Project 2 Report-Hot Spot Analysis

Name: Nigar Sultana

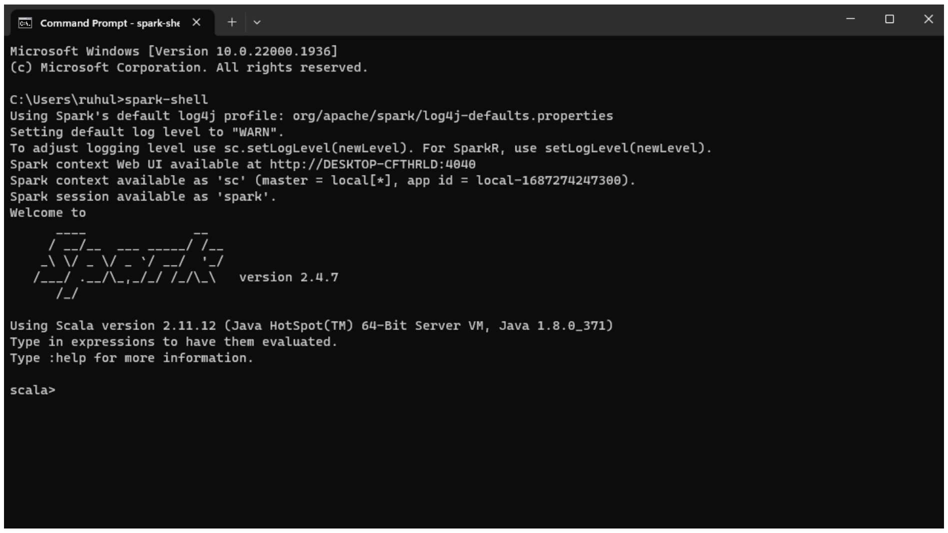
ID:1221371489

Project Overview

The partitioning process involves dividing the given dataset into distinct zones and cells, aiming to identify the zone that contains the highest concentration of data points. This project uses the Getis-Ortd statistic (z-score) to find the hot cells. The New York taxicab trip record is used as geospatial input data for this project.

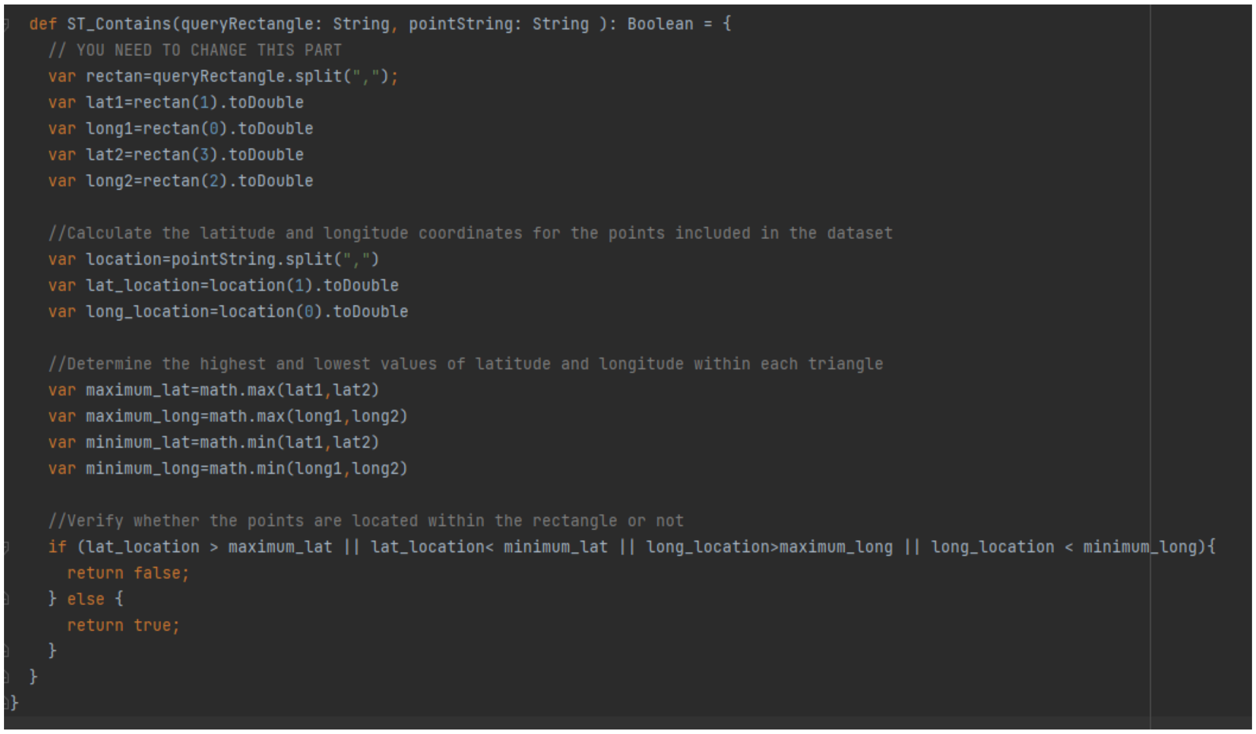
1.Reflection:

This project requires some specialized tools to process data in the Apache Spark environment which are specified below in the Technology requirements section. The appropriate installation of the correct version of these tools is required to ensure the smooth operation of the required software, as demonstrated in the screenshot below.



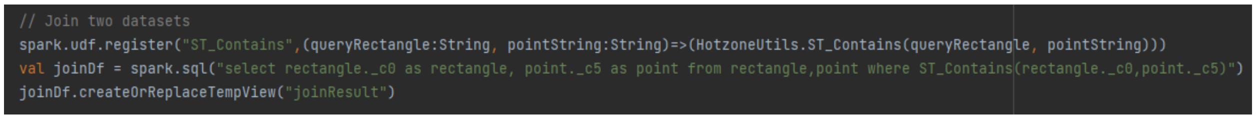
```
Microsoft Windows [Version 10.0.22000.1936]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ruhul>spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://DESKTOP-CFTHRLD:4040
Spark context available as 'sc' (master = local[*], app id = local-1687274247300).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.4.7
      /_/

Using Scala version 2.11.12 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_371)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

The first part of the project is the Hot Zone Analysis, in which the ST_Contains function is used to find the location of the given input nodes in the rectangles previously defined. The function executes a query for counting all the points falls in each rectangle (zone). To identify the hottest zone the above query is used. Below is the ST_Contains function that is implemented in the HotzoneUtils.scala



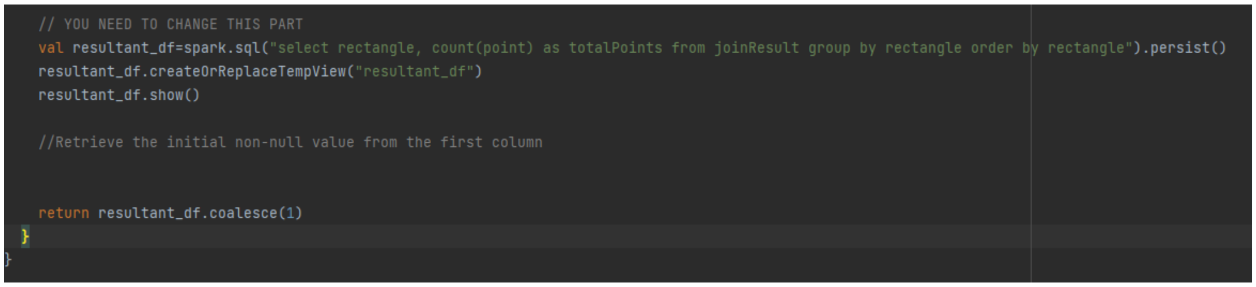
```scala
def ST_Contains(queryRectangle: String, pointString: String ): Boolean = {
  // YOU NEED TO CHANGE THIS PART
  var rectan=queryRectangle.split(",");
  var lat1=rectan(1).toDouble
  var long1=rectan(0).toDouble
  var lat2=rectan(3).toDouble
  var long2=rectan(2).toDouble

  //Calculate the latitude and longitude coordinates for the points included in the dataset
  var location=pointString.split(",")
  var lat_location=location(1).toDouble
  var long_location=location(0).toDouble

  //Determine the highest and lowest values of latitude and longitude within each triangle
  var maximum_lat=math.max(lat1,lat2)
  var maximum_long=math.max(long1,long2)
  var minimum_lat=math.min(lat1,lat2)
  var minimum_long=math.min(long1,long2)

