# Arizona State University – MCS

## CSE 511-DATA PROCESSING AT SCALE

# TEAM 6 - SYSTEM DOCUMENTATION REPORT

**28th April 2019**

**Team 6 Members:**

- Kathleen Suarez
- Neelamani Gopalakrishnan
- Ioana Tiriac
- Madhusudanan Sivasankaran

# INTRODUCTION

## Background:

We were given a collection of New York City Yellow Cab taxi trip records which spanned from January 2009 to June 2015. This contained geographic data and real-time location data of their customers. We were provided with a coding template in SparkSQL. SparkSQL is Big Data software application which provides functionality to run spatial queries. The coding template has already taken care of loading the data. Our team was tasked with implementing spatial queries and analysis on the data using SparkSQL.

## Team Goals / Objectives

For Phase1:

1) Write user defined functions ST_Contains and ST_Within using SparkSQL
2) Use user defined functions from 1) in the following spatial queries: range query, range join query, distance query and distance join query.

For Phase2:

1) Design and implement a hot zone analysis which includes performing a range join operation on rectangle datasets and a point dataset in order to calculate the hotness of rectangle.
2) Design and implement a hot cell analysis which includes translating spatial statistics problem into a workable algorithm and identifying any assumptions that can help in solving the analysis.

As a team, with members of team in different time zones, distribute the workload, work out any communication needs and support each other to meet Phase1 and Phase2 objectives.

Learn about and gain experience with spatial queries and SparkSQL including translating the spatial solutions into spatial queries/code and having a working local environment to run and debug the code on Apache Spark and submit the code to Spark.

## User Stories

| # | User Story Title | User Story Description | Priority |
|---|---|---|---|
| 1 | Range Query | For the given geographical boundary in a city (represented by Rectangle) and a set of points P (customer pickup point) calculate all points within the rectangle. | Must Have |

| | | | |
|---|---|---|---|
| 2 | Range Join Query | For the given rectangles set R and a set of points P (customer pickup point), calculate all (point, rectangle) pairs where the point is within the rectangle. | Must Have |
| 3 | Distance Query | For the given combination of fixed-point location P and distance D, calculate all points that are present within a distance D from the location. | Must Have |
| 4 | Distance Join Query | For the given two sets of points P1 and P2, and a distance D, calculate all (P1, P2) pairs where P1 is within a distance D from P2. | Must Have |
| 5 | Hot Zone Analysis | For the given rectangle dataset R and a point dataset P, calculate no of points within the Rectangle dataset R in order to find the hotness of the Rectangle R. i.e., classify a hot zone based on total points each Rectangle contains. | Must Have |
| 6 | Hot Cell Analysis Algorithm Design | Translate the spatial statistics problem into an an algorithm to calculate Getis-Ord for each pickup point in the dataset. | Must Have |
| 7 | Hot Cell Analysis Pickup Points Aggregation | Find the total count of pickups for each cell within the space-time cube. | Must Have |
| 8 | Hot Cell Analysis WijXj Aggregation | Find the total count of pickups for each cell's neighborhood within the space-time cube. | Must Have |
| 9 | Hot Cell Analysis Getis Ord Calculation | Find the Getis-ord statistic for each cell in the space-time cube and return top 50 cells. | Must Have |
| 10 | Hot Cell Analysis Edge Neighborhood | Account for neighborhood at the edge of time-space cube which could have less than 27 neighbors. | Should Have |

## DESIGN

The entire approach works on Apache Spark framework with code development using Scala programming language.

### Work Products

We divided the work between us and tried to produce the output and the combined effort helped to produce the deliverables as a team.

- Analysis & Research: Everyone
- Hot Zone: Neelamani
- Hot Cell: Kathleen
- ST_Contains / ST_Within: Neelamani
- Documentation: Everyone

**Tools**:

1. Java 1.8 (jdk1.8.0_202)
2. Scala - scala-sdk-2.11.11
3. Apache Spark 2.4.1
4. Hadoop 2.7
5. SBT 1.2.8
6. IntelliJ IDEA (2019.1)

Installation reference : https://medium.com/@dvainrub/how-to-install-apache-spark-2-x-in-your-pc-e2047246ffc3

