### Course: MScCS

### Year: 2020-2022

### Semester: IV

### Program: Computer Science

### Subject: Big Data Engineering Tools and Frameworks

### Subject Code: PS-SCS-404

### Seat Number: KSMSCCS012

### Name: Tanay Kulkarni

### University: HSNC University

### College: KC College, Churchgate

### 

### 

### 

### Signature

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Date

### \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### Practical No 1

Steps for Install Hadoop on Windows Based Platform

First, we need to make sure that the following prerequisites are installed:

1. Java 8 runtime environment (JRE): Hadoop 3 requires a Java 8 installation. I prefer using the offline installer.

2. Java 8 development Kit (JDK)

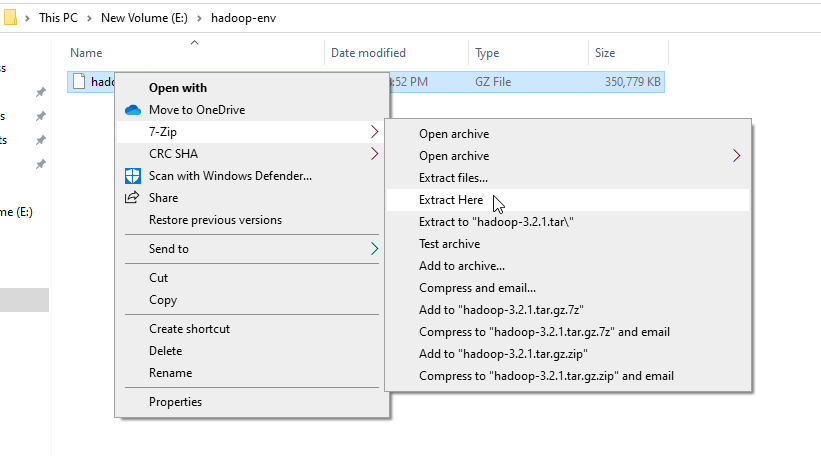
3. To unzip downloaded Hadoop binaries, we should install 7zip.

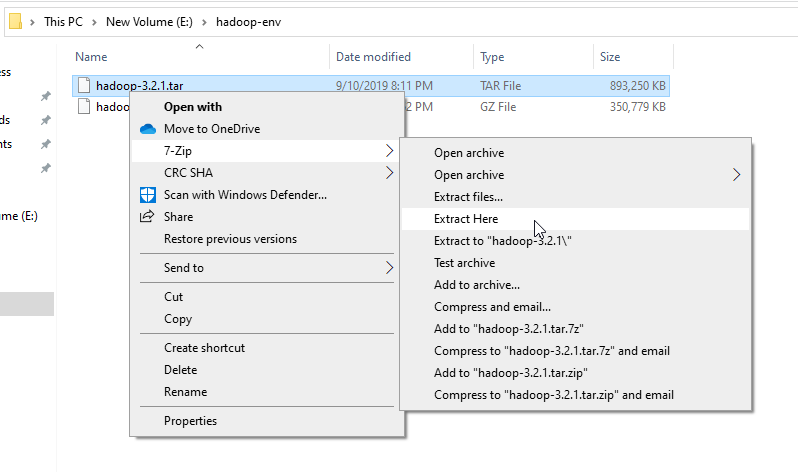
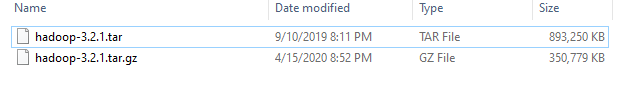
4. I will create a folder “E:\hadoop-env” on my local machine to store downloaded files.

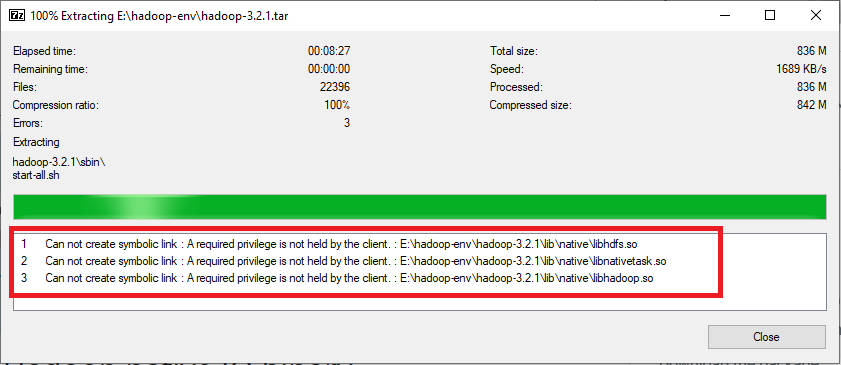
2. Download Hadoop binaries

The first step is to download Hadoop binaries from the official website.

<https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.2.1/hadoop-3.2.1.tar.gz>







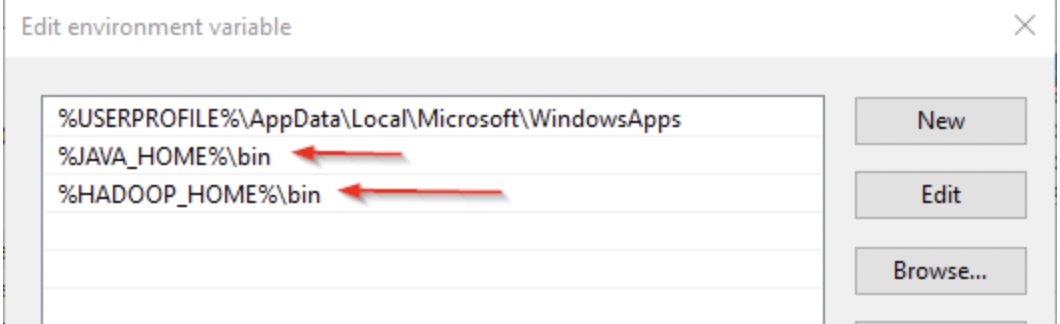
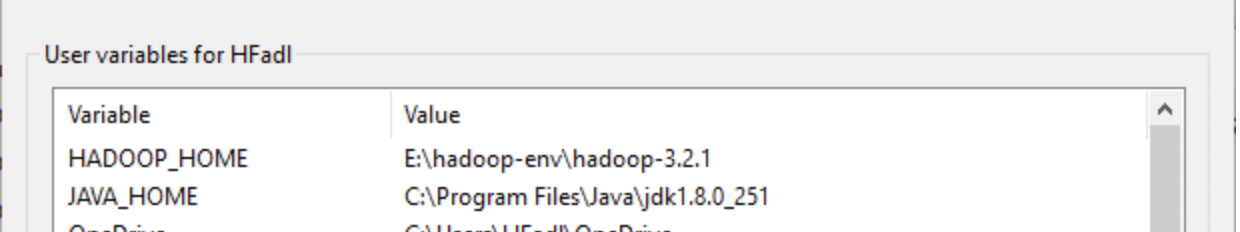
After unpacking the package, add the Hadoop native IO libraries, which can be found in the following GitHub repository:

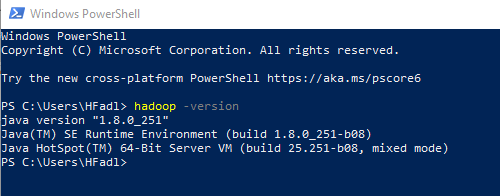
<https://github.com/cdarlint/winutils>

Since we are installing Hadoop 3.2.1, download the files located in <https://github.com/cdarlint/winutils/tree/master/hadoop-3.2.1/bin> and copy them into the “hadoop-3.2.1\bin” directory.

3. Setting up environment variables

After installing Hadoop and its prerequisites, we should configure the environment variables to define Hadoop and Java default paths.





open “hdfs-site.xml” file located in “%HADOOP\_HOME%\etc\hadoop” directory, and we should add the following properties within the <configuration></configuration> element:

<property>

<name>dfs.replication</name>

<value>1</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>file:///E:/hadoop-env/hadoop-3.2.1/data/dfs/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>file:///E:/hadoop-env/hadoop-3.2.1/data/dfs/datanode</value>

</property>

configure the name node URL adding the following XML code into the <configuration></configuration> element within “core-site.xml”:

<property>

<name>fs.default.name</name>

<value>hdfs://localhost:9820</value>

</property>

add the following XML code into the <configuration></configuration> element within “mapred-site.xml”:

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

<description>MapReduce framework name</description>

</property>

add the following XML code into the <configuration></configuration> element within “yarn-site.xml”:

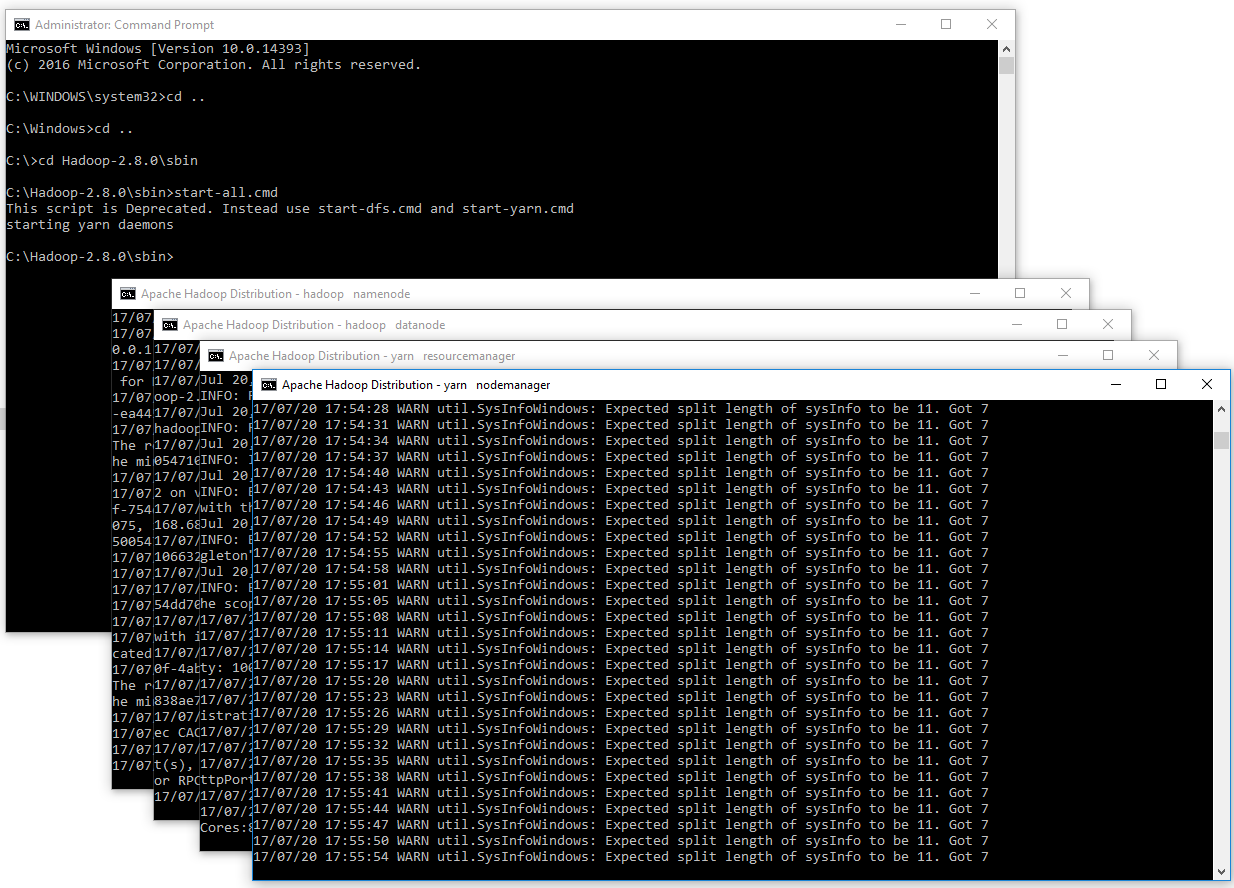
<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

