Assignment 1: Data Analysis

Teenu Prathyush
DCU Student Id: 21262966
Email: teenu.prathyush2@mail.dcu.ie

GitLab link: https://gitlab.computing.dcu.ie/prathyt2/ca675_cloud_technologies_assignment_1

In this assignment, I have perform data analysis on the top 200000 posts by ViewCount acquired on the platform StackExchange. The dataset contains 23 columns and around 200000 records which needs to be cleaned and processed before it is used for analysis as it contains a lot of messy data which is not needed in the analysis. The following technologies/platforms have been used in this assignment: Excel, RStudio, and Pig/Hive/MapReduce on Dataproc in Google Cloud Platform.

1. Documentation and steps undertaken for acquiring the top 200000 posts by ViewCount from StackExchange (Task 1):
Step 1: The following queries were executed one after the other using the data explorer feature on StackExchange (https://data.stackexchange.com/stackoverflow/query/new).

1. select count(*) from posts where posts.ViewCount > 100000
2. select top 50000 * from posts where posts.ViewCount > 100000 ORDER BY posts.ViewCount DESC
3. select top 50000 * from posts where posts.ViewCount < 127755 and posts.Id != 8618374 ORDER BY posts.ViewCount DESC
4. select top 50000 * from posts where posts.ViewCount < 74786 and posts.Id != 13836848 ORDER BY posts.ViewCount DESC
5. select top 50000 * from posts where posts.ViewCount < 53403 ORDER BY posts.ViewCount DESC

The 1st query is run to determine the number of records close to 50000 as StackExchange only allows to download 50000 records at a time. We assume that the view count is greater than 100000. In this case, we get a total of 68943 records.

The 2nd query is run to acquire the top 50000 posts having the most view count. From the previous query we know that the number of records having view count greater than 100000 is 68943 which is around 50000. So to extract the data we use – **ORDER by posts.ViewCount DESC** as it will determine the top 50000 records having the most view count, in descending order.

The 3rd query is run to determine the next 50000 posts having the most view count. We use the where clause - **posts.ViewCount < 127755 and posts.Id != 8618374**. This will ensure that there are no missing records in between queries.

The 4th query is run to determine the next 50000 posts having the most view count. We use the where clause - **posts.ViewCount < 74786 and posts.Id != 13836848.** This will ensure that there are no missing records and finally the 5th query is run to get the remaining posts having the most view count.

Step 2: After the queries have been executed, I have downloaded the four CSV files into the local system i.e., QueryResult1.csv, QueryResult2.csv, QueryResult3.csv, and QueryResult4.csv.

2. Steps undertaken to clean the data using RStudio (Tasks 2 & 3): The downloaded CSV files contains a lot of messy data which needs to be removed before loading it into the Google Cloud Platform for further processing. I have chosen RStudio to remove some of the messy data in the CSV files as I

found RStudio to be easy to navigate and code. The commands used in the R-Script are fairly simple and processes the data very quickly when compared to Python, Excel, or any other platform. In this case, RStudio has been used to clean only some of the messy data, further cleaning/processing of data has been done using PIG on GCP.

Link to the R-Script:
https://gitlab.computing.dcu.ie/prathyt2/ca675_cloud_technologies_assignment_1/blob/master/Code/clean_data.R

3. Steps undertaken to further clean and process the data using Apache Pig.

Apache Pig has been used in this assignment to further clean and process the data before loading into hive for querying. I found Pig Latin to be easy to use for processing the data when compared to other Hadoop platforms. In this case, Pig has been used to load the data, remove unnecessary columns, process the data and join the four CSV files which was cleaned earlier.

Link to the Pig Script:
https://gitlab.computing.dcu.ie/prathyt2/ca675_cloud_technologies_assignment_1/blob/master/Code/PigScript.pig

Step 1: Uploading the CSV files into the folder - cleaned_data on the Google Cloud Staging Bucket.

Step 2: Opening the SSH connection and entering the command pig. It will take us to the grunt shell. All the code executed below is provided in the link to the Pig Script mentioned above. The complete script is executed with the command:

exec gs://dataproc-staging-us-central1-795277444073-rzswho19/PigScript.pig

The following command is used for loading CSV files with support of multi-line fields.

define CSVExcelStorage org.apache.pig.piggybank.storage.CSVExcelStorage();

Step 3: Loading the data into Pig using the following code snippet.

csv1=load'gs://dataproc-staging-us-central1-795277444073rzswho19/cleaned_data/QueryResult1.csv' using CSVExcelStorage(',', 'YES_MULTILINE','NOCHANGE','SKIP_INPUT_HEADER') AS (Id:int, PostTypeId:int, AcceptedAnswerId:int, ParentId:int, CreationDate:chararray, DeletionDate:chararray, Score:int,ViewCount:int,Body:chararray,OwnerUserId:int,OwnerDisplayName:chararray,LastEditorUserId:int,LastEditorDisplayName:chararray,LastEditDate:chararray,LastActivityDate:chararray,Title:chararray,Tags:chararray,AnswerCount:int,CommentCount:int,FavoriteCount:int,ClosedDate:chararray,CommunityOwnedDate:chararray, ContentLicense:chararray);

The above code snippet is used to load the first CSV file – QueryResult1.csv into Pig environment. The YES_MULTILINE argument allows new lines inside of fields and the SKIP_INPUT_HEADER field skips the headers in the CSV file. The code to load the other three CSV files has been provided in the link to the Pig Script mentioned above.

Step 4: Joining all the four processed CSV files using the UNION command.

union_data = UNION csv1, csv2, csv3, csv4;

Step 5: Keeping only the necessary columns for processing.

required_data = FOREACH union_data GENERATE Id AS Id, Score AS Score, ViewCount AS ViewCount, Body AS Body, OwnerUserId AS OwnerUserId, Title AS Title, Tags AS Tags;

Step 6: Filtering data to remove all the null values from Score and OwnerUserId columns.

filter_data = FILTER required_data BY (OwnerUserId is NOT NULL);

final_data = FILTER filter_data BY (Score IS NOT NULL);

Step 7: Using the STORE command to store the processed data in the Google Cloud Staging Bucket

STORE final_data INTO 'gs://dataproc-staging-us-central1-795277444073-rzswho19/processed_data' USING org.apache.pig.piggybank.storage.CSVExcelStorage(',');

4. Using Hive to query the data to get the following:

Hive has been used in this assignment to query the data. I found Hive to be very useful for analysing the data as it is very similar to MySQL. It only requires a few lines of code and it processes data very quickly.

Link to the Hive Queries:
https://gitlab.computing.dcu.ie/prathyt2/ca675_cloud_technologies_assignment_1/blob/master/Code/HiveQueries.sql

Step 1: Loading processed_data into the local file system and then on to HDFS.

hdfs dfs -get "gs://dataproc-staging-us-central1-795277444073-rzswho19/processed_data" "/home/teenu_prathyush2/"

hdfs dfs -put "/home/teenu_prathyush2/processed_data/" "/user/pig"

Step 2: Opening Hive using the hive command on the terminal and then creating a table called as **stack_posts** using the create table command.

CREATE TABLE IF NOT EXISTS stack_posts(Id INT, Score INT, ViewCount INT, Body STRING, OwnerUserId STRING, Title STRING, Tags STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LOCATION '/user/pig/processed_data';

Step 3: Querying the table to check if the data has been loaded properly

SELECT Id, Score FROM stack_posts LIMIT 10;

Step 4: Query the table to get the results

Task 2.2.1. The top 10 posts by score

SELECT Id, Score, ViewCount, OwnerUserId, Title FROM stack_posts ORDER BY Score DESC LIMIT 10;

Task 2.2.2. The top 10 users by post score

SELECT OwnerUserId, SUM(Score) AS TOTAL_SCORE FROM stack_posts GROUP BY OwnerUserId ORDER BY TOTAL_SCORE DESC LIMIT 10;

Task 2.2.3. The number of distinct users, who used the word “cloud” in one of their posts

SELECT COUNT(DISTINCT OwnerUserId) AS distinct_users_count FROM stack_posts WHERE(lower(body) LIKE '%cloud%' OR lower(title) LIKE '%cloud%' OR lower(tags) LIKE '%cloud%');

Step 5: Creating a table called as top_users_scores to store the top 10 users by post scores

CREATE TABLE IF NOT EXISTS top_users_scores(OwnerUserId INT, TotalScore INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

INSERT INTO top_users_scores SELECT OwnerUserId, SUM(Score) AS TOTAL_SCORE FROM stack_posts GROUP BY OwnerUserId ORDER BY TOTAL_SCORE DESC LIMIT 10;

Step 6: Creating another table called as top_users_posts to store the text content from all of the top 10 users.

