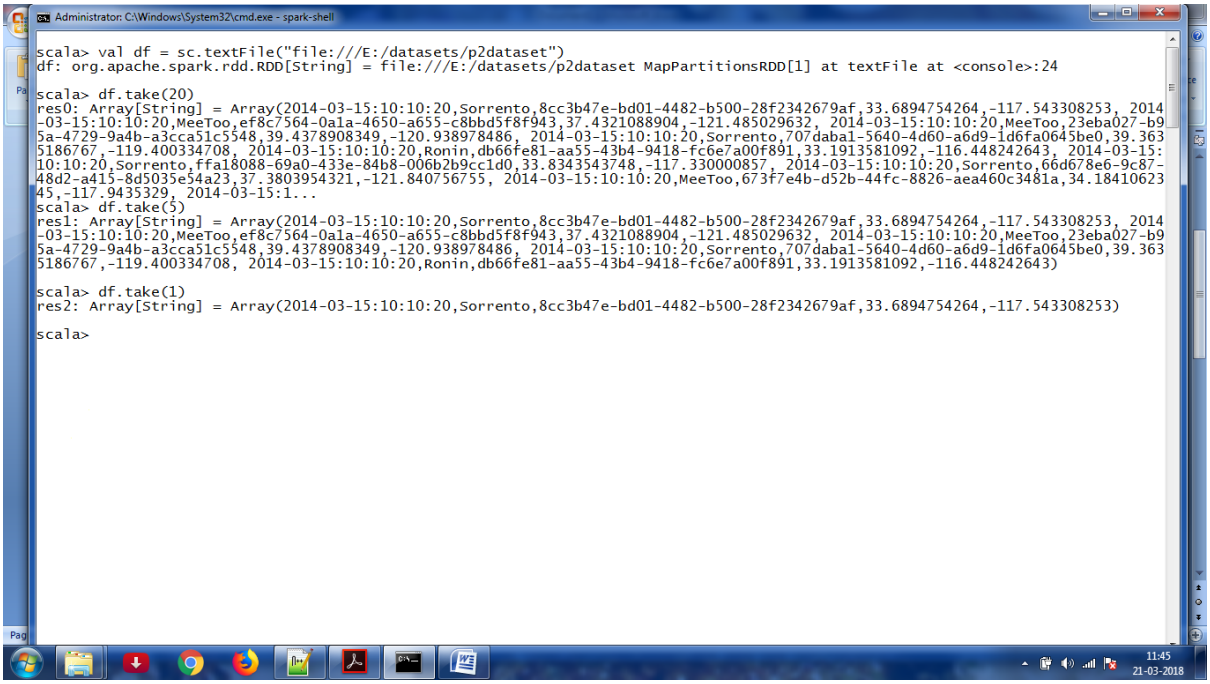
k-means—Real Time Project— LoudAcre Mobile

LoudAcre Mobile is a mobile phone service provider which has introduced a new Open Network campaign. As part of this campaign, the company has invited users to raise a request to initiate a complaint about the towers in their locality, if they face issues with their mobile network. LoudAcre has collected the dataset of users who had raised the complaint. The fourth and the fifth field of dataset has latitude and longitude of users which is an important information for the company. You have to find this information of latitude and longitude on the basis of available dataset and create three clusters of users with k-means algorithm. This will help Loudacre maximize the coverage for its users.

Loading test file and created RDD :

```
val df = sc.textFile("file:///E:/datasets/p2dataset")
df.take(20)
df.take(1)
```



```
Administrator: C:\Windows\System32\cmd.exe - spark-shell

scala> val df = sc.textFile("file:///E:/datasets/p2dataset")
df: org.apache.spark.rdd.RDD[String] = file:///E:/datasets/p2dataset MapPartitionsRDD[1] at textFile at <console>:24

scala> df.take(20)
res0: Array[String] = Array(2014-03-15:10:10:20,Sorrento,8cc3b47e-bd01-4482-b500-28f2342679af,33.6894754264,-117.543308253, 2014
-03-15:10:10:20,MeeToo,ef8c7564-0a1a-4650-a655-c8bbd5f8f943,37.4321088904,-121.485029632, 2014-03-15:10:10:20,MeeToo,23eba027-b9
5a-4729-9a4b-a3cca51c5548,39.4378908349,-120.938978486, 2014-03-15:10:10:20,Sorrento,707daba1-5640-4d60-a6d9-1d6fa0645be0,39.363
5186767,-119.400334708, 2014-03-15:10:10:20,Ronin,db66fe81-aa55-43b4-9418-fc6e7a00f891,33.1913581092,-116.448242643, 2014-03-15:
10:10:20,Sorrento,ffa18088-69a0-433e-84b8-006b2b9cc1d0,33.8343543748,-117.330000857, 2014-03-15:10:10:20,Sorrento,66d678e6-9c87-
48d2-a415-8d5035e54a23,37.3803954321,-121.840756755, 2014-03-15:10:10:20,MeeToo,673f7e4b-d52b-44fc-8826-aea460c3481a,34.18410623
45,-117.9435329, 2014-03-15:1...
scala> df.take(5)
res1: Array[String] = Array(2014-03-15:10:10:20,Sorrento,8cc3b47e-bd01-4482-b500-28f2342679af,33.6894754264,-117.543308253, 2014
-03-15:10:10:20,MeeToo,ef8c7564-0a1a-4650-a655-c8bbd5f8f943,37.4321088904,-121.485029632, 2014-03-15:10:10:20,MeeToo,23eba027-b9
5a-4729-9a4b-a3cca51c5548,39.4378908349,-120.938978486, 2014-03-15:10:10:20,Sorrento,707daba1-5640-4d60-a6d9-1d6fa0645be0,39.363
5186767,-119.400334708, 2014-03-15:10:10:20,Ronin,db66fe81-aa55-43b4-9418-fc6e7a00f891,33.1913581092,-116.448242643)

scala> df.take(1)
res2: Array[String] = Array(2014-03-15:10:10:20,Sorrento,8cc3b47e-bd01-4482-b500-28f2342679af,33.6894754264,-117.543308253)

scala>
```

