Sharon Laurance

Complex Data Lab: K-means clustering in a Distributed Setting

Distributed K-means Clustering

1. Implement K-means

Description of Algorithm:

Input:

- Vector representing TF-IDF scores of words in 20newsgroup Dataset. Size (11314 X 101322)
- Initial centroids randomly selected from data. Size (3 X 101322)

Output:

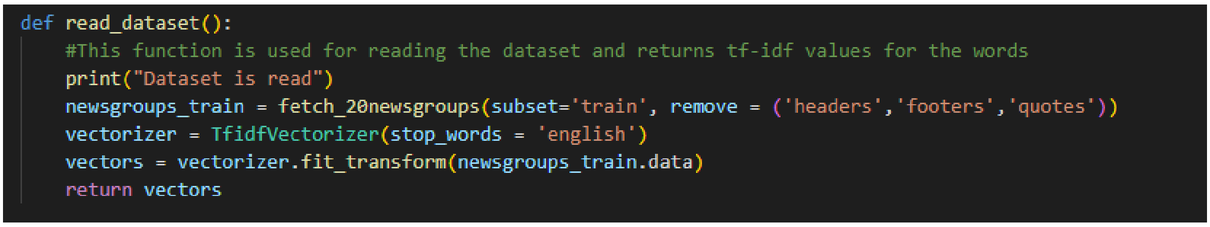
- New computed Centroids which represent the cluster membership. Size (3 X 101322)

Parallelization Strategy

- Collective communication is used for distributed K-Means Implementation.
- The vector having TF-IDF values is split based on the number of workers. (i.e according to size)
- This is sent to all the workers using scatter operation.The global centroid array is broadcasted to all the workers.
- Each worker performs local centroid computation. Euclidean distance is computed between the vector and centroid array. Membership is computed for each point and based on this cluster is assigned. After the cluster is computed new centroid is calculated by taking the mean.
- All local centroids are gathered. Mean of local centroids is computed to find global centroid.
- Convergence criteria is checked to see whether to terminate the algorithm.
- Finally, status of flag is broadcasted so that all workers can terminate.

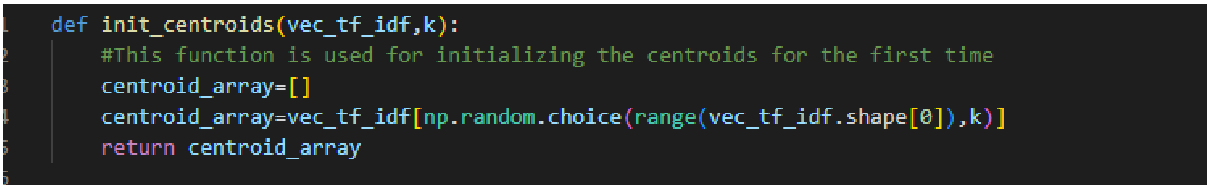
Functions used for Implementation:

1. read_dataset() - This function is used for reading the dataset and returns TF-IDF values for the words. Return value of the function is scipy.sparse.csr.csr_matrix.



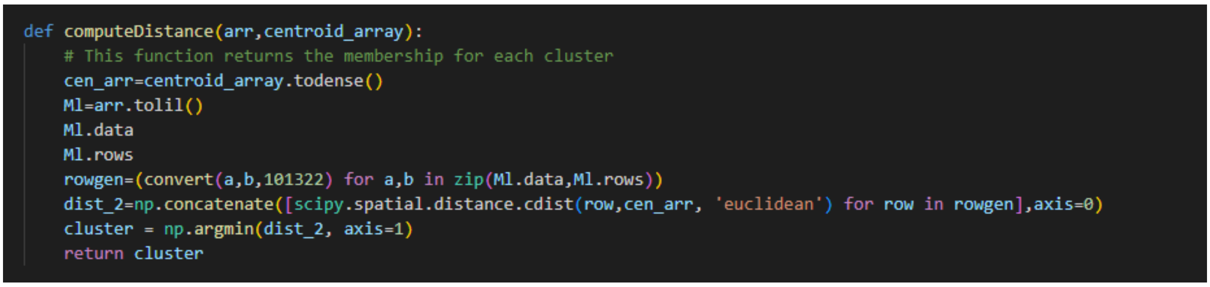
```
def read_dataset():
    #This function is used for reading the dataset and returns tf-idf values for the words
    print("Dataset is read")
    newsgroups_train = fetch_20newsgroups(subset='train', remove = ('headers','footers','quotes'))
    vectorizer = TfidfVectorizer(stop_words = 'english')
    vectors = vectorizer.fit_transform(newsgroups_train.data)
    return vectors
```

2. init_centroids() - This function is used for initializing the centroids for the first time.