CREATE TABLE IF NOT EXISTS top_users_posts(OwnerUserId INT, Body STRING, Title STRING, Tags STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

INSERT INTO top_users_posts SELECT OwnerUserId, Body, Title, Tags FROM stack_posts WHERE OwnerUserId IN (SELECT OwnerUserId from top_users_scores) GROUP BY OwnerUserID, Body, Title, Tags;

Step 7: Copying the data into HDFS

INSERT OVERWRITE DIRECTORY '/user/hive/stack_data' ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' SELECT OwnerUserId, Body, Title, Tags FROM top_users_posts GROUP BY OwnerUserId, Body, Title, Tags;

5. Steps undertaken to calculate the per-user TF-IDF of the top 10 terms for each of the top 10 users. (Task 4)

The TF-IDF implementation has been accomplished in four different phases using three mappers and reducers each. A fourth mapper file is used to get the TF-IDF values of the top 10 terms for each of the top 10 users in a file. Since I have multiple mappers and reducer files, I have used MapReduce to implement TF-IDF as it is a programming model which processes data in a parallel manner and it can be implemented with Java/Python. I have used Python as it is user-friendly and easy to code.

Link to the mapper and reducer files:
https://gitlab.computing.dcu.ie/prathyt2/ca675_cloud_technologies_assignment_1/tree/master/Code/python_files

Step 1: Loading the mapreduce programs into local profile: /home/teenu_prathyush2 and giving full permissions to the directory – **python_files/**

hdfs dfs -get 'gs://dataproc-staging-us-central1-795277444073-rzswho19/python_files' '/home/teenu_prathyush2/'

chmod 777 -R /home/teenu_prathyush2/python_files/

Step 2: Executing the first mapper and reducer file. The output will be stored in HDFS in the directory /user/hive/output1

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -file /home/teenu_prathyush2/python_files/mapper1.py /home/teenu_prathyush2/python_files/reducer1.py -mapper "python mapper1.py" -reducer "python reducer1.py" -input /user/hive/stack_data/000000_0 -output /user/hive/output1

Step 3: Executing the second mapper and reducer file. The output will be stored in HDFS in the directory /user/hive/output2

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar –file

/home/teenu_prathyush2/python_files/mapper2.py /home/teenu_prathyush2/python_files/reducer2.py -mapper "python mapper2.py" -reducer "python reducer2.py" -input /user/hive/output1 -output /user/hive/output2

Step 4: Executing the third mapper and reducer file. The output will be stored in HDFS in the directory /user/hive/output3

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar -file /home/teenu_prathyush2/python_files/mapper3.py /home/teenu_prathyush2/python_files/reducer3.py -mapper "python mapper3.py" -reducer "python reducer3.py" -input /user/hive/output2 -output /user/hive/output3

Step 5: Executing the fourth mapper file. The output will be stored in HDFS in the directory /user/hive/output4

hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar –file /home/teenu_prathyush2/python_files/mapper4.py -mapper "python mapper4.py" -input /user/hive/output3 -output /user/hive/output4

Step 6: Creating an empty file – **output.csv** and merging all the files from outputdata4 into output.csv then replacing all the spaces and loading it into another CSV file – **output1.csv**.

touch /home/teenu_prathyush2/output.csv

hadoop fs -getmerge /user/hive/output4 /home/teenu_prathyush2/output.csv

sed -e 's/\s/,/g' /home/teenu_prathyush2/output.csv > /home/teenu_prathyush2/output1.csv

Step 7: Creating a table called **tfidf_data** and then loading the output (output1.csv) into the newly created table and then to view the per-user TF-IDF of the top 10 terms for each of the top 10 users.

CREATE TABLE IF NOT EXISTS tfidf_data(Term STRING, OwnerUserId INT, tfidf DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

LOAD DATA LOCAL INPATH '/home/teenu_prathyush2/output1.csv' OVERWRITE INTO TABLE tfidf_data;

SELECT rank, OwnerUserId as userid, tfidf as tfidf_value, term FROM (SELECT ROW_NUMBER() OVER(PARTITION BY OwnerUserId ORDER BY tfidf DESC) AS rank, * FROM tfidf_data) n WHERE rank <= 10 and OwnerUserID IS NOT NULL;

REFERENCES:

1. https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
2. https://pig.apache.org/docs/r0.17.0/
3. https://cwiki.apache.org/confluence/display/Hive/Tutorial
4. https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/
5. https://github.com/kirthy21/Data-Analysis-Stack-Exchange-Hadoop-Pig-Hive-MapReduce-TFIDF
6. https://github.com/swathikiran86/pig-Hive-programmming-on-StackExchange-data
7. https://github.com/rajesh-codes/Stack-Exchange-Data-Analysis
8. https://gitlab.computing.dcu.ie/khaira2/ca675_cloud_technologies_assignment_1/tree/master

SCREENSHOTS

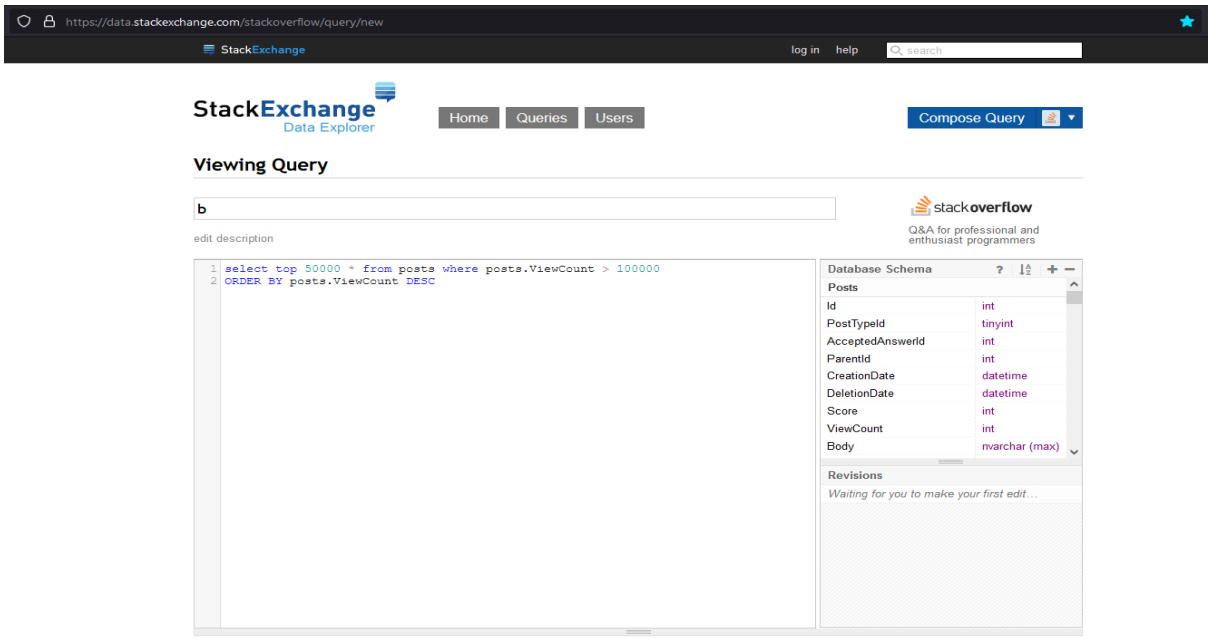


Figure 1: StackExchange Query 1

Figure 1 shows the first query being executed to acquire the top 50000 posts by ViewCount from StackExchange

