In my project, I used AWS with EC2 to run 4 instances and performed some analyses using Apache Spark and Apache Hadoop. Despite encountering many errors due to insufficient CPU during the process, I managed to execute the queries and aggregation operations I intended to. I wanted to share these with you, and I hope you like them.

GÖKHAN ERGÜL

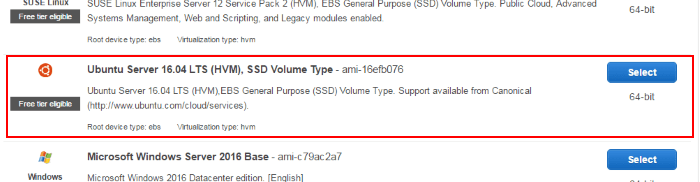
CONTENTS:

1. **Instance Settings and Necessary Configurations**: Settings and configurations required for instances.
2. **Installation of Apache Hadoop on AWS EC2 and Resolution of Some Errors Encountered**: Installation process of Apache Hadoop on AWS EC2 and information on how certain encountered errors were resolved.
3. **Installation of Apache Spark**: Steps for installing Apache Spark.
4. **How to Implement WordCount Application with Hadoop and Java Code**: Explanation of how to implement the WordCount application with Hadoop, including Java code.
5. **How to Implement WordCount Application with Spark and Java Code**: Explanation of how to implement the WordCount application with Spark, including Java code.
6. **How to Implement a Python Application:** The implementation of sorting software developers' average annual salaries received according to sectors in terms of dollars from highest to lowest in the year 2024 using Python**.**

**1) Instance Settings and Necessary Configurations:**

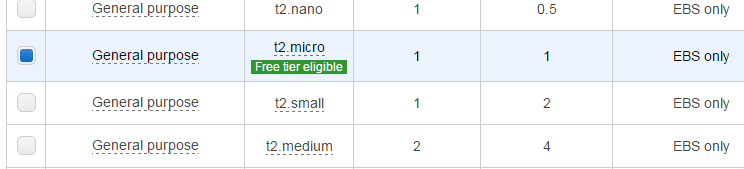
**Select Instance**

Go to your AWS Console, Click on Launch Instance and select Ubuntu Server 16.04 LTS.



**Instance Type**

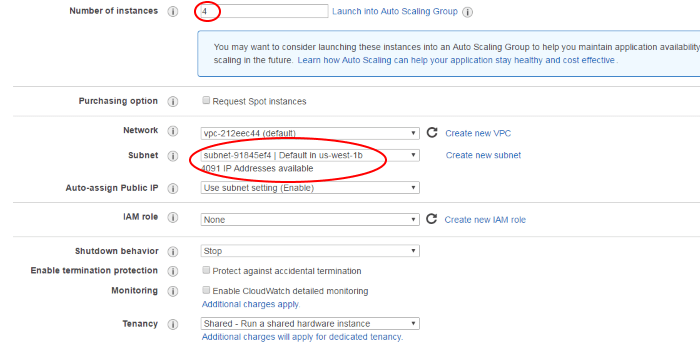
For the instance type, we choose **t2.micro** since that is sufficient for the purposes of the demo. If you have a need for a high-memory or high-cpu instance, you can select one of those.



Click Next to Configure Instance Details

**Instance Details**

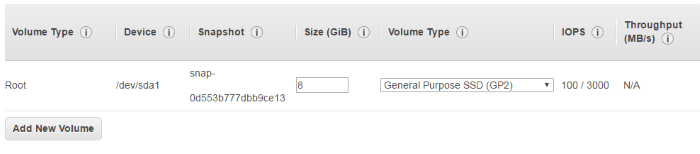
Here, we request 4 instances of the selected machine type. We also choose a subnet (us-west-1b) just so we can launch into the same location if we need more machines.

