CSC-8101 ENGINEERING FOR AI
Neo4j COURSE-WORK

NAME: KAILASH BALACHANDIRAN
STUDENT ID: 220243160

SCENARIO:

A taxi company operating in New York City (NYC) is rethinking its fleet allocation strategy. Their restructuring process is motivated by recent changes in passenger travel patterns (provoked by Covid) and the arrival of new competitors. To minimise costs, the company wants to station its fleet in a maximum of 20 city zones, spread across the city. At the same time, the company wants to maximise trips served and therefore choose zones that have hub-like characteristics, i.e. that are well-connected and near centres of high user activity. Lastly, the company considers operating purely outside the Manhattan borough where there is less competition.

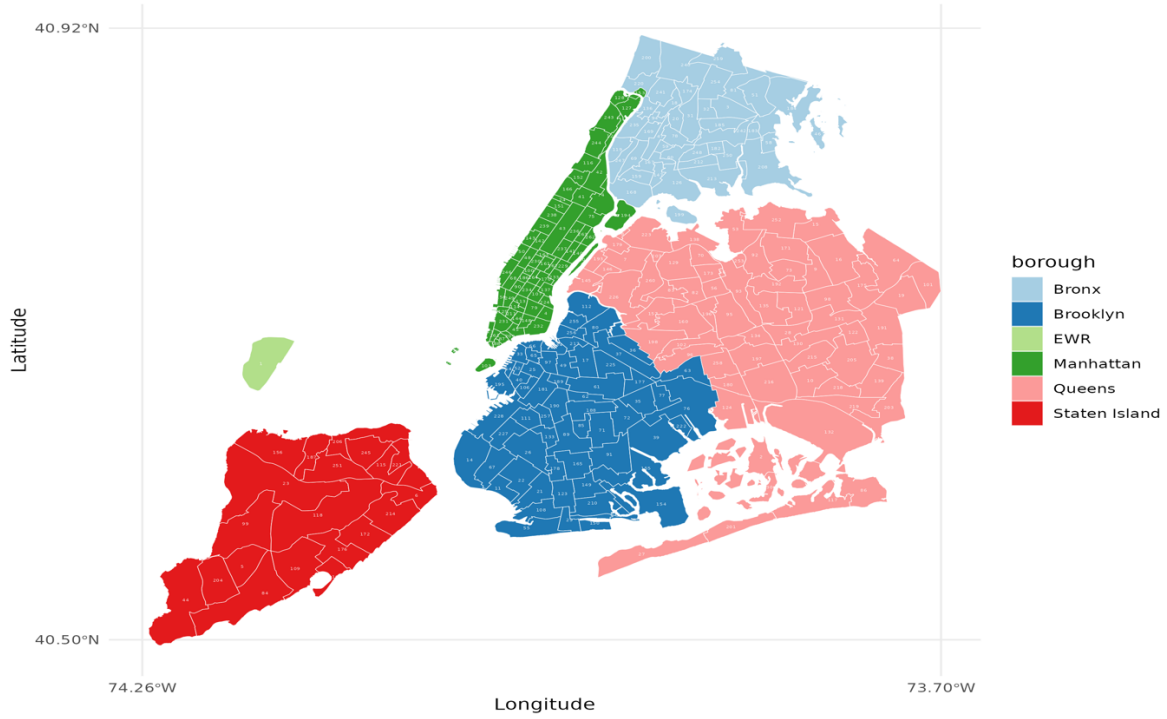

Figure 1.1

To help with this task, the taxi company contracts you to analyse its historical trip records and recommend a list of city zones where they should station their fleets. To that end, the company gives you a copy of its Neo4j graph database, available here, containing aggregate statistics of taxi trips undertaken in 2021. There are two types of nodes: borough and zone. Two zones are connected by a relationship of type :CONNECTS if a minimum of one trip was recorded between them. In addition, CONNECTS relationships have property trips representing the total number of trips observed in the year. Lastly, a node is related to a borough via a relationship of type: IN.

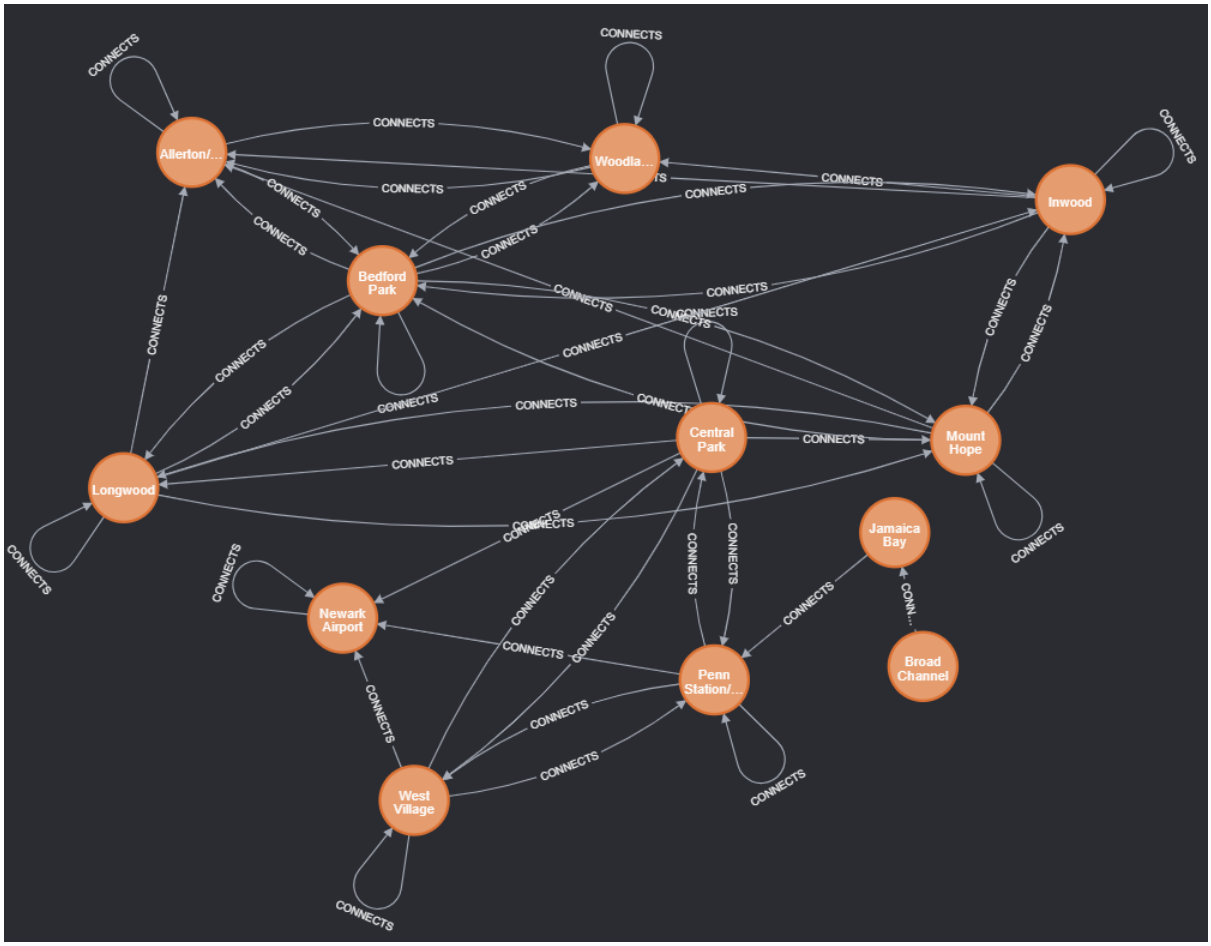

Figure 1.2

Based on preliminary exploration of the database and the NYC zones and boroughs map (shown above), you decide to tackle the challenge by combining two methods of network analysis available in the Neo4j Data Science Library. Your action plan is divided into three parts:

1. Perform community detection to identify clusters of strongly connected zones.
2. Perform centrality analysis to measure the hub-like features of each zone.

3. Combine the results above by identifying the top 3 zones with highest centrality score within each community cluster. This processed is done twice for the entire city:
   i. once including Manhattan, and
   ii. once excluding Manhattan.

In addition to your Neo4j analysis, you will use the visualisation app below to help interpret the results of your network analysis and present them to your client (the taxi company). To use the app, export your results at the end of each stage to a csv file and load it into the app (more details given in each task).

- http://csc8101-neo4j-shiny.uksouth.cloudapp.azure.com/

TASK DETAILS:

There are four tasks:

1. Find isolated nodes
2. Compute the community cluster of each node
3. Compute the centrality score of each node
4. Find the top centrality zones within each community

In task 0 you will develop a cypher query to find nodes and relationship of a certain type.

In tasks 1 and 2 you will execute two different algorithms available in the Neo4j Data Science Library (GDS). Both tasks follow the typical workflow described here:

- Create an (in-memory) graph projection.
- Estimate the memory necessary to run the algorithm (optional).
- Run the algorithm in stats mode to summarise the output of the algorithm.
- Run the algorithm in stream mode to run the algorithm but not store the results in the original graph.

In task 3 you will develop a cypher query that combines the outcomes of tasks 1 and 2.

Due to database writing restrictions, the coursework is split across three neo4j databases - one for Task 0, another for tasks 1 and 2 and a third for task 3.
Go here for connection details.

