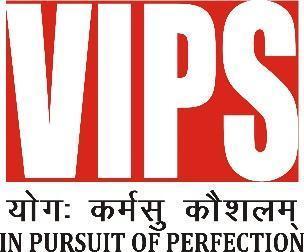
****

**VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS**

**Grade A++ Accredited Institution by NAAC**

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;

Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

An ISO 9001:2015 Certified Institution

**SCHOOL OF ENGINEERING & TECHNOLOGY**

**BTECH Programme: AIDS-**

**Course Title: Big Data Analytics Lab**

**Course Code: AIDS306P**

**Submitted By**

**Name: Neeraj Rathore**

**Enrollment No:** 03817711922

**VISION OF INSTITUTE**

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

**MISSION OF INSTITUTE**

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building  
capabilities to enable them to make significant contributions to the world.

**INDEX**

| **S.No** | **Experiment Name** | **Date** | **Marks** | | | **Remark** | **Updated Marks** | **Faculty Signature** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Laboratory Assessment (15 Marks)** | **Class Participation (5 Marks)** | **Viva (5 Marks)** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

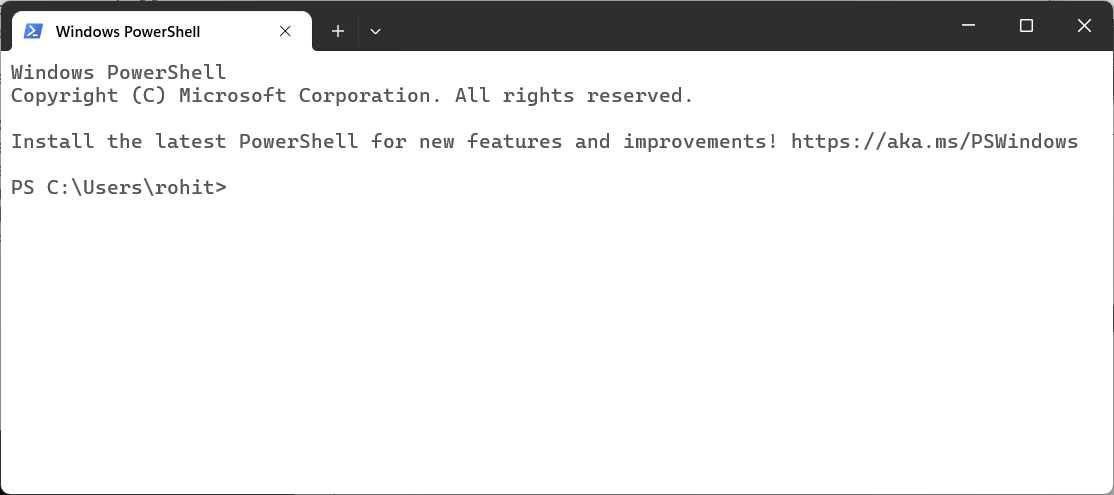
# Experiment 1

## Experiment 1: Install Apache Hadoop.

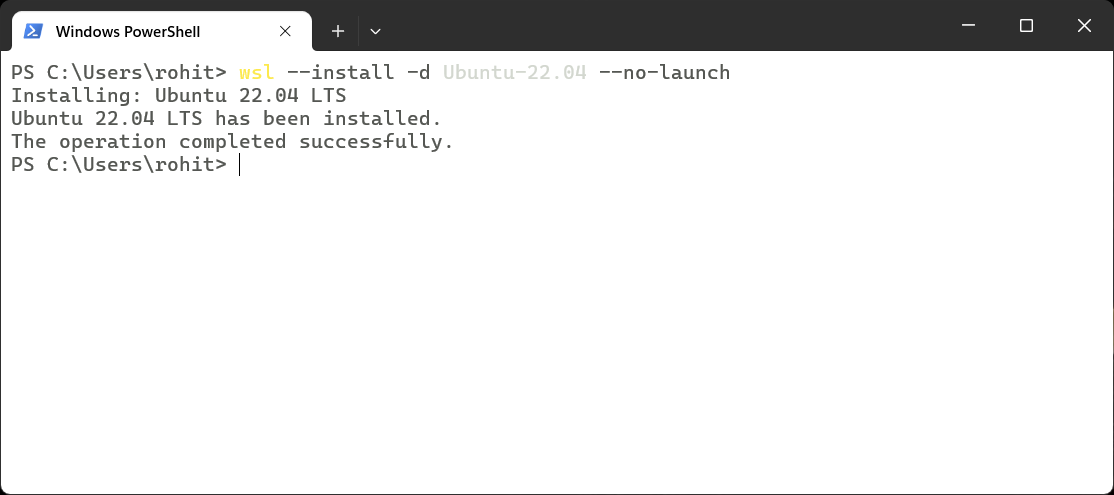
### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

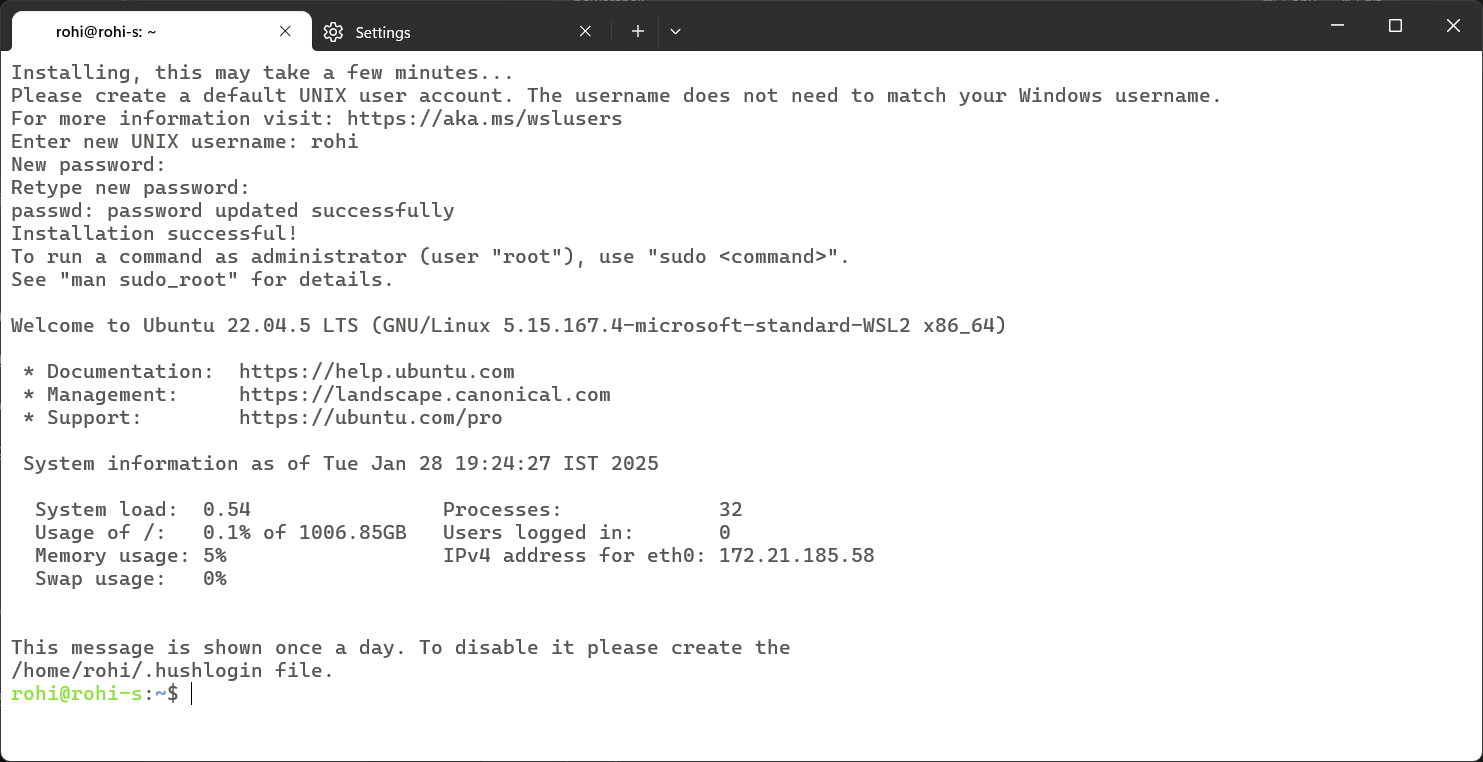
### Installing Windows Subsystem for Linux.





