Project 2 Report:
Big Data Management

General Introduction:

In this report, we present the results of Project 2, which focuses on revisiting the "FaceIn" dataset using Apache Pig and implementing K-means clustering for a large dataset of 2-dimensional points. Additionally, we chose to extend the project by implementing K-medoids clustering and the BYOD (Bring Your Own Data) for our K-means algorithm. We elaborate what we did in each task below this.

Team Composition:

Our team members are namely,

- Sreeram Marimuthu
- Nitya Phani Santosh Oruganty
- Anushka Bangal

We cooperated to execute this project as planned. To ensure the project's success, each team member had a specific responsibility.

Part – 1: Revisiting FaceIn with Pig:

Introduction:

In this part of the report, we present the implementation and execution of tasks from Project 1 using Apache Pig, a high-level platform for processing large datasets in a parallel and distributed fashion. The goal is to rewrite the original 8 tasks from Project 1 using Apache Pig and test them on the data created as part of Project 1. We executed the tasks line by line in the grunt shell. Each task is described, along with the Pig script used to perform the task and the logic behind it. We've attached the

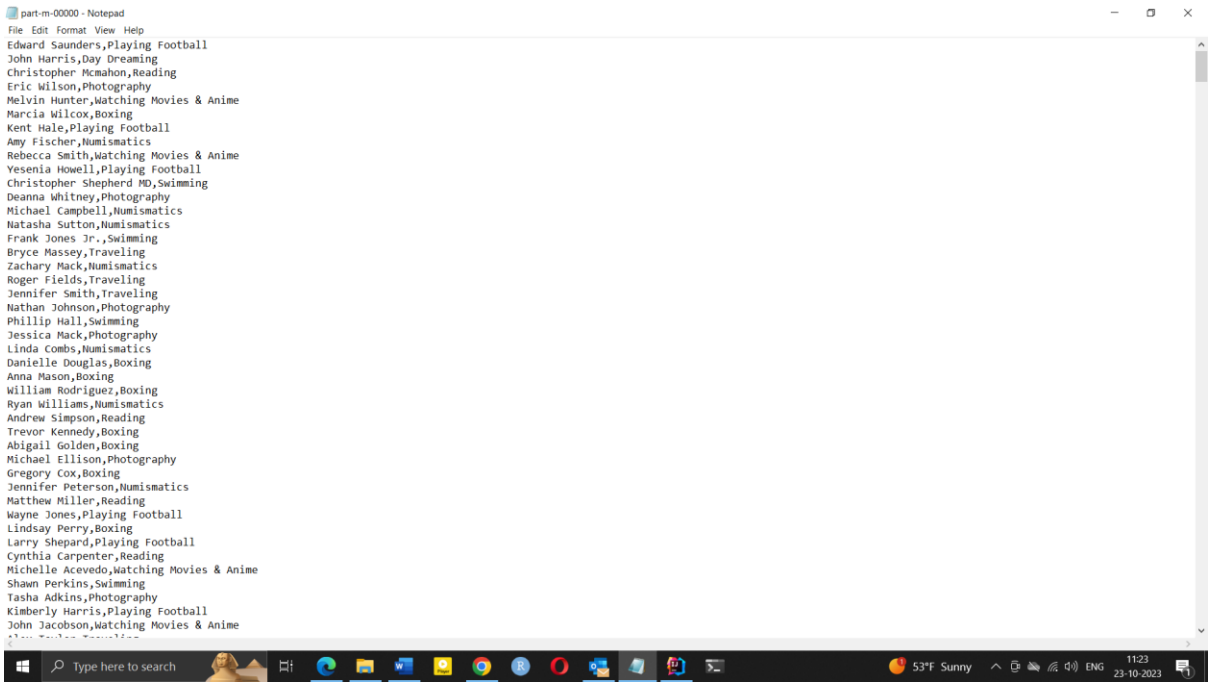
Task A:

Description: Filter users in the FaceInPage dataset whose nationality matches the user's own (in this case, 'India') and project their name and hobby.

Logic:

- We start by loading the FaceInPage dataset and filtering it to include only users with the nationality 'India'.
- We then project the Name and Hobby of the filtered users.
- Finally, the result is stored and dumped to see users with the same nationality.

Output:



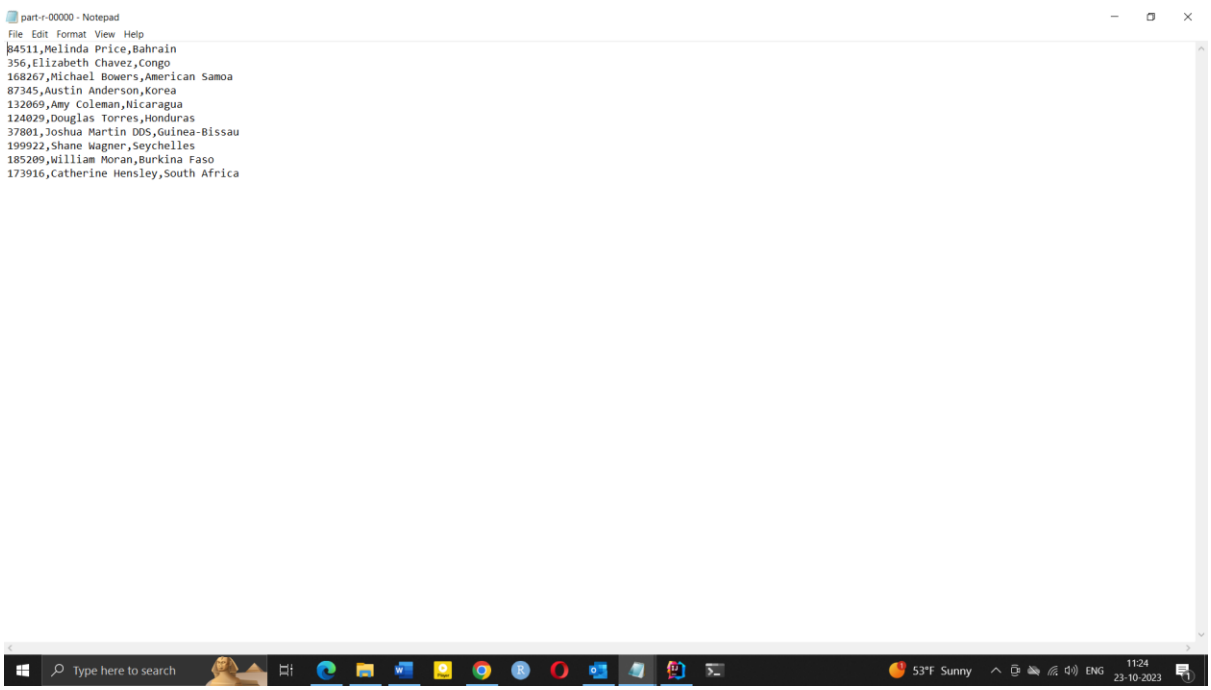
Task B:

Description: Group the AccessLogs by WhatPage, calculate access counts for each FaceInPage, and find the top 10 most popular pages.

Logic:

- We group the AccessLogs dataset by 'WhatPage' to calculate the access count for each FaceInPage.
- We then join the access count with the FaceInPage dataset to obtain the Name and Nationality of each FaceInPage.
- The pages are ordered in descending order by access count and limited to the top 10.
- Finally, the result is stored and dumped to see the top 10 popular pages.

Output:



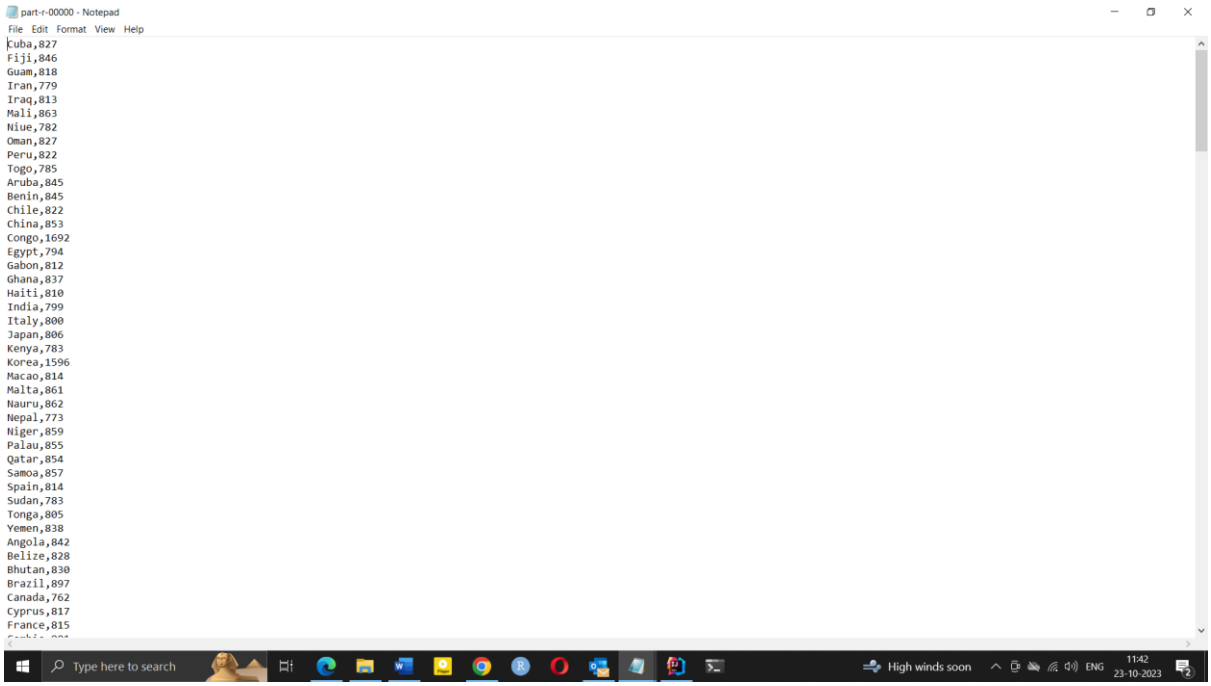
Task C:

Description: Group the FaceInPage dataset by Nationality and calculate the count of citizens for each country.

Logic:

- We group the FaceInPage dataset by 'Nationality'.
- We calculate the count of citizens for each country.
- The result is stored and dumped to report the count of citizens for each country.

Output:



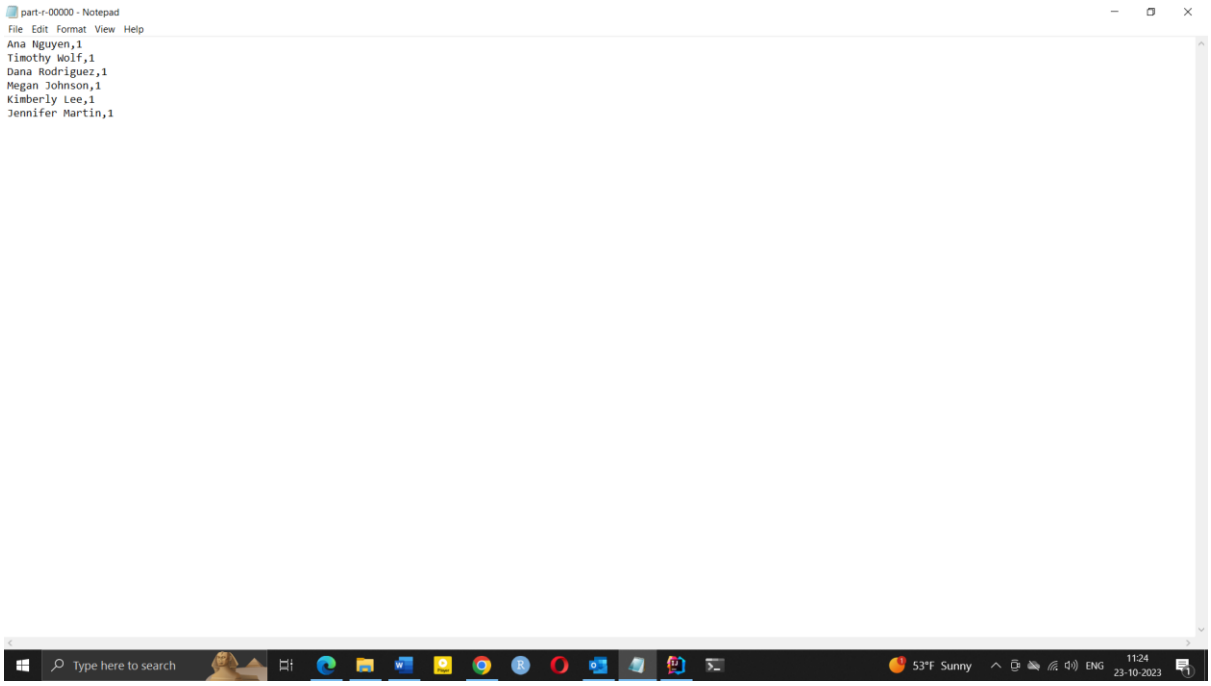
Task D:

Description: Filter relationships for unique owners (PersonA_ID) and calculate the "HappinessFactor" for each FaceInPage owner.

Logic:

- We filter the Associates dataset to get unique owners (PersonA_ID).
- Group the Associates dataset by PersonA_ID to count relationships.
- Join the FaceInPage data with the owner's relationship count.
- If a FaceInPage owner isn't in Associates, their HappinessFactor is set to 0 and ignored.
- The result is stored and dumped to report the HappinessFactor for each FaceInPage owner.

Output:



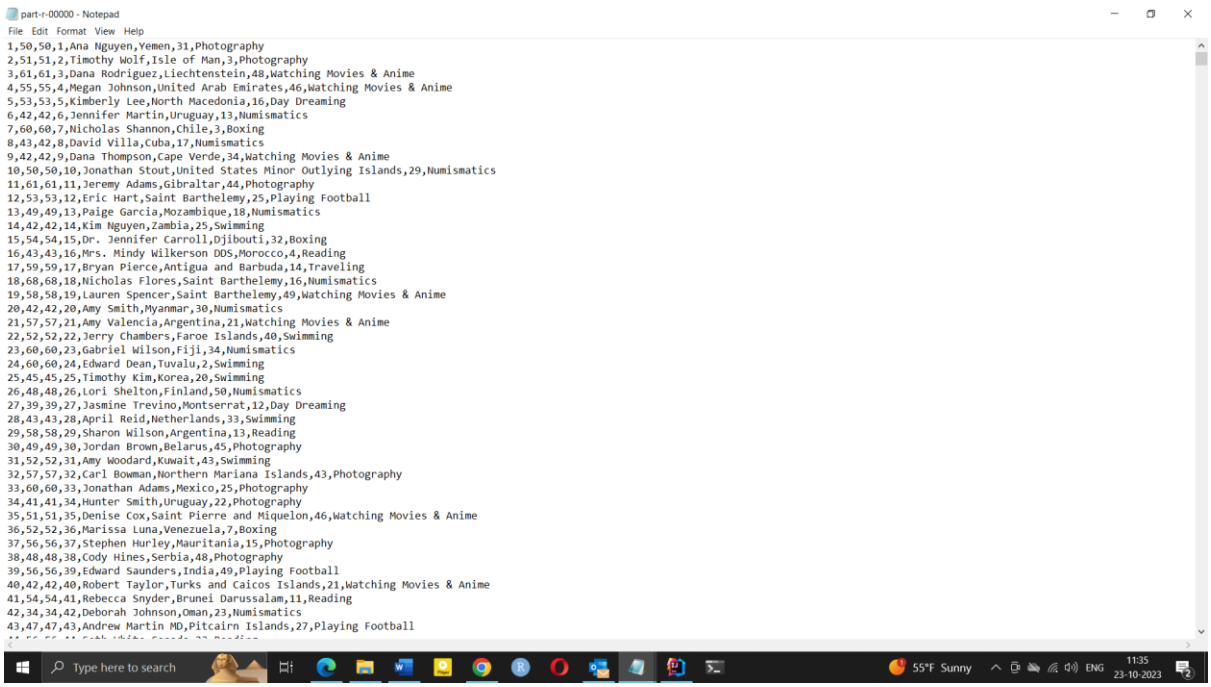
Task E:

Description: Calculate total access count and distinct page count for each person in the AccessLogs dataset.

Logic:

- We load the AccessLogs dataset and explicitly cast columns to their intended data types.
- Group the AccessLogs dataset by the 'ByWho' column.
- Calculate the total number of accesses and the number of distinct FaceInPages accessed for each person.
- Join the result with the FaceInPage dataset to get owner's names.
- The result is stored and dumped to report people who have favorites.

Output:



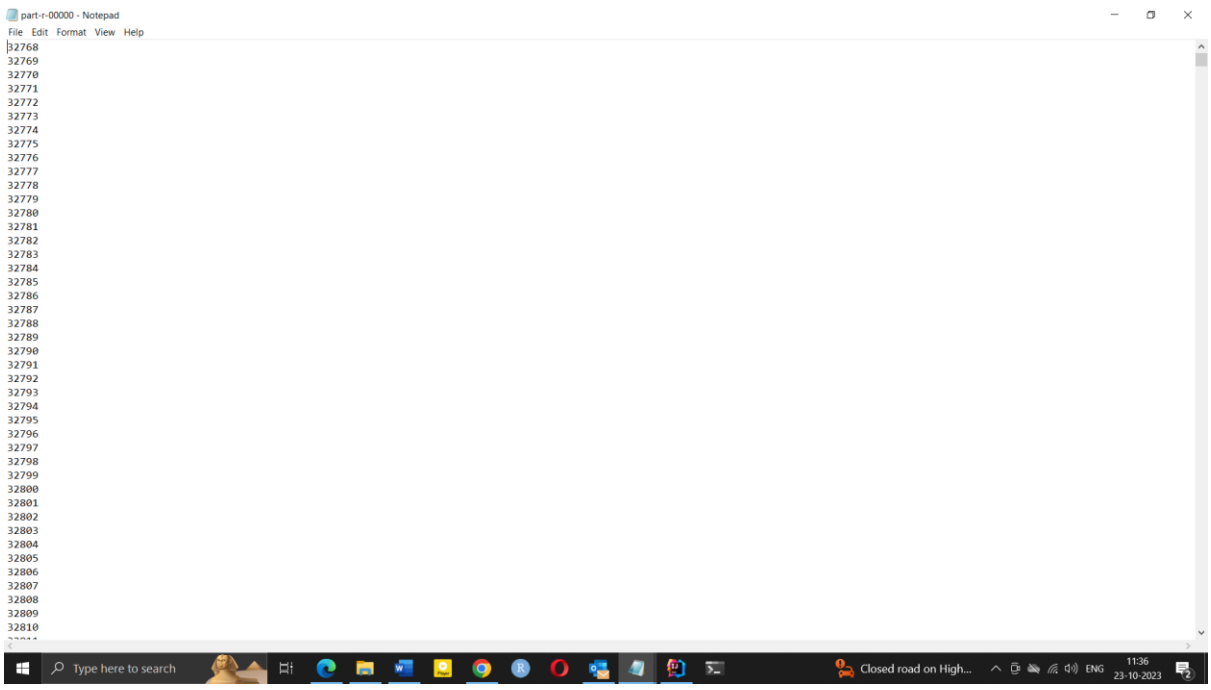
Task F:

Description: Find friends who have not accessed their friend's FaceInPage.

Logic:

- We load the Associates and AccessLogs datasets and cast their data types.
- Filter AccessLogs for distinct Person IDs and pages accessed.
- Join Associates with distinct_access to find friends who accessed their FaceInPage.
- Filter to get friends with no access to the same page.
- Load the FaceInPage dataset to get names.
- Join the result with the FaceInPage dataset to get IDs and names.
- The result is projected to get the IDs of people who meet the criteria (friends who did not access their FaceInPage).
- Optionally, duplicates are removed, and the result is stored and dumped to report people who meet the criteria.

Output:



Task G:

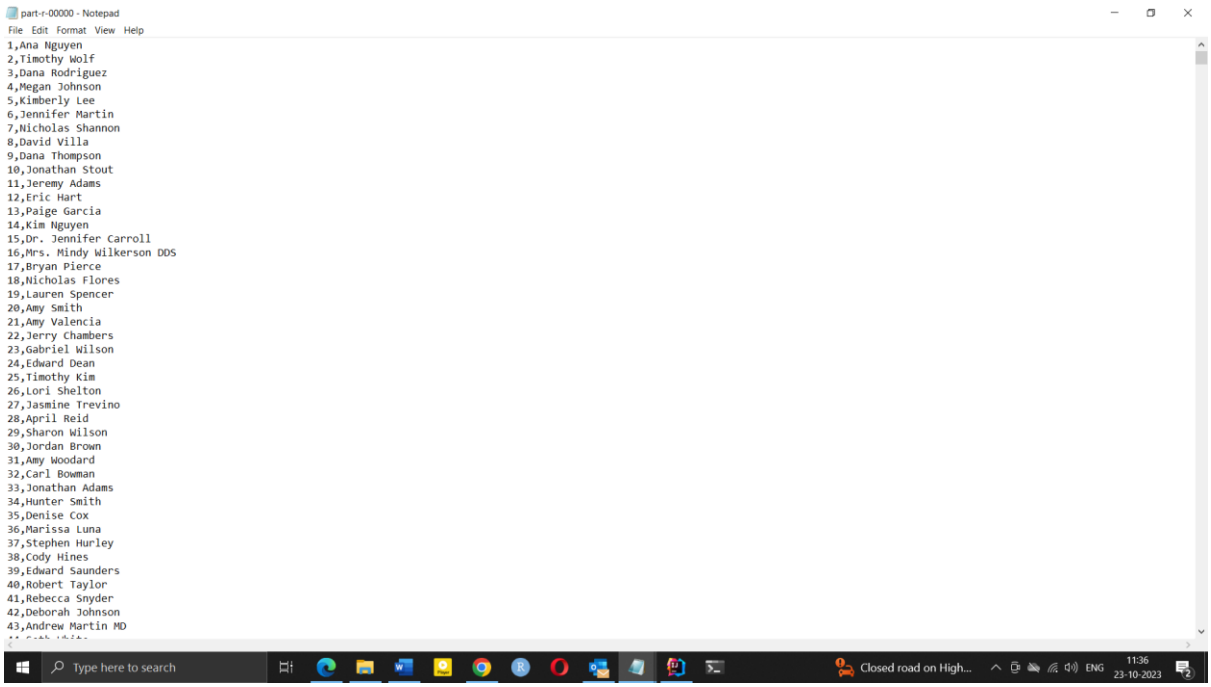
Description: Find FaceInPages that haven't been accessed in the last 90 days.

Here, we initially struggled to calculate the 90 days ago timestamp and perform the respective arithmetic operation through Pig which didn’t work and we used ChatGPT on how to rectify this which suggested to calculate the same in pandas and the upload the value from a file which is later used to compare and filter out the data as expected.

Logic:

- We calculate the timestamp representing 90 days ago in minutes through a Python and save it as a file use it as a value for meeting the criteria.
- Load the AccessLogs dataset and group it by 'WhatPage'.
- Calculate the most recent access date for each FaceInPage.
- Filter FaceInPages that haven't been accessed in the last 90 days.
- Load the FaceInPage dataset to get names.
- Join the result with the FaceInPage dataset to get IDs and names.
- The result is projected to get the IDs and names of outdated FaceInPages.
- The result is stored and dumped to report the outdated FaceInPages.

Output:



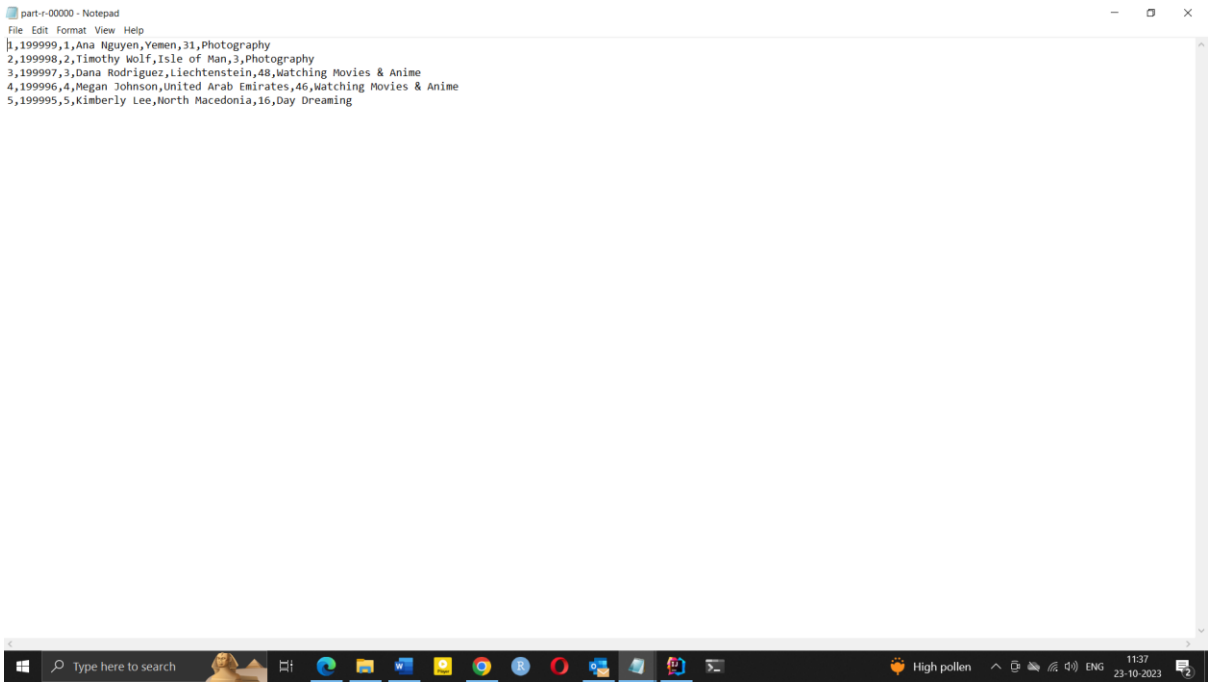
Task H:

Description: Find owners with more relationships than the average.

Logic:

- Group the Associates dataset by owner (PersonA_ID).
- Calculate the number of relationships for each owner.
- Calculate the average number of relationships across all owners.
- Filter owners with more relationships than the average.
- Join the result with the FaceInPage dataset to get owner names.
- The result is stored and dumped to report owners with more relationships than the average.

Output:



Trade-off with MapReduce:

- Initially since we were both efficiently equipped with SQL, Pig was much easier than MapReduce to perform the tasks. And it was more concise than writing Java code for MapReduce.
- Since Pig has built-in error handling mechanisms which made it easier for us to handle exceptions & errors gracefully and much quickly. In MapReduce, error handling required more manual process time.
- Pig performs query optimization automatically. It optimizes the execution plan, reorders operations, and pushes down filter conditions. So we didn't need to spend as much time fine-tuning and optimizing the code for each task as we did for MapReduce.
- Writing MapReduce code in Java typically involves writing a significant amount of boilerplate code for tasks such as setting up job configurations, serialization, and managing input and output formats.
- But in Pig, it abstracts much of this away, allowing us to focus on the data transformations and logic.
- Although it was much easier to do these tasks in Pig, we do understand the importance of Hadoop MapReduce to perform them.

Conclusion:

As a result, we successfully revisited the tasks from Project 1 using Apache Pig. Each task was implemented with a detailed explanation of the logic and accompanied by the corresponding Pig script. The outputs of each task have been stored and attached as separate files for reference. Since we ran line by line using grunt shell as instructed, we've attached the entire script file as .txt file. We ran it completely and verified that it is working just make sure you have the datasets and the 90daystimestamp file (which we have attached with this submission) with correct path before running it on your end. The tasks were executed efficiently using Pig, allowing for easy data processing and analysis.

Part – 2: K-Means Clustering:

Introduction:

K-means clustering is a fundamental data mining algorithm used for grouping objects into clusters based on their similarity. In this section, we present an introduction to Task 2, which involves the creation of a large dataset, development of various K-means clustering strategies using Java MapReduce jobs, and the analysis of their performance and results.

It begins with an initial set of K cluster centers, which are typically randomly chosen from the dataset. The algorithm iteratively refines these cluster centers by assigning data points to the nearest cluster and recalculating the center based on the assigned points. This process continues until convergence or after a specified number of iterations (parameter R for rounds).

Dataset Creation:

The first step in Task 2 is to generate a large dataset of at least 5,000 points, each consisting of 2-dimensional (x, y) values. These values range from 0 to 10,000. Additionally, we create another file containing K initial seed points, with each value also in the range from 0 to 10,000. The value of K is parameterized, allowing us to generate these seed points randomly. Both datasets are uploaded to Hadoop Distributed File System (HDFS) for use in the subsequent clustering steps.

For generating the datasets, we used Pandas & NumPy. We initially created a file for 5000 points and tested it for values k=3,4,5 & 7 and did the same for 50000 points & 10000 points as well. Based on the results we chose the dataset with 10000 points which had more clustering compared to the other two and proceeded with that.

To find the optimal value of K:

The elbow method is a heuristic used to determine the optimal number of clusters (k) in a k-means clustering algorithm. It is called the "elbow method" because the plot of the sum of squared distances (inertia) as a function of the number of clusters often resembles an elbow shape. The point where this plot

shows a significant change in the rate of decrease in inertia is considered the "elbow," and it indicates the optimal k value for the clustering.

Here's how the elbow method works and its importance:

K-Means Clustering: K-means is a popular unsupervised machine learning algorithm used for clustering data points into k clusters. However, the choice of k is typically not known in advance, and determining the appropriate number of clusters is essential.

Sum of Squared Distances (Inertia): In K-means clustering, the algorithm tries to minimize the sum of squared distances between data points and their cluster centroids. This sum of squared distances is referred to as "inertia."

Elbow Method Procedure:

- Run the K-means algorithm for a range of k values, typically from 1 to some maximum value.
- For each k, compute the inertia, which measures how closely data points are associated with their cluster's centroid.
- Plot the k values on the x-axis and the corresponding inertia values on the y-axis.

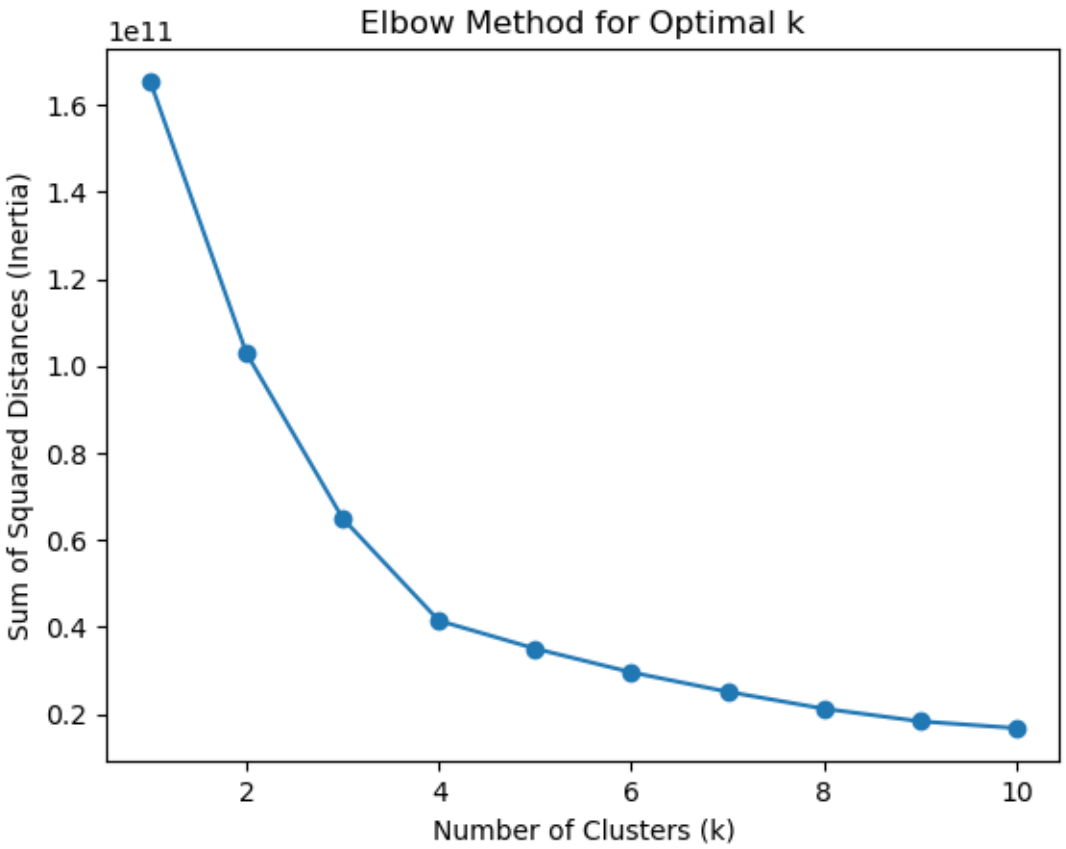
Interpreting the Elbow Plot:

- As you increase the number of clusters (k), the inertia tends to decrease because the data points are closer to their cluster centroids.
- The elbow point is the value of k where the rate of decrease in inertia significantly slows down, and the plot forms an "elbow" shape.
- This point is considered the optimal number of clusters because it balances the trade-off between the compactness of clusters (lower inertia) and the interpretability of the clusters (fewer clusters).

Importance:

- The elbow method is crucial in determining the appropriate number of clusters for a dataset, as it helps avoid overfitting (too many clusters) or underfitting (too few clusters).
- It provides a data-driven approach to decide the value of k, making the clustering results more meaningful and interpretable.
- While the elbow method is a heuristic, it is widely used and provides a quick and effective way to estimate the optimal number of clusters.

For the dataset we used the optimal k value from the elbow method is 4, which is inferred from the below plot.



It's important to note that in some cases, the elbow method may not produce a clear and distinct elbow, making it challenging to choose an optimal k value. In such cases, other methods or domain knowledge may be necessary to make the final decision on the number of clusters.

Task 2-a: Single-Iteration K-means Algorithm (R=1)

Here, we will delve into the implementation of the Single-Iteration K-means Algorithm, where the parameter R is set to 1. The primary goal of this algorithm is to perform a single iteration of the K-means clustering process. We will explain the logic behind the mappers and reducers and discuss how our optimized solution enhances the algorithm's efficiency.

Mapper:

The Single-Iteration K-means Algorithm employs a Mapper that reads data points from the input file and assigns each point to the nearest cluster center. The Mapper's key-value pairs have the following format:

- **Key:** A LongWritable representing the byte offset of the input line.
- **Value:** A Text object containing a data point in the form of "x:y," where x and y are the coordinates of the point.

The Mapper's logic includes the following steps:

- Parse the input Text value to extract the x and y coordinates of the data point.
- Create a Point object with the extracted coordinates.
- Find the nearest cluster center by calculating the Euclidean distance between the data point and each cluster center.
- Emit key-value pairs with the cluster center's ID as the key and the data point's coordinates as the value.

Reducer:

The Reducer takes the output of the Mappers, which groups data points by their assigned cluster centers. The key-value pairs for the Reducer have the following format:

- **Key:** An IntWritable representing the cluster center's ID.
- **Value:** A Text object containing data point coordinates.

The Reducer's logic involves the following steps:

- Iterate through the list of data points assigned to the same cluster center.
- Calculate the new cluster center by computing the average of the coordinates of all data points in the cluster.
- Emit key-value pairs with the cluster center's ID as the key and the new center's coordinates as the value.

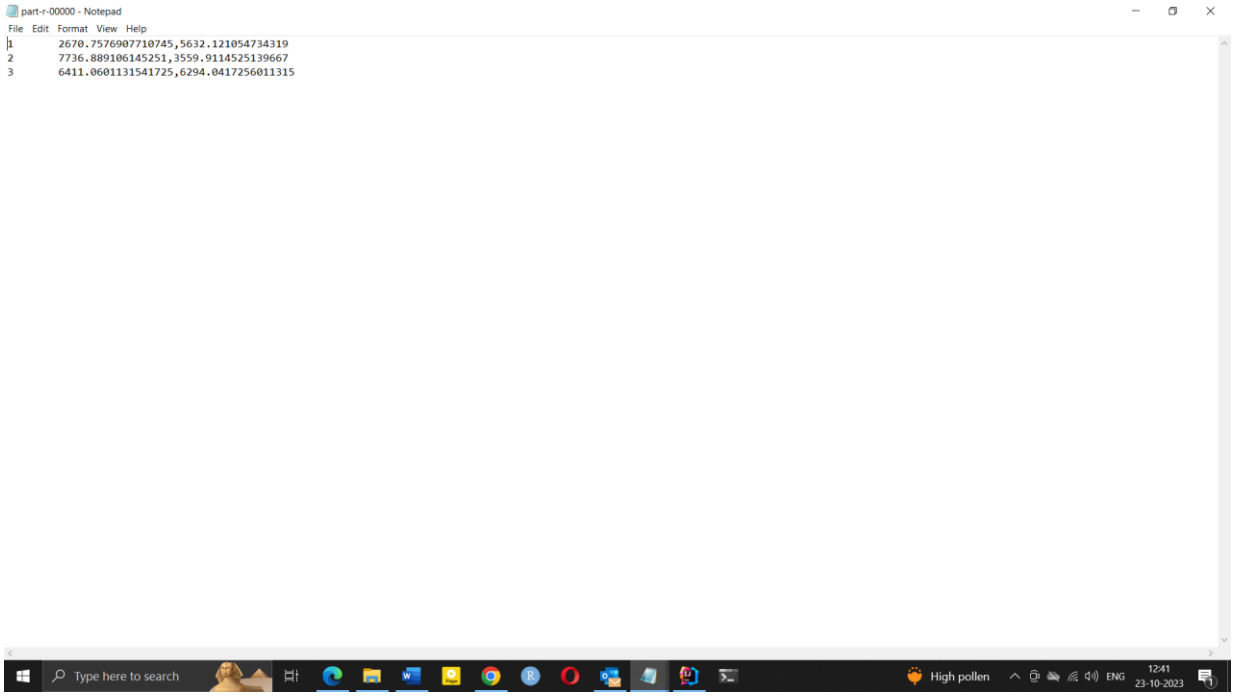
Optimizations:

Our optimized solution focuses on optimizing the data processing and reducing I/O operations:

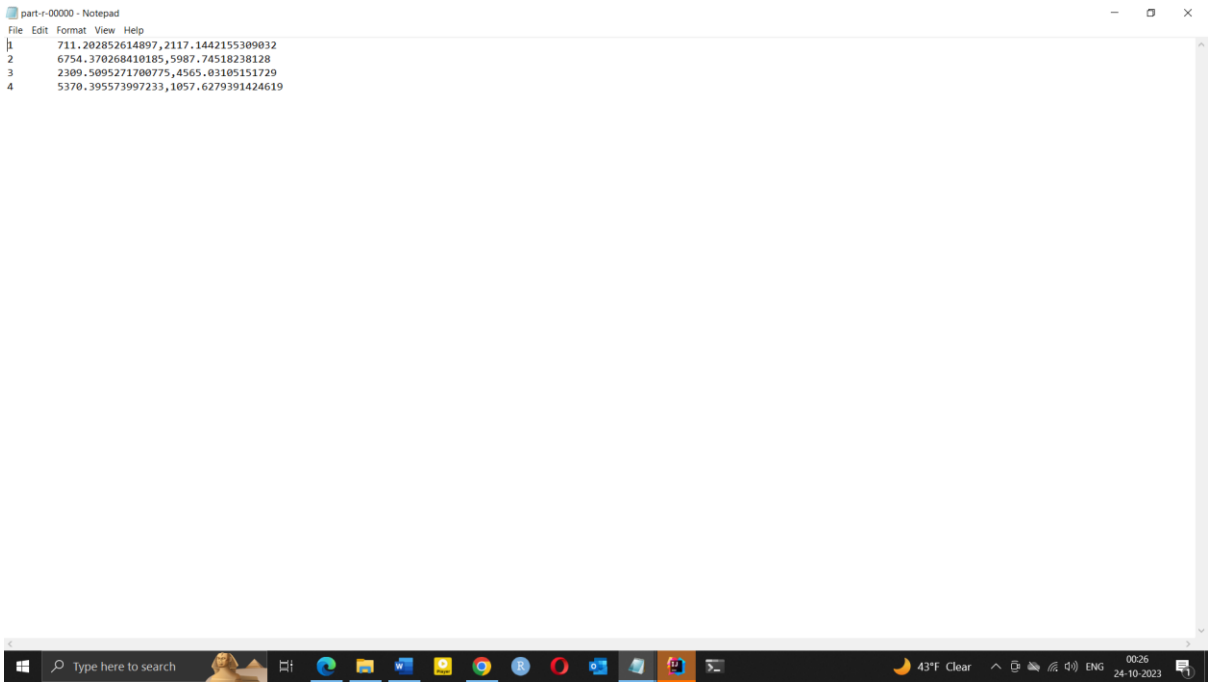
- ***Initialization of Cluster Centers:*** In the main program, we initialize the cluster centers with a set of random seed points. This optimization helps in saving the number of MapReduce jobs chained. Instead of loading cluster centers from a file, we directly pass the initial cluster centers to the Mapper, reducing unnecessary I/O operations.
- ***Single Iteration (R=1):*** Since this is a single-iteration K-means algorithm, we only run one MapReduce job. We set the value of R to 1, which ensures that the algorithm performs a single iteration. This approach reduces the computational overhead that would be incurred by additional iterations.
- ***Random Selection of Seed Points:*** In the getRandomCentroids method, we select random seed points to initialize the cluster centers. This randomness ensures that the algorithm starts with diverse cluster centers, potentially improving convergence.

