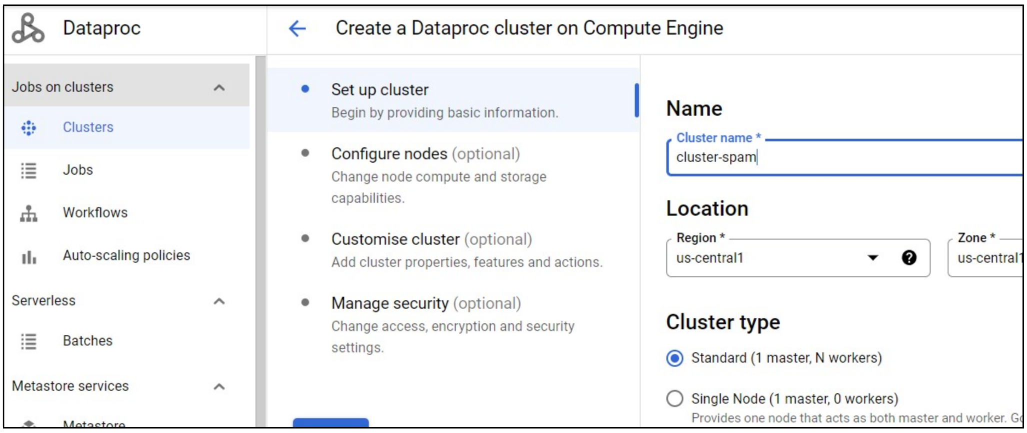
Spam Detection System in Hadoop

1) Cloud Infrastructure Setup

a. Create a Hadoop cluster using GCP (Google Cloud Platform) Dataproc.

I have chosen my cloud platform as GCP and I have created a cluster on it using Dataproc Compute Engine. The cluster is set to have 1 master and 3 worker nodes with the region selected as us-central1 and the zone as us-central1-f.



Create a Dataproc cluster on Compute Engine

- Set up cluster
  Begin by providing basic information.
- Configure nodes (optional)
  Change node compute and storage capabilities.
- Customise cluster (optional)
  Add cluster properties, features and actions.
- Manage security (optional)
  Change access, encryption and security settings.

Name

Cluster name * cluster-spam

Location

Region * us-central1

Zone * us-central1

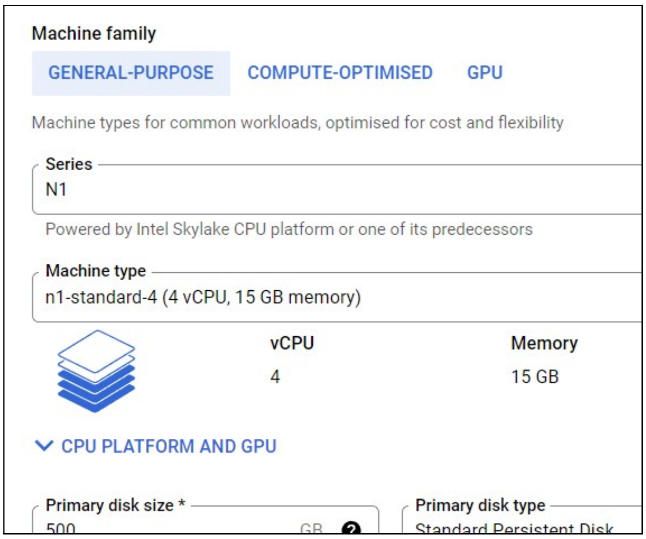
Cluster type

● Standard (1 master, N workers)

○ Single Node (1 master, 0 workers)
Provides one node that acts as both master and worker. G

Figure 1: Dataproc with 1master, N-worker

To configure my nodes, I selected the machine from the family of General Purpose of N1 series and machine type as ‘n1-standard-4 (4 vCPU, 15 GB memory)’. I have taken the same configuration for my worker nodes as well.



Machine family: GENERAL-PURPOSE

Machine types for common workloads, optimised for cost and flexibility

Series: N1

Powered by Intel Skylake CPU platform or one of its predecessors

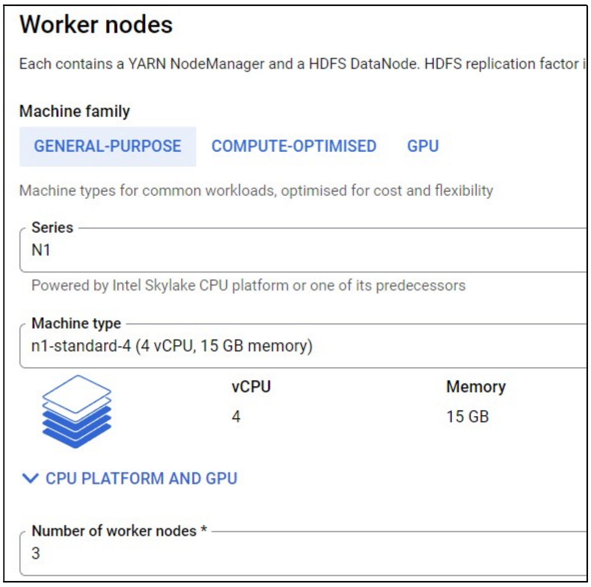
Machine type: n1-standard-4 (4 vCPU, 15 GB memory)

vCPU	Memory
4	15 GB

Primary disk size * 500 GB

Primary disk type: Standard Persistent Disk

Figure 2: Master node



Worker nodes

Each contains a YARN NodeManager and a HDFS DataNode. HDFS replication factor i

Machine family

Machine types for common workloads, optimised for cost and flexibility

Series: N1

Powered by Intel Skylake CPU platform or one of its predecessors

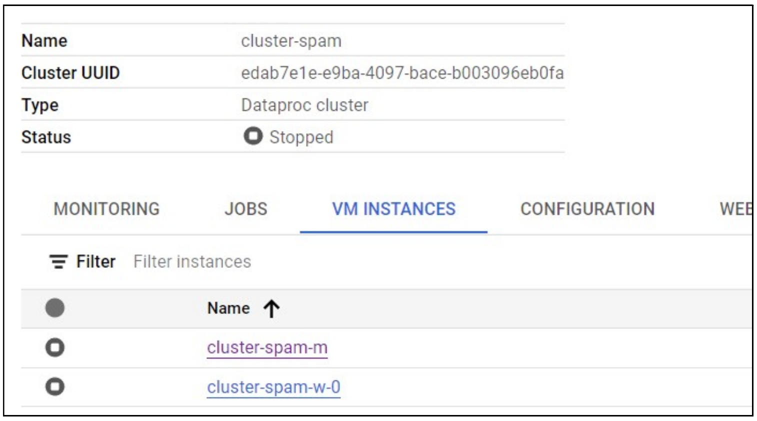
Machine type: n1-standard-4 (4 vCPU, 15 GB memory)

vCPU	Memory
4	15 GB

Number of worker nodes *: 3

Figure 3: Worker Node

After the cluster has been created, we can verify the same from the Dataproc VM instances of our cluster.



Name	cluster-spam
Cluster UUID	edab7e1e-e9ba-4097-bace-b003096eb0fa
Type	Dataproc cluster
Status	○ Stopped

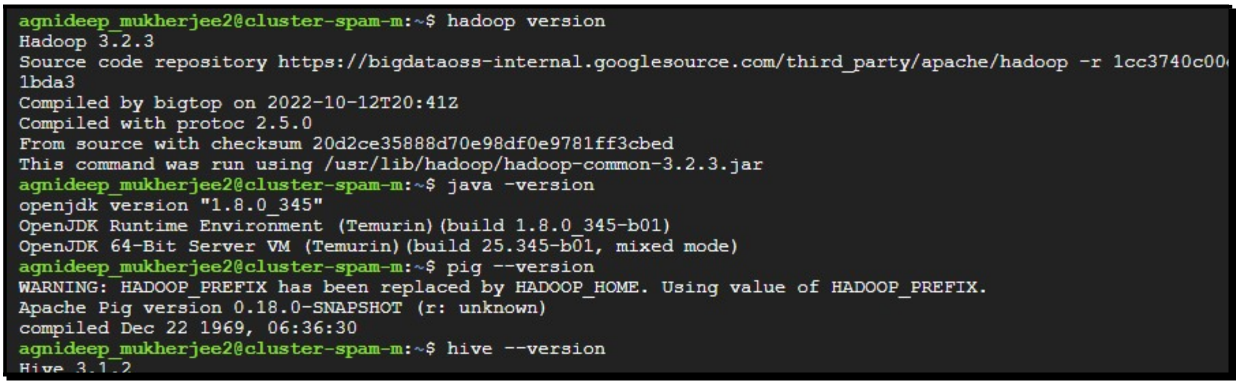
●	Name ↑
○	cluster-spam-m
○	cluster-spam-w-0

Figure 4: Cluster with 1 master, 3 worker

b. Environment check at the cluster.

In the dataproc cluster, Hadoop, pig, hive, and java was successfully installed. We have also created a directory of my user name under the '/user/' with the command: hdfs dfs -mkdir /<user_name>.

