

# **ASSIGNMENT 2 REPORT**

**CSE 587 : DATA INTENSIVE COMPUTING**

**Akshay S Gehlot (50243400)**

**Akash Malla (50336850)**

**Rohit (50336890)**

*The objective of this assignment is to get started with big data processing with Hadoop. The goals of the assignment are to implement basic text processing tasks from scratch on the Hadoop framework*

**PART1 - Setup and WordCount - 5 Points**

- Get familiar with the VM and the Hadoop framework
- In the folder gutenber, located in your home directory, there are 3 documents. Use necessary commands to transfer data to Hadoop distributed file system
- Implement a MapReduce algorithm to produce count of every word in the document

Setting up :

Once we had downloaded the VM and installed the image we then had to get the hadoop framework up and running . We used the start-all.sh command to get the various nodes started and running

To copy the contents of our Gutenberg folder to HDFS we utilized -copyFromLocal command which transfers local data to the HDFS .

Word Count :

Word count is one of the basic Hadoop algorithms and to perform this via Mapreduce model we created a separate mapper.py and reducer.py file

We use the strip() and split() commands in the mapper to get words from each line and then print them out one by one , For now we keep count as 1 and will iterate it further in the reducer . The mapper returns output in the format

Worda 1

Worda 1

Worda 1

Wordb 1

Wordc 1 ...

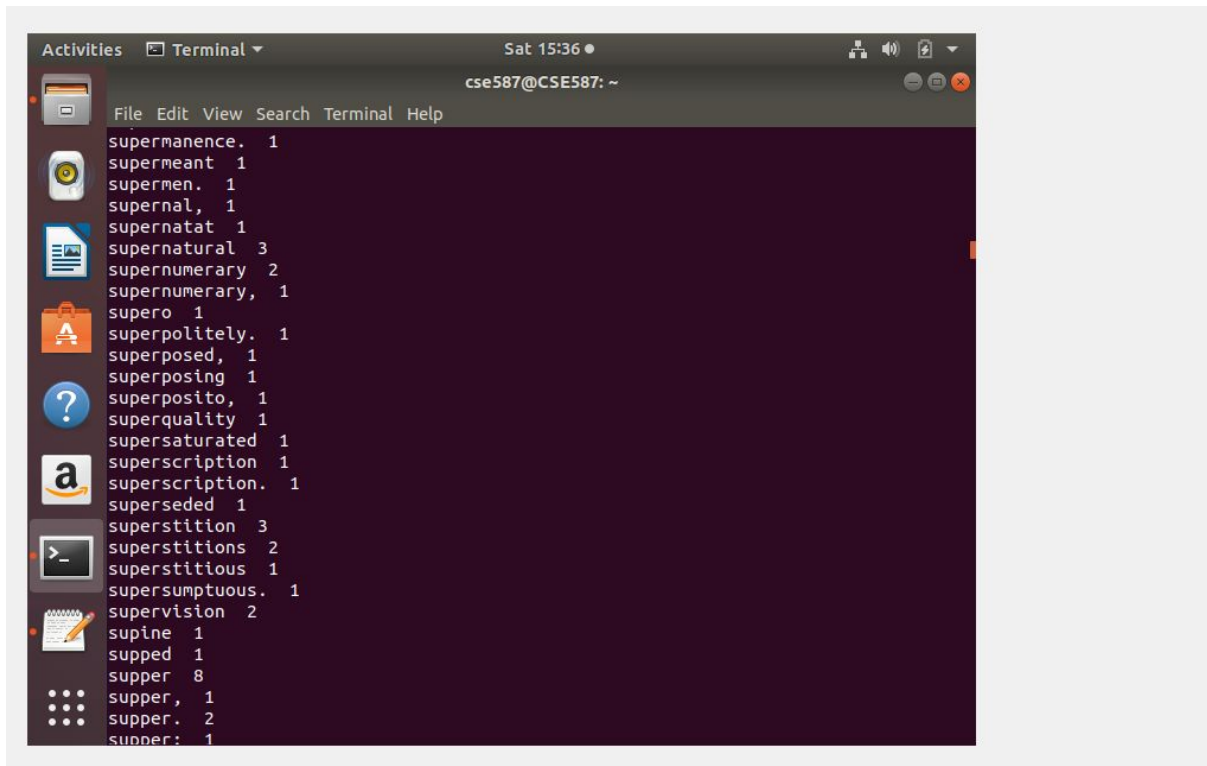
The reducer similarly takes in as input the output from mapper and we again utilize strip() and split() to differentiate the word and count values , Now here we make an index of the word and increase its count every time index[word] shows up in the list of words we got from mapper

