Spark Machine Learning-Java Application: Read CSV file into Data Frame and Clustering data using KMeans Algorithm(Unsupervised Learning).

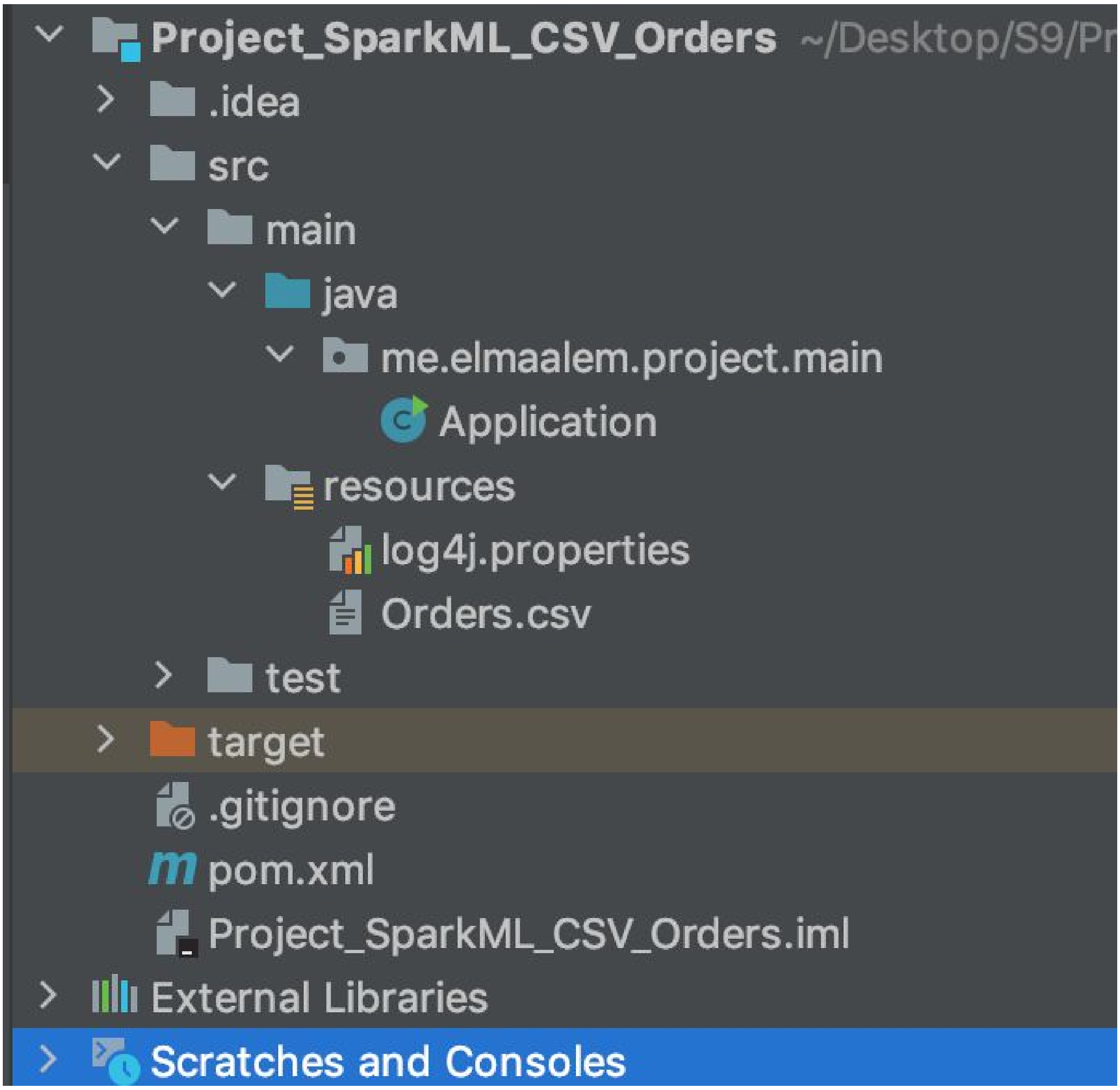
I. Introduction:

In this documentation, we are focused to load data from a CSV file and store them in **Dataset<Row>**, and Identify a number of Clusters (categories) to describe data using KMeans Algorithm. Finally, evaluating Cluster Quality by **Silhouette Score**.

