BIG DATA LABORATORY

Course Code: ISL75 **Credit: 0:0:1**
Prerequisite: Nil **Contact Hours: 14L**

Course Coordinator: Dr. Rajeshwari S B

PART-A

Write MapReduce programs for the following using Java:

1. To count the number of occurrences of each word in a given input text.
2. To read N natural numbers and display the sum along with Odd and Even count.
3. To analyse the given **Employee Data** and generate a statistics report with the total number of Female and Male Employees and their average Salary.
4. To analyse the **Titanic Ship Data** and find the average age of the people (both male and female) who died in the tragedy and also how many people are survived in each class.
5. To analyse the **Earthquake Data** and generate statistics with region and magnitude/region and depth/region and latitude/region and longitude.
6. To analyse the given **Sales Records** over a period of time and generate data about the country's total sales and the total number of products. Country's total sales and the frequency of the payment mode.

PART-B

7. Write a **Spark program using Python**, to analyse the given **Weather Report Data** and to generate a report with cities having maximum and minimum temperature for a particular year.

8. Write a **Spark program using Python**, to analyse the given **Insurance Data** and generate a statistics report with the construction building name and the count of building/ county name and its frequency.

9. Write **Pig Latin scripts** for the following queries on **Crop Production Dataset**:
   a. Calculate total production of each crop.
   b. Find the average production per year for each crop.
   c. Filter all crops grown in 'Karnataka'.
   d. Calculate the total area used for each crop in the year 2010.

10. Write **Pig Latin scripts** for the following queries on **Olympic Athletes and Hosts Dataset**:
    a. Filter athletes participated in the "Tokyo 2020" games.
    b. Filter the games held in "China".
    c. Group games by season and count the number of games in each session.
    d. Filter games that occurred after the year 2000.

11. Write **Hive Query** for the following on **Online Retail Dataset**:
    a. Total number of unique customers in the "given country".
    b. Filter the country from which the maximum revenue was collected from sales in the month of March 2010.
    c. Filter the month of 2010 in which maximum number of items were sold.
    d. In the month of January 2010, find the country in which maximum number of items were sold.

12. Write **Hive Query** for the following on **Coffee Sales Dataset**:
    a. Number of customers had “Hot Chocolate “coffee on the given date.
    b. Amount of money collected by “Latte” coffee sales.
    c. Number of customers done the payment through card.
    d. Most favourite coffee.

References:

1. “Hadoop: The Definitive Guide”, Tom White ,4th Edition, O’Reilly Media, 2015.
2. "Learning Spark: Lightning-Fast Big Data Analysis", Holden Karau, Andy Konwinski, Patrick Wendell, and Matei Zaharia, O'Reilly Media, 2016.
3. “Programming pig: Dataflow scripting with Hadoop", Gates, Alan, and Daniel Dai. O'Reilly Media, 2016.
4. “Programming Hive: Data warehouse and query language for Hadoop", Capriolo, Edward, Dean Wampler, and Jason Rutherglen. O'Reilly Media, 2012.

Course Outcomes (COs):

1. Explore the Hadoop ecosystem and MapReduce programming model to develop and execute a range of applications for Big Data analytics. (PO-1, 2, 3, 5, 9, 10) (PSO- 1, 2)
2. Dissect Apache Spark, a faster alternative to MapReduce, by implementing large-scale data applications. (PO-1, 2, 3, 5, 9, 10) (PSO- 1, 2)
3. Analyse Big Data applications using Pig and Hive for efficient data processing and querying. (PO-1, 2, 3, 5, 9, 10) (PSO- 1, 2)

1. Map Reduce program to count the number of occurrences of each word in a given input text.

driver.java

```
package word count;
import java.io. *;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;
public class driver
{
   public static void main(String args[]) throws IOException
   {
         JobConf conf=new JobConf(driver.class);
         conf.setMapperClass(mapper.class);
         conf.setReducerClass(reducer.class);
         conf.setOutputKeyClass(Text.class);
         conf.setOutputValueClass(IntWritable.class);
         FileInputFormat.addInputPath(conf, new Path(args[0]));
         FileOutputFormat.setOutputPath(conf,new Path(args[1]));
         JobClient.runJob(conf);
   }
}
```