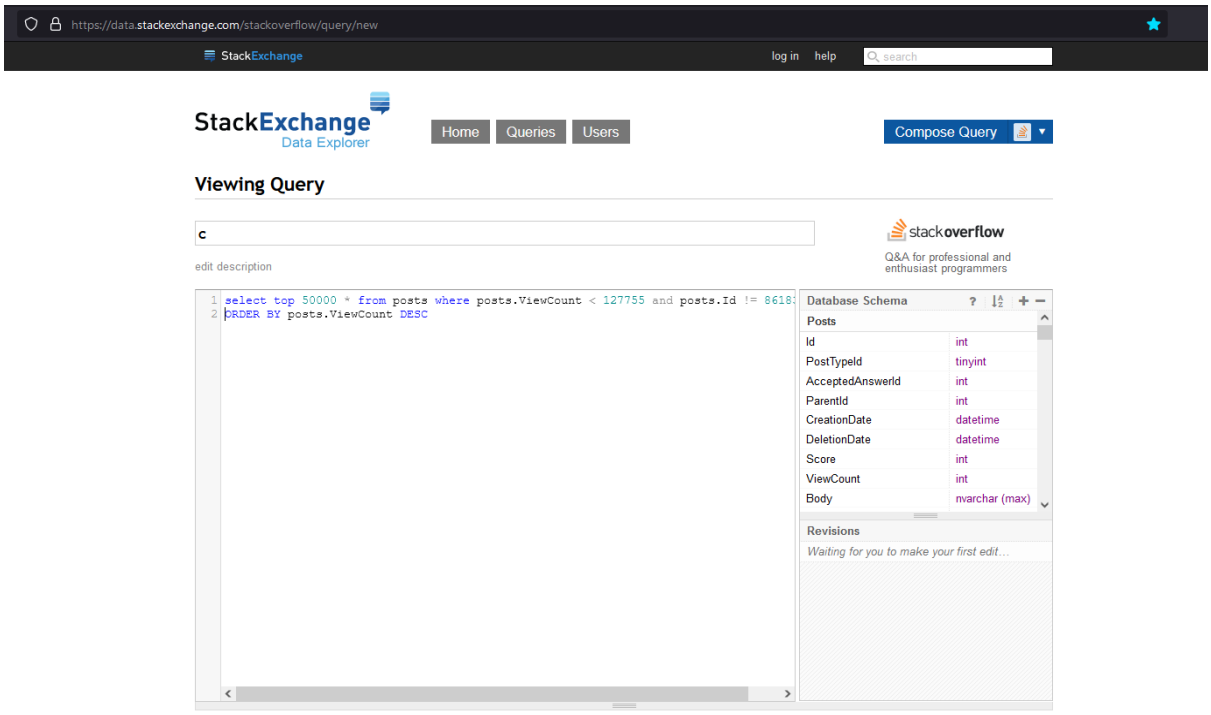


Figure 2: StackExchange Query 2

Figure 2 shows the second query being executed to acquire the next 50000 posts by ViewCount from StackExchange. The remaining 2 queries are run to get the next 100000 posts by ViewCount on StackExchange

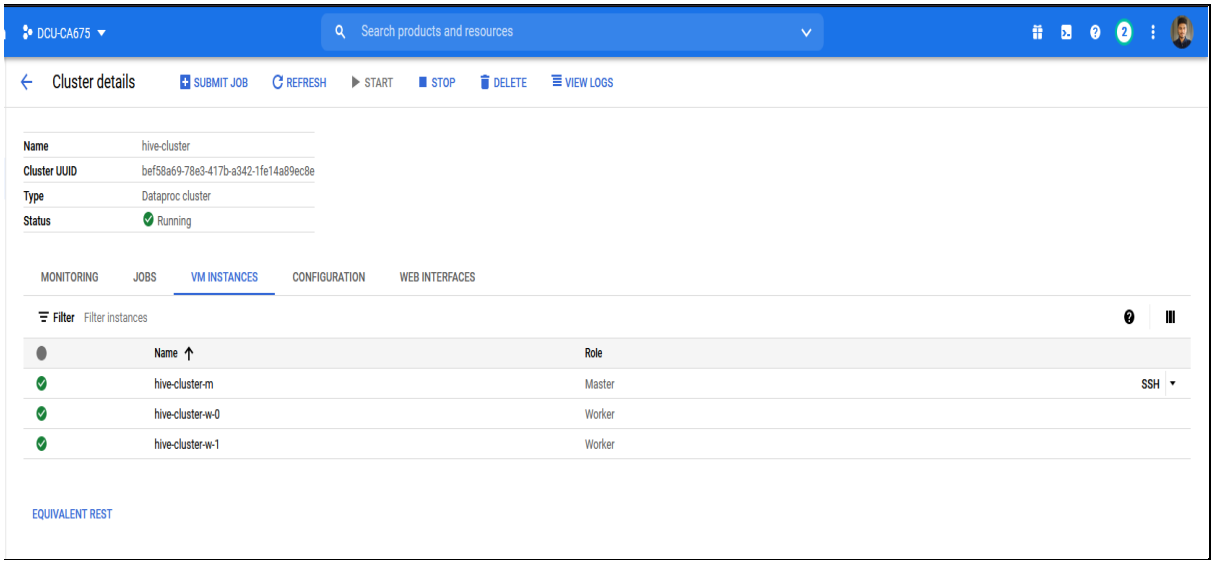


Figure 3: Cluster Details

Figure 3 shows the cluster details on Google Cloud Platform. The cluster name is **hive-cluster**

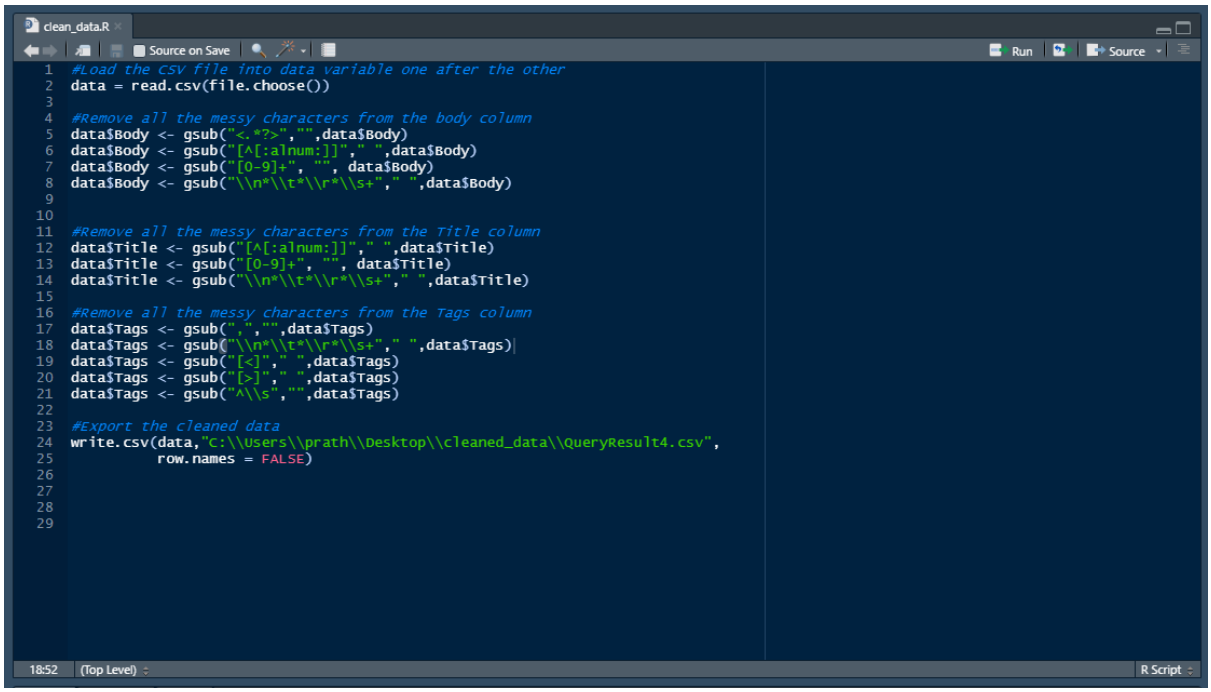


Figure 4: R-Script

Figure 4 shows the R-Script used for cleaning the messy data in the dataset. This includes removing special characters, numbers, new lines, tab spaces etc. Cleaning is done on the body, title, and tags column.