[](https://i0.wp.com/www.novixys.com/blog/wp-content/uploads/2017/03/ec2-3-700.png)

Click Next to Add Storage

**Storage**

For our purpose, the default instance storage of 8GB is sufficient. If you need more storage, either increase the size or attach a disk by clicking “Add Volume”. If you add a volume, you will need to attach the volume to your instance, format it and mount it. Since this is a beginner tutorial, these steps are not covered here.



Click Next to Add Tags to your instances.

After these steps I created 4 instances, here they instances:

metin, yazılım, multimedya yazılımı, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Since the public IP address changes every time, I shut down and start up the AWS instance, I've decided to use Elastic IP addresses for each node. Here they are:

ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

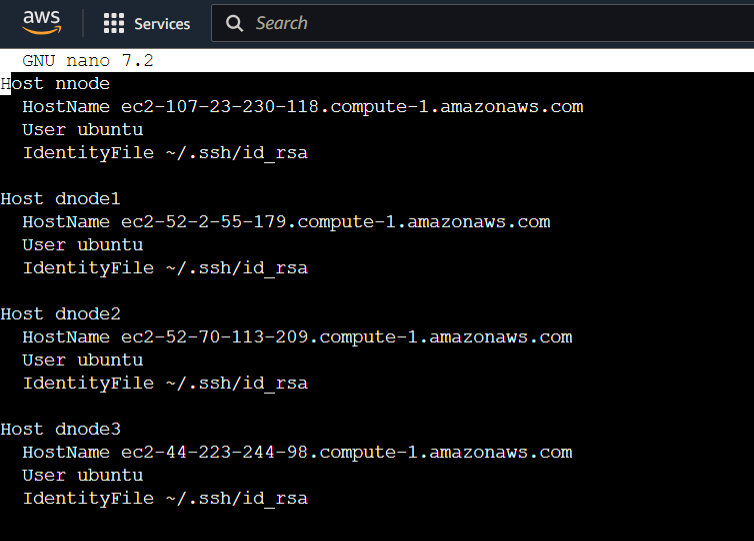
Açıklama otomatik olarak oluşturuldu

**2) Installation of Apache Hadoop on AWS EC2 and Resolution of Some Errors Encountered**:

Before starting to install Hadoop, we downloaded Java, then we downloaded Apache Hadoop for each node.

I edited the core\_site.xml file for all nodes, and then I installed "ssh-keygen" for the connection between the namenode and datanodes.

This establishes a key on the namenode, which is used to connect to the other nodes. Then, on the namenode, we made a modification in the ~/.ssh/config directory like this to be able to connect to other nodes:



After that, I edited hdfs-site.xml file, mapred-site.xml and yarn-site.xml file

Here these files contents:

Core-site.xml:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Hdfs-site.xml:

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

Mapred-site.xml:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yarn-site.xml:

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

After these steps, we start HDFS, YARN, and the history server to view the operations performed here:

./hadoop/sbin/start-dfs.sh

./hadoop/sbin/start-yarn.sh

./hadoop/sbin/mr-jobhistory-daemon.sh start historyserver

Troubleshooting during Hadoop installation:

