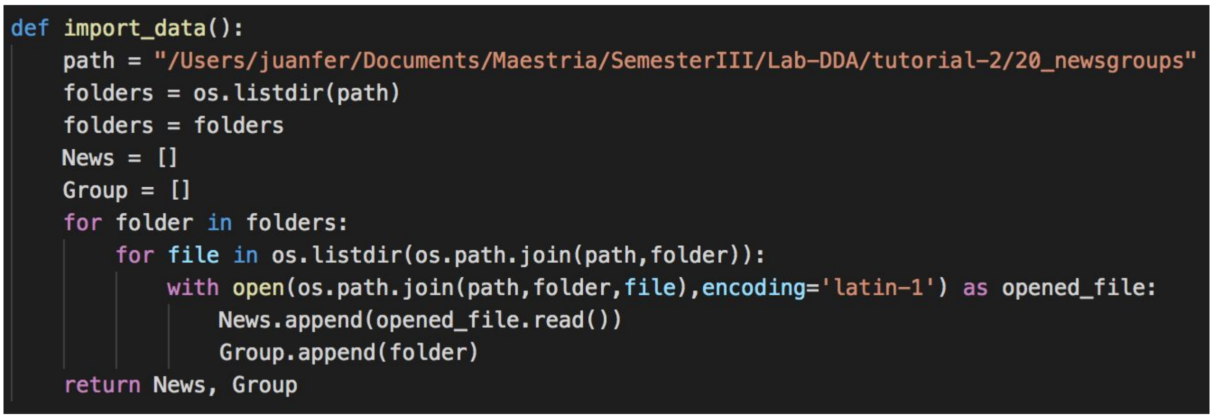
LAB DISTRIBUTED DATA ANALYTICS I
TUTORIAL II

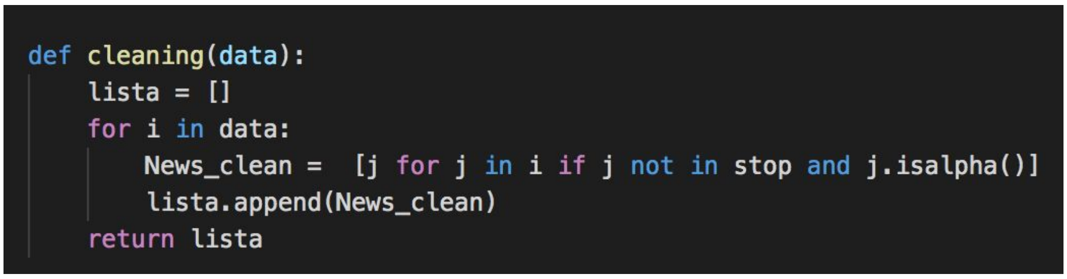
Juan Fernando Espinosa Reinoso
303158
Group 1: Monday Tutorial

Exercise 1: Data cleaning and text tokenization (5 points)

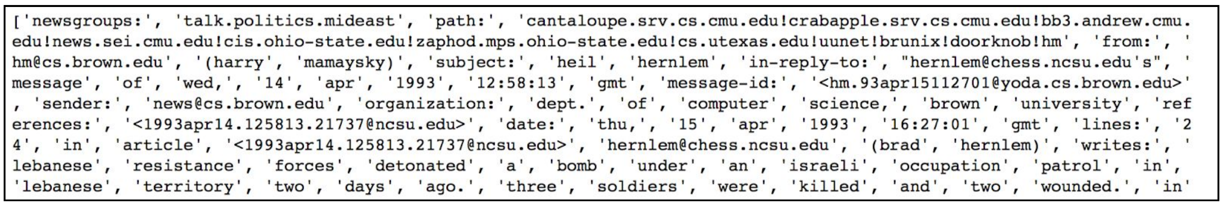
The dataset is composed of a folder/subfolder/document organization. Therefore, first I import the file from the OS and then iterate through all the subfolders extracting the documents and appending into a list. all this process is done in the function **import data.**


```python
def import_data():
    path = "/Users/juanfer/Documents/Maestria/SemesterIII/Lab-DDA/tutorial-2/20_newsgroups"
    folders = os.listdir(path)
    folders = folders
    News = []
    Group = []
    for folder in folders:
        for file in os.listdir(os.path.join(path,folder)):
            with open(os.path.join(path,folder,file),encoding='latin-1') as opened_file:
                News.append(opened_file.read())
                Group.append(folder)
    return News, Group
```

