ENGINEERING FOR AI

AKHIL RAJ RAJAN

210409183

NEWCASTLE UNIVERSITY

3 INTRODUCTION

- The aim is to analyse the performance of the spark-architecture based on scalability.
- The data sets selected for this task is NYC taxi data sets which contains the taxi trips details in New York city.
- The time taken for the execution of data sets whose size vary in different scale(3M-130M) is evaluated for understanding the system's scalability.

4 OVERVIEW OF THE TABLE STRUCTURE

PULocationID	DOLocationID	trip_distance	passenger_count	total_amount
90	68	0.8	1	8.8
113	90	0.9	1	8.8
88	232	2.8	1	13.8
79	249	1.4	1	12.3

Table : trips

LocationID	Borough	Zone	service_zone
1	EWR	Newark Airport	EWR
2	Queens	Jamaica Bay	Boro Zone
3	Bronx	Allerton/Pelham Gardens	Boro Zone
4	Manhattan	Alphabet City	Yellow Zone

Table: zone_names

6 TASK 1.1

- Removing records where trips distance is 0 and passenger count as well as the total amount is also 0.

```
def t11_remove_zeros(df):
    # input: trips dataset
    df=df.filter(~(df.trip_distance ==0))
    df=df.filter(~((df.passenger_count == 0) & (df.total_amount ==0)))
    df=df.select(['PULocationID', 'DOLocationID', 'trip_distance', 'passenger_count', 'total_amount'])
    # THE query will allow (passenger_count=0,total_amount>0) - delivery, (passenger_count>0,total_amount=0)-I dont know but not (passenger_count=0,total_amount=0)-invalid
    return df
```

7 TASK 1.2

- Removing outlier records.
- $z_i = \frac{x_i - median(x)}{MAD}$;
- $\text{MAD} = 1.483 * \text{median}(|x_i\text{-median(x)}|)$
- $|\ z_i\ |<3.5$

```
def t12_remove_outliers(df):
  median_total_amount=df.agg(pf.percentile_approx("total_amount", 0.5, pf.lit(1000))).collect()[0][0]
  median_trip_distance=df.agg(pf.percentile_approx("trip_distance", 0.5, pf.lit(1000))).collect()[0][0]
  delta_median_total_amount_abs=pf.abs(df.total_amount-median_total_amount)
  delta_median_trip_distance_abs=pf.abs(df.trip_distance-median_trip_distance)
  mad_total_amount=df.agg(pf.percentile_approx(delta_median_total_amount_abs, 0.5, pf.lit(1000))).collect()[0][0]*1.483
  mad_trip_distance=df.agg(pf.percentile_approx(delta_median_trip_distance_abs, 0.5, pf.lit(1000))).collect()[0][0]*1.483
  z_score_total_amount=((df.total_amount-median_total_amount)/mad_total_amount)
  z_score_trip_distance=((df.trip_distance-median_trip_distance)/mad_trip_distance)
  df=df.withColumn('z_score_total_amount',z_score_total_amount)
  df=df.withColumn('z_score_trip_distance',z_score_trip_distance)
  df=df.filter((pf.abs(df.z_score_total_amount)<3.5) & (pf.abs(df.z_score_trip_distance)<3.5))
  df=df.select(['PULocationID', 'DOLocationID', 'trip_distance', 'passenger_count', 'total_amount'])
  return df
```

8 TASK 2

- Finding Unit profitability

```
# Your solution implementation to task 2.1 goes HERE
def t21_join_zones(df, zones_df = zone_names):
    # input: output of task 1.2 and zone_names dataset
    k=df.alias('a').join(zones_df,(df.PULocationID==zones_df.LocationID),'left').select('a.*',zones_df.Zone.alias("start_zone"))
    df=k.alias('c').join(zones_df.alias("f"),(k.DOLocationID==zones_df.LocationID),'left').select('c.*',pf.col('f.Zone').alias("end_zone"))
    return df
```

```
# Your solution implementation to task 2.2 goes HERE
def t22_calc_profit(df):
    # input: output of task 2.1
    df=df.withColumn('unit_profitabilty',pf.when(df.trip_distance==0,0).otherwise(df.total_amount/df.trip_distance))
    return df
```

PULocationID	DOLocationID	trip_distance	passenger_count	total_amount	start_zone	end_zone	unit_profitabilty
90	68	0.8	1	8.8	Flatiron	East Chelsea	11
113	90	0.9	1	8.8	Greenwich Village North	Flatiron	9.777777778
88	232	2.8	1	13.8	Financial District South	Two Bridges/Seward Park	4.928571429

Code output

9 TASK 3

- This task will summarize the data to find out the average unit profitability, total trips and passenger linked to particular record.

```python
## Your solution to task 3.1 goes HERE
def t31_summarise_trips(df):
    # input: output of task 2.2
    df=df.groupby(df.PULocationID,df.DOLocationID).agg(pf.avg(df.unit_profitabilty).alias('average_unit_profit'),pf.count('*').alias('trips_count'),pf.sum(df.passenger_count).alias('total_passengers'))
    return df
```

