

A dark blue vertical bar runs down the left side of the page. A blue arrow points to the right from the bar, containing the text 'Big Data Lab'. Below the bar, several thin, curved lines in shades of blue and grey sweep upwards and to the right.

Big Data Lab

Lab Manual

Contents

| Sl. No. | Topic | Pg. No. |
|--------------------|------------------------|--------------------|
| 1. | Hadoop Installation | 1 |
| 2. | Hadoop Programs | 6 |
| 3. | Spark Installation | 27 |
| 4. | Spark Programs | 33 |
| 5. | Pig Installation | 43 |
| 6. | Pig Programs | 48 |
| 7. | Hive Installation | 51 |
| 8. | Hive Programs | 64 |
| 9. | HBase Installation | 70 |
| 10. | HBase Programs | 77 |
| 11. | Steps to run on docker | 81 |

Hadoop Installation

1.Hadoop Installation

1.1 From Source :-

- Open terminal and cd into Downloads folder.
- Download Hadoop source code tarball using the following command in the terminal.

```
wget  
http://apache.mirrors.lucidnetworks.net/hadoop/common/hadoop-  
3.2.2/hadoop-3.2.2.tar.gz
```

Or download the source code tarball directly from the [website](#).

- Unzip the file

```
tar -xvf hadoop-3.2.2.tar.gz
```

Note: hadoop_path is the path of where the extracted Hadoop folder is present. It can be found by using the following commands :-

```
cd hadoop-3.2.2
```

```
pwd
```

- Set Hadoop globally by updating the system bash file.

```
gedit ~/.bashrc
```

- Paste these 2 lines at the end of the file

```
export HADOOP_HOME=hadoop_path
```

```
export  
CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-  
p-mapreduce-client-core-  
3.2.2.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-  
mapreduce-client-common-  
3.2.2.jar:$HADOOP_HOME/share/hadoop/common/hadoop-  
common-3.2.2.jar:$HADOOP_HOME/lib/*"
```

- Save and close the bashrc file and run the source command to load and save the new variables globally.

```
source ~/.bashrc
```

- Run the following command to set the Hadoop java path

```
echo export JAVA_HOME=/usr/lib/jvm/default-java >>  
$HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

- To verify installation run the following command, it will return a long list of commands.

```
$HADOOP_HOME/bin/hadoop
```

1.2. Installation using a shell script (easier method) :-

Running this shell script automates the complete installation process.

- Open the terminal and create a new file named install-hadoop.sh
- Copy paste the below code into the new file and save it.

```
#!/bin/sh

cd ~/Downloads
echo ""
wget "https://dlcdn.apache.org/hadoop/common/hadoop-3.2.2/hadoop-3.2.2.tar.gz"
echo Downloaded Hadoop-3.2.2 successfully
echo ""
tar -xvf hadoop-3.2.2.tar.gz
echo Unzipped Hadoop-3.2.2 successfully
echo ""
rm hadoop-3.2.2.tar.gz
cd hadoop-3.2.2
HADOOP_HOME=`pwd`
cd ..
echo export HADOOP_HOME=$HADOOP_HOME >> ~/.bashrc
source ~/.bashrc
echo ""
echo HADOOP_HOME set to $HADOOP_HOME
echo ""
echo ""
echo export JAVA_HOME=/usr/lib/jvm/default-java >>
$HADOOP_HOME/etc/hadoop/hadoop-env.sh
echo Hadoop JAVA_HOME set to /usr/lib/jvm/default-java
echo ""
CLASSPATH="$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-client-
core-3.2.2.jar:$HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-
client-common-3.2.2.jar:$HADOOP_HOME/share/hadoop/common/hadoop-common-
3.2.2.jar:$HADOOP_HOME/lib/*"
echo export CLASSPATH=$CLASSPATH >> ~/.bashrc
source ~/.bashrc
echo ""
echo CLASSPATH set to $CLASSPATH
echo ""
echo "Installation completed successfully"
```

After saving the file run the following command.

```
bash install-hadoop.sh
```

2.Steps to run MapReduce programs

- Create a folder for a new program, inside that create 3 files for Mapper, Reducer and Driver classes. Write the relevant business logic in it. In the same folder run the following commands :-

- Compile all java files

```
javac -d . *.java
```

- Set driver class in manifest

```
echo Main-Class: package-name.driver > Manifest.txt
```

- Create an executable jar file

```
jar cfm customname.jar Manifest.txt package-name/*.class
```

- Run the jar file

```
$HADOOP_HOME/bin/hadoop jar customname.jar  
inputfile output
```

- View output

```
cat output/*
```

Note: Replace package-name, customname, inputfile with the relevant names and files used in your program.

A dark blue vertical bar runs down the left side of the slide. A blue arrow points to the right from this bar, containing the text 'Big Data Lab'.

Big Data Lab

Hadoop MapReduce Programs

1. Write a map reduce program to analyze the given weather report data and to generate a report with cities having maximum temperature for a particular year.

driver.java

```
package temp;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class driver {
    public static void main(String args[])throws Exception{
        if(args.length!=2){
            System.out.println("input valid arg");
            System.exit(-1);
        }
        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 temp;
import java.io.IOException;

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

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

    public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> values, Reporter r)
        throws IOException {
        String s=value.toString();
        String s1=s.substring(15, 19);
```

```

        int temperature;

        if(s.charAt(87)=='+'){
            temperature=Integer.parseInt(s.substring(88,92));
        } else{
            temperature=Integer.parseInt(s.substring(87,92));
        }
        values.collect(new Text(s1),new IntWritable(temperature));

    }
}

```

reducer.java

```

package temp;
import java.io.IOException;
import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

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

    public void reduce(Text key, Iterator<IntWritable> value1,
        OutputCollector<Text, IntWritable> values, Reporter r)
        throws IOException {

        int maxvalue=Integer.MIN_VALUE;
        while(value1.hasNext()){
            maxvalue=Math.max(maxvalue, value1.next().get());
        }
        values.collect(key, new IntWritable(maxvalue));
    }
}

```

Manifest.txt

```
Main-Class: temp.driver
```

2. Write a map reduce program to print the multiplication of two different matrices.

MatrixMultiplication.java

```
package Matrix;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
public class MatrixMultiplication {
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
// A is an m-by-n matrix; B is an n-by-p matrix.
conf.set("m", "2");
conf.set("n", "5");
conf.set("p", "3");
Job job = new Job(conf, "MatrixMultiplication");
job.setJarByClass(MatrixMultiplication.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);
job.setMapperClass(MatrixMapper.class);
job.setReducerClass(MatrixReducer.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.waitForCompletion(true);
}
}
```

MatrixMapper.java

```
package Matrix;

import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
public class MatrixMapper extends Mapper<LongWritable, Text,
Text, Text> {
public void map(LongWritable key, Text value, Context
```

```

context) throws IOException, InterruptedException {
Configuration conf = context.getConfiguration();
int m = Integer.parseInt(conf.get("m"));
int p = Integer.parseInt(conf.get("p"));
String line = value.toString();
String[] indicesAndValue = line.split(",");
Text outputKey = new Text();
Text outputValue = new Text();
if (indicesAndValue[0].equals("A")) {
for (int k = 0; k < p; k++) {
outputKey.set(indicesAndValue[1] + "," + k);
outputValue.set("A," + indicesAndValue[2] + "," +
indicesAndValue[3]);
context.write(outputKey, outputValue);
}
} else {
for (int i = 0; i < m; i++) {
outputKey.set(i + "," + indicesAndValue[2]);
outputValue.set("B," + indicesAndValue[1] + "," +
indicesAndValue[3]);
context.write(outputKey, outputValue);
}
}
}
}
}
}

```

MatrixReducer.java

```

package Matrix;

import java.io.IOException;
import java.util.HashMap;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MatrixReducer extends Reducer<Text, Text, Text, Text> {
public void reduce(Text key, Iterable<Text> values, Context
context) throws IOException, InterruptedException {
String[] value;
HashMap<Integer, Float> hashA = new HashMap<Integer,
Float>();
HashMap<Integer, Float> hashB = new HashMap<Integer,
Float>();
for (Text val : values) {
value = val.toString().split(",");
if (value[0].equals("A")) {
hashA.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
} else {

```

```

hashB.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
}
}
int n =
Integer.parseInt(context.getConfiguration().get("n"));
float result = 0.0f;
float a_ij;
float b_jk;
for (int j = 0; j < n; j++) {
a_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
b_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
result += a_ij * b_jk;
}
if (result != 0.0f) {
context.write(null, new Text(key.toString() + "," +
Float.toString(result)));
}}
}

```

Manifest.txt

```

Main-Class: Matrix.MatrixMultiplication

```

3. Write a Map Reduce program to analyze the given Earthquake data and generate statistics.

App.java