Convert to dataframe:

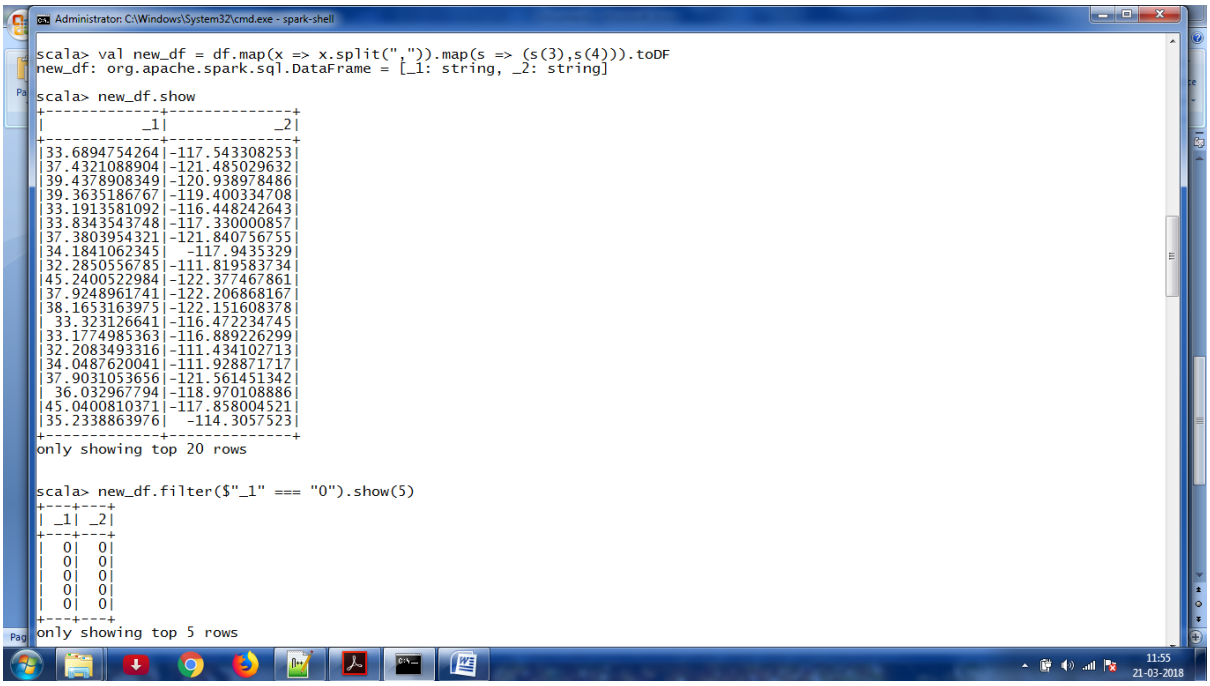
val new_df = df.map(x => x.split(",")).map(s => (s(3),s(4))).toDF

Display dataframe :

new_df.show

Check, is “0” value is present of not :

new_df.filter($"_1" === "0").show(5)



```
Administrator: C:\Windows\System32\cmd.exe - spark-shell

scala> val new_df = df.map(x => x.split(",")).map(s => (s(3),s(4))).toDF
new_df: org.apache.spark.sql.DataFrame = [_1: string, _2: string]

scala> new_df.show
+-------------+--------------+
|           _1|            _2|
+-------------+--------------+
|33.6894754264|-117.543308253|
|37.4321088904|-121.485029632|
|39.4378908349|-120.938978486|
|39.3635186767|-119.400334708|
|33.1913581092|-116.448242643|
|33.8343543748|-117.330000857|
|37.3803954321|-121.840756755|
|34.1841062345|  -117.9435329|
|32.2850556785|-111.819583734|
|45.2400522984|-122.377467861|
|37.9248961741|-122.206868167|
|38.1653163975|-122.151608378|
| 33.323126641|-116.472234745|
|33.1774985363|-116.889226299|
|32.2083493316|-111.434102713|
|34.0487620041|-111.928871717|
|37.9031053656|-121.561451342|
| 36.032967794|-118.970108886|
|45.0400810371|-117.858004521|
|35.2338863976|  -114.3057523|
+-------------+--------------+
only showing top 20 rows


scala> new_df.filter($"_1" === "0").show(5)
+---+---+
| _1| _2|
+---+---+
|  0|  0|
|  0|  0|
|  0|  0|
|  0|  0|
|  0|  0|
+---+---+
only showing top 5 rows
```