10

```
def t32_summarise_zones_pairs(df, zones_df = zone_names):
    df=df.alias('a').join(zone_names.alias('b'),(df.PULocationID==zone_names.LocationID),'left').select('a.*','b.Zone')
    df=df.alias('a').groupby(df.Zone).agg(pf.sum(df.trips_count).alias('total_trips_count'),pf.sum(df.total_passengers).alias('total_passengers'),
                                          pf.avg(df.average_unit_profit).alias('average_unit_profit'))
    return df

# Top 10 ranked zones by traffic (trip volume)
def t32_top10_trips(df_zones):
    # input: output of task 3.2
    df_zones=df_zones.sort(df_zones.total_trips_count.desc()).take(10)
    return df_zones

# Top 10 ranked zones by profit
def t32_top10_profit(df_zones):
    # input: output of task 3.2
    df_zones=df_zones.sort(df_zones.average_unit_profit.desc()).take(10)
    return df_zones

# Top 10 ranked zones by passenger volume
def t32_top10_passenger(df_zones):
    # input: output of task 3.2
    df_zones=df_zones.sort(df_zones.total_passengers.desc()).take(10)
    return df_zones
```

12 ZONES HAVING TOP 10 AVERAGE UNIT PROFIT

Zone	total_trips_count	total_passengers	average_unit_profit
Bay Terrace/Fort Totten	49	5	311.9330684
Baisley Park	320	101	100.0059271
Prospect Park	74	64	90.58902851
Saint Michaels Cemetery/Woodside	29	29	66.14679218
Van Cortlandt Village	205	48	48.6905998
Grymes Hill/Clifton	2	2	38.65
Crotona Park	7	3	36.84437138
Flatbush/Ditmas Park	681	118	32.94031965
Eastchester	147	16	31.20815779
Flushing Meadows-Corona Park	53	61	29.53958084

14 RESULT AND ANALYSIS

metric	S	M	L	XL	XXL
rows (M)	2,898,033	15,571,166	41,953,716	90,443,069	132,396,785
execution time (w/o 1.2)	7.19s	1.53m	13.63m	18.26m	19.21m
execution time	15.21s	3.14m	30.63m	40.62m	41.69m
sec / 1M records (w/o 1.2)	2.48	5.89s	19.52s	12.11s	8.70s
sec / 1M records	5.24	12.09s	43.80s	26.95s	18.89s

Parquet file type

15

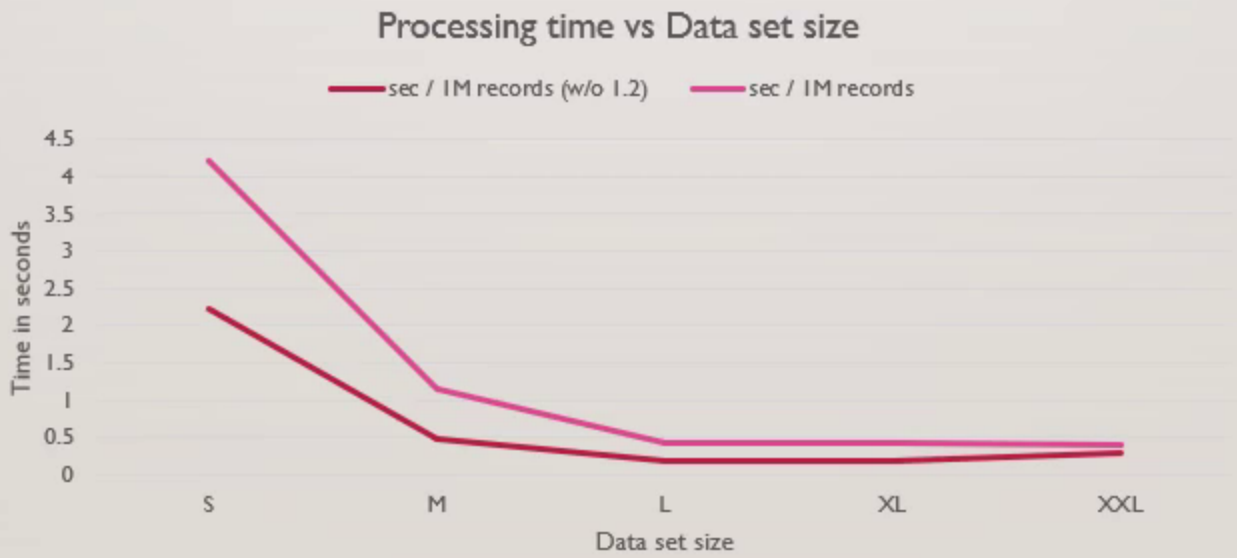
metric	S	M	L	XL	XXL
rows (M)	2,898,033	15,571,166	41,953,716	90,443,069	132,396,785
execution time (w/o 1.2)	6.44s	7.58s	8.74s	18.60s	39.44s
execution time	12.20s	18.05s	18.79s	39.45s	51.89s
sec / 1M records (w/o 1.2)	2.22s	0.486s	0.20s	0.20s	0.29s
sec / 1M records	4.20s	1.15s	0.44s	0.44s	0.39s

