# Big Data Analysis Spring 2022

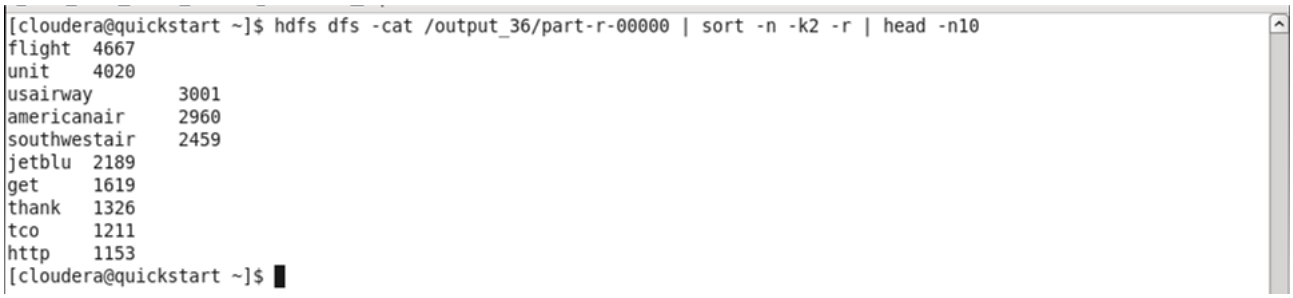
## Question # 1

### - Code

```
import java.io.IOException;
import java.util.StringTokenizer;
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;

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);
      }
    }
  }

  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);
    }
  }

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

- **Output**



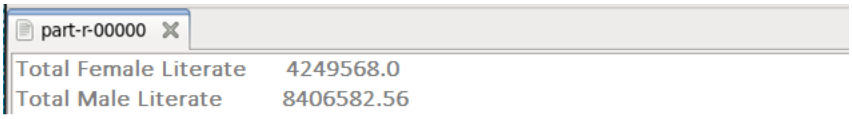
# Question # 2

- **Part A**

```
import java.awt.List;
import java.io.IOException;
import java.util.*;
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {


	public static class Map extends Mapper<LongWritable, Text, Text, FloatWritable> {

		private Text result = new Text();

		private FloatWritable counter = new FloatWritable();

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

			String lineParser = value.toString();
			String strArr[] = lineParser.split(",");
			if (strArr.length >= 3) {
				if(!strArr[1].equalsIgnoreCase("Males")) {
					result.set("Total Males Literate");

					int count = Integer.parseInt(strArr[1]);
					float percentage = Float.parseFloat(strArr[3]);
					percentage = (float) (percentage /100.0);
					float total = percentage * (float)count;
					counter.set(total);
					context.write(result, counter);

					result.set("Total Females Literate");
					count = Integer.parseInt(strArr[2]);
					percentage = Float.parseFloat(strArr[4]);
					percentage = (float) (j/100.0);
					total = percentage * (float) count;
					counter.set(k);
					context.write(result, counter);
				}
			}
		}
	}
	public static class Reduce extends Reducer<Text, FloatWritable, Text, FloatWritable> {
```

```
                public void reduce(Text key, Iterable<FloatWritable> values,
                                                        Context context) throws IOException, InterruptedException {

                        float sum = 0;
                        for (FloatWritable val : values) {
                                sum += val.get();
                        }
                        context.write(key, new FloatWritable(sum));
                }
        }

        public static void main(String[] args) throws Exception {
                Configuration conf = new Configuration();
                Job job = Job.getInstance(conf, "word count");
           job.setJarByClass(WordCount.class);
           job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(FloatWritable.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(FloatWritable.class);
                job.setMapperClass(Map.class);
                job.setReducerClass(Reduce.class);
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                job.waitForCompletion(true);
        }
}
```

- **Output**



- **Part B**

```
import java.awt.List;
import java.io.IOException;
import java.util.*;
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {


	public static class Map extends Mapper<LongWritable, Text, Text, FloatWritable> {

		private Text result = new Text();

		private FloatWritable counter = new FloatWritable();

