**Stack Overflow Posts Data Analysis**

|  |  |  |
| --- | --- | --- |
| Name: Nikhil Kolloju | Student No: 21261762 | Email: Nikhil.kolloju2@mail.dcu.ie |

Git Hub Link: <https://github.com/NKOLLOJU2/CA675_Cloud_Assignment1>

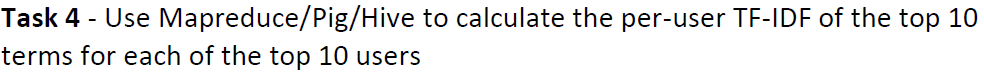
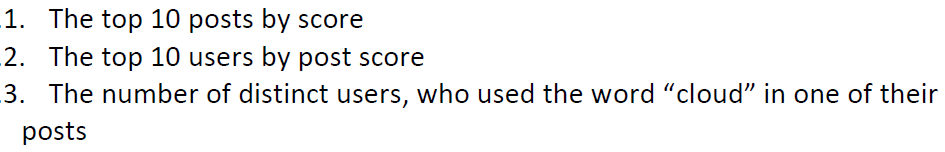
## **Requirement:**

The Assignment is to use Cloud Technologies and perform Data Analysis on Stack Overflow posts data. This assignment has 4 tasks described as follows:

**Task 1:** Fetch top 200,000 posts from Stack Exchange sorted by View Count.

**Task 2:** Extract, Transform and Load the fetched data using MapReduce/Hive/Pig.

**Task 3:** Fetch the following data using MapReduce/Hive/Pig



## **Steps for Task completion:**

1. Data Extraction from Stack Exchange.
2. Moving the extracted data from Stack Exchange into HDFS.
3. Cleaning the extracted data using PIG
4. Creating a table to store the cleaned data using HIVE
5. Running the queries to analyse table data for Task 3 using HIVE
6. Run Query to find TF-IDF for Task 4 using HIVE

## **Process Flow of Task completion:**

|  |
| --- |
|  |

## **1. Data Extraction from Stack Exchange:**

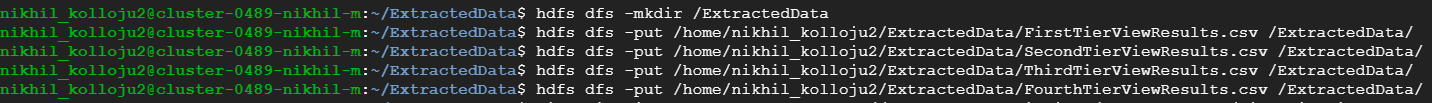
The [website](https://data.stackexchange.com/stackoverflow/query/new) uses SQL language to run the desired queries on the available tables. Since the maximum limit a query can fetch on the website is only 50,000 records, I have written multiple queries with different ranges to fetch data with 50,000 records each run and with 4 such iterations I was be able to download the top 200,000 records in csv format with 50,000 records in each csv file. The runtime to rank all the records in the Posts table was too long and would timeout, so it was important to filter out the data first and then rank. I first selected data from Posts table with ViewCount greater than 35,000. It amounted to more than 200,000 records and so I ranked this data using ROW\_NUMBER() [g] based on descending ViewCount. From the ranked data, I got the first 50,000 posts using a nested query. Similar approach has been done to next set of queries with the rank ranging from 50,001 to 100,000 and 100,001 to 150,000 and 150,001 to 200,000. A few records have been removed to avoid duplicate records in the data extraction. Fetched data has been downloaded into 4 CSV files.

Below are the set of queries used:

|  |  |
| --- | --- |
| SELECT A.\* FROM (SELECT \*  , ROW\_NUMBER() OVER (ORDER BY ViewCount DESC) rank  FROM Posts  WHERE ViewCount > 35000) A  WHERE A.rank BETWEEN 1 AND 50000)  ORDER BY A.rank ASC; | SELECT A.\* FROM (SELECT \*  , ROW\_NUMBER() OVER (ORDER BY ViewCount DESC) rank  FROM Posts  WHERE ViewCount > 35000) A  WHERE A.rank BETWEEN 50001 AND 100000  AND A.Id NOT IN (8733179) --to remove duplicate records  ORDER BY A.rank ASC; |
| SELECT A.\* FROM (SELECT \*  , ROW\_NUMBER() OVER (ORDER BY ViewCount DESC) rank  FROM Posts  WHERE ViewCount > 35000) A  WHERE A.rank BETWEEN 100000 AND 150000  AND A.Id NOT IN (8733179,54444538) --to remove duplicate records  ORDER BY A.rank ASC; | SELECT A.\* FROM (SELECT \*  , ROW\_NUMBER() OVER (ORDER BY ViewCount DESC) rank  FROM Posts  WHERE ViewCount > 35000) A  WHERE A.rank BETWEEN 150001 AND 200000  AND A.Id NOT IN (8733179,54444538,9587907) --to remove duplicate records  ORDER BY A.rank ASC; |