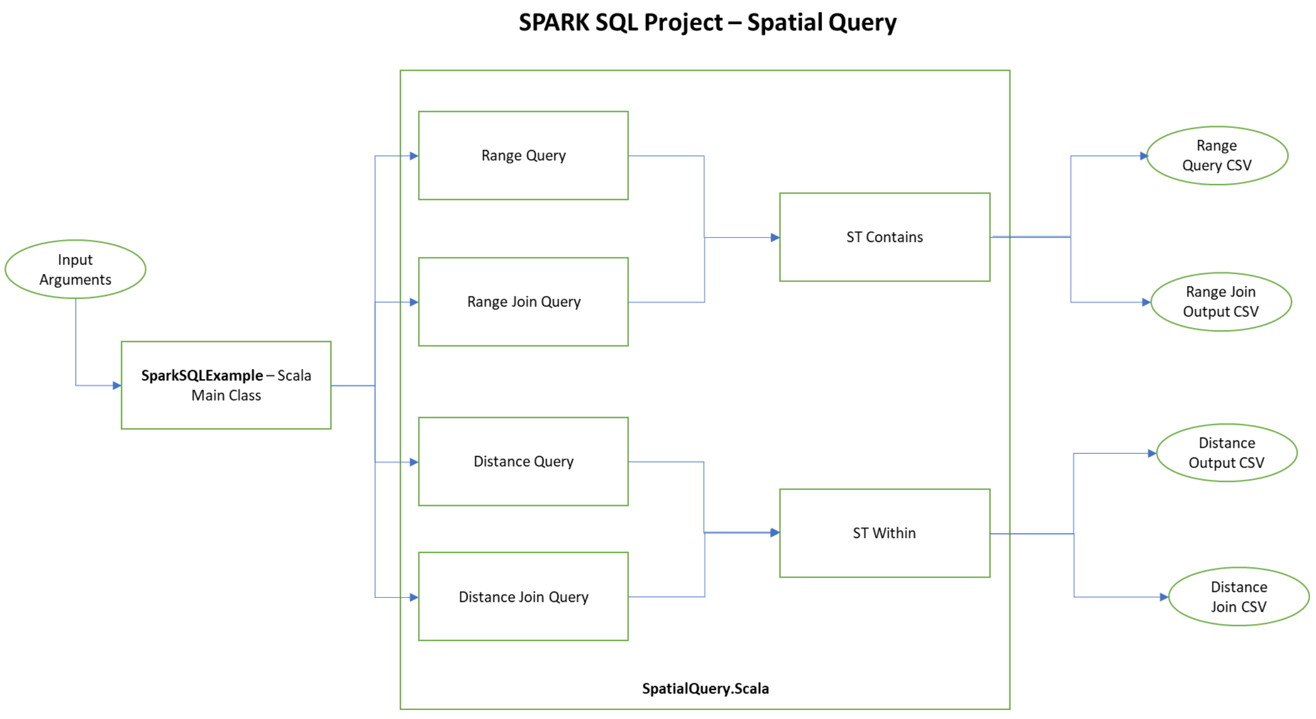
### USER STORY - SOLUTION TRACEABILITY MATRIX

| No # | User Story Title | File Modified | Function/Module |
|---|---|---|---|
| **1** | Range Query | SpatialQuery.scala | ST_Contains |
| **2** | Range Join | SpatialQuery.scala | ST_Contains |
| **3** | Distance Query | SpatialQuery.scala | ST_Within |
| **4** | Distance Join Query | SpatialQuery.scala | ST_Within |
| **5** | Hot Zone Analysis | HotzoneAnalysis.scala, HotzoneUtils.scala | HotzoneUtils.ST Contains |

| | | | |
|---|---|---|---|
| **6** | Hot Cell Analysis Algorithm Design | NA | NA |
| **7** | Hot Cell Analysis Pickup Points Aggregation | HotcellAnalysis.scala | HotcellAnalysis.runHotcellAnalysis |
| **8** | Hot Cell Analysis WijXj Aggregation | HotcellAnalysis.scala | HotcellAnalysis.runHotcellAnalysis |
| **9** | Hot Cell Analysis Getis Ord Calculation | HotcellAnalysis.scala | HotcellAnalysis.runHotcellAnalysis HotcellAnalysis.calcMean HotcellAnalysis.calcStdDev HotcellAnalysis.calcGetisOrd |
| **10** | Hot Cell Analysis Edge Neighborhood | HotcellAnalysis.scala | HotcellAnalysis.calcNeigborsCount (Not Implemented) |