Count how many “0” value is present in dataframe. It should be remove from dataframe. Because “0” value does not mention any location :

```
new_df.filter($"_1" === "0").count
val new_df2 = new_df.filter($"_1" =!= "0")
```

Check , is there any “0” value in new dataframe :

```
new_df2.filter($"_1" === "0").count
```

Again convert Dataframe to RDD which store list value :

```
val predata = new_df2.rdd.map(row => List(row.getString(0),row.getString(1)))
```

Display data:

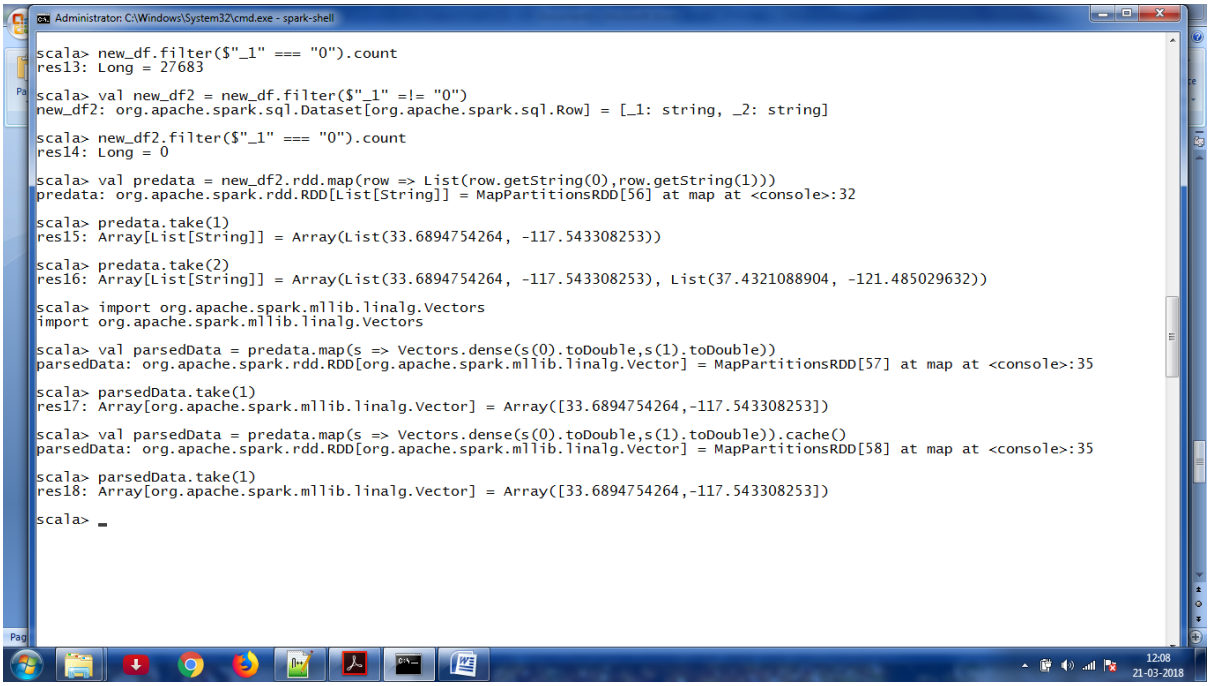
```
predata.take(1)
predata.take(2)
```

Import Vectors and create a Vector RDD :

```
import org.apache.spark.mllib.linalg.Vectors
val parsedData = predata.map(s => Vectors.dense(s(0).toDouble,s(1).toDouble)).cache()
```

Display data:

```
parsedData.take(1)
```



Import Kmeans and KeansModel :

import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}

Here Initialised cluster no . and Iteration no :

val numClusters = 3

val numIterations = 20

Run model with parsedData and others value :

val clusters = KMeans.train(parsedData, numClusters, numIterations)

Display cluster centers which created by this model :