TASK 0 - Find isolated nodes

Find all:

1. Self-pointing relationships, that is, all relationship instances of type: CONNECTS where the start node is the same as the end node.

```
MATCH (k)-[r:CONNECTS]->(k)
RETURN k,r
```

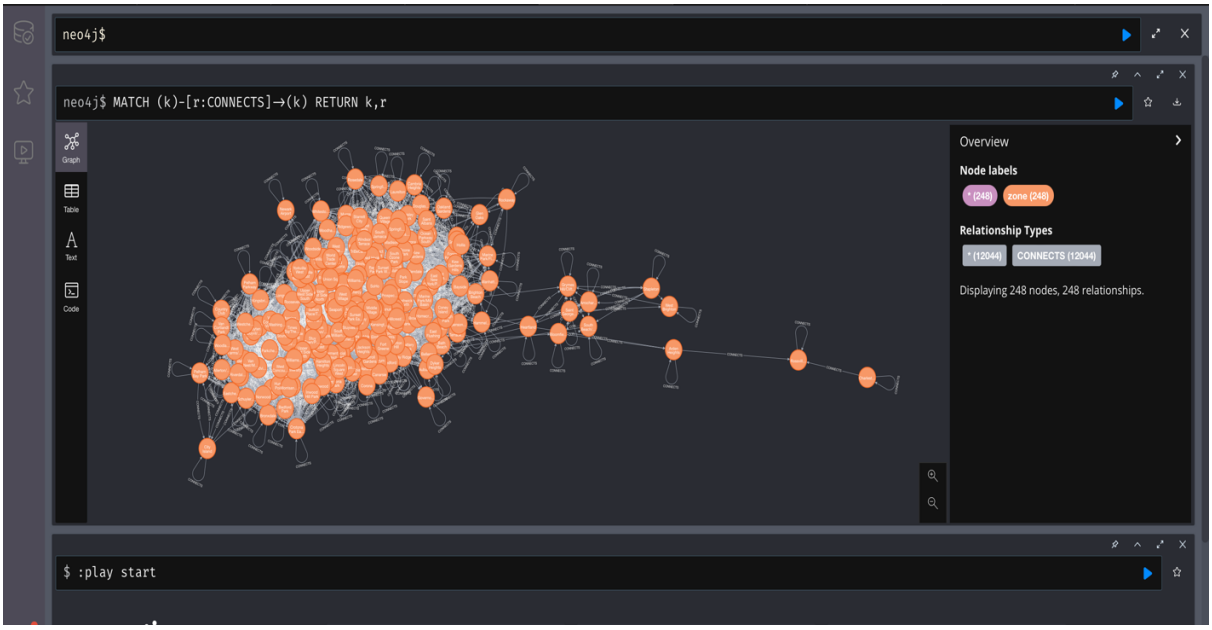

Figure 0.1

Figure 0.2

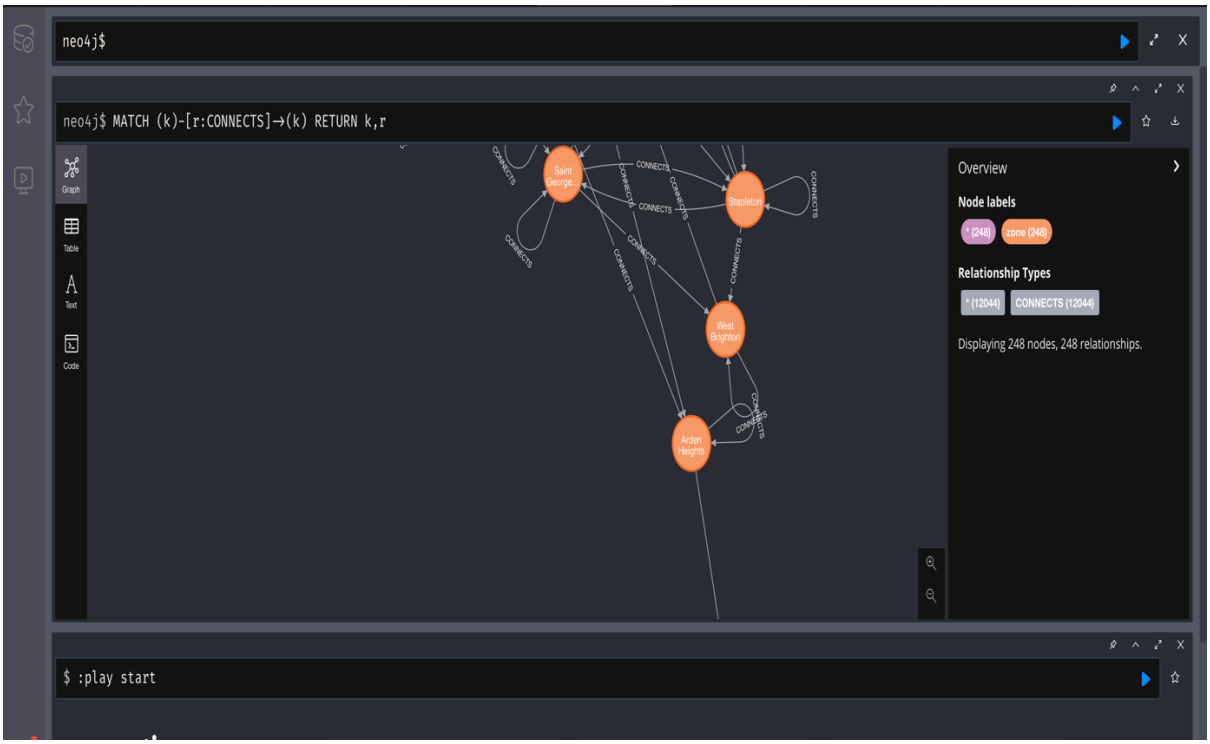

Figure 0.3

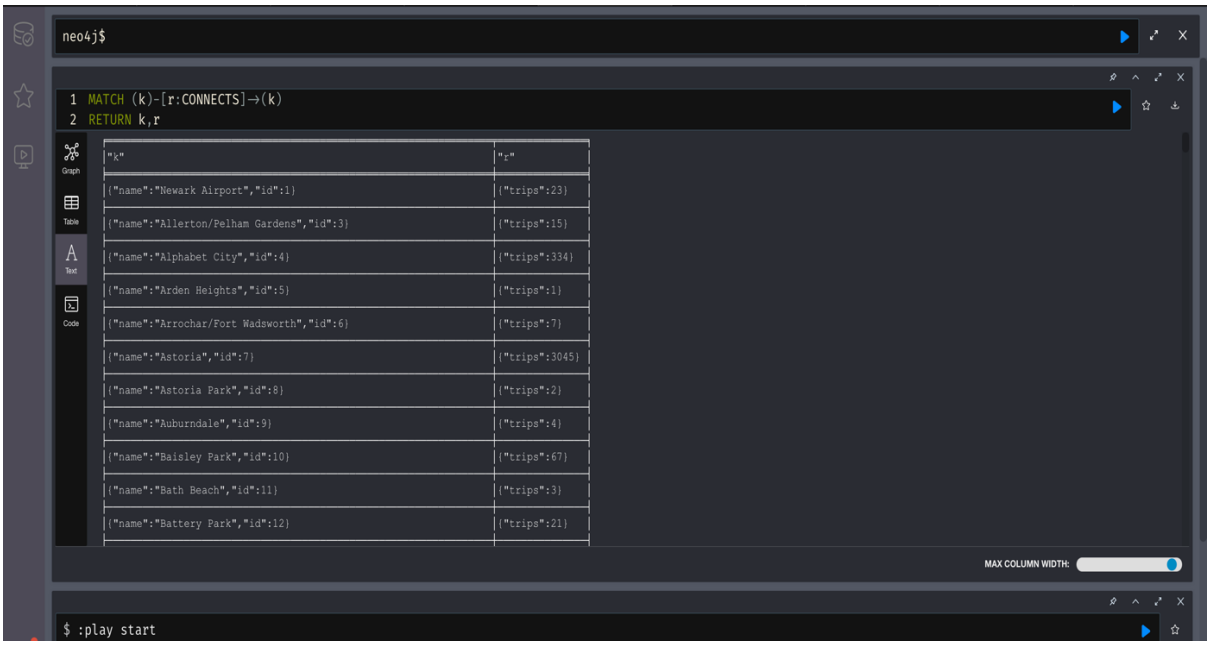

Figure 0.4

2. Find all isolated nodes, i.e., all nodes that have no relationship instances of type: CONNECTS to other nodes except possibly to themselves.

```
MATCH (k)
WHERE NOT (k)-[:CONNECTS]-() AND NOT (k)--(k)
RETURN k
```