II. Technologies:

- Java 8
- Spark Core 2.4.7
- Spark Machine Learning 2.4.7
- Maven
- Intellij IDEA

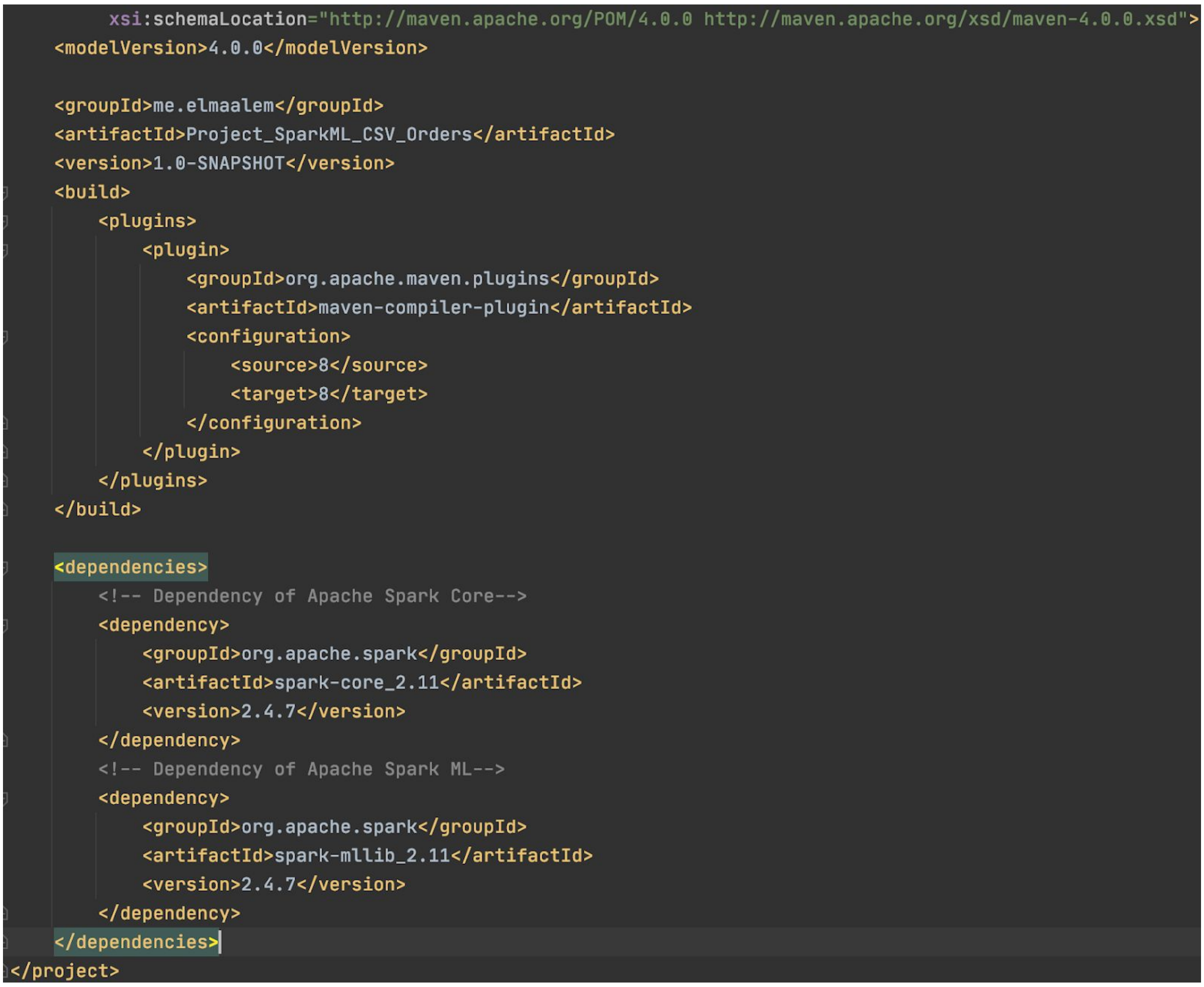
III. Project Structure :



```
Project_SparkML_CSV_Orders ~/Desktop/S9/Pr
  .idea
  src
    main
      java
        me.elmaalem.project.main
          Application
      resources
        log4j.properties
        Orders.csv
    test
  target
  .gitignore
  pom.xml
  Project_SparkML_CSV_Orders.iml
External Libraries
Scratches and Consoles
```

IV. Setup Dependencies on **pom.xml** :

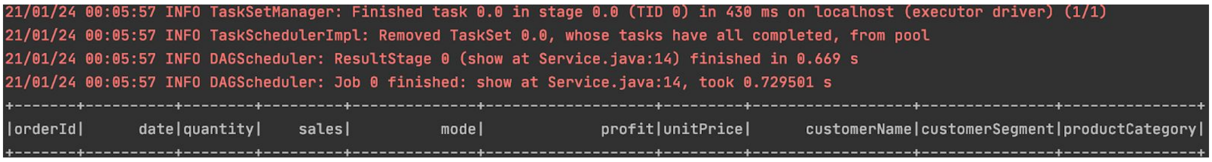
After Adding the below dependencies on **pom.xml**, It will download all the required packages.



```xml
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>me.elmaalem</groupId>
    <artifactId>Project_SparkML_CSV_Orders</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <!-- Dependency of Apache Spark Core-->
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-core_2.11</artifactId>
            <version>2.4.7</version>
        </dependency>
        <!-- Dependency of Apache Spark ML-->
        <dependency>
            <groupId>org.apache.spark</groupId>
            <artifactId>spark-mllib_2.11</artifactId>
            <version>2.4.7</version>
        </dependency>
    </dependencies>
</project>
```

