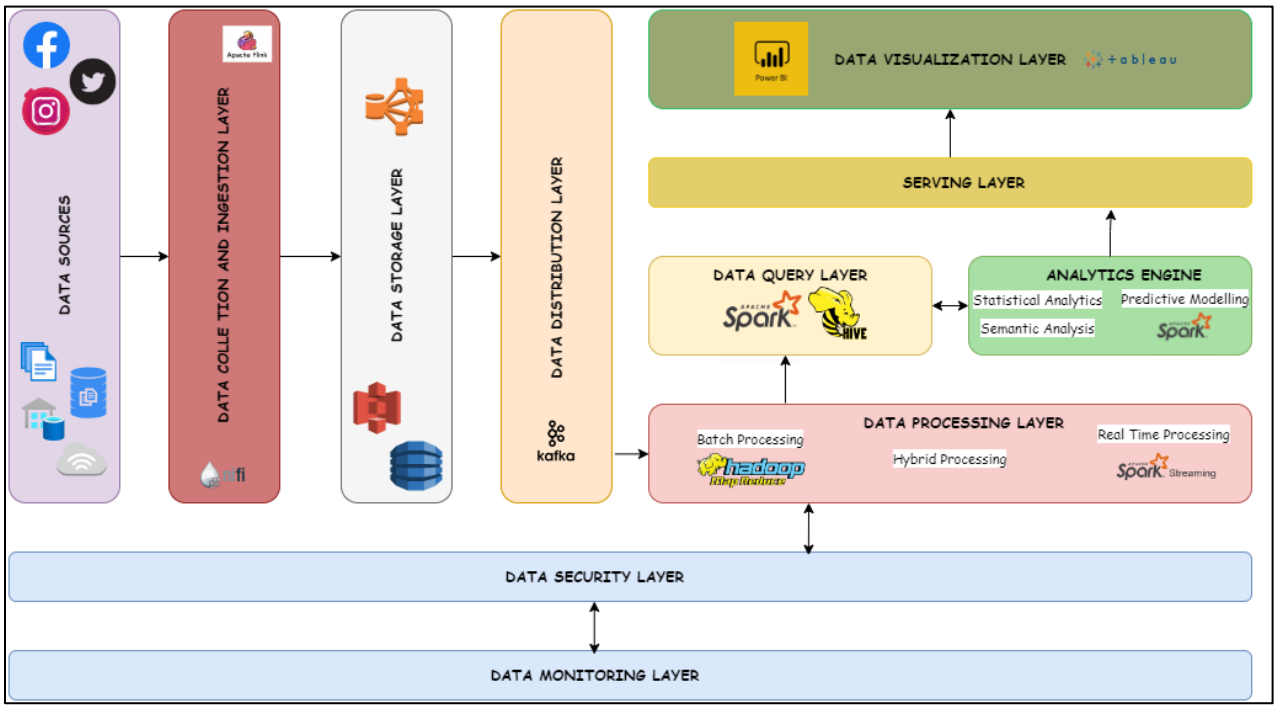
CMM705 BIG DATA PROGRAMMING

Sahdiya 2330842/20231624
MSc in Big Data Analytics
12/31/23

QUESTION 1

- **1.1**



- **1.2**

Lambda architecture and Kappa architecture are two widely recognized standard frameworks for managing both real-time and historical data. In the lambda architecture, data records are simultaneously processed in both real-time and batch modes. On the other hand, the Kappa architecture doesn't designate a specific component for batch processing; instead, a single system handles both real-time and historical data processing. The choice between these architectures depends entirely on the desired analytics output.

In the provided diagram, a system is proposed to collect real-time data on movies from various platforms, storing it in scalable storage, and generating summarized results for real-time analytics through a chosen data visualization method. The architecture comprises multiple components, each responsible for specific functionalities:

1. **Data Sources**: Diverse movie data sources, varying in volume, velocity, and variety, are included in the big data set for analysis. These sources, mainly the social media or data feeds related to movies, need to be subscribed and connected to the data ingestion layer.
2. **Data Ingestion Layer**: Responsible for separating noise from relevant information, this layer cleanses, validates, transforms, and incorporates data into the big data technology for subsequent processing.

3. **Distributed Storage Layer**: The ingested data is stored in a distributed storage layer, which can take the form of key-value stores, column-oriented stores, document stores, or graph stores. S3 and HDFS are highlighted as prominent storage mechanisms due to their durability, scalability, and security.
4. **Data Processing Layer**: Comprising batch processing, real-time processing, and hybrid processing, this layer generates batch views and indexes using technologies like Hadoop and Hive. The hybrid processing combines batch and real-time processing to create views enriched with both historical and up-to-date data.
5. **Data Query Layer/Analytics Engine**: Utilizing big data and traditional business intelligence methods, this layer runs data queries on top of the data in the processing layer. Hive is commonly used as a querying platform for its scalability and fast results generation.
6. **Data Serving Layer**: Merges real-time and batch analytics summarized output data based on indexes, facilitating faster results merging with separate layers for batch and real-time analytics.
7. **Visualization Layer**: Refreshes visualizations periodically to prevent information overload, incorporating aggregated data and real-time views. Tableau, PowerBI, and Quicksight are preferred tools for presenting processed data effectively.
8. **Data Monitoring Layer**: Essential for obtaining a comprehensive overview of the big data tech stack, ensuring SLA availability with minimal downtime. Performance is a key parameter in this monitoring process.
9. **Data Security Layer**: Addresses security concerns in the distributed architecture by implementing measures such as node authentication, file-layer encryption, trusted keys, certificates, and tools like Chef or Puppet for validation during deployments. Ensures secure communication between nodes.

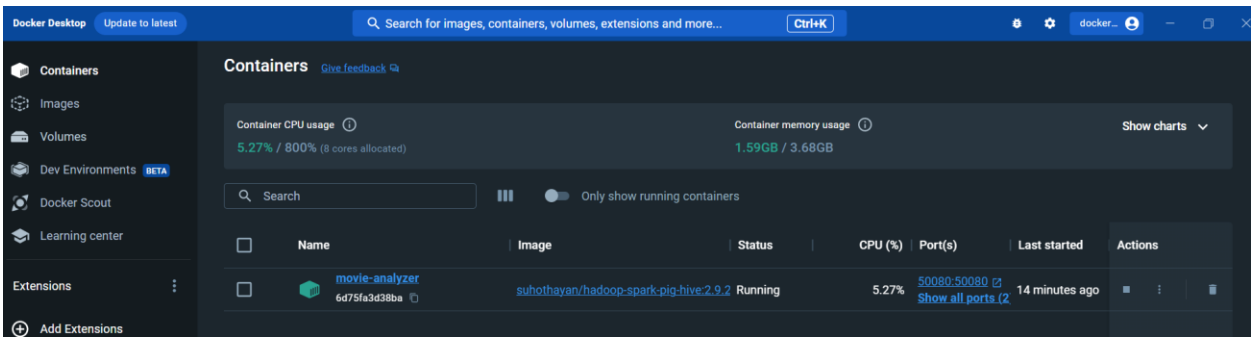
QUESTION 2

As the initial step a docker container is created by mounting the data set available in the local machine. It uses the image ‘suhothayan/hadoop-spark-pig-hive:2.9.2’

- *docker run -it -p 8081:8081 -p 50080:50080 --name movie-analyzer -v C:\Users\Shahd\OneDrive\Desktop\Movie:/resource -d suhothayan/hadoop-spark-pig-hive:2.9.2*

```
PS C:\Users\Shahd> docker run -it -p 8081:8081 -p 50080:50080 --name movie-analyzer -v C:\Users\Shahd\OneDrive\Desktop\M
ovie:/resource -d suhothayan/hadoop-spark-pig-hive:2.9.2
6d75fa3d38ba38e25cc8ce1bd90460f9cb4ecf09aa581e1d33d636b0d8160898
PS C:\Users\Shahd> docker ps
CONTAINER ID   IMAGE                                   COMMAND                  CREATED             STATUS          PO
RTS                                                                                                                
                                                                                                   NAMES
6d75fa3d38ba   suhothayan/hadoop-spark-pig-hive:2.9.2   "/etc/bootstrap.sh -d"   About a minute ago   Up 24 seconds   80
30-8033/tcp, 8040/tcp, 8042/tcp, 8080/tcp, 8088/tcp, 9000/tcp, 13562/tcp, 40661/tcp, 50010/tcp, 50020/tcp, 50070/tcp, 0.
0.0.0:8081->8081/tcp, 50075/tcp, 50090/tcp, 0.0.0.0:50080->50080/tcp   movie-analyzer
```

Created container as viewed from docker desktop:



Get into the docker container to view the mounted resource

- *docker exec -it movie-analyzer bash*

```
PS C:\Users\Shahd> docker exec -it movie-analyzer bash
root@6d75fa3d38ba:/# ls
bin   derby.log  etc   lib    media         mnt  proc      root  sbin  sys  usr
boot  dev        home  lib64  metastore_db  opt  resource  run   srv   tmp  var
root@6d75fa3d38ba:/# cd resource/
root@6d75fa3d38ba:/resource# ls
MoviesTopRated.csv
```