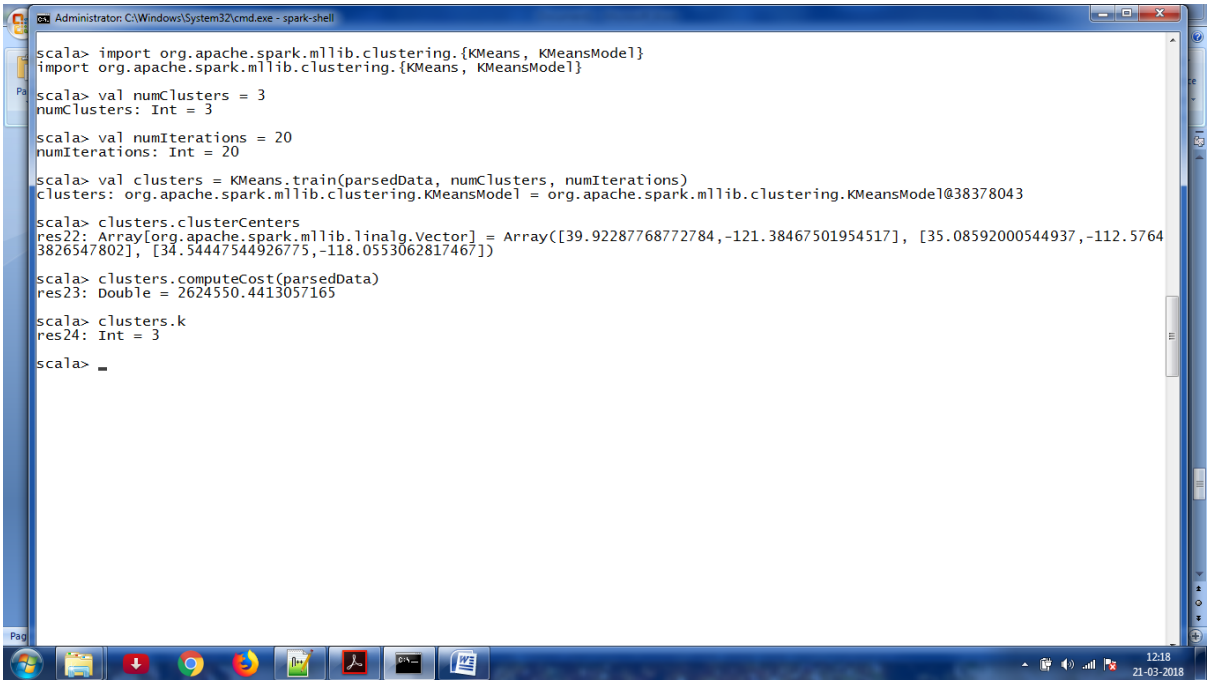
clusters.clusterCenters

Display compute Cost which is Within Sum of Squared Errors :

clusters.computeCost(parsedData)

Display cluster no :

clusters.k



Here created broadcast variable for kmeans model :

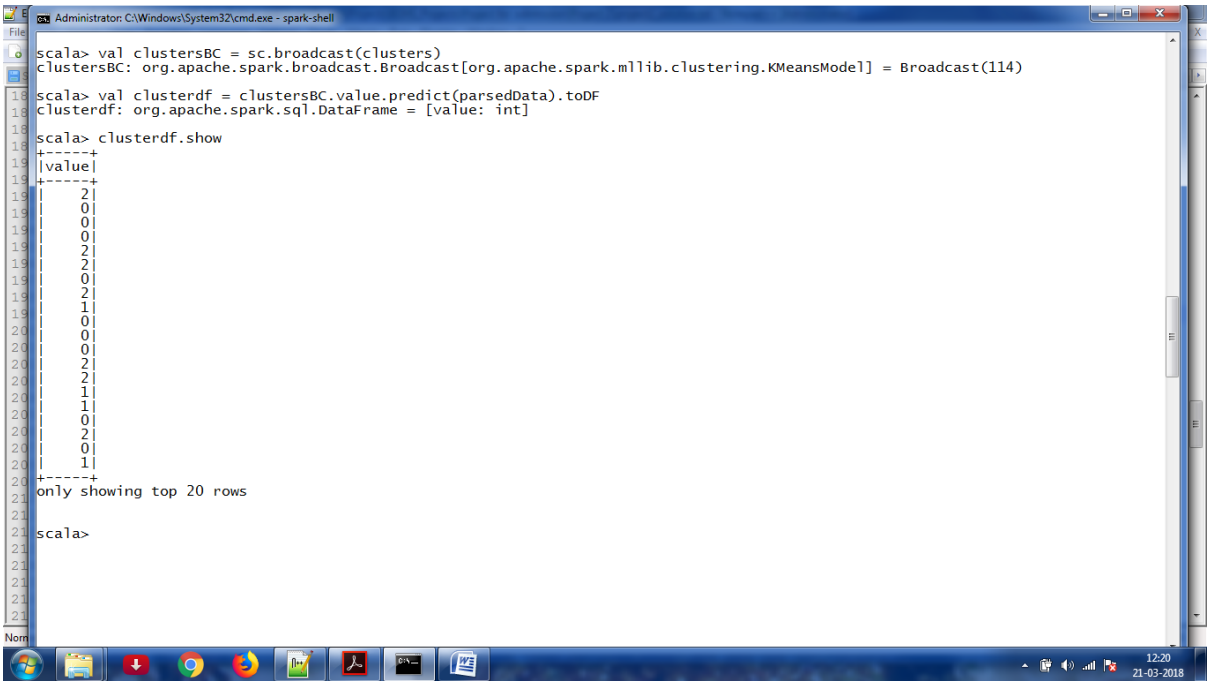
val clustersBC = sc.broadcast(clusters)

