DISCRETE MATHEMATICAL STRUCTURES[MAT1003]

PROJECT REPORT

SLOT:F1+TF1
GROUP 5

ROAD CONSTRUCTION ANALYZER

Submitted By:

Dasari Mohan Amritesh 18bcn7038

Sharaj Raja Chandran 18bcd7044

Kathal Aditya Rajendra 18bcd7008

N.M Keerthiga 18bce7047

Arushi Jha 18bce7007

Viswanadha Vedvyas 18bce7190

Guided By:

Dr. Santanu Mandal

INDEX

ABSTRACT

- The ability to predict the final result of construction projects based on limited initial input data could be a very valuable tool for every project manager and/or construction enterprise.
- India is a country with the second largest road network in the world. Out of the total stretch of 5.4 million km of road network, almost 97,991 km is covered by national highways.
- It is already a huge challenge for the Indian government to provide world-class roads, due to the sheer magnitude.
- On an average, a person spends anywhere between 30 minutes to two hours of their day driving. Which means, in a year, it is almost 360 hours.
- If India has to maintain its growth, it will require around 15,000 km of new expressways in the coming 10-12 years.
- Keeping in mind the scale at which road infrastructure is required to grow; a prior analysis on where new roads should be paved will hugely reduce the pre-planning time and logistics required.
- This project will help analyze traffic intensity and flow; enabling us to predict where to construct new roads resulting in reduced traffic overload, lower travel time and sustainable infrastructure.

METHODOLOGY

- Data collection:
  - Dataset 1:
    - The whole surveyed road network of India
    - Pickup/Source and Dropoff/Destination locations (Latitude and Longitude)

- Dataset 2:
    - New York cab dataset to simulate traffic intensity through a day.
- Designing an algorithm to analyze,simulate traffic flow and intensity before and after the addition of a new road(s).
- Implementing the algorithm on the collected data.
- Concluding whether the proposed road should be constructed or not based on the analysis.

DATASET

Train datasets have been used.

DATASET 1:

	PickUp_Latiude	Pickup_Longitude	Drop_Latitude	Drop_Longitude
2	14.43333	79.96667	25.35	74.63333
3	26.22361	78.17917	23.21667	72.68333
4	16.7	74.21667	10.98333	77.3
5	11.35	77.73333	25.6	85.11667
6	17.68333	75.91667	26.85	80.91667
7	26.85	80.91667	20.66667	85.6
8	27.5	77.68333	26.46667	80.35
9	23.31667	75.06667	14.23333	76.4
10	16.85438	74.56417	19.98333	73.8
11	28.35	79.41667	16.51667	80.61667
12	13.65	79.41667	25.18333	75.83333
13	15.86667	74.5	23.83333	78.71667
14	18.51957	73.85535	10.80289	78.69875
15	21.01667	75.56667	22.56667	72.93333
16	15.2	76.68333	30.32295	78.03168
17	26.61667	81.36667	22.3	73.2
18	15.48333	73.83333	19.95	79.3
19	15.15	76.93333	25.6	85.11667
20	25.45	81.85	25.6	85.11667
21	28.66667	77.43333	23.03333	72.61667
22	16.51667	80.61667	26.16667	75.78333
23	31.32556	75.57917	23.6	72.95
24	23.16697	79.95006	21.7	72.96667
25	28.66667	77.43333	17.68333	75.91667
26	22.56972	88.36972	19.01441	72.84794
27	29.68333	76.98333	24.58333	80.83333
28	23.21667	72.68333	13.08784	80.27847
29	24.7	84.98333	25.18333	75.83333
30	26.61667	81.36667	20.63333	72.93333
31	22.6	75.3	20.9	74.78333
32	28.83333	78.78333	15.35	75.16667
33	8.48333	78.11667	19.98333	73.8
34	24.7	84.98333	25.45	81.85

- The dataset used contains Pickup and Drop Latitude and Longitudes.
- The data has 4 columns and rows.
- Using these values the whole roadway network has been plotted and visualized.

DATASET 2:

M7

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	id	vendor_id	pickup_da	dropoff_datetir	passenger_c	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	store_and_fwd_flag	trip_duration		
2	id2875421	2	########	############	1	-73.98215485	40.76793671	-73.96463013	40.76560211	N	455		
3	id2377394	1	########	############	1	-73.98041534	40.73856354	-73.9994812	40.73115158	N	663		
4	id3858529	2	########	############	1	-73.97902679	40.7639389	-74.00533295	40.71008682	N	2124		
5	id3504673	2	########	############	1	-74.01004028	40.7199707	-74.01226807	40.70671844	N	429		
6	id2181028	2	########	############	1	-73.97305298	40.79320908	-73.97292328	40.78252029	N	435		
7	id0801584	2	########	############	6	-73.98285675	40.74219513	-73.99208069	40.74918365	N	443		
8	id1813257	1	########	############	4	-73.96901703	40.7578392	-73.95740509	40.76589584	N	341		
9	id1324603	2	########	############	1	-73.96927643	40.79777908	-73.92247009	40.76055908	N	1551		
10	id1301050	1	########	############	1	-73.9994812	40.73839951	-73.98578644	40.73281479	N	255		
11	id0012891	2	########	############	1	-73.98104858	40.74433899	-73.97299957	40.78998947	N	1225		
12	id1436371	2	########	############	1	-73.98265076	40.76383972	-74.00222778	40.73299026	N	1274		
13	id1299289	2	########	############	4	-73.99153137	40.74943924	-73.95654297	40.77062988	N	1128		
14	id1187965	2	########	############	2	-73.96298218	40.75667953	-73.98440552	40.7607193	N	1114		
15	id0799785	2	########	############	1	-73.95630646	40.76794052	-73.96611023	40.76300049	N	260		
16	id2900608	2	########	############	1	-73.99219513	40.72722626	-73.97465515	40.78306961	N	1414		
17	id3319787	1	########	############	1	-73.955513	40.76859283	-73.94876099	40.77154541	N	211		
18	id3379579	2	########	############	1	-73.99116516	40.75556183	-73.99929047	40.72535324	N	2316		
19	id1154431	1	########	############	1	-73.99425507	40.74580383	-73.99965668	40.7233429	N	731		
20	id3552682	1	########	############	1	-74.00398254	40.7130127	-73.97919464	40.74992371	N	1317		
21	id3390316	2	########	############	1	-73.98388672	40.73819733	-73.99120331	40.72787094	N	251		
22	id2070428	1	########	############	1	-73.98036957	40.7424202	-73.96285248	40.76063538	N	486		
23	id0809232	2	########	############	1	-73.97953796	40.75336075	-73.96399689	40.76345825	N	652		
24	id2352683	1	########	4/9/2016 3:41	1	-73.99586487	40.75881195	-73.99332428	40.74032211	N	423		
25	id1603037	1	########	############	1	-73.99355316	40.74717331	-74.00614166	40.70438385	N	1163		
26	id3321406	2	########	6/3/2016 8:56	1	-73.95523071	40.77713394	-73.78874969	40.64147186	N	2485		
27	id0129640	2	########	############	1	-73.95658112	40.77135849	-73.97496796	40.7327919	N	1283		
28	id3587298	1	########	############	1	-73.98376465	40.74987411	-73.95883179	40.80096054	N	1130		
29	id2104175	1	########	############	1	-73.95843506	40.71319199	-73.94953918	40.68025208	N	694		
30	id3973319	2	########	############	1	-73.99421692	40.71330643	-73.98284912	40.69229889	N	892		
31	id1410897	1	########	############	1	-73.9821167	40.75635147	-73.86569214	40.77098846	N	2331		
32	id0266980	1	########	############	1	-73.97093964	40.76166916	-73.99623871	40.7253685	N	1479		
33	id2822549	2	########	############	1	-73.99826813	40.72003937	-74.01036072	40.67213821	N	1048		
34	id3075902	2	########	############	2	-74.00550079	40.70658875	-73.98581696	40.74420166	N	1022		

- We have used this cab ride data for New York to visualize the traffic flow in the city through the day.
- The dataset contains 11 columns and 1458644 rows.

- This is a **time series** data.
- Out of the 11 only 6 columns have been used. These are pickup and dropoff date and time, pickup and dropoff latitudes and longitudes.