Figure 0.5

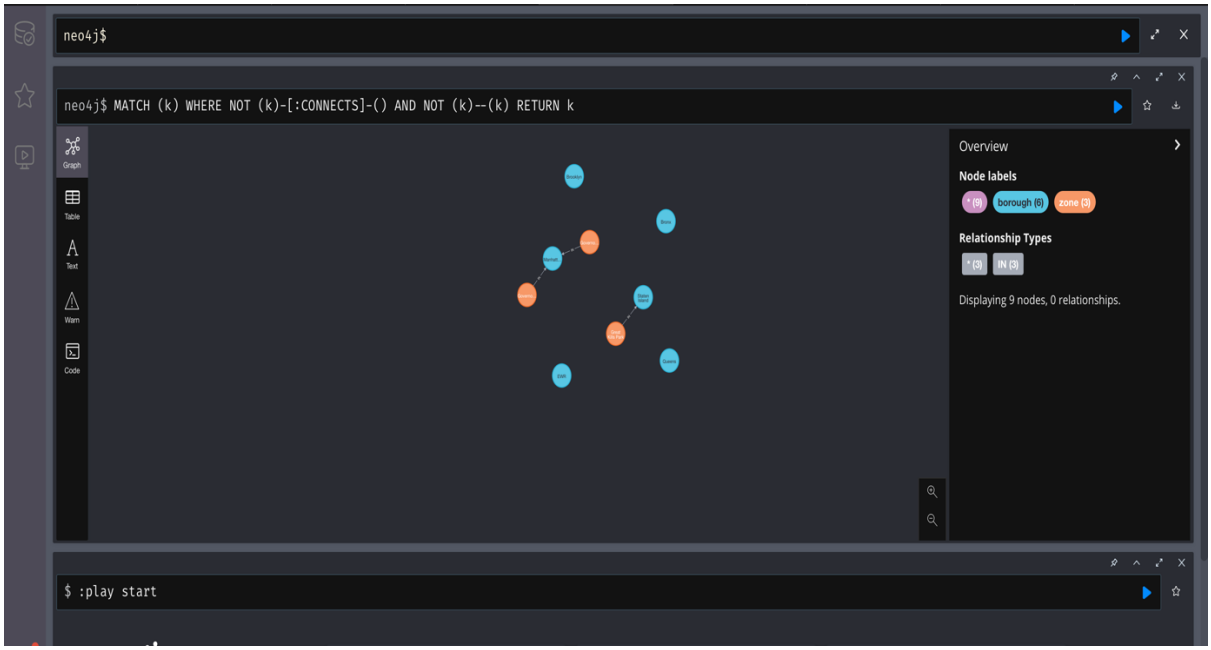

Figure 0.6

Figure 0.7

Task 1 - Community detection

Run the Louvain algorithm for community detection, available in the GDS library, with the following arguments:

- Graph projection of type UNDIRECTED.
- Name the graph projection as USERNAME-communities where USERNAME is your student number (this is necessary to avoid naming conflicts between students).
- Weighted by the trips property in :CONNECTS type of relationships.

```
CALL gds.graph.create(
'220243160-communities_7892',
'zone',
{
CONNECTS: {
type: 'CONNECTS',
orientation: 'UNDIRECTED',
properties: 'trips'
 }
}
 )
```

```
CALL gds.louvain.stats('220243160-communities_7892')
YIELD communityCount
```

```
CALL gds.louvain.stream('220243160-communities_7892', {
relationshipWeightProperty: 'trips' })
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).id AS zone_id,communityId AS community_id
```

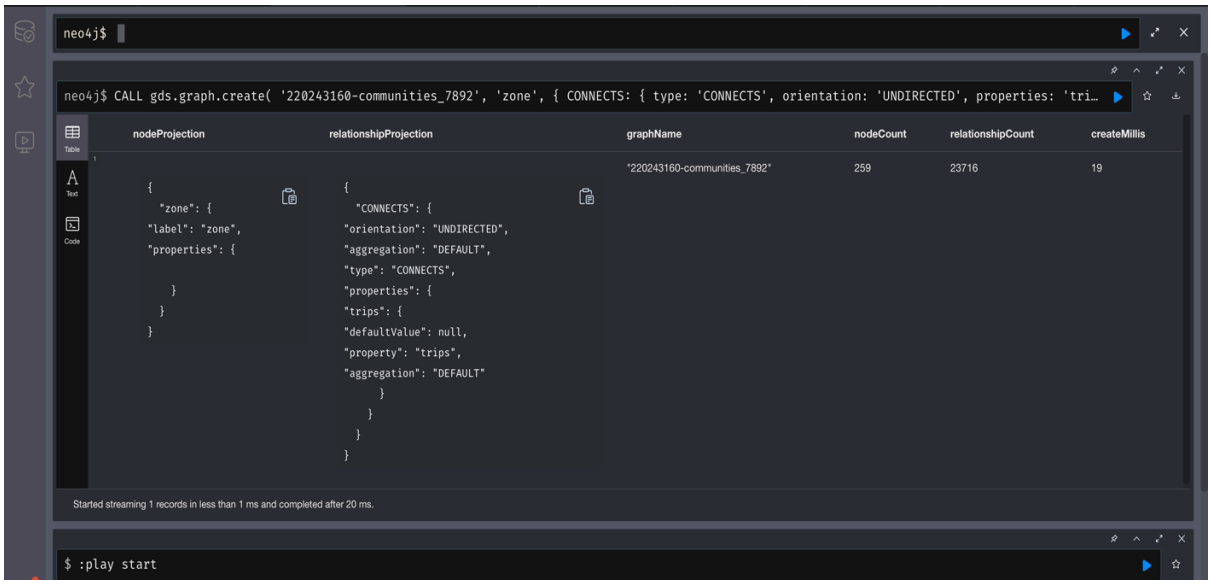

Figure 1.1

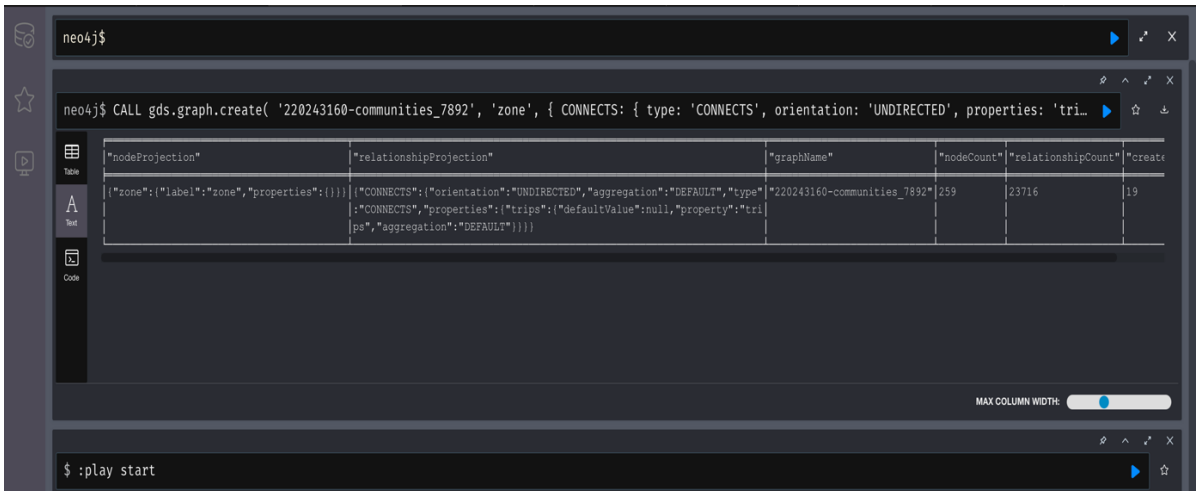

Figure 1.2

As specified above, run the algorithm in 2 modes: stats and stream:

- Report the number of communities using the stats mode,
- In stream mode, return the id and community properties of each zone (in this order),
- Export the results of running the algorithm in stream as a CSV file with two columns named zone_id, community_id and
- Visualise the results in the app by uploading the produced csv file (right-click to download image).