**2. Uploading the CSV files to HDFS:**

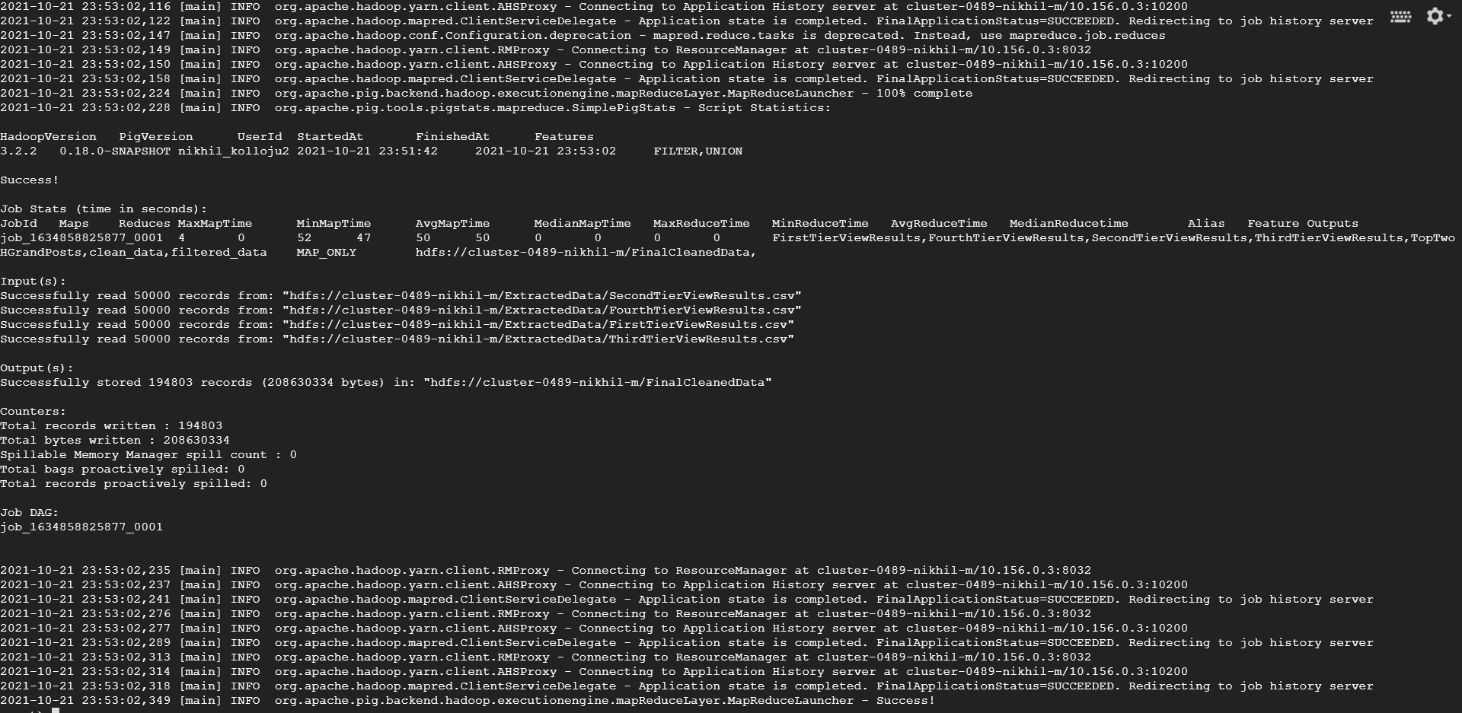
I created a Master-Worker Cluster in Google Cloud Platform and with the Master Node SSH, I was able to upload the csv files from my local machine to cluster directory using the settings > upload file option on the SSH window. Once the files are uploaded, I used the below commands to run in the CLI to move the files from my cluster to HDFS directory.



