DATA PROCESSING AT SCALE - CSE 511

Submitted By: Aman Peshin (1225476655)

PROJECT 2 REPORT

Purpose of Project:

Why did we implement hot zone analysis?

ST_Contains function essentially checks if a point is inside rectangular coordinates. The data input are split and corners are found based on which we give a True/False value back.

In this function, we work with some data only. We do a "rangeJoin" query on the rectangle and point dataset. We then count the number of points inside each rectangle and then sort it based on value. If a rectangle has the max points then we can label it as the "hottest" among all rectangles. We also have a function called run_hot_zone_analysis which will sort the data based on point and rectangle Path it will give an output that gives the rectangle and no of points in that particular rectangle.

Why did we implement hot cell analysis?

In this, we process spatial data of New York City taxi trips. We calculate Getis-Ord coordinate statistics to help us in the ST function. Getis-ord Gi is calculated as:

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

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

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

For reach cell we calculate the impact of current cell to its adjacent ones. The Z score is then calculated for each cell and ordered by largest to smallest in the final csv output. We use the concepts of "Spacetime cube" and "SpatialWeight" to calculate scores.

Reflection:

What I particularly enjoyed in this project was the implementation of the function ST_Contains where we had a data point and a rectangle and we needed to find whether the point was inside it. We do range and range join queries inside this function to find all points inside the rectangle and range join to find points and rectangles where the point is inside it. Spatial function, ST Within calculates whether two points are within range for given points and distances. The most enjoyable part was working with apache spark, I thoroughly enjoyed the project right from the setup of JDK, Spark and Microsoft C++ distributions to working with scala files.

DATA PROCESSING AT SCALE - CSE 511

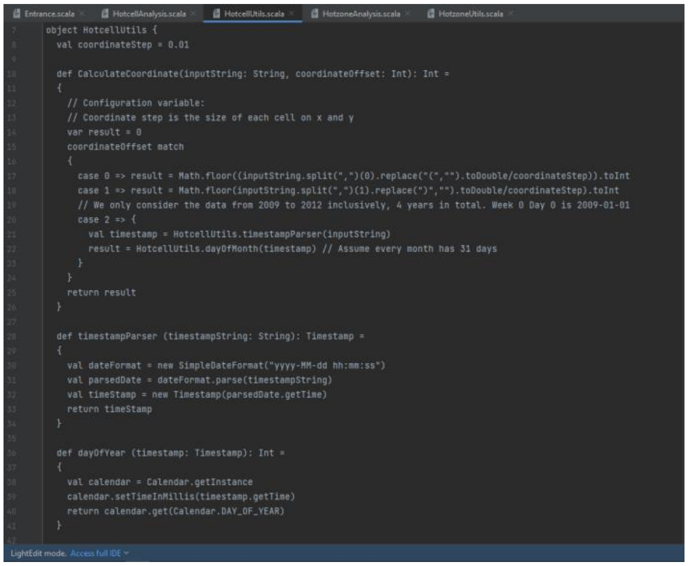
Submitted By: Aman Peshin (1225476655)

Lessons learned:

In this project, we got to explore a lot of new technologies and implement them for hot cell and hot zone analysis. We use scala which supports OOPs and functional programming and runs on JVM. We use Java development kit to run scala. Apache spark is used for large-scale data processing as in the case of yellow taxi 2009 01.csv file. The main point to do this project was to execute spatial queries on large amount of data. Along the way we learned a lot about concepts like Getis-Ord statistic. Scala allows for immutable codes which allows for concurrent transactions and can be run on Java runtime env.

