# london_underground

February 2, 2023

[601]:
```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import itertools
import matplotlib.cm as cmx
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

## 1 Understanding the Dataset

[602]:
```
folder = ""
file = "london_underground_updated"
ext = ".npy"
london_underground = np.load( folder+file+ext )




#old dataset - just undergound_tube stations

folder = ""
file = "london_underground"
ext = ".npy"
london_underground_old = np.load( folder+file+ext )
```

[603]:
```
N = len(london_underground_G)
```

The Nodes represent stations in London's underground transportation network.

[639]:
```
print('There are', N, 'nodes.')

just_underground = nx.node_connected_component(london_underground_G, 41)
G_sub = nx.subgraph (london_underground_G, just_underground)
print("There are", len(G_sub), "underground stations")
```

```
There are 369 nodes.
There are 271 underground stations
```

271 of these 369 nodes represent london undergound train stations, the rest of the nodes represent bus stations in London.

Note: There's 272 stations as of right now in the actual London underground system but our dataset only accounts for 271, since it's slightly outdated.

[605]:
```
nx.is_weighted(london_underground_G, edge=None, weight='weight')
```

[605]:
```
False
```

[606]:
```
nx.is_directed(london_underground_G)
```

[606]:
```
False
```

[607]:
```
nx.is_connected(london_underground_G)
```

[607]:
```
False
```

[608]:
```
print('There are',london_underground_G.number_of_edges(),'edges.')
```

```
There are 312 edges.
```

The edges represent transit routes between stations. The dataset originally has weighted edges, however, they are not relevant to the underground network as they represent the different number of ways to get to the station by public transit (tube, bus, etc.). Nonetheless, the weights were useful to help us double check measures like centrality, etc.

This graph is undirected, as an edge connects two nodes if there is (at least) one transit route between them.

This graph is disconnected, but has a giant component consisting of the 271 london undergound stations, which makes sense contextually since there should be a path between any two stations.

Construction of the London Underground began in the early 19th century, with the first line to open being the "Metropolitan Railway" in 1863. Now, the London Underground (informally known as "the tube") has 272 transit stations. The tube has 11 transit lines. Roughly 5 million passengers use the tube every day. Our network we will consider contains both London Underground (tube) stations as well as above ground bus stations.

In terms of node metadata, we have the station names and their geographic coordinates.

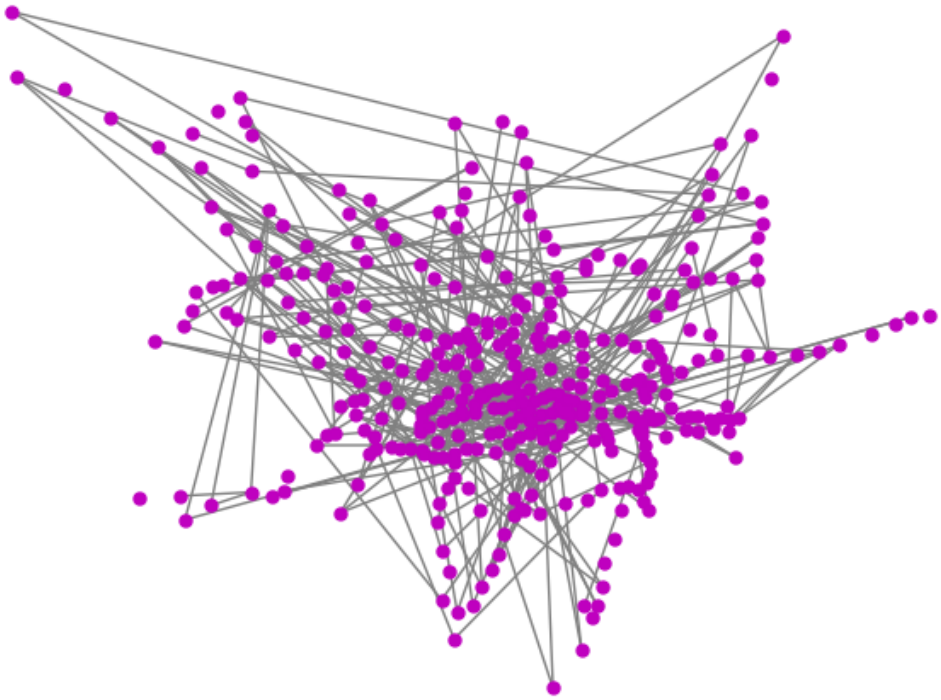
## 2 Plotting the Dataset

Here's our adjacency matrix, that we will turn into a networkx graph.

[609]:
```
print(london_underground)

london_underground_G = nx.to_networkx_graph(london_underground)

london_underground_G_old = nx.to_networkx_graph(london_underground_old)
```

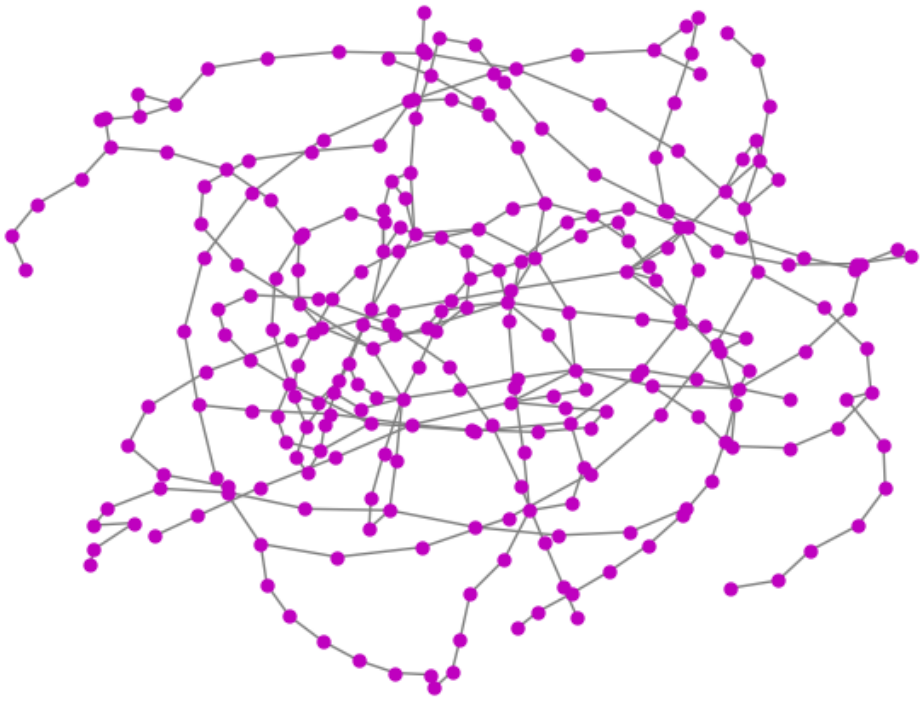
```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