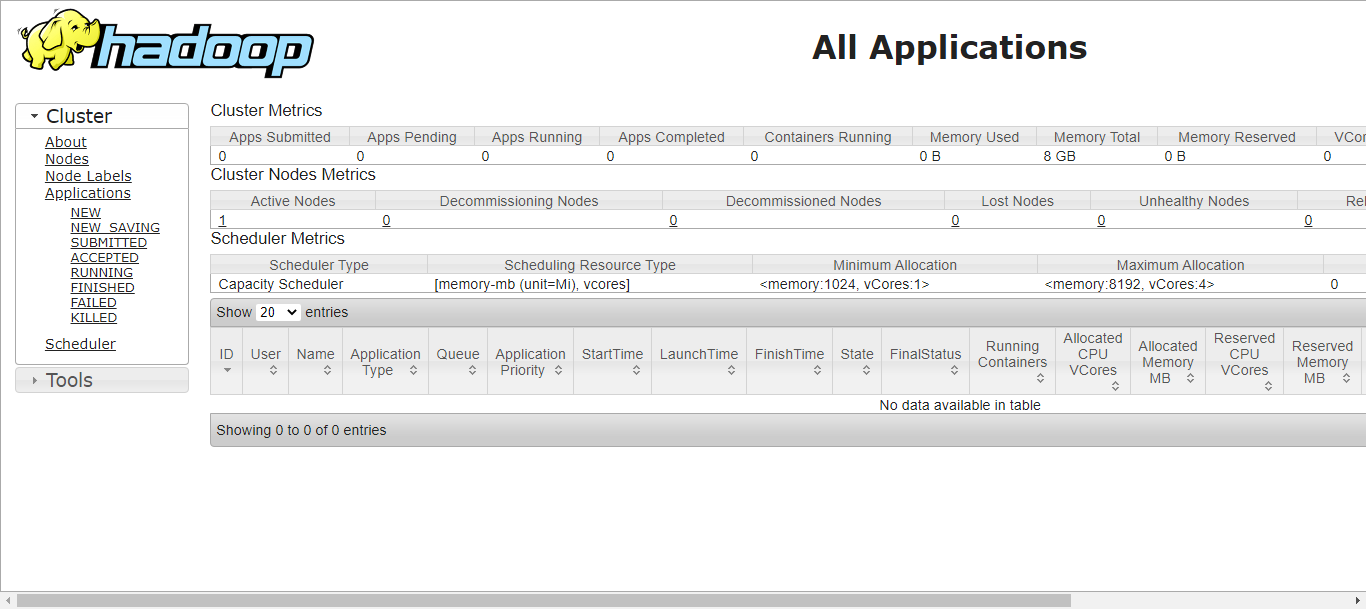
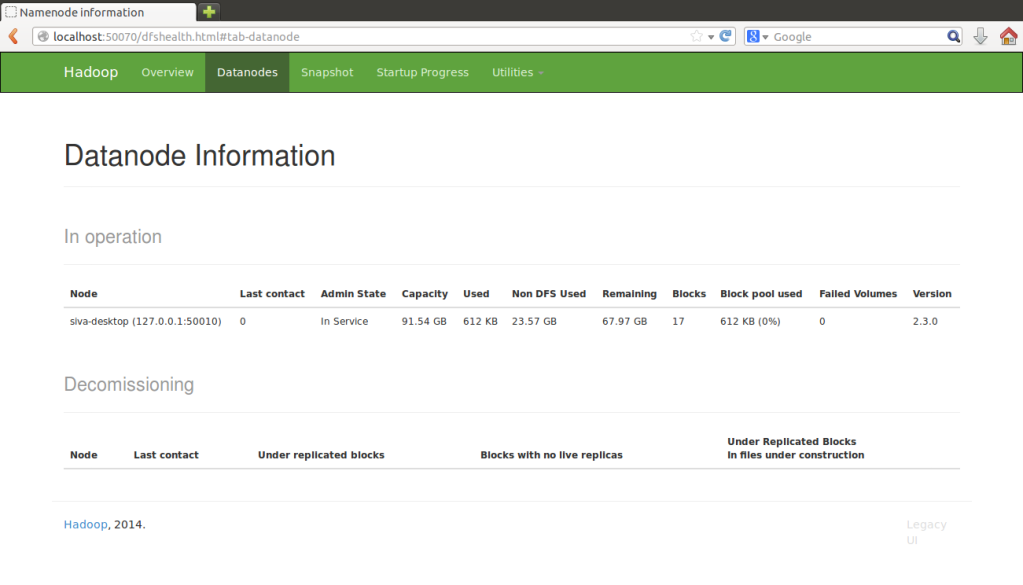
<description>Yarn Node Manager Aux Service</description>

</property>

hdfs namenode -format







### Practical No 2

Hadoop Word Count

Code:

import java.io.IOException;

import java.util.\*;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

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

import org.apache.hadoop.mapreduce.\*;

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

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

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

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

public class WordCount {

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

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

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

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

context.write(word, one);

}

}

}

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

public void reduce(Text key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

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

}

}

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

Configuration conf = new Configuration();

conf.set("mapred.job.tracker", "hdfs://localhost:50001");

conf.set("fs.default.name", "hdfs://localhost:50000");

Job job = new Job(conf, "wordcount");

job.setJarByClass(WordCount.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