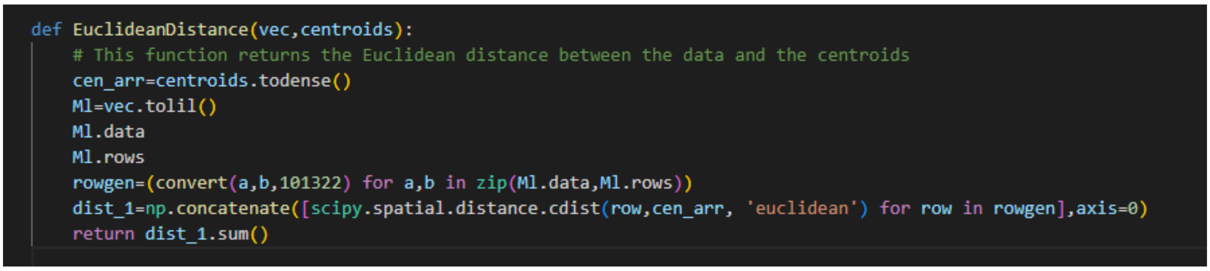
```
def init_centroids(vec_tf_idf,k):
    #This function is used for initializing the centroids for the first time
    centroid_array=[]
    centroid_array=vec_tf_idf[np.random.choice(range(vec_tf_idf.shape[0]),k)]
    return centroid_array
```

3. computeDistance() - This function returns the membership for each cluster.



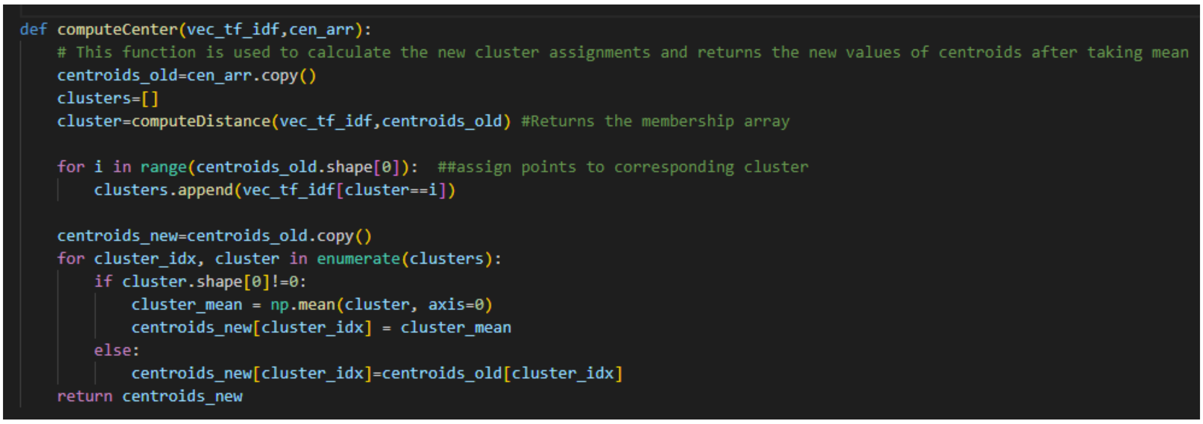
```python
def computeDistance(arr,centroid_array):
    # This function returns the membership for each cluster
    cen_arr=centroid_array.todense()
    Ml=arr.tolil()
    Ml.data
    Ml.rows
    rowgen=(convert(a,b,101322) for a,b in zip(Ml.data,Ml.rows))
    dist_2=np.concatenate([scipy.spatial.distance.cdist(row,cen_arr, 'euclidean') for row in rowgen],axis=0)
    cluster = np.argmin(dist_2, axis=1)
    return cluster
```

4. EuclideanDistance() - This function returns the Euclidean distance between the data and the centroids.



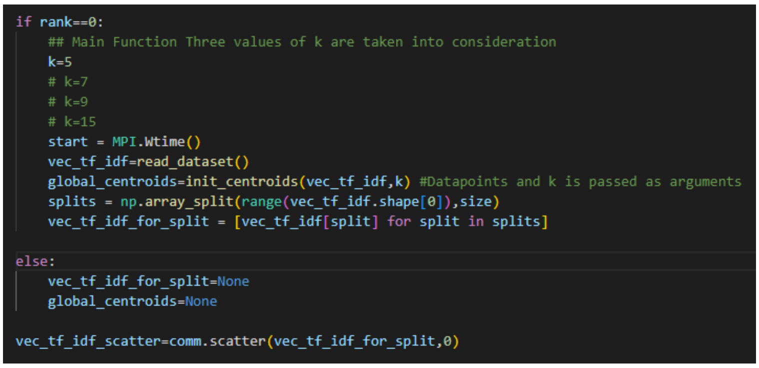
```python
def EuclideanDistance(vec,centroids):
    # This function returns the Euclidean distance between the data and the centroids
    cen_arr=centroids.todense()
    Ml=vec.tolil()
    Ml.data
    Ml.rows
    rowgen=(convert(a,b,101322) for a,b in zip(Ml.data,Ml.rows))
    dist_1=np.concatenate([scipy.spatial.distance.cdist(row,cen_arr, 'euclidean') for row in rowgen],axis=0)
    return dist_1.sum()
```

5. computeCenter() - This function is used to calculate the new cluster assignments and returns the new values of centroids after taking mean.