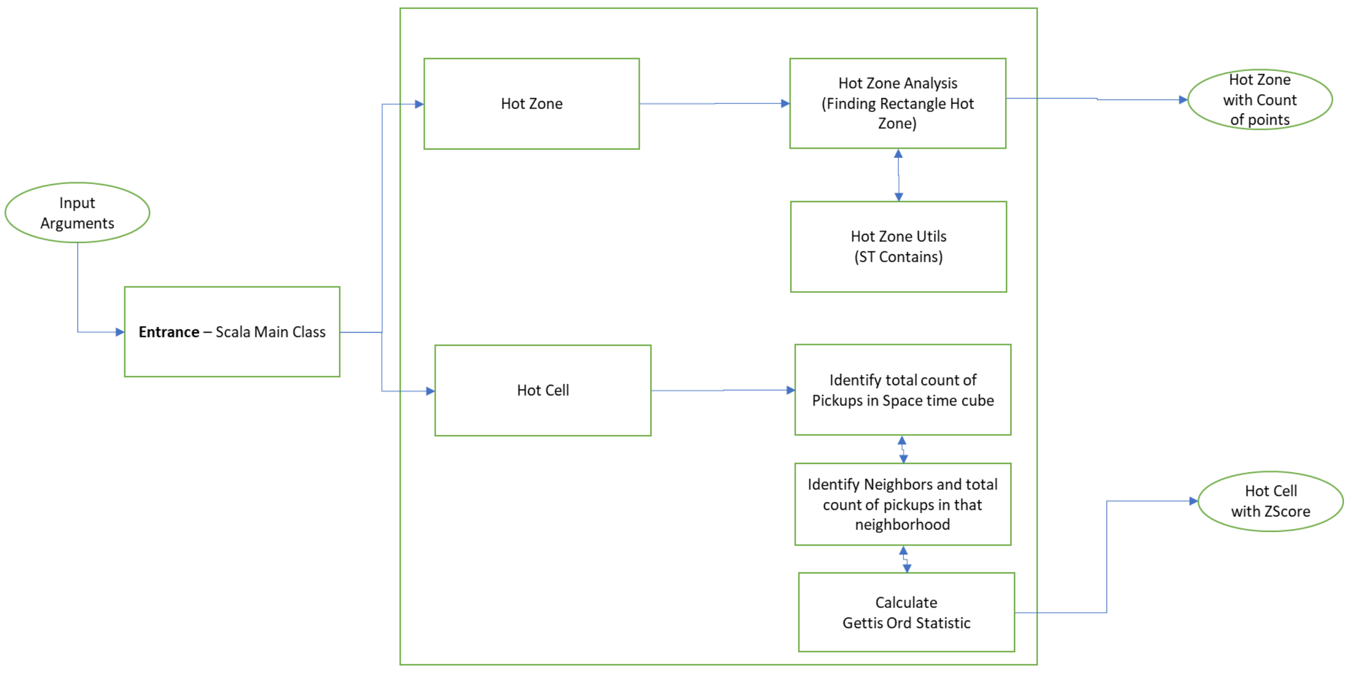
## DIAGRAMMATIC REPRESENTATION OF THE SOLUTION

### Process Flow: Phase 1 >>> ST_Contains and ST_Within Implementation logic



## Process Flow :Phase 2 >>>Hot Zone Analysis & Hot Cell Analysis

**Hot Spot Analysis – Hot Zone / Hot Cell**


Input Arguments
Entrance – Scala Main Class
Hot Zone
Hot Zone Analysis (Finding Rectangle Hot Zone)
Hot Zone Utils (ST Contains)
Hot Zone with Count of points
Hot Cell
Identify total count of Pickups in Space time cube
Identify Neighbors and total count of pickups in that neighborhood
Calculate Gettis Ord Statistic
Hot Cell with ZScore

## DEVELOPMENT/ PROGRAMMING

### Phase 1: ST_CONTAINS/ST_WITHIN

**SparkSQLExample.scala** calls for **SpatialQuery.scala** file mainly for Range Query, Range Join Query, Distance Query and Distance Join Query. The functions are defined in Spatialquery.scala and internally they use ST_Contains / ST_Within (defined as separate functions).