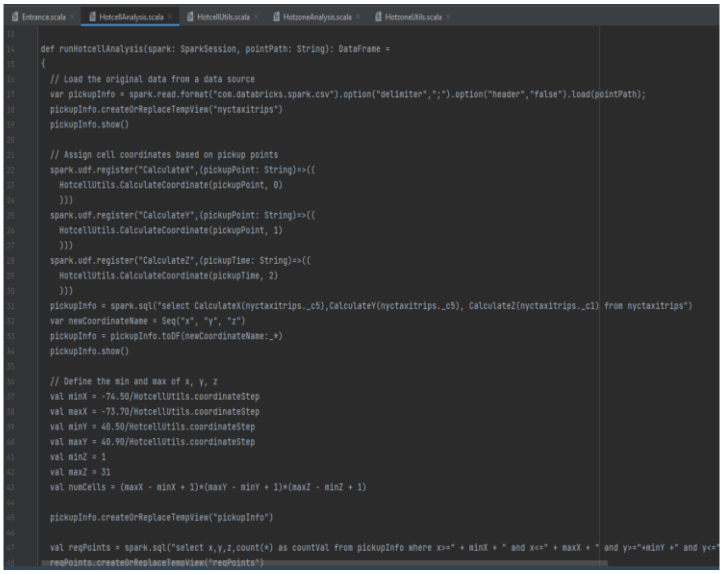
Code:

The code is written in scala and then compiler into a jar file which is then run on inputs to generate outputs which are then compared to sample outputs.



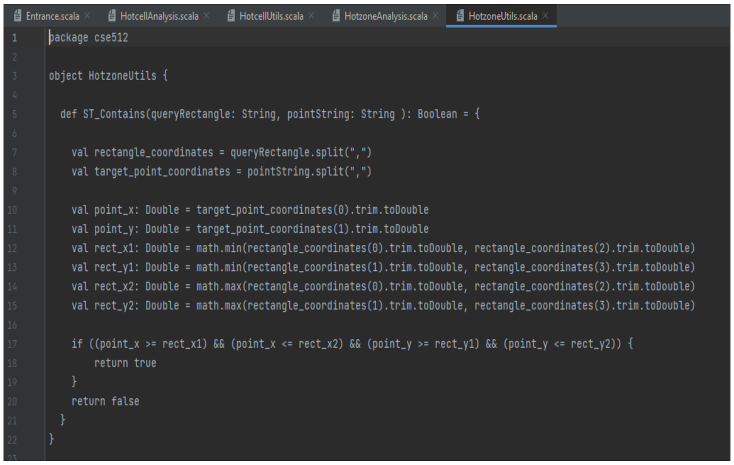
```scala
object HotcellUtils {
  val coordinateStep = 0.01

  def CalculateCoordinate(inputString: String, coordinateOffset: Int): Int =
  {
    // Configuration variable:
    // Coordinate step is the size of each cell on x and y
    var result = 0
    coordinateOffset match
    {
      case 0 => result = Math.floor((inputString.split(",")(0).replace("(","").toDouble/coordinateStep)).toInt
      case 1 => result = Math.floor(inputString.split(",")(1).replace(")","").toDouble/coordinateStep).toInt
      // We only consider the data from 2009 to 2012 inclusively, 4 years in total. Week 0 Day 0 is 2009-01-01
      case 2 => {
        val timestamp = HotcellUtils.timestampParser(inputString)
        result = HotcellUtils.dayOfMonth(timestamp) // Assume every month has 31 days
      }
    }
    return result
  }

  def timestampParser (timestampString: String): Timestamp =
  {
    val dateFormat = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss")
    val parsedDate = dateFormat.parse(timestampString)
    val timeStamp = new Timestamp(parsedDate.getTime)
    return timeStamp
  }

  def dayOfYear (timestamp: Timestamp): Int =
  {
    val calendar = Calendar.getInstance
    calendar.setTimeInMillis(timestamp.getTime)
    return calendar.get(Calendar.DAY_OF_YEAR)
  }
```