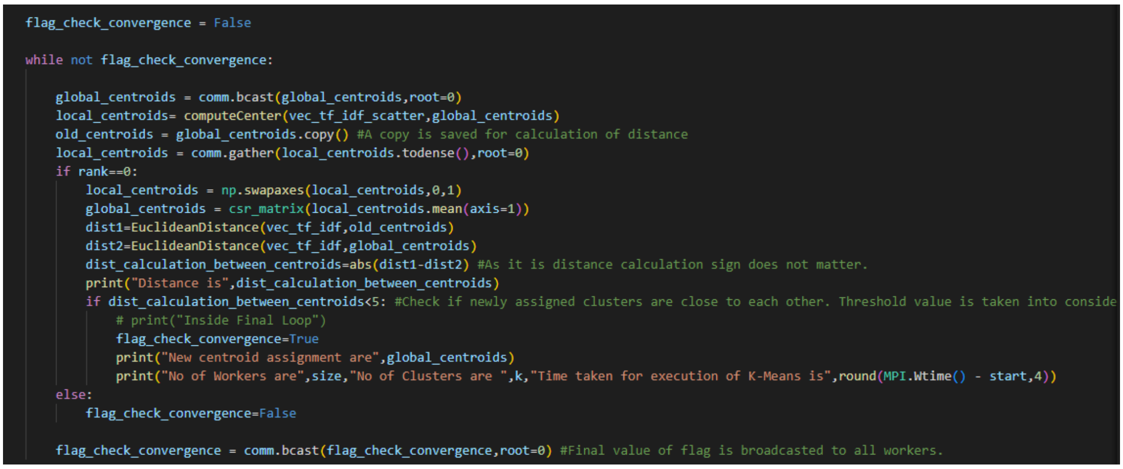
```python
def computeCenter(vec_tf_idf,cen_arr):
    # This function is used to calculate the new cluster assignments and returns the new values of centroids after taking mean
    centroids_old=cen_arr.copy()
    clusters=[]
    cluster=computeDistance(vec_tf_idf,centroids_old) #Returns the membership array

    for i in range(centroids_old.shape[0]):  ##assign points to corresponding cluster
        clusters.append(vec_tf_idf[cluster==i])

    centroids_new=centroids_old.copy()
    for cluster_idx, cluster in enumerate(clusters):
        if cluster.shape[0]!=0:
            cluster_mean = np.mean(cluster, axis=0)
            centroids_new[cluster_idx] = cluster_mean
        else:
            centroids_new[cluster_idx]=centroids_old[cluster_idx]
    return centroids_new
```

When rank==0, data initialization takes place.



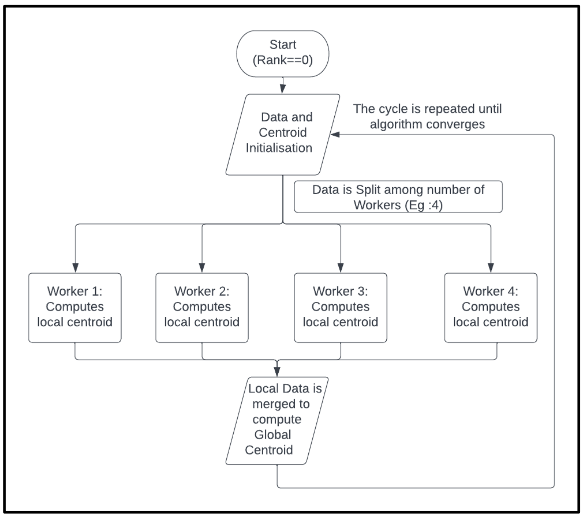
```python
if rank==0:
    ## Main Function Three values of k are taken into consideration
    k=5
    # k=7
    # k=9
    # k=15
    start = MPI.Wtime()
    vec_tf_idf=read_dataset()
    global_centroids=init_centroids(vec_tf_idf,k) #Datapoints and k is passed as arguments
    splits = np.array_split(range(vec_tf_idf.shape[0]),size)
    vec_tf_idf_for_split = [vec_tf_idf[split] for split in splits]

else:
    vec_tf_idf_for_split=None
    global_centroids=None

vec_tf_idf_scatter=comm.scatter(vec_tf_idf_for_split,0)
```

The while is run until the convergence criteria is fulfilled. Absolute value od difference of distance is taken into consideration.