```
package earthquake;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

/**
 * The main application class.
 *
 * @author Umer Mansoor
 */
public class App
{
    /**
     * Application entry point.
     * @param args
     * @throws Exception - Bad idea but produces less cluttered code.
     */
    public static void main(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: hadoopex <input path> <output path>");
            System.exit(-1);
        }

        // Create the job specification object
        Job job = new Job();
        job.setJarByClass(App.class);
        job.setJobName("Earthquake Measurment");

        // Setup input and output paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // Set the Mapper and Reducer classes
        job.setMapperClass(EarthquakeMapper.class);
        job.setReducerClass(EarthquakeReducer.class);

        // Specify the type of output keys and values
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        // Wait for the job to finish before terminating
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

```
}  
}
```

EarthquakeMapper.java

```
package earthquake;  
  
import org.apache.hadoop.io.DoubleWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
import java.io.IOException;  
  
/**  
 * This is the main Mapper class.  
 *  
 * @author umermansoor  
 */  
public class EarthquakeMapper extends  
    Mapper<LongWritable, Text, Text, DoubleWritable>  
{  
  
    /**  
     * The `Mapper` function. It receives a line of input from the file,  
     * extracts `region name` and `earthquake magnitude` from it, which becomes  
     * the output. The output key is `region name` and the output value is  
     * `magnitude`.  
     * @param key - Input key - The line offset in the file - ignored.  
     * @param value - Input Value - This is the line itself.  
     * @param context - Provides access to the OutputCollector and Reporter.  
     * @throws IOException  
     * @throws InterruptedException  
     */  
    @Override  
    public void map(LongWritable key, Text value, Context context) throws  
        IOException, InterruptedException {  
  
        String[] line = value.toString().split(",", 12);  
  
        // Ignore invalid lines  
        if (line.length != 12) {  
            System.out.println("- " + line.length);  
            return;  
        }  
  
        // The output `key` is the name of the region  
        String outputKey = line[11];
```

```

        // The output `value` is the magnitude of the earthquake
        double outputValue = Double.parseDouble(line[8]);
        // Record the output in the Context object
        context.write(new Text(outputKey), new DoubleWritable(outputValue));
    }
}

```

EarthquakeReducer.java

```

package earthquake;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;
import org.apache.hadoop.io.Text;
public class EarthquakeReducer extends

    Reducer<Text, DoubleWritable, Text, DoubleWritable>
{

    /**
     * The `Reducer` function. Iterates through all earthquake magnitudes for a
     * region to find the maximum value. The output key is the `region name` and
     * the value is the `maximum magnitude` for that region.
     * @param key - Input key - Name of the region
     * @param values - Input Value - Iterator over quake magnitudes for region
     * @param context - Used for collecting output
     * @throws IOException
     * @throws InterruptedException
     */
    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values,
        Context context) throws IOException, InterruptedException {

        // Standard algorithm for finding the max value
        double maxMagnitude = Double.MIN_VALUE;
        for (DoubleWritable value : values) {
            maxMagnitude = Math.max(maxMagnitude, value.get());
        }

        context.write(key, new DoubleWritable(maxMagnitude));
    }
}

```

Manifest.txt

```
Main-Class: earthquake.App
```


4. Write a Map Reduce program to analyze the given natural numbers and generate statistics for the number as odd or even and print their sum.

driver.java

```
package sum;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class driver extends Configured implements Tool {

    @Override
    public int run(String[] args) throws Exception
    {
        if (args.length < 2)
        {
            System.out.println("Please enter valid arguments");
            return -1;
        }

        JobConf conf = new JobConf(driver.class);
        FileInputFormat.setInputPaths(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));
        conf.setMapperClass mapper.class);
        conf.setReducerClass(reducer.class);
        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);
        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(IntWritable.class);

        JobClient.runJob(conf);
        return 0;
    }

    // Main Method
    public static void main(String args[]) throws Exception
    {
        int exitcode = ToolRunner.run(new driver(), args);
        System.out.println(exitcode);
    }
}
```

mapper.java

```
package sum;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;

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

    @Override
    // Map function
    public void map(LongWritable key, Text value, OutputCollector<Text,
                                                                    IntWritable> output, Reporter rep)
        throws IOException
    {
        // Splitting the line into spaces
        String data[] = value.toString().split(" ");

        for (String num : data)
        {

            int number = Integer.parseInt(num);

            if (number % 2 == 1)
            {
                // For Odd Numbers
                output.collect(new Text("ODD"), new IntWritable(number));
            }

            else
            {
                // For Even Numbers
                output.collect(new Text("EVEN"),
                               new IntWritable(number));
            }
        }
    }
}
```

reducer.java

```
package sum;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;

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

    @Override
    // Reduce Function
    public void reduce(Text key, Iterator<IntWritable> value,
        OutputCollector<Text, IntWritable> output, Reporter rep)
        throws IOException
    {
        // For finding sum and count of even and odd
        // you don't have to take different variables
        int sum = 0, count = 0;
        if (key.equals("ODD"))
        {
            while (value.hasNext())
            {
                IntWritable i = value.next();

                // Finding sum and count of ODD Numbers
                sum += i.get();
                count++;
            }
        }
        else
        {
            while (value.hasNext())
            {
                IntWritable i = value.next();

                // Finding sum and count of EVEN Numbers
                sum += i.get();
                count++;
            }
        }
    }
}
```

```
// First sum then count is printed  
output.collect(key, new IntWritable(sum));  
output.collect(key, new IntWritable(count));  
}  
}
```

Manifest.txt

```
Main-Class: sum.driver
```

5. Write a map-reduce program to analyze the given Insurance data and generate a statistics report.

InsuranceDriver.java

```
package Insurance;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;

public class InsuranceDriver {
    public static void main(String[] args) {
        JobClient my_client = new JobClient();
        // Create a configuration object for the job
        JobConf job_conf = new JobConf(InsuranceDriver.class);

        // Set a name of the Job
        job_conf.setJobName("ConstructionType");

        // Specify data type of output key and value
        job_conf.setOutputKeyClass(Text.class);
        job_conf.setOutputValueClass(IntWritable.class);

        // Specify names of Mapper and Reducer Class
        job_conf.setMapperClass(Insurance.InsuranceMapper.class);
        job_conf.setReducerClass(Insurance.InsuranceReducer.class);

        // Specify formats of the data type of Input and output
        job_conf.setInputFormat(TextInputFormat.class);
        job_conf.setOutputFormat(TextOutputFormat.class);

        // Set input and output directories using command line arguments,
        //arg[0] = name of input directory on HDFS, and arg[1] = name of output
        //directory to be created to store the output file.

        FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));

        my_client.setConf(job_conf);
        try {
            // Run the job
            JobClient.runJob(job_conf);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

InsuranceMapper.java

```
package Insurance;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

public class InsuranceMapper extends MapReduceBase implements Mapper<LongWritable,
Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);

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

        String valueString = value.toString();
        String[] SingleCountryData = valueString.split(",");
        output.collect(new Text(SingleCountryData[16]), one);
    }
}
```

InsuranceReducer.java

```
package Insurance;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.*;

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

    public void reduce(Text t_key, Iterator<IntWritable> values,
OutputCollector<Text,IntWritable> output, Reporter reporter) throws IOException {
        Text key = t_key;
        int frequencyForCountry = 0;
        while (values.hasNext()) {
            // replace type of value with the actual type of our value
            IntWritable value = (IntWritable) values.next();
            frequencyForCountry += value.get();
        }
    }
}
```

```
        output.collect(key, new IntWritable(frequencyForCountry));  
    }  
}
```

Manifest.txt

```
Main-Class: Insurance.InsuranceDriver
```

6. Write a map-reduce program to analyze the given employee record data and generate a statistics report with the total number of female and male employees and their average salary.

driver.java

```
package avg;

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

public class driver {
    public static void main(String args[]) throws Exception {
        if (args.length != 2) {
            System.out.println("input valid arg");
            System.exit(-1);
        }
        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 avg;

import java.io.IOException;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.fs.*;

public class mapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, DoubleWritable> {
    public void map(LongWritable key, Text empRecord, OutputCollector<Text,
DoubleWritable> values1, Reporter r) throws IOException {
```



```

        String[] word = empRecord.toString().split("\\t");
        String sex = word[3];
        Double salary = Double.parseDouble(word[8]);
        values1.collect(new Text(sex), new DoubleWritable(salary));
    }
}

```

reducer.java

```

package avg;

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

public class reducer extends MapReduceBase implements Reducer<Text, DoubleWritable,
Text, DoubleWritable>{
    //last arg is Double