  //Verify whether the points are located within the rectangle or not
  if (lat_location > maximum_lat || lat_location< minimum_lat || long_location>maximum_long || long_location < minimum_long){
    return false;
  } else {
    return true;
  }
}
}
```

The ST_Contains function is used to execute the request to generate a temporary view through merging of two datasets (point, rectangle) together. The purpose was to determine whether expected points are located within the specified rectangle.



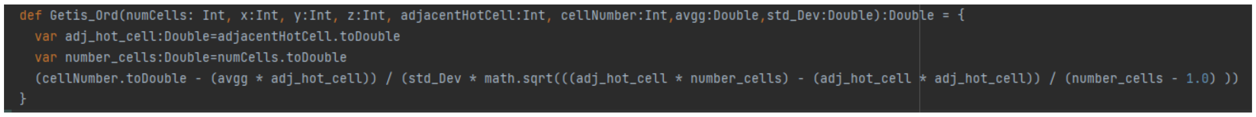
```scala
// Join two datasets
spark.udf.register("ST_Contains",(queryRectangle:String, pointString:String)=>(HotzoneUtils.ST_Contains(queryRectangle, pointString)))
val joinDf = spark.sql("select rectangle._c0 as rectangle, point._c5 as point from rectangle,point where ST_Contains(rectangle._c0,point._c5)")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

The "joinResult" is utilized as the input dataset in this query, allowing for the grouping of points based on rectangles. The count of total points within each rectangle is then determined. Upon sorting the results in descending order, the hottest zones are revealed in the top rows. The implementation of this query is in the image below:



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

  //Retrieve the initial non-null value from the first column


  return resultant_df.coalesce(1)
}
}
```

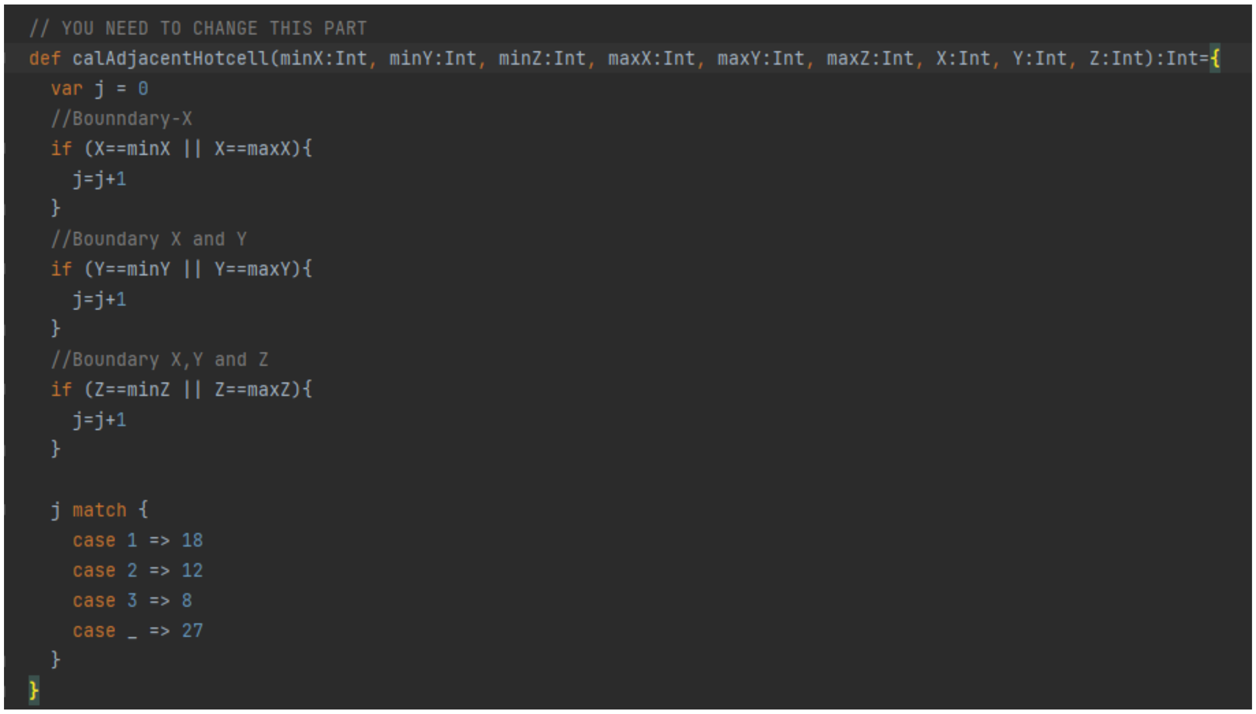
In the second part of this project the following G*t formula is used to find the hot cells. To compute the G*t function the Getis_Ord function is executed which is shown below:



```scala
def Getis_Ord(numCells: Int, x:Int, y:Int, z:Int, adjacentHotCell:Int, cellNumber:Int,avgg:Double,std_Dev:Double):Double = {
  var adj_hot_cell:Double=adjacentHotCell.toDouble
  var number_cells:Double=numCells.toDouble
  (cellNumber.toDouble - (avgg * adj_hot_cell)) / (std_Dev * math.sqrt(((adj_hot_cell * number_cells) - (adj_hot_cell * adj_hot_cell)) / (number_cells - 1.0) ))
}
```

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

The function calAdjacentHotcell is employed to determine the count of neighboring cells for each individual cell. Total number of neighbors are needed for every cell to compute variable Wij which represents the spatial weight between i and j .



```
// YOU NEED TO CHANGE THIS PART
def calAdjacentHotcell(minX:Int, minY:Int, minZ:Int, maxX:Int, maxY:Int, maxZ:Int, X:Int, Y:Int, Z:Int):Int={
  var j = 0
  //Bounndary-X
  if (X==minX || X==maxX){
    j=j+1
  }
  //Boundary X and Y
  if (Y==minY || Y==maxY){
    j=j+1
  }
  //Boundary X,Y and Z
  if (Z==minZ || Z==maxZ){
    j=j+1
  }

  j match {
    case 1 => 18
    case 2 => 12
    case 3 => 8
    case _ => 27
  }
}
```

2. Lessons Learned:

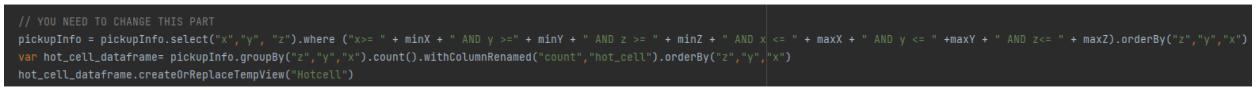
The lesson learned from this is as follows:

- Proper utilization of relevant tools and technologies.
- Understanding and effectively implementing function like ST_Contains for data processing and validation.
- Conducting thorough data analysis to extract patterns.
- Leveraging suitable tools, functions, and techniques to ensure accurate and valuable results in geospatial data analysis.

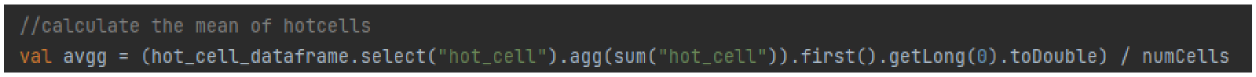
3. Implementation:

The following steps were undertaken for the z-score calculation:

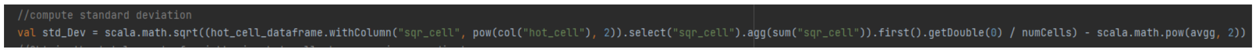
1. Extraction of cells and creation of a temporary dataset(inputCells) based on the provided maximum and minimum values of x, y, and z.
2. Computation of the average (mean) of the points (X) in each cell to determine the G* statistic (z-score).
3. Calculation of the standard deviation (S) of the points in each cell to determine the G* statistic (z-score)
4. Executing a request to identify neighboring points and calculate the cells weights, which were stored in the 'joinResult' variable.
5. Executing a query utilizing the "calAdjacentHotcell" function to determine the total weight sum between cells, which was saved in the Query2 variable.
6. Data from Query2 was utilized to calculating the z-score using Getis_Ord method and storing the result in the z-score



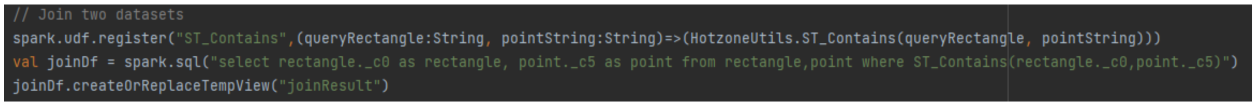
```
// YOU NEED TO CHANGE THIS PART
pickupInfo = pickupInfo.select("x","y", "z").where ("x>= " + minX + " AND y >=" + minY + " AND z >= " + minZ + " AND x <= " + maxX + " AND y <= " +maxY + " AND z<= " + maxZ).orderBy("z","y","x")
var hot_cell_dataframe= pickupInfo.groupBy("z","y","x").count().withColumnRenamed("count","hot_cell").orderBy("z","y","x")
hot_cell_dataframe.createOrReplaceTempView("Hotcell")
```



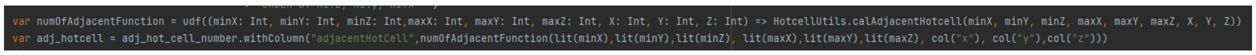
```
//calculate the mean of hotcells
val avgg = (hot_cell_dataframe.select("hot_cell").agg(sum("hot_cell")).first().getLong(0).toDouble) / numCells
```



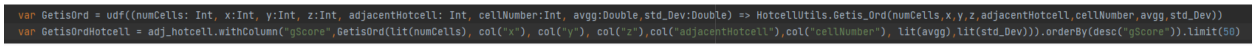
```
//compute standard deviation
val std_Dev = scala.math.sqrt((hot_cell_dataframe.withColumn("sqr_cell", pow(col("hot_cell"), 2)).select("sqr_cell").agg(sum("sqr_cell")).first().getDouble(0) / numCells) - scala.math.pow(avgg, 2))
```



