Networks-complete

September 21, 2022

0.1 Notebook for ENCN375 Lecture A: Systems Analysis

In this notebook, we will look a little bit at networks, network analysis, Open Street Map, and plotting.

> To run notebook cells, click inside the block (add to the code or write your own) and press Ctrl+Enter If you've never used a Jupyter notebook before, you can practice with the first two blocks of code below

[2]:
```
# We'll start by defining a simple function
def hello_world(name):
    # name is a string input
    print('Hello, world this is '+name+'!')
```

[3]:
```
# Now we can practice calling our function - try inputting the code you need␣
 ↪below
hello_world("Bec")
```

```
Hello, world this is Bec!
```

0.1.1 Coding begins here

Now that you've practiced, you can use the notebook below for the lecture activities. Blocks will add themselves to the notebook automatically, or you can use the '+' button on the top ribbon to add more. You can save your notebook and outputs when you're finished.

[4]:
```
# Often we start by loading any packages we think we might need for our code
# There's a few key ones we require for today, but you can call more if you're␣
 ↪getting fancy:
import pandas as pd
import numpy as np
import networkx as nx
import osmnx as ox
import matplotlib.pyplot as plt
from matplotlib import cm
%matplotlib inline
```

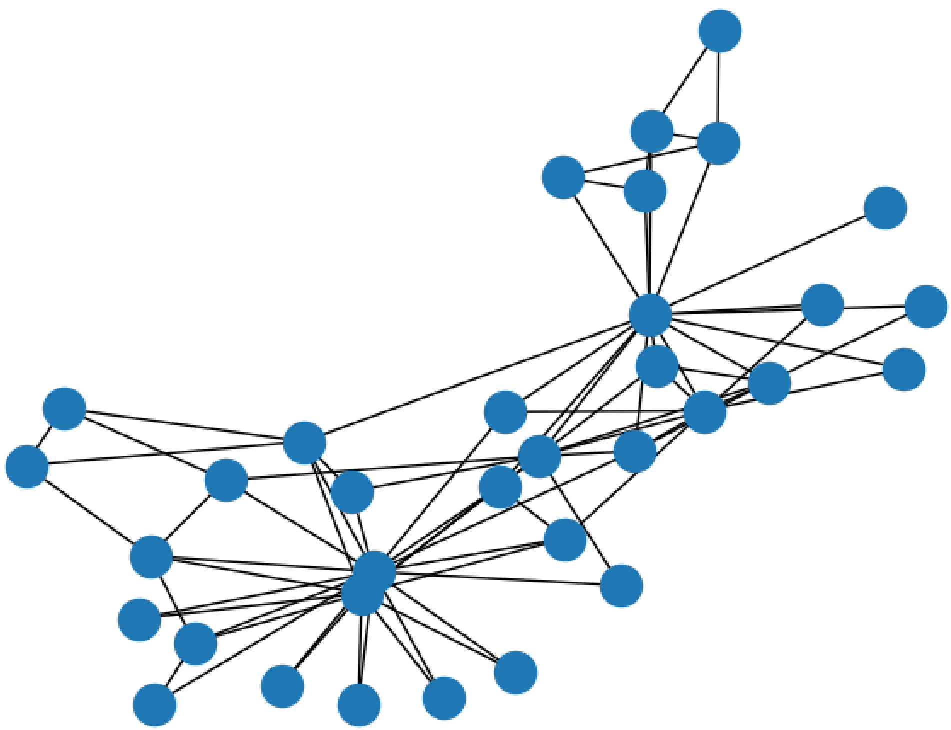
We'll start by looking at some network data and doing some simple network analysis. The networkx package has some empirical data that we will take advantage of for this lecture.

We'll be loading and investigating the dataset called Zachary's Karate Club, a dataset that describes social connections between members of a university Karate Club. The network has 34 members of a karate club, and shows links between members who interacted outside the club. This club was studied (for social cohesion and conflict) and during the study, a conflict between the administrator "John A" and instructor "Mr. Hi" (pseudonyms) led to the split of the club.