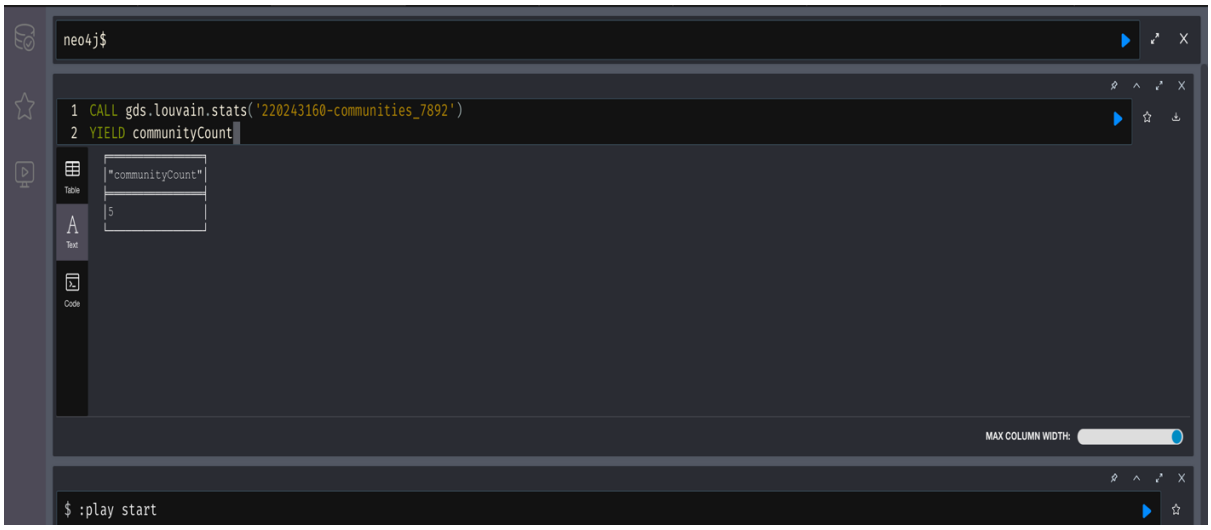

Figure 1.3

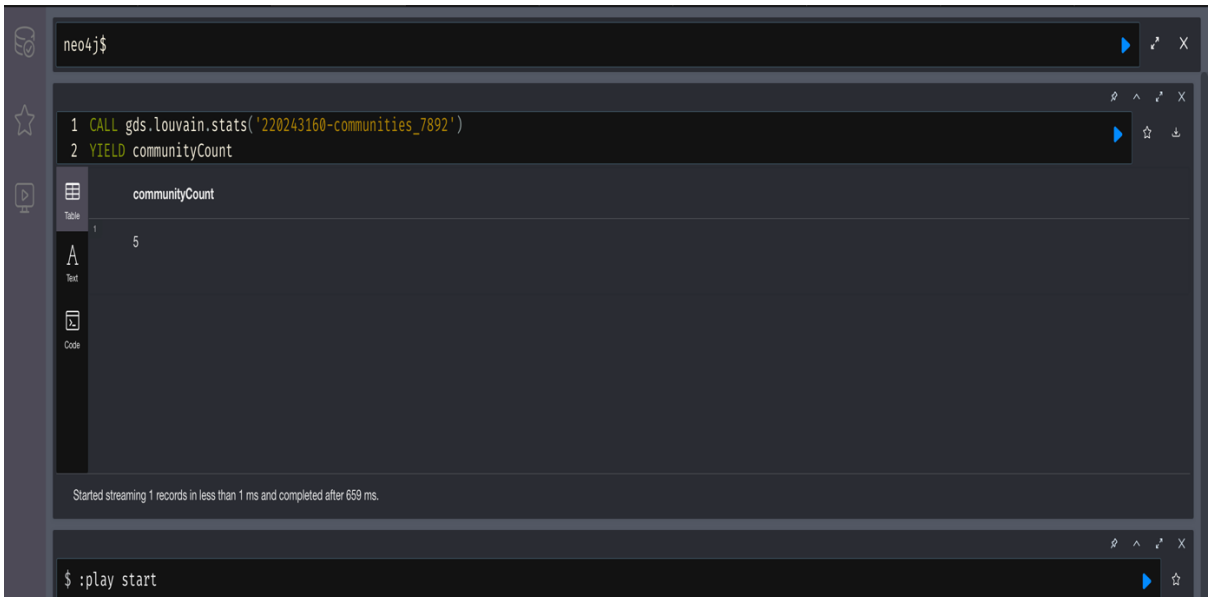

Figure 1.4

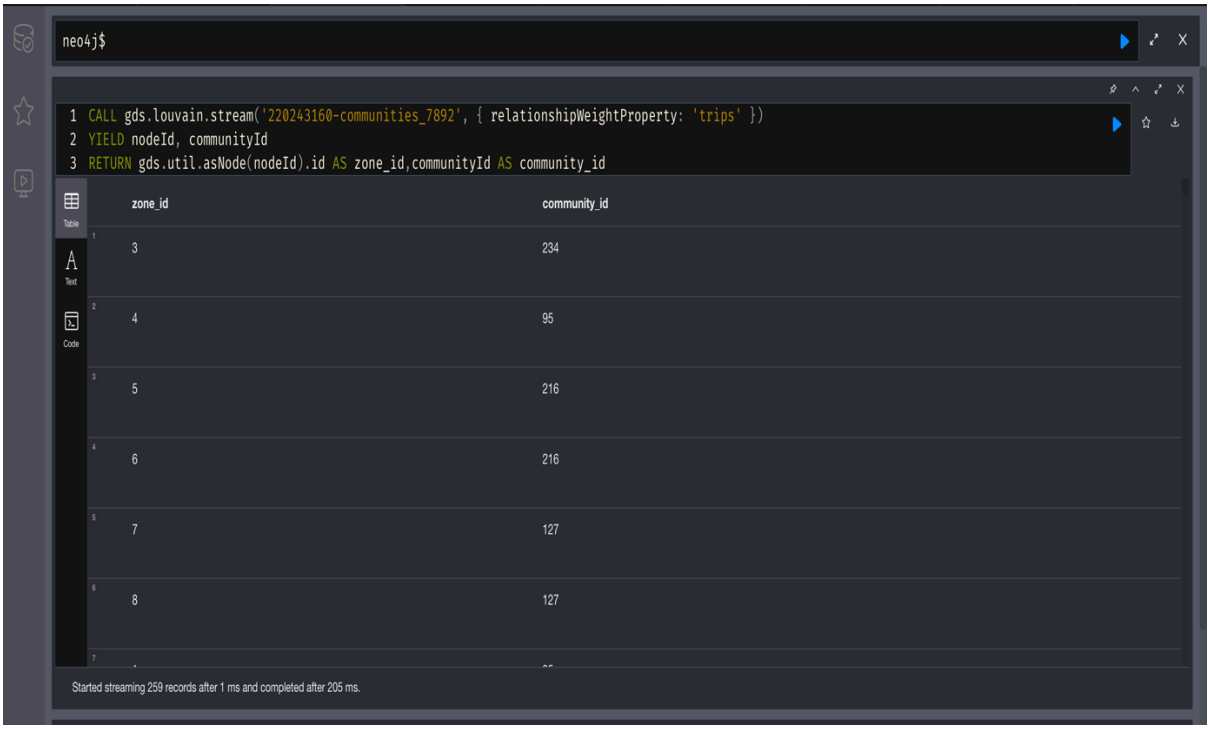

Figure 1.5

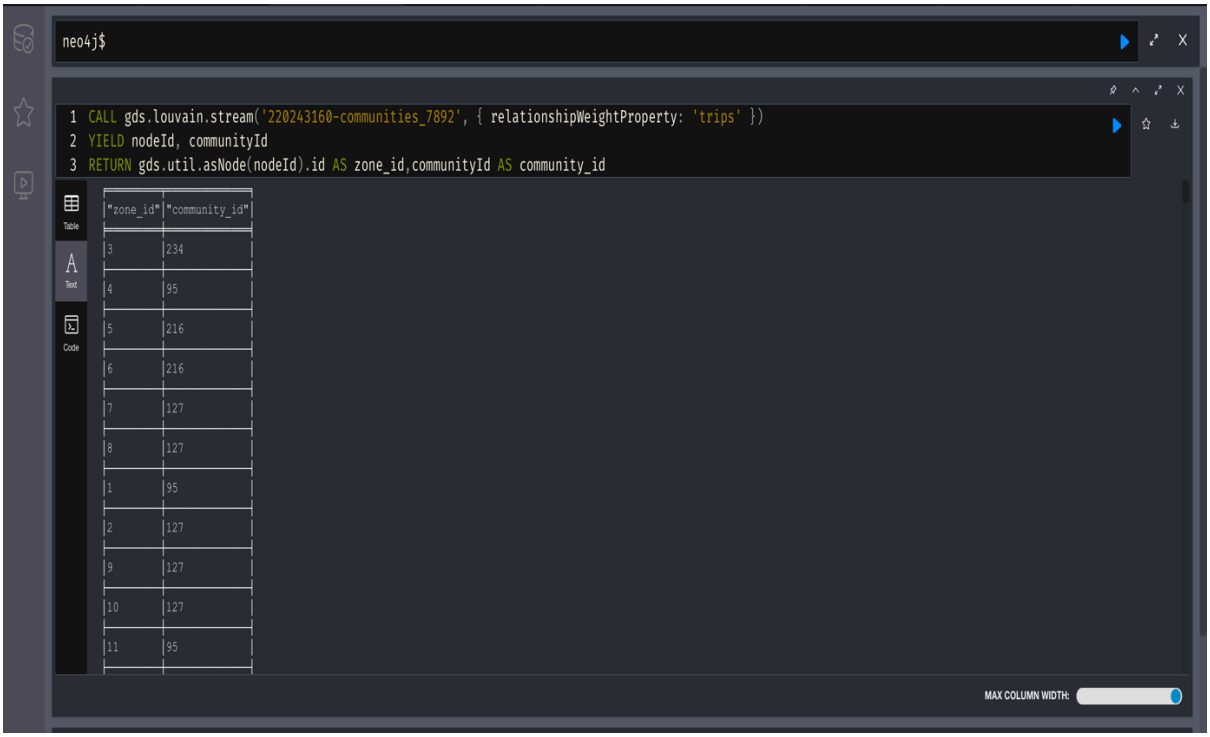

Figure 1.6

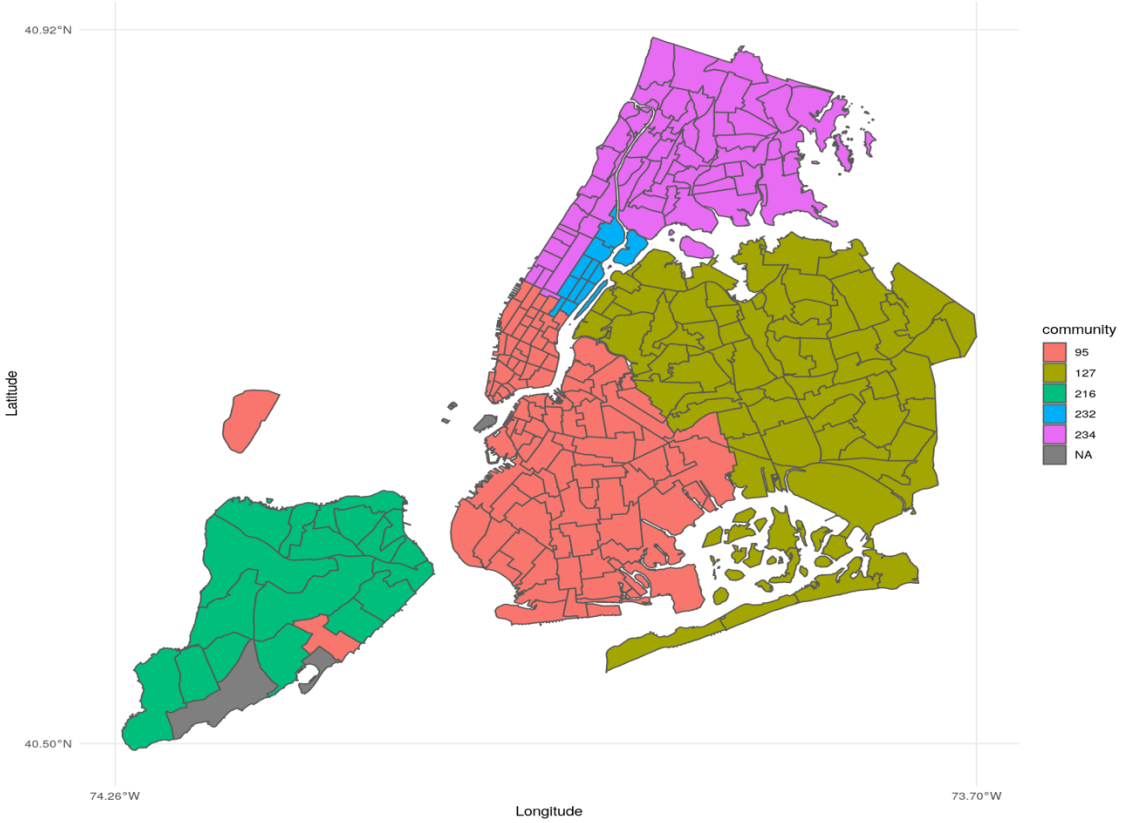

Figure 1.7

Task 2 - Centrality analysis

Run the Page Rank algorithm for centrality analysis, available in the GDS library, with the following arguments:

- Directed graph projection
- Name the graph projection as USERNAME-centrality where USERNAME is your student number (this is necessary to avoid naming conflicts between students).
- Damping factor: 0.75
- Weighted by the trips property in :CONNECTS type of relationships.

As specified above, run the algorithm in 2 modes: stats and stream:

- Report the maximum and minimum centrality score using the stats mode,
- In stream mode, return the id, centrality properties of each zone (in this order),
- Export the results of running the algorithm in stream as a CSV file with two columns named zone_id and centrality_score and
- Visualise the results in the app by uploading the produced csv file (right-click to download image).

```
CALL gds.graph.create(
'220243160-centrality_789',
'zone',
'CONNECTS',

{
relationshipProperties: 'trips'
}
)
```

```
  CALL gds.pageRank.write.estimate('220243160-centrality_789', {
 writeProperty: 'pageRank',
 maxIterations: 20,
 dampingFactor: 0.75
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory


CALL gds.pageRank.stats('220243160-centrality_789', {
 dampingFactor: 0.75,
 relationshipWeightProperty: 'trips'
})
YIELD centralityDistribution
RETURN centralityDistribution.max AS MAX , centralityDistribution.min AS MIN


CALL gds.pageRank.stream('220243160-centrality_789', {
 dampingFactor: 0.75,
 relationshipWeightProperty: 'trips'
})
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS zone_id, score AS centrality_score
ORDER BY score DESC
```