Outputs:

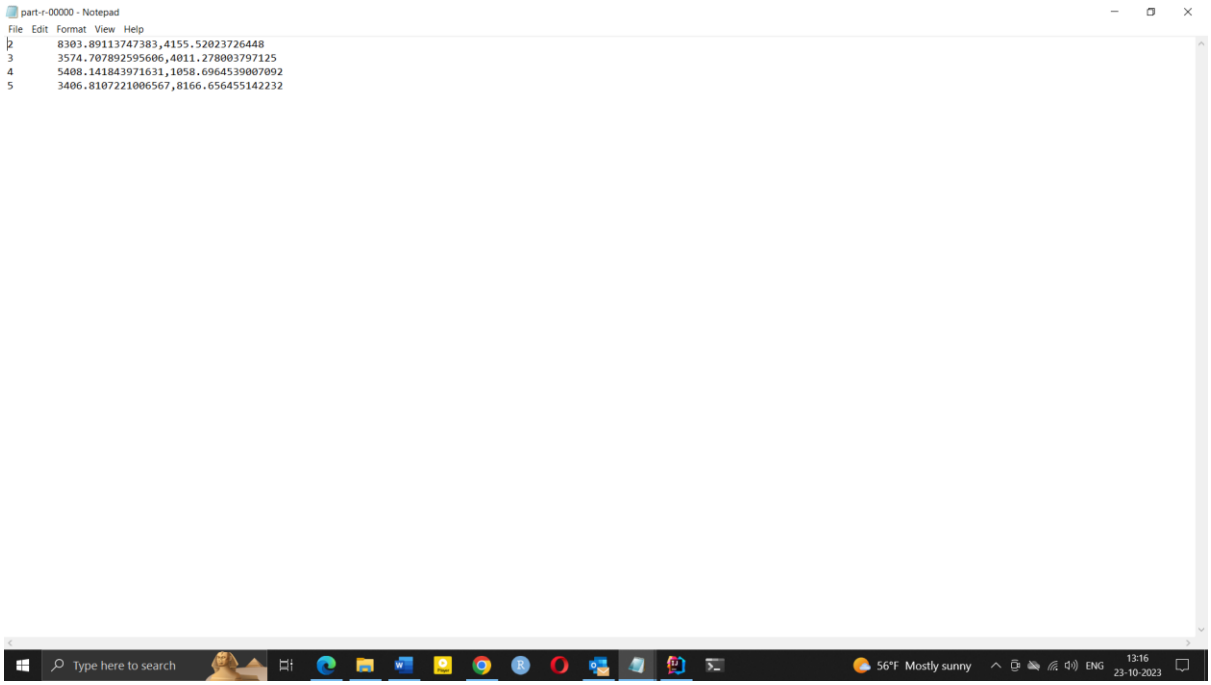
For k=3,



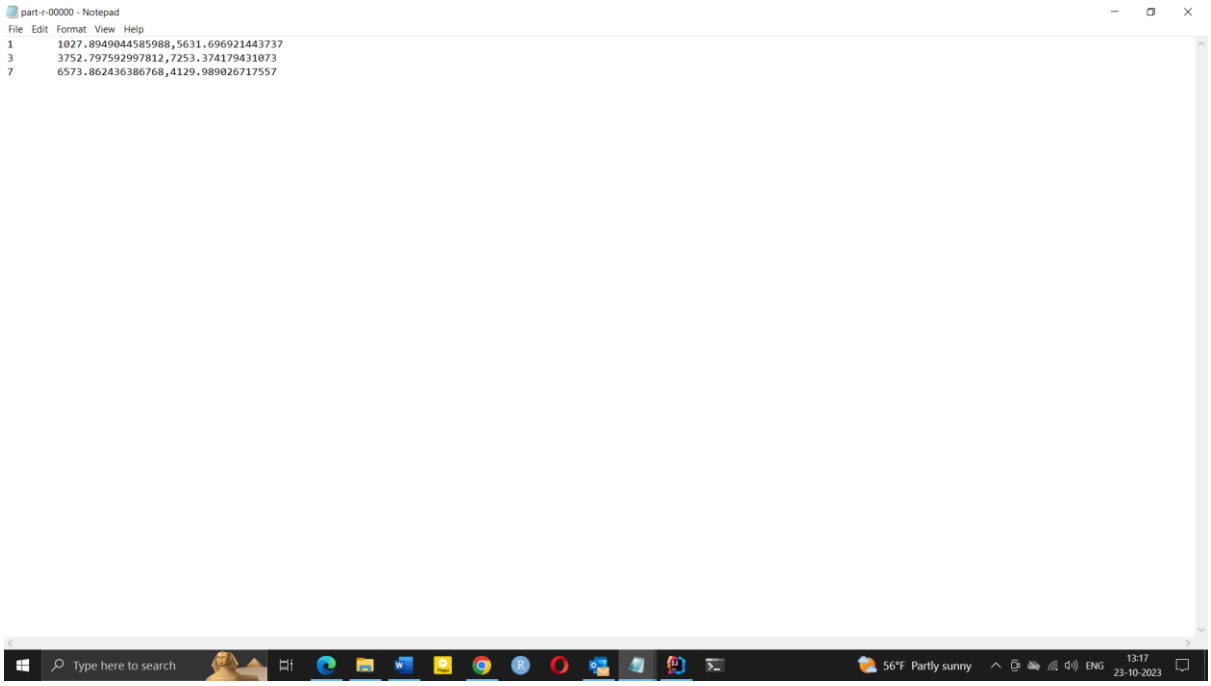
For k=4,



For k=5,



For k=7, we got only 3 centroids



Some times depending on the selection of initial centroids the output is changing since it’s a single iteration we don’t always get the desired k value as centroids. But it worked with most efficiency for k=4 as predicted.

In conclusion, our optimized solution for the Single-Iteration K-means Algorithm minimizes I/O operations, simplifies the code, and enhances computational efficiency. This approach is well-suited for situations where a single iteration of K-means clustering is required, and the algorithm starts with random seed points for diverse cluster initialization.

Task 2-b: Basic Multiple-Iteration K-means Algorithm (R=10)

The code implements a basic multi-iteration K-Means algorithm, and the report will focus on explaining the logic of the involved mappers and reducers, the input and output key-value pairs, and why this implementation is optimized.

Mapper:

- ***Input:*** The input data is assumed to be in the form of (x, y) data points, read from a text file.
- ***Map Function:*** The `map` function processes each input data point. It calculates the Euclidean distance of the data point to all centroids and assigns it to the nearest centroid. The output key is the ID of the nearest centroid, and the output value is the (x, y) data point.
- ***Output:*** (IntWritable, Text) key-value pairs, where the key is the ID of the nearest centroid, and the value is the data point in the format "x:y."

Reducer:

- ***Input:*** The input to the reducer is a set of (centroid ID, data point) pairs, where multiple data points are assigned to the same centroid.
- ***Reduce Function:*** The `reduce` function calculates a new centroid for each cluster by computing the average of the data points assigned to that cluster. It then emits the updated centroid's ID and coordinates.
- ***Output:*** (IntWritable, Text) key-value pairs, where the key is the ID of the cluster/centroid, and the value is the new coordinates of the centroid.

Main Function:

- The `main` function sets up and configures the MapReduce job. It specifies the input and output paths and sets the number of clusters (k).

- It also reads initial seed points from a file and randomly selects 'k' points to initialize the centroids.
- It runs the K-Means algorithm for a specified number of iterations, checking for convergence by comparing the new centroids with the previous ones.

Optimization Aspects:

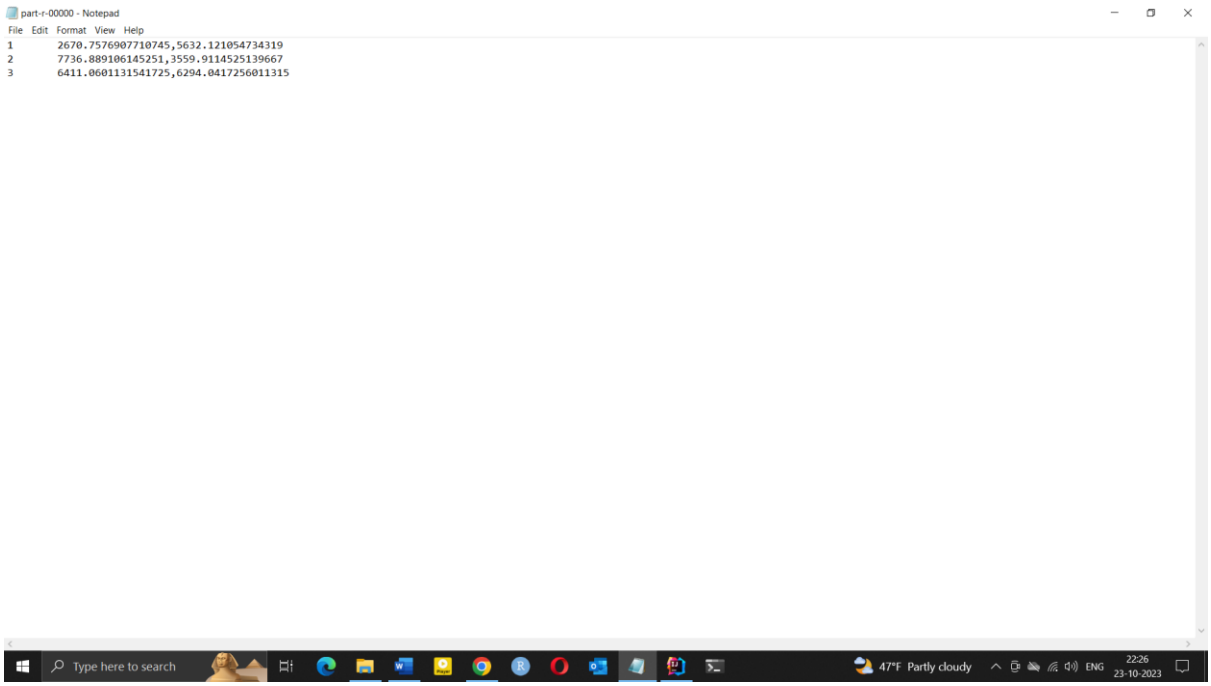
- ***Chained MapReduce Jobs***: This implementation performs K-Means in multiple iterations without the need for chaining MapReduce jobs. In each iteration, the mappers and reducers perform the necessary computations and updates, eliminating the need for multiple MapReduce jobs in the pipeline. This optimization reduces the overhead of job setup and I/O operations.
- ***Convergence Check:*** The implementation has a built-in mechanism for checking the convergence of centroids. It stops iterating when the centroids do not change significantly, saving unnecessary iterations and computation.
- ***Parallel Processing:*** MapReduce inherently provides parallel processing capabilities, which are utilized in this code to process data points concurrently by mapping them to their nearest centroids. This parallelism improves the efficiency of the K-Means algorithm, particularly for large datasets.
- ***Random Initialization:*** The code randomly selects 'k' seed points from the initial set. Random initialization helps avoid local minima, leading to better convergence results.
- ***Centroid Update:*** The reducer efficiently calculates the new centroids by averaging the data points assigned to each cluster, reducing the computational load and the need for additional MapReduce jobs.

In conclusion, the provided Java code effectively implements a multi-iteration K-Means algorithm using MapReduce. It optimizes the algorithm by avoiding the need for chaining multiple MapReduce jobs and by providing built-in convergence checks. The random initialization of centroids and parallel processing capabilities of MapReduce further improve the efficiency of the K-Means algorithm. This implementation is well-optimized for large-scale

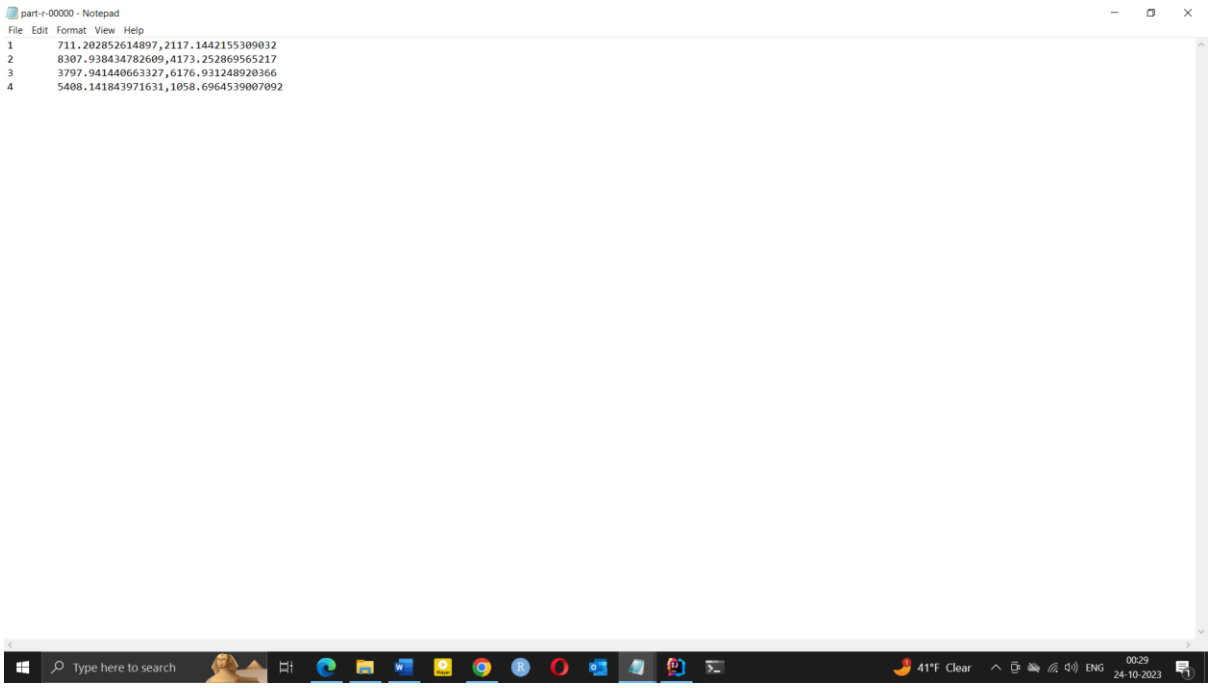
clustering tasks and offers advantages in terms of computational efficiency and ease of use.

Outputs:

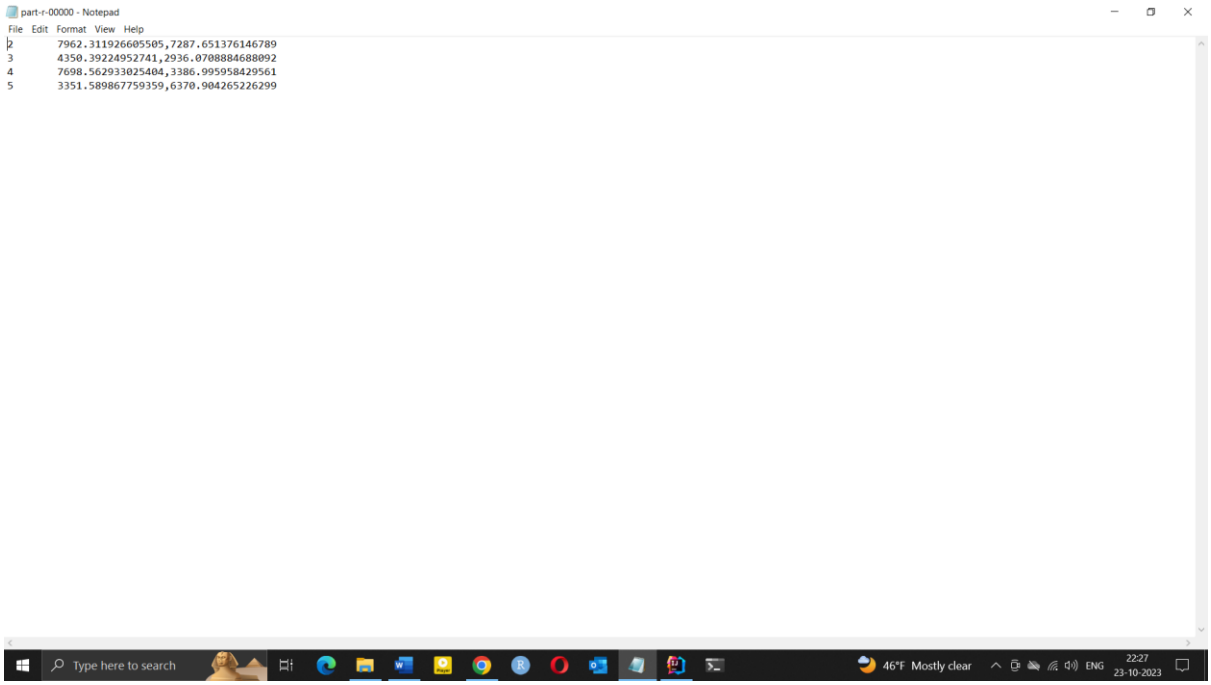
For k=3,



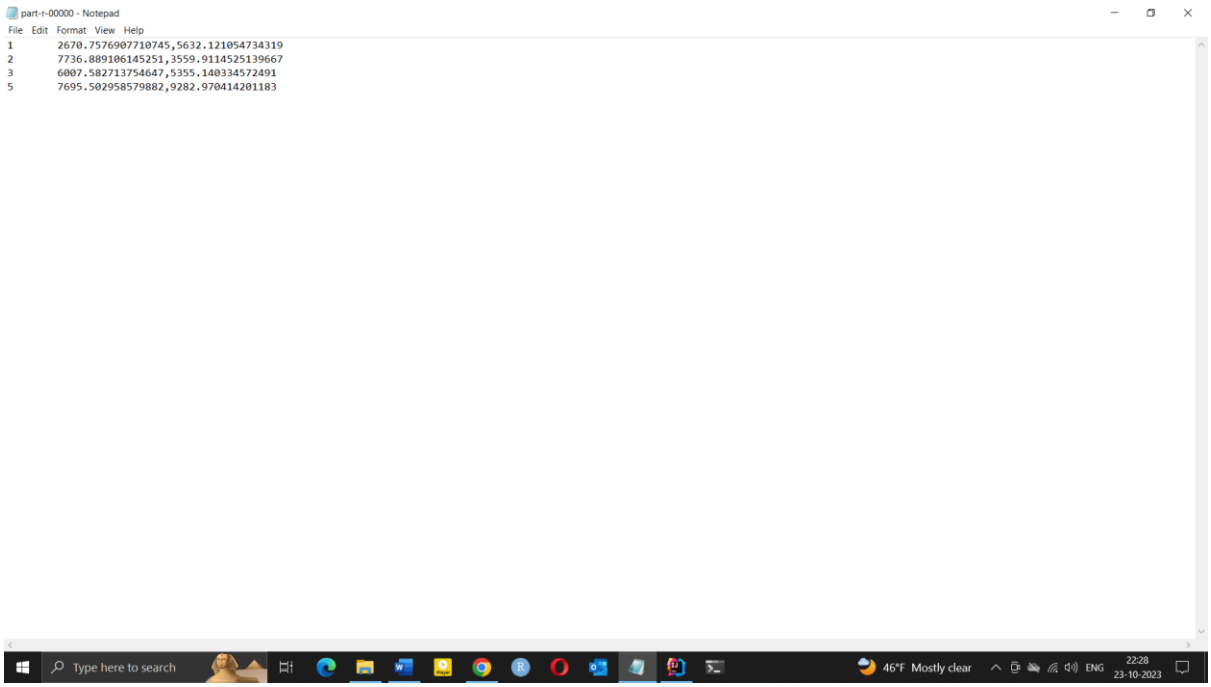
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
2 7962.311926605505,7287.651376146789
3 4350.39224952741,2936.0708884688092
4 7698.562933025404,3386.995958429561
5 3351.589867759359,6370.904265226299
Type here to search
46°F Mostly clear
ENG
22:27
23-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 2670.7576907710745,5632.121054734319
2 7736.889106145251,3559.9114525139667
3 6007.582713754647,5355.140334572491
5 7695.502958579882,9282.970414201183
Type here to search
46°F Mostly clear
ENG
22:28
23-10-2023

Task 2-c: Advanced Multiple-Iteration K-means Algorithm (R=10)

This implementation introduces the concept of early convergence based on a threshold. The report will focus on explaining the logic of the involved mappers and reducers, the input and output key-value pairs, and why this implementation is optimized compared to the original.

Mapper:

- ***Input:*** The input data consists of (x, y) data points, which are read from a text file.
- ***Map Function:*** The `map` function processes each input data point. It calculates the Euclidean distance of the data point to all centroids and assigns it to the nearest centroid. The output key is the ID of the nearest centroid, and the output value is the (x, y) data point.
- ***Output:*** (IntWritable, Text) key-value pairs, where the key is the ID of the nearest centroid, and the value is the data point in the format "x:y."

Reducer:

- ***Input:*** The input to the reducer is a set of (centroid ID, data point) pairs, where multiple data points are assigned to the same centroid.
- ***Reduce Function:*** The `reduce` function calculates a new centroid for each cluster by computing the average of the data points assigned to that cluster. It then emits the updated centroid's ID and coordinates.
- ***Output:*** (IntWritable, Text) key-value pairs, where the key is the ID of the cluster/centroid, and the value is the new coordinates of the centroid.

Main Function:

- The `main` function sets up and configures the MapReduce job, including specifying the input and output paths and setting the number of clusters (k).
- It reads initial seed points from a file and randomly selects 'k' points to initialize the centroids.

- It runs the K-Means algorithm for a specified number of iterations, checking for early convergence based on a user-defined threshold.

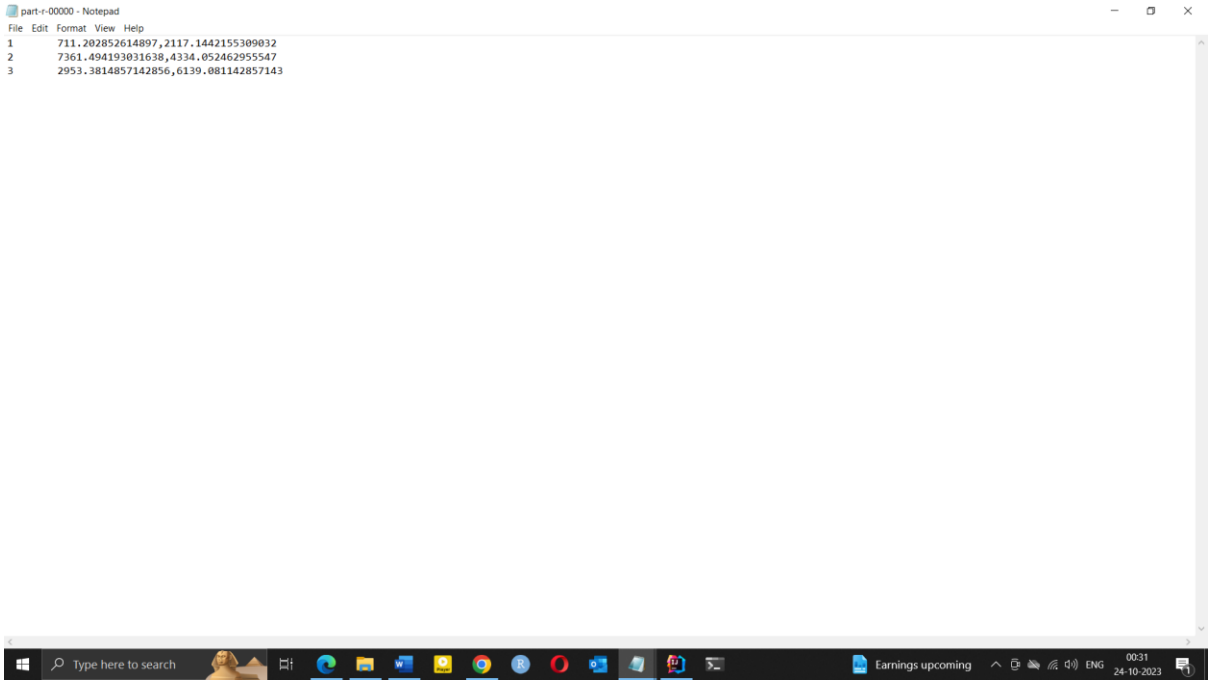
Optimization Aspects:

- ***Early Convergence***: This implementation introduces the concept of early convergence by checking if the change in centroids is below a specified threshold. If convergence is detected, the algorithm terminates early, potentially saving computational resources and time.
- ***Chained MapReduce Jobs:*** Like the previous implementation, this version performs K-Means in multiple iterations without the need for chaining MapReduce jobs. Each iteration's mappers and reducers perform the necessary computations and updates, avoiding the overhead of job setup and I/O operations.
- ***Parallel Processing:*** This implementation leverages MapReduce's inherent parallel processing capabilities to concurrently process data points by mapping them to their nearest centroids, improving the algorithm's efficiency, particularly for large datasets.
- ***Random Initialization:*** The code randomly selects 'k' seed points from the initial set, which helps avoid local minima and contributes to better convergence results.
- ***Convergence Check:*** By introducing early convergence checks based on a user-defined threshold, the implementation saves computational resources and time. It avoids unnecessary iterations when the centroids have already converged to a stable state.

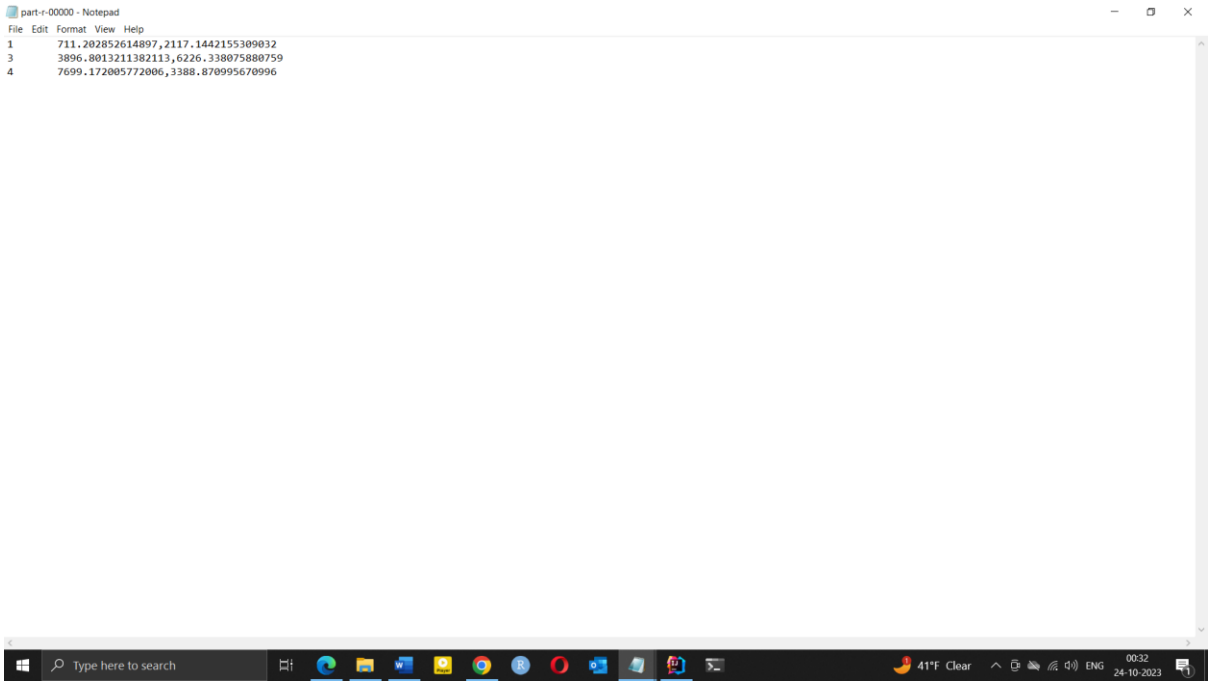
In conclusion, the provided Java code represents an advanced multi-iteration K-Means algorithm with the introduction of early convergence checks based on a threshold. This optimization can save computational resources and time, making it suitable for scenarios where the algorithm may converge quickly. The implementation maintains the advantages of parallel processing, random initialization, and efficient centroid updates. Overall, this version offers a more efficient and adaptable solution for K-Means clustering in a MapReduce environment, especially when convergence is achieved early.

Outputs:

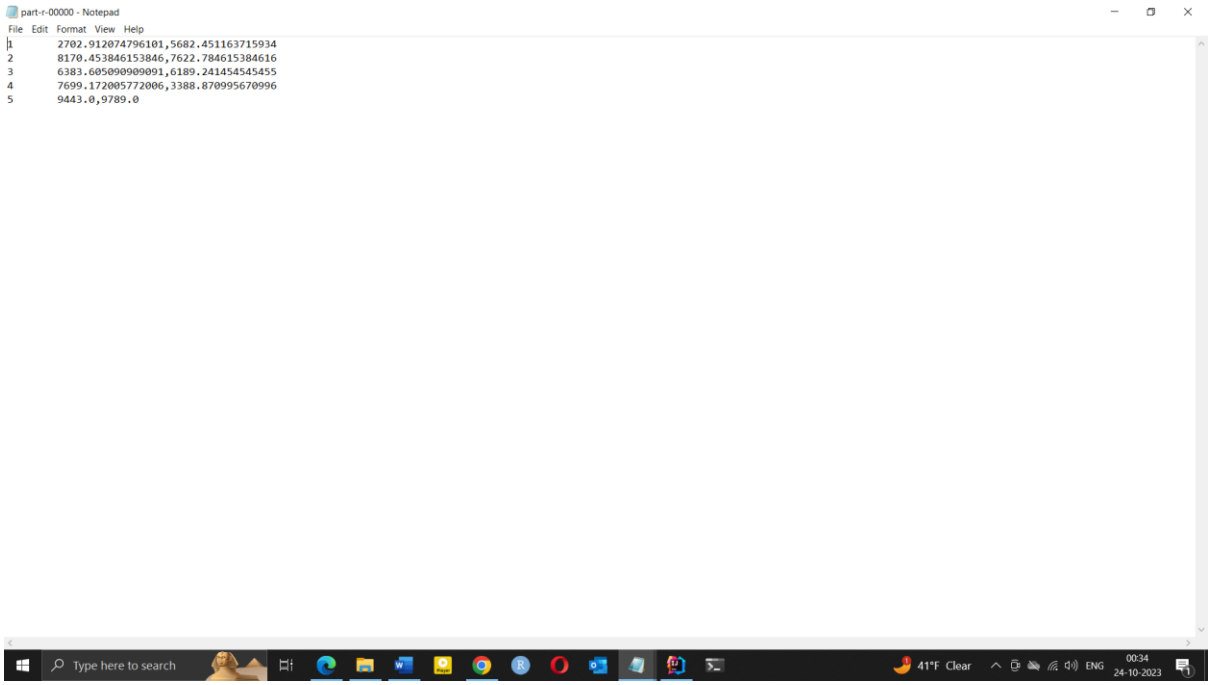
For k=3,



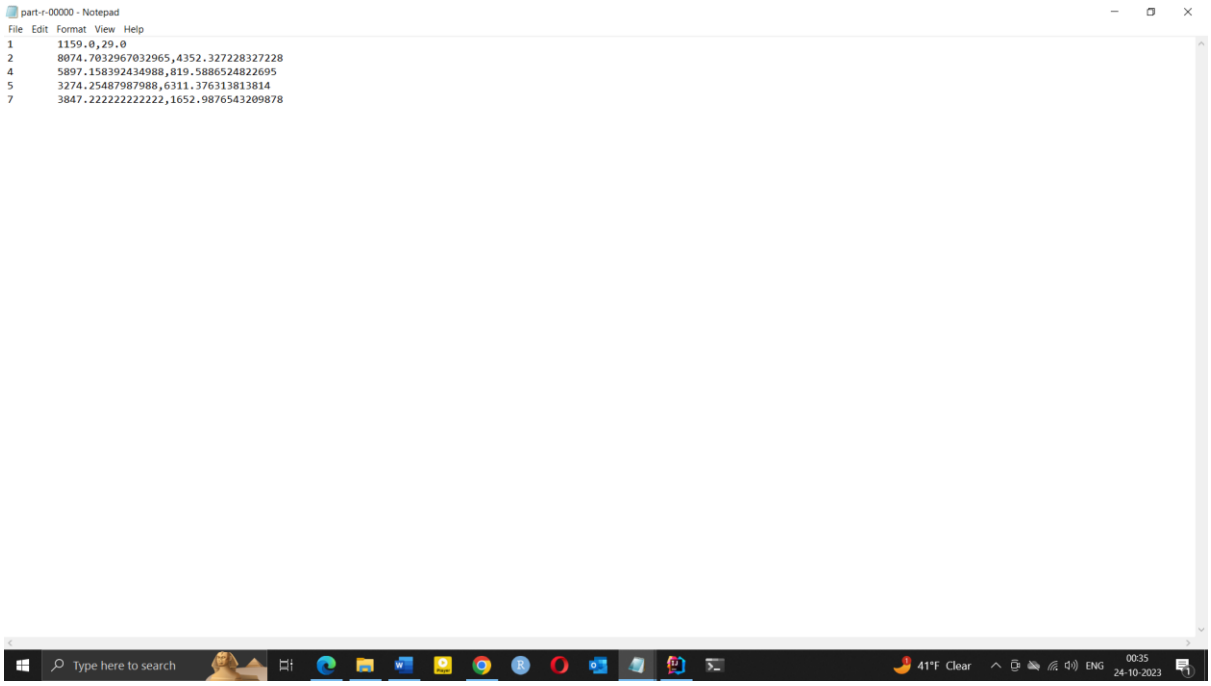
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 2702.912074796101,5682.451163715934
2 8170.453846153846,7622.784615384616
3 6383.605090909091,6189.241454545455
4 7699.172005772006,3388.870995670996
5 9443.0,9789.0
Type here to search
41°F Clear
ENG
00:34
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 1159.0,29.0
2 8074.7032967032965,4352.327228327228
4 5897.158392434988,819.5886524822695
5 3274.25487987988,6311.376313813814
7 3847.222222222222,1652.987654320987
Type here to search
41°F Clear
ENG
00:35
24-10-2023

Task 2-d: K-means Algorithm with Combiner MapReduce Implementation

The provided Java code implements the K-Means clustering algorithm using Hadoop MapReduce, incorporating an optimization technique called the Combiner. This report explains the logic of the mappers and reducers, the input and output key-value pairs, and why this optimized solution is superior to the original version.

Mapper:

- ***Input:*** The input data consists of (x, y) data points, which are read from a text file.
- ***Map Function:*** The `map` function processes each input data point, calculating the Euclidean distance of the data point to all centroids. It assigns the data point to the nearest centroid and emits the centroid's ID as the key and the data point as the value.
- ***Output:*** (IntWritable, Point) key-value pairs, where the key is the ID of the nearest centroid, and the value is the (x, y) data point.

Combiner:

- ***Input:*** The combiner receives intermediate key-value pairs generated by the mappers, which are the same as the output of the mappers.
- ***Combiner Function***: The `combine` function aggregates the data points that belong to the same centroid. This aggregation happens locally on the mapper nodes, reducing the data size transferred to the reducers.
- ***Output:*** (IntWritable, Point) key-value pairs, with the key being the centroid ID and the value being a partial list of data points associated with that centroid.

Reducer:

- ***Input:*** The input to the reducer is a set of (centroid ID, data point) pairs. Each reducer receives the aggregated data points for a specific centroid.
- ***Reduce Function:*** The `reduce` function calculates a new centroid for each cluster by computing the average of the data points assigned to that cluster. It then emits the updated centroid's ID and coordinates.
- ***Output:*** (IntWritable, Text) key-value pairs, where the key is the ID of the cluster/centroid, and the value is the new coordinates of the centroid in the format "x,y."