Once we have this we can print the output out in the format

Worda 3

Wordb 1

Wordc 1



The screenshot shows a terminal window titled 'Terminal' with a menu bar (File, Edit, View, Search, Terminal, Help) and a status bar (Sat 15:36, cse587@CSE587: ~). The terminal displays a list of words and their frequencies, sorted in descending order of frequency. The words are: supermanence. 1, supermeant 1, supermen. 1, supernal, 1, supernatant 1, supernatural 3, supernumerary 2, supernumerary, 1, supero 1, superpolitely. 1, superposed, 1, superposing 1, superposito, 1, superquality 1, supersaturated 1, superscription 1, superscription. 1, superseded 1, superstition 3, superstitions 2, superstitious 1, supersumptuous. 1, supervision 2, supine 1, supped 1, supper 8, supper, 1, supper. 2, and supper: 1.

```
supermanence. 1
supermeant 1
supermen. 1
supernal, 1
supernatant 1
supernatural 3
supernumerary 2
supernumerary, 1
supero 1
superpolitely. 1
superposed, 1
superposing 1
superposito, 1
superquality 1
supersaturated 1
superscription 1
superscription. 1
superseded 1
superstition 3
superstitions 2
superstitious 1
supersumptuous. 1
supervision 2
supine 1
supped 1
supper 8
supper, 1
supper. 2
supper: 1
```

**PART 2 - N-grams- 10 Points** • Using the same gutenber dataset, implement a MapReduce algorithm that will produce modified tri-grams around the key words, after replacing the key word with '\$'. Example: cat was sitting on a roof ---> if the key word was 'sitting' ---> the modified tri-grams would be cat\_was\_\$, was\_\$\_on,\$\_on\_a, • The key words to look for in the gutenber dataset are 'science', 'sea', 'fire'. • The algorithm after producing these modified tri-grams, should return the 10 most occurred modified tri-gram in the dataset.

Description of how you solved this (including the logic )

Mapper Logic:

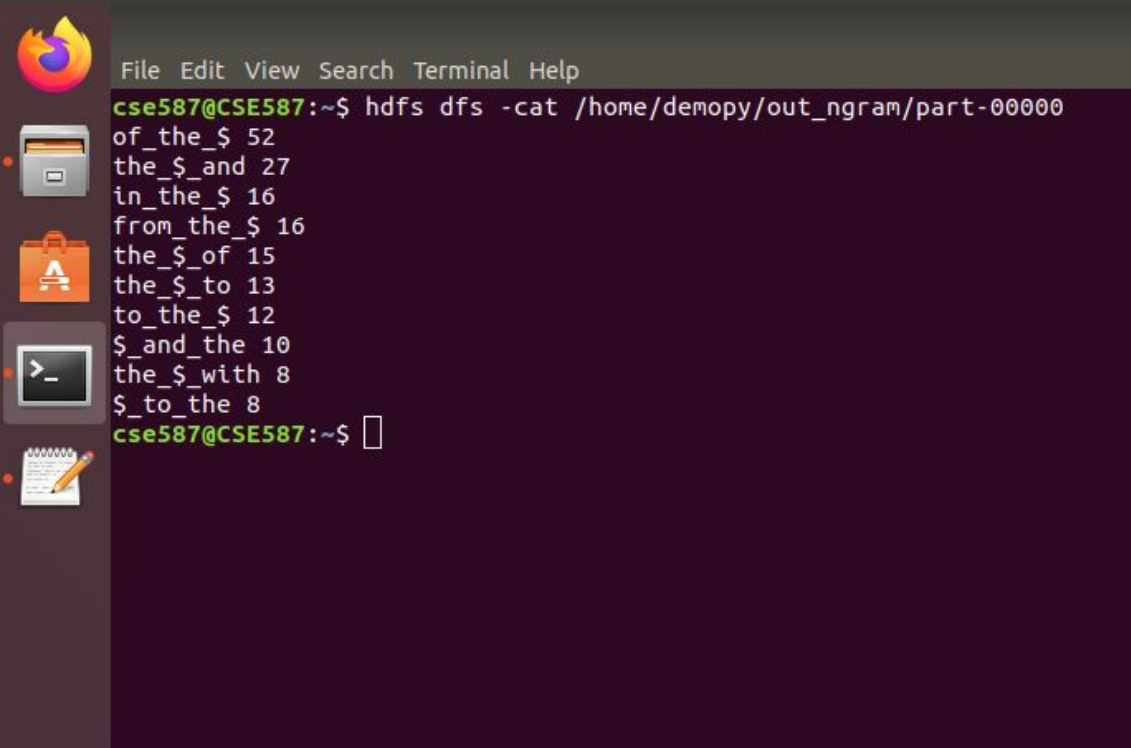
We are dividing the given input files into lines and removing the spaces from beginning to end of the line using the strip() function. Then we are calculating the length of line and we are printing the line and the lengths which will be the input for the reducer. The mapper returns the output of the format as follows:

Example: will be made on the earth 6

### Reducer Logic:

We are getting the input from the mapper as line and Length of line and then splitting the line into words. Then we are running the loop from the start of the line till the end of it (i.e. till the length of the line) and checking the 3 words whether they belong to the key words (sea, science, fire) or not. If they belong to any of these 3 key words, we are replacing it by the \$ and making a tri-gram of those 3 words which we have selected. We are storing it to a dictionary and keeping a count of it and increasing the count of each tri-gram if they appear again. Thereby, we are sorting all the words by count of each tri-gram and displaying the top 10 most occurring tri-grams.