    @Override
    public void reduce(Text arg0, Iterator<DoubleWritable> arg1, OutputCollector<Text,
DoubleWritable> arg2, Reporter arg3)
        throws IOException {
        // TODO Auto-generated method stub
        try {
            Double total = (Double) 0.0;
            int count = 0;
            while (arg1.hasNext()) {
                total += arg1.next().get();
                count++;
            }
            Double avg = (Double) total / count;
            String out = "Total: " + total + " :: " + "Average: " + avg;
            arg2.collect(arg0, new DoubleWritable(avg));
            arg2.collect(arg0, new DoubleWritable(total));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

Manifest.txt

```
Main-Class: avg.driver
```

7. Write a map-reduce program to analyze the given sales records over a period of time.

driver.java

```
package sales;

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.io.*;
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 driver {

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "driver");
        // job.setJarByClass(WordCount.class);
        job.setMapperClass(mapper.class);
        // job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(reducer.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);
    }
}
```

mapper.java

```
package sales;

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

```

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.*;
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 mapper 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);
        // }
        String[] line = value.toString().split(",");
        if(line[0].equals("Transaction_date")){
            return; //header of csv
        }
        // for(String val: line){
        //     System.out.print(val + " | ");
        // }
        // System.out.println();

        String country = "_country_" + line[7];
        String payment_type = "_payment_type_" + line[3];
        int price = Integer.parseInt(line[2]);
        context.write(new Text(country), new IntWritable(price));
        context.write(new Text(payment_type), new IntWritable(1));
    }
}

```

reducer.java

```

package sales;
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.io.*;
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 reducer extends Reducer<Text,IntWritable,Text,IntWritable> {
    // private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
        Context context
        ) throws IOException, InterruptedException {
        String temp = key.toString();
        if(temp.substring(0, 9) == "_country"){
            int total_sales = 0;
            for(IntWritable val: values){
                total_sales+=val.get();
            }
            context.write(key, new IntWritable(total_sales));
        } else{
            int payment_freq = 0;
            for(IntWritable val: values){
                payment_freq+=val.get();
            }
            context.write(key, new IntWritable(payment_freq));
        }
    }
}

```

Manifest.txt

```
Main-Class: sales.driver
```

Spark Installation

1. IntelliJ Installation

1.1 From Source :-

- Download IntelliJ **Community Version** from [here](#).
- Unzip the file
- Using the terminal go to the downloaded folder and cd into the bin folder.
- Run the following command to start the IDE

```
./idea.sh
```

1.2. Installation using a shell script :-

Running this shell script automates the complete installation process.

- Open the terminal and create a new file named install-intellij.sh
- Copy paste the below code into the new file and save it.

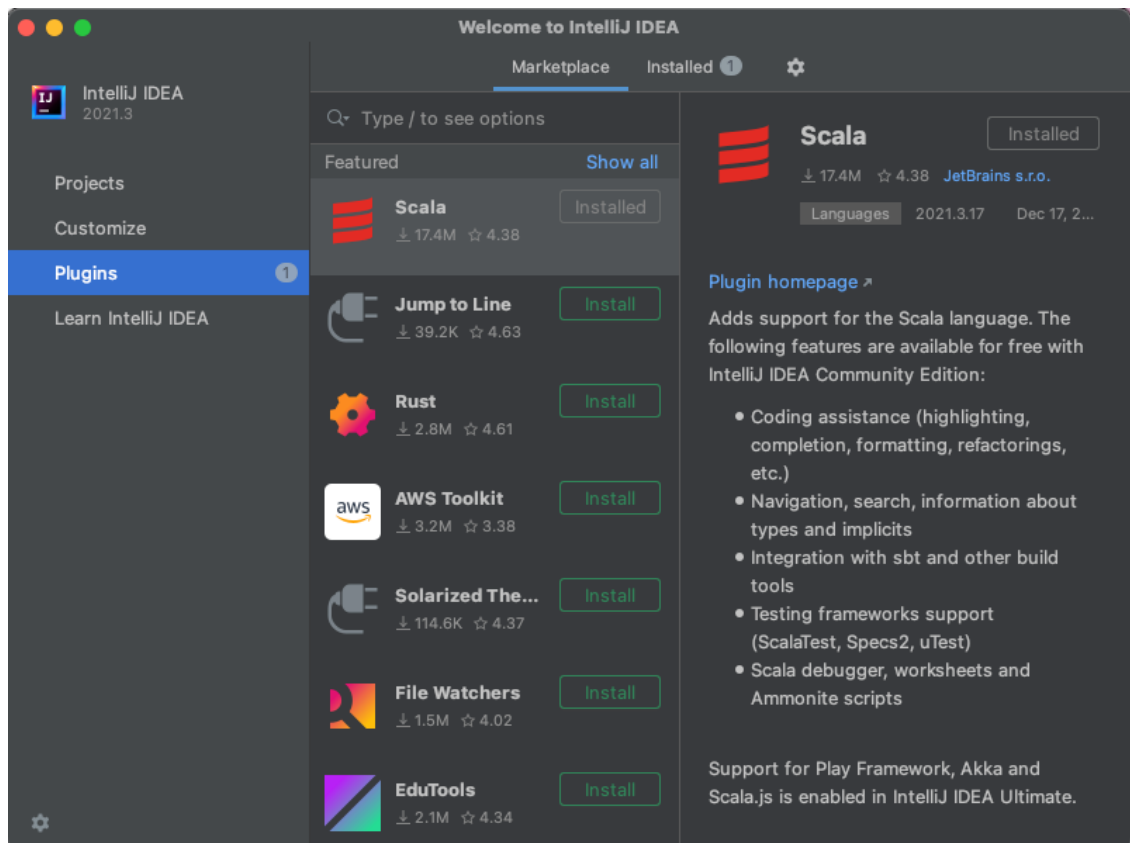
```
#!/bin/sh
cd ~/Downloads
wget "https://download.jetbrains.com/idea/ideaIC-
2021.3.tar.gz?_gl=1*14ggh2a*_ga*MTk0NjM1MjAzNS4xNjQwNTk3NTQz*_ga_V0XZL7QHEB*MTY0MDU
5NzU0My4xLjEuMTY0MDU5Nzg0S4w&_ga=2.226875417.589077562.1640597544-
1946352035.1640597543"
mv "ideaIC-
2021.3.tar.gz?_gl=1*14ggh2a*_ga*MTk0NjM1MjAzNS4xNjQwNTk3NTQz*_ga_V0XZL7QHEB*MTY0MDU
5NzU0My4xLjEuMTY0MDU5Nzg0S4w&_ga=2.226875417.589077562.1640597544-
1946352035.1640597543" "ideaIC-2021.3.tar.gz"
sudo tar -xvzf ideaIC-2021.3.tar.gz
cd ~/Downloads/
cd idea-IC-213.5744.223//bin
./idea.sh
```

After saving the file run the following command.

