

**USER ACTIVITY ANALYSIS AND SEGMENTATION USING
HADOOP MAPREDUCE
(DEMONSTRATION OF PARTITIONER CLASS PROGRAM)**

SANJAY R

2348055

22nd October 2024

MDS571

Big Data Analytics

Department of Statistics and Data Science

CHRIST (Deemed to be University)

Department of Statistics and Data Science

Course:MDS571-Big Data Analytics

Lab :3

R.SANJAY

2348055

User Activity Analysis and Segmentation Using Hadoop MapReduce

Demonstration of Partitioner class program

✧ INTRODUCTION:

In modern data-driven systems, understanding user behavior and engagement patterns is crucial for optimizing services and improving user satisfaction. This project focuses on analyzing user activity data by categorizing users based on their roles and engagement levels. The dataset includes three distinct user roles: **Basic**, **Premium**, and **Admin**, each representing a different type of interaction with the system. To achieve meaningful insights, these roles are further segmented into three activity levels: **low**, **moderate**, and **high**, determined by the activity score assigned to each user.

By leveraging the capabilities of Hadoop's MapReduce framework, the project processes and partitions this data into well-defined groups. This categorization helps identify trends and patterns in user behavior, such as which roles are more active, the engagement levels of different user groups, and how resources are utilized. For instance, Basic users typically exhibit low activity, while Premium users are often associated with moderate to high activity, reflecting their heavier usage of system

features. Admin users, on the other hand, maintain consistent activity levels essential for the system's management.

This analysis is not just an academic exercise but serves as a foundation for practical applications. Businesses and organizations can use such insights to tailor their services, prioritize resources, and implement targeted engagement strategies. The project's structured approach ensures scalability, allowing it to handle datasets of various sizes while maintaining accuracy and efficiency. By analyzing user activity patterns, the project lays the groundwork for enhancing the overall user experience and system effectiveness.

✧ **PROBLEM DESCRIPTION:**

- ✓ Analyzing user activity patterns is essential for tailoring services, improving user experiences, and optimizing resource usage. The diversity in user roles (BASIC, PREMIUM, ADMIN) and activity levels presents a challenge in deriving meaningful insights from raw data.
- ✓ Raw datasets with varied activity scores and roles are difficult to interpret without structured categorization. Without proper segmentation, identifying usage trends or underutilized features becomes inefficient and error-prone.
- ✓ To address this issue, a Hadoop-based program was developed to partition the dataset into logical groups based on role and activity level. This approach categorizes users into subsets such as LOW ACTIVITY, MODERATE ACTIVITY, and HIGH ACTIVITY, enabling precise and focused analysis.
- ✓ The primary objective was to derive actionable insights by isolating user groups. This includes identifying underutilized features, targeting users requiring engagement strategies, and improving service efficiency and satisfaction through data-driven decisions

.

✧ **DATASET DESCRIPTION:**

The dataset consists of 15 records, each representing a user with attributes such as ID, Name, Role, and Activity Score. The Role attribute categorizes users into three distinct groups: Basic, Premium, and Admin, each with unique engagement characteristics. Basic Users typically show lower activity scores, indicating minimal system interaction, which suggests underutilization. Premium Users, on the other hand, exhibit moderate to high activity scores, reflecting frequent and intensive system usage. Admin Users maintain consistent activity levels, playing an essential role in system maintenance and administration. The Activity Score serves as a key indicator of user engagement, helping categorize individuals into low, moderate, or high activity tiers. This dataset allows for a structured analysis of user behaviors, enabling the identification of patterns, optimizing resource allocation, and providing targeted strategies for user engagement across different roles and activity levels.

