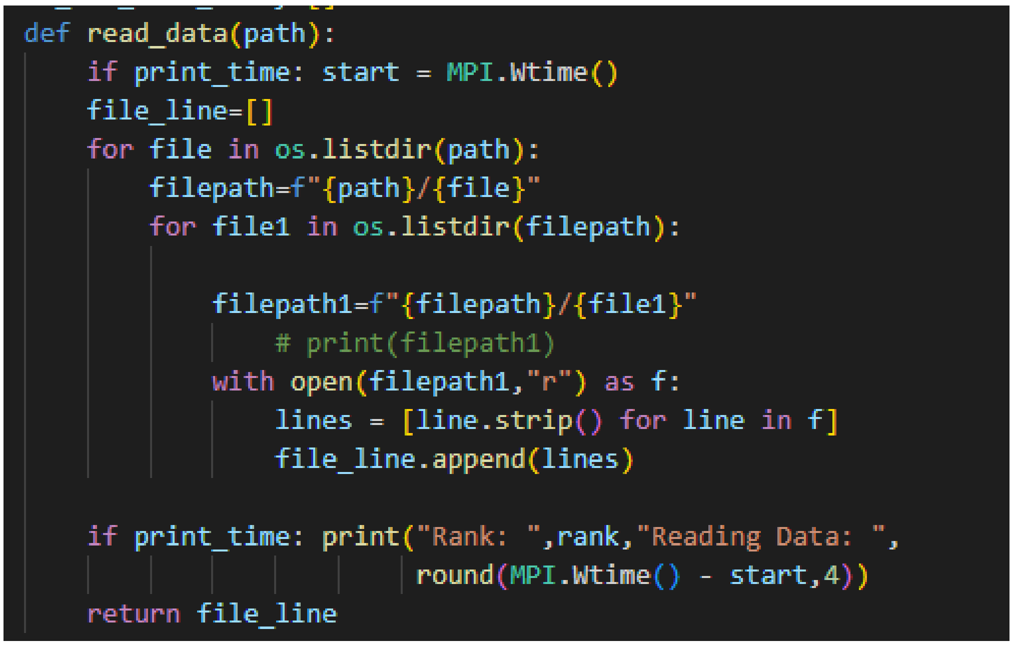
Sharon Laurance

Complex Data Lab: Processing Text Data in a Distributed Setting

Exercise 1: Data cleaning and text tokenization

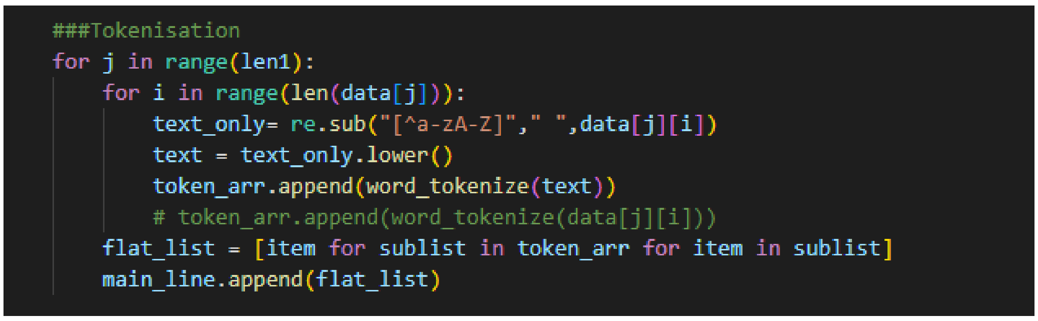
1. Cleaning: Remove all punctuations, numbers and common English stopwords.
2. Tokenize: Tokenize your documents so it is easy to process in the next task i.e. Tokenize words and output as a comma separated document.

First data is read from the folder. File structure is parsed using os.listdir(). File is opened using with open() function. Entire data is returned.



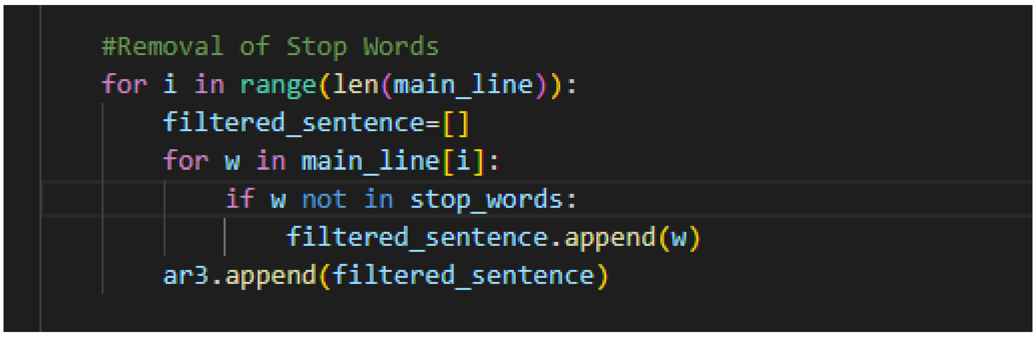
```python
def read_data(path):
    if print_time: start = MPI.Wtime()
    file_line=[]
    for file in os.listdir(path):
        filepath=f"{path}/{file}"
        for file1 in os.listdir(filepath):

            filepath1=f"{filepath}/{file1}"
                # print(filepath1)
            with open(filepath1,"r") as f:
                lines = [line.strip() for line in f]
                file_line.append(lines)

    if print_time: print("Rank: ",rank,"Reading Data: ",
                            round(MPI.Wtime() - start,4))
    return file_line
```

Tokenization and removal of Stop words both take place in clean_data() function. Along with this, punctuation is removed and text is converted to lower case for faster processing of data.



