1 INTRODUCTION

This report presents and explains the results derived from urban modelling methods to critically evaluate the resilience of London's underground network and the limitations of the methods adopted. In the following

2 LONDON'S UNDERGROUND RESILIENCE

In this part, the infrastructure of the underground network is discussed regarding the layout of the stations.

2.1 TOPOLOGICAL NETWORK

The importance of stations (nodes) are measured on centrality and impact dimension, and resilience of the tube network is tested via pressure testing where certain nodes being removed from the network in this section.

2.1.1 Centrality measures

In the context of London Tube Network (as shown in Figure 1 where the nodes in yellow represents the stations and lines connecting nods represent the tracks), the centrality measures to characterize nodes in terms of their quantified importance are the degree centrality, the betweenness centrality, the topological closeness centrality and the topological eigenvector centrality. As the eigenvector centrality incorporates degree centrality, the latter 3 measures are adopted in this analysis.

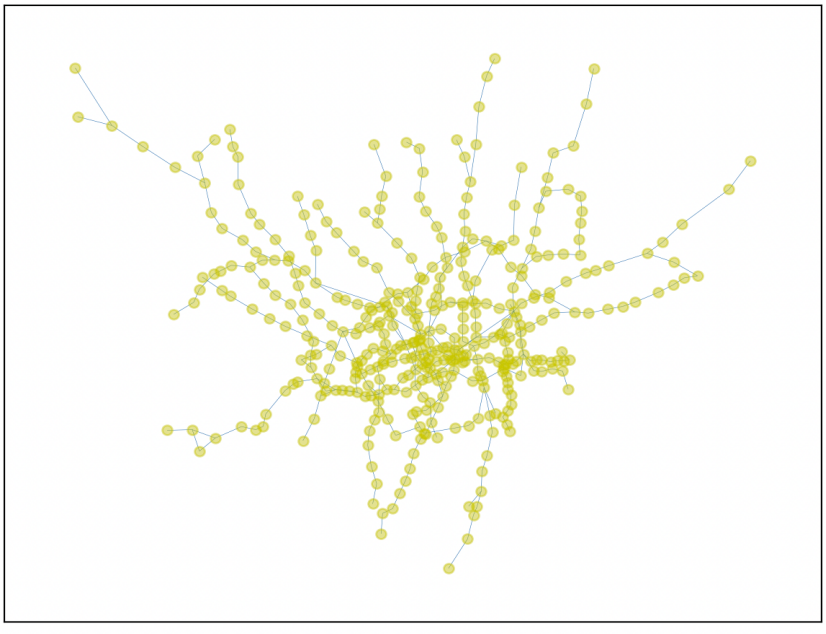


Figure 1 London Tube Network

2.1.1.1 Betweenness Centrality

In the network, betweenness centrality reflects the number of shortest paths through a node. The more the number of shortest paths passing through a node, the higher the betweenness centrality. Between centrality is

calculated by the ratio of the shortest path passing through a node and connecting the two nodes in the network to the total number of shortest path lines between the two nodes. The calculation formula is as follows:

$$C_B(\mathrm{p_i}) = \sum_{j=1}^{N} \sum_{k=1}^{j-1} \frac{g_{jk}(\mathrm{p_i})}{g_{jk}}$$

Where denominator represents the accumulation of the number of shortest paths between all nodes, and the numerator represents the accumulation of the number of shortest paths between all nodes (pi).

If the betweenness centrality of a certain node is high, this node tend to appear amidst the shortest path between other nodes, which is to say, it is capable of block the shortest path between other nodes, this node is more capable of determining the interchanges between other nodes. In this way, it will have a greater impact on the transfer of the whole network.

In the context of London tube network, betweenness centrality examines the control of one station over the passenger dissemination of other stations. If a station is located on multiple shortest paths of other stations, it scores high in betweenness centrality. If so, the probability the passengers from the connected nodes pass through this station to get to other stations becomes higher. In other words, this station assumes the role of a core station.

The calculation results are visualized in Figure 2 (right) with a heat map adapted from Figure 1, where the color indicates the level of betweenness centrality, transiting from the highest indicated with yellow and lowest purple. Moreover, the top 10 stations with highest scores, i.e. the most crucial ones in this regard, are listed at the left.

	betweenness_topo
Stratford	23768.093434
Bank and Monument	23181.058947
Liverpool Street	21610.387049
King's Cross St. Pancras	20373.521465
Waterloo	19464.882323
Green Park	17223.622114
Euston	16624.275469
Westminster	16226.155916
Baker Street	15287.107612
Finchley Road	13173.758009

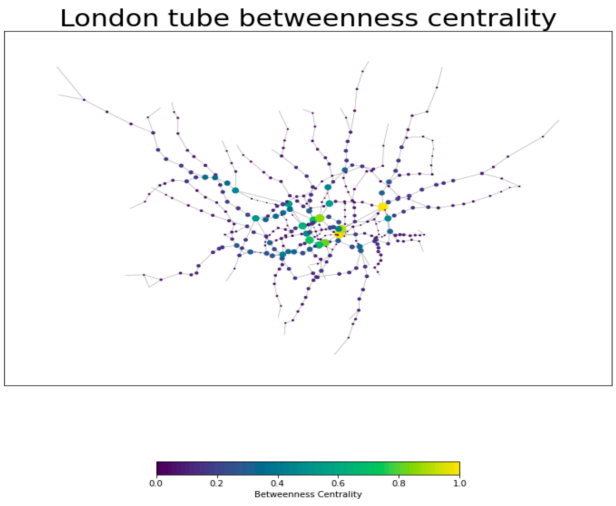


Figure 2 Top 10 nodes with highest betweenness centrality (left) and heat map of betweenness centrality (right)

The range of betweenness centrality among the top 10 stations spreads quite widely, among which Stratford and Bank & Monument are the core of the core stations in the sense that they 'guard' the most routes of passenger flows between stations.

2.1.1.2 Topological Closeness Centrality

Closeness centrality considers the average length of the shortest path from each node to other nodes. In other words, for a node, the closer it is to other nodes, the higher its centrality. The closeness centrality of nodes can be characterized by the geometric distance between nodes (the number of edges contained in the shortest path

between two nodes). Specifically, if the shortest path from a node to other nodes is very short, the closeness centrality of the node is high. The closeness centrality of this point is based on the sum of the shortest paths from this node to all other nodes in the network normalized to the number of nodes to find the average shortest distance from this node to these other nodes. The formula is as follows

$$C_C(\mathrm{p}_i) = \frac{N-1}{\sum_{k=1}^{N} \mathrm{d}(\mathrm{p}_i, \mathrm{p}_k)}$$

As indicated by the formula, closeness centrality is more similar to the geometric center than between centrality. The smaller the average shortest distance of a node, the greater the closeness centrality. If there is no path reachable between node i and node k, $\mathrm{d}(\mathrm{p}_i, \mathrm{p}_k)$ is defined as infinitively small and $C_C(\mathrm{p}_i)$ infinitively large.