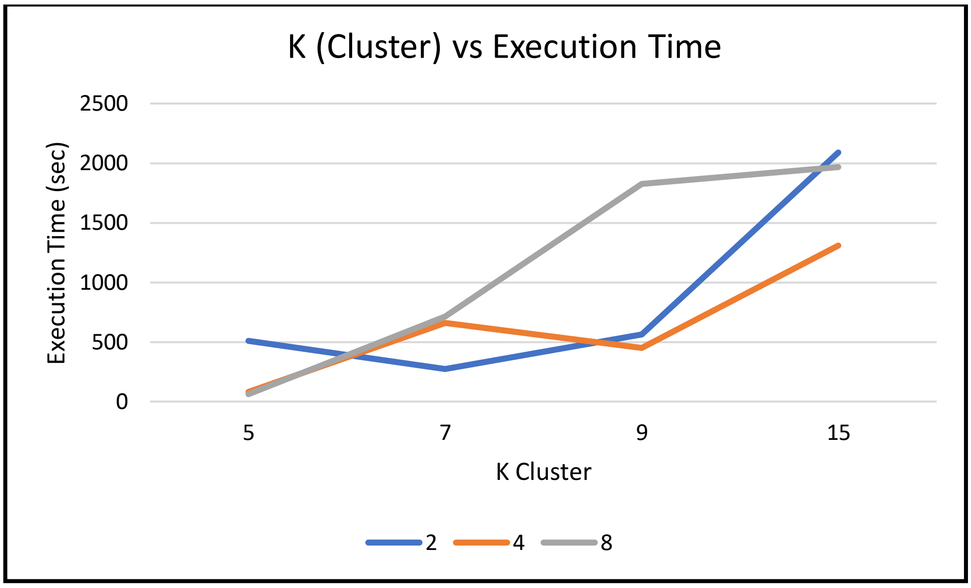
```
flag_check_convergence = False

while not flag_check_convergence:

    global_centroids = comm.bcast(global_centroids,root=0)
    local_centroids= computeCenter(vec_tf_idf_scatter,global_centroids)
    old_centroids = global_centroids.copy() #A copy is saved for calculation of distance
    local_centroids = comm.gather(local_centroids.todense(),root=0)
    if rank==0:
        local_centroids = np.swapaxes(local_centroids,0,1)
        global_centroids = csr_matrix(local_centroids.mean(axis=1))
        dist1=EuclideanDistance(vec_tf_idf,old_centroids)
        dist2=EuclideanDistance(vec_tf_idf,global_centroids)
        dist_calculation_between_centroids=abs(dist1-dist2) #As it is distance calculation sign does not matter.
        print("Distance is",dist_calculation_between_centroids)
        if dist_calculation_between_centroids<5: #Check if newly assigned clusters are close to each other. Threshold value is taken into conside
            # print("Inside Final Loop")
            flag_check_convergence=True
            print("New centroid assignment are",global_centroids)
            print("No of Workers are",size,"No of Clusters are ",k,"Time taken for execution of K-Means is",round(MPI.Wtime() - start,4))
    else:
        flag_check_convergence=False

    flag_check_convergence = comm.bcast(flag_check_convergence,root=0) #Final value of flag is broadcasted to all workers.
```

Flowchart for Algorithm


Start (Rank==0)
Data and Centroid Initialisation
The cycle is repeated until algorithm converges
Data is Split among number of Workers (Eg :4)
Worker 1: Computes local centroid
Worker 2: Computes local centroid
Worker 3: Computes local centroid
Worker 4: Computes local centroid
Local Data is merged to compute Global Centroid

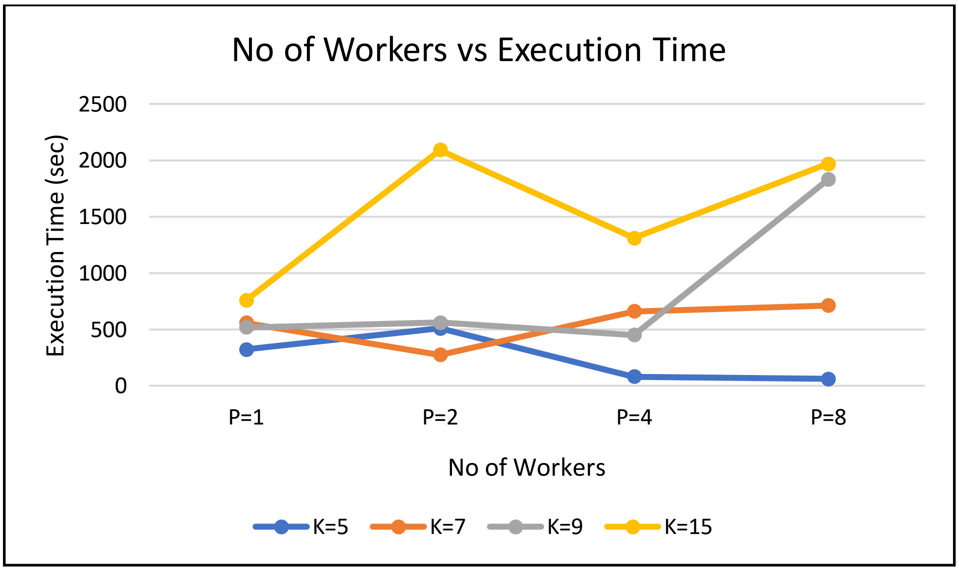
2. Performance Analysis

Cluster No	P=1	P=2	P=4	P=8
K=5	320.6227 sec	509.74 sec	80.43 sec	61.5541 sec
K=7	558.18 sec	272.9113 sec	660.0 sec	711.025 sec
K=9	517 sec	561.27 sec	449.18 sec	1828 sec
K=15	758.46 sec	2089.19 sec	1308.55 sec	1967.02 sec



Here 2,4 and 8 are the number of workers.

We can analyse that as the number of clusters increase the time increases for K-means algorithm. As the number of workers increase the execution time decreases.