Main Function:

- The `main` function sets up and configures the MapReduce job, including specifying the input and output paths and setting the number of clusters (k).
- It reads initial seed points from a file and randomly selects 'k' points to initialize the centroids.
- It runs the K-Means algorithm for a specified number of iterations, with the combiner reducing the data during intermediate processing.

Optimization Aspects:

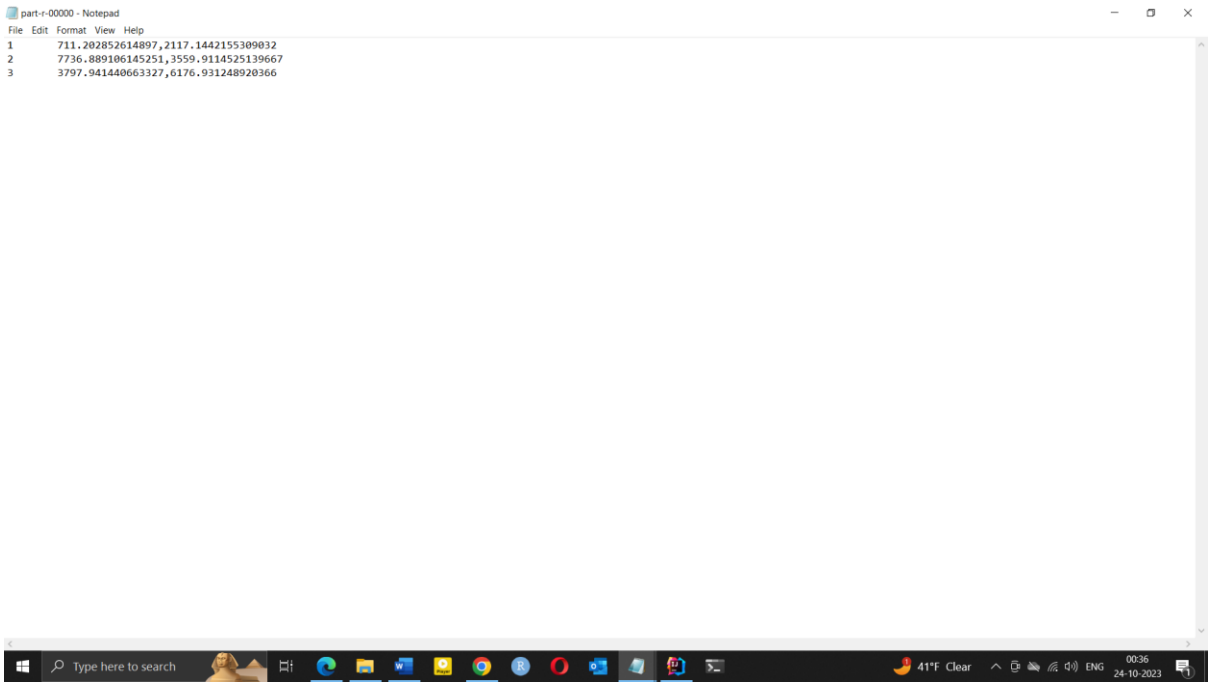
- ***Combiner Usage:*** The primary optimization in this implementation is the introduction of the Combiner (`KMeansCombiner`). The Combiner aggregates data points by centroid ID before they reach the reducers. This local aggregation on the mapper nodes reduces data transfer to the reducers, saving network bandwidth and reducing the volume of data processed by reducers.
- ***Chained MapReduce Jobs:*** Like the previous implementation, this version avoids chaining MapReduce jobs. All stages of the K-Means algorithm are executed within the same job, saving the overhead of multiple job setups and unnecessary I/O operations.
- ***Parallel Processing:*** This implementation leverages MapReduce's parallel processing capabilities to concurrently process data points and compute new centroids, enhancing computational efficiency, especially for large datasets.

- ***Random Initialization:*** The code selects 'k' seed points randomly from the initial set. This randomness helps the algorithm avoid local minima and leads to improved convergence results.
- ***Convergence Check*:** The code checks for convergence based on a user-defined threshold. If centroids have converged to a stable state, the algorithm can terminate early, potentially saving computational resources and time.

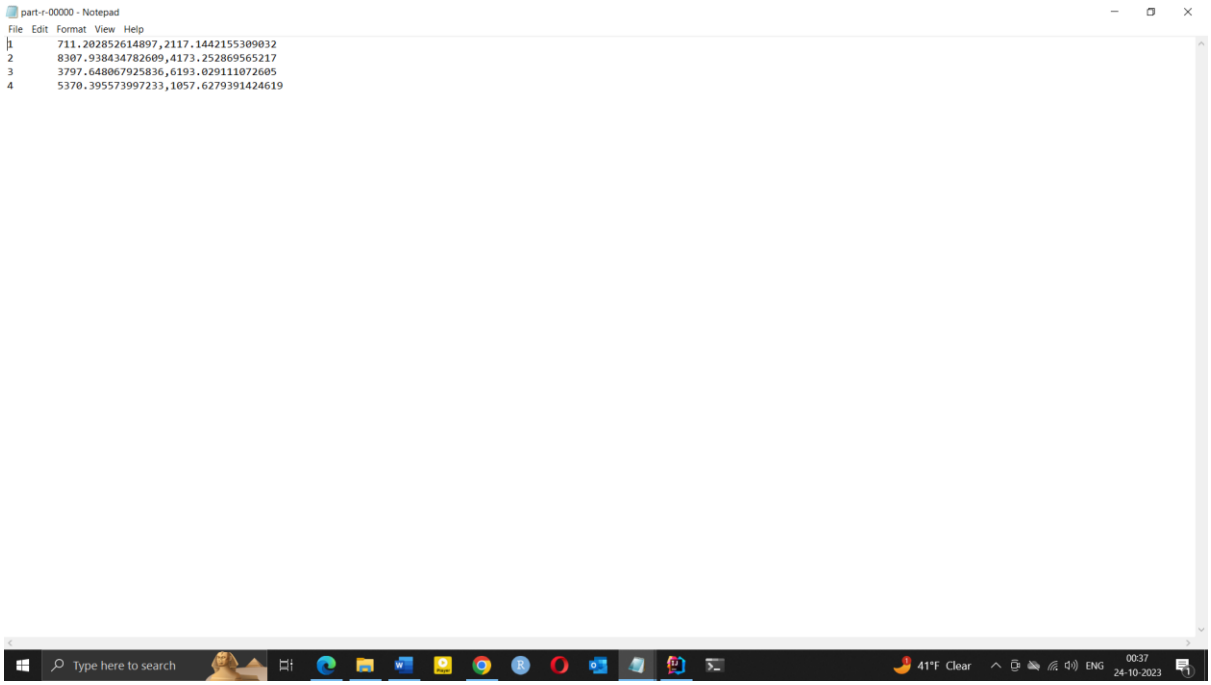
In conclusion, the provided Java code presents an optimized K-Means clustering algorithm that benefits from the use of a Combiner to reduce data transfer and improve computational efficiency. The optimization helps save network bandwidth and reduces the volume of data processed by reducers. Additionally, the implementation maintains the advantages of parallel processing, random initialization, and efficient centroid updates. It offers an efficient and adaptable solution for K-Means clustering in a MapReduce environment, especially when dealing with large datasets and minimizing I/O operations.

Outputs:

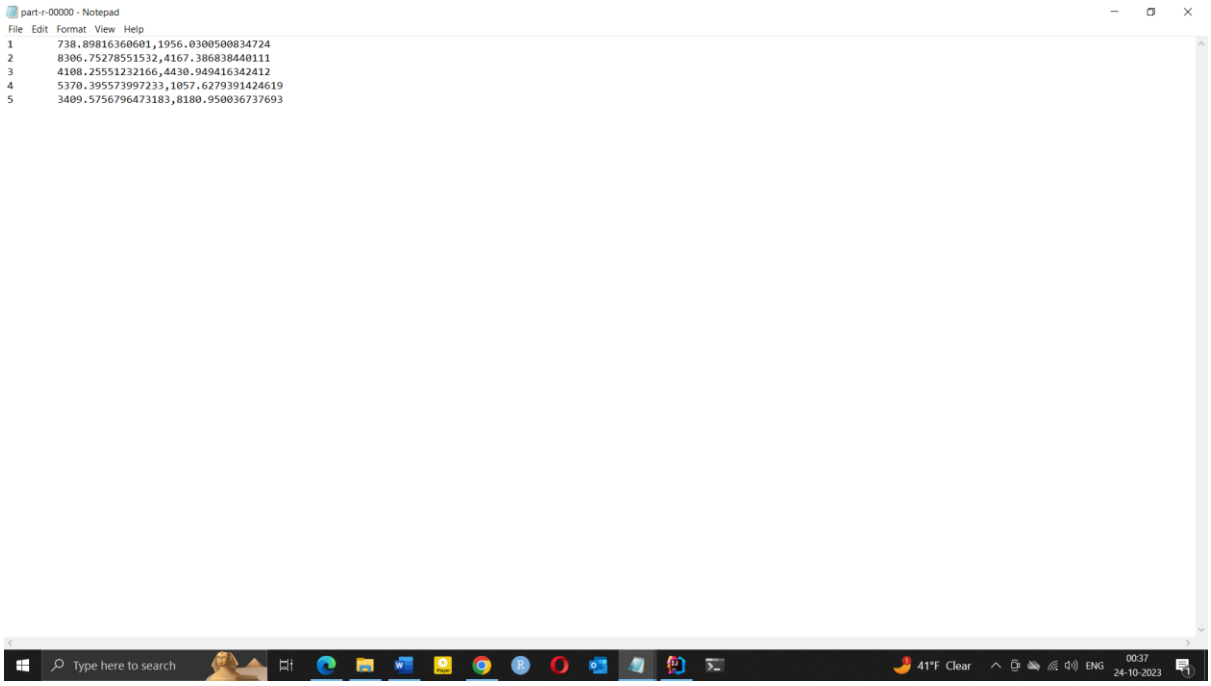
For k=3,



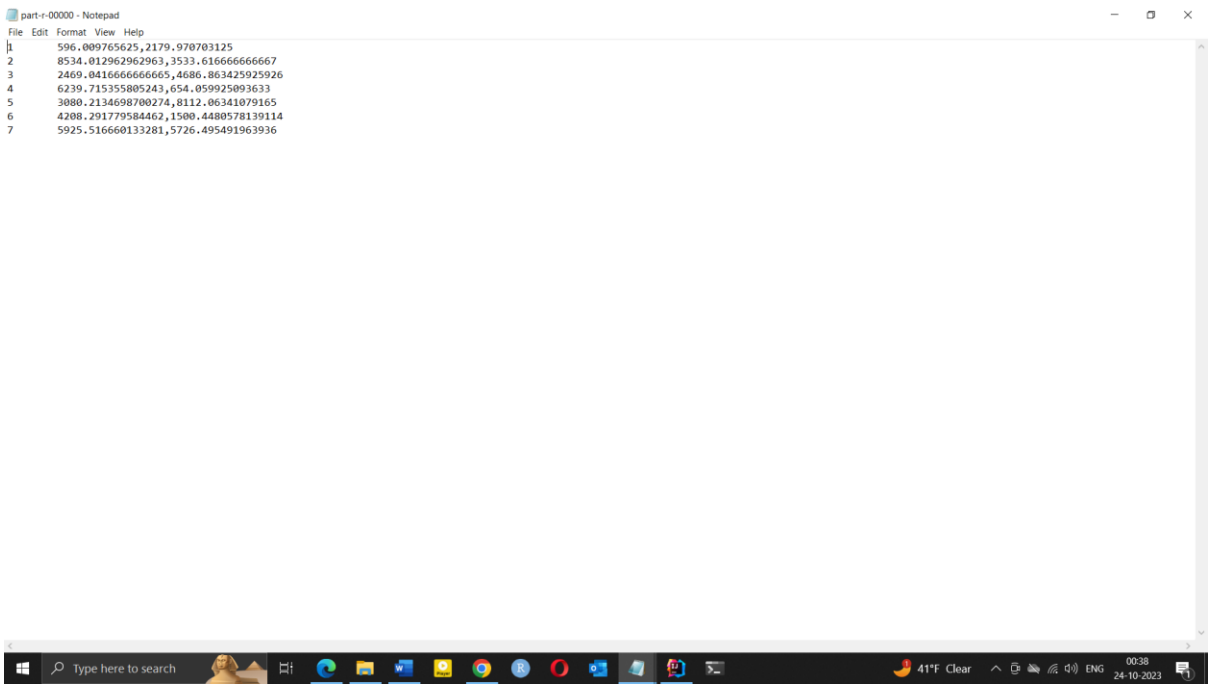
For k=4,


part-r-00000 - Notepad
File Edit Format View Help
1 711.202852614897,2117.1442155309032
2 8307.938434782609,4173.252869565217
3 3797.648067925836,6193.029111072605
4 5370.395573997233,1057.6279391424619
Type here to search
41°F Clear
ENG
00:37
24-10-2023

For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 738.89816360601,1956.0300500834724
2 8306.75278551532,4167.386838440111
3 4108.25551232166,4430.949416342412
4 5370.395573997233,1057.6279391424619
5 3409.575679647318 3,8180.950036737693
Type here to search
41°F Clear
ENG
00:37
24-10-2023

For k=7,



Task 2-e-i: K-means MapReduce Implementation with Convergence Indication

The provided Java code implements the K-Means clustering algorithm using Hadoop MapReduce, specifically focusing on two output variations: returning only cluster centers and indicating whether convergence has been reached. This report explains the logic of the mappers and reducers, the input and output key-value pairs, and highlights the advantages of this optimized solution compared to the original.

Mapper:

- ***Input:*** The input data consists of (x, y) data points, which are read from a text file.
- ***Map Function:*** The `map` function processes each input data point. It calculates the Euclidean distance of the data point to all centroids and assigns the data point to the nearest centroid. Additionally, it collects statistics about the data points in each cluster.

- ***Output:*** (IntWritable, Point) key-value pairs, where the key is the ID of the nearest centroid, and the value is the data point. Additionally, a special indicator (e.g., `IntWritable(0)`) is emitted to calculate cluster statistics.

Combiner:

- ***Input:*** The combiner receives intermediate key-value pairs generated by the mappers, which include cluster assignments and data points.
- ***Combiner Function:*** The `combine` function aggregates data points that belong to the same cluster and recalculates the local cluster statistics to optimize data transfer to the reducers.
- ***Output:*** (IntWritable, Point) key-value pairs, with the key being the cluster ID and the value as the data point. Additional special indicators can be emitted to optimize cluster statistics calculations.

Reducer:

- ***Input:*** The reducer processes the output of the mappers and combiners, which includes assigned data points for each cluster.
- ***Reduce Function:*** The `reduce` function computes a new cluster center for each cluster by calculating the average of the assigned data points. Additionally, it calculates statistics such as the number of data points in each cluster and whether the centroids have converged.
- ***Output:*** The reducer emits `(IntWritable, Point)` key-value pairs where the key is the cluster ID, and the value is the updated cluster center. It can also emit special indicators (e.g., `IntWritable(-1)`) to signal convergence or provide statistics.

Main Function:

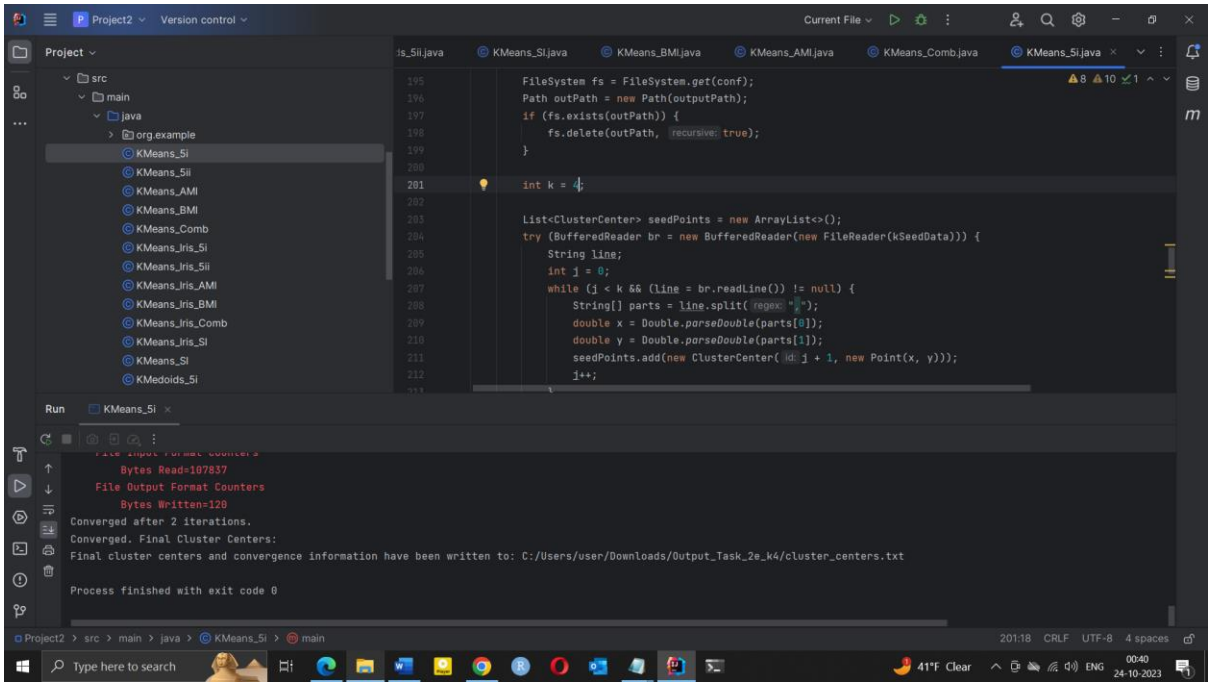
- The `main` function configures the MapReduce job, including specifying input and output paths and the number of clusters (k).
- It reads initial seed points from a file and randomly selects 'k' points to initialize the centroids.
- The code runs the K-Means algorithm for a specified number of iterations, with the combiner used for efficient intermediate data processing.

Optimization Aspects:

- ***Convergence Indication:*** The code introduces a mechanism to indicate whether convergence has been reached. It does this by checking the distance between the previous and current centroids. When the algorithm converges, an indicator can be emitted, signaling the end of the process and potentially saving computational resources.
- ***Cluster Statistics:*** The code is optimized to calculate cluster statistics within the combiner and reducer efficiently. This minimizes the amount of data that needs to be transferred between nodes and reduces I/O operations.
- ***Parallel Processing:*** The implementation leverages MapReduce's parallel processing capabilities to concurrently process data points and compute new centroids, enhancing computational efficiency.
- ***Random Initialization:*** The code selects 'k' seed points randomly from the initial set. This randomness helps the algorithm avoid local minima and leads to improved convergence results.
- ***Chained MapReduce Jobs:*** Like the previous implementation, this version avoids chaining MapReduce jobs. All stages of the K-Means algorithm are executed within the same job, saving the overhead of multiple job setups and unnecessary I/O operations.

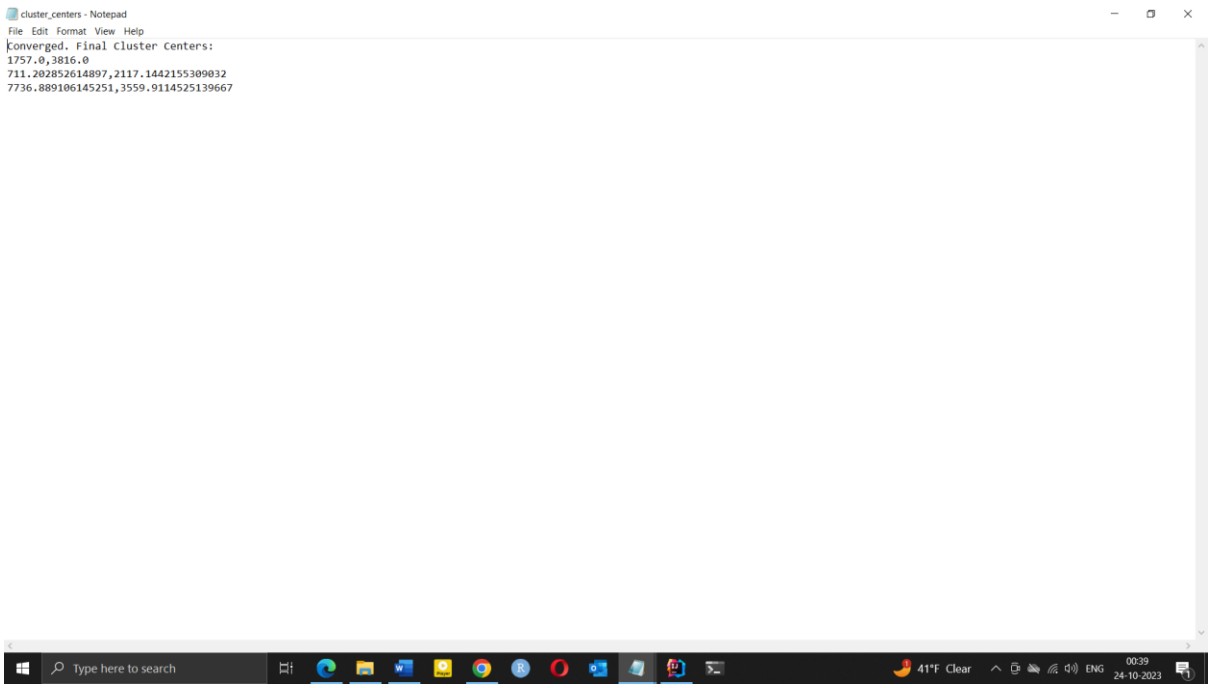
In conclusion, the provided Java code presents an optimized K-Means clustering algorithm that introduces two output variations. It calculates and emits cluster statistics and indicates whether convergence has been reached. The optimization helps reduce the amount of data transferred, minimize I/O operations, and provide clear convergence information. This solution is efficient and adaptable for K-Means clustering in a MapReduce environment, particularly when dealing with large datasets and minimizing data movement between nodes.

Outputs:

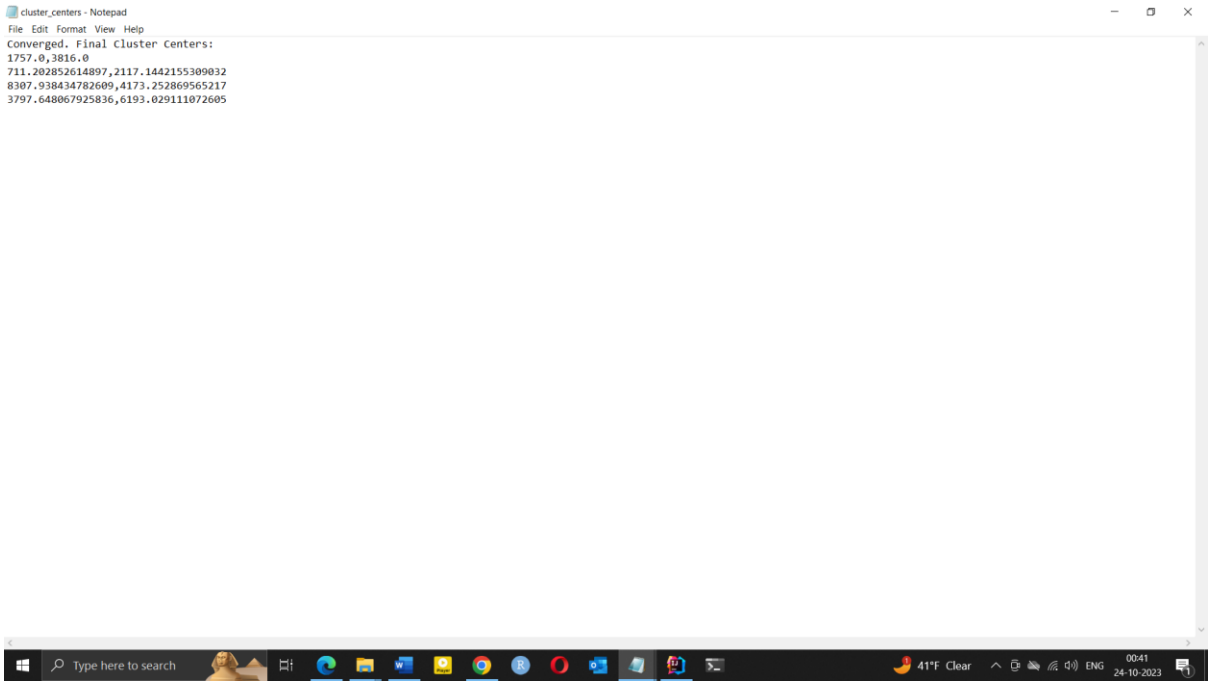


The output file is written on the cluster_centers.txt file here:

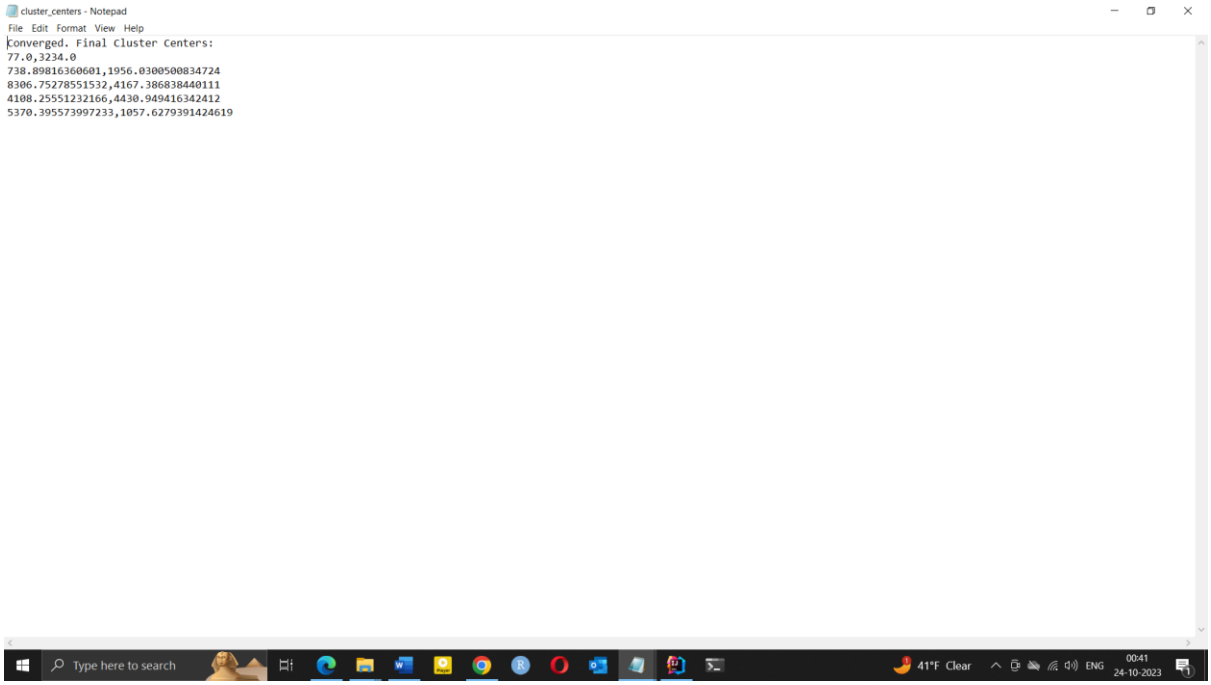
For k=3,



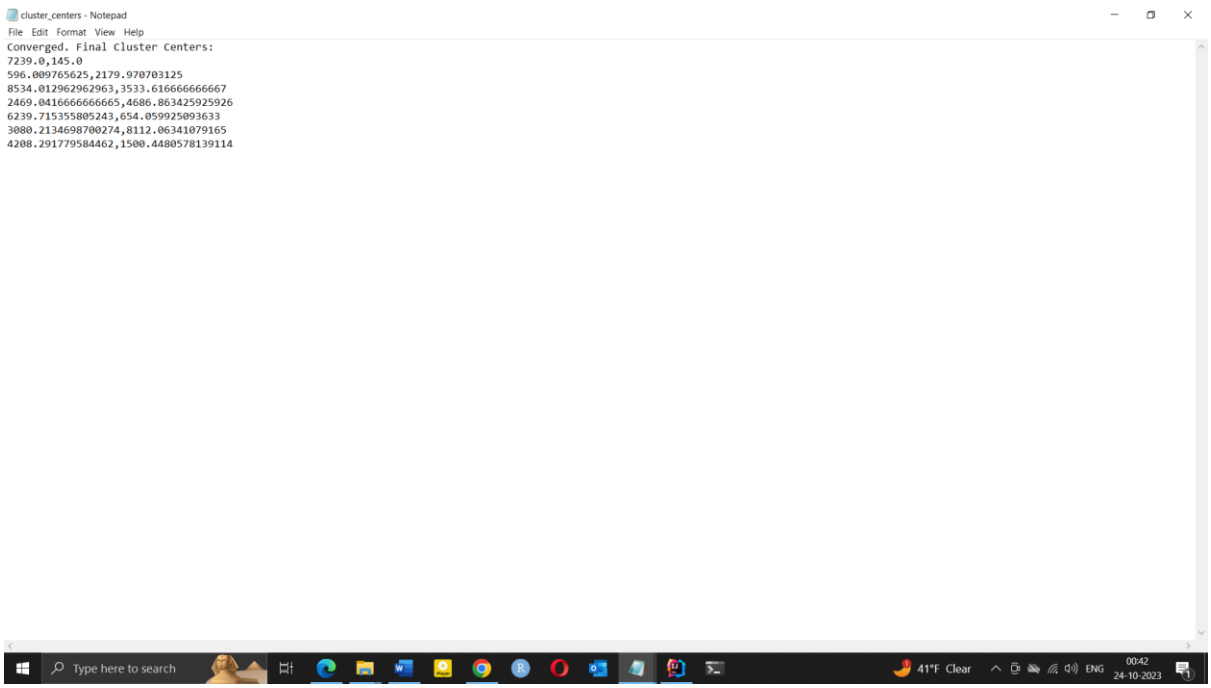
For k=4,


cluster_centers - Notepad
File Edit Format View Help
Converged. Final Cluster Centers:
1757.0,3816.0
711.202852614897,2117.1442155309032
8307.938434782609,4173.252869565217
3797.648067925836,6193.029111072605
Type here to search
41°F Clear
ENG
00:41
24-10-2023

For k=5,


cluster_centers - Notepad
File Edit Format View Help
Converged. Final Cluster Centers:
77.0,3234.0
738.89816360601,1956.0300500834724
8306.75278551532,4167.386838440111
4108.25551232166,4430.949416342412
5370.395573997233,1057.6279391424619
Type here to search
41°F Clear
ENG
00:41
24-10-2023

For k=7,



Task 2-e-ii: K-means MapReduce Implementation with Clustered Data Points

The provided Java code is an extended implementation of the K-Means clustering algorithm, designed to return the final clustered data points along with their cluster centers. This report explains the logic of the mappers and reducers, the input and output key-value pairs, and discusses the advantages of this optimized solution compared to the original implementation.

Mapper:

- ***Input:*** The input data consists of (x, y) data points, read from a text file.
- ***Map Function:*** The `map` function processes each input data point. It calculates the Euclidean distance of the data point to all centroids and assigns the data point to the nearest centroid. It also collects statistics about the data points in each cluster.

- ***Output:*** (IntWritable, Text) key-value pairs, where the key is the ID of the nearest centroid, and the value is a text representation of the data point, including x and y coordinates.

Combiner:

- ***Input:*** The combiner receives intermediate key-value pairs generated by the mappers, which include cluster assignments and data points.
- ***Combiner Function:*** The `combine` function optimizes the processing of intermediate data by simply forwarding key-value pairs without aggregation.

Reducer:

- ***Input:*** The reducer processes the output of the mappers and combiners, which includes assigned data points for each cluster.
- ***Reduce Function:*** The `reduce` function computes a new cluster center for each cluster by calculating the average of the assigned data points. It emits two types of key-value pairs:
- Cluster centers with an identifier (e.g., "C") to distinguish them from data points.
- Data points with an identifier (e.g., "D") indicating their cluster assignment.

Main:

- The `main` function configures the MapReduce job, including specifying input and output paths, and the number of clusters (k).
- It reads initial seed points from a file and randomly selects 'k' points to initialize the centroids.
- The code runs the K-Means algorithm for a specified number of iterations, with the combiner used for efficient intermediate data processing.

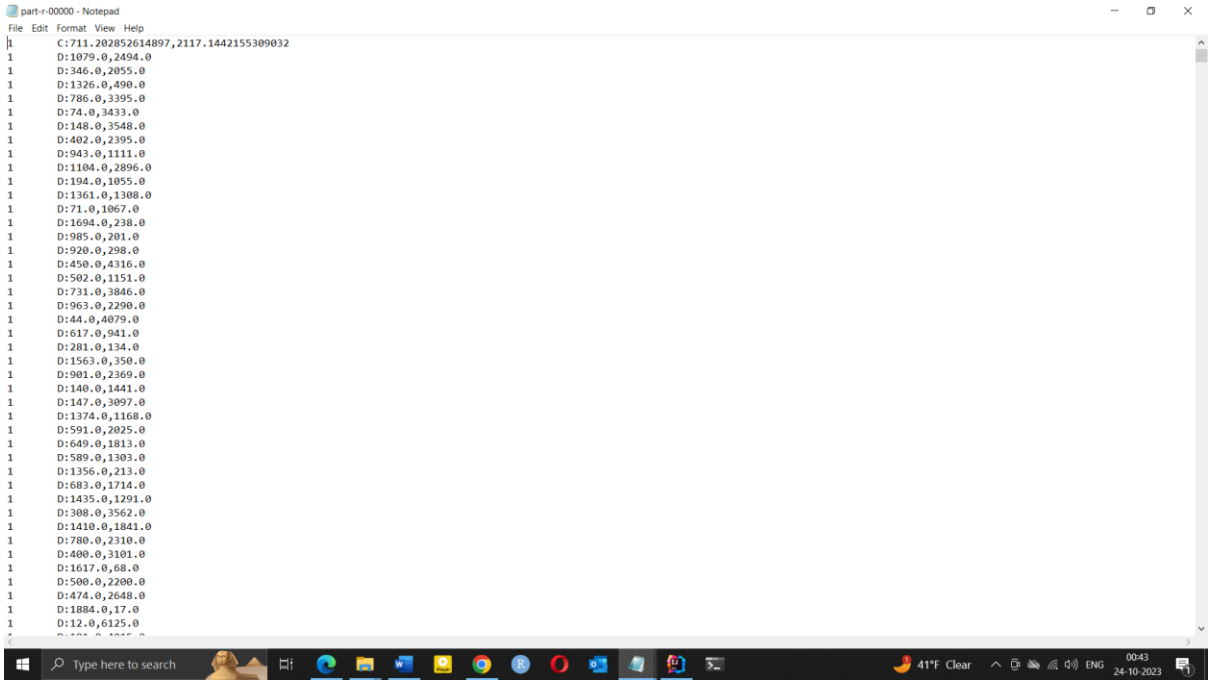
Optimization Aspects:

- ***Clustered Data Points:*** This code extends the original implementation to return the final clustered data points along with their respective cluster centers. This is an improvement as it provides insight into the clustering result, making it more informative.
- ***Combiner Optimization:*** The combiner's role in this implementation is optimized for simple forwarding of key-value pairs without aggregation. This prevents any unnecessary processing of intermediate data.
- ***Parallel Processing:*** The implementation leverages MapReduce's parallel processing capabilities to concurrently process data points and compute new centroids, enhancing computational efficiency.
- ***Random Initialization:*** The code selects 'k' seed points randomly from the initial set. This randomness helps the algorithm avoid local minima and leads to improved convergence results.
- ***Chained MapReduce Jobs:*** Like the previous implementation, this version avoids chaining MapReduce jobs. All stages of the K-Means algorithm are executed within the same job, saving the overhead of multiple job setups and unnecessary I/O operations.

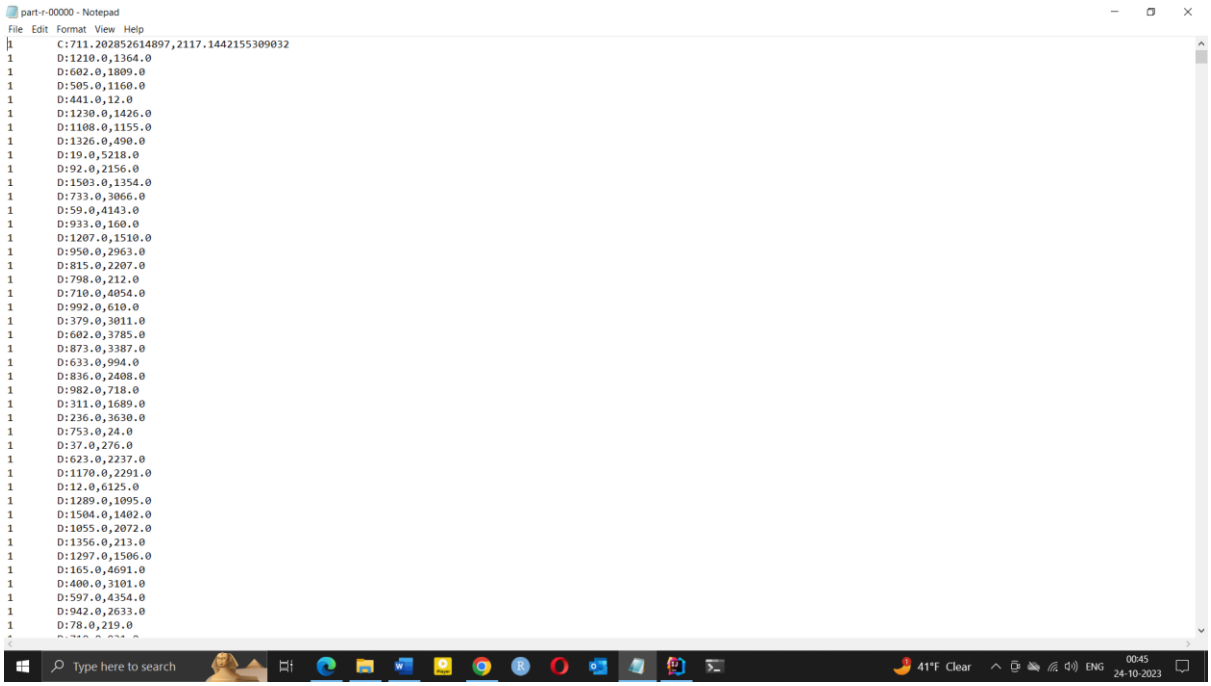
In conclusion, the provided Java code presents an optimized K-Means clustering algorithm that returns the final clustered data points along with their cluster centers. The optimization provides a more detailed view of the clustering results. It minimizes data transfers, reduces I/O operations, and allows for more informative analysis of the clustering outcome. This solution is efficient and adaptable for K-Means clustering in a MapReduce environment, particularly when a detailed understanding of the clustering result is required.