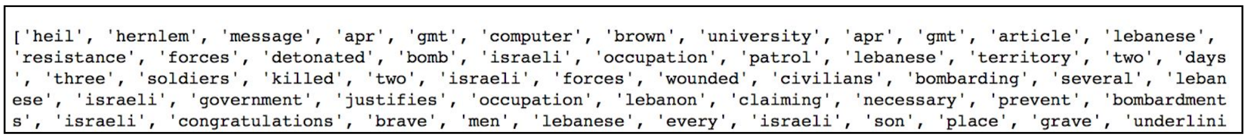
Next, as it is required to clean the documents a function **cleaning** which main goal is to erase stop words and tokenize each document that they receives.


```python
def cleaning(data):
    lista = []
    for i in data:
        News_clean =  [j for j in i if j not in stop and j.isalpha()]
        lista.append(News_clean)
    return lista
```

To show a small example, the data input of a document present tokens that are not currently words:


```
['newsgroups:', 'talk.politics.mideast', 'path:', 'cantaloupe.srv.cs.cmu.edu!crabapple.srv.cs.cmu.edu!bb3.andrew.cmu.
edu!news.sei.cmu.edu!cis.ohio-state.edu!zaphod.mps.ohio-state.edu!cs.utexas.edu!uunet!brunix!doorknob!hm', 'from:', '
hm@cs.brown.edu', '(harry', 'mamaysky)', 'subject:', 'heil', 'hernlem', 'in-reply-to:', "hernlem@chess.ncsu.edu's", '
message', 'of', 'wed,', '14', 'apr', '1993', '12:58:13', 'gmt', 'message-id:', '<hm.93apr15112701@yoda.cs.brown.edu>'
, 'sender:', 'news@cs.brown.edu', 'organization:', 'dept.', 'of', 'computer', 'science,', 'brown', 'university', 'ref
erences:', '<1993apr14.125813.21737@ncsu.edu>', 'date:', 'thu,', '15', 'apr', '1993', '16:27:01', 'gmt', 'lines:', '2
4', 'in', 'article', '<1993apr14.125813.21737@ncsu.edu>', 'hernlem@chess.ncsu.edu', '(brad', 'hernlem)', 'writes:', '
lebanese', 'resistance', 'forces', 'detonated', 'a', 'bomb', 'under', 'an', 'israeli', 'occupation', 'patrol', 'in',
'lebanese', 'territory', 'two', 'days', 'ago.', 'three', 'soldiers', 'were', 'killed', 'and', 'two', 'wounded.', 'in'
```

Then, after applying the function **cleaning** all the links, stop-words are erased:



```
['heil', 'hernlem', 'message', 'apr', 'gmt', 'computer', 'brown', 'university', 'apr', 'gmt', 'article', 'lebanese',
'resistance', 'forces', 'detonated', 'bomb', 'israeli', 'occupation', 'patrol', 'lebanese', 'territory', 'two', 'days
', 'three', 'soldiers', 'killed', 'two', 'israeli', 'forces', 'wounded', 'civilians', 'bombarding', 'several', 'leban
ese', 'israeli', 'government', 'justifies', 'occupation', 'lebanon', 'claiming', 'necessary', 'prevent', 'bombardment
s', 'israeli', 'congratulations', 'brave', 'men', 'lebanese', 'every', 'israeli', 'son', 'place', 'grave', 'underlini
```

1. The initialization of the workers starts and the worker 0 (root) will work as the organizer and it will process the first batch of data: It sends the data to the workers in the following manner: from the list of documents extracted with the function **import_data** it divides it taking the len(News)/total_workers and sending each batch to the workers. Then, it will apply the function **cleaning** to preprocess some
2. Worker 1 to n receives each batch of the data and apply the function **cleaning** to pre-process its batch of data and return them to the root.
3. Finally, the root worker joins the lists together, and exports each document preprocessed to a folder called Preprocessing. This files will be used in the next exercise.

The timing measure of the process with different number of workers is presented next:

Note: MPI crashes after dealing with more than 9.998 documents due to a memory error/processes degradation by oversubscribe more than 2 workers. Therefore, the following comparison will be done until that process.