2.1

In order to perform the task, the data set should be added to the HDFS Folder. Initially create a directory in HDFS and then move the file to that folder

- *hdfs dfs -mkdir moviedata*
- *hdfs dfs -put MoviesTopRated.csv moviedata/MoviesTopRated.csv*

- *hdfs dfs -ls moviedata /*

```
root@6d75fa3d38ba:/resource# hdfs dfs -mkdir moviedata
root@6d75fa3d38ba:/resource# hdfs dfs -ls
Found 2 items
drwxr-xr-x   - root supergroup          0 2019-07-21 16:09 input
drwxr-xr-x   - root supergroup          0 2023-12-27 03:34 moviedata
root@6d75fa3d38ba:/resource# hdfs dfs -put MoviesTopRated.csv moviedata/MoviesTopRated.csv
root@6d75fa3d38ba:/resource# hdfs dfs -ls moviedata
Found 1 items
-rw-r--r--   1 root supergroup    3434842 2023-12-27 03:35 moviedata/MoviesTopRated.csv
root@6d75fa3d38ba:/resource#
```

- **2.1.1**

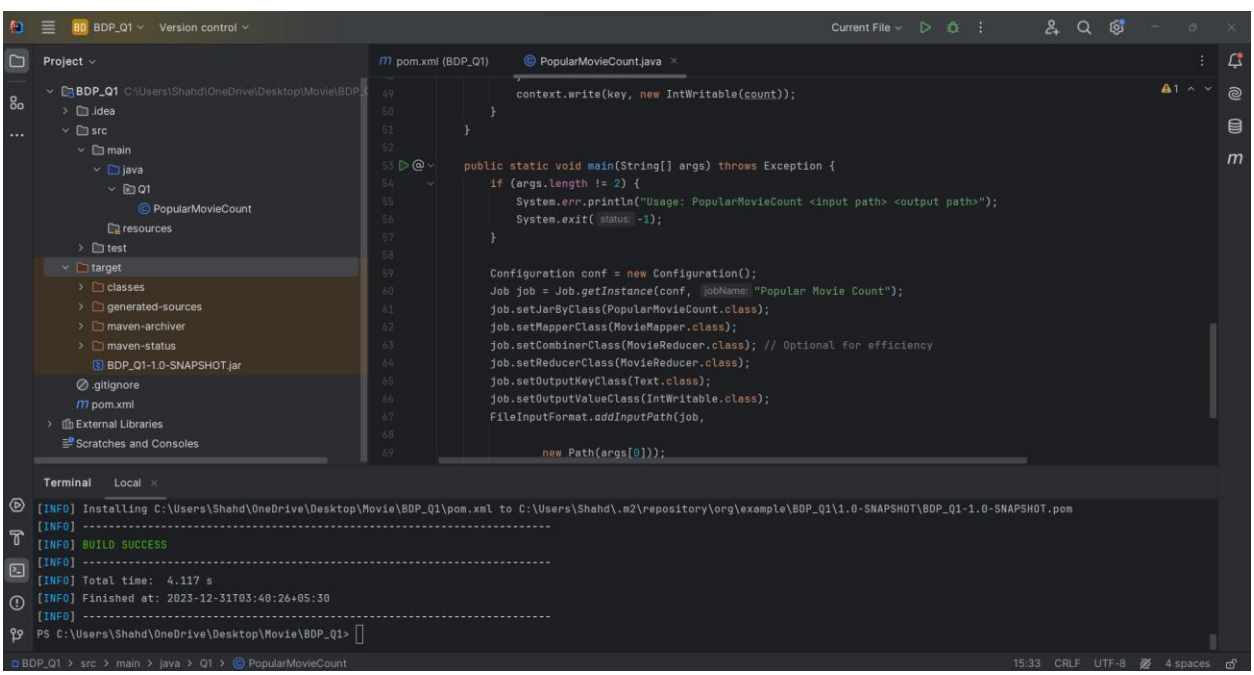
PopularMovieCount.java class is created with mapper, reducer and main function to count the movies based on the condition

```java
1 usage
public static class MovieMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] movieData = value.toString().split( regex: ",");

        // Skip header and invalid records
        if (!movieData[0].equals("id") && !movieData[1].equals("#VALUE!") && !movieData[4].equals("#VALUE!") && !mov
            try {
                double popularity = Double.parseDouble(movieData[1]);
                double voteAverage = Double.parseDouble(movieData[4]);
                double voteCount = Double.parseDouble(movieData[5]);

                if (popularity > 500.0 && voteAverage > 8.0 && voteCount > 10000.0) {
                    context.write(new Text( string: "popular_movies"), new IntWritable( value: 1));
                }
            } catch (NumberFormatException e) {
                // Handle invalid data (e.g., log a warning)
                System.err.println("Invalid numerical data encountered in record: " + value.toString());
            }
        }
    }
}
```

```java
2 usages
public static class MovieReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, Interrupt
        int count = 0;
        for (IntWritable value : values) {
            count += value.get();
        }
        context.write(key, new IntWritable(count));
    }
}
```

```java
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: PopularMovieCount <input path> <output path>");
            System.exit( status: -1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, jobName: "Popular Movie Count");
        job.setJarByClass(PopularMovieCount.class);
        job.setMapperClass(MovieMapper.class);
        job.setCombinerClass(MovieReducer.class); // Optional for efficiency
        job.setReducerClass(MovieReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job,

                new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
    }
}
```

The project is then built to generate the jar file in the target folder



Since the jar file is created, the task is executed using the YARN command

- *yarn jar BDP_Q1/target/BDP_Q1-1.0-SNAPSHOT.jar Q1.PopularMovieCount moviedata/Movies.csv output/Q02_01_outputs*

```
root@1e82f378c65e:/resource# yarn jar BDP_Q1/target/BDP_Q1-1.0-SNAPSHOT.jar Q1.PopularMovieCount moviedata/Movies.csv output/Q02_01_outputs
23/12/30 22:22:36 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/12/30 22:22:37 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
23/12/30 22:22:37 INFO input.FileInputFormat: Total input files to process : 1
23/12/30 22:22:39 INFO mapreduce.JobSubmitter: number of splits:1
23/12/30 22:22:39 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publ
isher.enabled
23/12/30 22:22:39 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1703944994811_0014
23/12/30 22:22:40 INFO impl.YarnClientImpl: Submitted application application_1703944994811_0014
```

The output will be stored in the specified folder 'output/Q02_01_outputs'. It can be viewed as below

- *hdfs dfs -ls output/Q02_01_outputs*
- *hdfs dfs -cat output/Q02_01_outputs/part-r-00000*

```
root@1e82f378c65e:/resource# hdfs dfs -ls output/Q02_01_outputs
Found 2 items
-rw-r--r--   1 root supergroup          0 2023-12-30 22:24 output/Q02_01_outputs/_SUCCESS
-rw-r--r--   1 root supergroup          0 2023-12-30 22:24 output/Q02_01_outputs/part-r-00000
root@1e82f378c65e:/resource# hdfs dfs -cat output/Q02_01_outputs/part-r-00000
root@1e82f378c65e:/resource#
```

There are no outputs displayed since there are no movies satisfying all three conditions

Therefore, the map reduce task will be run for each condition

<u>Map Reduce Task for popularity>500</u>

The Class is modified as below and built to generate the jar file

```
public static class MovieMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] movieData = value.toString().split( regex: ",");

        // Skip header and invalid records
        if (!movieData[0].equals("id") && !movieData[1].equals("#VALUE!") && !movieData[4].equals("#VALUE!") && !mov
            try {
                double popularity = Double.parseDouble(movieData[1]);

                if (popularity > 500.0) {
                    context.write(new Text( string: "popular_movies"), new Text( string: "1")); // Emit "1" as a Text val
                }
            } catch (NumberFormatException e) {
                // Handle invalid data (e.g., log a warning)
                System.err.println("Invalid numerical data encountered in record: " + value.toString());
            }
        }
    }
}
```

```
2 usages
public static class MovieReducer extends Reducer<Text, IntWritable, Text, Text> {

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, Interrupte
        {
            int count = 0;
            for (IntWritable value : values) {
                count += value.get();
            }
            context.write(new Text( string: "Movies with popularity greater than 500:"), new Text(String.valueOf(cou
        }
    }
}
```

```
public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: PopularMovies <input path> <output path>");
        System.exit( status: -1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf,  jobName: "Popular Movie Count");
    job.setJarByClass(PopularMovies.class);
    job.setMapperClass(MovieMapper.class);
    job.setCombinerClass(MovieReducer.class); // Optional for efficiency
    job.setReducerClass(MovieReducer.class);

    // Set output key and value classes to Text
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job,

            new Path(args[1]));
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
}
```