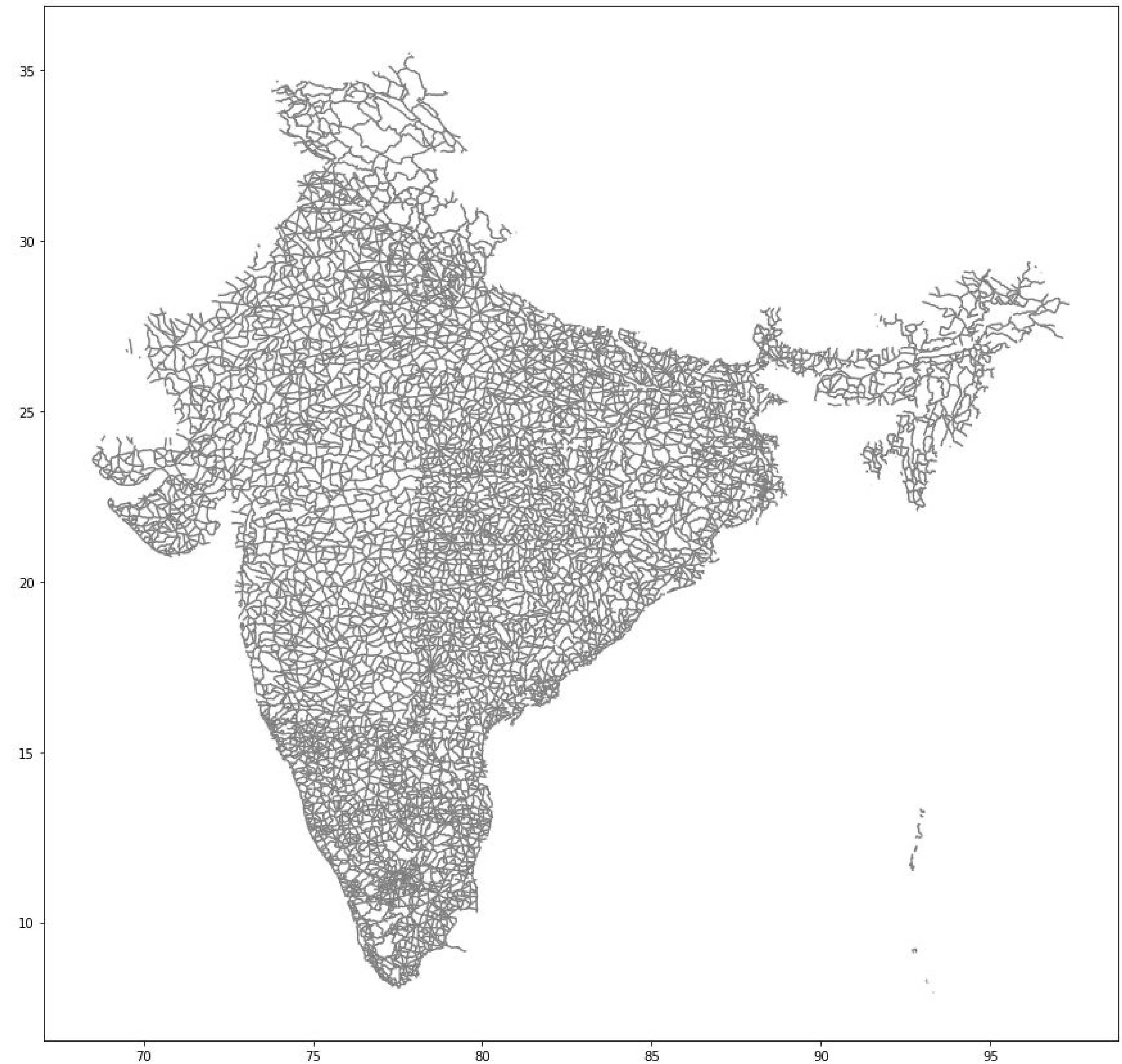
INTRODUCTION

- Road transportation sector plays a crucial role in accessing the growth of any country.
- India is a country with the second largest road network in the world. Out of the total stretch of 5.4 million km of road network, almost 97,991 km is covered by national highways.
- Though enormous types of roads are available in India, low volume village roads connecting small villages with each other and also with other categories of roads play a vital role towards the economic growth of the country as such class of roads establishes direct linkages with agricultural and production sectors.
- Even though the Government of India (GoI) is investing huge amounts of money on road construction every year, poor control over the quality of road construction and its subsequent maintenance is leading to the faster road deterioration.
- Many roads after pavement are either deserted or rarely used; failing to give a substantial return for the put resources and investment.
- During the implementation of a construction project, the meticulous "Project Evaluation" at the beginning loses much of its importance during latter stages because of various uncertainties related to the environment resulting in varying degrees of cost overruns.
- In this regard, it is essential that scientific procedures are evolved for predicting if the construction of a new road will be beneficial in traffic diversion and traffic intensity reduction .
- This will ensure resources are allocated and utilised highly productively with high returns.
- Even though many scientific models are available for assessing the performance of flexible pavements, cost estimation, maintenance scheduling etc. very few exist if any to determine if a new construction is even required.
- This project aims to fill that gap and develop a prediction and analysis model for the same.

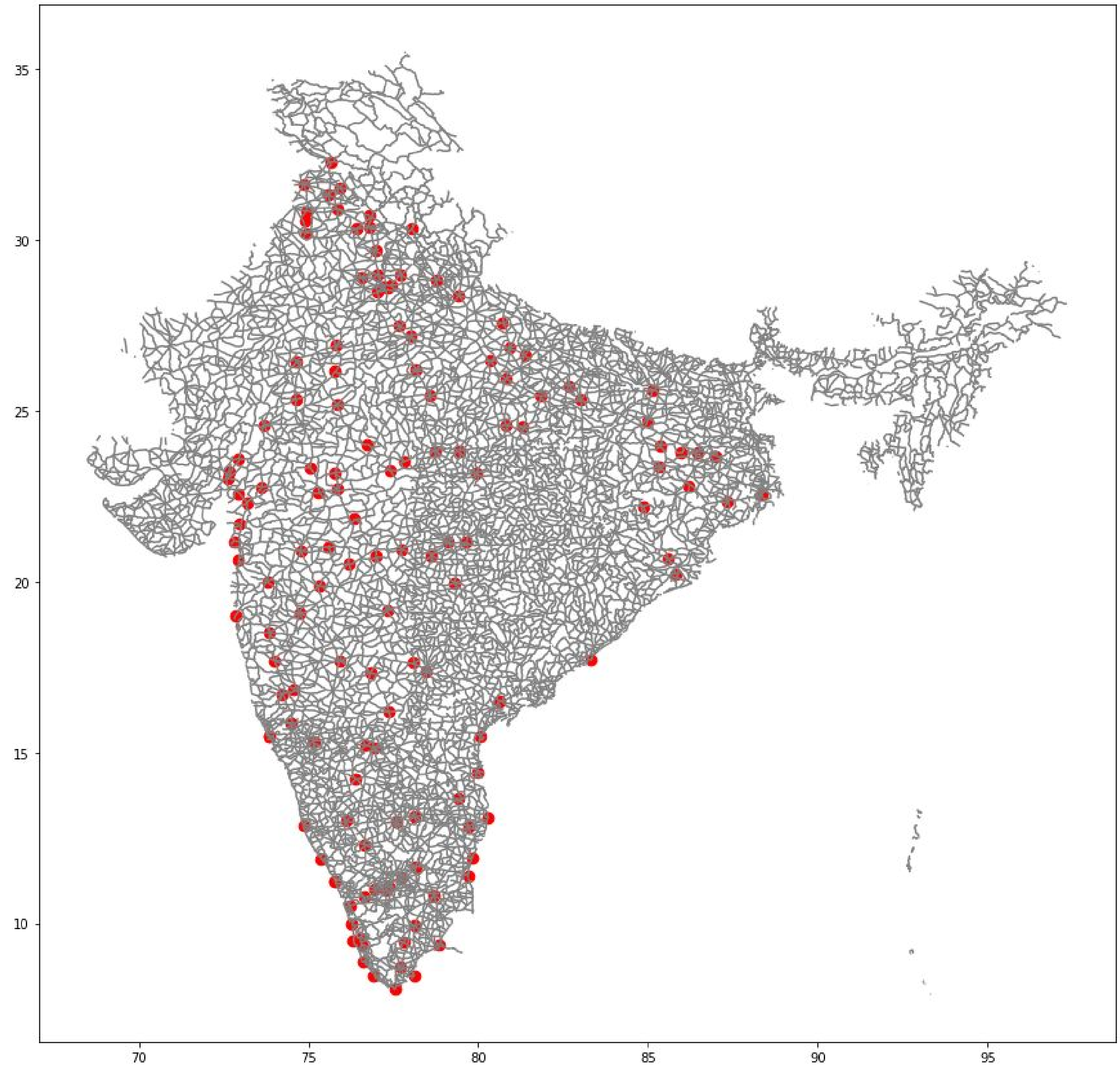
ANALYSIS and VISUALIZATION

DATASET 1:

- Python has been used to import, visualize and analyze the data.
- The concepts of **GRAPHS and DIJKSTRA'S ALGORITHM** have been used.

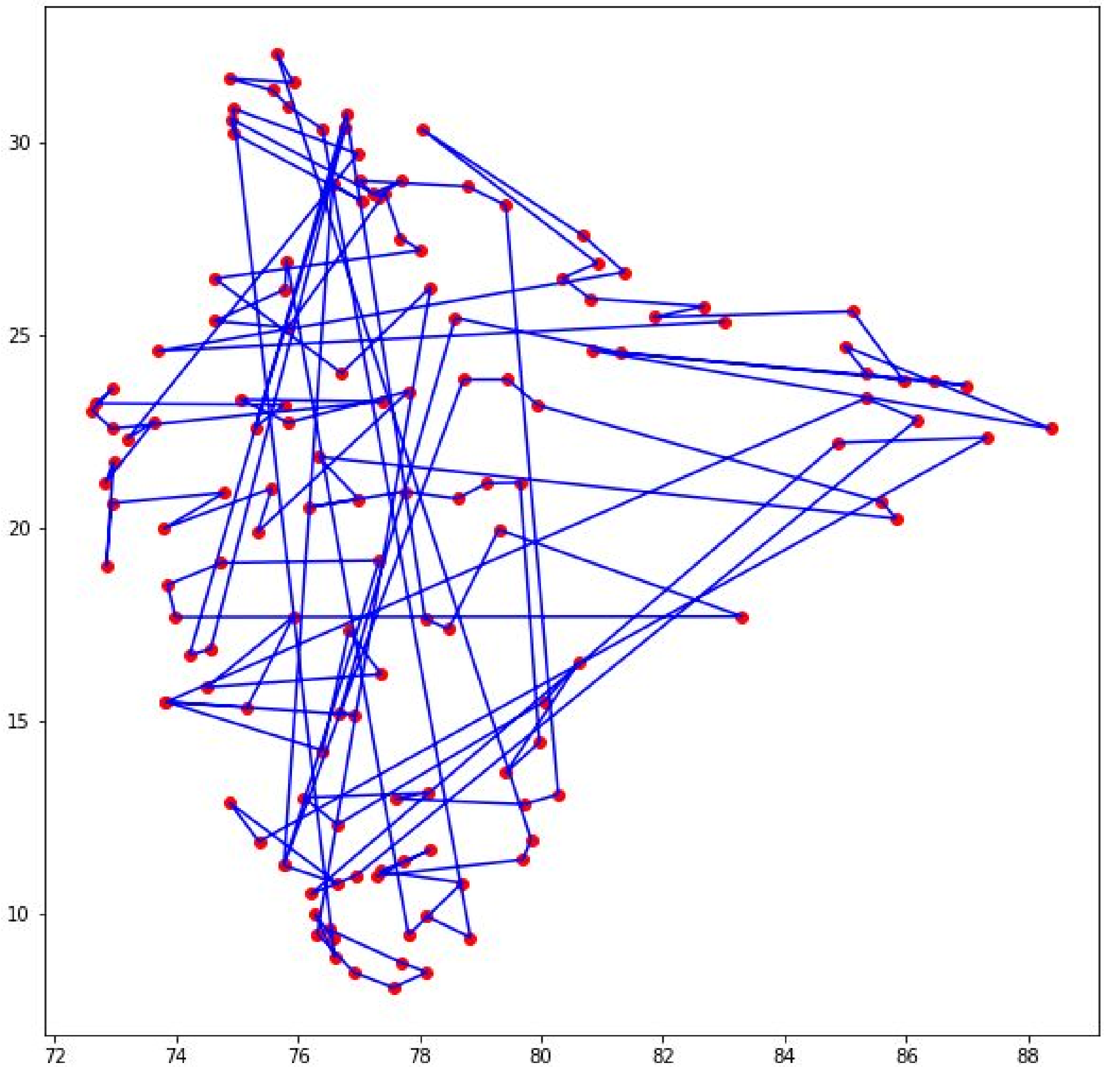


- After importing all the required libraries and packages the graph is read into the code.
- It consists of the whole surveyed road network of India as in the above figure.
- The southern and central regions have a denser network due to higher available data for those regions (more extensively surveyed).