Functions calling  ST_Contains in the SpatialQuery.scala file are

- runRangeQuery
- runRangeJoinQuery.

Functions calling  ST_Within in the SpatialQuery.scala file are

- runDistanceQuery
- runDistanceJoinQuery

### ST_Contains

The ST_Contains logic was defined as follows,

- The rectangle co-ordinates and point values are taken as an input array and converted to double datatype and then assigned to variables.
- For checking if point values are within the rectangle, min and max values of the rectangle co-ordinates are first calculated and then verified if the point value is within this range by returning a Boolean value.
- ST_Contains return value is processed by Range and Range Join query functions to get the final count of records.
- Comments are provided for easy readability.

### ST_Within

The ST_Within logic was defined as follows,

- Two set of point values are taken as input array and converted to double data type and assigned to variables.
- Using Coordinate Geometry - Pythagoras formula, distance between 2 points are calculated i.e., First find length and breadth and then calculate diagonal distance between point co-ordinates.
- Compare our calculated value with the given input distance to see if the point co-ordinates are within the range.

## Phase 2: Hot Zone Analysis

- For identifying the hot zone, two datasets rectangle and point set are parsed and ST_Contains function is called for to identify the points within the rectangle. For ST_Contains logic refer above.
- Next, all points within the rectangle are grouped together and listed and it implies that the rectangle that has the maximum points can be identified from the set.

## Phase 2: Hot Cell Analysis

User Story: As a user, I want to be able to retrieve list of 50 most statistically significant hot spots using Getis-Ord statistic.

Given: Dataset which contains a collection of New York City Yellow Cab taxi trip records spanning from January 2009 to June 2015

When: hotcellanalysis is run on the data set

Then: the program outputs the list of 50 hot spot cells which has the most statistically significant Getis-Ord values.

## Assumptions

- The pickup location is at the center of the neighborhood cube.
- The spatial weight to any neighbor is 1. The spatial weight to itself is 1. Hence, when the neighborhood is within space-time cube, the sum of Wij = 27.
- For user stories with Priority = Must Have, the Wij (spatial weight) for a pickup point's neighborhood is a constant equal to 27.
- To support computing Wij when part of neighborhood is outside of the space-time cube, a user story Edge Neighborhood is in the backlog. The spatial weight will be calculated based on the position of cell on the time-space cube. There are 3 scenarios that would be supported: 1) pickup location is on the edge of time space cube, 2) pickup location is on the corner of time space cube, 3) pickup location is on the side of the time space cube.

## Design

1. Perform select query on PickupInfo aggregated on x,y,z to create PickupPoints dataframe.

$$PickupPoints\ dataframe\ =\ \{x, y, z, xi, xisquare\},$$
$$where\ xi\ =\ total\ pickup\ points\ for\ cell\ x, y, z,$$
$$xisquare\ =\ xi * xi$$

2. Perform join query on PickupPoints to create PickupWIJXJ dataframe. This is a join of PickupPoints with itself to map each pickup point to its neighbors and aggregate the count of pickup points of its neighbors.

$$PickupupWIJXJ\ dataframe\ =\ \{x, y, z, wijxj\}$$
$$where\ wijxj\ is\ total\ pickup\ points\ for\ the\ cell's\ neighbors.$$

3. Perform the Getis-Ord calculation for each pickup point and output 50 cells with highest Getis-Ord statistic value.

$$numCells\ =\ total\ number\ of\ cells\ in\ space\ time\ cube.$$
$$Xj\ =\ aggregate\ sum\ of\ xi$$
$$Xjsquare\ =\ aggregate\ sum\ of\ xisquare$$
$$mean\ =\ Xj\ /\ numCells$$
$$stddev\ =\ sqrt\ (xjsquare\ /\ numCells\ -\ mean * mean)$$

$$Getis - Ord = (wijxj\ -\ mean\ *\ 27)\ /\ stddev\ *\ math.sqrt((numCells * 27\ -\ (27 * 27))/numCells\ - 1)$$

## <u>QUALITY ASSURANCE/ TESTING</u>

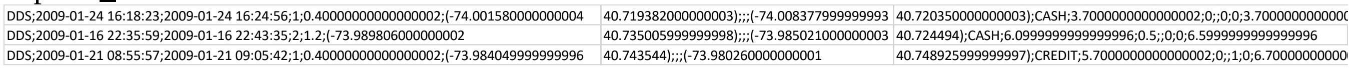
Once the code was compiled in IntelliJ, the. scala file will be run for testing the input data.