In the context of London tube network, this indicator can be used to measure the time (assuming speed of cars is standardized) passengers take to transmit from the focal station to all other stations. Like between centrality, closeness centrality makes use of the characteristics of the whole tube network, that is, it illustrates the position of a station in the whole network structure. Generally speaking, the higher the closeness centrality of one station the more likely this station being used by as many passengers as possible to get to other stations, i.e. important in the network.

	closeness_topo
Green Park	0.114778
Bank and Monument	0.113572
King's Cross St. Pancras	0.113443
Westminster	0.112549
Waterloo	0.112265
Oxford Circus	0.111204
Bond Street	0.110988
Farringdon	0.110742
Angel	0.110742
Moorgate	0.110314

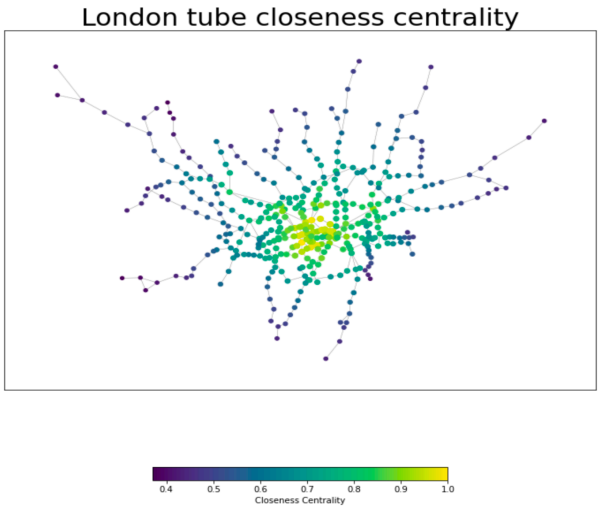


Figure 3 Top 10 nodes with highest closeness centrality (left) and heat map of closeness centrality (right)

Similar to that in section 2.1.1.1, the calculation results are presented with a list of top 10 stations and a diagram adapted from Figure 1 to illustrate the different levels of closeness centrality of stations (see Figure 3). Along the same logic, higher closeness centralities are colored yellow transiting from purple which representing lower closeness centralities.

As predicted, the closer a station's location is to the geometric center, the higher closeness centrality this station features.

2.1.1.3 Topological Eigenvector Centrality

The above measures of centrality are direct indicators. Next, a more complex yet profound indicator that is indirect measure of centrality is introduced, namely, eigenvector centrality. The basic idea of eigenvector centrality is that the centrality of a node is a function of the centrality of adjacent nodes. In other words, the

more important the neighboring nodes the focal node connects with, the more important the focal node is. In short, the centrality of the eigenvector is related to the importance of the neighbor nodes of the node.

$$EC(i) = x_i = c\sum_{j=1}^{n} a_{ij}x_j$$

Where xi indicates the importance of the focal node and xj that of the neighboring nodes. And x is the eigenvector of the matrix A such that $x = cAx$.

The outputs are presented in Figure 4, where

	eigenvector_t
Bank and Monument	0.383427
Liverpool Street	0.328824
Stratford	0.269179
Waterloo	0.249731
Moorgate	0.215137
Green Park	0.197630
Oxford Circus	0.184094
Tower Hill	0.171676
Westminster	0.168610
Shadwell	0.159090

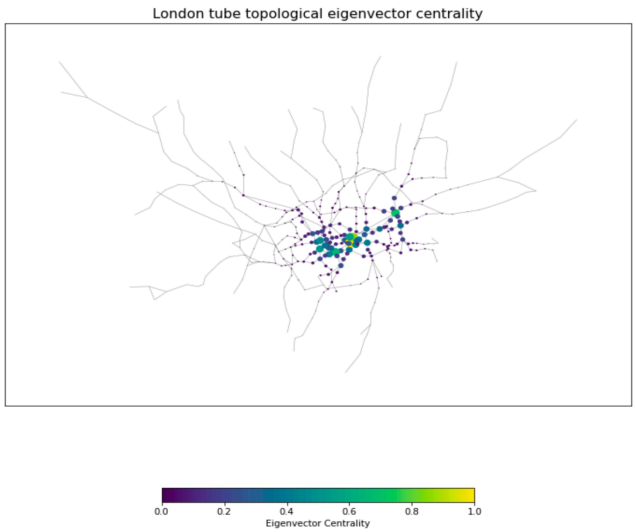


Figure 4 Top 10 nodes with highest eigenvector centrality (left) and heat map of eigenvector centrality (right)

2.1.2 Impact measures

Two metrics were chosen to assess the impact of deleting nodes.

It would be used in the Topological Betweenness Centrality on nodes and the Topological Closeness Centrality on nodes.

The reason for using these two metrics is that they are relatively easier to compare. Unlike the Eigenvector Centrality which relies on surrounding nodes for feedback and is difficult to compare after changes. The Topological Betweenness Centrality on nodes and the Topological Closeness Centrality on nodes are chosen so that by censoring nodes, the changes can be seen visually and the importance of the nodes can be easily determined. And it is quicker to observe the severity caused by the deletion of nodes.

Judging Criteria:

Average clustering coefficient and average global efficiency

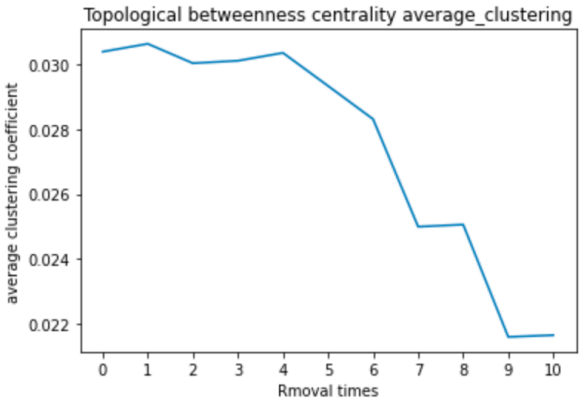
The average clustering coefficient is a global measure of network isolation, reflecting the clustered connectivity around a single node. Therefore the study of this metric is more appropriate for the analysis of London Underground lines and stations. Similarly, in network science, the efficiency of a network is a measure of how

efficiently it exchanges information, and it is also known as communication efficiency. So when analysing the efficiency of certain grids, the method can also be used to evaluate whether which nodes can be pruned.

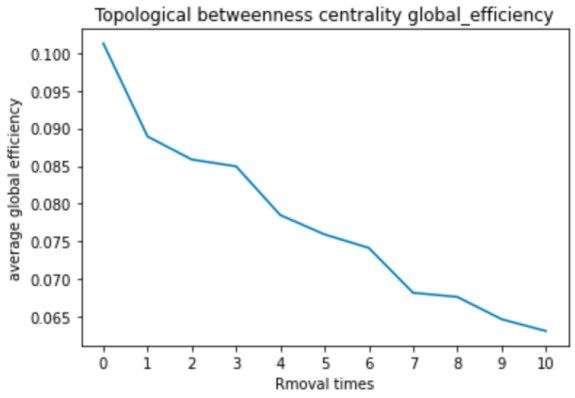
2.1.3 Node removal

A. Non-sequential Removal

The Topological Betweenness Centrality:

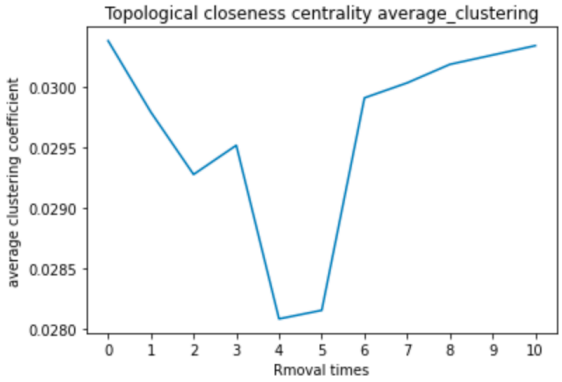


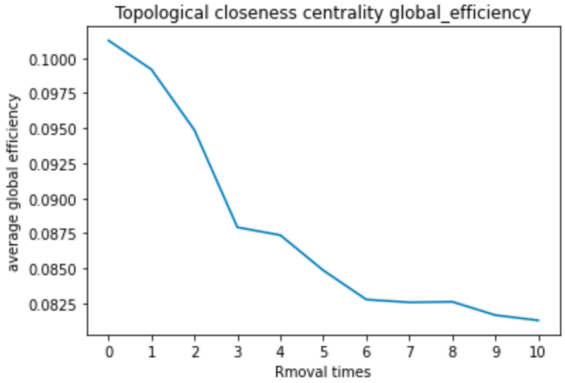
The first few nodes drop slowly and have less impact on the whole, probably because: there are many stations around these nodes that can replace most of the routes of this station. The latter nodes drop fast and have a greater overall impact; the surrounding stations cannot replace most of the routes of these nodes, so they drop faster.