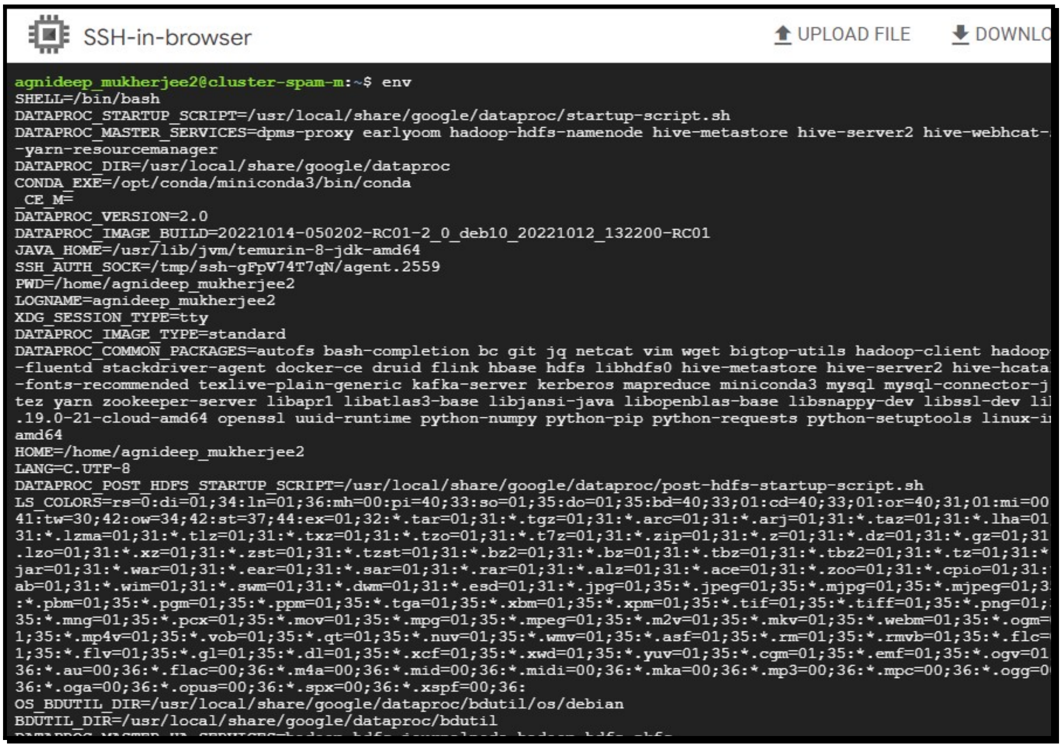
We have also checked the version of the same to verify whether the installations have been properly done.



```
agnideep_mukherjee2@cluster-spam-m:~$ hadoop version
Hadoop 3.2.3
Source code repository https://bigdataoss-internal.googlesource.com/third_party/apache/hadoop -r 1cc3740c00
1bda3
Compiled by bigtop on 2022-10-12T20:41Z
Compiled with protoc 2.5.0
From source with checksum 20d2ce35888d70e98df0e9781ff3cbed
This command was run using /usr/lib/hadoop/hadoop-common-3.2.3.jar
agnideep_mukherjee2@cluster-spam-m:~$ java -version
openjdk version "1.8.0_345"
OpenJDK Runtime Environment (Temurin)(build 1.8.0_345-b01)
OpenJDK 64-Bit Server VM (Temurin)(build 25.345-b01, mixed mode)
agnideep_mukherjee2@cluster-spam-m:~$ pig --version
WARNING: HADOOP_PREFIX has been replaced by HADOOP_HOME. Using value of HADOOP_PREFIX.
Apache Pig version 0.18.0-SNAPSHOT (r: unknown)
compiled Dec 22 1969, 06:36:30
agnideep_mukherjee2@cluster-spam-m:~$ hive --version
Hive 3.1.2
```

Figure 5: Version check for Hadoop, Java, Pig, Hive

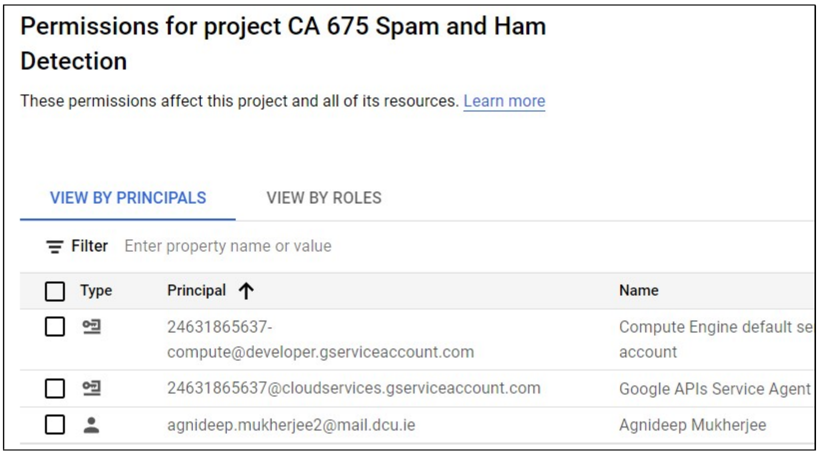
I have checked whether the cluster has been set up properly by using the '~$ env' command.



```
agnideep_mukherjee2@cluster-spam-m:~$ env
SHELL=/bin/bash
DATAPROC_STARTUP_SCRIPT=/usr/local/share/google/dataproc/startup-script.sh
DATAPROC_MASTER_SERVICES=dpms-proxy earlyoom hadoop-hdfs-namenode hive-metastore hive-server2 hive-webhcat-
-yarn-resourcemanager
DATAPROC_DIR=/usr/local/share/google/dataproc
CONDA_EXE=/opt/conda/miniconda3/bin/conda
_CE_M=
DATAPROC_VERSION=2.0
DATAPROC_IMAGE_BUILD=20221014-050202-RC01-2_0_deb10_20221012_132200-RC01
JAVA_HOME=/usr/lib/jvm/temurin-8-jdk-amd64
SSH_AUTH_SOCK=/tmp/ssh-gFpV74T7qN/agent.2559
PWD=/home/agnideep_mukherjee2
LOGNAME=agnideep_mukherjee2
XDG_SESSION_TYPE=tty
DATAPROC_IMAGE_TYPE=standard
DATAPROC_COMMON_PACKAGES=autofs bash-completion bc git jq netcat vim wget bigtop-utils hadoop-client hadoop
-fluentd stackdriver-agent docker-ce druid flink hbase hdfs libhdfs0 hive-metastore hive-server2 hive-hcata
-fonts-recommended texlive-plain-generic kafka-server kerberos mapreduce miniconda3 mysql mysql-connector-j
tez yarn zookeeper-server libapr1 libatlas3-base libjansi-java libopenblas-base libsnappy-dev libssl-dev li
.19.0-21-cloud-amd64 openssl uuid-runtime python-numpy python-pip python-requests python-setuptools linux-i
amd64
HOME=/home/agnideep_mukherjee2
LANG=C.UTF-8
DATAPROC_POST_HDFS_STARTUP_SCRIPT=/usr/local/share/google/dataproc/post-hdfs-startup-script.sh
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00
41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01
31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31
.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*
jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:
ab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;3
:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;
35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=
1;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=
1;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01
36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=0
36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
OS_BDUTIL_DIR=/usr/local/share/google/dataproc/bdutil/os/debian
BDUTIL_DIR=/usr/local/share/google/dataproc/bdutil
```

Figure 6: Environment check

I have also provided access to professor Manoj on my cloud system, such that my project can by verified.



Permissions for project CA 675 Spam and Ham Detection

These permissions affect this project and all of its resources. Learn more

	Type	Principal ↑	Name
□		24631865637-compute@developer.gserviceaccount.com	Compute Engine default se account
□		24631865637@cloudservices.gserviceaccount.com	Google APIs Service Agent
□		agnideep.mukherjee2@mail.dcu.ie	Agnideep Mukherjee

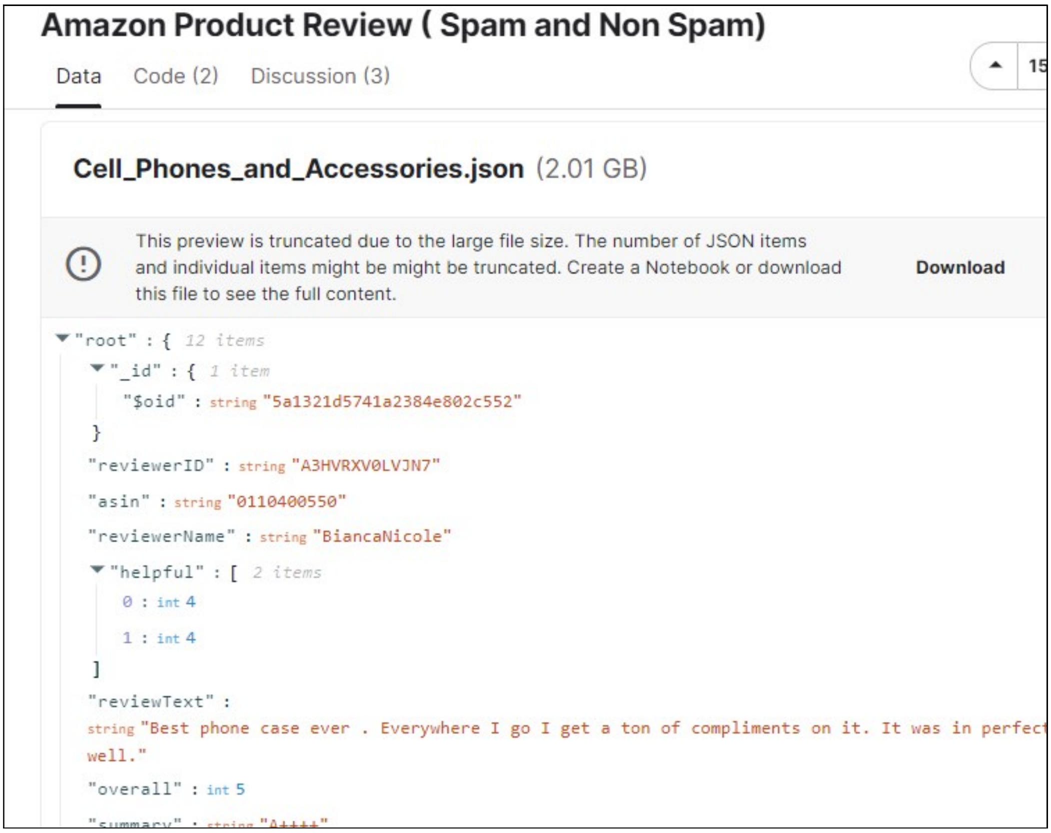
Figure 7: Access control

2) Choosing the dataset.

a. Data from Kaggle.com

Link: https://www.kaggle.com/datasets/naveedhn/amazon-product-review-spam-and-non-spam

I have chosen a fairly large and complex dataset from Kaggle.com about the spam and ham reviews that different user on Amazon provides under the 'Cell phone and accessories' category.



Amazon Product Review (Spam and Non Spam)

Cell_Phones_and_Accessories.json (2.01 GB)

This preview is truncated due to the large file size. The number of JSON items and individual items might be might be truncated. Create a Notebook or download this file to see the full content.