### Screenshot after running first mapreduce



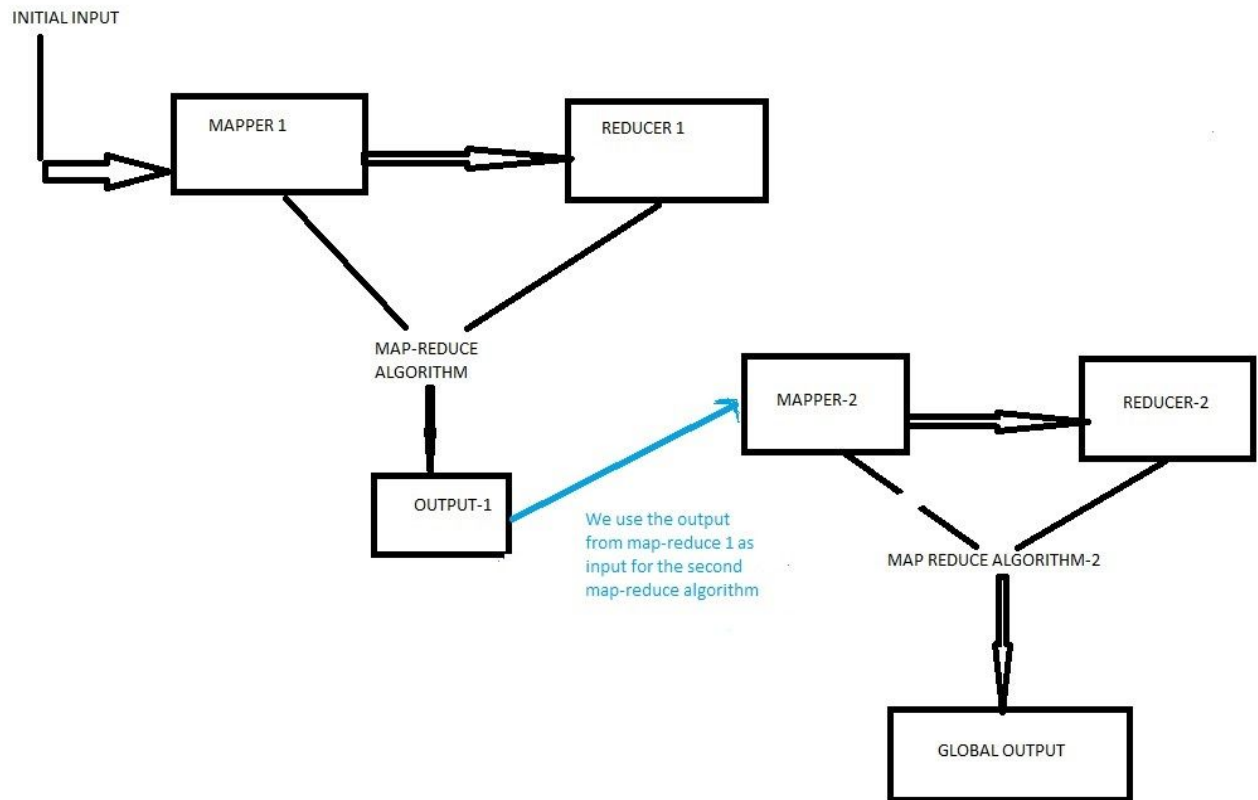
```
File Edit View Search Terminal Help
cse587@CSE587:~$ hdfs dfs -cat /home/demopy/out_ngram/part-00000
of_the_$ 52
the_$_and 27
in_the_$ 16
from_the_$ 16
the_$_of 15
the_$_to 13
to_the_$ 12
$_and_the 10
the_$_with 8
$_to_the 8
cse587@CSE587:~$
```

### For Multiple MapReduce -

We have implemented the second map-reduce i.e. mapper\_ngram\_2.py and reducer\_ngram\_2.py.

### 2nd Mapper Logic :

We are taking the output of the first reducer as input and splitting the line by tri-gram and count and printing 3 strings - ngram (as a temporary string), tri-gram word, count. We are printing the ngram temporary string so that all the printing values go to the same reducer.