Yarn execution and output

- *Yarn jar BDP_Q1/target/BDP_Q1-1.0-SNAPSHOT.jar popularity.PopularMovies moviedata/Movies.csv output/Q02_01_popularity*

```
root@1e82f378c65e:/resource# yarn jar BDP_Q1/target/BDP_Q1-1.0-SNAPSHOT.jar popularity.PopularMovies moviedata/Movies.csv output/Q02_01_popularity
23/12/30 22:51:11 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/12/30 22:51:11 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
23/12/30 22:51:12 INFO input.FileInputFormat: Total input files to process : 1
23/12/30 22:51:13 INFO mapreduce.JobSubmitter: number of splits:1
23/12/30 22:51:14 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publ
isher.enabled
23/12/30 22:51:14 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1703944994811_0017
23/12/30 22:51:14 INFO impl.YarnClientImpl: Submitted application application_1703944994811_0017
23/12/30 22:51:15 INFO mapreduce.Job: The url to track the job: http://1e82f378c65e:8088/proxy/application_1703944994811_0017/
23/12/30 22:51:15 INFO mapreduce.Job: Running job: job_1703944994811_0017
23/12/30 22:51:25 INFO mapreduce.Job: Job job_1703944994811_0017 running in uber mode : false
23/12/30 22:51:25 INFO mapreduce.Job:  map 0% reduce 0%
23/12/30 22:51:32 INFO mapreduce.Job:  map 100% reduce 0%
23/12/30 22:51:48 INFO mapreduce.Job:  map 100% reduce 100%
23/12/30 22:51:51 INFO mapreduce.Job: Job job_1703944994811_0017 completed successfully
23/12/30 22:51:52 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=443
                FILE: Number of bytes written=398641
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
```

```
        File Output Format Counters
                Bytes Written=44
root@1e82f378c65e:/resource# hdfs dfs -cat output/Q02_01_popularity/part-r-00000
Movies with popularity greater than 500:        23
root@1e82f378c65e:/resource#
```

Map Reduce Task for Vote Average > 8

Modified Map Reduce Task and Yarn Executions are as below

```java
public static class AverageMovieMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] movieData = value.toString().split( regex: ",");

        // Skip header and invalid records (adjust indices if needed)
        if (!movieData[0].equals("id") && !movieData[2].equals("#VALUE!")) {
            try {
                double voteAverage = Double.parseDouble(movieData[2]); // Assuming voteAverage is in the 3rd column

                if (voteAverage > 8.0) {
                    context.write(new Text( string: "popular_movies"), new Text( string: "1"));
                }
            } catch (NumberFormatException e) {
                // Handle invalid data (e.g., log a warning)
                System.err.println("Invalid numerical data encountered in record: " + value.toString());
            }
        }
    }
}
```

```java
    public static class MovieReducer extends Reducer<Text, Text, Text, Text> {

        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            int count = 0;
            for (Text value : values) {
                count += Integer.parseInt(value.toString());
            }
            context.write(new Text( string: "Movies with vote average a> 8 :"), new Text(String.valueOf(count)));
        }
    }

    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: AveragePopularity <input path> <output path>");
            System.exit( status: -1);
        }

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, jobName: "Average Popularity Count");
        job.setJarByClass(AveragePopularity.class);
        job.setMapperClass(AverageMovieMapper.class);
        // job.setCombinerClass(MovieReducer.class); // Commented out as before
        job.setReducerClass(MovieReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
    }
```

```
root@1e82f378c65e:/resource# yarn jar BDP_Q1/target/BDP_Q1-1.0-SNAPSHOT.jar average.AveragePopularity moviedata/Movies.csv output/Q02_01_averageVote
23/12/30 22:59:52 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/12/30 22:59:52 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
23/12/30 22:59:53 INFO input.FileInputFormat: Total input files to process : 1
23/12/30 22:59:55 INFO mapreduce.JobSubmitter: number of splits:1
23/12/30 22:59:55 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publ
isher.enabled
23/12/30 22:59:56 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1703944994811_0018
23/12/30 22:59:56 INFO impl.YarnClientImpl: Submitted application application_1703944994811_0018
23/12/30 22:59:56 INFO mapreduce.Job: The url to track the job: http://1e82f378c65e:8088/proxy/application_1703944994811_0018/
23/12/30 22:59:56 INFO mapreduce.Job: Running job: job_1703944994811_0018
23/12/30 23:00:10 INFO mapreduce.Job: Job job_1703944994811_0018 running in uber mode : false
23/12/30 23:00:10 INFO mapreduce.Job:  map 0% reduce 0%
23/12/30 23:00:19 INFO mapreduce.Job:  map 100% reduce 0%
23/12/30 23:00:29 INFO mapreduce.Job:  map 100% reduce 100%
23/12/30 23:00:34 INFO mapreduce.Job: Job job_1703944994811_0018 completed successfully
23/12/30 23:00:34 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=6
                FILE: Number of bytes written=397797
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
```

```
root@1e82f378c65e:/resource# hdfs dfs -cat output/Q02_avg_vote/
Movies with vote average a> 8 : 263
```

Map Reduce Task for Vote Count > 10000

Mapper, reducer and yarn executions are as follows

```java
public static class CountMovieMapper extends Mapper<LongWritable, Text, Text, Text> {

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] movieData = value.toString().split(regex: ",");

        // Skip header row and invalid records (adjust indices if needed)
        if (!movieData[0].equals("id") && !movieData[2].equals("#VALUE!") && key.get() > 0) {
            try {
                int voteCount = Integer.parseInt(movieData[3]); // Assuming voteCount is in the 4th column

                if (voteCount > 10000) {
                    context.write(new Text(string: "popular_movies"), new Text(string: "1"));
                }
            } catch (NumberFormatException e) {
                // Handle invalid data (e.g., log a warning)
                System.err.println("Invalid numerical data encountered in record: " + value.toString());
            }
        }
    }
}

1 usage
```

```java
public static class MovieReducer extends Reducer<Text, Text, Text, Text> {

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        int count = 0;
        for (Text value : values) {
            count += Integer.parseInt(value.toString());
        }
        context.write(new Text(string: "Movies with vote count > 10000:"), new Text(String.valueOf(count)));
    }
}

public static void main(String[] args) throws Exception {
    if (args.length != 2) {
        System.err.println("Usage: CountPopularity <input path> <output path>");
        System.exit(status: -1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, jobName: "Count Popularity");
    job.setJarByClass(CountPopularity.class);
    job.setMapperClass(CountMovieMapper.class);
    // job.setCombinerClass(MovieReducer.class); // Optional combiner
    job.setReducerClass(MovieReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(verbose: true) ? 0 : 1);
}
```

```
root@1e82f378c65e:/resource# yarn jar BDP_Q1/target/BDP_Q1-1.0-SNAPSHOT.jar count.CountPopularity moviedata/Movies.csv output/Q2_count
23/12/31 00:51:06 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/12/31 00:51:07 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
23/12/31 00:51:08 INFO input.FileInputFormat: Total input files to process : 1
23/12/31 00:51:09 INFO mapreduce.JobSubmitter: number of splits:1
23/12/31 00:51:09 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publ
isher.enabled
23/12/31 00:51:10 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1703944994811_0029
23/12/31 00:51:10 INFO impl.YarnClientImpl: Submitted application application_1703944994811_0029
23/12/31 00:51:10 INFO mapreduce.Job: The url to track the job: http://1e82f378c65e:8088/proxy/application_1703944994811_0029/
23/12/31 00:51:10 INFO mapreduce.Job: Running job: job_1703944994811_0029
23/12/31 00:51:21 INFO mapreduce.Job: Job job_1703944994811_0029 running in uber mode : false
23/12/31 00:51:21 INFO mapreduce.Job:  map 0% reduce 0%
23/12/31 00:51:30 INFO mapreduce.Job:  map 100% reduce 0%
23/12/31 00:51:49 INFO mapreduce.Job:  map 100% reduce 100%
23/12/31 00:51:54 INFO mapreduce.Job: Job job_1703944994811_0029 completed successfully
23/12/31 00:51:54 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=44
                FILE: Number of bytes written=398197
```

```
root@1e82f378c65e:/resource# hdfs dfs -cat output/Q2_count/part-r-00000
Movies with vote count > 10000: 266
```