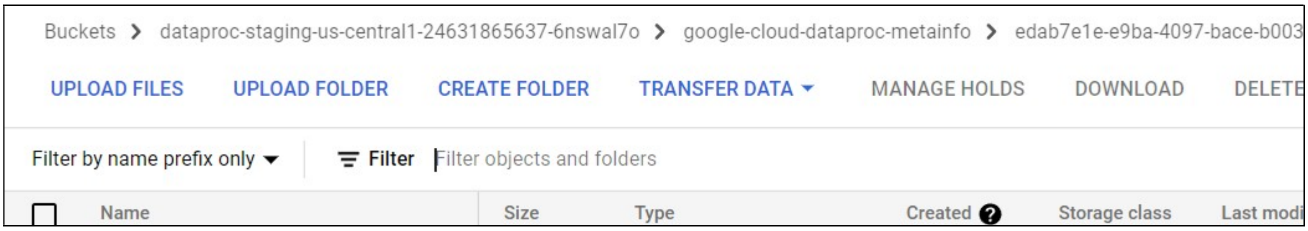
```
▼"root" : { 12 items
  ▼"_id" : { 1 item
     "$oid" : string "5a1321d5741a2384e802c552"
  }
  "reviewerID" : string "A3HVRXV0LVJN7"
  "asin" : string "0110400550"
  "reviewerName" : string "BiancaNicole"
  ▼"helpful" : [ 2 items
     0 : int 4
     1 : int 4
  ]
  "reviewText" :
  string "Best phone case ever . Everywhere I go I get a ton of compliments on it. It was in perfect
  well."
  "overall" : int 5
  "summary" : string "A++++"
```

Figure 8: Dataset preview

The dataset is of 1.87 Gb in size and of type .JSON and has 12 attributes which makes it more complex, and interesting to apply analytics on.

b. Loading the data in the GCP bucket

I have loaded the data in the GCP bucket with the gsutil path as "*gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0fa/cluster-spam-m/Cell_Phones_and_Accessories.json*"



Buckets > dataproc-staging-us-central1-24631865637-6nswal7o > google-cloud-dataproc-metainfo > edab7e1e-e9ba-4097-bace-b003

Figure 9: Dataset uploaded in Google Bucket

3) Cleaning the data

The data has been cleaned has been performed with the following commands in Apache Pig.

```
>>cleanedtable1= FOREACH tableframe GENERATE id AS id, reviewerID AS reviewerID, reviewerName AS reviewerName,
REPLACE(reviewText,'([^a-zA-Z\\s]+)','') AS reviewText, class AS class;
```

```
>> cleanedtable1_5= FOREACH tableframe GENERATE id AS id, reviewerID AS reviewerID, reviewerName AS reviewerName,
REPLACE(reviewText, ‘[\r\n]+',' ') AS reviewText, class AS class;

>>cleanedtable2= FOREACH cleanedtable1_5 GENERATE id AS id, reviewerID AS reviewerID, reviewerName AS reviewerName,
LOWER(reviewText) AS reviewText, class AS class;

>>cleanedtable3 = FILTER cleanedtable2 BY reviewerName != 'Amazon Customer' AND reviewerName != 'No Name';

>>cleanedtable4 = FILTER cleanedtable3 BY reviewerName is not null AND reviewerID is not null;

>>cleanedtable5 = FILTER cleanedtable4 by reviewText is not null;

>>cleanedtable = FILTER cleanedtable5 by class==0 OR class==1;

>>STORE cleanedtable INTO
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/cleanedtable.json' USING JsonStorage();
```

In the first cleanedtable1 I used Regular Expression (Regex) to remove the special characters and numbers present in the review text. Also, I have restricted the unnecessary attributes from the original table, which made the new data more concise and useful for analysis.

In the cleanedtable1_5 I took the help of Regular Expression again to replace the new lines with spaces.

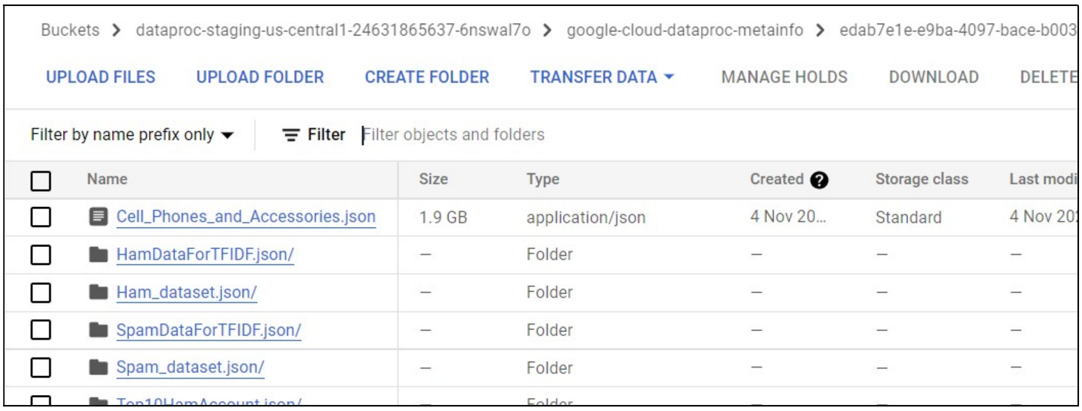
In cleanedtable2 I have loaded the review text by lowercasing the entire review text such that my analysis will be irrespective of the cases on the same word.

I have verified in cleanedtable3 that there are a fairly large amount of reviewers as ‘Amazon Customer’ and ‘No Name’, perhaps because there have been comments from Guest Accounts, so I am removing them to perform a better analysis.

In cleanedtable4 and cleanedtable5 I have removed all the rows where the reviewer name, id and text are null.

Finally since the ‘class’ is the attribute which classifies if the data is spam or ham, I have taken only those rows where the data has been already classified.

At the last I have stored the cleaned table in the cloud bucket such that when we go further on the analysis we will not need to perform the cleaning over and over again, we can directly load the data in the pig using the JsonLoader();



Buckets > dataproc-staging-us-central1-24631865637-6nswal7o > google-cloud-dataproc-metainfo > edab7e1e-e9ba-4097-bace-b003
UPLOAD FILES
UPLOAD FOLDER
CREATE FOLDER
TRANSFER DATA
MANAGE HOLDS
DOWNLOAD
DELETE
Filter by name prefix only
Filter
Filter objects and folders

□	Name	Size	Type	Created	Storage class	Last modi
□	Cell_Phones_and_Accessories.json	1.9 GB	application/json	4 Nov 20…	Standard	4 Nov 20
□	HamDataForTFIDF.json/	–	Folder	–	–	–
□	Ham_dataset.json/	–	Folder	–	–	–
□	SpamDataForTFIDF.json/	–	Folder	–	–	–
□	Spam_dataset.json/	–	Folder	–	–	–
□	Top10HamAccount.json/	–	Folder	–	–	–

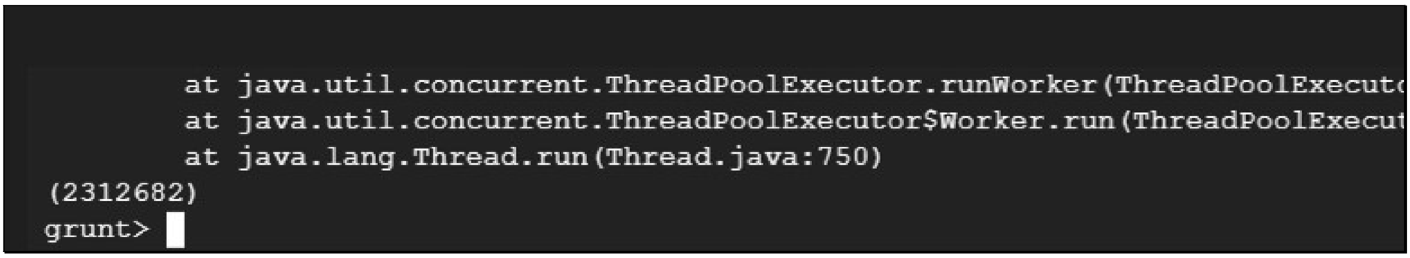
Figure 10: Cleaned Table uploaded in Google Bucket

4) Ham or Spam using Apache Pig

a. Differentiate Ham and Spam Dataset

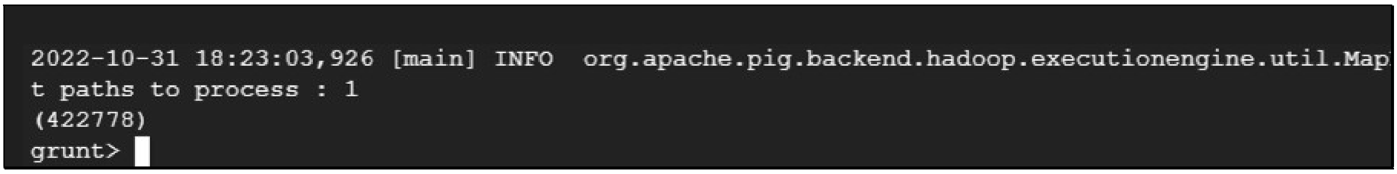
```
>>filter_spam = FILTER cleanedtable BY class == 1;
>>spamgroup= GROUP filter_spam ALL;
>>spamcount= FOREACH spamgroup GENERATE COUNT (filter_spam.reviewerID);
>>dump spamcount;

>>filter_ham = FILTER cleanedtable BY class == 0;
>>hamgroup= GROUP filter_ham ALL;
>>hamcount= FOREACH hamgroup GENERATE COUNT (filter_ham.reviewerID);
>>dump hamcount;


>>STORE filter_spam INTO
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/Spam_dataset.json' USING JsonStorage();

>>STORE filter_ham INTO
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/Ham_dataset.json' USING JsonStorage();
```

I have split the spam and the ham dataset in Apache pig using the ‘class’ attribute present in the original dataset, which represents whether a data is a spam or ham. After the split, I stored them separately in two variables filter_spam and filter_ham variable. To check whether the split has been performed correctly I have grouped the dataset and with foreach loop I have taken the count of the ‘reviewerID’, which is unique.



```
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecut
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecut
        at java.lang.Thread.run(Thread.java:750)
(2312682)
grunt>
```