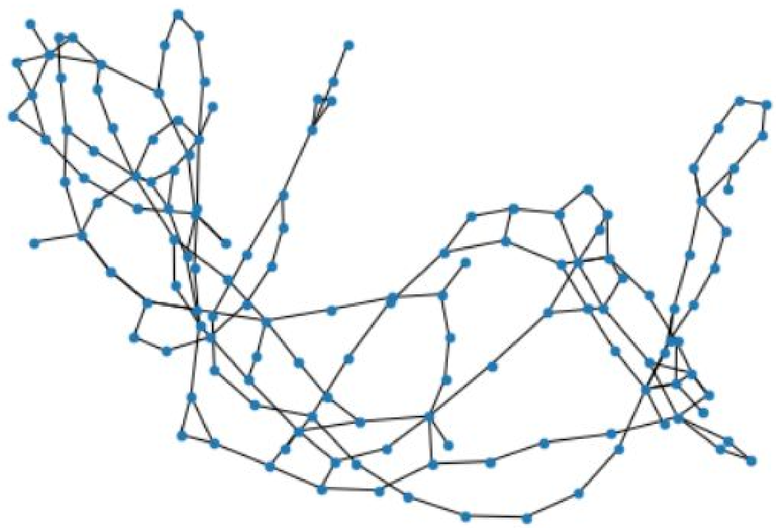


- The points where two or more roads intersect make a **NODE.**
- For simplicity out of the many nodes in the dataset, 144 nodes with the **maximum in or out degree** and the most clustered points are chosen.
- These chosen nodes have been plotted in red.

```
Out[7]: [<matplotlib.lines.Line2D at 0x1fcf83f4c50>]
```



- After obtaining the nodes the whole map was converted to a **graph**.
- The red dots represent the intersections whereas the blue lines represent the connecting roads.



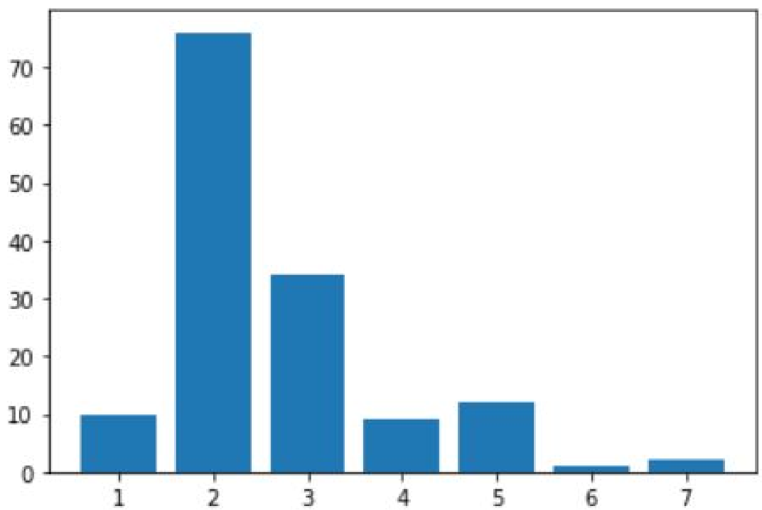
- A graph **G = (V, E)** consists of a set of **nodes** V (or vertices, points) and a set of **edges** E (links, lines) which illustrate how the nodes in the network are interacting with each other. Edges can be **directed** or **undirected**.
- This graph contains undirected edges.
- Using the networkx library the schematic has been converted into a proper **network graph** with nodes and edges for analysis.

```
Name:
Type: MultiGraph
Number of nodes: 144
Number of edges: 190
Average degree:   2.6389

 After Removingthe cyclicity:

Name:
Type: Graph
Number of nodes: 144
Number of edges: 182
Average degree:   2.5278
```

- The properties of the graph have been calculated before and after removing the cyclicity.
- Average degree = (No. of in degree + No. of out degree)/Total
- Removing cyclicity => removing the edges due to which cyclicity exists.
- This ensures the shortest paths between nodes are used for analysis.

- Clearly, after the cyclicity is removed the number of edges decrease and so does the average degree; proving the existence and removal of cyclicity.

```
Out[10]: <BarContainer object of 7 artists>
```



- This is a visual representation of the various degrees of nodes present, i.e. number of nodes/degree.
- A **degree** of a node stands for the number of edges it has to other nodes.

```
Density :
0.018453768453768452
```

- Then we calculate the density of the graph.
- Graph density = no of edges/total no of possible edges.
- The density is 0 for a graph without edges and 1 for a complete graph. The density of multigraphs can be higher than 1.
- Self loops are counted in the total number of edges so graphs with self loops can have density higher than 1.
- Having a density of 0.01845 makes sense because in a street network not all nodes can be connected to all other nodes.

```
The longest route is: 28
The average shortest path: 10.718822843822844
Node Connectivity: 1
Algebraic Connectivity: 0.010650367637297337
```

- Here we have calculated various parameters.
- The **Node Connectivity** describes the number of nodes we must delete from the Graph G until it is **disconnected**.
- Disconnected means that if every node in our graph G can reach any other node in the network via edges it is **connected**. If this is not the case the graph is disconnected.
- As expected the output of the node connectivity is 1, meaning our graph is disconnected after removing just one node.
- This does not matter, as the size of the removed subgraph is just a single node and the rest of the network is still connected. If however the size of the resulting disconnected part is relatively big, this indicates a problem in the structure of the network.

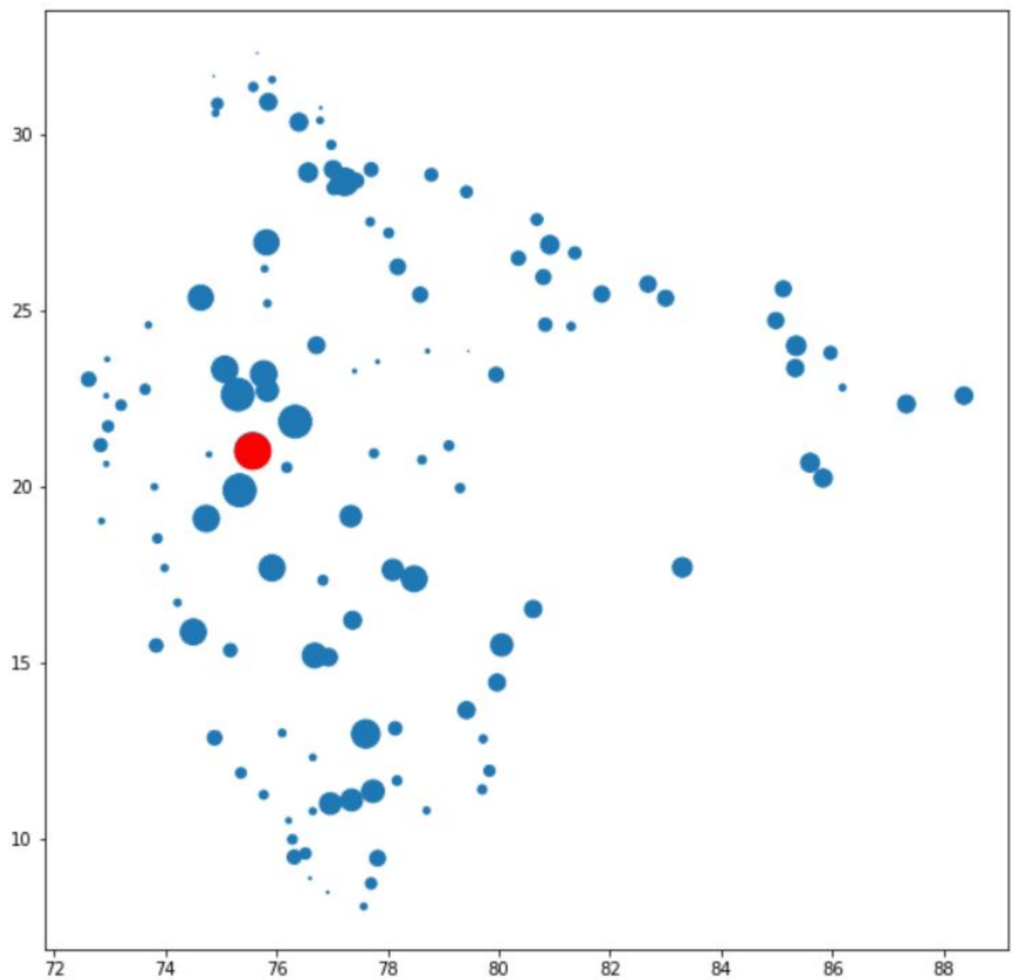
```
Maximum centrality value:
0.42247048725922
Node with maximum centrality is:
98
Latitude is : 21.01667
Longitude is : 75.56667
```

- Here, the **Betweenness Centrality** has been calculated.
- In graph theory, betweenness centrality is a measure of centrality in a graph based on shortest paths.
- For every pair of vertices in a connected graph, there exists at least one shortest path between the vertices such that either the number of edges that the path passes through (for unweighted graphs) or the sum of the weights of the edges (for weighted graphs) is minimized.
- The betweenness centrality for each vertex is the number of these shortest paths that pass through the vertex.
- The betweenness centrality of a node {\displaystyle v} v is given by the expression:

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

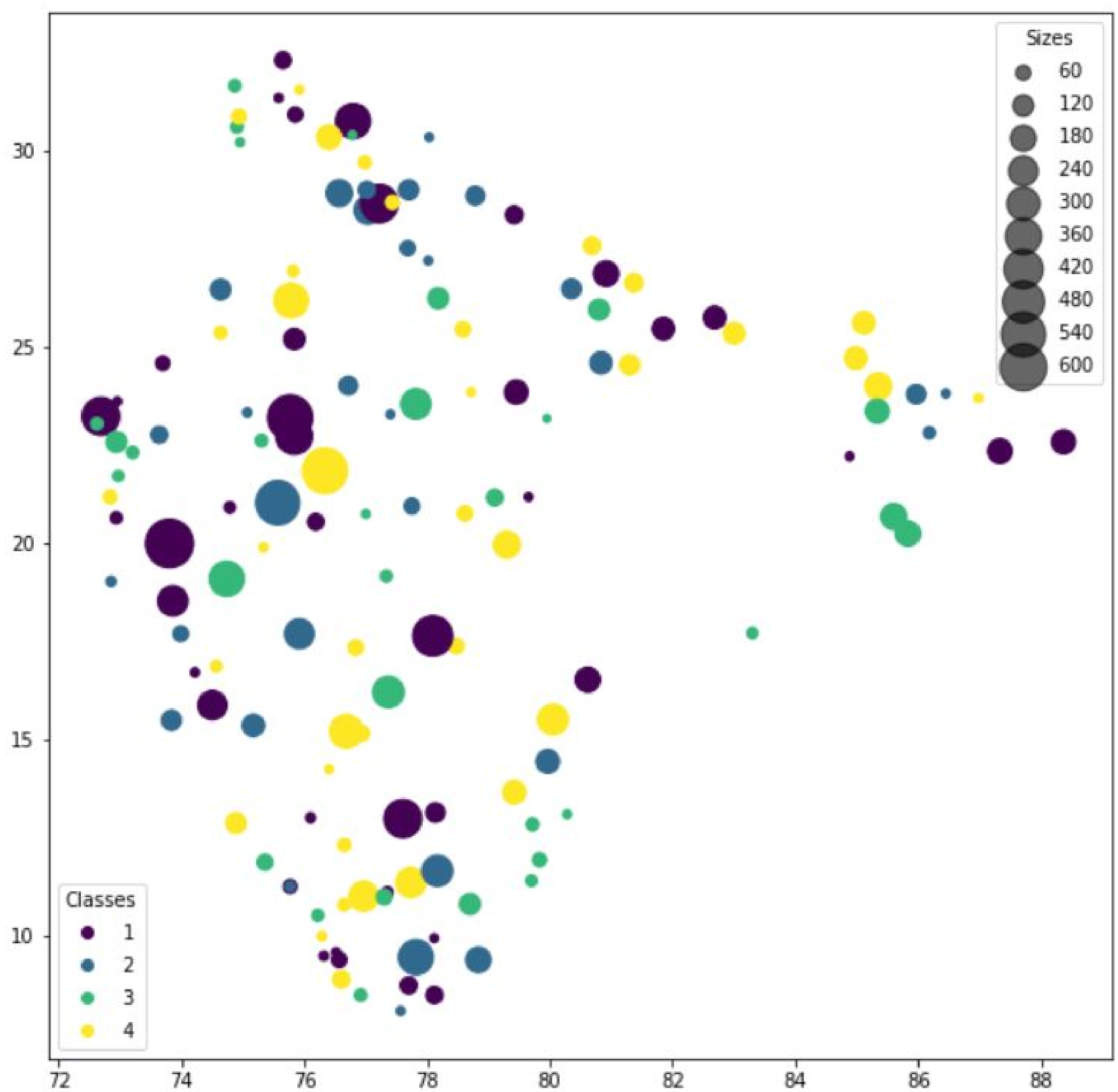
- Where σ_{st} is the total number of shortest paths from node *s* to node *t* and $\sigma_{st}(v)$ is the number of those paths that pass through *v*.

```
Out[14]: <matplotlib.collections.PathCollection at 0x1fcf876cbe0>
```