The graph shows that by removing each point in turn from the I.1 table, the Average clustering coefficient and Average global efficiency both decrease, indicating that the top ten sites are very important and that these ten sites have a high influence on the London Underground.

Topological Closeness Centrality on nodes:

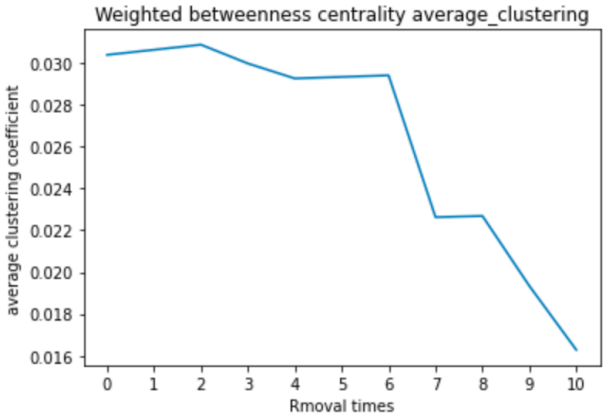


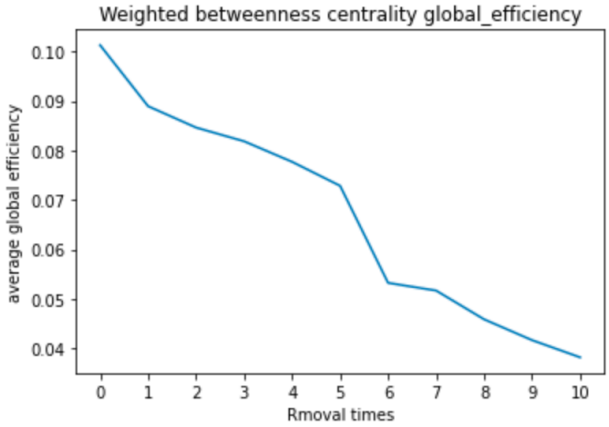


The average global efficiency decreases and the average clustering coefficient decreases and then increases as each point is removed in turn from the I.1 scale, which reflects the clustering connectivity around a single node. The average clustering coefficient reflects the clustering connectivity around a single node, and from a global perspective, the clustering connectivity of individual nodes in the metro network will decrease and then increase after removing a few sites with the highest values.

B) Sequential Removal:

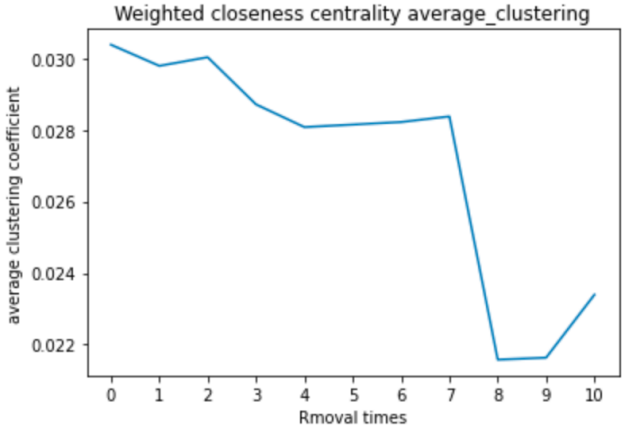
The Topological Betweenness Centrality:

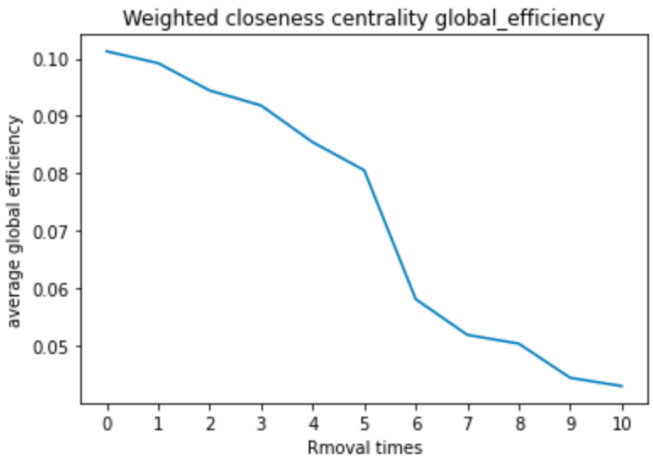




The average clustering coefficient and average global efficiency both drop, and both have a large gap between 5 and 6 times.

Topological Closeness Centrality on nodes：





The Average clustering coefficient and Average global efficiency both decreases, with a slight increase behind the Average clustering coefficient.

The Topological Betweenness Centrality is a better reflection of the impact of removing stations on the rail network, as the measured values are all in a decreasing state. The definition and concept of The Topological Betweenness Centrality is also the most consistent with the rail road analysis.

2.2 FLOWS: WEIGHTED NETWORK

2.2.1

Weighted Betweenness Centrality on nodes:

	betweenness_w
West Hampstead	28401.50
Gospel Oak	21947.00
Finchley Road & Frognal	21446.00
Hampstead Heath	21372.00
Willesden Junction	19166.50
Leicester Square	17097.75
Brondesbury	16526.00
Brondesbury Park	16411.50
Kensal Rise	16299.75
Blackhorse Road	16091.75

Weighted Closeness Centrality on nodes:

	closeness_w
Holborn	0.000079
King's Cross St. Pancras	0.000079
Tottenham Court Road	0.000079
Oxford Circus	0.000079
Leicester Square	0.000078
Piccadilly Circus	0.000078
Charing Cross	0.000078
Chancery Lane	0.000078
Covent Garden	0.000078
Embankment	0.000078
Russell Square	0.000078
Warren Street	0.000077
Euston	0.000077
Bank and Monument	0.000077
Moorgate	0.000077

Weighted Eigenvector Centrality on nodes：

	eigenvector_w
Waterloo	0.527355
Westminster	0.481937
Bank and Monument	0.417409
Green Park	0.374465
Liverpool Street	0.214570
Bond Street	0.156550
Oxford Circus	0.140253
Victoria	0.132111
Moorgate	0.118185
Southwark	0.100186

What can be noticed is that the stations ranked by the three methods change considerably after the addition of weighted value.

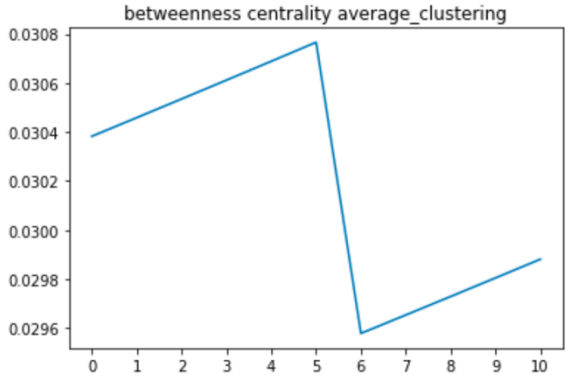
2.2.2

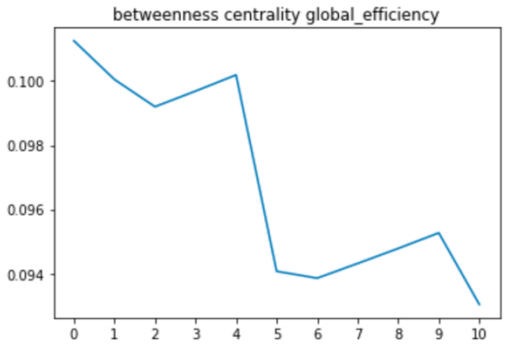
Two metrics were still chosen to assess the impact of deleting sites.

