DS 503 - Project 3

By :
Abdulaziz Alajaji - asalajaji@wpi.edu
Yousef Fadila - yousef@fadila.net

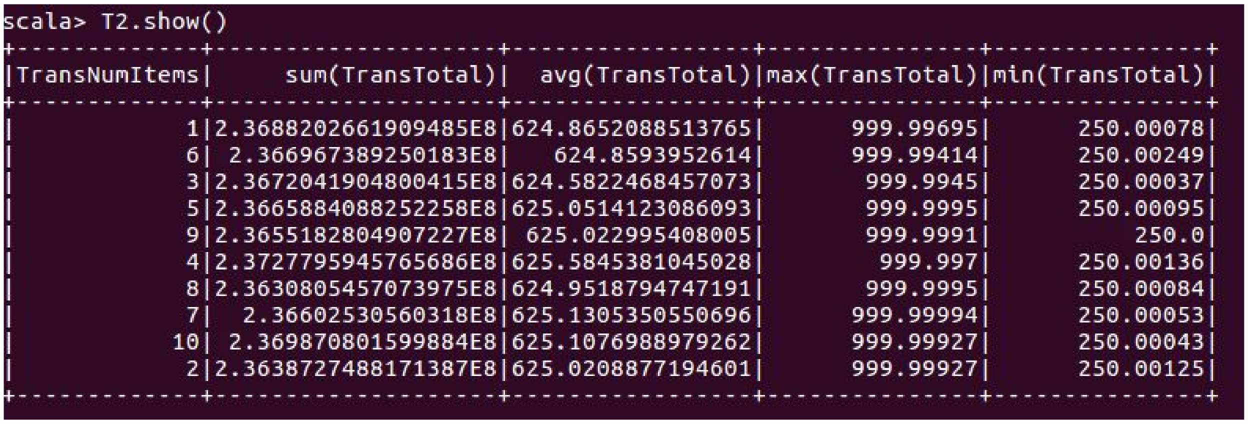
Question 1)

The code is supposed to run under spark-shell
It reads the Transactions file from /home/mqp/Transactions

Find the code with the on **sparkSql.scala** file or below with output sample

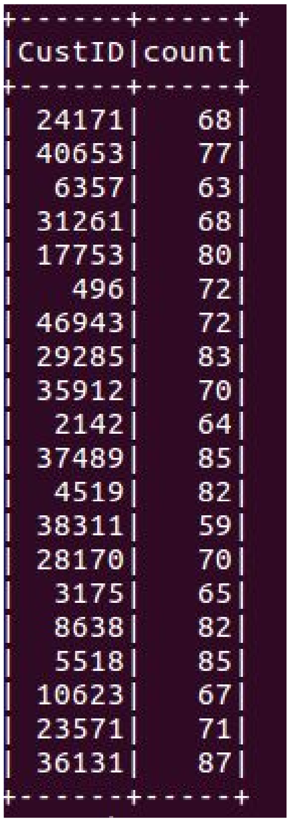
```
case class Record(TransID: Int, CustID: Int, TransTotal: Float, TransNumItems:Int, TransDesc:String)

val data = sc.textFile("/home/mqp/Transactions").map(line => line.split(",")).map {
  case Array(r1,r2,r3,r4,r5) => Record(r1.toInt, r2.toInt, r3.toFloat, r4.toInt, r5)
}
val T0 = data.toDF()
val T1 = T0.where(T0("TransTotal") >= 250)
val T2 = T1.groupBy(T1("TransNumItems")).agg(sum("TransTotal"), avg("TransTotal"), max("TransTotal"), min("TransTotal"))
T2.show()
```