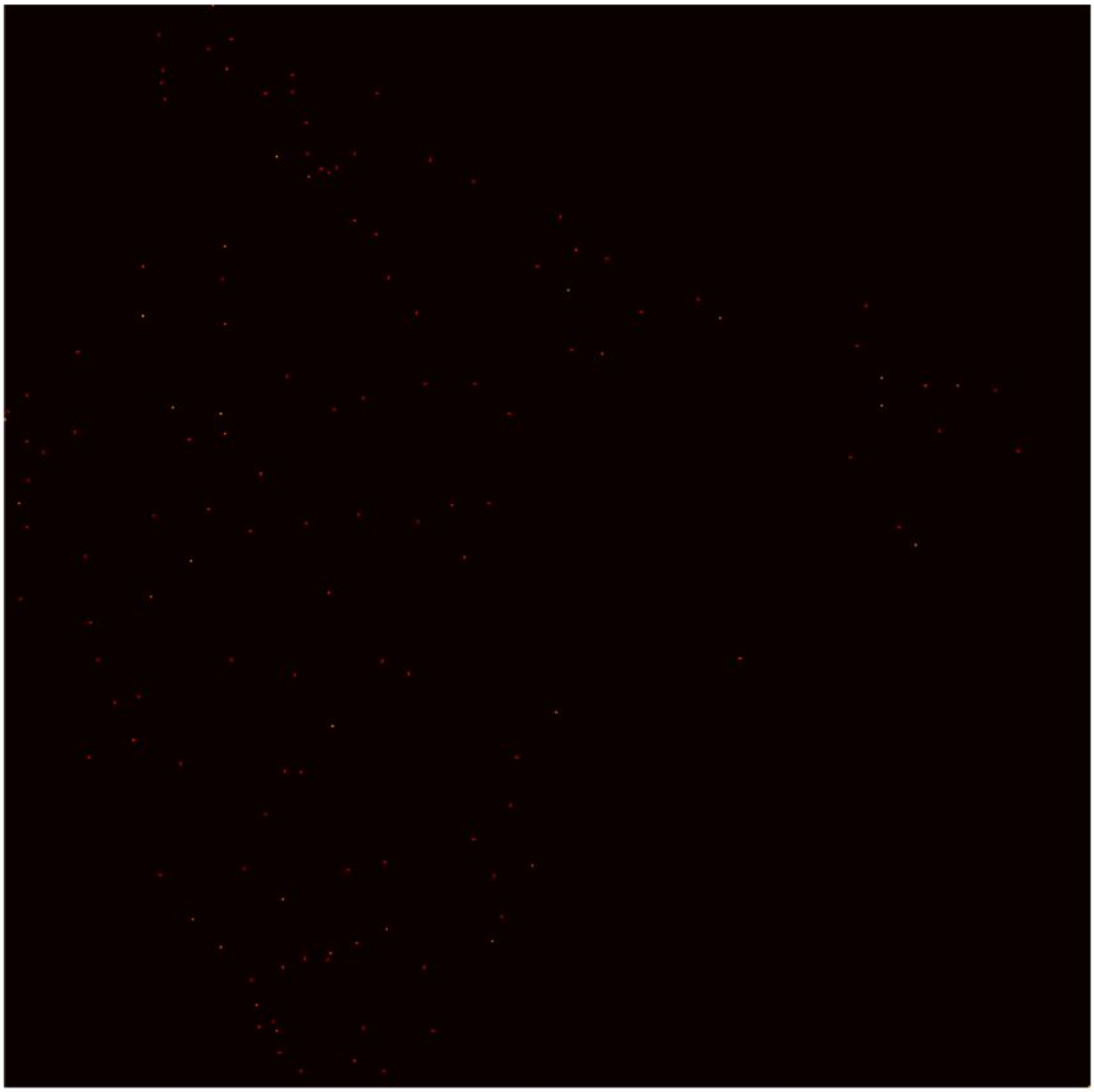


- The calculated **Betweenness Centrality** has been plotted
- The maximum value of the Betweenness Centrality is represented by 'red'.

- After declaring all the required parameters. We run a function to identify the pickup and drop location out of a possible **1458644** values.
- Then a function attaches the pickup and drop latitudes and longitudes to the list.
- Then we define a function to return a string with the **shortest path** using **DIJKSTRA'S ALGORITHM** between a pickup and drop location containing all the nodes visited in the path.
- After finding the shortest for all possible routes, we find the number of times **each node has been visited** through all the possible calculated paths.
- **Dictionary** data type has been used.
- Dictionary in Python is an unordered collection of data values, used to store **data** values like a map, which unlike other data Types that hold only a single value as an element, Dictionary holds **key:value pair.**

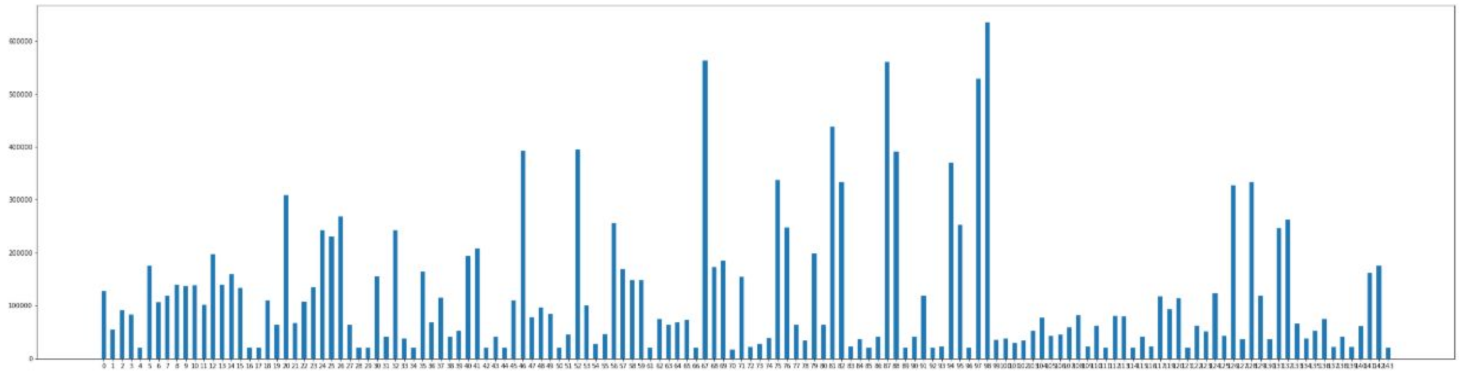


- The frequency of visit to each node has been plotted in the figure above.
- The bigger the circle, the greater the intensity.
- The values have been scaled down by 1000 for mapping (eg. real value 600=600*1000).
- The greater the number of the times the node has been visited, the higher its importance.
- We only have one plot as it is not a time series data.
- The values have been **binned** and allotted classes by Python as in the figure below.



- The frequency of the nodes visited has also been visualized in the form of a heat map as seen in the above figure.
- The darker the red, the greater the frequency.

Out[22]: <BarContainer object of 142 artists>

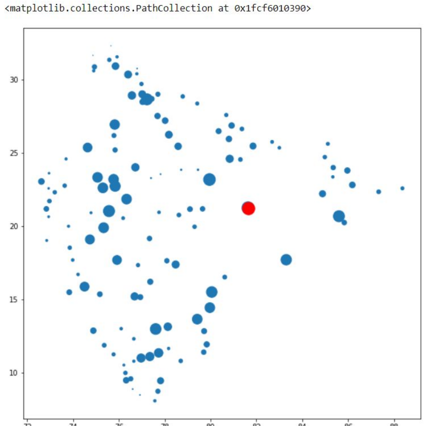


- The frequency has also been visualized in the form of a bar plot as shown in the figure above.

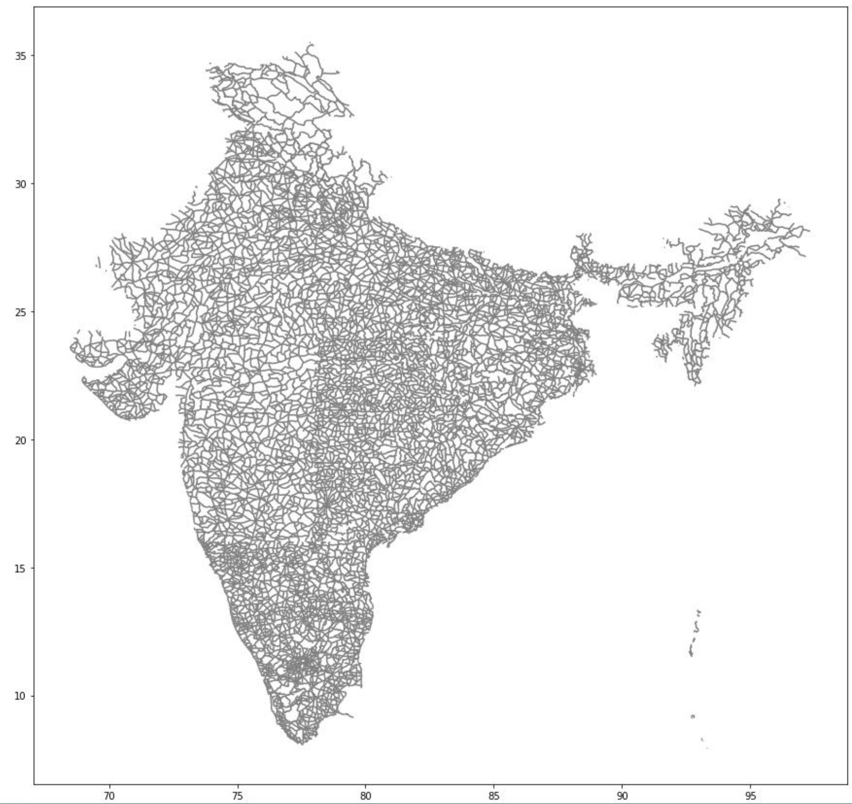
NOW WE REPEAT THE SAME STEPS AFTER ADDING A NODE