[610]:
```
london_node_coords = pd.read_csv(r"/Users/hedavamsolano/Documents/Winter Term␣
↪2023/Final_Project/london_transport_nodes.txt",  delim_whitespace=True)

coordinates = []
with open('london_transport_nodes.txt', 'r') as fd:
    lines = fd.readlines()
    for line in lines:
        x = line.split(' ')[2]
        y = line.split(' ')[3][:-1]
        coords = tuple((float(y), float(x)))
        coordinates.append(coords)

Names = []
with open('london_underground_names.txt', 'r') as fd:
    lines = fd.readlines()
for line in lines:
    name = line.split('\n')[0]
    Names.append(name)

nx.draw(london_underground_G, pos = coordinates, node_color='m', node_size=25,␣
↪edge_color='grey',  with_labels=False)
```



Above is the geographical plot of our network.

[611]:
```
l_pos_old = nx.spring_layout(london_underground_G_old, k = .1)

nx.draw(london_underground_G_old, l_pos_old, node_color='m', node_size=25,␣
↪edge_color='grey', with_labels=False)
```



Above is a non-geographical model of our network, which makes it easier to understand.

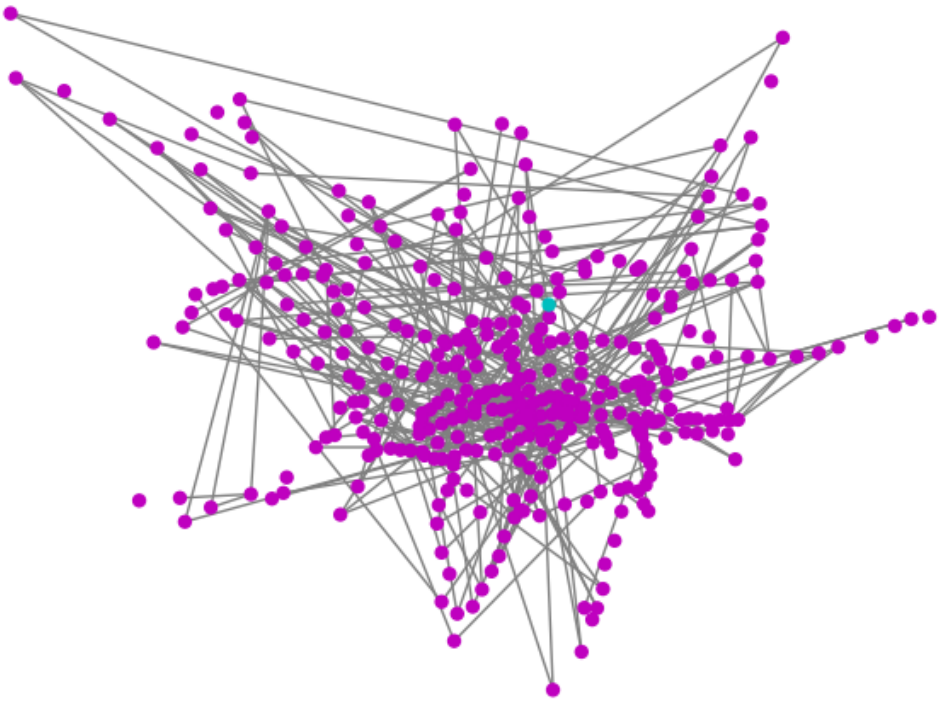
## 3 Centrality Measures

[612]:
```
a = nx.degree_centrality(london_underground_G)
b = nx.closeness_centrality(london_underground_G)
c = nx.eigenvector_centrality(london_underground_G)
d = nx.betweenness_centrality(london_underground_G)

print('The station with highest degree centrality is', max(a, key=lambda key:␣
 ↪a[key]), "which is", Names[max(a, key=lambda key: a[key])], "station")
print('The station with highest closeness centrality is', max(b, key=lambda key:␣
 ↪b[key]), "which is", Names[max(b, key=lambda key: b[key])], "station")
print('The station with highest eigenvector centrality is', max(c, key=lambda␣
 ↪key: c[key]), "which is", Names[max(c, key=lambda key: c[key])], "station")
print('The station with highest betweenness centrality is', max(d, key=lambda␣
 ↪key: d[key]), "which is", Names[max(d, key=lambda key: d[key])], "station\n")
```

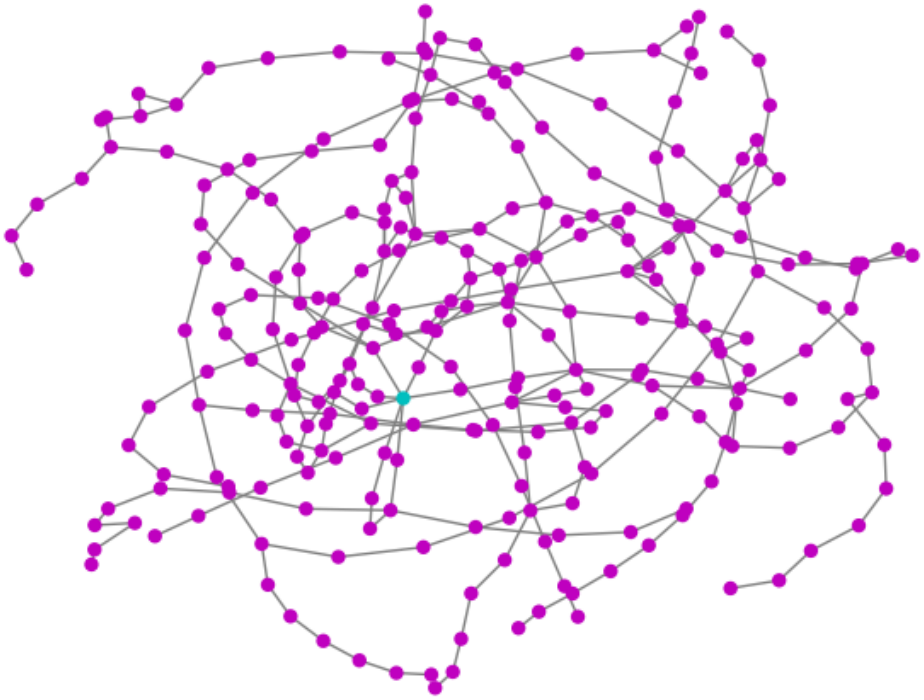
```
#old dataset - just london underground stations

#central nodes appear to be different, but they match up to the central nodes␣
 ↪calculated w/ updated dataset

a_old = nx.degree_centrality(london_underground_G_old)
b_old = nx.closeness_centrality(london_underground_G_old)
c_old = nx.eigenvector_centrality(london_underground_G_old)
d_old = nx.betweenness_centrality(london_underground_G_old)

print('The station with highest degree centrality is', max(a_old, key=lambda key:
 ↪ a_old[key]))
print('The station with highest closeness centrality is', max(b_old, key=lambda␣
 ↪key: b_old[key]))
print('The station with highest eigenvector centrality is', max(c_old,␣
 ↪key=lambda key: c_old[key]))
print('The station with highest betweenness centrality is', max(d_old,␣
 ↪key=lambda key: d_old[key]))
```

```
The station with highest degree centrality is 26 which is finsburypark station
The station with highest closeness centrality is 66 which is eastindia station
The station with highest eigenvector centrality is 67 which is greenpark station
The station with highest betweenness centrality is 26 which is finsburypark
station

The station with highest degree centrality is 35
The station with highest closeness centrality is 58
The station with highest eigenvector centrality is 59
The station with highest betweenness centrality is 50
```

