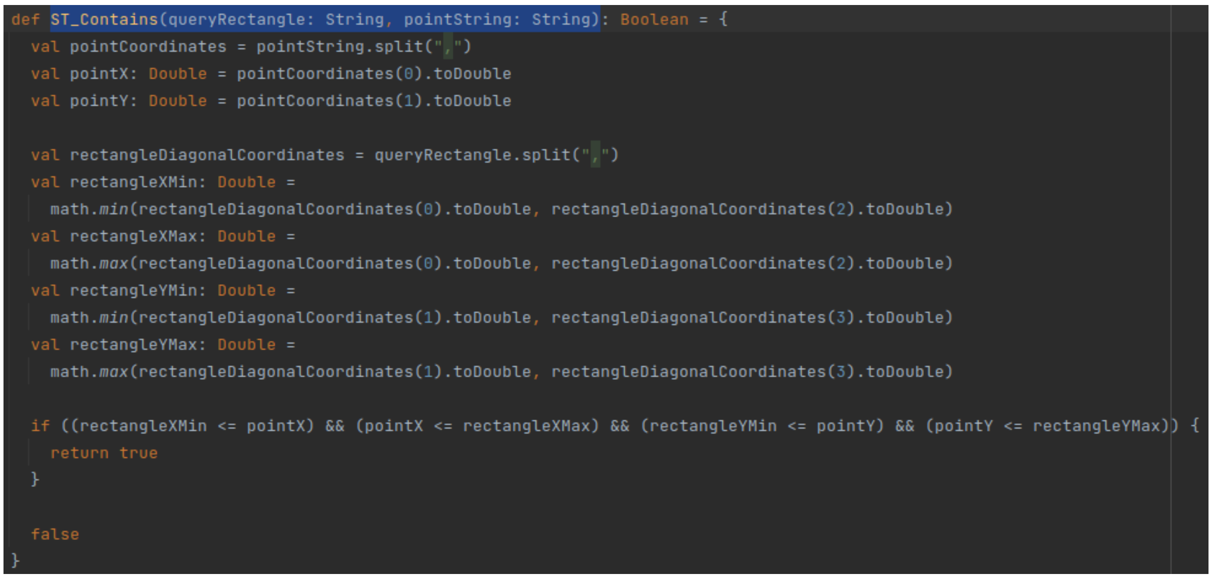
CSE 511: Project 2 Report

Anurag Banerjee

Reflection:

In this assignment, we had to do Hot Zone Analysis and Hot Cell analysis on the New York taxi trips dataset using spark and scala.:

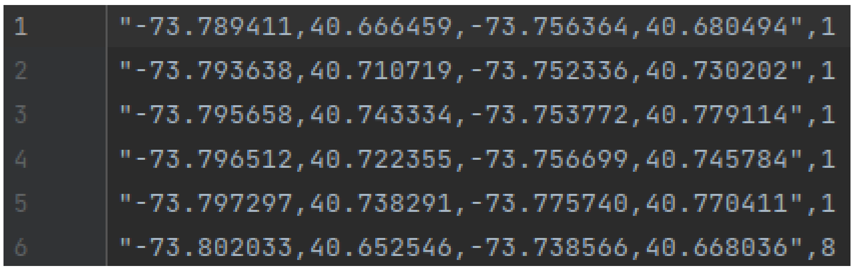
1. **Hot Zone Analysis:**
   In hot zone analysis we needed to find the hotness of a zone. We were given the coordinates of diagonals of rectangles and the zone was the entire area encompassed by the respective rectangles. The hotness of a zone is the total number of taxi trips that started from any point within that zone. We were required to find the hotness of all the zones that were given to us in the **zone-hotzone.csv** file using the points that were provided to us in the **point_hotzone.csv** file.
   We were provided with the code template in which some basic preprocessing was done on the data. A high-level overview of the task that we did was to perform the join operation between the zone (rectangular) dataset and the point dataset. This was achieved as follows
   In order to complete our task first we had to complete the function **ST_Contains(queryRectangle: String, pointString: String)**.
   a) Inside this function, we basically checked whether a point was inside the rectangle that was given to us. The rectangle was provided as the coordinates of the endpoints of a diagonal in the rectangle in a comma-separated format. We calculated the maximum and minimum x and y coordinates of the rectangle from this data.
   b) We considered the point to be inside the zone (rectangle) when the x coordinate and y coordinate were both in between the minimum and maximum x and y coordinates of the rectangle respectively. This was achieved in the code as depicted in the picture below.



```
def ST_Contains(queryRectangle: String, pointString: String): Boolean = {
  val pointCoordinates = pointString.split(",")
  val pointX: Double = pointCoordinates(0).toDouble
  val pointY: Double = pointCoordinates(1).toDouble

  val rectangleDiagonalCoordinates = queryRectangle.split(",")
  val rectangleXMin: Double =
    math.min(rectangleDiagonalCoordinates(0).toDouble, rectangleDiagonalCoordinates(2).toDouble)
  val rectangleXMax: Double =
    math.max(rectangleDiagonalCoordinates(0).toDouble, rectangleDiagonalCoordinates(2).toDouble)
  val rectangleYMin: Double =
    math.min(rectangleDiagonalCoordinates(1).toDouble, rectangleDiagonalCoordinates(3).toDouble)
  val rectangleYMax: Double =
    math.max(rectangleDiagonalCoordinates(1).toDouble, rectangleDiagonalCoordinates(3).toDouble)

  if ((rectangleXMin <= pointX) && (pointX <= rectangleXMax) && (rectangleYMin <= pointY) && (pointY <= rectangleYMax)) {
    return true
  }

  false
}
```