Delta file type

16 TIME TAKEN FOR PROCESSING DIFFERENT SIZE OF DATA FOR DELTA FILE.

17 TIME TAKEN FOR THE DATA SETS TO PROCESS 1MILLION RECORDS FOR DELTA FILE TYPE.

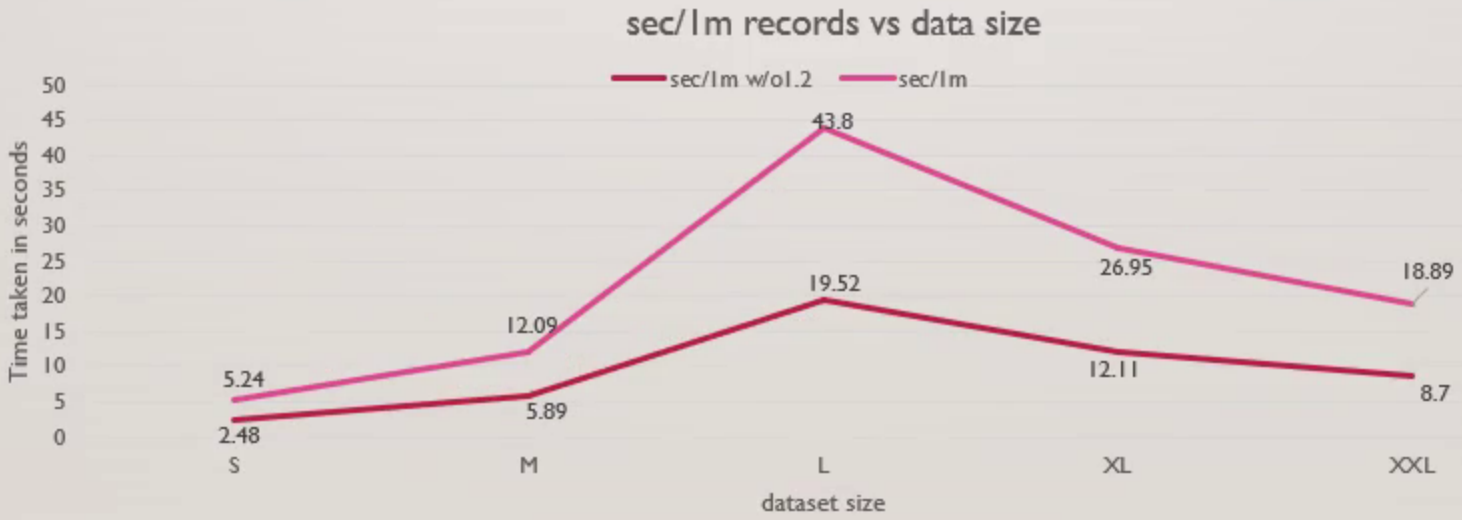
S	M	L	XL	XXL
2,898,033	15,571,166	41,953,716	90,443,069	132,396,785



18 TIME TAKEN FOR PROCESSING DIFFERENT SIZE OF DATA FOR PARQUET FILE.

TIME TAKEN FOR THE DATA SETS TO PROCESS 1MILLION RECORDS FOR PARQUET.

S	M	L	XL	XXL
2,898,033	15,571,166	41,953,716	90,443,069	132,396,785



20 CONCLUSION

- As size increases processing time also increases.
- Delta type data execute faster than parquet data type. Also delta files execution time is in scale of seconds but at the same time parquet is in minutes.
- For delta file, the value of sec/1 m data is minimum for L and XL sizes, that is when the size is between 40 M and 90 M
- For parquet file, the value of sec/1 m data is maximum for size L, whose size is around 40M.
- As expected, the execution time is lower without executing the task 1.2.

PART B

Graph Data Science assignment

22 INTRODUCTION

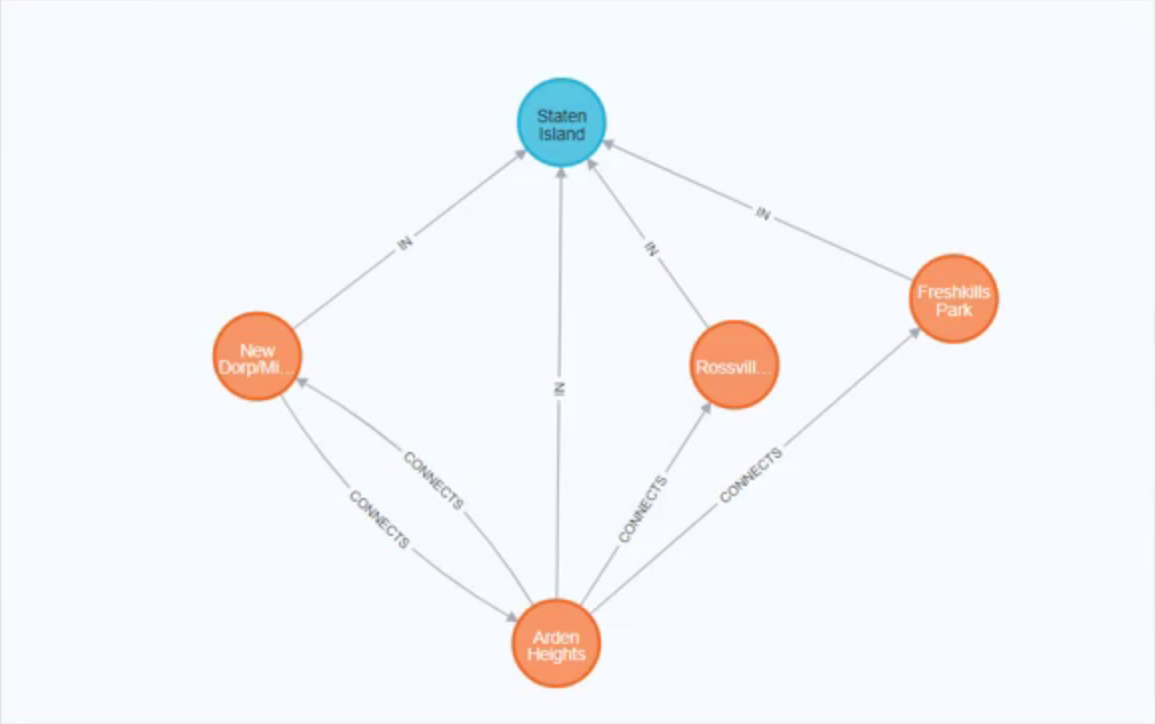
- The aim is to choose the zones having hub like characteristics for minimizing the cost under this conditions.

a. maximum 20 zone spread across the city and maximum trip served

b. Considers operating outside Manhattan city.

