    3.8747815958066396|
|     2866|     5.623124448367167|
|      463|     7.198387096774193|
|      471|    3.2491812227074237|
|     2366|     11.51820728291365|
|     3175|    1.5789473684210527|
|     7340|     6.686182669789227|
|     7253|     6.302222222222222|
|     7240|    2.7661971830985914|
|     5518|    2.9819277108433737|
+---------+----------------------+
only showing top 20 rows
```

Query 3: Month-wise samples

We can print month-wise samples for each month available in the dataset.

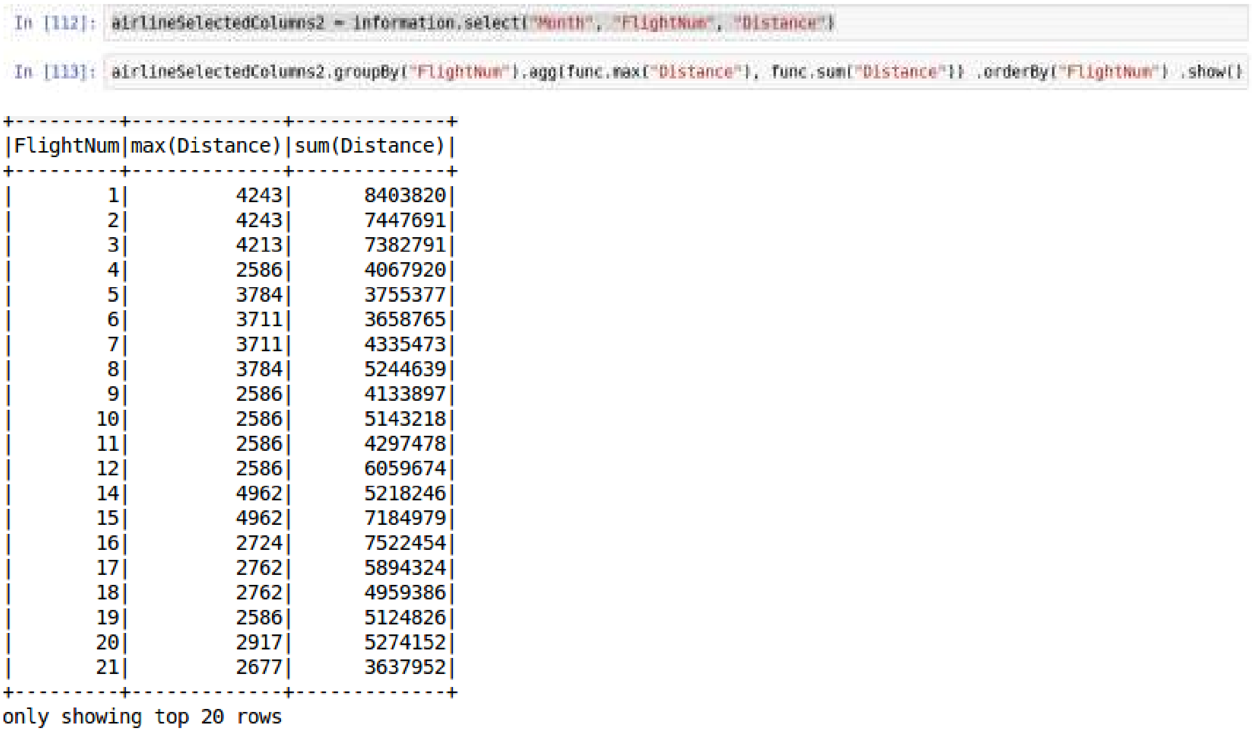


```
In [111]: information.groupBy("Month") .count() .orderBy("Month") .show()