mapper.java

```
package word count;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;

public class mapper extends MapReduceBase implements Mapper<LongWritable, Text, Text,
IntWritable> {

   // Static final variable for the count of 1
   private final static IntWritable one = new IntWritable(1);

   // Reusable Text object to hold each word
   private Text word = new Text();

   // The map function
   public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
Reporter reporter)
         throws IOException {

      // Convert the input value (line of text) to a string
      String line = value.toString();

      // Tokenize the line into words
      StringTokenizer tokenizer = new StringTokenizer(line);
```

```
        // Iterate through the tokens (words)
        while (tokenizer.hasMoreTokens()) {
            // Set the current word into the Text object
            word.set(tokenizer.nextToken());

            // Collect the word and emit (word, 1) as key-value pairs
            output.collect(word, one);
        }
    }
}
```

reducer.java

```
package word count;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;

public class reducer extends MapReduceBase implements Reducer<Text, IntWritable, Text,
IntWritable> {

    public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
output,
            Reporter reporter) throws IOException {

        int sum = 0;

        // Sum up the counts for each word
        while (values.hasNext()) {
            sum += values.next().get();
        }

        // Emit the word with the total count
        output.collect(key, new IntWritable(sum));
    }
}
```

Steps to run

1. Create a New File named Bash.sh
2. Copy the Below code and Paste inside Bash.sh and save that File.
   ```
   export JAVA_HOME=$(readlink -f $(which javac) | awk 'BEGIN {FS="/bin"} {print $1}')
   export PATH=$(echo $PATH):$(pwd)/bin
   export CLASSPATH=$(hadoop classpath)
   ```
3. Execute the bash.sh File using following command source Bash.sh.
4. Verify JAVA_HOME variable to be set to Java Path and PATH variable has your USN Hadoop Folder.
   If any previous PATH set to Hadoop Folder remove that inside .bashrc file.

5. Verify Hadoop is Installed or not by executing hadoop command.if command gives Information about
   Hadoop command then Hadoop is Successfully Installed.
6. Create a folder word count and move to that folder.
7. Make the driver.java , mapper.java and reducer.java files.
8. Compile all java files (driver.java mapper.java reducer.java)
   javac -d . *.java
9. Set driver class in manifest
   echo Main-Class: wordcount.driver > Manifest.txt
10. Create an executable jar file
    jar cfm wordcount.jar Manifest.txt word count/*.class
11. oe.txt is input file for Oddeven create Input File
    echo “hello good morning, hello have a nice day” > input.txt
12. Run the jar file
    hadoop jar wordcount.jar input.txt output
13. To see the Output
    cat output/*

1. Write a MapReduce program using Java, to analyze the given natural numbers and generate statistics for the number as Odd or Even and print their sum.

driver.java

```
package oddeven;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;

public class driver
{
   public static void main(String args[]) throws IOException
   {
          JobConf conf=new JobConf(driver.class);
          conf.setMapperClass(mapper.class);
          conf.setReducerClass(reducer.class);
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(IntWritable.class);
          FileInputFormat.addInputPath(conf, new Path(args[0]));
          FileOutputFormat.setOutputPath(conf,new Path(args[1]));
          JobClient.runJob(conf);
   }
}
```

mapper.java

```
package oddeven;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;

public class mapper extends MapReduceBase implements Mapper<LongWritable , Text , Text , IntWritable>
{
   public void map(LongWritable key,Text value,OutputCollector<Text,IntWritable> output,Reporter r) throws
IOException
   {
          String[] line=value.toString().split(" ");
          for(String num:line){
                 int number=Integer.parseInt(num);
                 if(number%2==0) {
                        output.collect(new Text("even"),new IntWritable(number));
                 }
                 else{
                        output.collect(new Text("odd"),new IntWritable(number));
                 }
          }
   }
}
```

reducer.java

```
package oddeven;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
public class reducer extends MapReduceBase implements Reducer<Text,IntWritable,Text,IntWritable>
{
   public void reduce(Text key,Iterator<IntWritable> value,OutputCollector<Text,IntWritable> output ,Reporter
r) throws IOException
   {
          int sum=0,count=0;
          while(value.hasNext()){
                    sum+=value.next().get();
                    count++;
          }
          output.collect(new Text("Sum of "+key+" Numbers"),new IntWritable(sum));
          output.collect(new Text(key+" Number count"),new IntWritable(count));
   }
}
```

