Practical No. 11

Title -: Write a code in JAVA for a simple Word Count application that counts the number of occurrences of each word in a given input set using the Hadoop Map Reduce framework on local-standalone set-up.

```
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 -:

```
D:\Demo> cmd /C ""C:\Program Files\Eclipse Adoptium\jdk-17.0.13.11-hotspot\bin\java.exe"
4c3c193555f13\redhat.java\jdt_ws\Demo_72d793fc\bin LocalWordCount "
hadoop  1
hadoop  1
hello   1
hello   1
map     1
of      1
program 1
reduce  1
world   1
world   1
```

Practical No. - 12

April 30, 2025

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[4]:
```
file_name = "sample_weather.txt"
```

[ ]:
```
try:
    weather_df = pd.read_csv(file_name, delim_whitespace=True)
except:
    weather_df = pd.read_csv(file_name, delimiter=',')
```

[6]:
```
weather_df.head()
```

[6]:

	690190	13910	20060201_0	51.75	33.0	24	1006.3	24.1	943.9	24.2	\
0	690190	13910	20060201_1	54.74	33.0	24	1006.3	24	943.9	24	
1	690190	13910	20060201_2	50.59	33.0	24	1006.3	24	943.9	24	
2	690190	13910	20060201_3	51.67	33.0	24	1006.3	24	943.9	24	
3	690190	13910	20060201_4	65.67	33.0	24	1006.3	24	943.9	24	
4	690190	13910	20060201_5	55.37	33.0	24	1006.3	24	943.9	24	

	15.0	24.3	10.7	24.4	22.0	28.9	0.00I	999.9	000000
0	15.0	24	10.7	24	22.0	28.9	0.00I	999.9	0
1	15.0	24	10.7	24	22.0	28.9	0.00I	999.9	0
2	15.0	24	10.7	24	22.0	28.9	0.00I	999.9	0
3	15.0	24	10.7	24	22.0	28.9	0.00I	999.9	0
4	15.0	24	10.7	24	22.0	28.9	0.00I	999.9	0

[7]:
```
weather_df.columns = weather_df.columns.str.lower().str.replace(' ', '_')
```

[8]:
```
print("Missing values per column:")
print(weather_df.isnull().sum())
```

```
Missing values per column:
690190        0
13910         0
20060201_0    0
51.75         0
33.0          0
24            0
```

```
1006.3        0
24.1          0
943.9         0
24.2          0
15.0          0
24.3          0
10.7          0
24.4          0
22.0          0
28.9          0
0.00i         0
999.9         0
000000        0
dtype: int64
```

[9]:
```
weather_df = weather_df.dropna()
```

[10]:
```
numeric_cols = ['temperature', 'dew_point', 'wind_speed']
for col in numeric_cols:
    if col in weather_df.columns:
        weather_df[col] = pd.to_numeric(weather_df[col], errors='coerce')
```

[11]:
```
weather_df = weather_df.dropna()
```

[ ]:
```
averages = {
    'Average Temperature': weather_df['temperature'].mean(),
    'Average Dew Point': weather_df['dew_point'].mean(),
    'Average Wind Speed': weather_df['wind_speed'].mean()
}
```

[ ]:
```
for metric, value in averages.items():
    print(f"{metric}: {value:.2f}")
```

[14]:
```
plt.figure(figsize=(15, 5))
```

[14]:
```
<Figure size 1500x500 with 0 Axes>
```

```
<Figure size 1500x500 with 0 Axes>
```

[ ]:
```
plt.subplot(1, 3, 1)
plt.hist(weather_df['temperature'], bins=20, color='red', alpha=0.7)
plt.title('Temperature Distribution')
plt.xlabel('Temperature')
plt.ylabel('Frequency')
```

[16]:
```
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

[ ]:
```
averages_df = pd.DataFrame.from_dict(averages, orient='index',␣
↪columns=['Value'])
averages_df.to_csv('weather_averages.csv')
print("Averages saved to weather_averages.csv")
```

[ ]:
```

```

Practical No. 13

Title -: Write a simple program in SCALA using Apache Spark framework

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

object WeatherAnalysis {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession.builder()
      .appName("WeatherDataAnalysis")
      .master("local[*]")
      .getOrCreate()

    import spark.implicits._

    val filePath = "sample_weather.txt"

    val weatherDF = spark.read
      .option("header", "true")
      .option("inferSchema", "true")
      .option("delimiter", " ")
      .csv(filePath)

    val finalDF = if (weatherDF.columns.length <= 1) {
      spark.read
        .option("header", "true")
        .option("inferSchema", "true")
        .option("delimiter", ",")
        .csv(filePath)
    } else {
      weatherDF
    }

    val columnsRenamed = finalDF.columns.map(_.toLowerCase.replace(" ",
"_"))
    val renamedDF = finalDF.toDF(columnsRenamed: _*)
```

```
    println("Data Schema:")
    renamedDF.printSchema()
    println("\nSample Data:")
    renamedDF.show(5)

    val averages = renamedDF.select(
      avg("temperature").as("avg_temperature"),
      avg("dew_point").as("avg_dew_point"),
      avg("wind_speed").as("avg_wind_speed")
    )

    println("\nAverage Values:")
    averages.show()

    spark.stop()
  }
}
```

build.sbt

```
name := "WeatherAnalysis"
version := "1.0"
scalaVersion := "2.12.15"

libraryDependencies += "org.apache.spark" %% "spark-sql" % "3.3.0"
```

Output -:

```
D:\SparkDemo> cmd /C ""C:\Program Files\Java\jdk-17.0.13.11-hotspot\bin\java.exe" -XX:+ShowCodeD
etailsInExceptionMessages -cp "C:\spark\spark-3.3.0-bin-hadoop3\jars\*" WeatherAnalysis"

Loading data from sample_weather.txt...
First 5 rows:
+-----------+---------+----------+
|temperature|dew_point|wind_speed|
+-----------+---------+----------+
|72.5       |65.3     |8.2       |
|68.1       |62.7     |5.5       |
|74.3       |68.9     |10.1      |
|70.8       |64.2     |7.3       |
|69.4       |63.5     |6.8       |
+-----------+---------+----------+

Calculated Averages:
+--------------+------------+-------------+
|avg_temperature|avg_dew_point|avg_wind_speed|
+--------------+------------+-------------+
|71.02         |64.92       |7.58         |
+--------------+------------+-------------+
```