[613]:
```
colors = ['m'] * len(london_underground_G)
len(london_underground_G)
colors[26] = 'c'
nx.draw(london_underground_G, pos = coordinates, node_color=colors,␣
 ↪node_size=25, edge_color='grey', with_labels=False)
```

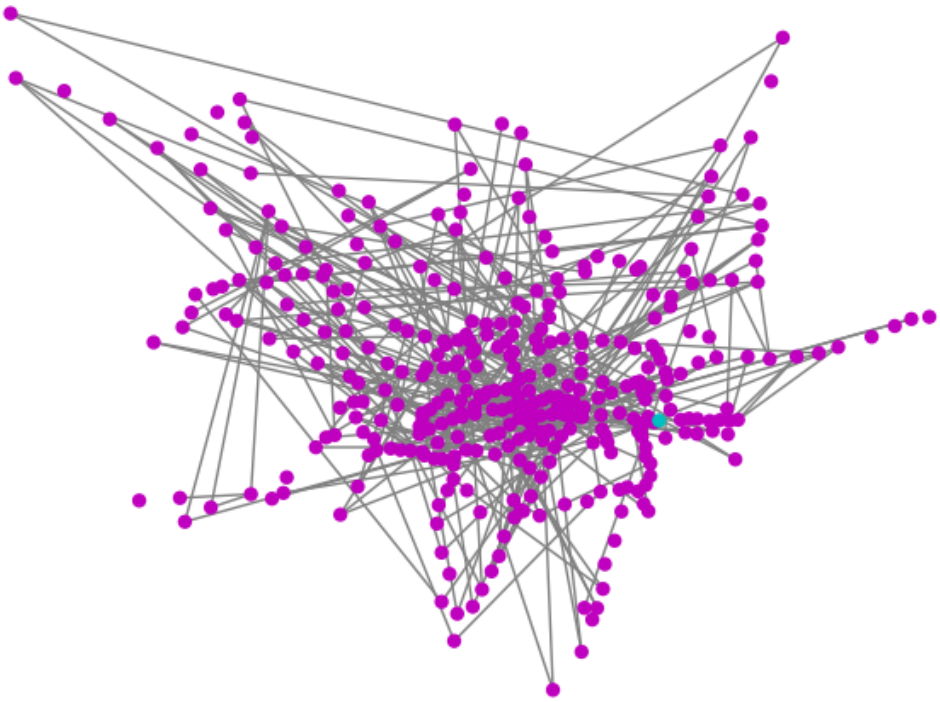


[614]:
```
colors = ['m'] * len(london_underground_G_old)
len(london_underground_G_old)
colors[35] = 'c'
nx.draw(london_underground_G_old, l_pos_old, node_color=colors, node_size=25,␣
 ↪edge_color='grey', with_labels=False)
```

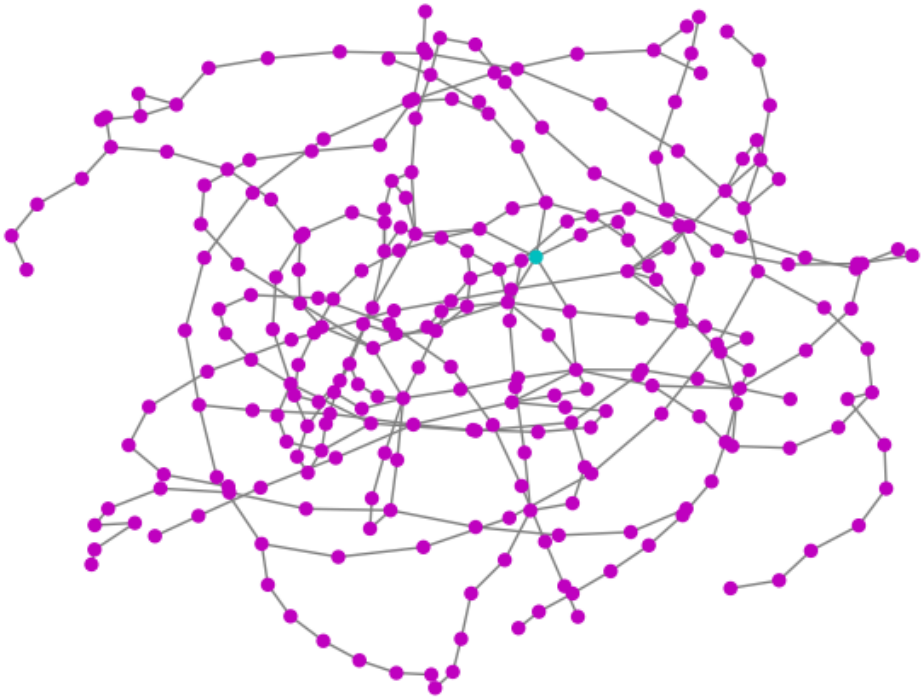


The above network visualizations display one of the four metrics of centrality in which node 26 (node 35 in the old dataset), or Finsbury Park station, has the highest degree centrality.