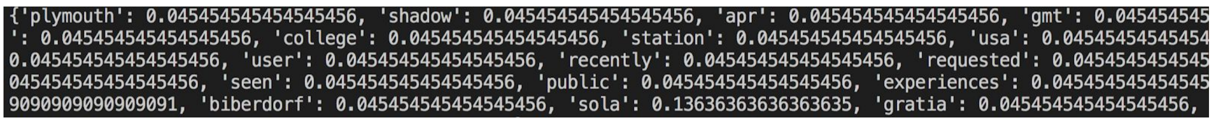
WORKERS / time in seconds - files downloaded		
No workers	2	4
22.99 - 19.997	**14.19** - 19.997	23.32 - Incomplete file numbers

As it is possible to see, parallelizing has reduce the processing time in **38%**.

Exercise 2: Calculate Term Frequency (TF) (5 points)

In the previous step, the cleaned and tokenized documents were save in a folder called "preprocessing". For this exercise, I imported those files to work on. First, there is a function to import the data. now the file structure is folder/documents erasing one for loop in comparison with exercise 1.

A second function called **TF** is in charge for taking the data, iterate through it, take the total number of unique words and finally normalize the counter for each word by the total # of unique words. A example is shown below.



```
{'plymouth': 0.045454545454545456, 'shadow': 0.045454545454545456, 'apr': 0.045454545454545456, 'gmt': 0.045454545
': 0.045454545454545456, 'college': 0.045454545454545456, 'station': 0.045454545454545456, 'usa': 0.04545454545454
0.045454545454545456, 'user': 0.045454545454545456, 'recently': 0.045454545454545456, 'requested': 0.0454545454545
045454545454545456, 'seen': 0.045454545454545456, 'public': 0.045454545454545456, 'experiences': 0.045454545454545
9090909090909091, 'biberdorf': 0.045454545454545456, 'sola': 0.13636363636363635, 'gratia': 0.045454545454545456,
```

The output is a dictionary with each word and the respective TF.

Next, for MPI purposes worker 0, the root, will be the one who calls all the documents, split in chunks to each worker in equally parts and deal with the first batch of data.

Workers **{1 to n}** are in charge of receiving the data, calculating each TF by applying the formula above explained and send back the results to the root.

Finally, the root will merge all the list of dictionaries from the workers and print an output. The experiments are presented below:

TF with 2 files testing	**WORKERS**		
	2	4	**Correct**
Market	0.0128205128	0.0128205128	**True**
Access	0.0128205128	0.0128205128	**True**
Miners	0.0128205128	0.0128205128	**True**
factor	0.025641025	0.025641025	**True**
suffers	0.03846153	0.03846153	**True**

Timing of the process:

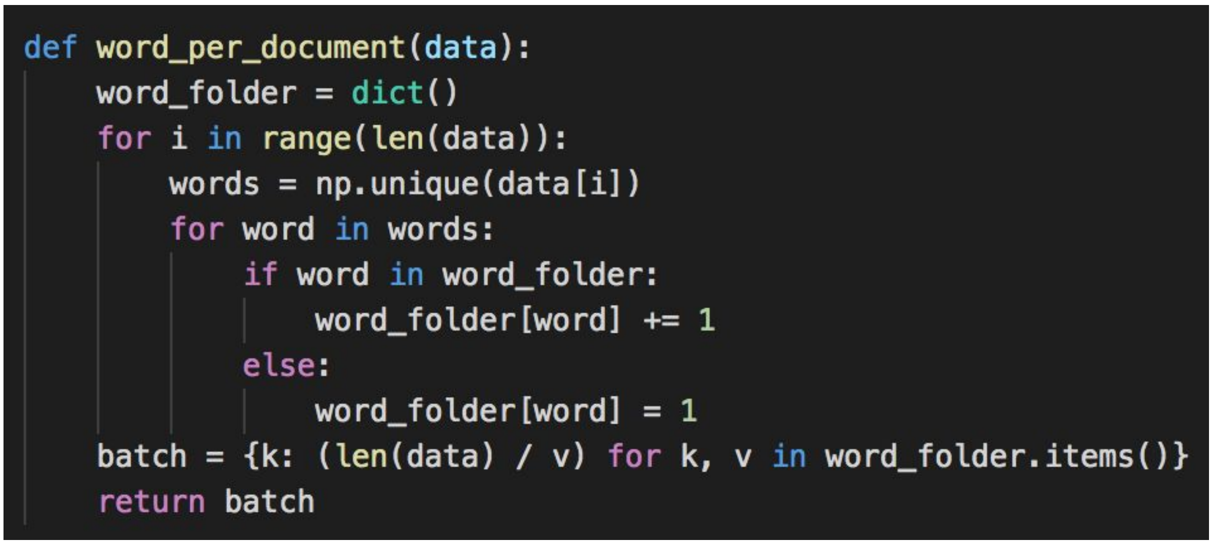
	WORKERS / time in seconds		
No workers	2	4	6
7.55	**6.70**	24.068	23.96 / process failed due to memory in workers.