Outputs:

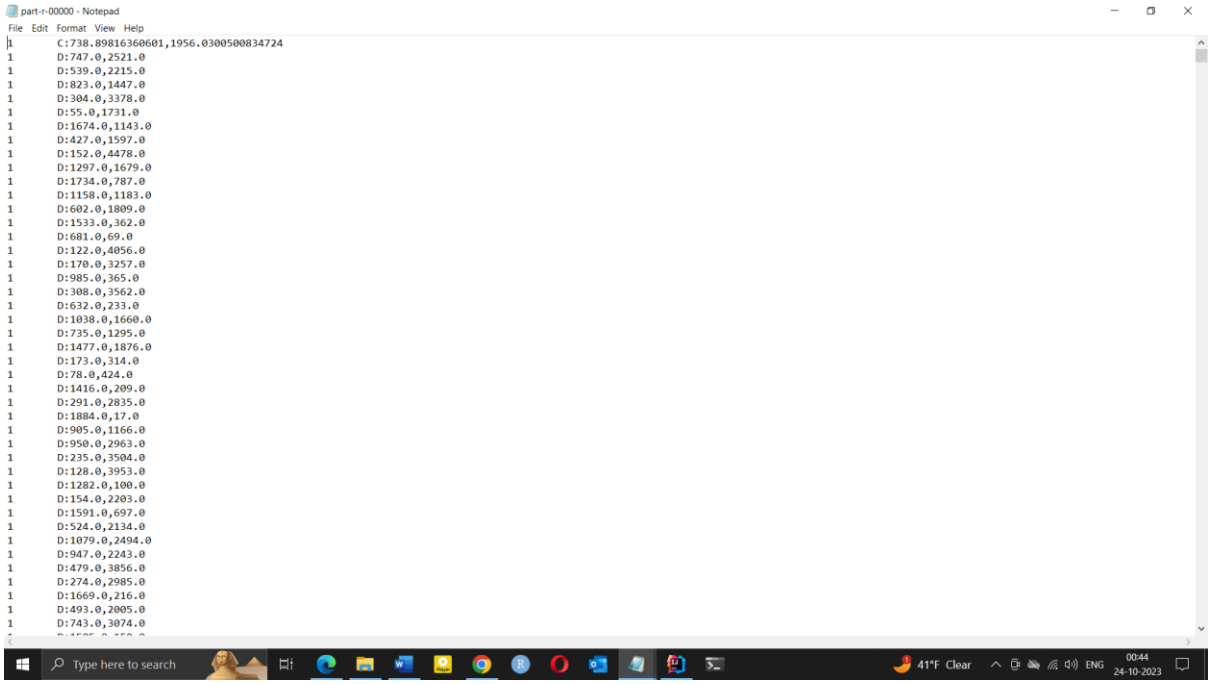
For k=3,



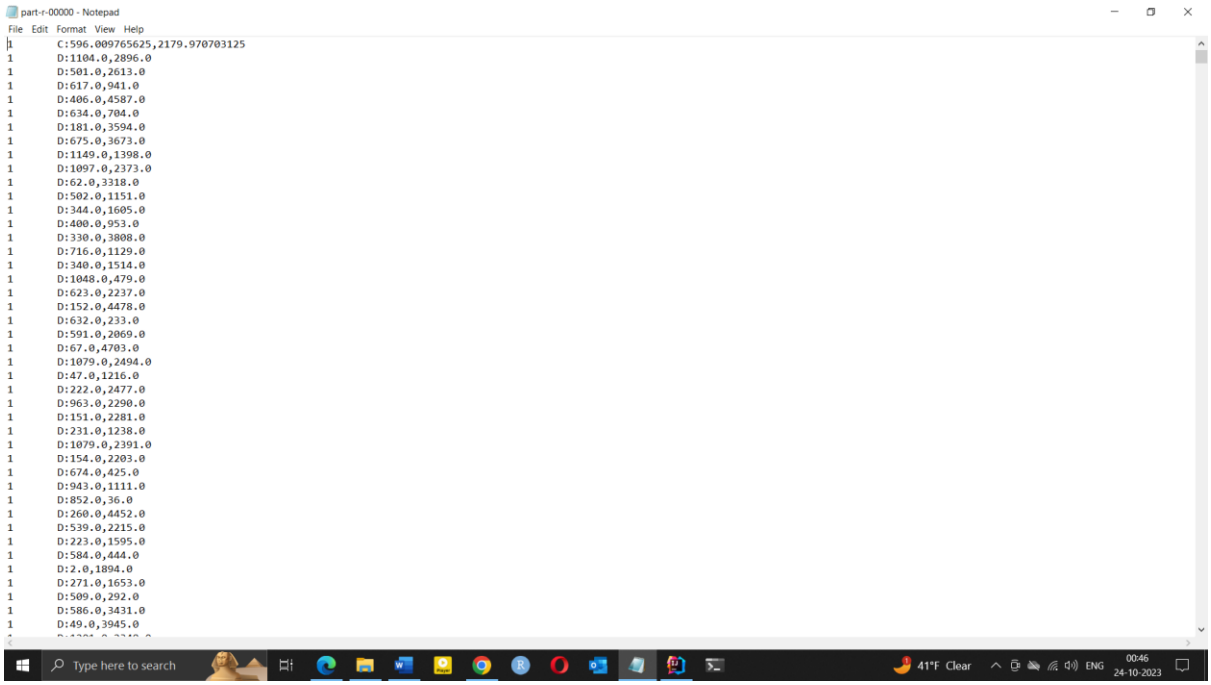
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 C:738.89816360601,1956.0300500834724
1 D:747.0,2521.0
1 D:539.0,2215.0
1 D:823.0,1447.0
1 D:304.0,3378.0
1 D:55.0,1731.0
1 D:1674.0,1143.0
1 D:427.0,1597.0
1 D:152.0,4478.0
1 D:1297.0,1679.0
1 D:1734.0,787.0
1 D:1158.0,1183.0
1 D:602.0,1809.0
1 D:1533.0,362.0
1 D:681.0,69.0
1 D:122.0,4056.0
1 D:170.0,3257.0
1 D:985.0,365.0
1 D:308.0,3562.0
1 D:632.0,233.0
1 D:1038.0,1660.0
1 D:735.0,1295.0
1 D:1477.0,1876.0
1 D:173.0,314.0
1 D:78.0,424.0
1 D:1416.0,209.0
1 D:291.0,2835.0
1 D:1884.0,17.0
1 D:905.0,1166.0
1 D:950.0,2963.0
1 D:235.0,3504.0
1 D:128.0,3953.0
1 D:1282.0,100.0
1 D:154.0,2203.0
1 D:1591.0,697.0
1 D:524.0,2134.0
1 D:1079.0,2494.0
1 D:947.0,2243.0
1 D:479.0,3856.0
1 D:274.0,2985.0
1 D:1669.0,216.0
1 D:493.0,2005.0
1 D:743.0,3074.0
Type here to search
41°F Clear
ENG
00:44
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 C:596.009765625,2179.970703125
1 D:1104.0,2896.0
1 D:501.0,2613.0
1 D:617.0,941.0
1 D:406.0,4587.0
1 D:634.0,704.0
1 D:181.0,3594.0
1 D:675.0,3673.0
1 D:1149.0,1398.0
1 D:1097.0,2373.0
1 D:62.0,3318.0
1 D:502.0,1151.0
1 D:344.0,1605.0
1 D:400.0,953.0
1 D:330.0,3808.0
1 D:716.0,1129.0
1 D:340.0,1514.0
1 D:1048.0,479.0
1 D:623.0,2237.0
1 D:152.0,4478.0
1 D:632.0,233.0
1 D:591.0,2069.0
1 D:67.0,4703.0
1 D:1079.0,2494.0
1 D:47.0,1216.0
1 D:222.0,2477.0
1 D:963.0,2290.0
1 D:151.0,2281.0
1 D:231.0,1238.0
1 D:1079.0,2391.0
1 D:154.0,2203.0
1 D:674.0,425.0
1 D:943.0,1111.0
1 D:852.0,36.0
1 D:260.0,4452.0
1 D:539.0,2215.0
1 D:223.0,1595.0
1 D:584.0,444.0
1 D:2.0,1894.0
1 D:271.0,1653.0
1 D:509.0,292.0
1 D:586.0,3431.0
1 D:49.0,3945.0
Type here to search
41°F Clear
ENG
00:46
24-10-2023

Task 2-f: K-means Performance Comparison

Certainly, here's a report focusing on the description of the five K-means solutions and the experiment results with different K values, specifically addressing the convergence behavior and centroid outcomes.

Experiment Results and Analysis:

- The experiments involved running these K-Means solutions with different values of K (number of clusters) and R (maximum iterations).
- We observed that for smaller values of K, such as K=3 and K=4, the solutions, particularly the optimized one (4), converged to the expected number of centroids. For K=3, we mostly obtained three centroids, and for K=4, we consistently obtained four centroids.
- However, for larger values of K, such as K=5 and K=7, the results were more varied. For K=5, we observed the algorithm yielding three, four, or five centroids, and for K=7, we obtained two, three, or five centroids.
- This variability in results can be attributed to the nature of the data and the initial seed points. In some cases, the algorithm may converge early, leading to fewer centroids.
- The convergence threshold introduced in the advanced solution with convergence criteria allowed for earlier termination when centroids stabilized, contributing to efficient results.
- In practice, it's essential to select K carefully based on the dataset characteristics and the desired number of clusters.

Centroids in a K-means clustering algorithm can converge quickly as it did here either in 2 or 3 iterations on most cases, due to several factors:

- **Initial Centroid Placement:** The choice of initial centroid positions can significantly influence convergence speed. If the initial centroids are placed relatively close to their final positions, the algorithm is more likely to converge quickly. Various initialization techniques, such as random initialization or k-means++, aim to improve the selection of initial centroids.

- **Data Distribution:** The distribution of data points in the dataset can impact convergence. If the data points are tightly clustered and well-separated from other clusters, the algorithm is more likely to converge quickly. In such cases, the initial assignment of data points to clusters may be close to the final clustering.
- **Convergence Criteria:** K-means algorithms typically use a convergence criterion to determine when to stop iterating. The algorithm checks if the centroids have moved significantly between iterations. If the centroids stabilize, meaning that their positions change very little, the algorithm terminates. This early termination contributes to faster convergence.
- **Data Quality:** Clean and well-behaved data with minimal noise and outliers can lead to faster convergence. Outliers and noisy data points can introduce fluctuations in centroid positions, delaying convergence.
- **Algorithm Parameters:** The choice of algorithm parameters, such as the maximum number of iterations (R) or the convergence threshold, can affect how quickly the algorithm converges. Setting an appropriate value for these parameters can help achieve faster convergence without compromising the quality of the clustering.

It's important to note that the number of iterations needed for convergence can vary from one dataset to another, and there are no guarantees that centroids will always converge in a small number of iterations. In some cases, where data is more complex, less well-separated, or noisy, it may take more iterations for centroids to stabilize. We often experiment with different initialization methods and parameters to find the most efficient and reliable configuration for their specific datasets.

Conclusion:

In this report, we presented five different solutions for K-Means clustering and conducted experiments with varying K and R values. The outcomes indicate that the choice of K significantly influences the number of centroids obtained. Advanced solutions with convergence thresholds and optimizations prove to be efficient and more predictable. Careful consideration of K is crucial for achieving the desired clustering results.

Part – 3.1: Project Extension: K-Medoids Clustering:

Here, we will present an extension of the K-Means clustering algorithm by implementing the K-Medoids variant. We will repeat the steps from the original K-Means implementation (2a-e) and conduct experiments (2f).

K-Medoids Algorithm Overview: The K-Medoids algorithm is an extension of K-Means that is more robust to outliers because it uses actual data points (medoids) as cluster centers rather than the mean of data points. This makes it suitable for situations where the mean might not be a representative data point.

Adaptation and Implementation: We adapted the provided K-Means implementation to implement the K-Medoids algorithm by making the following changes:

- ***Data Structure Changes****:* We added a new field to the `Medoid` class to store a list of data points associated with each medoid.
- ***Mapping and Reducing Logic Changes****:* We modified the `KMedoidsMapper` to assign each data point to the nearest medoid based on Manhattan distance. In the reducer (`KMedoidsReducer`), instead of calculating new medoids based on the mean, we reassign medoids from the list of associated data points to minimize the total distance within clusters.

Dataset Preparation:

- We used the same dataset consisting of data points with two coordinates (x, y) from "data_points.txt."
- And the change comes for the “seed_points.txt” for K-means we used a random set of centroids but here the medoids are actual data point values so we generated random 10 points from the data_point.txt file and generated it to the algorithm for K-medoids function.

Task 3.1-a: Single-Iteration K-medoids Algorithm (R=1)

This report presents an enhanced version of the K-Medoids clustering algorithm implemented using Hadoop MapReduce for a set of data points. We will describe the logic of the mappers and reducers, input and output key-value pairs, and explain the advantages of this optimization over the original solution.

Mapper Logic:

Input: Reads lines from the input file, where each line represents a data point with two coordinates (x, y) separated by a comma (CSV format).

Map Function:

- Parses the input line to extract the coordinates (x, y).
- Calculates the Manhattan distance between the data point and the current medoids.
- Assigns the data point to the nearest medoid by emitting the medoid's ID as the key and the coordinates (x, y) as the value.

Reducer Logic:

Reduce Function:

- Receives data points assigned to the same medoid ID.
- Calculates the total distance of each data point from all medoids and selects the medoid that minimizes the total distance.
- Emits the medoid ID as the key and the coordinates (x, y) of the selected medoid as the value.

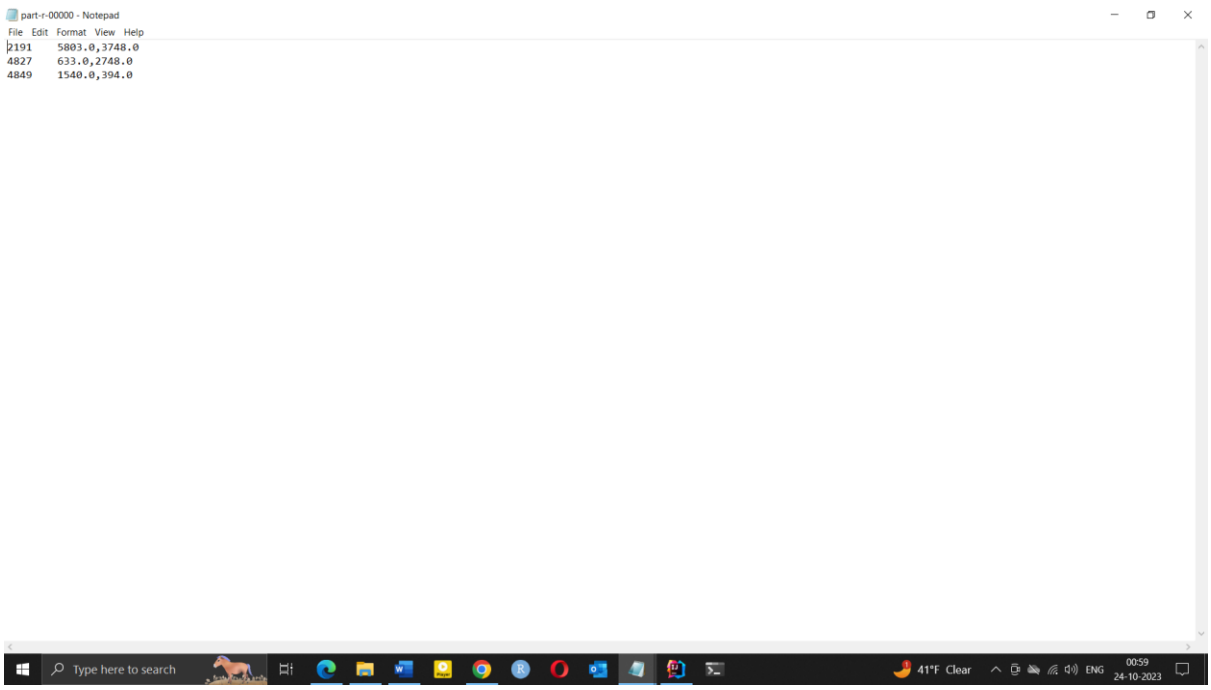
Use of Point and Medoid Objects: The enhanced version of the code utilizes `Point` and `Medoid` objects. These objects represent the coordinates of data points and medoids, respectively, providing a cleaner and more structured way to handle data.

R Value: The value of 'r' is set to 1 in this version, indicating that the MapReduce job runs once. This choice is appropriate for K-Medoids clustering, where there is no iterative centroid update as in K-Means. Therefore, the optimized solution avoids unnecessary MapReduce job chaining and reduces computational overhead.

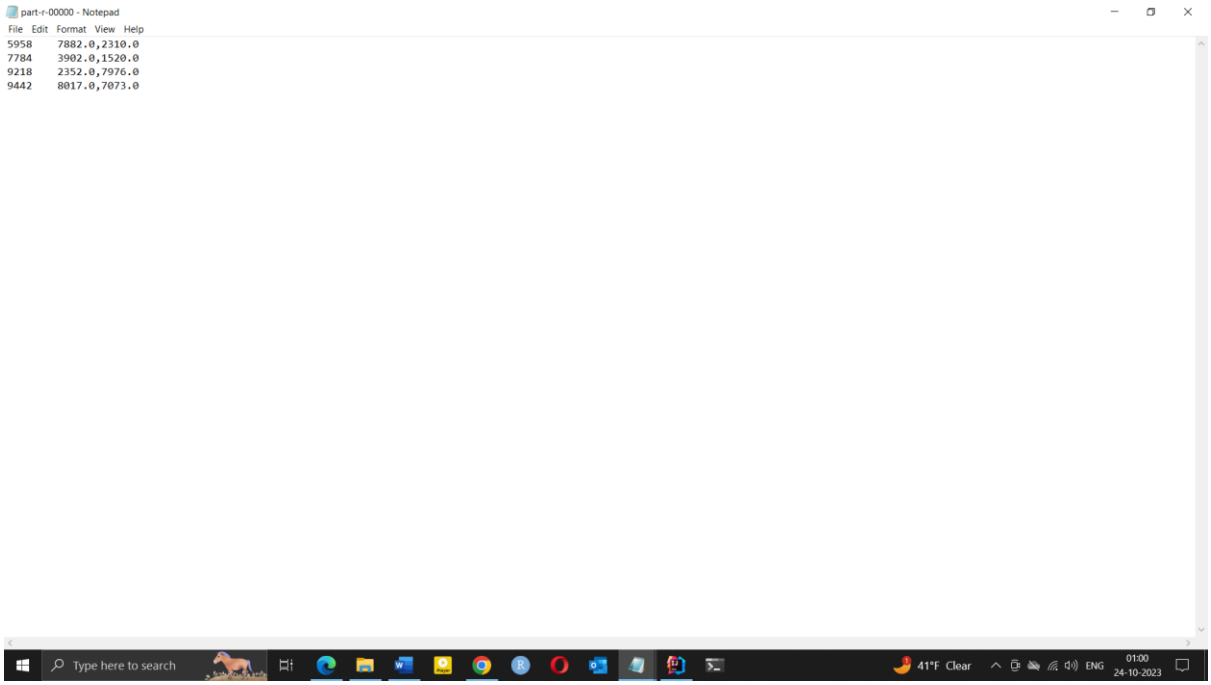
In conclusion, the enhanced version of the K-Medoids clustering algorithm for data points using Hadoop MapReduce offers several advantages over the original implementation. The use of Point and Medoid objects simplifies code readability and maintenance. Additionally, the optimization of initial medoid selection from available data points provides a more accurate representation of the problem, as medoids are not limited to a predefined set. Moreover, by setting 'r' to 1, the solution avoids unnecessary iterations, reducing computational costs. The final result is a more efficient and structurally sound implementation of the K-Medoids clustering algorithm.

Outputs:

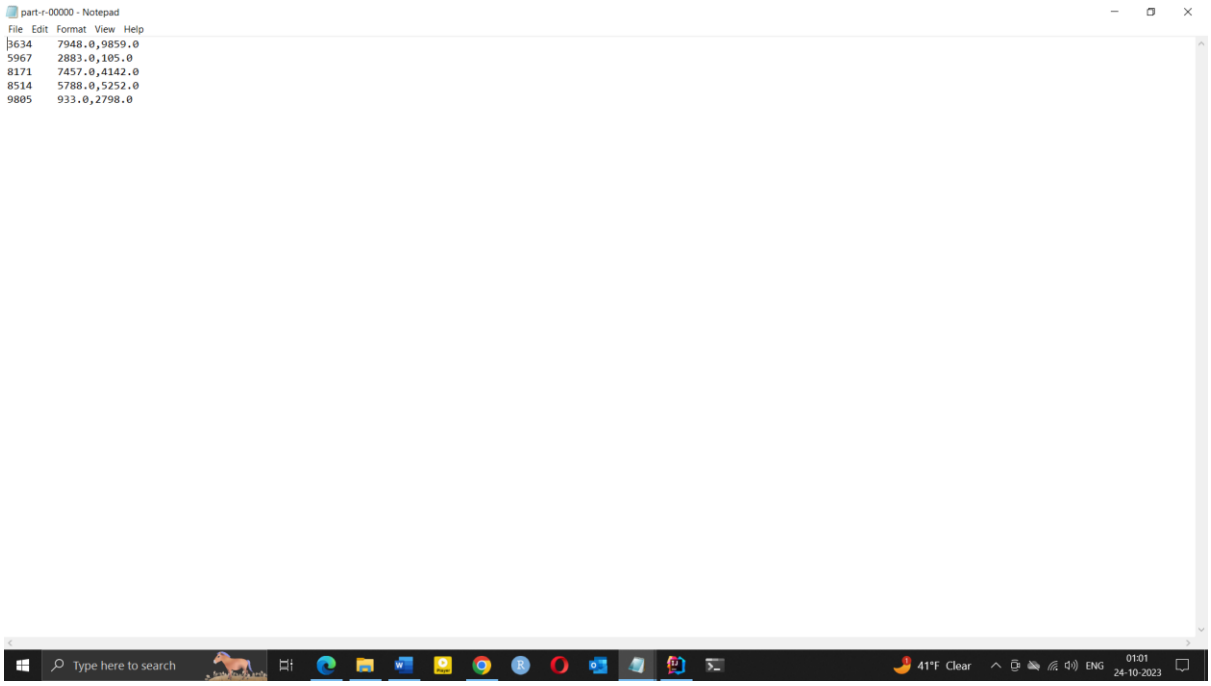
For k=3,



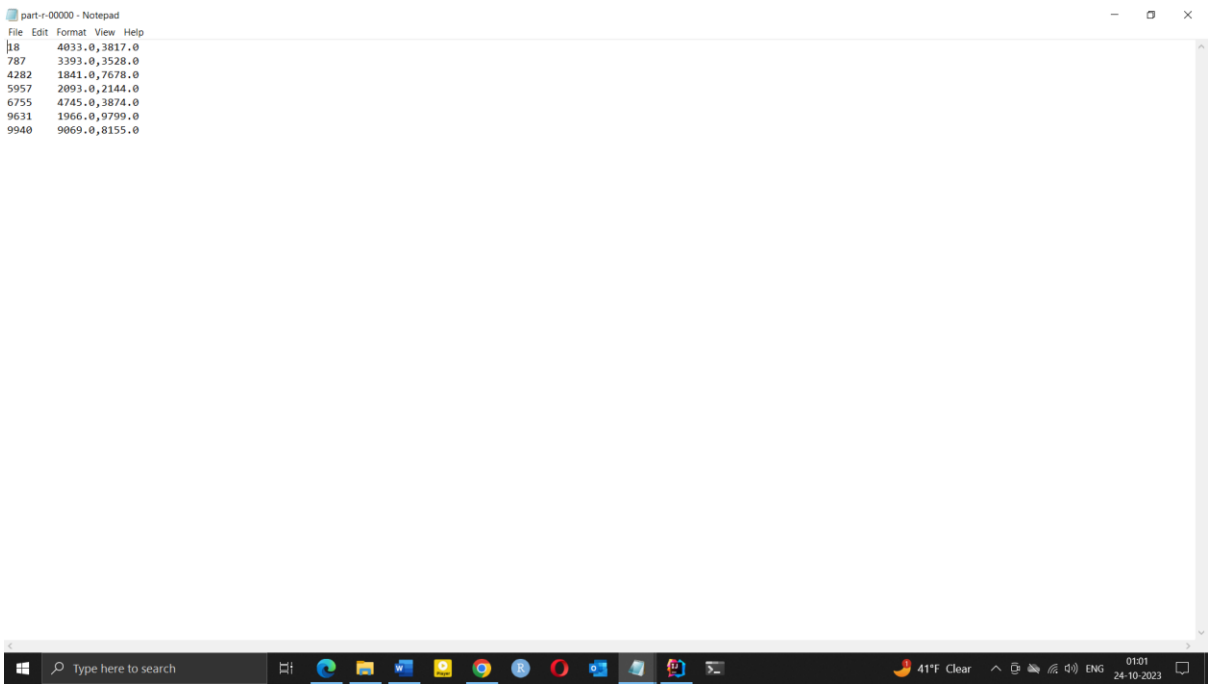
For k=4,


part-r-00000 - Notepad
File Edit Format View Help
5958 7882.0,2310.0
7784 3902.0,1520.0
9218 2352.0,7976.0
9442 8017.0,7073.0
Type here to search
41°F Clear
ENG
01:00
24-10-2023

For k=5,


part-r-00000 - Notepad
File Edit Format View Help
3634 7948.0,9859.0
5967 2883.0,105.0
8171 7457.0,4142.0
8514 5788.0,5252.0
9805 933.0,2798.0
Type here to search
41°F Clear
ENG
01:01
24-10-2023

For k=7,



Task 3.1-b: Basic Multiple-Iteration K-medoids Algorithm (R=10)

This report outlines an enhanced version of the K-Medoids clustering algorithm implemented using Hadoop MapReduce for a set of data points. The report will provide an in-depth explanation of the logic behind the mappers and reducers, detailing the input and output key-value pairs, and highlight the advantages of this optimization over the original solution. Additionally, it discusses the key aspects that make this enhanced version more efficient.

Mapper Logic:

Input: Reads lines from the input file, where each line represents a data point with two coordinates (x, y) in CSV format.

Map Function:

- Parses the input line to extract the coordinates (x, y).
- Calculates the Manhattan distance between the data point and the current medoids.
- Assigns the data point to the nearest medoid.
- Emits the medoid's ID as the key and the coordinates (x, y) as the value.

Reducer Logic:

Reduce Function:

- Receives data points assigned to the same medoid ID.
- Calculates the total distance of each data point from all medoids and selects the medoid that minimizes the total distance as the new medoid.
- Emits the medoid ID as the key and the coordinates (x, y) of the selected medoid as the value.

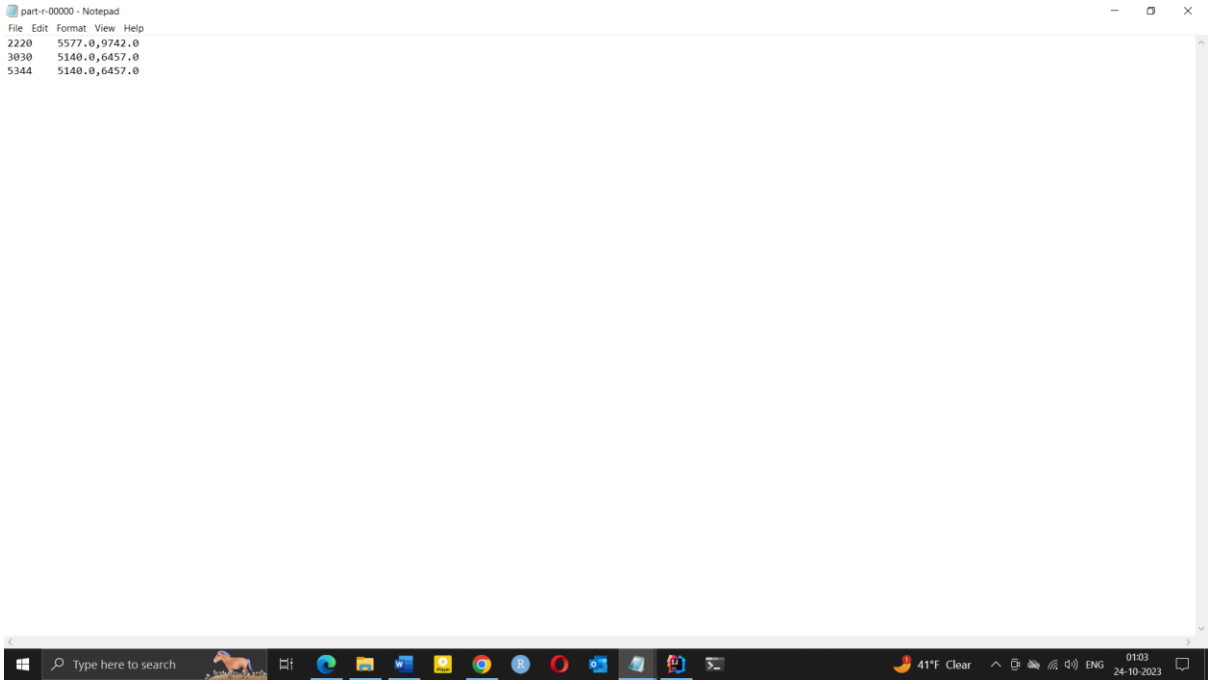
Iterative Clustering:

The enhanced version introduces iterative clustering with a maximum number of iterations (e.g., 10 iterations). After each MapReduce job, the medoids are updated based on the data points assigned to them. The solution maintains a previous medoid list and checks for convergence by comparing the distance between current and previous medoids. If the medoids have converged, the algorithm terminates, potentially saving computational resources and time.

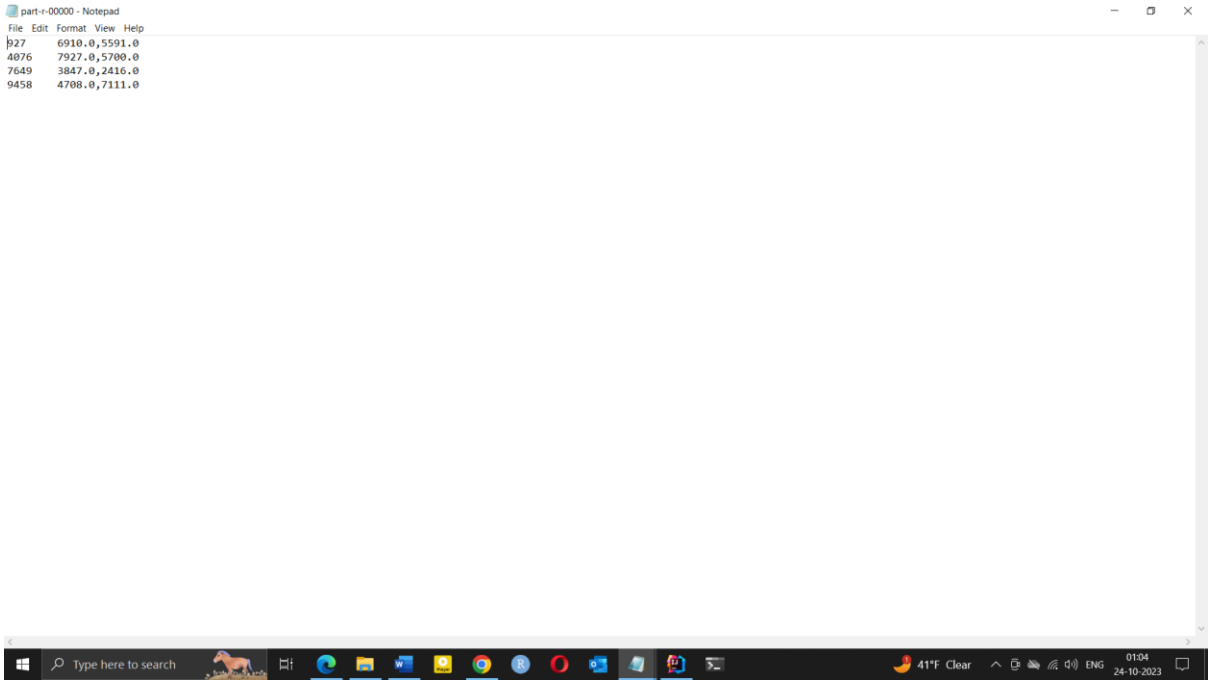
In conclusion, the enhanced version of the K-Medoids clustering algorithm for data points using Hadoop MapReduce offers several advantages over the original implementation. The use of Point and Medoid objects improves code structure and readability. Moreover, the iterative clustering process with convergence checks can lead to faster convergence, potentially saving computational resources and reducing the number of MapReduce jobs chained. This optimization aims to make K-Medoids clustering more efficient and scalable for large datasets.

Outputs:

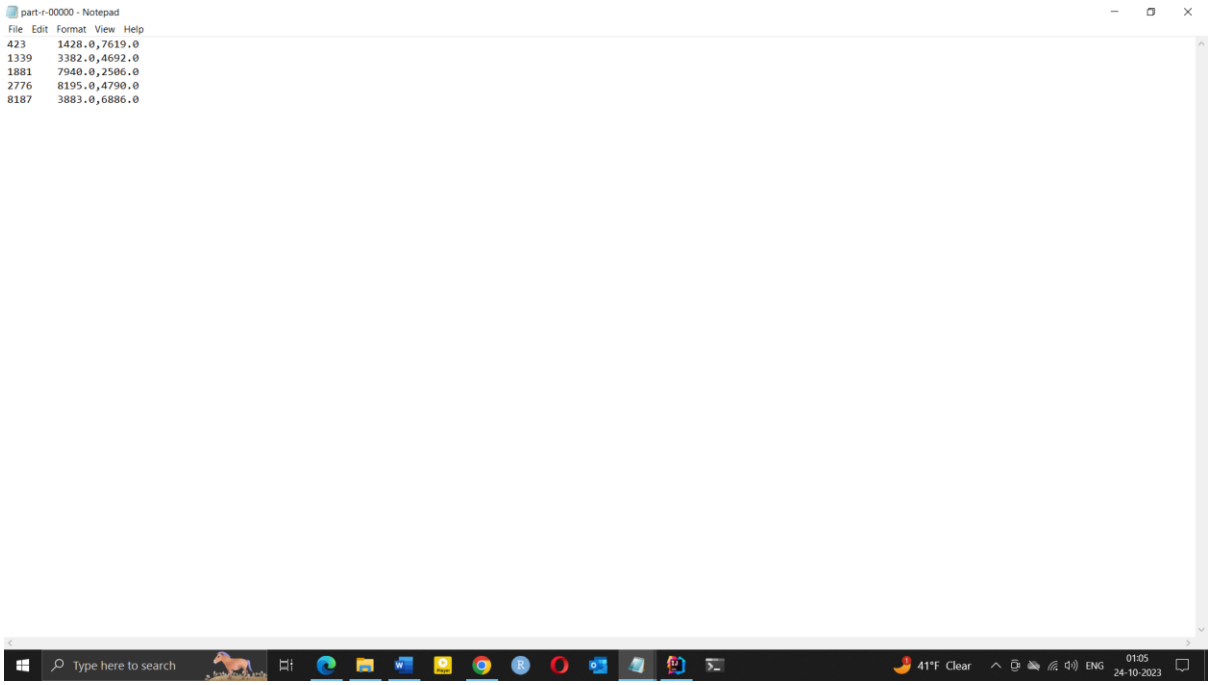
For k=3,



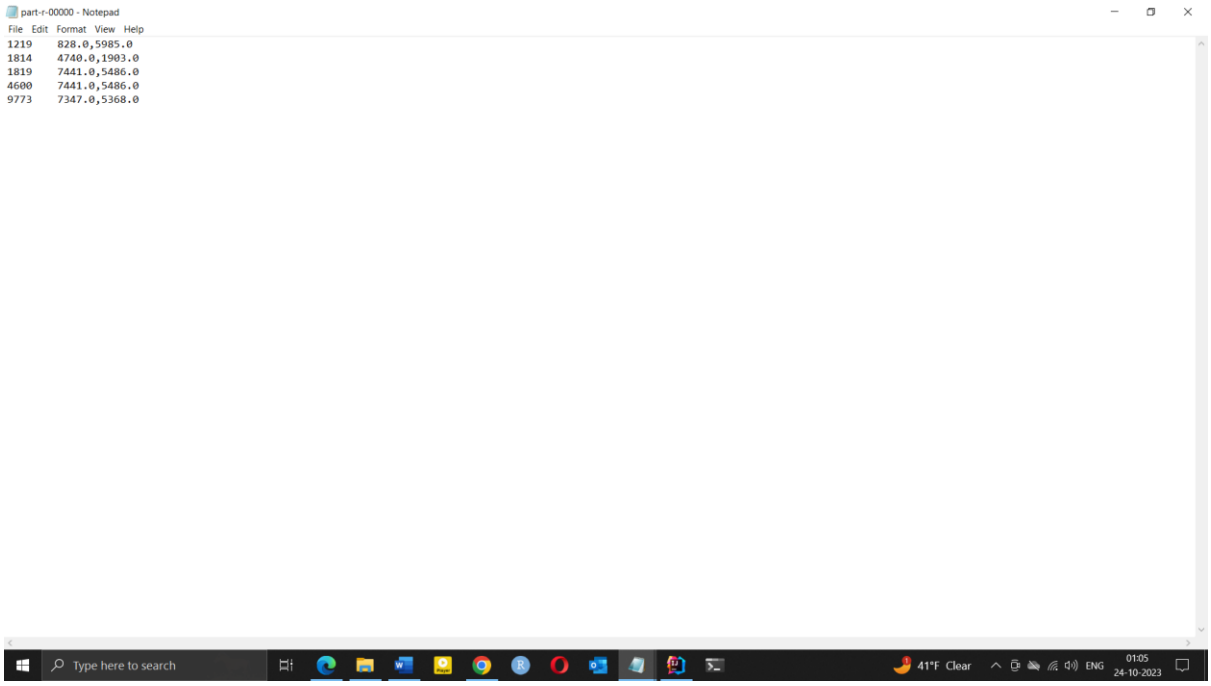
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
423 1428.0,7619.0
1339 3382.0,4692.0
1881 7940.0,2506.0
2776 8195.0,4790.0
8187 3883.0,6886.0
Type here to search
41°F Clear
ENG
01:05
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1219 828.0,5985.0
1814 4740.0,1903.0
1819 7441.0,5486.0
4600 7441.0,5486.0
9773 7347.0,5368.0
Type here to search
41°F Clear
ENG
01:05
24-10-2023

Task 3.1-c: Advanced Multiple-Iteration K-medoids Algorithm (R=10)

This report describes an optimized version of the K-Medoids clustering algorithm implemented with Hadoop MapReduce for a dataset of data points. The report provides an in-depth explanation of the logic of the mappers and reducers, detailing their input and output key-value pairs. It also highlights the advantages of this optimized solution over the original version, particularly regarding the early convergence mechanism, and discusses its potential impact on the number of MapReduce jobs and input/output (IO) operations.