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

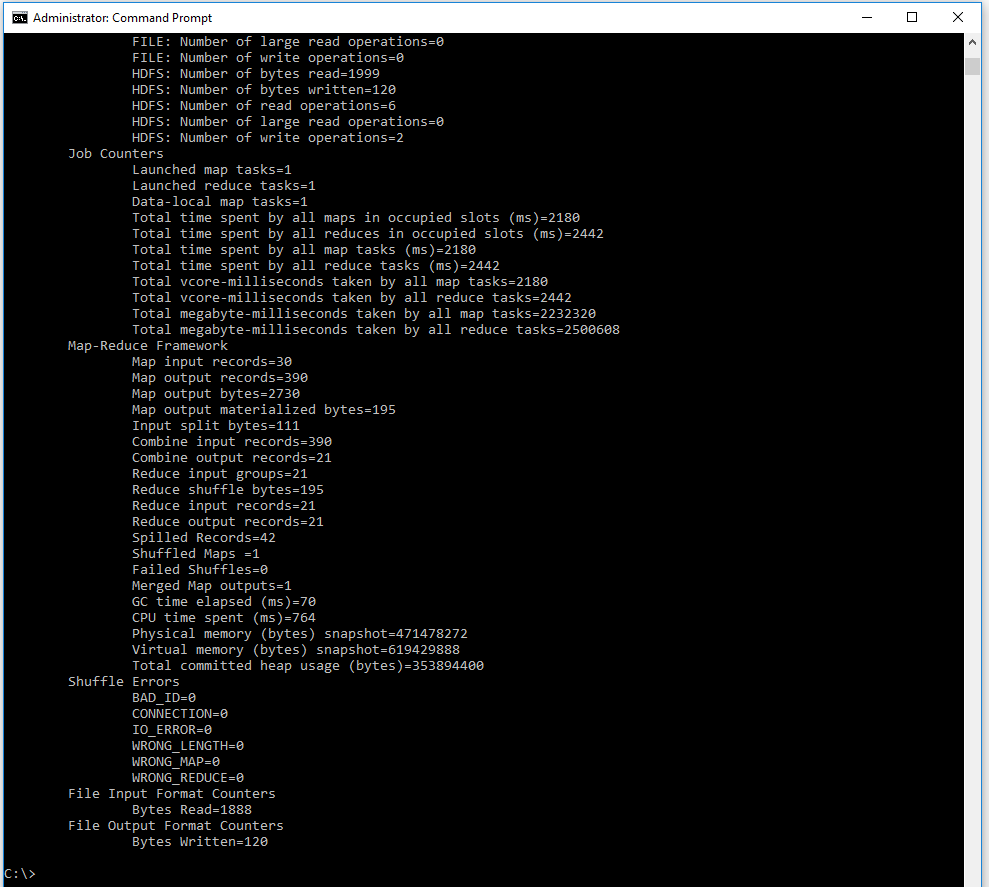
FileOutputFormat.setOutputPath(job, new Path(args[1]));

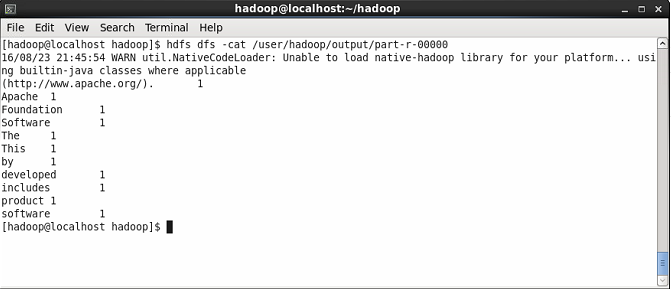
job.waitForCompletion(true);

}

}

Output:



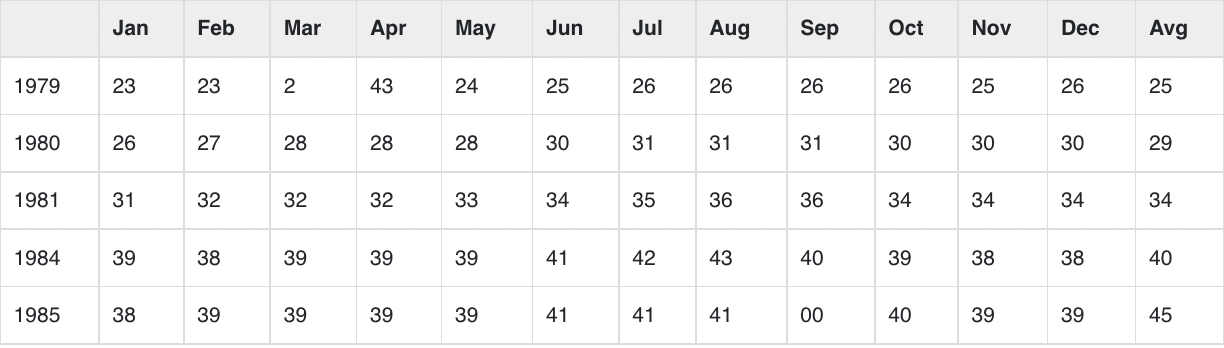


### Practical No 3

Example working with Hadoop Map Reduce

Given below is the data regarding the electrical consumption of an organization. It contains the monthly electrical consumption and the annual average for various years.

If the above data is given as input, write applications to process it and produce results such as finding the year of maximum usage, year of minimum usage, and so on.



package hadoop;

import java.util.\*;

import java.io.IOException;

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.conf.\*;

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

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

import org.apache.hadoop.util.\*;

public class ProcessUnits {

//Mapper class

public static class E\_EMapper extends MapReduceBase implements

Mapper<LongWritable ,/\*Input key Type \*/

Text, /\*Input value Type\*/

Text, /\*Output key Type\*/

IntWritable> /\*Output value Type\*/

{

//Map function

public void map(LongWritable key, Text value,

OutputCollector<Text, IntWritable> output,

Reporter reporter) throws IOException {

String line = value.toString();

String lasttoken = null;

StringTokenizer s = new StringTokenizer(line,"\t");

String year = s.nextToken();

while(s.hasMoreTokens()) {

lasttoken = s.nextToken();

}

int avgprice = Integer.parseInt(lasttoken);

output.collect(new Text(year), new IntWritable(avgprice));

}

}

//Reducer class

public static class E\_EReduce extends MapReduceBase implements Reducer< Text, IntWritable, Text, IntWritable > {

//Reduce function

public void reduce( Text key, Iterator <IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int maxavg = 30;

int val = Integer.MIN\_VALUE;

while (values.hasNext()) {

if((val = values.next().get())>maxavg) {

output.collect(key, new IntWritable(val));

}

}

}

}

//Main function

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

JobConf conf = new JobConf(ProcessUnits.class);

conf.setJobName("max\_eletricityunits");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(E\_EMapper.class);

conf.setCombinerClass(E\_EReduce.class);

conf.setReducerClass(E\_EReduce.class);

conf.setInputFormat(TextInputFormat.class);

conf.setOutputFormat(TextOutputFormat.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