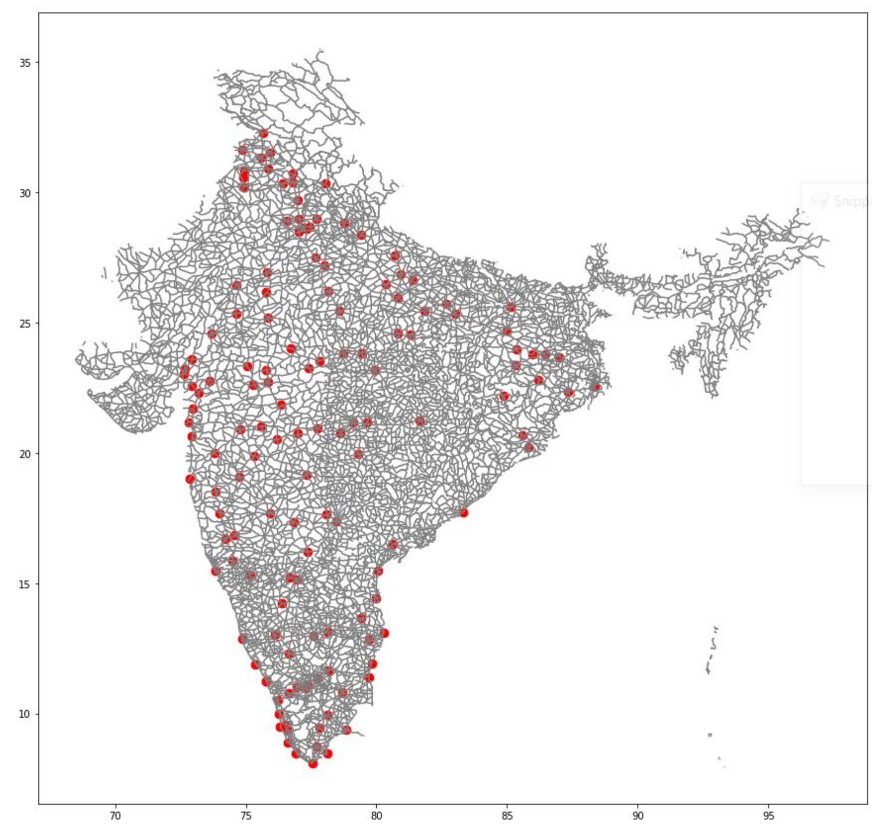
- We assume that the node we are adding is a new road that is being considered to be constructed. The node has connectivity with all adjacent nodes.
- We will now analyze if the intensity at the other nodes reduces due to the addition of the new node and conclude if the new node adds any value or not.

Out[32]: <matplotlib.collections.PathCollection at 0x1fcf6010390>

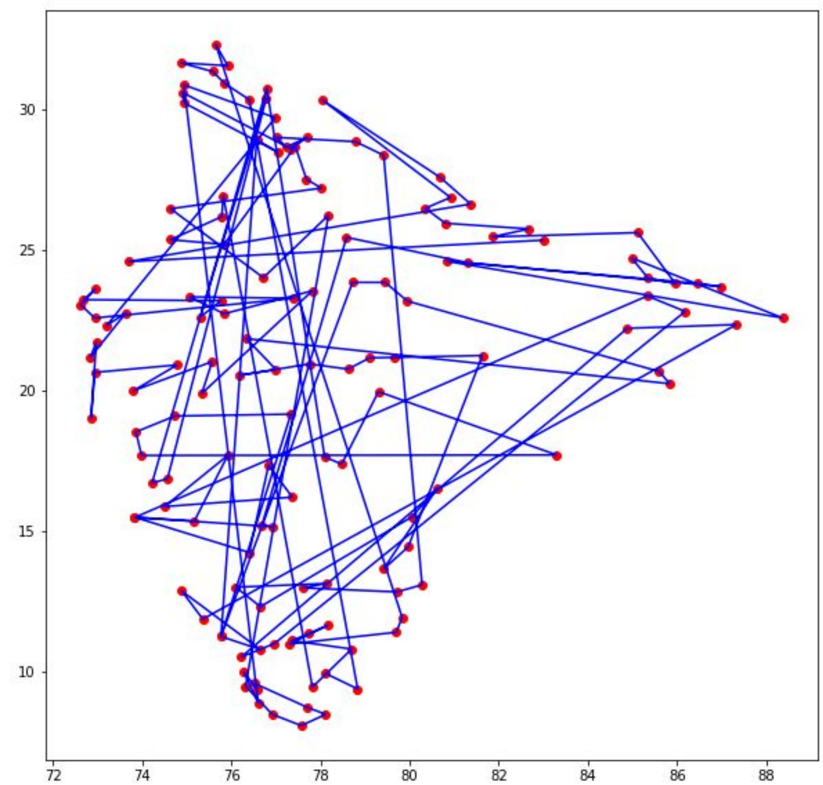


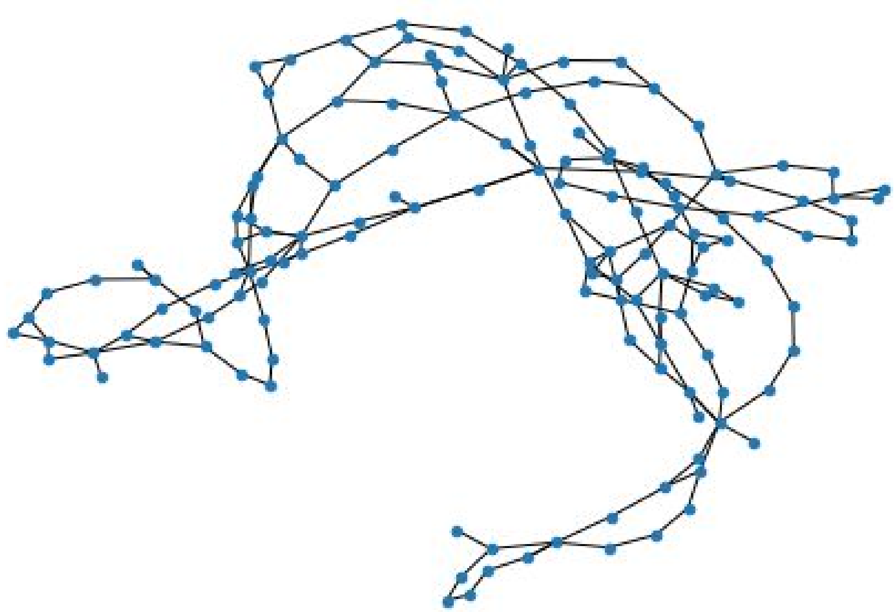
The red dot denotes the added node.


35
30
25
20
15
10
70
75
80
85
90
95


35
30
25
20
15
10
70
75
80
85
90
95

```
Out[25]: [<matplotlib.lines.Line2D at 0x1fcf5fd79b0>]
```



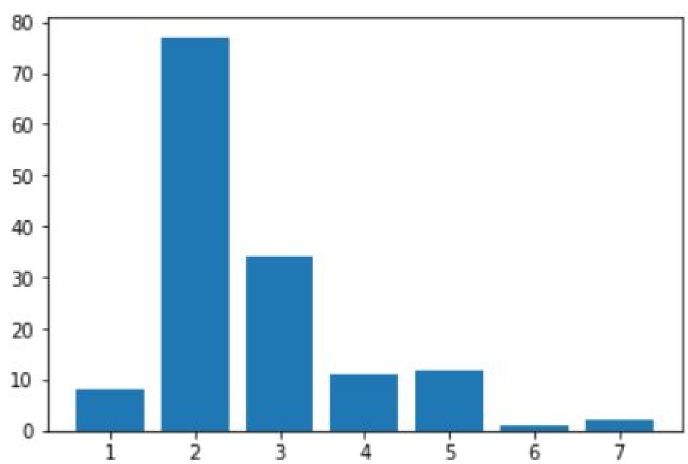


- Here, we can see the change in the shape of the network graph due to the addition of a node.

```
Name:
Type: MultiGraph
Number of nodes: 145
Number of edges: 194
Average degree:   2.6759

 After Removingthe cyclicity:

Name:
Type: Graph
Number of nodes: 145
Number of edges: 186
Average degree:   2.5655
```

- Clearly, the number of **edges have increased** and so has the average degree even after cyclicity has been removed.

```
Out[28]: <BarContainer object of 7 artists>
```



- The number of two degree nodes have specifically **increased**.

```
Density :
0.018453768453768452
```

- We observe no change in the graph density.

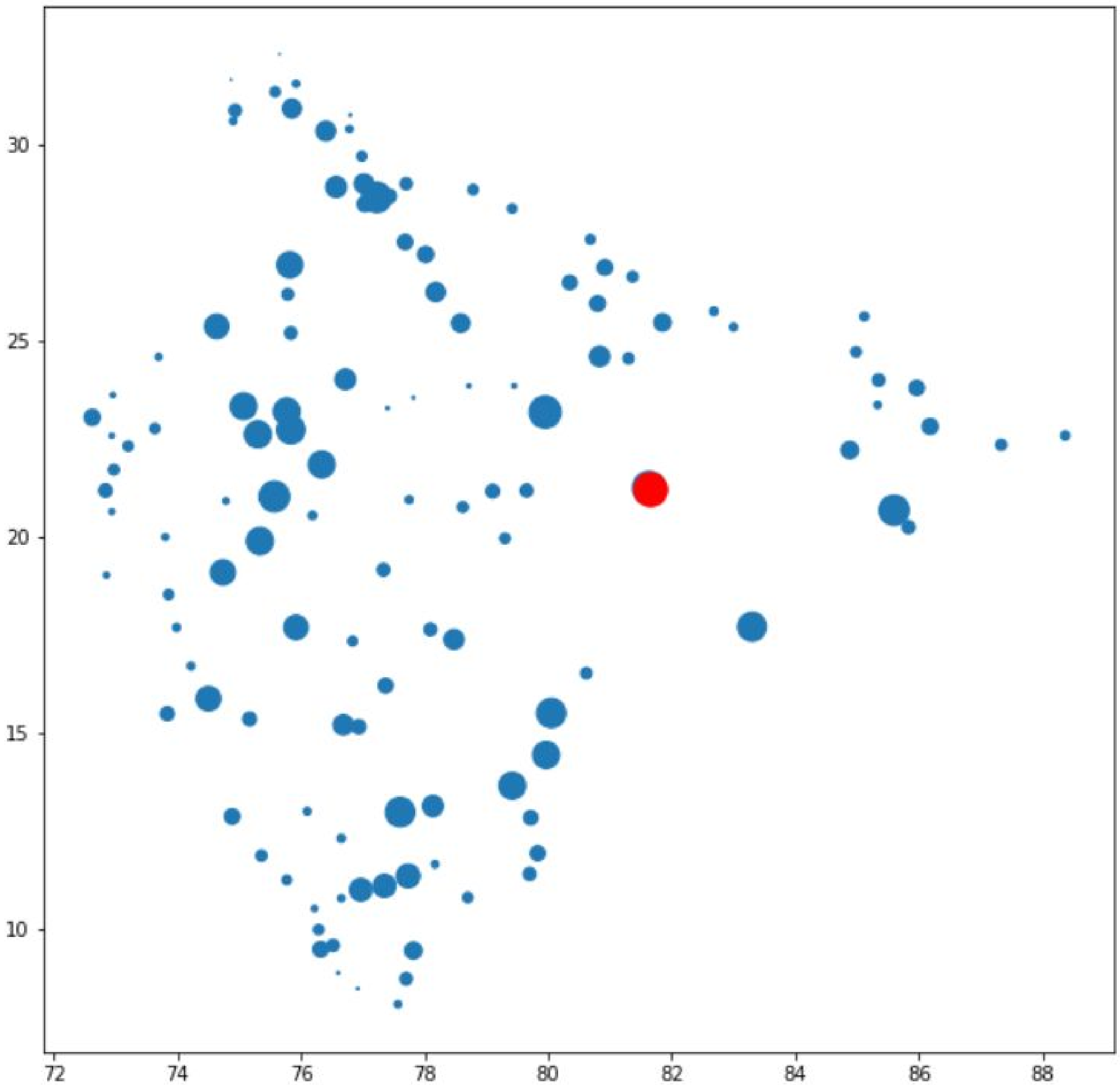
```
The longest route is: 28
The average shortest path: 9.849712643678162
Node Connectivity: 1
Algebraic Connectivity: 0.013163293110879687
```