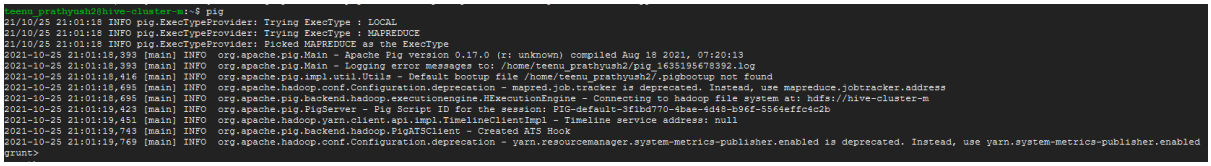


Figure 5: Grunt Shell

Figure 5 shows the Grunt shell being loaded using the PIG command

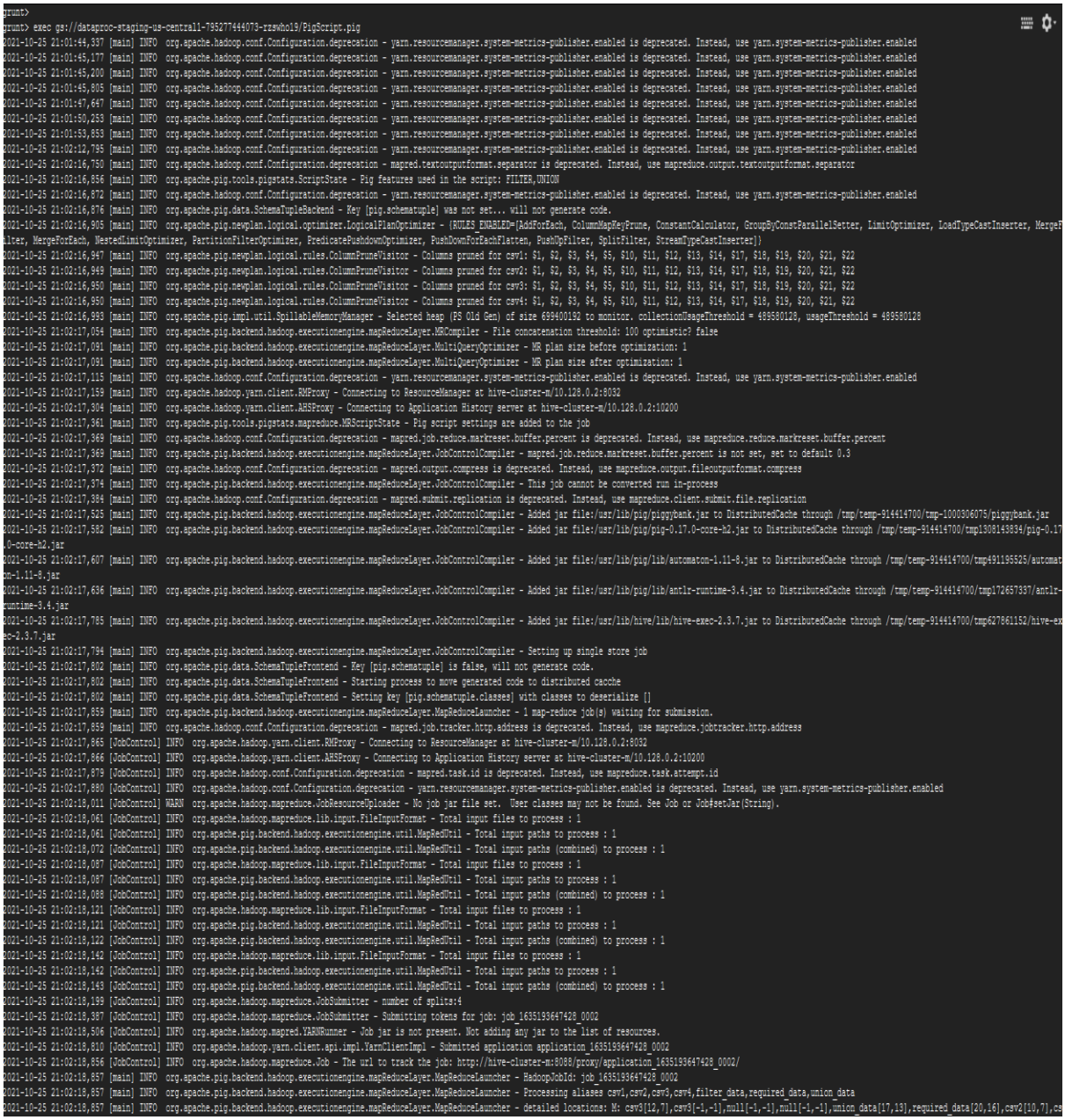


Figure 6: Executing PigScript.pig

Figure 6 shows the pig script – PigScript.pig being executed.

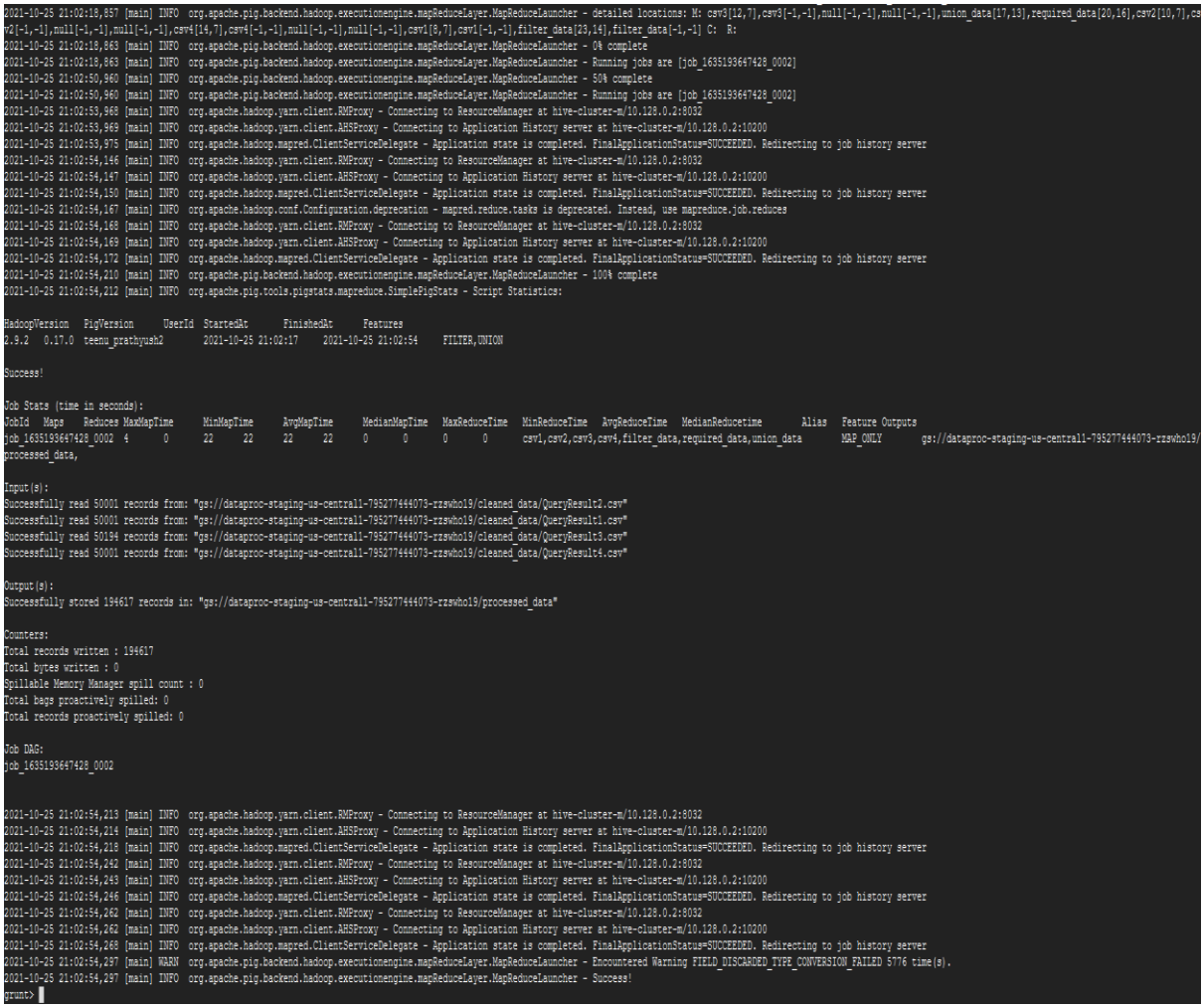


Figure 7 (a): Output of PigScript.pig

Figure 7(a) shows the output of PigScript.pig. It shows that 194617 records were processed successfully and stored in the folder – **processed_data** on Cloud Storage.

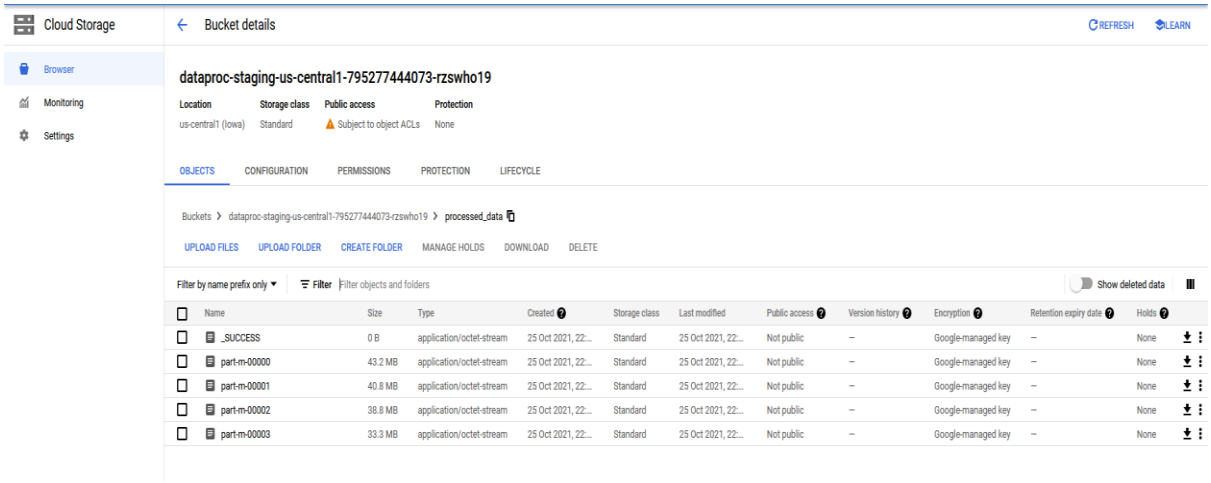


Figure 7 (b): Output of PigScript.pig

Figure 7(b) shows the output being stored in Cloud Storage. The output will be then moved to the HDFS.