```
// 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")
```

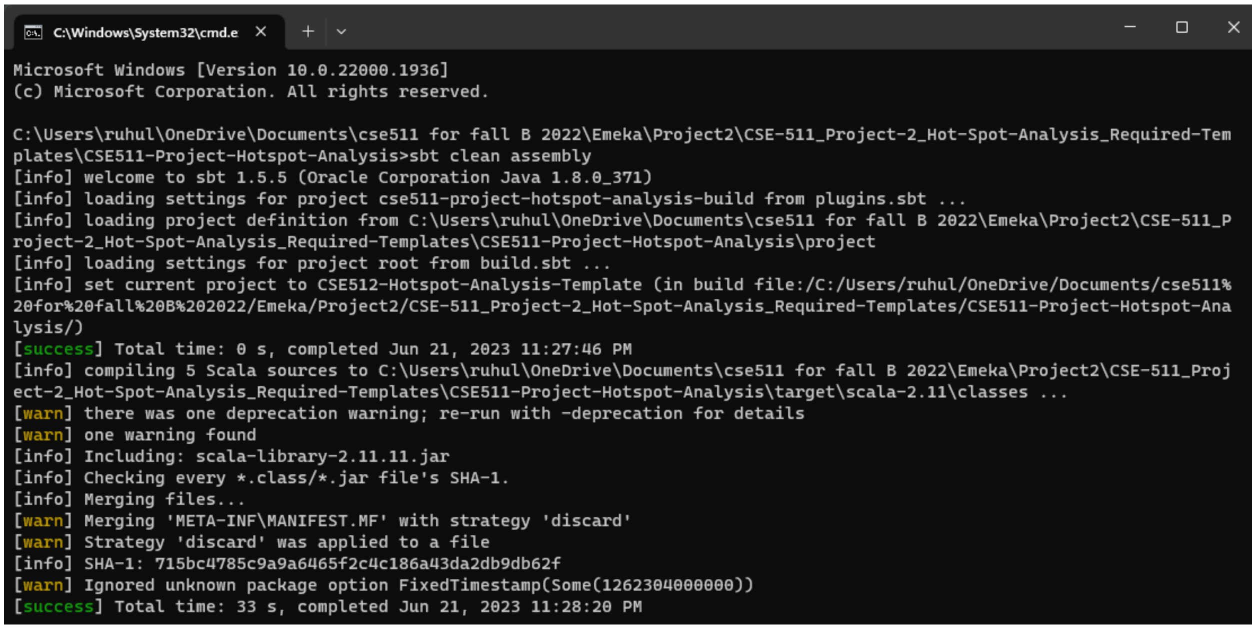