There is an improvement of execution time of **11%** with two workers.

Exercise 3: Calculate Inverse Document Frequency ((IDF) (5 points)

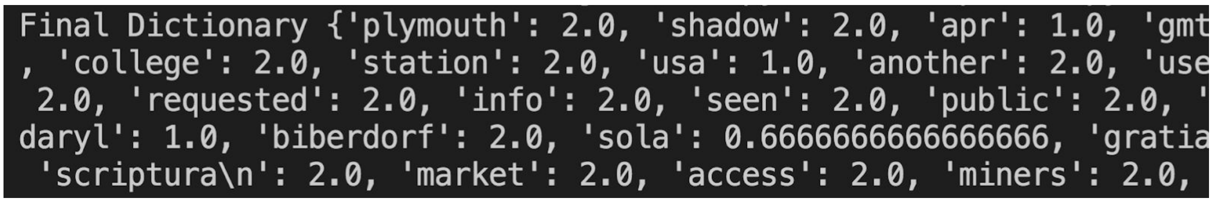
First using the same function of the previous exercise to import the already tokenized data. Followed by a function called **word_per_document** which steps are the following:

- Take the data imported and initialize an empty dictionary.
- Next, find all the unique words in each document.
- Next, add the words that are not yet in the **word_folder** dictionary or sum up 1 if the word is already in the dictionary.
- Finally, applied the formula to the new dictionary representing all documents in the worker: Len(data received by worker) / value of each word



```python
def word_per_document(data):
    word_folder = dict()
    for i in range(len(data)):
        words = np.unique(data[i])
        for word in words:
            if word in word_folder:
                word_folder[word] += 1
            else:
                word_folder[word] = 1
    batch = {k: (len(data) / v) for k, v in word_folder.items()}
    return batch
```

- Showing a partial results from the above function:



```
Final Dictionary {'plymouth': 2.0, 'shadow': 2.0, 'apr': 1.0, 'gmt
, 'college': 2.0, 'station': 2.0, 'usa': 1.0, 'another': 2.0, 'use
 2.0, 'requested': 2.0, 'info': 2.0, 'seen': 2.0, 'public': 2.0, '
daryl': 1.0, 'biberdorf': 2.0, 'sola': 0.6666666666666666, 'gratia
 'scriptura\n': 2.0, 'market': 2.0, 'access': 2.0, 'miners': 2.0,
```

MPI

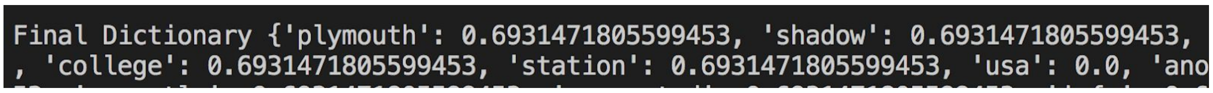
Initializing parallelization structure worker 0 as the root receives all the documents. It is in charge of splitting all the documents in equal parts in the following manner:

- p = Divide the number of documents by the number of workers.
- for all workers (except root) send p documents to each worker.
- Worker 0 will manage the first batch of data.
- Each **worker** will call the function **word_document** and return a dictionary with the partial $\frac{|C|}{DF}$ results for that batch of information.
- The root recollects the information from all workers and executes a final operation:
    - Apply the **log** to the resultant values gathered from different workers.

The decision of applying the “log” in the root instead in each worker is because it is almost impossible to take the right log if applied in the workers and then merge them together.

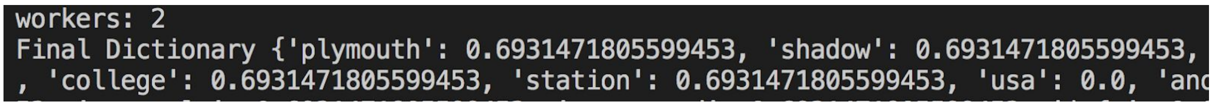
Making an output comparison between the exercise with no workers and 2 workers the result for the first words return exactly the same IDF as shown below.