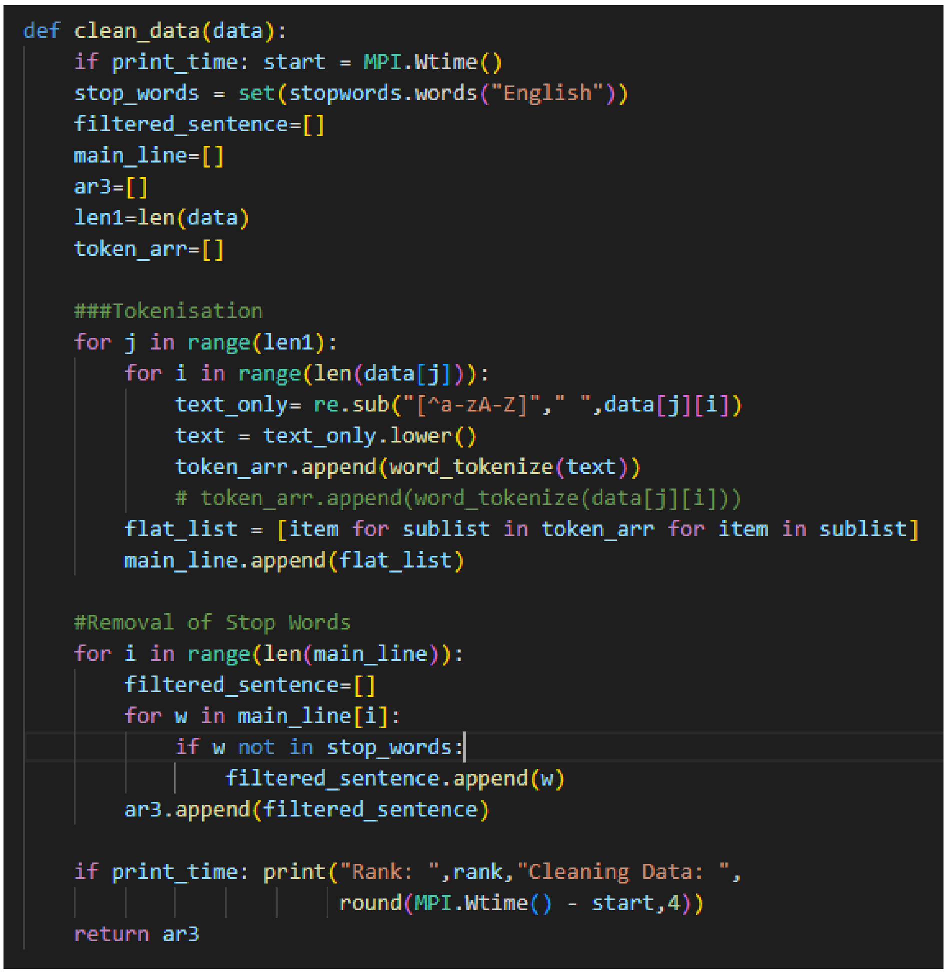
```python
###Tokenisation
for j in range(len1):
    for i in range(len(data[j])):
        text_only= re.sub("[^a-zA-Z]"," ",data[j][i])
        text = text_only.lower()
        token_arr.append(word_tokenize(text))
        # token_arr.append(word_tokenize(data[j][i]))
    flat_list = [item for sublist in token_arr for item in sublist]
    main_line.append(flat_list)
```

For removal of Stop Words, stopwords are used from nltk.corpus. This provides set of documents that are clean and tokenized.



```python
#Removal of Stop Words
for i in range(len(main_line)):
    filtered_sentence=[]
    for w in main_line[i]:
        if w not in stop_words:
            filtered_sentence.append(w)
    ar3.append(filtered_sentence)
```

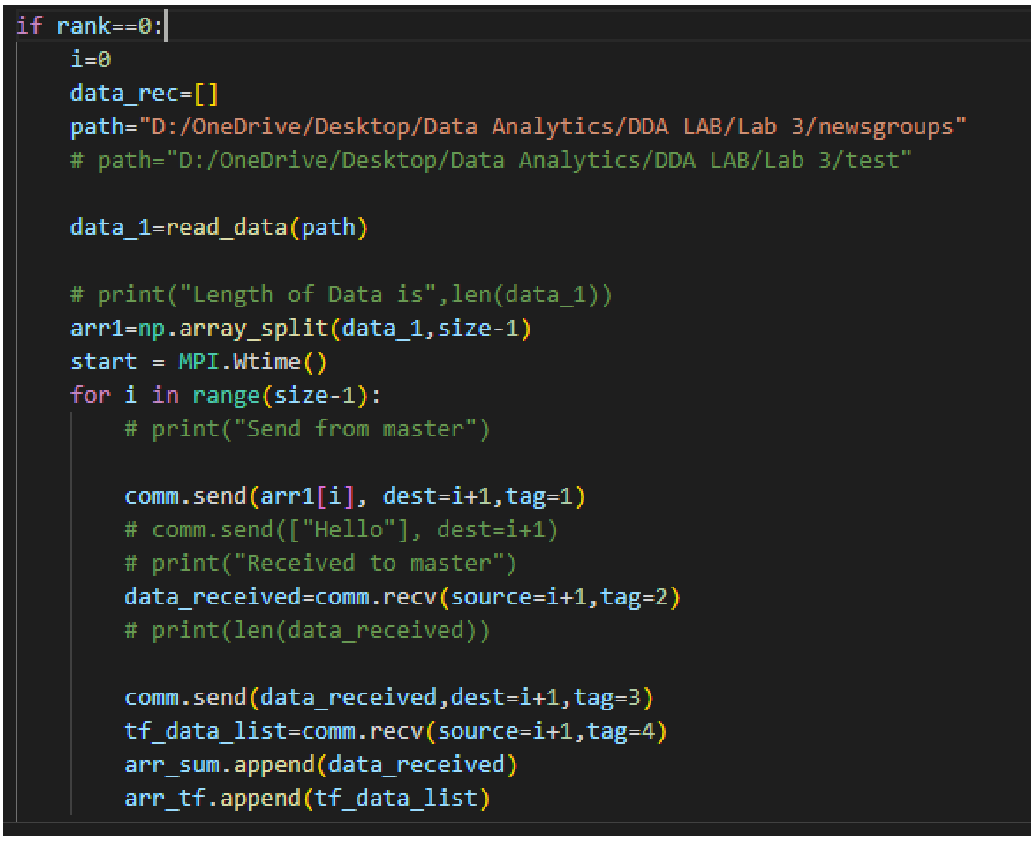
Entire function is given below.



```python
def clean_data(data):
    if print_time: start = MPI.Wtime()
    stop_words = set(stopwords.words("English"))
    filtered_sentence=[]
    main_line=[]
    ar3=[]
    len1=len(data)
    token_arr=[]

    ###Tokenisation
    for j in range(len1):
        for i in range(len(data[j])):
            text_only= re.sub("[^a-zA-Z]"," ",data[j][i])
            text = text_only.lower()
            token_arr.append(word_tokenize(text))
            # token_arr.append(word_tokenize(data[j][i]))
        flat_list = [item for sublist in token_arr for item in sublist]
        main_line.append(flat_list)

    #Removal of Stop Words
    for i in range(len(main_line)):
        filtered_sentence=[]
        for w in main_line[i]:
            if w not in stop_words:
                filtered_sentence.append(w)
        ar3.append(filtered_sentence)

    if print_time: print("Rank: ",rank,"Cleaning Data: ",
                          round(MPI.Wtime() - start,4))
    return ar3
```

Parallelization MPI Framework:

- When the rank is 0, data is read from the dataset .
- The Data is split into parts using np.array_split() based on the number of workers. (i.e for 8 workers data is split in 8 parts.)
- Parts of Data is sent to the worker and this is coordinated with the help of tag.
- Here Point-to-Point Communication is used. Functions send() and receive() have been used. Capital (Send and Receive) are not used to avoid buffers being used and increasing time consumed.
- The Worker process the data and performs tokenization and removal of stop words.