input.txt

```
1 2 3 4 5 6 7 8 9 10
```

Steps to run

1. Create a New File named Bash.sh
2. Copy the Below code and Paste inside Bash.sh and save that File.
   ```
   export JAVA_HOME=$(readlink -f $(which javac) | awk 'BEGIN {FS="/bin"} {print $1}')
   export PATH=$(echo $PATH):$(pwd)/bin
   export CLASSPATH=$(hadoop classpath)
   ```
3. Execute the bash.sh File using following command source Bash.sh.
4. Verify JAVA_HOME variable to be set to Java Path and PATH variable has your USN Hadoop Folder. If any previous PATH set to Hadoop Folder remove that inside .bashrc file.
5. Verify Hadoop is Installed or not by executing hadoop command.if command gives Information about Hadoop command then Hadoop is Successfully Installed.
6. Create a folder oddeven and move to that folder.
7. Make the driver.java , mapper.java and reducer.java files.
8. Compile all java files (driver.java mapper.java reducer.java)
   ```
   javac -d . *.java
   ```
9. Set driver class in manifest
   ```
   echo Main-Class: oddeven.driver > Manifest.txt
   ```
10. Create an executable jar file
    ```
    jar cfm oddeven.jar Manifest.txt oddeven/*.class
    ```
11. oe.txt is input file for Oddeven create Input File
    ```
    echo 1 2 3 4 5 6 7 8 9 10 > input.txt
    ```
12. Run the jar file
    ```
    hadoop jar oddeven.jar input.txt output
    ```
13. To see the Output
    ```
    cat output/*
    ```

3. Write MapReduce program in Java to analyse the given **Employee Data** and generate a statistics report with the total number of Female and Male Employees and their average Salary.

driver.java

```
package employee;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;

public class driver
{
   public static void main(String args[]) throws IOException
   {
          JobConf conf=new JobConf(driver.class);
          conf.setMapperClass(mapper.class);
          conf.setReducerClass(reducer.class);
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(DoubleWritable.class);
          FileInputFormat.addInputPath(conf,new Path(args[0]));
          FileOutputFormat.setOutputPath(conf,new Path(args[1]));
          JobClient.runJob(conf);
   }
}
```

mapper.java

```
package employee;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;

class mapper extends MapReduceBase implements Mapper<LongWritable , Text , Text , DoubleWritable>
{
   public void map(LongWritable key, Text value, OutputCollector<Text,DoubleWritable> output ,Reporter r) throws IOException
   {
          String[] line=value.toString().split(",");
          Double salary=Double.parseDouble(line[4]);
          output.collect(new Text(line[3]), new DoubleWritable(salary));
   }
}
```

reducer.java

```
package employee;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
class reducer extends MapReduceBase implements Reducer<Text,DoubleWritable,Text,DoubleWritable>
{
public void reduce(Text key,Iterator<DoubleWritable> value , OutputCollector<Text,DoubleWritable> output ,Reporter r) throws IOException
   {
          int count=0;
          Double sum=0.0;
```

```
            while(value.hasNext()){
                    sum+=value.next().get();
                    count+=1;
            }
            output.collect(new Text(key+" Average"), new DoubleWritable(sum/count));
            output.collect(new Text(key+" Count"), new DoubleWritable(count));
    }
}
```

Steps to run

1. Create a New File named Bash.sh
2. Copy the Below code and Paste inside Bash.sh and save that File.
   ```
   export JAVA_HOME=$(readlink -f $(which javac) | awk 'BEGIN {FS="/bin"} {print$1}')
   export PATH=$(echo $PATH):$(pwd)/bin
   export CLASSPATH=$(hadoop classpath)
   ```
3. Execute the bash.sh File using following command source bash.sh.
4. Verify JAVA_HOME variable to be set to Java Path and PATH variable has your USN Hadoop Folder.If any previous PATH set to Hadoop Folder remove that inside .bashrc file.
5. Verify Hadoop is Installed or not by executing hadoop command.if command gives Information about Hadoop command then Hadoop is Successfully Installed.
6. Create a folder oddeven and move to that folder
7. Make the driver.java , mapper.java and reducer.java files
8. Compile all java files (driver.java mapper.java reducer.java)
   ```
   javac -d . *.java
   ```
9. Set driver class in manifest
   ```
   echo Main-Class: employee.driver > Manifest.txt
   ```
10. Create an executable jar file
    ```
    jar cfm employee.jar Manifest.txt employee/*.class
    ```
11. input.csv is input file for employee create Input File
12. Run the jar file
    ```
    hadoop jar employee.jar input.csv output
    ```
13. To see the Output
    ```
    cat output/*
    ```