[615]:
```
colors = ['m'] * len(london_underground_G)
len(london_underground_G)
colors[66] = 'c'
nx.draw(london_underground_G, pos = coordinates, node_color=colors,␣
 ↪node_size=25, edge_color='grey', with_labels=False)
```

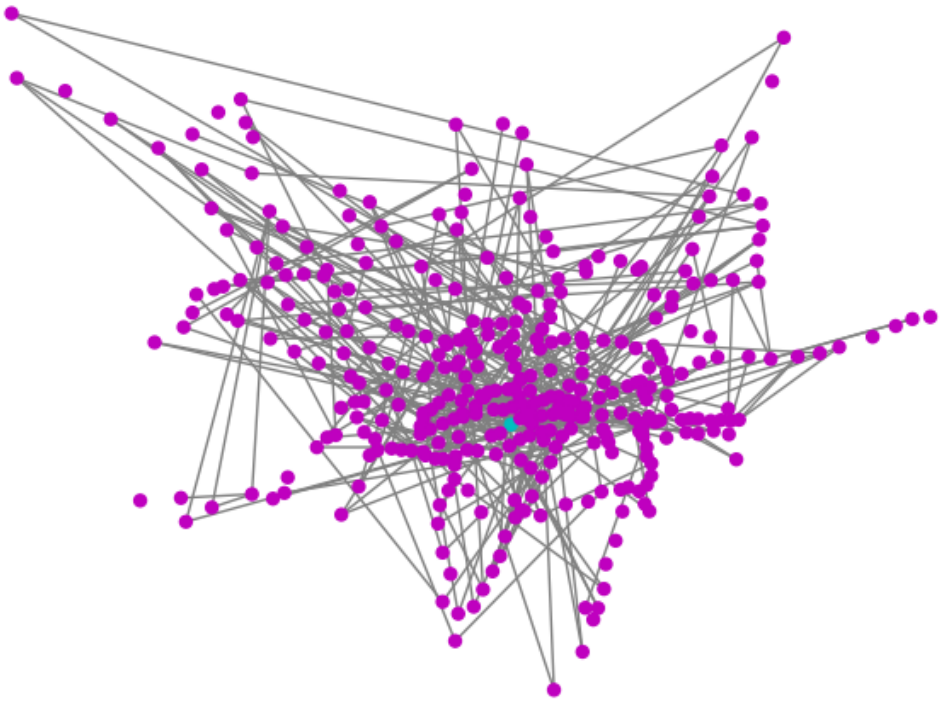


[616]:
```
colors = ['m'] * len(london_underground_G_old)
len(london_underground_G_old)
colors[58] = 'c'
nx.draw(london_underground_G_old, l_pos_old, node_color=colors, node_size=25,␣
↪edge_color='grey', with_labels=False)
```

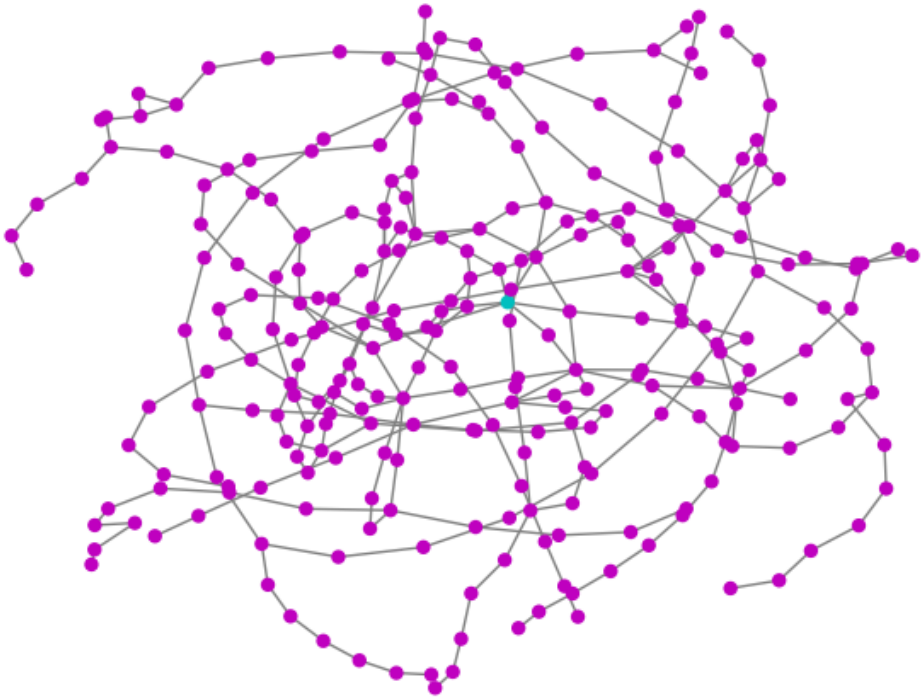


The above network displays one of the four metrics of centrality in wich node 66 (node 58 in old dataset), or East India station, has the highest closeness centrality.

[617]:
```
colors = ['m'] * len(london_underground_G)
len(london_underground_G)
colors[67] = 'c'
nx.draw(london_underground_G, pos = coordinates, node_color=colors,␣
 ↪node_size=25, edge_color='grey', with_labels=False)
```

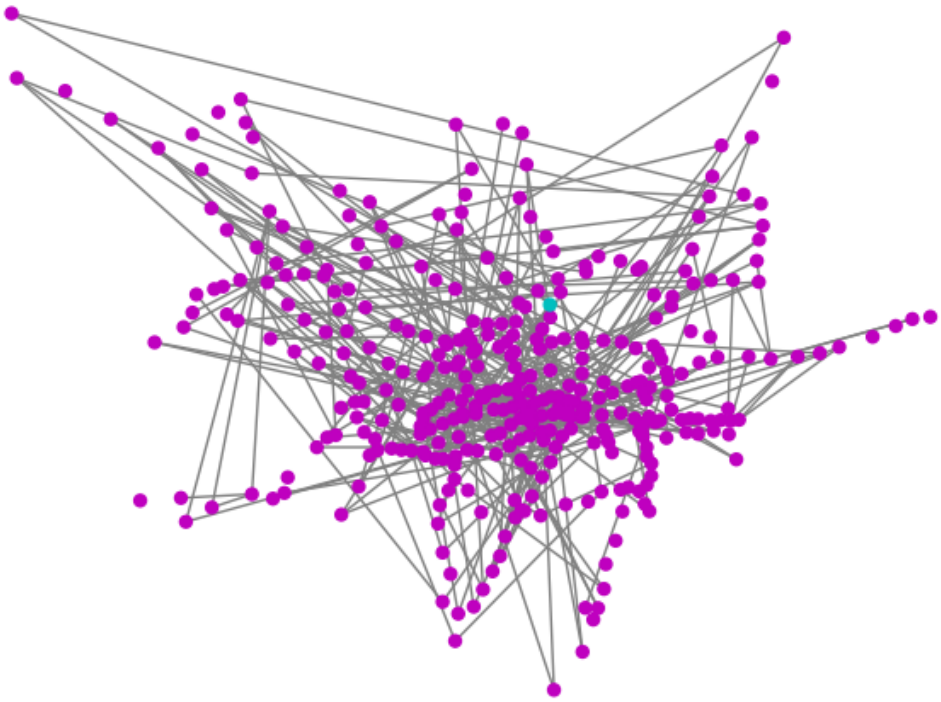


[618]:
```
colors = ['m'] * len(london_underground_G_old)
len(london_underground_G_old)
colors[59] = 'c'
nx.draw(london_underground_G_old, l_pos_old, node_color=colors, node_size=25,␣
↪edge_color='grey', with_labels=False)
```



The above network displays one of four metrics of centrality in which node 67 (node 59 in the old dataset), or Green Park station, has the highest eigenvector centrality.