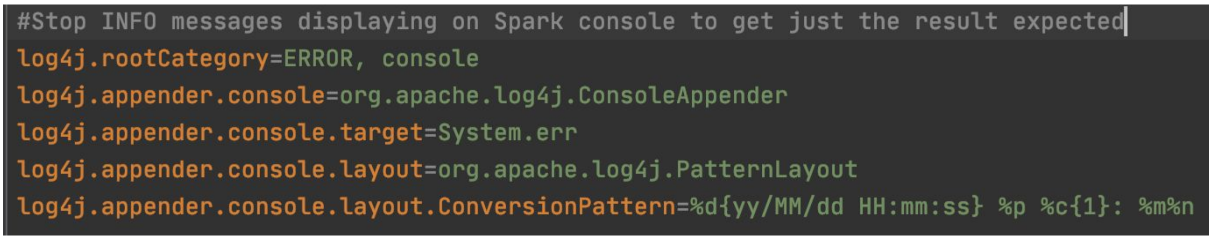
I. Configure Log4j file on spark console :

We would like to stop various **INFO messages** that are coming on the spark console to get just the result on the console without logging messages.



```
21/01/24 00:05:57 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 430 ms on localhost (executor driver) (1/1)
21/01/24 00:05:57 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
21/01/24 00:05:57 INFO DAGScheduler: ResultStage 0 (show at Service.java:14) finished in 0.669 s
21/01/24 00:05:57 INFO DAGScheduler: Job 0 finished: show at Service.java:14, took 0.729501 s
+-------+----------+--------+---------+-------------+------------------+---------+------------------+---------------+---------------+
|orderId|      date|quantity|    sales|         mode|            profit|unitPrice|      customerName|customerSegment|productCategory|
+-------+----------+--------+---------+-------------+------------------+---------+------------------+---------------+---------------+
```

We create a new file **log4j.properties** in order to stop these messages. Here are the contents of **log4j.properties:**



```
#Stop INFO messages displaying on Spark console to get just the result expected
log4j.rootCategory=ERROR, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n
```

V. Main Application & Output:

Let’s create an **Application** class under package **main** to load CSV file into **Dataset<Row>** and Clustering data using **KMeans Algorithm**. Then, display results in the console.

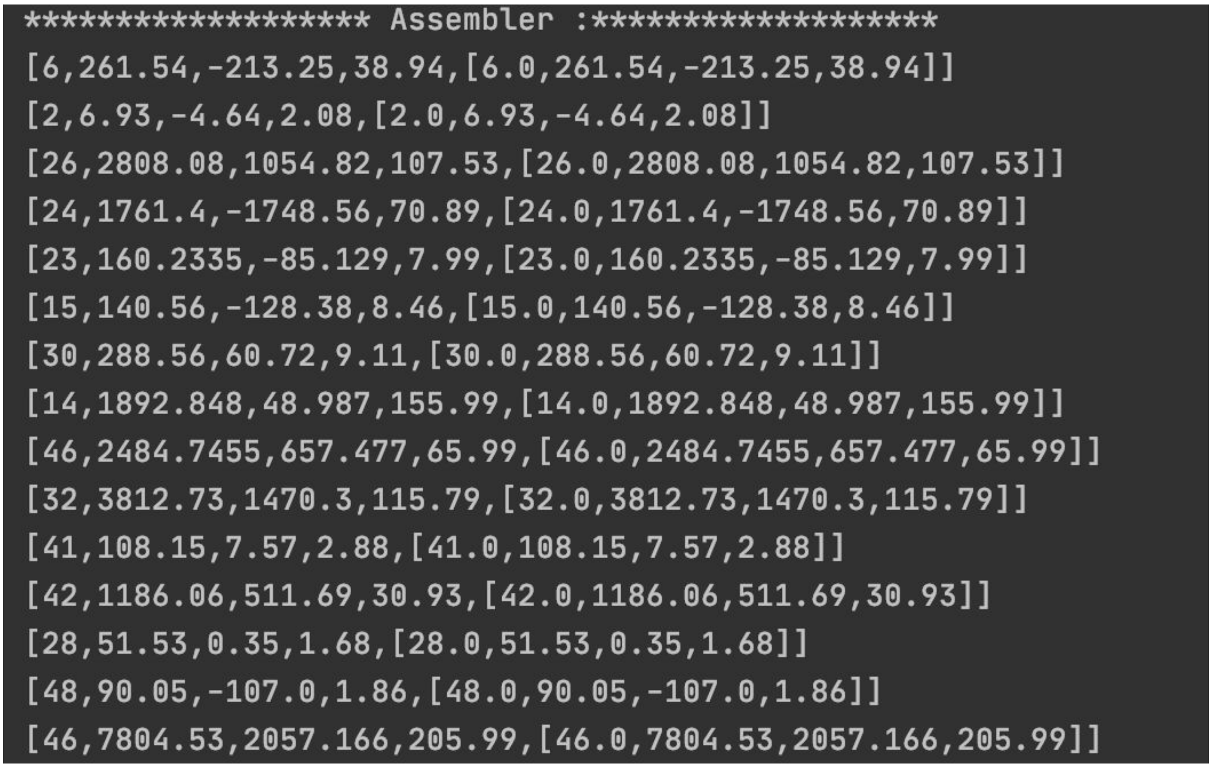
1. *SparkSession & Load CSV file into Dataset<Row>:*

```
SparkSession spark = SparkSession
        .builder()
        .appName("Spring Boot App with Spark SQL")
        .master("local[*]")
        .getOrCreate();

Dataset<Row> csvDF = spark.read()
        .option("header","true")
        .option("treatEmptyValuesAsNulls", "true")
        .option("inferSchema", "true")
        .option("mode","DROPMALFORMED")
        .option("dateFormat", "MM-dd-yyyy")
        .option("delimiter",",")
        .csv("src/main/resources/Orders.csv")
        .select("quantity","sales","profit","unitPrice");
```