5. **Write a MapReduce program using Java, to analyze the given Earthquake Data and generate region wise statistics of largest earthquake with respect to longitude.**

driver.java

```
package earthquake;
import java.util.*;
import java.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;

public class driver
{
   public static void main(String args[]) throws IOException
   {
          JobConf conf=new JobConf(driver.class);
          conf.setMapperClass(mapper.class);
          conf.setReducerClass(reducer.class);
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(DoubleWritable.class);
          FileInputFormat.addInputPath(conf, new Path(args[0]));
          FileOutputFormat.setOutputPath(conf,new Path(args[1]));
          JobClient.runJob(conf);
   }
```

mapper.java

```
package earthquake;
import java.util.*;
import java.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
public class mapper extends MapReduceBase implements Mapper<LongWritable,
Text,Text,DoubleWritable>
{
   public void map(LongWritable key , Text value , OutputCollector<Text,DoubleWritable> output,
Reporter r) throws IOException
   {
          String[] line=value.toString().split(",");
          Double longi=Double.parseDouble(line[7]);
          output.collect(new Text(line[11]), new DoubleWritable(longi));
   }
}
```

reducer.java

```
package earthquake;
import java.util.*;
import java.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
```

```
class reducer extends MapReduceBase implements
Reducer<Text,DoubleWritable,Text,DoubleWritable> {

   public void reduce(Text key, Iterator<DoubleWritable> value,
OutputCollector<Text,DoubleWritable> output, Reporter r) throws IOException
   {
          Double max=-9999.0;
          while(value.hasNext())
          {
                 Double temp=value.next().get();
                 max=Math.max(max,temp);
          }
          output.collect(new Text(key), new DoubleWritable(max));
   }

}
```

Steps to run

1. Create a New File named Bash.sh
2. Copy the Below code and Paste inside Bash.sh and save that File.
```
export JAVA_HOME=$(readlink -f $(which javac) | awk 'BEGIN {FS="/bin"} {print $1}')
export PATH=$(echo $PATH):$(pwd)/bin
export CLASSPATH=$(hadoop classpath)
```
3. Execute the bash.sh File using following command source Bash.sh.
4. Verify JAVA_HOME variable to be set to Java Path and PATH variable has your USN Hadoop Folder.If any previous PATH set to Hadoop Folder remove that inside .bashrc file.
5. Verify Hadoop is Installed or not by executing hadoop command.if command gives Information about Hadoop command then Hadoop is Successfully Installed.
6. Create a folder oddeven and move to that folder
7. Make the driver.java , mapper.java and reducer.java files
8. Compile all java files (driver.java mapper.java reducer.java)
```
javac -d . *.java
```
9. Set driver class in manifest
```
echo Main-Class: earthquake.driver > Manifest.txt
```
10. Create an executable jar file
```
jar cfm earthquake.jar Manifest.txt earthquake/*.class
```
11. input.csv is input file for earthquake create Input File
12. Run the jar file
```
hadoop jar earthquake.jar input.csv output
```
13. To see the Output
```
cat output/*
```

6. **Write a MapReduce program using Java, to analyze the given Sales Records over a period of time and generate data about the country's total sales, and the total number of the products. / Country's total sales and the frequency of the payment mode.**

driver.java

```
package sales;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.Path;

public class driver
{
   public static void main(String args[]) throws IOException
   {
          JobConf conf=new JobConf(driver.class);
          conf.setMapperClass(mapper.class);
          conf.setReducerClass(reducer.class);
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(IntWritable.class);
          FileInputFormat.addInputPath(conf, new Path(args[0]));
          FileOutputFormat.setOutputPath(conf,new Path(args[1]));
          JobClient.runJob(conf);
   }
}
```

mapper.java

```
package sales;
import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;

public class mapper extends MapReduceBase implements Mapper<LongWritable , Text , Text ,
IntWritable>
{
   public void map(LongWritable key,Text value,OutputCollector<Text,IntWritable> output,Reporter
r) throws IOException
   {
          String[] line=value.toString().split(",");
          int price=Integer.parseInt(line[2]);
          String cardtype=line[3];
          String Country=line[7];
          output.collect(new Text("Country "+Country),new IntWritable(price));
          output.collect(new Text("CardType "+cardtype),new IntWritable(1));
   }
}
```