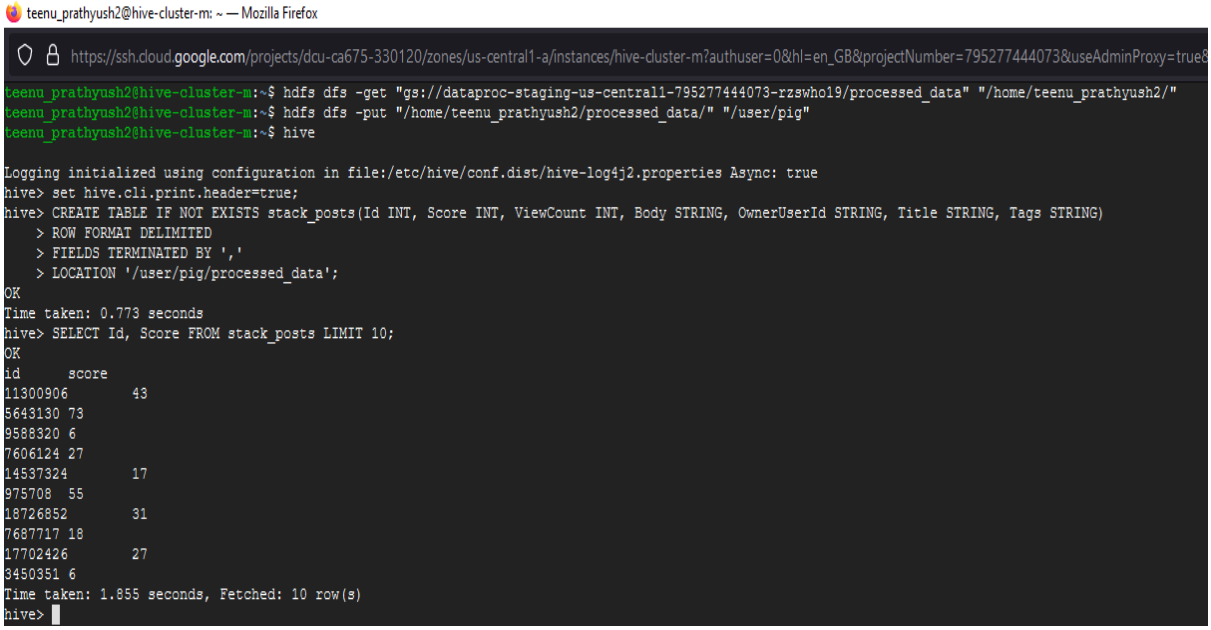


Figure 8: Create table stack_posts

In figure 8, a table called as stack_posts is created using the create table command and the previously processed data from Pig is loaded into the Hive table.

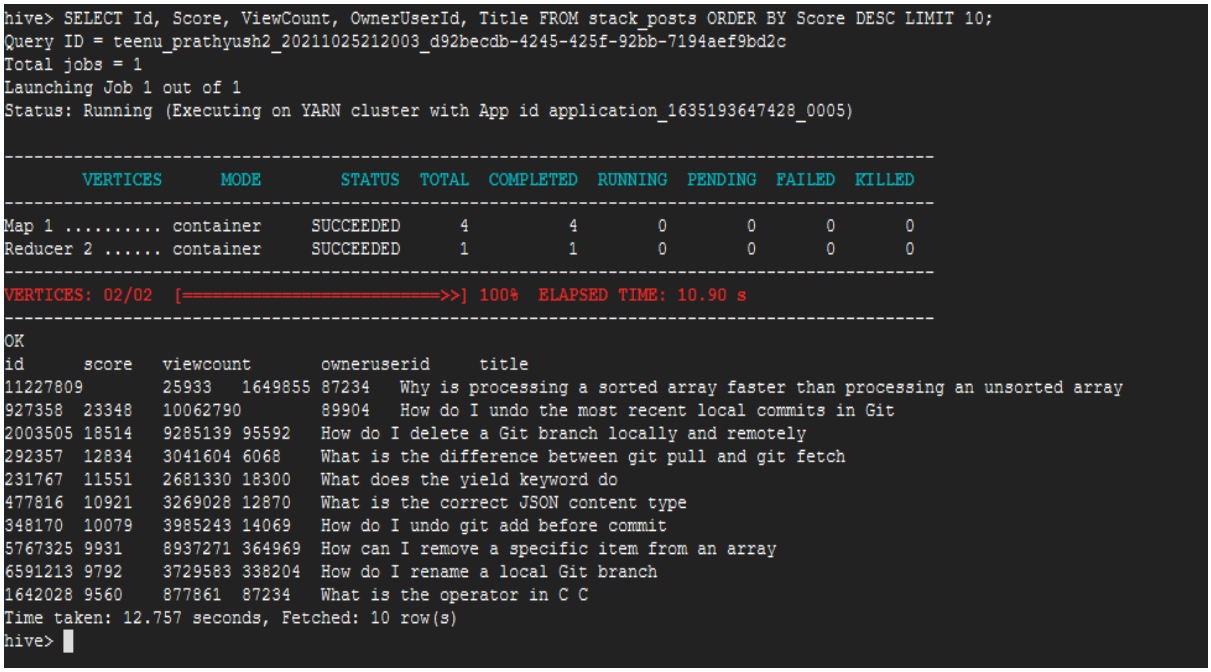


Figure 9: Query to get the top 10 posts by score

Figure 9 shows the query to get the top 10 posts by score and the result of the query (Task 2.2.1).

```
hive> SELECT OwnerUserId, SUM(Score) AS TOTAL_SCORE FROM stack_posts GROUP BY OwnerUserId ORDER BY TOTAL_SCORE DESC LIMIT 10;
Query ID = teenu_prathyush2_20211025212125_b69934fb-e48b-4e3e-b0db-1b6648817dfe
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1635193647428_0005)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 12.18 s
----------------------------------------------------------------------------------------------
OK
owneruserid     total_score
87234   37672
4883    28817
9951    26799
6068    25944
89904   24024
51816   23719
49153   20203
179736  19530
95592   19479
63051   19345
Time taken: 13.296 seconds, Fetched: 10 row(s)
hive>
```

Figure 10: Query to get the Top 10 users by post score

Figure 10 shows the query to get the top 10 users by post score and the result of the query (Task 2.2.2)

```
hive> SELECT COUNT(DISTINCT OwnerUserId) AS distinct_users_count FROM stack_posts
    > WHERE(lower(body) LIKE '%cloud%' OR lower(title) LIKE '%cloud%' OR lower(tags) LIKE '%cloud%');
Query ID = teenu_prathyush2_20211025212303_f964eb57-ec55-4834-9e2f-b0dd47087bde
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1635193647428_0005)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 .......... container     SUCCEEDED      4          4        0        0       0       0
Reducer 2 ...... container     SUCCEEDED      1          1        0        0       0       0
Reducer 3 ...... container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 03/03  [==========================>>] 100%  ELAPSED TIME: 15.16 s
----------------------------------------------------------------------------------------------
OK
distinct_users_count
915
Time taken: 16.083 seconds, Fetched: 1 row(s)
hive>
```

Figure 11: No. of Distinct Users using the word cloud

Figure 11 shows the query to get the number of distinct users who have used the word cloud in their posts and the result of the query (Task 2.2.3).