Predicted value base on parsedData and convert to a dataframe :

val clusterdf = clustersBC.value.predict(parsedData).toDF

Display dataframe :

clusterdf.show

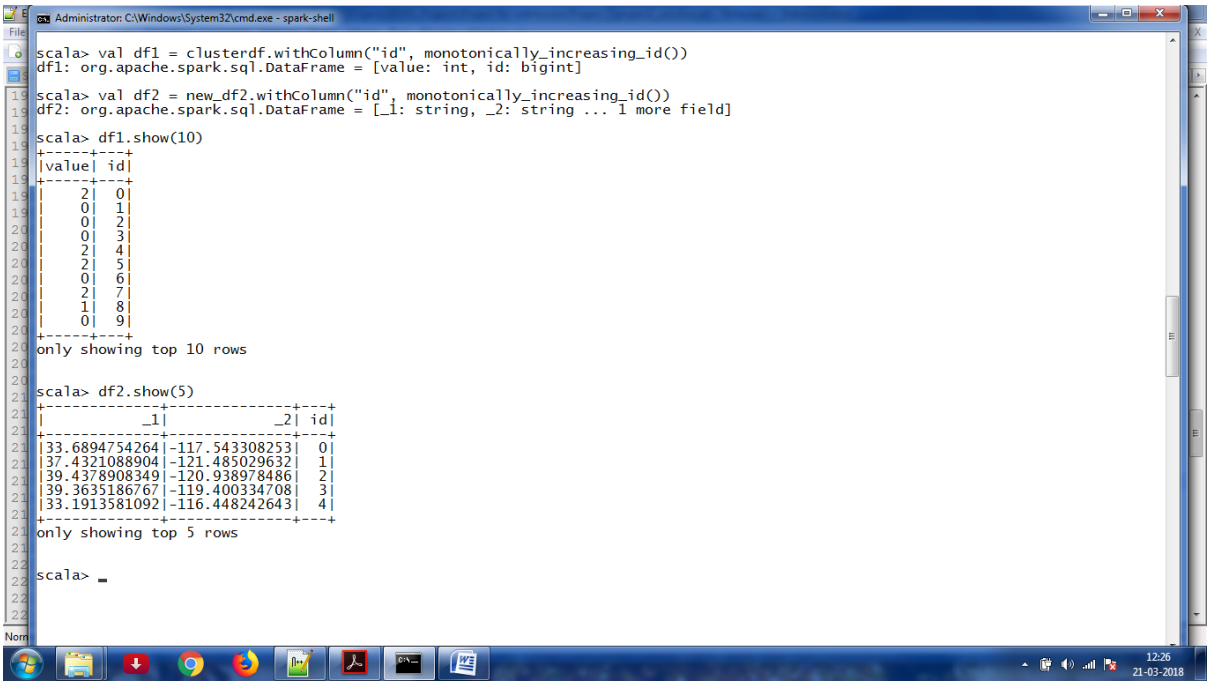


Here added "Id" columns for each dataframe for join two dataframe :

```
val df1 = clusterdf.withColumn("id", monotonically_increasing_id())
val df2 = new_df2.withColumn("id", monotonically_increasing_id())
```

Display dataframe :

```
df1.show(10)
df2.show(5)
```



Join two dataframe base on "id" :

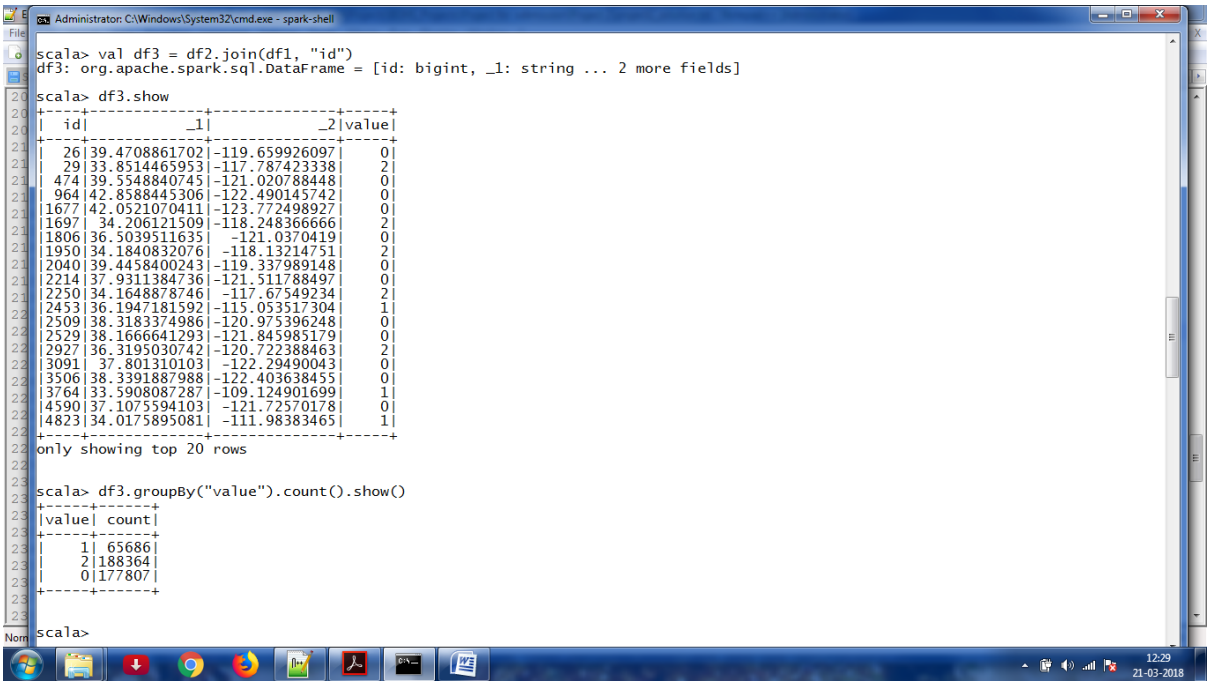
val df3 = df2.join(df1, "id")