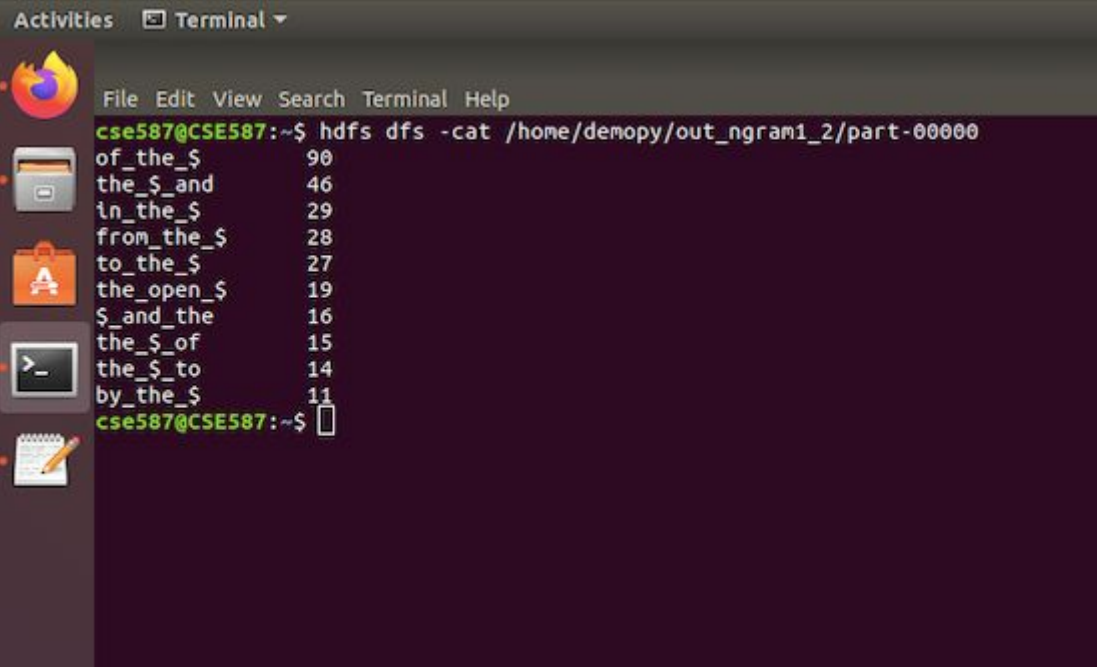
( Flow diagram of the process)

2nd Reducer Logic:

We are taking the output of the 2nd mapper as input and here we are storing the initial count of each trigram in a list and increasing the count if that same trigram is appeared again in the input. And then we print the global top 10 most occurring trigrams.

For running the 2nd MapReduce we have taken the output of the 1st mapreduce as input

ScreenShot after running the second map-reduce :-



The screenshot shows a terminal window with a dark background. The title bar at the top says "Activities" and "Terminal". The menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The prompt is "cse587@CSE587:~\$". The command entered is "hdfs dfs -cat /home/demopy/out\_ngram1\_2/part-00000". The output is a list of n-grams and their counts:

of_the_\$	90
the_\$_and	46
in_the_\$	29
from_the_\$	28
to_the_\$	27
the_open_\$	19
\$_and_the	16
the_\$_of	15
the_\$_to	14
by_the_\$	11

The prompt "cse587@CSE587:~\$" is shown again at the bottom.

**PART 3 - Inverted Index - 5 Points • Using the gutenber dataset, implement a MapReduce algorithm to produce inverted index for the whole dataset. • A small explanation of what inverted index is can be found in the link [Inverted index](#)**

Mapper Logic :

To begin with we needed a document id for each file ,I chose the name of the file as document id and stored it via map\_input\_file function ,To get it in the desired format we used the last 2 outputs of map\_input\_file and concatenate them with a “.” in between

So for ex. [ ‘arthur’,’txt’ ] became -> arthur.txt

Then we moved on pre-processing the gutenber input files which included removing spaces and other unnecessary information via the strip() and split() commands , To find all the words in the text we used .findall() command which works with regular expressions and ensures that we didn’t miss out on any words . And Finally for the last mapper step we printed out the word and its corresponding doc\_id in the format