reducer.java

```
package sales;

import java.io.*;
import java.util.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;

public class reducer extends MapReduceBase implements
Reducer<Text,IntWritable,Text,IntWritable>
{
   public void reduce(Text key,Iterator<IntWritable> value,OutputCollector<Text,IntWritable> output
,Reporter r) throws IOException
   {
           int sum=0;
           while(value.hasNext())
           {
                   sum+=value.next().get();
           }
           output.collect(new Text(key),new IntWritable(sum));
   }
}
```

Steps to run

1. Create a New File named Bash.sh
2. Copy the Below code and Paste inside Bash.sh and save that File.
```
export JAVA_HOME=$(readlink -f $(which javac) | awk 'BEGIN {FS="/bin"} {print $1}')
export PATH=$(echo $PATH):$(pwd)/bin
export CLASSPATH=$(hadoop classpath)
```
3. Execute the bash.sh File using following command source Bash.sh.
4. Verify JAVA_HOME variable to be set to Java Path and PATH variable has your USN Hadoop Folder.If any previous PATH set to Hadoop Folder remove that inside .bashrc file.
5. Verify Hadoop is Installed or not by executing hadoop command.if command gives Information about Hadoop command then Hadoop is Successfully Installed.
6. Create a folder oddeven and move to that folder
7. Make the driver.java , mapper.java and reducer.java files
8. Compile all java files (driver.java mapper.java reducer.java)
```
javac -d . *.java
```
9. Set driver class in manifest
```
echo Main-Class: sales.driver > Manifest.txt
```
10. Create an executable jar file
```
jar cfm sales.jar Manifest.txt sales/*.class
```
11. sales.txt is input file for Sales create Input File
12. Run the jar file
```
hadoop jar sales.jar sales.txt output
```
13. To see the Output
```
cat output/*
```

Pig Latin Programs Solution

1. Write Pig Latin scripts for Crop Production Dataset.

Solution:

```
crop_prod = LOAD 'crop_production.csv' USING PigStorage (',') AS
(State_Name:chararray, District_Name:chararray, Crop_Year:int, Season:chararray,
Crop:chararray, Area:float, Production:float);
DESCRIBE crop_prod;
```

a. **Calculate total production of each crop.**

```
total_production = GROUP crop_prod BY Crop;
sum_production = FOREACH total_production GENERATE group AS Crop,
SUM(crop_prod.Production) AS Total_Production;
DUMP sum_production;
```

b. Find the average production per year for each crop.

```
grouped_by_crop_year = GROUP crop_prod BY (Crop, Crop_Year);
average_production = FOREACH grouped_by_crop_year GENERATE group.Crop
AS Crop, group.Crop_Year AS Crop_Year, AVG(crop_prod.Production) AS
Avg_Production;
DUMP average_production;
```

c. Filter all crops grown in ‘Karnataka’

```
specific_state = FILTER crop_prod BY State_Name == 'Karnataka';
unique_crops = GROUP specific_state BY Crop;
DUMP unique_crops;
```

[Note: Ensure that the string 'Karnataka' uses straight quotes (' ') instead of typographic quotes (‘’), as Pig does not recognize the latter.]

d. Calculate the total area used for each crop in the year 2010.

```
specific_year = FILTER crop_prod BY Crop_Year == 2010;
total_area = GROUP specific_year BY Crop;
sum_area = FOREACH total_area GENERATE group AS Crop, SUM (specific_year.
Area) AS Total_Area;
DUMP sum_area;
```

2. Write Pig Latin scripts for the following queries on Olympic Athletes and Hosts Dataset

```
athletes = LOAD 'olympic_athletes.csv' USING PigStorage (',') AS (athlete_url:
chararray, athlete_full_name: chararray, games_participations: int, first_game: chararray,
athlete_year_birth: float, athlete_medals: chararray, bio: chararray);

hosts = LOAD 'olympic_hosts.csv' USING PigStorage (',') AS (game_slug: chararray,
game_end_date: chararray, game_start_date: chararray, game_location: chararray,
game_name: chararray, game_season: chararray, game_year: int);

DESCRIBE athletes;

DESCRIBE hosts;
```

a. **Filter athletes participated in the “Tokyo 2020” games**.

```
tokyo_2020_athletes = FILTER athletes BY first_game == 'Tokyo 2020';
DUMP tokyo_2020_athletes;
```