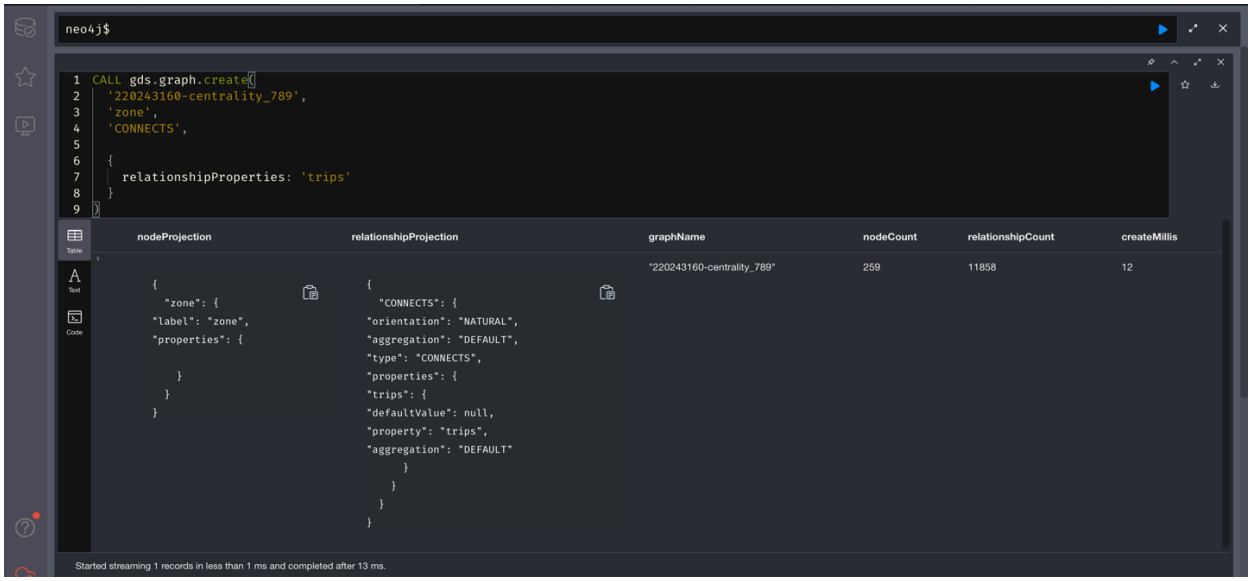

Figure 2.1

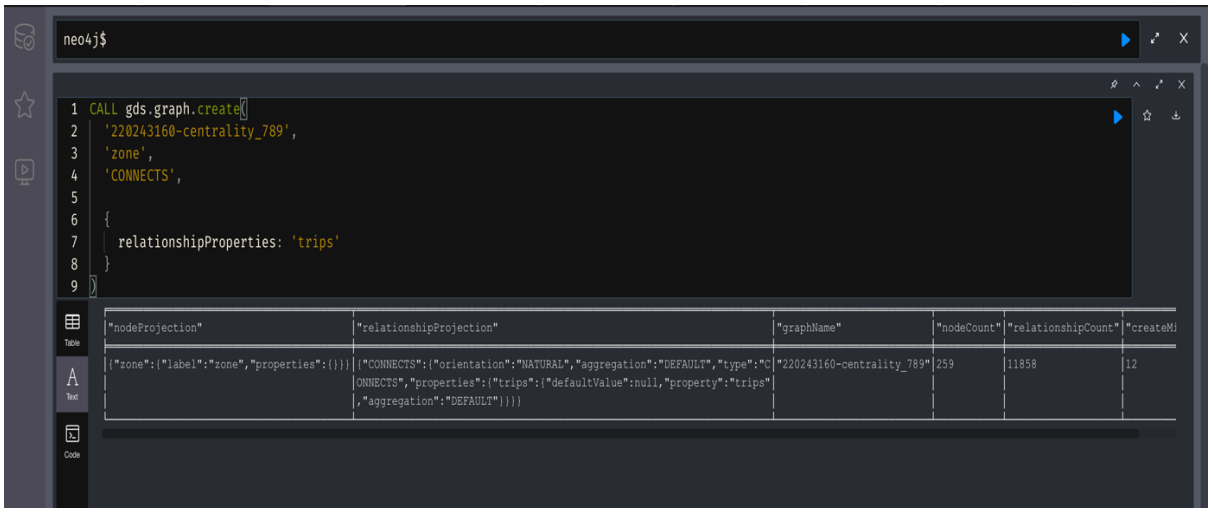

Figure 2.2

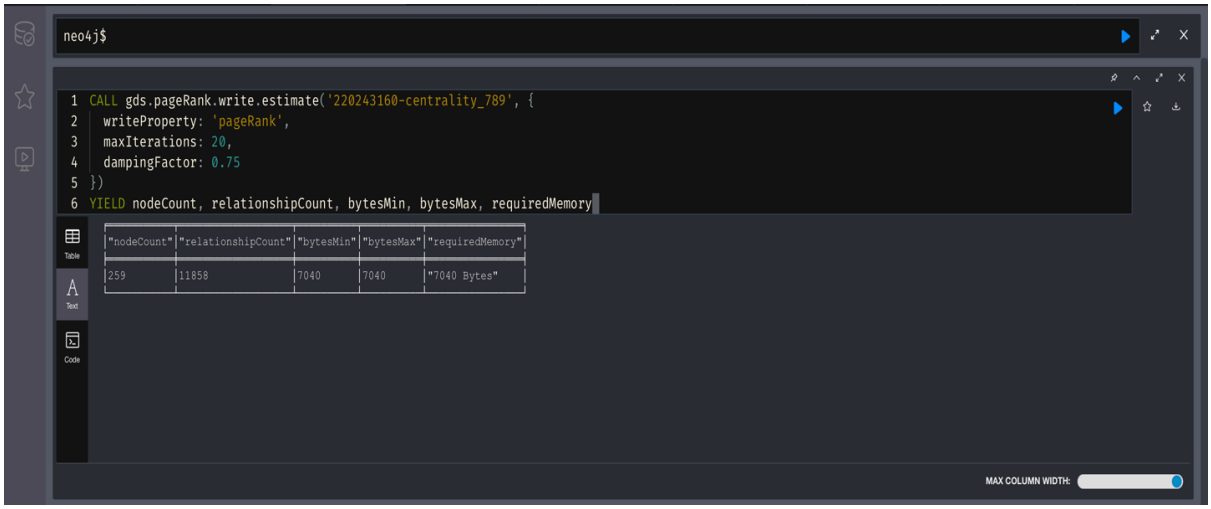

Figure 2.3

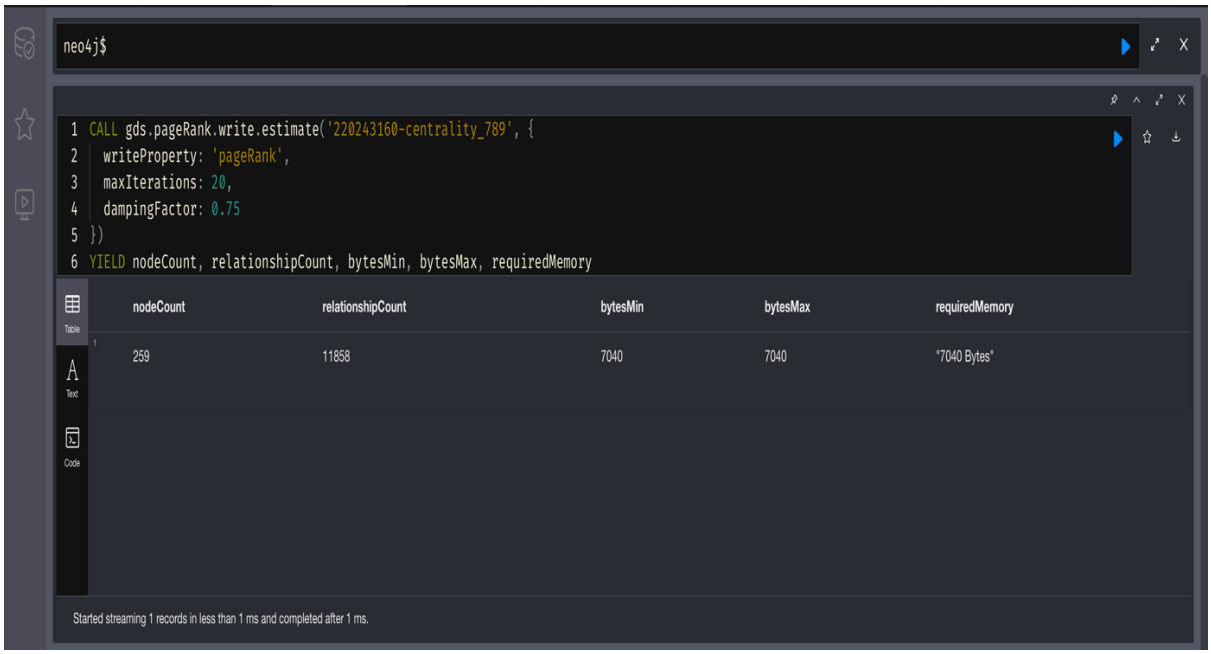

Figure 2.4

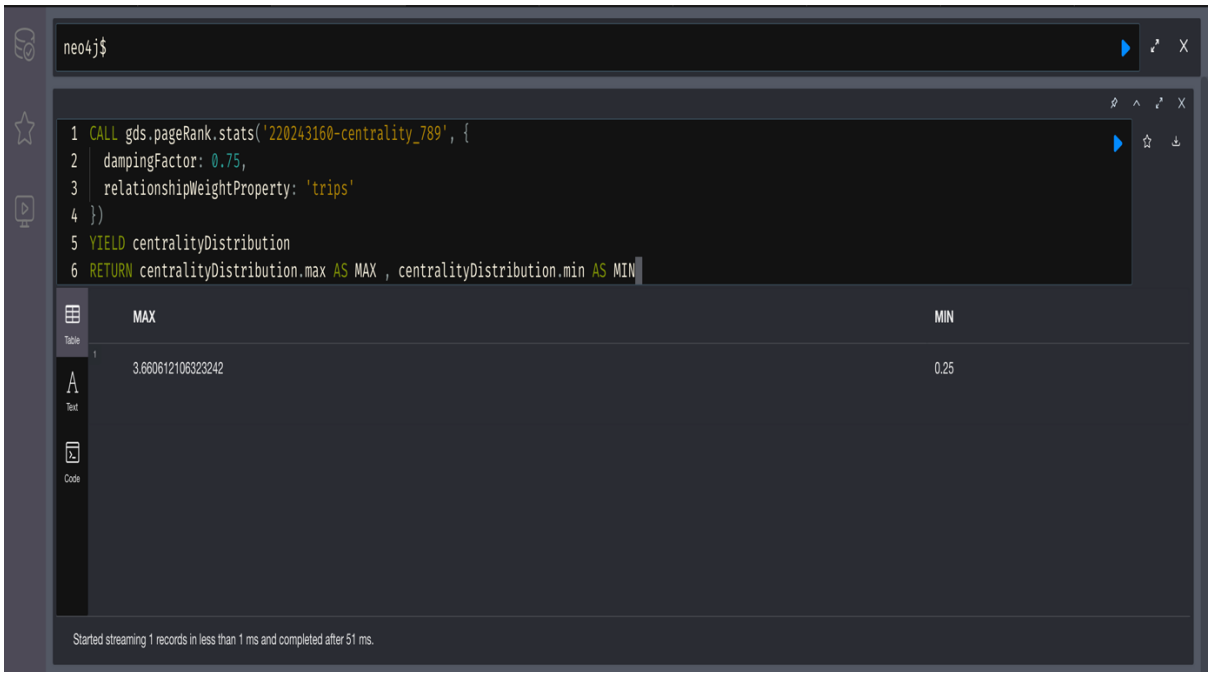

Figure 2.5

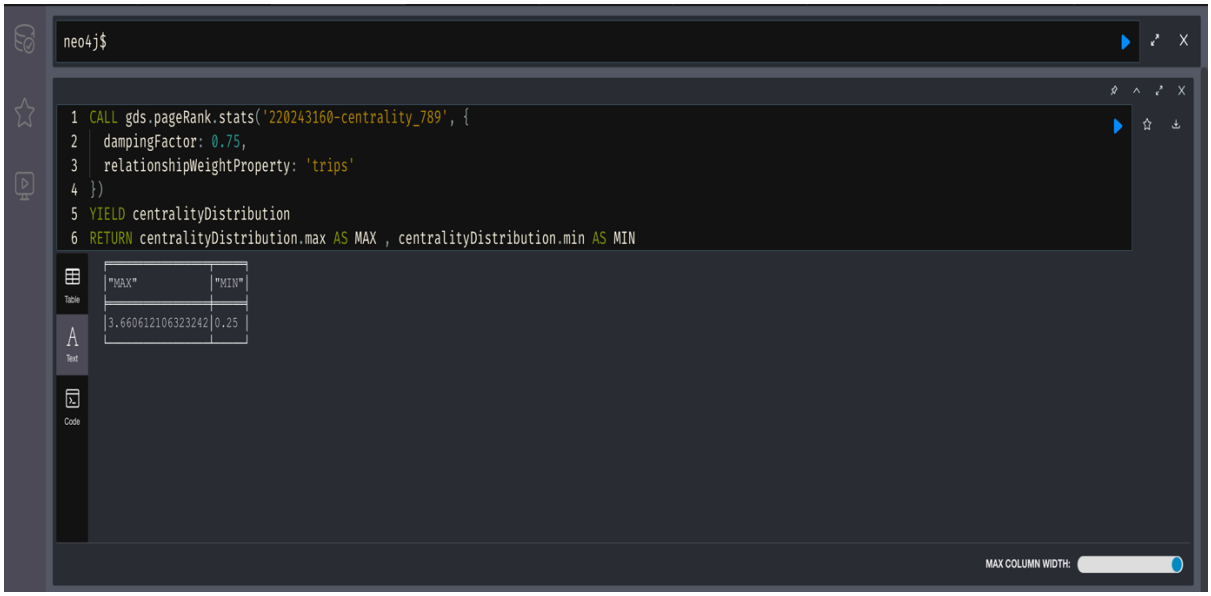

Figure 2.6

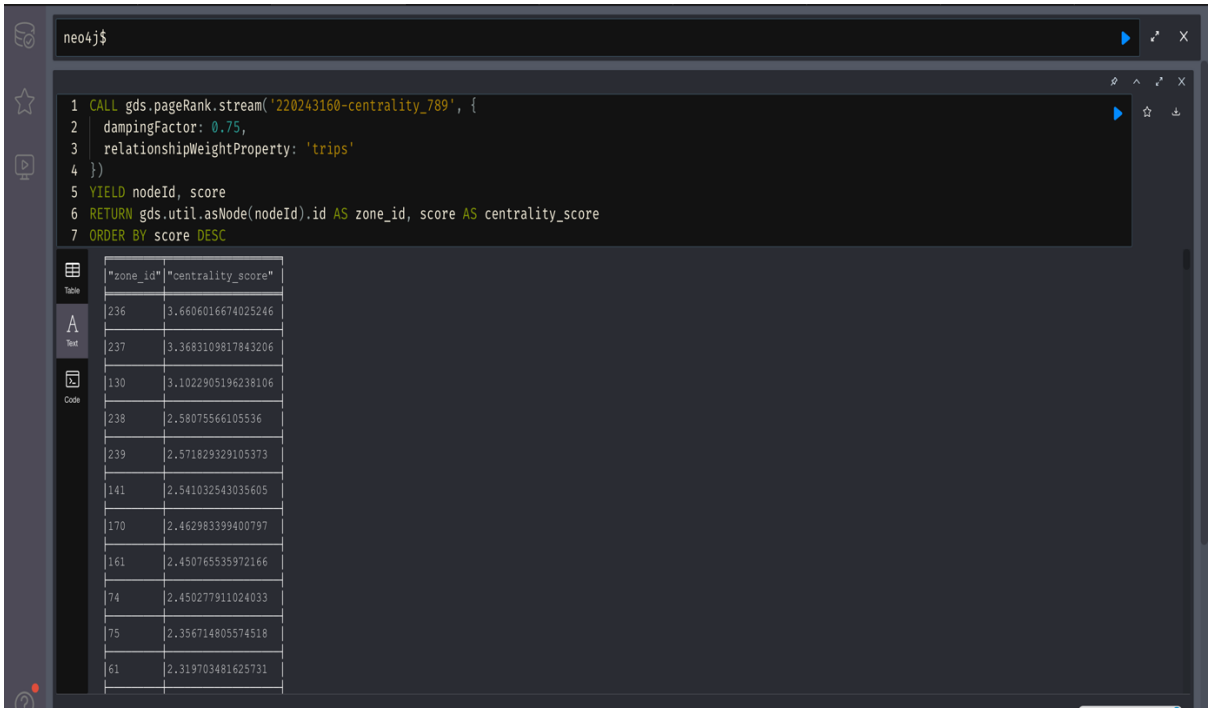

Figure 2.7

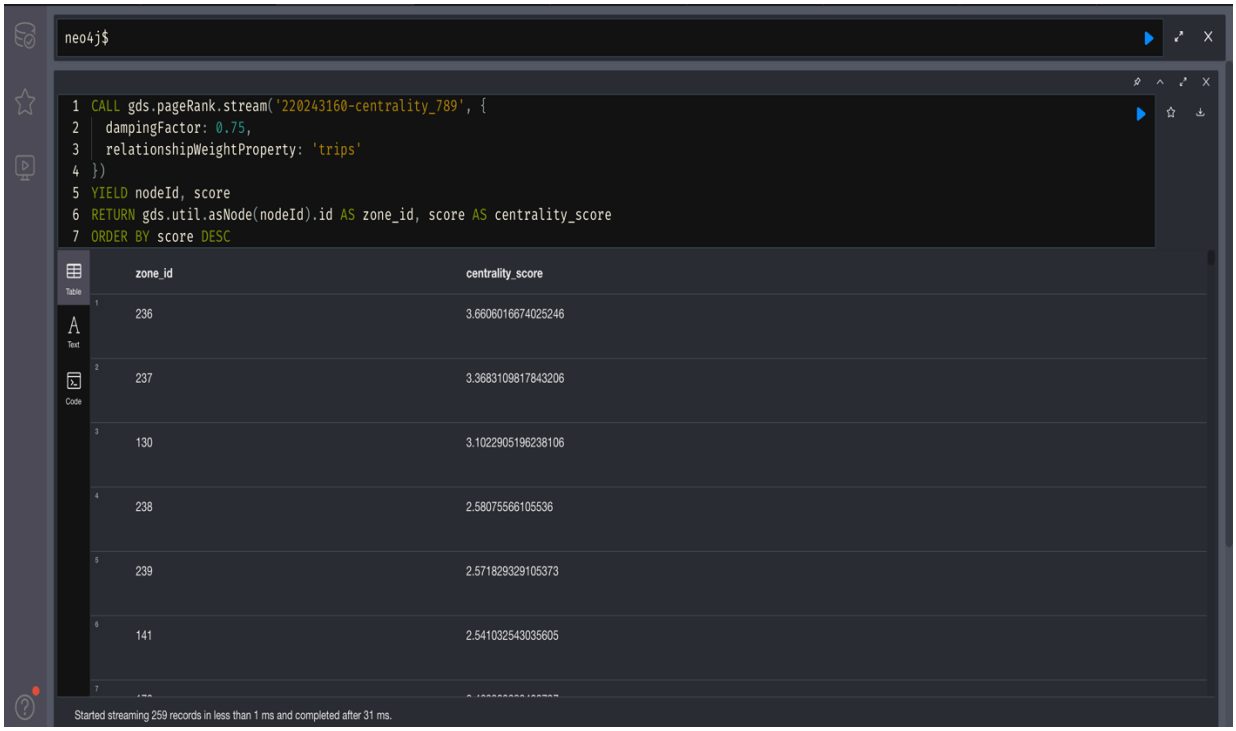

Figure 2.8

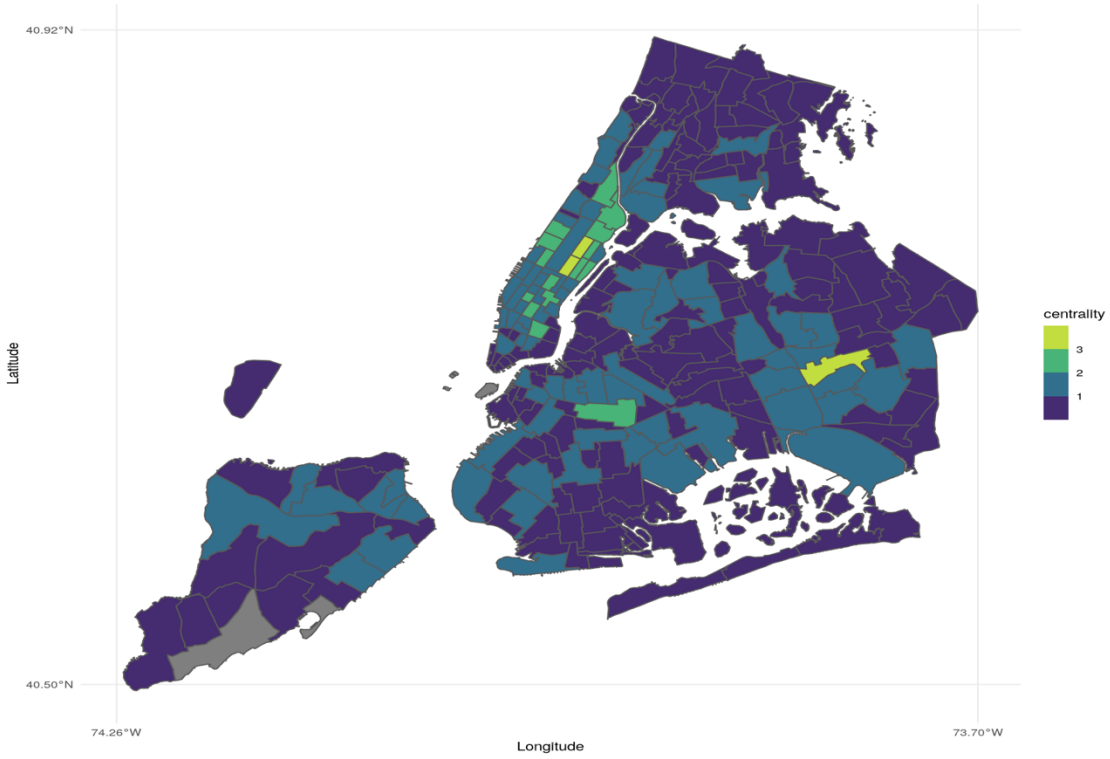

Figure 2.9

Task 3 - Zone recommendation

Find the top 3 highest centrality zones per each community:

1. Including zones in the 'Manhattan' borough.
2. Excluding zones in the 'Manhattan' borough.

Do this using the available zone properties community (representing the community id obtained in 1) and centrality (representing the centrality score obtained in 2).
For each case, return two columns: zone_id and community_id and export the query results to a csv file. Then, using the visualisationapp, upload these (one at a time) to the 'Task-3' file input and produce a separate and produce two separate images.

```
MATCH (n:zone)
with n order by n.centrality desc
with n.community as class,collect({id:n.id, score:n.centrality}) as listt
UNWIND listt[0..3] AS l
return l.id as zone_id ,class as community_id
order by class
```

```
MATCH (n:zone)-[r:IN]->(b:borough) WHERE b.name <> 'Manhattan' with n order by
n.centrality desc
with n.community as class,collect({id:n.id, score:n.centrality}) as listt
UNWIND listt[0..3] AS l
return l.id as zone_id ,class as community_id
order by class
```