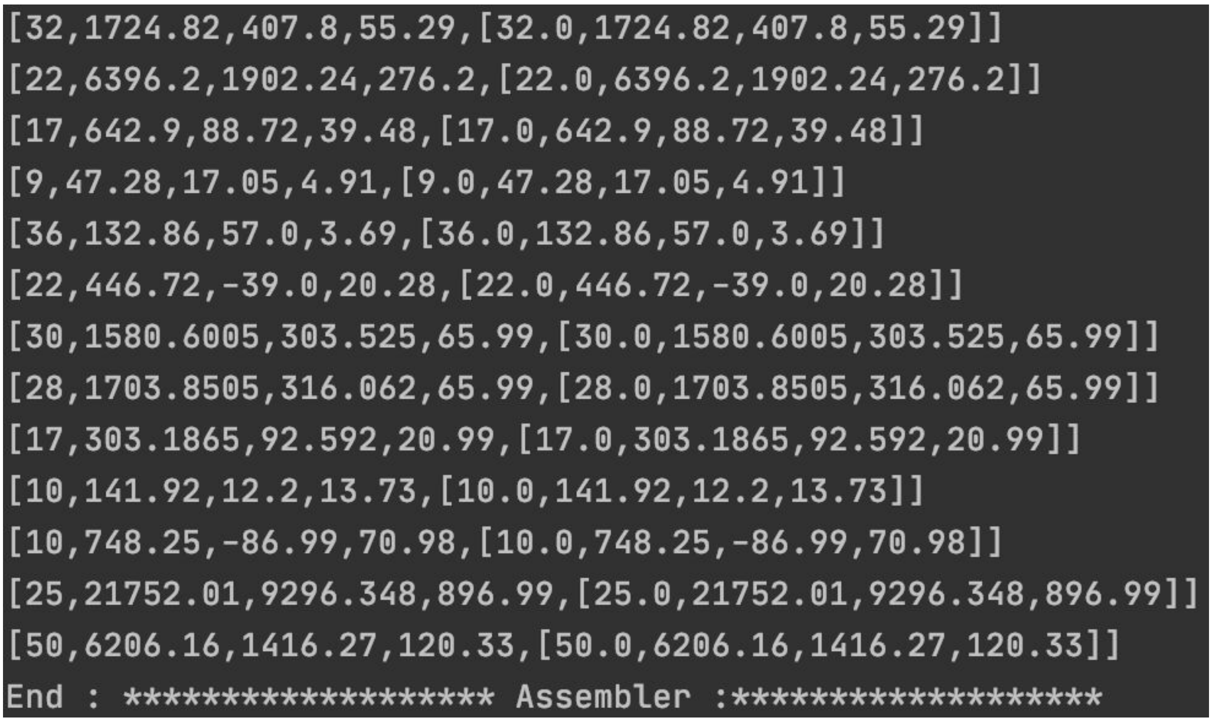
2. *Assembling Columns into Features:*

```
System.out.println("******************* Assembler :*******************");
VectorAssembler assembler = new VectorAssembler()
        .setInputCols(csvDF.columns())
        .setOutputCol("features");

Dataset<Row> orders = assembler.setHandleInvalid("skip").transform(csvDF);
orders.foreach((ForeachFunction<Row>) row-> System.out.println(row));
System.out.println("End : ******************* Assembler
:*******************");
```

->Output:

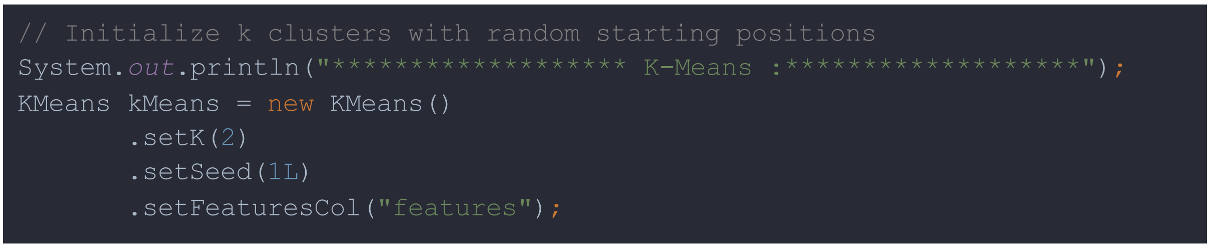


```
******************* Assembler :*******************
[6,261.54,-213.25,38.94,[6.0,261.54,-213.25,38.94]]
[2,6.93,-4.64,2.08,[2.0,6.93,-4.64,2.08]]
[26,2808.08,1054.82,107.53,[26.0,2808.08,1054.82,107.53]]
[24,1761.4,-1748.56,70.89,[24.0,1761.4,-1748.56,70.89]]
[23,160.2335,-85.129,7.99,[23.0,160.2335,-85.129,7.99]]
[15,140.56,-128.38,8.46,[15.0,140.56,-128.38,8.46]]
[30,288.56,60.72,9.11,[30.0,288.56,60.72,9.11]]
[14,1892.848,48.987,155.99,[14.0,1892.848,48.987,155.99]]
[46,2484.7455,657.477,65.99,[46.0,2484.7455,657.477,65.99]]
[32,3812.73,1470.3,115.79,[32.0,3812.73,1470.3,115.79]]
[41,108.15,7.57,2.88,[41.0,108.15,7.57,2.88]]
[42,1186.06,511.69,30.93,[42.0,1186.06,511.69,30.93]]
[28,51.53,0.35,1.68,[28.0,51.53,0.35,1.68]]
[48,90.05,-107.0,1.86,[48.0,90.05,-107.0,1.86]]
[46,7804.53,2057.166,205.99,[46.0,7804.53,2057.166,205.99]]
```



```
[32,1724.82,407.8,55.29,[32.0,1724.82,407.8,55.29]]
[22,6396.2,1902.24,276.2,[22.0,6396.2,1902.24,276.2]]
[17,642.9,88.72,39.48,[17.0,642.9,88.72,39.48]]
[9,47.28,17.05,4.91,[9.0,47.28,17.05,4.91]]
[36,132.86,57.0,3.69,[36.0,132.86,57.0,3.69]]
[22,446.72,-39.0,20.28,[22.0,446.72,-39.0,20.28]]
[30,1580.6005,303.525,65.99,[30.0,1580.6005,303.525,65.99]]
[28,1703.8505,316.062,65.99,[28.0,1703.8505,316.062,65.99]]
[17,303.1865,92.592,20.99,[17.0,303.1865,92.592,20.99]]
[10,141.92,12.2,13.73,[10.0,141.92,12.2,13.73]]
[10,748.25,-86.99,70.98,[10.0,748.25,-86.99,70.98]]
[25,21752.01,9296.348,896.99,[25.0,21752.01,9296.348,896.99]]
[50,6206.16,1416.27,120.33,[50.0,6206.16,1416.27,120.33]]
End : ******************* Assembler :*******************
```

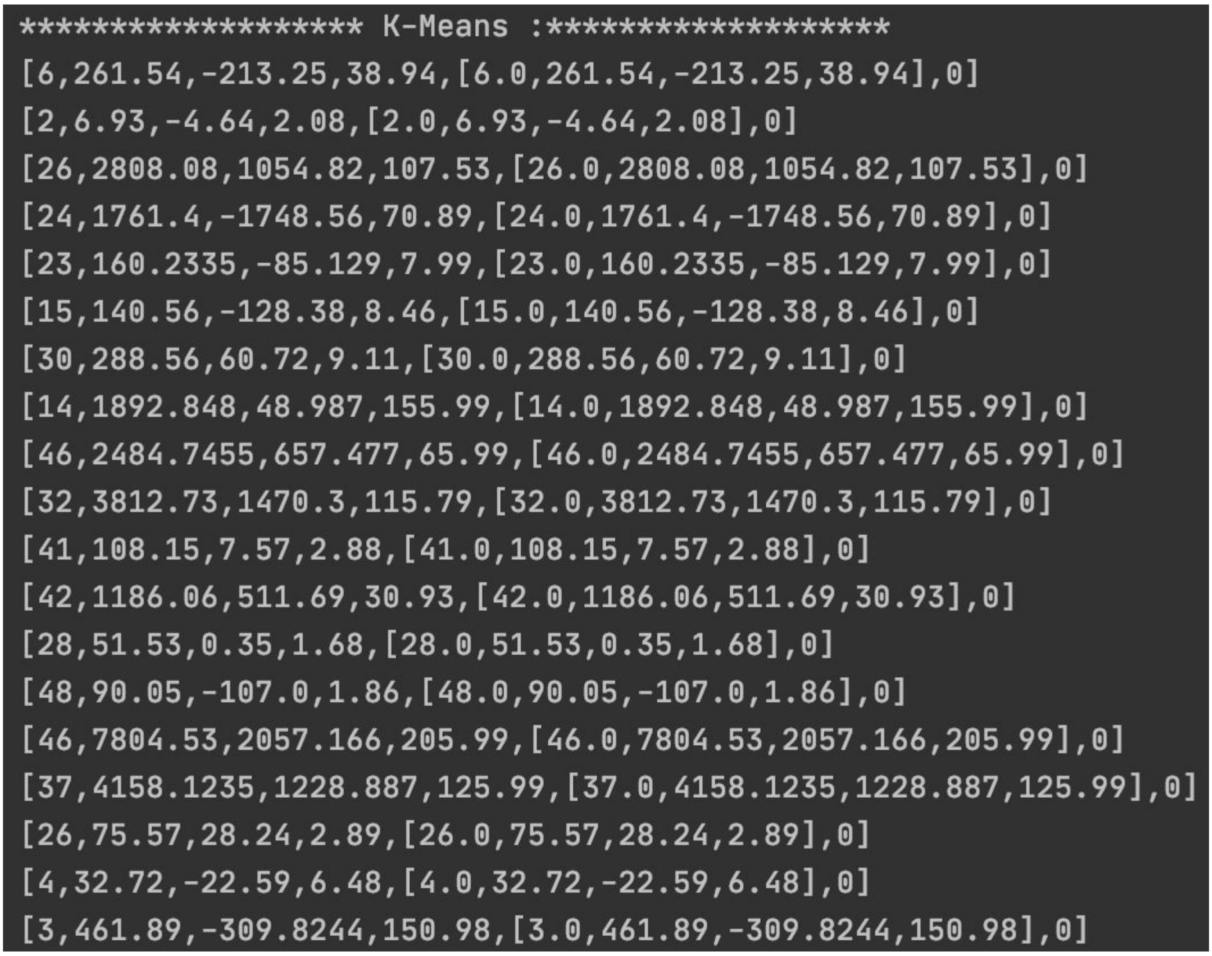
3. ***Initialize k clusters with random starting positions & Trains a k-means model:***