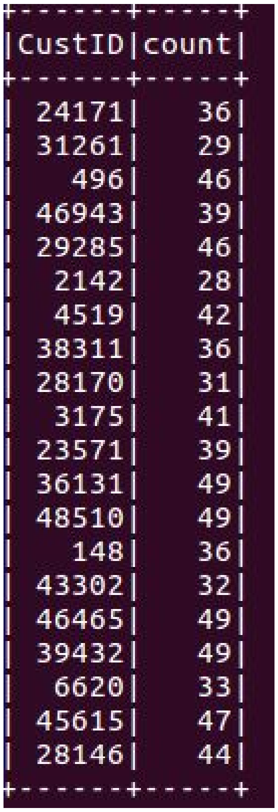
```
scala> T2.show()
+-------------+--------------------+-----------------+---------------+---------------+
|TransNumItems|     sum(TransTotal)|  avg(TransTotal)|max(TransTotal)|min(TransTotal)|
+-------------+--------------------+-----------------+---------------+---------------+
|            1|2.3688202661909485E8|624.8652088513765|      999.99695|      250.00078|
|            6| 2.366967389250183E8|   624.8593952614|      999.99414|      250.00249|
|            3|2.3672041904800415E8|624.5822468457073|       999.9945|      250.00037|
|            5|2.3665884088252258E8|625.0514123086093|       999.9995|      250.00095|
|            9|2.3655182804907227E8| 625.022995408005|       999.9991|          250.0|
|            4|2.3727795945765686E8|625.5845381045028|        999.997|      250.00136|
|            8|2.3630805457073975E8|624.9518794747191|       999.9995|      250.00084|
|            7|  2.36602530560318E8|625.1305350550696|      999.99994|      250.00053|
|           10| 2.369870801599884E8|625.1076988979262|      999.99927|      250.00043|
|            2|2.3638727488171387E8|625.0208877194601|      999.99927|      250.00125|
+-------------+--------------------+-----------------+---------------+---------------+
```

```
val T3 = T1.groupBy(T1("CustID")).count()
T3.show()
```



```
+------+-----+
|CustID|count|
+------+-----+
| 24171|   68|
| 40653|   77|
|  6357|   63|
| 31261|   68|
| 17753|   80|
|   496|   72|
| 46943|   72|
| 29285|   83|
| 35912|   70|
|  2142|   64|
| 37489|   85|
|  4519|   82|
| 38311|   59|
| 28170|   70|
|  3175|   65|
|  8638|   82|
|  5518|   85|
| 10623|   67|
| 23571|   71|
| 36131|   87|
+------+-----+
```

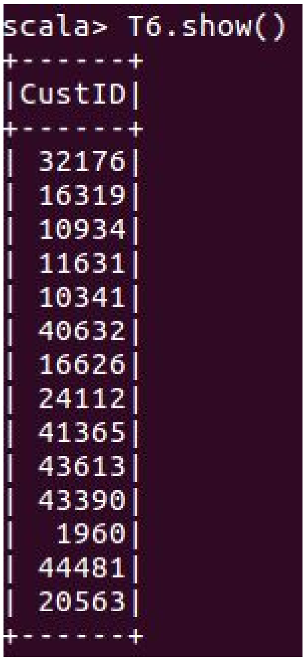
```
val T4 = T0.where(T0("TransTotal") >= 600)
val T5 = T4.groupBy(T1("CustID")).count()
T5.show()
```



```
+------+-----+
|CustID|count|
+------+-----+
| 24171|   36|
| 31261|   29|
|   496|   46|
| 46943|   39|
| 29285|   46|
|  2142|   28|
|  4519|   42|
| 38311|   36|
| 28170|   31|
|  3175|   41|
| 23571|   39|
| 36131|   49|
| 48510|   49|
|   148|   36|
| 43302|   32|
| 46465|   49|
| 39432|   49|
|  6620|   33|
| 45615|   47|
| 28146|   44|
+------+-----+
```

```
val _T6 = T5.join(T3, "CustID").where(T5("count") * 3 < T3("count"))
_T6.show()
```

```
val T6 = _T6.select("CustID")
T6.show()
```



```
scala> T6.show()
+------+
|CustID|
+------+
| 32176|
| 16319|
| 10934|
| 11631|
| 10341|
| 40632|
| 16626|
| 24112|
| 41365|
| 43613|
| 43390|
|  1960|
| 44481|
| 20563|
+------+
```

Question 2)

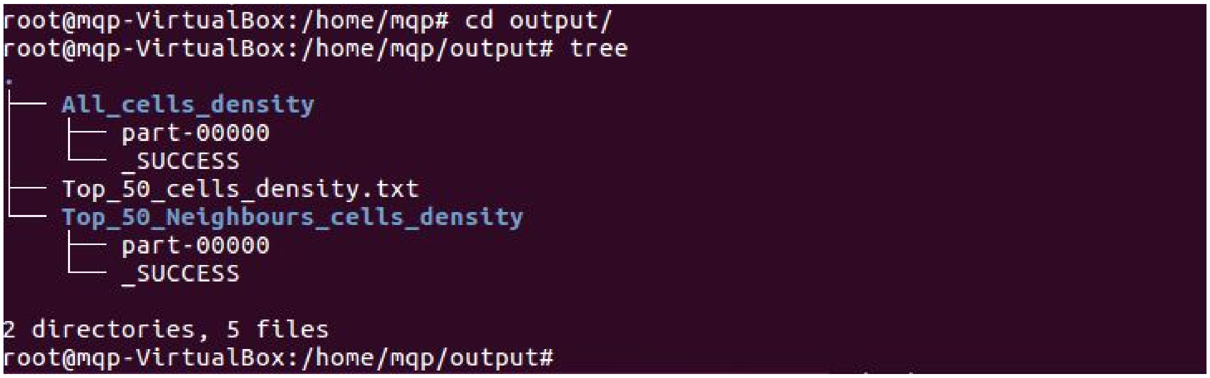
Run the project:

Run mvn package to create project3-1.0.jar

In the virtual machine run
/home/mqp/spark-2.1.0-bin-hadoop2.7/bin/spark-submit --class com.ds503.project3.App \
--master local --deploy-mode client \
project3-1.0.jar **[INPUT_FILE - could be hdfs] [OUTPUT_DIR]**

Example
/home/mqp/spark-2.1.0-bin-hadoop2.7/bin/spark-submit --class com.ds503.project3.App \
--master local --deploy-mode client \
project3-1.0.jar hdfs://localhost:54310/input/p_dataset /home/mqp/output/

After running the code, the output directory [/home/mqp/output/] will include:
1) Top_50_cells_density.txt - answer for step 2 text file includes the top 50; one per line
2) Top_50_Neighbours_cells_density- answer for step 3.
3) All_cells_density - for debugging only.
For example:



```
root@mqp-VirtualBox:/home/mqp# cd output/
root@mqp-VirtualBox:/home/mqp/output# tree
.
├── All_cells_density
│   ├── part-00000
│   └── _SUCCESS
├── Top_50_cells_density.txt
└── Top_50_Neighbours_cells_density
    ├── part-00000
    └── _SUCCESS

2 directories, 5 files
root@mqp-VirtualBox:/home/mqp/output#
```

Code Explained - Utilities:

1) Get the cell number from point

((500 - Y/20) - 1) * 500 + X/20 + 1 = (499 - Y/20) * 500 + X/20 + 1

assumption , the points from [0,10000) range (include 0, not including 10000)

2) Get neighbours per cell:

6	3	5
2	0	1
8	4	7

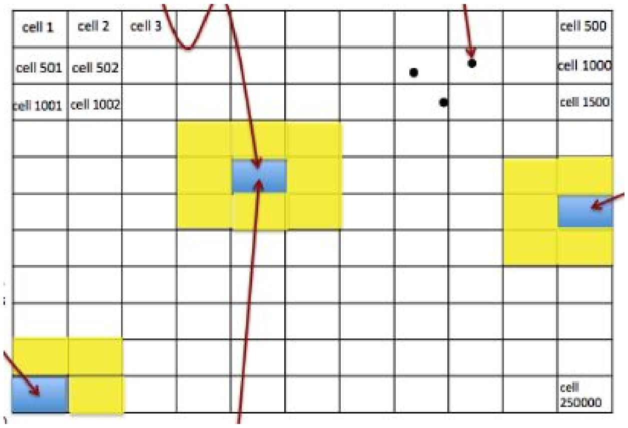
X0: the cell itself;	Neighbour is exist if :
1 - X0 + 1	If (X0 % 500 != 0)
2- X0 -1	If (X0 - 1 % 500 != 0)
3- X0 - 500	If (X0 - 500 > 0)
4 - X0 + 500	If (X0 + 500 <250000)
5- X0 -500 +1	If and only if **both 3 & 1 exist**,
6) X0 -500 -1	If and only if **both 3 & 2 exist,**
7) X0 + 500 +1	If and only if **both 4 & 1 exist,**
8) X0 + 500 -1	If and only if **both 2 & 4 exist,**

The java function implements the logic above is `getNeighbours(int x0);`

the function testGetNeighbours() tests the getNeighbours using cells from the examples given the pdf.

The output of the test is:

```
Neighbours of 1: [2, 501, 502]
Neighbours of 2: [3, 1, 502, 503, 501]
Neighbours of 1000: [999, 500, 1500, 499, 1499]
Neighbours of 1001: [1002, 501, 1501, 502, 1502]
Neighbours of 1002: [1003, 1001, 502, 1502, 503,
501, 1503, 1501]
Neighbours of 1500: [1499, 1000, 2000, 999, 1999]
Neighbours of 250000: [249999, 249500, 249499]
```



Code Explained - Flow:

1st step) read the input file, map each point to the cell it belongs to, then reduce by counting point in each cell

```
JavaRDD<String> input = sc.textFile( inputFile );
// the first pair is cell number, 2nd is the density (count)
JavaPairRDD<Integer, Integer> cellDensity =
       input.mapToPair(new PairFunction<String, Integer, Integer>() {
                           public Tuple2<Integer, Integer> call(String s) {
                               String[] point = s.split(",");
                               int x = Integer.valueOf(point[0].substring(1));
                               int y = Integer.valueOf(point[1].substring(0, point[1].length() -1));
                               int cellNumber = (499 - y / 20) * 500 + x / 20 + 1;
                               return new Tuple2<Integer, Integer>(cellNumber, 1);
                           }
                       }
       ).reduceByKey(
               new Function2<Integer, Integer, Integer>(){
                   public Integer call(Integer x, Integer y){ return x + y; }
               });
```