- We can see a decrease in the average shortest path value from 10.71 to 9.84 .
- The algebraic connectivity increases from 0.010 to 0.013.

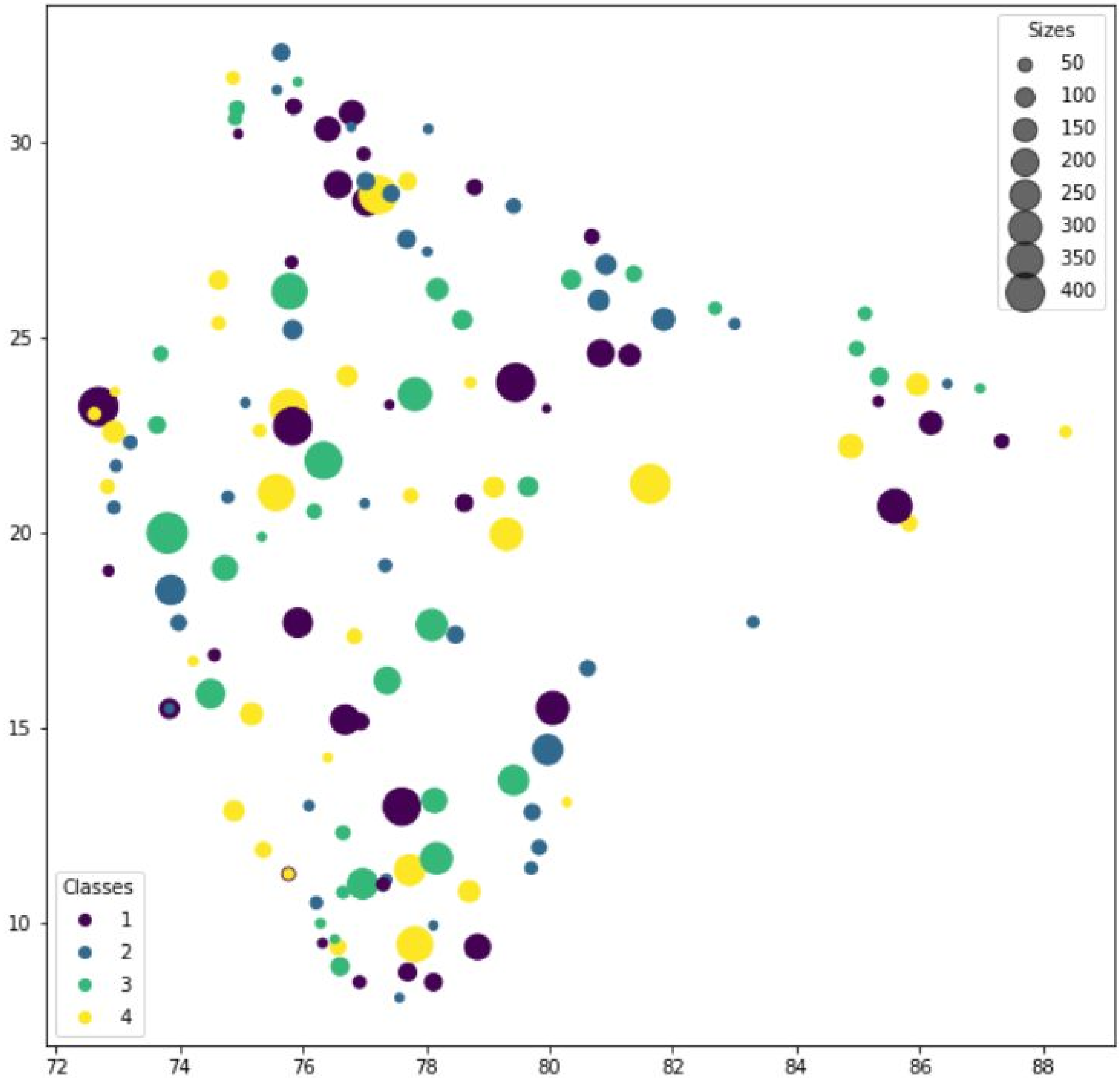
```
Maximum centrality value:
0.301520239020239
Node with maximum centrality is:
60
Latitude is : 21.23333
Longitude is : 81.63333
```

- The maximum centrality magnitude changes from 0.422 to 0.301
- The node with the **maximum centrality** value also changes from **98 to 60.**

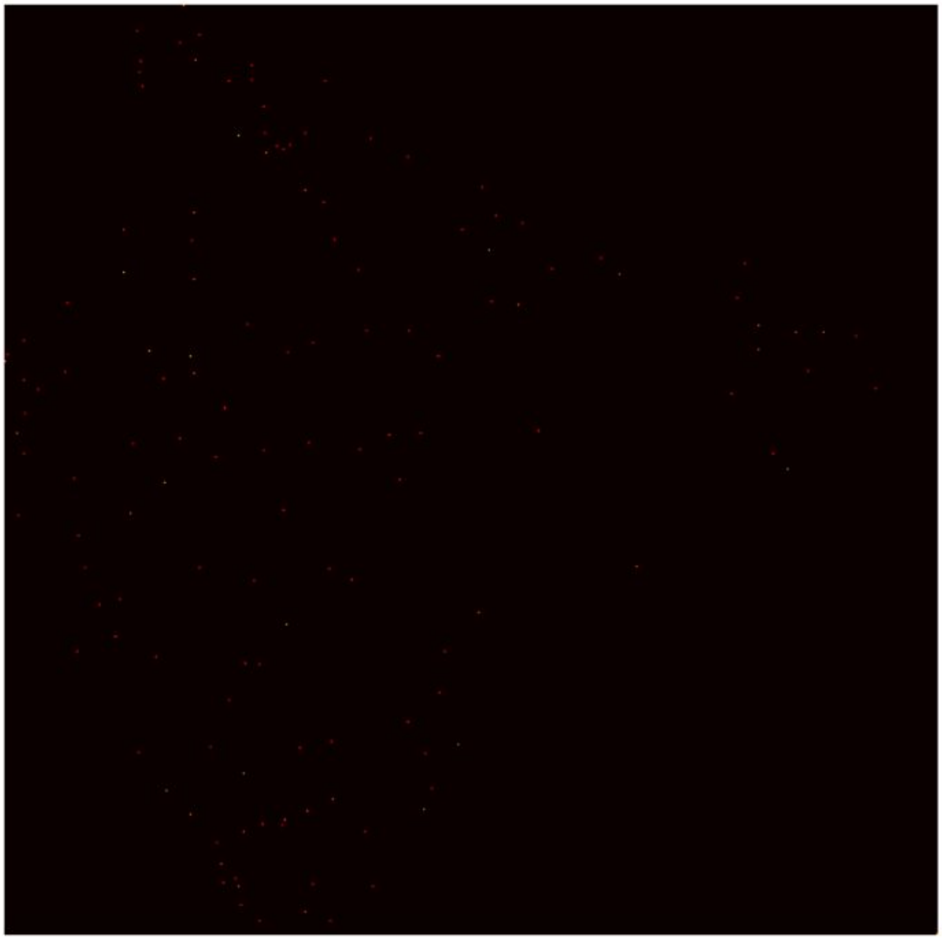
Out[32]: <matplotlib.collections.PathCollection at 0x1fcf6010390>



- We can clearly see that the **position of the maximum centrality** has changed after the addition on a node.
- Here the maximum centrality is also the added node.

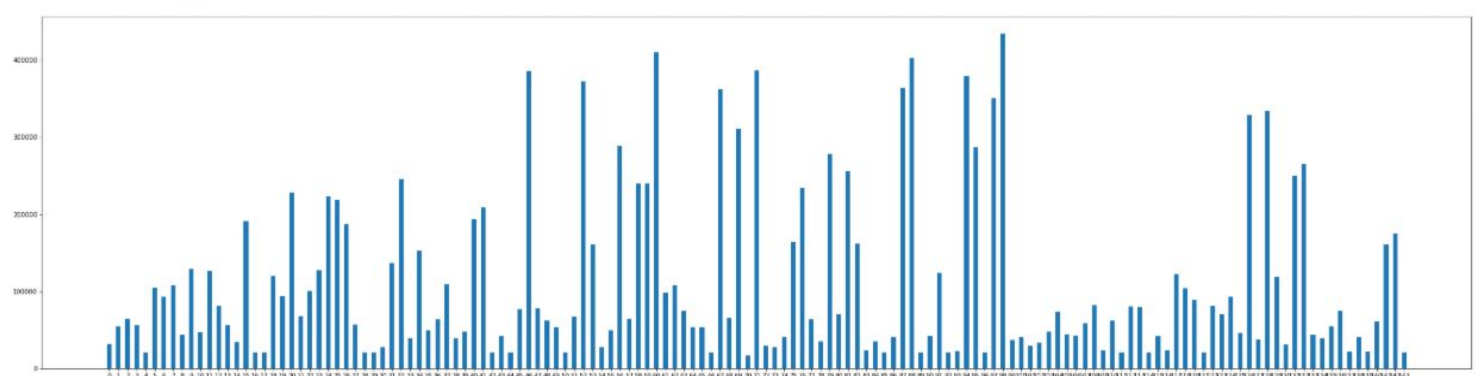


- Here we can clearly see that the **intensity/frequency** has **decreased** from the previous results across all bins. The maximum drops from **600 to 400**.
- **From here we can confirm and conclude that the addition of this node will be valuable and will reduce the traffic intensity significantly across most nodes, especially major nodes.**



- The reduced intensity can also be inferred from the heat map and the bar plot.

```
Out[38]: <BarContainer object of 143 artists>
```