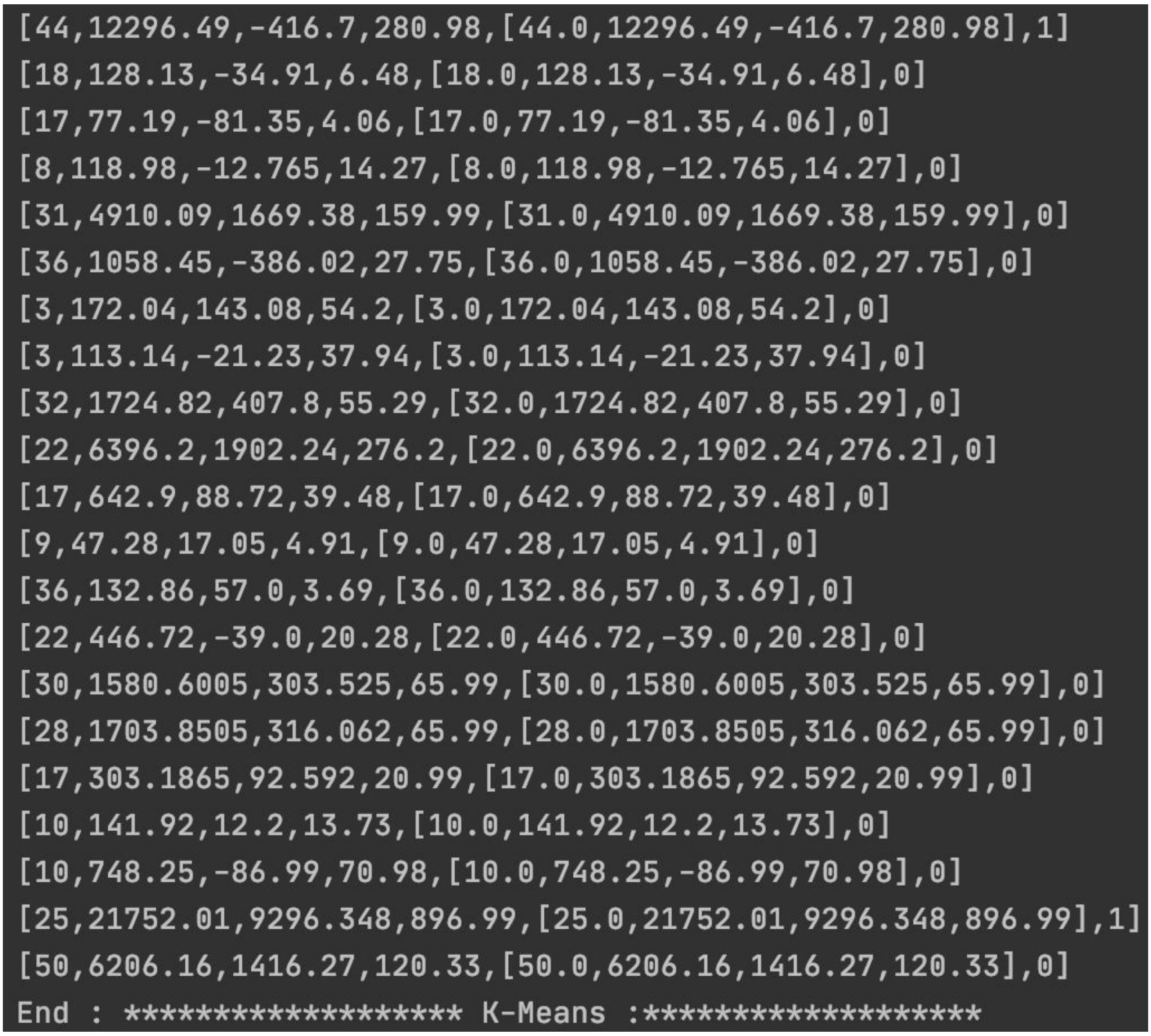
```
// Initialize k clusters with random starting positions
System.out.println("******************* K-Means :*******************");
KMeans kMeans = new KMeans()
        .setK(2)
        .setSeed(1L)
        .setFeaturesCol("features");
```

```
// Trains a k-means model
KMeansModel model = kMeans.fit(orders);
Dataset<Row> predictions = model.transform(orders);

predictions.foreach((ForeachFunction<Row>) row-> System.out.println(row));
System.out.println("End : ******************** K-Means
:********************");
```