Display dataframe :

df3.show

Here count how many value are distributed for each cluster :

df3.groupBy("value").count().show()



Here mention column names :

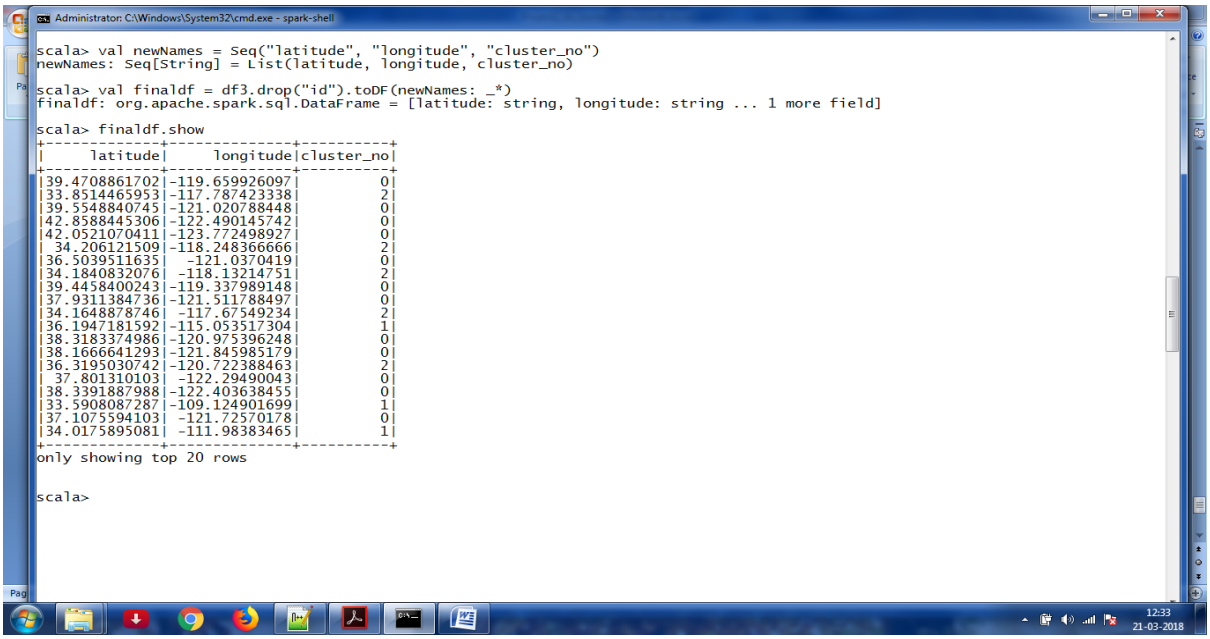
val newNames = Seq("latitude", "longitude", "cluster_no")

Change columns name after drop "Id" column :

val finaldf = df3.drop("id").toDF(newNames: _*)

Display dataframe :