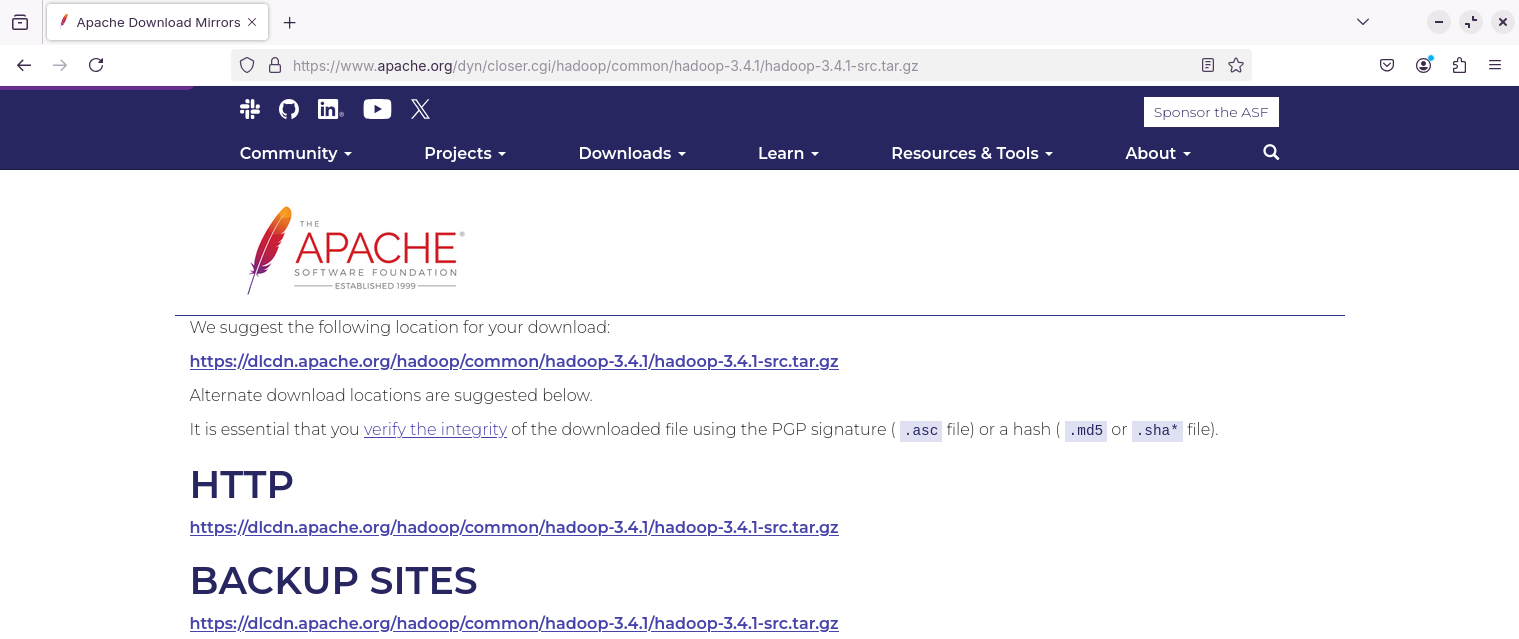




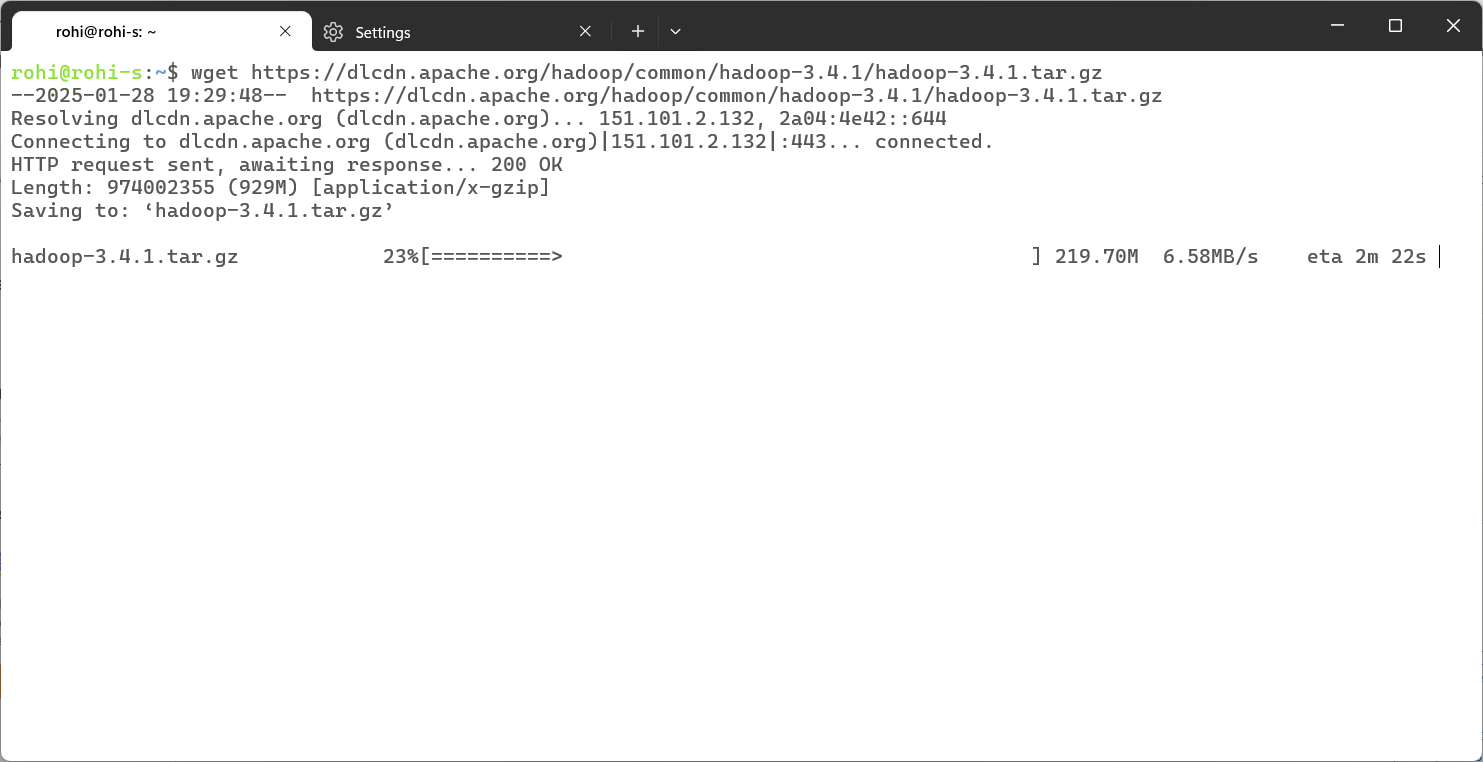


### Installing Apache Hadoop

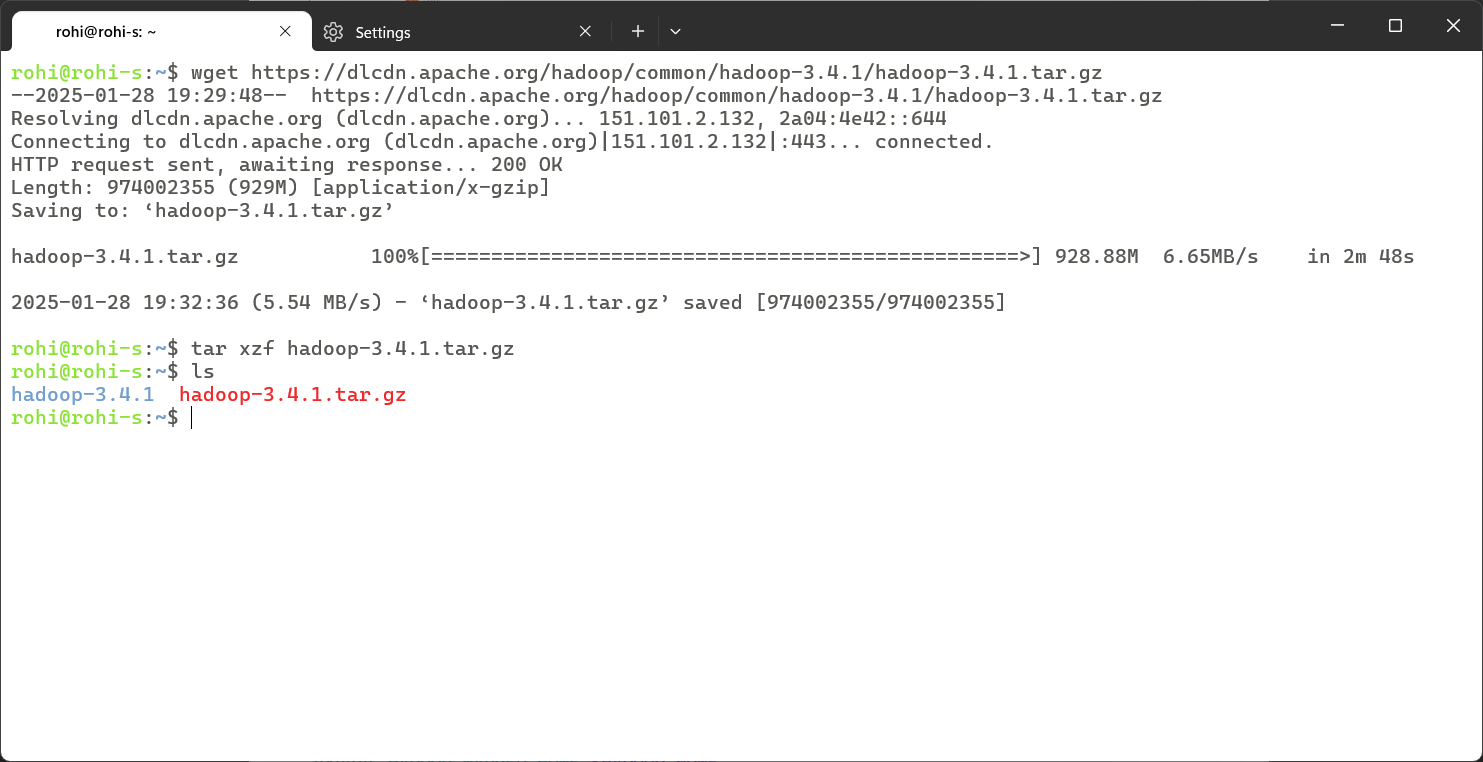






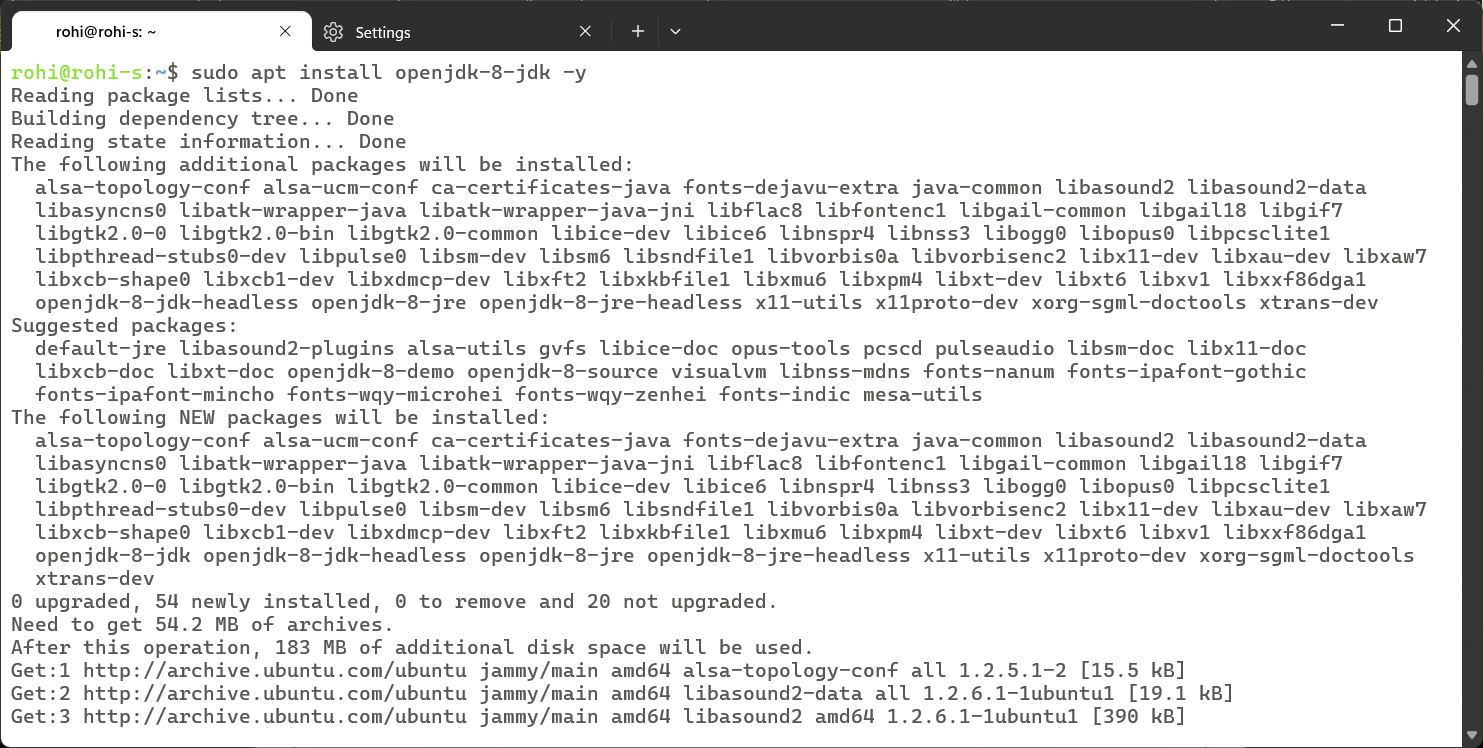




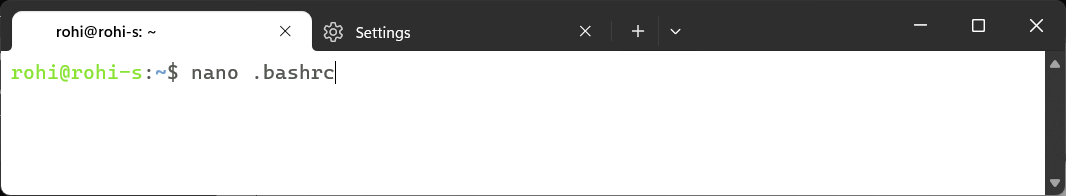


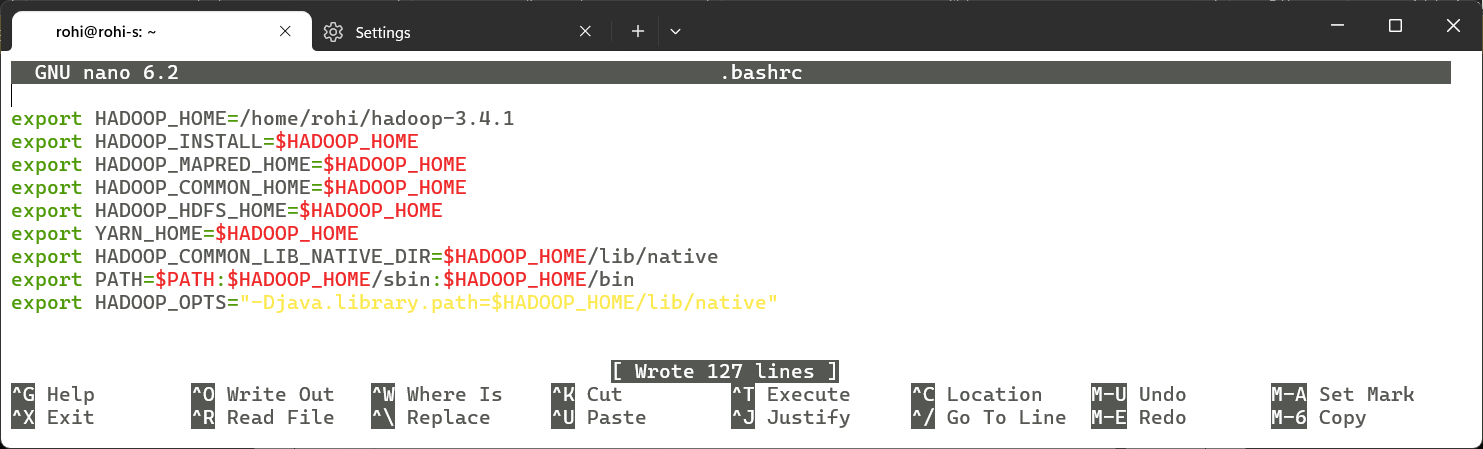
### Installing Java



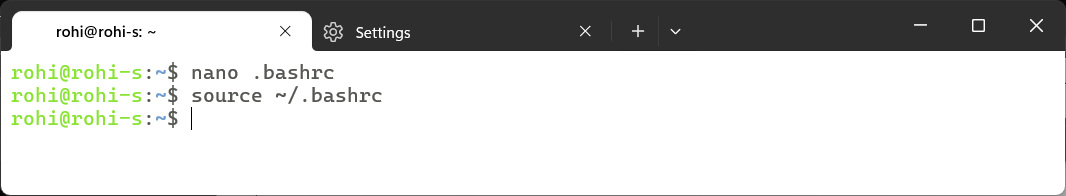


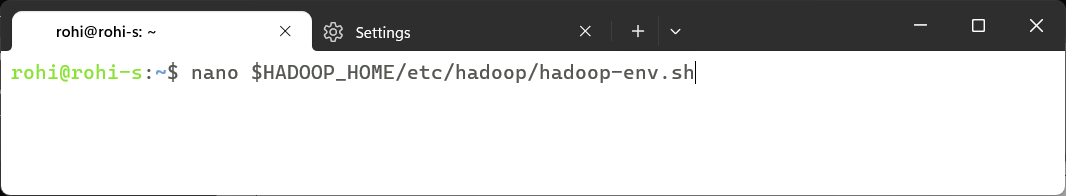
### Setting up Environment



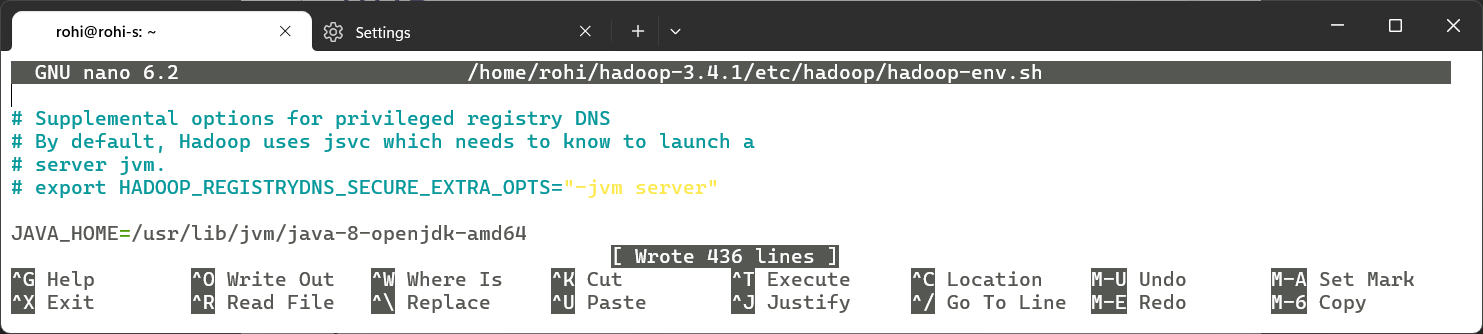






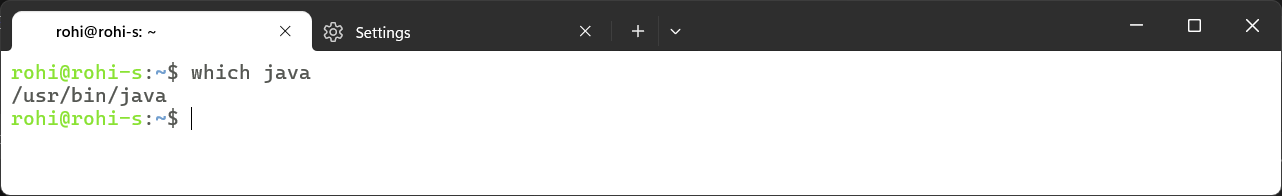




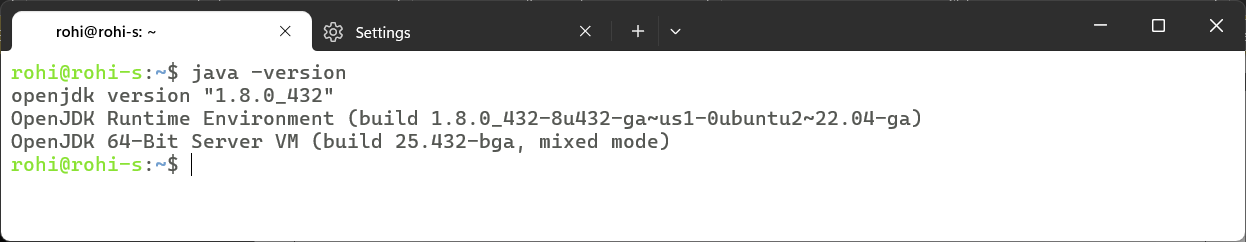


### Java path and Version



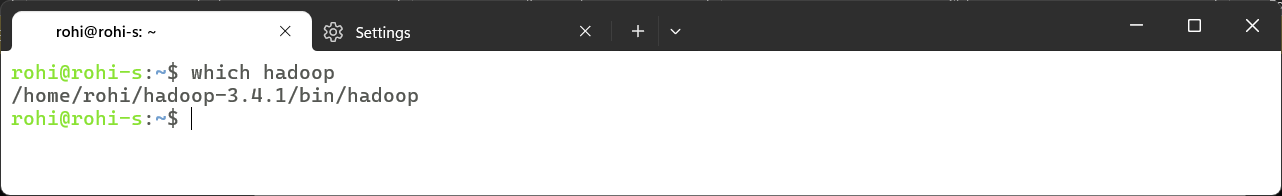


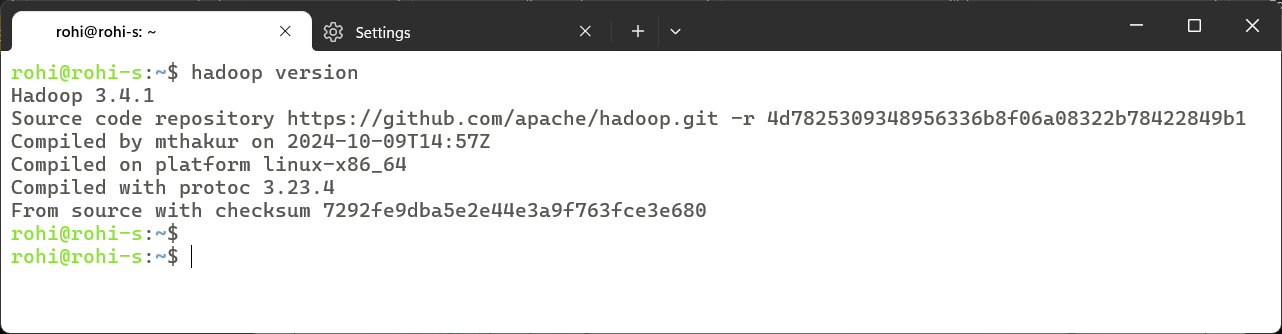




### Hadoop Path and Version







### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 2

## Experiment 2: Develop a map reduce program to calculate the frequency of a given word in a given file.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 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);

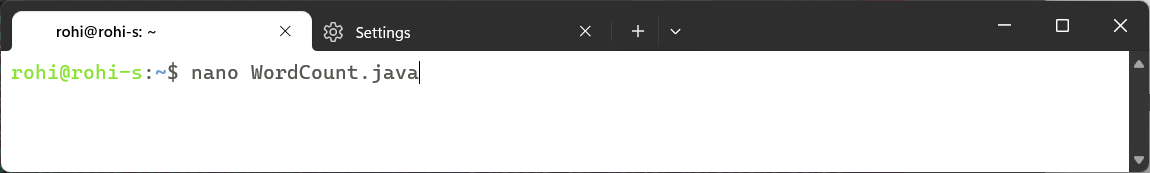
    FileInputFormat.addInputPath(job, new Path(args[0]));

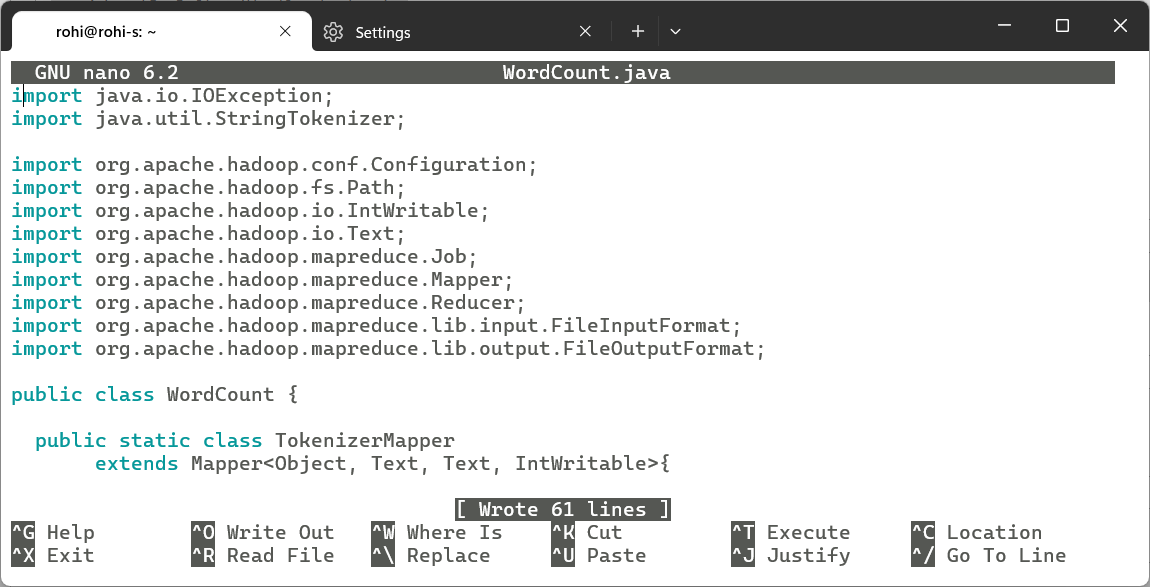
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

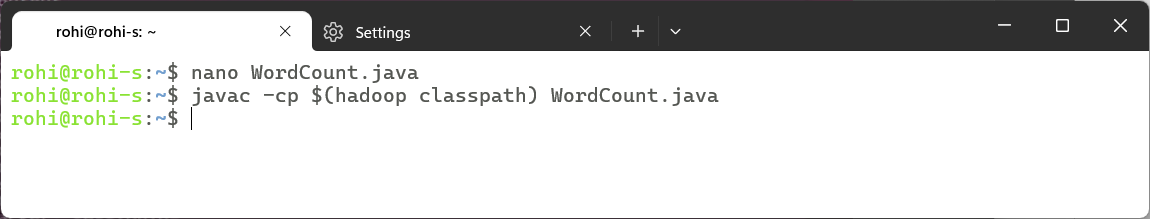
    System.exit(job.waitForCompletion(true) ? 0 : 1);

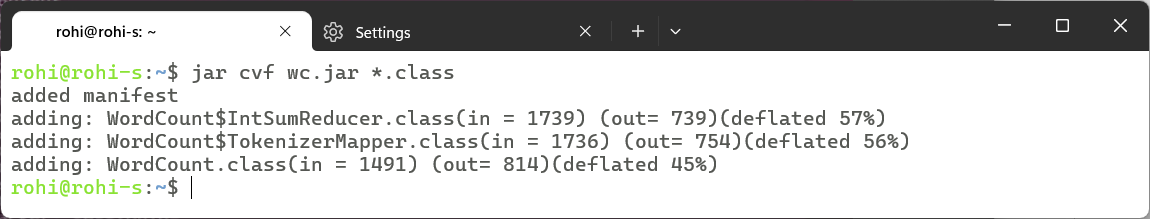
  }

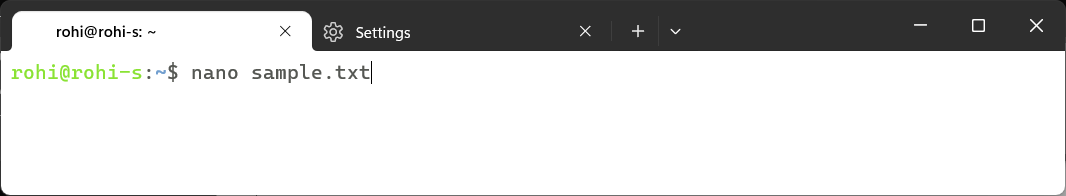
}

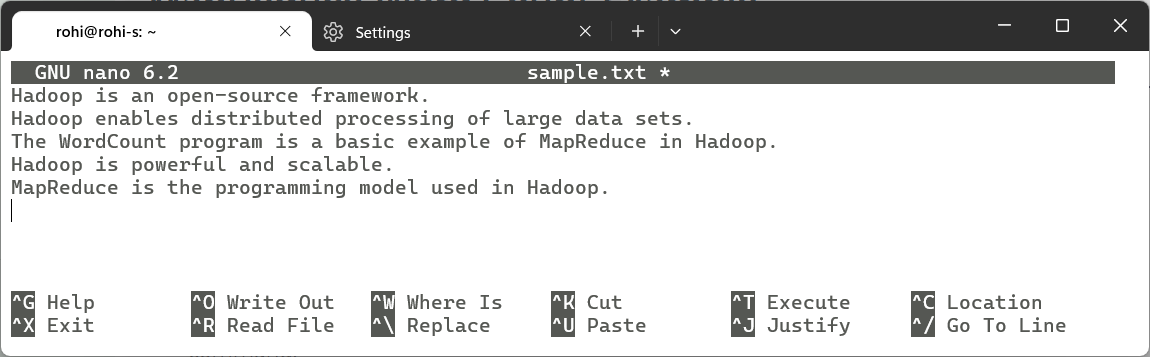


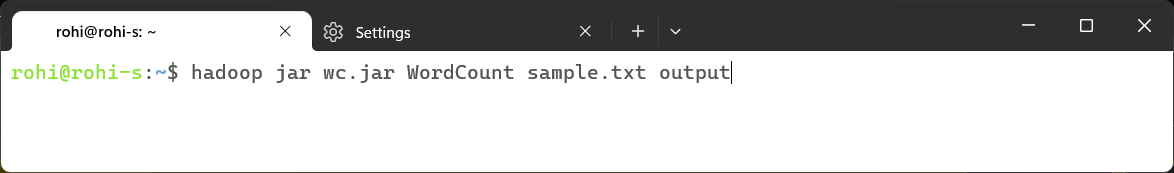


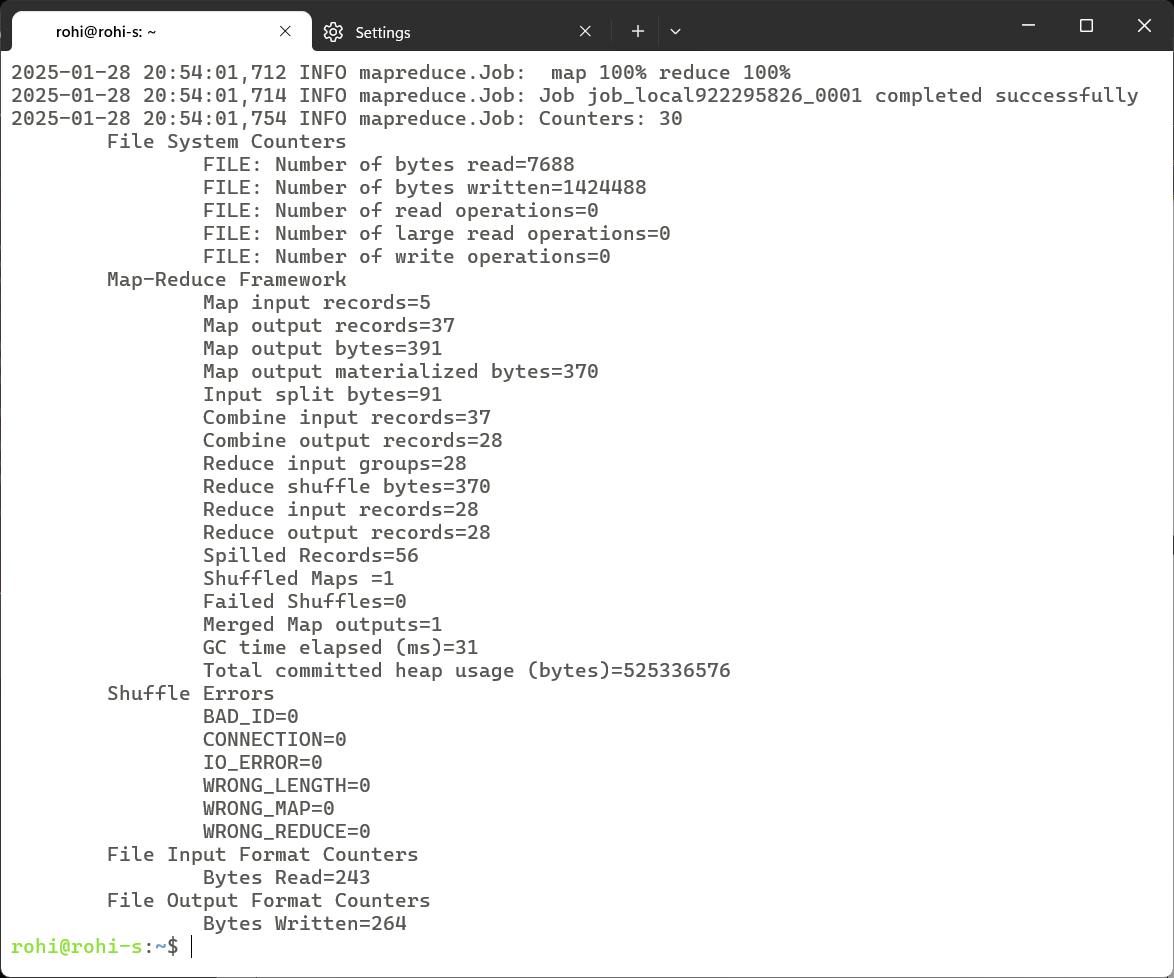














### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 3

## Experiment 3: Develop a map reduce program to find the maximum temperature in each year.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 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 Max\_temp {

    public static class TemperatureMapper

        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 {

            StringTokenizer itr = new StringTokenizer(value.toString());

            if (itr.hasMoreTokens()) {

                String yearString = itr.nextToken();