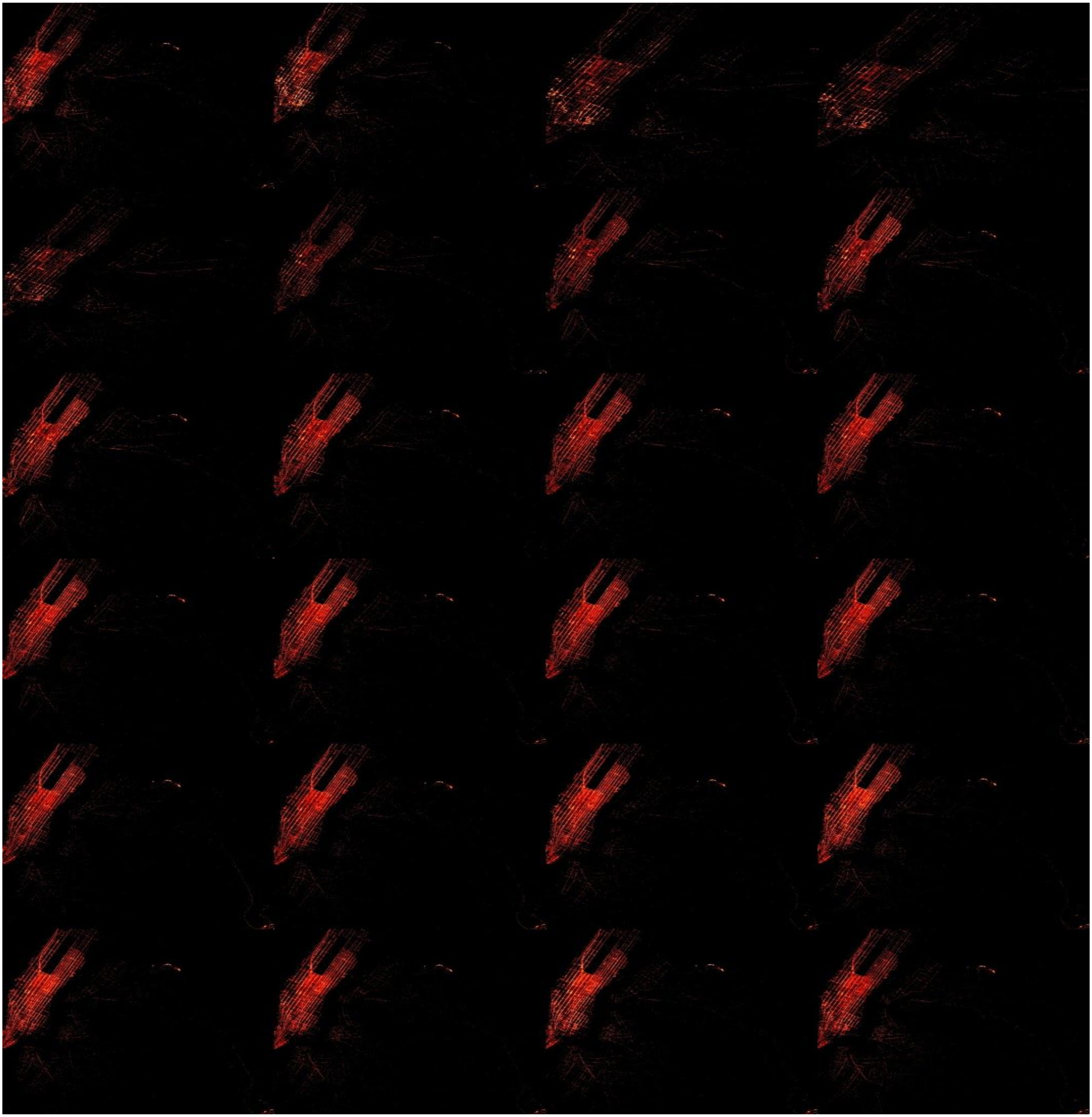
- Hence, the addition of our assumed node results in **reduced intensity** across nodes and will therefore be a recommended construction.

DATASET 2:

- Many times in big cities and major metropolitans constructing roads make very little difference to the traffic flow diversion and visualization.
- In such cases it is better to visualize the traffic throughout the day over the whole area and identify the pattern of traffic.
- This way resources such as barricades, traffic police etc. can be allocated to areas according to the traffic intensity in an efficient way.

We first visualize the traffic **without adding any node(s)** for 24 hours (left to right). The first image is hour 0 and the last is hour 23.

Now we visualize the traffic **after adding a node** for 24 hours (right to left).



- We can clearly see that the addition of a node results in a little or no difference in the traffic intensity through the area, the traffic pattern remains the same.

- Here, we can observe that from **3-5 am** the traffic **intensity reduces** and then gradually increases with the **maximum intensity** around **1900hrs.**

- This way this visualization helps us to allocate resources to the required location at a given time of the day.

CONCLUSION

- With the increasing population the state has to keep up with the infrastructure.
- New roads have to be paved to distribute traffic flow and intensity.
- Using this model we can predict whether a road if constructed will be able to help in doing so or not.
- For the used dataset and the assumed added node we can clearly conclude that the added node will successfully be able to reduce traffic and flow across other nodes in the network.
- For the 2nd dataset we can see that the addition of a new node makes a little or no difference in reducing the traffic intensity.
- Hence, we switch to data visualization.
- We visualize the traffic flow of the area for the whole day and identify a pattern.
- After successful identification of a pattern in the traffic intensity through a day, the resources can be allocated accordingly by the respective authorities.

REFERENCES

- https://shapely.readthedocs.io/en/latest/manual.html
- http://graphml.graphdrawing.org/
- https://www.kaggle.com/drgilermo/dynamics-of-new-york-city-animation

- https://www.kaggle.com/c/nyc-taxi-trip-duration
- https://www.geeksforgeeks.org/betweenness-centrality-centrality-measure/
- https://www.diva-gis.org/gdata
- https://towardsdatascience.com/geopandas-101-plot-any-data-with-a-latitude-and-longitude-on-a-map-98e01944b972
- https://osmnx.readthedocs.io/en/stable/osmnx.html

APPENDIX

DATASET 1:

In [2]:
```
import networkx as nx
from shapely.geometry import Point ,Polygon
import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd
import geopandas as gdp
from geopandas import GeoDataFrame
import numpy as np
from collections import Counter
import random
```

- Importing all the required libraries and packages.

In [3]:
```
##Reading the Graph in the code

G = nx.read_graphml('try.graphml')
G.remove_node('60')
```

- Reading the graph into the code.

In [4]:
```
#ploting the initial roadways map

map = gpd.read_file("./IND_rds/IND_roads.shp")
fig,ax = plt.subplots(figsize=(15,15))
Rmap = map.plot(ax = ax , color = "grey")
```

- To plot the surveyed road network of India.

In [5]:

```
##Finding latitude and longitude for plotting
lat = nx.get_node_attributes(G , 'Latitude')
long = nx.get_node_attributes(G,'Longitude')
geometry = []
for i in lat.keys():
    geometry.append(Point(long[str(i)],lat[str(i)]))
xs = [point.x for point in geometry]
ys = [point.y for point in geometry]
min_longitude = min(xs)
max_longitude = max(xs)
min_latitude = min(ys)
max_latitude = max(ys)
BBox = [min_longitude , max_longitude , min_latitude , max_latitude]
```

- We find the latitude and longitude of nodes to plot.

In [6]:

```
#Plotting the points

df = pd.DataFrame(list(zip(xs,ys)),columns = ['Longitude' , 'Latitude'])
geometry = [Point(xy) for xy in zip(df['Longitude'], df['Latitude'])]
gdf = GeoDataFrame(df, geometry=geometry)
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
gdf.plot(ax= map.plot(figsize=(15,15) , color = 'grey'), marker='o', color='red', markersize=75);
```

- To plot the nodes.

In [7]:

```
#Drawing the graph

fig,ax = plt.subplots(figsize = (10,10))
ax.scatter(xs,ys,color = 'red')
ax.plot(xs,ys,color = 'blue')
```

- To plot the graph after node identification.

In [8]:

```
#Basic Graph Properties:

nx.draw(G, node_size = 20, width = 1)
```

- Converted into a proper **network graph.**

```python
print(nx.info(G))
print("\n After Removingthe cyclicity:\n")
G_simple = nx.Graph(G)
print(nx.info(G_simple))
```

- To calculate the properties of the graph.

```python
degree_cal = dict(G.degree())
degree_dic = Counter(degree_cal.values())
plt.bar(degree_dic.keys() , degree_dic.values())
```

- To visualize the node frequency in a bar plot.

```python
print("Density : ")
print(nx.density(G))
```

- Returns the graph density.

```python
a = nx.diameter(G)
b = nx.average_shortest_path_length(G_simple)
c = nx.node_connectivity(G_simple)
d = nx.algebraic_connectivity(G_simple)

print("The longest route is: {} \nThe average shortest path: {} \nNode Connectivity: {} \nAlgebraic Connectivity: {}".format(a,
b,c,d))
```

- Returns the shortest path between two nodes.

In [13]:
```
#Betweenness Centrality :
bet_centrality = nx.betweenness_centrality(G_simple, normalized = True,endpoints = False)
print("Maximum centrality value:")
value_maxcen = max(bet_centrality.values())
print(value_maxcen)
keymax = max(bet_centrality , key = bet_centrality.get)
print("Node with maximum centrality is:")
print(keymax)
print("Latitude is : {}" .format(G.nodes[str(keymax)]['Latitude']))
print("Longitude is : {}".format(G.nodes[str(keymax)]['Longitude']))
lat_maxcen = G.nodes[str(keymax)]['Latitude']
long_maxcen = G.nodes[str(keymax)]['Longitude']
```

- Calculates the maximum centrality value.

In [14]:
```
del bet_centrality['70']
del bet_centrality['118']
fig,ax = plt.subplots(figsize = (10,10))
size = list(bet_centrality.values())
size = [s*1000 for s in size]
ax.scatter(xs,ys,s=size)
ax.scatter(long_maxcen,lat_maxcen,s=value_maxcen*1000,c = "red")
```

- Plots the calculated centrality values.

In [15]:
```
#Pickup and Drop Latitude and Longitude

pickup_Lat = []
pickup_Long = []
drop_Lat = []
drop_Long = []
pickup_node = []
drop_node = []
```

- Declaring all the required parameters.

In [16]:
```
for i in range(1458644):
    c,m = 1,1
    while(True):
        c = random.randint(1,143)
        if c != 70 and c != 118 and c != 60:
            break
    pickup_node.append(c)
    while(True):
        m = random.randint(1,143)
        if m != 70 and  m != 118 and m != 60:
            if m != c:
                break
    drop_node.append(m)
```

- Function to identify the pickup and drop location out of a possible **1458644** values.

In [17]:
```
for x in pickup_node:
    y = lat[str(x)]
    z = long[str(x)]
    pickup_Lat.append(y)
    pickup_Long.append(z)

for x in drop_node:
    y = lat[str(x)]
    z = long[str(x)]
    drop_Lat.append(y)
    drop_Long.append(z)
```

- Function to attach the pickup and drop latitudes and longitudes to the list.