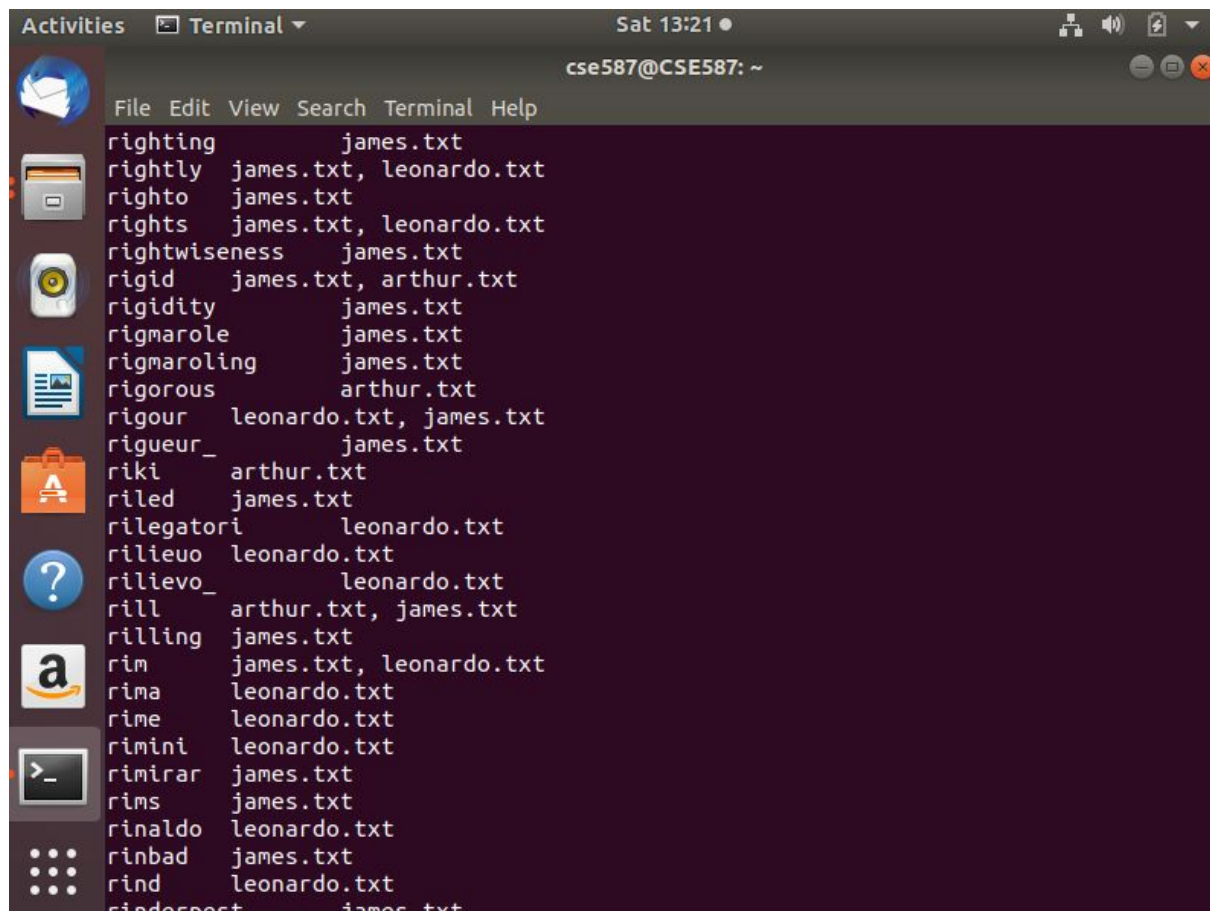
Word1 doc\_id  
Word2 doc\_id  
Word3 doc\_id

#### Reducer Logic :

The reducer takes in as input the output from the mapper.py program , So we had a list of words and their corresponding doc\_ids but we still needed to reduce them as a word occurring in multiple files would show up separately for each doc\_id

We again used strip() and split() functions to separate the words from their doc\_ids and proceeded to create a list of the doc\_ids for each word via an index

Once we had the list we are joining them and printing out the desired output which was of the type word : doc\_id1,doc\_id2



A terminal window titled 'Terminal' with a dark purple background. The window shows a list of words and their corresponding document IDs (doc\_ids) separated by a colon. The words are listed on the left, and the doc\_ids are listed on the right. The doc\_ids are either a single file name or a list of file names separated by commas. The window has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The status bar at the top shows 'Sat 13:21' and 'cse587@CSE587: ~'.

```
righting      james.txt
rightly       james.txt, leonardo.txt
righto        james.txt
rights        james.txt, leonardo.txt
rightwiseness james.txt
rigid         james.txt, arthur.txt
rigidity      james.txt
rigmarole     james.txt
rigmaroling   james.txt
rigorous      arthur.txt
rigour        leonardo.txt, james.txt
rigueur_      james.txt
riki          arthur.txt
riled         james.txt
rilegatori    leonardo.txt
rilieu        leonardo.txt
rilievo_      leonardo.txt
rill          arthur.txt, james.txt
rilling       james.txt
rim           james.txt, leonardo.txt
rima          leonardo.txt
rime          leonardo.txt
rimini        leonardo.txt
rimirar       james.txt
rims          james.txt
rinaldo       leonardo.txt
rinbad        james.txt
rind          leonardo.txt
rinderpest    james.txt
```

**PART 4 - Relational Join - 5 Points • Using the Dataset provided along with the assignment, Implement a MapReduce algorithm to join two datasets using a primary key • The assumed primary key is the ‘Employee ID’**

After importing the csv files to hadoop our main task was implementing a join mapreduce function . We used the strip() and split functions to go line by line in the data ,