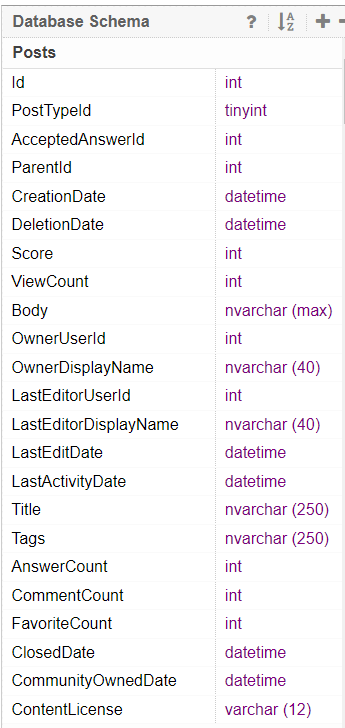
**3. Data Cleaning with Pig:**

Pig is available as one of the add-ons in the Hadoop network. I chose Pig to perform data cleaning because it is much simpler to do the aforementioned task without having to write complex Java programs with mapper and reducer classes.

After loading Pig in local mode in GCP Hadoop, I ran the below grunt commands like LOAD & CSVExcelStorage [a] to load the csv data into local variables and UNION to combine all the data into one variable for easier reference. I then filtered out all the records that didn’t have any OwnerUserId and Id. After cleaning[i] the Title, Body, Tags, OwnerDisplayName, OwnerUserId, ViewCount, Id and Score for unwanted characters, the cleaned data is then loaded into a file in HDFS using PigStorage.



## **4. Database and Table Creation with Hive:**

Hive is closely integrated with Hadoop and is available as an add-on in GCP Hadoop. Hive allows us to read, write and manage large amounts of data using HQL or Hive-QL which is similar to SQL. By using the below commands I was able to create a database and a table[e] to load the cleaned data from Pig.

CREATE DATABASE IF NOT EXISTS cloudtechdb;

CREATE TABLE cloudtechdb.top2gpoststb (Id int,PostTypeId int,AcceptedAnswerId int,ParentId int,CreationDate timestamp,DeletionDate timestamp,Score int,ViewCount int,Body string,OwnerUserId int,OwnerDisplayName string,LastEditorUserId int,LastEditorDisplayName string,LastEditDate timestamp,LastActivityDate timestamp,Title string,Tags string,AnswerCount int,CommentCount int,FavoriteCount int,ClosedDate timestamp,CommunityOwnedDate timestamp,ContentLicense string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

LOAD DATA INPATH 'hdfs://cluster-0489-nikhil-m/FinalCleanedData' INTO TABLE cloudtechdb.top2gpoststb;

## **5.Querying data for Task 3 with Hive:**

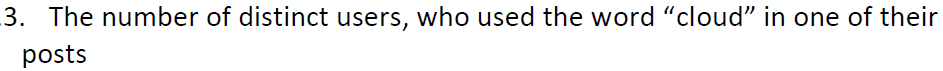
After the cleaned data is loaded into Hive table as above, the below queries are run to get the result to fetch the data.



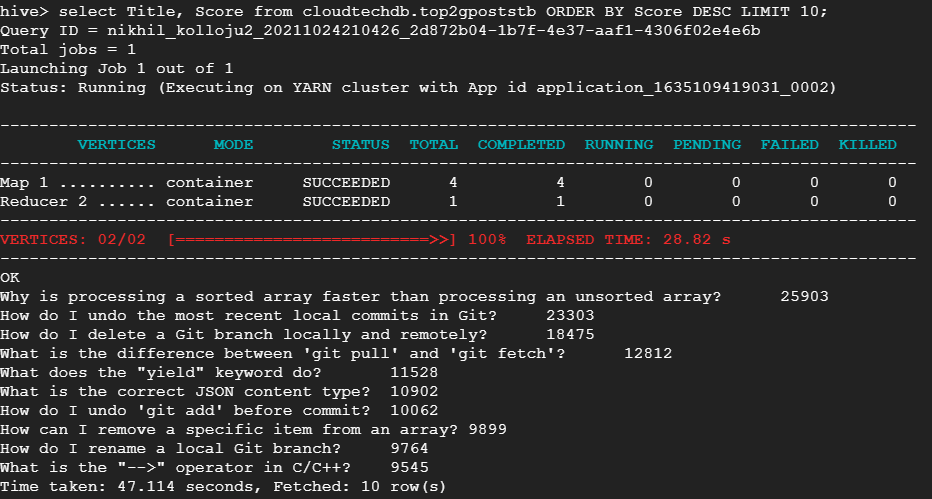
SELECT Title, Score AS Post\_Score FROM cloudtechdb.top2gpoststb ORDER BY Post\_Score DESC LIMIT 10;