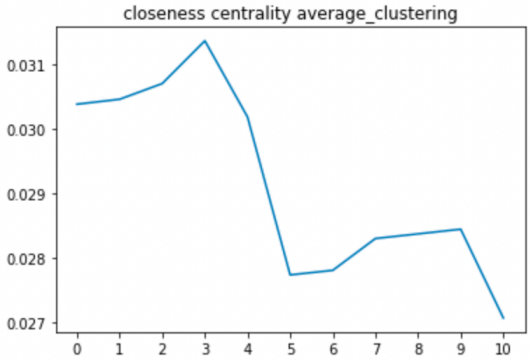
The Topological Betweenness Centrality on nodes and the Topological Closeness Centrality on nodes.

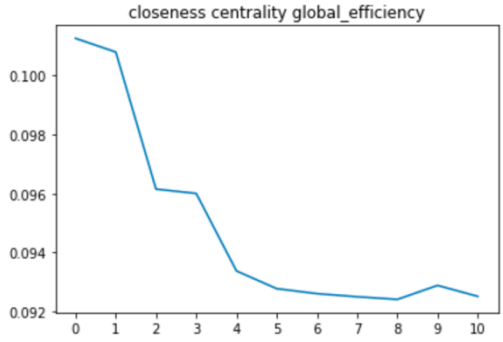
Judging criteria:

Both Average clustering coefficient and Average global efficiency are decreasing until the sixth time, when there is a slight upward trend.









The weighted network metric is not adjusted because the topological betweenness centrality on nodes and topological closeness centrality on nodes can be represented visually by removing nodes. Therefore, no further adjustment is required.

2.2.3

As seen in 2.1 the Topological Closeness Centrality on nodes is the best performing centrality metric as there is a clear downward trend in both Average clustering coefficient and Average global efficiency.

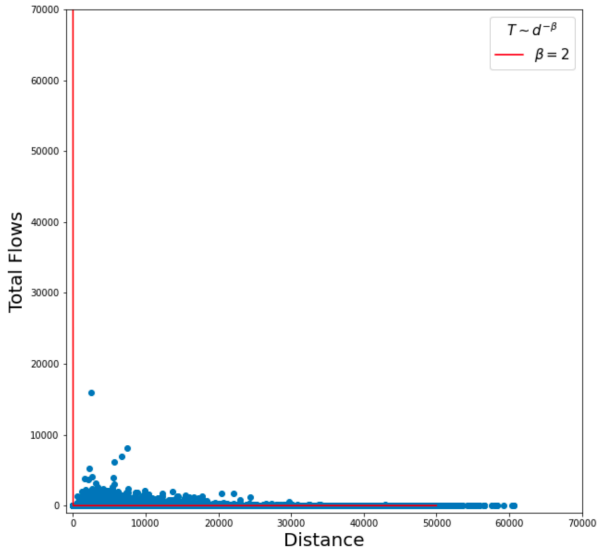
For the Topological Closeness Centrality on nodes, turning off the third node has a significant impact on the metric, as can be seen in the two line graphs.

3 SPATIAL INTERACTION MODELS

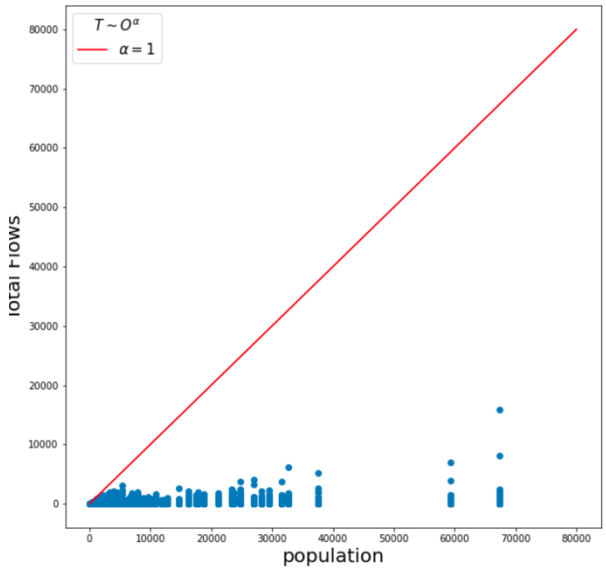
3.1 MODELS AND CALIBRATION

3.1.1

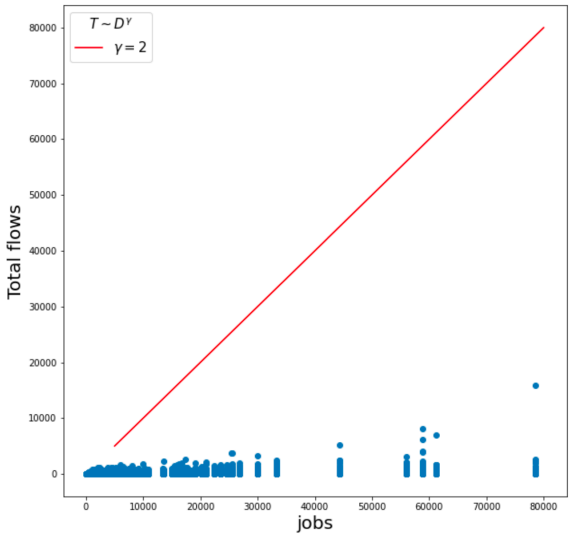
Firstly, plot the Total commuter flows denoted by T against distance denoted by d. and then fit a model line T ~ d^-beta with beta=2. The result is: There is a relationship between commuter flow and distance.



and then fit a model line T ~ O^alpha, for alpha=1, the result is: There is a positive relationship between commuter flows and population.



and then fit a model line T ~ D^gamma. We will fit a line for gamma=2. The result is: There is a positive relationship between commuting flows and work.



3.1.2

First set the model parameters. Model parameters: alpha = 1 gamma = 1 beta = 2 k=1. Model test: R-Squared = 0.28746769499714825, and R-Squared between [0,1], the closer to 1 the better the fit. root Mean Squared Error (RMSE) = 728.385 Then, improving our model by calibrating parameters.

```
formula = 'flows ~ log_population + log_jobs +log_distance'

uncosim = smf.glm(formula = formula,

data=cdatasub,

family=sm.families.Poisson()).fit()
```

Parameters derived from uncosim:

```
K = uncosim.params[0]

alpha = uncosim.params[1]

gamma = uncosim.params[2]

beta = -uncosim.params[3]
```

R-Squared：0.5654865489046319

RMSE：517.258

It can be seen from the conclusions that our model fits better and has a higher level of confidence.

3.2 SCENARIOS

Scenarios A

Model:

Take the variables and produce logarithms of them, then run the regression:

```
formula = 'flows ~ log_population + log_jobs +log_distance'

uncosim = smf.glm(formula = formula,
                  data=cdatasub,
                  family=sm.families.Poisson()).fit()
```

Firstly, assign the parameter values from the model to the appropriate variables:

Result:

K= -8.0291,

alpha=0.9482,

gamma=1.1525,

beta=0.8529

```
cdatasub["unconstrained_increase_cost"] = np.exp(K
                                                 + alpha*cdatasub["log_population"]
                                                 + gamma*cdatasub["log_jobs"]
                                                 - beta*cdatasub["log_distance"])
```

RSqaured=0.3024924032875866

RMSE=665.248

Scenarios B

Model:

```
cdatasub["unconstrained_increase_cost"] = np.exp(K
                                                 + alpha*cdatasub["log_population"]
                                                 + gamma*cdatasub["log_jobs"]
                                                 - beta*cdatasub["log_distance"])
```

Firstly, assign the parameter values from the model to the appropriate variables:

K= -8.0291

alpha=0.9482

gamma=1.1525

beta=0.9

R-Squared = 0.5656407640326113

RMSE = 578.058

Changing the beta value:

When beta = 1.1 :

RSqaured=0.5529440531520704

RMSE=864.129

With different values of beta, it was found that beta 0.9 was more appropriate as the R-Squared was greater and the RMSE was smaller. On balance using the model in scenario B gives better results.