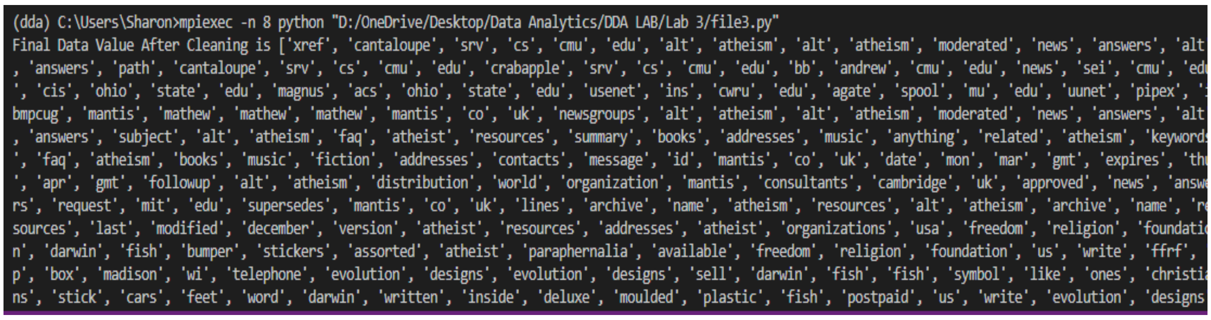


```python
if rank==0:
    i=0
    data_rec=[]
    path="D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 3/newsgroups"
    # path="D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 3/test"

    data_1=read_data(path)

    # print("Length of Data is",len(data_1))
    arr1=np.array_split(data_1,size-1)
    start = MPI.Wtime()
    for i in range(size-1):
        # print("Send from master")

        comm.send(arr1[i], dest=i+1,tag=1)
        # comm.send(["Hello"], dest=i+1)
        # print("Received to master")
        data_received=comm.recv(source=i+1,tag=2)
        # print(len(data_received))

        comm.send(data_received,dest=i+1,tag=3)
        tf_data_list=comm.recv(source=i+1,tag=4)
        arr_sum.append(data_received)
        arr_tf.append(tf_data_list)
```

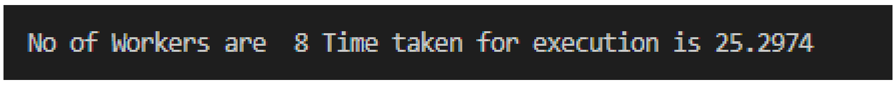
When run with P=2,4 this takes a very long time for processing and ultimately gives memory error. So I have used higher number of workers. Here P=8,16,32 for processing. We see time is decreasing as we increase number of workers.

<u>Output</u>

Number of Workers	P=8	P=16	P=32
Time taken for execution	25.2974 sec	17 sec	14.7532 sec



```
(dda) C:\Users\Sharon>mpiexec -n 8 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 3/file3.py"
Final Data Value After Cleaning is ['xref', 'cantaloupe', 'srv', 'cs', 'cmu', 'edu', 'alt', 'atheism', 'alt', 'atheism', 'moderated', 'news', 'answers', 'alt
, 'answers', 'path', 'cantaloupe', 'srv', 'cs', 'cmu', 'edu', 'crabapple', 'srv', 'cs', 'cmu', 'edu', 'bb', 'andrew', 'cmu', 'edu', 'news', 'sei', 'cmu', 'edu
', 'cis', 'ohio', 'state', 'edu', 'magnus', 'acs', 'ohio', 'state', 'edu', 'usenet', 'ins', 'cwru', 'edu', 'agate', 'spool', 'mu', 'edu', 'uunet', 'pipex', '
bmpcug', 'mantis', 'mathew', 'mathew', 'mathew', 'mantis', 'co', 'uk', 'newsgroups', 'alt', 'atheism', 'alt', 'atheism', 'moderated', 'news', 'answers', 'alt
, 'answers', 'subject', 'alt', 'atheism', 'faq', 'atheist', 'resources', 'summary', 'books', 'addresses', 'music', 'anything', 'related', 'atheism', 'keywords
', 'faq', 'atheism', 'books', 'music', 'fiction', 'addresses', 'contacts', 'message', 'id', 'mantis', 'co', 'uk', 'date', 'mon', 'mar', 'gmt', 'expires', 'thu
', 'apr', 'gmt', 'followup', 'alt', 'atheism', 'distribution', 'world', 'organization', 'mantis', 'consultants', 'cambridge', 'uk', 'approved', 'news', 'answe
rs', 'request', 'mit', 'edu', 'supersedes', 'mantis', 'co', 'uk', 'lines', 'archive', 'name', 'atheism', 'resources', 'alt', 'atheism', 'archive', 'name', 're
sources', 'last', 'modified', 'december', 'version', 'atheist', 'resources', 'addresses', 'atheist', 'organizations', 'usa', 'freedom', 'religion', 'foundatio
n', 'darwin', 'fish', 'bumper', 'stickers', 'assorted', 'atheist', 'paraphernalia', 'available', 'freedom', 'religion', 'foundation', 'us', 'write', 'ffrf', 
p', 'box', 'madison', 'wi', 'telephone', 'evolution', 'designs', 'evolution', 'designs', 'sell', 'darwin', 'fish', 'fish', 'symbol', 'like', 'ones', 'christia
ns', 'stick', 'cars', 'feet', 'word', 'darwin', 'written', 'inside', 'deluxe', 'moulded', 'plastic', 'fish', 'postpaid', 'us', 'write', 'evolution', 'designs
```



```
No of Workers are  8 Time taken for execution is 25.2974
```

The time taken is for 20newsgroup Dataset.

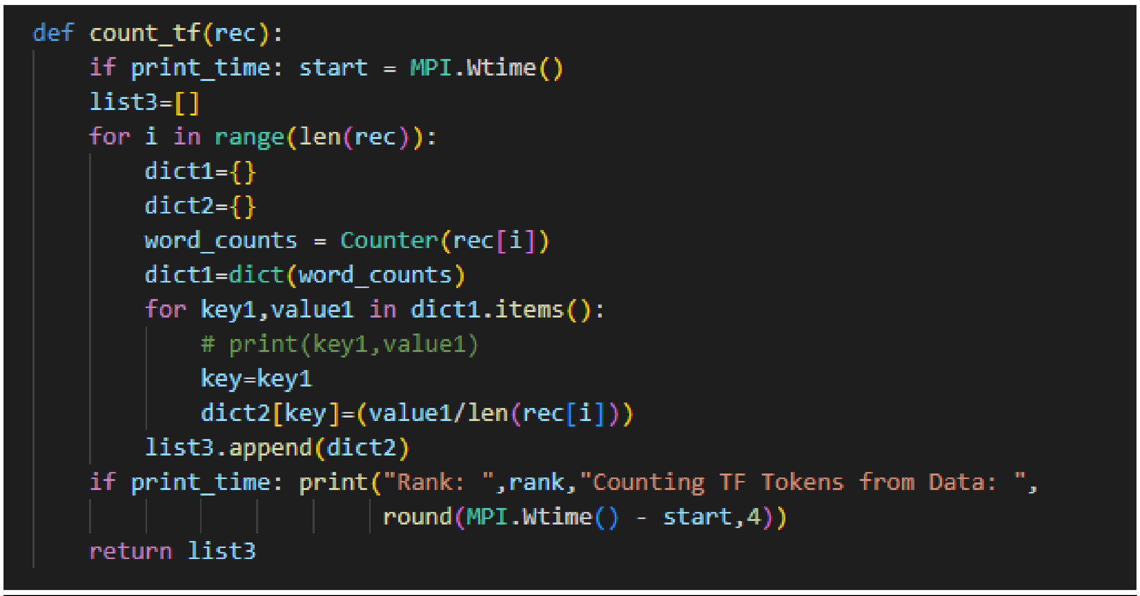
Exercise 2: Calculate Term Frequency (TF)