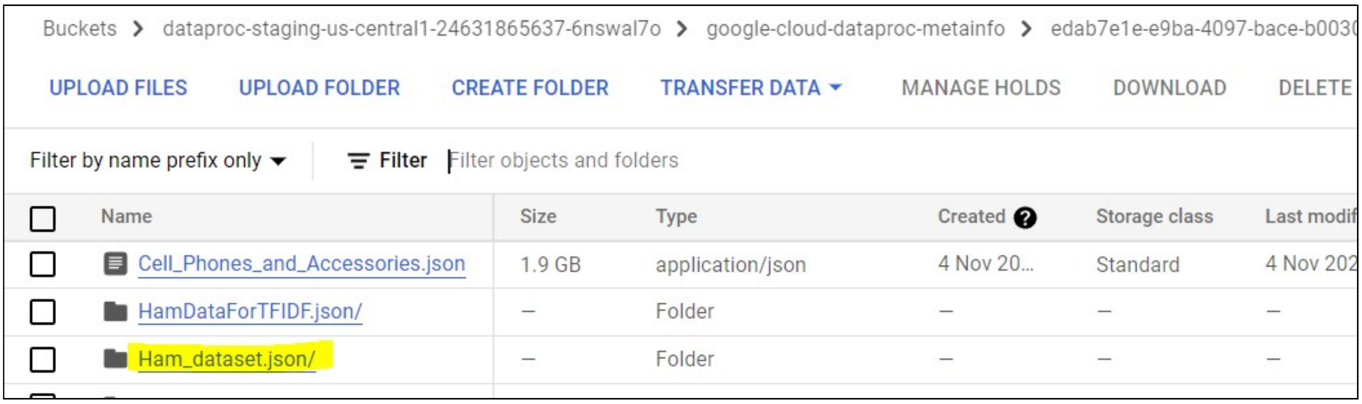
Figure 11: Total count of Spam Data



```
2022-10-31 18:23:03,926 [main] INFO  org.apache.pig.backend.hadoop.executionengine.util.Map
t paths to process : 1
(422778)
grunt>
```

Figure 12: Total count of Ham Data

After that, I stored the 2 different datasets in the google cloud bucket, such that the data remains ready even after the instances are closed.


Buckets > dataproc-staging-us-central1-24631865637-6nswal7o > google-cloud-dataproc-metainfo > edab7e1e-e9ba-4097-bace-b003
UPLOAD FILES
UPLOAD FOLDER
CREATE FOLDER
TRANSFER DATA ▾
MANAGE HOLDS
DOWNLOAD
DELETE
Filter by name prefix only ▾
Filter
Filter objects and folders

□	Name	Size	Type	Created	Storage class	Last modif
□	Cell_Phones_and_Accessories.json	1.9 GB	application/json	4 Nov 20...	Standard	4 Nov 202
□	HamDataForTFIDF.json/	–	Folder	–	–	–
□	Ham_dataset.json/	–	Folder	–	–	–

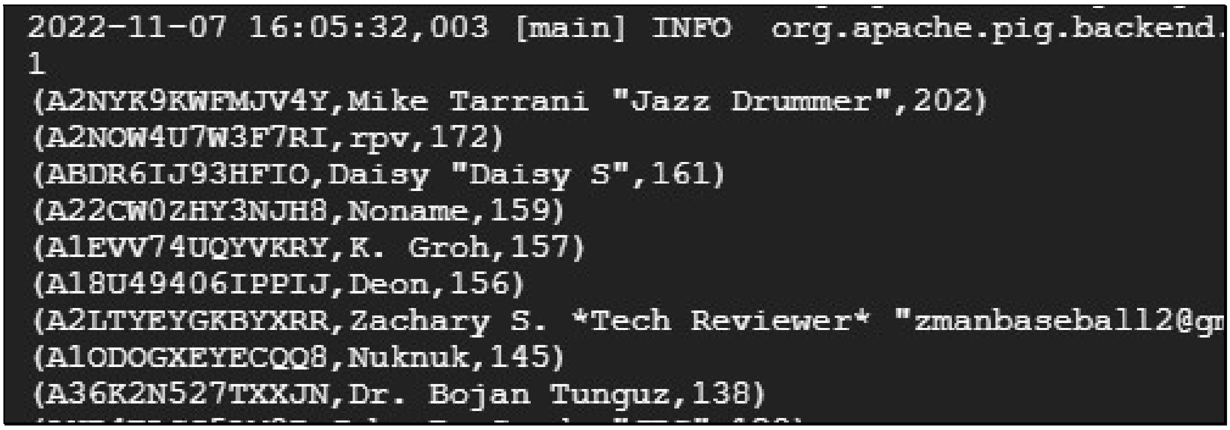
Figure 13: Ham and Spam Dataset stored in Google Bucket

b. Top 10 Spam Account

I have written the following code in Apache pig to find the top 10 Spam Account.

```
>>spamtest = LOAD
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/Spam_dataset.json' USING JsonLoader('id:(oid:chararray), reviewerID:chararray, reviewerName:chararray,
reviewText:chararray, summary:chararray, class:int');

>>spamReviewerGroup = GROUP spamtest by reviewerID;
>>spamReviewerCount = FOREACH spamReviewerGroup GENERATE FLATTEN(group) as reviewerID, COUNT($1) as count;
>>spamReviewerOrdered= ORDER spamReviewerCount by count DESC;
>>top10SpamReviewer= LIMIT spamReviewerOrdered 10;
>>SpamTotalData1= JOIN top10SpamReviewer by reviewerID, spamtest by reviewerID;
>>SpamNameAndId= FOREACH SpamTotalData1 GENERATE top10SpamReviewer::reviewerID as reviewerID, spamtest::reviewerName
as reviewerName, top10SpamReviewer::count as count;
Dump SpamNameAndId;
>>SpamNameAndId1= DISTINCT SpamNameAndId;
>>Dump SpamNameAndId1;

>>STORE SpamNameAndId1 INTO
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/Top10SpamAccount.json' USING JsonStorage();
```

At first, I loaded the spam data from the google cloud bucket, which we had previously saved. Furthermore, I created a new table frame by grouping the spam table with the reviewerId and adding the count of the rows in another column mentioned as 'count'.
I have used 'FLATTEN()' operartor to un-nest the tuples which got created when I grouped spamtest by the reviewerId.
Following that I have sorted the table by descending order of the count because my aim is to find the Top 10 spam accounts and stored them in 'spamReviewerOrdered'.
Now, the challenge is the table frame, spamReviewerOrdered does not have the name of the reviewer, it only possesses the reviewerId and the count. So, in order to overcome this, I have joined the table frame top10SpamReviewer and spamtest with reviewerId as the pivot.
After the join is successfully done, with the FOREACH loop I have iterated through the table and picked only those attributes which are important to me i.e Reviewer ID, Reviewer Name and Count.
In the variable SpamNameAndId1, we have taken only the distinct values and printed in the console.



```
2022-11-07 16:05:32,003 [main] INFO  org.apache.pig.backend.
1
(A2NYK9KWFMJV4Y,Mike Tarrani "Jazz Drummer",202)
(A2NOW4U7W3F7RI,rpv,172)
(ABDR6IJ93HFIO,Daisy "Daisy S",161)
(A22CW0ZHY3NJH8,Noname,159)
(A1EVV74UQYVKRY,K. Groh,157)
(A18U49406IPPIJ,Deon,156)
(A2LTYEYGKBYXRR,Zachary S. *Tech Reviewer* "zmanbaseball2@gm
(A1ODOGXEYECQQ8,Nuknuk,145)
(A36K2N527TXXJN,Dr. Bojan Tunguz,138)
```

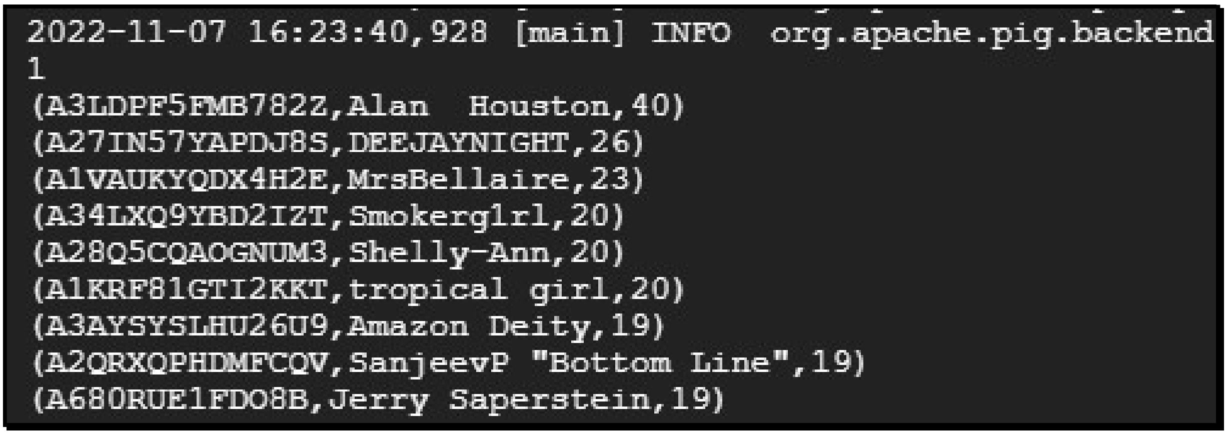
Figure 14: Top 10 Spam Accounts

a. Top 10 Ham Account

I have performed the steps with the same logic which I had previously performed while calculating the Top 10 Spam accounts.

The code is as follows.