```
bash install-intellij.sh
```

2. Setup IntelliJ for Spark

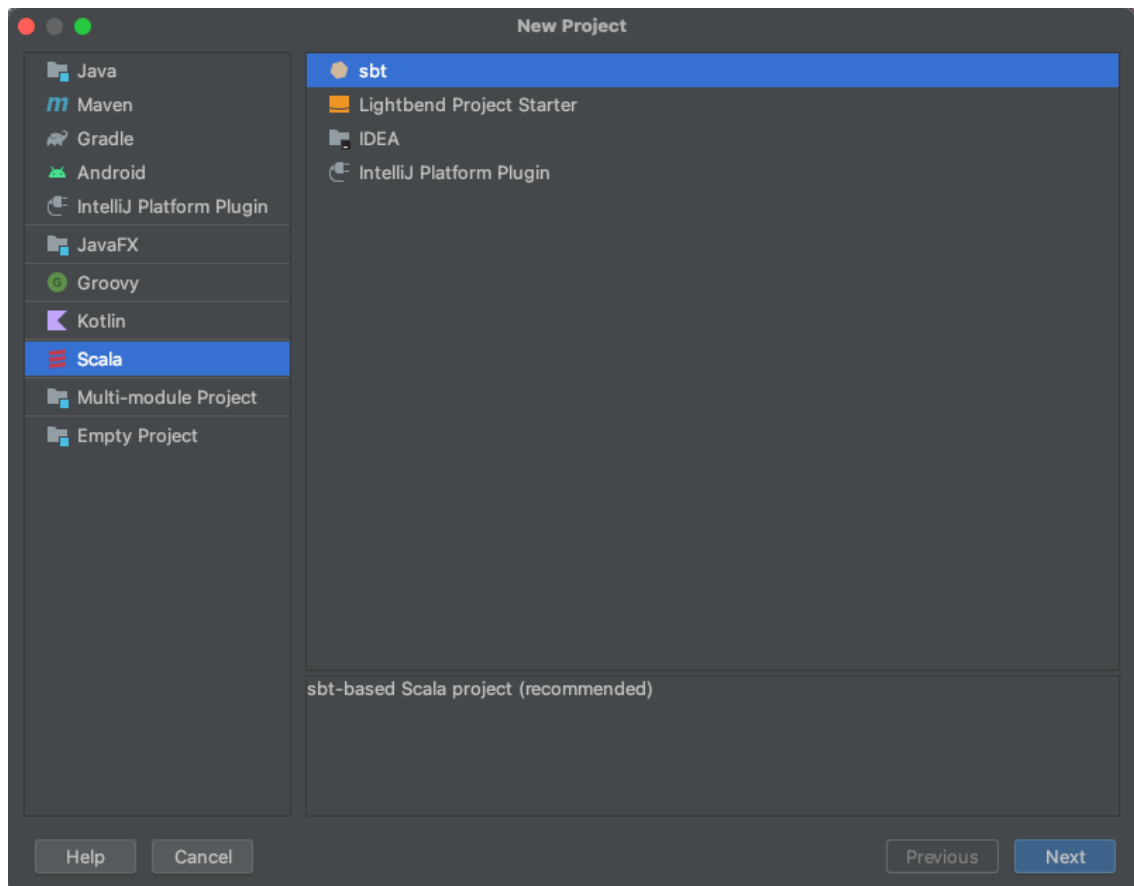
- Navigate to Plugins on the taskbar and install Scala.



- Restart the IDE once the plugin is installed

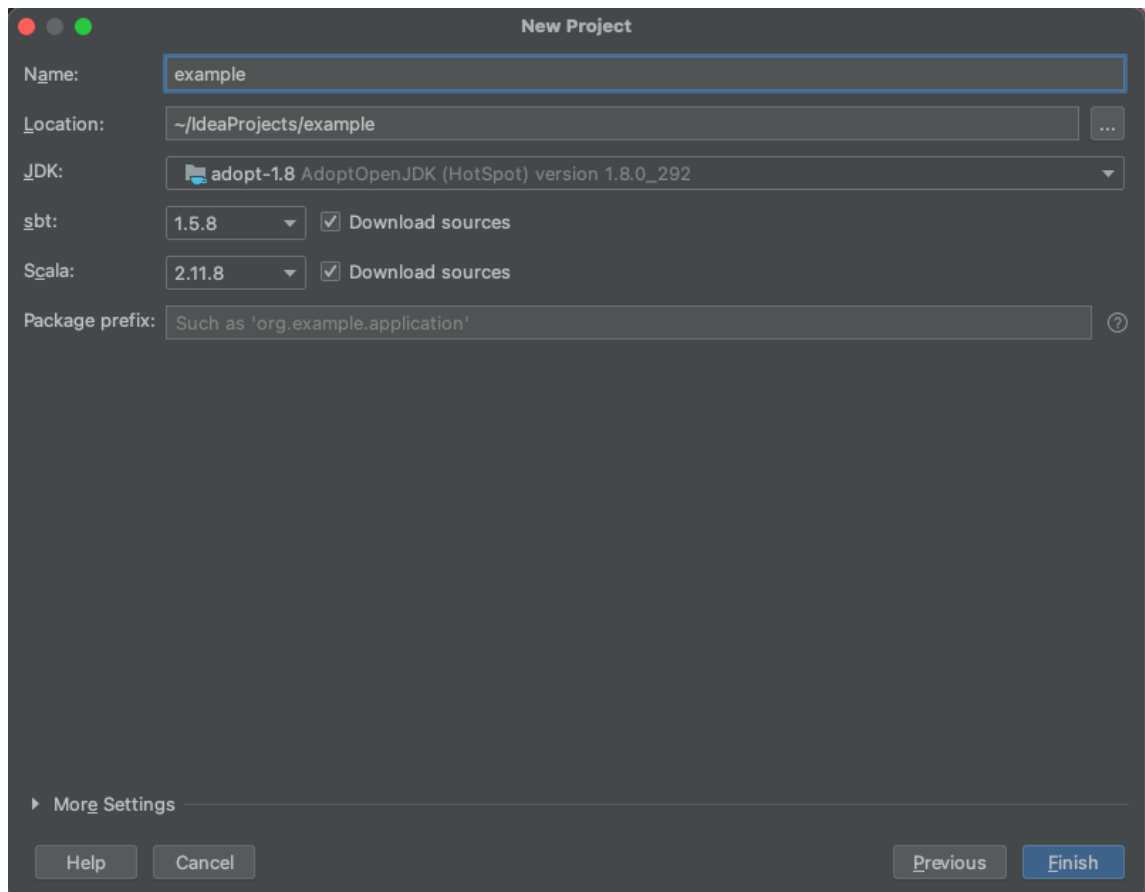
3. Creating Scala Project

- Click on create a project, select scala and use sbt option(default) to create an sbt-based scala project.



- Project settings :-

- Select JDK : 1.8
- SBT version : 1.5.8, check download sources
- Scala version : 2.11.8, check download sources

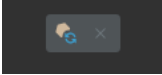


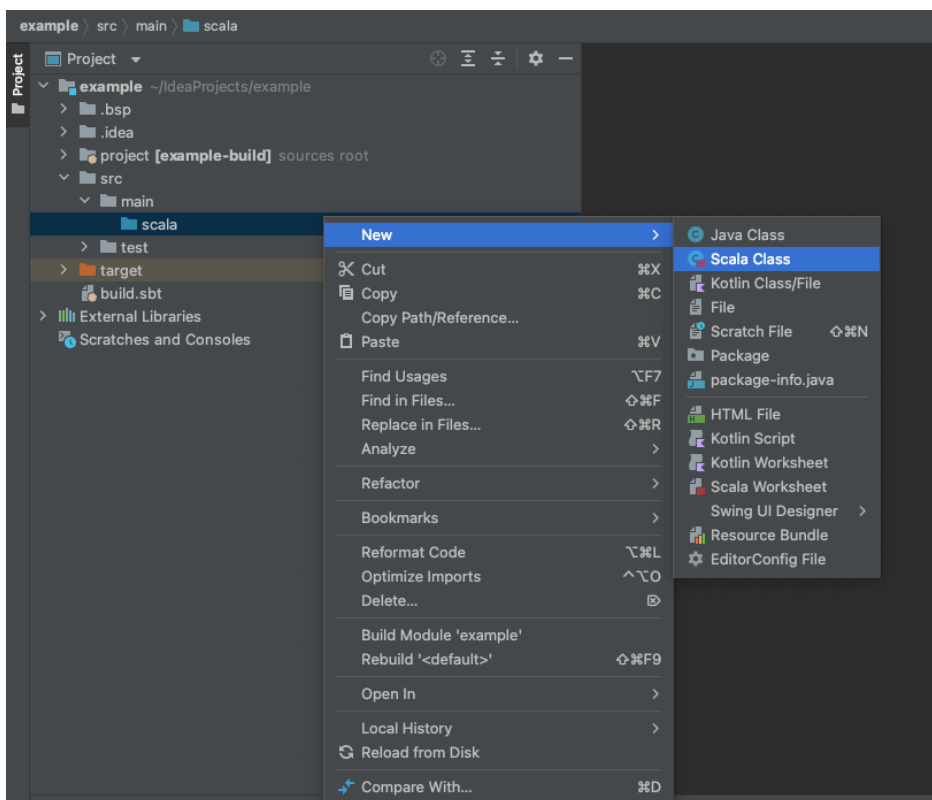
Note: if the versions aren't set properly the programs won't run due to version mismatch errors.

- Open build.sbt file and add the following dependencies

```
val sparkVersion = "2.4.0"

libraryDependencies += Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion,
  "org.apache.spark" %% "spark-sql" % sparkVersion,
)
```

-  When this icon appears in your project please click on it to sync your changes made in the build.sbt file.
- Once the build is successful, head over to src/main/scala. Click on *new Scala class*. Give a file name and create a Scala Object .Write the business logic here and run the file.



Note: If you are unable to create a new scala class then the IDE is downloading some important files from the internet, please look at the progress bar found near the bottom right corner of the screen. Once the progress bar finishes all processes and vanishes you will be able to create a new scala class/object.

A dark blue vertical bar runs down the left side of the slide. A blue arrow points to the right from this bar, containing the text 'Big Data Lab'.

Big Data Lab

Spark Programs

1. Demonstrate case classes and tuples in spark by creating an object, copying it, and checking for equality.

CaseClassExample.scala

```
case class CaseClassExample(v1: String, v2: String)

object Test {
  def main(args: Array[String]) : Unit = {
    val caseClassExampleTest = CaseClassExample("abc", "def")

    println(caseClassExampleTest.v1)
    println(caseClassExampleTest.v2)
  }
}
```

build.sbt

```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.11.8"

lazy val root = (project in file("."))
  .settings(
    name := "Case Class & Tuples"
  )
```

2. Demonstrate WordCount using spark with scala.

WordCount.scala

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.rdd.RDD

object WordCount {
  def main(args: Array[String]) {
    val file = "src/main/words.txt"
    val spark: SparkSession = SparkSession.builder
      .appName("Word Count")
      .config("spark.master", "local")
      .getOrCreate()
    val fileRdd: RDD[String] = spark.sparkContext.textFile(file)

    // create the counts
    val counts = fileRdd.map(_._replaceAll("[.,]", ""))
      .map(_._replace("-", " "))
      .flatMap(line => line.split(" "))
      .map(word => (word, 1))
      .reduceByKey(_ + _)
      .sortBy(_._2)
      .collect

    println( "-----")
    counts.foreach(println)
    println( "-----")

    spark.stop()
  }
}
```

build.sbt

```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.11.8"

lazy val root = (project in file("."))
  .settings(
    name := "WordCount"
  )

val sparkVersion = "2.4.0"

libraryDependencies += Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion,
  "org.apache.spark" %% "spark-sql" % sparkVersion,
)
```

words.txt

```
hi  
hello  
hey  
hi  
hi
```

3. Demonstrate use of aggregate functions (`approx_count_distinct()`, `collect_list()`, `collect_set()`, `avg()`, `count()`, `countDistinct()`) by using SQL libraries in spark.