**Datanode did not work:**

After set Hadoop, I got

2576 NameNode

2912 ResourceManager

2995 JobHistoryServer

3254 Jps

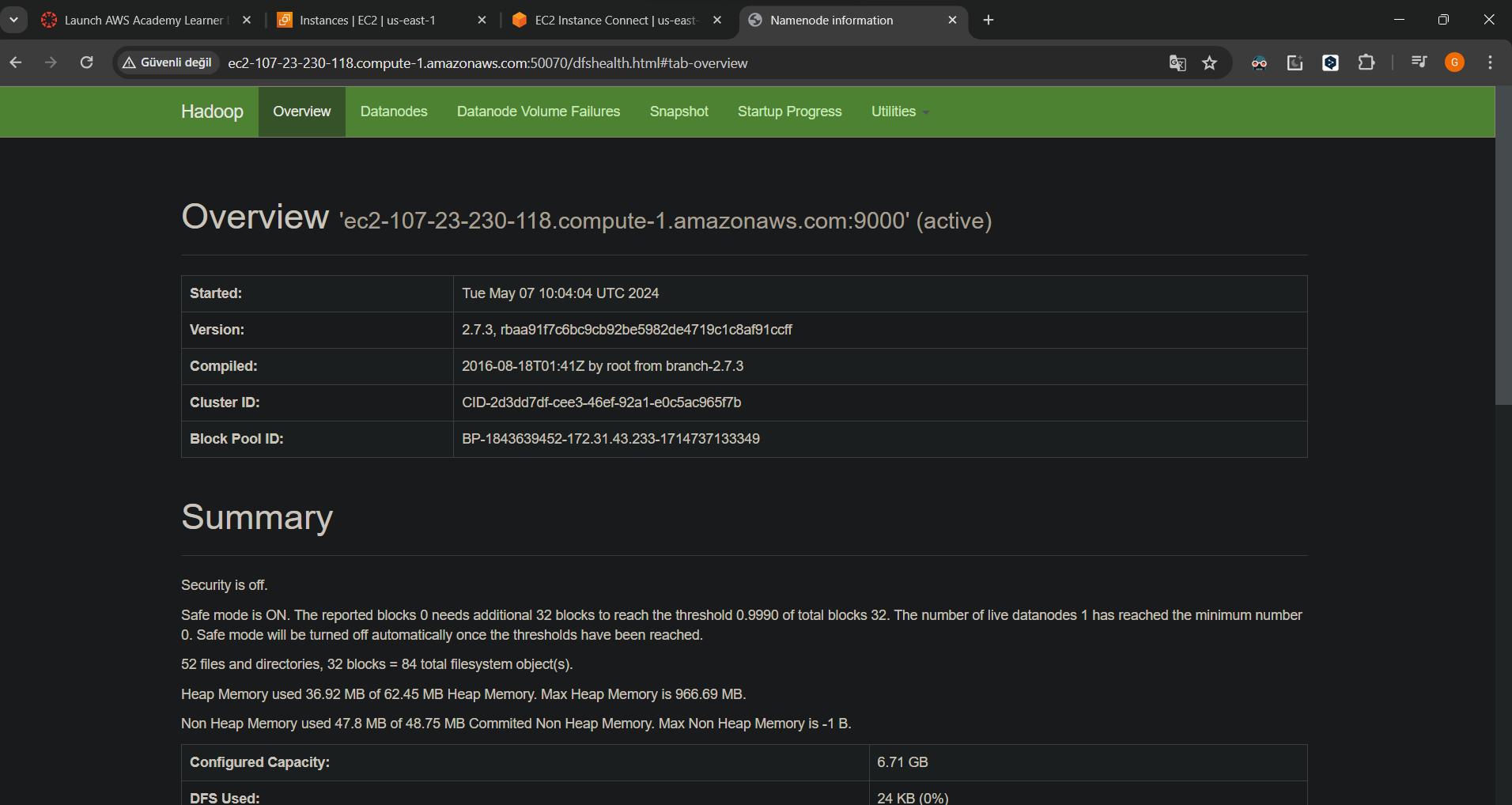
2783 SecondaryNameNode

When I try ‘jps’ but I had to get ‘Datanode’ too, for solve this problem I use this line: ./hadoop/sbin/hadoop-daemon.sh start datanode

**Not being able to Connect to datanodes via SSH and getting permission denied error:**

When I try to connect my datanode1 with ‘ssh dnode1’ I got ‘Permision denied’ error but I solve that with manually adding your key to the ~/.ssh/authorized\_keys file.

