ASSIGMENT 1

SUBJECT: HADOOP
ENTROLLMENT NUMBER: 202300819010032
NAME: Pinjani Monica Chanderlal

1.Develop a MapReduce job that will split each line of the input text file into tokens and then count the occurrences of each unique token. The input text file should be available on the HDFS. The output will be in the form of:
Hadoop 5
Hotspot 2

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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.util.StringTokenizer;

public class WordCount {


    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
```

```
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{

        StringTokenizer itr = new StringTokenizer(value.toString());
        while (itr.hasMoreTokens()) {
          word.set(itr.nextToken());
          context.write(word, one);  // Emit each word with count 1
        }
      }
  }


  public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
        int sum = 0;

        for (IntWritable val : values) {
          sum += val.get();
        }
        result.set(sum);
        context.write(key, result);  // Emit word and its total count
      }
  }


  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path
    FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

2.Develop a MapReduce job that will analyze the minimum temperature for each year. The input text file should be available on HDFS. The output will be in the form of:
2014 -1
2015 20

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class MinTemperature {


   public static class TempMapper extends Mapper<Object, Text, Text, IntWritable> {

      private Text year = new Text();
      private IntWritable temperature = new IntWritable();

      public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{

         String[] parts = value.toString().split("\\s+");
         if (parts.length == 2) {

            year.set(parts[0]);  // Set the year as the key
            temperature.set(Integer.parseInt(parts[1]));  // Set the temperature as the value
            context.write(year, temperature);  // Emit year as key and temperature as value
         }
      }
   }


   public static class MinTempReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
    private IntWritable minTemperature = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
        int minTemp = Integer.MAX_VALUE;


        for (IntWritable value : values) {
            minTemp = Math.min(minTemp, value.get());
        }

        minTemperature.set(minTemp);
        context.write(key, minTemperature);  // Emit the year and the minimum temperature
    }
  }


  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "min temperature");
    job.setJarByClass(MinTemperature.class);
    job.setMapperClass(TempMapper.class);
    job.setReducerClass(MinTempReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path
    FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

3.Develop a MapReduce job that will split each line of the input text file into tokens and then find the average count of all the tokens in the input file. The input text file should be available on the HDFS. The output will be in the form of:
Hadoop 5
Hotspot 2
AverageCount = 3.5

TokenCount.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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.util.StringTokenizer;

public class TokenCount {

   public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

      private final static IntWritable one = new IntWritable(1);
      private Text word = new Text();

      public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
         StringTokenizer itr = new StringTokenizer(value.toString());
         while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            context.write(word, one);  // Emit each word with count 1
         }
      }
   }
```

```
    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);  // Emit word and its total count
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "token count");
        job.setJarByClass(TokenCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

AverageCount.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class AverageCount {
```

```
   public static class SumMapper extends Mapper<Object, Text, Text, IntWritable> {
      private final static Text tokenKey = new Text("TokenCount");
      private IntWritable count = new IntWritable();

      public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
         String[] parts = value.toString().split("\\s+");
         if (parts.length == 2) {
            count.set(Integer.parseInt(parts[1]));
            context.write(tokenKey, count);  // Emit a single key for summing all counts
         }
      }
   }


   public static class AverageReducer extends Reducer<Text, IntWritable, Text, Text> {
      public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
         int sum = 0;
         int count = 0;

         for (IntWritable value : values) {
            sum += value.get();
            count++;
         }

         double average = (double) sum / count;  // Calculate the average count
         context.write(new Text("AverageCount = "), new Text(String.valueOf(average)));
      }
   }

   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "average count");
      job.setJarByClass(AverageCount.class);
      job.setMapperClass(SumMapper.class);
      job.setReducerClass(AverageReducer.class);
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(IntWritable.class);
      FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path
      FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path
      System.exit(job.waitForCompletion(true) ? 0 : 1);
```

```
    }
}
```

4.Develop a MapReduce job that will take text file input and will produce the output:
(1) Count of the total token having length greater than or equal to four characters.
Total count for token = 5

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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.util.StringTokenizer;

public class TokenLengthCount {


   public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable> {

      private final static IntWritable one = new IntWritable(1);
      private final static Text tokenKey = new Text("TokenCount");

      public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
         StringTokenizer itr = new StringTokenizer(value.toString());
         while (itr.hasMoreTokens()) {
            String token = itr.nextToken();
            if (token.length() >= 4) {
               context.write(tokenKey, one);  // Emit 'TokenCount' key with value 1
            }
         }
      }
   }


   public static class TokenCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
      private IntWritable result = new IntWritable();
```

```
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(new Text("Total count for token ="), result);  // Emit total token count
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "token length count");
        job.setJarByClass(TokenLengthCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setReducerClass(TokenCountReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

5.Develop a MapReduce job that will count the total number of females voters in the record file Voters.txt. The input text file should be available on the HDFS. The file must have fields: ID, NAME, GENDER, AGE. The output will be in the form of:
No. of female voters are: 5

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class FemaleVoterCount {

    // Mapper Class
    public static class VoterMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private final static Text femaleVoterKey = new Text("FemaleVoter");

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{

            String[] fields = value.toString().split(",");


            if (fields.length == 4 && fields[2].equalsIgnoreCase("Female")) {
                context.write(femaleVoterKey, one);  // Emit 'FemaleVoter' key with value 1
            }
        }
    }

    // Reducer Class
    public static class VoterReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();
```

```
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
        int sum = 0;
        // Sum up all the female voter counts
        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        // Emit the total count of female voters
        context.write(new Text("No. of female voters are :"), result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "female voter count");
    job.setJarByClass(FemaleVoterCount.class);
    job.setMapperClass(VoterMapper.class);
    job.setReducerClass(VoterReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path
    FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

6.Develop a MapReduce job that will count the number of reviews given by each unique user for the Musical Instruments. The file should be available on HDFS. It's available on Moodle (Musical_instruments_reviews.csv). The output should be in the form of
A00625243BI8W1SSZNLMD 3
A10044ECXDUVKS 2
A102MU6ZC9H1N6 3
A109JTUZXO61UY 3
A109ME7C09HM2M 4
A10APIDAZISWQF 1
A10E3QH2FQUBLF 3
A10FM4ILBIMJJ7 5
A10H2F00ZOT8S2 4
A10HYGDU2NITYQ 3
A10KH8EN77ZKWH 3
A10N243R7A5ZW3 1

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class ReviewCount {


    public static class ReviewMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text userId = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
```

```
        String[] fields = value.toString().split(",");


        if (fields.length > 0) {
            String userID = fields[0];  // Replace with the correct index if different
            userId.set(userID);
            context.write(userId, one);  // Emit the user ID and count 1 for each review
        }
    }
}


public static class ReviewReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
        int sum = 0;

        for (IntWritable val : values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "review count");
    job.setJarByClass(ReviewCount.class);
    job.setMapperClass(ReviewMapper.class);
    job.setReducerClass(ReviewReducer.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(true) ? 0 : 1);
}
}
```