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

JobClient.runJob(conf);

}

}

$ mkdir units

$ javac -classpath hadoop-core-1.2.1.jar -d units ProcessUnits.java

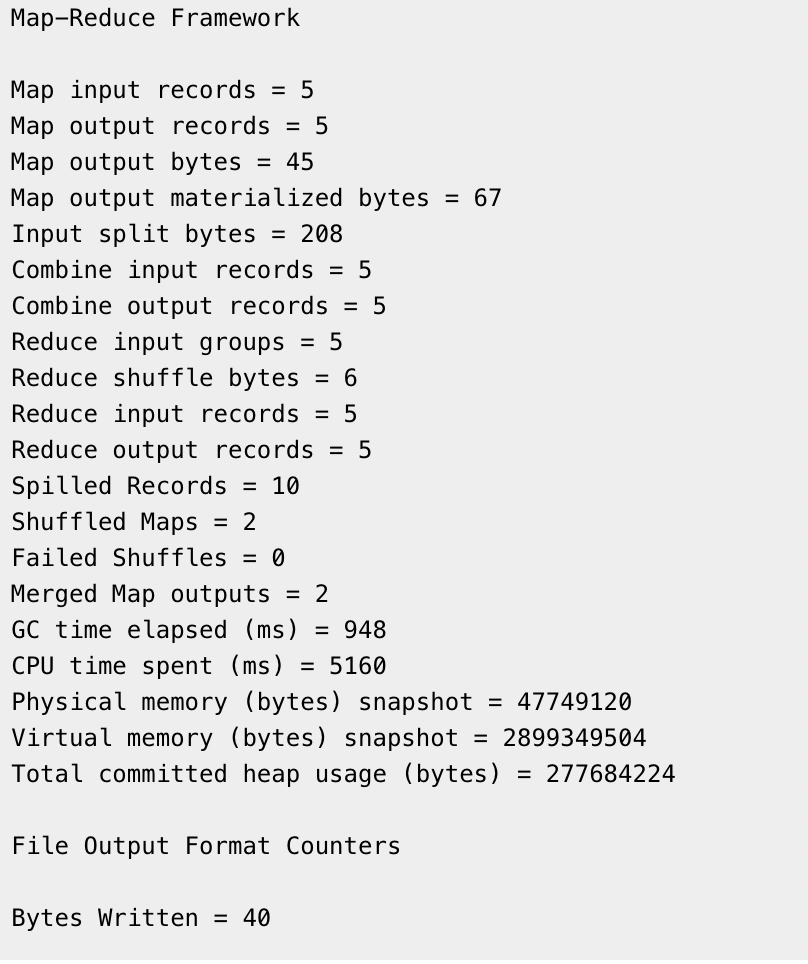
$ jar -cvf units.jar -C units/ .

$HADOOP\_HOME/bin/hadoop fs -mkdir input\_dir

$HADOOP\_HOME/bin/hadoop fs -put /home/hadoop/sample.txt input\_dir

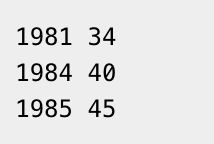
$HADOOP\_HOME/bin/hadoop fs -ls input\_dir/

$HADOOP\_HOME/bin/hadoop jar units.jar hadoop.ProcessUnits input\_dir output\_dir



$HADOOP\_HOME/bin/hadoop fs -ls output\_dir/

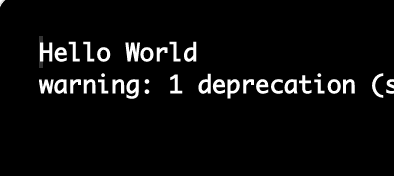
$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000



copy the output folder from HDFS to the local file system.

$HADOOP\_HOME/bin/hadoop fs -cat output\_dir/part-00000/bin/hadoop dfs get output\_dir /home/hadoop

### Practical No 4

a. Write a Scala program to print "hello world" 

object MainObject

{

def main(args:Array[String])

{

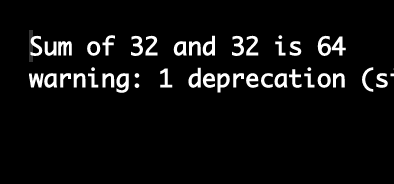
print("Hello World")

}

}

b. Write a Scala program to compute the sum of the two given integer value, if the value are the same then return their sum.

object MainObject

{

def main(args:Array[String])

{

val a = scala.io.StdIn.readInt()

val b = scala.io.StdIn.readInt()

if(a==b){

print(s"Sum of $a and $b is " + (a + b))

}else {

print(s"$a and $b are not same")

}

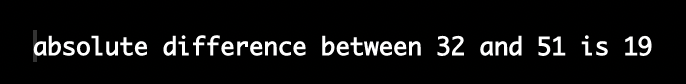
}

}

c. Write a Scala program to get the absolute difference between n to 51. If n is greater than 51 , error message should display.

object MainObject

{

def main(args:Array[String])

{

val n = scala.io.StdIn.readInt()

val n\_abs = n.abs

if(n\_abs > 51) {

println(s"Invalid Input")

}else {

val diff = 51 - n\_abs

println(s"absolute difference between $n and 51 is $diff")

}

}

}

### Practical No 5

a.Write a Scala program to check if a given number is present in first or the last position of given array

object MainObject

{

def main(args:Array[String])

{

val list = List(4,3,5,66,8,3,2,1,9,8)

val n = scala.io.StdIn.readInt()

if(n == list.head){

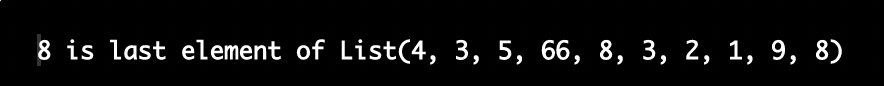
println(s"$n is first element of $list")

}else if(n == list.last){

println(s"$n is last element of $list")

}else {

println(s"$n is not first or last element of list")