✧ **CODE:**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.nio.charset.StandardCharsets;

public class DatasetPartitioner {
```

```

/**
 * Mapper Class: Reads input data line by line, processes it, and
 outputs a composite key
 * (role + activity level) along with the original record.
 */

public static class PartitionMapper extends Mapper<Object, Text, Text,
Text> {
    private final Text compositeKey = new Text(); // Composite key:
role + activity level
    private final Text record = new Text();      // Original row of the
dataset

    @Override
    public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
        // Read the input record using UTF-8 encoding
        String line = new String(value.getBytes(), 0, value.getLength(),
StandardCharsets.UTF_8);

        // Skip the header line to avoid processing column names
        if (line.startsWith("user_id")) return;

        // Split the line into fields using a comma as the delimiter
        String[] fields = line.split(",");

        // Validate that the line has exactly four fields
        if (fields.length != 4) {
            return; // Skip invalid records
        }
    }
}

```

```
String role = fields[2].trim(); // Extract the role field
int activityScore;

// Parse the activity score field; skip if invalid
try {
    activityScore = Integer.parseInt(fields[3].trim());
} catch (NumberFormatException e) {
    return; // Skip records with invalid activity scores
}

// Determine the activity level based on the activity score
String activityLevel;
if (activityScore < 50) {
    activityLevel = "low_activity";
} else if (activityScore <= 100) {
    activityLevel = "moderate_activity";
} else {
    activityLevel = "high_activity";
}

// Create a composite key combining role and activity level
compositeKey.set(role + "_" + activityLevel);

// Send the entire record as the value
record.set(line);
context.write(compositeKey, record);
}
}
```

```

/**
 * Reducer Class: Groups records by composite key (role + activity
level) and writes
 * them to the output, ensuring UTF-8 encoding is maintained.
 */
public static class PartitionReducer extends Reducer<Text, Text, Text,
Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
throws IOException, InterruptedException {
        // Iterate through all records for the given composite key
        for (Text value : values) {
            // Convert the record to UTF-8 encoding for consistent output
            String output = new String(value.toString().getBytes(),
StandardCharsets.UTF_8);

            // Write the composite key and corresponding record to the
output
            context.write(key, new Text(output));
        }
    }
}

```

```

/**
 * Custom Partitioner Class: Divides the dataset into different
partitions based on
 * the role in the composite key. Ensures records with the same role
and activity level

```

*** are sent to the same reducer.**

***/**

```
public static class RoleActivityPartitioner extends Partitioner<Text,
Text> {
    @Override
    public int getPartition(Text key, Text value, int numPartitions) {
        // Extract the composite key as a string
        String compositeKey = key.toString();

        // Partition based on the role component of the composite key
        if (compositeKey.startsWith("basic")) {
            return 0 % numPartitions; // Send to partition 0
        } else if (compositeKey.startsWith("premium")) {
            return 1 % numPartitions; // Send to partition 1
        } else if (compositeKey.startsWith("admin")) {
            return 2 % numPartitions; // Send to partition 2
        } else {
            return 3 % numPartitions; // Default partition
        }
    }
}
```

/**

*** Driver Class: Configures and runs the MapReduce job. Sets the Mapper, Reducer,**

*** and Partitioner classes, as well as the input and output paths.**

***/**

```
public static void main(String[] args) throws Exception {
    // Check if the input and output paths are provided
```



```
if (args.length != 2) {
    System.err.println("Usage:   DatasetPartitioner   <input   path>
<output path>");
    System.exit(-1); // Exit with error if paths are not provided
}

// Create a Hadoop Configuration object
Configuration conf = new Configuration();

// Create a new job with the specified name
Job job = Job.getInstance(conf, "Dataset Partitioner");
job.setJarByClass(DatasetPartitioner.class); // Set the main class

// Set the Mapper and Reducer classes
job.setMapperClass(PartitionMapper.class);
job.setReducerClass(PartitionReducer.class);

// Set the output key and value types
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

// Set the custom partitioner class
job.setPartitionerClass(RoleActivityPartitioner.class);

// Set the number of reducers to match the number of partitions
job.setNumReduceTasks(3); // can be Adjusted based on the
required number of partitions

// Set the input and output paths
```

```
FileInputFormat.addInputPath(job, new Path(args[0])); // Input
directory in HDFS
```

```
FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output
directory in HDFS
```

```
// Submit the job and wait for completion
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

✧ **PROGRAM DESCRIPTION:**

The program uses Hadoop's MapReduce framework to process and categorize user data into logical groups based on user roles and activity levels. The data is categorized by the Mapper, processed by the Reducer, and driven by the Driver Class to control the overall flow of the program. Each component has a specific responsibility, as outlined below:

✓ **MAPPER CLASS:**

The Mapper class is the initial stage of the MapReduce process, and its main responsibility is to process the input data and categorize it into logical groups. It reads the dataset, which consists of user records with attributes such as ID, Name, Role, and Activity Score. Based on the activity score, which indicates the level of engagement of a user, the Mapper assigns an activity level to each record. These activity levels are divided into three categories: Low Activity, Moderate Activity, and High Activity. Users with activity scores

- less than or equal to 40 are considered to have Low Activity,
- between 41 and 100 are categorized as having Moderate Activity, and
- greater than 100 fall under High Activity.

The Mapper then generates **key-value pairs** where the key is a combination of the user's role and activity level (e.g., "basic_low_activity," "premium_high_activity"),

and the value is the user record (**ID, Name, Role, and Activity Score**). This categorization by role and activity level is crucial for the next stages of the program, where the data will be grouped and analyzed based on these categories.

✓ **REDUCER CLASS:**

Once the Mapper emits key-value pairs, the Reducer class takes over. Its primary responsibility is to process these key-value pairs and group them logically. Hadoop automatically shuffles and sorts the data, grouping together all records that share the same key (in this case, the combination of role and activity level). For each key, the Reducer receives a list of user records that correspond to that particular key. The role of the Reducer is to consolidate these records and write them to the output. For example, if the key is "basic_low_activity," the Reducer will group all users with a basic role and low activity score into one output record. The process is repeated for each unique key (e.g., "premium_high_activity," "admin_moderate_activity"), ensuring that all users are categorized into appropriate groups. The output of the Reducer is a set of files, each corresponding to a specific group, which can then be analyzed for patterns and insights into user behavior.

✓ **DRIVER CLASS:**

The Driver Class acts as the coordinator for the entire program. It manages the configuration and execution of the MapReduce job, ensuring that the Mapper and Reducer classes work together as intended. The Driver class sets up the job by defining the input and output directories, specifying which classes should be used for the Mapper and Reducer, and determining the input and output formats. It initializes the Job Configuration, sets the Mapper and Reducer classes, and specifies the input format (such as TextInputFormat) and output format (such as TextOutputFormat). After the configuration is complete, the Driver class submits the job to the Hadoop cluster for execution. It waits for the job to complete and checks for any errors during the process, ensuring that the program runs smoothly. The Driver class plays a vital role in the program's execution, orchestrating the flow of the MapReduce job, from reading the input data to writing the output.