7.Perform the following on the movies.csv dataset.

7.1 Write a MapReduce job to display all the details of the comedy movies.

```
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class ComedyMovies {

   public static class ComedyMapper extends Mapper<Object, Text, Text, Text> {

      public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
         String[] fields = value.toString().split(",");
         if (fields.length == 3 && fields[2].contains("Comedy")) {
            context.write(new Text(fields[0]), value);
         }
      }
   }

   public static class ComedyReducer extends Reducer<Text, Text, Text, Text> {
      public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
         for (Text val : values) {
            context.write(key, val);
         }
      }
   }

   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
```

```
        Job job = Job.getInstance(conf, "comedy movies");
        job.setJarByClass(ComedyMovies.class);
        job.setMapperClass(ComedyMapper.class);
        job.setReducerClass(ComedyReducer.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(true) ? 0 : 1);
    }
}
```

7.2 Write a mapreduce job to find the count of the Documentary movies released in the year 1995.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class Documentary1995Count {

    public static class DocumentaryMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
            String[] fields = value.toString().split(",");
            if (fields.length == 3) {
                String title = fields[1];
                String genres = fields[2];

                if (title.contains("(1995)") && genres.contains("Documentary")) {
                    context.write(new Text("DocumentaryMovies1995"), one);
                }
            }
        }
    }

    public static class DocumentaryReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
```

```
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "documentary 1995 count");
        job.setJarByClass(Documentary1995Count.class);
        job.setMapperClass(DocumentaryMapper.class);
        job.setReducerClass(DocumentaryReducer.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(true) ? 0 : 1);
    }
}
```

7.3 Write a MapReduce job that will count the total number of missing records where ‘genres’ are missing. (Note: Remove genre from few records before processing.)

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class MissingGenreCount {

    public static class MissingGenreMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
            String[] fields = value.toString().split(",");
            if (fields.length < 3 || fields[2].isEmpty()) {
                context.write(new Text("MissingGenres"), one);
            }
        }
    }

    public static class MissingGenreReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
```

```
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "missing genres count");
        job.setJarByClass(MissingGenreCount.class);
        job.setMapperClass(MissingGenreMapper.class);
        job.setReducerClass(MissingGenreReducer.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(true) ? 0 : 1);
    }
}
```

7.4 Write a MapReduce job to display only titles of the movie having “Gold” anywhere in the title.

```
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class GoldMovies {

    // Mapper Class
    public static class GoldMovieMapper extends Mapper<Object, Text, Text, Text> {

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
            // Split the CSV line by commas
            String[] fields = value.toString().split(",");

            if (fields.length > 1) {
                String title = fields[1]; // Assuming the title is in the second column
                // Check if the title contains "Gold"
                if (title.contains("Gold")) {
                    // Emit the movie title if it contains "Gold"
                    context.write(new Text(title), new Text(""));
                }
            }
        }
    }

    // Reducer Class
    public static class GoldMovieReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
            // Directly write the movie title to the output
            context.write(key, new Text(""));
        }
```

```
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "gold movie titles");
        job.setJarByClass(GoldMovies.class);
        job.setMapperClass(GoldMovieMapper.class);
        job.setReducerClass(GoldMovieReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path on HDFS
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path on HDFS
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

7.5 Write a MapReduce that will display the count of the movies which belong to both Drama and Romantic genre.

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
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;

public class DramaRomanticMovieCount {

    // Mapper Class
    public static class GenreMapper extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text genreKey = new Text("DramaAndRomanticMovies");

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException
{
            // Split the CSV line by commas
            String[] fields = value.toString().split(",");

            if (fields.length == 3) {
                String genres = fields[2]; // Assuming the genres are in the third column

                // Check if the genres contain both "Drama" and "Romantic"
                if (genres.contains("Drama") && genres.contains("Romantic")) {
                    // Emit the key "DramaAndRomanticMovies" with a count of 1
                    context.write(genreKey, one);
                }
            }
        }
    }

    // Reducer Class
    public static class GenreReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
```

```
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            // Sum up all the counts for movies belonging to both genres
            for (IntWritable val : values) {
                sum += val.get();
            }
            // Output the final count
            context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "drama and romantic movie count");
        job.setJarByClass(DramaRomanticMovieCount.class);
        job.setMapperClass(GenreMapper.class);
        job.setReducerClass(GenreReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path on HDFS
        FileOutputFormat.setOutputPath(job, new Path(args[1]));  // Output path on HDFS
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```