The Term Frequency (TF) is calculated by counting the number of times a token occurs in the document. This TF score is relative to a specific document, therefore you need to normalized it by dividing with the total number of tokens appearing in the document. A normalized TF score for a specific token t in a document d can be calculated as,

$$\mathrm{TF}(t,d) = \frac{n^d(t)}{\sum_{t' \in d} n^d(t')} ,$$

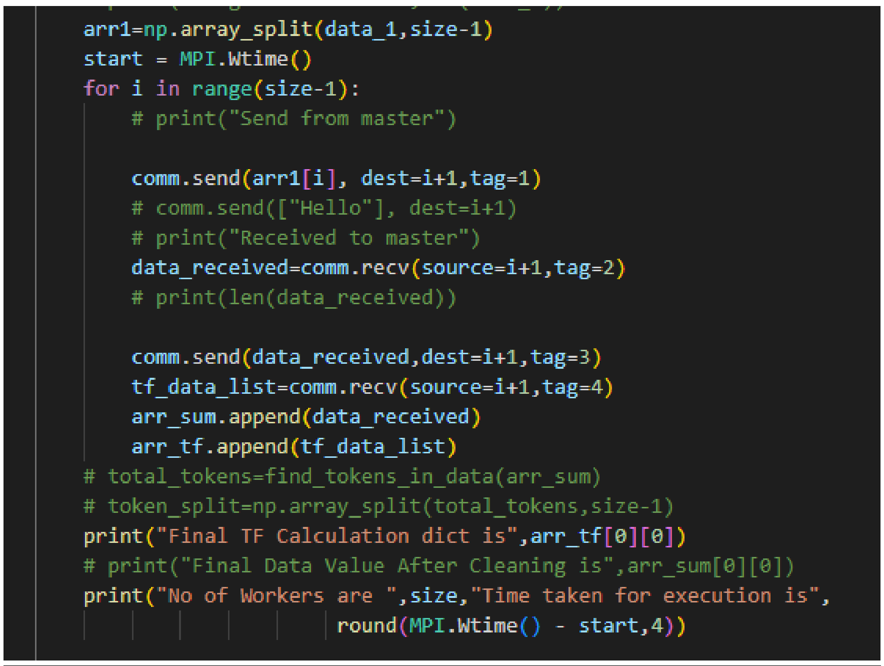
where $n^d(t)$ is the number of times a token t appears in a document d and |d| is the total number of tokens in the document d.

Program using count_tf() function for calculation of Term Frequency.



```
def count_tf(rec):
    if print_time: start = MPI.Wtime()
    list3=[]
    for i in range(len(rec)):
        dict1={}
        dict2={}
        word_counts = Counter(rec[i])
        dict1=dict(word_counts)
        for key1,value1 in dict1.items():
            # print(key1,value1)
            key=key1
            dict2[key]=(value1/len(rec[i]))
        list3.append(dict2)
    if print_time: print("Rank: ",rank,"Counting TF Tokens from Data: ",
                         round(MPI.Wtime() - start,4))
    return list3
```

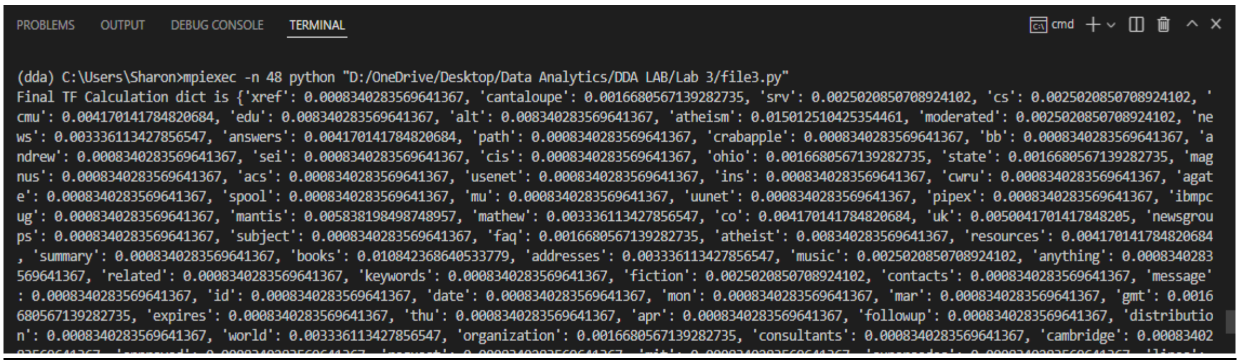
Code below shows how master splits the data, sends it to different words and after receiving the data appends it to array arr_tf.



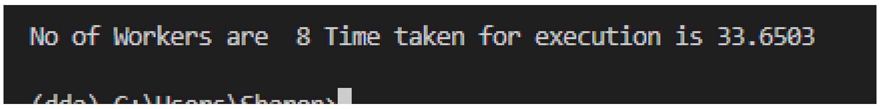
```python
    arr1=np.array_split(data_1,size-1)
    start = MPI.Wtime()
    for i in range(size-1):
        # print("Send from master")

        comm.send(arr1[i], dest=i+1,tag=1)
        # comm.send(["Hello"], dest=i+1)
        # print("Received to master")
        data_received=comm.recv(source=i+1,tag=2)
        # print(len(data_received))

        comm.send(data_received,dest=i+1,tag=3)
        tf_data_list=comm.recv(source=i+1,tag=4)
        arr_sum.append(data_received)
        arr_tf.append(tf_data_list)
    # total_tokens=find_tokens_in_data(arr_sum)
    # token_split=np.array_split(total_tokens,size-1)
    print("Final TF Calculation dict is",arr_tf[0][0])
    # print("Final Data Value After Cleaning is",arr_sum[0][0])
    print("No of Workers are ",size,"Time taken for execution is",
                         round(MPI.Wtime() - start,4))
```

When run with P=2,4 this takes a very long time for processing and ultimately gives memory error. So I have used higher number of workers. Here P=8,16,32 for processing.