                String temperatureString = itr.nextToken();

                try {

                    year.set(yearString);

                    temperature.set(Integer.parseInt(temperatureString));

                    context.write(year, temperature);

                } catch (NumberFormatException e) {

                    System.err.println("Skipping invalid record: " +

                    value.toString());

                }

            }

        }

    }

    public static class MaxTemperatureReducer

        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 maxTemp = Integer.MIN\_VALUE;

            for (IntWritable val : values) {

                maxTemp = Math.max(maxTemp, val.get());

            }

            result.set(maxTemp);

            context.write(key, result);

        }

    }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Maximum Temperature");

        job.setJarByClass(Max\_temp.class);

        job.setMapperClass(TemperatureMapper.class);

        job.setReducerClass(MaxTemperatureReducer.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);

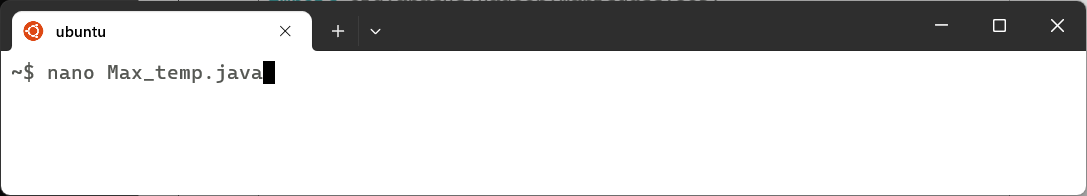
        FileInputFormat.addInputPath(job, new Path(args[0]));

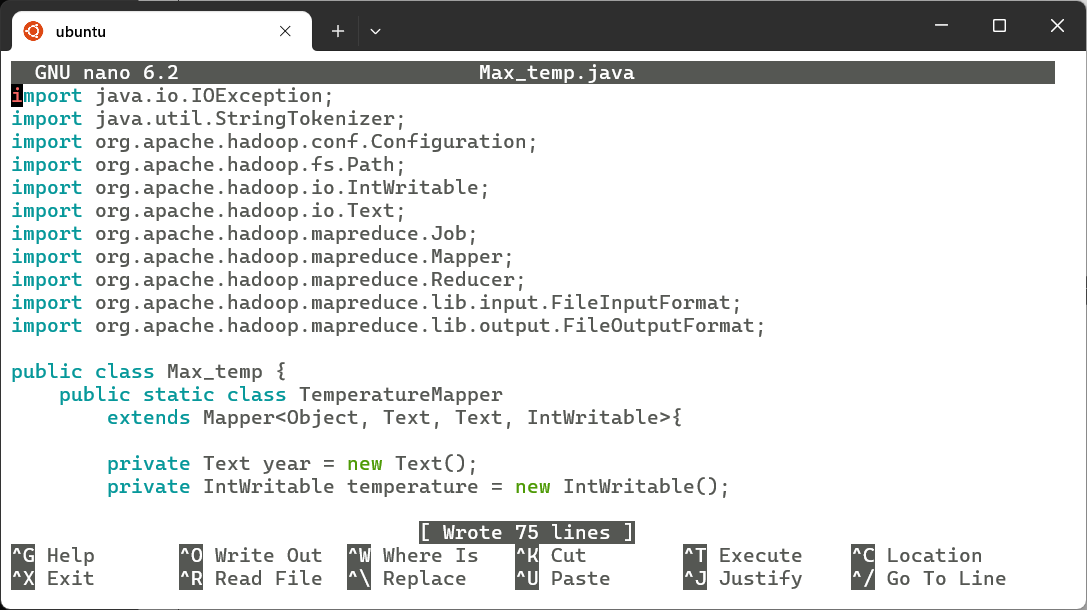
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);

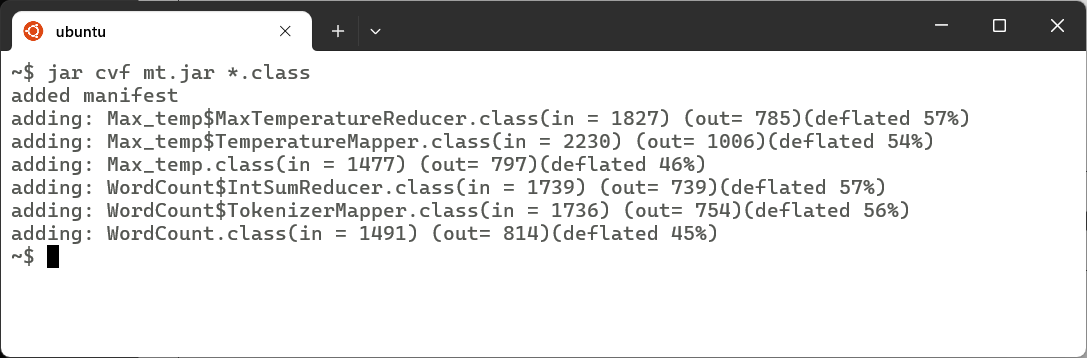
    }

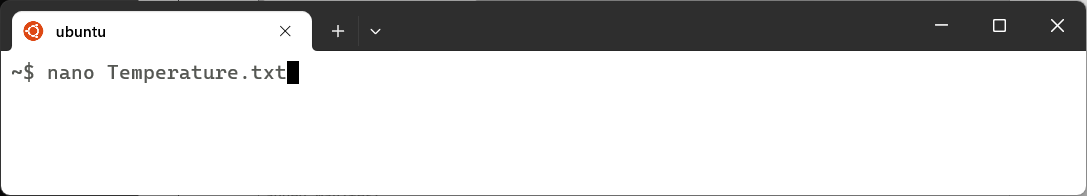
}

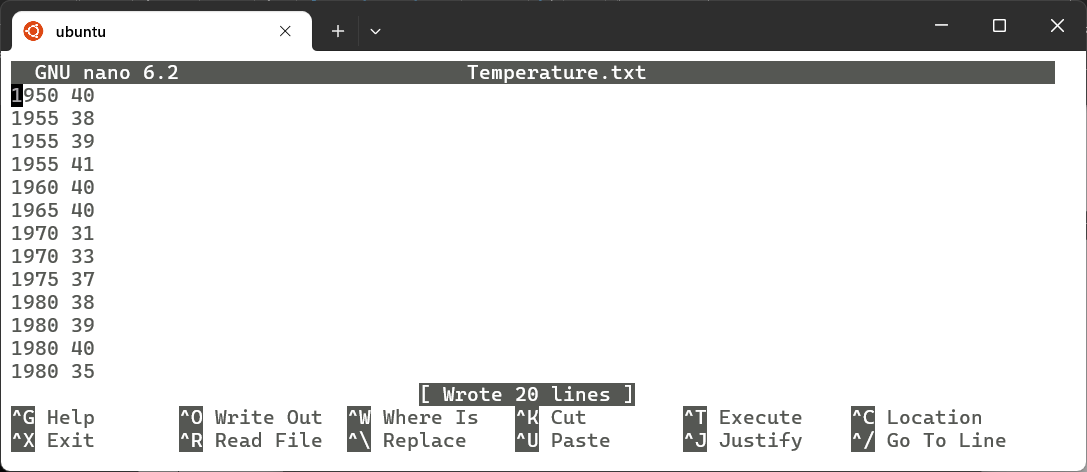


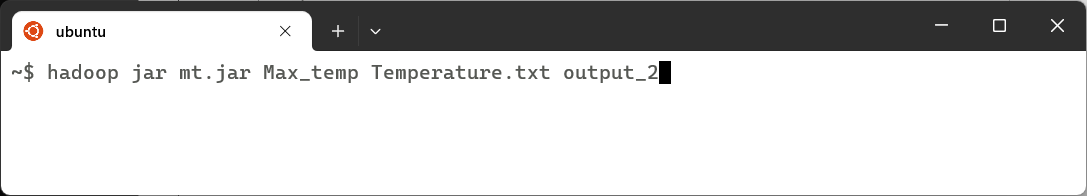


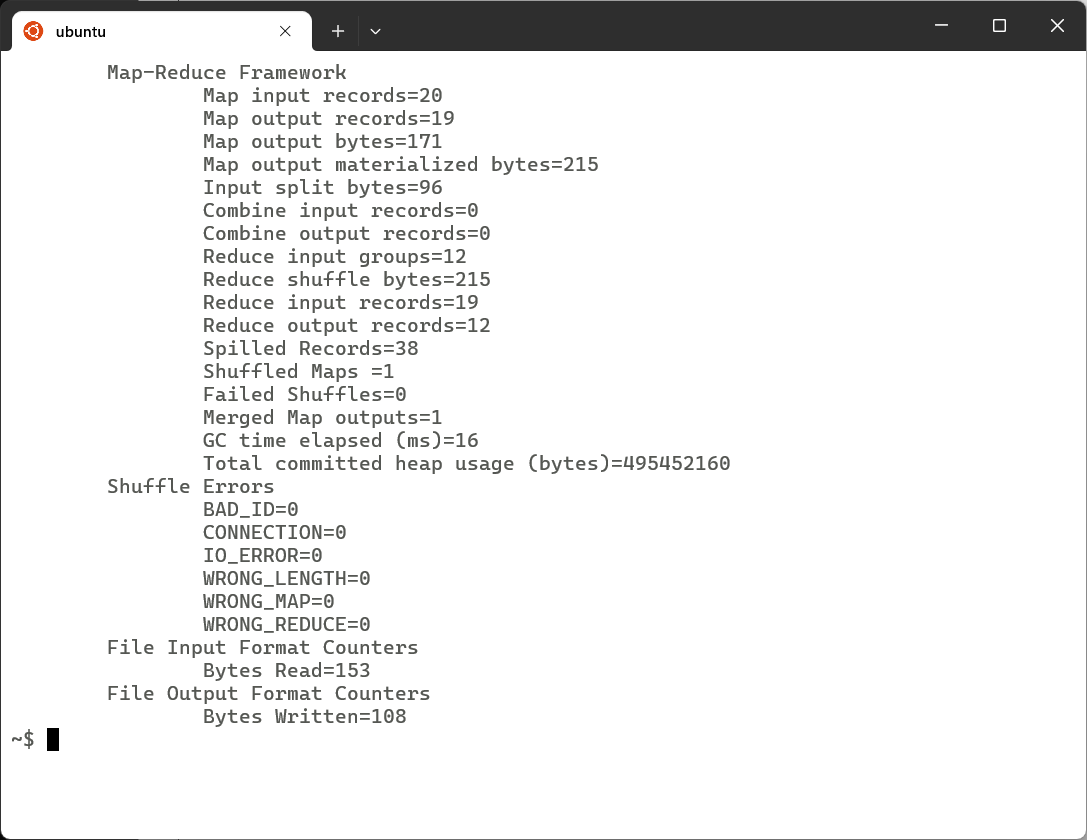














### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 4

## Experiment 4: Develop a map reduce program to find the grade of students.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 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.Text;

import org.apache.hadoop.io.IntWritable;

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 StudentGrades {

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

        private Text studentName = new Text();

        private Text grade = new Text();

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

            StringTokenizer itr = new StringTokenizer(value.toString());

            if (itr.hasMoreTokens()) {

                String name = itr.nextToken();  // Student name

                String marksStr = itr.nextToken();  // Marks

                try {

                    int marks = Integer.parseInt(marksStr);

                    String gradeLetter;

                    if (marks >= 90) {

                        gradeLetter = "A";

                    } else if (marks >= 80) {

                        gradeLetter = "B";

                    } else if (marks >= 70) {

                        gradeLetter = "C";

                    } else if (marks >= 60) {

                        gradeLetter = "D";

                    } else {

                        gradeLetter = "F";

                    }

                    studentName.set(name);

                    grade.set(gradeLetter);

                    context.write(studentName, grade);

                } catch (NumberFormatException e) {

                    System.err.println("Skipping invalid record: " + value.toString());

                }

            }

        }

    }

    public static class GradeReducer 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); // Output (StudentName -> Grade)

            }

        }

    }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Student Grades Calculation");

        job.setJarByClass(StudentGrades.class);

        job.setMapperClass(GradeMapper.class);

        job.setReducerClass(GradeReducer.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(Text.class);

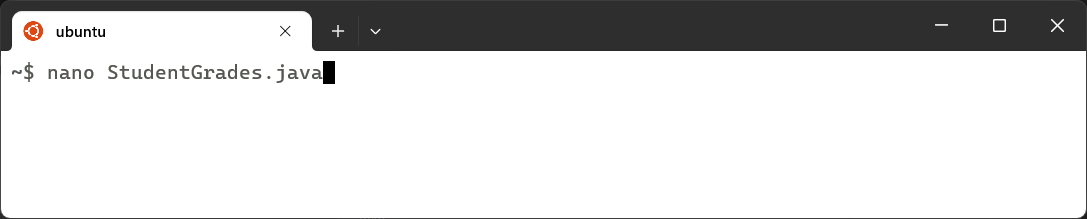
        FileInputFormat.addInputPath(job, new Path(args[0]));

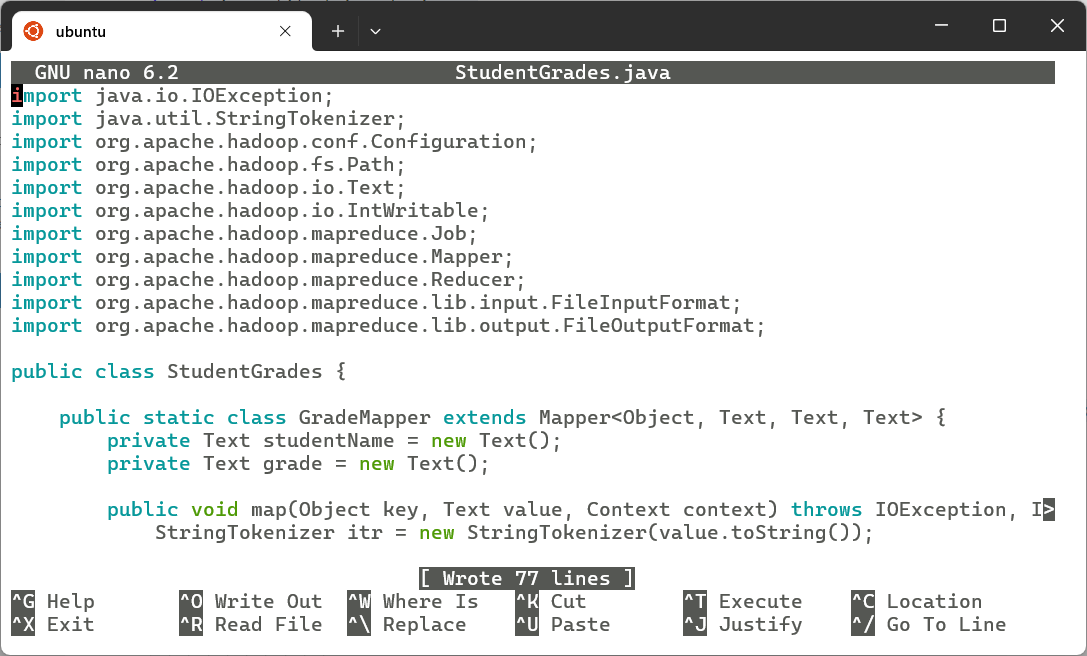
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);