NO WORKERS:



```
Final Dictionary {'plymouth': 0.6931471805599453, 'shadow': 0.6931471805599453,
, 'college': 0.6931471805599453, 'station': 0.6931471805599453, 'usa': 0.0, 'ano
```

WORKERS = 2



```
workers: 2
Final Dictionary {'plymouth': 0.6931471805599453, 'shadow': 0.6931471805599453,
, 'college': 0.6931471805599453, 'station': 0.6931471805599453, 'usa': 0.0, 'anc
```

Finally, the result of experiments are the following:

NORMAL RUN	**WORKERS /** time in seconds	
No workers	2	4
11.593171	**10.044829**	6.84 but stop running due to share memory initialization.

There is a small improvement in the processing time by parallelizing: an improvement of **14%.**

Exercise 4: Calculate Term Frequency Inverse Document Frequency (TF-IDF) scores (5 points)

A function to import the tokenized documents is created. Next, the root will be the one who sends batches of data to the remaining workers. Each worker will accomplish 2 tasks: first get the TF for each document and secondly it will get the IDF for all the documents that each has received from the root. Similarly, and due to the workers of the laptop, worker 0 will process the first batch of documents.

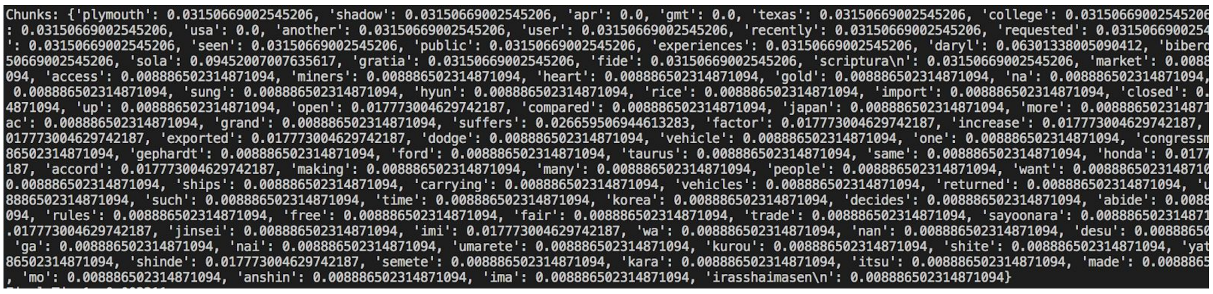
Finally, worker 0 will merge the data from all the workers, measure the final IDF and multiply with each token TF. A final dictionary will be outputted. I chose worker 0 to manage the final activities because of the log function in the IDF. It is not possible to get the right value if the log is applied inside the workers. Additionally, for measuring the global TF-IDF it is necessary to get the IDF of all documents.

Finally, the result of experiments are the following:

	WORKERS	
No workers	2	4
11.2793450355	6.86121	9.483646 / Experience performance degradation.

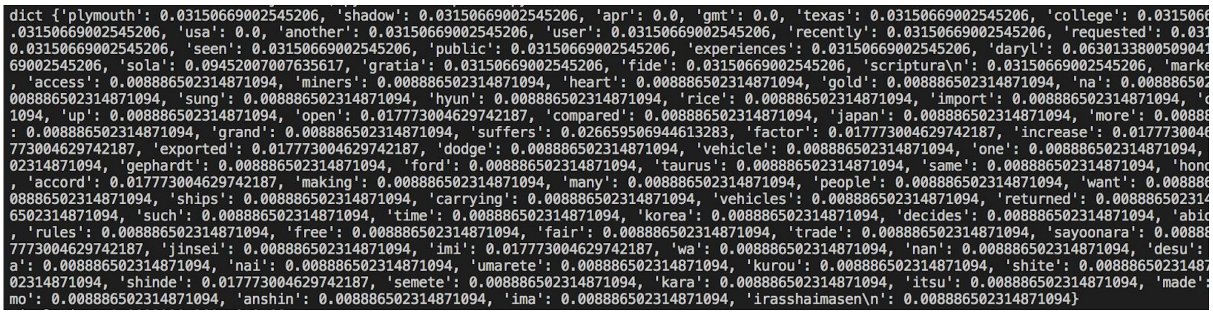
Checking if the outputs are the same. With just 2 files I run a test checking the accuracy of the algorithm with no workers and two workers.