AggregateFunctions.scala

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

object AggregateFunctions extends App {
  val spark: SparkSession = SparkSession.builder()
    .master("local[1]")
    .appName("SparkByExamples.com")
    .getOrCreate()

  spark.sparkContext.setLogLevel("ERROR")

  import spark.implicits._

  val simpleData = Seq(("James", "Sales", 3000),
    ("Michael", "Sales", 4600),
    ("Robert", "Sales", 4100),
    ("Maria", "Finance", 3000),
    ("James", "Sales", 3000),
    ("Scott", "Finance", 3300),
    ("Jen", "Finance", 3900),
    ("Jeff", "Marketing", 3000),
    ("Kumar", "Marketing", 2000),
    ("Saif", "Sales", 4100)
  )
  val df = simpleData.toDF("employee_name", "department", "salary")

  println("approx_count_distinct: "+
    df.select(approx_count_distinct("salary")).collect()(0)(0))

  println("avg: "+ df.select(avg("salary")).collect()(0)(0))

  df.select(collect_list("salary")).show(false)

  df.select(collect_set("salary")).show(false)

  val df2 = df.select(countDistinct("department", "salary"))
  df2.show(false)
  println("Distinct Count of Department & Salary: "+df2.collect()(0)(0))

  println("count: "+
    df.select(count("salary")).collect()(0))
}
```

build.sbt

```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.11.8"

lazy val root = (project in file("."))
  .settings(
    name := "Aggregate Functions"
  )

val sparkVersion = "2.4.0"

libraryDependencies += Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion, "org.apache.spark" %%
  "spark-sql" % sparkVersion,
)
```


4. Demonstrate basic dataframe transformation functions by using SQL libraries in spark.

TransformationFunctions.scala

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

object TransformationFunctions extends App {
  val spark: SparkSession = SparkSession.builder()
    .master("local[1]")
    .appName("SparkByExamples.com")
    .getOrCreate()

  spark.sparkContext.setLogLevel("ERROR")

  import spark.implicits._

  val simpleData = Seq(("James", "Sales", 3000),
    ("Michael", "Sales", 4600),
    ("Robert", "Sales", 4100),
    ("Maria", "Finance", 3000),
    ("James", "Sales", 3000),
    ("Scott", "Finance", 3300),
    ("Jen", "Finance", 3900),
    ("Jeff", "Marketing", 3000),
    ("Kumar", "Marketing", 2000),
    ("Saif", "Sales", 4100)
  )

  val df = simpleData.toDF("employee_name", "department", "salary")
  //show all
  df.select("*").show(truncate = false)

  //first row
  df.select(first("salary")).show(false)

  //last row
  df.select(last("salary")).show(false)

  //Select first 2 columns.
  df.select(df.columns.slice(0,2).map(m=>col(m)):_*).show()

  //Selects 3rd column (index starts from zero)
  df.select(df.columns(2)).show()

  //Selects columns from index 1 to 3
  df.select(df.columns.slice(1,3).map(m=>col(m)):_*).show()
}
```

build.sbt

```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.11.8"

lazy val root = (project in file("."))
  .settings(
    name := "Transformation Functions"
  )

val sparkVersion = "2.4.0"
libraryDependencies += Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion,
  "org.apache.spark" %% "spark-sql" % sparkVersion,
)
```

5. Demonstrate basic math functions on a dataframe using SQL libraries in spark.

MathFunctions.scala

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

object MathFunctions extends App {
  val spark: SparkSession = SparkSession.builder()
    .master("local[1]")
    .appName("SparkByExamples.com")
    .getOrCreate()

  spark.sparkContext.setLogLevel("ERROR")

  import spark.implicits._

  val simpleData = Seq(("James", "Sales", 3000),
    ("Michael", "Sales", 4600),
    ("Robert", "Sales", 4100),
    ("Maria", "Finance", 3000),
    ("James", "Sales", 3000),
    ("Scott", "Finance", 3300),
    ("Jen", "Finance", 3900),
    ("Jeff", "Marketing", 3000),
    ("Kumar", "Marketing", 2000),
    ("Saif", "Sales", 4100)
  )
  val df = simpleData.toDF("employee_name", "department", "salary")

  df.select(max("salary")).show(false)

  df.select(min("salary")).show(false)

  df.select(mean("salary")).show(false)

  df.select(skewness("salary")).show(false)

  df.select(stddev("salary"), stddev_samp("salary"),
    stddev_pop("salary")).show(false)

  df.select(sum("salary")).show(false)

  df.select(sumDistinct("salary")).show(false)

  df.select(variance("salary"), var_samp("salary"),
    var_pop("salary")).show(false)
}
```

build.sbt

```
ThisBuild / version := "0.1.0-SNAPSHOT"

ThisBuild / scalaVersion := "2.11.8"

lazy val root = (project in file("."))
  .settings(
    name := "Math Functions"
  )

val sparkVersion = "2.4.0"
libraryDependencies += Seq(
  "org.apache.spark" %% "spark-core" % sparkVersion,
  "org.apache.spark" %% "spark-sql" % sparkVersion,
)
```

PIG Installation

1. Pig Installation

1.1 From Source :-

- Open terminal and cd into Downloads folder.
- Download Pig source code tarball using the following command in the terminal.

```
wget https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz
```

Or download the source code tarball directly from the [website](#).

- Unzip the file

```
tar -xvf pig-0.17.0.tar.gz
```

Note: pig_path is the path of where the extracted PIG folder is present.

It can be found by using the following commands.

```
cd pig-0.17.0
```

```
pwd
```

- Set PIG globally by updating the system bash file.

```
gedit ~/.bashrc
```

- Paste the following code at the end of the file

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

```
export PIG_HOME=pig_path
```

```
export PATH=$PATH:$PIG_PATH/bin
```

```
export PIG_CLASSPATH=$HADOOP_HOME/conf
```

- Save and close the bashrc file and run the source command to load and save the new variables globally.

```
source ~/.bashrc
```

- To verify installation run the following command, it will return pig version number.

```
pig -version
```

1.2. Installation using a shell script (easier method) :-

Running this shell script automates the complete installation process.

- Open the terminal and create a new file named install-pig.sh
- Copy paste the below code into the new file and save it.

```
#!/bin/sh

cd ~/Downloads
echo ""
wget "https://downloads.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz"
echo Downloaded pig-0.17.0 successfully
echo ""
tar -xvf pig-0.17.0.tar.gz
echo Unzipped pig-0.17.0 successfully
echo ""
rm pig-0.17.0.tar.gz
cd pig-0.17.0
PIG_PATH=`pwd`
cd ..
echo export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64 >> ~/.bashrc
source ~/.bashrc
echo ""
echo JAVA_HOME set to /usr/lib/jvm/java-8-openjdk-amd64
echo ""
echo export PIG_HOME=$PIG_PATH >> ~/.bashrc
source ~/.bashrc
echo ""
echo PIG_HOME set to $PIG_PATH
echo ""
echo export PATH=$PATH:$PIG_PATH/bin >> ~/.bashrc
source ~/.bashrc
echo ""
echo PATH set to $PATH:$PIG_PATH/bin
echo ""
echo export PIG_CLASSPATH=$HADOOP_HOME/conf >> ~/.bashrc
source ~/.bashrc
echo ""
echo PIG_CLASSPATH set to $HADOOP_HOME/conf
echo ""
echo "Installation completed successfully"
```

After saving the file run the following command.