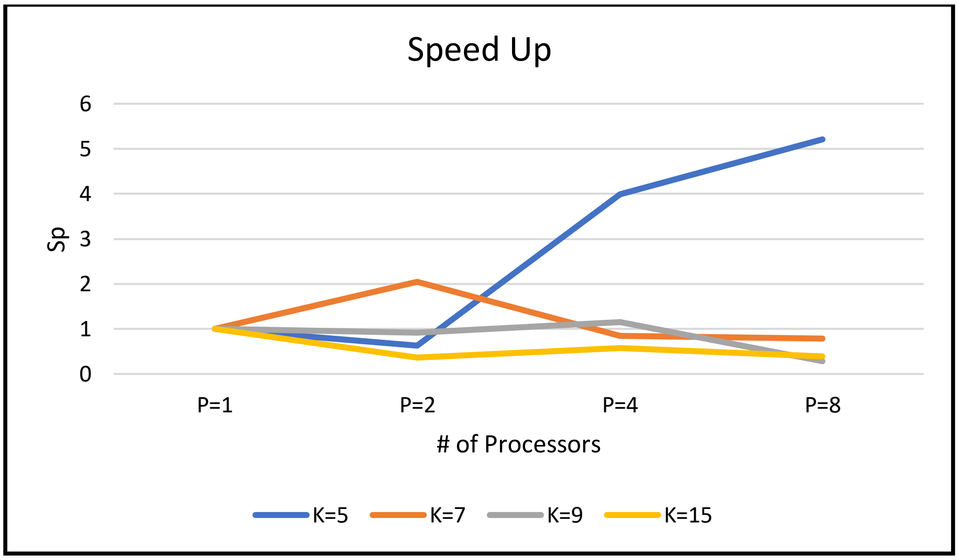


Here K=5,7,9,15 are the number of clusters

As the number of workers increase, the execution time decreases which can seen for all clusters. This is seen till P=4 after that communication overhead increases for P=8 so time can be seen as increasing. Also, in K-Means algorithm initial random centroid selection plays an part.

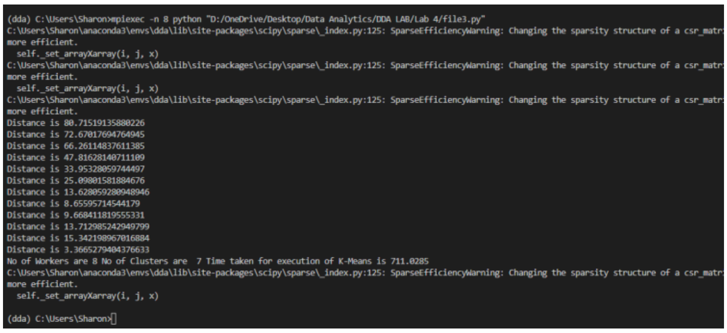
Speedup Curve:

The formula for S_p= (Best Serial Execution Time/Execution Time on p processes). Sub-linear speedup is the most common.

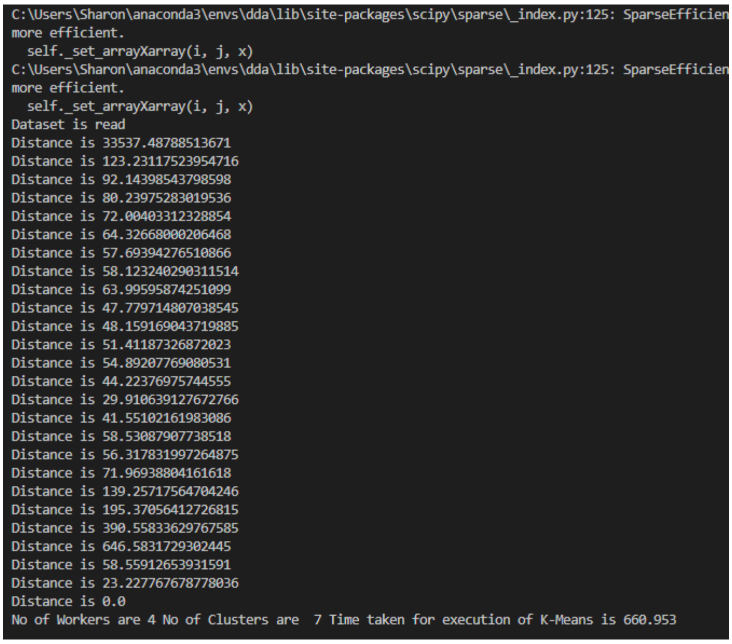


We can see that speedup occurs as the number of processes increases till P=4.When speedup is greater than 1 it is better. The best speedup occurs for Cluster K=5.