```
var numOfAdjacentFunction = udf((minX: Int, minY: Int, minZ: Int,maxX: Int, maxY: Int, maxZ: Int, X: Int, Y: Int, Z: Int) => HotcellUtils.calAdjacentHotcell(minX, minY, minZ, maxX, maxY, maxZ, X, Y, Z))
var adj_hotcell = adj_hot_cell_number.withColumn("adjacentHotCell",numOfAdjacentFunction(lit(minX),lit(minY),lit(minZ), lit(maxX),lit(maxY),lit(maxZ), col("x"), col("y"),col("z")))
```

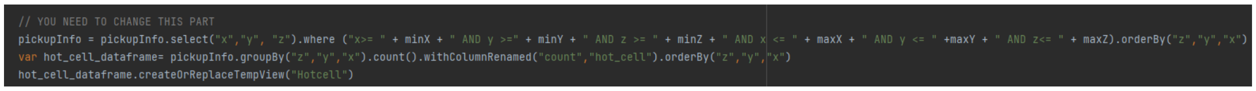
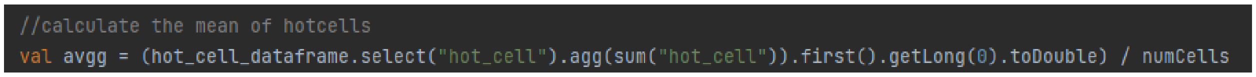
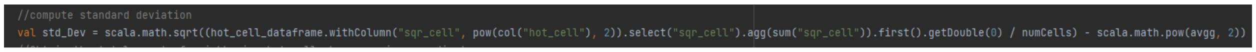
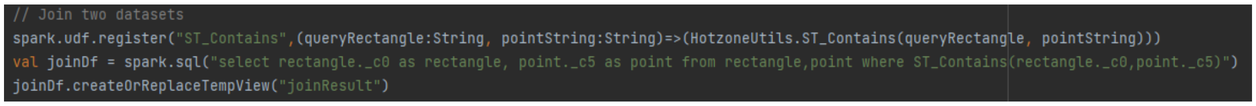
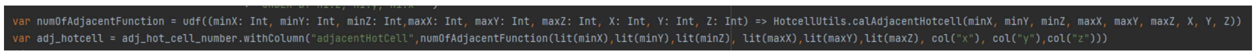
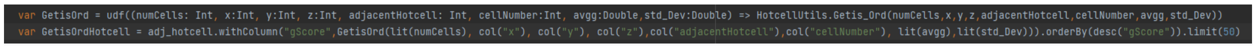
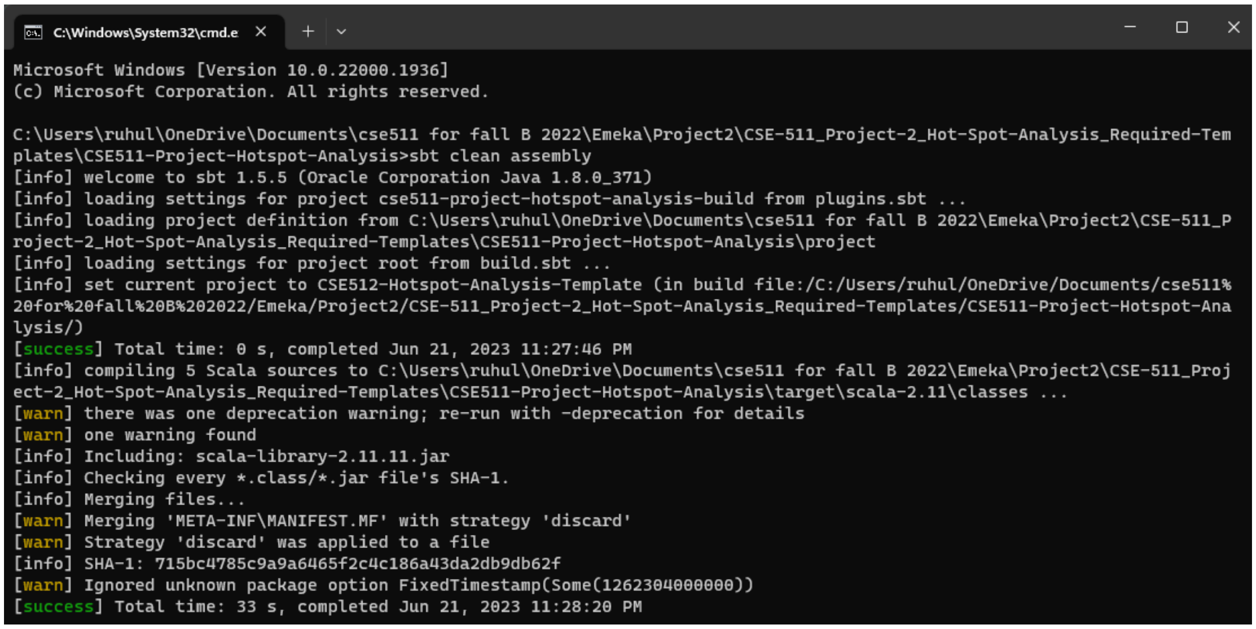