**Mapper Logic:**

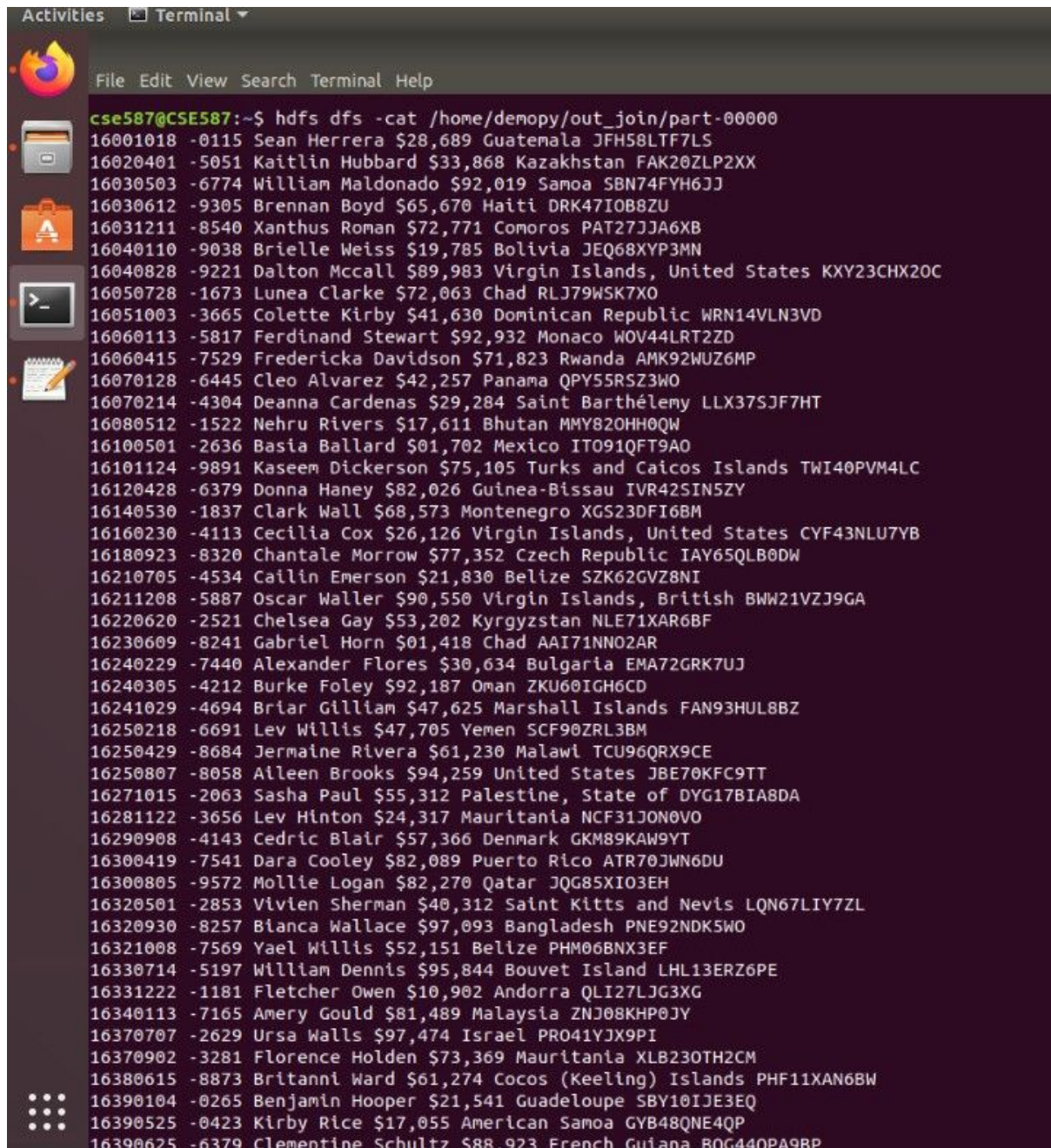
In mapper we are getting the input from Join1.tsv and Join2.tsv and reading both the files by line and splitting the line into words and printing the words.As Join1.tsv has only 2 column and Join2.tsv has 4 columns, while reading the line we distinguish the line by the number of fields in each row,if the length of the line is 2 then we are printing the Employee ID and Name(line from Join1) otherwise we are printing the Employee ID, Salary,Place,Code(line from Join2) and giving this output to the reducer.

**Reducer Logic:**

In reducer, we will fetch the data of same Employee ID in same reducer as the Employee ID is our key .The input from mapper will be in sorted order by key as Employee ID, so there will be 2 rows with the same Employee ID, one from Join1 data and the other row from Join2 data.So while reading the first row, for each new Employee ID we are storing the details(Employee ID , name or Employee ID, salary,place,code ) and joining them with the details we are getting from the second row.And then printing the merged output.