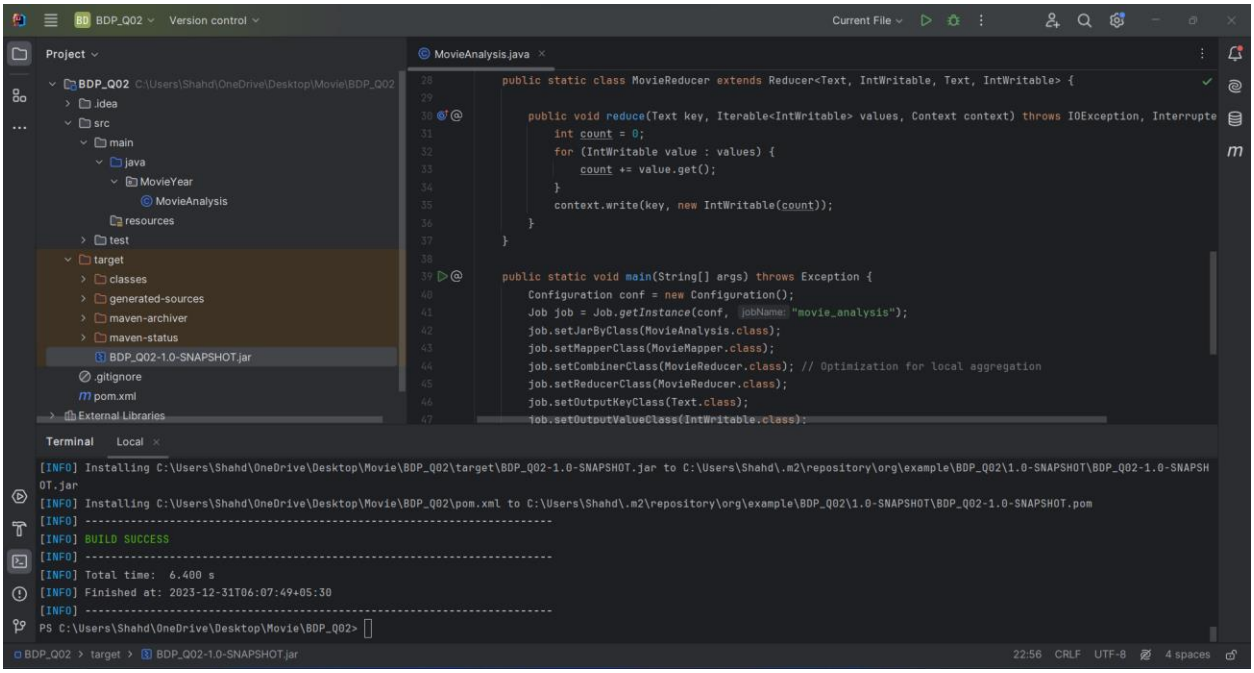
- **2.1.2**

Movie Analysis Java class is created with mapper, reducer and main function to count the movies based on the condition.

```java
public class MovieAnalysis {

    1 usage
    public static class MovieMapper extends Mapper<Object, Text, Text, IntWritable> {

        1 usage
        private static final IntWritable ONE = new IntWritable( value: 1);

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] fields = value.toString().split( regex: ","); // Assuming tab-separated data
            String year = fields[5]; // Extract year from the sixth column
            context.write(new Text(year), ONE);
        }
    }

    2 usages
    public static class MovieReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            int count = 0;
            for (IntWritable value : values) {
                count += value.get();
            }
            context.write(key, new IntWritable(count));
        }
    }
```

```java
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, jobName: "movie_analysis");
        job.setJarByClass(MovieAnalysis.class);
        job.setMapperClass(MovieMapper.class);
        job.setCombinerClass(MovieReducer.class); // Optimization for local aggregation
        job.setReducerClass(MovieReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
    }
}
```

The project is built to generate the jar file to be used for execution



Since the jar file is created the task is executed using the yarn command.

- *yarn jar BDP_Q02/target/BDP_Q02-1.0-SNAPSHOT.jar MovieYear.MovieAnalysis moviedata/Movies.csv output/Q02_mv*

```
root@1e82f378c65e:/resource# yarn jar BDP_Q02/target/BDP_Q02-1.0-SNAPSHOT.jar MovieYear.MovieAnalysis moviedata/Movies.csv output/Q02_mv
23/12/31 00:38:10 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/12/31 00:38:12 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your applic
ation with ToolRunner to remedy this.
23/12/31 00:38:14 INFO input.FileInputFormat: Total input files to process : 1
23/12/31 00:38:16 INFO mapreduce.JobSubmitter: number of splits:1
23/12/31 00:38:16 INFO Configuration.deprecation: yarn.resourcemanager.system-metrics-publisher.enabled is deprecated. Instead, use yarn.system-metrics-publ
isher.enabled
23/12/31 00:38:16 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1703944994811_0028
23/12/31 00:38:17 INFO impl.YarnClientImpl: Submitted application application_1703944994811_0028
23/12/31 00:38:17 INFO mapreduce.Job: The url to track the job: http://1e82f378c65e:8088/proxy/application_1703944994811_0028/
23/12/31 00:38:17 INFO mapreduce.Job: Running job: job_1703944994811_0028
23/12/31 00:38:36 INFO mapreduce.Job: Job job_1703944994811_0028 running in uber mode : false
23/12/31 00:38:36 INFO mapreduce.Job:  map 0% reduce 0%
23/12/31 00:38:52 INFO mapreduce.Job:  map 100% reduce 0%
23/12/31 00:39:12 INFO mapreduce.Job:  map 100% reduce 100%
23/12/31 00:39:17 INFO mapreduce.Job: Job job_1703944994811_0028 completed successfully
23/12/31 00:39:17 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=1252
                FILE: Number of bytes written=400615
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
```

The output will be stored in the specified folder ‘output/Q02_01_outputs’. It can be viewed as below

- *hdfs dfs -ls output/Q02_mv*
- *hdfs dfs -cat output/Q02_mv /part-r-00000*

```
root@1e82f378c65e:/resource# hdfs dfs -cat output/Q02_mv/part-r-00000
#VALUE! 1
1902    1
1903    1
1911    1
1915    1
1916    1
1917    1
1918    1
1920    2
1921    2
1922    1
1923    2
1924    2
1925    4
1926    2
1927    4
1928    5
1929    3
1930    4
1931    7
1932    10
1933    7
1934    5
1935    6
1936    3
1937    5
1938    7
1939    9
1940    10
1941    10
1942    8
1943    6
1944    9
1945    13
1946    17
```

2.2

Before proceeding with hive commands, following directories are created. These directories are used to store input data and outputs

- *hdfs dfs -mkdir /user/movie_data*
- *hdfs dfs -mkdir /user/hive*
- *hdfs dfs -mkdir /user/hive/movie_data*

```
root@1e82f378c65e:/# hdfs dfs -mkdir user
root@1e82f378c65e:/# hdfs dfs -mkdir /user/movie_data
root@1e82f378c65e:/# hdfs dfs -mkdir /user/hive
mkdir: `/user/hive': File exists
root@1e82f378c65e:/# hdfs dfs -mkdir /user/hive/movie_data
root@1e82f378c65e:/# hdfs dfs -ls /user
Found 3 items
drwxr-xr-x   - root supergroup          0 2023-12-30 14:54 /user/hive
drwxr-xr-x   - root supergroup          0 2023-12-30 14:53 /user/movie_data
drwxr-xr-x   - root supergroup          0 2023-12-30 14:53 /user/root
```

The dataset is now moved to the location '/user/hive/movie_data'

- *hdfs dfs -put /resource/MoviesTopRated.csv /user/movie_data/MoviesTopRated.csv*

```
root@1e82f378c65e:/# hdfs dfs -put /resource/MoviesTopRated.csv /user/movie_data/MoviesTopRated.csv
root@1e82f378c65e:/# hdfs dfs -ls /user/movie_data
Found 1 items
-rw-r--r--   1 root supergroup    3434842 2023-12-30 14:54 /user/movie_data/MoviesTopRated.csv
root@1e82f378c65e:/#
```

A new database and table is created to hold the dataset

- *create database movies_db;*

  show databases;

  use movies_db;

  CREATE EXTERNAL TABLE movies_table (serial_number INT, id INT, genre_ids STRING, title STRING,overview STRING, popularity DOUBLE, release_date STRING, vote_average DOUBLE, vote_count INT)

  ROW FORMAT DELIMITED

  FIELDS TERMINATED BY ','

  LOCATION '/user/hive/movie_data/';

```
hive> create database movies_db;
OK
Time taken: 0.838 seconds
hive> show databases;
OK
default
movies_db
Time taken: 0.234 seconds, Fetched: 2 row(s)
hive> use movies_db;
OK
Time taken: 0.071 seconds
```

```
hive> CREATE EXTERNAL TABLE movies_table (
    >   serial_number INT,
    >   id INT,
    >   genre_ids STRING,
    >   title STRING,
    >   overview STRING,
    >   popularity DOUBLE,
    >   release_date STRING,
    >   vote_average DOUBLE,
    >   vote_count INT
    > )
    > ROW FORMAT DELIMITED
    > FIELDS TERMINATED BY ','
    > LOCATION '/user/hive/movie_data/';
OK
Time taken: 0.572 seconds
```