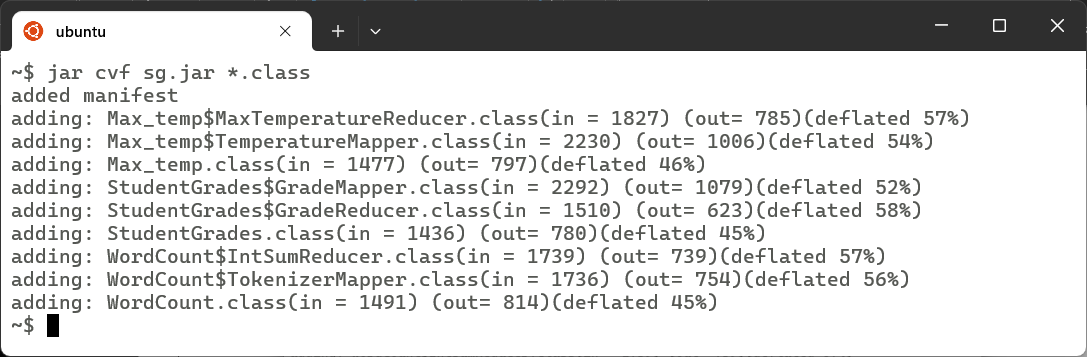
    }

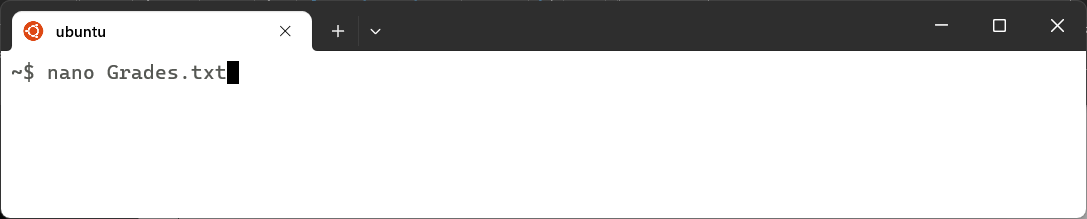
}

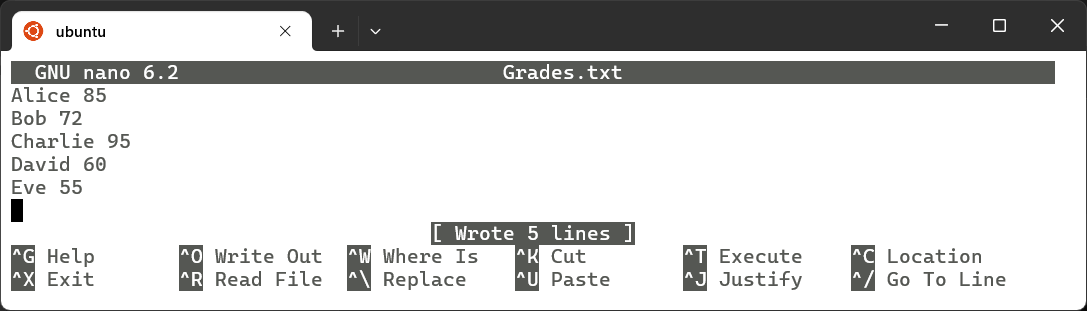


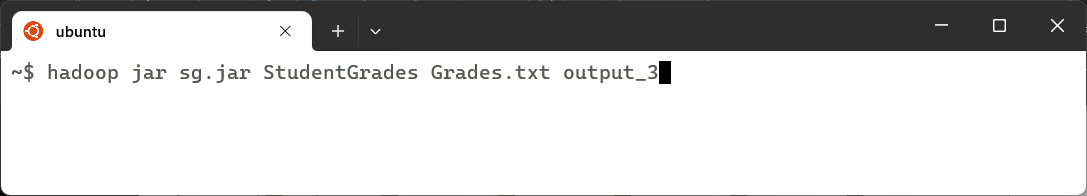


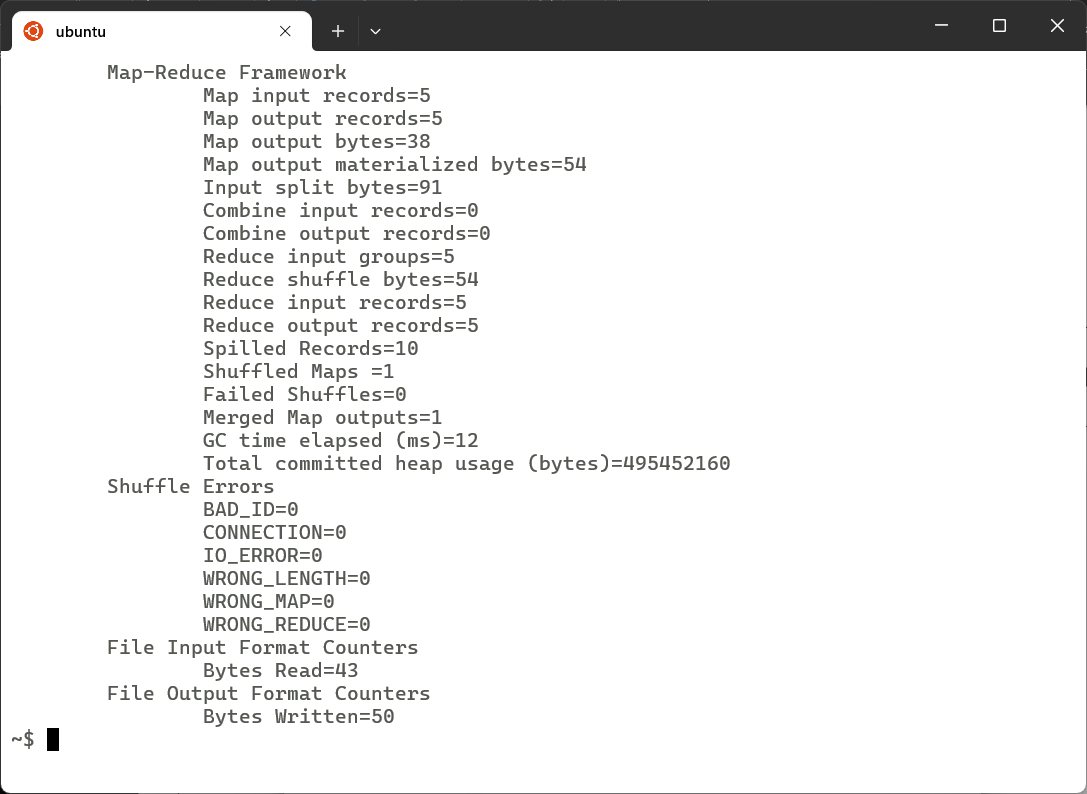


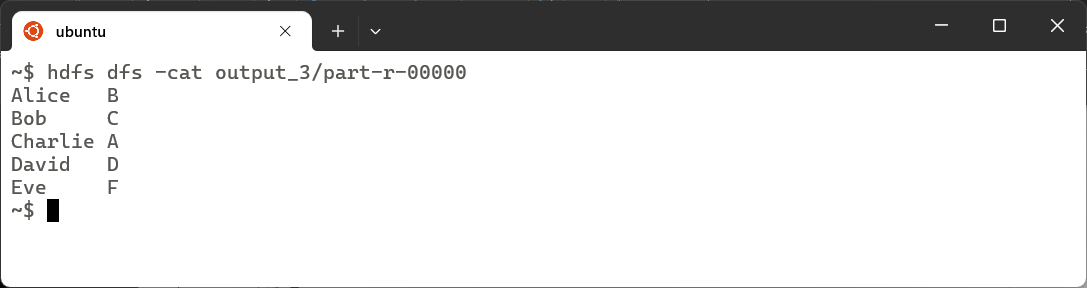












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 5

## Experiment 5: Develop a map reduce program to implement matrix multiplication.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Code:

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.io.DoubleWritable;

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.input.TextInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.input.FileSplit;

import java.util.HashMap;

import java.util.Map;

public class MatrixMultiplication {

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

        private Text outputKey = new Text();

        private Text outputValue = new Text();

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

            String[] tokens = value.toString().split("\\s+");

            if (tokens.length != 3) return; // Ensure valid format

            String fileName = ((FileSplit) context.getInputSplit()).getPath().getName();

            int row = Integer.parseInt(tokens[0]);

            int col = Integer.parseInt(tokens[1]);

            double val = Double.parseDouble(tokens[2]);

            if (fileName.contains("matrixA")) {

                for (int k = 0; k < 2; k++) { // Assuming 2x2 matrix

                    outputKey.set(row + "," + k);

                    outputValue.set("A," + col + "," + val);

                    context.write(outputKey, outputValue);

                }

            } else if (fileName.contains("matrixB")) {

                for (int i = 0; i < 2; i++) { // Assuming 2x2 matrix

                    outputKey.set(i + "," + col);

                    outputValue.set("B," + row + "," + val);

                    context.write(outputKey, outputValue);

                }

            }

        }

    }

    public static class MatrixReducer extends Reducer<Text, Text, Text, DoubleWritable> {

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

            Map<Integer, Double> matrixA = new HashMap<>();

            Map<Integer, Double> matrixB = new HashMap<>();

            for (Text val : values) {

                String[] parts = val.toString().split(",");

                if (parts[0].equals("A")) {

                    matrixA.put(Integer.parseInt(parts[1]), Double.parseDouble(parts[2]));

                } else if (parts[0].equals("B")) {

                    matrixB.put(Integer.parseInt(parts[1]), Double.parseDouble(parts[2]));

                }

            }

            double sum = 0.0;

            for (int k : matrixA.keySet()) {

                if (matrixB.containsKey(k)) {

                    sum += matrixA.get(k) \* matrixB.get(k);

                }

            }

            context.write(key, new DoubleWritable(sum));

        }

    }

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

        if (args.length < 2) {

            System.err.println("Usage: MatrixMultiplication <Input Directory> <Output Directory>");

            System.exit(1);

        }

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Matrix Multiplication");

        job.setJarByClass(MatrixMultiplication.class);

        job.setMapperClass(MatrixMapper.class);

        job.setReducerClass(MatrixReducer.class);

        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(DoubleWritable.class);

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

        job.setInputFormatClass(TextInputFormat.class);

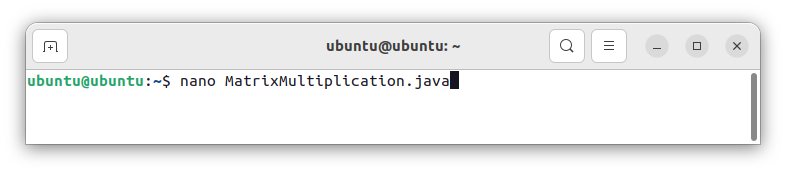
        FileOutputFormat.setOutputPath(job, new Path(args[1

        System.exit(job.waitForCompletion(true) ? 0 : 1);

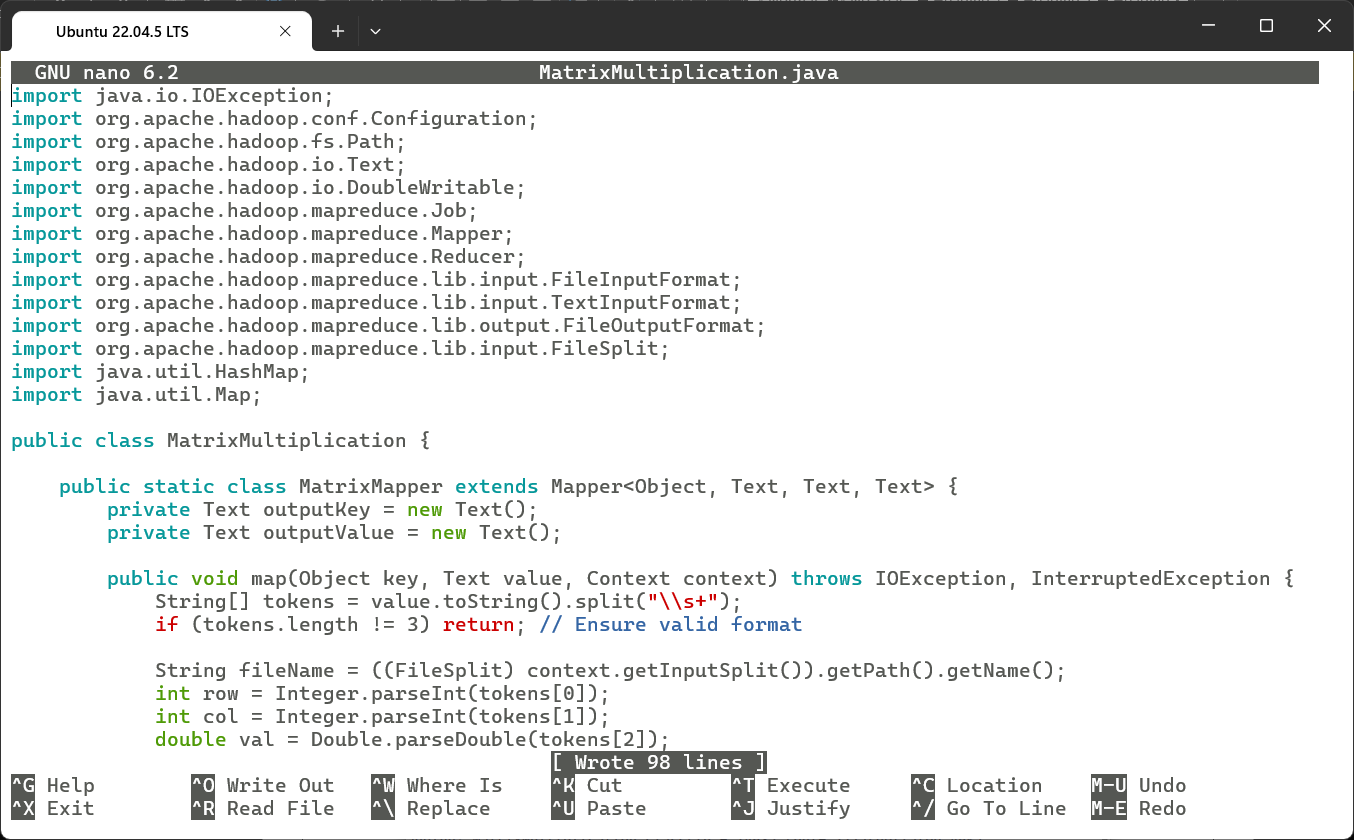
    }

}

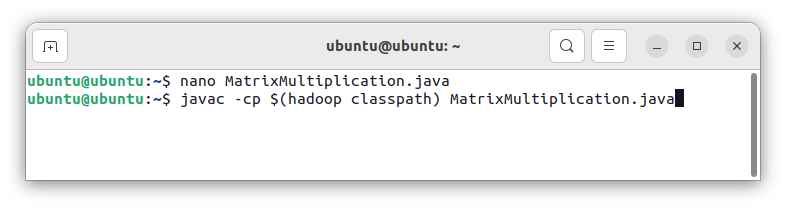




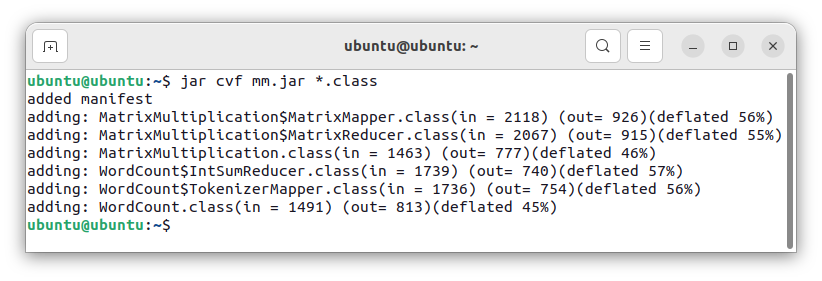




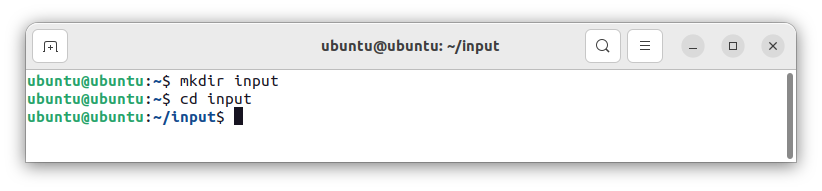




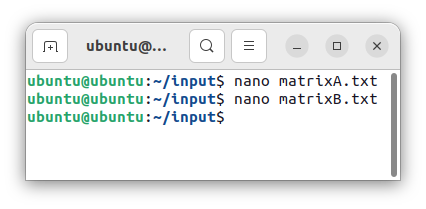




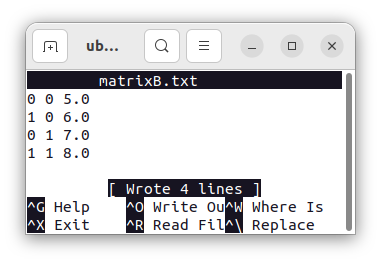
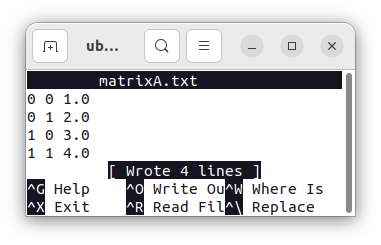




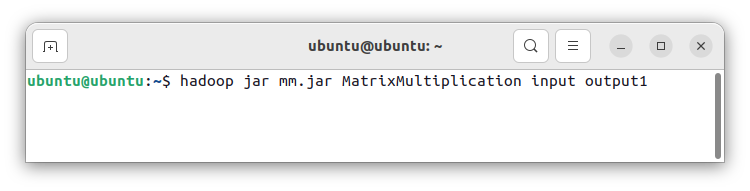




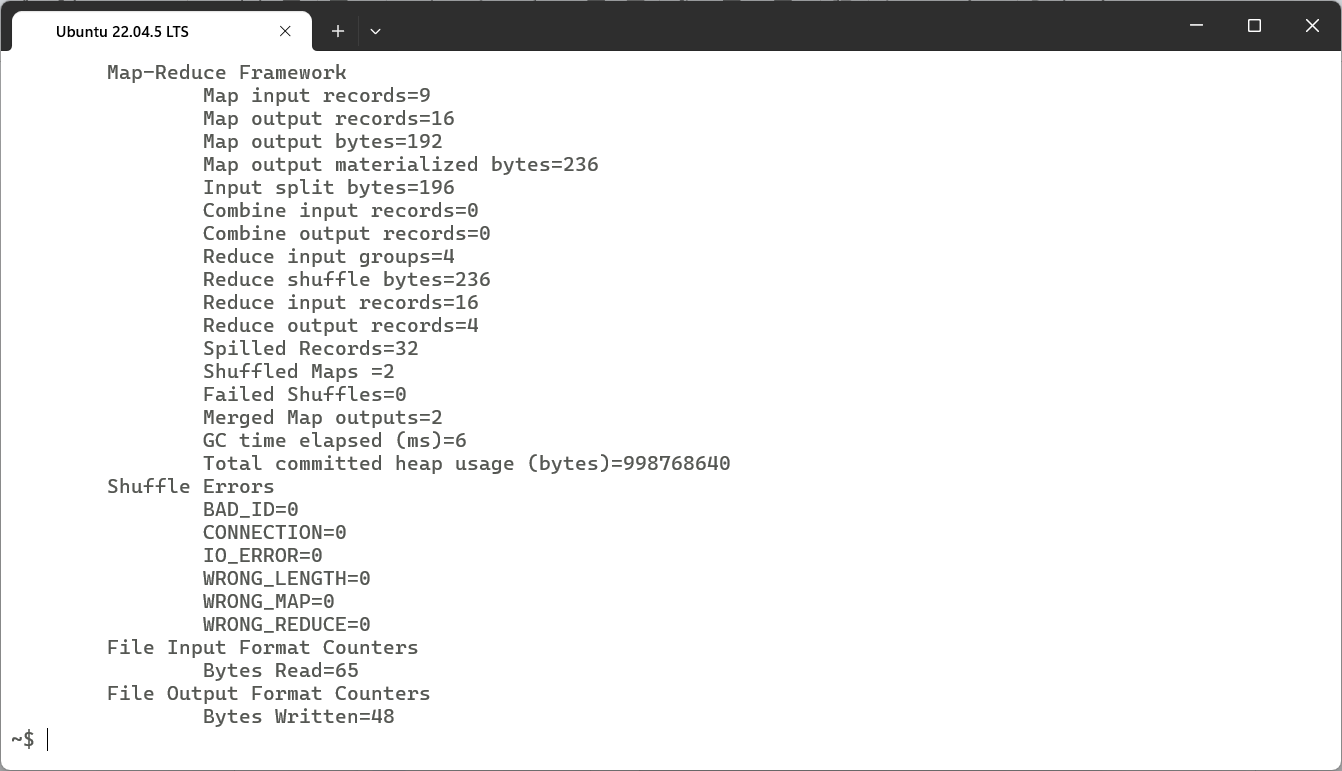




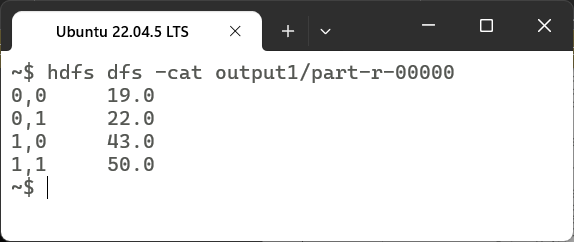












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |

# Experiment 6

## Experiment 6: Develop a map reduce program to find the maximum electrical consumption in each year given electrical consumption for each month in each year.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 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 MaxElectricalConsumption {

    // Mapper class to extract month and consumption data

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

        private Text month = new Text();

        private IntWritable consumption = new IntWritable();

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

            StringTokenizer itr = new StringTokenizer(value.toString());

            // Expected format: "Month Consumption"

            if (itr.hasMoreTokens()) {

                String monthString = itr.nextToken(); // Month (e.g., "January")

                String consumptionString = itr.nextToken(); // Consumption value

                try {

                    month.set(monthString);

                    consumption.set(Integer.parseInt(consumptionString)); // Parse the consumption value

                    context.write(month, consumption); // Emit (month, consumption)

                } catch (NumberFormatException e) {

                    System.err.println("Skipping invalid record: " + value.toString());

                }

            }

        }

    }

    // Reducer class to calculate the maximum consumption across all months

    public static class MaxConsumptionReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private Text resultMonth = new Text();

        private IntWritable resultConsumption = new IntWritable();

        private String maxMonth = "";

        private int maxConsumption = Integer.MIN\_VALUE;

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

            for (IntWritable val : values) {

                if (val.get() > maxConsumption) {

                    maxConsumption = val.get();

                    maxMonth = key.toString();

                }

            }

        }

        @Override

        protected void cleanup(Context context) throws IOException, InterruptedException {

            resultMonth.set(maxMonth);

            resultConsumption.set(maxConsumption);

            context.write(resultMonth, resultConsumption);

        }

    }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Maximum Electrical Consumption");

        job.setJarByClass(MaxElectricalConsumption.class);

        job.setMapperClass(ConsumptionMapper.class);

        job.setReducerClass(MaxConsumptionReducer.class);

        job.setOutputKeyClass(Text.class); // Key: month

        job.setOutputValueClass(IntWritable.class); // Value: consumption

        // Set the input and output paths

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

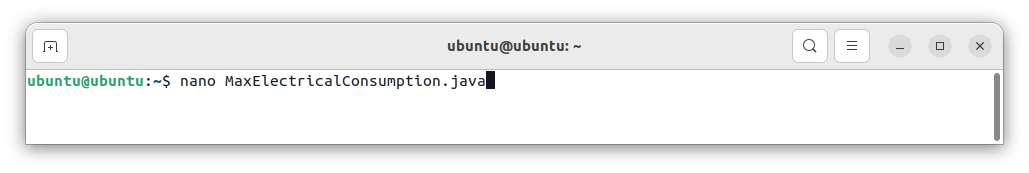
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