```
var GetisOrd = udf((numCells: Int, x:Int, y:Int, z:Int, adjacentHotcell: Int, cellNumber:Int, avgg:Double,std_Dev:Double) => HotcellUtils.Getis_Ord(numCells,x,y,z,adjacentHotcell,cellNumber,avgg,std_Dev))
var GetisOrdHotcell = adj_hotcell.withColumn("gScore",GetisOrd(lit(numCells), col("x"), col("y"), col("z"),col("adjacentHotcell"),col("cellNumber"), lit(avgg),lit(std_Dev))).orderBy(desc("gScore")).limit(50)
```

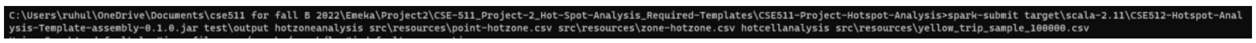
In the project root folder, I used "sbt clean assembly" and found it ran successfully.



```
Microsoft Windows [Version 10.0.22000.1936]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ruhul\OneDrive\Documents\cse511 for fall B 2022\Emeka\Project2\CSE-511_Project-2_Hot-Spot-Analysis_Required-Tem
plates\CSE511-Project-Hotspot-Analysis>sbt clean assembly
[info] welcome to sbt 1.5.5 (Oracle Corporation Java 1.8.0_371)
[info] loading settings for project cse511-project-hotspot-analysis-build from plugins.sbt ...
[info] loading project definition from C:\Users\ruhul\OneDrive\Documents\cse511 for fall B 2022\Emeka\Project2\CSE-511_P
roject-2_Hot-Spot-Analysis_Required-Templates\CSE511-Project-Hotspot-Analysis\project
[info] loading settings for project root from build.sbt ...
[info] set current project to CSE512-Hotspot-Analysis-Template (in build file:/C:/Users/ruhul/OneDrive/Documents/cse511%
20for%20fall%20B%202022/Emeka/Project2/CSE-511_Project-2_Hot-Spot-Analysis_Required-Templates/CSE511-Project-Hotspot-Ana
lysis/)
[success] Total time: 0 s, completed Jun 21, 2023 11:27:46 PM
[info] compiling 5 Scala sources to C:\Users\ruhul\OneDrive\Documents\cse511 for fall B 2022\Emeka\Project2\CSE-511_Proj
ect-2_Hot-Spot-Analysis_Required-Templates\CSE511-Project-Hotspot-Analysis\target\scala-2.11\classes ...
[warn] there was one deprecation warning; re-run with -deprecation for details
[warn] one warning found
[info] Including: scala-library-2.11.11.jar
[info] Checking every *.class/*.jar file's SHA-1.
[info] Merging files...
[warn] Merging 'META-INF\MANIFEST.MF' with strategy 'discard'
[warn] Strategy 'discard' was applied to a file
[info] SHA-1: 715bc4785c9a9a6465f2c4c186a43da2db9db62f
[warn] Ignored unknown package option FixedTimestamp(Some(1262304000000))
[success] Total time: 33 s, completed Jun 21, 2023 11:28:20 PM
```

The following command is to submit the code in Spark:



```
C:\Users\ruhul\OneDrive\Documents\cse511 for fall B 2022\Emeka\Project2\CSE-511_Project-2_Hot-Spot-Analysis_Required-Templates\CSE511-Project-Hotspot-Analysis>spark-submit target\scala-2.11\CSE512-Hotspot-Anal
ysis-Template-assembly-0.1.0.jar test\output hotzoneanalysis src\resources\point-hotzone.csv src\resources\zone-hotzone.csv hotcellanalysis src\resources\yellow_trip_sample_100000.csv
```

Analysis

By completing it was understood how to process geospatial data using SQL and Scala in the Apache Spark environment, with the objective being identification of significant zones and cells known as hot zones and cells. Based on the results, a plan can be created or modified to improve services to customers. The G* statistic was utilized to find the location significant statistically. Metric aids in identifying spatial clusters of attributes with high or low values, where a higher G-score assigned to a particular cell signifies a grater concentration of points compared to the average number of points in other cells.

Tools used for this project:

- Apache Spark
- Spark SQL
- Scala
- Java
- Hadoop

References:

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