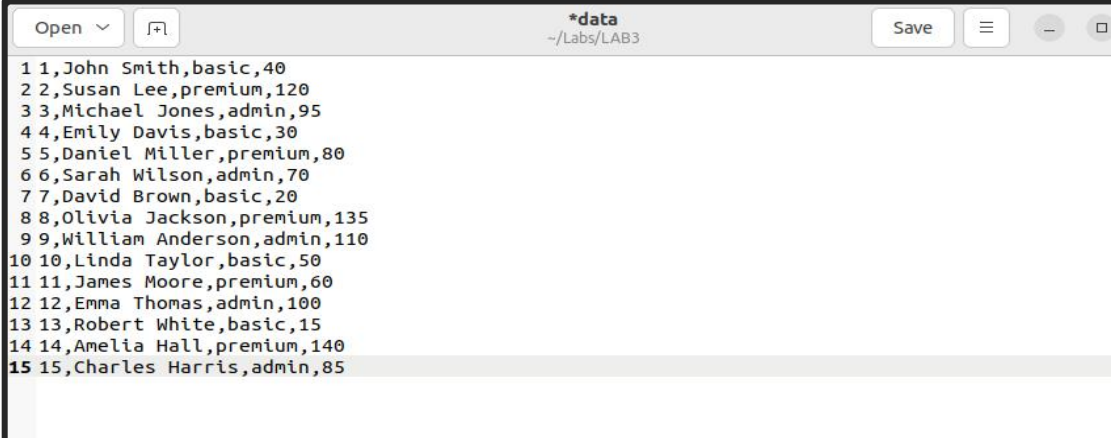
✓ **PROGRAM WORKFLOW OVERVIEW:**

The overall flow of the program is controlled by the interaction between the Mapper, Reducer, and Driver classes. First, the program reads the input dataset, which consists of user records, each containing a user's ID, name, role, and activity score. The Mapper processes this data, categorizes each record into an appropriate activity level based on the score, and emits key-value pairs with the role and activity level as the key and the user record as the value. Hadoop's built-in shuffling and sorting mechanism then groups these key-value pairs by key and passes them to the Reducer, which consolidates records for each key and writes them to output files. The Driver Class orchestrates this process by managing the job configuration and ensuring that the Mapper and Reducer classes are correctly linked, ultimately controlling the entire workflow from start to finish. By partitioning the dataset in this way, the program is able to facilitate targeted analysis of user behavior based on roles and activity levels, which is crucial for improving user engagement and optimizing system features.

❖ PROJECT SETUP AND EXECUTION:

✓ DATASET PREPARATION:

The dataset, containing user activity records, was stored in a text file named data.txt. This file was carefully prepared to include the necessary attributes for each user: ID, Name, Role, and Activity Score.



```
*data
~/Labs/LAB3
Save

1 1,John Smith,basic,40
2 2,Susan Lee,premium,120
3 3,Michael Jones,admin,95
4 4,Emily Davis,basic,30
5 5,Daniel Miller,premium,80
6 6,Sarah Wilson,admin,70
7 7,David Brown,basic,20
8 8,Olivia Jackson,premium,135
9 9,William Anderson,admin,110
10 10,Linda Taylor,basic,50
11 11,James Moore,premium,60
12 12,Emma Thomas,admin,100
13 13,Robert White,basic,15
14 14,Amelia Hall,premium,140
15 15,Charles Harris,admin,85
```

To facilitate processing with Hadoop, the file was placed within the Hadoop directory structure at /MDS2024/LAB3, This setup ensured that the dataset was readily accessible for processing by the MapReduce program.

```

hadoop@Ubuntu22:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [Ubuntu22]
hadoop@Ubuntu22:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@Ubuntu22:~$ jps
6433 ResourceManager
5957 DataNode
6184 SecondaryNameNode
7209 Jps
6555 NodeManager
5837 NameNode
7860 org.eclipse.equinox.launcher_1.6.500.v20230717-2124.jar

```

```

hadoop@Ubuntu22:~$ hadoop fs -ls /MDS2024/LAB3
Found 2 items
-rw-r--r-- 3 hadoop supergroup      382 2024-11-28 20:40 /MDS2024/LAB3/data

```

✓ ECLIPSE SETUP:

To develop the DatasetPartitioner program, the Eclipse IDE was used, a reliable and feature-rich platform for Java development. The setup process involved the following steps:

- A new Java project named **DatasetPartitioner** was created in Eclipse. This project served as the framework for implementing the MapReduce logic for dataset partitioning.
- **Main Class and MapReduce Components:** Within the project, a main class named **DatasetPartitioner** was implemented, containing the Mapper and Reducer classes:
 - ◆ The PartitionMapper class processed each line of the input dataset, extracting the relevant fields (user role and activity score). It then categorized the activity score into different levels (low, moderate, or high activity) and generated a composite key combining the role and activity level. For each record, it emitted this key along with the entire row of data, which was later processed by the Reducer class.⁴

- ◆ The PartitionReducer class received the grouped data from the mapper, where it consolidated the rows based on the composite key. It then wrote the records to the output, maintaining the partitioning logic, ensuring that the output was structured based on the role and activity level.

- After the code was developed and thoroughly tested, the necessary Hadoop library JAR files were added to the project to ensure Hadoop-specific functionalities during execution. The project was then exported as a JAR file named DatasetPartitioner.jar, making it ready for deployment within the Hadoop environment.