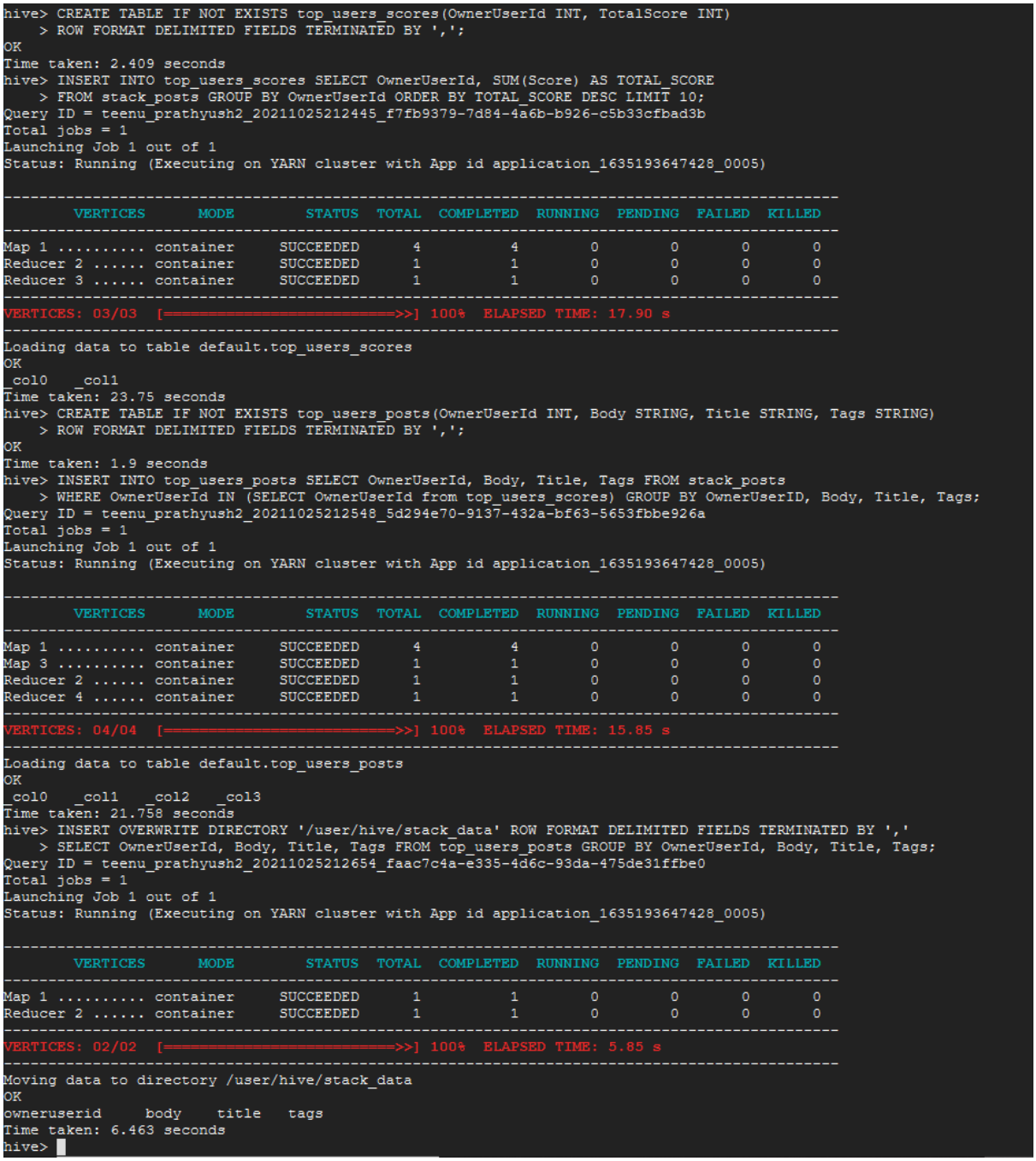


Figure 12: Creating table top_users_scores and top_users_posts

Figure 12 shows the creation of tables top_users_scores and top_users_posts which will be needed to calculate TFIDF later.

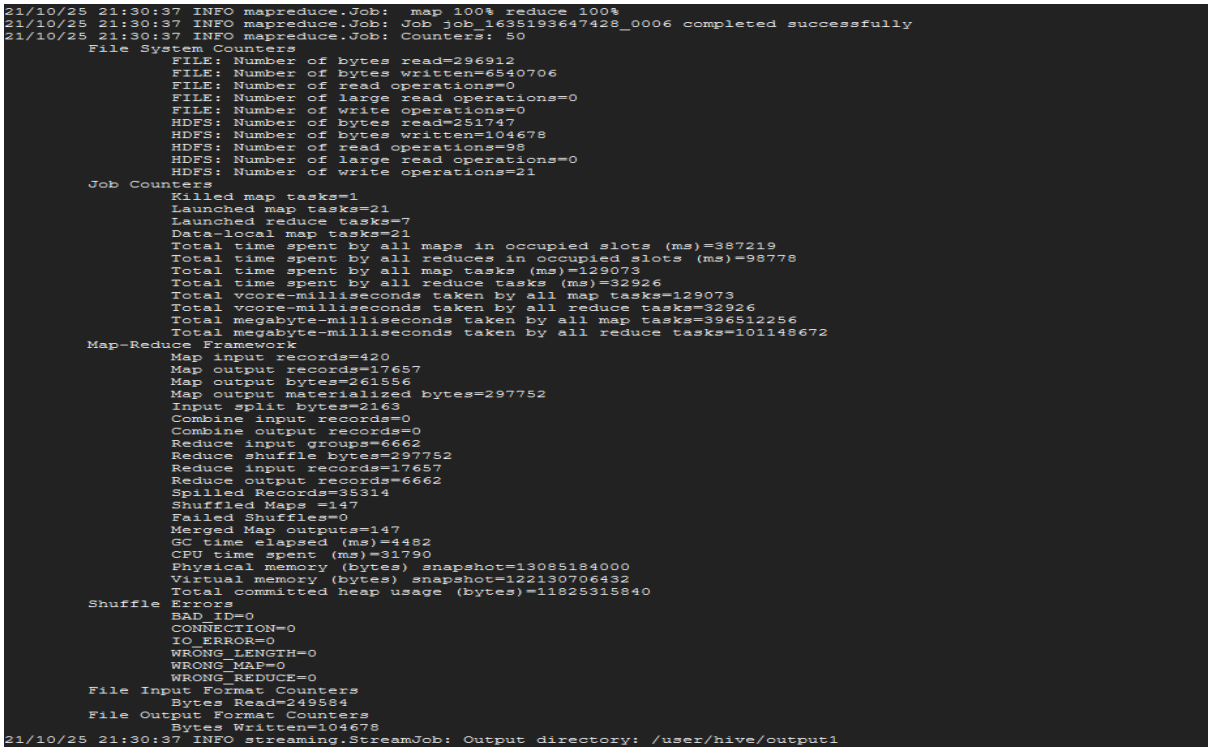


Figure 13: Output of Mapper1 and Reducer1 files

Figure 13 shows the output of the 1st Mapper and 1st Reducer files. The output is stored in HDFS - /user/hive/output1

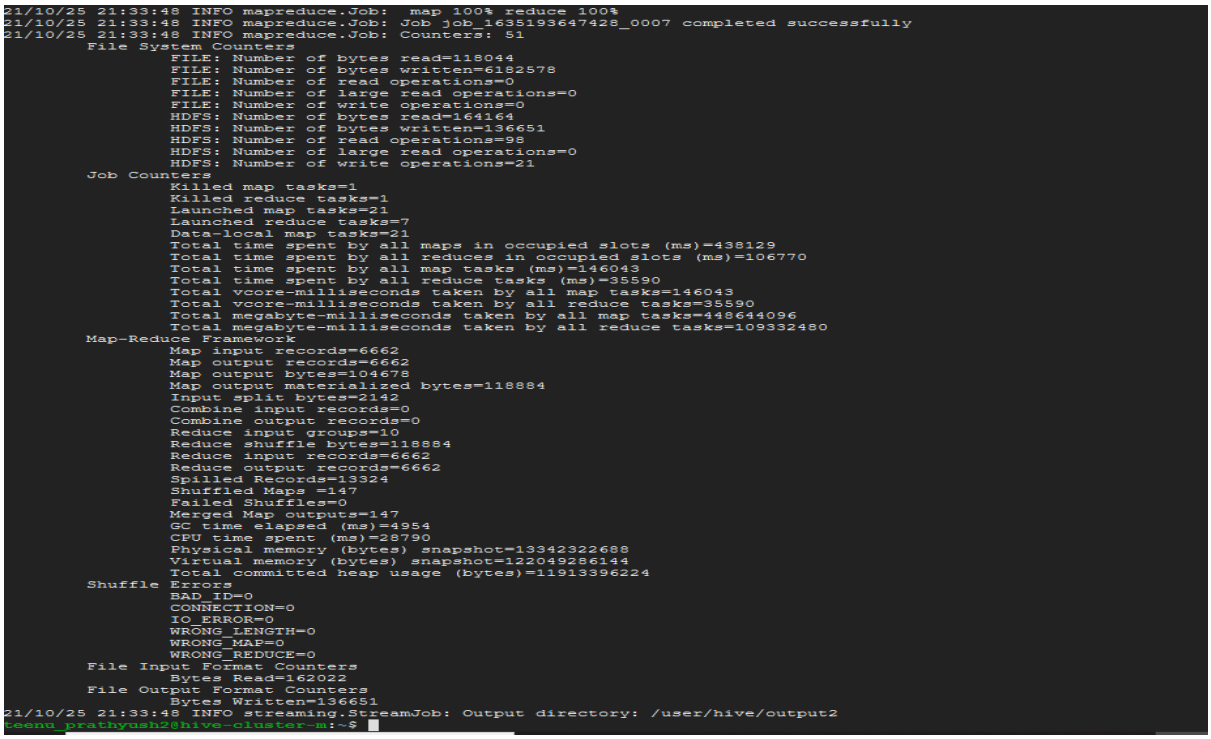


Figure 14: Output of Mapper2 and Reducer2 files

Figure 14 shows the output of the 2nd Mapper and 2nd Reducer files. The output is stored in HDFS - /user/hive/output2