HotCellUtils.scala



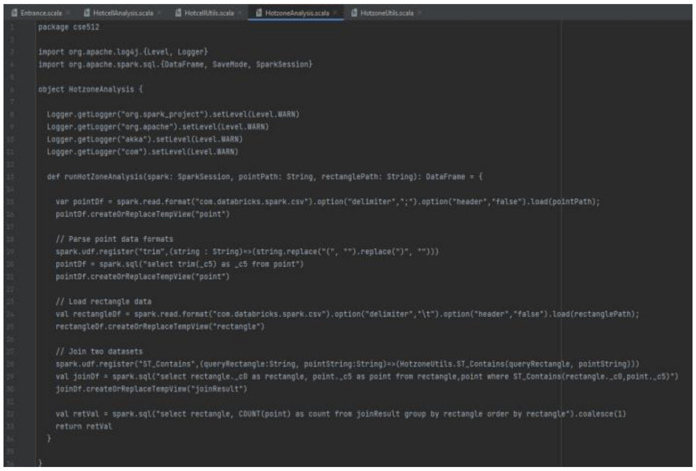
```scala
def runHotcellAnalysis(spark: SparkSession, pointPath: String): DataFrame =
{
  // Load the original data from a data source
  var pickupInfo = spark.read.format("com.databricks.spark.csv").option("delimiter",";").option("header","false").load(pointPath);
  pickupInfo.createOrReplaceTempView("nyctaxitrips")
  pickupInfo.show()

  // Assign cell coordinates based on pickup points
  spark.udf.register("CalculateX",(pickupPoint: String)=>((
    HotcellUtils.CalculateCoordinate(pickupPoint, 0)
    )))
  spark.udf.register("CalculateY",(pickupPoint: String)=>((
    HotcellUtils.CalculateCoordinate(pickupPoint, 1)
    )))
  spark.udf.register("CalculateZ",(pickupTime: String)=>((
    HotcellUtils.CalculateCoordinate(pickupTime, 2)
    )))
  pickupInfo = spark.sql("select CalculateX(nyctaxitrips._c5),CalculateY(nyctaxitrips._c5), CalculateZ(nyctaxitrips._c1) from nyctaxitrips")
  var newCoordinateName = Seq("x", "y", "z")
  pickupInfo = pickupInfo.toDF(newCoordinateName:_*)
  pickupInfo.show()

  // Define the min and max of x, y, z
  val minX = -74.50/HotcellUtils.coordinateStep
  val maxX = -73.70/HotcellUtils.coordinateStep
  val minY = 40.50/HotcellUtils.coordinateStep
  val maxY = 40.90/HotcellUtils.coordinateStep
  val minZ = 1
  val maxZ = 31
  val numCells = (maxX - minX + 1)*(maxY - minY + 1)*(maxZ - minZ + 1)

  pickupInfo.createOrReplaceTempView("pickupInfo")

  val reqPoints = spark.sql("select x,y,z,count(*) as countVal from pickupInfo where x>=" + minX + " and x<=" + maxX + " and y>="+minY +" and y<="
  reqPoints.createOrReplaceTempView("reqPoints")
```

HotCellAnalysis.scala



```scala
package cse512

object HotzoneUtils {

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

    val rectangle_coordinates = queryRectangle.split(",")
    val target_point_coordinates = pointString.split(",")

    val point_x: Double = target_point_coordinates(0).trim.toDouble
    val point_y: Double = target_point_coordinates(1).trim.toDouble
    val rect_x1: Double = math.min(rectangle_coordinates(0).trim.toDouble, rectangle_coordinates(2).trim.toDouble)
    val rect_y1: Double = math.min(rectangle_coordinates(1).trim.toDouble, rectangle_coordinates(3).trim.toDouble)
    val rect_x2: Double = math.max(rectangle_coordinates(0).trim.toDouble, rectangle_coordinates(2).trim.toDouble)
    val rect_y2: Double = math.max(rectangle_coordinates(1).trim.toDouble, rectangle_coordinates(3).trim.toDouble)

    if ((point_x >= rect_x1) && (point_x <= rect_x2) && (point_y >= rect_y1) && (point_y <= rect_y2)) {
      return true
    }
    return false
  }
}
```

HotZoneUtils.scala



```scala
package cse512

import org.apache.log4j.{Level, Logger}
import org.apache.spark.sql.{DataFrame, SaveMode, SparkSession}

object HotzoneAnalysis {

  Logger.getLogger("org.spark_project").setLevel(Level.WARN)
  Logger.getLogger("org.apache").setLevel(Level.WARN)
  Logger.getLogger("akka").setLevel(Level.WARN)
  Logger.getLogger("com").setLevel(Level.WARN)

  def runHotZoneAnalysis(spark: SparkSession, pointPath: String, rectanglePath: String): DataFrame = {

    var pointDf = spark.read.format("com.databricks.spark.csv").option("delimiter",";").option("header","false").load(pointPath);
    pointDf.createOrReplaceTempView("point")

    // Parse point data formats
    spark.udf.register("trim",(string : String)=>(string.replace("(", "").replace(")", "")))
    pointDf = spark.sql("select trim(_c5) as _c5 from point")
    pointDf.createOrReplaceTempView("point")

    // Load rectangle data
    val rectangleDf = spark.read.format("com.databricks.spark.csv").option("delimiter","\t").option("header","false").load(rectanglePath);
    rectangleDf.createOrReplaceTempView("rectangle")

    // Join two datasets
    spark.udf.register("ST_Contains",(queryRectangle:String, pointString:String)=>(HotzoneUtils.ST_Contains(queryRectangle, pointString)))
    val joinDf = spark.sql("select rectangle._c0 as rectangle, point._c5 as point from rectangle,point where ST_Contains(rectangle._c0,point._c5)")
    joinDf.createOrReplaceTempView("joinResult")

    val retVal = spark.sql("select rectangle, COUNT(point) as count from joinResult group by rectangle order by rectangle").coalesce(1)
    return retVal
  }

}
```