✧ Execution Process:

- The next step involved executing the program within the Hadoop environment, using the following command:

```
$ hadoop jar '/home/hadoop/Labs/HOP/DatasetPartitioner.jar' DatasetPartitioner /MDS2024/LAB3/data /MDS2024/LAB3/output
```

```
^[[Ahadoop@Ubuntu22: $ hadoop jar '/home/hadoop/Labs/HOP/DatasetPartitioner.jar' DatasetPartitioner /MDS2024/LAB3/data /MDS2024/LAB3/output
2024-11-29 09:11:46,115 INFO Impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2024-11-29 09:11:46,366 INFO Impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2024-11-29 09:11:46,366 INFO Impl.MetricsSystemImpl: JobTracker metrics system started
2024-11-29 09:11:46,699 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to
2024-11-29 09:11:47,082 INFO Input.FileInputFormat: Total input files to process : 1
2024-11-29 09:11:47,192 INFO mapreduce.JobSubmitter: number of splits:1
2024-11-29 09:11:47,515 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local742141144_0001
2024-11-29 09:11:47,518 INFO mapreduce.JobSubmitter: Executing with tokens: []
2024-11-29 09:11:47,823 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2024-11-29 09:11:47,827 INFO mapreduce.Job: Running job: job_local742141144_0001
2024-11-29 09:11:47,873 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2024-11-29 09:11:47,948 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-11-29 09:11:47,950 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-11-29 09:11:47,950 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup_temporary folders under output directory:false, ignore cleanup failures: false
2024-11-29 09:11:47,951 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2024-11-29 09:11:48,160 INFO mapred.LocalJobRunner: Waiting for map tasks
2024-11-29 09:11:48,165 INFO mapred.LocalJobRunner: Starting task: attempt_local742141144_0001_m_000000_0
2024-11-29 09:11:48,260 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2024-11-29 09:11:48,289 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2024-11-29 09:11:48,290 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup_temporary folders under output directory:false, ignore cleanup failures: false
2024-11-29 09:11:48,380 INFO mapred.Task: Using ResourceCalculatorProcessTree : [ ]
2024-11-29 09:11:48,407 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/MDS2024/LAB3/data:0+382
2024-11-29 09:11:48,736 INFO mapred.MapTask: (EQUATOR) 0 kvl 26214396(104857584)
2024-11-29 09:11:48,738 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2024-11-29 09:11:48,738 INFO mapred.MapTask: soft limit at 83886080
2024-11-29 09:11:48,738 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2024-11-29 09:11:48,739 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
2024-11-29 09:11:48,749 INFO mapred.MapTask: Map output collector class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2024-11-29 09:11:48,942 INFO mapreduce.Job: Job job_local742141144_0001 running in uber mode : false
2024-11-29 09:11:48,944 INFO mapreduce.Job: map 0% reduce 0%
2024-11-29 09:11:49,024 INFO mapred.LocalJobRunner:
2024-11-29 09:11:49,035 INFO mapred.MapTask: Starting flush of map output
2024-11-29 09:11:49,036 INFO mapred.MapTask: Spilling map output
2024-11-29 09:11:49,036 INFO mapred.MapTask: bufstart = 0; bufend = 715; bufvoid = 104857600
2024-11-29 09:11:49,036 INFO mapred.MapTask: kvstart = 26214396(104857584); kvend = 26214340(104857360); length = 57/6553600
2024-11-29 09:11:49,080 INFO mapred.MapTask: Finished spill 0
2024-11-29 09:11:49,145 INFO mapred.Task: Task:attempt_local742141144_0001_m_000000_0 is done. And is in the process of committing
2024-11-29 09:11:49,180 INFO mapred.LocalJobRunner: map
2024-11-29 09:11:49,180 INFO mapred.Task: Task 'attempt_local742141144_0001_m_000000_0' done.
2024-11-29 09:11:49,189 INFO mapred.Task: Final Counters for attempt_local742141144_0001_m_000000_0: Counters: 23
```