NO WORKERS:



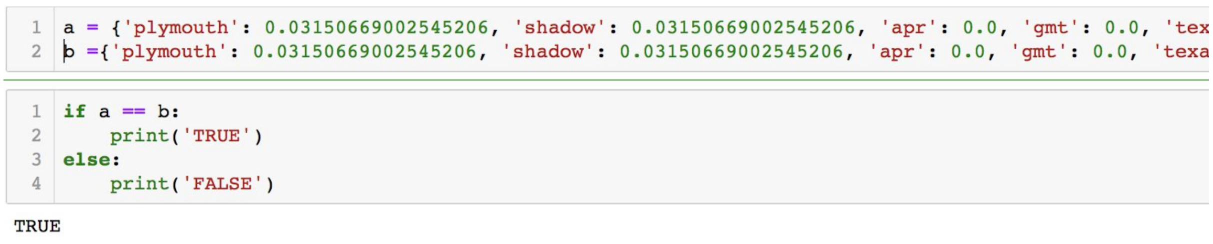
```
Chunks: {'plymouth': 0.03150669002545206, 'shadow': 0.03150669002545206, 'apr': 0.0, 'gmt': 0.0, 'texas': 0.03150669002545206, 'college': 0.03150669002545206
: 0.03150669002545206, 'usa': 0.0, 'another': 0.03150669002545206, 'user': 0.03150669002545206, 'recently': 0.03150669002545206, 'requested': 0.0315066900254
': 0.03150669002545206, 'seen': 0.03150669002545206, 'public': 0.03150669002545206, 'experiences': 0.03150669002545206, 'daryl': 0.06301338005090412, 'biberd
50669002545206, 'sola': 0.09452007007635617, 'gratia': 0.03150669002545206, 'fide': 0.03150669002545206, 'scriptura\n': 0.03150669002545206, 'market': 0.0088
094, 'access': 0.008886502314871094, 'miners': 0.008886502314871094, 'heart': 0.008886502314871094, 'gold': 0.008886502314871094, 'na': 0.008886502314871094,
 0.008886502314871094, 'sung': 0.008886502314871094, 'hyun': 0.008886502314871094, 'rice': 0.008886502314871094, 'import': 0.008886502314871094, 'closed': 0.
4871094, 'up': 0.008886502314871094, 'open': 0.017773004629742187, 'compared': 0.008886502314871094, 'japan': 0.008886502314871094, 'more': 0.008886502314871
ac': 0.008886502314871094, 'grand': 0.008886502314871094, 'suffers': 0.026659506944613283, 'factor': 0.017773004629742187, 'increase': 0.017773004629742187,
017773004629742187, 'exported': 0.017773004629742187, 'dodge': 0.008886502314871094, 'vehicle': 0.008886502314871094, 'one': 0.008886502314871094, 'congressm
86502314871094, 'gephardt': 0.008886502314871094, 'ford': 0.008886502314871094, 'taurus': 0.008886502314871094, 'same': 0.008886502314871094, 'honda': 0.0177
187, 'accord': 0.017773004629742187, 'making': 0.008886502314871094, 'many': 0.008886502314871094, 'people': 0.008886502314871094, 'want': 0.0088865023148710
0.008886502314871094, 'ships': 0.008886502314871094, 'carrying': 0.008886502314871094, 'vehicles': 0.008886502314871094, 'returned': 0.008886502314871094, 'u
8886502314871094, 'such': 0.008886502314871094, 'time': 0.008886502314871094, 'korea': 0.008886502314871094, 'decides': 0.008886502314871094, 'abide': 0.0088
094, 'rules': 0.008886502314871094, 'free': 0.008886502314871094, 'fair': 0.008886502314871094, 'trade': 0.008886502314871094, 'sayoonara': 0.008886502314871
.017773004629742187, 'jinsei': 0.008886502314871094, 'imi': 0.017773004629742187, 'wa': 0.008886502314871094, 'nan': 0.008886502314871094, 'desu': 0.00888650
 'ga': 0.008886502314871094, 'nai': 0.008886502314871094, 'umarete': 0.008886502314871094, 'kurou': 0.008886502314871094, 'shite': 0.008886502314871094, 'yat
86502314871094, 'shinde': 0.017773004629742187, 'semete': 0.008886502314871094, 'kara': 0.008886502314871094, 'itsu': 0.008886502314871094, 'made': 0.0088865
, 'mo': 0.008886502314871094, 'anshin': 0.008886502314871094, 'ima': 0.008886502314871094, 'irasshaimasen\n': 0.008886502314871094}
```