Appendix

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

G = nx.read_graphml('/Users/brianlee/downloads/london.graph.xml')

list(G.nodes(data = True))[0]

fig, ax = plt.subplots(figsize=(15,10))

pos = nx.get_node_attributes(G, 'coords')

nx.draw_networkx_nodes(G,pos,node_size=25,node_color='y',alpha=0.5)
nx.draw_networkx_edges(G,pos,arrows=False,width=0.2,edge_color='#004C99',alpha=1)
#nx.draw_networkx_labels(G,pos, node_labels, font_size=10, font_color='black')

plt.title("The London tube network",fontsize=30)
plt.axis("on")
plt.show()

df = nx.to_pandas_edgelist(G)
df[0:10]

nx.set_node_attributes(G,dict(deg_london),'degree_topo')
```

```
df = pd.DataFrame(index=G.nodes())

df['degree_topo'] = pd.Series(nx.get_node_attributes(G, 'degree_topo'))

df_sorted = df.sort_values(["degree_topo"], ascending=False)
df_sorted[0:10]



# Lets set colour and size of nodes according to betweenness values
degree_values=[(i[1]['degree_topo']) for i in G.nodes(data=True)]

deg_color=[(i[1]['degree_topo']/(max(degree_values))) for i in G.nodes(data=True)]
deg_size=[(i[1]['degree_topo']/(max(degree_values)))*100 for i in G.nodes(data=True)]



pos=nx.get_node_attributes(G, 'coords')

fig, ax = plt.subplots(figsize=(12,12))



nx.draw_networkx_edges(G, pos,edge_color='black',
                       width=0.4)
nod=nx.draw_networkx_nodes(G,
                       pos = pos,
                       node_color= deg_color,
                       node_size= deg_size)

plt.colorbar(nod,label="Degree Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
```

```
plt.title("The London tube degree centrality",fontsize=30)
plt.show()

bet_london_topo=nx.betweenness_centrality(G, normalized=False)
nx.set_node_attributes(G,bet_london_topo,'betweenness_topo')
df = pd.DataFrame(index=G.nodes())

df['betweenness_topo'] = pd.Series(nx.get_node_attributes(G, 'betweenness_topo'))

df_sorted = df.sort_values(["betweenness_topo"], ascending=False)
df_sorted[0:10]

# Lets set colour and size of nodes according to betweenness values
betweenness_t_values=[(i[1]['betweenness_topo']) for i in G.nodes(data=True)]

bet_t_color=[(i[1]['betweenness_topo']/max(betweenness_t_values)) for i in G.nodes(data=True)]
bet_t_size=[(i[1]['betweenness_topo']/max(betweenness_t_values))*100 for i in G.nodes(data=True)]

fig, ax = plt.subplots(figsize=(12,12))

nx.draw_networkx_edges(G, pos,edge_color='black', width=0.4)

nod=nx.draw_networkx_nodes(G, pos = pos, node_color= bet_t_color, node_size= bet_t_size)

plt.colorbar(nod,label="Betweenness Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
plt.title("The London tube betweenness centrality",fontsize=30)
plt.show()
```

```
clos_topo=nx.closeness_centrality(G)
nx.set_node_attributes(G,clos_topo,'closeness_topo')

df = pd.DataFrame(index=G.nodes())

df['closeness_topo'] = pd.Series(nx.get_node_attributes(G, 'closeness_topo'))

df_sorted = df.sort_values(["closeness_topo"], ascending=False)
df_sorted[0:10]

# Lets set color and width of nodes according to the closeness values
clos_t_val=[(i[1]['closeness_topo']) for i in G.nodes(data=True)]

closs_t_color=[(i[1]['closeness_topo']-min(clos_t_val))/(max(clos_t_val)-min(clos_t_val)) for i in G.nodes(data=True)]
closs_t_size=[((i[1]['closeness_topo']-min(clos_t_val))/(max(clos_t_val)-min(clos_t_val))*50) for i in G.nodes(data=True)]

fig, ax = plt.subplots(figsize=(12,12))

nx.draw_networkx_edges(G, pos,edge_color='black',
                       width=0.4)

nod=nx.draw_networkx_nodes(G,
                       pos = pos,
                       node_color= closs_t_color,
                       node_size= closs_t_size)

plt.colorbar(nod,label="Closeness Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
```

```
plt.title("The London tube closeness centrality",fontsize=30)
plt.show()

# Lets set color and width of nodes according to the closeness values
clos_t_val=[(i[1]['closeness_topo']) for i in G.nodes(data=True)]

closs_t_color=[i[1]['closeness_topo']/max(clos_t_val) for i in G.nodes(data=True)]
closs_t_size=[i[1]['closeness_topo']/max(clos_t_val)*50 for i in G.nodes(data=True)]

fig, ax = plt.subplots(figsize=(12,12))

nx.draw_networkx_edges(G, pos,edge_color='black',
                       width=0.4)

nod=nx.draw_networkx_nodes(G,
                       pos = pos,
                       node_color= closs_t_color,
                       node_size= closs_t_size)

plt.colorbar(nod,label="Closeness Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
plt.title("The London tube closeness centrality",fontsize=30)
plt.show()

#compute the igenvector centrality for the network, without using weights:
eig_london_t=nx.eigenvector_centrality(G,max_iter=1000)
nx.set_node_attributes(G,eig_london_t,'eigenvector_t')
df = pd.DataFrame(index=G.nodes())
df['eigenvector_t'] = pd.Series(nx.get_node_attributes(G, 'eigenvector_t'))
```

```
df_sorted = df.sort_values(["eigenvector_t"], ascending=False)
df_sorted[0:10]

# Lets set colour and size of nodes according to eigenvector values
eigenvector_t_values=[(i[1]['eigenvector_t']) for i in G.nodes(data=True)]

eig_t_color=[(i[1]['eigenvector_t']/max(eigenvector_t_values)) for i in G.nodes(data=True)]
eig_t_size=[(i[1]['eigenvector_t']/max(eigenvector_t_values))*100 for i in G.nodes(data=True)]

fig, ax = plt.subplots(figsize=(12,12))

nx.draw_networkx_edges(G, pos,edge_color='gray', width=0.4)

nod=nx.draw_networkx_nodes(G, pos = pos, node_color= eig_t_color, node_size= eig_t_size)

plt.colorbar(nod,label="Eigenvector Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
plt.title("The London tube topological eigenvector centrality",fontsize=15)
plt.show()

#1.3 A)
G1=G.copy()

bet_london_topo_1=nx.betweenness_centrality(G1, normalized=False)

nx.set_node_attributes(G1,bet_london_topo_1,'betweenness_topo')

df_bet_1 = pd.DataFrame(index=G1.nodes())
```

```
df_bet_1['betweenness_topo'] = pd.Series(nx.get_node_attributes(G1, 'betweenness_topo'))

df_sorted_bet_1 = df_bet_1.sort_values(["betweenness_topo"], ascending=False)
df_sorted_bet_1[0:10]

bet_ave_clu = []
bet_glo_eff = []

print(f'average_clustering before removal: {nx.average_clustering(G1)} ')
print(f'global_efficiency before removal: {nx.global_efficiency(G1)} ')

print(nx.info(G1))
bet_ave_clu.append(nx.average_clustering(G1))
bet_glo_eff.append(nx.global_efficiency(G1))

for i in range(10):
    print('-------------------------------------------------------')
    print(f'{i + 1} time node removal')

    betweenness_t_values = [(i[1]['betweenness_topo']) for i in G1.nodes(data=True)]

    maxbets = [n for n in G1.nodes() if G1.nodes[n]['betweenness_topo'] == max(betweenness_t_values)]

    print(f'remove node: {maxbets}')