}

}

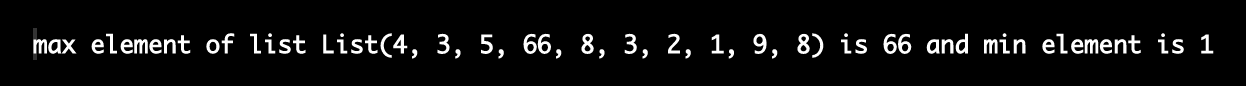
}

b. Write a Scala program to find the maximum and minimum value of an array of integers.

object MainObject

{

def main(args:Array[String])

{

val list = List(4,3,5,66,8,3,2,1,9,8)

val mx = list.max

val mn = list.min

println(s"max element of list $list is $mx and min element is $mn")

}

}

c. Write a Scala program to find the common element between two arrays of string.

object MainObject

{

def main(args:Array[String])

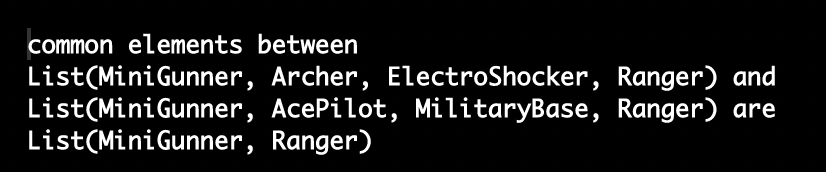
{

val s1 = List("MiniGunner", "Archer", "ElectroShocker", "Ranger")

val s2 = List("MiniGunner", "AcePilot", "MilitaryBase", "Ranger")

val common = s1.intersect(s2)

println(s"common elements between $s1 and $s2 are $common")

}

}

### Practical No 6

a. Write a Scala program to calculate the length of a given list.

object MainObject

{

def main(args:Array[String])

{

val s1 = List("MiniGunner", "Archer", "ElectroShocker", "Ranger")

val len = s1.length

println(s"List of length $s1 is $len")

}

}

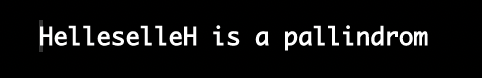
b. Write a Scala program to check a given list is a palindrome or not.

object MainObject

{

def main(args:Array[String])

{

val str = scala.io.StdIn.readLine()

val str\_r = str.reverse

if(str == str\_r){

println(s"$str is a pallindrom")

}else {

println(s"$str is not a pallindrome")

}

}

}

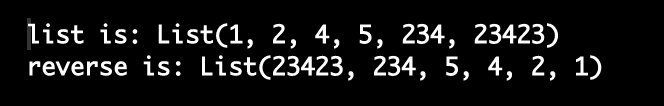
c. Write a Scala program to reverse a given list.

object MainObject

{

def main(args:Array[String])

{

val str = scala.io.StdIn.readLine()

val str\_r = str.reverse

if(str == str\_r){

println(s"$str is a pallindrom")

}else {

println(s"$str is not a pallindrome")

}

}

}

### Practical No 7

Working with Spark

1.What are the column name

2.What does the Schema look like

3.Print out the first 5 columns

4.Display mean, count, stdev, min, max

5. Create a new dataframe with a column called hvratio that is the ratio of the high price verus volume of stock traded for a day

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('walmart').getOrCreate()

df = spark.read.csv('walmart\_stock.csv', inferSchema=True, header=True)

print(df.columns)

df.printSchema()

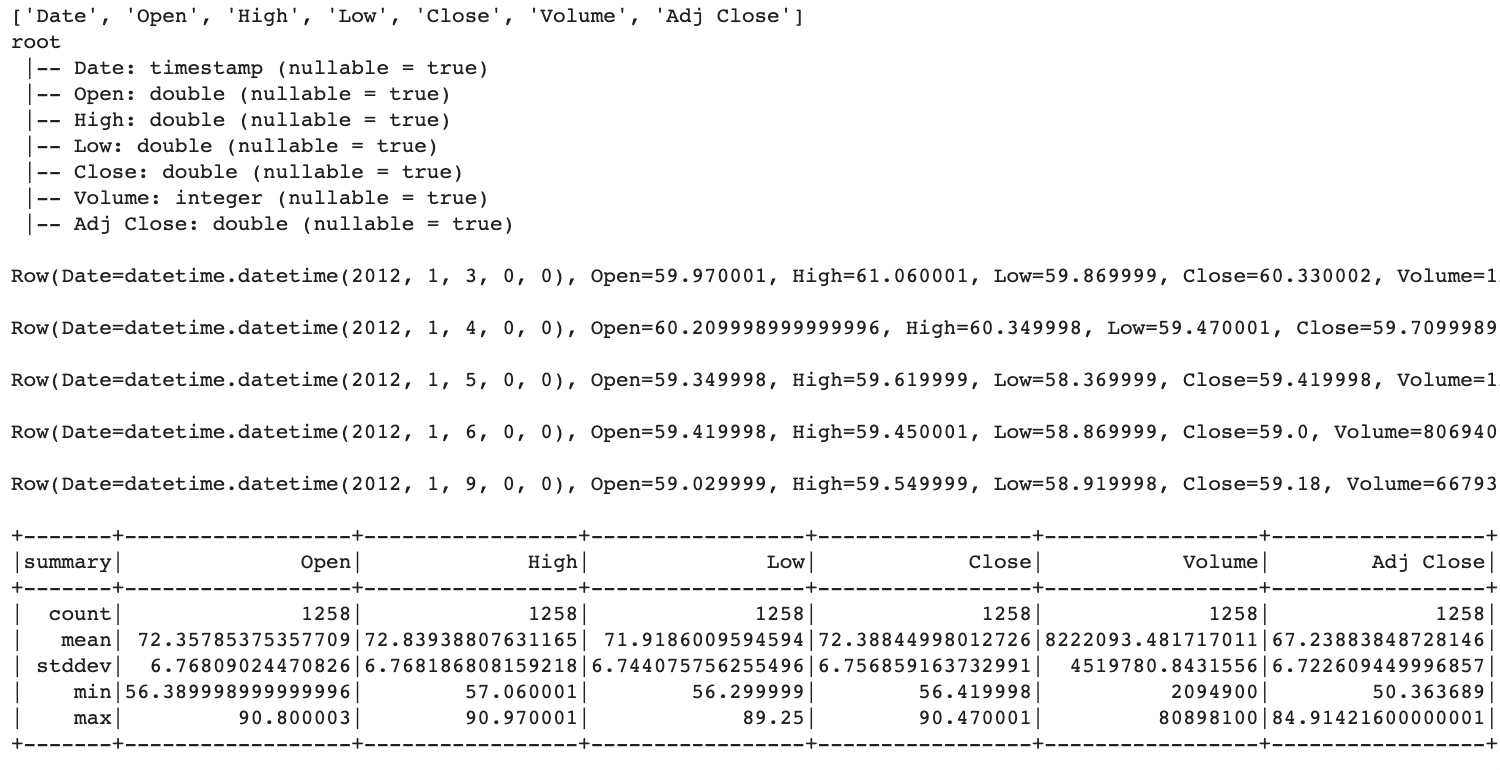
for line in df.head(5):