   The **ST_Contains** function was used as a helper when we performed the join of the two datasets. We performed the joining of two rows from the respective datasets when the **ST_Contains** function returned a true value.
   After performing the join, we grouped the rows based on the rectangle and then did a count of the number of rows per rectangle and ordered them by the coordinates of the

rectangle. The first few lines of the output were as follows:

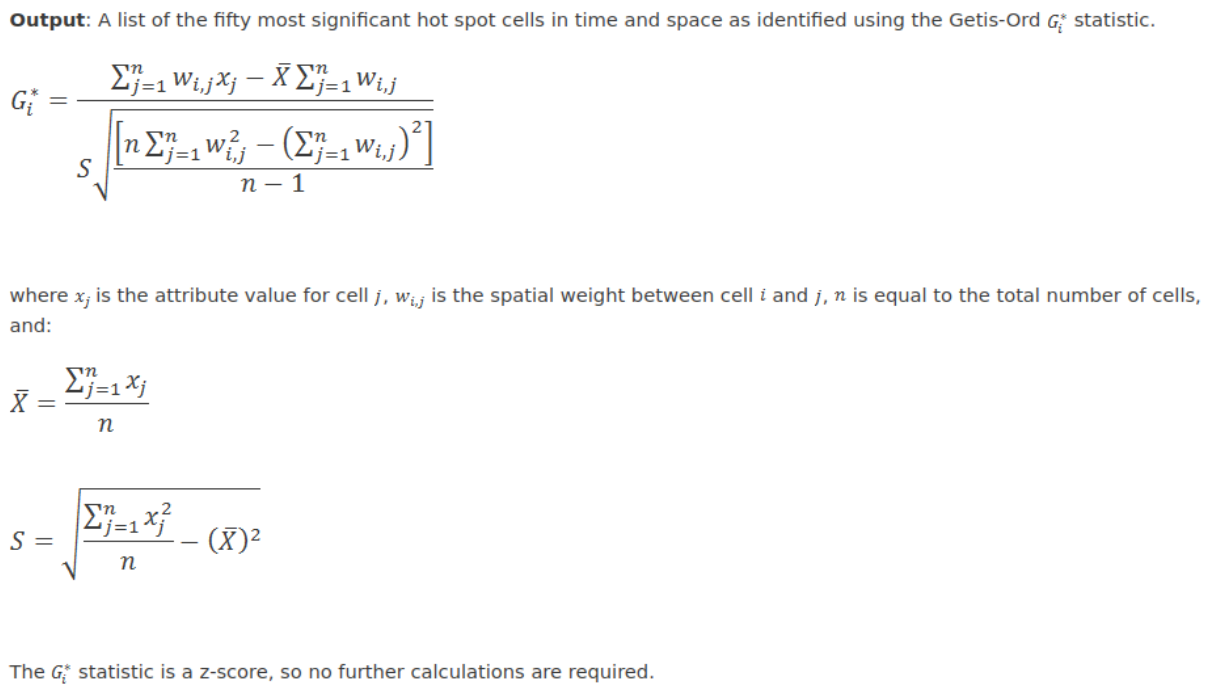


```
"-73.789411,40.666459,-73.756364,40.680494",1
"-73.793638,40.710719,-73.752336,40.730202",1
"-73.795658,40.743334,-73.753772,40.779114",1
"-73.796512,40.722355,-73.756699,40.745784",1
"-73.797297,40.738291,-73.775740,40.770411",1
"-73.802033,40.652546,-73.738566,40.668036",8
```

In this way, we were able to find out the hotness of the zones that were provided to us.