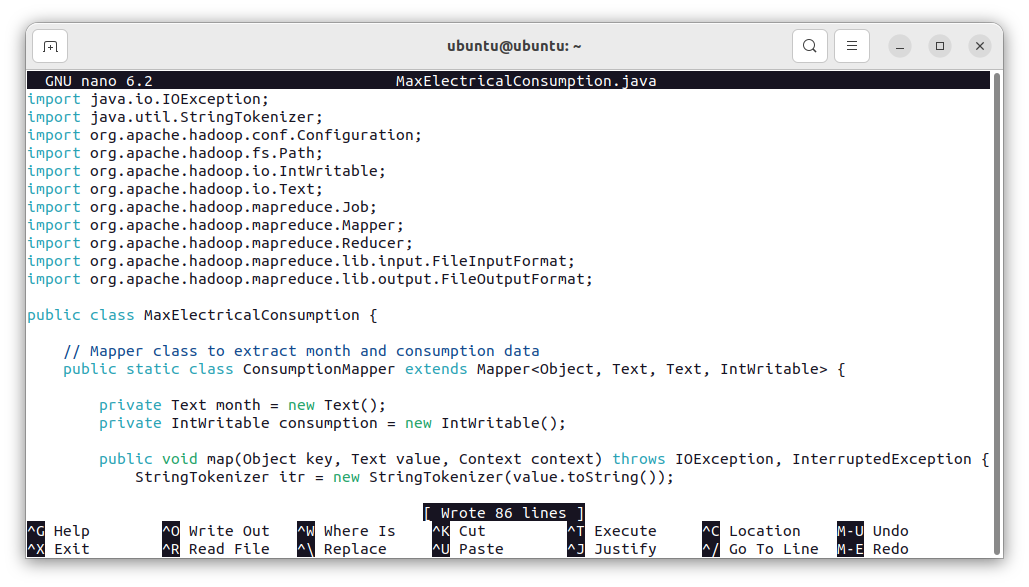
        // Wait for the job to complete

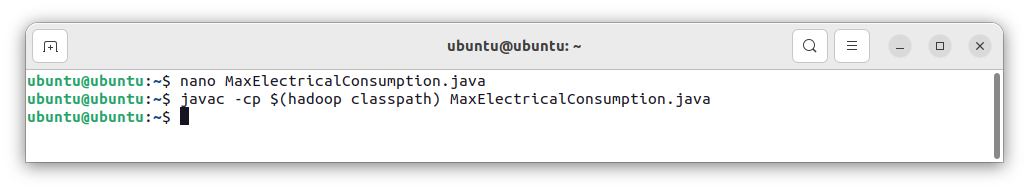
        System.exit(job.waitForCompletion(true) ? 0 : 1);

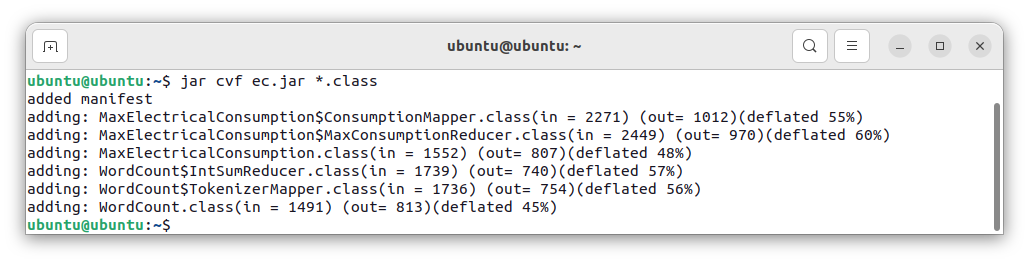
    }

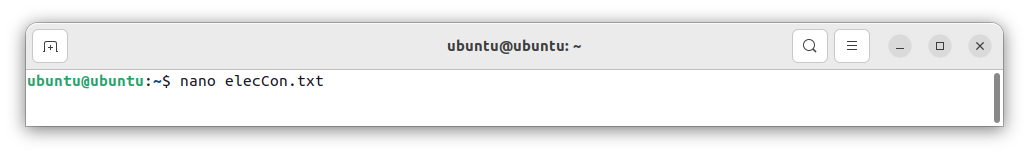
}



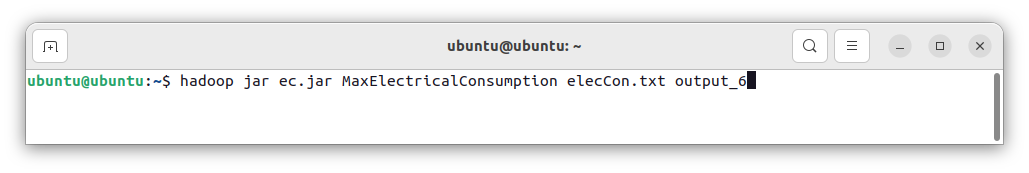


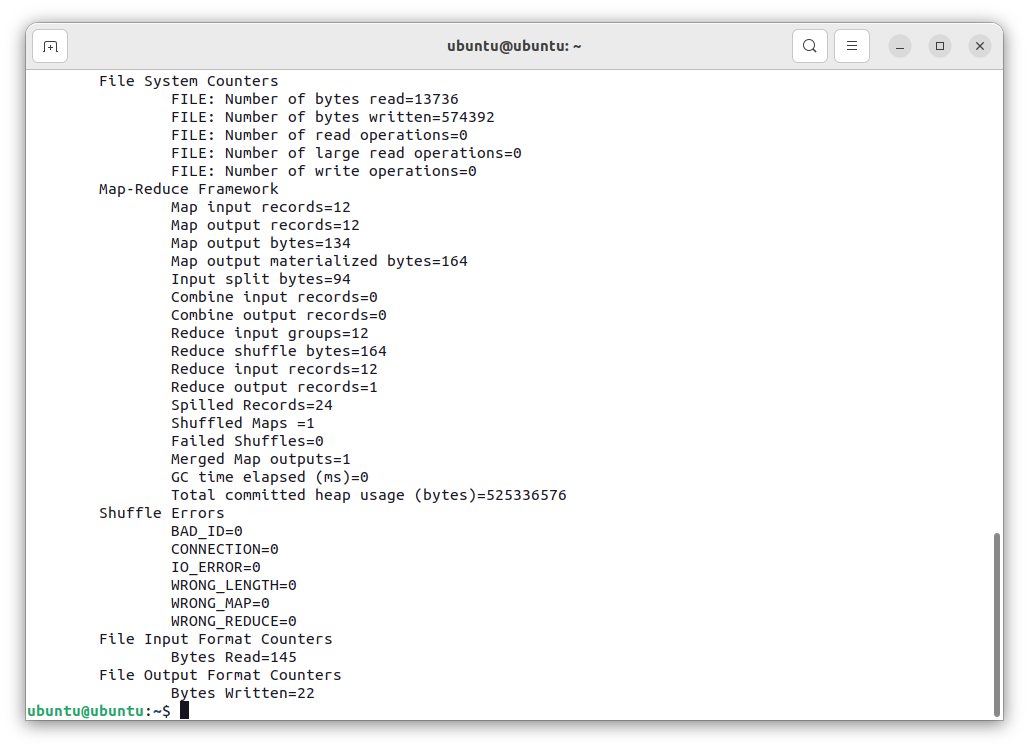


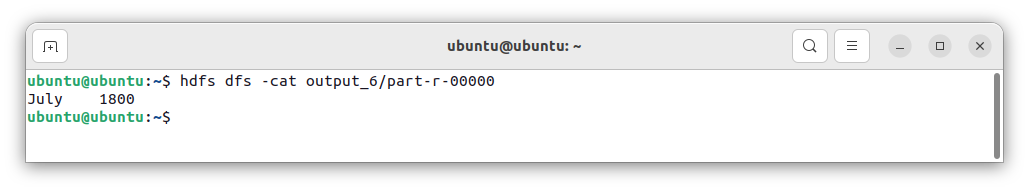












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 7

## Experiment 7: Develop a map reduce program to analyze weather data set and print whether the day is shiny or cool day.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### 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 MaxElectricalConsumption {

    // Mapper class to extract month and consumption data

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

        private Text month = new Text();

        private IntWritable consumption = new IntWritable();

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

            StringTokenizer itr = new StringTokenizer(value.toString());

            // Expected format: "Month Consumption"

            if (itr.hasMoreTokens()) {

                String monthString = itr.nextToken(); // Month (e.g., "January")

                String consumptionString = itr.nextToken(); // Consumption value

                try {

                    month.set(monthString);

                    consumption.set(Integer.parseInt(consumptionString)); // Parse the consumption value

                    context.write(month, consumption); // Emit (month, consumption)

                } catch (NumberFormatException e) {

                    System.err.println("Skipping invalid record: " + value.toString());

                }

            }

        }

    }

    // Reducer class to calculate the maximum consumption across all months

    public static class MaxConsumptionReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        private Text resultMonth = new Text();

        private IntWritable resultConsumption = new IntWritable();

        private String maxMonth = "";

        private int maxConsumption = Integer.MIN\_VALUE;

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

            for (IntWritable val : values) {

                if (val.get() > maxConsumption) {

                    maxConsumption = val.get();

                    maxMonth = key.toString();

                }

            }

        }

        @Override

        protected void cleanup(Context context) throws IOException, InterruptedException {

            resultMonth.set(maxMonth);

            resultConsumption.set(maxConsumption);

            context.write(resultMonth, resultConsumption);

        }

    }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Maximum Electrical Consumption");

        job.setJarByClass(MaxElectricalConsumption.class);

        job.setMapperClass(ConsumptionMapper.class);

        job.setReducerClass(MaxConsumptionReducer.class);

        job.setOutputKeyClass(Text.class); // Key: month

        job.setOutputValueClass(IntWritable.class); // Value: consumption

        // Set the input and output paths

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

        FileOutputFormat.setOutputPath(job, new Path(args[1]));

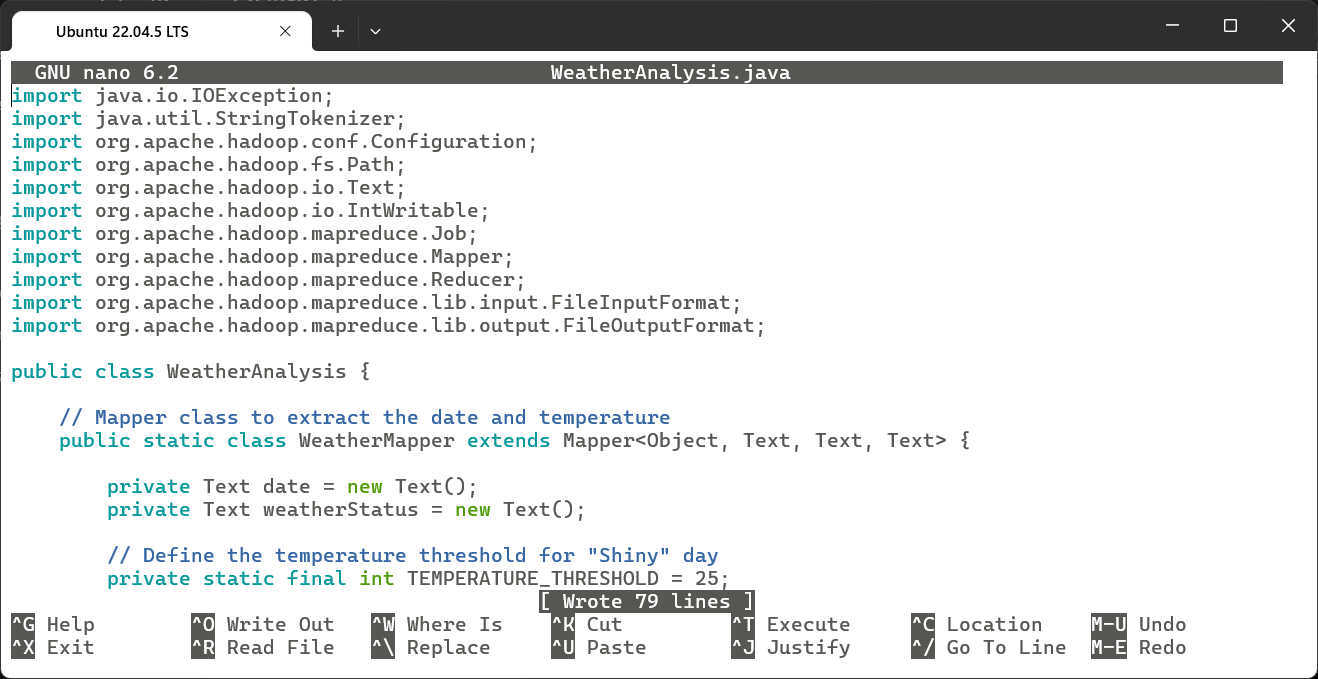
        // Wait for the job to complete

        System.exit(job.waitForCompletion(true) ? 0 : 1);

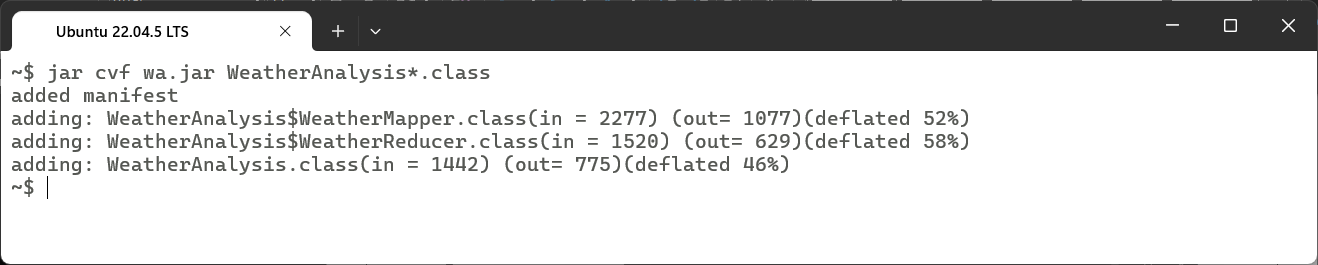
    }

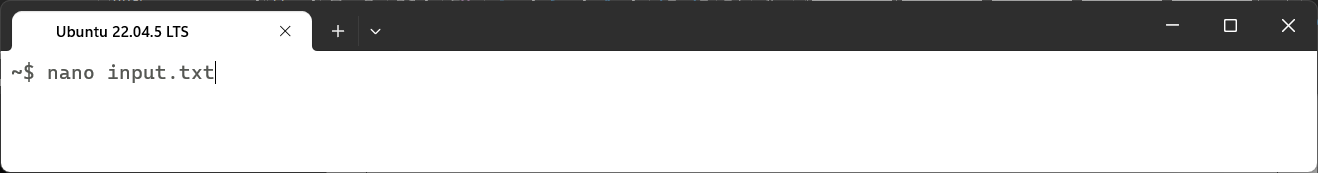
}

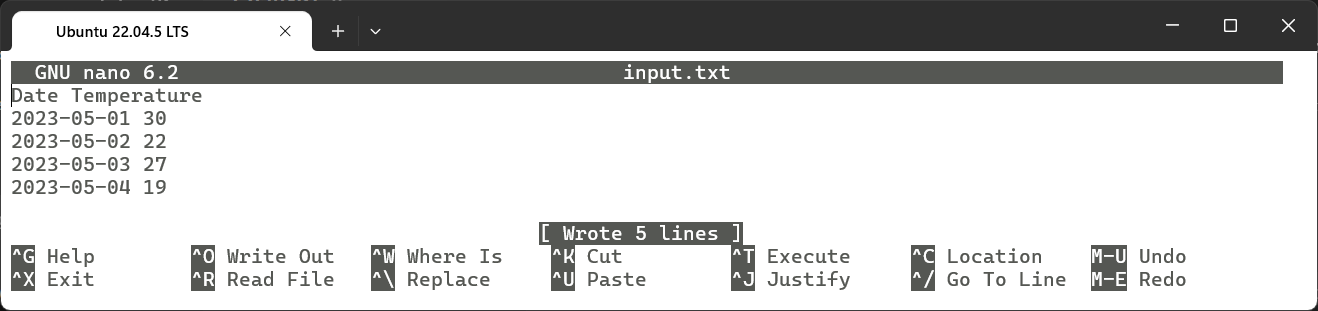


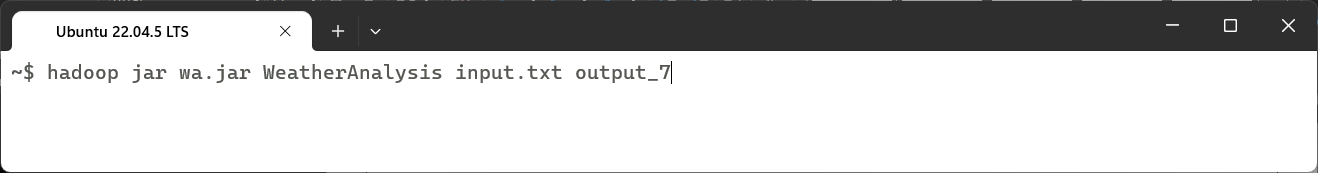


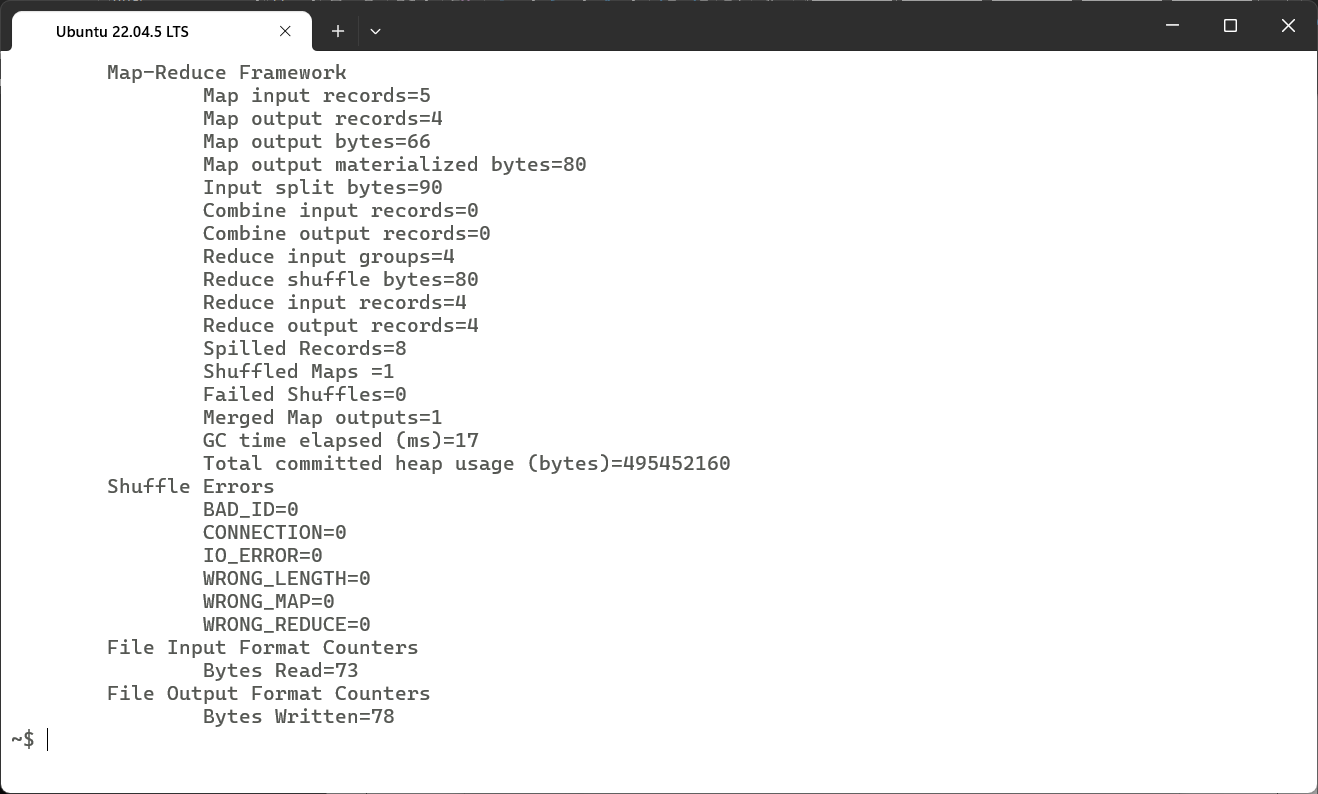


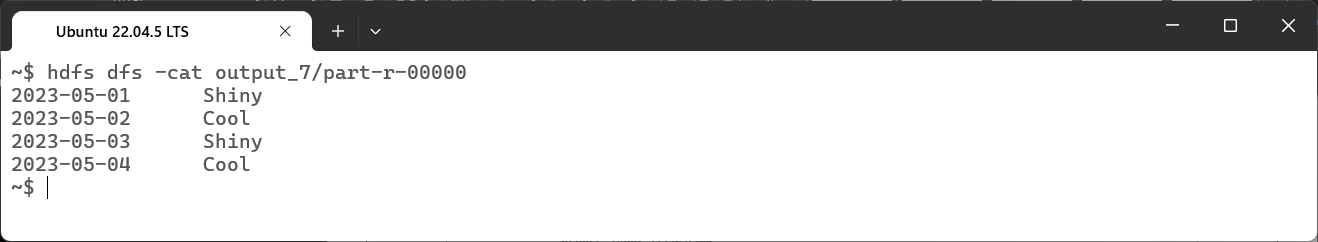












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 8

## Experiment 8: Develop a map reduce program to find the tags associated with each movie by analyzing movie lens data.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Code:

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

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;

public class MovieTagsAnalysis {

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

        private Text movieID = new Text();

        private Text tag = new Text();

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

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

            if (fields.length >= 3) {  // Ensuring proper data format

                movieID.set(fields[1]);  // Movie ID is in the second column

                tag.set(fields[2]);  // Tag is in the third column

                context.write(movieID, tag);

            }

        }

    }

    public static class TagsReducer extends Reducer<Text, Text, Text, Text> {

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

            List<String> tagsList = new ArrayList<>();

            for (Text val : values) {

                tagsList.add(val.toString());

            }

            context.write(key, new Text(String.join(", ", tagsList)));

        }

    }

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

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Movie Tags Analysis");

        job.setJarByClass(MovieTagsAnalysis.class);

        job.setMapperClass(TagsMapper.class);

        job.setReducerClass(TagsReducer.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(Text.class);

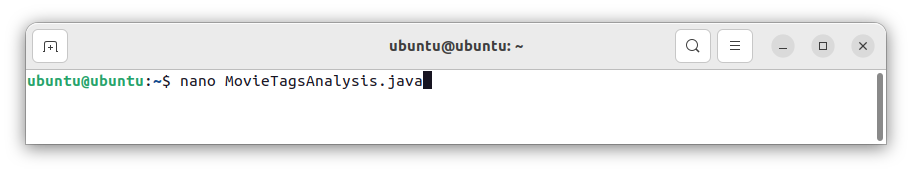
        FileInputFormat.addInputPath(job, new Path(args[0])); // Input directory (tags.csv)

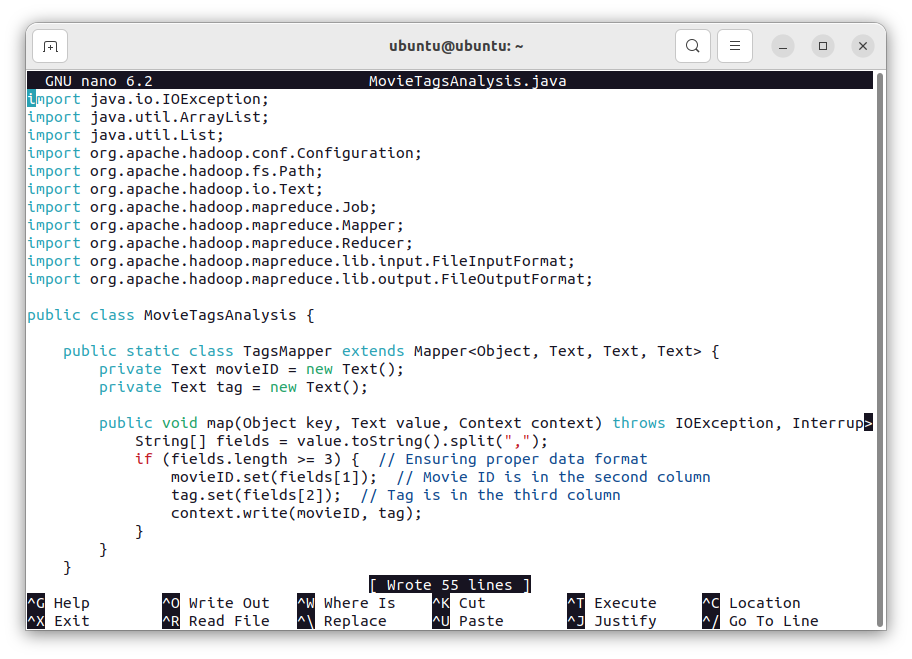
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory

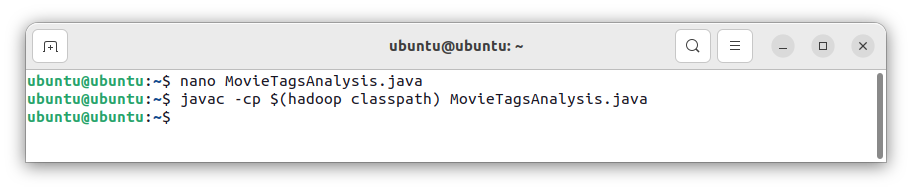
        System.exit(job.waitForCompletion(true) ? 0 : 1);

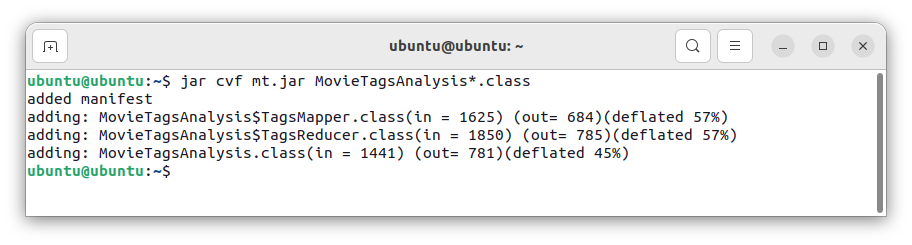
    }

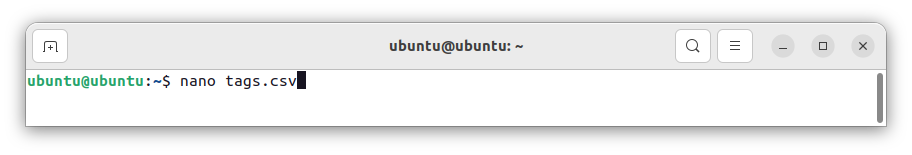
}

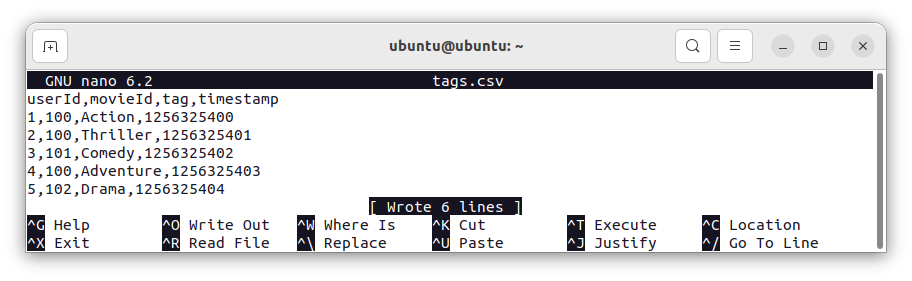


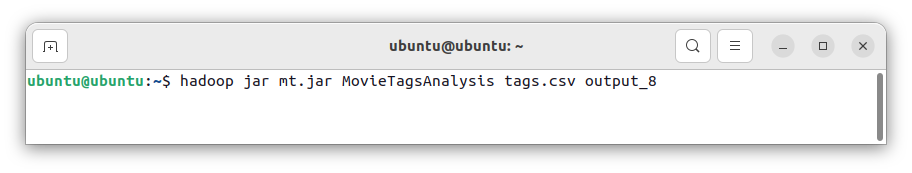


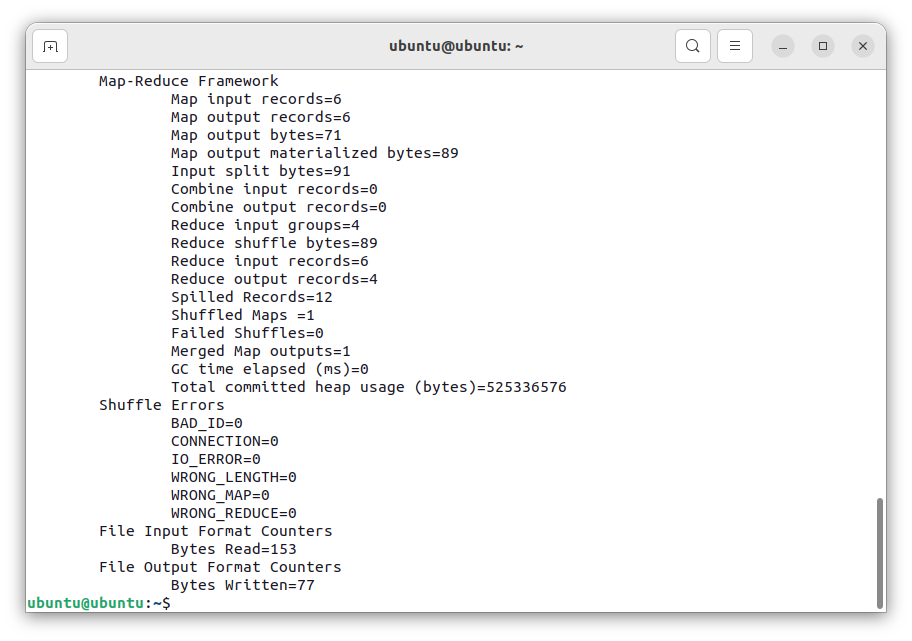


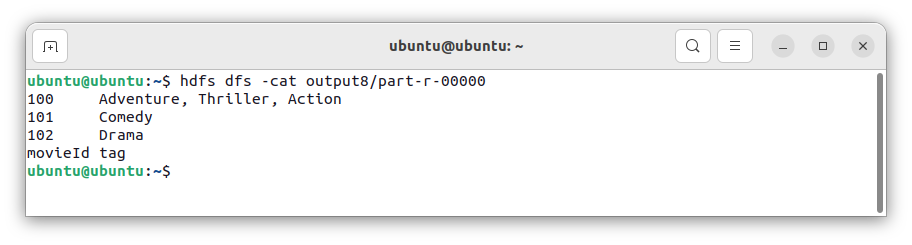












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 9

## Experiment 9: Develop a map reduce program to analyze Uber data set to find the days on which each basement has more trips using the following data set. The uber data set consists of four columns they are: Dispatching base number, date, active vehicle, trips.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Code:

import java.io.IOException;

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;

public class UberTripsAnalysis {

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

        private Text baseId = new Text();

        private Text dateAndTrips = new Text();

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

            String[] tokens = value.toString().split("\\s+");  // Split by spaces

            if (tokens.length == 4) {  // Ensure the correct number of columns

                String base = tokens[0];      // Base ID

                String date = tokens[1];      // Date

                String trips = tokens[3];     // Number of trips

                baseId.set(base);

                dateAndTrips.set(date + "," + trips);

                context.write(baseId, dateAndTrips);

            }

        }

    }

    public static class UberReducer extends Reducer<Text, Text, Text, Text> {

        private Text maxTripDate = new Text();

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

            int maxTripsValue = Integer.MIN\_VALUE;

            String maxDay = "";

            for (Text val : values) {

                String[] parts = val.toString().split(",");

                String date = parts[0];

                int trips = Integer.parseInt(parts[1]);

                if (trips > maxTripsValue) {

                    maxTripsValue = trips;

                    maxDay = date;

                }

            }

            maxTripDate.set(maxDay + " " + maxTripsValue);

            context.write(key, maxTripDate);

        }

    }

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

        if (args.length < 2) {

            System.err.println("Usage: UberTripsAnalysis <input file> <output dir>");

            System.exit(1);

        }

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Uber Trips Analysis");

        job.setJarByClass(UberTripsAnalysis.class);

        job.setMapperClass(UberMapper.class);

        job.setReducerClass(UberReducer.class);

        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(Text.class);

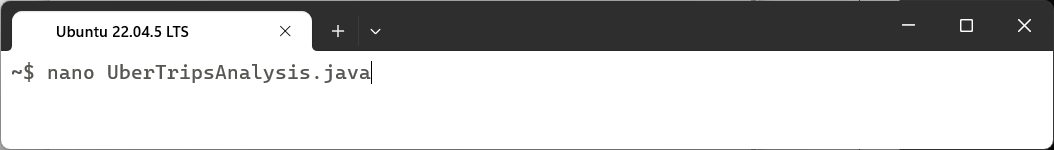
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Takes file directly

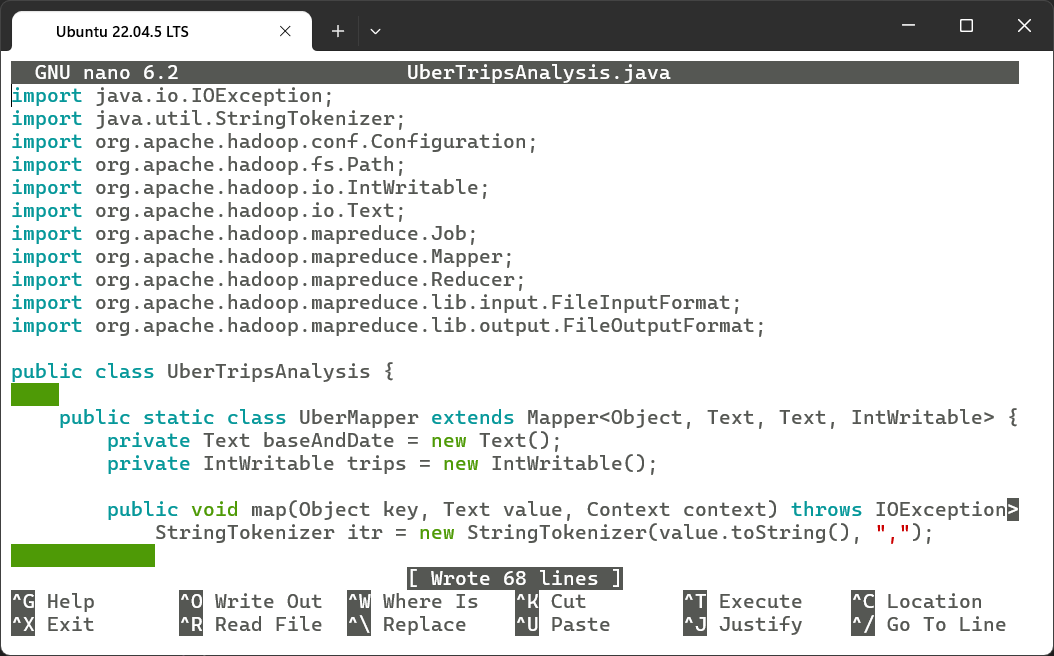
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

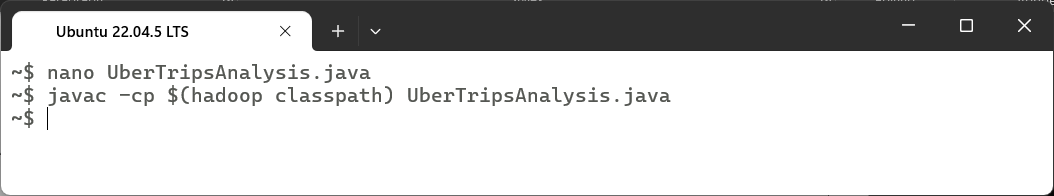
        System.exit(job.waitForCompletion(true) ? 0 : 1);

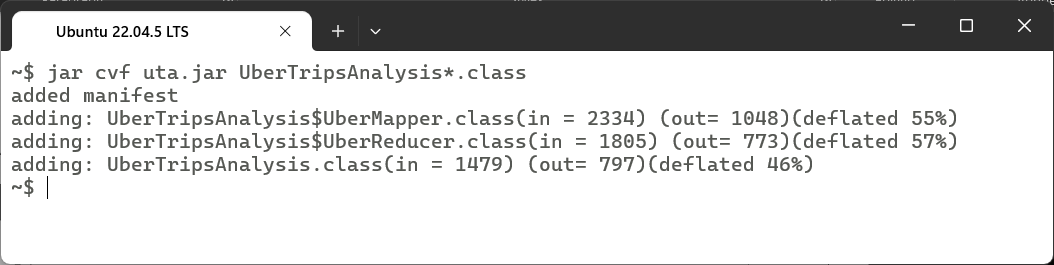
    }

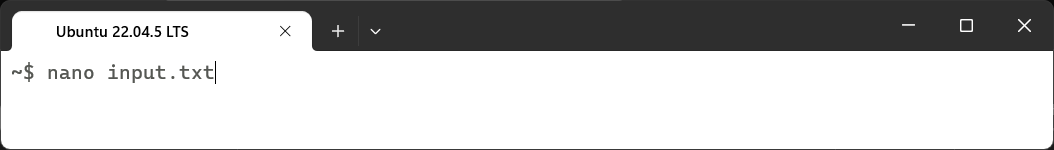
}

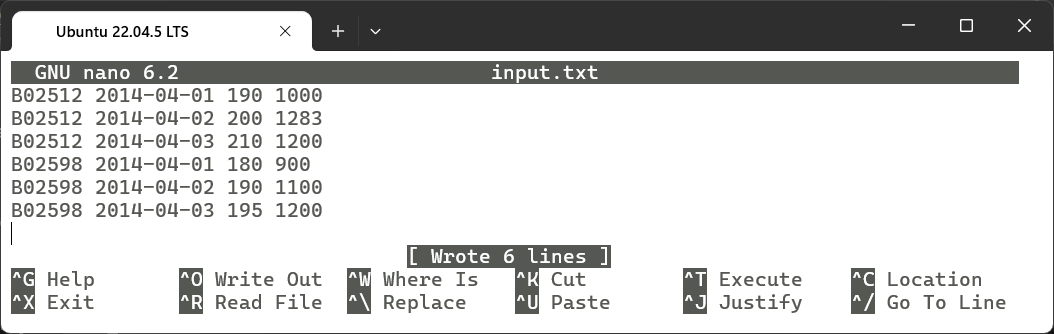


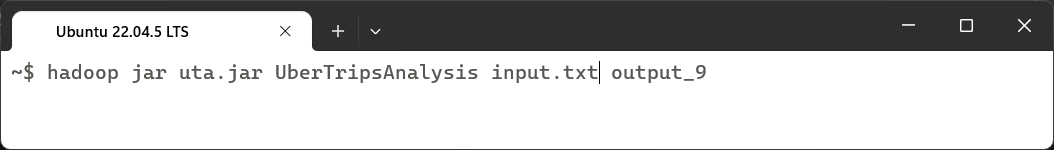


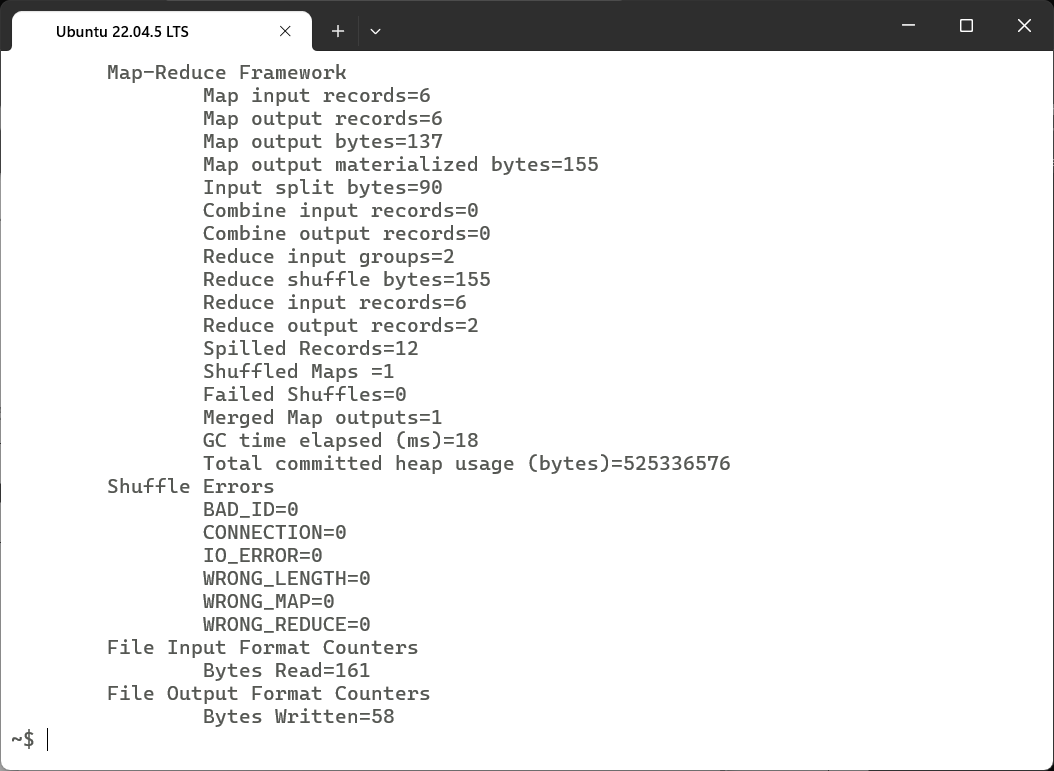


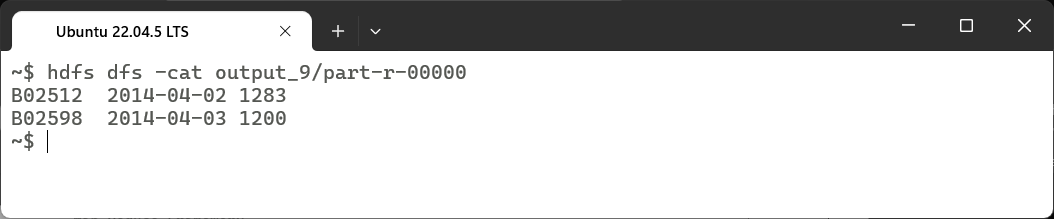












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 10

## Experiment 10: Develop a map reduce program to analyze titanic dataset to find the average age of the people (both male and female) who died in the tragedy. How many people survived in each class.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Code:

import java.io.IOException;