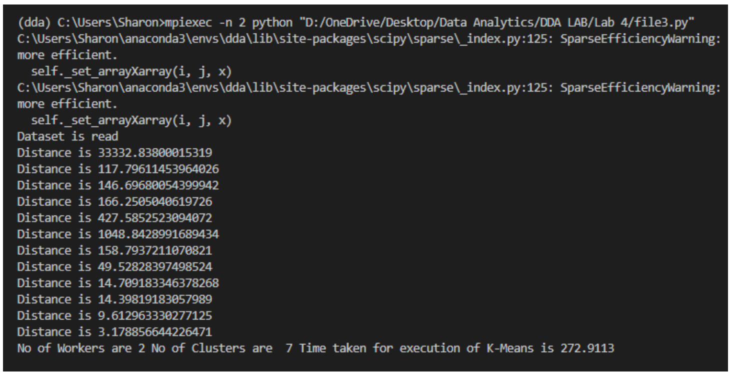
Output Screenshots:



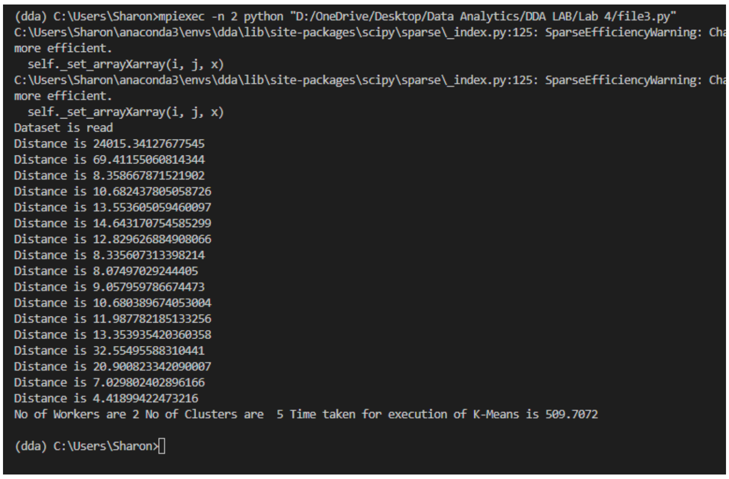
```
(dda) C:\Users\Sharon>mpiexec -n 8 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 4/file3.py"
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matr
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matr
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matr
more efficient.
Distance is 80.71519135880226
Distance is 72.67017694764945
Distance is 66.26114837611385
Distance is 47.81628140711109
Distance is 33.95328059744497
Distance is 25.09801581884676
Distance is 13.628059280948946
Distance is 8.65595714544179
Distance is 9.668411819555331
Distance is 13.712985242949799
Distance is 15.342198967016884
Distance is 3.3665279404376633
No of Workers are 8 No of Clusters are  7 Time taken for execution of K-Means is 711.0285
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changing the sparsity structure of a csr_matr
more efficient.
  self._set_arrayXarray(i, j, x)

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



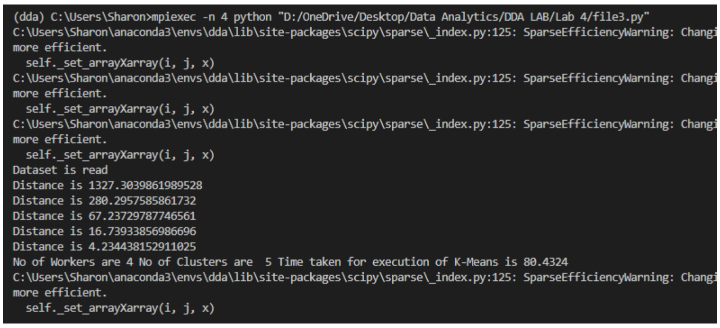
```
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficien
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficien
more efficient.
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 33537.48788513671
Distance is 123.23117523954716
Distance is 92.14398543798598
Distance is 80.23975283019536
Distance is 72.00403312328854
Distance is 64.32668000206468
Distance is 57.69394276510866
Distance is 58.123240290311514
Distance is 63.99595874251099
Distance is 47.779714807038545
Distance is 48.159169043719885
Distance is 51.41187326872023
Distance is 54.89207769080531
Distance is 44.22376975744555
Distance is 29.910639127672766
Distance is 41.55102161983086
Distance is 58.53087907738518
Distance is 56.317831997264875
Distance is 71.96938804161618
Distance is 139.25717564704246
Distance is 195.37056412726815
Distance is 390.55833629767585
Distance is 646.5831729302445
Distance is 58.55912653931591
Distance is 23.227767678778036
Distance is 0.0
No of Workers are 4 No of Clusters are  7 Time taken for execution of K-Means is 660.953
```



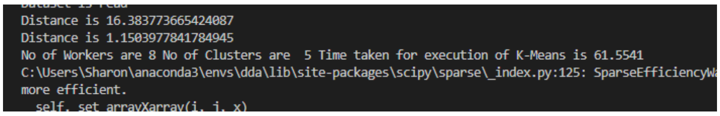
```
(dda) C:\Users\Sharon>mpiexec -n 2 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 4/file3.py"
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning:
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning:
more efficient.
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 33332.83800015319
Distance is 117.79611453964026
Distance is 146.69680054399942
Distance is 166.2505040619726
Distance is 427.5852523094072
Distance is 1048.8428991689434
Distance is 158.7937211070821
Distance is 49.52828397498524
Distance is 14.709183346378268
Distance is 14.39819183057989
Distance is 9.612963330277125
Distance is 3.178856644226471
No of Workers are 2 No of Clusters are  7 Time taken for execution of K-Means is 272.9113
```



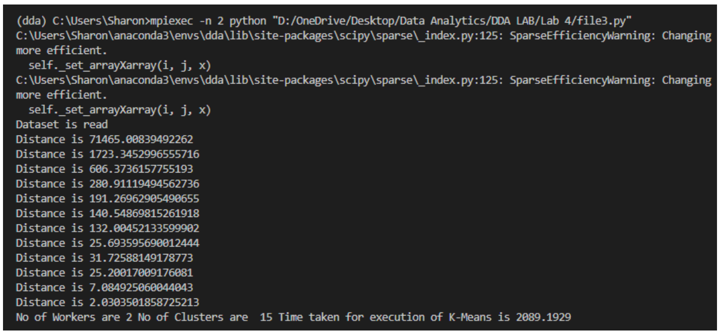
```
(dda) C:\Users\Sharon>mpiexec -n 2 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 4/file3.py"
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Cha
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Cha
more efficient.
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 24015.34127677545
Distance is 69.41155060814344
Distance is 8.358667871521902
Distance is 10.682437805058726
Distance is 13.553605059460097
Distance is 14.643170754585299
Distance is 12.829626884908066
Distance is 8.335607313398214
Distance is 8.07497029244405
Distance is 9.057959786674473
Distance is 10.680389674053004
Distance is 11.987782185133256
Distance is 13.353935420360358
Distance is 32.55495588310441
Distance is 20.900823342090007
Distance is 7.029802402896166
Distance is 4.41899422473216
No of Workers are 2 No of Clusters are  5 Time taken for execution of K-Means is 509.7072