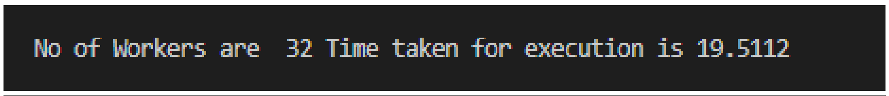
Output



```
(dda) C:\Users\Sharon>mpiexec -n 48 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 3/file3.py"
Final TF Calculation dict is {'xref': 0.0008340283569641367, 'cantaloupe': 0.0016680567139282735, 'srv': 0.0025020850708924102, 'cs': 0.0025020850708924102, '
cmu': 0.004170141784820684, 'edu': 0.008340283569641367, 'alt': 0.008340283569641367, 'atheism': 0.015012510425354461, 'moderated': 0.0025020850708924102, 'ne
ws': 0.003336113427856547, 'answers': 0.004170141784820684, 'path': 0.0008340283569641367, 'crabapple': 0.0008340283569641367, 'bb': 0.0008340283569641367, 'a
ndrew': 0.0008340283569641367, 'sei': 0.0008340283569641367, 'cis': 0.0008340283569641367, 'ohio': 0.0016680567139282735, 'state': 0.0016680567139282735, 'mag
nus': 0.0008340283569641367, 'acs': 0.0008340283569641367, 'usenet': 0.0008340283569641367, 'ins': 0.0008340283569641367, 'cwru': 0.0008340283569641367, 'agat
e': 0.0008340283569641367, 'spool': 0.0008340283569641367, 'mu': 0.0008340283569641367, 'uunet': 0.0008340283569641367, 'pipex': 0.0008340283569641367, 'ibmpc
ug': 0.0008340283569641367, 'mantis': 0.005838198498748957, 'mathew': 0.003336113427856547, 'co': 0.004170141784820684, 'uk': 0.0050041701417848205, 'newsgrou
ps': 0.0008340283569641367, 'subject': 0.0008340283569641367, 'faq': 0.0016680567139282735, 'atheist': 0.008340283569641367, 'resources': 0.004170141784820684
, 'summary': 0.0008340283569641367, 'books': 0.010842368640533779, 'addresses': 0.003336113427856547, 'music': 0.0025020850708924102, 'anything': 0.0008340283
569641367, 'related': 0.0008340283569641367, 'keywords': 0.0008340283569641367, 'fiction': 0.0025020850708924102, 'contacts': 0.0008340283569641367, 'message'
: 0.0008340283569641367, 'id': 0.0008340283569641367, 'date': 0.0008340283569641367, 'mon': 0.0008340283569641367, 'mar': 0.0008340283569641367, 'gmt': 0.0016
680567139282735, 'expires': 0.0008340283569641367, 'thu': 0.0008340283569641367, 'apr': 0.0008340283569641367, 'followup': 0.0008340283569641367, 'distributio
n': 0.0008340283569641367, 'world': 0.003336113427856547, 'organization': 0.0016680567139282735, 'consultants': 0.0008340283569641367, 'cambridge': 0.00083402
```



```
No of Workers are  8 Time taken for execution is 33.6503

(dda) C:\Users\Sharon>
```



```
No of Workers are  32 Time taken for execution is 19.5112
```

We see time is decreasing as we increase number of workers.

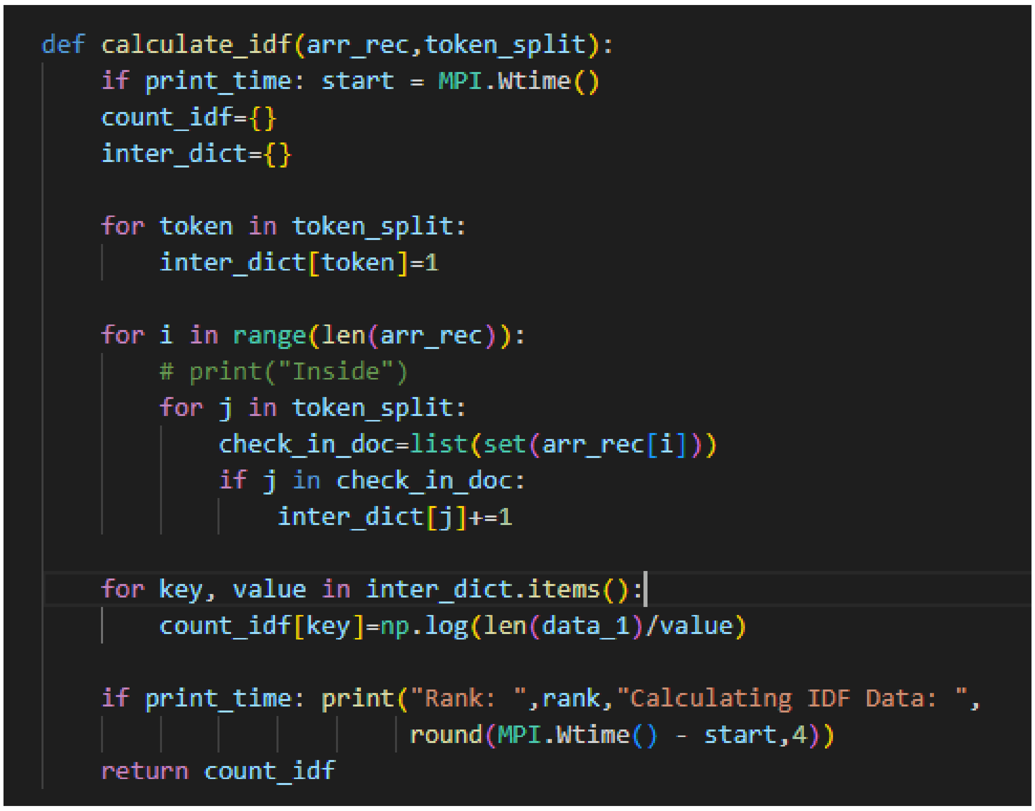
Number of Workers	P=8	P=16	P=32
Time taken for execution	33 sec	25.1997 sec	19.5112 sec

Exercise 3: Calculate Inverse Document Frequency

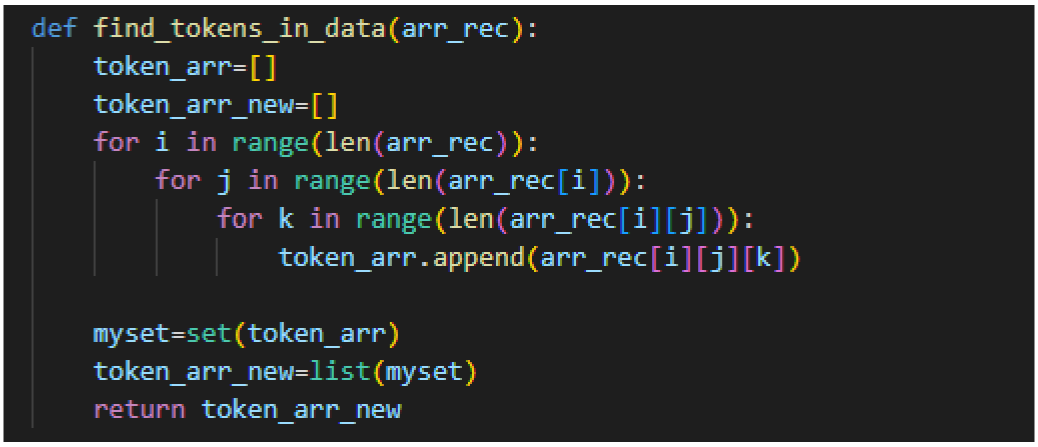
The Inverse Document Frequency ((IDF) is counting the number of documents in the corpus and counting the number of documents that contain a token. The IDF formula is given as

$$\mathrm{IDF}(t) = \log \frac{|C|}{\sum_{d \in C} \mathbb{1}(t,d)} \ ,$$

Program uses calculate_idf function for calculating Inverse Document Frequency.