```
bash install-pig.sh
```


2. Steps to run Pig programs

- Create a pig script and write the business logic in this file

```
gedit file_name.pig
```

- To run the code

```
pig -x local file_name.pig
```

Pig Programs

1. Write a program to filter data by city.

pig_script_filter.pig

```
student_details = LOAD '/home/msrit/Downloads/pig/test/Filter/student_details.txt'
USING PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray);

filter_data = FILTER student_details BY city == 'Chennai';

Dump filter_data;
```

2. Write a program to group students by age.

pig_script_grouping.pig

```
student = LOAD '/home/msrit/Downloads/pig/test/Grouping/student_details.txt' USING
PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray);

group_data = GROUP student by age;

Dump group_data;
```

3. Write a program to join 2 tables by id and order by customer_id

pig_script_joining.pig

```
customers = LOAD '/home/msrit/Downloads/pig/test/Joining/customer.txt' USING
PigStorage(',') as (id:int, name:chararray, age:int, address:chararray,
salary:int);

orders = LOAD '/home/msrit/Downloads/pig/test/Joining/order.txt' USING
PigStorage(',') as (oid:int, date:chararray, customer_id:int, amount:int);

join_result = JOIN customers BY id, orders BY customer_id;

Dump join_result
```

4. Write a program to sort student by age in descending order and limit the results to a maximum of 4 records.

pig_script_sorting.pig

```
student = LOAD '/home/msrit/Downloads/pig/test/Sorting/student_details.txt' USING
PigStorage(',') as (id:int, firstname:chararray, lastname:chararray, age:int,
phone:chararray, city:chararray);

student_order = ORDER student BY age DESC;

student_limit = LIMIT student_order 4;

Dump student_limit;
```

5. Write a program to create a union between 2 tables.

pig_script_union.pig

```
cust1 = LOAD '/home/msrit/Downloads/pig/test/Union/customer1.txt' USING
PigStorage(',') as (id:int, firstname:chararray, lastname:chararray,
phone:chararray, city:chararray);

cust2 = LOAD '/home/msrit/Downloads/pig/test/Union/customer2.txt' USING
PigStorage(',') as (id:int, firstname:chararray, lastname:chararray,
phone:chararray, city:chararray);

cust = UNION cust1, cust2;

Dump cust
```

Hive Installation

Hadoop Setup For Hive

- Check your java version, if it is not Java8, then you need to install Java8 and set it as the default version.

```
java -version
```

- Steps to install java 8

```
sudo apt update  
sudo apt install openjdk-8-jdk -y
```

- Select java 8 version as the default version from the existing versions.

```
sudo update-alternatives --config java  
(Type the selection number corresponding to java 8)
```

- -Create a java.sh file and export the JAVA_HOME and PATH variables

```
vi java8.sh
```

- Type the below under java8.sh file

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-  
amd64/  
export PATH=$PATH:$JAVA_HOME
```

- Bash the file

```
bash java8.sh
```

- Export java path

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

- Setting up a Non-Root User for Hadoop Environment. Therefore, Install the OpenSSH server and client using the following command:

```
sudo apt install openssh-server openssh-client -y
```

- Generate an SSH key pair and define the location it is to be stored in:

```
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
```

- Use the cat command to store the public key as authorized_keys in the ssh directory:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

- Set the permissions for your user with the chmod command:

```
chmod 0600 ~/.ssh/authorized_keys
```

- The new user is now able to SSH without needing to enter a password every time. Verify everything is set up correctly by using the hdoop user to SSH to localhost:

```
ssh localhost
```

- Open and Edit the .bashrc shell configuration file

```
sudo vi .bashrc
```

- Once you add the variables, save and exit the .bashrc file. It is vital to apply the changes to the current running environment by using the following command :-

#Hadoop Related Options

```
export HADOOP_HOME=/home/msrit/Downloads/hadoop-3.2.2 export HADOOP_INSTALL=$HADOOP_HOME export HADOOP_MAPRED_HOME=$HADOOP_HOME export HADOOP_COMMON_HOME=$HADOOP_HOME export HADOOP_HDFS_HOME=$HADOOP_HOME export YARN_HOME=$HADOOP_HOME export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native" export HIVE_HOME=/home/msrit/Downloads/apache-hive-3.1.2-bin export PATH=$PATH:$HIVE_HOME/bin
```

```
source ~/.bashrc
```

- Open the hadoop-env.sh file and make few changes:

```
sudo vi $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

- Uncomment the \$JAVA_HOME variable (i.e., remove the # sign) and add the full path to the OpenJDK installation on your system.

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

- Open the core-site.xml file

```
sudo vi $HADOOP_HOME/etc/hadoop/core-site.xml
```


- add the following configuration in between `<configuration>` and `</configuration>` to override the default values for the temporary directory and add your HDFS URL to replace the default local file system setting:

```
<configuration>
<property>
  <name>hadoop.tmp.dir</name>
  <value>/home/msrit/Downloads/tmpdata</value>
</property>
<property>
  <name>fs.default.name</name>
  <value>hdfs://127.0.0.1:9000</value>
</property>
</configuration>
```

- Use the following command to open the `hdfs-site.xml` file for editing:

```
sudo vi $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

- Add the following configuration in between `<configuration>` and `</configuration>`

```
<property>
<name>dfs.data.dir</name>
<value>/home/msrit/Downloads/dfsdata/namenode</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>/home/msrit/Downloads/dfsdata/datanode</value>
</property>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
```

- Use the following command to access the mapred-site.xml file and define MapReduce values:

```
sudo vi $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

- Add the following configuration in between <configuration> and </configuration> to change the default MapReduce framework name value to yarn:

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

- Open the yarn-site.xml file in a text editor:

```
sudo vi $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

- Add the following configuration in between <configuration> and </configuration>

```
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
<property>
  <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
  <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
<property>
  <name>yarn.resourcemanager.hostname</name>
  <value>127.0.0.1</value>
</property>
<property>
  <name>yarn.acl.enable</name>
  <value>0</value>
</property>
<property>
  <name>yarn.nodemanager.env-whitelist</name>
  <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_
_DIR,CLASSPATH_PERPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_H
OME</value>
</property>
```

- Format the NameNode before starting Hadoop services for the first time

```
hdfs namenode -format
```

- cd to sbin directory of hadoop

```
cd Downloads/hadoop-3.2.2/sbin
```

- execute the following commands to start the NameNode and DataNode:

```
./start-dfs.sh
```

- Once the namenode, datanodes, and secondary namenode are up and running, start the YARN resource and nodemanagers by typing:

```
./start-yarn.sh
```

- Type this simple command to check if all the daemons are active and running as Java processes:

```
jps
```

- cd to the main directory

```
cd ..  
cd ..  
cd ..
```

Hive Installation

- Open terminal and cd into Downloads folder.
- Download Hive source code tarball using the following command in the terminal.

```
wget https://downloads.apache.org/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

Or download the source code tarball directly from the [website](#).

- Unzip the file and go to home directory.

```
tar xzf apache-hive-3.1.2-bin.tar.gz
```

```
cd ..
```

- Set Hive globally by updating the system bash file.

```
gedit ~/.bashrc
```

- Paste these 2 lines at the end of the file

```
export HIVE_HOME=/home/msrit/Downloads/apache-hive-3.1.2-bin
```

```
export PATH=$PATH:$HIVE_HOME/bin
```

- Save and close the bashrc file and run the source command to load and save the new variables globally.

```
source ~/.bashrc
```

- Access the *hive-config.sh* file using the previously created **\$HIVE_HOME** variable:

```
sudo gedit $HIVE_HOME/bin/hive-config.sh
```

- Add the **HADOOP_HOME** variable and the full path to your Hadoop directory in *hive-config.sh* file. Save the edits and exit the *hive-config.sh* file.

```
export HADOOP_HOME=/home/msrit/hadoop-3.2.1
```

- Create a *tmp* directory within the HDFS storage layer. This directory is going to store the intermediary data Hive sends to the HDFS:

```
hdfs dfs -mkdir /tmp
```

- Add write and execute permissions to tmp group members:

```
hdfs dfs -chmod g+w /tmp
```

- Check if the permissions were added correctly:

```
hdfs dfs -ls /
```

- Create the *warehouse* directory within the */user/hive/* parent directory:

```
hdfs dfs -mkdir -p /user/hive/warehouse
```

- Add **write** and **execute** permissions to *warehouse* group members:

```
hdfs dfs -chmod g+w /user/hive/warehouse
```

- Check if the permissions were added correctly:

```
hdfs dfs -ls /user/hive
```

Apache Hive distributions contain template configuration files by default. The template files are located within the Hive *conf* directory and outline default Hive settings.

- Use the following command to locate the correct file:

```
cd $HIVE_HOME/conf
```

- Use the *hive-default.xml.template* to create the *hive-site.xml* file:

```
cp hive-default.xml.template hive-site.xml
```

- Access the *hive-site.xml* file using the nano text editor:

```
sudo gedit hive-site.xml
```

- You can configure the system to use your local storage rather than the HDFS layer by setting the *hive.metastore.warehouse.dir* parameter value to the location of your Hive *warehouse* directory.

Check the below lines are there in *hive-site.xml* file.