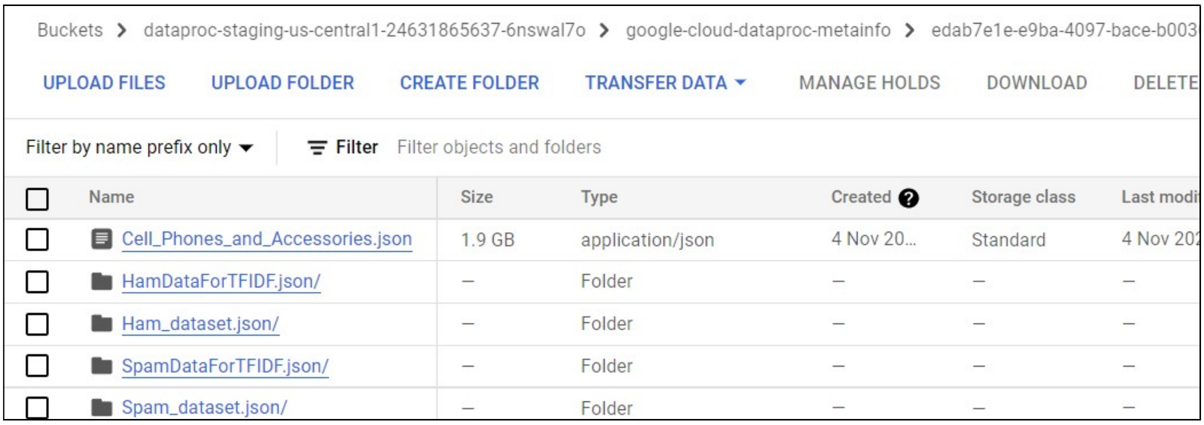
```
>>hamtest = LOAD 'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0fa/cluster-ham-m/ham_dataset.json' USING JsonLoader('id:(oid:chararray), reviewerID:chararray, reviewerName:chararray, reviewText:chararray, summary:chararray, class:int');

>>hamReviewerGroup = GROUP hamtest by reviewerID;

>>hamReviewerCount = FOREACH hamReviewerGroup GENERATE FLATTEN(group) as reviewerID, COUNT($1) as count;

>>hamReviewerOrdered= ORDER hamReviewerCount by count DESC;

>>top10hamReviewer= LIMIT hamReviewerOrdered 10;

>>hamTotalData1= JOIN top10hamReviewer by reviewerID, hamtest by reviewerID;

>>hamNameAndId= FOREACH hamTotalData1 GENERATE top10hamReviewer::reviewerID as reviewerID, hamtest::reviewerName as reviewerName, top10hamReviewer::count as count;

>>hamNameAndId1= DISTINCT hamNameAndId;


>>STORE hamNameAndId1 INTO 'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0fa/cluster-spam-m/Top10hamAccount.json' USING JsonStorage();
```



```
2022-11-07 16:23:40,928 [main] INFO  org.apache.pig.backend
1
(A3LDPF5FMB782Z,Alan  Houston,40)
(A27IN57YAPDJ8S,DEEJAYNIGHT,26)
(A1VAUKYQDX4H2E,MrsBellaire,23)
(A34LXQ9YBD2IZT,Smokerg1rl,20)
(A28Q5CQAOGNUM3,Shelly-Ann,20)
(A1KRF81GTI2KKT,tropical girl,20)
(A3AYSYSLHU26U9,Amazon Deity,19)
(A2QRXQPHDMFCQV,SanjeevP "Bottom Line",19)
(A680RUE1FDO8B,Jerry Saperstein,19)
```

Figure 15: Top 10 Ham Accounts

I have saved both the top 10 Spam and Ham accounts in the google cloud bucket using JsonLoader()



Buckets > dataproc-staging-us-central1-24631865637-6nswal7o > google-cloud-dataproc-metainfo > edab7e1e-e9ba-4097-bace-b003
UPLOAD FILES
UPLOAD FOLDER
CREATE FOLDER
TRANSFER DATA ▾
MANAGE HOLDS
DOWNLOAD
DELETE
Filter by name prefix only ▾
Filter
Filter objects and folders

☐	Name	Size	Type	Created	Storage class	Last modi
☐	Cell_Phones_and_Accessories.json	1.9 GB	application/json	4 Nov 20...	Standard	4 Nov 202
☐	HamDataForTFIDF.json/	–	Folder	–	–	–
☐	Ham_dataset.json/	–	Folder	–	–	–
☐	SpamDataForTFIDF.json/	–	Folder	–	–	–
☐	Spam_dataset.json/	–	Folder	–	–	–

Figure 16: Top 10 Ham and Spam Accounts stored in Google Cloud Bucket

5) TF-IDF using Map Reduce

a. Top 10 Spam keywords for top 10 spam accounts.

The statistical technique known as Term Frequency - Inverse Document Frequency (TF-IDF) is frequently employed in information retrieval and natural language processing. It gauges a term's significance within a document in relation to a group of papers (i.e., relative to a corpus).

Formula-

- Tf=(Total number of times the term appears in a document/total number of terms in the document)
- Idf= Log(number of the document in the corpus/number of documents in the corpus containing the term)
- TfIdf= Tf*Idf

i. Data Preparation

In the data preparation step, we have taken the help of Apache Pig to load the spam data and find the top 10 spam reviewer (the logic has been previously explained when we calculated the top 10 spam reviewers).

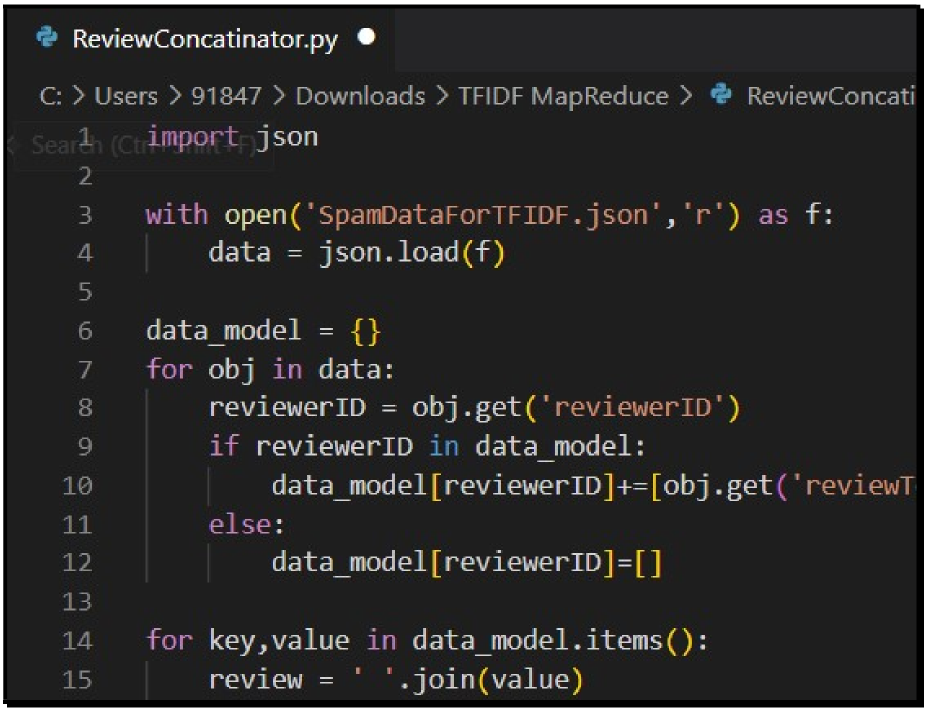
In the variable SpamTotalData1 we have joined top10SpamReviewer and spamtest by reviewerId. After that from the resultant variable we have iterated the through the rows and only picked the attributes I require for the TF-IDF process i.e the reviewer name, reviewer id and the text and saved it the cloud storage.

So, the data which we possess right now has all the text comments by the top the spam accounts.

```
>>spamtest = LOAD
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/Spam_dataset.json' USING JsonLoader('id:(oid:chararray), reviewerID:chararray, reviewerName:chararray,
reviewText:chararray, summary:chararray, class:int');


>>spamReviewerGroup = GROUP spamtest by reviewerID;
>>spamReviewerCount = FOREACH spamReviewerGroup GENERATE FLATTEN(group) as reviewerID, COUNT($1) as count;
>>spamReviewerOrdered= ORDER spamReviewerCount by count DESC;
>>top10SpamReviewer= LIMIT spamReviewerOrdered 10;
>>SpamTotalData1= JOIN top10SpamReviewer by reviewerID, spamtest by reviewerID;
>>SpamTotalData= FOREACH SpamTotalData1 GENERATE top10SpamReviewer::reviewerID as reviewerID, spamtest::reviewerName as
reviewerName, spamtest::reviewText as reviewText;

>>STORE SpamTotalData INTO
'gs://dataproc-staging-us-central1-24631865637-6nswal7o/google-cloud-dataproc-metainfo/edab7e1e-e9ba-4097-bace-b003096eb0
fa/cluster-spam-m/SpamDataForTFIDF.json' USING JsonStorage();
```

The next challenge is to concatenate all the rows of different comments and store it in a text file which will be required while performing TFIDF on top 10 spam accounts.

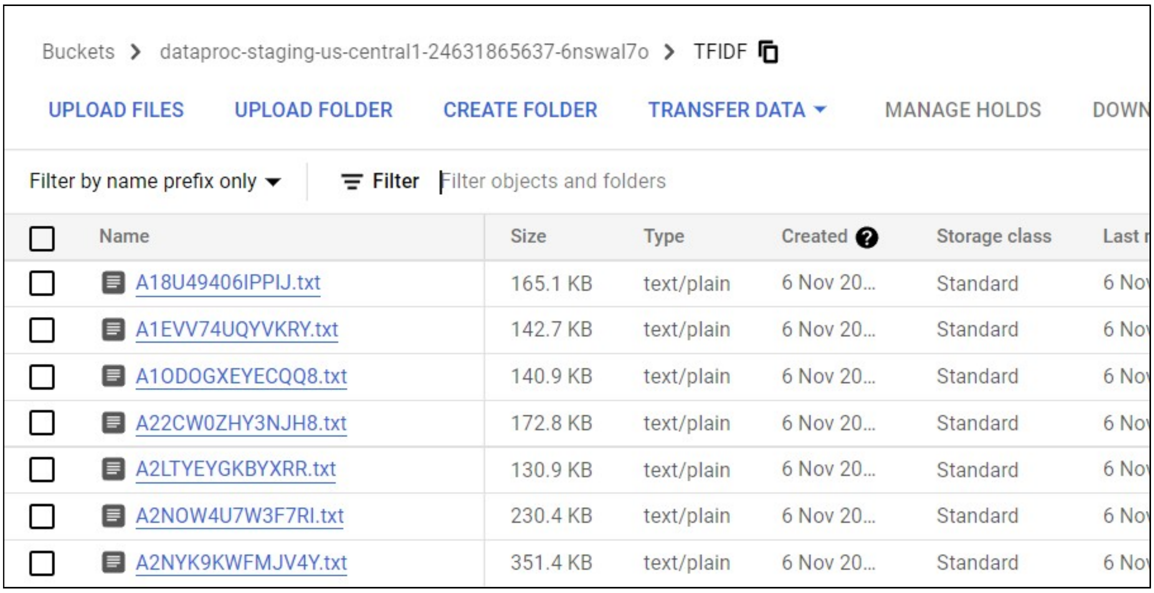
To overcome this, we have taken the aid of python to concatenate the data of each row and create a text file for each reviewer. We have loaded the json file which we had stored in cloud: SpamDataForTFIDF in our python code. Using data model we have joined the reviewText for each reviewer and have written the data in text file.