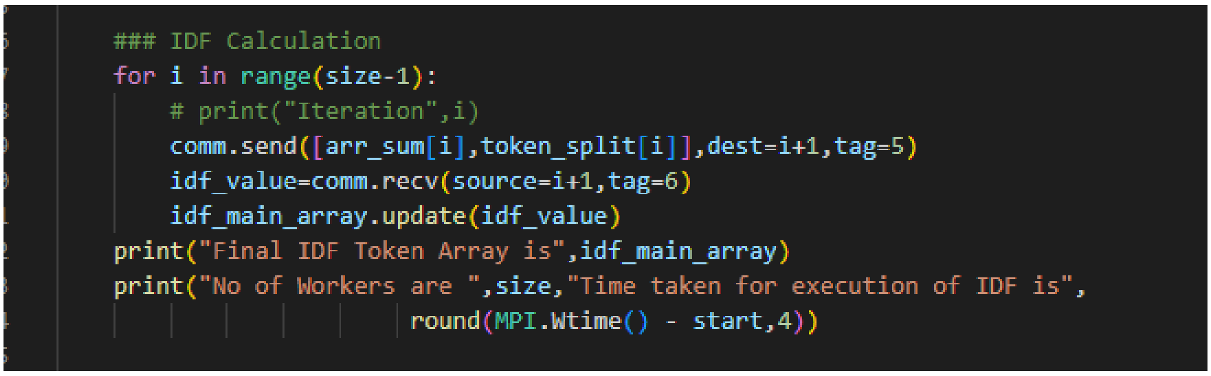


```python
def calculate_idf(arr_rec,token_split):
    if print_time: start = MPI.Wtime()
    count_idf={}
    inter_dict={}

    for token in token_split:
        inter_dict[token]=1

    for i in range(len(arr_rec)):
        # print("Inside")
        for j in token_split:
            check_in_doc=list(set(arr_rec[i]))
            if j in check_in_doc:
                inter_dict[j]+=1

    for key, value in inter_dict.items():
        count_idf[key]=np.log(len(data_1)/value)

    if print_time: print("Rank: ",rank,"Calculating IDF Data: ",
                         round(MPI.Wtime() - start,4))
    return count_idf
```



```python
def find_tokens_in_data(arr_rec):
    token_arr=[]
    token_arr_new=[]
    for i in range(len(arr_rec)):
        for j in range(len(arr_rec[i])):
            for k in range(len(arr_rec[i][j])):
                token_arr.append(arr_rec[i][j][k])

    myset=set(token_arr)
    token_arr_new=list(myset)
    return token_arr_new
```

Code below shows how master splits the data, sends it to different words and after receiving the data updates the Final Dictionary idf_main_array.



```
### IDF Calculation
for i in range(size-1):
    # print("Iteration",i)
    comm.send([arr_sum[i],token_split[i]],dest=i+1,tag=5)
    idf_value=comm.recv(source=i+1,tag=6)
    idf_main_array.update(idf_value)
print("Final IDF Token Array is",idf_main_array)
print("No of Workers are ",size,"Time taken for execution of IDF is",
                    round(MPI.Wtime() - start,4))
```

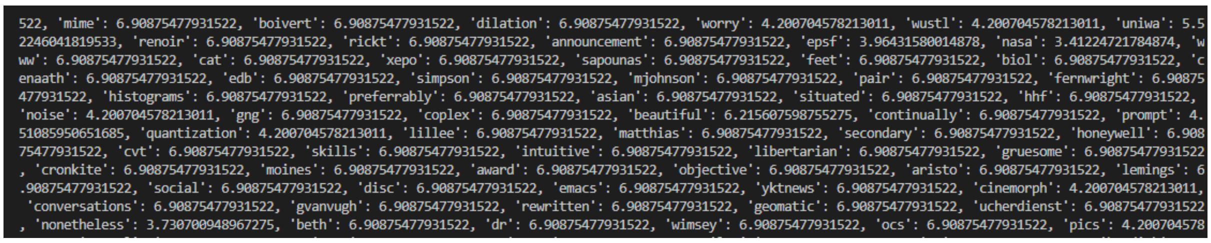
Parallelization MPI Framework:

- For IDF Calculation entire corpus needs to be utilized. For this token data needs to be collected together. Here find_tokens_in_data () function is used to find unique tokens.
- After this the entire token list is split into parts based on the number of workers.
- Then token array and the document array is passed using send().
- Here Point-to-Point Communication is used. Functions send() and receive() have been used. Capital Send and Receive and not used to avoid buffers being used and increasing time consumed.
- The Worker process the data and performs IDF Calculation. For merging the data from different workers, the result gets stored in a dictionary and this dictionary gets updated every time.
- This is the most time consuming process as entire corpus needs to be taken into consideration. All other steps are relatively fast compared to IDF Calculation.

When run with P=2,4 this takes a very long time for processing and ultimately gives memory error. So I have used higher number of workers. Here P=8,16,32 for processing. The time denoted includes above steps (i.e total time for tokenization, Finding TF and Performing IDF Calculation). We see time is decreasing as we increase number of workers.

<u>Output</u>

Number of Workers	P=8	P=16	P=32
Time taken for execution	290 sec	132 sec	113 sec

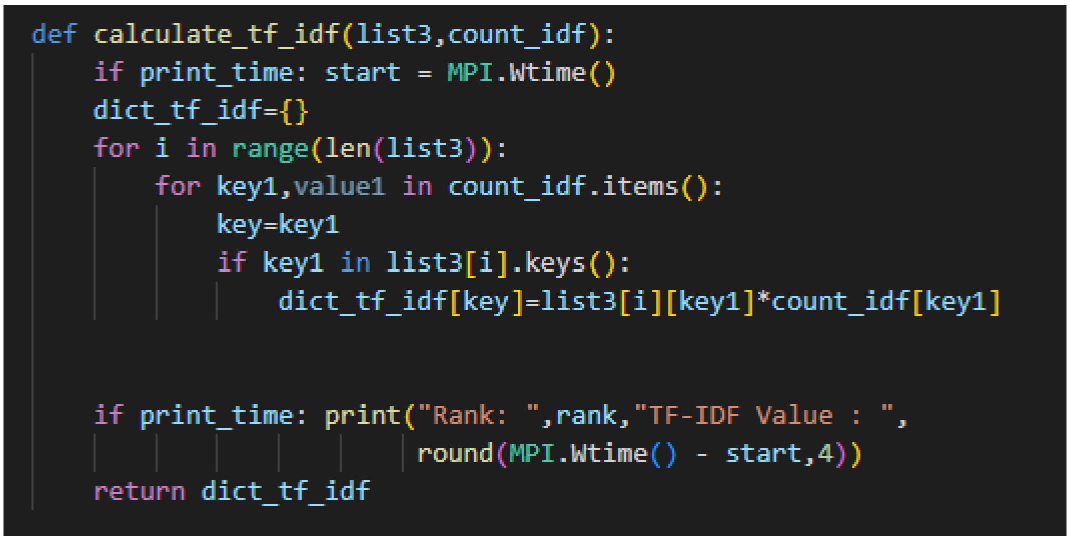


```
522, 'mime': 6.90875477931522, 'boivert': 6.90875477931522, 'dilation': 6.90875477931522, 'worry': 4.200704578213011, 'wustl': 4.200704578213011, 'uniwa': 5.5
2246041819533, 'renoir': 6.90875477931522, 'rickt': 6.90875477931522, 'announcement': 6.90875477931522, 'epsf': 3.96431580014878, 'nasa': 3.41224721784874, 'w
ww': 6.90875477931522, 'cat': 6.90875477931522, 'xepo': 6.90875477931522, 'sapounas': 6.90875477931522, 'feet': 6.90875477931522, 'biol': 6.90875477931522, 'c
enaath': 6.90875477931522, 'edb': 6.90875477931522, 'simpson': 6.90875477931522, 'mjohnson': 6.90875477931522, 'pair': 6.90875477931522, 'fernwright': 6.90875
477931522, 'histograms': 6.90875477931522, 'preferrably': 6.90875477931522, 'asian': 6.90875477931522, 'situated': 6.90875477931522, 'hhf': 6.90875477931522,
'noise': 4.200704578213011, 'gng': 6.90875477931522, 'coplex': 6.90875477931522, 'beautiful': 6.215607598755275, 'continually': 6.90875477931522, 'prompt': 4.
5108595065l685, 'quantization': 4.200704578213011, 'lillee': 6.90875477931522, 'matthias': 6.90875477931522, 'secondary': 6.90875477931522, 'honeywell': 6.908
75477931522, 'cvt': 6.90875477931522, 'skills': 6.90875477931522, 'intuitive': 6.90875477931522, 'libertarian': 6.90875477931522, 'gruesome': 6.90875477931522
, 'cronkite': 6.90875477931522, 'moines': 6.90875477931522, 'award': 6.90875477931522, 'objective': 6.90875477931522, 'aristo': 6.90875477931522, 'lemings': 6
.90875477931522, 'social': 6.90875477931522, 'disc': 6.90875477931522, 'emacs': 6.90875477931522, 'yktnews': 6.90875477931522, 'cinemorph': 4.200704578213011,
 'conversations': 6.90875477931522, 'gvanvugh': 6.90875477931522, 'rewritten': 6.90875477931522, 'geomatic': 6.90875477931522, 'ucherdienst': 6.90875477931522
, 'nonetheless': 3.730700948967275, 'beth': 6.90875477931522, 'dr': 6.90875477931522, 'wimsey': 6.90875477931522, 'ocs': 6.90875477931522, 'pics': 4.200704578
```