I access to web site use my\_namenode\_DNS\_ip\_address:50070



**3)Installation of Apache Spark**: Steps for installing Apache Spark:

1. **Install Apache Spark**:
   * Download Apache Spark from the official website.
   * Extract the downloaded file on each node.
   * Set the **$SPARK\_HOME** path in the **~/.bashrc** file on each node.
2. **Configuration of Master Nodes**:
   * Modify **spark-defaults.conf** and **spark-env.sh** files in the **conf** directory of Spark installation.
   * Configure **spark.master** and **spark.jars.packages** properties in **spark-defaults.conf**.
   * Set **SPARK\_LOCAL\_IP** and **SPARK\_MASTER\_HOST** in **spark-env.sh**.
3. **Configuration of Slave Nodes**:
   * Repeat the same modification process for **spark-defaults.conf** and **spark-env.sh** files on slave nodes.
4. **Add Dependencies to Connect Spark and Cassandra**:
   * Download required JAR files for Spark-Cassandra connectivity and place them in the **jars** directory of Spark.
5. **Launch Master and Slave Nodes**:
   * Start Spark Master and Worker nodes using the appropriate scripts (**start-master.sh** and **start-slave.sh**).

To start master:

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

To start slaves try this for each datanodes:

ekran görüntüsü, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

**4)How to Implement WordCount Application with Hadoop and Java Code**: Explanation of how to implement the WordCount application with Hadoop, including Java code:

Here is content of my WordCount.java file:

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

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

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

public class WordCount {

  public static class TokenizerMapper

       extends Mapper<Object, Text, Text, IntWritable>{

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

    private Text word = new Text();

    public void map(Object key, Text value, Context context

                    ) throws IOException, InterruptedException {

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

      while (itr.hasMoreTokens()) {

        word.set(itr.nextToken());

        context.write(word, one);

      }

    }

  }

  public static class IntSumReducer

       extends Reducer<Text,IntWritable,Text,IntWritable> {

    private IntWritable result = new IntWritable();

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

                       Context context

                       ) throws IOException, InterruptedException {

      int sum = 0;

      for (IntWritable val : values) {

        sum += val.get();

      }

      result.set(sum);

      context.write(key, result);

    }

  }

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

    Configuration conf = new Configuration();

    Job job = Job.getInstance(conf, "word count");

    job.setJarByClass(WordCount.class);

    job.setMapperClass(TokenizerMapper.class);

    job.setCombinerClass(IntSumReducer.class);

    job.setReducerClass(IntSumReducer.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(IntWritable.class);

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

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

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

  }

}

Here is content of my test.txt file:

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

I used the **javac** function to compile the Java file into a class file.