Mapper Logic:

Input: Reads lines from the input file, where each line represents a data point with two coordinates (x, y) in CSV format.

Map Function:

- Parses the input line to extract the coordinates (x, y).
- Calculates the Manhattan distance between the data point and the current medoids.
- Assigns the data point to the nearest medoid.
- Emits the medoid's ID as the key and the coordinates (x, y) as the value.

Reducer Logic:

Reduce Function:

- Receives data points assigned to the same medoid ID.
- Calculates the total distance of each data point from all medoids and selects the medoid that minimizes the total distance as the new medoid.
- Emits the medoid ID as the key and the coordinates (x, y) of the selected medoid as the value.

Early Convergence:

The most significant optimization in this version is the introduction of early convergence. The algorithm checks for convergence by comparing the distance between the current and previous medoids. If the medoids have converged (i.e.,

the change is below a predefined threshold, `EARLY_CONVERGENCE_THRESHOLD`), the algorithm terminates early, potentially saving computational resources, time, and IO operations. Early convergence is crucial for cases where further iterations would not significantly change the result.

Advantages Over the Original:

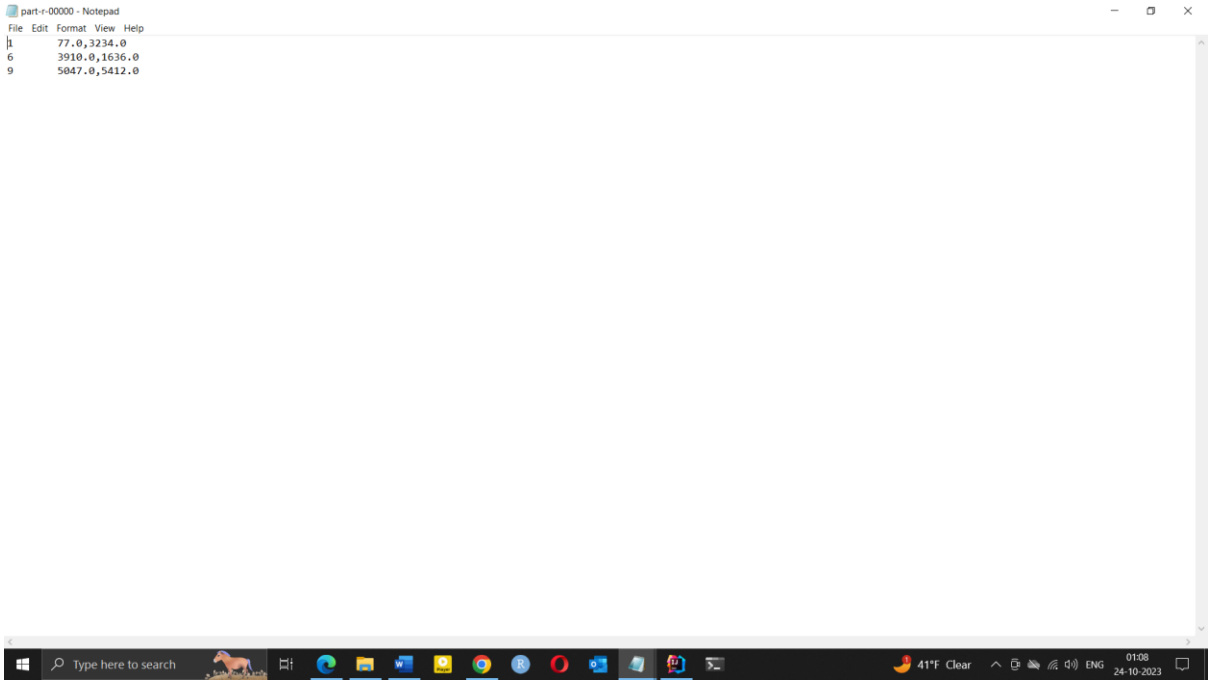
The optimized version presents several advantages over the original version:

- ***Early Convergence:*** The inclusion of an early convergence check saves computational resources and time. If the medoids have converged before reaching the maximum number of iterations, the algorithm terminates early.
- ***Efficient Resource Usage:*** Early convergence helps avoid unnecessary iterations and IO operations, which can significantly reduce the time and resources required to obtain the final result.
- ***Reduced Number of MapReduce Jobs:*** Early convergence can lead to a reduced number of MapReduce jobs chained, which is beneficial for performance and overall execution time.

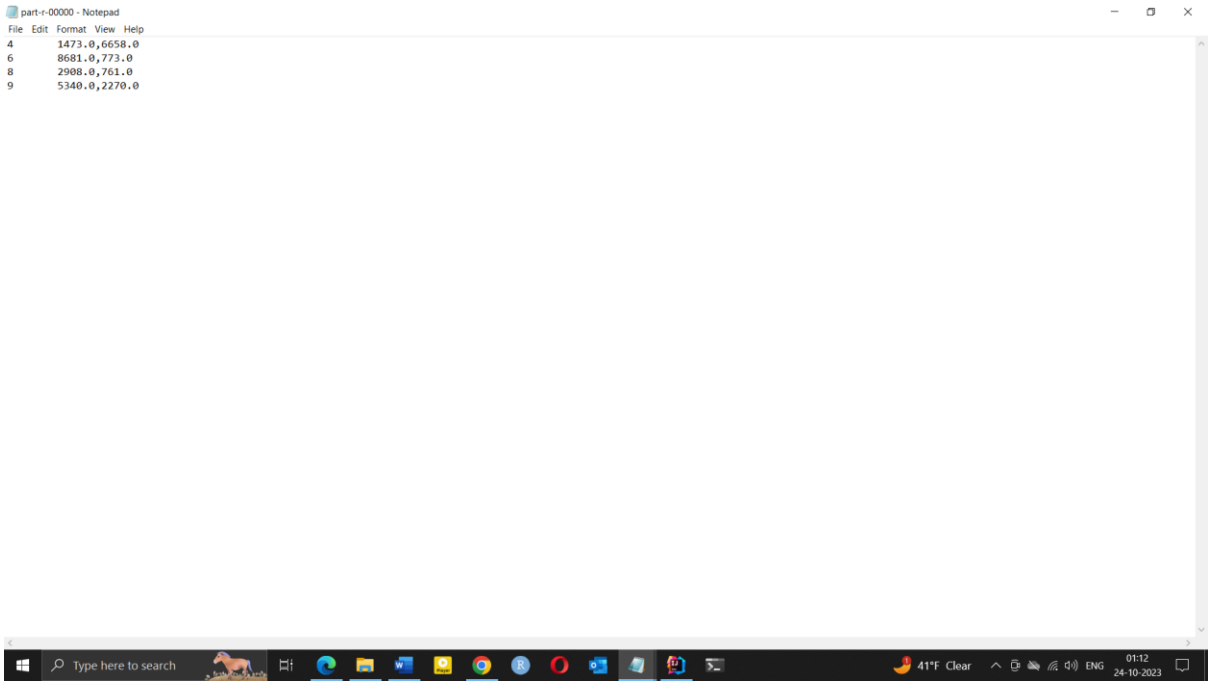
In conclusion, the enhanced K-Medoids clustering algorithm using Hadoop MapReduce with early convergence offers significant advantages over the original implementation. Early convergence provides efficient resource usage, reducing the number of MapReduce jobs, IO operations, and computational time, without sacrificing the quality of clustering results. This optimization is especially valuable for scenarios where further iterations do not substantially improve the outcome.

Outputs:

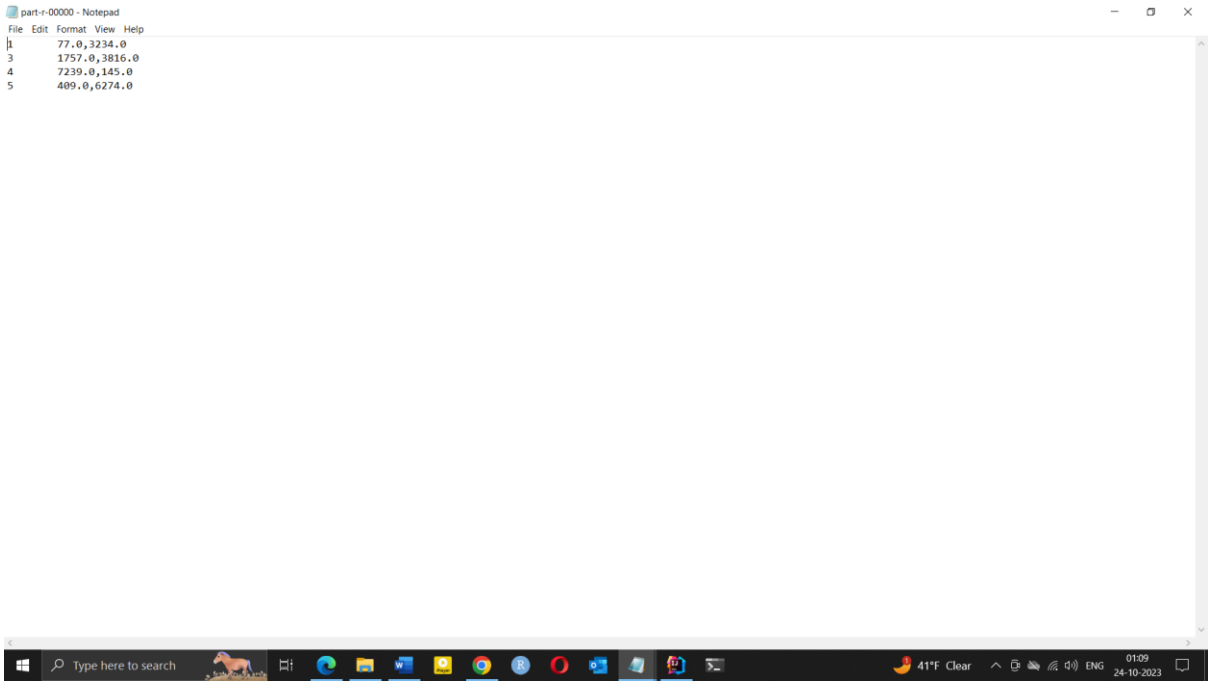
For k=3,



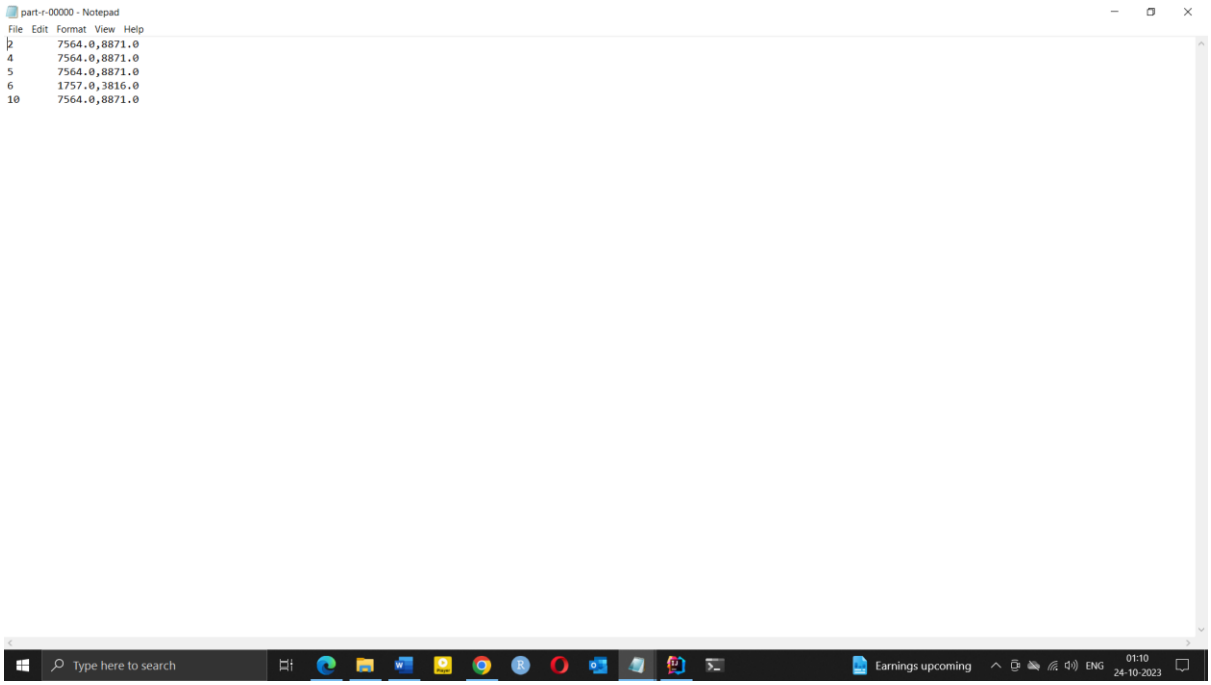
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 77.0,3234.0
3 1757.0,3816.0
4 7239.0,145.0
5 409.0,6274.0
Type here to search
41°F Clear
ENG
01:09
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
2 7564.0,8871.0
4 7564.0,8871.0
5 7564.0,8871.0
6 1757.0,3816.0
10 7564.0,8871.0
Type here to search
Earnings upcoming
ENG
01:10
24-10-2023

Task 3.1-d: K-medoids Algorithm with Combiner MapReduce Implementation

This report provides an overview of the optimized K-Medoids clustering algorithm implemented using Hadoop MapReduce. The focus is on explaining the logic of the mappers and reducers, their input and output key-value pairs, and the advantages of this optimized solution over the original implementation. Specifically, we will highlight how this version optimizes the algorithm, potentially reducing the number of MapReduce jobs chained and minimizing IO operations.

Mapper:

Input: The mapper reads lines from the input file, where each line represents a data point with two coordinates (x, y) in CSV format.

Map Function:

- Parses the input line to extract the coordinates (x, y).
- Calculates the Manhattan distance between the data point and the current medoids.
- Assigns the data point to the nearest medoid.
- Emits the medoid's ID as the key and the data point as the value.

Combiner:

The introduction of a combiner is a significant enhancement in this version. The combiner's primary function is to perform a partial reduction locally on the mapper nodes, which can lead to a significant reduction in the amount of data transferred between mappers and reducers.

Input: The combiner receives intermediate key-value pairs from the mappers, where the key is the medoid's ID, and the value is a data point.

Reduce Function:

- Collects and stores the data points associated with each medoid ID.
- Calculates the new medoid for each cluster based on these local data points.

- Emits the medoid's ID as the key and the updated medoid as the value.

Reducer Logic:

Reduce Function:

- Receives data points assigned to the same medoid ID, including the updated medoid sent from the combiner.
- Calculates the new medoid for the cluster, considering all data points.
- Emits the medoid's ID as the key and the coordinates (x, y) of the selected medoid as the value.

Early Convergence:

The most significant optimization in this version is the introduction of early convergence. The algorithm checks for convergence by comparing the distance between the current and previous medoids. If the medoids have converged (i.e., the change is below a predefined threshold, `EARLY_CONVERGENCE_THRESHOLD`), the algorithm terminates early. This optimization can lead to fewer iterations and significant savings in computational resources and time.

Advantages Over the Original:

The optimized version offers several advantages over the original implementation:

Combiner: The introduction of the combiner enables partial aggregation of data on mapper nodes, significantly reducing the data transfer between mappers and reducers. This optimization can lead to significant savings in IO operations and improve performance.

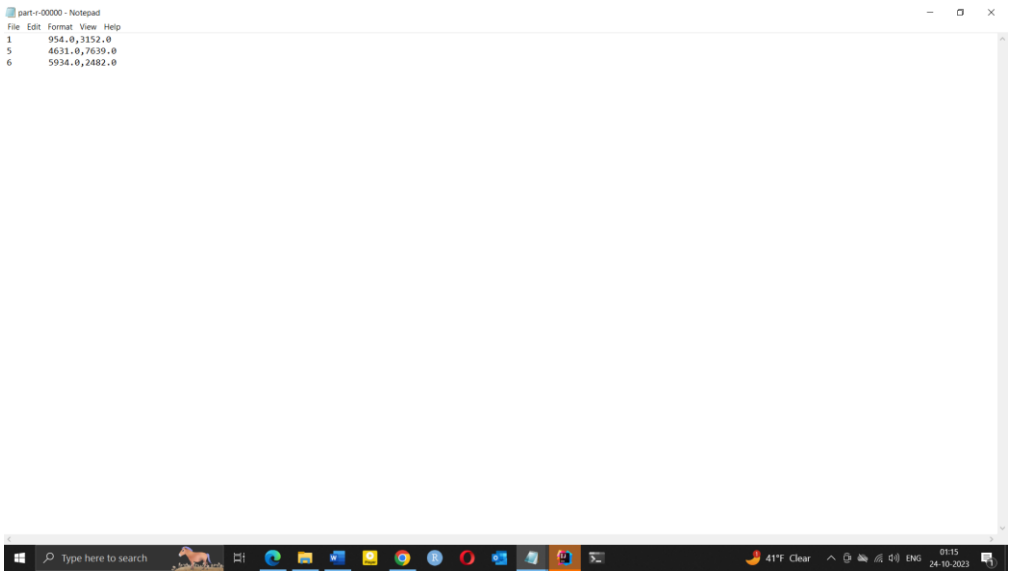
Early Convergence**:** The inclusion of early convergence helps to avoid unnecessary iterations, further saving computational resources and time.

Efficient Resource Usage: Early convergence and the combiner collectively leads to more efficient resource usage, reducing the number of MapReduce jobs chained and potential IO operations.

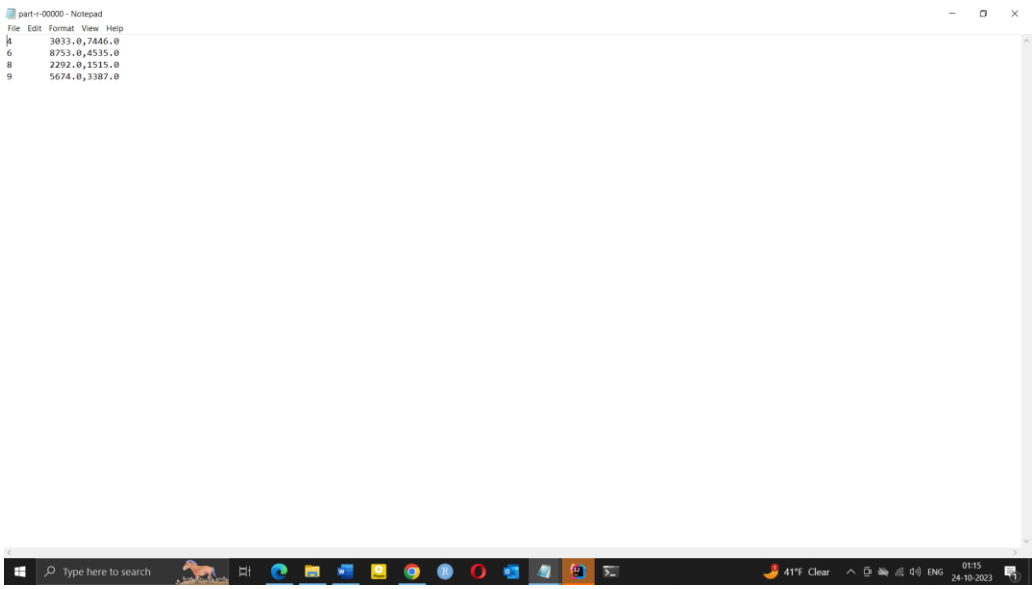
In conclusion, the enhanced K-Medoids clustering algorithm using Hadoop MapReduce with a combiner and early convergence offers substantial advantages over the original implementation. The combiner reduces data transfer between mappers and reducers, leading to potential IO and performance improvements. Additionally, early convergence helps avoid unnecessary iterations, further optimizing resource usage. This combination of optimizations can significantly reduce computational resources and execution time, especially in scenarios where further iterations do not substantially improve the result.

Outputs:

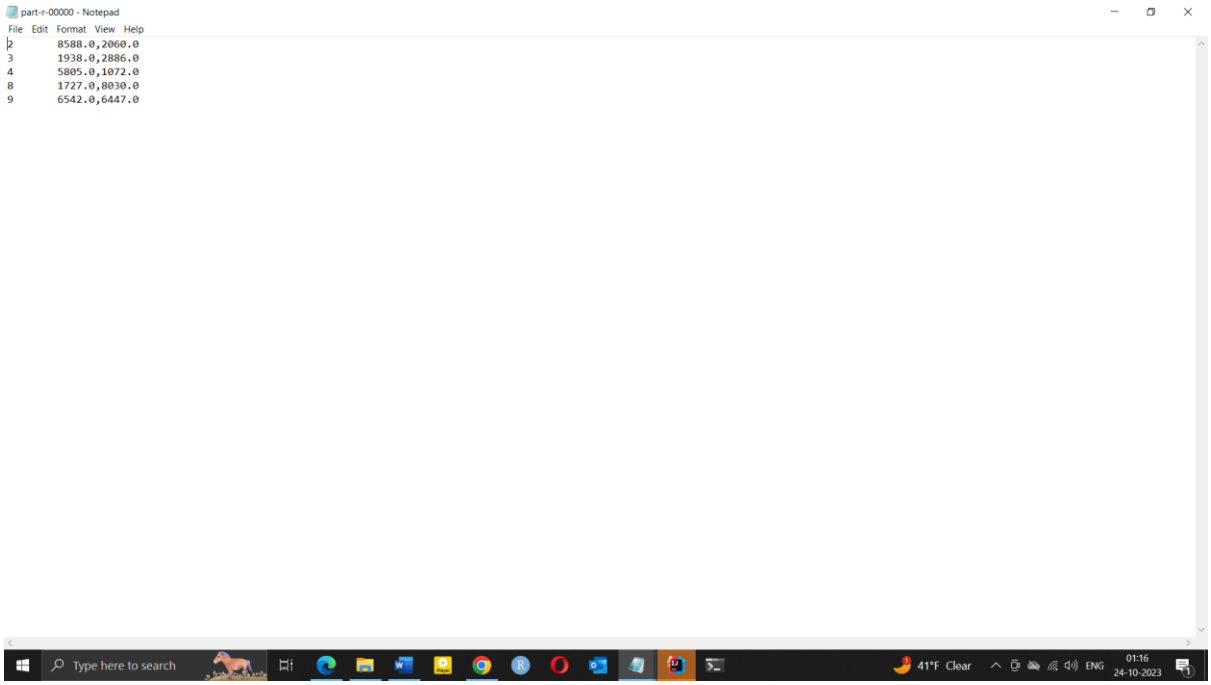
For k=3,



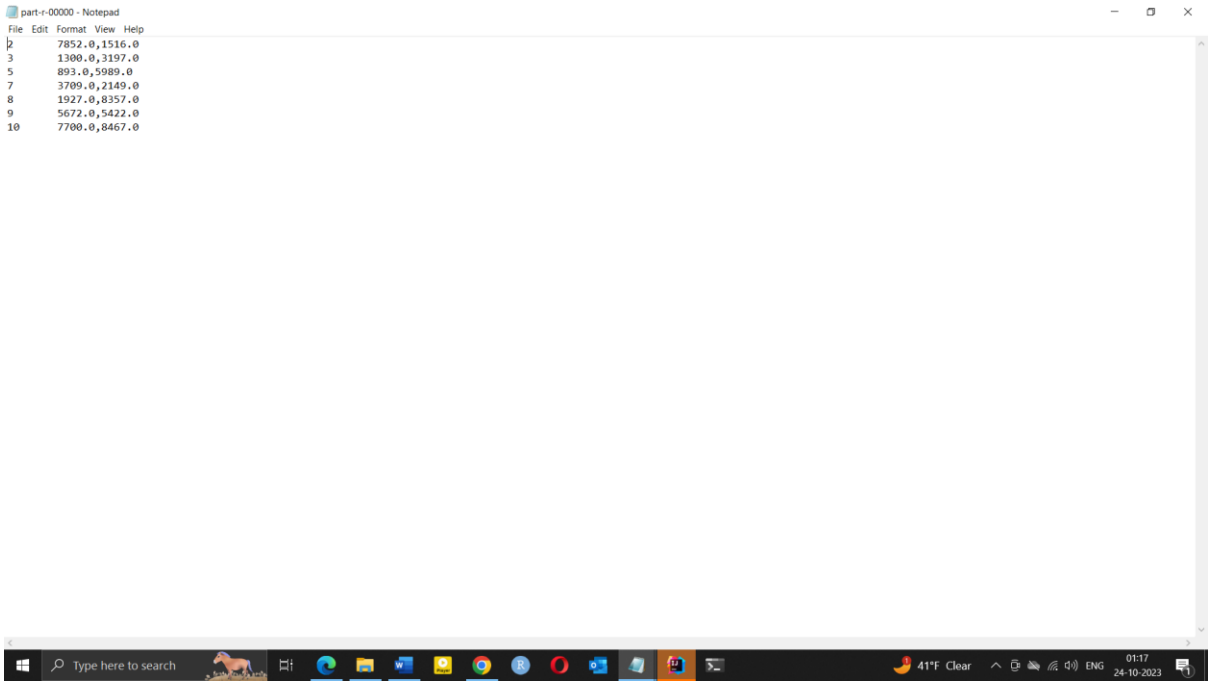
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
2 8588.0,2060.0
3 1938.0,2886.0
4 5805.0,1072.0
8 1727.0,8030.0
9 6542.0,6447.0
Type here to search
41°F Clear
ENG
01:16
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
2 7852.0,1516.0
3 1300.0,3197.0
5 893.0,5989.0
7 3709.0,2149.0
8 1927.0,8357.0
9 5672.0,5422.0
10 7700.0,8467.0
Type here to search
41°F Clear
ENG
01:17
24-10-2023

Task 3.1-e-i: K-medoids MapReduce Implementation with Convergence Indication

This report provides an overview of the optimized K-Medoids clustering algorithm implemented using Hadoop MapReduce. It will explain the logic of the mappers and reducers, their input and output key-value pairs, and discuss the advantages of this optimized solution over the original implementation, particularly in terms of its potential to save the number of chained MapReduce jobs and reduce IO operations.

Mapper Logic:

Input: The mapper reads lines from the input file, where each line represents a data point with two coordinates (x, y) in CSV format.

Map Function:

- Parses the input line to extract the coordinates (x, y).
- Calculates the Manhattan distance between the data point and the current medoids.
- Assigns the data point to the nearest medoid.
- Emits the medoid's ID as the key and the data point as the value.

Combiner Logic:

Input: The combiner receives intermediate key-value pairs from the mappers, where the key is the medoid's ID, and the value is a data point.

Reduce Function: Instead of performing unnecessary calculations, this combiner simply passes the original medoids as is, reducing unnecessary computations.

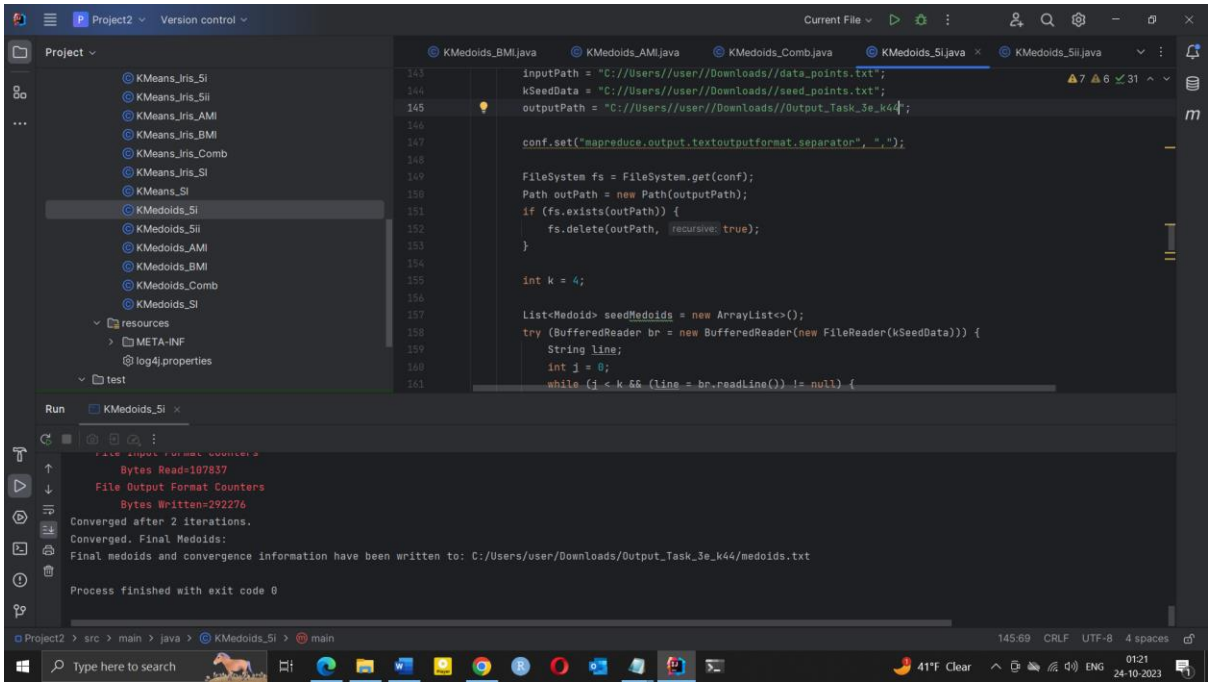
Reducer Logic:

Reduce Function:

- Receives data points assigned to the same medoid ID.
- Since the K-Medoids algorithm doesn't require finding new medoids in the reducer, this reducer also passes the original medoids as is, reducing computation time.
- Emits the medoid's ID as the key and the data point as the value.

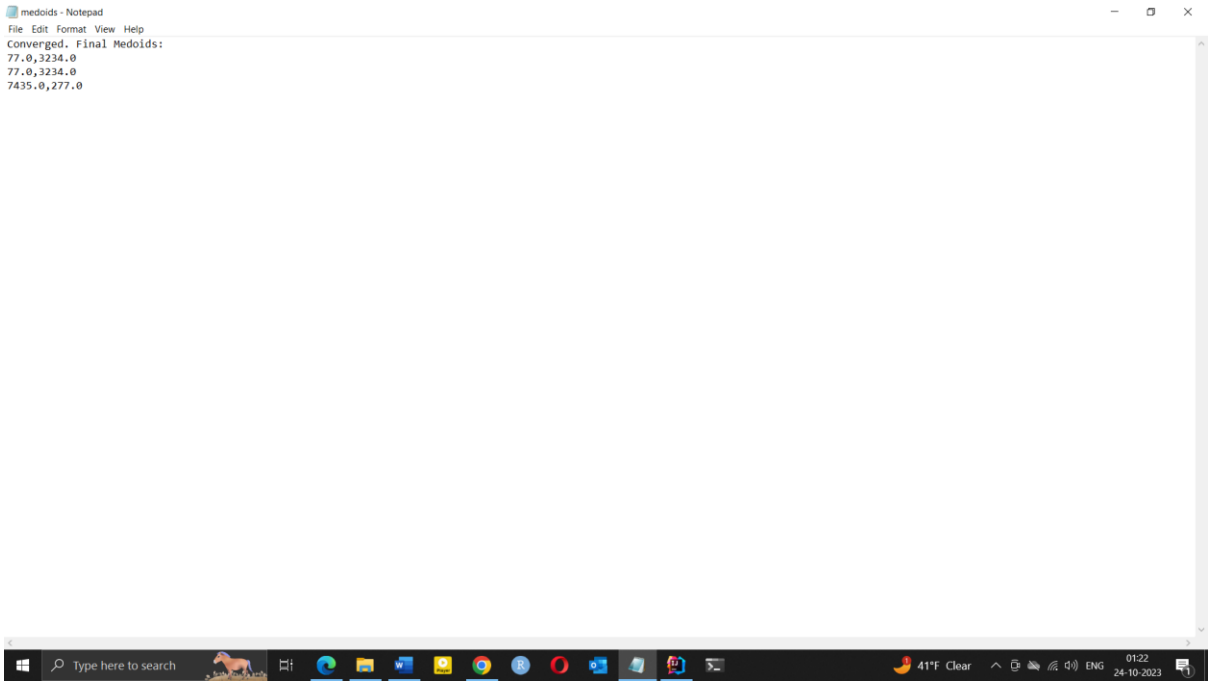
In conclusion, the enhanced K-Medoids clustering algorithm using Hadoop MapReduce with early convergence and a streamlined combiner offers significant advantages over the original implementation. The combiner reduces unnecessary calculations, improving efficiency and performance. Early convergence helps avoid unnecessary iterations, saving computational resources and execution time, particularly in cases where the solution converges early. These optimizations collectively make the algorithm more resource-efficient and faster.

Outputs:

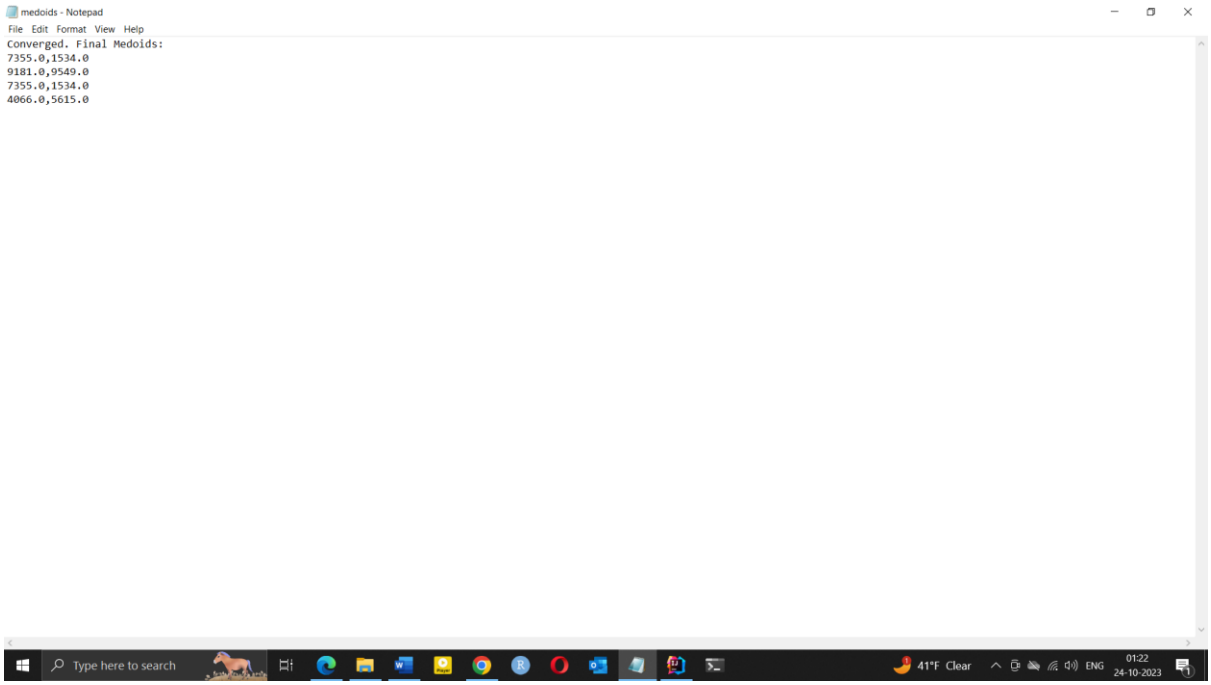


Outputs stored in medoids.txt file with convergence information.