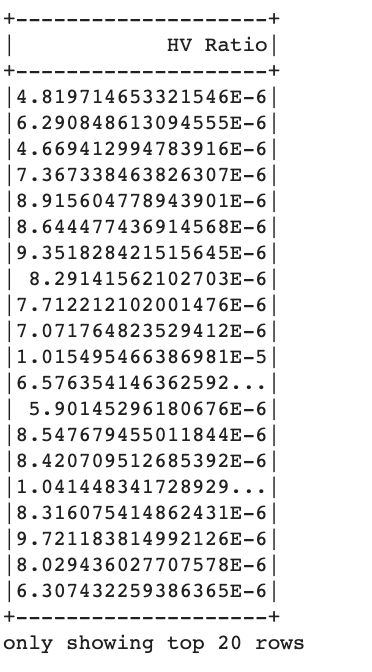
print(line, '\n')

df.describe().show()

df\_hv = df.withColumn('HV Ratio', df['High']/df['Volume']).select(['HV Ratio'])

df\_hv.show()





### Practical No 8

1. What day had the Peak high in price

2. What is the mean of the close column

3. What is the max and min of the volume column

4. How many days was the close lower than 220

5. What percentage of the time was the high greater than 250

6. What is the max high per year

7. What is the average close for each Calendar Month

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('walmart').getOrCreate()

df = spark.read.csv('walmart\_stock.csv', inferSchema=True, header=True)

print(df.orderBy(df['High'].desc()).select(['Date']).head(1)[0]['Date'])

from pyspark.sql.functions import mean

df.select(mean('Close')).show()

from pyspark.sql.functions import min, max

df.select(max('Volume'),min('Volume')).show()

print(df.filter(df['Close'] < 220).count())

print(df.filter('High > 250').count() \* 100/df.count())

from pyspark.sql.functions import (dayofmonth, hour,

dayofyear, month,

year, weekofyear,

format\_number, date\_format)

year\_df = df.withColumn('Year', year(df['Date']))

year\_df.groupBy('Year').max()['Year', 'max(High)'].show()

#Create a new column Month from existing Date column

month\_df = df.withColumn('Month', month(df['Date']))

#Group by month and take average of all other columns

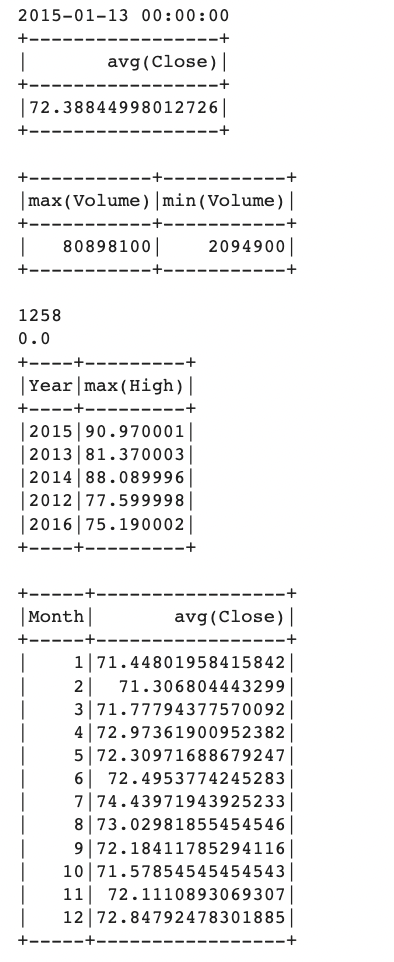
month\_df = month\_df.groupBy('Month').mean()

#Sort by month

month\_df = month\_df.orderBy('Month')

#Display only month and avg(Close), the desired columns

month\_df['Month', 'avg(Close)'].show()



### Practical No 9:

Spark SQL connecting with Data Source : Display all the High and Closing price

What is the average high price , close price, low price

What is the lowest price in high, close and low price

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('walmart').getOrCreate()

df = spark.read.csv('walmart\_stock.csv', inferSchema=True, header=True)

print(df.orderBy(df['High'].desc()).select(['Date']).head(1)[0]['Date'])

from pyspark.sql.functions import mean

avg\_high = df.agg({"High": "avg"}).collect()[0][0]

print("average high: " + str(avg\_high))

avg\_close = df.agg({"Close": "avg"}).collect()[0][0]

print("average close: " + str(avg\_close))

avg\_low = df.agg({"Low": "avg"}).collect()[0][0]

print("average low: " + str(avg\_low))

low\_high = df.agg({"High": "min"}).collect()[0][0]