[Note: uses straight quotes (' ')]

b. Filter the games held in “China”.

```
games_in_china = FILTER hosts BY game_location == 'China';
DUMP games_in_china;
```

[Note: uses straight quotes (' ')]

c. Group games by season and count the number of games in each session.

```
grouped_by_season = GROUP hosts BY game_season;
counted_by_season = FOREACH grouped_by_season GENERATE group AS
game_season, COUNT(hosts) AS num_games;
DUMP counted_by_season;
```

d. Filter games that occurred after the year 2000.

```
games_after_2000 = FILTER hosts BY game_year > 2000;
DUMP games_after_2000;
```

PART-B

SPARK PROGRAMS USING PYTHON

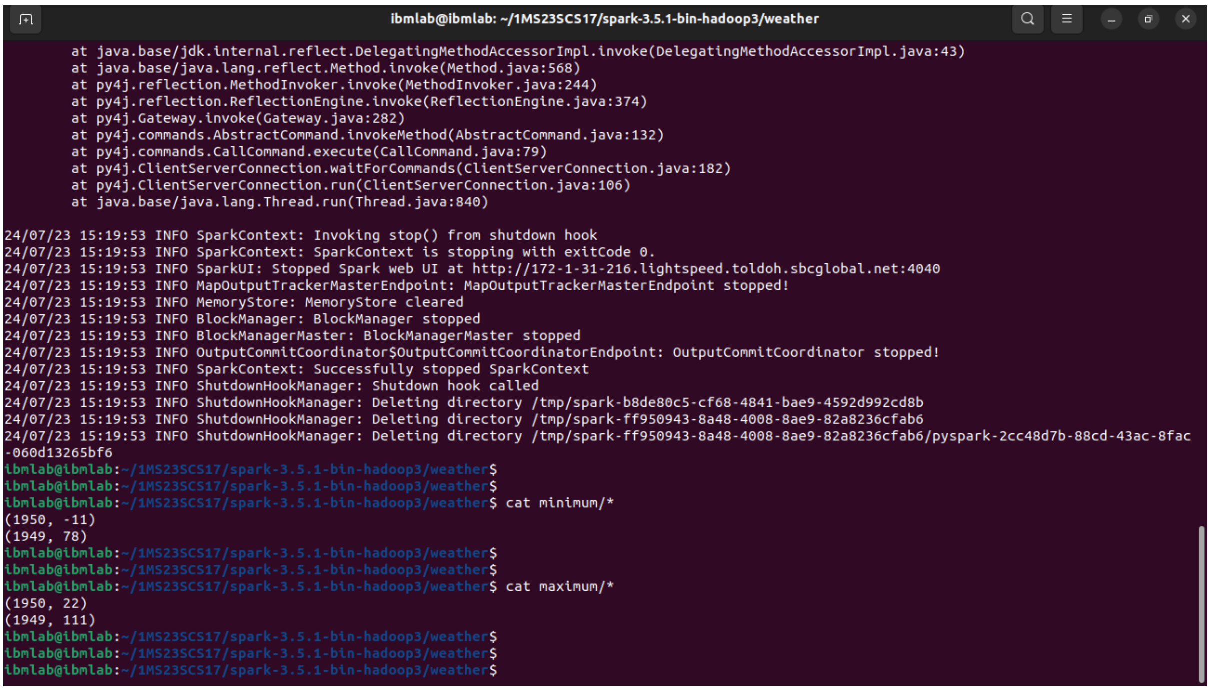
1. Write a **Spark program using Python**, to analyse the given **Weather Report Data** and to generate a report with cities having maximum and minimum temperature for a particular year.