ReviewConcatinator.py
C: > Users > 91847 > Downloads > TFIDF MapReduce > ReviewConcati

```python
import json

with open('SpamDataForTFIDF.json','r') as f:
    data = json.load(f)

data_model = {}
for obj in data:
    reviewerID = obj.get('reviewerID')
    if reviewerID in data_model:
        data_model[reviewerID]+=[obj.get('reviewT
    else:
        data_model[reviewerID]=[]

for key,value in data_model.items():
    review = ' '.join(value)
```

Figure 17: Python code to merge all texts for each reviewer

After this code has generated the 10 text files, I have stored them in the cloud bucket, such that they can be used further by the TF-IDF map reduce code. Now we have the all the data required to perform the further steps.


Buckets > dataproc-staging-us-central1-24631865637-6nswal7o > TFIDF
UPLOAD FILES
UPLOAD FOLDER
CREATE FOLDER
TRANSFER DATA
MANAGE HOLDS
DOWN
Filter by name prefix only
Filter
Filter objects and folders

☐	Name	Size	Type	Created	Storage class	Last r
☐	A18U49406IPPIJ.txt	165.1 KB	text/plain	6 Nov 20...	Standard	6 No
☐	A1EVV74UQYVKRY.txt	142.7 KB	text/plain	6 Nov 20...	Standard	6 No
☐	A1ODOGXEYECQQ8.txt	140.9 KB	text/plain	6 Nov 20...	Standard	6 No
☐	A22CW0ZHY3NJH8.txt	172.8 KB	text/plain	6 Nov 20...	Standard	6 No
☐	A2LTYEYGKBYXRR.txt	130.9 KB	text/plain	6 Nov 20...	Standard	6 No
☐	A2NOW4U7W3F7RI.txt	230.4 KB	text/plain	6 Nov 20...	Standard	6 No
☐	A2NYK9KWFMJV4Y.txt	351.4 KB	text/plain	6 Nov 20...	Standard	6 No

Figure 18: Text Files uploaded in Google Cloud Bucket

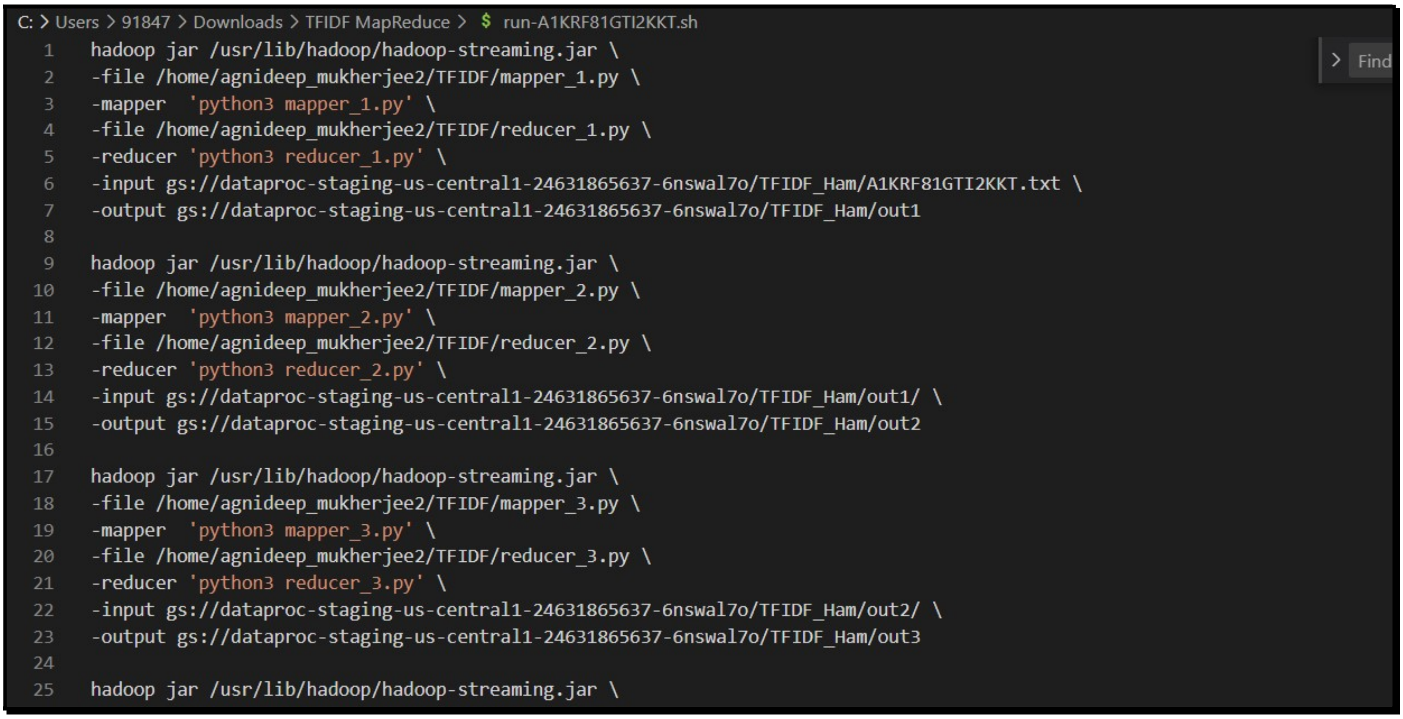
ii. Creating Mappers and Reducers

Python scripts are used in Hadoop to implement TFIDF. There are four mapper program files and three reduction program files in all. There are four stages to the implementation process. Three mappers and three reducers are used in the first phase. In the initial phase we remove the stopwords from the text and clean it further, following that we calculate the word count and the TF. The final phase calculates the TF-IDF and creates a single file containing the spam or ham word lists of 10 users and their TF-IDF scores using the fourth mapper.

One phase's output is used as an input for the following phase. Each text file is provided to the MapReduce.sh script as an input.

This method is applied to each of the ten text files, and the resulting text file for each of the ten text files gets merged at '/home/agnideep_mukherjee2/TFIDF/results/'.

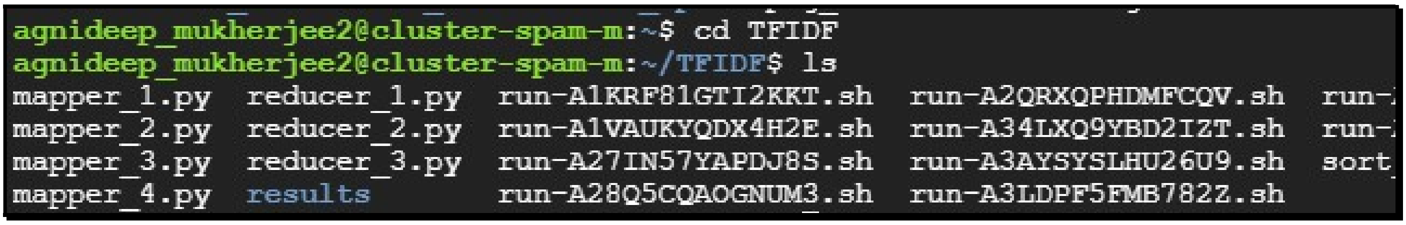
I have also created run shell scripts to integrate the mappers and reducers and provide the input and output path of each phase.


C: > Users > 91847 > Downloads > TFIDF MapReduce > $ run-A1KRF81GTI2KKT.sh
Find

```
hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
-file /home/agnideep_mukherjee2/TFIDF/mapper_1.py \
-mapper  'python3 mapper_1.py' \
-file /home/agnideep_mukherjee2/TFIDF/reducer_1.py \
-reducer 'python3 reducer_1.py' \
-input gs://dataproc-staging-us-central1-24631865637-6nswal7o/TFIDF_Ham/A1KRF81GTI2KKT.txt \
-output gs://dataproc-staging-us-central1-24631865637-6nswal7o/TFIDF_Ham/out1

hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
-file /home/agnideep_mukherjee2/TFIDF/mapper_2.py \
-mapper  'python3 mapper_2.py' \
-file /home/agnideep_mukherjee2/TFIDF/reducer_2.py \
-reducer 'python3 reducer_2.py' \
-input gs://dataproc-staging-us-central1-24631865637-6nswal7o/TFIDF_Ham/out1/ \
-output gs://dataproc-staging-us-central1-24631865637-6nswal7o/TFIDF_Ham/out2

hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
-file /home/agnideep_mukherjee2/TFIDF/mapper_3.py \
-mapper  'python3 mapper_3.py' \
-file /home/agnideep_mukherjee2/TFIDF/reducer_3.py \
-reducer 'python3 reducer_3.py' \
-input gs://dataproc-staging-us-central1-24631865637-6nswal7o/TFIDF_Ham/out2/ \
-output gs://dataproc-staging-us-central1-24631865637-6nswal7o/TFIDF_Ham/out3

hadoop jar /usr/lib/hadoop/hadoop-streaming.jar \
```