finaldf.show



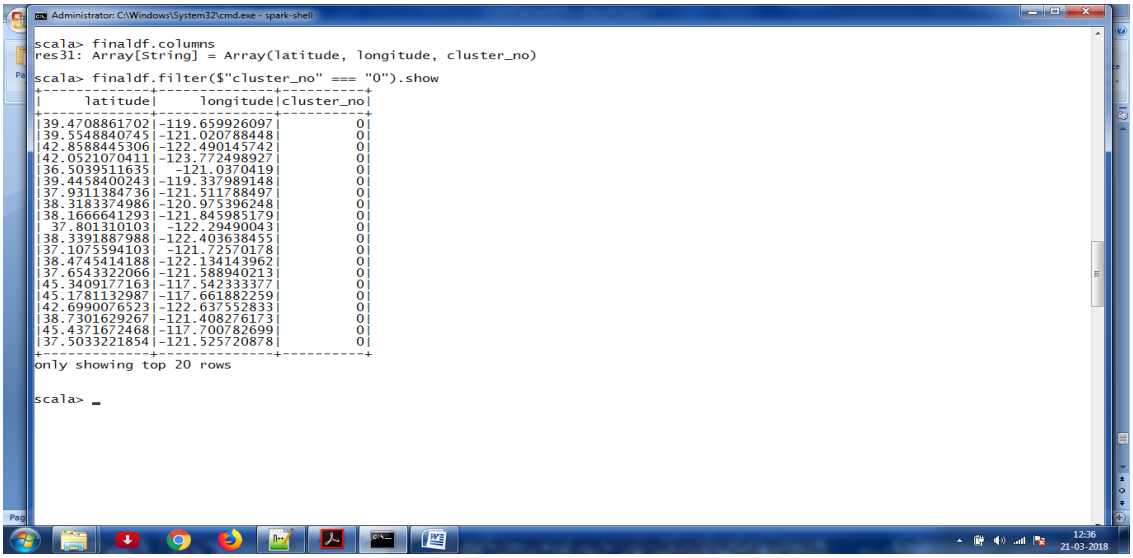
```
scala> val newNames = Seq("latitude", "longitude", "cluster_no")
newNames: Seq[String] = List(latitude, longitude, cluster_no)

scala> val finaldf = df3.drop("id").toDF(newNames: _*)
finaldf: org.apache.spark.sql.DataFrame = [latitude: string, longitude: string ... 1 more field]

scala> finaldf.show
+-------------+--------------+----------+
|     latitude|     longitude|cluster_no|
+-------------+--------------+----------+
|39.4708861702|-119.659926097|         0|
|33.8514465953|-117.787423338|         2|
|39.5548840745|-121.020788448|         0|
|42.8588445306|-122.490145742|         0|
|42.0521070411|-123.772498927|         0|
| 34.206121509|-118.248366666|         2|
|36.5039511635|  -121.0370419|         0|
|34.1840832076| -118.13214751|         2|
|39.4458400243|-119.337989148|         0|
|37.9311384736|-121.511788497|         0|
|34.1648878746| -117.67549234|         2|
|36.1947181592|-115.053517304|         1|
|38.3183374986|-120.975396248|         0|
|38.1666641293|-121.845985179|         0|
|36.3195030742|-120.722388463|         2|
| 37.801310103| -122.29490043|         0|
|38.3391887988|-122.40363845|         0|
|33.5908087287|-109.124901699|         1|
|37.1075594103| -121.72570178|         0|
|34.0175895081| -111.98383465|         1|
+-------------+--------------+----------+
only showing top 20 rows

scala>
```

Display column names :

finaldf.columns

Display only 1st Cluster with latitude and longitude :

finaldf.filter($"cluster_no" === "0").show



```
scala> finaldf.columns
res31: Array[String] = Array(latitude, longitude, cluster_no)

scala> finaldf.filter($"cluster_no" === "0").show
+-------------+--------------+----------+
|     latitude|     longitude|cluster_no|
+-------------+--------------+----------+
|39.4708861702|-119.659926097|         0|
|39.5548840745|-121.020788448|         0|
|42.8588445306|-122.490145742|         0|
|42.0521070411|-123.772498927|         0|
|36.5039511635|  -121.0370419|         0|
|39.4458400243|-119.337989148|         0|
|37.9311384736|-121.511788497|         0|
|38.3183374986|-120.975396248|         0|
|38.1666641293|-121.845985179|         0|
| 37.801310103| -122.29490043|         0|
|38.3391887988|-122.40363845|         0|
|37.1075594103| -121.72570178|         0|
|38.4745414188|-122.134143962|         0|
|37.6543322066|-121.588940213|         0|
|45.3409177163|-117.542333377|         0|
|45.1781132987|-117.661882250|         0|
|42.6990076523|-122.637552833|         0|
|38.7301629267|-121.408276173|         0|
|45.4371672468|-117.700782699|         0|
|37.5033221854|-121.525720878|         0|
+-------------+--------------+----------+
only showing top 20 rows

scala> _
```

Display only 2nd Cluster with latitude and longitude :

finaldf.filter($"cluster_no" === "1").show

```
scala> finaldf.filter($"cluster_no" === "1").show
+-------------+-------------+----------+
|     latitude|    longitude|cluster_no|
+-------------+-------------+----------+
|36.1947181592|-115.053517304|         1|
|33.5908087287|-109.124901699|         1|
|34.0175895081| -111.98383465|         1|
|39.6628949081| -114.562891137|         1|
|33.6230375795|-111.542615553|         1|
|35.0789202973|-111.408292531|         1|
|36.0838304774|-114.731425185|         1|
|36.2981261276|-113.613971565|         1|
|32.2734814339|-111.641629449|         1|
|33.5206745813|-112.003312523|         1|
| 36.661607919|-115.006223963|         1|
|33.3561047647|-111.484732167|         1|
|33.3954391971|-111.861572714|         1|
|32.2866036613|-111.791915607|         1|
|32.3938041755|-111.506443379|         1|
|33.6603116416|-111.513103129|         1|
|35.0455801515|-111.316399941|         1|
|33.7285608333|-111.825495401|         1|
|32.5350707596|-110.614014808|         1|
|33.6029085259|-111.304704398|         1|
+-------------+-------------+----------+
only showing top 20 rows

scala>
```