Code: weather.py

```python
import sys
if(len(sys.argv)!=4):
    print("Provide Input File and Output Directory")
    sys.exit(0)

from pyspark import SparkContext
sc =SparkContext()
f = sc.textFile(sys.argv[1])

temp=f.map(lambda x: (int(x[15:19]),int(x[87:92])))

mini=temp.reduceByKey(lambda a,b:a if a<b else b)
mini.saveAsTextFile(sys.argv[2])

maxi=temp.reduceByKey(lambda a,b:a if a>b else b)
maxi.saveAsTextFile(sys.argv[3])
```

Execution:

```
spark-submit weather.py input.txt minimum maximum
```



ibmlab@ibmlab: ~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather

```
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:568)
        at py4j.reflection.MethodInvoker.invoke(MethodInvoker.java:244)
        at py4j.reflection.ReflectionEngine.invoke(ReflectionEngine.java:374)
        at py4j.Gateway.invoke(Gateway.java:282)
        at py4j.commands.AbstractCommand.invokeMethod(AbstractCommand.java:132)
        at py4j.commands.CallCommand.execute(CallCommand.java:79)
        at py4j.ClientServerConnection.waitForCommands(ClientServerConnection.java:182)
        at py4j.ClientServerConnection.run(ClientServerConnection.java:106)
        at java.base/java.lang.Thread.run(Thread.java:840)

24/07/23 15:19:53 INFO SparkContext: Invoking stop() from shutdown hook
24/07/23 15:19:53 INFO SparkContext: SparkContext is stopping with exitCode 0.
24/07/23 15:19:53 INFO SparkUI: Stopped Spark web UI at http://172-1-31-216.lightspeed.toldoh.sbcglobal.net:4040
24/07/23 15:19:53 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
24/07/23 15:19:53 INFO MemoryStore: MemoryStore cleared
24/07/23 15:19:53 INFO BlockManager: BlockManager stopped
24/07/23 15:19:53 INFO BlockManagerMaster: BlockManagerMaster stopped
24/07/23 15:19:53 INFO OutputCommitCoordinator$OutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
24/07/23 15:19:53 INFO SparkContext: Successfully stopped SparkContext
24/07/23 15:19:53 INFO ShutdownHookManager: Shutdown hook called
24/07/23 15:19:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-b8de80c5-cf68-4841-bae9-4592d992cd8b
24/07/23 15:19:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-ff950943-8a48-4008-8ae9-82a8236cfab6
24/07/23 15:19:53 INFO ShutdownHookManager: Deleting directory /tmp/spark-ff950943-8a48-4008-8ae9-82a8236cfab6/pyspark-2cc48d7b-88cd-43ac-8fac
-060d13265bf6
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$ cat minimum/*
(1950, -11)
(1949, 78)
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$ cat maximum/*
(1950, 22)
(1949, 111)
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/weather$
```

1. Write a **Spark program using Python**, to analyse the given **Insurance Data** and generate a statistics report with the construction building name and the count of building/ county name and its frequency.

Code: insurance.py

```python
import sys
from pyspark import SparkContext

if(len(sys.argv)!=4):
    print("Provide Input File and Output Directory")
    sys.exit(0)

sc =SparkContext()
f = sc.textFile(sys.argv[1])

# Construction building or Count of building
temp=f.map(lambda x: (x.split(',')[16],1))
data=temp.countByKey()
dd=sc.parallelize(data.items())
dd.saveAsTextFile(sys.argv[2])

# County name and its frequency
temp=f.map(lambda x: (x.split(',')[2],1))
data=temp.countByKey()
dd=sc.parallelize(data.items())
dd.saveAsTextFile(sys.argv[3])
```

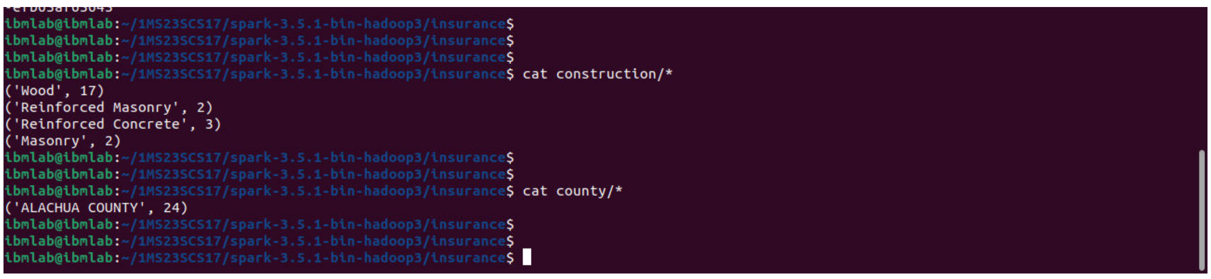
Execution:

```
spark-submit insurance.py input-insurance.csv construction county
```