Screenshot:



```
Activities Terminal
File Edit View Search Terminal Help

cse587@CSE587:~$ hdfs dfs -cat /home/demopy/out_join/part-00000
16001018 -0115 Sean Herrera $28,689 Guatemala JFH58LTF7LS
16020401 -5051 Kaitlin Hubbard $33,868 Kazakhstan FAK20ZLP2XX
16030503 -6774 William Maldonado $92,019 Samoa SBN74FYH6JJ
16030612 -9305 Brennan Boyd $65,670 Haiti DRK47IOB8ZU
16031211 -8540 Xanthus Roman $72,771 Comoros PAT27JJA6XB
16040110 -9038 Brielle Weiss $19,785 Bolivia JEQ68XYP3MN
16040828 -9221 Dalton Mccall $89,983 Virgin Islands, United States KXY23CHX20C
16050728 -1673 Luneia Clarke $72,063 Chad RLJ79WSK7XO
16051003 -3665 Colette Kirby $41,630 Dominican Republic WRN14VLN3VD
16060113 -5817 Ferdinand Stewart $92,932 Monaco WOV44LRT2ZD
16060415 -7529 Fredericka Davidson $71,823 Rwanda AMK92WUZ6MP
16070128 -6445 Cleo Alvarez $42,257 Panama QPY55RSZ3WO
16070214 -4304 Deanna Cardenas $29,284 Saint Barthélemy LLX37SJF7HT
16080512 -1522 Nehru Rivers $17,611 Bhutan MMY82OH0QW
16100501 -2636 Basia Ballard $01,702 Mexico IT091QFT9AO
16101124 -9891 Kaseem Dickerson $75,105 Turks and Caicos Islands TWI40PVM4LC
16120428 -6379 Donna Haney $82,026 Guinea-Bissau IVR42SIN5ZY
16140530 -1837 Clark Wall $68,573 Montenegro XGS23DFI6BM
16160230 -4113 Cecilia Cox $26,126 Virgin Islands, United States CYF43NLU7YB
16180923 -8320 Chantale Morrow $77,352 Czech Republic IAY65QLB0DW
16210705 -4534 Cailin Emerson $21,830 Belize SZK62GVZ8NI
16211208 -5887 Oscar Waller $90,550 Virgin Islands, British BWW21VZJ9GA
16220620 -2521 Chelsea Gay $53,202 Kyrgyzstan NLE71XAR6BF
16230609 -8241 Gabriel Horn $01,418 Chad AAI71NNO2AR
16240229 -7440 Alexander Flores $30,634 Bulgaria EMA72GRK7UJ
16240305 -4212 Burke Foley $92,187 Oman ZKU60IGH6CD
16241029 -4694 Briar Gilliam $47,625 Marshall Islands FAN93HUL8BZ
16250218 -6691 Lev Willis $47,705 Yemen SCF90ZRL3BM
16250429 -8684 Jermaine Rivera $61,230 Malawi TCU96QRX9CE
16250807 -8058 Aileen Brooks $94,259 United States JBE70KFC9TT
16271015 -2063 Sasha Paul $55,312 Palestine, State of DYG17BIA8DA
16281122 -3656 Lev Hinton $24,317 Mauritania NCF31JON0VO
16290908 -4143 Cedric Blair $57,366 Denmark GKM89KAW9YT
16300419 -7541 Dara Cooley $82,089 Puerto Rico ATR70JWN6DU
16300805 -9572 Mollie Logan $82,270 Qatar JQG85XIO3EH
16320501 -2853 Vivien Sherman $40,312 Saint Kitts and Nevis LQN67LIY7ZL
16320930 -8257 Bianca Wallace $97,093 Bangladesh PNE92NDK5WO
16321008 -7569 Yael Willis $52,151 Belize PHM06BNX3EF
16330714 -5197 William Dennis $95,844 Bouvet Island LHL13ERZ6PE
16331222 -1181 Fletcher Owen $10,902 Andorra QLI27LJG3XG
16340113 -7165 Amery Gould $81,489 Malaysia ZNJ08KHP0JY
16370707 -2629 Ursa Walls $97,474 Israel PRO41YJX9PI
16370902 -3281 Florence Holden $73,369 Mauritania XLB230TH2CM
16380615 -8873 Britanni Ward $61,274 Cocos (Keeling) Islands PHF11XAN6BW
16390104 -0265 Benjamin Hooper $21,541 Guadeloupe SBY10IJE3EQ
16390525 -0423 Kirby Rice $17,055 American Samoa GYB48QNE4QP
16390625 -6379 Clementine Schultz $88,923 French Guiana B0G440PA9BP
```

**BONUS: K-Nearest Neighbour - 5 Points • Using the train and test set provided along with the assignment, Implement KNN algorithm using MapReduce. • You can assume the test set is small. • The algorithm should return the corresponding predicted label for each test instance**

We are reading the 2 files, Train.csv and Test.csv. We have copied the Train file in hdfs directory - /home/demopy/knn/ and reading the Test.csv file which are giving in the command i.e

```
hadoop jar /home/cse587/hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar  
-file /home/cse587/mapper_knn.py -mapper /home/cse587/mapper_knn.py -file  
/home/cse587/reducer_knn.py -reducer /home/cse587/reducer_knn.py -file  
/home/cse587/Test.csv -input /home/demopy/knn/ -output /home/demopy/out_knn4
```