- by understanding the tools and environment used for Graph programming using neo4j.
- The different algorithm used for finding the centrality and detection of community is analysed in this assignment.
- The data set contains aggregate statistics of taxi trips undertaken in 2021.

23 GRAPH STRUCTURE



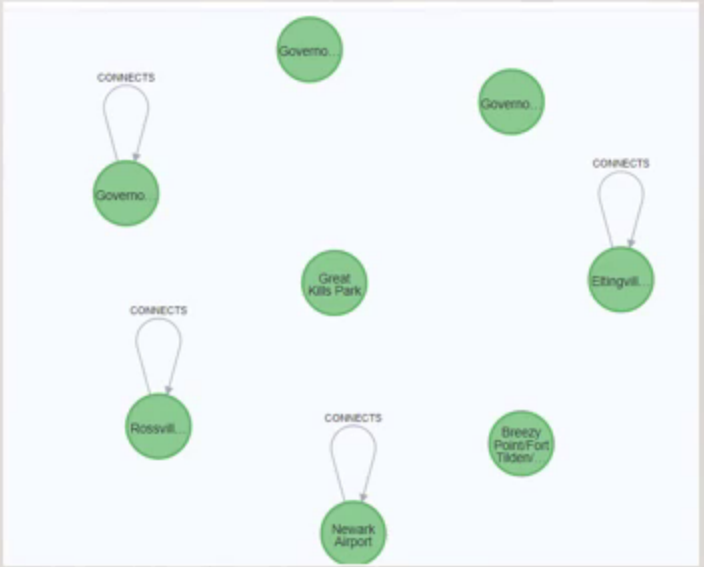
24 TASK 0

- Self pointing node that is node having relation to itself
- Cql :-*match (z:zone)-[r:CONNECTS]->(z1:zone) where z.id=z1.id return z.id*
- Isolated node that is the node which does not having any connection to a different node
- Cql:-

MATCH (z:zone)WHERE NOT

EXISTS { match (z:zone)-[r:CONNECTS]->(z_:zone)

where not z=z_}RETURN z



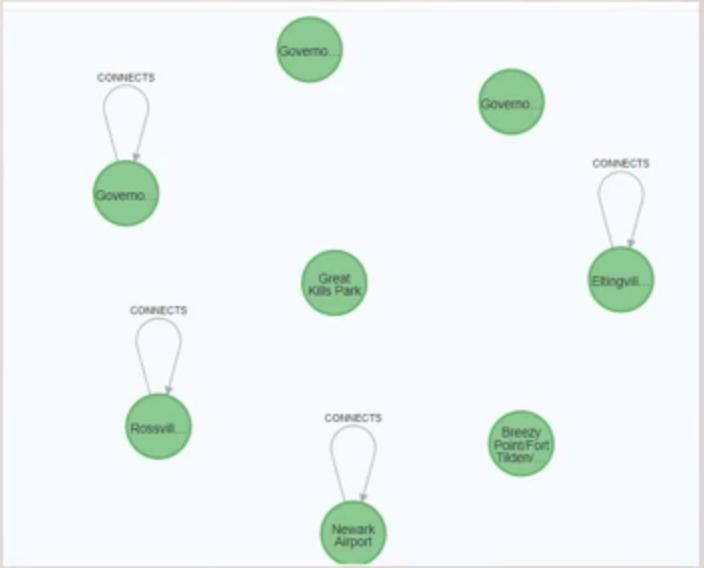
24 TASK 0

- Self pointing node that is node having relation to itself
- Cql :-*match (z:zone)-[r:CONNECTS]->(z1:zone) where z.id=z1.id return z.id*
- Isolated node that is the node which does not having any connection to a different node
- Cql:-

MATCH (z:zone)WHERE NOT

EXISTS { match (z:zone)-[r:CONNECTS]->(z_:zone)

where not z=z_}RETURN z



25 TASK 1

- Finding the community detection.
- Community detection helps to reveal the hidden clustering with in the network.
- It uses Louvain algorithm
- Cql:- graph projection code given

```
call gds.graph.create(
    'c1040918-communities',
    'zone',
    {
        CONNECTS:{
        orientation:'undirected'
        }
    },
    {
        relationshipProperties:'trips'
    }
)
```

