Project Report2 – Hot Spot Analysis

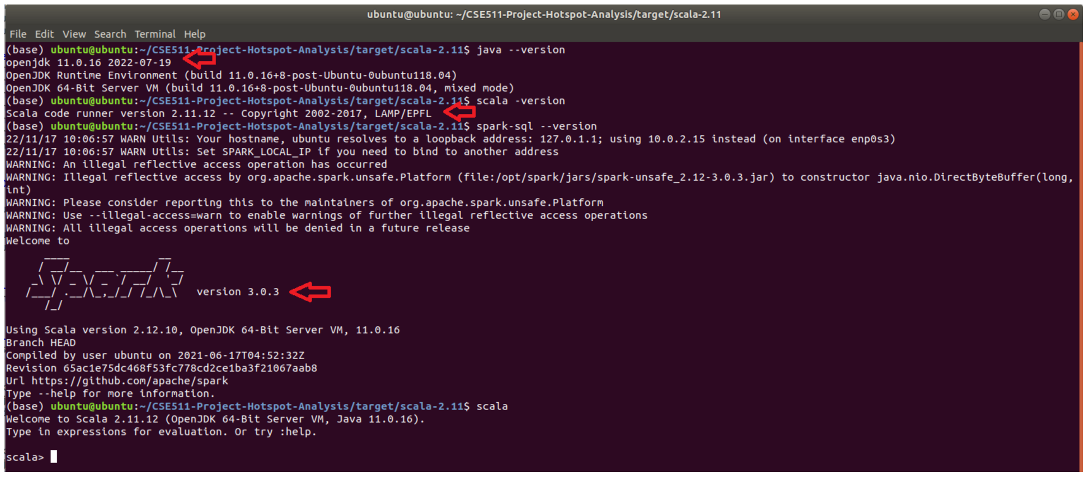
Student Name: Mehran Tajbakhsh
Email: mtajbakh@asu.edu
Submission Date: 30th Nov, 2022
Class Name and Term: CSE511 Fall 2022

Project Overview

In this project, I'm going to use the New York Cab taxi trip records as a geospatial input data and use various partitioning techniques to partition the input data into zones and cells and find the zones with the most points (hot zone) and find the hot cells using the Getis-Ord statistic (z-score).

I. Reflection

In this section I described the steps that I followed to prepare the VM and the functions that I implemented. I used a Linux Ubuntu 18.04 VM to install and use the technologies that I need to implement this project. I installed the software listed on the "Technology Requirements" section. I checked the version of the installed software to make sure that I installed the correct version and the software runs correctly (below image).

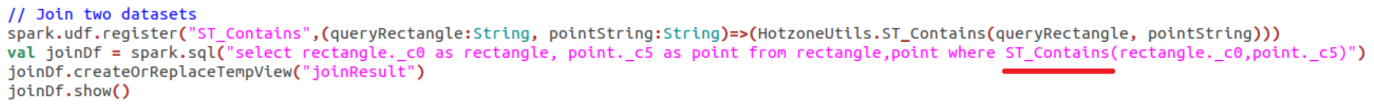

ubuntu@ubuntu: ~/CSE511-Project-Hotspot-Analysis/target/scala-2.11
File Edit View Search Terminal Help

```
(base) ubuntu@ubuntu:~/CSE511-Project-Hotspot-Analysis/target/scala-2.11$ java --version
openjdk 11.0.16 2022-07-19
OpenJDK Runtime Environment (build 11.0.16+8-post-Ubuntu-0ubuntu118.04)
OpenJDK 64-Bit Server VM (build 11.0.16+8-post-Ubuntu-0ubuntu118.04, mixed mode)
(base) ubuntu@ubuntu:~/CSE511-Project-Hotspot-Analysis/target/scala-2.11$ scala -version
Scala code runner version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
(base) ubuntu@ubuntu:~/CSE511-Project-Hotspot-Analysis/target/scala-2.11$ spark-sql --version
22/11/17 10:06:57 WARN Utils: Your hostname, ubuntu resolves to a loopback address: 127.0.1.1; using 10.0.2.15 instead (on interface enp0s3)
22/11/17 10:06:57 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3.0.3.jar) to constructor java.nio.DirectByteBuffer(long,
int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.3
      /_/

Using Scala version 2.12.10, OpenJDK 64-Bit Server VM, 11.0.16
Branch HEAD
Compiled by user ubuntu on 2021-06-17T04:52:32Z
Revision 65ac1e75dc468f53fc778cd2ce1ba3f21067aab8
Url https://github.com/apache/spark
Type --help for more information.
(base) ubuntu@ubuntu:~/CSE511-Project-Hotspot-Analysis/target/scala-2.11$ scala
Welcome to Scala 2.11.12 (OpenJDK 64-Bit Server VM, Java 11.0.16).
Type in expressions for evaluation. Or try :help.

scala> 
```