**$** **javac -classpath ${HADOOP\_CLASSPATH} -d 'tutorial\_classes' wordCount.java**

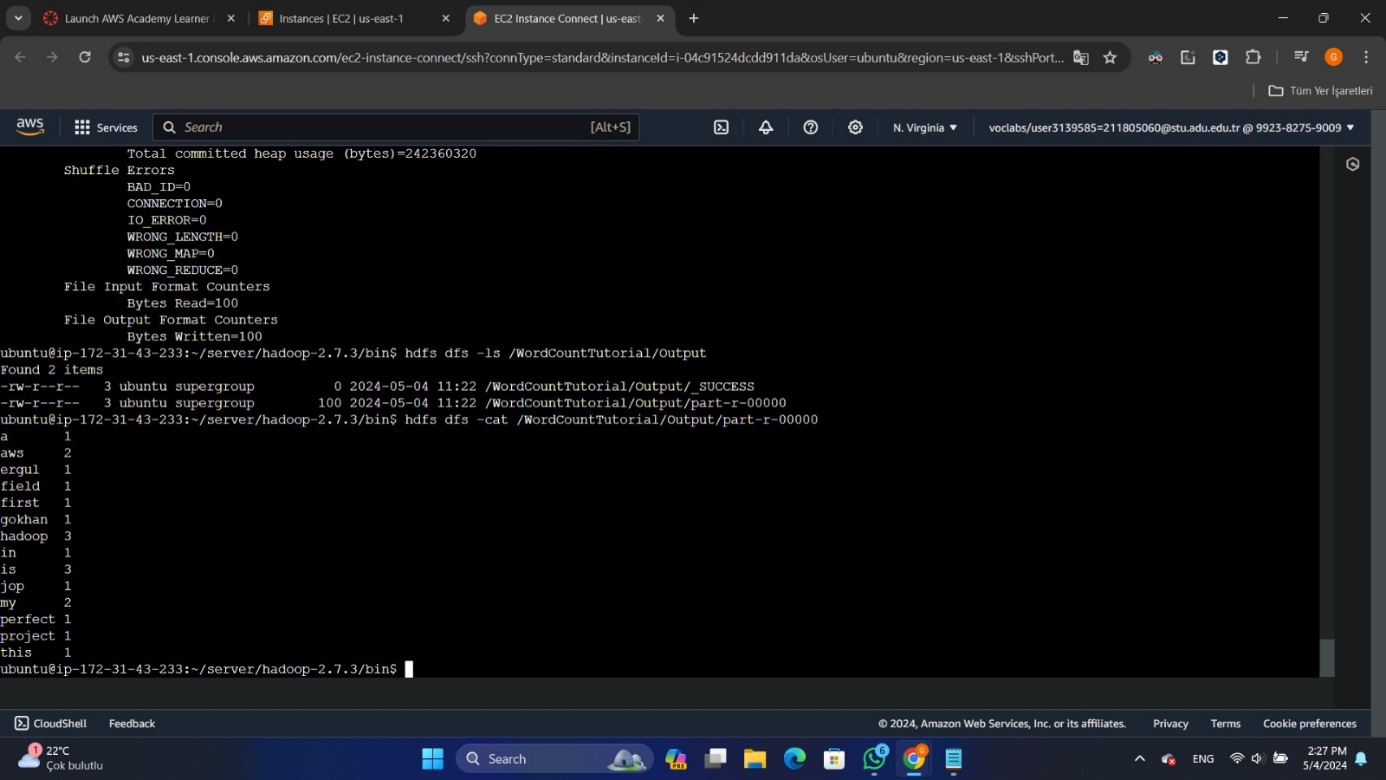
Then, I used the **jar** function to create a JAR file from the class file.

**$ jar -cvf wordCount.jar -C tutorial\_classes .**

Finally, I applied the word count operation to my **test.txt** file using the **java** function.

**$ hadoop jar wordCount.jar wordCount /WordCountTutorial/Input /WordCountTutorial/Output**

Finaly the result:



**5) How to Implement WordCount Application with Spark and Java Code**: Explanation of how to implement the WordCount application with Spark, including Java code:

I made txt file with information about ottoman empire , I got that

from Wikipedia.

Here is InputWordCount.txt:

metin, ekran görüntüsü, yazılım, bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturuldu

Content of JavaWordCount.java:

import org.apache.spark.api.java.JavaPairRDD;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.SparkConf;

import scala.Tuple2;

import java.util.Arrays;

import java.util.regex.Pattern;

public class JavaWordCount {

    private static final Pattern SPACE = Pattern.compile(" ");

    public static void main(String[] args) {

        if (args.length < 2) {

            System.err.println("Usage: JavaWordCount <inputFile> <outputDirectory>");

            System.exit(1);

        }

        // Create a SparkConf object to configure the application

        SparkConf sparkConf = new SparkConf().setAppName("JavaWordCount");

        // Create a JavaSparkContext object to run the application

        JavaSparkContext sc = new JavaSparkContext(sparkConf);

        // Load the input data from the file into an RDD

        JavaRDD<String> lines = sc.textFile(args[0]);

        // Split each line into words using a space as the delimiter

        JavaRDD<String> words = lines.flatMap(line -> Arrays.asList(SPACE.split(line)).iterator());

        // Map each word to a tuple (word, 1)

        JavaPairRDD<String, Integer> ones = words.mapToPair(word -> new Tuple2<>(word, 1));

        // Reduce by key to count the occurrences of each word

        JavaPairRDD<String, Integer> counts = ones.reduceByKey((Integer i1, Integer i2) -> i1 + i2);

        // Save the word counts to a file

        counts.saveAsTextFile(args[1]);

        // Stop the Spark context

        sc.stop();

    }

}

I used the **javac** function to compile the Java file into a class file.