27 ZONES AND COMMUNITY

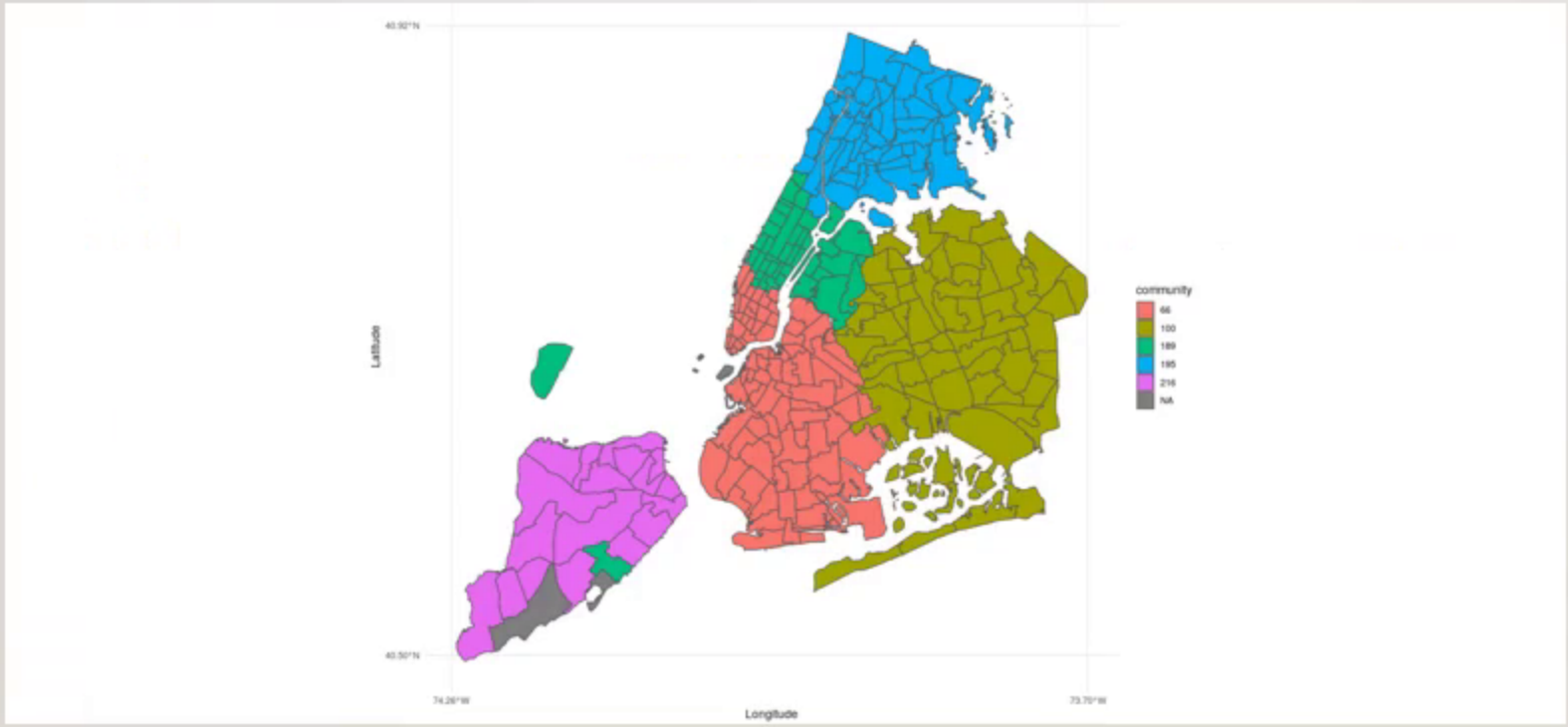
- The community can be identified by using the community id generated in the below cql.

```
1 CALL gds.louvain.stream('c1040918-communities')
2 YIELD nodeId,communityId
3 RETURN gds.util.asNode(nodeId).id as zone_id,communityId as community_id
```

Table | Text | Code

	zone_id	community_id
1	3	195
2	4	66

28 MAP SHOWING COMMUNITY



29 TASK 2

- Finding the centrality of the zone using page rank algorithm
- Cql for graph projection:

```
call gds.graph.create(
    'c1040918-centrality',
    'zone',
    {
        CONNECTS:{
        orientation:'natural'
        }
    },
    {
        relationshipProperties:'trips'
    }
)
```

30 CENTRALITY USING STATS MODE

```
CALL gds.pageRank.stats('c1040918-centrality', {
    dampingFactor: 0.75
})
YIELD centralityDistribution
RETURN centralityDistribution.max AS max,centralityDistribution.min as min
```