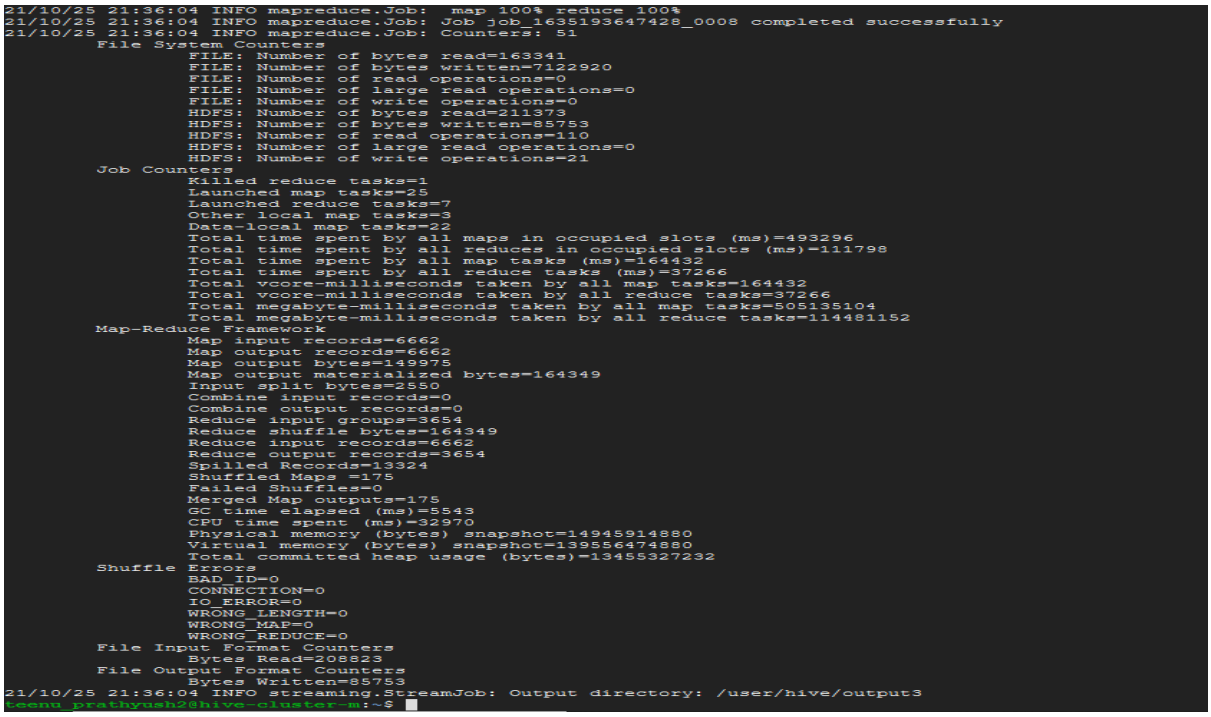


Figure 15: Output of Mapper3 and Reducer3 files

Figure 15 shows the output of the 3rd Mapper and 3rd Reducer files. The output is stored in HDFS - /user/hive/output3

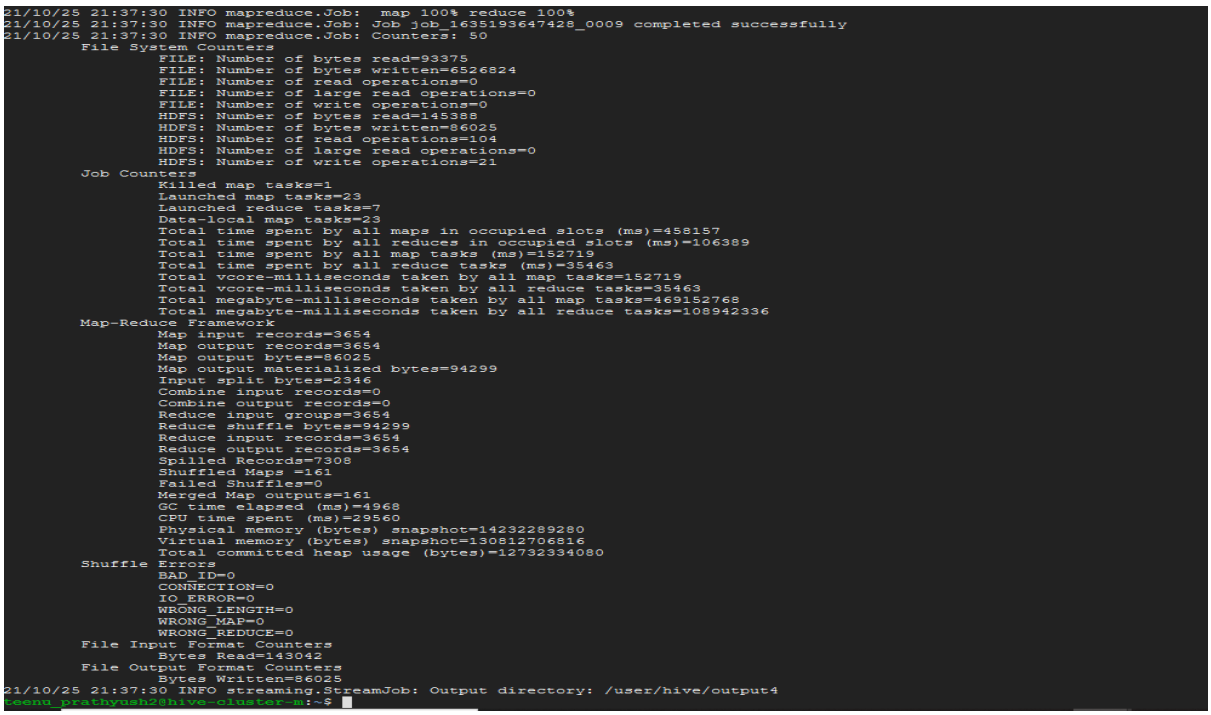


Figure 16: Output of Mapper4 file

Figure 16 shows the output of the 4th Mapper file. The final output is stored in HDFS - /user/hive/output4

```
teenu_prathyush2@hive-cluster-m:~$ touch /home/teenu_prathyush2/output.csv
teenu_prathyush2@hive-cluster-m:~$ hadoop fs -getmerge /user/hive/output4 /home/teenu_prathyush2/output.csv
teenu_prathyush2@hive-cluster-m:~$ sed -e 's/\s/,/g' /home/teenu_prathyush2/output.csv > /home/teenu_prathyush2/output1.csv
teenu_prathyush2@hive-cluster-m:~$ ls
output.csv  output1.csv  processed_data  pyhon files
```

Figure 17: Merging the output and replacing the spaces

Figure 17 shows the merging of all the output parts into a csv file (output.csv) and then replacing all the spaces and copying it to another csv file (output1.csv).

```
teenu_prathyush2@hive-cluster-m:~$ hive

Logging initialized using configuration in file:/etc/hive/conf.dist/hive-log4j2.properties Async: true
hive> set hive.cli.print.header=true;
hive> CREATE TABLE IF NOT EXISTS tfidf_data(Term STRING, OwnerUserId INT, tfidf DOUBLE) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
OK
Time taken: 2.441 seconds
hive> LOAD DATA LOCAL INPATH '/home/teenu_prathyush2/output1.csv' OVERWRITE INTO TABLE tfidf_data;
Loading data to table default.tfidf_data
OK
Time taken: 2.628 seconds
```