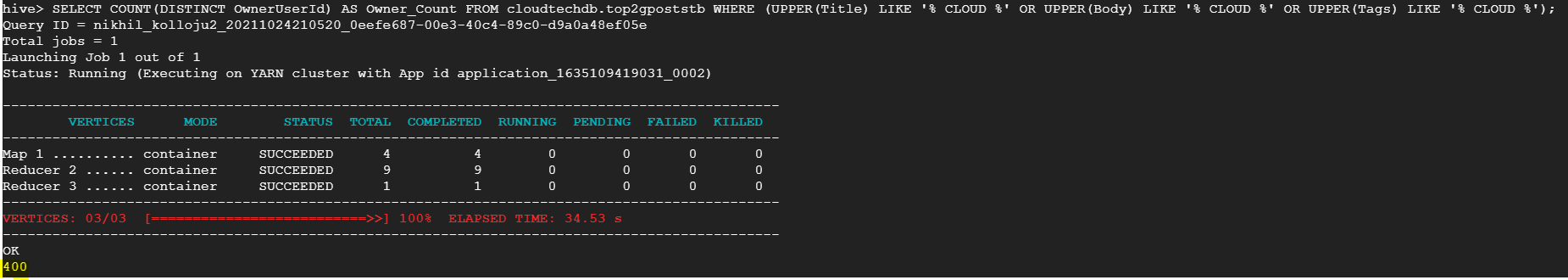
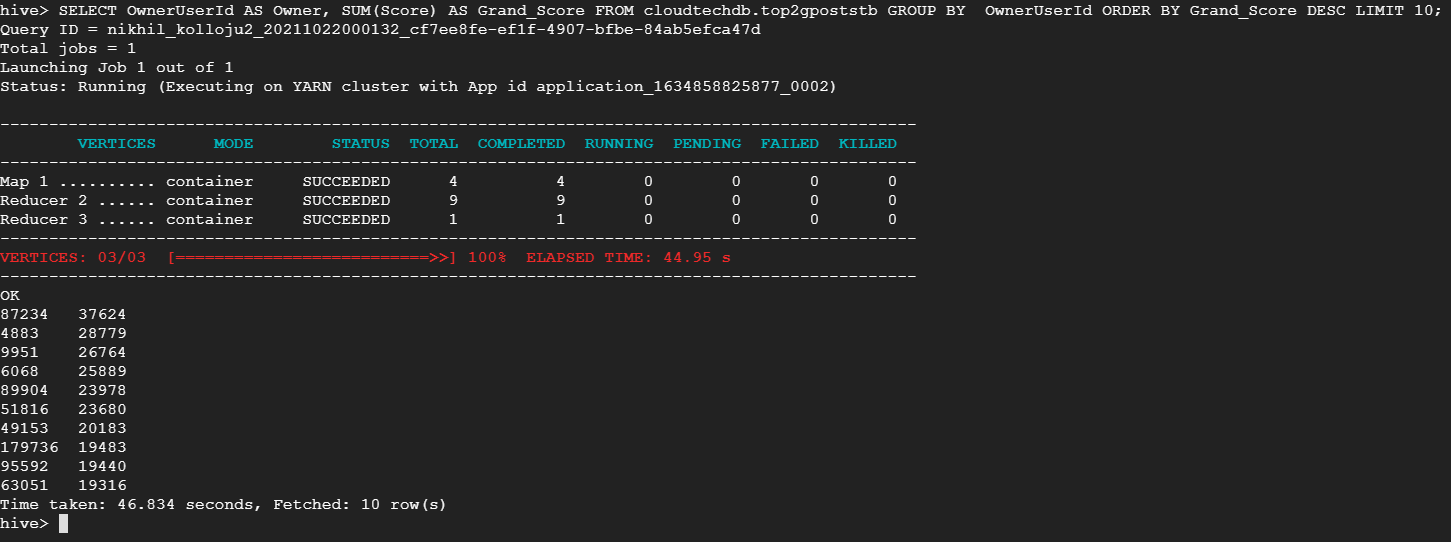