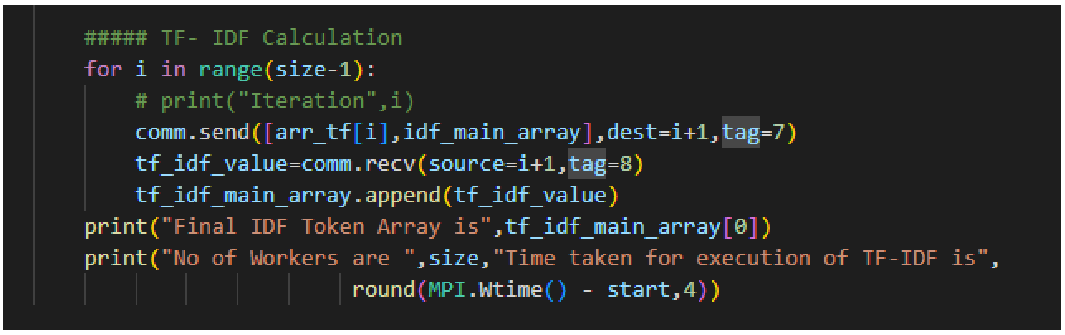
Exercise 4: Calculate Term Frequency Inverse Document Frequency (TF-IDF) scores

TF-IDF (t,d) for a given token t and a document d in the corpus C is the product of TF (t,d) and IDF (t), which is represented as

$$TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)$$

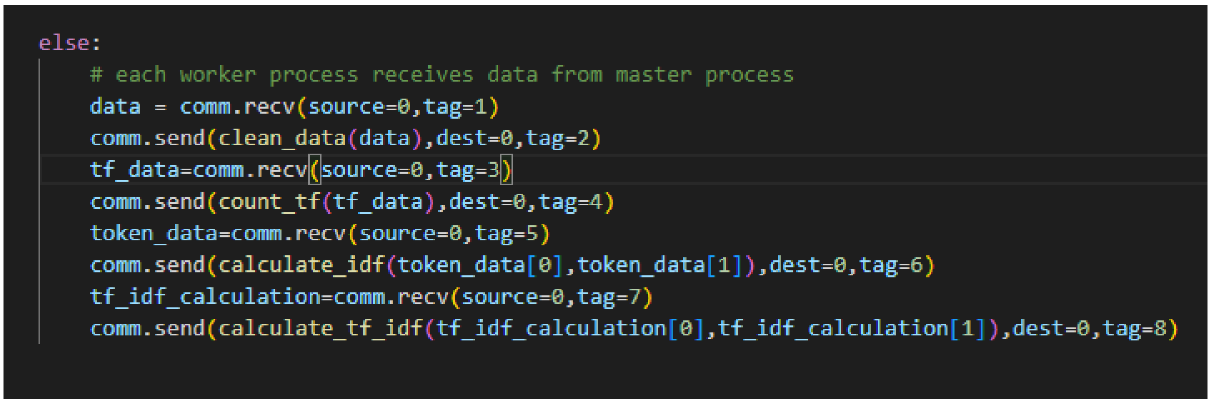


```
def calculate_tf_idf(list3,count_idf):
    if print_time: start = MPI.Wtime()
    dict_tf_idf={}
    for i in range(len(list3)):
        for key1,value1 in count_idf.items():
            key=key1
            if key1 in list3[i].keys():
                dict_tf_idf[key]=list3[i][key1]*count_idf[key1]


    if print_time: print("Rank: ",rank,"TF-IDF Value : ",
                                round(MPI.Wtime() - start,4))
    return dict_tf_idf
```



```
    ##### TF- IDF Calculation
    for i in range(size-1):
        # print("Iteration",i)
        comm.send([arr_tf[i],idf_main_array],dest=i+1,tag=7)
        tf_idf_value=comm.recv(source=i+1,tag=8)
        tf_idf_main_array.append(tf_idf_value)
    print("Final IDF Token Array is",tf_idf_main_array[0])
    print("No of Workers are ",size,"Time taken for execution of TF-IDF is",
                        round(MPI.Wtime() - start,4))
```

This is code for worker processes. Worker processes are coordinated by tag to ensure they send and receive the right communication.



```
else:
    # each worker process receives data from master process
    data = comm.recv(source=0,tag=1)
    comm.send(clean_data(data),dest=0,tag=2)
    tf_data=comm.recv(source=0,tag=3)
    comm.send(count_tf(tf_data),dest=0,tag=4)
    token_data=comm.recv(source=0,tag=5)
    comm.send(calculate_idf(token_data[0],token_data[1]),dest=0,tag=6)
    tf_idf_calculation=comm.recv(source=0,tag=7)
    comm.send(calculate_tf_idf(tf_idf_calculation[0],tf_idf_calculation[1]),dest=0,tag=8)
```