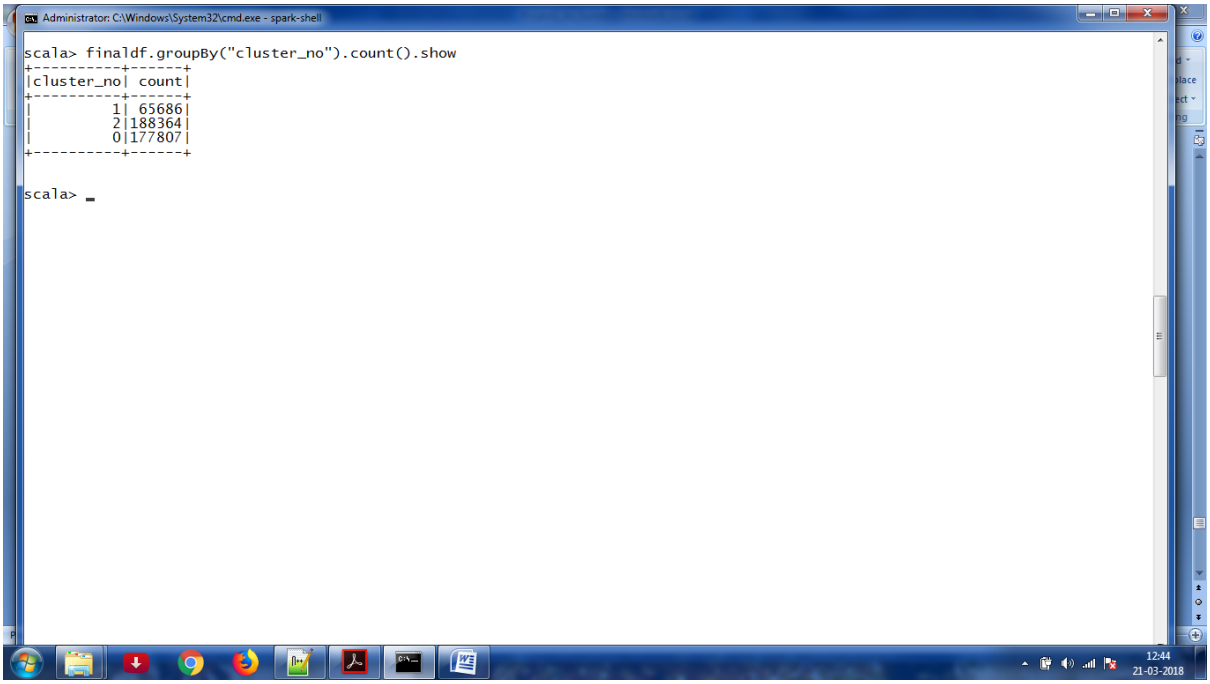
Display only 3rd Cluster with latitude and longitude :

finaldf.filter($"cluster_no" === "2").show

```
scala> finaldf.filter($"cluster_no" === "2").show
+-------------+-------------+----------+
|     latitude|    longitude|cluster_no|
+-------------+-------------+----------+
|33.8514465953|-117.787423338|         2|
| 34.206121509|-118.248366666|         2|
|34.1840832076| -118.13214751|         2|
|34.1648878746| -117.67549234|         2|
|36.3195030742|-120.722388463|         2|
|34.2298811925|-117.877102556|         2|
|34.0669923073|-118.192176947|         2|
|33.0262077564|-116.812861708|         2|
|34.1521333194|-116.638723297|         2|
|32.9596188021|-116.805661125|         2|
|33.9249343611|-117.881397679|         2|
|33.0239999005|-116.674524717|         2|
|33.0339156527|-116.837511787|         2|
| 37.038360427|-119.588745345|         2|
|32.9965920016|-116.685453769|         2|
|35.6625077365|-120.285702621|         2|
|34.2486240248|  -118.2386197|         2|
|33.1863998188|-116.724525124|         2|
|32.9127253094|-116.307005506|         2|
|33.8233479881|-117.770690361|         2|
+-------------+-------------+----------+
only showing top 20 rows

scala>
```

Here count how many value are distributed for each cluster in finaldf :

finaldf.groupBy("cluster_no").count().show



Here saved text file to a particular path :
finaldf.rdd.saveAsTextFile("file:///E:/datasets/project2output")

Here load text file from that path and created RDD :
val textdata = sc.textFile("file:///E:/datasets/project2output/")

Display save data :
textdata.take(5)

Save model to a particular path :
clusters.save(sc, "file:///E:/datasets/KMeansModel")

Load the model from that path :
val samemodel = KMeansModel.load(sc, "file:///E:/datasets/KMeansModel")

Verify model :

Display cluster no. :
samemodel.k

Display compute Cost which is Within Sum of Squared Errors :
samemodel.computeCost(parsedData)

Display cluster Center :
samemodel.clusterCenters