Project 2 Report

1.Reflection

There are two sub-problems in this project:

Sub-Problem A: HotZone Analysis

For this problem, we are given a rectangle string and a point string. In the rectangle string, we have two coordinates on an x-y plane that represent the diagonal points of the rectangle. I had to complete the ST_Contains function, in which we need to check whether the given point is inside the rectangle boundaries, contributing to the rectangle's hotness. I approached the ST_Contains function by checking:

1. If the x-coordinate of the point lies between the two x-coordinates of the rectangle points.
2. Similarly, if the y-coordinate of the given point lies between the two y-coordinates of the rectangle points.
3. If both the above conditions are satisfied, return True; otherwise, return False.

In the end, using a Spark SQL query, select the rectangle (in terms of its coordinates) and its count (number of points inside that rectangle) while grouping and ordering by the rectangle.

Code snippet of ST_Contains:

```
def ST_Contains(queryRectangle: String, pointString: String ): Boolean = {

val rect_coordinates = queryRectangle.split(",")
val point_coordinates = pointString.split(",")

val lat_point: Double = point_coordinates(0).trim.toDouble
val lon_point: Double = point_coordinates(1).trim.toDouble
val lat1_rec: Double = math.min(rect_coordinates(0).trim.toDouble, rect_coordinates(2).trim.toDouble)
val lat2_rec: Double = math.max(rect_coordinates(0).trim.toDouble, rect_coordinates(2).trim.toDouble)
val lon1_rec: Double = math.min(rect_coordinates(1).trim.toDouble, rect_coordinates(3).trim.toDouble)
val lon2_rec: Double = math.max(rect_coordinates(1).trim.toDouble, rect_coordinates(3).trim.toDouble)

if ((lat_point >= lat1_rec) && (lon_point >= lon1_rec) && (lat_point <= lat2_rec) && (lon_point <= lon2_rec)) {
  return true
}
return false
}
```

Sub-Problem B: Hot cell/Hotspot Analysis

A cell is defined as a point (x, y, z), where (x, y) represents the pickup location in terms of longitude (longitudinal coordinate divided by 0.01) and latitude (latitudinal coordinate divided by 0.01), and z represents the day of the month. For the 2nd function, we need to calculate the hotness of a cell using Getis-Ord statistic which is defined as below:

$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j}x_j - \bar{X}\sum_{j=1}^{n} w_{i,j}}{S\sqrt{\frac{\left[n\sum_{j=1}^{n} w_{i,j}^2 - \left(\sum_{j=1}^{n} w_{i,j}\right)^2\right]}{n-1}}}$$

where xj is the attribute value for cell j, wij is the spatial weight between cell i and j, n is equal to the total number of cells and X-bar & S are as follows:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n} \qquad S = \sqrt{\frac{\sum_{j=1}^{n} x_j^2}{n} - (\bar{X})^2}$$

I calculated the following metrics for each cell:

1. Count: Grouped cells by (x, y, z) and calculated the count for each unique combination.
2. Mean: Obtained the mean by dividing the sum of counts for each cell by the total number of cells (numCells).
3. Standard Deviation: Computed the standard deviation using the formula sqrt (sum of (Count^2) / numCells - mean^2).
4. number Of Adjacent Cells: Difference of 1 unit between one of co-ordinates: (x,y,z).

```
val adj_df = spark.sql("select S1.x as x , S1.y as y, S1.z as z, count(*) as numNeighbors, sum(S2.total_count) as sigma from sample
_data as S1 inner join sample_data as S2 on ((abs(S1.x-S2.x) <= 1 and  abs(S1.y-S2.y) <= 1 and abs(S1.z-S2.z) <= 1)) group by S1.x, S
1.y, S1.z").persist()

adj_df.createOrReplaceTempView("adj_df")
```

5. Finally, using these above metrics, calculate the Zscore:

```
def ST_ZScore(mean: Double, std_dev: Double, numofNeighbors: Int, sigmaxi: Int, numCells: Int): Double =
{
  val numer = sigmaxi - (mean * numofNeighbors)
  val denom = std_dev * Math.sqrt((numCells * numofNeighbors - numofNeighbors * numofNeighbors) / (numCells-1))
  return numer / denom
}
```

At last, I sort these cells according to their Zscore in a descending order and return the top 50 hottest ones.

2. Lessons Learned

This project has equipped me with valuable skills in big data analysis using Apache Spark and Scala. I learned how to install and configure Spark, write SQL queries in Scala, and create user-defined functions. The project also provided hands-on experience in determining the 'hotness' of cells or areas in a dataset, a crucial aspect of spatial analysis.

Additionally, I gained insights into data type transformations, such as converting between double and long in Scala. Understanding how to aggregate data across multiple rows was a key takeaway, offering practical knowledge for real-world data analysis scenarios.

Overall, this project has provided a practical glimpse into the day-to-day work of data scientists and analysts in corporate settings, showcasing the importance of Spark, Scala, and SQL in tackling challenges in data analytics and decision-making.

3. Output/Results

Sub-Problem A:

-73.789411,40.666459,-73.756364,40.680494	1
-73.793638,40.710719,-73.752336,40.730202	1
-73.795658,40.743334,-73.753772,40.779114	1
-73.796512,40.722355,-73.756699,40.745784	1
-73.797297,40.738291,-73.775740,40.770411	1
-73.802033,40.652546,-73.738566,40.668036	8
-73.805770,40.666526,-73.772204,40.690003	3
-73.815233,40.715862,-73.790295,40.738951	2
-73.816380,40.690882,-73.768447,40.715693	1
-73.819131,40.582343,-73.761289,40.609861	1

Sub-Problem B:

-7399	4075	15
-7399	4075	22
-7399	4075	14
-7399	4075	29
-7398	4075	15
-7399	4075	16
-7399	4075	21
-7399	4075	28
-7399	4075	23
-7399	4075	30