Figure 19: Example of a run shell script

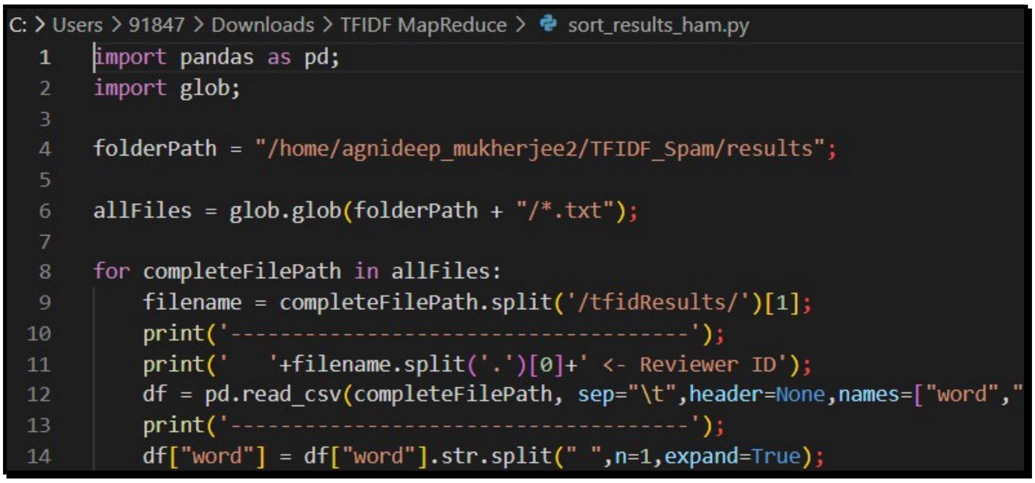
I have upload the 4 mappers and 3 Reducers and Run scripts for the 10 spam text files generated in data processing phase in our Hadoop File system as



```
agnideep_mukherjee2@cluster-spam-m:~$ cd TFIDF
agnideep_mukherjee2@cluster-spam-m:~/TFIDF$ ls
mapper_1.py  reducer_1.py  run-A1KRF81GTI2KKT.sh  run-A2QRXQPHDMFCQV.sh  run-
mapper_2.py  reducer_2.py  run-A1VAUKYQDX4H2E.sh  run-A34LXQ9YBD2IZT.sh  run-
mapper_3.py  reducer_3.py  run-A27IN57YAPDJ8S.sh  run-A3AYSYSLHU26U9.sh  sort
mapper_4.py  results       run-A28Q5CQAOGNUM3.sh  run-A3LDPF5FMB782Z.sh
```

Figure 20: Files uploaded in Hadoop File System

I have also created a python script 'sort_results.py' for sorting the top 10 results and displaying them in an understandable tabular format.

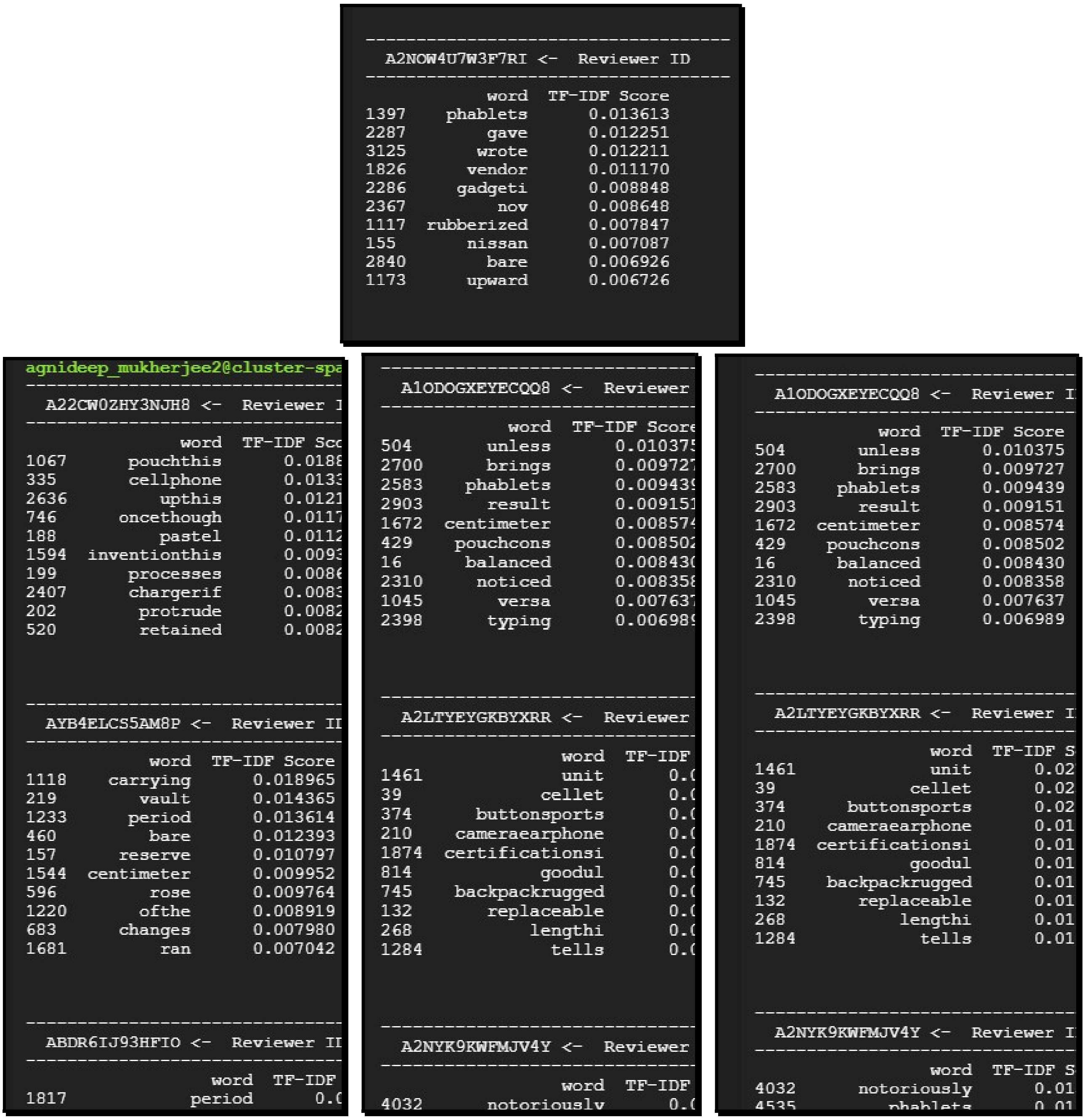

C: > Users > 91847 > Downloads > TFIDF MapReduce > sort_results_ham.py

```python
import pandas as pd;
import glob;

folderPath = "/home/agnideep_mukherjee2/TFIDF_Spam/results";

allFiles = glob.glob(folderPath + "/*.txt");

for completeFilePath in allFiles:
    filename = completeFilePath.split('/tfidResults/')[1];
    print('-------------------------------------');
    print('    '+filename.split('.')[0]+' <- Reviewer ID');
    df = pd.read_csv(completeFilePath, sep="\t",header=None,names=["word","
    print('-------------------------------------');
    df["word"] = df["word"].str.split(" ",n=1,expand=True);
```

Figure 21: Python code to print all the TF-IDF results in tabular format

iii. Initiating the process through cloud ssh.

Now I shall run the shell scripts run-<reviewerID> one by one for 10 Spam account reviewers and Print the data in tabular format using sort_results.py



```
-----------------------------------
  A2NOW4U7W3F7RI <-  Reviewer ID
-----------------------------------
            word  TF-IDF Score
1397    phablets      0.013613
2287        gave      0.012251
3125       wrote      0.012211
1826      vendor      0.011170
2286     gadgeti      0.008848
2367         nov      0.008648
1117  rubberized      0.007847
155       nissan      0.007087
2840        bare      0.006926
1173      upward      0.006726
```

```
agnideep_mukherjee2@cluster-spa
-----------------------------------
  A22CW0ZHY3NJH8 <-  Reviewer I
-----------------------------------
              word  TF-IDF Sco
1067      pouchthis      0.0188
335       cellphone      0.0133
2636         upthis      0.0121
746     oncethough      0.0117
188          pastel      0.0112
1594  inventionthis      0.0093
199       processes      0.0086
2407      chargerif      0.0083
202        protrude      0.0082
520        retained      0.0082


-----------------------------------
  AYB4ELCS5AM8P <-  Reviewer II
-----------------------------------
           word  TF-IDF Score
1118   carrying      0.018965
219       vault      0.014365
1233     period      0.013614
460        bare      0.012393
157     reserve      0.010797
1544  centimeter      0.009952
596        rose      0.009764
1220      ofthe      0.008919
683     changes      0.007980
1681        ran      0.007042


-----------------------------------
  ABDR6IJ93HFIO <-  Reviewer II
-----------------------------------
                word  TF-IDF
1817          period      0.0
```

```
-----------------------------------
  A1ODOGXEYECQQ8 <-  Reviewer
-----------------------------------
            word  TF-IDF Score
504       unless      0.010375
2700      brings      0.009727
2583    phablets      0.009439
2903      result      0.009151
1672  centimeter      0.008574
429    pouchcons      0.008502
16      balanced      0.008430
2310     noticed      0.008358
1045       versa      0.007637
2398      typing      0.006989


-----------------------------------
  A2LTYEYGKBYXRR <-  Reviewer
-----------------------------------
                 word  TF-IDF
1461             unit      0.0
39             cellet      0.0
374      buttonsports      0.0
210    cameraearphone      0.0
1874  certificationsi      0.0
814            goodul      0.0
745    backpackrugged      0.0
132       replaceable      0.0
268           lengthi      0.0
1284            tells      0.0


-----------------------------------
  A2NYK9KWFMJV4Y <-  Reviewer
-----------------------------------
                 word  TF-IDF
4032       notoriously     0.0
```