Table

"max"	"min"
1.6610088348388672	0.25

Text

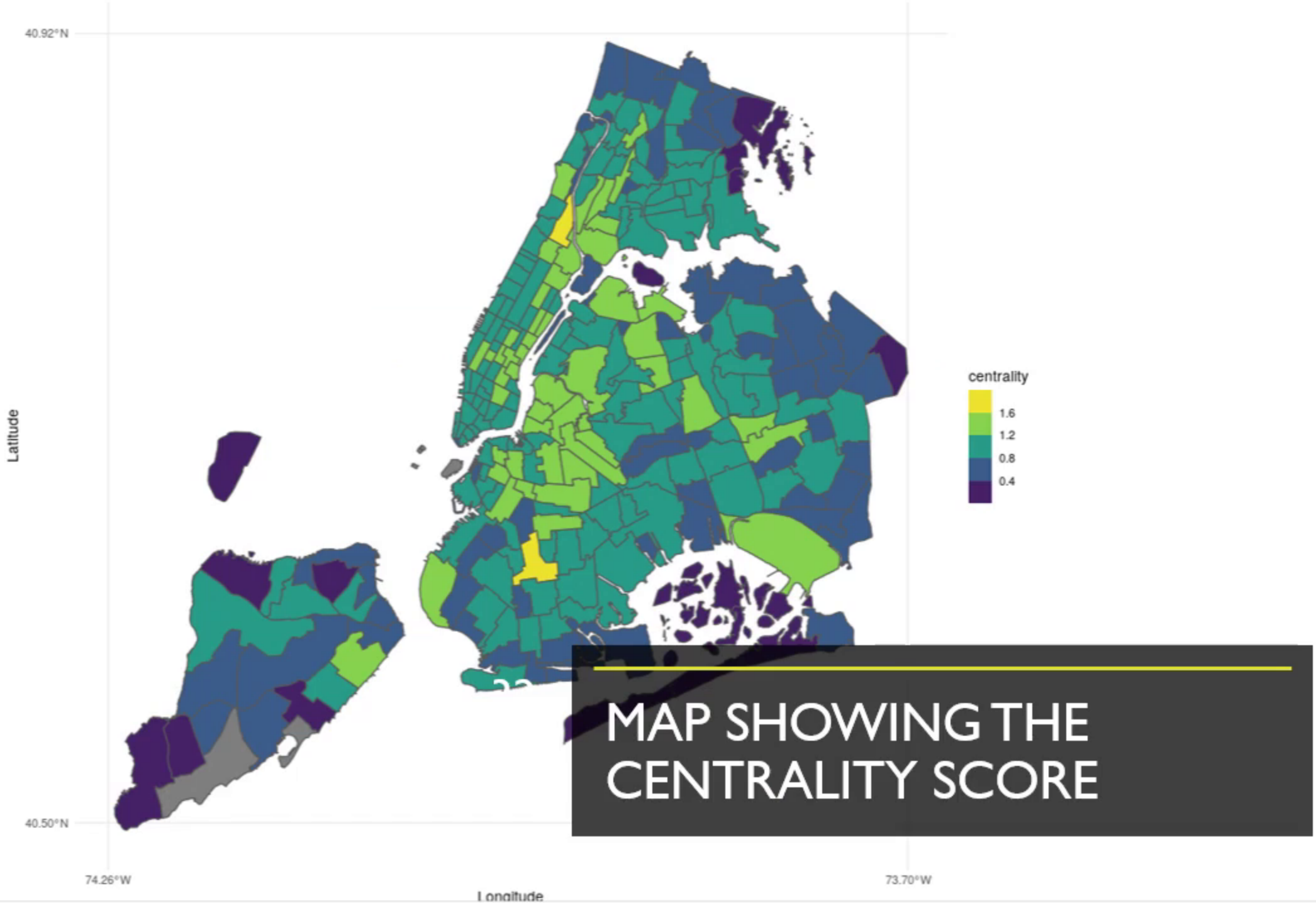
31 STREAM MODE CQL QUERY

```
1 CALL gds.pageRank.stream('c1040918-centrality')
2 YIELD nodeId, score
3 RETURN gds.util.asNode(nodeId).id AS zone_id, score as centrality_score
4 ORDER BY zone_id asc
```