+-----+------+
|Month| count|
+-----+------+
|    1|621559|
|    2|565604|
|    3|639209|
|    4|614648|
|    5|631609|
|    6|629280|
|    7|648560|
|    8|653279|
|    9|600187|
|   10|629992|
|   11|605149|
|   12|614139|
+-----+------+
```

Query 4: Average and Total distance travelled by each Flight

Command is: airlineSelectedColumns.groupBy("FlightNum").agg(func.max("Distance"), func.sum("Distance")) .orderBy("FlightNum") .show()

First we have created another subset of the main dataset with 3 columns only. Than we apply the query to calculate total distance travelled by a flight and maximum distance travelled by a flight. Top 20 results are also shown below:



```
In [112]: airlineSelectedColumns2 = information.select("Month", "FlightNum", "Distance")
```

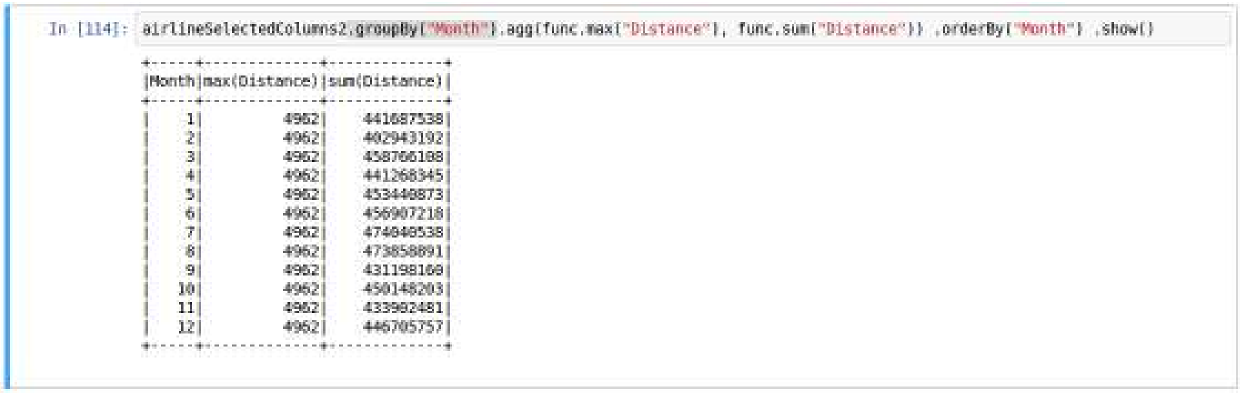
```
In [113]: airlineSelectedColumns2.groupBy("FlightNum").agg(func.max("Distance"), func.sum("Distance")) .orderBy("FlightNum") .show()
```

```
+---------+-------------+-------------+
|FlightNum|max(Distance)|sum(Distance)|
+---------+-------------+-------------+
|        1|         4243|      8403820|
|        2|         4243|      7447691|
|        3|         4213|      7382791|
|        4|         2586|      4067920|
|        5|         3784|      3755377|
|        6|         3711|      3658765|
|        7|         3711|      4335473|
|        8|         3784|      5244639|
|        9|         2586|      4133897|
|       10|         2586|      5143218|
|       11|         2586|      4297478|
|       12|         2586|      6059674|
|       14|         4962|      5218246|
|       15|         4962|      7184979|
|       16|         2724|      7522454|
|       17|         2762|      5894324|
|       18|         2762|      4959386|
|       19|         2586|      5124826|
|       20|         2917|      5274152|
|       21|         2677|      3637952|
+---------+-------------+-------------+
only showing top 20 rows
```

This result is displayed in ordered format starting from 1 as we have used “.orderBy()” call.

Query 5: Average and Total distance travelled in each Month

Command is: airlineSelectedColumns.groupBy("Month").agg(func.max("Distance"), func.sum("Distance")) .orderBy("Month") .show()



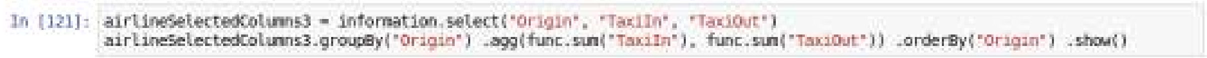
```
In [114]: airlineSelectedColumns2.groupBy("Month").agg(func.max("Distance"), func.sum("Distance")) .orderBy("Month") .show()