2nd step) "send" each cell to all cells that require it to calculate the relative density. The neighbourhood is a commutative relationship (if a is neighbour of b then b is neighbour for a too) That means, the cells require cell A for calculating the relative density are the cells that A requires for calculating its own density, so we need to "send" cell A point count value to all it is neighbours.

```
JavaPairRDD<Integer, Iterable<Tuple2<Integer, Integer>>> groupNeighboursByCell =
cellDensity.flatMapToPair(new PairFlatMapFunction<Tuple2<Integer, Integer>, Integer, Tuple2<Integer,
Integer>>() {

   @Override
   public Iterator<Tuple2<Integer, Tuple2<Integer, Integer>>> call(Tuple2<Integer, Integer> cell) throws
Exception {
       int x0 = cell._1();
       Tuple2 cellClone = new Tuple2(cell._1(), cell._2());
       List<Tuple2<Integer, Tuple2<Integer, Integer>>> result = new ArrayList<Tuple2<Integer, Tuple2<Integer,
Integer>>>(8);

       result.add(new Tuple2<Integer, Tuple2<Integer, Integer>>(x0, cellClone));
       List<Integer> neighbours = getNeighbours(x0);
       for (Integer neighbour : neighbours) {
           result.add(new Tuple2<Integer, Tuple2<Integer, Integer>>(neighbour, cellClone));
       }
       return result.iterator();
   }
}).groupByKey();
```

3rd step) now, after all points grouped by center cell, calculate the average by sum all points and divide by ***getNeighboursNumber*** . Please note that we can't count to cells in the iterable to know how many neighbours they had because cells that don't have any point in the dataset won't appear here, but in calculating the average we still need to consider them. The actual number of neighbours is obtained from getNeighboursNumber()

Calculate the relative

```
JavaPairRDD<Integer, Float> cellDensityPairsRDD = groupNeighboursByCell.mapToPair(new
PairFunction<Tuple2<Integer, Iterable<Tuple2<Integer, Integer>>>, Integer, Float>() {
   @Override
   public Tuple2<Integer, Float> call(Tuple2<Integer, Iterable<Tuple2<Integer, Integer>>> cell_neighbours)
throws Exception {
       int cell = cell_neighbours._1();
       int cell_density = 0;

       int sum = 0;
       for (Tuple2<Integer, Integer> t : cell_neighbours._2()) {
           if (t._1() == cell) {
               cell_density = t._2();
               continue;
           }
           sum += t._2();
       }
       int count = getNeighboursNumber(cell);
       float avg = ((float)sum) / count;
       float relativeDensity = avg != 0.0f ? (cell_density / avg): 0.0f;
       return new Tuple2<Integer, Float>(cell, relativeDensity);
   }
});
```

4th step) report the top 50. To do that we use the takeOrdered function with custom comparator that compare based on value not key.

```
List<Tuple2<Integer, Float>> top50 = cellDensityPairsRDD.takeOrdered(50,
ValueComparator.INSTANCE);
```

5th step)
for each of the reported top 50 grid cells, report the IDs and the relative-density indexes of its neighbor cells.
- first we create the top50neighboursRdd , then do a join operation to end up with all neighbours relative-density indexes and then a map in order to get rid of the common column and have only tuple in the output.

```
List<Tuple2<Integer, Integer>> top50neighbours = new ArrayList<Tuple2<Integer, Integer>>(400);
for(Tuple2<Integer, Float> tuple2: top50) {
   for (int n : getNeighbours(tuple2._1())) {
       top50neighbours.add(new Tuple2<Integer, Integer>(n,tuple2._1()));
   }
}

JavaPairRDD<Integer, Integer>  top50neighboursRdd = sc.parallelizePairs(top50neighbours);

JavaPairRDD<Integer, Float> Top50NeighboursRDD =
cellDensityPairsSortedRDD.join(top50neighboursRdd).mapToPair(new PairFunction<Tuple2<Integer, Tuple2<Float,
Integer>>, Integer, Float>() {
   @Override
   public Tuple2<Integer, Float> call(Tuple2<Integer, Tuple2<Float, Integer>> integerTuple2Tuple2) throws
Exception {
       return new Tuple2<Integer, Float>(integerTuple2Tuple2._1(), integerTuple2Tuple2._2()._1());
   }
});

Top50NeighboursRDD.saveAsTextFile(outputDir + "Top_50_Neighbours_cells_density");
```