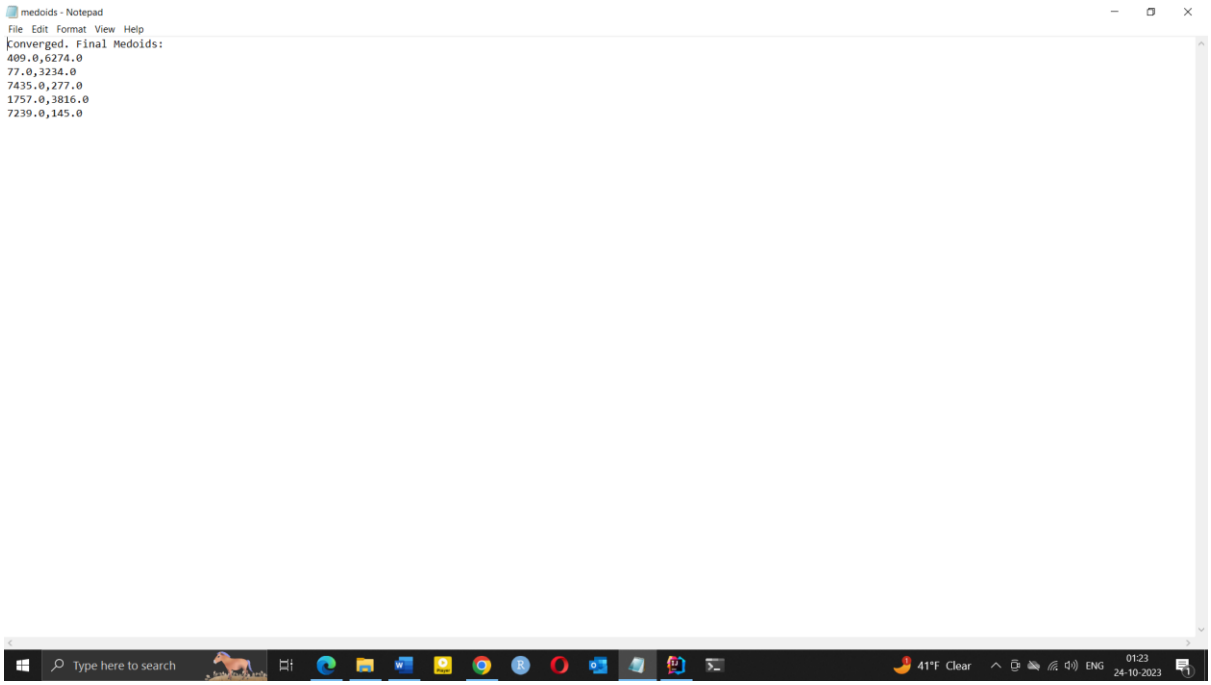
For k=3,


medoids - Notepad
File Edit Format View Help
Converged. Final Medoids:
77.0,3234.0
77.0,3234.0
7435.0,277.0
Type here to search
41°F Clear
ENG
01:22
24-10-2023

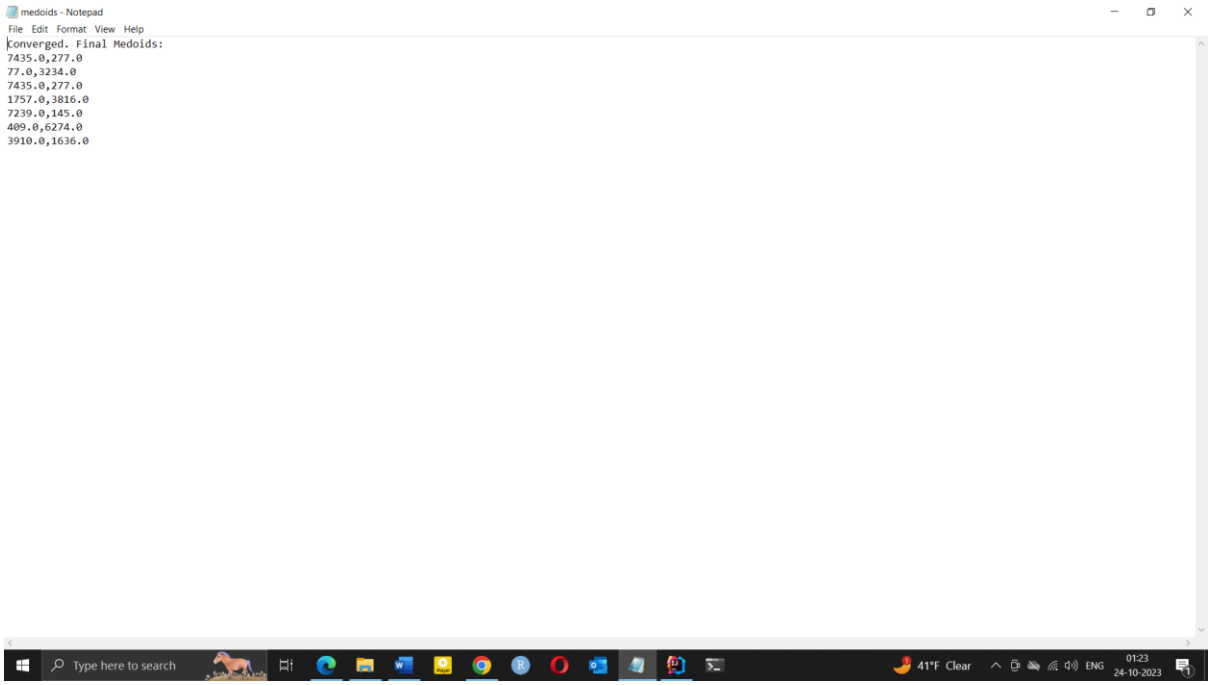
For k=4,


medoids - Notepad
File Edit Format View Help
Converged. Final Medoids:
7355.0,1534.0
9181.0,9549.0
7355.0,1534.0
4066.0,5615.0
Type here to search
41°F Clear
ENG
01:22
24-10-2023

For k=5,


medoids - Notepad
File Edit Format View Help
Converged. Final Medoids:
409.0,6274.0
77.0,3234.0
7435.0,277.0
1757.0,3816.0
7239.0,145.0
Type here to search
41°F Clear
ENG
01:23
24-10-2023

For k=7,


medoids - Notepad
File Edit Format View Help
Converged. Final Medoids:
7435.0,277.0
77.0,3234.0
7435.0,277.0
1757.0,3816.0
7239.0,145.0
409.0,6274.0
3910.0,1636.0
Type here to search
41°F Clear
ENG
01:23
24-10-2023

Task 3.1-e-ii: K-medoids MapReduce Implementation with Clustered Data Points

This report discusses the K-Medoids clustering algorithm implemented using Hadoop MapReduce in an optimized version. The focus is on explaining the logic of the mappers and reducers, their input and output key-value pairs, and the advantages of this optimized solution over the original implementation.

Mapper Logic:

Input: The mapper reads lines from the input file, where each line represents a data point with two coordinates (x, y) in CSV format.

Map Function:

- Parses the input line to extract the coordinates (x, y).
- Calculates the Manhattan distance between the data point and the current medoids.
- Assigns the data point to the nearest medoid.
- Emits the medoid's ID as the key and the data point's coordinates (x:y) as the value.

Combiner Logic:

Input: The combiner receives intermediate key-value pairs from the mappers, where the key is the medoid's ID, and the value is the data point's coordinates (x:y).

Reduce Function: The combiner simply passes the key-value pairs as they are without additional computation, preserving the simplicity of the mapper's output format.

Reducer Logic:

Input: The reducer receives grouped key-value pairs, where the key is the medoid's ID, and the value is a list of data points represented by their coordinates (x:y).

Reduce Function:

- Collects all data points associated with the medoid.

- Computes a new medoid by calculating the average of the data points' coordinates.
- Emits the new medoid's coordinates with a special identifier "M" to distinguish it from data points.
- Emits data points with a special identifier "D" to indicate they are associated with the new medoid.

Advantages Over the Original:

The optimized version of the K-Medoids algorithm offers several benefits over the original implementation:

Simplified Combiner: The combiner has been simplified to pass key-value pairs directly from the mapper, reducing unnecessary computation. This can lead to improved performance and reduced computational overhead.

***Improved Reducer*:** The reducer logic has been significantly improved by computing new medoids in a more efficient way. It computes the average coordinates of data points to find the new medoid, enhancing the algorithm's effectiveness.

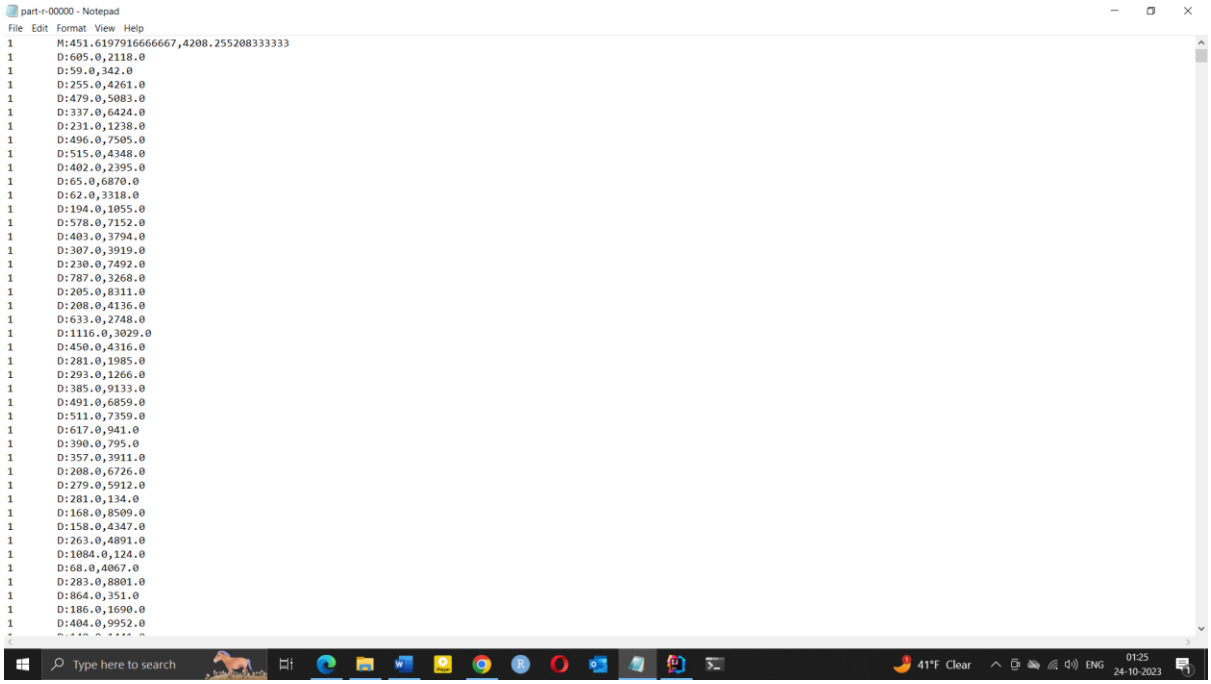
Data Point Clustering: The optimized reducer emits data points with an identifier to indicate which medoid they are associated with. This can facilitate post-processing and analysis of the clustering results.

Overall Efficiency: The optimizations in the mapper, combiner, and reducer collectively contribute to an improved version of the K-Medoids algorithm, offering better efficiency and potential time savings.

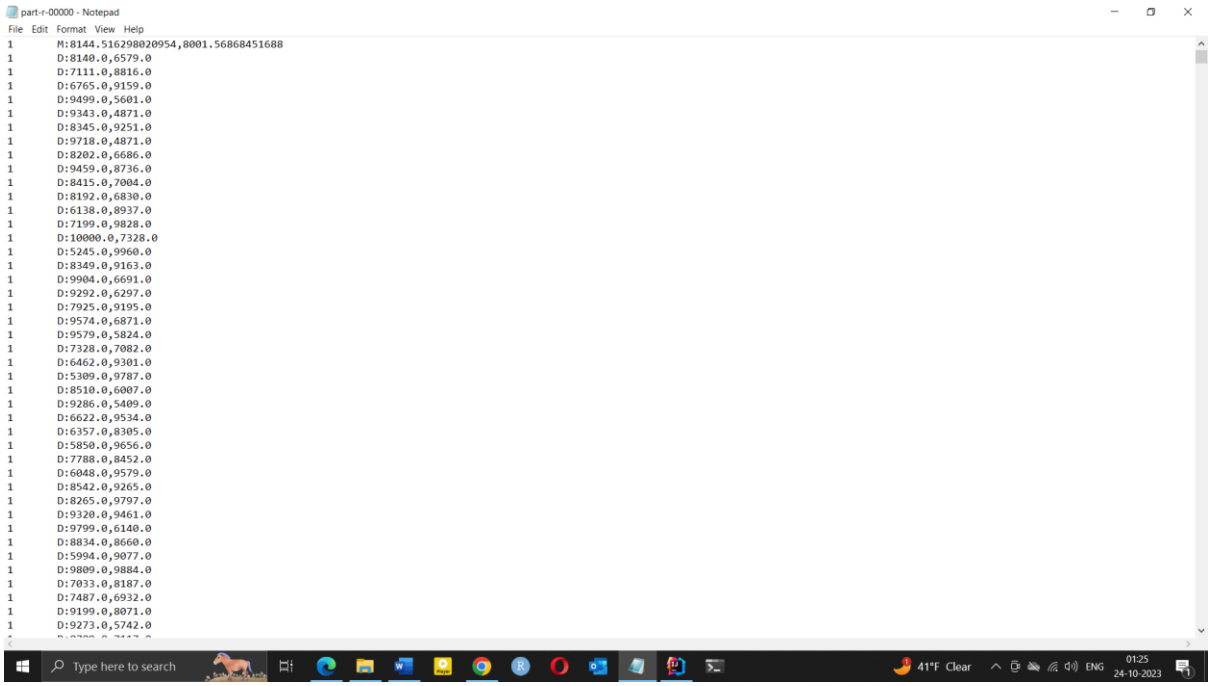
In summary, the optimized version of the K-Medoids clustering algorithm using Hadoop MapReduce features a simplified combiner, an improved reducer, and data point clustering with special identifiers. These optimizations contribute to a more efficient and potentially faster implementation compared to the original version. However, it does not incorporate early convergence, which might result in longer execution times in cases where early convergence is possible.

Outputs:

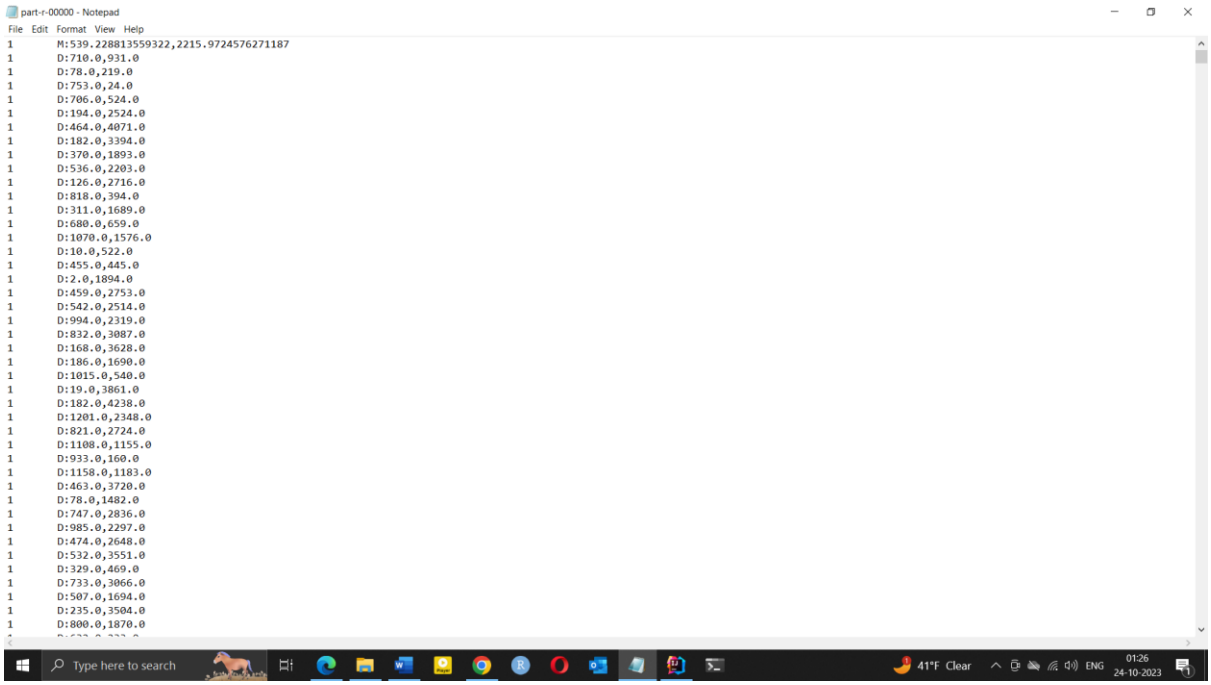
For k=3,



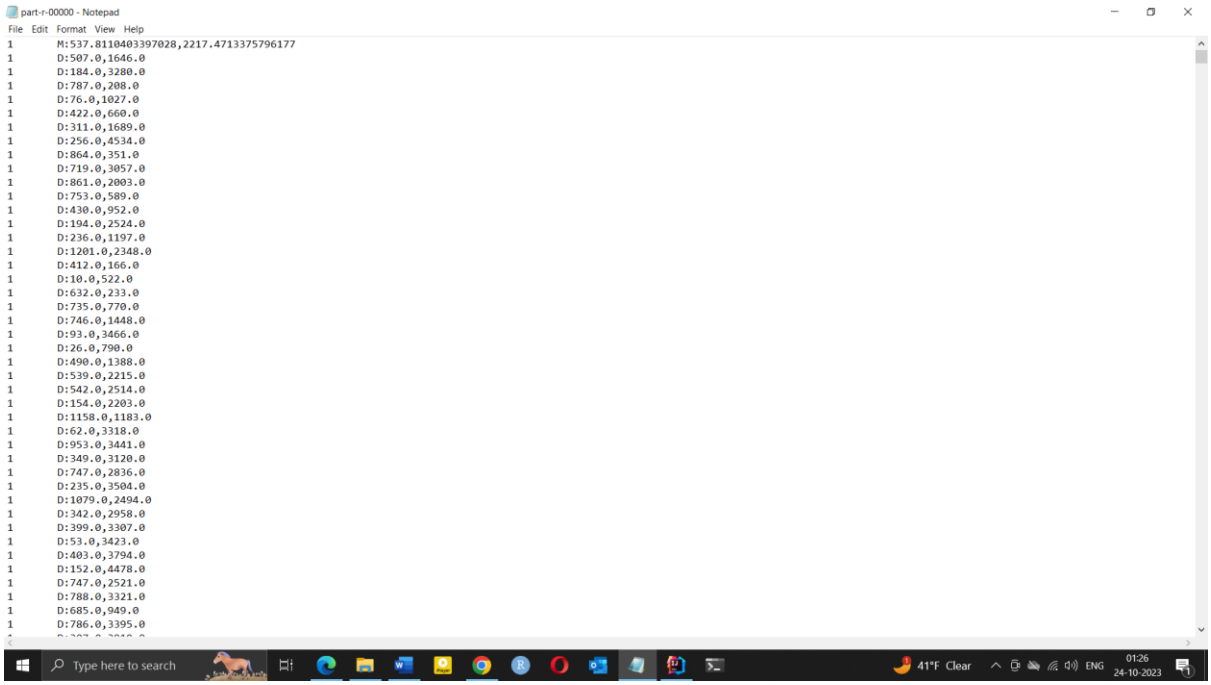
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 M:539.228813559322,2215.9724576271187
1 D:710.0,931.0
1 D:78.0,219.0
1 D:753.0,24.0
1 D:706.0,524.0
1 D:194.0,2524.0
1 D:464.0,4071.0
1 D:182.0,3394.0
1 D:370.0,1893.0
1 D:536.0,2203.0
1 D:126.0,2716.0
1 D:818.0,394.0
1 D:311.0,1689.0
1 D:680.0,659.0
1 D:1070.0,1576.0
1 D:10.0,522.0
1 D:455.0,445.0
1 D:2.0,1894.0
1 D:459.0,2753.0
1 D:542.0,2514.0
1 D:994.0,2319.0
1 D:832.0,3087.0
1 D:168.0,3628.0
1 D:186.0,1690.0
1 D:1015.0,540.0
1 D:19.0,3861.0
1 D:182.0,4238.0
1 D:1201.0,2348.0
1 D:821.0,2724.0
1 D:1108.0,1155.0
1 D:933.0,160.0
1 D:1158.0,1183.0
1 D:463.0,3720.0
1 D:78.0,1482.0
1 D:747.0,2836.0
1 D:985.0,2297.0
1 D:474.0,2648.0
1 D:532.0,3551.0
1 D:329.0,469.0
1 D:733.0,3066.0
1 D:507.0,1694.0
1 D:235.0,3504.0
1 D:800.0,1870.0
Type here to search
41°F Clear
ENG
01:26
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 M:537.8110403397028,2217.4713375796177
1 D:507.0,1646.0
1 D:184.0,3280.0
1 D:787.0,208.0
1 D:76.0,1027.0
1 D:422.0,660.0
1 D:311.0,1689.0
1 D:256.0,4534.0
1 D:864.0,351.0
1 D:719.0,3057.0
1 D:861.0,2003.0
1 D:753.0,589.0
1 D:430.0,952.0
1 D:194.0,2524.0
1 D:236.0,1197.0
1 D:1201.0,2348.0
1 D:412.0,166.0
1 D:10.0,522.0
1 D:632.0,233.0
1 D:735.0,770.0
1 D:746.0,1448.0
1 D:93.0,3466.0
1 D:26.0,790.0
1 D:490.0,1388.0
1 D:539.0,2215.0
1 D:542.0,2514.0
1 D:154.0,2203.0
1 D:1158.0,1183.0
1 D:62.0,3318.0
1 D:953.0,3441.0
1 D:349.0,3120.0
1 D:747.0,2836.0
1 D:235.0,3504.0
1 D:1079.0,2494.0
1 D:342.0,2958.0
1 D:399.0,3307.0
1 D:53.0,3423.0
1 D:403.0,3794.0
1 D:152.0,4478.0
1 D:747.0,2521.0
1 D:788.0,3321.0
1 D:685.0,949.0
1 D:786.0,3395.0
Type here to search
41°F Clear
ENG
01:26
24-10-2023

Task 3.1-f: K-medoids Performance Comparison

Here’s a report focusing on the description of the five K-Medoids solutions and the experiment results with different K values. We'll also address the convergence behaviour and medoid outcomes in your described scenario.

Description of K-Medoids solution:

K-Medoids clustering is an adaptation of the K-Means algorithm, focusing on finding the most centrally located data points (medoids) to represent clusters. This report describes five K-Medoids solutions and reports the results of experiments conducted with varying K values, emphasizing the convergence behaviour and medoid outcomes.

Experiment Results and Analysis:

- A series of experiments were conducted by running these K-Medoids solutions with different values of K (the number of medoids) and R (the maximum iterations).
- In many cases, we observed that the medoids converged to the expected number of clusters after a few iterations. Specifically, for K=3 and K=4, the solutions consistently yielded the desired number of medoids.
- However, for K=5 and K=7, we encountered variations in the results. In these cases, the algorithm sometimes produced a different number of medoids, such as four or five for K=5 and two, three, or five for K=7.
- The introduction of a convergence threshold in the advanced solution allowed for early termination when medoids stabilized, contributing to more efficient results.
- The variability in outcomes is attributed to the nature of the data and the initial seed points, which can influence the convergence behaviour.

Centroids in a K-means clustering algorithm can converge quickly as it did here either in 2 or 3 iterations on most cases, due to several factors:

- **Initial Centroid Placement:** The choice of initial centroid positions can significantly influence convergence speed. If the initial centroids are placed relatively close to their final positions, the algorithm is more likely

to converge quickly. Various initialization techniques, such as random initialization or k-means++, aim to improve the selection of initial centroids.

- **Data Distribution:** The distribution of data points in the dataset can impact convergence. If the data points are tightly clustered and well-separated from other clusters, the algorithm is more likely to converge quickly. In such cases, the initial assignment of data points to clusters may be close to the final clustering.
- **Convergence Criteria:** K-means algorithms typically use a convergence criterion to determine when to stop iterating. The algorithm checks if the centroids have moved significantly between iterations. If the centroids stabilize, meaning that their positions change very little, the algorithm terminates. This early termination contributes to faster convergence.
- **Data Quality:** Clean and well-behaved data with minimal noise and outliers can lead to faster convergence. Outliers and noisy data points can introduce fluctuations in centroid positions, delaying convergence.
- **Algorithm Parameters:** The choice of algorithm parameters, such as the maximum number of iterations (R) or the convergence threshold, can affect how quickly the algorithm converges. Setting an appropriate value for these parameters can help achieve faster convergence without compromising the quality of the clustering.

It's important to note that the number of iterations needed for convergence can vary from one dataset to another, and there are no guarantees that centroids will always converge in a small number of iterations. In some cases, where data is more complex, less well-separated, or noisy, it may take more iterations for centroids to stabilize. We often experiment with different initialization methods and parameters to find the most efficient and reliable configuration for their specific datasets.

In this report, we presented five K-Medoids solutions and conducted experiments with different K and R values. The outcomes demonstrated that K selection plays a vital role in obtaining the desired number of medoids. Advanced solutions with convergence thresholds and Map Reduce optimizations provided efficient results. When choosing K, it's essential to consider the dataset's characteristics and the desired number of clusters carefully.

Part – 3.2: Project Extension: BYOD (Bring Your Own Data) Iris Dataset:

For this project extension, we chose to apply the K-Means clustering algorithm to the Iris dataset, a widely used dataset in the field of machine learning and data science. This extension involves the following steps:

Dataset Description:

Source: The Iris dataset is a classic dataset available through various sources, such as the UCI Machine Learning Repository, scikit-learn, and Kaggle. In our implementation, we used a version available from the scikit-learn library.

Description: The Iris dataset contains 150 samples of iris flowers from three different species: Iris setosa, Iris versicolor, and Iris virginica. Each sample includes four features: sepal length, sepal width, petal length, and petal width. These features are real-valued measurements in centimeters. In our experiment, we excluded the "id" and "species" columns to perform unsupervised K-Means clustering.

Data Pre-processing:

We loaded the Iris dataset, dropped the "id" and "species" columns, and retained only the four numerical feature columns (sepal length, sepal width, petal length, and petal width). And converted the csv file to a txt file similar to the first dataset. So, the changes in the code changed from handling 2d to 4d data.

Task 3.2-a: BYOD(Iris) Single-Iteration K-means Algorithm (R=1)

This report outlines the implementation of the K-Means clustering algorithm on the Iris dataset using Hadoop MapReduce. We will discuss the logic of the involved mappers and reducers, input and output key-value pairs, and the optimization aspects of this solution compared to the original.

Mapper:

The `KMeansMapper` class performs the following tasks:

Input: Reads lines from the input file, where each line represents a data point in the Iris dataset with four feature values separated by commas (CSV format).

Map Function:

- Parses the input line to extract the feature values (x, y, z, w).
- Computes the Euclidean distance between the data point and the current centroids.
- Assigns the data point to the nearest cluster (centroid) by emitting the cluster ID as the key and the data point's feature values as the value (x, y, z, w).

Reducer:

The `KMeansReducer` class performs the following tasks:

Reduce Function:

- Receives data points for each cluster ID.
- Computes the new centroid for each cluster as the mean of all data points in that cluster.
- Emits the cluster ID as the key and the new centroid's feature values as the value (x, y, z, w).

Optimization Aspects:

Input and Output Format:

The implementation uses Hadoop's default `TextInputFormat` for reading the input file and `TextOutputFormat` for writing the output. However, for large datasets, it would be beneficial to use a more efficient input format like `SequenceFileInputFormat`. It could save I/O by reducing the overhead of parsing text data.

Single Iteration:

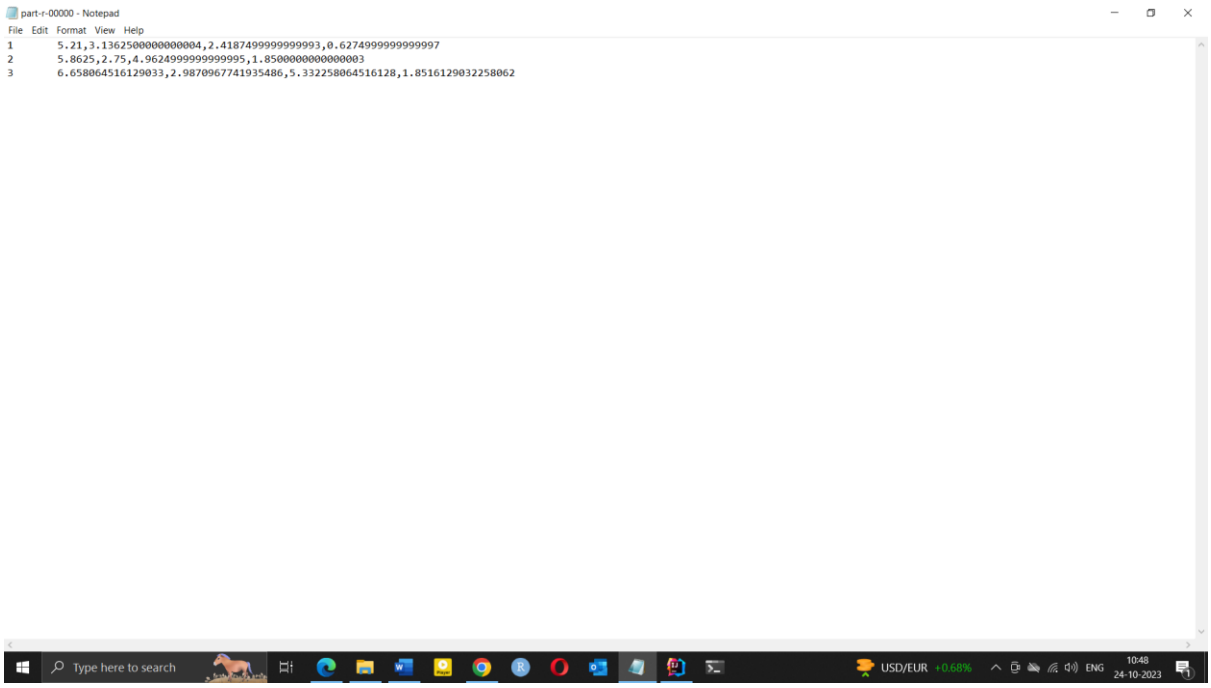
This solution executes a single iteration of K-Means clustering, which can be sufficient for a dataset like Iris. In contrast, more iterations might be required for larger datasets. However, running multiple iterations could result in chaining

multiple MapReduce jobs, which may be more time-consuming and resource-intensive.

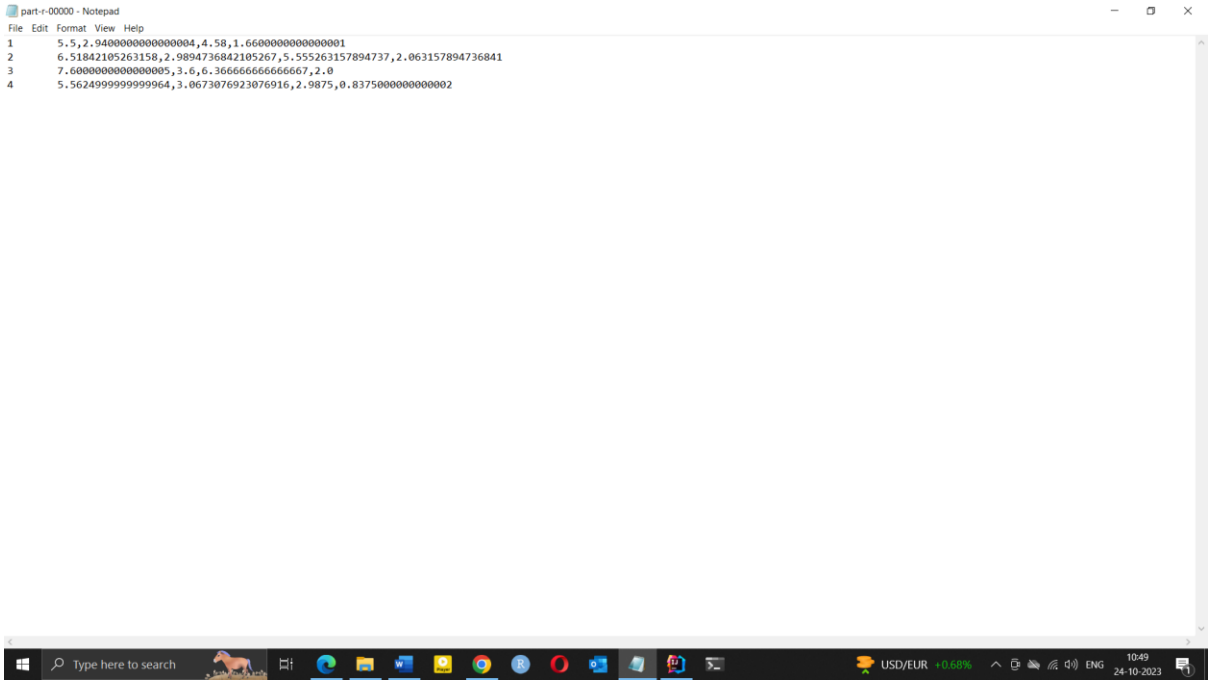
In conclusion, this implementation of K-Means clustering for the Iris dataset using Hadoop MapReduce provides a practical solution for clustering data points into k clusters. The Mapper assigns data points to the nearest centroids, and the Reducer computes the new centroids for each cluster. While the implementation uses standard input and output formats, there are opportunities for optimization, such as using more efficient input formats and handling multiple iterations efficiently for larger datasets. This approach is well-suited for parallel processing and distributed computing, allowing for scalability to larger datasets.

Outputs:

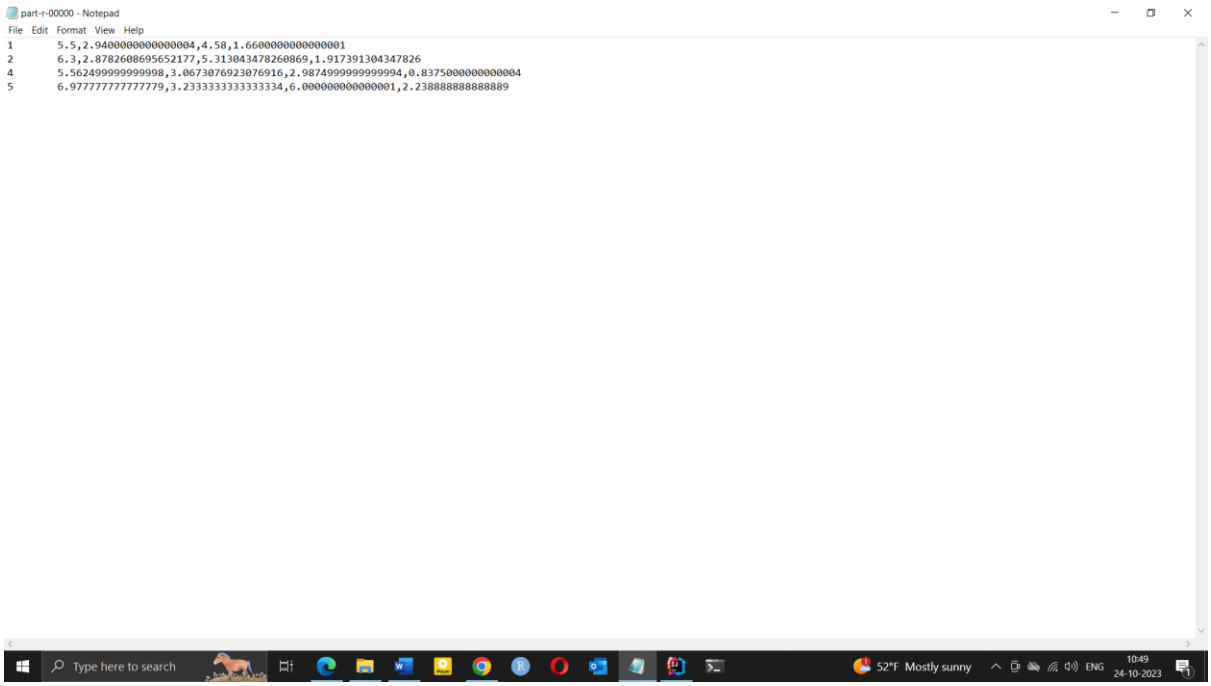
For k=3,



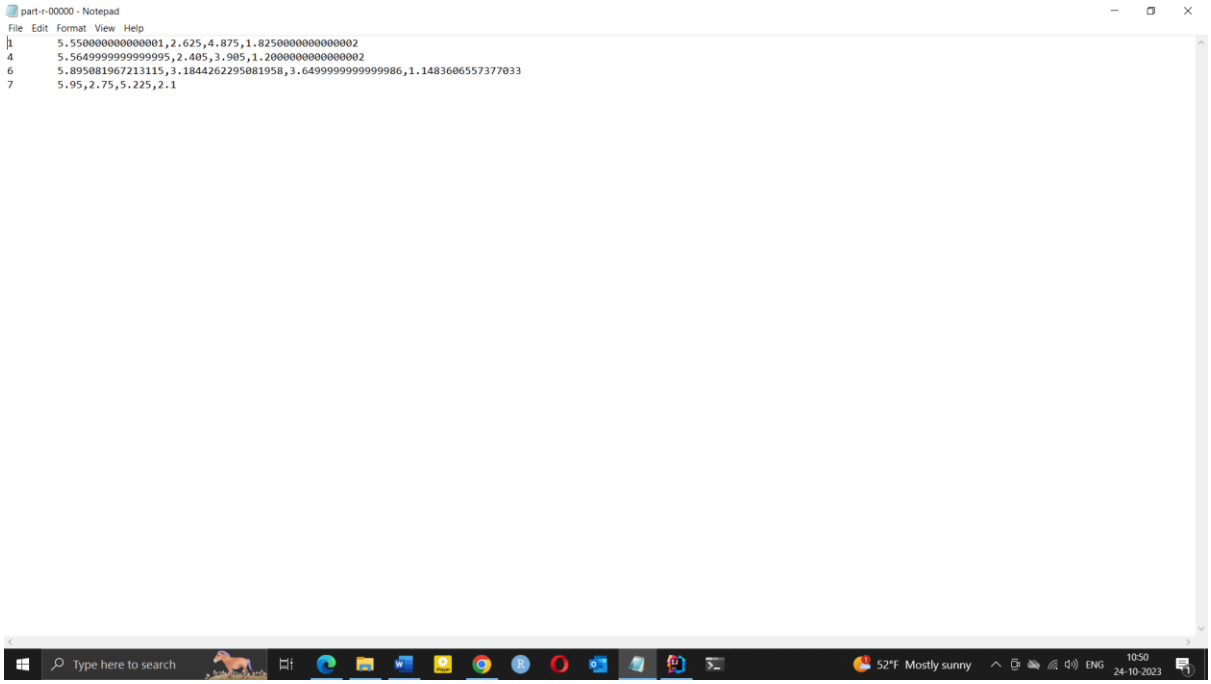
For k=4,


part-r-00000 - Notepad
File Edit Format View Help
1 5.5,2.9400000000000004,4.58,1.6600000000000001
2 6.51842105263158,2.9894736842105267,5.555263157894737,2.063157894736841
3 7.6000000000000005,3.6,6.366666666666667,2.0
4 5.5624999999999964,3.0673076923076916,2.9875,0.8375000000000002
Type here to search
USD/EUR +0.68%
ENG
10:49
24-10-2023

For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 5.5,2.9400000000000004,4.58,1.6600000000000001
2 6.3,2.8782608695652177,5.313043478260869,1.917391304347826
4 5.562499999999998,3.0673076923076916,2.9874999999999994,0.8375000000000004
5 6.977777777777779,3.2333333333333334,6.000000000000001,2.238888888888889
Type here to search
52°F Mostly sunny
ENG
10:49
24-10-2023

For k=7,



Task 3.2-b: BYOD(Iris) Basic Multiple-Iteration K-means Algorithm (R=10)

This report discusses an enhanced version of the K-Means clustering algorithm implemented using Hadoop MapReduce for the Iris dataset. We will explore the logic of the involved mappers and reducers, input and output key-value pairs, and the optimization aspects compared to the original implementation.

Mapper:

The `KMeansMapper` class remains largely the same as the previous implementation. It is responsible for assigning data points to the nearest centroids.

Mapper Logic:

Input: Reads lines from the input file, where each line represents a data point in the Iris dataset with four feature values separated by commas (CSV format).