1. To pass the testing parameters, Input arguments are passed in IntellJ IDEA using **Edit Configuration – Program Arguments** section.

2. <u>Sample Input arguments –</u>

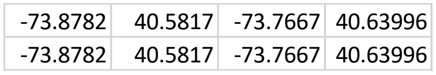
   *test/output hotzoneanalysis src/resources/point-hotzone.csv src/resources/zone-hotzone.csv*

   <u>For Hot Zone Analysis – Input files</u>
   a. point_hotzone.csv



| | | |
|---|---|---|
| DDS;2009-01-24 16:18:23;2009-01-24 16:24:56;1;0.40000000000000002;(-74.001580000000004 | 40.719382000000003);;;(-74.008377999999993 | 40.720350000000003);CASH;3.7000000000000002;0;;0;0;3.700000000000 |
| DDS;2009-01-16 22:35:59;2009-01-16 22:43:35;2;1.2;(-73.989806000000002 | 40.735005999999998);;;(-73.985021000000003 | 40.724494);CASH;6.0999999999999996;0.5;;0;0;6.5999999999999996 |
| DDS;2009-01-21 08:55:57;2009-01-21 09:05:42;1;0.40000000000000002;(-73.984049999999996 | 40.743544);;;(-73.980260000000001 | 40.748925999999997);CREDIT;5.7000000000000002;0;;1;0;6.70000000000 |

   b. zone-hotzone.csv



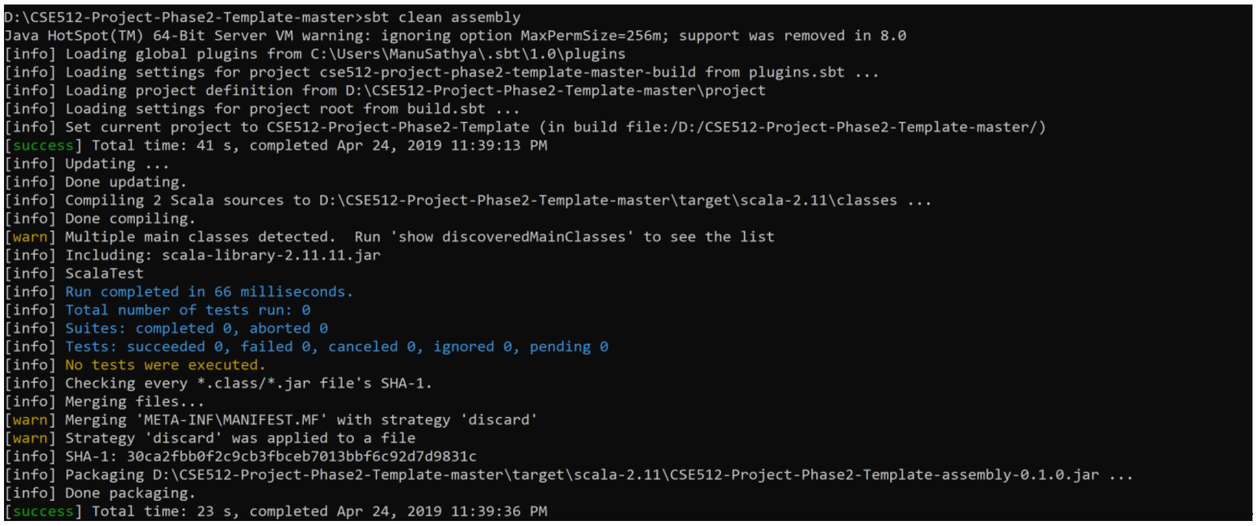
| | | | |
|---|---|---|---|
| -73.8782 | 40.5817 | -73.7667 | 40.63996 |
| -73.8782 | 40.5817 | -73.7667 | 40.63996 |

3. <u>Sample Test Results from IntelliJ for ST Contains</u>

```
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
19/04/28 13:33:10 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
+--------------------+
|                 _c0|
+--------------------+
|-93.579565,33.205413|
|-93.417285,33.171084|
|-93.493952,33.194597|
|-93.436889,33.214568|
+--------------------+
+--------------------+--------------------+
|                 _c0|                 _c0|
+--------------------+--------------------+
|-93.63173,33.0183...|-93.579565,33.205413|
|-93.63173,33.0183...|-93.417285,33.171084|
|-93.63173,33.0183...|-93.493952,33.194597|
|-93.63173,33.0183...|-93.436889,33.214568|
```

   Output CSV files will contain the name of Query (Range Query, Distance Join etc) and the count of records. Verified if the count matches with example answer file provided in project template document.