Dataset is loaded to the table specific folder where it is stored

- *LOAD DATA INPATH 'hdfs:/user/movie_data/MoviesTopRated.csv' OVERWRITE INTO TABLE movies_table;*
- *ALTER TABLE movies_table SET TBLPROPERTIES ("skip.header.line.count"="1");*

```
hive> LOAD DATA INPATH 'hdfs:/user/movie_data/MoviesTopRated.csv' OVERWRITE INTO TABLE movies_table;
Loading data to table movies_db.movies_table
OK
Time taken: 59.398 seconds
hive> ALTER TABLE movies_table SET TBLPROPERTIES ("skip.header.line.count"="1");
OK
Time taken: 0.119 seconds
hive> SELECT * FROM movies_table LIMIT 5;
OK
0       238     "[18     80]"   The Godfather   NULL     a chronicle of the fictional Italian-American Corleone crime family. When organized crime family pa
triarch NULL    NULL
1       278     "[18     80]"   The Shawshank Redemption        NULL     upstanding banker Andy Dufresne begins a new life at the Shawshank prison      NULL
NULL
2       240     "[18     80]"   The Godfather Part II   NULL     a young Vito Corleone grows up in Sicily and in 1910s New York. In the 1950s   NULL    NULL
3       424     "[18     36      10752]"        NULL    The true story of how businessman Oskar Schindler saved over a thousand Jewish lives from the Nazis
while they worked as slaves in his factory during World War II. 48.096  NULL
4       19404   "[35     18      10749]"        NULL    "Raj is a rich  NULL    NULL
Time taken: 11.733 seconds, Fetched: 5 row(s)
```

- **2.2.1**

A CTE is defined using WITH clause and stores the details required for the question:

- Extracts year using YEAR() function from ‘release_date’ and assigns it as ‘release_year’
- Retrieves the movie title and vote count and assigns ranking using ROW_NUMBER().
- Data is partitioned by ‘release_year’ and ordered in descending order

Next, the ‘release_year’, ‘title’, and ‘vote_count’ columns are selected from the ‘ranked_movies’ CTE, filtering the results to include only instances where the ranking is equal to 1 which gives the top movies.

Query and Output are as follows:

- *WITH ranked_movies AS (*

  SELECT

  year(release_date) AS release_year,

  title,

  vote_count,

  ROW_NUMBER() OVER (PARTITION BY year(release_date) ORDER BY vote_count DESC) AS ranking

  FROM movies_table

)

  SELECT release_year, title, vote_count

  FROM ranked_movies

  WHERE ranking = 1;

```
hive> WITH ranked_movies AS (
    >   SELECT
    >     year(release_date) AS release_year,
    >     title,
    >     vote_count,
    >     ROW_NUMBER() OVER (PARTITION BY year(release_date) ORDER BY vote_count DESC) AS ranking
    >   FROM movies_table
    > )
    > SELECT release_year, title, vote_count
    > FROM ranked_movies
    > WHERE ranking = 1;
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e.
r using Hive 1.X releases.
Query ID = root_20231230153148_a8b7ab18-d0ac-412f-bb22-0a9a974c699f
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1703944994811_0002, Tracking URL = http://1e82f378c65e:8088/proxy/application_1703944994811_0002/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1703944994811_0002
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-12-30 15:31:58,603 Stage-1 map = 0%,  reduce = 0%
2023-12-30 15:32:59,236 Stage-1 map = 0%,  reduce = 0%
2023-12-30 15:33:15,194 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 27.1 sec
2023-12-30 15:33:38,281 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 34.29 sec
MapReduce Total cumulative CPU time: 34 seconds 290 msec
Ended Job = job_1703944994811_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 34.29 sec   HDFS Read: 3446697 HDFS Write: 2562 SUCCESS
Total MapReduce CPU Time Spent: 34 seconds 290 msec
OK
```

```
OR
NULL     12     856
1911    The Right of Youth      235
1918    A Dog's Life    282
1933    Sons of the Desert      210
1941    How Green Was My Valley 342
1950    Sunset Boulevard        2278
1951    A Streetcar Named Desire        1213
1957    Gunfight at the O.K. Corral     285
1960    Black Sunday    447
1961    Accattone       428
1962    Vivre Sa Vie    558
1964    Zorba the Greek 304
1965    The Cincinnati Kid      239
1967    Mouchette       211
1968    Carne   304
1969    The Wild Bunch  1043
1971    And Now for Something Completely Different      457
1972    Don't Torture a Duckling        317
1973    Enter the Dragon        1640
1974    The Front Page  302
1976    The Second Tragic Fantozzi      562
1977    Suspiria        2596
1978    La Cage aux Folles      357
1979    The Amityville Horror   785
1980    Airplane!       4016
1981    The Evil Dead   3519
1982    Pieces  211
1983    Trading Places  2897
1984    The Natural     524
1985    Pale Rider      921
1986    Department Store        277
1987    3 Men and a Baby        896
1988    Twins   2015
1989    She-Devil       334
1990    Fantozzi to the Rescue  324
1991    Paprika 349
1992    Lorenzo's Oil   475
1993    Philadelphia    3866
1994    Blown Away      470
```

- 2.2.2

The year is extracted from ‘release_date’ using YEAR() function and a COUNT(*) function is employed to calculate the number of rows of movies and is assigned the alias 'action_movie_count.' A filter is applied using WHERE clause to filter the rows based on the condition that the ‘genre_ids’ column must contain the genre ID '28,' ensuring that only action movies are included in the subsequent count. The results are grouped by the extracted release year.

```
SELECT
  year(release_date) AS release_year,
  COUNT(*) AS action_movie_count
FROM movies_table
WHERE genre_ids LIKE '%28%'
GROUP BY year(release_date);
```

The query and results are as follows:

```
hive> SELECT
    >   year(release_date) AS release_year,
    >   COUNT(*) AS action_movie_count
    > FROM movies_table
    > WHERE genre_ids LIKE '%28%'
    > GROUP BY year(release_date);
WARNING: Hive-on-MR is deprecated in Hive 2 and may not be available in the future versions. Consider using a different execution engine (i.e. spark, tez) o
r using Hive 1.X releases.
Query ID = root_20231230154408_d39be170-fb75-4e3c-8c63-94becbd0a2db
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Starting Job = job_1703944994811_0003, Tracking URL = http://1e82f378c65e:8088/proxy/application_1703944994811_0003/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1703944994811_0003
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2023-12-30 15:45:06,651 Stage-1 map = 0%,  reduce = 0%
2023-12-30 15:46:05,319 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 21.15 sec
2023-12-30 15:46:23,948 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 28.08 sec
MapReduce Total cumulative CPU time: 28 seconds 80 msec
Ended Job = job_1703944994811_0003
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 28.08 sec   HDFS Read: 3444768 HDFS Write: 278 SUCCESS
Total MapReduce CPU Time Spent: 28 seconds 80 msec
OK
```

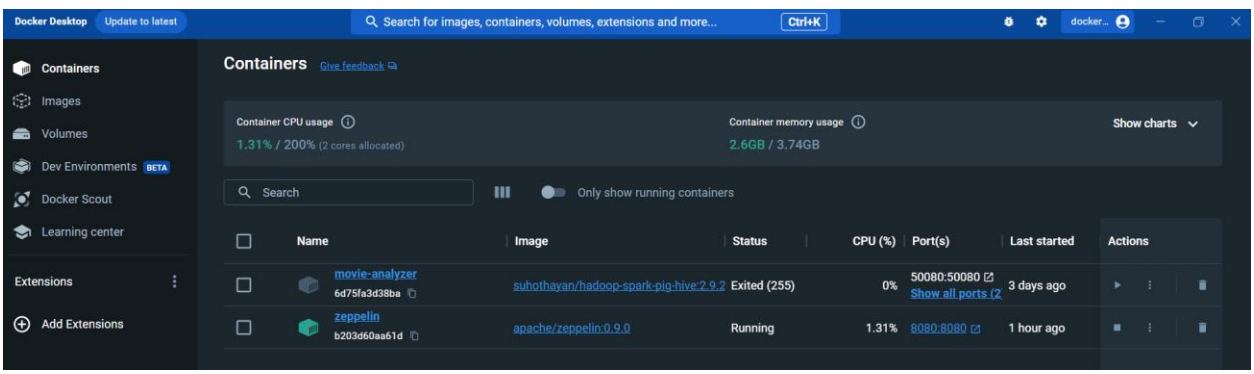
```
OK
NULL    1213
1973    1
1985    1
1989    1
1994    1
1995    1
1998    1
2003    1
2006    1
2021    2
Time taken: 139.189 seconds, Fetched: 10 row(s)
```