[619]:
```
colors = ['m'] * len(london_underground_G)
len(london_underground_G)
colors[26] = 'c'
nx.draw(london_underground_G, pos = coordinates, node_color=colors,␣
 ↪node_size=25, edge_color='grey', with_labels=False)
```



The above network displays one of four metrics of centrality in which node 26, or Finsbury Park station, has the highest betweenness centrality.

Note: Using the old dataset changed the node with the highest betwenness centrality, so its graph was ommitted.

We find that the betweenness centrality measure most accurately defines a central node to our network. Thus, Finsburry Park Station is our most central node. This most central node is the node that falls on the greatest number of shortest paths between two nodes. Intuitively, Finsbury Park station will be a stop along the most possible trips using the tube. So in taking some arbitrary route in the tube, one is most likely to pass through this station.

Interestingly, Finsbury Park station also has the highest degree centrality, which further proves its importance as a hub.

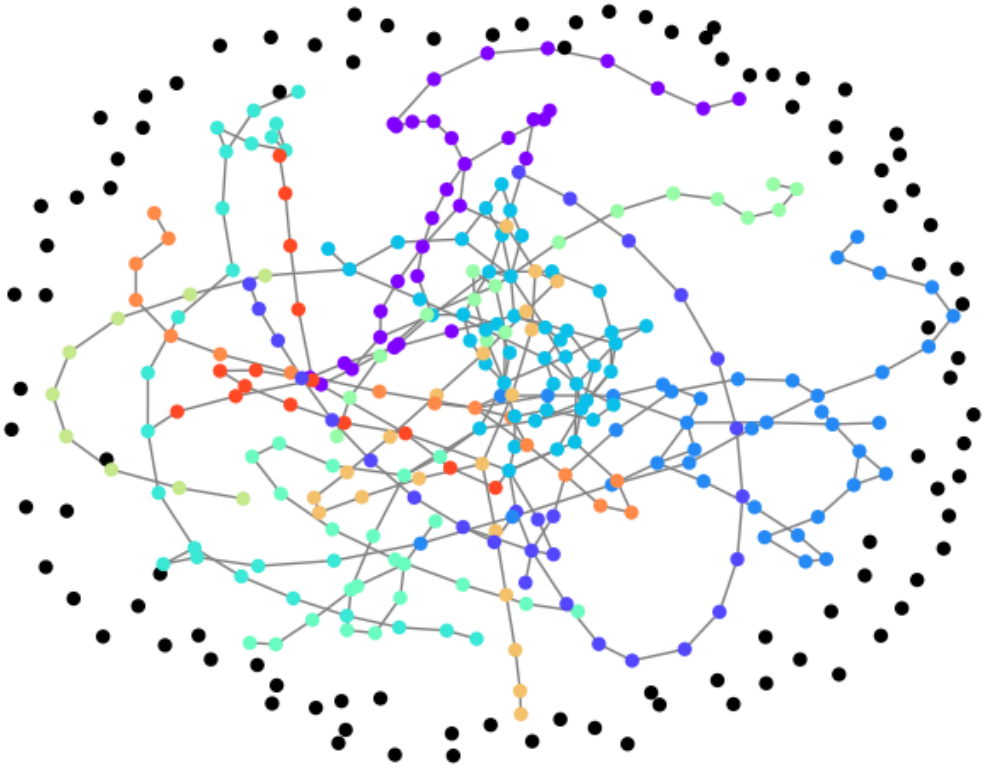
## 4 Graph Partitioning

[636]:
```
pos_part = coordinates
#nx.draw(london_underground_G, pos = pos_part, node_color='m', node_size=25,␣
↪edge_color='lightgrey',  with_labels=True, width = .5)
just_underground = nx.node_connected_component(london_underground_G, 41)
```

```
G_sub = nx.subgraph (london_underground_G, just_underground)
just_underground_coords = []
for node in just_underground:
    just_underground_coords.append(coordinates[node])
```

271

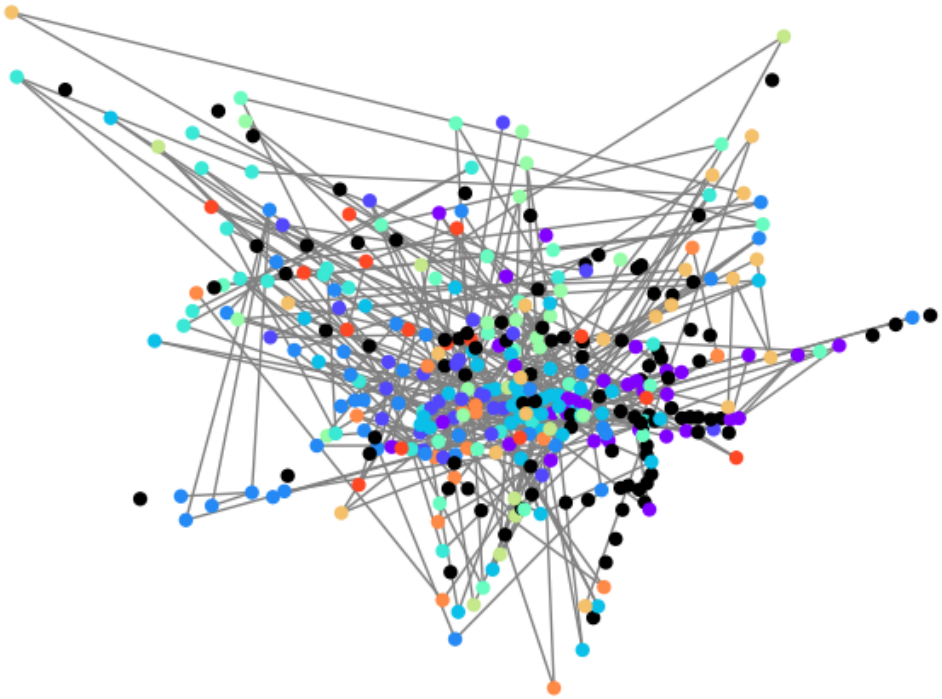
[621]:
```
comm = nx.algorithms.community.girvan_newman(G_sub)
#l_u_communities = tuple(sorted(c) for c in next(comm))

k = 11 #number of partitions

limited = itertools.takewhile(lambda c: len(c) <= k, comm)

for communities in limited:
    l_u_communities = (tuple(sorted(c) for c in communities))


def Plot_Comm(Network, C, position = None):
    cmap = cmx.get_cmap(name='rainbow')
    N = len(Network.nodes())
    K = len(C)
    color_map = ['k']*N
    for i in range(K):
        for j in range(len(C[i])):
            color_map[ C[i][j] ] = cmap(i/K)
    if position is None:
        pos = nx.spring_layout(Network, k=0.25,iterations=20)
    else:
        pos = position


    nx.draw(Network, pos, node_color=color_map, node_size=25, edge_color='grey',␣
 ↪with_labels=False)
    plt.show()
    return




commT = nx.algorithms.community.girvan_newman(london_underground_G)
l_u_communitiesT = tuple(sorted(c) for c in next(commT))



Plot_Comm(london_underground_G, l_u_communities, pos)
```