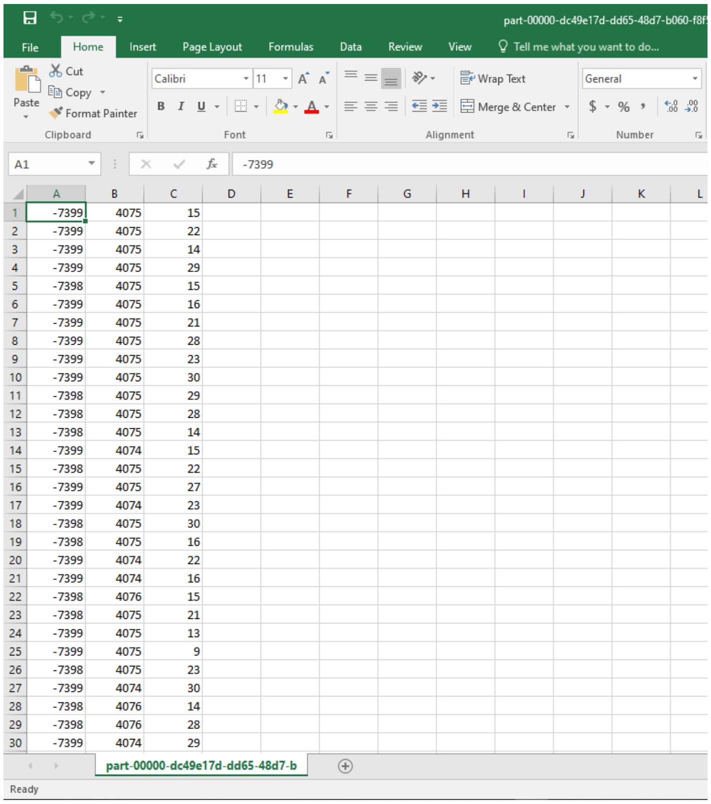
HotZoneAnalysis.scala

DATA PROCESSING AT SCALE - CSE 511

Submitted By: Aman Peshin (1225476655)

OUTPUT: HotCell and HotZone Analysis

Executed Output(HotCell)