**$** **javac -cp "/home/ubuntu/server/spark-2.3.2-bin-hadoop2.7/jars/\*" JavaWordCount.java**

Then, I used the **jar** function to create a JAR file from the class file.

**$ jar cf JavaWordCount.jar JavaWordCount.class**

Finally, I applied the word count operation to my **test.txt** file using the **spark-submit** function.

**$ spark-submit --driver-memory 2g --class JavaWordCount --master local[4] JavaWordCount.jar InputWordCount.txt outputfile/.**

And here my result:

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Troubleshooting during make wordcound example on spark:

‘java.lang.IllegalArgumentException: System memory 243269632 must be at least 471859200. Please increase heap size using the --driver-memory option or spark.driver.memory in Spark configuration. in spark’ error:

To address this issue, I initially increased the **spark.driver.memory** to 5g in the Spark configuration directory. However, despite this adjustment, I continued to encounter the same error. After extensive research on platforms such as GitHub and Stack Overflow, I discovered that adjusting memory settings while running the application using the **spark-submit** method instead of the Java function could potentially resolve the problem.

**6)** **The implementation of sorting software developers' average annual salaries received according to sectors in terms of dollars from highest to lowest in the year 2024 using Python.**

Here is the input:

metin, ekran görüntüsü, yazılım, bilgisayar içeren bir resim

Açıklama otomatik olarak oluşturuldu

My Python codes:

import sys

from pyspark.sql import SparkSession

from pyspark.sql.functions import avg, format\_number

from pyspark import SparkContext

from pyspark.sql.types import FloatType

# Check if the correct number of arguments is provided

if len(sys.argv) != 3:

    print("Usage: spark-submit script.py input\_file output\_file")

    sys.exit(1)

# Extract input and output file paths from command-line arguments

input\_file = sys.argv[1]

output\_file = sys.argv[2]

# Create SparkContext

sc = SparkContext("local", "Average Salary Calculation")

# Create SparkSession

spark = SparkSession.builder \

    .appName("Average Salary Calculation") \

    .getOrCreate()

# Read input CSV file with comma delimiter

df = spark.read \

    .option("header", "true") \

    .option("inferSchema", "true") \

    .option("delimiter", "\t") \

    .csv(input\_file)

# Perform the required DataFrame operations

average\_salary\_df = df.filter(df["work\_year"] == 2024).groupBy("job\_title") \

    .agg(avg("salary").alias("average\_salary"))

# Convert "average\_salary" column to float type

average\_salary\_df = average\_salary\_df.withColumn("average\_salary", average\_salary\_df["average\_salary"].cast(FloatType()))

# Sort the DataFrame by "average\_salary" column in descending order

average\_salary\_df = average\_salary\_df.orderBy("average\_salary", ascending=False)

# Format the average\_salary to 3 decimal places

average\_salary\_df = average\_salary\_df.withColumn("average\_salary", format\_number("average\_salary", 3))

# Write the DataFrame to a CSV file

average\_salary\_df.write \

    .mode("overwrite") \

    .option("header", "true") \

    .csv(output\_file)

# Stop the SparkSession

spark.stop()

# Stop the SparkContext

sc.stop()

Here is the output:

metin, elektronik donanım, ekran görüntüsü, ekran, görüntüleme içeren bir resim

Açıklama otomatik olarak oluşturuldu

Note: I examined the data and indeed confirmed that there are employees earning '3,300,000.000' and '3,000,000.000' salaries.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

Bibliography:

<https://www.ayto.org.tr/tr/kurumsal/102>

<https://maelfabien.github.io/bigdata/Spark/>

<https://www.novixys.com/blog/setup-apache-hadoop-cluster-aws-ec2/>

<https://www.kaggle.com/datasets/chopper53/data-engineer-salary-in-2024>