We decided to disregard the bus stations in our partitioning, and only partitioned the giant component (the train stations) with 11 partitions since there's 11 train lines in the london underground network. On the non-geographical model, these partitions can be viewed clearly in the visualization above.

[622]:
```
comm = nx.algorithms.community.girvan_newman(G_sub)
#l_u_communities = tuple(sorted(c) for c in next(comm))

k = 11 #number of partitions

limited = itertools.takewhile(lambda c: len(c) <= k, comm)

for communities in limited:
    l_u_communities = (tuple(sorted(c) for c in communities))


def Plot_Comm(Network, C, position = None):
    cmap = cmx.get_cmap(name='rainbow')
    N = len(Network.nodes())
    K = len(C)
    color_map = ['k']*N
```

```
    for i in range(K):
        for j in range(len(C[i])):
            color_map[ C[i][j] ] = cmap(i/K)
    if position is None:
        pos = nx.spring_layout(Network, k=0.25,iterations=20)
    else:
        pos = position


    nx.draw(Network, pos = coordinates, node_color=color_map, node_size=25,␣
↪edge_color='grey', with_labels=False)
    plt.show()
    return




commT = nx.algorithms.community.girvan_newman(london_underground_G)
l_u_communitiesT = tuple(sorted(c) for c in next(commT))



Plot_Comm(london_underground_G, l_u_communities, pos)
```



The partitions are much less defined, however, in our geographical model of the network.

## 5 Clustering and Shortest Path

[623]:
```
clustering = nx.algorithms.clustering(london_underground_G)
print('The node with lowest clustering coefficient is', min(clustering,␣
 ↪key=lambda key: clustering[key]))

average_cc = nx.algorithms.average_clustering(london_underground_G)
print('The average clustering coefficient for the London Underground w/ bus␣
 ↪stations network is ', average_cc)
```

```
The node with lowest clustering coefficient is 0
The average clustering coefficient for the London Underground w/ bus stations
network is  0.02285456187895212
```

[624]:
```
clustering_old = nx.algorithms.clustering(london_underground_G_old)
print('The node with lowest clustering coefficient is', min(clustering_old,␣
 ↪key=lambda key: clustering_old[key]))

average_cc_old = nx.algorithms.average_clustering(london_underground_G_old)
print('The average clustering coefficient for the London Underground network␣
 ↪(using the old dataset) is ', average_cc_old)
```

```
The node with lowest clustering coefficient is 0
The average clustering coefficient for the London Underground network (using the
old dataset) is  0.03111931119311194
```

Considering this is the larger dataset, including bus stations, this node could be disconnected hence its low clustering coefficient of 0 or it could just be a tube station whose neighbors don't connect with any of each other, which is the case for the older dataset, as seen below.

The average clustering coefficient for the London Undergound network excluding bus stations is higher, which makes sense because all the disconnectd bus stations bring down the clustering coefficient of the network that does include them in the calculation.

The network above displays the node 0, or Abbey Road station, with the lowest clustering coefficient of 0. Of Abbey Road station's four connections, Westham station, Acton Central Station, Willesden Junction station, and Acton Town station, none of them are connected amongst each other. In the context of the underground, the only way to commute via tube from any of those previously mentioned stations to another is to travel to Abbey Road station. The average clustering coefficient for the London Underground network is 0.03111931119311194. This means that, on average, the only way to commute between two stations that share a common neighbor is to travel to the station they share an edge with.

Note: Other nodes may also have clustering coefficient of 0, but the min function only returns one of these.

[628]:
```
nx.average_shortest_path_length(london_underground_G_old)
```

[628]:
```
13.963318299849664
```

The average shortest path of approximately 13.9633 represents the shortest number of stations (on average) that it takes to travel between all of the pairs of stations.

Note: The old dataset needed to be used for this calculation because it only works on connected graphs.

[629]:
```
nx.shortest_path(london_underground_G_old, source=1, target=123)
```

[629]:
```
[1, 108, 107, 22, 17, 20, 70, 123]
```

Given a source station and a target station we could find the shortest path between the two, which could be useful to estimate the time it takes to take from origin to destination given that the time the train stops at a station is known and uniform. Otherwise, it could help a traveler know how many stops are left on their trip so they can be on the lookout and aware of when they have to get off at their desired stop.

Note: The old dataset needed to be used for this calculation because it only works on connected graphs.

## 6 Miscellaneous

[630]:
```
#Used this article to plot graph geographically
#https://towardsdatascience.com/
 ↪easy-steps-to-plot-geographic-data-on-a-map-python-11217859a2db
df = pd.read_csv(r"/Users/hedavamsolano/Documents/Winter Term 2023/Final_Project/
 ↪london_coords.txt",  delim_whitespace=True)
df.head()
```

[630]:

| | latitude | longitude |
|---|---|---|
| 0 | 51.531952 | 0.003738 |
| 1 | 51.528526 | 0.005332 |
| 2 | 51.508758 | -0.263416 |
| 3 | 51.532234 | -0.243895 |
| 4 | 51.503071 | -0.280288 |