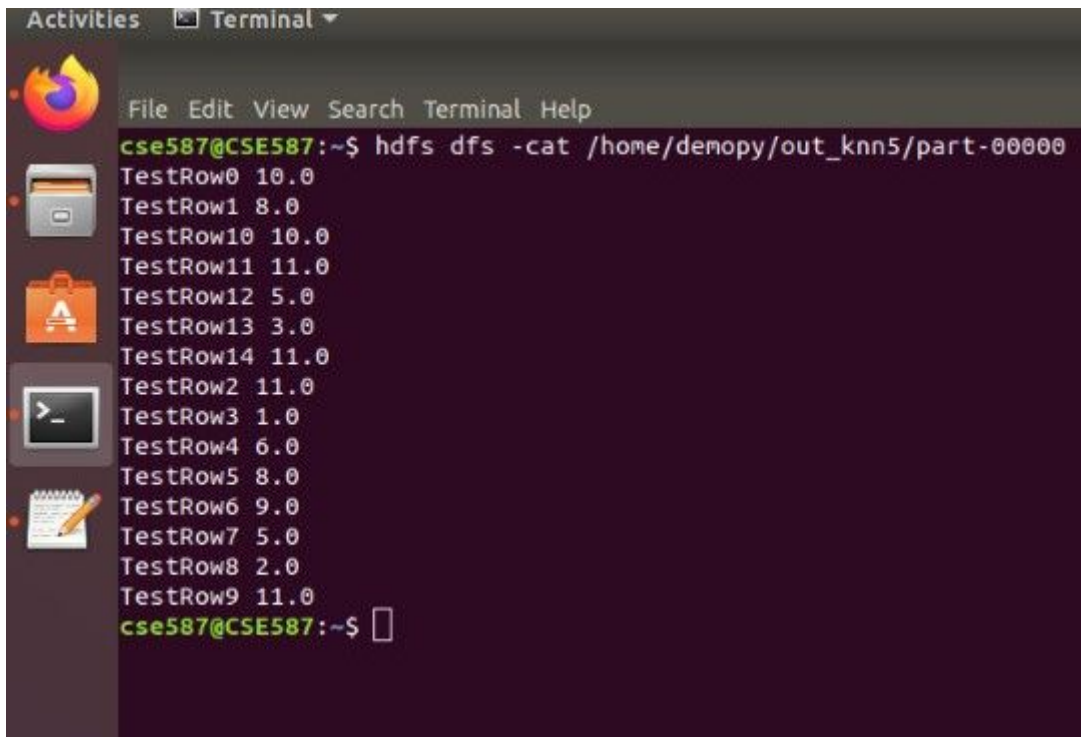
Mapper Logic:

In Mapper, as Test.csv is small in size, we are reading the Test as csv and converting it into a numpy array. And then we are reading the Train.csv by system standard input by line and converting each line to numpy array and after storing the label i.e the last column of the Train.csv, we are deleting the last column. While reading each Train row, we are calculating the euclidean distance between each Train row and every Test rows, and then we are printing the Test row id, euclidean distance we calculated for that Test row and the Train label which we have stored. The mapper output will be the input for reducer.

Reducer Logic:

In Reducer, as we will get the input from the mapper in sorted order by test row id, data with the same test row id will be in the same reducer. While reading the input line by line, for each new test row id, we are creating a list for that test row id otherwise, we are appending the euclidean distance and label to that list of test row id. Moreover, we are sorting the list of test row id, by the euclidean distance and taking the top k(=3) minimum values. Thereby, we are calculating the votes of each label among the 3 values and the label which gets the maximum votes will be the predictive label for that particular test row id. Then, we are printing the test row id and the predicted label.

Screenshots:



```
Activities Terminal
File Edit View Search Terminal Help
cse587@CSE587:~$ hdfs dfs -cat /home/demopy/out_knn5/part-00000
TestRow0 10.0
TestRow1 8.0
TestRow10 10.0
TestRow11 11.0
TestRow12 5.0
TestRow13 3.0
TestRow14 11.0
TestRow2 11.0
TestRow3 1.0
TestRow4 6.0
TestRow5 8.0
TestRow6 9.0
TestRow7 5.0
TestRow8 2.0
TestRow9 11.0
cse587@CSE587:~$
```

Link of Video -

<https://drive.google.com/drive/folders/1qk5TWnLMTg7uzAKA5JoMKTnmmNogho4D?usp=sharing>

## CITATIONS AND REFERENCES

In Solving this assignment we took help of various tools and resources which are listed below

1. <https://en.wikipedia.org/wiki/MapReduce>
2. <https://www.geeksforgeeks.org/inverted-index/>
3. [https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html)
4. <https://lintool.github.io/MapReduceAlgorithms/ed1n/MapReduce-algorithms.pdf>  
(^^ Great Textbook for map reduce content)
5. <https://stackabuse.com/the-python-string-strip-function/>
6. <https://medium.com/@rrfd/your-first-map-reduce-using-hadoop-with-python-and-osx-ca3b6f3dfe78>
7. <https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python>  
(Great tutorial for Mapreduce and Hadoop beginners)

8. <https://helpdeskgeek.com/virtualization/virtualbox-share-folder-host-guest/>

9. Hadoop Python Demo in class lectures

10. [https://www.python-course.eu/k\\_nearest\\_neighbor\\_classifier.php](https://www.python-course.eu/k_nearest_neighbor_classifier.php)

11. <https://acadgild.com/blog/k-nearest-neighbor-algorithm>

12. [https://en.wikipedia.org/wiki/Inverted\\_index](https://en.wikipedia.org/wiki/Inverted_index)

13. <https://www.edureka.co/blog/hadoop-streaming-mapreduce-program/>