Output:

$ cat construction/*

$ cat county/*



```
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$ cat construction/*
('Wood', 17)
('Reinforced Masonry', 2)
('Reinforced Concrete', 3)
('Masonry', 2)
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$ cat county/*
('ALACHUA COUNTY', 24)
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
ibmlab@ibmlab:~/1MS23SCS17/spark-3.5.1-bin-hadoop3/insurance$
```

Instruction Guide

JAVA

1. Verify whether you have JAVA installed on your system using the following command.

   ```
   $ javac --version
   ```

2. If not, to install JDK in Linux, use the following command.

   ```
   $ sudo apt install default-jdk
   ```

3. Then, the JRE File of Java will be installed using the following command.

   ```
   $ sudo apt install default-jre
   ```

4. To verify the installation, the following command you can use. It will prompt the Java version used there.

   ```
   $ javac --version
   ```

5. Create a new folder named <1MS23SCS/SCN**> and navigate inside the folder using the following command.

   ```
   $ cd <1MS23SCS/SCN**>
   ```

Hadoop

1. Now download the hadoop tar file to install using the given url.
2. Once you have downloaded hadoop-3.2.2.tar.gz, extract this file with the below command (make sure to check your tar filename).

   ```
   $ tar xvzf hadoop-3.2.2.tar.gz
   ```

3. Now navigate inside the folder using the below command.

   ```
   $ cd hadoop-3.2.2/
   ```

4. Create and open a new ***bash.sh*** file inside the directory.

   ```
   $ gedit bash.sh
   ```

5. We configure the file, copy the below command inside this file and save it.

   ```
   export JAVA_HOME=$(readlink -f $(which javac) | awk 'BEGIN {FS="/bin"} {print $1}')
   export PATH=$(echo $PATH):$(pwd)/bin
   export CLASSPATH=$(hadoop classpath)
   ```

6. Execute the bash.sh File using following command

   ```
   $ source bash.sh
   ```

7. Verify ***JAVA_HOME*** variable to be set to Java Path and ***PATH*** variable has your Hadoop Folder.
8. Verify Hadoop is Installed or not by executing hadoop command. If command gives Information about Hadoop command, then Hadoop is Successfully Installed.

   ```
   $ hadoop
   ```

Spark

1. Verify that hadoop is installed and running in your system.

   ```
   $ hadoop
   ```
2. Download Apache Spark from the given url inside your folder.
3. Now we extract this tar file with the help of below command (make sure to check your tar filename).

   ```
   $ tar xvzf spark-3.5.2-bin-hadoop3.tgz
   ```
4. Now navigate inside the folder using the below command.

   ```
   $ cd spark-3.5.2-bin-hadoop3/
   ```
5. Create a new file named bash.sh inside your folder by using any text editor.

   ```
   $ gedit bash.sh
   ```
6. Copy below code and paste inside bash.sh file

   ```
   export PATH=$(echo $PATH):$(pwd)/bin
   ```
7. In terminal, execute bash.sh file using the following command.

   ```
   $ source bash.sh
   ```
8. Verify spark version with the below command.

   ```
   $ spark-shell --version
   ```

PIG

1. Verify that hadoop is installed and running in your system.

   ```
   $ hadoop
   ```
2. Download Apache Pig from the given url inside your folder.
3. Now we extract this tar file with the help of below command (make sure to check your tar filename).

   ```
   $ tar -xvf pig-0.17.0.tar.gz
   ```
4. Now navigate inside the folder using the below command.

$ cd pig-0.17.0/

5. Create and open a new bash.sh file inside the directory.

   $ gedit bash.sh

6. We configure file, copy the below command inside this file and save it.

   export PIG_INSTALL=$(pwd)

   export PATH=$PATH:$(pwd)/bin

7. Execute the bash.sh File using following command

   $ source bash.sh

[kindly note: goto hadoop folder,run source bash.sh, then goto pig folder,run source bash.sh]

8. You can check your pig version with the below command.

   $ pig -version

9. Once you get it correct that's it we have successfully install pig to our Hadoop single node setup, now we start pig with below pig command.

   $ pig

   [grunt> will be displayed, execute the commands one by one]