    G1.remove_nodes_from(maxbets)

    print(nx.info(G1))
```

```
        bet_ave_clu.append(nx.average_clustering(G1))
        bet_glo_eff.append(nx.global_efficiency(G1))

        print(f'average_clustering: {nx.average_clustering(G1)} ')
        print(f'global_efficiency: {nx.global_efficiency(G1)} ')

plt.plot(bet_ave_clu)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average clustering coefficient')
plt.title('Topological betweenness centrality and    average clustering')
plt.show()

plt.plot(bet_glo_eff)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average global efficiency')
plt.title('Topological betweenness centrality and global efficiency')
plt.show()

G2=G.copy()

clos__london_topo_1=nx.closeness_centrality(G2)

nx.set_node_attributes(G2,clos__london_topo_1,'closeness_topo')

df_clo_1 = pd.DataFrame(index=G2.nodes())
```

```
df_clo_1 ['closeness_topo'] = pd.Series(nx.get_node_attributes(G2, 'closeness_topo'))

df_sorted_clo_1 = df_clo_1 .sort_values(["closeness_topo"], ascending=False)
df_sorted_clo_1[0:10]

clo_ave_clu = []
clo_glo_eff = []

print(f'average_clustering before removal: {nx.average_clustering(G2)} ')
print(f'global_efficiency before removal: {nx.global_efficiency(G2)} ')

print(nx.info(G2))
clo_ave_clu.append(nx.average_clustering(G2))
clo_glo_eff.append(nx.global_efficiency(G2))

for i in range(10):
    print('-----------------------------------------------------')
    print(f'{i + 1} time node removal')

    closeness_t_values = [(i[1]['closeness_topo']) for i in G2.nodes(data=True)]

    maxclos = [n for n in G2.nodes() if G2.nodes[n]['closeness_topo'] == max(closeness_t_values)]

    print(f'remove node: {maxclos}')

    G2.remove_nodes_from(maxclos)

    print(nx.info(G2))
```

```
        clo_ave_clu.append(nx.average_clustering(G2))
        clo_glo_eff.append(nx.global_efficiency(G2))

        print(f'average_clustering: {nx.average_clustering(G2)} ')
        print(f'global_efficiency: {nx.global_efficiency(G2)} ')

plt.plot(clo_ave_clu)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average clustering coefficient')
plt.title('Topological closeness centrality and average clustering')
plt.show()

plt.plot(clo_glo_eff)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average global efficiency')
plt.title('Topological closeness centrality and global efficiency')
plt.show()



#1.3 B)
G5=G.copy()
bet_ave_clu = []
bet_glo_eff = []

print(f'average_clustering before removal: {nx.average_clustering(G5)} ')
print(f'global_efficiency before removal: {nx.global_efficiency(G5)} ')
print(nx.info(G5))
```

```
bet_ave_clu.append(nx.average_clustering(G5))
bet_glo_eff.append(nx.global_efficiency(G5))

for i in range(10):
    print('------------------------------------------------------')
    print(f'{i + 1} time node removal')

    bet_london_topo_2 = nx.betweenness_centrality(G5, normalized=False)

    nx.set_node_attributes(G5, bet_london_topo_2, 'betweenness_topo')

    betweenness_t_values = [(i[1]['betweenness_topo']) for i in G5.nodes(data=True)]

    maxbets = [n for n in G5.nodes() if G5.nodes[n]['betweenness_topo'] == max(betweenness_t_values)]

    print(f'remove node: {maxbets}')

    G5.remove_nodes_from(maxbets)

    print(nx.info(G5))

    bet_ave_clu.append(nx.average_clustering(G5))
    bet_glo_eff.append(nx.global_efficiency(G5))

    print(f'average_clustering: {nx.average_clustering(G5)} ')
    print(f'global_efficiency: {nx.global_efficiency(G5)} ')

plt.plot(bet_ave_clu)
```

```
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average clustering coefficient')
plt.title('Weighted betweenness centrality and average clustering')
plt.show()

plt.plot(bet_glo_eff)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average global efficiency')
plt.title('Weighted betweenness centrality and global_efficiency')
plt.show()

G6 = G.copy()

clo_ave_clu = []
clo_glo_eff = []

print(f'average_clustering before removal: {nx.average_clustering(G6)} ')
print(f'global_efficiency before removal: {nx.global_efficiency(G6)} ')
clo_ave_clu.append(nx.average_clustering(G6))
clo_glo_eff.append(nx.global_efficiency(G6))

for i in range(10):
    print('-----------------------------------------------------')
    print(f'{i + 1} time node removal')

    clo_london_topo_2 = nx.closeness_centrality(G6)
```

```
        nx.set_node_attributes(G6, clo_london_topo_2, 'closeness_topo')

        closeness_t_values = [(i[1]['closeness_topo']) for i in G6.nodes(data=True)]

        maxclos = [n for n in G6.nodes() if G6.nodes[n]['closeness_topo'] == max(closeness_t_values)]

        print(f'remove node: {maxclos}')

        G6.remove_nodes_from(maxclos)

        print(nx.info(G6))

        clo_ave_clu.append(nx.average_clustering(G6))
        clo_glo_eff.append(nx.global_efficiency(G6))

        print(f'average_clustering: {nx.average_clustering(G6)} ')
        print(f'global_efficiency: {nx.global_efficiency(G6)} ')

plt.plot(clo_ave_clu)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average clustering coefficient')
plt.title('Weighted closeness centrality average_clustering')
plt.show()

plt.plot(clo_glo_eff)
plt.xticks(np.arange(0,11))
plt.xlabel('Rmoval times')
plt.ylabel('average global efficiency')
```

```
plt.title('Weighted closeness centrality global_efficiency')
plt.show()

#2.1
bet_london_w=nx.betweenness_centrality(G,weight='flows',normalized=False)

nx.set_node_attributes(G,bet_london_w,'betweenness_w')
df = pd.DataFrame(index=G.nodes())
df['betweenness_w'] = pd.Series(nx.get_node_attributes(G, 'betweenness_w'))
df_sorted = df.sort_values(["betweenness_w"], ascending=False)
df_sorted[0:10]

# Lets set colour and size of nodes according to betweenness values
betweenness_w_values=[(i[1]['betweenness_w']) for i in G.nodes(data=True)]

bet_w_color=[(i[1]['betweenness_w']/max(betweenness_w_values)) for i in G.nodes(data=True)]
bet_w_size=[(i[1]['betweenness_w']/max(betweenness_w_values))*100 for i in G.nodes(data=True)]

pos=pos

fig, ax = plt.subplots(figsize=(12,12))


nx.draw_networkx_edges(G, pos,edge_color='black',
                       width=0.4)
nod=nx.draw_networkx_nodes(G,
                       pos = pos,
                       node_color= bet_w_color,
                       node_size= bet_w_size)
```

```
plt.colorbar(nod,label="Weighted Betweenness Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
plt.title("The London tube weighted betweenness centrality",fontsize=20)
plt.show()

clos_w=nx.closeness_centrality(G,distance='length')
nx.set_node_attributes(G,clos_w,'closeness_w')
df = pd.DataFrame(index=G.nodes())
df['closeness_w'] = pd.Series(nx.get_node_attributes(G, 'closeness_w'))
df_sorted = df.sort_values(["closeness_w"], ascending=False)
df_sorted[0:15]