2.3

Zeppelin docker container is started using the zeppelin image

- *docker run -it --name zeppelin -p 8080:8080 -v D:\IIT\Semester3-Sep\CMM705-Coursework:/data apache/zeppelin:0.9.0*

```
PS C:\Users\Shahd> docker run -it --name zeppelin -p 8080:8080 -v C:\Users\Shahd\OneDrive\Desktop\Movie:/data apache/zep
pelin:0.9.0
Unable to find image 'apache/zeppelin:0.9.0' locally
0.9.0: Pulling from apache/zeppelin
83ee3a23efb7: Pull complete
db98fc6f11f0: Pull complete
f611acd52c6c: Pull complete
c68f056c1360: Pull complete
d2893470c61e: Pull complete
f8cf2bd9216a: Pull complete
78e3dee51c69: Pull complete
8bfdc346b9be: Pull complete
d760722ddd93: Pull complete
Digest: sha256:20ad134275d061a27bc8888ab6d7d44534f4c751bdc0fb8a0df2111437e57d1d
Status: Downloaded newer image for apache/zeppelin:0.9.0
 WARN [2023-12-29 10:45:41,381] ({main} ZeppelinConfiguration.java[create]:168) - Failed to load configuration, proceedi
ng with a default
 INFO [2023-12-29 10:45:41,448] ({main} ZeppelinConfiguration.java[create]:180) - Server Host: 0.0.0.0
 INFO [2023-12-29 10:45:41,448] ({main} ZeppelinConfiguration.java[create]:182) - Server Port: 8080
 INFO [2023-12-29 10:45:41,448] ({main} ZeppelinConfiguration.java[create]:186) - Context Path: /
 INFO [2023-12-29 10:45:41,449] ({main} ZeppelinConfiguration.java[create]:187) - Zeppelin Version: 0.9.0
 INFO [2023-12-29 10:45:41,794] ({main} Log.java[initialized]:169) - Logging initialized @5226ms to org.eclipse.jetty.ut
```



A notebook is created and a Spark session is initiated. Then the dataset is loaded

```
%spark.pyspark
from pyspark.sql import SparkSession

# Initialize Spark session
spark = SparkSession.builder.appName("MoviesAnalysis").getOrCreate()
```

Took 0 sec. Last updated by anonymous at December 30 2023, 5:04:36 AM.

```
%spark.pyspark

# Load the dataset
data = spark.read.csv("/data/MoviesTopRated.csv", header=True, inferSchema=True)
data.show()
```

```
+---+------+-----------------+--------------------+--------------------+----------+------------+------------+----------+
|_c0|    id|        genre_ids|               title|            overview|popularity|release_date|vote_average|vote_count|
+---+------+-----------------+--------------------+--------------------+----------+------------+------------+----------+
|  0|   238|         [18, 80]|       The Godfather|Spanning the year...|   119.438|  1972-03-14|         8.7|     18448|
|  1|   278|         [18, 80]|The Shawshank Red...|Framed in the 194...|    90.415|  1994-09-23|         8.7|     24376|
|  2|   240|         [18, 80]|The Godfather Par...|In the continuing...|    70.637|  1974-12-20|         8.6|     11144|
|  3|   424|  [18, 36, 10752]|    Schindler's List|The true story of...|    48.096|  1993-12-15|         8.6|     14421|
|  4| 19404|  [35, 18, 10749]|Dilwale Dulhania ...|Raj is a rich, ca...|    26.588|  1995-10-20|         8.6|      4225|
|  5|   389|             [18]|        12 Angry Men|The defense and t...|    40.754|  1957-04-10|         8.5|      7529|
|  6|   129|  [16, 10751, 14]|       Spirited Away|A young girl, Chi...|    73.067|  2001-07-20|         8.5|     14730|
|  7|372058|  [10749, 16, 18]|          Your Name.|High schoolers Mi...|    66.572|  2016-08-26|         8.5|     10152|
|  8|496243|     [35, 53, 18]|            Parasite|All unemployed, K...|    62.752|  2019-05-30|         8.5|     16210|
|  9|   155| [18, 28, 80, 53]|     The Dark Knight|Batman raises the...|    83.163|  2008-07-14|         8.5|     30333|
| 10|   497|     [14, 18, 80]|      The Green Mile|A supernatural ta...|    65.211|  1999-12-10|         8.5|     15763|
| 11|   680|         [53, 80]|        Pulp Fiction|A burger-loving h...|      66.1|  1994-09-10|         8.5|     25678|
| 12|    13|  [35, 18, 10749]|        Forrest Gump|A man with a low ...|   136.867|  1994-06-23|         8.5|     25172|
| 13|   122|     [12, 14, 28]|The Lord of the R...|Aragorn is reveal...|    75.514|  2003-12-01|         8.5|     22053|
```

- 2.3.1

```
%spark.pyspark

# Percentage of movies above three genres

from pyspark.sql.functions import size

# Remove brackets and split the "genre_ids", then filter out rows with invalid genre_ids
df = df.withColumn("genre_ids_array", split(regexp_replace(df["genre_ids"], "[\\[\\]]", ""), ",").cast("array<int>"))

# Percentage of movies classified under at least 3 genres
df_with_genre_count = df.withColumn("genre_count", size(df["genre_ids_array"]))

# Filter out rows with invalid genre counts and calculate percentages
valid_genre_counts = df_with_genre_count.filter(df_with_genre_count["genre_count"] >= 3)

at_least_3_genres_count = valid_genre_counts.count()
total_movies_count = df_with_genre_count.count()

percentage_at_least_3_genres = (at_least_3_genres_count / total_movies_count) * 100

print(f"Number of movies with at least 3 genres: {at_least_3_genres_count}")
print(f"Total Number of movies: {total_movies_count}")
print(f"Percentage of movies with at least 3 genres: {percentage_at_least_3_genres:.2f}%")
```

```
Number of movies with at least 3 genres: 5232
Total Number of movies: 10004
Percentage of movies with at least 3 genres: 52.30%
```

The ‘genre_ids’ are of type String. The split function is used to split the String representation of the array and is then casted to array of integers. Next the number of movies containing three or more genres and the total number of movies are calculated. The percentage is calculated by considering this ratio

- 2.3.2

```
%spark.pyspark

# Total number of movies for each genre

from pyspark.sql.functions import explode, count, regexp_replace, split

# Remove brackets and split the "genre_ids", then explode the resulting array
df_exploded_genres = df.select("title", explode(split(regexp_replace(df["genre_ids"], "[\\[\\]]", ""), ",")).alias("genre_id"))

# Filter out rows with invalid genre_id values
df_valid_genres = df_exploded_genres.filter(df_exploded_genres["genre_id"].cast("int").isNotNull())

# Total number of movies released for each genre
genre_counts = df_valid_genres.groupBy("genre_id").agg(count("title").alias("movie_count"))

genre_counts.show()
```

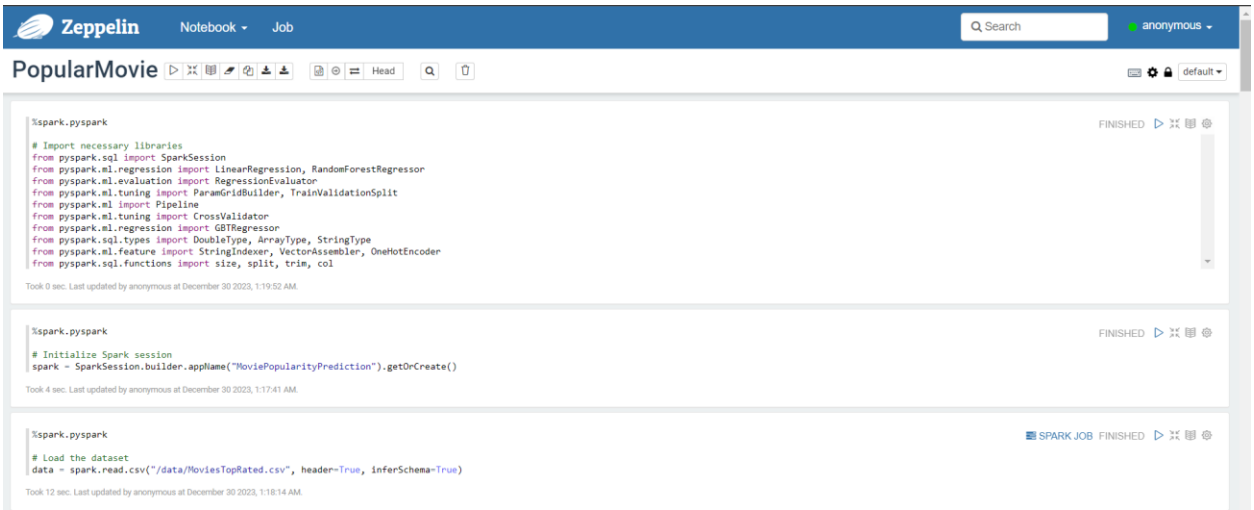
```
+--------+-----------+
|genre_id|movie_count|
+--------+-----------+
|   10752|         91|
|     878|        264|
|      28|       1223|
|      35|       2048|
|      16|        598|
|   10770|         28|
|      18|       2418|
|      27|        684|
|   10402|         78|
|   10749|        312|
|      53|        492|
|   10751|        274|
|      36|         48|
|      37|         75|
|    9648|        115|
```

Initially the ‘regexp_replace’ is used to remove the square brackets from ‘genre_ids’ and then split to remove the string representation. The filter method is then used to keep only rows where the ‘genre_id’ can be successfully cast to an integer. The cast("int").isNotNull() condition ensures that only valid integer values are retained. The resulting array is then exploded and the counts are calculated grouped by the movie title

QUESTION 3

The zeppelin container is already created and the same will be used here.