[631]:
```
df.columns
```

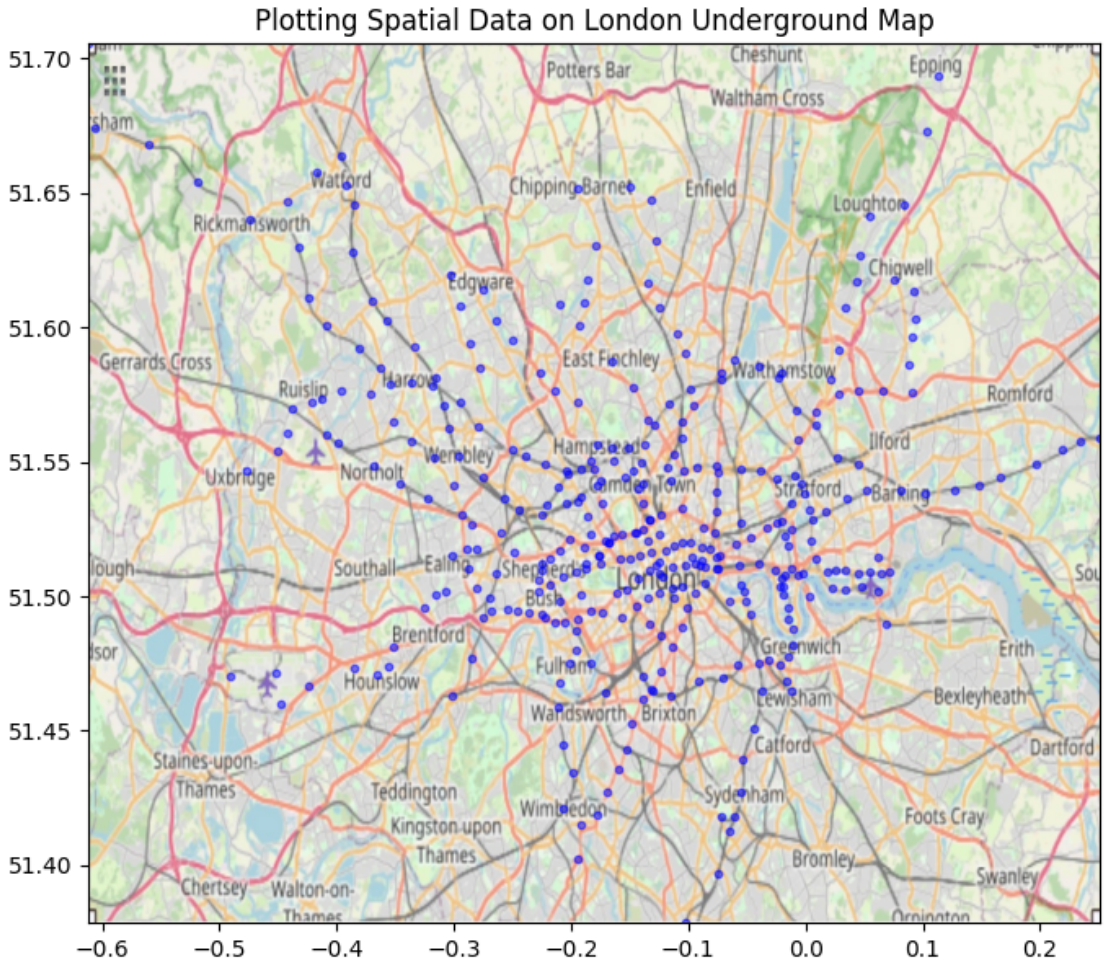
[631]:
```
Index(['latitude', 'longitude'], dtype='object')
```

[632]:
```
BBox = ((df.longitude.min(),   df.longitude.max(),
         df.latitude.min(), df.latitude.max()))
print(BBox)
```

```
(-0.61141518839627, 0.2514152239117, 51.378552186111, 51.705380095434)
```

[633]:
```
fig, ax = plt.subplots(figsize = (8,7))
ax.scatter(df.longitude, df.latitude, zorder=1, alpha= 0.5, c='b', s=10)
ax.set_title('Plotting Spatial Data on London Underground Map')
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])
ruh_m = plt.imread(r'/Users/hedavamsolano/Documents/Winter Term 2023/
 ↪Final_Project/london_undergound_map.png')
ax.imshow(ruh_m, zorder=0, extent = BBox, aspect= 'auto')
```

[633]:
```
<matplotlib.image.AxesImage at 0x16cc2f040>
```




[634]:
```
weight_attr = nx.get_edge_attributes(london_underground_G, "weight")
print(weight_attr)
```

```
{(0, 76): 2.0, (0, 105): 1.0, (0, 218): 1.0, (0, 320): 2.0, (2, 223): 1.0, (2,
257): 1.0, (3, 4): 1.0, (3, 5): 1.0, (3, 159): 2.0, (3, 300): 1.0, (4, 5): 1.0,
(5, 214): 1.0, (5, 220): 1.0, (5, 344): 1.0, (6, 7): 2.0, (6, 294): 1.0, (7, 8):
```