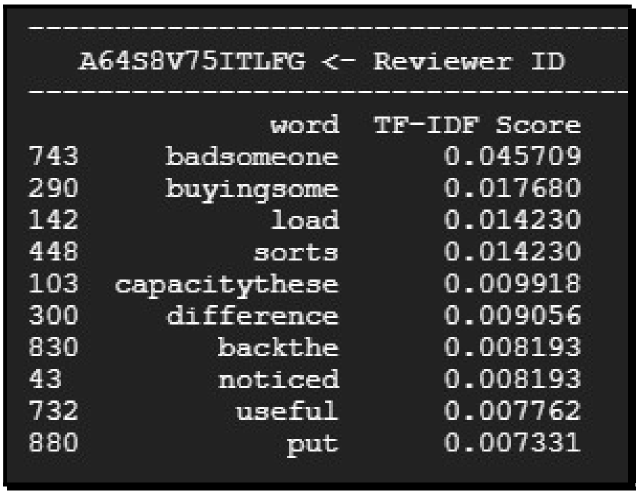
```
-----------------------------------
  A1ODOGXEYECQQ8 <-  Reviewer I
-----------------------------------
            word  TF-IDF Score
504       unless      0.010375
2700      brings      0.009727
2583    phablets      0.009439
2903      result      0.009151
1672  centimeter      0.008574
429    pouchcons      0.008502
16      balanced      0.008430
2310     noticed      0.008358
1045       versa      0.007637
2398      typing      0.006989


-----------------------------------
  A2LTYEYGKBYXRR <-  Reviewer I
-----------------------------------
                 word  TF-IDF S
1461             unit      0.02
39             cellet      0.02
374      buttonsports      0.02
210    cameraearphone      0.01
1874  certificationsi      0.01
814            goodul      0.01
745    backpackrugged      0.01
132       replaceable      0.01
268           lengthi      0.01
1284            tells      0.01


-----------------------------------
  A2NYK9KWFMJV4Y <-  Reviewer I
-----------------------------------
                 word  TF-IDF S
4032      notoriously      0.01
4535         phablets      0.01
```

Figure 22: TF-IDF results for Spam

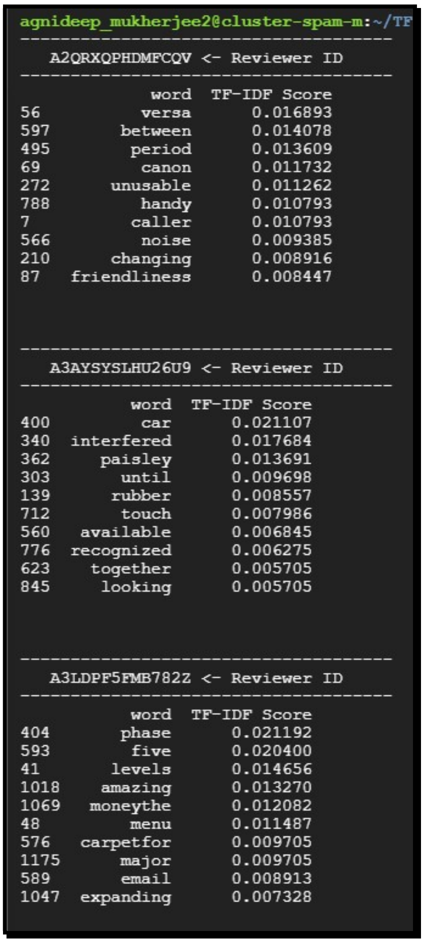
b. TF-IDF of Top 10 ham keywords for top 10 ham accounts

I have followed the same steps and logic that I have previously applied while calculating TF-IDF for top 10 spam keywords in the top 10 spam accounts.

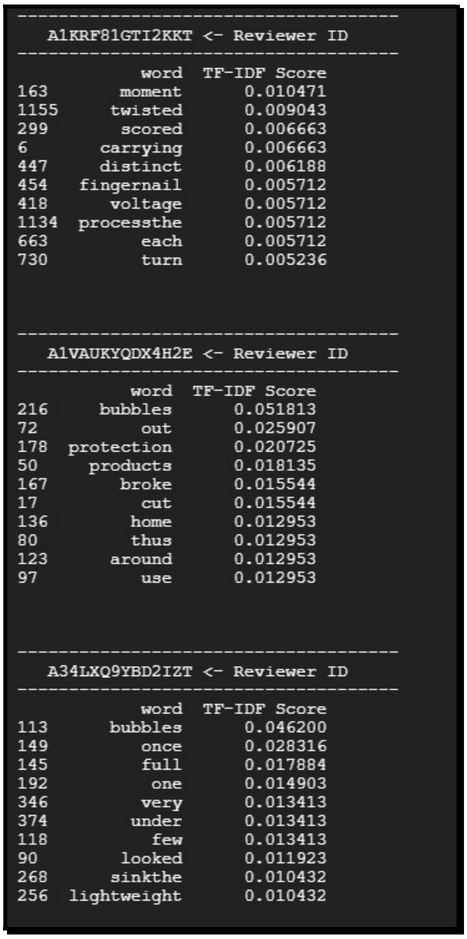
I am attaching the result for the same.



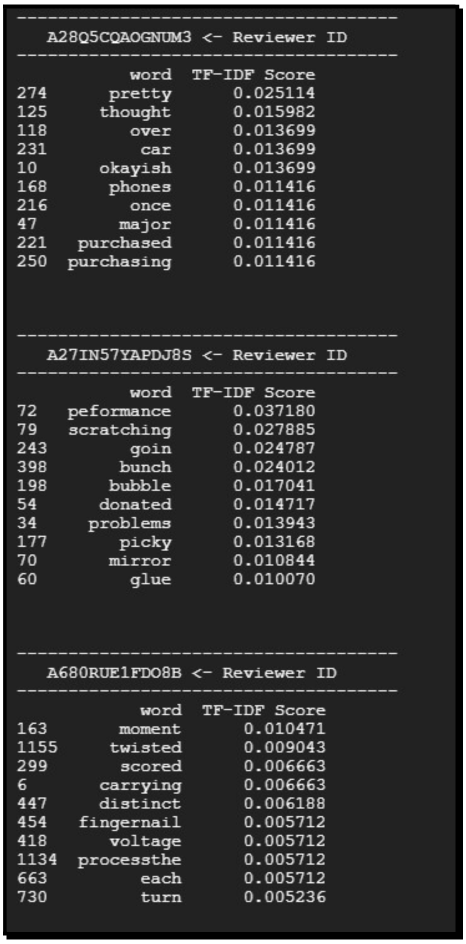
```
----------------------------------------
    A64S8V75ITLFG <- Reviewer ID
----------------------------------------
             word  TF-IDF Score
743    badsomeone      0.045709
290    buyingsome      0.017680
142          load      0.014230
448         sorts      0.014230
103  capacitythese     0.009918
300    difference      0.009056
830       backthe      0.008193
43        noticed      0.008193
732        useful      0.007762
880           put      0.007331
```



```
agnideep_mukherjee2@cluster-spam-m:~/TF
----------------------------------------
   A2QRXQPHDMFCQV <- Reviewer ID
----------------------------------------
              word  TF-IDF Score
56           versa      0.016893
597        between      0.014078
495         period      0.013609
69           canon      0.011732
272       unusable      0.011262
788          handy      0.010793
7           caller      0.010793
566          noise      0.009385
210       changing      0.008916
87    friendliness      0.008447



----------------------------------------
   A3AYSYSLHU26U9 <- Reviewer ID
----------------------------------------
            word  TF-IDF Score
400          car      0.021107
340   interfered      0.017684
362      paisley      0.013691
303        until      0.009698
139       rubber      0.008557
712        touch      0.007986
560    available      0.006845
776   recognized      0.006275
623     together      0.005705
845      looking      0.005705



----------------------------------------
   A3LDPF5FMB782Z <- Reviewer ID
----------------------------------------
            word  TF-IDF Score
404        phase      0.021192
593         five      0.020400
41        levels      0.014656
1018     amazing      0.013270
1069    moneythe      0.012082
48          menu      0.011487
576    carpetfor      0.009705
1175       major      0.009705
589        email      0.008913
1047   expanding      0.007328
```



```
----------------------------------------
   A1KRF81GTI2KKT <- Reviewer ID
----------------------------------------
             word  TF-IDF Score
163        moment      0.010471
1155      twisted      0.009043
299        scored      0.006663
6        carrying      0.006663
447      distinct      0.006188
454    fingernail      0.005712
418       voltage      0.005712
1134   processthe      0.005712
663          each      0.005712
730          turn      0.005236



----------------------------------------
   A1VAUKYQDX4H2E <- Reviewer ID
----------------------------------------
            word  TF-IDF Score
216      bubbles      0.051813
72           out      0.025907
178   protection      0.020725
50      products      0.018135
167        broke      0.015544
17           cut      0.015544
136         home      0.012953
80          thus      0.012953
123       around      0.012953
97           use      0.012953



----------------------------------------
   A34LXQ9YBD2IZT <- Reviewer ID
----------------------------------------
             word  TF-IDF Score
113       bubbles      0.046200
149          once      0.028316
145          full      0.017884
192           one      0.014903
346          very      0.013413
374         under      0.013413
118           few      0.013413
90         looked      0.011923
268       sinkthe      0.010432
256   lightweight      0.010432
```



```
----------------------------------------
   A28Q5CQAOGNUM3 <- Reviewer ID
----------------------------------------
            word  TF-IDF Score
274       pretty      0.025114
125      thought      0.015982
118         over      0.013699
231          car      0.013699
10       okayish      0.013699
168       phones      0.011416
216         once      0.011416
47         major      0.011416
221    purchased      0.011416
250   purchasing      0.011416



----------------------------------------
   A27IN57YAPDJ8S <- Reviewer ID
----------------------------------------
            word  TF-IDF Score
72    peformance      0.037180
79    scratching      0.027885
243         goin      0.024787
398        bunch      0.024012
198       bubble      0.017041
54       donated      0.014717
34      problems      0.013943
177        picky      0.013168
70        mirror      0.010844
60          glue      0.010070



----------------------------------------
   A680RUE1FDO8B <- Reviewer ID
----------------------------------------
             word  TF-IDF Score
163        moment      0.010471
1155      twisted      0.009043
299        scored      0.006663
6        carrying      0.006663
447      distinct      0.006188
454    fingernail      0.005712
418       voltage      0.005712
1134   processthe      0.005712
663          each      0.005712
730          turn      0.005236
```

Figure 23: TF-IDF for Ham Accounts

6) References

1. https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/
2. https://pig.apache.org/docs/r0.17.0/api/org/apache/pig/piggybank/storage/CSVExcelStorage.html
3. https://www.geeksforgeeks.org/tf-idf-model-for-page-ranking/
4. https://data.stackexchange.com/stackoverflow/query/new
5. https://stackoverflow.com/questions/20731966/regex-remove-all-special-characters-except-numbers
6. https://www.geeksforgeeks.org/removing-stop-words-nltk-python/