As the initial step necessary libraries are imported and spark session is started, followed by the loading of the dataset.



The question requires to predict the popularity of a movie given two genres. Therefore, initially the data is filtered to obtain the movies that belong to any two genres.

```
%spark.pyspark

# Getting rows with two genres

# Split the 'genre_ids' string into an array of strings and trim whitespaces
df = data.withColumn(
    "genre_ids_array",
    split(trim(data["genre_ids"]), ",").cast(ArrayType(StringType()))
)

# Filter rows with exactly two elements in the 'genre_ids' array
df = df.filter(size("genre_ids_array") == 2)

# Show the resulting DataFrame
df.show()
```

```
+---+------+-----------+--------------------+--------------------+--------------------+--------------------+------------+---------+---------------+
|_c0|    id|  genre_ids|               title|            overview|          popularity|        release_date|vote_average|vote_count|genre_ids_array|
+---+------+-----------+--------------------+--------------------+--------------------+--------------------+------------+---------+---------------+
|  0|   238|   [18, 80]|       The Godfather|Spanning the year...|             119.438|          1972-03-14|         8.7|    18448|    [[18,  80]]|
|  1|   278|   [18, 80]|The Shawshank Red...|Framed in the 194...|              90.415|          1994-09-23|         8.7|    24376|    [[18,  80]]|
|  2|   240|   [18, 80]|The Godfather Par...|In the continuing...|              70.637|          1974-12-20|         8.6|    11144|    [[18,  80]]|
| 11|   680|   [53, 80]|        Pulp Fiction|A burger-loving h...|               66.1|          1994-09-10|         8.5|    25678|    [[53,  80]]|
| 15|   769|   [18, 80]|          GoodFellas|The true story of...|             108.118|          1990-09-12|         8.5|    11586|    [[18,  80]]|
| 17| 11216|[18, 10749]|     Cinema Paradiso|A filmmaker recal...|              27.051|          1988-11-17|         8.5|     3896| [[18,  10749]]|
| 18|667257|[10751, 18]|   Impossible Things|Matilde is a woma...|              19.13|          2021-06-17|         8.4|      351| [[10751,  18]]|
| 19|   637|   [35, 18]|   Life Is Beautiful|A touching story ...|              38.32|          1997-12-20|         8.5|    12111|    [[35,  18]]|
| 20|696374|[10749, 18]|   Gabriel's Inferno|An intriguing and...|              15.44|          2020-05-29|         8.5|     2355| [[10749,  18]]|
| 21|   346|   [28, 18]|       Seven Samurai|A samurai answers...|             74.414|          1954-04-26|         8.5|     3175|    [[28,  18]]|
| 23|772071|   [35, 14]|    Cuando Sea Joven|"70-year-old Male...| she becomes the ...| which she had to...|      11.555|2022-09-14|    [[35,  14]]|
| 27|   311|   [18, 80]|Once Upon a Time ...|A former Prohibit...|              31.708|          1984-05-23|         8.4|     4786|    [[18,  80]]|
| 29|   598|   [18, 80]|         City of God|In the slums of R...|              28.469|          2002-08-30|         8.4|     6619|    [[18,  80]]|
| 30|724089|[10749, 18]|Gabriel's Inferno...|Professor Gabriel...|              18.031|          2020-07-31|         8.4|     1481| [[10749,  18]]|
```

The genre ids are available as array, it is segregated to two different columns.

```
%spark.pyspark

# Removing the genre array to columns
import pyspark.sql.functions as f

df1 = df.select("id","title","popularity","genre_ids", f.translate(f.col("genre_ids"), "][ ", "").alias("genre_id"))
df2 = df1.withColumn("genre_id", f.split(df1.genre_id, ","))
df3 = df2.select(
    df2.id,
    df2.genre_ids,
    df2.title,
    df2.popularity,
    df2.genre_id.getItem(0).alias('genre1'),
    df2.genre_id.getItem(1).alias('genre2'),
)
df4 = df3.na.fill("0")
df5 = df4
df5.show()
```

```
+------+-----------+--------------------+--------------------+------+------+
|    id|  genre_ids|               title|          popularity|genre1|genre2|
+------+-----------+--------------------+--------------------+------+------+
|   238|   [18, 80]|       The Godfather|             119.438|    18|    80|
|   278|   [18, 80]|The Shawshank Red...|              90.415|    18|    80|
|   240|   [18, 80]|The Godfather Par...|              70.637|    18|    80|
|   680|   [53, 80]|        Pulp Fiction|                66.1|    53|    80|
|   769|   [18, 80]|          GoodFellas|             108.118|    18|    80|
| 11216|[18, 10749]|     Cinema Paradiso|              27.051|    18| 10749|
|667257|[10751, 18]|   Impossible Things|               19.13| 10751|    18|
|   637|   [35, 18]|   Life Is Beautiful|               38.32|    35|    18|
|696374|[10749, 18]|   Gabriel's Inferno|               15.44| 10749|    18|
|   346|   [28, 18]|       Seven Samurai|              74.414|    28|    18|
|772071|   [35, 14]|    Cuando Sea Joven| she becomes the ...|    35|    14|
|   311|   [18, 80]|Once Upon a Time ...|              31.708|    18|    80|
|   598|   [18, 80]|         City of God|              28.469|    18|    80|
|724089|[10749, 18]|Gabriel's Inferno...|              18.031| 10749|    18|
```

Took 1 sec. Last updated by anonymous at December 30 2023, 4:24:04 AM.

Since all columns are not required for model training, only the required columns are selected

```
%spark.pyspark

# Filtering the data
final_df = df5["title","popularity","genre1","genre2"]
final_df.show()
```

```
|     Cinema Paradiso|              27.051|    18| 10749|
|   Impossible Things|               19.13| 10751|    18|
|   Life Is Beautiful|               38.32|    35|    18|
|   Gabriel's Inferno|               15.44| 10749|    18|
|       Seven Samurai|              74.414|    28|    18|
|    Cuando Sea Joven| she becomes the ...|    35|    14|
|Once Upon a Time ...|              31.708|    18|    80|
|         City of God|              28.469|    18|    80|
|Gabriel's Inferno...|              18.031| 10749|    18|
|Dou kyu sei - Cla...|              12.028| 10749|    16|
|Gabriel's Inferno...|              30.889| 10749|    18|
|A Silent Voice: T...|              36.158|    16|    18|
|         The Pianist|               33.48|    18| 10752|
|            Whiplash|              56.287|    18| 10402|
|Red, White & Roya...|             727.969|    35| 10749|
+--------------------+--------------------+------+------+
only showing top 20 rows
```

The processed data set has column types which are string. These columns are then casted to numeric types for further processing followed by null value treatment

```
%spark.pyspark

# Converting the data types
from pyspark.sql.types import DoubleType

final_df = final_df.withColumn('popularity', final_df['popularity'].cast(DoubleType()))
final_df = final_df.withColumn('genre1', final_df['genre1'].cast(DoubleType()))
final_df = final_df.withColumn('genre2', final_df['genre2'].cast(DoubleType()))
```

Took 0 sec. Last updated by anonymous at December 30 2023, 4:24:55 AM.

```
%spark.pyspark

# Checking for null values

final_df = final_df.na.fill(0)
final_df.filter(final_df.title.isNull()).show()
```

```
+-----+----------+------+------+
|title|popularity|genre1|genre2|
+-----+----------+------+------+
+-----+----------+------+------+
```

Before feeding the data for model training, the data should be transformed into vectors. OneHotEncoderEstimator, StringIndexer and VectorAssembler are used for this purpose and the transformation is done via a pipeline

FINISHED

```
%spark.pyspark

# Vectorize

from pyspark.ml.feature import StringIndexer, VectorAssembler, OneHotEncoder, OneHotEncoderEstimator

categoricalColumns = ['title','genre1', 'genre2']
stages=[]

for col in categoricalColumns:
    stringIndexer = StringIndexer(inputCol = col , outputCol = col + 'Index', handleInvalid="skip")
    encoder = OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[col + "classVec"])
    stages += [stringIndexer,encoder]

assembler = VectorAssembler(inputCols=[c + "classVec" for c in categoricalColumns],
                            outputCol="features")
stages += [assembler]
```

Took 0 sec. Last updated by anonymous at December 30 2023, 4:25:51 AM.

SPARK JOB FINISHED

```
%spark.pyspark

from pyspark.ml import Pipeline

cols = final_df.columns
pipeline = Pipeline(stages=stages)
pipelineModel = pipeline.fit(final_df)
ml_transform=pipelineModel.transform(final_df)
selectedCols = ['features'] + cols
ml_data = ml_transform.select(selectedCols)
ml_data.printSchema()
```

```
root
 |-- features: vector (nullable = true)
 |-- title: string (nullable = false)
 |-- popularity: double (nullable = false)
 |-- genre1: double (nullable = false)
 |-- genre2: double (nullable = false)
```

Took 5 sec. Last updated by anonymous at December 30 2023, 4:25:58 AM.