Figure 18: Loading the output into the table - tfidf_data

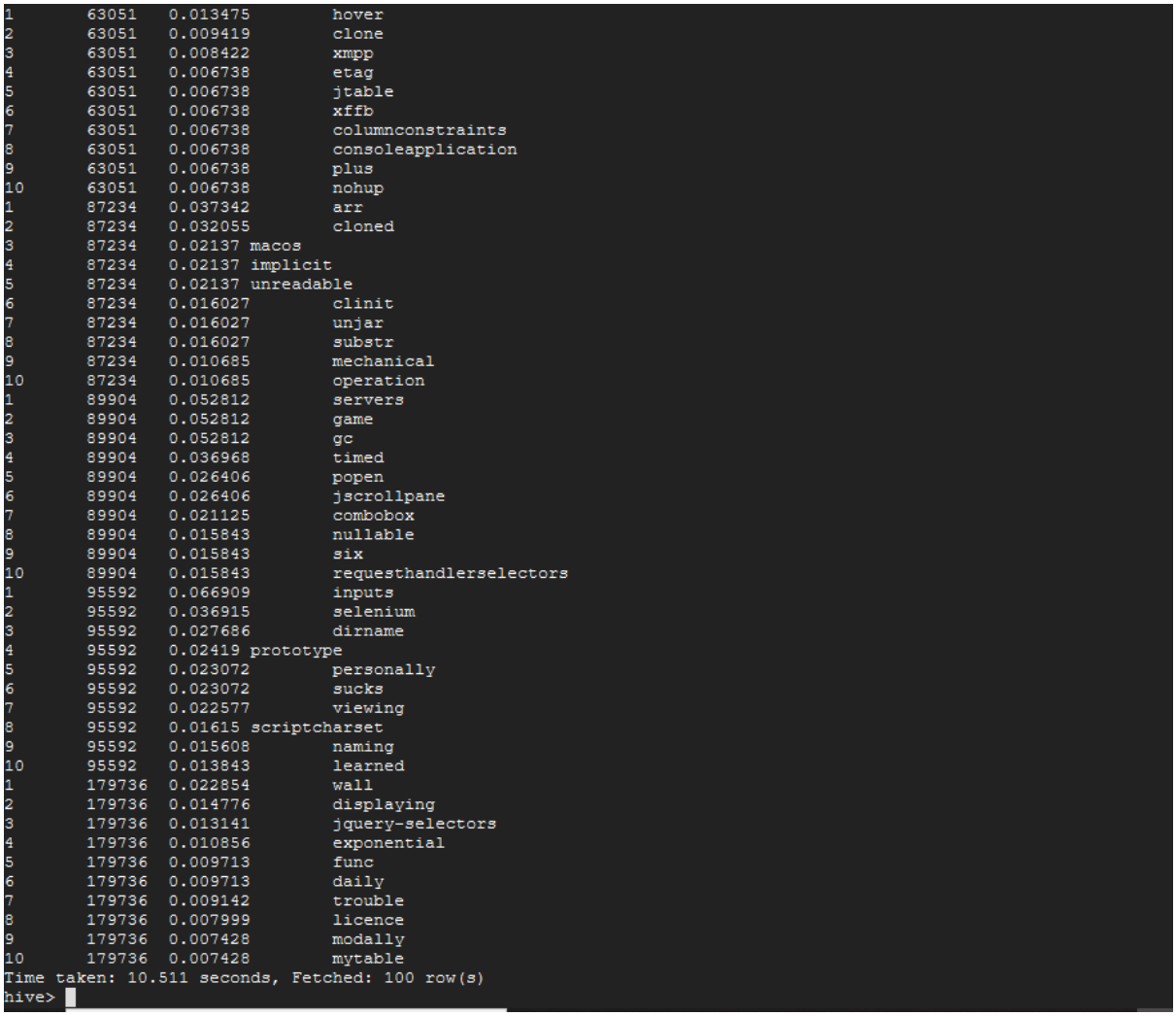
Figure 18 shows the creation of the table – tfidf_data and loading of the contents of output1.csv into tfidf_data table.

```
hive> SELECT rank, OwnerUserId as userid, tfidf as tfidf_value, term
    > FROM (SELECT ROW_NUMBER() OVER(PARTITION BY OwnerUserId ORDER BY tfidf DESC) AS rank, * FROM tfidf_data) n WHERE rank <= 10 and OwnerUserID IS NOT NULL;
Query ID = teenu_prathyush2_20211026095252_95813d50-e005-4e67-b0c7-30deffdf0b30
Total jobs = 1
Launching Job 1 out of 1
Status: Running (Executing on YARN cluster with App id application_1635239831360_0004)

----------------------------------------------------------------------------------------------
        VERTICES      MODE        STATUS  TOTAL  COMPLETED  RUNNING  PENDING  FAILED  KILLED
----------------------------------------------------------------------------------------------
Map 1 ..........   container     SUCCEEDED      1          1        0        0       0       0
Reducer 2 ......   container     SUCCEEDED      1          1        0        0       0       0
----------------------------------------------------------------------------------------------
VERTICES: 02/02  [==========================>>] 100%  ELAPSED TIME: 6.24 s
----------------------------------------------------------------------------------------------
OK
rank    userid  tfidf_value     term
1       4883    0.037364        writeobject
2       4883    0.023353        untracked
3       4883    0.014012        showattributes
4       4883    0.009341        trading
5       4883    0.009341        menu
6       4883    0.009341        remy
7       4883    0.009341        components
8       4883    0.009341        indexview
9       4883    0.009341        initially
10      4883    0.009341        offs
1       6068    0.017696        listening
2       6068    0.016655        asp
3       6068    0.013532        support
4       6068    0.013532        putting
5       6068    0.012491        ndez
6       6068    0.012491        spas
7       6068    0.009369        shaded
8       6068    0.008328        conditionmet
9       6068    0.008328        vectors
10      6068    0.007287        comp
1       9951    0.016316        omitted
2       9951    0.012237        fetched
3       9951    0.012237        dhtml
4       9951    0.012237        oriented
5       9951    0.012237        remember
6       9951    0.010197        benefits
7       9951    0.010197        resemblance
8       9951    0.010197        everybody
9       9951    0.008553        dictionaries
10      9951    0.008158        mytype
1       49153   0.02817 lower
2       49153   0.011985        person
3       49153   0.011635        utils
4       49153   0.009186        corresponding
5       49153   0.007349        tee
6       49153   0.007349        focus
7       49153   0.007349        saytime
8       49153   0.006124        parseprofilesjson
9       49153   0.005512        somebody
10      49153   0.005512        busy
1       51816   0.030152        grab
2       51816   0.018393        store
3       51816   0.016081        nvozeorryltracbmexugsdghuroiptnph
4       51816   0.015076        network
5       51816   0.012061        enums
6       51816   0.010051        wp
7       51816   0.010051        flash
8       51816   0.010051        noemit
9       51816   0.00843 buttons
10      51816   0.00804 focused
1       63051   0.013475        hover
2       63051   0.009419        clone
3       63051   0.008422        xmpp
4       63051   0.006738        etag
```

Figure 19(a): (Task 4) Per-User TFIDF of the top 10 terms for each of the top 10 users

Figure 19(a) shows the Per-User TFIDF values of the top 10 terms for each of the top 10 users.



```
1       63051   0.013475        hover
2       63051   0.009419        clone
3       63051   0.008422        xmpp
4       63051   0.006738        etag
5       63051   0.006738        jtable
6       63051   0.006738        xffb
7       63051   0.006738        columnconstraints
8       63051   0.006738        consoleapplication
9       63051   0.006738        plus
10      63051   0.006738        nohup
1       87234   0.037342        arr
2       87234   0.032055        cloned
3       87234   0.02137 macos
4       87234   0.02137 implicit
5       87234   0.02137 unreadable
6       87234   0.016027        clinit
7       87234   0.016027        unjar
8       87234   0.016027        substr
9       87234   0.010685        mechanical
10      87234   0.010685        operation
1       89904   0.052812        servers
2       89904   0.052812        game
3       89904   0.052812        gc
4       89904   0.036968        timed
5       89904   0.026406        popen
6       89904   0.026406        jscrollpane
7       89904   0.021125        combobox
8       89904   0.015843        nullable
9       89904   0.015843        six
10      89904   0.015843        requesthandlerselectors
1       95592   0.066909        inputs
2       95592   0.036915        selenium
3       95592   0.027686        dirname
4       95592   0.02419 prototype
5       95592   0.023072        personally
6       95592   0.023072        sucks
7       95592   0.022577        viewing
8       95592   0.01615 scriptcharset
9       95592   0.015608        naming
10      95592   0.013843        learned
1       179736  0.022854        wall
2       179736  0.014776        displaying
3       179736  0.013141        jquery-selectors
4       179736  0.010856        exponential
5       179736  0.009713        func
6       179736  0.009713        daily
7       179736  0.009142        trouble
8       179736  0.007999        licence
9       179736  0.007428        modally
10      179736  0.007428        mytable
Time taken: 10.511 seconds, Fetched: 100 row(s)
hive>
```

Figure 19(b): (Task 4) Per-User TFIDF of the top 10 terms for each of the top 10 users

Figure 19(b) shows the Per-User TFIDF values of the top 10 terms for each of the top 10 users.

APPENDIX (SCREENSHOTS)