->Output:

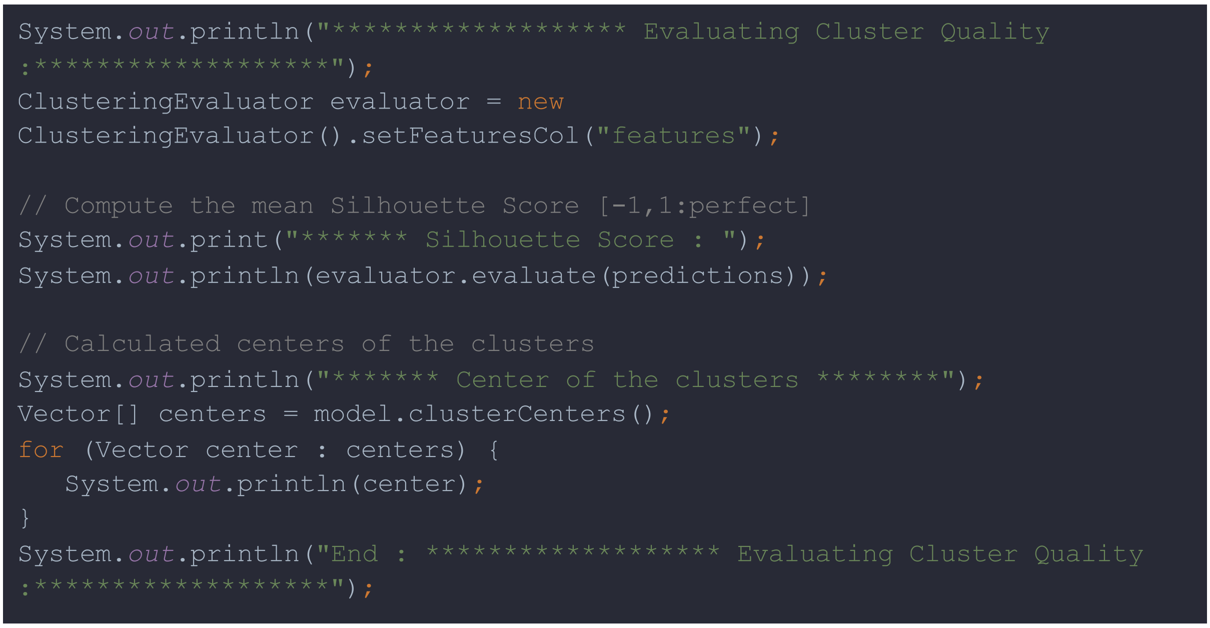


```
******************* K-Means :*******************
[6,261.54,-213.25,38.94,[6.0,261.54,-213.25,38.94],0]
[2,6.93,-4.64,2.08,[2.0,6.93,-4.64,2.08],0]
[26,2808.08,1054.82,107.53,[26.0,2808.08,1054.82,107.53],0]
[24,1761.4,-1748.56,70.89,[24.0,1761.4,-1748.56,70.89],0]
[23,160.2335,-85.129,7.99,[23.0,160.2335,-85.129,7.99],0]
[15,140.56,-128.38,8.46,[15.0,140.56,-128.38,8.46],0]
[30,288.56,60.72,9.11,[30.0,288.56,60.72,9.11],0]
[14,1892.848,48.987,155.99,[14.0,1892.848,48.987,155.99],0]
[46,2484.7455,657.477,65.99,[46.0,2484.7455,657.477,65.99],0]
[32,3812.73,1470.3,115.79,[32.0,3812.73,1470.3,115.79],0]
[41,108.15,7.57,2.88,[41.0,108.15,7.57,2.88],0]
[42,1186.06,511.69,30.93,[42.0,1186.06,511.69,30.93],0]
[28,51.53,0.35,1.68,[28.0,51.53,0.35,1.68],0]
[48,90.05,-107.0,1.86,[48.0,90.05,-107.0,1.86],0]
[46,7804.53,2057.166,205.99,[46.0,7804.53,2057.166,205.99],0]
[37,4158.1235,1228.887,125.99,[37.0,4158.1235,1228.887,125.99],0]
[26,75.57,28.24,2.89,[26.0,75.57,28.24,2.89],0]
[4,32.72,-22.59,6.48,[4.0,32.72,-22.59,6.48],0]
[3,461.89,-309.8244,150.98,[3.0,461.89,-309.8244,150.98],0]
```