Table

Text

"zone_id"	"centrality_score"
1	0.17989397409363472
2	0.21453017356802942



MAP SHOWING THE CENTRALITY SCORE

33 TASK 3

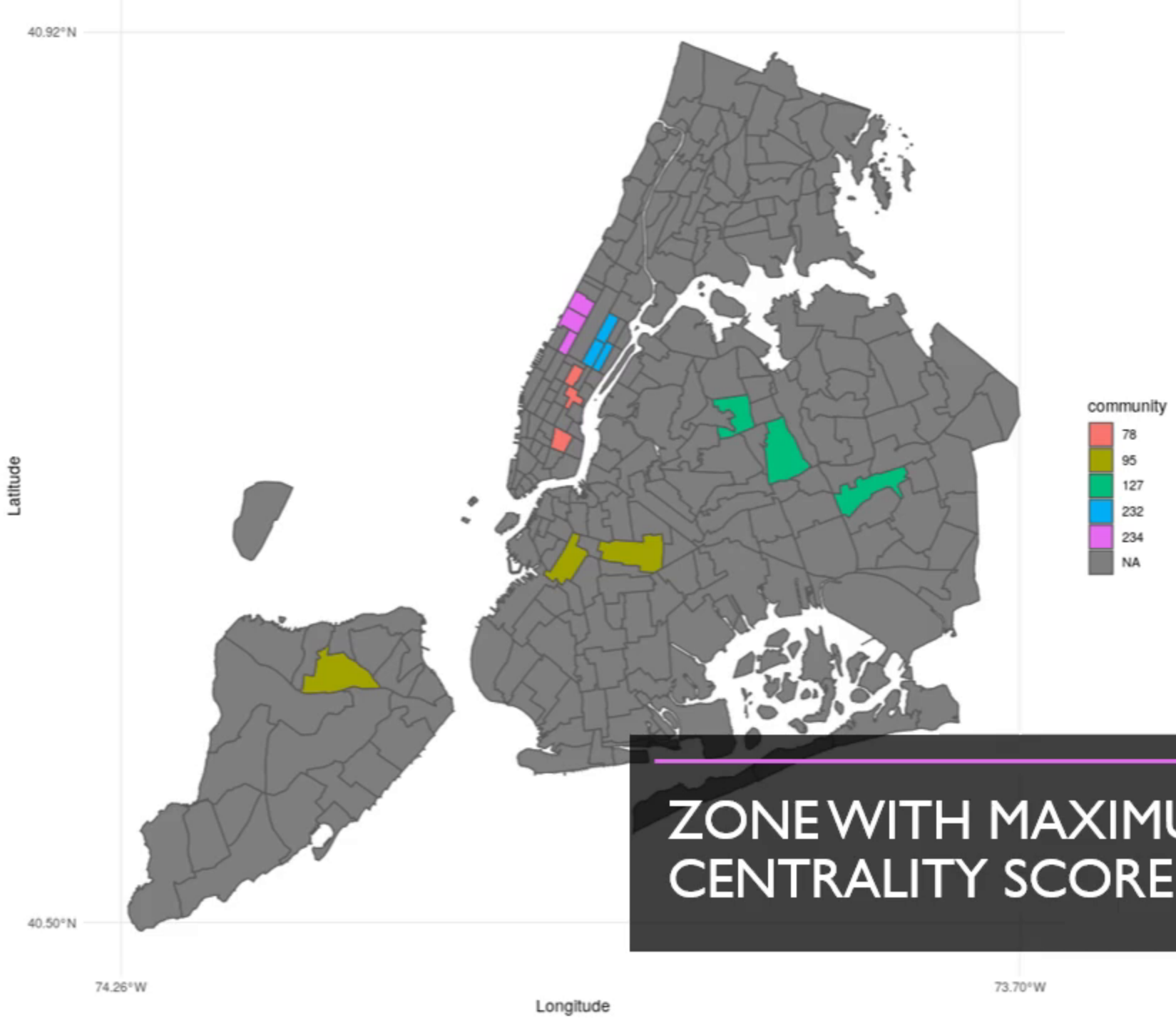
- Finding the top 3 zones having the maximum centrality for each community detected in the previous steps.
- Cql :

```
MATCH (n:zone)
WITH n.community AS group, n.centrality AS value,n.id as zone_id
ORDER BY group, value desc
WITH group, COLLECT(value) AS values
with group, values[0..3] AS top_3_values
unwind top_3_values as top_3
with group as community_id, top_3
match (z1:zone) where z1.centrality=top_3
return z1.id as zone_id,z1.community as community_id