2. **Hot Cell Analysis:**

In hot cell analysis also we were required to find out the hotspots from the given data but the major difference with hot zone analysis is that in hot cell we consider time also. Therefore, we make use of spatio-temporal data to calculate the hotspots. We had to make use of the getis-ord statistic which is depicted in the picture below.



Output: A list of the fifty most significant hot spot cells in time and space as identified using the Getis-Ord G_i^* statistic.

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

where x_j is the attribute value for cell j, $w_{i,j}$ is the spatial weight between cell i and j, n is equal to the total number of cells, and:

$$\bar{X} = \frac{\sum_{j=1}^{n} x_j}{n}$$

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

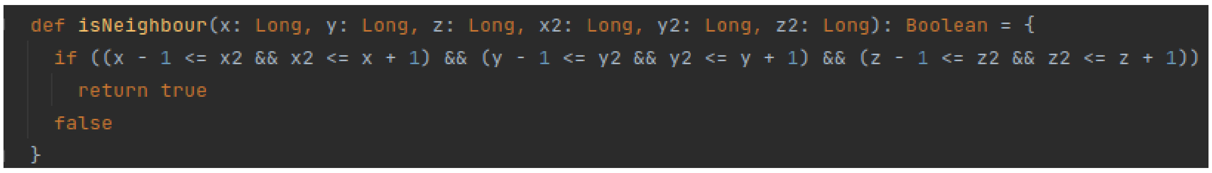
The G_i^* statistic is a z-score, so no further calculations are required.

The hot cell analysis was performed as follows:

a) First, we filtered the points according to the minimum and maximum x, y and z coordinates provided to us. Z coordinate specifies time here. In the same operation, the trips were grouped according to the x, y and z coordinates and the total number of trips for a particular coordinate was calculated which served as xj for the formula shown above.

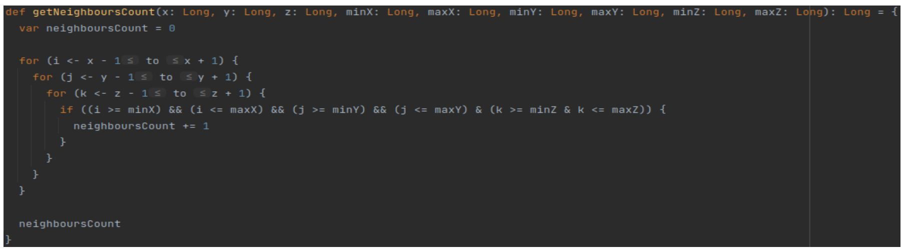
b) Next, we calculated the sum of xj and (xj)^2 using the result in the above step. Then we found out the mean and standard deviation of this data.

c) Next, a utility function was defined to check whether a cell was a neighbor of another cell or not. This was achieved as follows:



```
def isNeighbour(x: Long, y: Long, z: Long, x2: Long, y2: Long, z2: Long): Boolean = {
  if ((x - 1 <= x2 && x2 <= x + 1) && (y - 1 <= y2 && y2 <= y + 1) && (z - 1 <= z2 && z2 <= z + 1))
    return true
  false
}
```

d) Then, another utility function was defined to get the count of neighbors for a particular cell. We checked out of all possible neighbours which were inside the minimum and maximum x, y, z coordinates provided to us. This was achieved in the code as follows.:

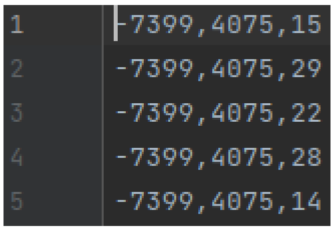


```scala
def getNeighboursCount(x: Long, y: Long, z: Long, minX: Long, maxX: Long, minY: Long, maxY: Long, minZ: Long, maxZ: Long): Long = {
  var neighboursCount = 0

  for (i <- x - 1 to x + 1) {
    for (j <- y - 1 to y + 1) {
      for (k <- z - 1 to z + 1) {
        if ((i >= minX) && (i <= maxX) && (j >= minY) && (j <= maxY) & (k >= minZ & k <= maxZ)) {
          neighboursCount += 1
        }
      }
    }
  }

  neighboursCount
}
```

e) Then we performed a join of the dataset obtained in step a with itself. The join condition was that it should be a neighbor and also found out the number of possible neighbors (sigmaW) for a cell. The sum of count of the neighbor trips gave us the sigmaWX for the above formula.

f) Then we placed all this in the getis ord statistic formula shown above to get the final z scores.

g) We arranged the results in descending order by their z scores. Snapshot of the result is as follows.



```
-7399,4075,15
-7399,4075,29
-7399,4075,22
-7399,4075,28
-7399,4075,14
```

In this way we obtained the hottest cells spatiotemporally.