SELECT SUM(Score) AS Grand\_Post\_Score, OwnerUserId AS Owner FROM cloudtechdb.top2gpoststb GROUP BY OwnerUserId ORDER BY Grand\_Post\_Score DESC LIMIT 10;



SELECT COUNT ( DISTINCT OwnerUserId ) AS Owner\_Count FROM cloudtechdb.top2gpoststb WHERE (UPPER(Title) LIKE '% CLOUD %' OR UPPER(body) LIKE '% CLOUD %' OR UPPER(Tags) LIKE '% CLOUD %');





## **6. Calculating TF-IDF for Task 4 with Hive:**

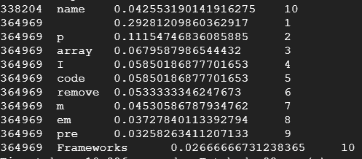
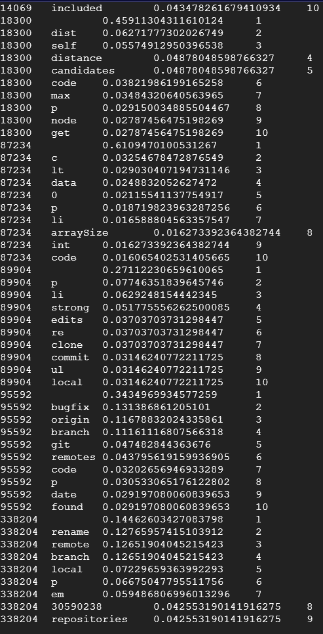
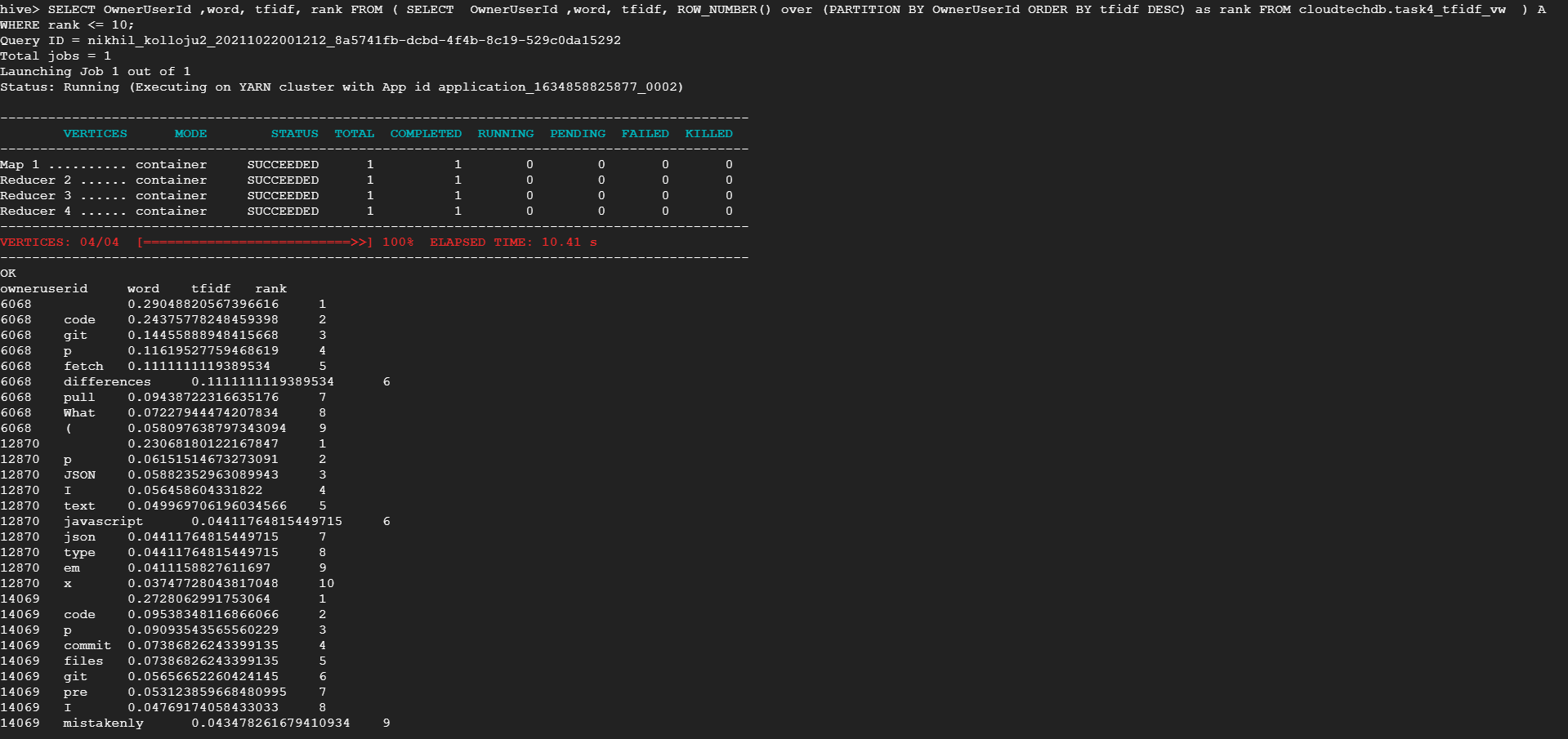
TF-IDF [b], which stands for term frequency — inverse document frequency and is intended to reflect how relevant a term is in a given document. A set of pre-requisites need to be in place before we can actually get to the task. First of which is to add Hivemall [c][d] as we will be using it to detach each word in the ‘Body’ column. Once I have downloaded the [hivemall jar file](https://search.maven.org/search?q=a:hivemall-all) and the [define-all.hive file](https://github.com/apache/incubator-hivemall/blob/master/resources/ddl/define-all.hive), I uploaded them using the same navigation Settings > Upload File to upload 2 files into GCP Master Cluster Node. I then ran the below commands to add hivemall to hive and set the source.