This project consists of two parts, Hot Zone/Cell Analysis. In the first part implemented the function ST_Contains to determine whether any of the given input nodes is located inside the predefined rectangles that partitioned geospatial dataset or not and used this function to run a query to count all the points in each zone (rectangles) and use this query to find the hottest zones. The function ST_Contains That I implemented is placed in the HotcellUtils.scala file:

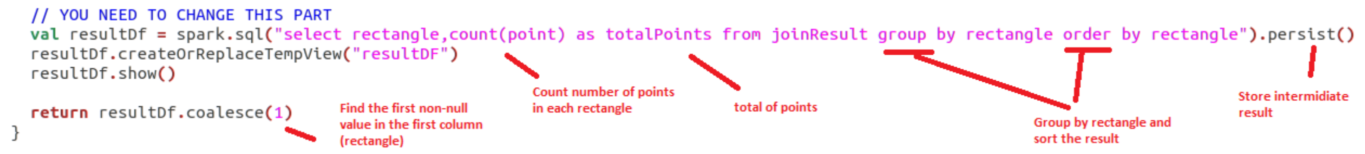
```
  // YOU NEED TO CHANGE THIS PART
  def ST_Contains(queryRectangle: String, pointString: String ): Boolean = {
    //Extract edges from the given rectangle
    var rectangle = queryRectangle.split(",");
    var longitude_1 = rectangle(0).toDouble
    var latitude_1 = rectangle(1).toDouble
    var longitude_2 = rectangle(2).toDouble
    var latitude_2 = rectangle(3).toDouble

    // Find Latitude and Logitude of the given points in the dataset
    var point = pointString.split(",");
    var longitude_point = point(0).toDouble
    var latitude_point = point(1).toDouble

    //Find Min and Max of the Latitude and Longitude in every rectangles
    var max_longitude = math.max(longitude_1, longitude_2)
    var min_longitude = math.min(longitude_1, longitude_2)
    var max_latitude = math.max(latitude_1, latitude_2)
    var min_latitude = math.min(latitude_1, latitude_2)

    //Check wether the given points in the dataset is located in the rectangle or not
    if (longitude_point > max_longitude || longitude_point < min_longitude || latitude_point > max_latitude || latitude_point < min_latitude) {
      false;
    }
    else {
      true;
    }
  }
}
```

I used the function ST_Contains to run the following query to create a temporary view (joinResult) by joining two datasets (rectangle, point) and specify either each point in located in the specified rectangle or not.



```
// 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")
joinDf.show()
```

In the following image I show the implemented query to find the hot zones. In this query I use the joinResult as input dataset and group the points based on the rectangles and then I count the total numbers of point in each rectangles. When I sort the result in the descending order, the top rows give me the hottest zones.