```

Table

	zone_id	community_id
1	170	78



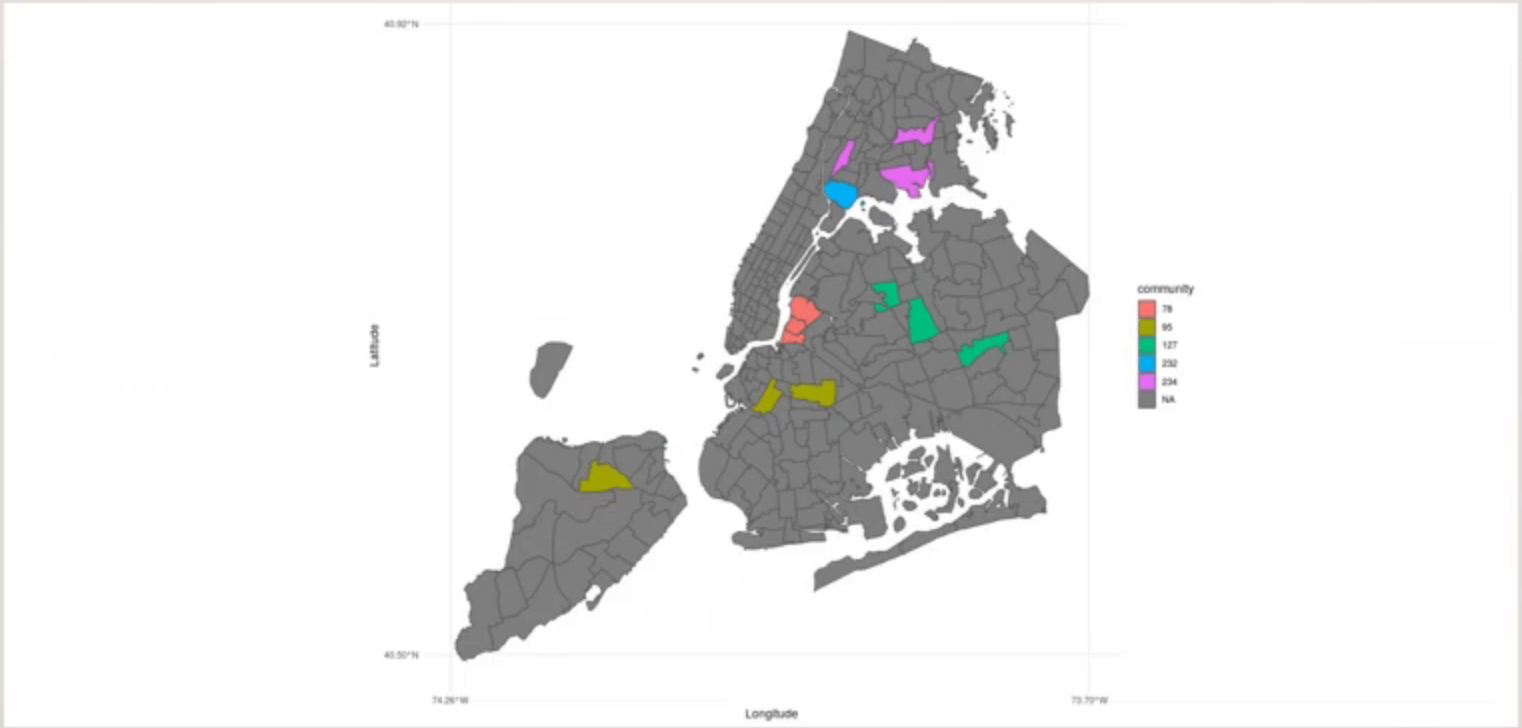
ZONE WITH MAXIMUM CENTRALITY SCORE

35 TASK 3

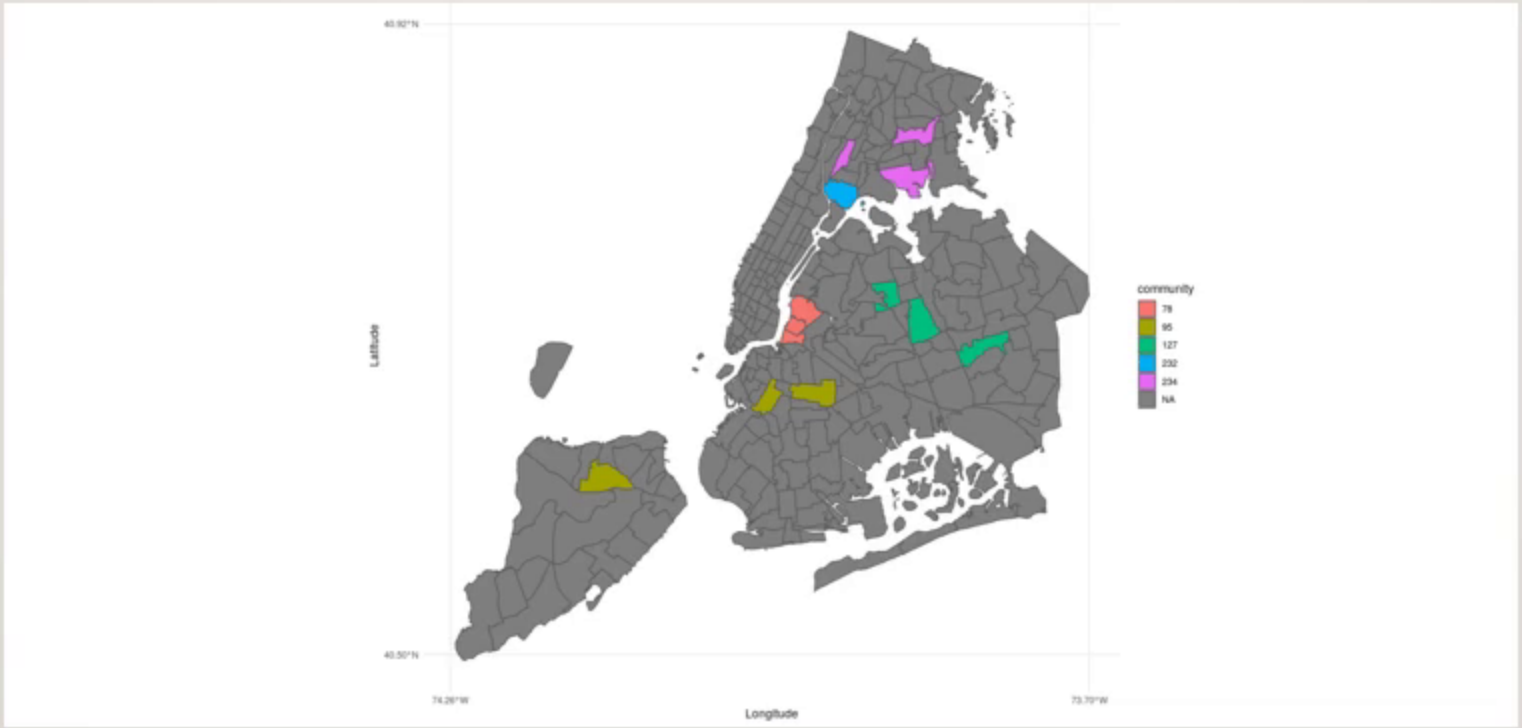
- Finding the top 3 zones having the maximum centrality for each community detected in the previous steps which is not in Manhattan borough.
- Cql :

```
match (n:zone)-[r:IN]→(b:borough) where b.name◇'Manhattan'
WITH n.community AS group, n.centrality AS value,n.id as zone_id
ORDER BY group, value desc
WITH group, COLLECT(value) AS values
with group, values[0..3] AS top_3_values
unwind top_3_values as top_3
with group as community_id, top_3
match (z1:zone) where z1.centrality=top_3
return z1.id as zone_id,z1.community as community_id
```

36 ZONE WITH MAXIMUM CENTRALITY SCORE EXCLUDING MANHATTAN BOROUGH



36 ZONE WITH MAXIMUM CENTRALITY SCORE EXCLUDING MANHATTAN BOROUGH



37 CONCLUSION

- Zones with ids 256,255,112,61,181,251,130,82,95,168,213,69,242 are the best zones where the company can focus their operation if Manhattan borough is ignored.
- The best zones where the company can focus their operation if all the boroughs are considered are the zone with ids 170,161,79,61,181,251,130,82,95,236,237,141,238,239,142