1.0, (7, 33): 1.0, (7, 38): 3.0, (7, 57): 1.0, (8, 9): 2.0, (8, 294): 1.0, (9, 347): 2.0, (12, 13): 1.0, (12, 353): 1.0, (13, 299): 1.0, (14, 15): 1.0, (15, 116): 1.0, (15, 121): 1.0, (18, 19): 1.0, (18, 180): 1.0, (19, 38): 1.0, (20, 21): 1.0, (20, 168): 1.0, (21, 261): 1.0, (22, 23): 1.0, (22, 72): 1.0, (23, 310): 1.0, (24, 25): 1.0, (24, 94): 1.0, (25, 96): 1.0, (25, 190): 1.0, (25, 191): 1.0, (26, 27): 1.0, (26, 28): 2.0, (26, 29): 3.0, (26, 30): 1.0, (26, 174): 1.0, (26, 186): 1.0, (26, 349): 1.0, (27, 66): 1.0, (27, 67): 1.0, (27, 278): 1.0, (28, 48): 1.0, (28, 314): 1.0, (29, 183): 3.0, (30, 67): 1.0, (31, 32): 1.0, (31, 126): 1.0, (32, 357): 1.0, (33, 34): 1.0, (33, 38): 1.0, (33, 118): 1.0, (33, 179): 1.0, (34, 55): 1.0, (34, 68): 1.0, (34, 342): 1.0, (36, 37): 3.0, (36, 38): 3.0, (37, 180): 3.0, (39, 40): 1.0, (39, 169): 2.0, (40, 51): 1.0, (41, 42): 1.0, (41, 198): 1.0, (42, 184): 1.0, (43, 44): 1.0, (43, 45): 1.0, (43, 220): 2.0, (44, 45): 1.0, (44, 160): 2.0, (44, 161): 1.0, (44, 195): 1.0, (44, 260): 1.0, (46, 47): 2.0, (46, 48): 2.0, (47, 161): 2.0, (47, 245): 1.0, (47, 309): 1.0, (48, 173): 1.0, (48, 291): 1.0, (48, 312): 1.0, (48, 313): 1.0, (51, 52): 1.0, (52, 146): 1.0, (53, 54): 1.0, (53, 200): 1.0, (54, 98): 1.0, (55, 56): 1.0, (56, 101): 1.0, (57, 58): 1.0, (58, 75): 2.0, (58, 218): 1.0, (58, 347): 2.0, (59, 60): 2.0, (59, 178): 2.0, (60, 108): 2.0, (61, 62): 1.0, (61, 335): 1.0, (66, 67): 1.0, (66, 212): 1.0, (66, 213): 1.0, (66, 251): 1.0, (66, 319): 1.0, (67, 182): 1.0, (67, 205): 1.0, (67, 212): 1.0, (68, 69): 1.0, (69, 175): 1.0, (69, 276): 1.0, (70, 71): 1.0, (70, 249): 1.0, (71, 300): 1.0, (72, 360): 1.0, (75, 76): 2.0, (77, 78): 1.0, (77, 130): 1.0, (78, 200): 1.0, (79, 80): 1.0, (80, 129): 1.0, (80, 256): 1.0, (80, 350): 1.0, (85, 186): 1.0, (85, 264): 1.0, (87, 88): 1.0, (87, 331): 1.0, (88, 150): 1.0, (89, 90): 1.0, (89, 172): 1.0, (90, 130): 1.0, (93, 94): 1.0, (93, 180): 1.0, (96, 180): 1.0, (98, 99): 1.0, (98, 100): 1.0, (98, 261): 1.0, (99, 100): 1.0, (99, 180): 2.0, (99, 182): 2.0, (101, 104): 1.0, (104, 105): 1.0, (108, 109): 2.0, (109, 294): 2.0, (113, 114): 1.0, (113, 345): 1.0, (114, 270): 1.0, (116, 124): 1.0, (117, 118): 1.0, (117, 134): 1.0, (119, 120): 2.0, (119, 212): 1.0, (119, 280): 1.0, (120, 178): 2.0, (120, 179): 2.0, (120, 213): 2.0, (122, 123): 1.0, (122, 209): 1.0, (123, 331): 1.0, (124, 295): 1.0, (125, 126): 1.0, (125, 129): 1.0, (131, 132): 1.0, (131, 357): 1.0, (132, 343): 1.0, (133, 134): 1.0, (133, 280): 1.0, (134, 205): 1.0, (134, 244): 1.0, (139, 140): 1.0, (139, 295): 1.0, (146, 147): 1.0, (147, 176): 1.0, (150, 151): 1.0, (151, 356): 1.0, (156, 157): 1.0, (156, 297): 1.0, (157, 264): 1.0, (158, 159): 1.0, (158, 298): 1.0, (159, 299): 1.0, (160, 161): 1.0, (160, 199): 3.0, (162, 163): 1.0, (162, 222): 1.0, (163, 337): 1.0, (164, 165): 2.0, (164, 166): 2.0, (165, 231): 1.0, (165, 328): 1.0, (166, 252): 2.0, (167, 168): 1.0, (167, 185): 1.0, (169, 321): 2.0, (170, 171): 1.0, (170, 340): 1.0, (171, 315): 1.0, (173, 174): 1.0, (175, 179): 1.0, (175, 256): 1.0, (176, 246): 1.0, (179, 213): 1.0, (179, 276): 1.0, (179, 342): 1.0, (180, 183): 3.0, (180, 244): 1.0, (182, 204): 1.0, (184, 209): 1.0, (185, 293): 1.0, (185, 364): 1.0, (186, 187): 1.0, (186, 271): 1.0, (187, 349): 1.0, (190, 292): 1.0, (191, 335): 1.0, (194, 195): 1.0, (194, 315): 1.0, (197, 198): 1.0, (197, 329): 1.0, (199, 272): 1.0, (199, 338): 2.0, (201, 202): 2.0, (201, 203): 2.0, (203, 279): 2.0, (204, 205): 1.0, (205, 280): 1.0, (210, 211): 1.0, (210, 302): 1.0, (211, 221): 1.0, (212, 280): 1.0, (213, 348): 2.0, (214, 263): 1.0, (218, 281): 1.0, (220, 327): 1.0, (221, 222): 1.0, (222, 298): 1.0, (223, 224): 1.0, (224, 304): 1.0, (226, 228): 1.0, (228, 262): 1.0, (229, 230): 1.0,

```
(229, 231): 1.0, (229, 301): 1.0, (230, 305): 1.0, (233, 234): 1.0, (233, 235):
1.0, (233, 236): 1.0, (234, 236): 1.0, (235, 247): 1.0, (236, 237): 1.0, (239,
240): 1.0, (240, 358): 1.0, (241, 242): 2.0, (241, 243): 2.0, (242, 252): 2.0,
(245, 337): 1.0, (246, 361): 1.0, (247, 248): 1.0, (248, 249): 1.0, (251, 272):
1.0, (257, 258): 1.0, (258, 266): 1.0, (262, 303): 1.0, (263, 330): 1.0, (266,
267): 1.0, (267, 291): 1.0, (270, 271): 1.0, (271, 297): 1.0, (271, 305): 1.0,
(273, 274): 2.0, (273, 275): 2.0, (274, 279): 2.0, (275, 313): 2.0, (277, 278):
1.0, (277, 309): 1.0, (281, 282): 1.0, (282, 285): 1.0, (282, 286): 1.0, (285,
339): 1.0, (286, 329): 1.0, (292, 360): 1.0, (295, 296): 1.0, (296, 306): 1.0,
(301, 307): 1.0, (302, 334): 1.0, (303, 304): 1.0, (306, 307): 1.0, (310, 311):
1.0, (313, 314): 1.0, (318, 319): 1.0, (318, 350): 1.0, (319, 338): 2.0, (319,
348): 2.0, (320, 321): 2.0, (327, 344): 1.0, (328, 341): 1.0, (331, 339): 1.0,
(333, 334): 1.0, (333, 365): 1.0, (340, 367): 1.0, (341, 353): 1.0, (358, 364):
1.0, (366, 367): 1.0}
```

[635]:
```python
def drop_weights(london_underground_G):
    '''Drop the weights from a networkx weighted graph.'''
    for node, edges in nx.to_dict_of_dicts(london_underground_G).items():
        for edge, attrs in edges.items():
            attrs.pop('weight', None)

drop_weights(london_underground_G)

#https://stackoverflow.com/questions/72045825/remove-weights-from-networkx-graph
```

# 7 Conclusion

Considering this network in terms of its central nodes, we have gained a new understanding of what it means to be a 'central' station. We found betweenness centrality to be a good measure of a central station. We've also found that data network science (at least an introductory level) has a vastly different interpretation of centrality and clusters than an urban planner when considering geography and location.

If we could find a way to weigh edges by the popularity of the travel route, we wonder if we could determine a different understanding of the most central nodes, since we may want to consider the most central stations to be those that the most people pass through. So, having access to the number of people that pass through each station/path might be useful metadata.