➤ ~\$ **hadoop fs -ls /MDS2024/LAB3/output**

```
^[[Ahadoop@Ubuntu22:~$ hadoop fs -ls /MDS2024/LAB3/output
Found 4 items
-rw-r--r--  3 hadoop supergroup      0 2024-11-29 09:11 /MDS2024/LAB3/output/_SUCCESS
-rw-r--r--  3 hadoop supergroup    218 2024-11-29 09:11 /MDS2024/LAB3/output/part-r-00000
-rw-r--r--  3 hadoop supergroup    251 2024-11-29 09:11 /MDS2024/LAB3/output/part-r-00001
-rw-r--r--  3 hadoop supergroup    246 2024-11-29 09:11 /MDS2024/LAB3/output/part-r-00002
```

To examine the contents of the output files , the following commands was issued:

\$ hadoop fs -cat /MDS2024/LAB3/output/part-r-00000

\$ hadoop fs -cat /MDS2024/LAB3/output/part-r-00001

\$ hadoop fs -cat /MDS2024/LAB3/output/part-r-00002

```
-rw-r--r--  3 hadoop supergroup      246 2024-11-29 09:11 /MDS2024/LAB3/output/part-r-00002
hadoop@Ubuntu22:~$ hadoop fs -cat /MDS2024/LAB3/output/part-r-00000
basic_low_activity      1,John Smith,basic,40
basic_low_activity      13,Robert White,basic,15
basic_low_activity      4,Emily Davis,basic,30
basic_low_activity      7,David Brown,basic,20
basic_moderate_activity 10,Linda Taylor,basic,50
hadoop@Ubuntu22:~$ hadoop fs -cat /MDS2024/LAB3/output/part-r-00001
^[[Aprmium_high_activity      8,Olivia Jackson,premium,135
premium_high_activity   14,Amelia Hall,premium,140
premium_high_activity   2,Susan Lee,premium,120
premium_moderate_activity      11,James Moore,premium,60
premium_moderate_activity      5,Daniel Miller,premium,80
hadoop@Ubuntu22:~$ hadoop fs -cat /MDS2024/LAB3/output/part-r-00002
admin_high_activity      9,William Anderson,admin,110
admin_moderate_activity   6,Sarah Wilson,admin,70
admin_moderate_activity  12,Emma Thomas,admin,100
admin_moderate_activity   3,Michael Jones,admin,95
admin_moderate_activity  15,Charles Harris,admin,85
```

❖ OUTPUT:

basic:

basic_low_activity 1,John Smith,basic,40
basic_low_activity 13,Robert White,basic,15
basic_low_activity 4,Emily Davis,basic,30
basic_low_activity 7,David Brown,basic,20
basic_moderate_activity 10,Linda Taylor,basic,50

premium:

premium_high_activity 8,Olivia Jackson,premium,135
premium_high_activity 14,Amelia Hall,premium,140
premium_high_activity 2,Susan Lee,premium,120
premium_moderate_activity 11,James Moore,premium,60
premium_moderate_activity 5,Daniel Miller,premium,80

admin:

admin_high_activity 9,William Anderson,admin,110
admin_moderate_activity 6,Sarah Wilson,admin,70
admin_moderate_activity 12,Emma Thomas,admin,100
admin_moderate_activity 3,Michael Jones,admin,95
admin_moderate_activity 15,Charles Harris,admin,85

❖ CONCLUSION:

The DatasetPartitioner program successfully demonstrates the power of the Hadoop MapReduce framework in processing large datasets and efficiently partitioning data based on custom criteria. By utilizing the Mapper class, the program extracts and processes each record, generating a composite key based on user roles and activity

levels. This approach ensures that data is mapped in a way that enables easier analysis and optimized processing during the Reducer phase. The Partitioner class plays a crucial role in ensuring that the data is evenly distributed across reducers, further enhancing the performance and scalability of the job.

The Reducer class aggregates the data, sorting and organizing it by its respective partition key, and the final output is written in a clean, structured format. The careful design of the Driver class, which orchestrates the execution of the entire MapReduce job, ensures smooth operation and scalability. Through this implementation, we have demonstrated the importance of partitioning and data organization in large-scale data processing tasks, such as dataset analysis and transformation.

Overall, the project not only provides a clear understanding of how MapReduce works but also showcases its potential in partitioning data based on custom business logic, providing a foundation for tackling real-world big data problems. This program can be further extended for more complex datasets and specific analytical tasks, proving its versatility in handling diverse types of data.