# Lets set color and width of nodes according to the closeness values
clos_w_val=[(i[1]['closeness_w']) for i in G.nodes(data=True)]

closs_w_color=[(i[1]['closeness_w']-min(clos_w_val))/(max(clos_w_val)-min(clos_w_val)) for i in G.nodes(data=True)]
closs_w_size=[((i[1]['closeness_w']-min(clos_w_val))/(max(clos_w_val)-min(clos_w_val))*50) for i in G.nodes(data=True)]

fig, ax = plt.subplots(figsize=(12,12))
nx.draw_networkx_edges(G, pos,edge_color='black',
                       width=0.4)
nod=nx.draw_networkx_nodes(G,
                       pos = pos,
                       node_color= closs_w_color,
                       node_size= closs_w_size)
plt.colorbar(nod,label="Weighted Closeness Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
```

```
plt.title("The London tube weighted closeness centrality",fontsize=20)
plt.show()

eig_london_w=nx.eigenvector_centrality(G,weight='flows',max_iter=1000)
nx.set_node_attributes(G,eig_london_w,'eigenvector_w')
df = pd.DataFrame(index=G.nodes())
df['eigenvector_w'] = pd.Series(nx.get_node_attributes(G, 'eigenvector_w'))
df_sorted = df.sort_values(["eigenvector_w"], ascending=False)
df_sorted[0:10]

# Lets set colour and size of nodes according to eigenvector values
eigenvector_w_values=[(i[1]['eigenvector_w']) for i in G.nodes(data=True)]

eig_w_color=[(i[1]['eigenvector_w']/max(eigenvector_w_values)) for i in G.nodes(data=True)]
eig_w_size=[(i[1]['eigenvector_w']/max(eigenvector_w_values))*100 for i in G.nodes(data=True)]
fig, ax = plt.subplots(figsize=(12,12))

nx.draw_networkx_edges(G, pos,edge_color='black', width=0.4)

nod=nx.draw_networkx_nodes(G, pos = pos, node_color= eig_w_color, node_size= eig_w_size)

plt.colorbar(nod,label="Weighted Eigenvector Centrality",orientation="horizontal", shrink=0.5)
plt.axis("on")
plt.title("The London tube weighted eigenvector centrality",fontsize=20)
plt.show()

G9=G.copy()

bet_london_topo_1=nx.betweenness_centrality(G9,weight='flows',normalized=False)
```

```
nx.set_node_attributes(G9,bet_london_topo_1,'betweenness_topo')

df_bet_1 = pd.DataFrame(index=G9.nodes())

df_bet_1['betweenness_topo'] = pd.Series(nx.get_node_attributes(G9, 'betweenness_topo'))

df_sorted_bet_1 = df_bet_1.sort_values(["betweenness_topo"], ascending=False)
df_sorted_bet_1[0:10]

bet_ave_clu = []
bet_glo_eff = []

print(f'average_clustering before removal: {nx.average_clustering(G9)} ')
print(f'global_efficiency before removal: {nx.global_efficiency(G9)} ')

print(nx.info(G9))
bet_ave_clu.append(nx.average_clustering(G9))
bet_glo_eff.append(nx.global_efficiency(G9))

for i in range(10):
    print('-----------------------------------------------------')
    print(f'{i + 1} time node removal')

    betweenness_t_values = [(i[1]['betweenness_topo']) for i in G9.nodes(data=True)]

    maxbets = [n for n in G9.nodes() if G9.nodes[n]['betweenness_topo'] == max(betweenness_t_values)]

    print(f'remove node: {maxbets}')
```

```
        G9.remove_nodes_from(maxbets)

        print(nx.info(G9))

        bet_ave_clu.append(nx.average_clustering(G9))
        bet_glo_eff.append(nx.global_efficiency(G9))

        print(f'average_clustering: {nx.average_clustering(G9)} ')
        print(f'global_efficiency: {nx.global_efficiency(G9)} ')

plt.plot(bet_ave_clu)
plt.xticks(np.arange(0,11))
plt.title('betweenness centrality average_clustering')
plt.show()

plt.plot(bet_glo_eff)
plt.xticks(np.arange(0,11))
plt.title('betweenness centrality global_efficiency')
plt.show()



G2=G.copy()

clos__london_topo_1=nx.closeness_centrality(G2,distance='length')
nx.set_node_attributes(G2,clos__london_topo_1,'closeness_topo')
df_clo_1 = pd.DataFrame(index=G2.nodes())

df_clo_1 ['closeness_topo'] = pd.Series(nx.get_node_attributes(G2, 'closeness_topo'))
```

```
df_sorted_clo_1 = df_clo_1 .sort_values(["closeness_topo"], ascending=False)
df_sorted_clo_1[0:10]
clo_ave_clu = []
clo_glo_eff = []

print(f'average_clustering before removal: {nx.average_clustering(G2)} ')
print(f'global_efficiency before removal: {nx.global_efficiency(G2)} ')

print(nx.info(G2))
clo_ave_clu.append(nx.average_clustering(G2))
clo_glo_eff.append(nx.global_efficiency(G2))

for i in range(10):
    print('-----------------------------------------------------')
    print(f'{i + 1} time node removal')

    closeness_t_values = [(i[1]['closeness_topo']) for i in G2.nodes(data=True)]

    maxclos = [n for n in G2.nodes() if G2.nodes[n]['closeness_topo'] == max(closeness_t_values)]

    print(f'remove node: {maxclos}')

    G2.remove_nodes_from(maxclos)

    print(nx.info(G2))

    clo_ave_clu.append(nx.average_clustering(G2))
    clo_glo_eff.append(nx.global_efficiency(G2))
```

```
        print(f'average_clustering: {nx.average_clustering(G2)} ')
        print(f'global_efficiency: {nx.global_efficiency(G2)} ')

plt.plot(clo_ave_clu)
plt.xticks(np.arange(0,11))
plt.title('closeness centrality average_clustering')
plt.show()

plt.plot(clo_glo_eff)
plt.xticks(np.arange(0,11))
plt.title('closeness centrality global_efficiency')

plt.show()



#Part two:
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

cdata=pd.read_csv('data/london_flows.csv')
cdata.head(5)

cdata.sort_values(by=["station_origin", "station_destination"], inplace = True)
cdata.reset_index(inplace=True)
cdata.drop(columns = ["index"], inplace=True)
cdata["TotalNoIntra"] = cdata.apply(lambda x: 0 if x['station_origin'] == x['station_destination'] else x['flows'], axis=1)
```

```
cdata["offset"]= cdata.apply(lambda x: 0.0000000001 if x['station_origin'] == x['station_destination'] else 0, axis=1)

#we will just select the first 7 boroughs by code

to_match = ["Westminster", "Canary Wharf", "Oxford Circus", "Old Street", "King's Cross St. Pancras", "Bank and Monument", "Stratford"]


#subset the data by the 7 sample boroughs

#first the origins

cdatasub = cdata[cdata["station_origin"].isin(to_match)]

cdatasub = cdatasub[cdata["station_destination"].isin(to_match)]

cdatasub = cdatasub[cdata["station_origin"] != cdata["station_destination"]]

beg = ["station_origin", "station_destination", "TotalNoIntra"]

cols = beg + [col for col in cdatasub.columns.tolist() if col not in beg]

cdatasub = cdatasub.reindex(columns = cols)


cdatasubmat = pd.pivot_table(cdatasub, values ="TotalNoIntra", index="station_origin", columns = "station_destination",

                                                aggfunc=np.sum, margins=True)