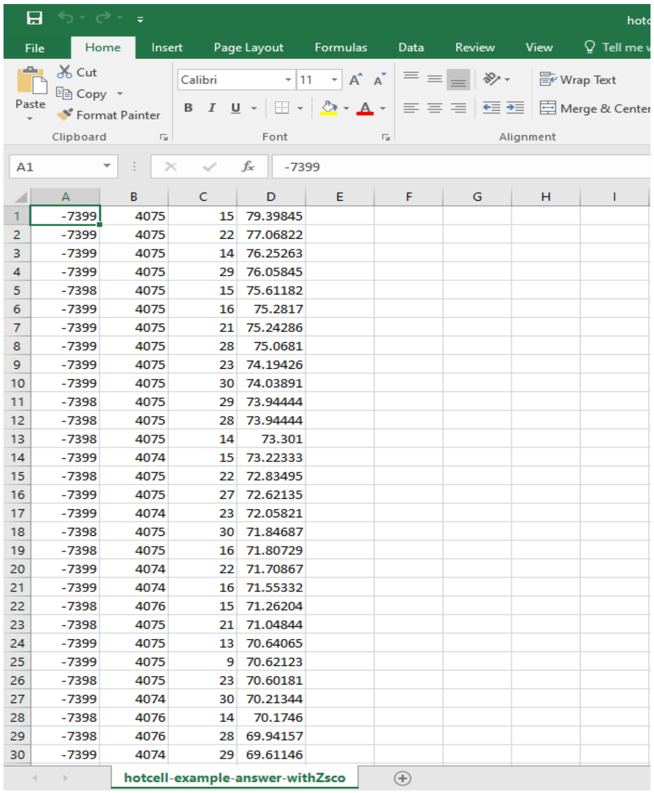
part-00000-dc49e17d-dd65-48d7-b060-f8f

A1 -7399

	A	B	C
1	-7399	4075	15
2	-7399	4075	22
3	-7399	4075	14
4	-7399	4075	29
5	-7398	4075	15
6	-7399	4075	16
7	-7399	4075	21
8	-7399	4075	28
9	-7399	4075	23
10	-7399	4075	30
11	-7398	4075	29
12	-7398	4075	28
13	-7398	4075	14
14	-7399	4074	15
15	-7398	4075	22
16	-7399	4075	27
17	-7399	4074	23
18	-7398	4075	30
19	-7398	4075	16
20	-7399	4074	22
21	-7399	4074	16
22	-7398	4076	15
23	-7398	4075	21
24	-7399	4075	13
25	-7399	4075	9
26	-7398	4075	23
27	-7399	4074	30
28	-7398	4076	14
29	-7398	4076	28
30	-7399	4074	29

part-00000-dc49e17d-dd65-48d7-b

Sample Output(HotCell)



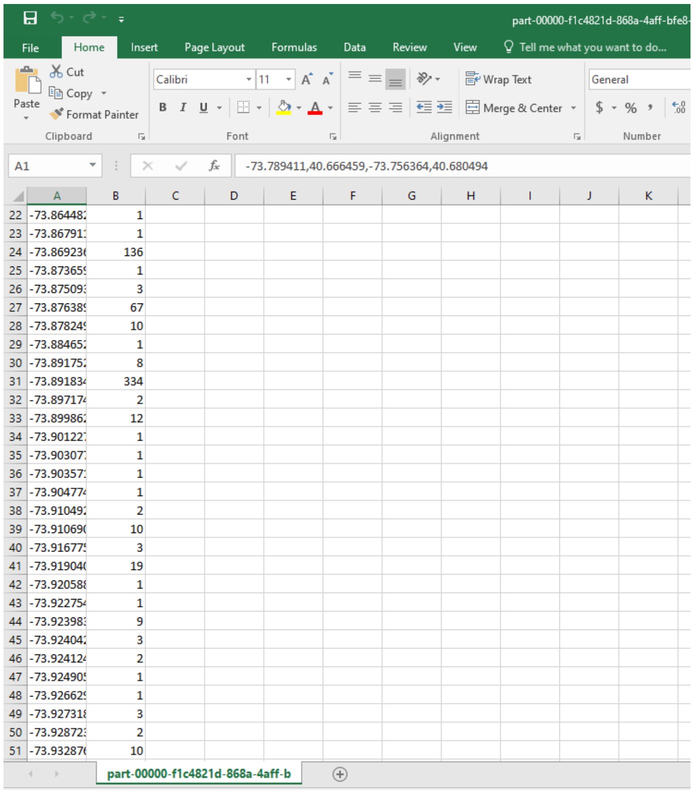
hotc

A1 -7399

	A	B	C	D
1	-7399	4075	15	79.39845
2	-7399	4075	22	77.06822
3	-7399	4075	14	76.25263
4	-7399	4075	29	76.05845
5	-7398	4075	15	75.61182
6	-7399	4075	16	75.2817
7	-7399	4075	21	75.24286
8	-7399	4075	28	75.0681
9	-7399	4075	23	74.19426
10	-7399	4075	30	74.03891
11	-7398	4075	29	73.94444
12	-7398	4075	28	73.94444
13	-7398	4075	14	73.301
14	-7399	4074	15	73.22333
15	-7398	4075	22	72.83495
16	-7399	4075	27	72.62135
17	-7399	4074	23	72.05821
18	-7398	4075	30	71.84687
19	-7398	4075	16	71.80729
20	-7399	4074	22	71.70867
21	-7399	4074	16	71.55332
22	-7398	4076	15	71.26204
23	-7398	4075	21	71.04844
24	-7399	4075	13	70.64065
25	-7399	4075	9	70.62123
26	-7398	4075	23	70.60181
27	-7399	4074	30	70.21344
28	-7398	4076	14	70.1746
29	-7398	4076	28	69.94157
30	-7399	4074	29	69.61146

hotcell-example-answer-withZsco

Executed Output(HotZone)