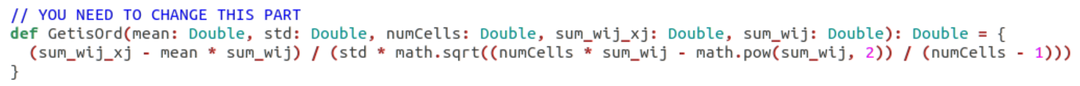
```
  // YOU NEED TO CHANGE THIS PART
  val resultDf = spark.sql("select rectangle,count(point) as totalPoints from joinResult group by rectangle order by rectangle").persist()
  resultDf.createOrReplaceTempView("resultDF")
  resultDf.show()

  return resultDf.coalesce(1)
}
```

Find the first non-null value in the first column (rectangle)
Count number of points in each rectangle
total of points
Group by rectangle and sort the result
Store intermidiate result

In the second part I need to compute Getis-Ord G*$_I$ statistic for each cell and find the hot cells. In order to do this I need to compute the following formula for each cell:

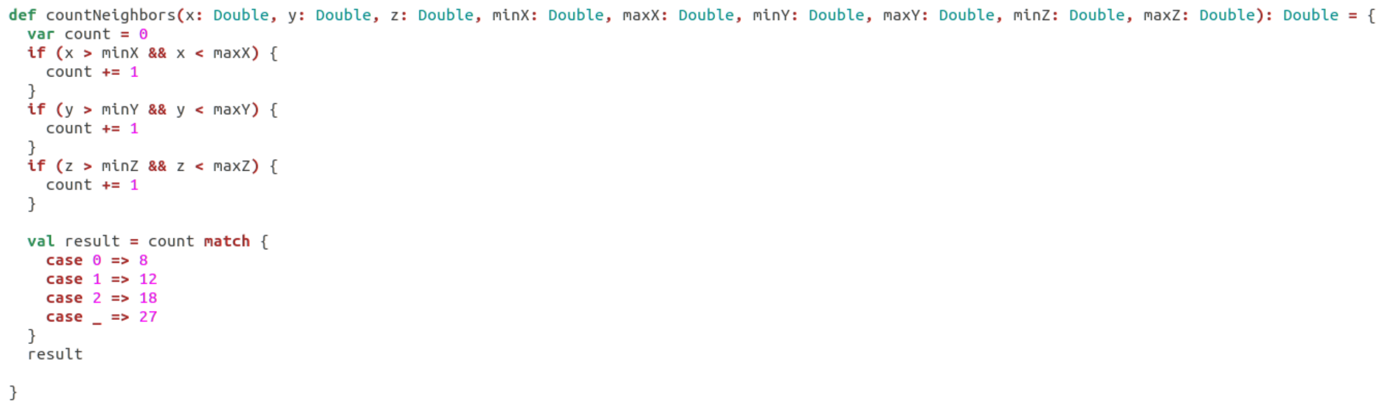
$$G_i^* = \frac{\sum_{j=1}^{n} w_{i,j} x_j - \bar{X} \sum_{j=1}^{n} w_{i,j}}{S \sqrt{\frac{\left[n \sum_{j=1}^{n} w_{i,j}^2 - \left(\sum_{j=1}^{n} w_{i,j} \right)^2 \right]}{n-1}}}$$

In the following image I implemented the below function (GetisOrd) to compute the above formula:



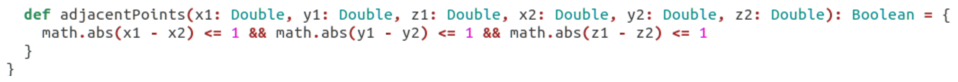
```
// YOU NEED TO CHANGE THIS PART
def GetisOrd(mean: Double, std: Double, numCells: Double, sum_wij_xj: Double, sum_wij: Double): Double = {
  (sum_wij_xj - mean * sum_wij) / (std * math.sqrt((numCells * sum_wij - math.pow(sum_wij, 2)) / (numCells - 1)))
}
```

In the above code the variable wij is the spatial weight between cell i and j. In order to compute this variable I need to find the total number of the neighbors for each cell. I wrote the function countNeighbors to find the total number of neighbors for each cell:



```
def countNeighbors(x: Double, y: Double, z: Double, minX: Double, maxX: Double, minY: Double, maxY: Double, minZ: Double, maxZ: Double): Double = {
  var count = 0
  if (x > minX && x < maxX) {
    count += 1
  }
  if (y > minY && y < maxY) {
    count += 1
  }
  if (z > minZ && z < maxZ) {
    count += 1
  }

  val result = count match {
    case 0 => 8
    case 1 => 12
    case 2 => 18
    case _ => 27
  }
  result

}
```