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

			String lineParser = value.toString();
			String strArr[] = lineParser.split(",");
			if (strArr.length >= 3) {
				if(!strArr[1].equalsIgnoreCase("Males")) {
					result.set("Literate Males");

					float percentage = Float.parseFloat(strArr[3]);
					counter.set(percentage);
					context.write(result, counter);

					result.set("Literate Females");
					percentage = Float.parseFloat(strArr[4]);
					counter.set(percentage);
					context.write(result, counter);
				}
			}
		}
	}
	public static class Reduce extends Reducer<Text, FloatWritable, Text, FloatWritable> {

		// overriding reduce method(runs each time for every key )
		public void reduce(Text key, Iterable<FloatWritable> values,
				Context context) throws IOException, InterruptedException {


			float sum = 0;
			int avgCounter = 0;
			float avg = 0.0
```

```
                for (FloatWritable val : values) {
                        avgCounter += 1;
                        sum += val.get();
                }
                avg =(float)sum/avgCounter;
                context.write(key, new FloatWritable(avg));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(FloatWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(FloatWritable.class);
        job.setMapperClass(Map.class);
        job.setReducerClass(Reduce.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.waitForCompletion(true);
    }
}
```

- **Output**

- **Part C**

```
import java.awt.List;
import java.io.IOException;
import java.util.*;
import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {


	public static class Map extends Mapper<LongWritable, Text, Text, FloatWritable> {

		private Text result = new Text();

		private FloatWritable counter = new FloatWritable();

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

			String lineParser = value.toString();
			String strArr[] = lineParser.split(",");
			if (strArr.length >= 3) {
				if(!strArr[1].equalsIgnoreCase("Males")) {

					result.set("Total Literate");
					int count1 = Integer.parseInt(strArr[1]);
					float percentage1 = Float.parseFloat(strArr[3]);
					int count2 = Integer.parseInt(strArr[2]);
					float percentage2 = Float.parseFloat(strArr[4]);
					percentage1 = (float) (percentage1/100.0);
					percentage2 = (float) (percentage2/100.0);
					float total = (percentage1 * (float)count1) + (percentage2 * (float)count2);
					counter.set(total);
					context.write(result, counter);

					result.set("Total Iliterate");
					int count1 = Integer.parseInt(strArr[1]);
					float percentage1 = Float.parseFloat(strArr[3]);
					int count2 = Integer.parseInt(strArr[2]);
					float percentage2 = Float.parseFloat(strArr[4]);
					percentage1 = 1- (float) (percentage1/100.0);
					percentage2 = 1 - (float) (percentage2/100.0);
					float total = (percentage1 * (float)count1) + (percentage2 * (float)count2);
					counter.set(total);
					context.write(result, counter);
```

```
                                        }
                                }
                        }
                }
        public static class Reduce extends Reducer<Text, FloatWritable, Text, FloatWritable> {

                // overriding reduce method(runs each time for every key )
                public void reduce(Text key, Iterable<FloatWritable> values,
                                                                Context context) throws IOException, InterruptedException {


                                float sum = 0;
                                for (FloatWritable val : values) {
                                        sum += val.get();
                                }
                                context.write(key, new FloatWritable(sum));
                        }
        }

        public static void main(String[] args) throws Exception {
                Configuration conf = new Configuration();
                Job job = Job.getInstance(conf, "word count");
           job.setJarByClass(WordCount.class);
           job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(FloatWritable.class);
                job.setOutputKeyClass(Text.class);
                job.setOutputValueClass(FloatWritable.class);
                job.setMapperClass(Map.class);
                job.setReducerClass(Reduce.class);
                job.setInputFormatClass(TextInputFormat.class);
                job.setOutputFormatClass(TextOutputFormat.class);
                FileInputFormat.addInputPath(job, new Path(args[0]));
                FileOutputFormat.setOutputPath(job, new Path(args[1]));
                job.waitForCompletion(true);
        }
}
```

- **Output**