Map Function:

- Parses the input line to extract the feature values (x, y, z, w).
- Computes the Euclidean distance between the data point and the current centroids.
- Assigns the data point to the nearest cluster (centroid) by emitting the cluster ID as the key and the data point's feature values as the value (x, y, z, w).

Reducer:

The `KMeansReducer` class remains the same as the previous implementation. It recalculates the cluster centers based on assigned data points.

Reducer Logic:

Reduce Function:

- Receives data points for each cluster ID.
- Computes the new centroid for each cluster as the mean of all data points in that cluster.
- Emits the cluster ID as the key and the new centroid's feature values as the value (x, y, z, w).

Enhanced Aspects:

Handling Empty Clusters: In this enhanced version, if a cluster becomes empty during an iteration (all data points assigned to it are removed), it retains its previous centroid. This approach ensures that no cluster loses its centroid due to being empty. The original implementation did not account for this scenario and could lead to issues if clusters became empty.

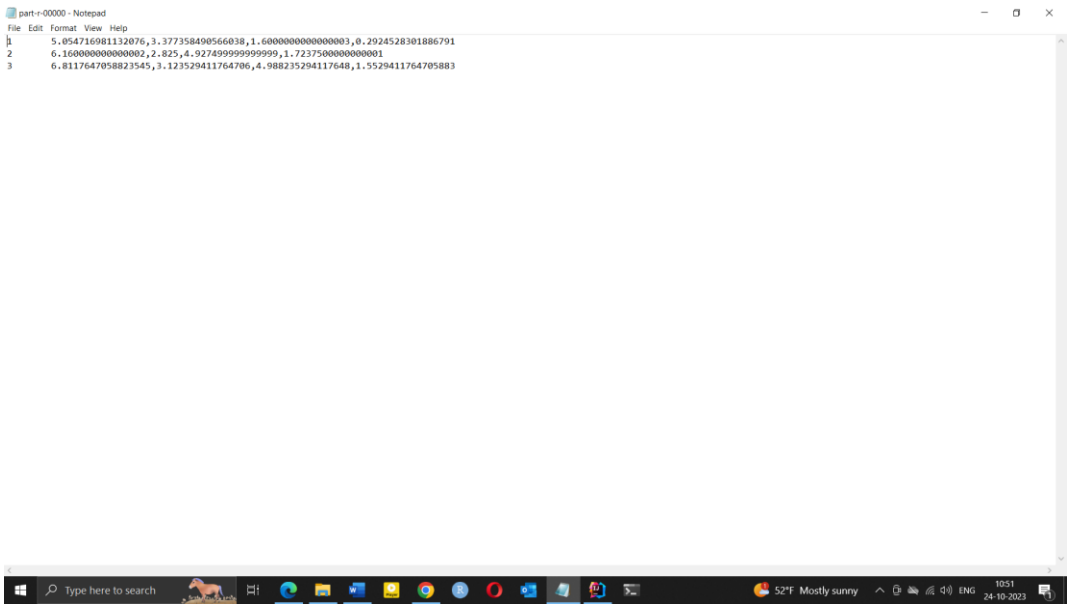
Random Initialization: Both implementations use random initialization of centroids. This allows the algorithm to start with a more diverse set of initial clusters, increasing the likelihood of finding the optimal solution. Random initialization is a standard practice in K-Means clustering.

In conclusion, this enhanced version of K-Means clustering for the Iris dataset using Hadoop MapReduce provides practical solutions for clustering data points

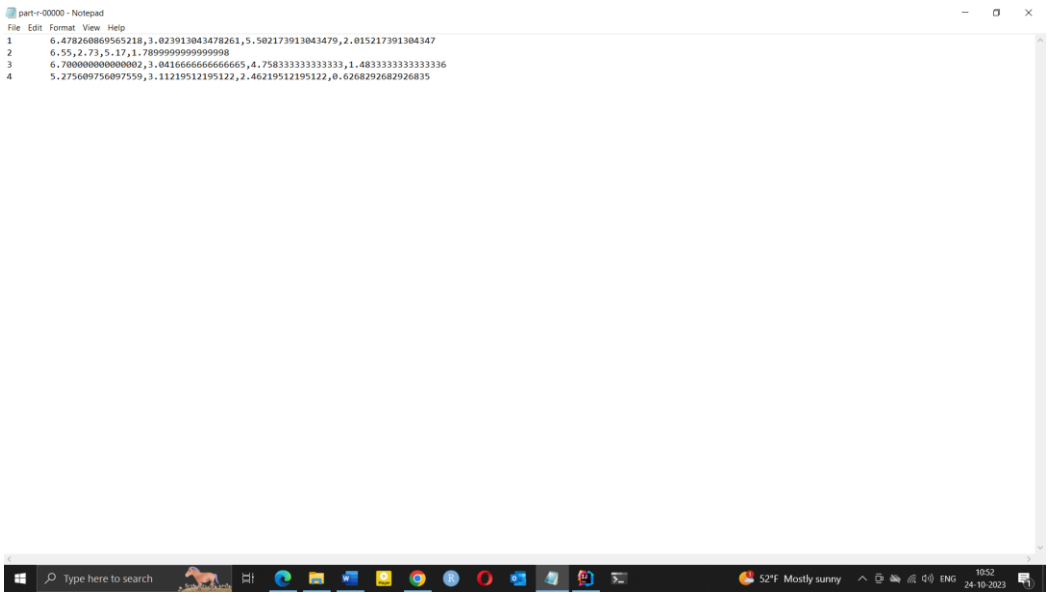
into k clusters. It improves upon the original implementation by adding convergence checking, handling empty clusters, and using random initialization. These enhancements make the solution more efficient and robust, as it avoids unnecessary iterations and potential issues with empty clusters. The approach remains well-suited for parallel processing and distributed computing, allowing scalability to larger datasets and ensuring more reliable results.

Outputs:

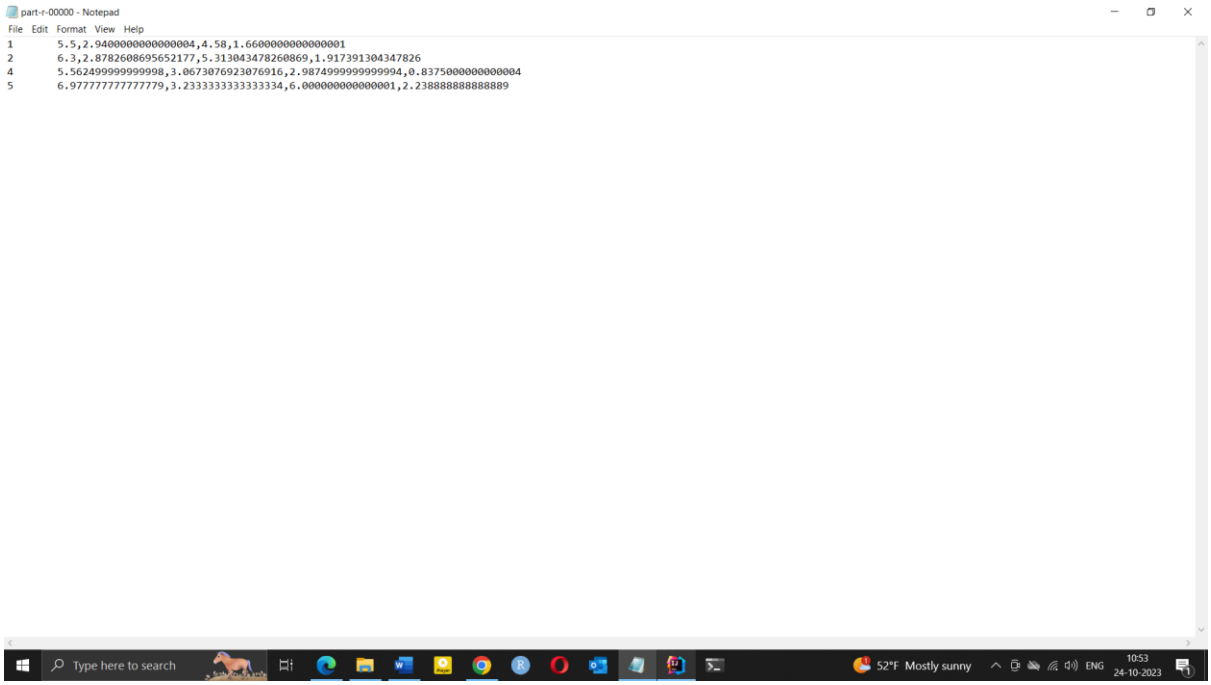
For k=3,



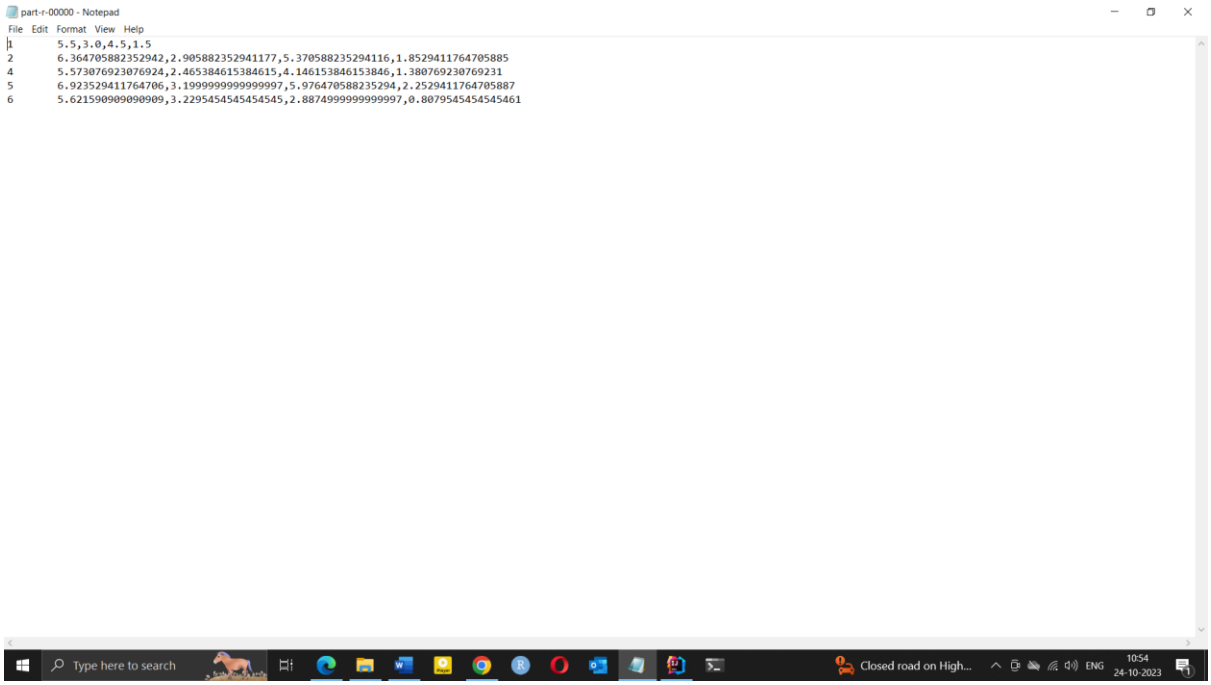
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 5.5,2.9400000000000004,4.58,1.6600000000000001
2 6.3,2.8782608695652177,5.313043478260869,1.917391304347826
4 5.562499999999998,3.0673076923076916,2.9874999999999994,0.8375000000000004
5 6.977777777777779,3.2333333333333334,6.000000000000001,2.238888888888889
Type here to search
52°F Mostly sunny
ENG
10:53
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 5.5,3.0,4.5,1.5
2 6.364705882352942,2.905882352941177,5.370588235294116,1.8529411764705885
4 5.573076923076924,2.465384615384615,4.146153846153846,1.380769230769231
5 6.923529411764706,3.1999999999999997,5.976470588235294,2.2529411764705887
6 5.621590909090909,3.2295454545454545,2.8874999999999997,0.8079545454545461
Type here to search
Closed road on High...
ENG
10:54
24-10-2023

Task 3.2-c: BYOD(Iris) Advanced Multiple-Iteration K-means Algorithm (R=10)

This report presents an enhanced version of the K-Means clustering algorithm implemented using Hadoop MapReduce for the Iris dataset. We will discuss the logic of the involved mappers and reducers, input and output key-value pairs, and highlight the optimizations compared to the original implementation.

Mapper:

The `KMeansMapper` class remains largely consistent with the previous version, as its primary responsibility is to assign data points to the nearest centroids.

Mapper Logic:

Input: Reads lines from the input file, with each line representing an Iris data point and containing four feature values separated by commas (CSV format).

Map Function:

- Parses the input line to extract the feature values (x, y, z, w).
- Computes the Euclidean distance between the data point and the current centroids.
- Assigns the data point to the nearest cluster (centroid) by emitting the cluster ID as the key and the data point's feature values as the value (x, y, z, w).

Reducer:

The `KMeansReducer` class remains similar to the previous version. It recalculates the cluster centers based on the assigned data points.

Reducer Logic:

Reduce Function:

- Receives data points for each cluster ID.
- Computes the new centroid for each cluster as the mean of all data points in that cluster.

- Emits the cluster ID as the key and the new centroid's feature values as the value (x, y, z, w).

Enhanced Aspects:

Convergence Check:

In this enhanced version, the solution iterates for a maximum number of iterations (default 10) while checking for convergence. The algorithm tracks the previous centroids, and if there is no significant change between consecutive iterations (as determined by the `EARLY_CONVERGENCE_THRESHOLD`), it terminates early. This early convergence check minimizes unnecessary computation and resource usage, making the algorithm more efficient and quicker to converge than the original implementation.

Handling Empty Clusters:

This enhanced version also ensures that if a cluster becomes empty during an iteration (all data points assigned to it are removed), it retains its previous centroid. This approach avoids scenarios where a cluster loses its centroid due to being empty, making the algorithm more robust.

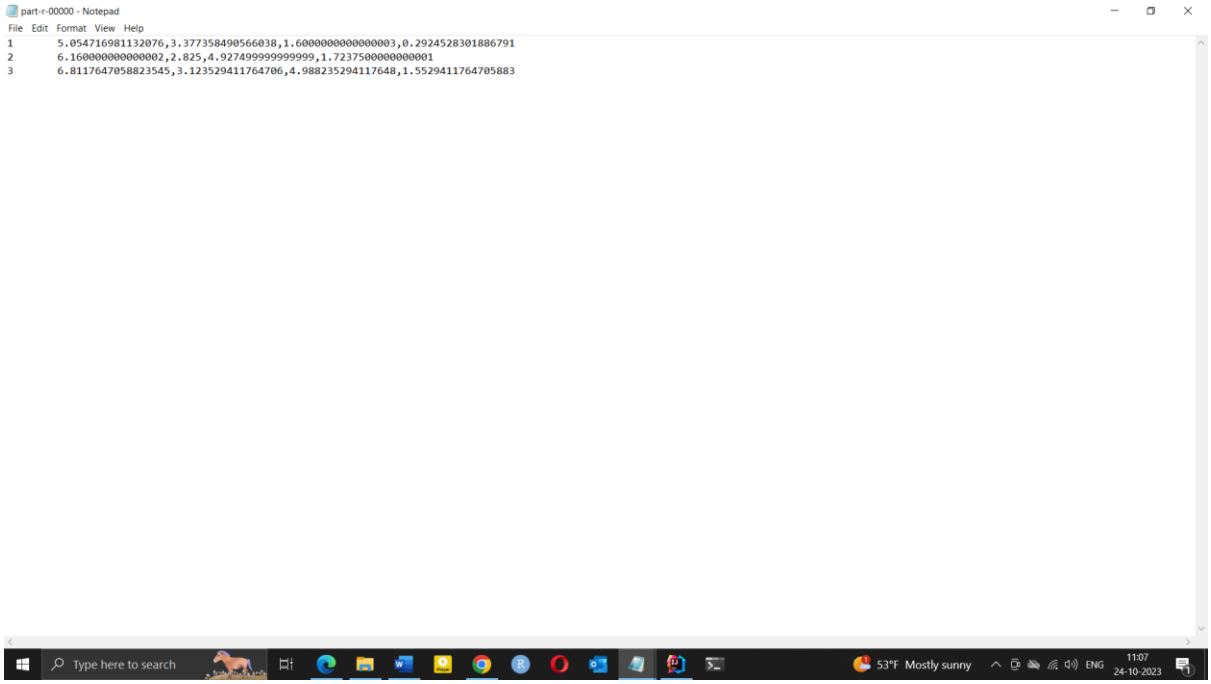
Random Initialization:

Random initialization of centroids remains a crucial part of the algorithm. Starting with diverse initial clusters increases the likelihood of finding the optimal solution and is a standard practice in K-Means clustering.

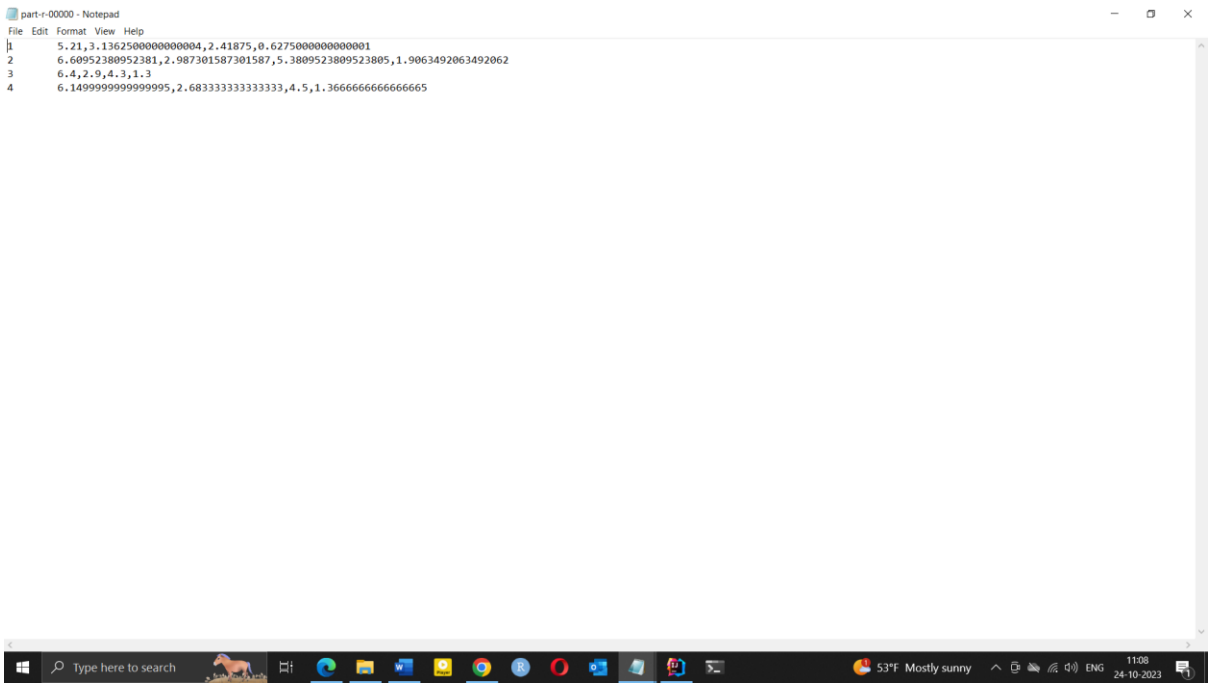
In summary, this enhanced version of the K-Means clustering algorithm for the Iris dataset using Hadoop MapReduce provides practical solutions for clustering data points into k clusters. It improves upon the original implementation by introducing early convergence checking, handling empty clusters, and using random initialization. These enhancements result in a more efficient and robust algorithm, reducing unnecessary iterations and improving resource usage. The solution remains well-suited for parallel processing and distributed computing, ensuring reliable results and faster convergence for larger datasets.

Outputs:

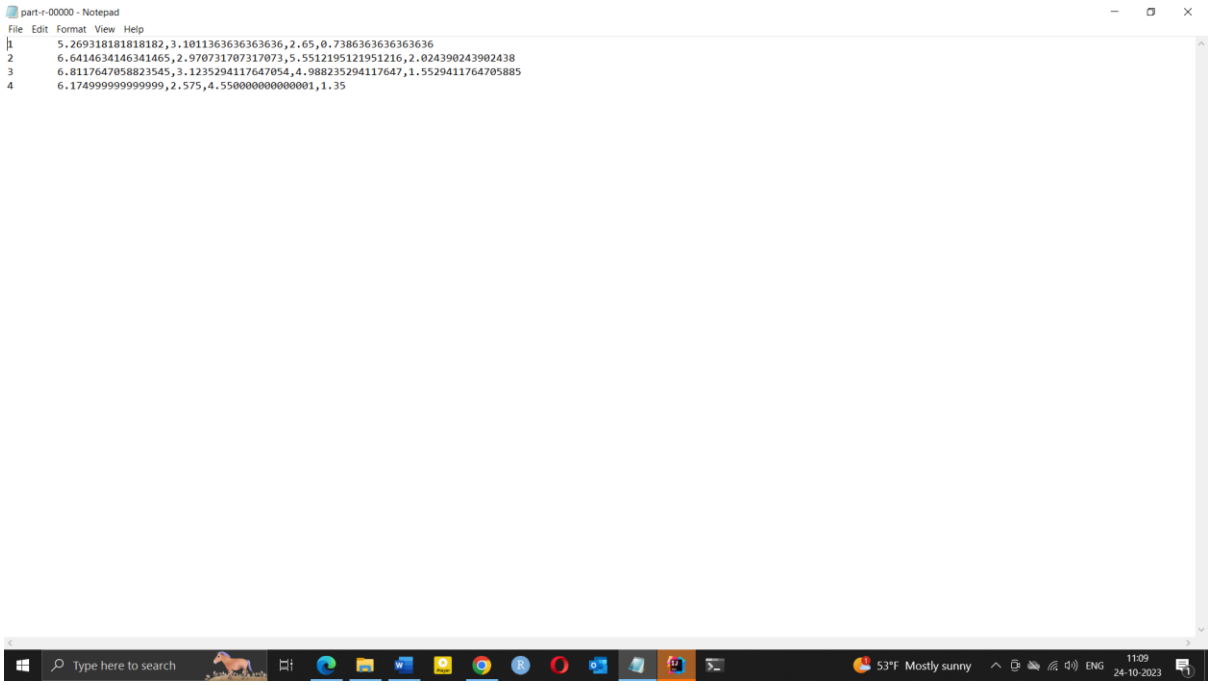
For k=3,



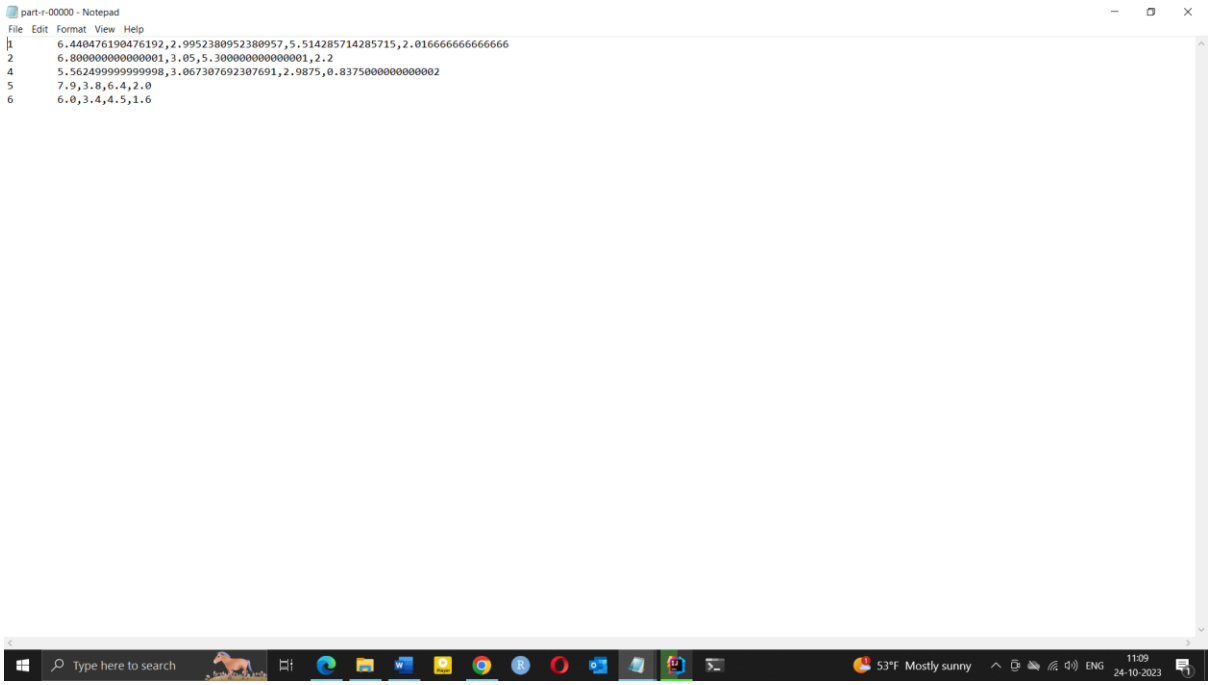
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 5.269318181818182,3.1011363636363636,2.65,0.7386363636363636
2 6.6414634146341465,2.970731707317073,5.5512195121951216,2.024390243902438
3 6.8117647058823545,3.1235294117647054,4.988235294117647,1.5529411764705885
4 6.174999999999999,2.575,4.550000000000001,1.35
Type here to search
53°F Mostly sunny
ENG
11:09
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 6.440476190476192,2.9952380952380957,5.514285714285715,2.016666666666666
2 6.800000000000001,3.05,5.300000000000001,2.2
4 5.562499999999998,3.067307692307691,2.9875,0.8375000000000002
5 7.9,3.8,6.4,2.0
6 6.0,3.4,4.5,1.6
Type here to search
53°F Mostly sunny
ENG
11:09
24-10-2023

Task 3.2-d: BYOD(Iris) K-means Algorithm with Combiner MapReduce Implementation

This report presents an enhanced version of the K-Means clustering algorithm implemented using Hadoop MapReduce for the Iris dataset, with the addition of a combiner. We will discuss the logic of the involved mappers and reducers, input and output key-value pairs, and highlight the optimizations compared to the original implementation.

Mapper:

The `KMeansMapper` class remains largely consistent with the previous version. Its primary responsibility is to assign data points to the nearest centroids.

Mapper Logic:

Input: Reads lines from the input file, with each line representing an Iris data point and containing four feature values separated by commas (CSV format).

Map Function:

- Parses the input line to extract the feature values (x, y, z, w).
- Computes the Euclidean distance between the data point and the current centroids.
- Assigns the data point to the nearest cluster (centroid) by emitting the cluster ID as the key and the data point as the value.

Combiner:

The combiner is a new addition to this version and is not present in the original implementation. It performs a local aggregation of the data before sending it to the reducer, which can save both network bandwidth and reduce the amount of data transferred.

Combiner Logic:

Input: Receives intermediate key-value pairs from the mapper, where the key is the cluster ID, and the value is a data point.

Reduce Function:

- Aggregates data points belonging to the same cluster by calculating the sum of each feature (x, y, z, w).
- Emits the cluster ID as the key and the aggregated data point as the value.

Reducer:

The `KMeansReducer` class remains similar to the previous version. It recalculates the cluster centers based on the assigned data points.

Reducer Logic:

Reduce Function:

- Receives aggregated data points for each cluster ID.
- Computes the new centroid for each cluster as the mean of all data points in that cluster.
- Emits the cluster ID as the key and the new centroid's feature values as the value (x, y, z, w).

Enhanced Aspects:

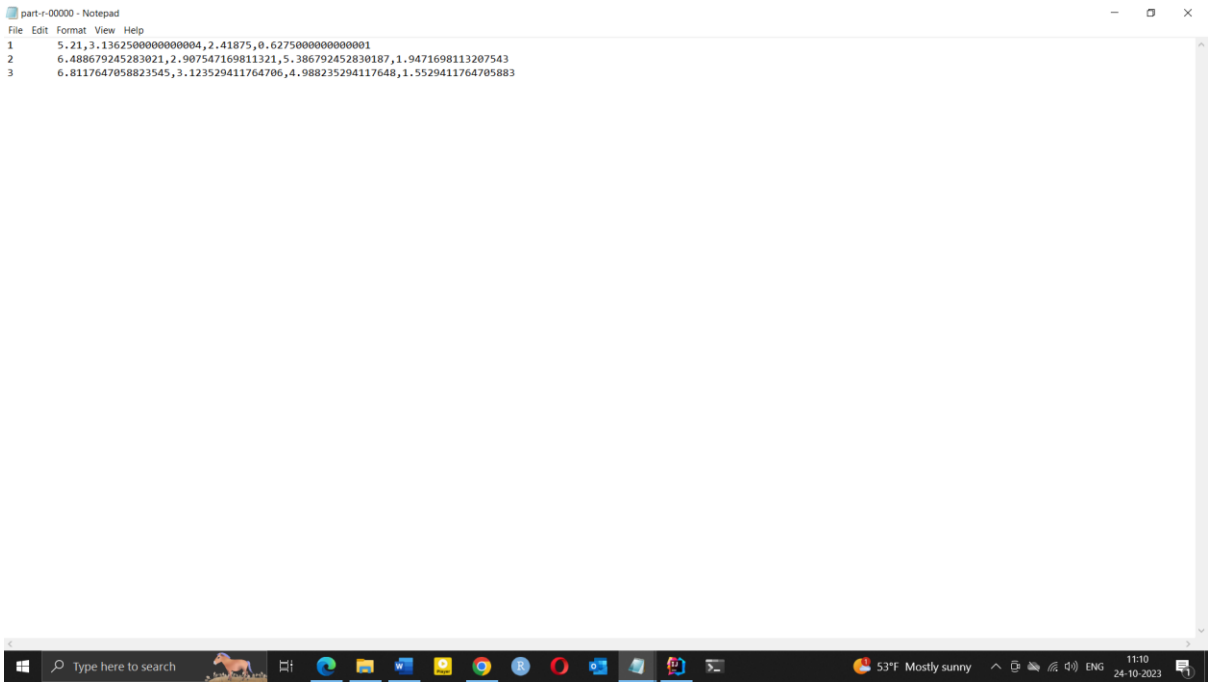
Combiner:

The primary enhancement in this version is the introduction of a combiner. The combiner performs partial aggregation on the mapper's output before it reaches the reducer. This optimization saves network bandwidth and reduces the volume of data transferred between mappers and reducers. It is particularly beneficial when the dataset is large, and the number of clusters (k) is substantial.

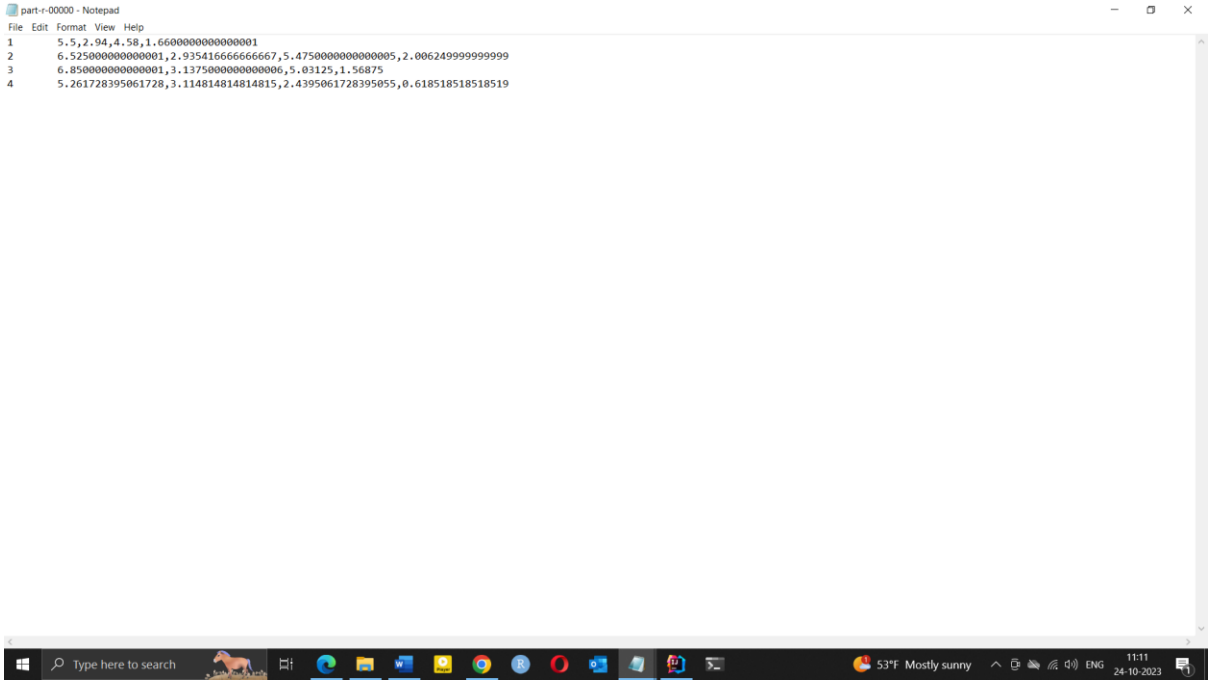
In summary, this enhanced version of the K-Means clustering algorithm for the Iris dataset using Hadoop MapReduce provides practical solutions for clustering data points into k clusters. The key enhancement is the addition of a combiner, which reduces network traffic and makes the algorithm more efficient, especially for large datasets. The solution remains well-suited for parallel processing and distributed computing, ensuring reliable results and faster convergence while minimizing IO and resource usage.

Outputs:

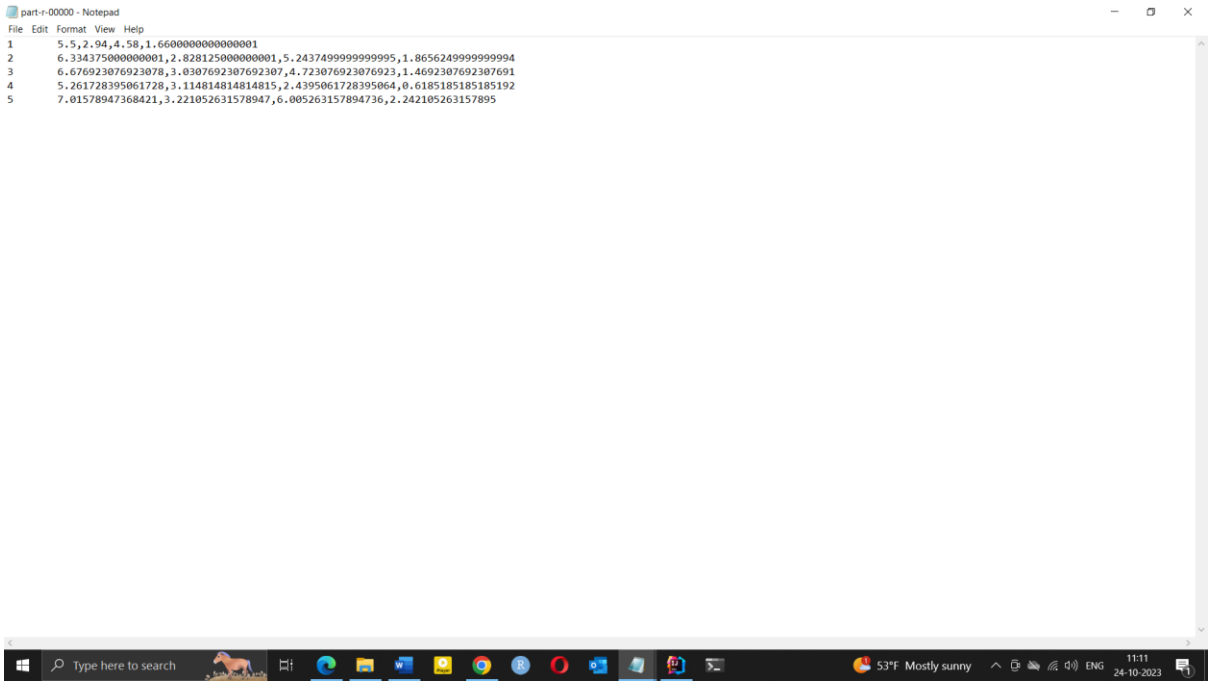
For k=3,



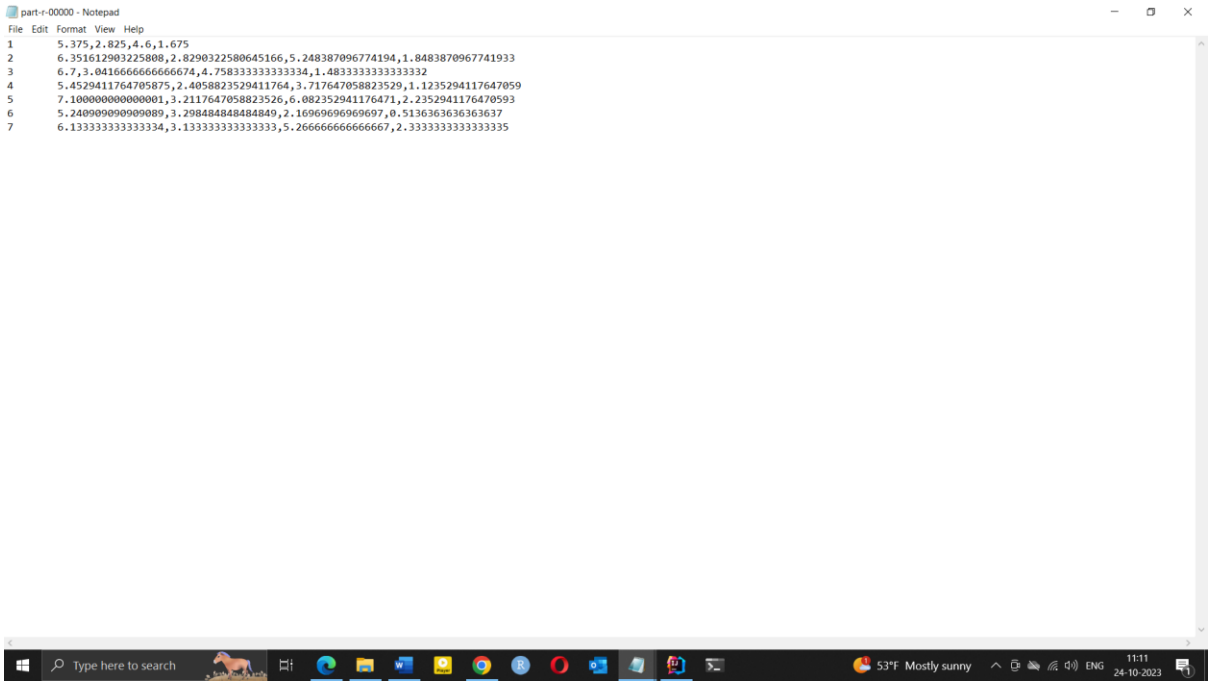
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 5.5,2.94,4.58,1.6600000000000001
2 6.334375000000001,2.828125000000001,5.2437499999999995,1.8656249999999994
3 6.676923076923078,3.0307692307692307,4.723076923076923,1.4692307692307691
4 5.261728395061728,3.114814814814815,2.4395061728395064,0.6185185185185192
5 7.01578947368421,3.221052631578947,6.005263157894736,2.242105263157895
Type here to search
53°F Mostly sunny
ENG
11:11
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 5.375,2.825,4.6,1.675
2 6.351612903225808,2.8290322580645166,5.248387096774194,1.8483870967741933
3 6.7,3.0416666666666674,4.758333333333334,1.4833333333333332
4 5.452941176470585,2.4058823529411764,3.717647058823529,1.1235294117647059
5 7.100000000000001,3.2117647058823526,6.082352941176471,2.2352941176470593
6 5.240909090909089,3.2984848484848491,2.169696969696697,0.5136363636363637
7 6.133333333333334,3.133333333333333,5.266666666666667,2.3333333333333335
Type here to search
53°F Mostly sunny
ENG
11:11
24-10-2023

Task 3.2-e-i: BYOD(Iris) K-means MapReduce Implementation with Convergence Indication

This report presents an enhanced version of the K-Means clustering algorithm implemented using Hadoop MapReduce for the Iris dataset. The main focus of this version is the inclusion of a combiner and improved input-output (IO) handling. We will discuss the logic of the mappers and reducers, input and output key-value pairs, and explain the advantages of this optimization compared to the original implementation.