The function adjacentPoints checks if two given points are placed in the same cell or not:



```
  def adjacentPoints(x1: Double, y1: Double, z1: Double, x2: Double, y2: Double, z2: Double): Boolean = {
    math.abs(x1 - x2) <= 1 && math.abs(y1 - y2) <= 1 && math.abs(z1 - z2) <= 1
  }
}
```

II. LESSON LEARNED

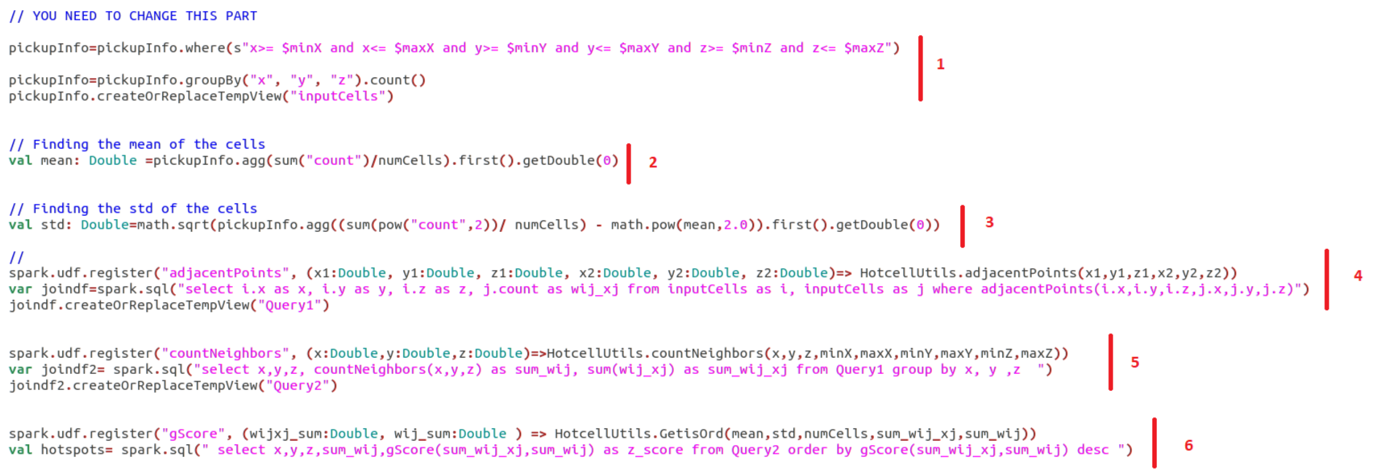
In this project I did Hot Spot Spatial Analysis on New York Cab taxi trip records. During this project I learned the following concepts, tools, and techniques:

- How to setup a framework using Apache Spark and Scala and using it to process on geospatial data.
- What is the difference between the Hot Zone/Cell Analysis?
- What is Getis-Ord G*i statistics and how to calculate it and use it as a metric to find hot cells.
- How to build a space-time cube based on the latitude and longitude attributes of the spatial data and find the neighborhood of each cell.

III. IMPLEMENTATION

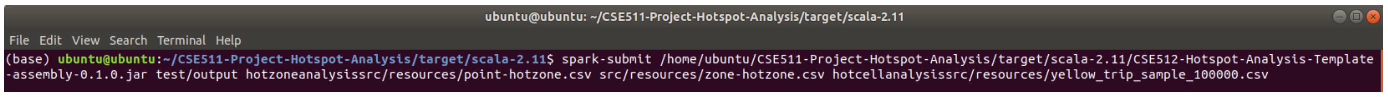
In order to calculate the z-score I did the following steps:

1- Based on the minimum and maximum value of x, y, and z that is given to me I extracted the cells and created a temporary dataset (inputCells).
2- I computed the average (mean) of the points ($\overline{X}$) in each cells to compute G* statistic (z-score).
3- I calculated the Standard Deviation of the points (S) in each cells to compute G* statistic (z-score).
4- I used a query to find the adjacent points and compute the weights between cells, I used the function adjacentPoints in this query, and saved the results in the temporary dataset (Query1)
5- I used a query to calculate the sum of the weights between cells, I used the function countNeighbors in this query, and I saved the results in the temporary dataset (Query2).
6- Finally, I used the data that I calculated in the Query2 to compute the z-score and sorted the result on the descending order based on the z-score. I used the function GetisOrd that I registered as gScore in the Scala to compute the z-score in this query.



```
// YOU NEED TO CHANGE THIS PART

pickupInfo=pickupInfo.where(s"x>= $minX and x<= $maxX and y>= $minY and y<= $maxY and z>= $minZ and z<= $maxZ")

pickupInfo=pickupInfo.groupBy("x", "y", "z").count()
pickupInfo.createOrReplaceTempView("inputCells")


// Finding the mean of the cells
val mean: Double =pickupInfo.agg(sum("count")/numCells).first().getDouble(0)


// Finding the std of the cells
val std: Double=math.sqrt(pickupInfo.agg((sum(pow("count",2))/ numCells) - math.pow(mean,2.0)).first().getDouble(0))

//
spark.udf.register("adjacentPoints", (x1:Double, y1:Double, z1:Double, x2:Double, y2:Double, z2:Double)=> HotcellUtils.adjacentPoints(x1,y1,z1,x2,y2,z2))
var joindf=spark.sql("select i.x as x, i.y as y, i.z as z, j.count as wij_xj from inputCells as i, inputCells as j where adjacentPoints(i.x,i.y,i.z,j.x,j.y,j.z)")
joindf.createOrReplaceTempView("Query1")


spark.udf.register("countNeighbors", (x:Double,y:Double,z:Double)=>HotcellUtils.countNeighbors(x,y,z,minX,maxX,minY,maxY,minZ,maxZ))
var joindf2= spark.sql("select x,y,z, countNeighbors(x,y,z) as sum_wij, sum(wij_xj) as sum_wij_xj from Query1 group by x, y ,z  ")
joindf2.createOrReplaceTempView("Query2")


spark.udf.register("gScore", (wijxj_sum:Double, wij_sum:Double ) => HotcellUtils.GetisOrd(mean,std,numCells,sum_wij_xj,sum_wij))
val hotspots= spark.sql(" select x,y,z,sum_wij,gScore(sum_wij_xj,sum_wij) as z_score from Query2 order by gScore(sum_wij_xj,sum_wij) desc ")
```

To submit the code in the spark I used the following command:


ubuntu@ubuntu: ~/CSE511-Project-Hotspot-Analysis/target/scala-2.11
File Edit View Search Terminal Help

```
(base) ubuntu@ubuntu:~/CSE511-Project-Hotspot-Analysis/target/scala-2.11$ spark-submit /home/ubuntu/CSE511-Project-Hotspot-Analysis/target/scala-2.11/CSE512-Hotspot-Analysis-Template
-assembly-0.1.0.jar test/output hotzoneanalysissrc/resources/point-hotzone.csv src/resources/zone-hotzone.csv hotcellanalysissrc/resources/yellow_trip_sample_100000.csv
```

ANALYSIS

In this project I learned how to process geospatial data using SQL and Scala in the Apache Spark environment. Finding most important zones and cells (hot zones/cells) is the biggest goal in the processing of the geographic information. Based on the results we can create/change a plan to improve our services to the customers. We used G* statistic to find statistically significant locations (hot cells). This metric help us to determine where attributes with either high or low values are grouped spatially.

The higher G-score (z-score) for a specific cell shows that cell contains more points than average points in the other cells.

TECHNOLOGY REQUIREMENTS

– Apache Spark
– SparkSQL
– Scala
– Java
– Hadoop

REFERENCES

1. http://sigspatial2016.sigspatial.org/giscup2016/problem - ACM SIGSPATIAL GISCUP 2016