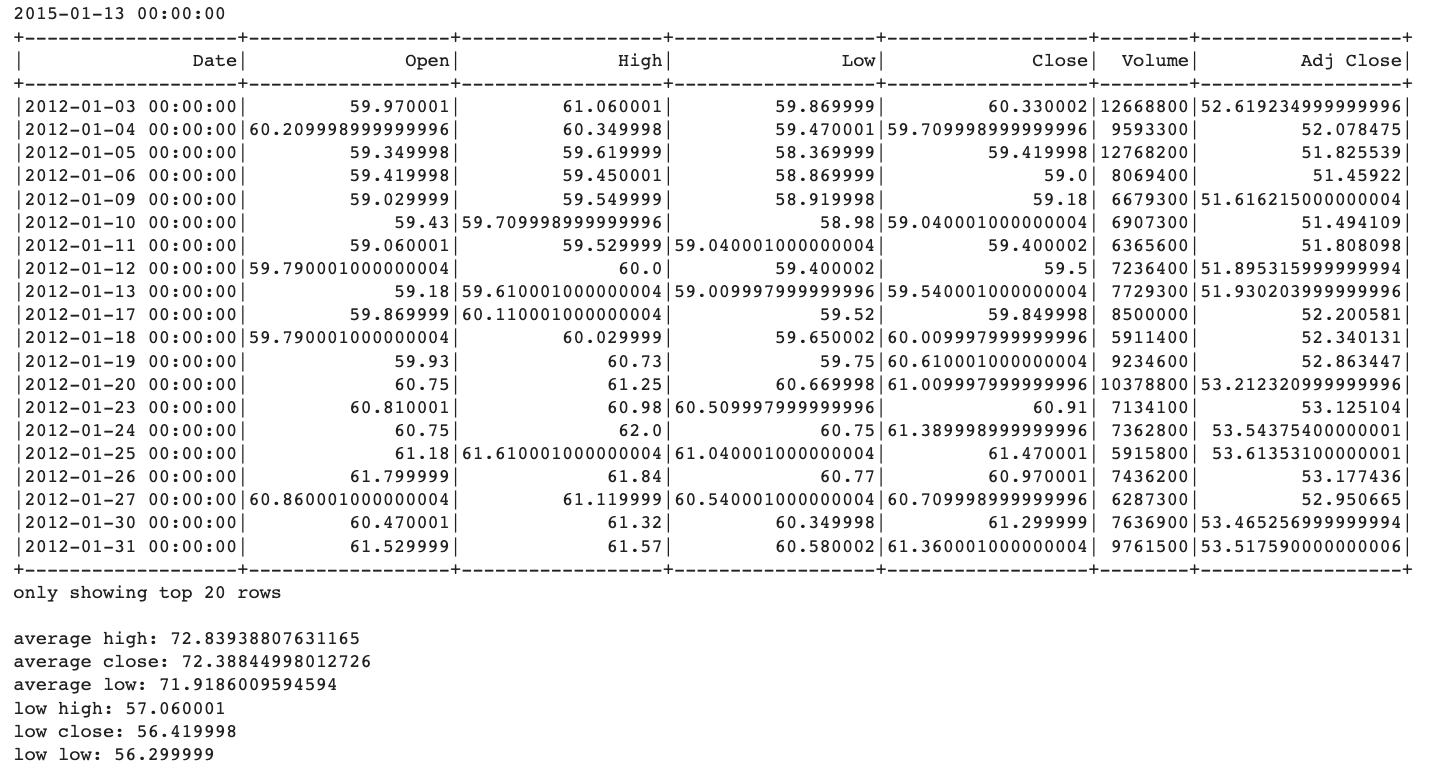
print("low high: " + str(low\_high))

low\_close = df.agg({"Close": "min"}).collect()[0][0]

print("low close: " + str(low\_close))

low\_low = df.agg({"Low": "min"}).collect()[0][0]

print("low low: " + str(low\_low))



### Practical No 10:

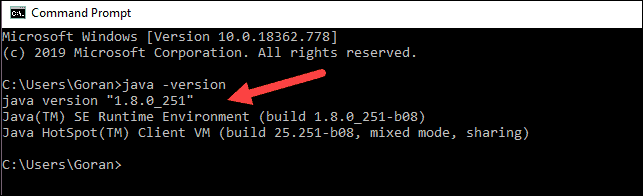
Installation SPark and Scala on window based application

Install Java 8

Apache Spark requires Java 8. You can check to see if Java is installed using the command prompt.

Open the command line by clicking Start > type cmd > click Command Prompt.

Type the following command in the command prompt:



Install Apache Spark

Installing Apache Spark involves extracting the downloaded file to the desired location.

1. Create a new folder named Spark in the root of your C: drive. From a command line, enter the following:

cd \

mkdir Spark

2. In Explorer, locate the Spark file you downloaded.

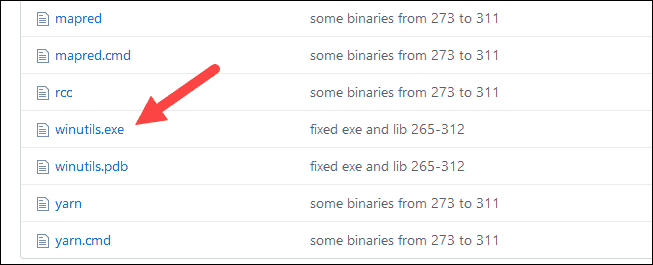
3. Right-click the file and extract it to C:\Spark using the tool you have on your system (e.g., 7-Zip).

4. Now, your C:\Spark folder has a new folder spark-2.4.5-bin-hadoop2.7 with the necessary files inside.

Add winutils.exe File

Download the winutils.exe file for the underlying Hadoop version for the Spark installation you downloaded.

1. Navigate to this URL https://github.com/cdarlint/winutils and inside the bin folder, locate winutils.exe, and click it.



2. Find the Download button on the right side to download the file.

3. Now, create new folders Hadoop and bin on C: using Windows Explorer or the Command Prompt.

4. Copy the winutils.exe file from the Downloads folder to C:\hadoop\bin.

For Hadoop, the variable name is HADOOP\_HOME and for the value use the path of the folder you created earlier: C:\hadoop. Add C:\hadoop\bin to the Path variable field, but we recommend using %HADOOP\_HOME%\bin.

For Java, the variable name is JAVA\_HOME and for the value use the path to your Java JDK directory (in our case it’s C:\Program Files\Java\jdk1.8.0\_251).

To start Spark, enter:

C:\Spark\spark-2.4.5-bin-hadoop2.7\bin\spark-shell