add jar /home/nikhil\_kolloju2/hivemall-all-0.6.0-incubating.jar;

source /home/nikhil\_kolloju2/define-all.hive;

Since the Task 4 asks to Calculate the per-user TF-IDF [h] of the top 10 terms for each of the top 10 users’, I first stored the top 10 users along with their scores and Body column in one table. I then created another table with just OwnerUserId and their scores. With this, I created a view which filters out the stop words and fetch us the Term Frequency. Then we calculate the document frequency and using both the views we finally get our TF-IDF view from which I ran the below query to fetch the Task-4 data.

SELECT A.OwnerUserId ,A.de\_word, A.wf\_tfidf, A.rank FROM ( SELECT OwnerUserId ,de\_word, wf\_tfidf, ROW\_NUMBER() over (PARTITION BY OwnerUserId ORDER BY wf\_tfidf DESC) as rank FROM cloudtechdb.task4\_tfidf\_vw ) A WHERE A.rank <= 10;



## **References:**

[a]CSVExcelStorage:

<https://pig.apache.org/docs/r0.17.0/api/org/apache/pig/piggybank/storage/CSVExcelStorage.html>

[b]TF-IDF:

<https://www.kdnuggets.com/2018/08/wtf-tf-idf.html>

[c]TF-IDF Term Weighting:

<https://hivemall.incubator.apache.org/userguide/ft_engineering/tfidf.html>.

[d]Hivemall User Manual:

<https://hivemall.incubator.apache.org/userguide/getting_started/installation.html>

[e]Website to download csv files from Stack Exchange and Create Table Schema:

<https://data.stackexchange.com/stackoverflow/query/new>

[f]To download and install hivemall and define-all.hive:

<https://search.maven.org/search?q=a:hivemall-all>

<https://github.com/apache/incubator-hivemall/blob/master/resources/ddl/define-all.hive>

[g]ROW\_NUMBER():

<https://www.sqlservertutorial.net/sql-server-window-functions/sql-server-row_number-function/>

[h]TF-IDF Calculation:

<https://github.com/daijyc/hivemall/wiki/TFIDF-calculation>

[i] Data Clean using Pig:

<https://pig.apache.org/docs/r0.17.0/basic.html>

Note:

All the code and screenshots have been uploaded into the github repository mentioned in the beginning.