Thus, the final preprocessed data set contains ‘populairty’ which is the dependent variable to be predicted while the required columns title and genre are independent columns

Preprocessed data set is split into training and testing datasets in a ration of 8:2. Training data is used to fit the model while the testing data is used to validate the trained model.

```
%spark.pyspark

training_data, validation_data = ml_data.randomSplit([0.8, 0.2], seed=42)
print("Training Data : "+str(training_data.count()))
print("Validation Data : "+str(validation_data.count()))

Training Data : 2544
Validation Data : 609

Took 2 sec. Last updated by anonymous at December 30 2023, 4:53:33 AM.
```

Three models ‘Linear Regression’, ‘Decision Tree Regression’ and ‘Gradient Boost Tree Regression’ are fitted using the training data followed by testing and evaluation. Root Mean Square Error (RMSE) is used as the evaluation metric to evaluate the model

```
%spark.pyspark                                                    SPARK JOB FINISHED

# Linear Regression
lr = LinearRegression(labelCol="popularity", featuresCol="features")
lr_model = lr.fit(training_data)

lr_predictions = lr_model.transform(validation_data)
lr_evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='popularity',metricName='rmse')

rmse = lr_evaluator.evaluate(lr_predictions)
print("RMSE : %g" % rmse)

RMSE : 35.6778

Took 11 sec. Last updated by anonymous at December 30 2023, 4:53:45 AM.
```

```
%spark.pyspark                                                    SPARK JOB FINISHED

from pyspark.ml.regression import DecisionTreeRegressor

dt = DecisionTreeRegressor(labelCol="popularity", featuresCol="features")
dt_model = dt.fit(training_data)

dt_predictions = dt_model.transform(validation_data)
dt_evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='popularity',metricName='rmse')

rmse = dt_evaluator.evaluate(dt_predictions)
print("RMSE : %g" % rmse)

RMSE : 34.2808

Took 3 sec. Last updated by anonymous at December 30 2023, 4:55:01 AM.
```

```
%spark.pyspark                                                    SPARK JOB FINISHED

from pyspark.ml.regression import GBTRegressor

gbt = GBTRegressor(labelCol="popularity", featuresCol="features")
gbt_model = gbt.fit(training_data)

gbt_predictions = gbt_model.transform(validation_data)
gbt_evaluator = RegressionEvaluator(predictionCol='prediction', labelCol='popularity',metricName='rmse')

rmse = gbt_evaluator.evaluate(gbt_predictions)
print("RMSE : %g" % rmse)

RMSE : 34.289
```

The models produce the following results:

Model	RMSE
Linear Regression	35.6778
Decision Tree Regression	34.2808
Gradient Boost Tree Regression	34.289

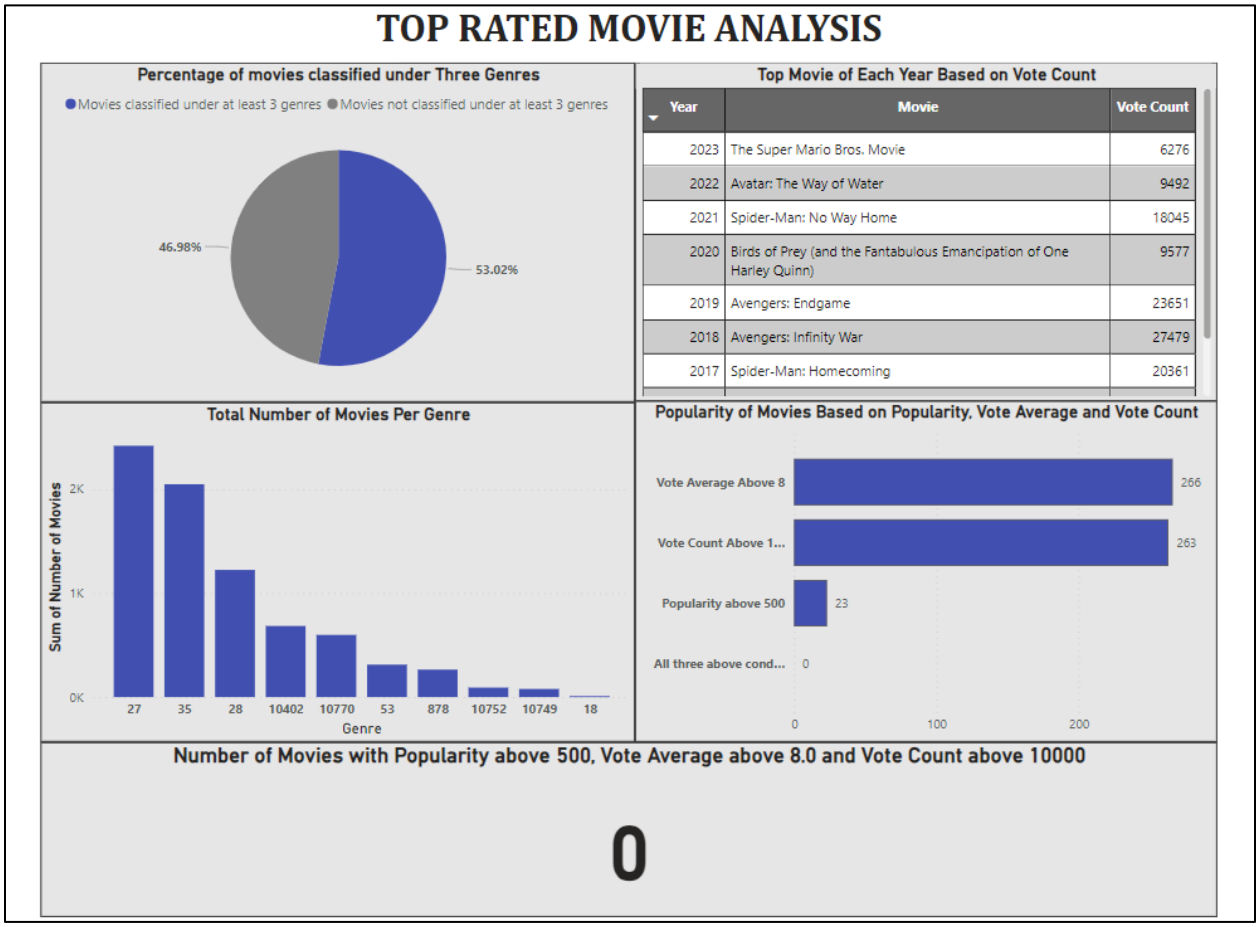
Lower the RMSE, better the model performance. Comparatively, the Decision Tree produces the best results. Accordingly, it will be used to predict the popularity of a movie given that it belongs to two genres

```
%spark.pyspark

dt_predictions.select('title','genre1','genre2','popularity').show()
```

```
+--------------------+-------+-------+----------+
|               title| genre1| genre2|popularity|
+--------------------+-------+-------+----------+
|   Wuthering Heights|   18.0|10749.0|     9.923|
|   Wuthering Heights|   18.0|10749.0|    11.781|
|  War of the Buttons|   35.0|   18.0|     5.844|
|           The Ledge|   53.0|   12.0|     58.06|
|      The Stepfather|   27.0|   53.0|     9.909|
|       The Innocents|   27.0| 9648.0|     8.335|
|     The Parent Trap|   35.0|10751.0|    17.163|
|     The Parent Trap|   35.0|10751.0|    53.308|
|              Legend|   12.0|   14.0|    23.621|
|       The Immigrant|   35.0|10749.0|     6.283|
|          Metropolis|   16.0|  878.0|    11.956|
|           Overboard|   35.0|10749.0|    19.661|
|          Sweetheart|   35.0|   18.0|     4.411|
|           Leap Year|   18.0|10749.0|     8.149|
|    Miss You Already|   35.0|   18.0|    10.597|
```

Took 1 sec. Last updated by anonymous at December 30 2023, 4:56:32 AM.

QUESTION 4


TOP RATED MOVIE ANALYSIS
Percentage of movies classified under Three Genres
Movies classified under at least 3 genres
Movies not classified under at least 3 genres
46.98%
53.02%
Top Movie of Each Year Based on Vote Count
Year
Movie
Vote Count
2023 The Super Mario Bros. Movie 6276
2022 Avatar: The Way of Water 9492
2021 Spider-Man: No Way Home 18045
2020 Birds of Prey (and the Fantabulous Emancipation of One Harley Quinn) 9577
2019 Avengers: Endgame 23651
2018 Avengers: Infinity War 27479
2017 Spider-Man: Homecoming 20361
Total Number of Movies Per Genre
Sum of Number of Movies
2K
1K
0K
27
35
28
10402
10770
53
878
10752
10749
18
Genre
Popularity of Movies Based on Popularity, Vote Average and Vote Count
Vote Average Above 8
266
Vote Count Above 1...
263
Popularity above 500
23
All three above cond...
0
0
100
200
Number of Movies with Popularity above 500, Vote Average above 8.0 and Vote Count above 10000
0