+-----+-------------+-------------+
|Month|max(Distance)|sum(Distance)|
+-----+-------------+-------------+
|    1|         4962|    441687538|
|    2|         4962|    402943192|
|    3|         4962|    458766188|
|    4|         4962|    441268345|
|    5|         4962|    453440873|
|    6|         4962|    456907218|
|    7|         4962|    474040538|
|    8|         4962|    473858891|
|    9|         4962|    431198169|
|   10|         4962|    450148203|
|   11|         4962|    433902481|
|   12|         4962|    446705757|
+-----+-------------+-------------+
```

Query 6: Origin of Flights and number of TaxiIn and TaxiOut done

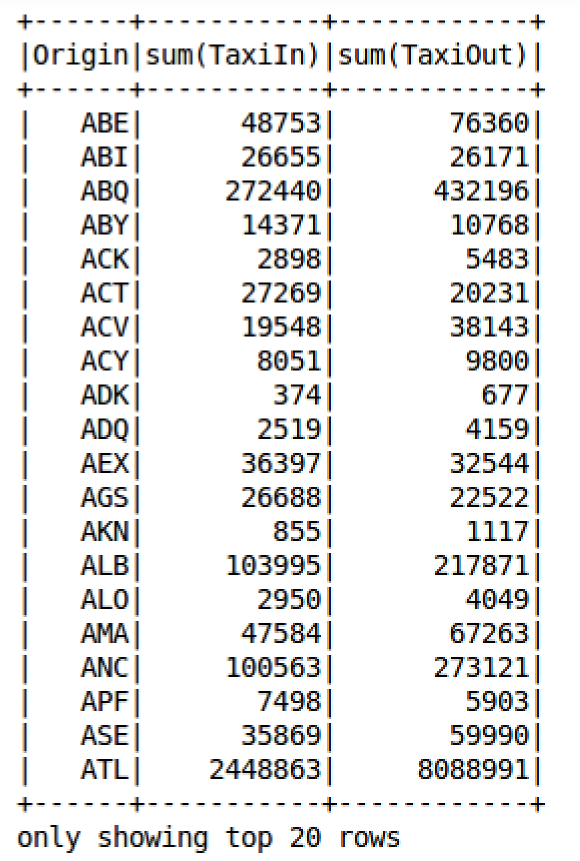
Command 1 is: airlineSelectedColumns3 = information.select("Origin", "TaxiIn", "TaxiOut")

Command 2 is: airlineSelectedColumns3.groupBy("Origin").agg(func.sum("TaxiIn"), func.sum("TaxiOut")) .orderBy("Origin") .show()

Here we have created 3rd subset based on origin. The results show sum of flights that “Taxied In” or “Taxied Out” from the “Origin”. Again top 20 results are displayed and order is based on “Origin”.



```
In [121]: airlineSelectedColumns3 = information.select("Origin", "TaxiIn", "TaxiOut")
          airlineSelectedColumns3.groupBy("Origin") .agg(func.sum("TaxiIn"), func.sum("TaxiOut")) .orderBy("Origin") .show()
```



```
+------+-----------+------------+
|Origin|sum(TaxiIn)|sum(TaxiOut)|
+------+-----------+------------+
|   ABE|      48753|       76360|
|   ABI|      26655|       26171|
|   ABQ|     272440|      432196|
|   ABY|      14371|       10768|
|   ACK|       2898|        5483|
|   ACT|      27269|       20231|
|   ACV|      19548|       38143|
|   ACY|       8051|        9800|
|   ADK|        374|         677|
|   ADQ|       2519|        4159|
|   AEX|      36397|       32544|
|   AGS|      26688|       22522|
|   AKN|        855|        1117|
|   ALB|     103995|      217871|
|   ALO|       2950|        4049|
|   AMA|      47584|       67263|
|   ANC|     100563|      273121|
|   APF|       7498|        5903|
|   ASE|      35869|       59990|
|   ATL|    2448863|     8088991|
+------+-----------+------------+
only showing top 20 rows
```