```
<property>
<name>hive.metastore.warehouse.dir</name>

<value>/user/hive/warehouse</value>
<description>location of default database for the warehouse
</description>
</property>
```

- Paste these lines to *hive-site.xml*:

```
<property>
<name>system:java.io.tmpdir</name>
<value>/tmp/hive/java</value>
</property>
<property>
<name>system:user.name</name>
<value>${user.name}</value>
</property>
<property>
<name>javax.jdo.option.ConnectionURL</name>
<value>jdbc:derby:/opt/hive-3.1.2-
bin/metastore_db;databaseName=metastore_db;create=true</value>
</property>
```

- Change the lines in *hive-site.xml* to given below lines:

```
<property>
<name>hive.txn.xlock.iow</name>
<value>true</value>
<description>
Ensures commands with OVERWRITE (such as INSERT
OVERWRITE) acquire Exclusive locks for transactional tables. This
ensures that inserts (w/o overwrite) running concurrently
are not hidden by the INSERT OVERWRITE.
</description>
</property>
```

- Locate the **guava jar** file in the Hive *lib* directory:

```
ls $HIVE_HOME/lib
```

- Locate the **guava jar** file in the Hadoop *lib* directory as well:

```
ls $HADOOP_HOME/share/hadoop/hdfs/lib
```

- Remove the existing **guava** file from the Hive *lib* directory:

```
rm $HIVE_HOME/lib/guava-19.0.jar
```

- Copy the **guava** file from the Hadoop *lib* directory to the Hive *lib* directory:

```
cp $HADOOP_HOME/share/hadoop/hdfs/lib/guava-27.0-jre.jar  
$HIVE_HOME/lib/
```

- Run the given command:

```
rm -rf metastore_db
```

- Use the **schematool** command once again to initiate the Derby database:

```
$HIVE_HOME/bin/schematool -dbType derby -initSchema
```


Steps To Run Hive

- Start the Hive command-line interface using the following commands:

```
cd $HIVE_HOME/bin
```

```
hive
```

Hive Programs

1.

- a. Create table “employees” with 6 attributes such as Id, Name, Age, Address, Salary, Department and load data into employees table from Employees.txt file.
- b. Display the list of corresponding employee id, name and their address.
- c. Write a query to group all the employees by their department and display the results.
- d. Create another table department with the attributes dno and dname. Now display the department id in which each employee works along with the employee id and their age.
- e. Drop the table employees

Ans.

1.a

```
create table employees(Id int, Name string, Age int, Address string, Salary float, DeptId int) row format delimited fields terminated by ',';  
  
load data local inpath '/home/msrit/Downloads/Employees.txt' into TABLE employees;
```

1.b

```
SELECT Id, Name, Address from employees;
```

1.c

```
SELECT DeptId, count(*) FROM employees GROUP BY DeptId;
```

1.d

```
create table department(DeptId int, DeptName string) row format delimited fields terminated by ',';  
  
load data local inpath '/home/msrit/Downloads/Departments.txt' into TABLE department;  
  
select e.DeptId, e.Name, e.Id, e.Age from employees e join department d on e.DeptId = d.DeptId;
```

1.e

```
drop table employees;
```

2.

- a. Create table “employees” with 6 attributes such as Id, Name, Age, Address, Salary, Department and load data into employees table from Employees.txt file
- b. Display the total number of employees whose details are present in the employees table.
- c. Write a query to sort the employee details by their ‘id’ in descending order.
- d. Create a view ‘employee_view’ by taking id and name of employee from ‘employees’ table and display the contents of the view.
- e. Drop the view and the table

Ans.

2.a

```
create table employees(Id int, Name string, Age int, Address string, Salary float, DeptId int) row format delimited fields terminated by ',';  
load data local inpath '/home/msrit/Downloads/Employees.txt' into TABLE employees;
```

2.b

```
select count(*) from employees;
```

2.c

```
SELECT * from employees SORT BY Id DESC;
```

2.d

```
create view employee_view as select Id, Name from employees;  
SELECT * from employee_view;
```

2.e

```
drop view employee_view;  
drop table employees;
```

3.

- a. Create table “employees” with 6 attributes such as Id, Name, Age, Address, Salary, Department and load data into employees table from Employees.txt file.
- b. Write a query to display the details of the employees from the ‘employees’ table ordered by the ‘department’ attribute using the order by clause.
- c. Create a table to static_part_employee with attributes id int, name String and partition it with the attribute salary int.
- d. Insert values into static_part_employee by overwriting the details from employees table and pass the value of salary as 25000 for partition.
- e. Display the static_part_employee table and then drop the same table

Ans.

3.a

```
create table employees(Id int, Name string, Age int, Address string, Salary float, DeptId int) row format delimited fields terminated by ',';
load data local inpath '/home/msrit/Downloads/Employees.txt' into TABLE employees;
```

3.b

```
SELECT * FROM employees ORDER BY department;
```

3.c

```
create table if not exists static_part_employee(Id int, name string) partitioned by (salary float) row format delimited fields terminated by ',';
```

3.d

```
insert overwrite table static_part_employee partition (salary = 25000) select Id, name
from employees where salary = 25000;
```

3.e

```
select * from static_part_employee;

drop table static_part_employee;
```

4.

- a. Create table “employees” with 6 attributes such as Id, Name, Age, Address, Salary, Department and load data into employees table from Employees.txt file.
- b. Create a table employee_rc with attributes (id int, name string, salary int) and store it as a rcfile.
- c. Insert values into the employee_ec by overwriting values from employee table.
- d. Display the employee and employee_rc table and also total salary given to all employees.
- e. Write a query to display names of employees whose salary is more than 25000.

Ans.

4.a.

```
create table employees(Id int, Name string, Age int, Address string, Salary float, DeptId int) row format delimited fields terminated by ',';
```

```
load data local inpath '/home/msrit/Downloads/Employees.txt' into TABLE employees;
```

4.b

```
create table employee_rc(Id int, name string, salary float) stored as rcfile;
```

4.c.

```
insert overwrite table employee_rc select Id, name, salary from employees;
```

4.d.

```
select * from employees;

select * from employee_rc;

select sum(salary) from employee_rc;
```

4.e.

```
select name from employees where salary>25000;
```

5.

- a. Create table “employees” with 6 attributes such as Id, Name, Age, Address, Salary, Department and load data into employees table from Employees.txt file.
- b. Create a table to dynamic_part_employee with attributes id int, name String and partition it with the attribute salary int.
- c. Insert values into dynamic_part_employee by overwriting the details from employees table and partition by salary. Display the dynamic_part_employee table.
- d. Write a query to display the average, minimum and maximum salary obtained by the employees.
- e. Write a query to display the number of employees under each department

Ans.

5.a.

```
create table employees(Id int, Name string, Age int, Address string, Salary float, DeptId int) row format delimited fields terminated by ',';
```

```
load data local inpath '/home/msrit/Downloads/Employees.txt' into TABLE employees;
```

5.b.

```
create table if not exists dynamic_part_employee(Id int, name string) partitioned by (salary float) row format delimited fields terminated by ',';
```

```
set hive.exec.dynamic.partition=true;
```

```
set hive.exec.dynamic.partition.mode=nonstrict;
```

5.c.

```
insert overwrite table dynamic_part_employee partition (salary) select Id, name, salary from employees;
```

```
select * from dynamic_part_employee;
```

5.d.

```
select avg(salary), min(salary), max(salary) from employees;
```

5.e.

```
SELECT DeptId, count(*) FROM employees GROUP BY DeptId;
```

HBase Installation

1. HBase Installation

1. From Source :-

- Open terminal and make sure you are inside the home directory.
- Download HBase source code tarball using the following command in the terminal.

```
wget https://dlcdn.apache.org/hbase/2.4.9/hbase-2.4.9-bin.tar.gz
```

- Unzip the file

```
tar -xvf hbase-2.4.9-bin.tar.gz
```

Note: Make sure Hadoop is installed already.
It can be found by using the following commands.

```
cd ~
```

```
hadoop version
```

Note:

1. Follow the Hadoop installation ppt to set paths in the “bashrc” file.
2. Copy the output of **echo \$JAVA_HOME**

- Set HBase paths in bashrc file

```
sudo nano .bashrc
```

- Paste these lines at the end of the file

#HBASE_SETUP settings

#"/home/hadoop" can be different, so,

#1- cd ~

#2- pwd

#3- copy the output and replace the "/home/hadoop" with the output.