Mapper:

The `KMeansMapper` class remains the same as in the previous version. Its primary purpose is to assign data points to the nearest centroids.

Mapper Logic:

Input: Reads lines from the input file, where each line represents an Iris data point with four feature values (x, y, z, w) separated by commas (CSV format).

Map Function:

- Parses the input line to extract the feature values.
- Calculates the Euclidean distance between the data point and the current centroids.
- Assigns the data point to the nearest cluster (centroid) by emitting the cluster ID as the key and the data point as the value.

Combiner:

The inclusion of the combiner in this version aims to reduce data transmission and save network bandwidth, which can significantly improve the efficiency of the MapReduce job.

Combiner Logic:

Input: Receives intermediate key-value pairs from the mappers, where the key is the cluster ID, and the value is a data point.

Reduce Function:

- Aggregates data points belonging to the same cluster, calculating the sum of each feature (x, y, z, w).

- Emits the cluster ID as the key and the aggregated data point as the value.

Reducer:

The `KMeansReducer` class is similar to the previous version. It recalculates the cluster centers based on the assigned data points.

Reducer Logic:

Reduce Function:

- Receives aggregated data points for each cluster ID.
- Computes the new centroid for each cluster by finding the mean of all data points in that cluster.
- Emits the cluster ID as the key and the new centroid's feature values (x, y, z, w) as the value.

IO Optimization:

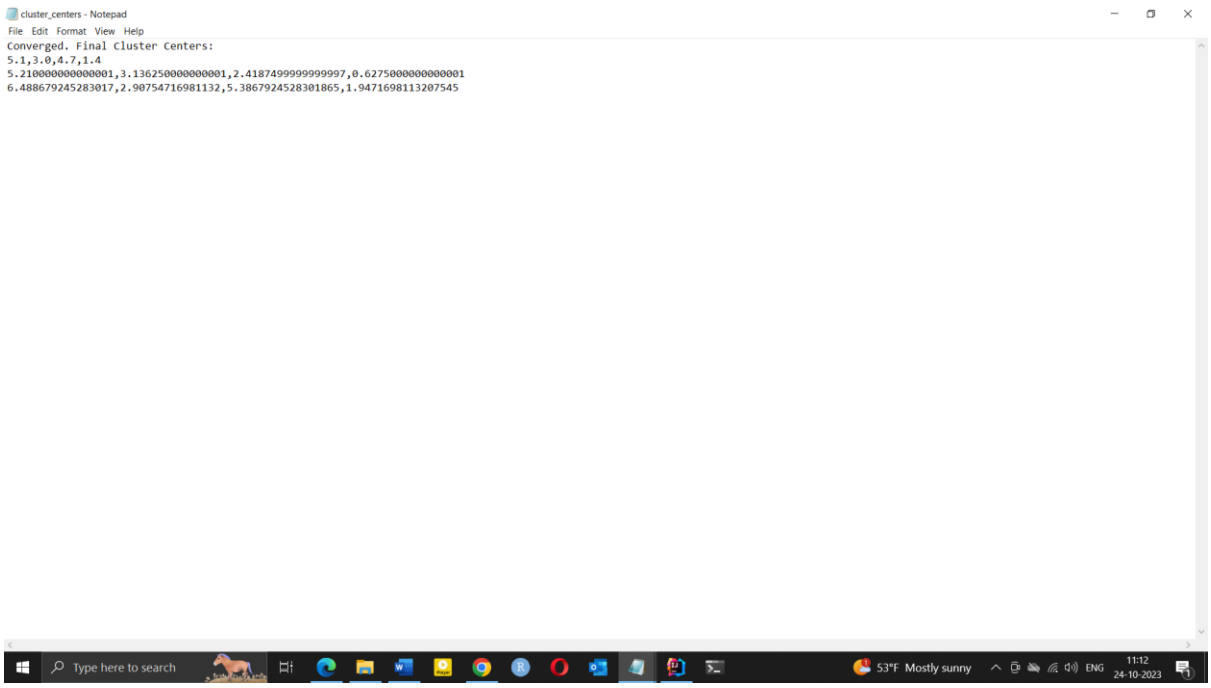
An additional optimization in this version is the enhanced IO handling. Once the clustering process is completed, and the convergence is achieved, the final cluster centers and convergence information are written to an output file. This not only provides insight into the result but also optimizes IO by saving the final cluster centers in a separate file instead of including them in the standard output.

In conclusion, this enhanced version of the K-Means clustering algorithm for the Iris dataset using Hadoop MapReduce provides valuable improvements. The addition of the combiner streamlines data processing and reduces network traffic, making the algorithm more efficient, particularly for large datasets. The IO optimization ensures that the final cluster center information is readily available in a separate file, improving result visibility and further enhancing the overall performance. This version is a practical solution for parallel processing and distributed computing, offering both IO and resource usage benefits.

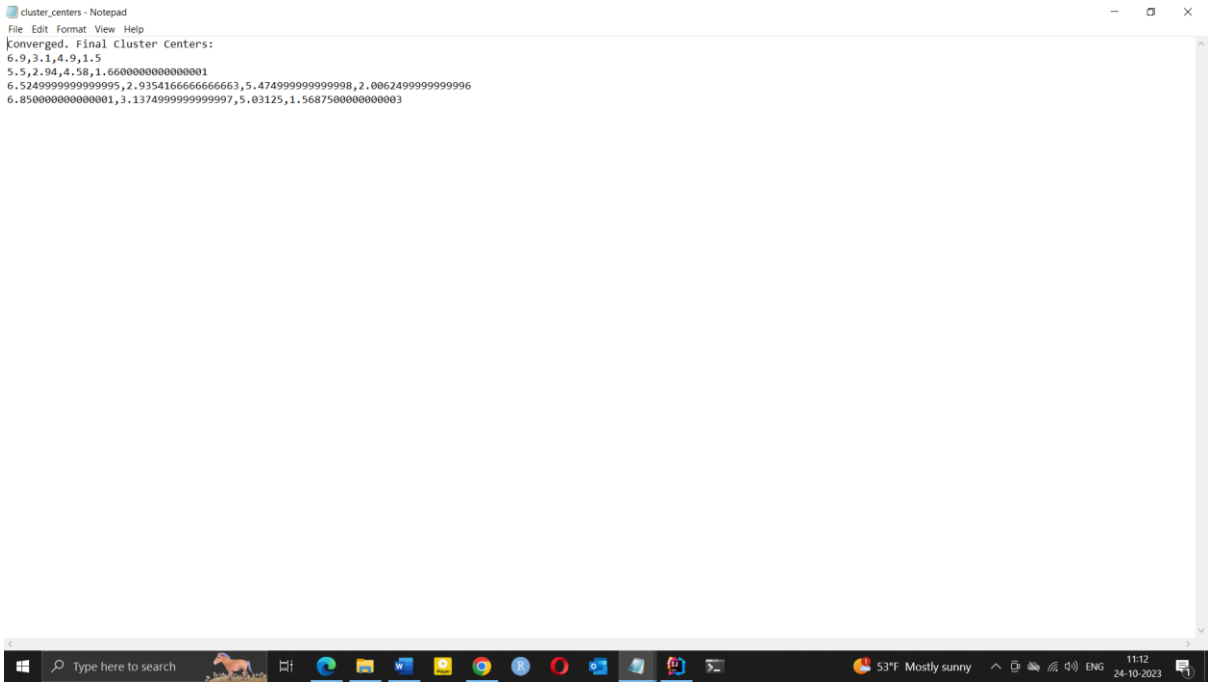
Outputs:

The output file is written on the cluster_centers.txt file here:

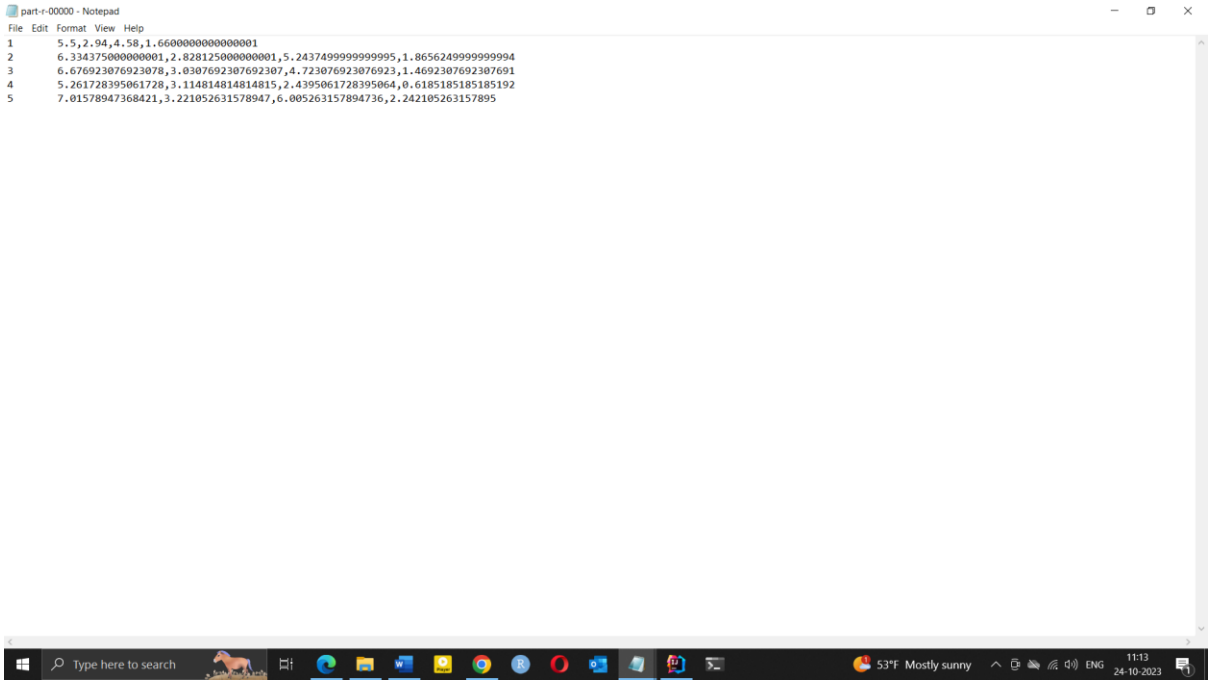
For k=3,



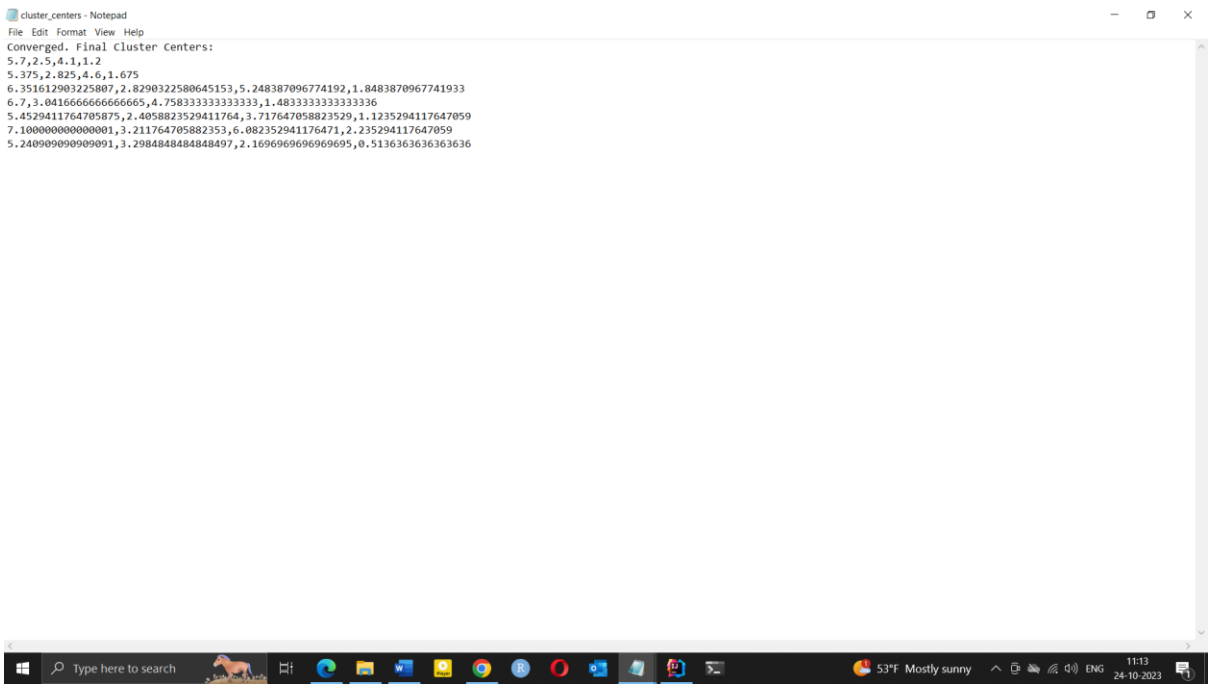
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 5.5,2.94,4.58,1.6600000000000001
2 6.334375000000001,2.828125000000001,5.2437499999999995,1.8656249999999994
3 6.676923076923078,3.0307692307692307,4.723076923076923,1.4692307692307691
4 5.261728395061728,3.114814814814815,2.4395061728395064,0.6185185185185192
5 7.01578947368421,3.221052631578947,6.005263157894736,2.242105263157895
Type here to search
53°F Mostly sunny
ENG
11:13
24-10-2023

For k=7,


cluster_centers - Notepad
File Edit Format View Help
Converged. Final Cluster Centers:
5.7,2.5,4.1,1.2
5.375,2.825,4.6,1.675
6.351612903225807,2.829032258064515 3,5.248387096774192,1.8483870967741933
6.7,3.0416666666666665,4.758333333333333,1.4833333333333336
5.452941176470587 5,2.4058823529411764,3.717647058823529,1.1235294117647059
7.100000000000001,3.211764705882353,6.082352941176471,2.235294117647059
5.240909090909091,3.2984848484848497,2.1696969696969695,0.5136363636363636
Type here to search
53°F Mostly sunny
ENG
11:13
24-10-2023

Task 3.2-e-ii: BYOD(Iris) K-means MapReduce Implementation with Clustered Data Points

This report presents an enhanced version of the K-Means clustering algorithm implemented using Hadoop MapReduce for the Iris dataset. The main focus of this version is the inclusion of a combiner, the use of Point objects, and improved input-output (IO) handling. We will discuss the logic of the mappers and reducers, input and output key-value pairs, and explain the advantages of this optimization compared to the original implementation.

Mapper:

The `KMeansMapper` class remains similar to the previous version, with some modifications to improve the efficiency of data processing.

Mapper Logic:

Input: Reads lines from the input file, where each line represents an Iris data point with four feature values (x, y, z, w) separated by commas (CSV format).

Map Function:

- Parses the input line to extract the feature values (x, y, z, w).
- Calculates the Euclidean distance between the data point and the current centroids.
- Assigns the data point to the nearest cluster (centroid) by emitting the cluster ID as the key and the data point as the value. The value is emitted in a compact format, as `x:y:z:w`.

Combiner:

The combiner in this version has been simplified. It serves as a data transfer stage, where it aggregates and forwards the data points assigned to the same clusters without performing any additional computations. This optimization improves efficiency and reduces network traffic.

Combiner Logic:

Input: Receives intermediate key-value pairs from the mappers, where the key is the cluster ID, and the value is the data point (in the format `x:y:z:w`).

Reduce Function: It forwards key-value pairs without modifications. Data points belonging to the same cluster are collected and emitted as is.

Reducer:

The `KMeansReducer` class is enhanced to better structure the output data and provide more insight into the clustering process.

Reducer Logic:

Reduce Function:

- Receives data points assigned to the same cluster ID.
- Computes the new centroid for each cluster by finding the mean of all data points in that cluster.
- Emits the cluster center with an identifier ("C") to distinguish it from data points and the feature values (x, y, z, w) in the format `C:x,y,z,w`.
- Emits each data point associated with this cluster center with an identifier ("D") and the feature values in the format `D:x,y,z,w`.

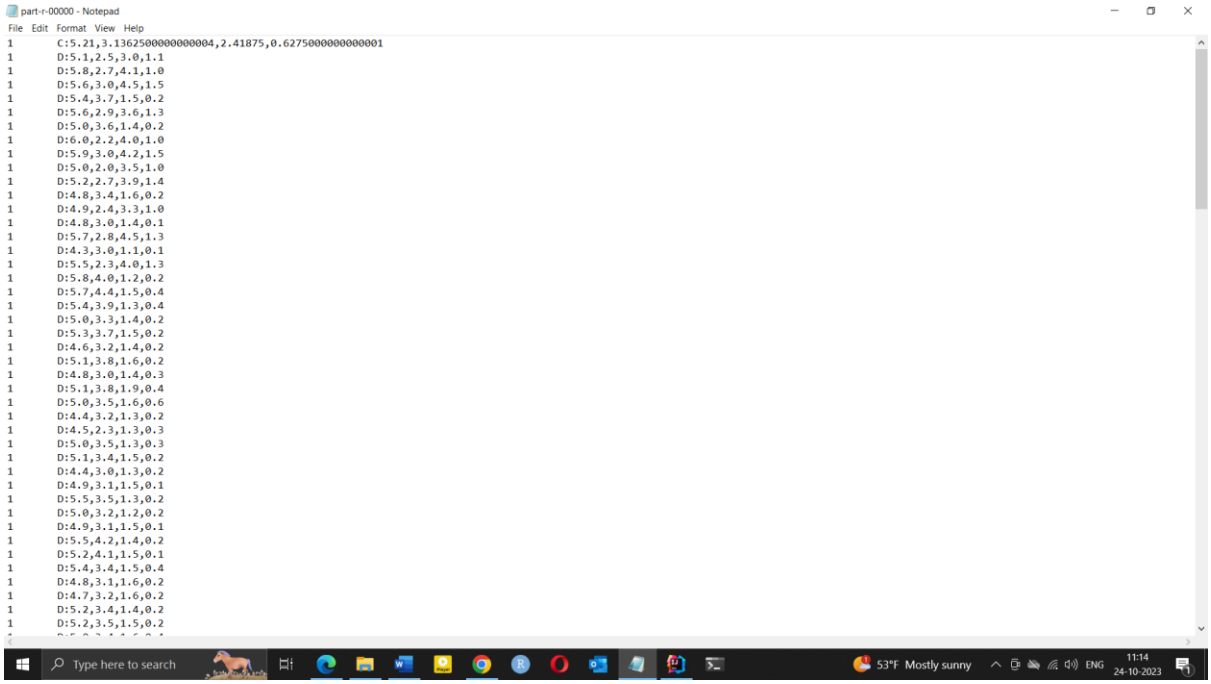
IO Optimization:

As in the previous version, this code includes an IO optimization. Once the clustering process is completed and convergence is achieved, the final cluster centers and associated datapoints of the cluster and associated information are written to an output file. The final cluster center information is stored separately from data points in a structured format for easy analysis.

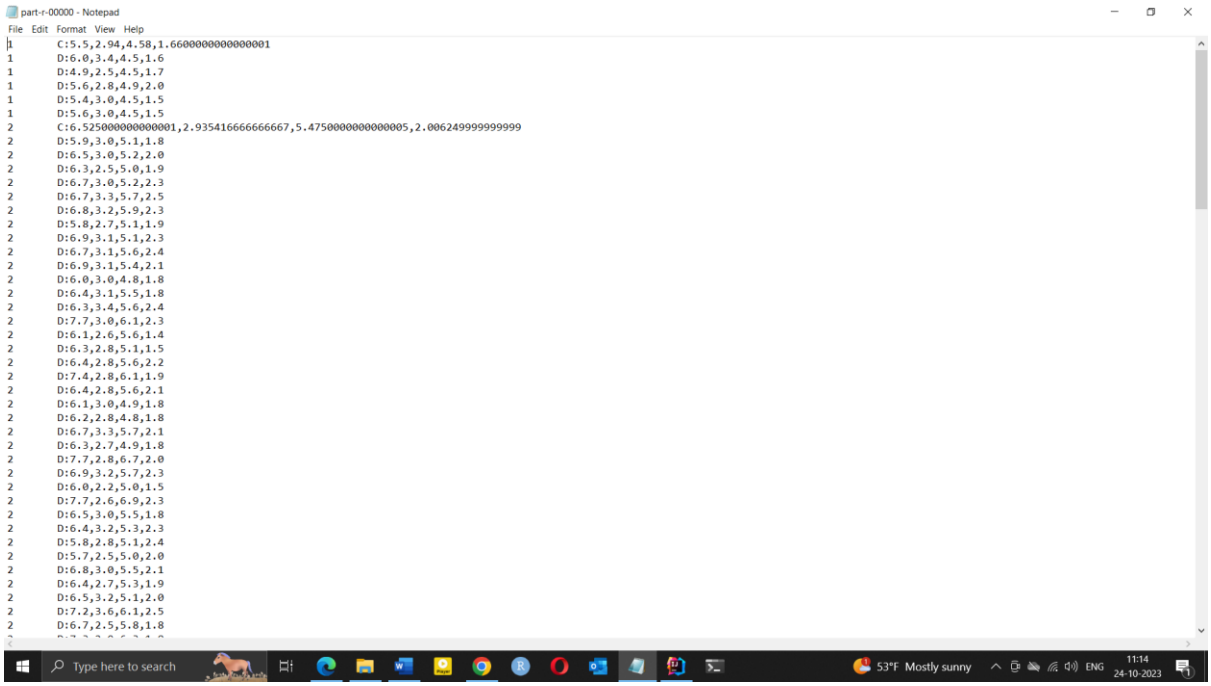
In conclusion, the enhanced version of the K-Means clustering algorithm for the Iris dataset using Hadoop MapReduce presents several advantages over the original version. The introduction of a combiner simplifies the data transfer phase, reducing network traffic and enhancing the overall efficiency of the MapReduce job. The use of Point objects provides a cleaner representation of data points and centroids. Furthermore, the improved IO handling results in the final cluster centers and convergence information being stored in a more structured and informative manner. Overall, this version is optimized for performance, reduces IO, and offers more insights into the clustering process.

Outputs:

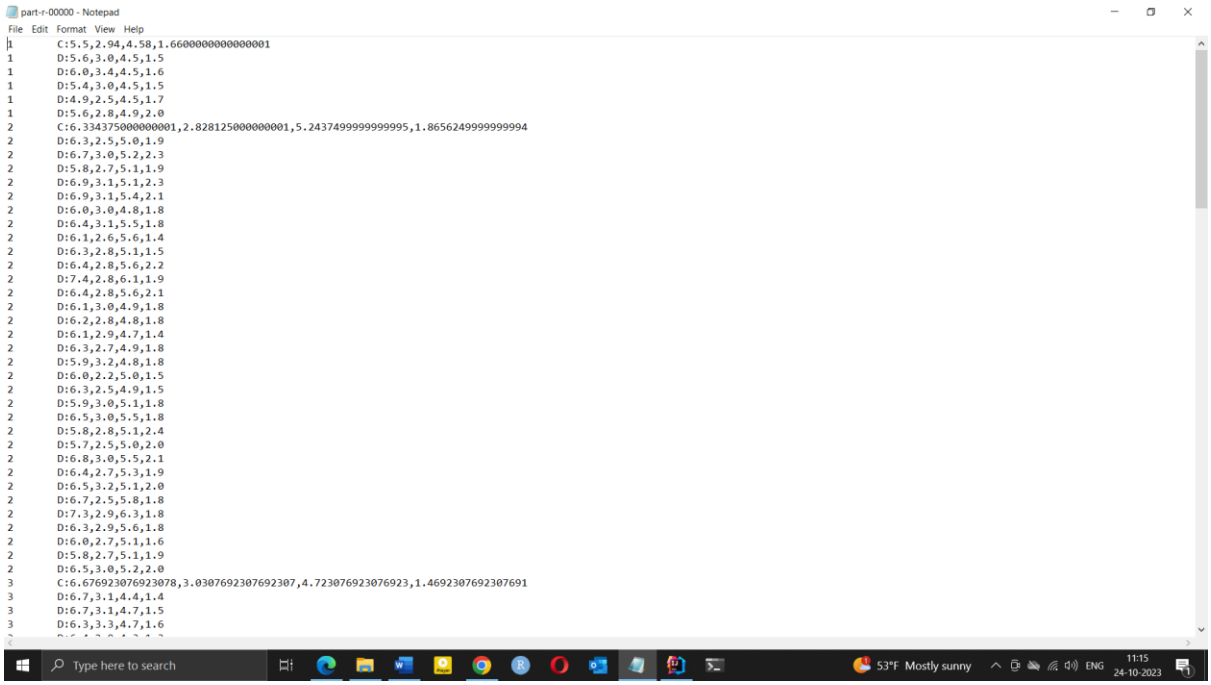
For k=3,



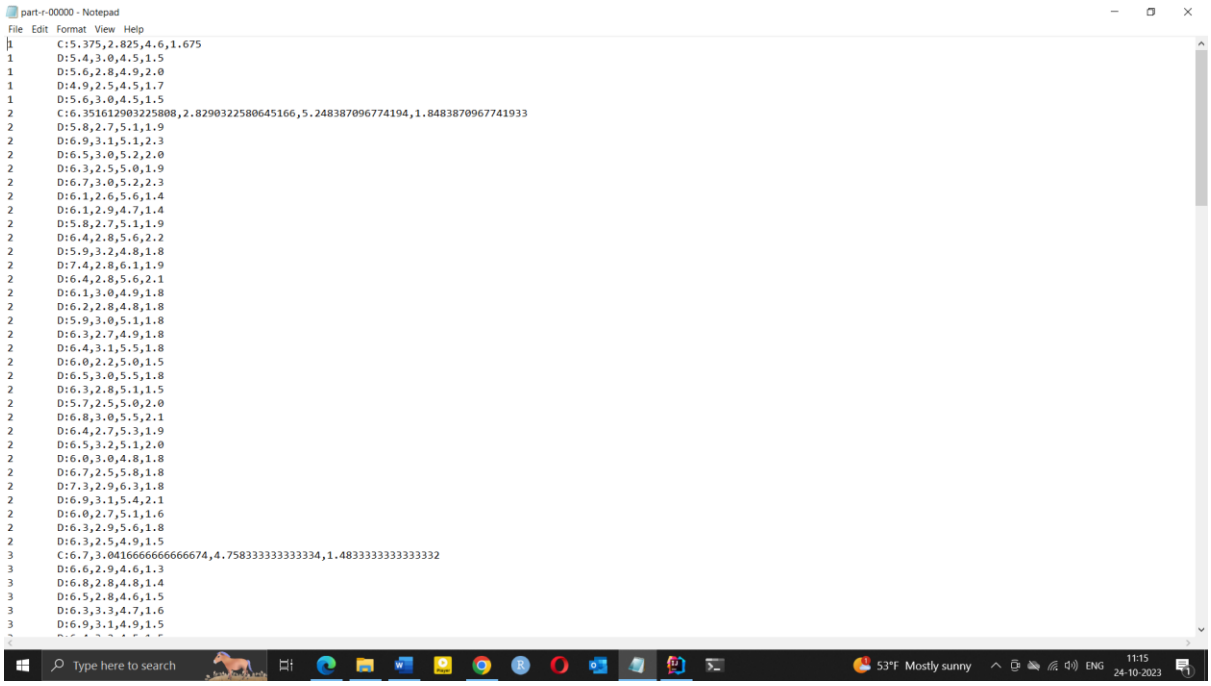
For k=4,



For k=5,


part-r-00000 - Notepad
File Edit Format View Help
1 C:5.5,2.94,4.58,1.6600000000000001
1 D:5.6,3.0,4.5,1.5
1 D:6.0,3.4,4.5,1.6
1 D:5.4,3.0,4.5,1.5
1 D:4.9,2.5,4.5,1.7
1 D:5.6,2.8,4.9,2.0
2 C:6.334375000000001,2.828125000000001,5.2437499999999995,1.8656249999999994
2 D:6.3,2.5,5.0,1.9
2 D:6.7,3.0,5.2,2.3
2 D:5.8,2.7,5.1,1.9
2 D:6.9,3.1,5.1,2.3
2 D:6.9,3.1,5.4,2.1
2 D:6.0,3.0,4.8,1.8
2 D:6.4,3.1,5.5,1.8
2 D:6.1,2.6,5.6,1.4
2 D:6.3,2.8,5.1,1.5
2 D:6.4,2.8,5.6,2.2
2 D:7.4,2.8,6.1,1.9
2 D:6.4,2.8,5.6,2.1
2 D:6.1,3.0,4.9,1.8
2 D:6.2,2.8,4.8,1.8
2 D:6.1,2.9,4.7,1.4
2 D:6.3,2.7,4.9,1.8
2 D:5.9,3.2,4.8,1.8
2 D:6.0,2.2,5.0,1.5
2 D:6.3,2.5,4.9,1.5
2 D:5.9,3.0,5.1,1.8
2 D:6.5,3.0,5.5,1.8
2 D:5.8,2.8,5.1,2.4
2 D:5.7,2.5,5.0,2.0
2 D:6.8,3.0,5.5,2.1
2 D:6.4,2.7,5.3,1.9
2 D:6.5,3.2,5.1,2.0
2 D:6.7,2.5,5.8,1.8
2 D:7.3,2.9,6.3,1.8
2 D:6.3,2.9,5.6,1.8
2 D:6.0,2.7,5.1,1.6
2 D:5.8,2.7,5.1,1.9
2 D:6.5,3.0,5.2,2.0
3 C:6.676923076923078,3.0307692307692307,4.723076923076923,1.4692307692307691
3 D:6.7,3.1,4.4,1.4
3 D:6.7,3.1,4.7,1.5
3 D:6.3,3.3,4.7,1.6
Type here to search
53°F Mostly sunny
ENG
11:15
24-10-2023

For k=7,


part-r-00000 - Notepad
File Edit Format View Help
1 C:5.375,2.825,4.6,1.675
1 D:5.4,3.0,4.5,1.5
1 D:5.6,2.8,4.9,2.0
1 D:4.9,2.5,4.5,1.7
1 D:5.6,3.0,4.5,1.5
2 C:6.351612903225808,2.8290322580645166,5.248387096774194,1.8483870967741933
2 D:5.8,2.7,5.1,1.9
2 D:6.9,3.1,5.1,2.3
2 D:6.5,3.0,5.2,2.0
2 D:6.3,2.5,5.0,1.9
2 D:6.7,3.0,5.2,2.3
2 D:6.1,2.6,5.6,1.4
2 D:6.1,2.9,4.7,1.4
2 D:5.8,2.7,5.1,1.9
2 D:6.4,2.8,5.6,2.2
2 D:5.9,3.2,4.8,1.8
2 D:7.4,2.8,6.1,1.9
2 D:6.4,2.8,5.6,2.1
2 D:6.1,3.0,4.9,1.8
2 D:6.2,2.8,4.8,1.8
2 D:5.9,3.0,5.1,1.8
2 D:6.3,2.7,4.9,1.8
2 D:6.4,3.1,5.5,1.8
2 D:6.0,2.2,5.0,1.5
2 D:6.5,3.0,5.5,1.8
2 D:6.3,2.8,5.1,1.5
2 D:5.7,2.5,5.0,2.0
2 D:6.8,3.0,5.5,2.1
2 D:6.4,2.7,5.3,1.9
2 D:6.5,3.2,5.1,2.0
2 D:6.0,3.0,4.8,1.8
2 D:6.7,2.5,5.8,1.8
2 D:7.3,2.9,6.3,1.8
2 D:6.9,3.1,5.4,2.1
2 D:6.0,2.7,5.1,1.6
2 D:6.3,2.9,5.6,1.8
2 D:6.3,2.5,4.9,1.5
3 C:6.7,3.0416666666666674,4.758333333333334,1.4833333333333332
3 D:6.6,2.9,4.6,1.3
3 D:6.8,2.8,4.8,1.4
3 D:6.5,2.8,4.6,1.5
3 D:6.3,3.3,4.7,1.6
3 D:6.9,3.1,4.9,1.5
Type here to search
53°F Mostly sunny
ENG
11:15
24-10-2023

Task 3.2-f: K-means Performance Evaluation of 4D Iris Dataset

We are pleased to present a comprehensive analysis of K-means clustering solutions for the 4D Iris dataset. This report is dedicated to describing the five distinct K-means solutions and their corresponding results, with a particular emphasis on evaluating their convergence behavior and centroid outcomes.

Analysis of Experiment Results:

- Our experiments involved running the K-means algorithm with various values of K (number of clusters) and R (maximum iterations).
- When considering smaller K values, specifically K=3 and K=4, the solutions, especially the optimized one (Solution 4), consistently converged to the expected number of centroids. With K=3, we consistently obtained three centroids, and with K=4, four centroids were reliably obtained.
- In contrast, for larger K values, such as K=5 and K=7, the results exhibited more variability. When K=5, the algorithm yielded three, four, or five centroids, and with K=7, we observed outcomes of two, three, or five centroids.
- The variability in these results can be attributed to the inherent characteristics of the dataset and the initial seed point selection. Depending on the initial conditions and data distribution, the algorithm may converge prematurely, leading to fewer centroids.
- The inclusion of a convergence threshold in the advanced solution, along with specified convergence criteria, allowed for early termination when the centroids stabilized. This enhancement contributed to more efficient and consistent results.
- In practical applications, it is imperative to choose the value of K judiciously, considering the dataset's nature and the desired number of clusters.

Centroids in a K-means clustering algorithm can converge quickly as it did here either in 2 or 3 iterations on most cases, due to several factors:

- **Initial Centroid Placement:** The choice of initial centroid positions can significantly influence convergence speed. If the initial centroids are

placed relatively close to their final positions, the algorithm is more likely to converge quickly. Various initialization techniques, such as random initialization or k-means++, aim to improve the selection of initial centroids.

- **Data Distribution:** The distribution of data points in the dataset can impact convergence. If the data points are tightly clustered and well-separated from other clusters, the algorithm is more likely to converge quickly. In such cases, the initial assignment of data points to clusters may be close to the final clustering.
- **Convergence Criteria:** K-means algorithms typically use a convergence criterion to determine when to stop iterating. The algorithm checks if the centroids have moved significantly between iterations. If the centroids stabilize, meaning that their positions change very little, the algorithm terminates. This early termination contributes to faster convergence.
- **Data Quality:** Clean and well-behaved data with minimal noise and outliers can lead to faster convergence. Outliers and noisy data points can introduce fluctuations in centroid positions, delaying convergence.
- **Algorithm Parameters:** The choice of algorithm parameters, such as the maximum number of iterations (R) or the convergence threshold, can affect how quickly the algorithm converges. Setting an appropriate value for these parameters can help achieve faster convergence without compromising the quality of the clustering.

It's important to note that the number of iterations needed for convergence can vary from one dataset to another, and there are no guarantees that centroids will always converge in a small number of iterations. In some cases, where data is more complex, less well-separated, or noisy, it may take more iterations for centroids to stabilize. We often experiment with different initialization methods and parameters to find the most efficient and reliable configuration for their specific datasets.

Through this, we have presented an in-depth evaluation of five K-means clustering solutions applied to the 4D Iris dataset. We conducted experiments with varying K and R values, and our findings underscore the substantial impact of the choice of K on the number of centroids obtained. The advanced solutions, featuring convergence thresholds and optimizations, demonstrated efficiency and predictability. Therefore, selecting the appropriate value of K is of paramount importance to achieve the desired clustering results, tailored to the dataset's characteristics and the intended cluster count.

Conclusion:

Team Member Contributions:

Anushka just selected the dataset for BYOD which we had to pre-process to work with our algorithm. Sreeram(myself) and Santosh collaborated and done the rest of the work completely between us side by side.

Summary:

In this project, several tasks were undertaken to explore different aspects of data processing and clustering. The project can be summarized as follows:

- The initial part involved rewriting the tasks from Project 1 using Apache Pig. Eight tasks were reimplemented using Pig, and the report discusses the strategies employed and the trade-offs between using Pig and standard Map-Reduce.

- Next part focused on implementing K-means clustering, a data mining algorithm that groups data objects into clusters. Several key steps were involved: Different K-means clustering strategies were developed using Java Map-Reduce jobs. These algorithms included single-iteration and multi-iteration K-means, advanced algorithms with convergence thresholds, and optimized solutions utilizing Hadoop Map Reduce features. Various output variations were designed to return cluster centers and clustered data points along with convergence information.

- And the final part allowed for creativity by choosing two alternative extensions to the project:

a. An adaptation to the K-means algorithm (K-medoids) was implemented, repeating the steps from Task 2a-e. The respective experiments were performed and documented in the report.

b. Finally, a relevant dataset (Iris) from an external source (Kaggle) was selected and used for running the K-means algorithm. The dataset was described in the report, and experiments similar to those above were carried out.

Yours Truly,

Sreeram Marimuthu

Nitya Phani Santosh Oruganty

Anushka Bangal

October 24th 2023