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.io.DoubleWritable;

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 TitanicAnalysis {

    // Mapper to process each line of the dataset

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

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

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

            // Skip the header row

            if (columns[0].equals("PassengerId")) {

                return;

            }

            try {

                int survived = Integer.parseInt(columns[1]);  // Survived column (0 = No, 1 = Yes)

                int pclass = Integer.parseInt(columns[2]);    // Passenger class (1st, 2nd, 3rd)

                String sex = columns[5];                     // Gender (male/female)

                String ageStr = columns[6];                  // Age (can be empty)

                // For average age of deceased passengers

                if (survived == 0 && !ageStr.isEmpty()) {

                    double age = Double.parseDouble(ageStr);

                    context.write(new Text("Age\_" + sex), new Text(ageStr + ",1"));

                }

                // For survivor count per class

                if (survived == 1) {

                    context.write(new Text("Class\_" + pclass), new Text("1,1"));

                }

            } catch (Exception e) {

                System.err.println("Skipping invalid record: " + value.toString());

            }

        }

    }

    // Reducer to compute results

    public static class TitanicReducer extends Reducer<Text, Text, Text, Text> {

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

            double totalAge = 0;

            int count = 0;

            int totalCount = 0;

            for (Text val : values) {

                String[] parts = val.toString().split(",");

                double value = Double.parseDouble(parts[0]);

                int occurrences = Integer.parseInt(parts[1]);

                totalAge += value;

                count += occurrences;

                totalCount += occurrences;

            }

            // Compute average age for deceased passengers

            if (key.toString().startsWith("Age\_")) {

                double avgAge = (count == 0) ? 0 : totalAge / count;

                context.write(new Text(key), new Text(String.format("Average Age: %.2f", avgAge)));

            }

            // Compute survivor count per class

            else if (key.toString().startsWith("Class\_")) {

                context.write(new Text(key), new Text("Total Survivors: " + totalCount));

            }

        }

    }

    // Main method to configure and run the MapReduce job

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

        if (args.length < 2) {

            System.err.println("Usage: TitanicAnalysis <input file> <output dir>");

            System.exit(1);

        }

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Titanic Data Analysis");

        job.setJarByClass(TitanicAnalysis.class);

        job.setMapperClass(TitanicMapper.class);

        job.setReducerClass(TitanicReducer.class);

        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(Text.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input file path

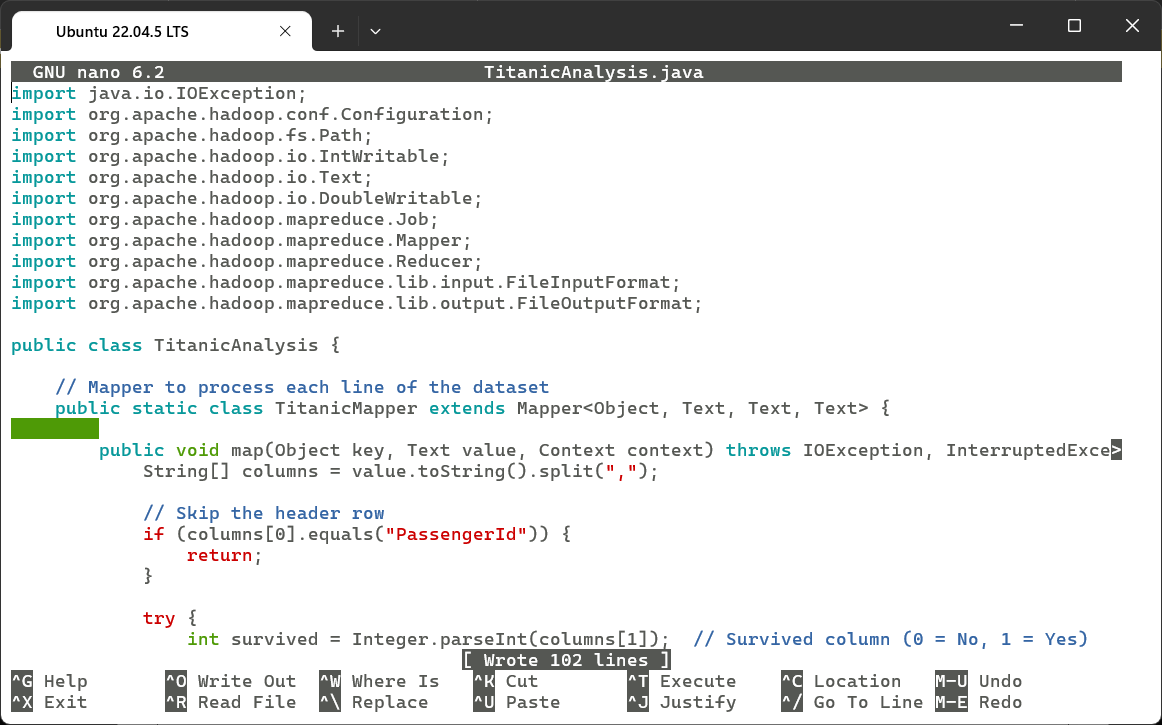
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory

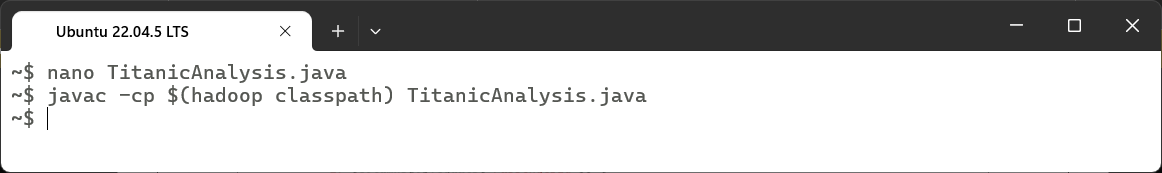
        System.exit(job.waitForCompletion(true) ? 0 : 1);

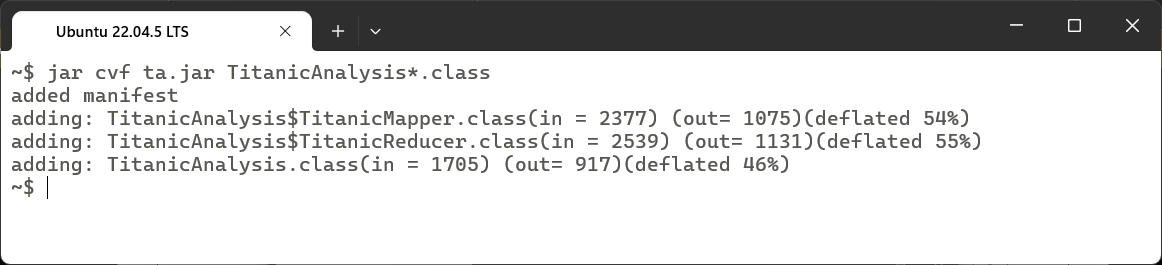
    }

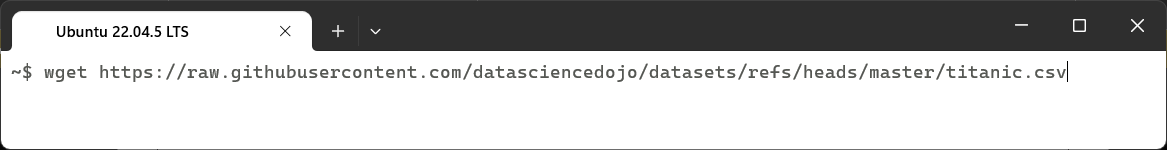
}

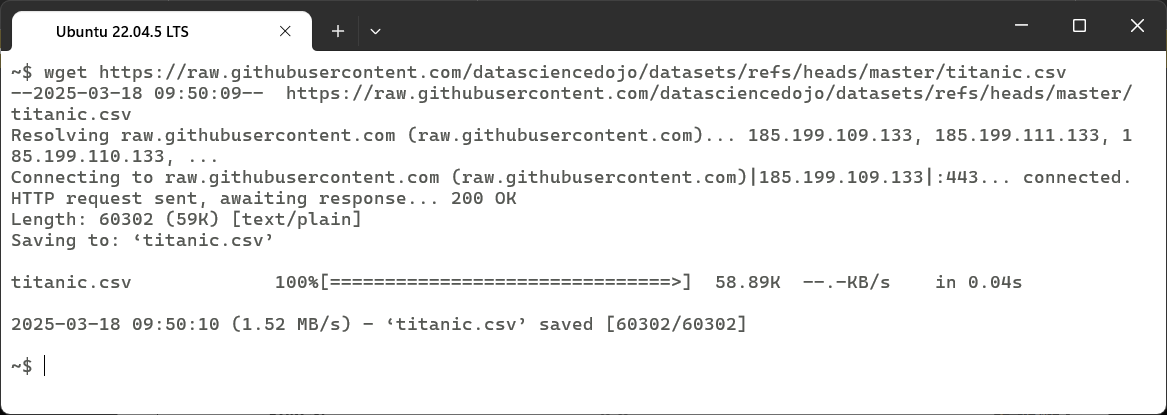


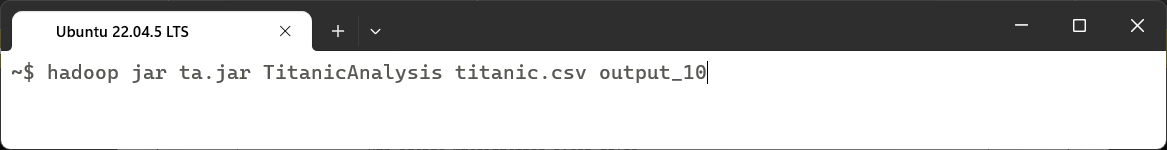


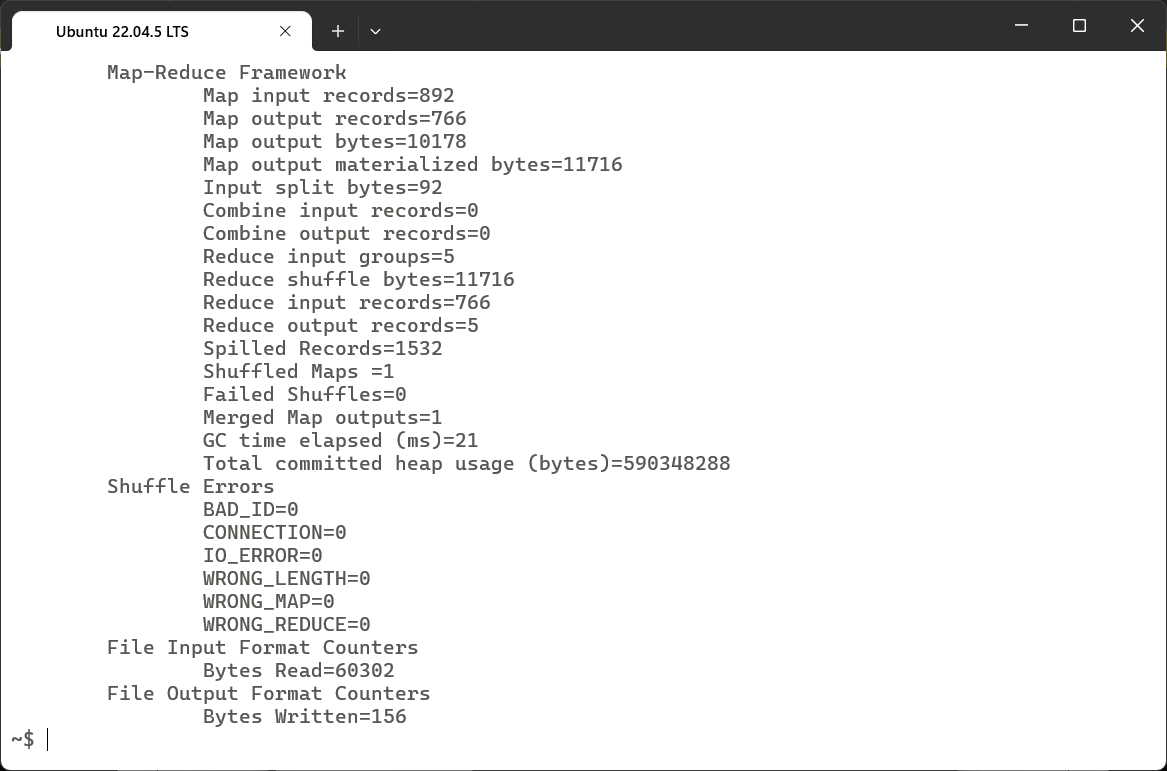


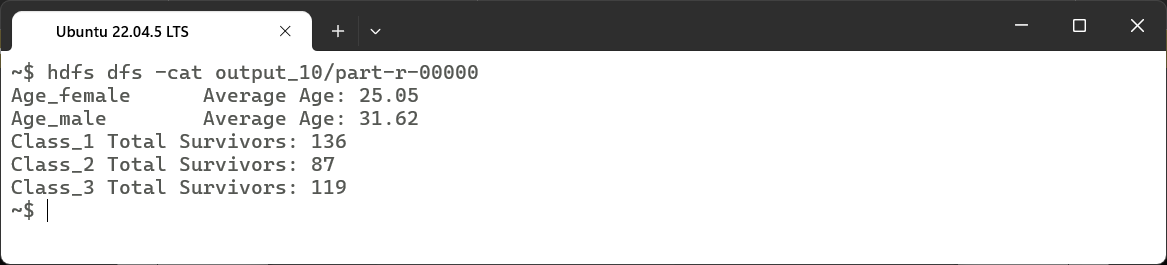












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 11

## Experiment 11: Develop a program to calculate the maximum recorded temperature year wise for the weather data set.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Code:

import java.io.IOException;

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 MaxTemperature {

    // Mapper Class

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

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

            String[] columns = value.toString().split("\\s+"); // Split by whitespace

            if (columns.length != 3) return; // Skip invalid lines

            String year = columns[0];         // Extract year

            String month = columns[1];        // Extract month

            String temp = columns[2];         // Extract temperature

            context.write(new Text(year), new Text(month + "\_" + temp)); // Emit (year, month\_temperature)

        }

    }

    // Reducer Class

    public static class MaxTempReducer extends Reducer<Text, Text, Text, IntWritable> {

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

            int maxTemp = Integer.MIN\_VALUE;

            String maxMonth = "";

            for (Text val : values) {

                String[] parts = val.toString().split("\_");

                String month = parts[0];

                int temperature = Integer.parseInt(parts[1]);

                if (temperature > maxTemp) {

                    maxTemp = temperature;

                    maxMonth = month;

                }

            }

            // Emit (Year, Month MaxTemperature)

            context.write(new Text(key.toString() + " " + maxMonth), new IntWritable(maxTemp));

        }

    }

    // Main Method

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

        if (args.length < 2) {

            System.err.println("Usage: MaxTemperature <input file> <output dir>");

            System.exit(1);

        }

        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "Max Temperature Finder");

        job.setJarByClass(MaxTemperature.class);

        job.setMapperClass(MaxTempMapper.class);

        job.setReducerClass(MaxTempReducer.class);

        job.setMapOutputKeyClass(Text.class);

        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);

        job.setOutputValueClass(IntWritable.class);

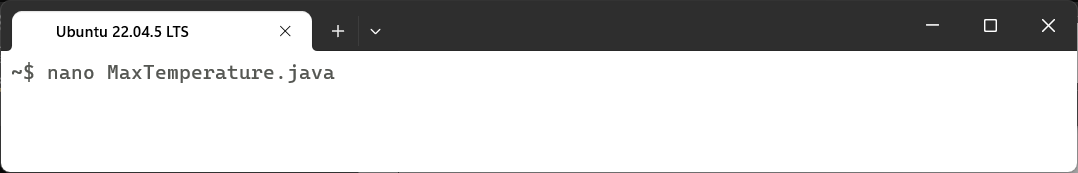
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input file path

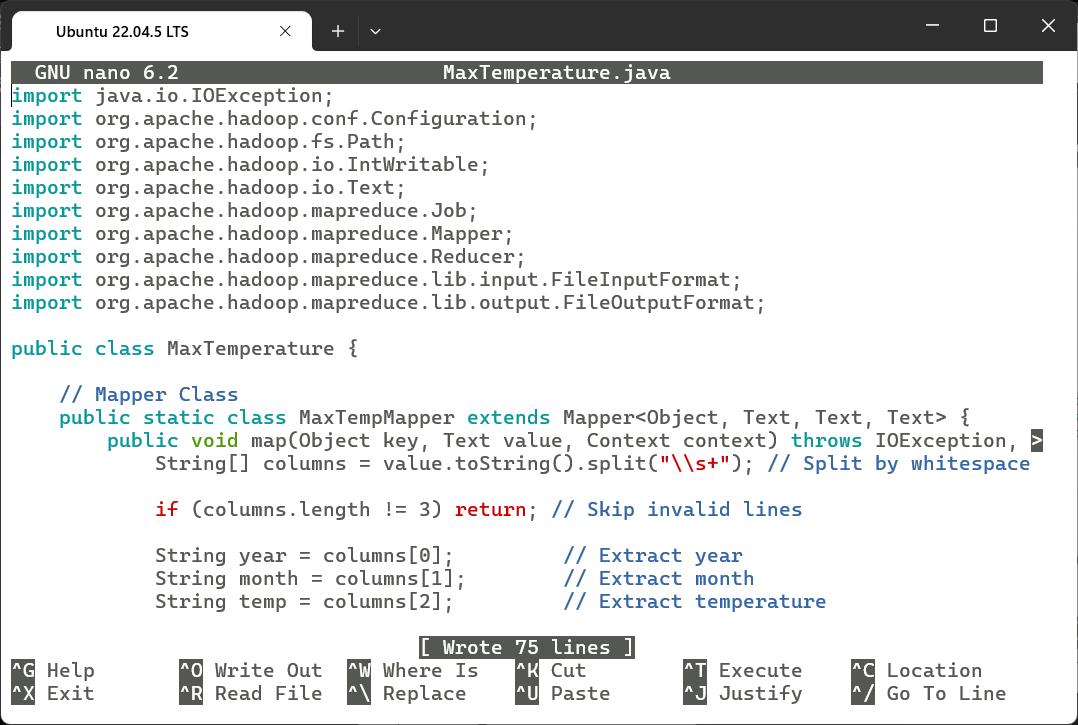
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory

        System.exit(job.waitForCompletion(true) ? 0 : 1);