```
export HBASE_HOME=/home/hadoop/hbase-2.4.9
```

```
export PATH=$PATH:$HBASE_HOME/bin
```

#JAVA_HOME settings

paste the output of "echo \$JAVA_HOME"

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-  
amd64
```

```
export PATH=$PATH:/usr/lib/jvm/java-1.8.0-openjdk-  
amd64/bin
```

- Check to confirm all of the below are present

#HADOOP_SETUP settings

#use the output of "echo \$HADOOP_HOME"

```
export HADOOP_HOME=/home/hadoop/hadoop-3.2.2
```

```
export HADOOP_INSTALL=$HADOOP_HOME
```

```
export HADOOP_MAPRED_HOME=$HADOOP_HOME
```

```
export HADOOP_COMMON_HOME=$HADOOP_HOME
```

```
export HADOOP_HDFS_HOME=$HADOOP_HOME
```

```
export YARN_HOME=$HADOOP_HOME
```

```
export
```

```
HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
```

```
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

```
export HADOOP_OPTS="-
```

```
Djava.library.path=$HADOOP_HOME/lib/native"
```

- Apply the changes in “bashrc” file

```
source ~/.bashrc
```

- create a folder named "zookeeper" in the home folder.

```
cd ~  
mkdir zookeeper
```

- Making changes in the HBase configuration files

```
sudo nano hbase-site.xml
```

```
cd conf
```

```
cd hbase-2.4.9/
```

- In hbase-site.xml; between <configuration></configuration>, paste the following code and make sure to replace **/home/hadoop** with the output of the path of home directory where **Hbase-2.4.9** is extracted.

```
cd ~
```

```
pwd
```

```
hadoop@abhishek-ubuntu:~$ cd ~  
hadoop@abhishek-ubuntu:~$ pwd  
/home/hadoop  
hadoop@abhishek-ubuntu:~$
```

- Copy the output and replace **/home/hadoop** with the output in the “**hbase-site.xml**”

```

<property>
  <name>hbase.cluster.distributed</name>
  <value>true</value>
</property>

<!--"/home/hadoop" can be different, so,
      1- cd ~
      2- pwd
      3- copy the output and replace the "/home/hadoop" with the output.

-->
<property>
  <name>hbase.rootdir</name>
  <value>/home/hadoop/hbase-2.4.9/hbasestorage</value>
</property>

<property>
  <name>hbase.zookeeper.property.dataDir</name>
  <value>/home/hadoop/zookeeper</value>
</property>

<property>
  <name>hbase.zookeeper.property.clientPort</name>
  <value>2181</value>
</property>

<property>
  <name>hbase.unsafe.stream.capability.enforce</name>
  <value>>false</value>
</property>

```

- Now, we need to make changes to “**hbase-env.sh**” file

```
sudo nano hbase-env.sh
```

- Paste the following at the end of the file

```
# paste the output of "echo $JAVA_HOME" and $JAVA_PATH
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64
export PATH=$PATH:/usr/lib/jvm/java-1.8.0-openjdk-amd64/bin
```

- Uncomment the following code (line)

```
export HBASE_DISABLE_HADOOP_CLASSPATH_LOOKUP="true"
```

- Start the services

```
sudo hdfs namenode -format
start-dfs.sh
start-yearn.sh
```

- Check the running services

```
jps
```

```
hdoop@abhishek-ubuntu:~$ jps
7137 HQuorumPeer
9617 JarBootstrapMain
13730 SecondaryNameNode
7250 HMaster
13571 DataNode
7894 HRegionServer
13463 NameNode
14247 Jps
```

- Start HBase

```
cd ~/hbase-2.4.9/bin
```

```
./start-hbase.sh
```

```
hdoop@abhishek-ubuntu:~/hbase-2.4.9/bin$ ./start-hbase.sh
127.0.0.1: running zookeeper, logging to /home/hdoop/hbase-2.4.9/bin/../logs/
hbase-hdoop-zookeeper-abhishek-ubuntu.out
running master, logging to /home/hdoop/hbase-2.4.9/logs/hbase-hdoop-master-ab
hishek-ubuntu.out
: running regionserver, logging to /home/hdoop/hbase-2.4.9/logs/hbase-hdoop-r
egionserver-abhishek-ubuntu.out
```

```
hadoop@abhishek-ubuntu:~/hbase-2.4.9/bin$ hbase shell
2022-01-06 14:43:04,845 WARN [main] util.NativeCodeLoader: Unable to load native-hadoop
op library for your platform... using builtin-java classes where applicable
HBase Shell
Use "help" to get list of supported commands.
Use "exit" to quit this interactive shell.
For Reference, please visit: http://hbase.apache.org/2.0/book.html#shell
Version 2.4.9, rc49f7f63fca144765bf7c2da41791769286dfccc, Fri Dec 17 19:02:09 PST 2021
Took 0.0023 seconds
hbase:001:0> █
```

HBase Programs

1. Create a table (table name = USN) with columns fields as name, age, sem, sec, add data to the table created and print the values in the table.

```
put '111', '1', 'name', 'Abhishek'
```

```
put '111', '1', 'age', '21'
```

```
put '111', '1', 'sem', '7'
```

```
put '111', '1', 'sec', 'A'
```

```
scan '111'
```

2. Create a table (table name = USN) with columns fields as name, age, sem, sec, add data to the table created and get the value of the name column of an specific row.

```
put '111', '1', 'name', 'Abhishek'
```

```
put '111', '1', 'age', '21'
```

```
put '111', '1', 'sem', '7'
```

```
put '111', '1', 'sec', 'A'
```

```
get '111', '1', 'name'
```


3. Create a table (table name = USN) with columns fields as name, age, sem, sec, add data to the table created. Then, delete all the cells (column data) of a row in that table.

```
put '111', '1', 'name', 'Abhishek'
```

```
put '111', '1', 'age', '21'
```

```
put '111', '1', 'sem', '7'
```

```
put '111', '1', 'sec', 'A'
```

```
deleteall '111', '1'
```

4. Create a table (table name = USN) with columns fields as name, age, sem, sec, add data to the table created and delete age column of any row.

```
put '111', '1', 'name', 'Abhishek'
```

```
put '111', '1', 'age', '21'
```

```
put '111', '1', 'sem', '7'
```

```
put '111', '1', 'sec', 'A'
```

```
delete '111', '1', 'age'
```

5. Create a table (table name = USN) with columns fields as name, age, sem, sec, add data to the table created and update the name column (change the value to 'Tom')

```
put '111', '1', 'name', 'Abhishek'
```

```
put '111', '1', 'age', '21'
```

```
put '111', '1', 'sem', '7'
```

```
put '111', '1', 'sec', 'A'
```

```
put '111', '1', 'name', 'Tom'
```

Steps To Run on Docker

Installing docker if not exists:

1. Check if docker already exists:

```
sudo docker --version
```

2. If not installed install it by following steps:

```
sudo apt-get update
```

```
echo \  
  "deb [arch=$(dpkg --print-architecture) signed-  
by=/usr/share/keyrings/docker-archive-keyring.gpg]  
https://download.docker.com/linux/ubuntu \  
  $(lsb_release -cs) stable" | sudo tee  
/etc/apt/sources.list.d/docker.list > /dev/null  
sudo apt-get update
```

```
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

Pull image and run:

```
docker run -it adityasm1238/cclab:2.2
```

Once inside the container to start all the data nodes and name nodes run:

```
start-nodes
```

Wait until hadoop **safemode** is turned **off**

Status of safe mode can be checked using following command:

```
hdfs dfsadmin -safemode get
```

To run hive:

Once the safe mode is off, run the following command to open hive shell:

```
run-hive
```

To run hbase:

Once the safe mode is off, run the following command to open hbase shell:

```
hbase shell
```

Then inside hbase shell try running :

```
status
```

```
ERROR: can not resolve c3be5f557d44,16000,1640721543862
For usage try 'help "status"'
Took 5.4762 seconds
```

Wait until the above command stops giving error, this happens because the hbase nodes take approx 3 min to start.

```
hbase(main):002:0> status
1 active master, 0 backup masters, 1 servers, 0 dead, 1.0000 average load
Took 0.4721 seconds
```

It means hbase is up and running.

To run pig:

Once the safe mode is off, use 'nano' to edit and save the program along with data file, Run the program by using:

```
pig -x local program_name.pig
```

To run hadoop:

Write all programs, and data files using nano, Once the safe mode is off, put the data files into hdfs:

```
hdfs dfs -put /path/to/file/in/local /
```

To check if files are present in hdfs:

```
hdfs dfs -ls /
```

To export classpath run the following command to get classpath value

```
hadoop classpath
```

Copy paste the value in class path variable:

```
export CLASSPATH=<copied_value>
```

Compile the programs:

```
javac *.java -d .
```

Create **Manifest.txt** file using nano and make jar file:

```
jar cfm filename.jar Manifest.txt packagename/*
```

Once the jar is created, before running confirm there is no previous output folder in hdfs if present delete it using:

```
hdfs dfs -rm -r -f /output
```

To run jar:

```
hadoop jar filename.jar /datafile.csv /output
```

To check output:

```
hdfs dfs -cat /output/*
```