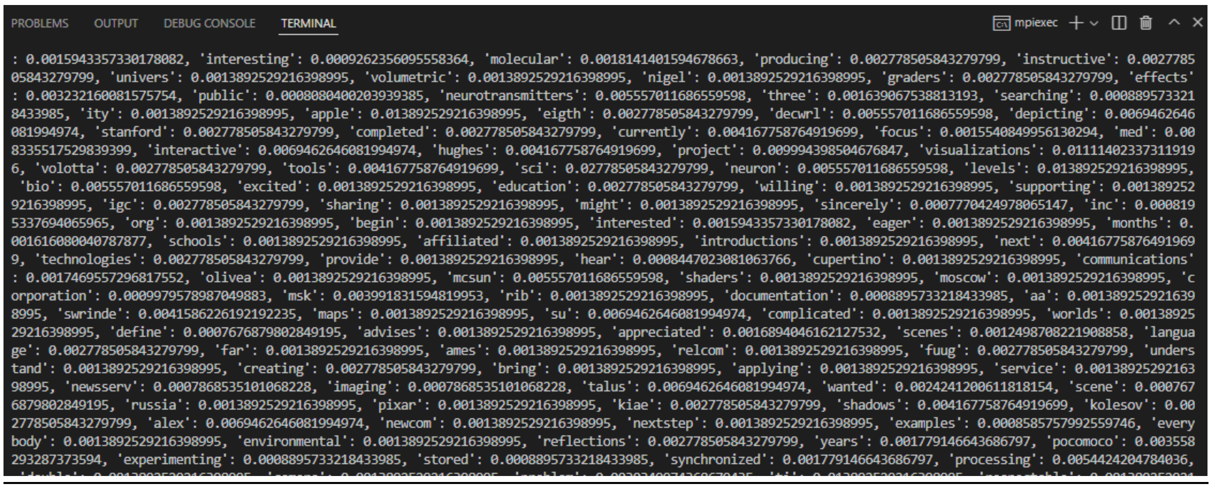
Parallelization MPI Framework:

- When the rank is 0, data which split into parts is send to workers. This includes the IDF Dictionary and TF list.

- One part of Data is sent to the worker and this is coordinated with the help of tag.
- Here Point-to-Point Communication is used. Functions send() and receive() have been used. Capital Send and Receive and not used to avoid buffers being used and increasing time consumed.
- The Worker process the data and performs TF-IDF Calculation which is done for each word . TF-IDF for a word is calculated by multiplying the TF value for the word with the IDF value.

When run with P=2,4 this takes a very long time for processing and ultimately gives memory error. So I have used higher number of workers. Here P=8,16,32 for processing. Time here is for the entire pipeline of distributed code.

Output:



```
: 0.0015943357330178082, 'interesting': 0.0009262356095558364, 'molecular': 0.0018141401594678663, 'producing': 0.002778505843279799, 'instructive': 0.0027785
05843279799, 'univers': 0.001389252921639895, 'volumetric': 0.001389252921639895, 'nigel': 0.001389252921639895, 'graders': 0.002778505843279799, 'effects'
: 0.003232160081575754, 'public': 0.0008080400203939385, 'neurotransmitters': 0.005557011686559598, 'three': 0.001639067538813193, 'searching': 0.000889573321
8433985, 'ity': 0.001389252921639895, 'apple': 0.01389252921639895, 'eigth': 0.002778505843279799, 'decwrl': 0.005557011686559598, 'depicting': 0.0069462646
081994974, 'stanford': 0.002778505843279799, 'completed': 0.002778505843279799, 'currently': 0.004167758764919699, 'focus': 0.001554084995613029 4, 'med': 0.00
8335517529839399, 'interactive': 0.0069462646081994974, 'hughes': 0.004167758764919699, 'project': 0.009994398504676847, 'visualizations': 0.011114023373119l9
6, 'volotta': 0.002778505843279799, 'tools': 0.004167758764919699, 'sci': 0.02778505843279799, 'neuron': 0.005557011686559598, 'levels': 0.01389252921639895,
 'bio': 0.005557011686559598, 'excited': 0.001389252921639895, 'education': 0.002778505843279799, 'willing': 0.001389252921639895, 'supporting': 0.001389252
921639895, 'igc': 0.002778505843279799, 'sharing': 0.001389252921639895, 'might': 0.001389252921639895, 'sincerely': 0.0007770424978065147, 'inc': 0.000819
5337694065965, 'org': 0.001389252921639895, 'begin': 0.001389252921639895, 'interested': 0.0015943357330178082, 'eager': 0.001389252921639895, 'months': 0.
001616080040787877, 'schools': 0.001389252921639895, 'affiliated': 0.001389252921639895, 'introductions': 0.001389252921639895, 'next': 0.00416775876491969
9, 'technologies': 0.002778505843279799, 'provide': 0.001389252921639895, 'hear': 0.0008447023081063766, 'cupertino': 0.001389252921639895, 'communications'
: 0.0017469557296817552, 'olivea': 0.001389252921639895, 'mcsun': 0.005557011686559598, 'shaders': 0.001389252921639895, 'moscow': 0.001389252921639895, 'c
orporation': 0.0009979578987049883, 'msk': 0.003991831594819953, 'rib': 0.001389252921639895, 'documentation': 0.0008895733218433985, 'aa': 0.00138925292163 9
8995, 'swrinde': 0.004158622619219223 5, 'maps': 0.001389252921639895, 'su': 0.0069462646081994974, 'complicated': 0.001389252921639895, 'worlds': 0.00138925
2921639895, 'define': 0.0007676879802849195, 'advises': 0.001389252921639895, 'appreciated': 0.0016894046162127532, 'scenes': 0.0012498708221908858, 'langua
ge': 0.002778505843279799, 'far': 0.001389252921639895, 'ames': 0.001389252921639895, 'relcom': 0.001389252921639895, 'fuug': 0.002778505843279799, 'unders
tand': 0.001389252921639895, 'creating': 0.002778505843279799, 'bring': 0.001389252921639895, 'applying': 0.001389252921639895, 'service': 0.00138925292163
98995, 'newsserv': 0.0007868535101068228, 'imaging': 0.0007868535101068228, 'talus': 0.0069462646081994974, 'wanted': 0.002424120061181815 4, 'scene': 0.000767
6879802849195, 'russia': 0.001389252921639895, 'pixar': 0.001389252921639895, 'kiae': 0.002778505843279799, 'shadows': 0.004167758764919699, 'kolesov': 0.00
2778505843279799, 'alex': 0.0069462646081994974, 'newcom': 0.001389252921639895, 'nextstep': 0.001389252921639895, 'examples': 0.0008585757992559746, 'every
body': 0.001389252921639895, 'environmental': 0.001389252921639895, 'reflections': 0.002778505843279799, 'years': 0.00177914664368679 7, 'pocomoco': 0.003558
293287373594, 'experimenting': 0.0008895733218433985, 'stored': 0.0008895733218433985, 'synchronized': 0.00177914664368679 7, 'processing': 0.0054424204784036,
```

We see time is decreasing as we increase number of workers.

Number of Workers	P=8	P=16	P=32
Time taken for execution	346.279 (sec)	177 (sec)	156 sec

References:

- What is Natural Language Processing? Introduction to NLP | DataRobot
- Tf-idf :: A Single-Page Tutorial - Information Retrieval and Text Mining (tfidf.com)