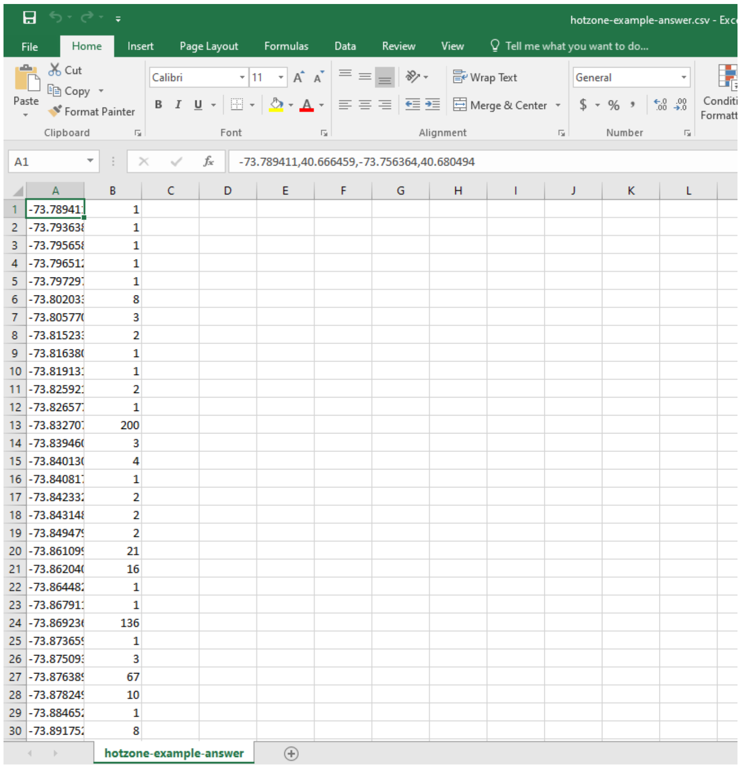
part-00000-f1c4821d-868a-4aff-bfe8

A1 -73.789411,40.666459,-73.756364,40.680494

	A	B
22	-73.864482	1
23	-73.867911	1
24	-73.869236	136
25	-73.873659	1
26	-73.875093	3
27	-73.876389	67
28	-73.878249	10
29	-73.884652	1
30	-73.891752	8
31	-73.891834	334
32	-73.897174	2
33	-73.899862	12
34	-73.901227	1
35	-73.903077	1
36	-73.903571	1
37	-73.904774	1
38	-73.910492	2
39	-73.910690	10
40	-73.916775	3
41	-73.919040	19
42	-73.920588	1
43	-73.922754	1
44	-73.923983	9
45	-73.924042	3
46	-73.924124	2
47	-73.924905	1
48	-73.926629	1
49	-73.927318	3
50	-73.928723	2
51	-73.932876	10

part-00000-f1c4821d-868a-4aff-b

Sample Output(HotZone)



hotzone-example-answer.csv - Exc

A1 -73.789411,40.666459,-73.756364,40.680494

	A	B
1	-73.789411	1
2	-73.793638	1
3	-73.795658	1
4	-73.796512	1
5	-73.797297	1
6	-73.802033	8
7	-73.805770	3
8	-73.815233	2
9	-73.816380	1
10	-73.819131	1
11	-73.825921	2
12	-73.826577	1
13	-73.832707	200
14	-73.839460	3
15	-73.840130	4
16	-73.840817	1
17	-73.842332	2
18	-73.843148	2
19	-73.849479	2
20	-73.861099	21
21	-73.862040	16
22	-73.864482	1
23	-73.867911	1
24	-73.869236	136
25	-73.873659	1
26	-73.875093	3
27	-73.876389	67
28	-73.878249	10
29	-73.884652	1
30	-73.891752	8

hotzone-example-answer