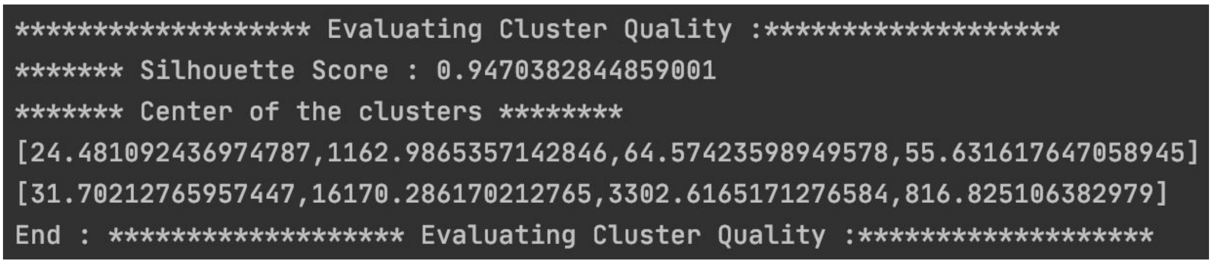
```
[44,12296.49,-416.7,280.98,[44.0,12296.49,-416.7,280.98],1]
[18,128.13,-34.91,6.48,[18.0,128.13,-34.91,6.48],0]
[17,77.19,-81.35,4.06,[17.0,77.19,-81.35,4.06],0]
[8,118.98,-12.765,14.27,[8.0,118.98,-12.765,14.27],0]
[31,4910.09,1669.38,159.99,[31.0,4910.09,1669.38,159.99],0]
[36,1058.45,-386.02,27.75,[36.0,1058.45,-386.02,27.75],0]
[3,172.04,143.08,54.2,[3.0,172.04,143.08,54.2],0]
[3,113.14,-21.23,37.94,[3.0,113.14,-21.23,37.94],0]
[32,1724.82,407.8,55.29,[32.0,1724.82,407.8,55.29],0]
[22,6396.2,1902.24,276.2,[22.0,6396.2,1902.24,276.2],0]
[17,642.9,88.72,39.48,[17.0,642.9,88.72,39.48],0]
[9,47.28,17.05,4.91,[9.0,47.28,17.05,4.91],0]
[36,132.86,57.0,3.69,[36.0,132.86,57.0,3.69],0]
[22,446.72,-39.0,20.28,[22.0,446.72,-39.0,20.28],0]
[30,1580.6005,303.525,65.99,[30.0,1580.6005,303.525,65.99],0]
[28,1703.8505,316.062,65.99,[28.0,1703.8505,316.062,65.99],0]
[17,303.1865,92.592,20.99,[17.0,303.1865,92.592,20.99],0]
[10,141.92,12.2,13.73,[10.0,141.92,12.2,13.73],0]
[10,748.25,-86.99,70.98,[10.0,748.25,-86.99,70.98],0]
[25,21752.01,9296.348,896.99,[25.0,21752.01,9296.348,896.99],1]
[50,6206.16,1416.27,120.33,[50.0,6206.16,1416.27,120.33],0]
End : ******************* K-Means :*******************
```

4. Evaluating Cluster Quality:



```
System.out.println("******************* Evaluating Cluster Quality
:*******************");
ClusteringEvaluator evaluator = new
ClusteringEvaluator().setFeaturesCol("features");

// Compute the mean Silhouette Score [-1,1:perfect]
System.out.print("******* Silhouette Score : ");
System.out.println(evaluator.evaluate(predictions));

// Calculated centers of the clusters
System.out.println("******* Center of the clusters ********");
Vector[] centers = model.clusterCenters();
for (Vector center : centers) {
   System.out.println(center);
}
System.out.println("End : ******************* Evaluating Cluster Quality
:*******************");
```

->Output:



```
******************* Evaluating Cluster Quality :*******************
******* Silhouette Score : 0.9470382844859001
******* Center of the clusters ********
[24.481092436974787,1162.9865357142846,64.57423598949578,55.631617647058945]
[31.70212765957447,16170.286170212765,3302.6165171276584,816.825106382979]
End : ******************* Evaluating Cluster Quality :*******************
```

VI. Wrapping Up:

In this project, we have created a spark application using **Spark Core** and **Spark Machine Learning** with **Java**. Here, we have loaded the CSV file into **Dataset<Row>**. Also, Clustering data using **KMeans Algorithm(Unsupervised Learning)**.
If you want to test the examples above, you will find my Github code link:
Load CSV file into Dataset And Clustering data using K-Means Algorithm