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



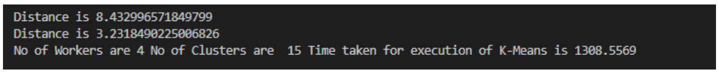
```
(dda) C:\Users\Sharon>mpiexec -n 4 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 4/file3.py"
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changi
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changi
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changi
more efficient.
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 1327.3039861989528
Distance is 280.2957585861732
Distance is 67.23729787746561
Distance is 16.73933856986696
Distance is 4.234438152911025
No of Workers are 4 No of Clusters are  5 Time taken for execution of K-Means is 80.4324
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changi
more efficient.
  self._set_arrayXarray(i, j, x)
```



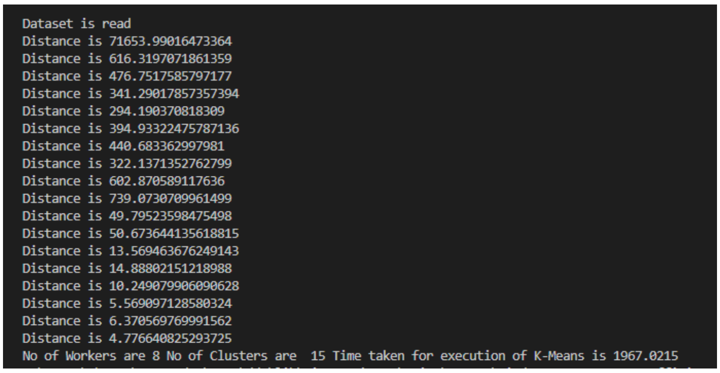
```
Dataset is read
Distance is 16.383773665424087
Distance is 1.1503977841784945
No of Workers are 8 No of Clusters are  5 Time taken for execution of K-Means is 61.5541
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWa
more efficient.
  self._set_arrayXarray(i, j, x)
```



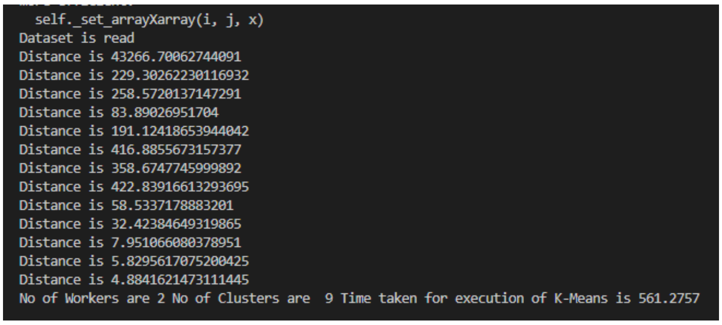
```
(dda) C:\Users\Sharon>mpiexec -n 2 python "D:/OneDrive/Desktop/Data Analytics/DDA LAB/Lab 4/file3.py"
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changing
more efficient.
  self._set_arrayXarray(i, j, x)
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficiencyWarning: Changing
more efficient.
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 71465.00839492262
Distance is 1723.3452996555716
Distance is 606.3736157755193
Distance is 280.91119494562736
Distance is 191.26962905490655
Distance is 140.54869815261918
Distance is 132.00452133599902
Distance is 25.693595690012444
Distance is 31.72588149178773
Distance is 25.20017009176081
Distance is 7.084925060044043
Distance is 2.0303501858725213
No of Workers are 2 No of Clusters are  15 Time taken for execution of K-Means is 2089.1929
```