# Estimating Model Parameters

fig, ax = plt.subplots(figsize=(10,10))

ax.scatter(x = cdata["distance"], y=cdata["flows"])

line = np.arange(0.0002, 50_000, 0.1)

ax.plot(line, line**-6, color ="r", label = "$\\beta=2$")

ax.legend(title = "$T \\sim d^{-\\beta}$", fontsize = 15, title_fontsize=15)

ax.set_xlim([-1000,70000])

ax.set_ylim([-1000, 70000])

ax.set_xlabel("Distance", fontsize = 20)

ax.set_ylabel("Total Flows", fontsize = 20)
```

```
#Let us now look at the behaviour of the flows with respect to the population at the origin denoted by O
#and then fit a model line T ~ O^alpha, for alpha=1
fig, ax = plt.subplots(figsize=(10,10))
ax.scatter(x = cdata["population"], y=cdata["flows"])
line = np.arange(0.0002, 80000, 0.1)
ax.plot(line, line**1, color ="r", label = "$\\alpha=1$")
ax.legend( fontsize = 15, title = "$T \sim O^{\\alpha}$", title_fontsize=15)
ax.set_xlabel("population", fontsize = 20)
ax.set_ylabel("Total Flows", fontsize = 20)

#Now let us look at the behaviour of the flows with respect to the salaries at destination denoted by D
# and then fit a model line    T ~ D^gamma. We will fit a line for gamma=1
fig, ax = plt.subplots(figsize=(10,10))
ax.scatter(x = cdata["jobs"], y=cdata["flows"])
line = np.arange(5_000, 80000, 0.1)
ax.plot(line, line**1, color ="r", label = "$\\gamma=2$")
ax.legend( fontsize = 15, title = "$T \sim D^{\\gamma}$", title_fontsize=15)
ax.set_xlabel("jobs", fontsize = 20)
ax.set_ylabel("Total flows", fontsize = 20)

#set up some variables to hold our parameter values in:
alpha = 1
gamma = 1
beta = 2
k = 1
T2 = sum(cdatasub["flows"])
Oi1_alpha = cdatasub["population"]**alpha
Dj2_gamma = cdatasub["jobs"]**gamma
dist_beta = cdatasub["distance"]**-beta
```

```
T1 = Oi1_alpha*Dj2_gamma*dist_beta
k = T2/sum(T1)

#R-squared
import scipy.stats
def CalcRSqaured(observed, estimated):

    r, p = scipy.stats.pearsonr(observed, estimated)
    R2 = r ** 2

    return R2
CalcRSqaured(cdatasub["flows"], cdatasub["unconstrainedEst1"])

#RSME
from math import sqrt


def CalcRMSE(observed, estimated):
    res = (observed - estimated) ** 2
    RMSE = round(sqrt(res.mean()), 3)
    return RMSE
CalcRMSE(cdatasub["flows"], cdatasub["unconstrainedEst1"])

plt.subplots(figsize=(10,10))

plt.hist(cdata["flows"], histtype="stepfilled" , bins = 100)
plt.xlabel("London travel to work flows histogram", fontsize = 15)
plt.ylabel("Count", fontsize= 15)
plt.title("London travel to work flows", fontsize = 20)
```

```
plt.grid(True)

#subset the dataframe to the flows we want
cdata_flows = cdata[["flows", "distance"]]
#remove all 0 values (logarithms can't deal with 0 values)
cdata_flows = cdata_flows[(cdata_flows!=0).all(1)]

x = np.log(cdata_flows["distance"])
y = np.log(cdata_flows["flows"])
fig, ax = plt.subplots(figsize = (10,10))
#plot the results along with the line of best fit
sns.regplot(x=x, y=y, marker="+", ax=ax)
ax.set_xlabel("log(distance)", fontsize = 20)
ax.set_ylabel("log(flows)", fontsize = 20)

import statsmodels.api as sm
import statsmodels.formula.api as smf

#take the variables and produce logarithms of them
x_variables = ["population", "jobs", "distance"]
log_x_vars = []
for x in x_variables:
    cdatasub[f"log_{x}"] = np.log(cdatasub[x])
    log_x_vars.append(f"log_{x}")

formula = 'flows ~ log_population + log_jobs +log_distance'
#run the regression
uncosim = smf.glm(formula = formula,
                  data=cdatasub,
```

```
                        family=sm.families.Poisson()).fit()


#4.1
def new_jobs(row):
    if row["station_destination"] == "Canary Wharf":
        val = 58772 / 2
    else:
        val = row["jobs"]
    return val
cdatasub["jobs_reduce"] = cdatasub.apply(new_jobs, axis=1)
cdatasub["jobs_reduce"] = cdatasub["jobs_reduce"].astype(int)
cdatasub.head(10)
import statsmodels.api as sm
import statsmodels.formula.api as smf

#take the variables and produce logarithms of them
x_variables = ["population", "jobs_reduce", "distance"]
log_x_vars = []
for x in x_variables:
    cdatasub[f"log_{x}"] = np.log(cdatasub[x])
    log_x_vars.append(f"log_{x}")

formula = 'flows ~ log_population + log_jobs_reduce +log_distance'

#run the regression
uncosim = smf.glm(formula = formula,
                        data=cdatasub,
                        family=sm.families.Poisson()).fit()
```

```
#first assign the parameter values from the model to the appropriate variables
K= -8.0291
alpha=0.9482
gamma=1.1525
beta=0.8529
cdatasub["unconstrained_job_reduce_Est"] = np.exp(K
                                                  + alpha*cdatasub["log_population"]
                                                  + gamma*cdatasub["log_jobs_reduce"]
                                                  - beta*cdatasub["log_distance"])
predictions = uncosim.get_prediction()
predictions_summary_frame = predictions.summary_frame()
cdatasub["fitted"] = predictions_summary_frame["mean"]

#beta=0.9
import statsmodels.api as sm
import statsmodels.formula.api as smf

#take the variables and produce logarithms of them
x_variables = ["population", "jobs", "distance"]
log_x_vars = []
for x in x_variables:
    cdatasub[f"log_{x}"] = np.log(cdatasub[x])
    log_x_vars.append(f"log_{x}")
formula = 'flows ~ log_population + log_jobs +log_distance'
#run the regression
uncosim = smf.glm(formula = formula,
                  data=cdatasub,
                  family=sm.families.Poisson()).fit()
```

```
K= -8.0291
alpha=0.9482
gamma=1.1525
beta=0.9

cdatasub["unconstrained_increase_cost"] = np.exp(K
                                                 + alpha*cdatasub["log_population"]
                                                 + gamma*cdatasub["log_jobs"]
                                                 - beta*cdatasub["log_distance"])
predictions = uncosim.get_prediction()
predictions_summary_frame = predictions.summary_frame()
cdatasub["fitted"] = predictions_summary_frame["mean"]

CalcRSqaured(cdatasub["flows"], cdatasub["unconstrained_increase_cost"])
CalcRMSE(cdatasub["flows"], cdatasub["unconstrained_increase_cost"])

#beta=1.1
import statsmodels.api as sm
import statsmodels.formula.api as smf

x_variables = ["population", "jobs", "distance"]
log_x_vars = []
for x in x_variables:
    cdatasub[f"log_{x}"] = np.log(cdatasub[x])
    log_x_vars.append(f"log_{x}")

formula = 'flows ~ log_population + log_jobs +log_distance'
uncosim = smf.glm(formula = formula,
                  data=cdatasub,
```

```
                        family=sm.families.Poisson()).fit()

K= -8.0291
alpha=0.9482
gamma=1.1525
beta=1.1

cdatasub["unconstrained_increase_cost"] = np.exp(K
                                                 + alpha*cdatasub["log_population"]
                                                 + gamma*cdatasub["log_jobs"]
                                                 - beta*cdatasub["log_distance"])
predictions = uncosim.get_prediction()
predictions_summary_frame = predictions.summary_frame()
cdatasub["fitted"] = predictions_summary_frame["mean"]

CalcRSqaured(cdatasub["flows"], cdatasub["unconstrained_increase_cost"])
CalcRMSE(cdatasub["flows"], cdatasub["unconstrained_increase_cost"])
```