2 WORKERS:



```
dict {'plymouth': 0.03150669002545206, 'shadow': 0.03150669002545206, 'apr': 0.0, 'gmt': 0.0, 'texas': 0.03150669002545206, 'college': 0.0315066
.03150669002545206, 'usa': 0.0, 'another': 0.03150669002545206, 'user': 0.03150669002545206, 'recently': 0.03150669002545206, 'requested': 0.031
0.03150669002545206, 'seen': 0.03150669002545206, 'public': 0.03150669002545206, 'experiences': 0.03150669002545206, 'daryl': 0.0630133800509041
69002545206, 'sola': 0.09452007007635617, 'gratia': 0.03150669002545206, 'fide': 0.03150669002545206, 'scriptura\n': 0.03150669002545206, 'marke
, 'access': 0.008886502314871094, 'miners': 0.008886502314871094, 'heart': 0.008886502314871094, 'gold': 0.008886502314871094, 'na': 0.008886502
008886502314871094, 'sung': 0.008886502314871094, 'hyun': 0.008886502314871094, 'rice': 0.008886502314871094, 'import': 0.008886502314871094, 'c
1094, 'up': 0.008886502314871094, 'open': 0.017773004629742187, 'compared': 0.008886502314871094, 'japan': 0.008886502314871094, 'more': 0.00888
: 0.008886502314871094, 'grand': 0.008886502314871094, 'suffers': 0.026659506944613283, 'factor': 0.017773004629742187, 'increase': 0.0177730046
773004629742187, 'exported': 0.017773004629742187, 'dodge': 0.008886502314871094, 'vehicle': 0.008886502314871094, 'one': 0.008886502314871094,
02314871094, 'gephardt': 0.008886502314871094, 'ford': 0.008886502314871094, 'taurus': 0.008886502314871094, 'same': 0.008886502314871094, 'hond
, 'accord': 0.017773004629742187, 'making': 0.008886502314871094, 'many': 0.008886502314871094, 'people': 0.008886502314871094, 'want': 0.008886
08886502314871094, 'ships': 0.008886502314871094, 'carrying': 0.008886502314871094, 'vehicles': 0.008886502314871094, 'returned': 0.008886502314
6502314871094, 'such': 0.008886502314871094, 'time': 0.008886502314871094, 'korea': 0.008886502314871094, 'decides': 0.008886502314871094, 'abid
, 'rules': 0.008886502314871094, 'free': 0.008886502314871094, 'fair': 0.008886502314871094, 'trade': 0.008886502314871094, 'sayoonara': 0.00888
7773004629742187, 'jinsei': 0.008886502314871094, 'imi': 0.017773004629742187, 'wa': 0.008886502314871094, 'nan': 0.008886502314871094, 'desu':
a': 0.008886502314871094, 'nai': 0.008886502314871094, 'umarete': 0.008886502314871094, 'kurou': 0.008886502314871094, 'shite': 0.00888650231487
02314871094, 'shinde': 0.017773004629742187, 'semete': 0.008886502314871094, 'kara': 0.008886502314871094, 'itsu': 0.008886502314871094, 'made':
mo': 0.008886502314871094, 'anshin': 0.008886502314871094, 'ima': 0.008886502314871094, 'irasshaimasen\n': 0.008886502314871094}
```

By running a if/else comparison in the outputs in Jupyter notebooks the output is **TRUE**



```python
a = {'plymouth': 0.03150669002545206, 'shadow': 0.03150669002545206, 'apr': 0.0, 'gmt': 0.0, 'tex
b ={'plymouth': 0.03150669002545206, 'shadow': 0.03150669002545206, 'apr': 0.0, 'gmt': 0.0, 'texa
```

```python
if a == b:
    print('TRUE')
else:
    print('FALSE')
```

```
TRUE
```

References:

- https://stackoverflow.com/questions/20906474/import-multiple-csv-files-into-pandas-and-concatenate-into-one-dataframe
- https://stackoverflow.com/questions/30964577/divide-each-python-dictionary-value-by-total-value
- https://stackoverflow.com/questions/53107463/writing-items-from-list-into-different-files-python
- https://stackoverflow.com/questions/53107463/writing-items-from-list-into-different-files-python
- http://www.textfixer.com/tutorials/common-english-words