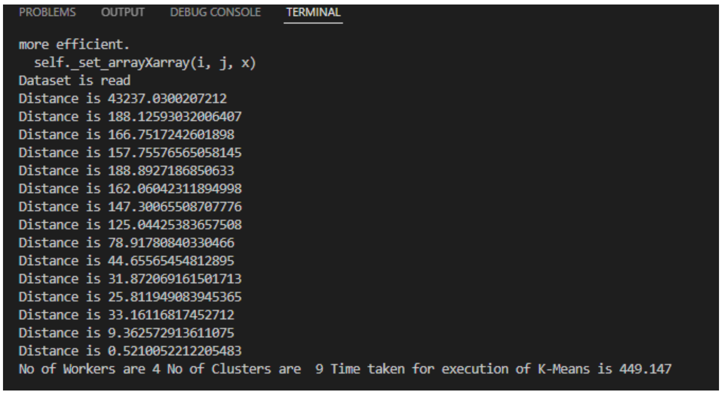
```
Distance is 8.432996571849799
Distance is 3.2318490225006826
No of Workers are 4 No of Clusters are  15 Time taken for execution of K-Means is 1308.5569
```



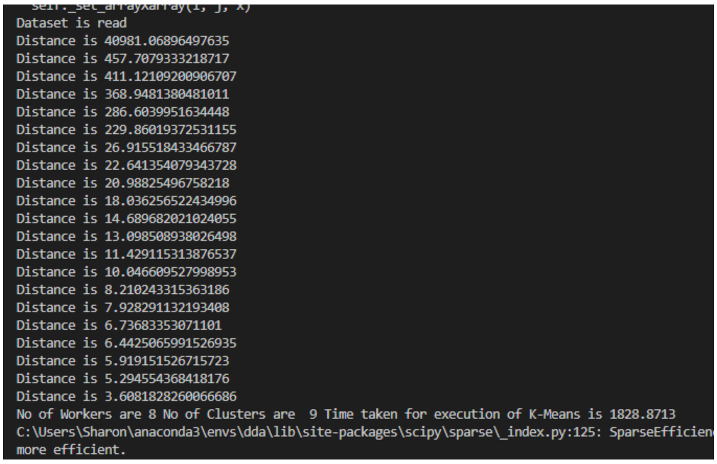
```
Dataset is read
Distance is 71653.99016473364
Distance is 616.3197071861359
Distance is 476.7517585797177
Distance is 341.29017857357394
Distance is 294.190370818309
Distance is 394.93322475787136
Distance is 440.683362997981
Distance is 322.1371352762799
Distance is 602.870589117636
Distance is 739.0730709961499
Distance is 49.79523598475498
Distance is 50.673644135618815
Distance is 13.569463676249143
Distance is 14.88802151218988
Distance is 10.249079906090628
Distance is 5.569097128580324
Distance is 6.370569769991562
Distance is 4.776640825293725
No of Workers are 8 No of Clusters are  15 Time taken for execution of K-Means is 1967.0215
```



```
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 43266.70062744091
Distance is 229.30262230116932
Distance is 258.5720137147291
Distance is 83.89026951704
Distance is 191.12418653944042
Distance is 416.8855673157377
Distance is 358.6747745999892
Distance is 422.83916613293695
Distance is 58.5337178883201
Distance is 32.42384649319865
Distance is 7.951066080378951
Distance is 5.8295617075200425
Distance is 4.8841621473111445
No of Workers are 2 No of Clusters are  9 Time taken for execution of K-Means is 561.2757
```



```
more efficient.
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 43237.0300207212
Distance is 188.12593032006407
Distance is 166.7517242601898
Distance is 157.75576565058145
Distance is 188.8927186850633
Distance is 162.06042311894998
Distance is 147.30065508707776
Distance is 125.04425383657508
Distance is 78.91780840330466
Distance is 44.65565454812895
Distance is 31.872069161501713
Distance is 25.811949083945365
Distance is 33.16116817452712
Distance is 9.362572913611075
Distance is 0.5210052212205483
No of Workers are 4 No of Clusters are  9 Time taken for execution of K-Means is 449.147
```



```
  self._set_arrayXarray(i, j, x)
Dataset is read
Distance is 40981.06896497635
Distance is 457.7079333218717
Distance is 411.12109200906707
Distance is 368.9481380481011
Distance is 286.6039951634448
Distance is 229.86019372531155
Distance is 26.915518433466787
Distance is 22.641354079343728
Distance is 20.98825496758218
Distance is 18.036256522434996
Distance is 14.689682021024055
Distance is 13.098508938026498
Distance is 11.429115313876537
Distance is 10.046609527998953
Distance is 8.210243315363186
Distance is 7.928291132193408
Distance is 6.73683353071101
Distance is 6.4425065991526935
Distance is 5.919151526715723
Distance is 5.294554368418176
Distance is 3.6081828260066686
No of Workers are 8 No of Clusters are  9 Time taken for execution of K-Means is 1828.8713
C:\Users\Sharon\anaconda3\envs\dda\lib\site-packages\scipy\sparse\_index.py:125: SparseEfficien
more efficient.
```

References:

- https://stackoverflow.com/questions/36557472/calculate-the-euclidean-distance-in-scipy-csr-matrix
- Shared Memory Programming (uni-hildesheim.de)