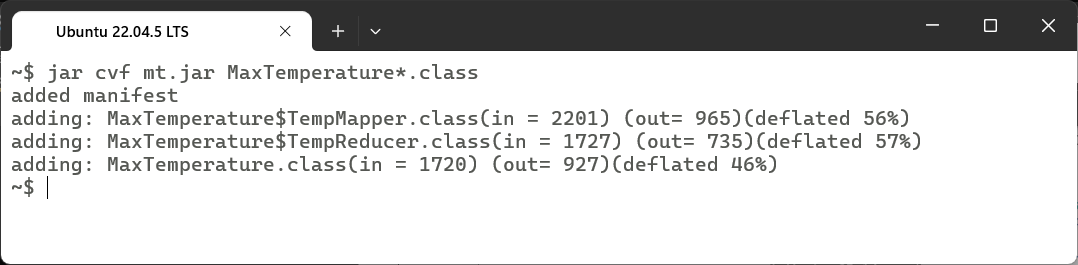
    }

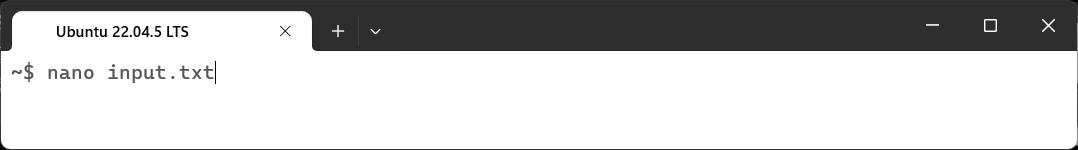
}

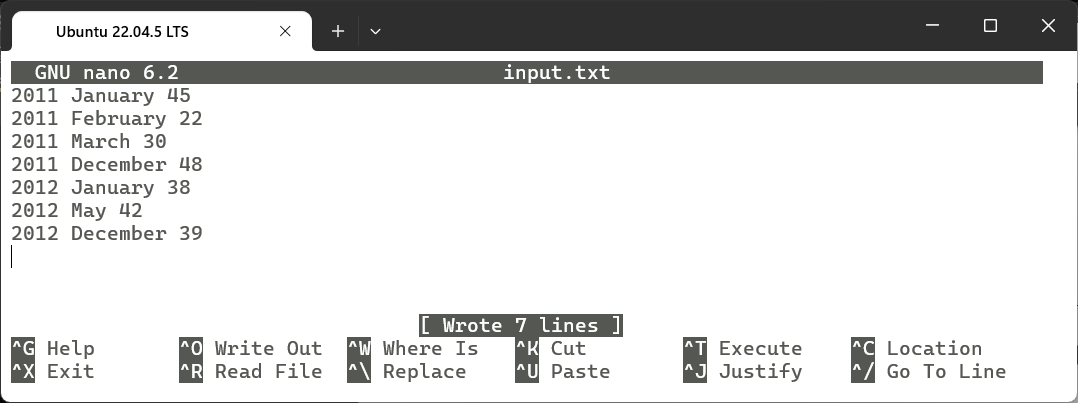


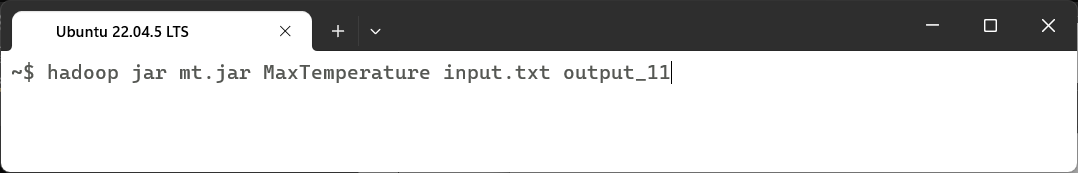


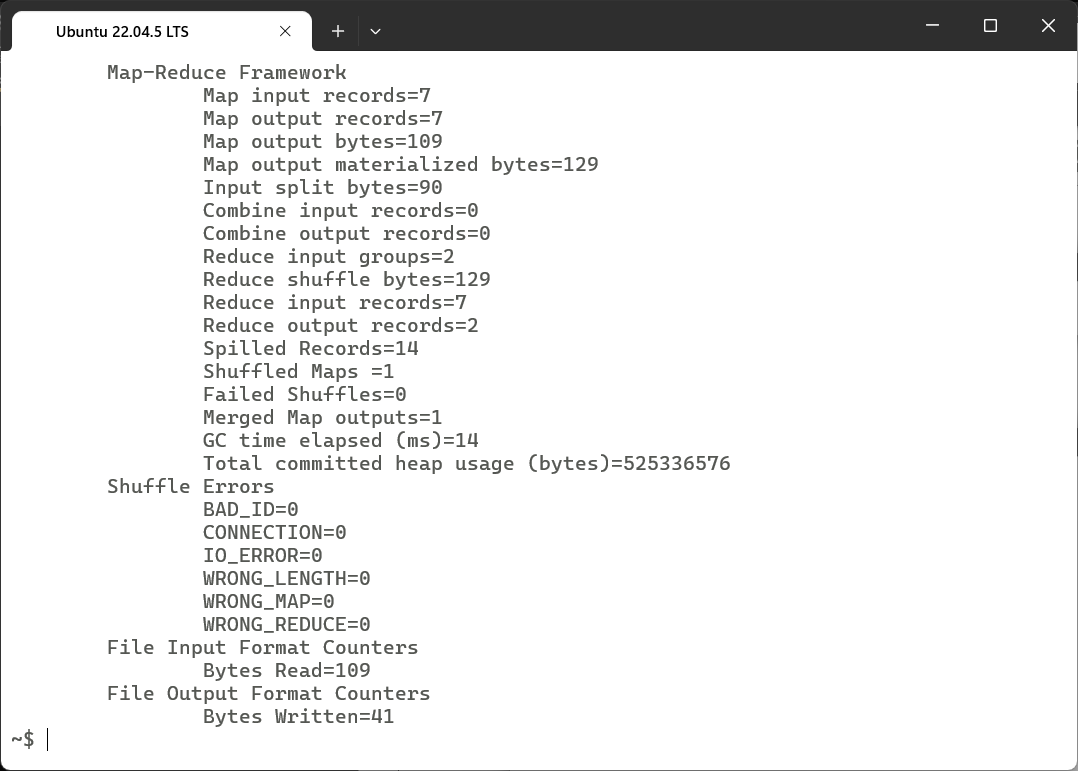


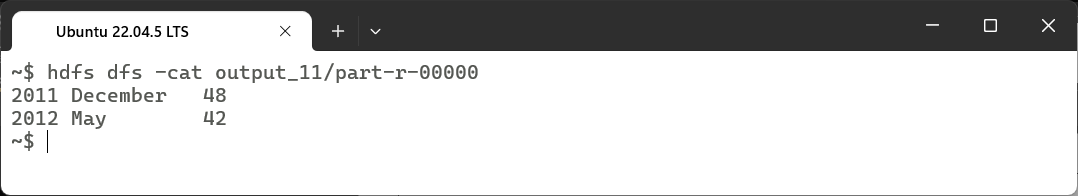












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

# Experiment 12

## Experiment 12: Write queries to sort and aggregate the data in a table.

### Theory:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

### Code:

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

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 org.apache.hadoop.io.Writable;

import java.io.DataInput;

import java.io.DataOutput;

public class Aggregate {

  public static class TitanicData implements Writable {

    int totalPassengers,

    survived;

    double totalAge;

    public TitanicData() {

      this.totalPassengers = 0;

      this.survived = 0;

      this.totalAge = 0.0;

    }

    public void addPassenger(int survived, double age) {

      totalPassengers++;

      if (survived == 1) {

        this.survived++;

      }

      if (age > 0) {

        this.totalAge += age;

      }

    }

    public double getSurvivalRate() {

      if (totalPassengers == 0) return 0.0;

      return (double) survived / totalPassengers \* 100;

    }

    public double getAverageAge() {

      if (totalPassengers == 0) return 0.0;

      return totalAge / totalPassengers;

    }

    public int getTotalPassengers() {

      return totalPassengers;

    }

    public void write(DataOutput out) throws IOException {

      out.writeInt(totalPassengers);

      out.writeInt(survived);

      out.writeDouble(totalAge);

    }

    public void readFields(DataInput in ) throws IOException {

      totalPassengers = in.readInt();

      survived = in.readInt();

      totalAge = in.readDouble();

    }

  }

  public static class TitanicMapper extends Mapper < LongWritable,

  Text,

  Text,

  TitanicData > {

    private Text classKey = new Text();

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

    InterruptedException {

      String line = value.toString();

      String[] tokens = line.split(",");

      try {

        String pclass = tokens[2];

        int survived = Integer.parseInt(tokens[1]);

        double age = tokens[5].equals("") ? 0 : Double.parseDouble(tokens[5]);

        classKey.set(pclass);

        TitanicData data = new TitanicData();

        data.addPassenger(survived, age);

        context.write(classKey, data);

      } catch(NumberFormatException | ArrayIndexOutOfBoundsException e) {}

    }

  }

  public static class TitanicReducer extends Reducer < Text,

  TitanicData,

  Text,

  Text > {

    private Text result = new Text();

    public void reduce(Text key, Iterable < TitanicData > values, Context context) throws IOException,

    InterruptedException {

      TitanicData aggregatedData = new TitanicData();

      for (TitanicData data: values) {

        aggregatedData.totalPassengers += data.getTotalPassengers();

        aggregatedData.survived += data.survived;

        aggregatedData.totalAge += data.totalAge;

      }

      double survivalRate = aggregatedData.getSurvivalRate();

      double avgAge = aggregatedData.getAverageAge();

      int totalPassengers = aggregatedData.getTotalPassengers();

      String resultString = String.format("Total Passengers: %d, Survival Rate: %.2f%%, Average Age: %.2f", totalPassengers, survivalRate, avgAge);

      result.set(resultString);

      context.write(key, result);

    }

  }

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

    Configuration conf = new Configuration();

    Job job = Job.getInstance(conf, "Titanic Aggregator");

    job.setJarByClass(Aggregate.class);

    job.setMapperClass(TitanicMapper.class);

    job.setReducerClass(TitanicReducer.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(TitanicData.class);

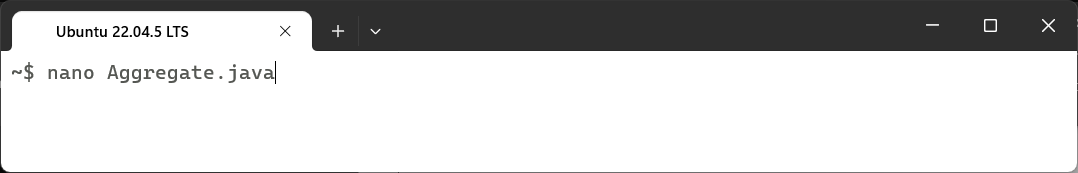
    FileInputFormat.addInputPath(job, new Path(args[0]));

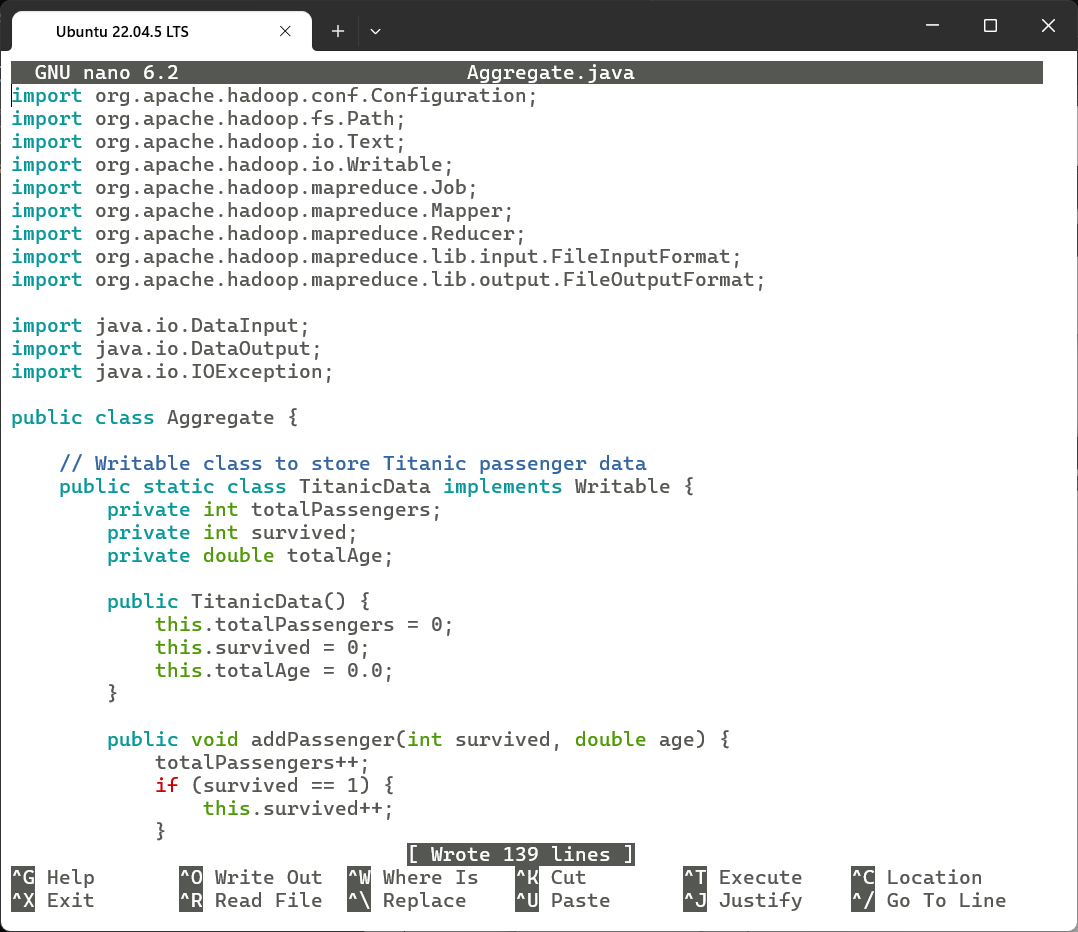
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

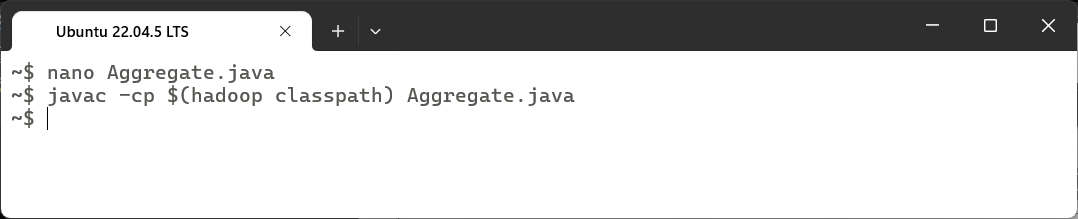
    System.exit(job.waitForCompletion(true) ? 0 : 1);

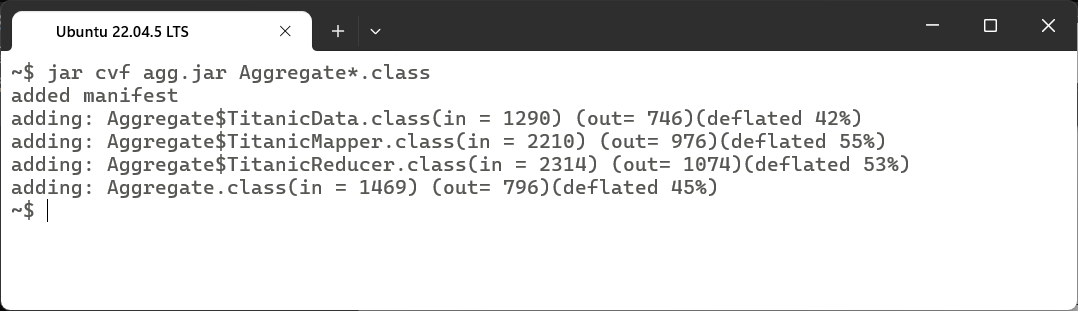
  }

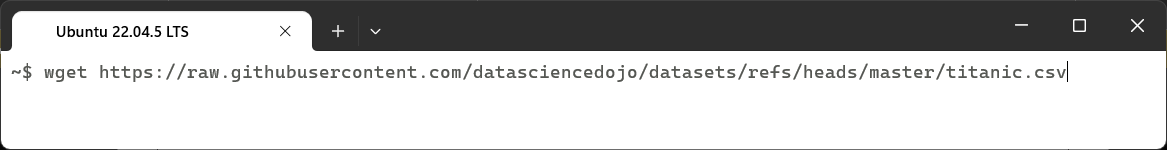
}

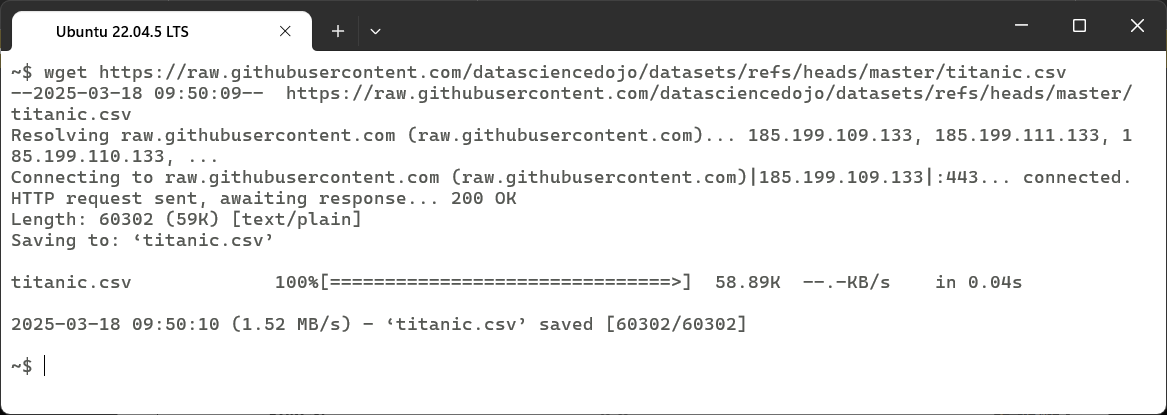




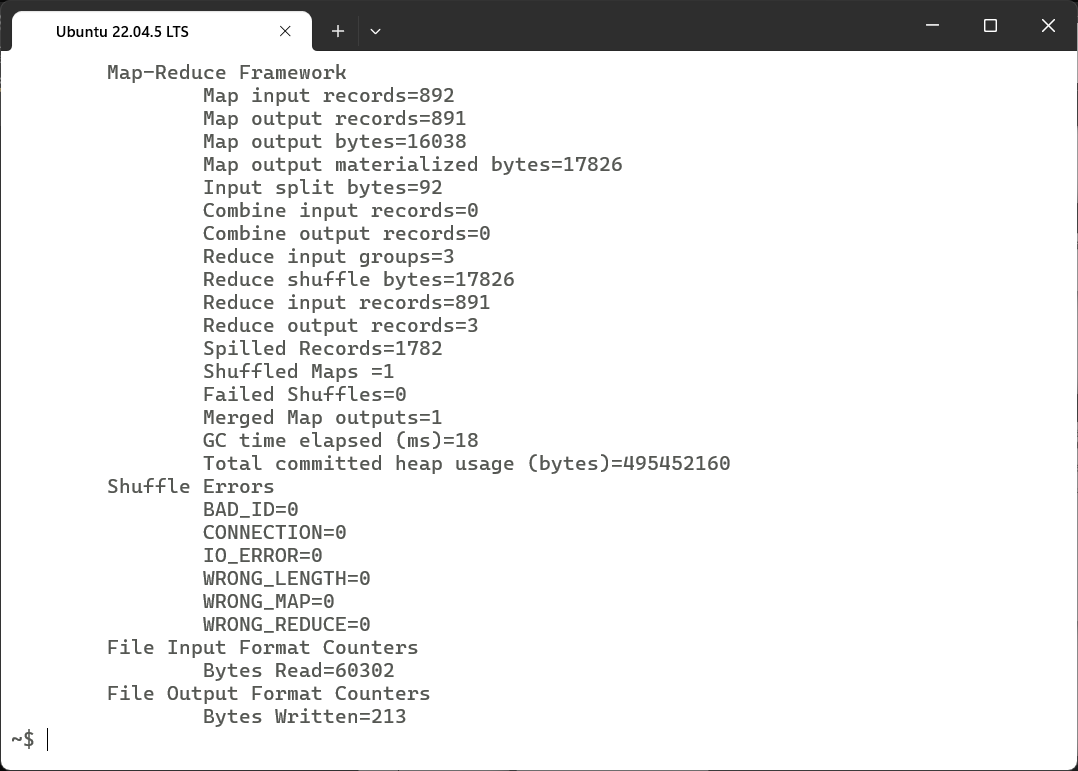


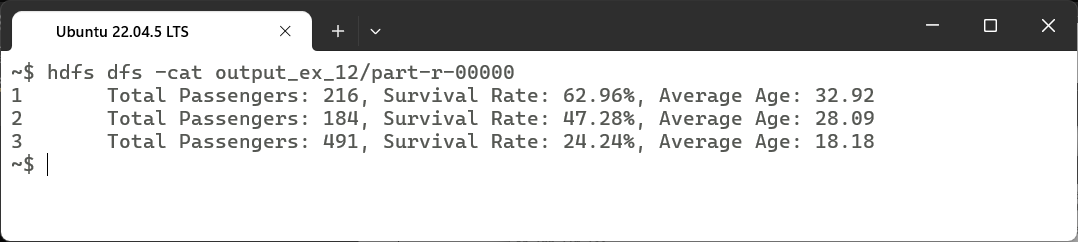












### Learning Outcomes:

|  |
| --- |
|  |
|  |
|  |
|  |
|  |