In [18]:
```
path = []
for i in range(1458644):
    x = pickup_node[i]
    y = drop_node[i]
    path.append(nx.shortest_path(G,str(x),str(y)))
```

- Returns the shortest between nodes.

In [19]:
```
allLat = []
allLong= []
Dict = {}
nodes = list(G.nodes())
for x in nodes:
    Dict[x] = 0
for pather in path:
    for i in pather:
        c = Dict[i]
        Dict[i] = c+ 1
        if int(i) != 70 and int(i) != 118:
            allLat.append(lat[str(i)])
            allLong.append(long[str(i)])
```

- Returns the number of times each node is visited.

In [20]:
```
N = len(xs)
del Dict['144']
del Dict['118']
sizevalues = [siz/1000 for siz in list(Dict.values())]
color = np.random.randint(1, 5, size=N)
fig ,ax = plt.subplots(figsize=(10,10))
scatter = ax.scatter(xs,ys,s=sizevalues,c=color)
legend1 = ax.legend(*scatter.legend_elements(),loc="lower left", title="Classes")
ax.add_artist(legend1)
handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
legend2 = ax.legend(handles, labels, loc="upper right", title="Sizes")
```

- Plots the visited frequency for each node.

In [21]:
```
imageSize = (700,700)
allLat = np.array(allLat)
allLong = np.array(allLong)
longRange = [min(allLong),max(allLong)]
latRange = [min(allLat),max(allLat)]

allLatInds  = imageSize[0] - (imageSize[0] * ((allLat  - latRange[0])  / (latRange[1]  - latRange[0]))).astype(int)
allLongInds = (imageSize[1] * ((allLong - longRange[0]) / (longRange[1] - longRange[0]))).astype(int)

locationDensityImage = np.zeros(imageSize)
for latInd, longInd in zip(allLatInds,allLongInds):
    if latInd == 700 or longInd == 700:
            locationDensityImage[699,699] += 1
    else:
        locationDensityImage[latInd,longInd] += 1

fig, ax = plt.subplots(nrows=1,ncols=1,figsize=(10,10))
ax.imshow(np.log(locationDensityImage+1),cmap='hot')
ax.set_axis_off()
```

- Plotting the heat map for frequency of nodes visited.

In [22]:
```
x = Dict.keys()
y = Dict.values()
fig ,ax = plt.subplots(figsize = (40,10))
ax.bar(x,y,width = 0.5)
```

- To visualize the frequency in the form of a bar graph.

NOW WE REPEAT THE SAME STEPS AFTER ADDING A NODE

In [23]:
```
G_added= nx.read_graphml('try.graphml')
```

In [24]:
```
map = gpd.read_file("./IND_rds/IND_roads.shp")
fig,ax = plt.subplots(figsize=(15,15))
Rmap = map.plot(ax = ax , color = "grey")


lat = nx.get_node_attributes(G_added , 'Latitude')
long = nx.get_node_attributes(G_added,'Longitude')
geometry = []
for i in lat.keys():
    geometry.append(Point(long[str(i)],lat[str(i)]))
xs = [point.x for point in geometry]
ys = [point.y for point in geometry]
min_longitude = min(xs)
max_longitude = max(xs)
min_latitude = min(ys)
max_latitude = max(ys)
BBox = [min_longitude , max_longitude , min_latitude , max_latitude]

df = pd.DataFrame(list(zip(xs,ys)),columns = ['Longitude' , 'Latitude'])
geometry = [Point(xy) for xy in zip(df['Longitude'], df['Latitude'])]
gdf = GeoDataFrame(df, geometry=geometry)
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
gdf.plot(ax= map.plot(figsize=(15,15) , color = 'grey'), marker='o', color='red', markersize=75);
```

```
In [25]: fig,ax = plt.subplots(figsize = (10,10))
         ax.scatter(xs,ys,color = 'red')
         ax.plot(xs,ys,color = 'blue')
```

```
In [26]: #Basic Graph Properties:

         nx.draw(G_added, node_size = 20, width = 1)
```

```
In [27]: print(nx.info(G_added))
         print("\n After Removingthe cyclicity:\n")
         G_simple = nx.Graph(G_added)
         print(nx.info(G_simple))
```

```
In [28]: degree_cal = dict(G_added.degree())
         degree_dic = Counter(degree_cal.values())
         plt.bar(degree_dic.keys() , degree_dic.values())
```

```
In [29]: print("Density : ")
         print(nx.density(G))

         Density :
         0.018453768453768452
```

```
In [30]: a = nx.diameter(G_added)
         b = nx.average_shortest_path_length(G_simple)
         c = nx.node_connectivity(G_simple)
         d = nx.algebraic_connectivity(G_simple)

         print("The longest route is: {} \nThe average shortest path: {} \nNode Connectivity: {} \nAlgebraic Connectivity: {}".format(a,
         b,c,d))
```

In [31]:
```python
#Betweenness Centrality :
bet_centrality = nx.betweenness_centrality(G_simple, normalized = True,endpoints
= False)
print("Maximum centrality value:")
value_maxcen = max(bet_centrality.values())
print(value_maxcen)
keymax = max(bet_centrality , key = bet_centrality.get)
print("Node with maximum centrality is:")
print(keymax)
print("Latitude is : {}" .format(G_added.nodes[str(keymax)]['Latitude']))
print("Longitude is : {}".format(G_added.nodes[str(keymax)]['Longitude']))
lat_maxcen = G_added.nodes[str(keymax)]['Latitude']
long_maxcen = G_added.nodes[str(keymax)]['Longitude']
```

In [32]:
```python
del bet_centrality['70']
del bet_centrality['118']
fig,ax = plt.subplots(figsize = (10,10))
size = list(bet_centrality.values())
size = [s*1000 for s in size]
ax.scatter(xs,ys,s=size)
ax.scatter(long_maxcen,lat_maxcen,s=value_maxcen*1000,c = "red")
```

In [33]:
```python
path = []
for i in range(1458644):
    x = pickup_node[i]
    y = drop_node[i]
    path.append(nx.shortest_path(G_added,str(x),str(y)))
```

In [34]:
```python
allLat = []
allLong= []
Dict = {}
nodes = list(G_added.nodes())
for x in nodes:
    Dict[x] = 0
for pather in path:
    for i in pather:
        c = Dict[i]
        Dict[i] = c+ 1
        if int(i) != 70 and int(i) != 118:
            allLat.append(lat[str(i)])
            allLong.append(long[str(i)])
```

In [35]:
```python
N = len(xs)
del Dict['144']
del Dict['118']
sizevalues = [siz/1000 for siz in list(Dict.values())]
color = np.random.randint(1, 5, size=N)
fig ,ax = plt.subplots(figsize=(10,10))
scatter = ax.scatter(xs,ys,s=sizevalues,c=color)
legend1 = ax.legend(*scatter.legend_elements(),loc="lower left", title="Classes")
ax.add_artist(legend1)
handles, labels = scatter.legend_elements(prop="sizes", alpha=0.6)
legend2 = ax.legend(handles, labels, loc="upper right", title="Sizes")
```

In [36]:
```python
imageSize = (700,700)
allLat = np.array(allLat)
allLong = np.array(allLong)
longRange = [min(allLong),max(allLong)]
latRange = [min(allLat),max(allLat)]

allLatInds  = imageSize[0] - (imageSize[0] * ((allLat  - latRange[0])  / (latRange[1]  - latRange[0]))).astype(int)
allLongInds = (imageSize[1] * ((allLong - longRange[0]) / (longRange[1] - longRange[0]))).astype(int)

locationDensityImage = np.zeros(imageSize)
for latInd, longInd in zip(allLatInds,allLongInds):
    if latInd == 700 or longInd == 700:
            locationDensityImage[699,699] += 1
    else:
        locationDensityImage[latInd,longInd] += 1
```

In [37]:
```python
fig, ax = plt.subplots(figsize=(10,10))
ax.imshow(np.log(locationDensityImage+1),cmap='hot')
ax.set_axis_off()
```

In [38]:
```python
x = Dict.keys()
y = Dict.values()
fig ,ax = plt.subplots(figsize = (40,10))
ax.bar(x,y,width = 0.5)
```

DATASET 2:

In [30]:
```
from datashader.utils import lnglat_to_meters as webm
from functools import partial
from datashader.utils import export_image
from datashader.colors import colormap_select , Greys9
from IPython.core.display import HTML , display
import pandas as pd
import datashader as ds
import datashader.transfer_functions as tf
from datetime import datetime
```

In [31]:
```
df = pd.read_csv('train.csv')
```

In [32]:
```
df.head()
```

In [33]:
```
df.drop(['id' , 'vendor_id' , 'passenger_count' , 'store_and_fwd_flag','trip_duration'] , axis = 1)
```

In [34]:
```
pickup_hour = []
dropoff_hour = []
for i in range(1458644):
    datetime_str = df['pickup_datetime'][i]
    datetime_object = datetime.strptime(datetime_str, '%Y-%m-%d %H:%M:%S')
    pickup_hour.append(datetime_object.hour)
    datetime_str = df['dropoff_datetime'][i]
    datetime_object = datetime.strptime(datetime_str, '%Y-%m-%d %H:%M:%S')
    dropoff_hour.append(datetime_object.hour)
df['pickup_hour'] = pickup_hour
df['dropoff_hour'] = dropoff_hour
```

In [87]:
```
num = 0
```

In [250]:
```
i= 23
df_hour = df.loc[df['pickup_hour'] == i].copy()
s = "Hour_" + str(i)
```

In [251]:
```
df_hour.drop(columns = ['id' , 'vendor_id' , 'passenger_count' , 'store_and_fwd_flag','trip_duration'])
df_hour
```

In [252]:
```
df_hour.loc[:,'easting'] , df_hour.loc[:,'northing'] = webm(df['pickup_longitude'] , df['pickup_latitude'])
df_hour.head()
```

In [253]:
```
y_range_min = df_hour['pickup_latitude'].quantile(0.01)
y_range_max = df_hour['pickup_latitude'].quantile(0.99)
x_range_min = df_hour['pickup_longitude'].quantile(0.01)
x_range_max = df_hour['pickup_longitude'].quantile(0.99)
sw = webm(x_range_min,y_range_min)
ne = webm(x_range_max,y_range_max)
SF = zip(sw,ne)
plot_width = int(500)
plot_height = int(500)
background = "black"
export = partial(export_image,background=background,export_path = "Analysis")
cm = partial(colormap_select , reverse = (background != "black"))

display(HTML("<style>.container {width : 100% !important;}</style>"))
cvs = ds.Canvas(plot_width,plot_height,*SF)
agg = cvs.points(df_hour,'easting','northing')
```

In [264]:
```
export(tf.shade(agg , cmap = cm(Greys9,0.5) ) , s)
```

In [263]:
```
from colorcet import fire
s = s + "_fire"
export(tf.shade(agg,cmap = cm(fire,0.5) , how = 'log') , s)
```

In [256]:
```
num = num + df_hour.shape[0]
del(df_hour)
```

In [257]:
```
num
```

In [258]:
```
df.shape[0]
```