Figure 3.1

Figure 3.2

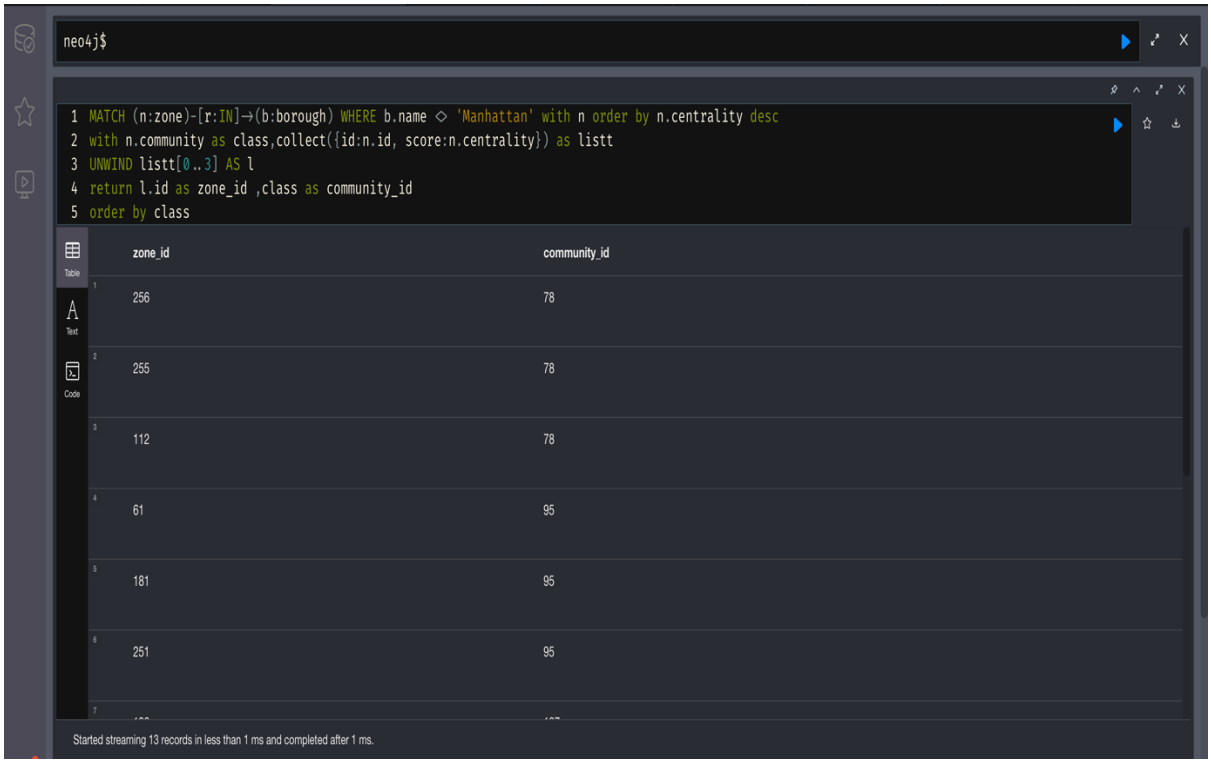

Figure 3.3

Figure 3.4

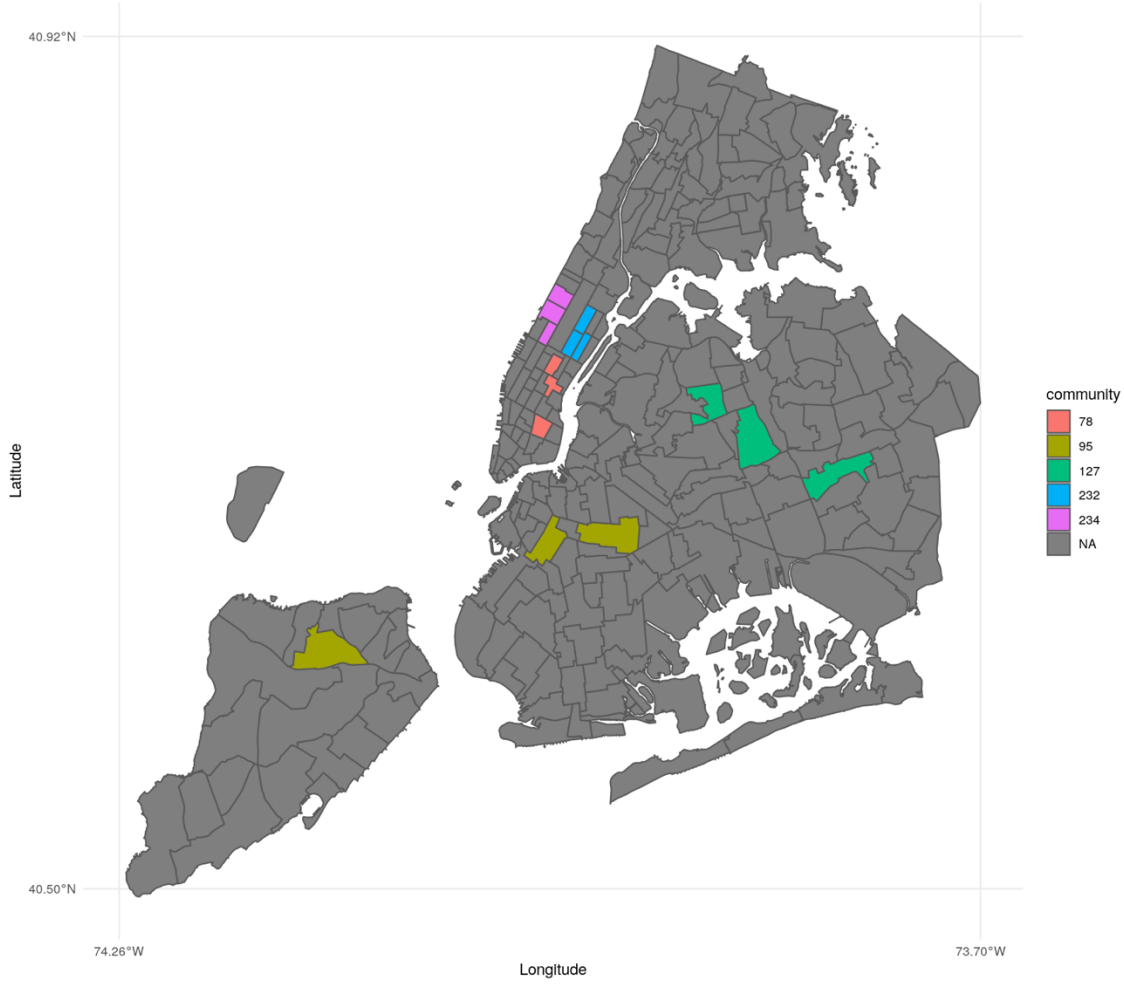
40.92°N
40.50°N
74.26°W
73.70°W
Latitude
Longitude
community
78
95
127
232
234
NA

Figure 3.5

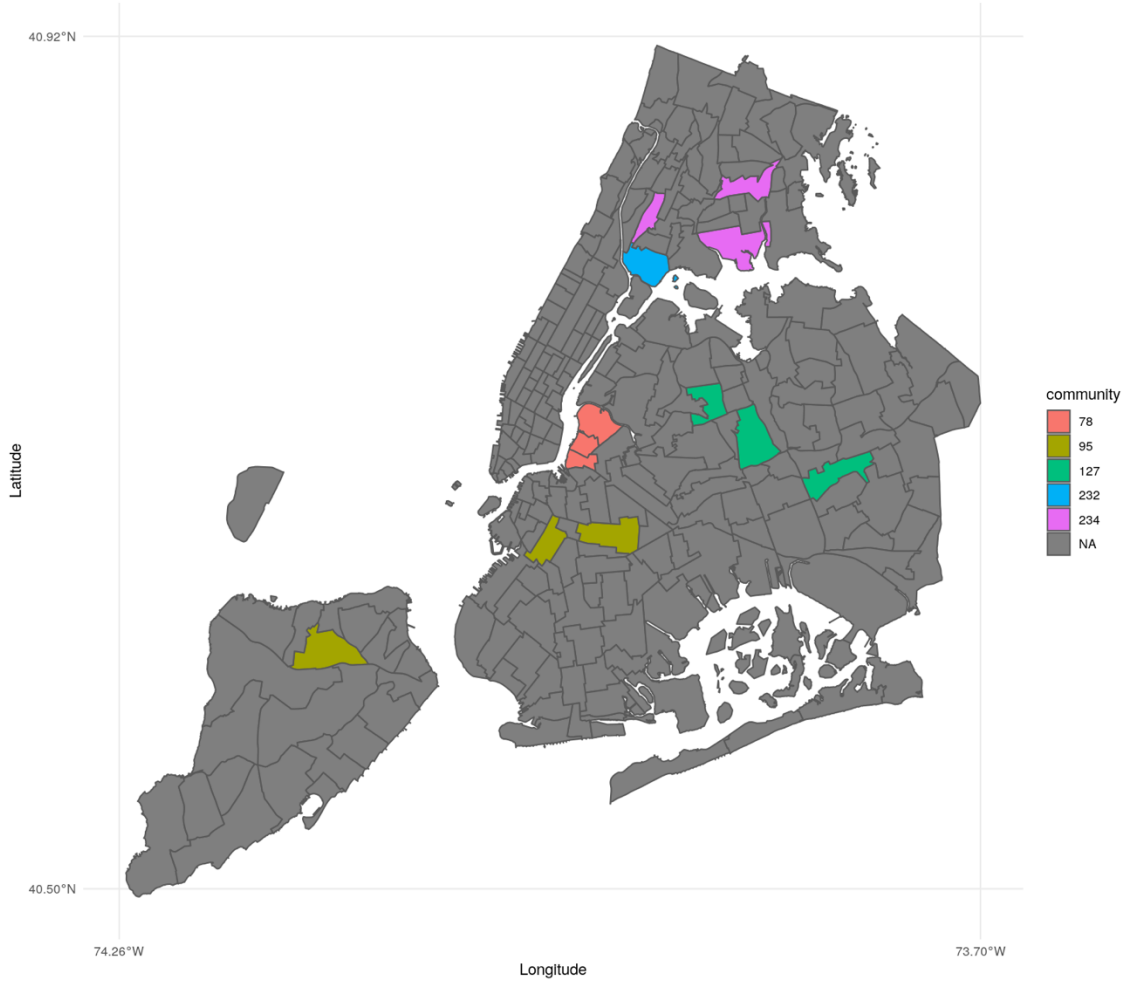
40.92°N
40.50°N
74.26°W
73.70°W
Latitude
Longitude
community
78
95
127
232
234
NA

Figure 3.6