4. Next step is to build the .jar file. We have to make sure the input files and output location are correct and run the following command **sbt clean assembly** from the project home folder.



```
D:\CSE512-Project-Phase2-Template-master>sbt clean assembly
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=256m; support was removed in 8.0
[info] Loading global plugins from C:\Users\ManuSathya\.sbt\1.0\plugins
[info] Loading settings for project cse512-project-phase2-template-master-build from plugins.sbt ...
[info] Loading project definition from D:\CSE512-Project-Phase2-Template-master\project
[info] Loading settings for project root from build.sbt ...
[info] Set current project to CSE512-Project-Phase2-Template (in build file:/D:/CSE512-Project-Phase2-Template-master/)
[success] Total time: 41 s, completed Apr 24, 2019 11:39:13 PM
[info] Updating ...
[info] Done updating.
[info] Compiling 2 Scala sources to D:\CSE512-Project-Phase2-Template-master\target\scala-2.11\classes ...
[info] Done compiling.
[warn] Multiple main classes detected.  Run 'show discoveredMainClasses' to see the list
[info] Including: scala-library-2.11.11.jar
[info] ScalaTest
[info] Run completed in 66 milliseconds.
[info] Total number of tests run: 0
[info] Suites: completed 0, aborted 0
[info] Tests: succeeded 0, failed 0, canceled 0, ignored 0, pending 0
[info] No tests were executed.
[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: 30ca2fbb0f2c9cb3fbceb7013bbf6c92d7d9831c
[info] Packaging D:\CSE512-Project-Phase2-Template-master\target\scala-2.11\CSE512-Project-Phase2-Template-assembly-0.1.0.jar ...
[info] Done packaging.
[success] Total time: 23 s, completed Apr 24, 2019 11:39:36 PM
```

5. From Spark Home/bin location, test the executable .Jar file created using **Spark-submit**.

   *E:\Spark\spark-2.4.1-bin-hadoop2.7\bin>spark-submit D:\CSE512-Project-Phase2-Template-master\target\scala-2.11\CSE512-Project-Phase2-Template-assembly-0.1.0.jar result/output rangequery src/resources/arealm10000.csv -93.63173,33.0183,-93.359203,33.219456 rangejoinquery src/resources/arealm10000.csv src/resources/zcta10000.csv distancequery src/resources/arealm10000.csv -88.331492,32.324142 1 distancejoinquery src/resources/arealm10000.csv src/resources/arealm10000.csv 0.1*

6. After verifying the output from Jar file and comparing with IntelliJ output and example answer output, Jar file is submitted to Coursera auto grader.

## **CHALLENGES / LESSONS LEARNT:**

1. Setting up the environment and make sure everything is ready for development. Installing Apache Spark and setting up environment variables including Scala installation. Different versions of Scala were available (We had to make sure build.sbt file had the same Scala version that was installed).
2. Scala was new and writing spatial queries and figuring out the logic was interesting and mainly running the .jar had difficulties. Correct libraries to be used and build.sbt modified with correct parameters (changing to % provided%)
3. Understanding and Calculating Getis-Ord Statistic - It took some time to parse and interpret the problem. There were assumptions about the problem that we were not aware of initially. For example, we didn't know that the pickup location is at the center of the cube or how the spatial weight is calculated. We attended virtual office hours and also synced with colleagues to gain better understanding of the problem.
4. There is ramp-up involved in using Spark/SparkSQL. Perhaps, on revisiting the problem and learning more about Spark/SparkSQL, there could be  a better approach. In our design, we currently have 3 DataFrames: 1 for the pickup points aggregation, 1 of the wijxj aggregation and1 for the Getis Ord calculation. We were looking at using UDF to be able to calculate wijxj and store in same dataframe however it seems we are not able to update the values of the dataframe with new column and values calculated from UDF. We also would like to look at the functionality such as FlatMap to see if those could instead be used to implement the solution.