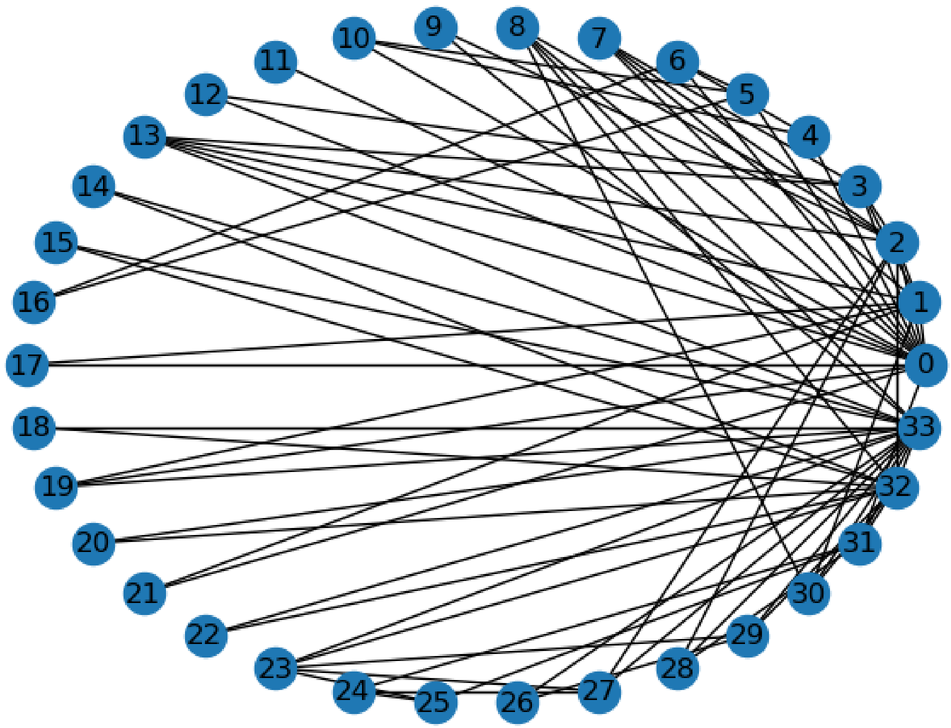
We'll refer to the package documents to help us find useful functions within networkx. They can be found here.

[5]:
```
# first step is to load in the data - I've done this one for you
g_karate = nx.karate_club_graph()
```

[6]:
```
# now, use the in-built functions to visualise the network
nx.draw(g_karate)
```



[7]:
```
# plot the network with a different layout using the built-in functions
nx.draw_circular(g_karate, with_labels=True)
```



[8]:
```
# Get a bit of information about the network. What sort of data are we looking␣
↪at?
# Each node contains information about the club that person belongs to. How␣
↪many belong to Mr. Hi vs John A?
# nx.info(g_karate)
nx.get_node_attributes(g_karate, "club")
```

[8]:
```
{0: 'Mr. Hi',
 1: 'Mr. Hi',
 2: 'Mr. Hi',
 3: 'Mr. Hi',
 4: 'Mr. Hi',
 5: 'Mr. Hi',
 6: 'Mr. Hi',
 7: 'Mr. Hi',
 8: 'Mr. Hi',
 9: 'Officer',
 10: 'Mr. Hi',
 11: 'Mr. Hi',
 12: 'Mr. Hi',
```

```
 13: 'Mr. Hi',
 14: 'Officer',
 15: 'Officer',
 16: 'Mr. Hi',
 17: 'Mr. Hi',
 18: 'Officer',
 19: 'Mr. Hi',
 20: 'Officer',
 21: 'Mr. Hi',
 22: 'Officer',
 23: 'Officer',
 24: 'Officer',
 25: 'Officer',
 26: 'Officer',
 27: 'Officer',
 28: 'Officer',
 29: 'Officer',
 30: 'Officer',
 31: 'Officer',
 32: 'Officer',
 33: 'Officer'}
```

[9]:
```
attrs = nx.get_node_attributes(g_karate, "club")
```

[10]:
```
for v in attrs.values():
    print(v)
```

```
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Officer
Mr. Hi
Mr. Hi
Mr. Hi
Mr. Hi
Officer
Officer
Mr. Hi
Mr. Hi
Officer
Mr. Hi
Officer
```

```
Mr. Hi
Officer
Officer
Officer
Officer
Officer
Officer
Officer
Officer
Officer
Officer
Officer
Officer
```

[11]:
```python
# if we wanted to sum the participants for each club and print out the answer
count_H = 0

for v in attrs.values():
    if v == "Mr. Hi":
        count_H += 1
    else:
        count_H = count_H

count_A = 34-count_H

print("There are {0} members in Mr. Hi's club, and {1} members in Officers's club".format(count_H,count_A))
```

```
There are 17 members in Mr. Hi's club, and 17 members in Officers's club
```

[12]:
```python
# now, use networkx to calculate some basic network statistics. Print out your results.
nodes =  g_karate.number_of_nodes()
links = g_karate.number_of_edges()
beta = links/nodes
density = nx.density(g_karate)

print("nodes: {0}, links: {1}, beta:{2:.2f}, density: {3:.2f}".format(nodes, links, beta, density))
```

```
nodes: 34, links: 78, beta:2.29, density: 0.14
```

[13]:
```python
# next we'll look at the nodes themselves. Calculate the degree, closeness, and betweenness of the nodes
# store your answers in a dataframe and print them out
node = list(g_karate.nodes())
degree = [v for (d,v) in g_karate.degree()]
cc = list(nx.closeness_centrality(g_karate).values())
```

```python
bc = list(nx.betweenness_centrality(g_karate).values())

node_sts = pd.DataFrame(zip(*[node, degree, cc, bc]),␣
 ↪columns=['nodes','degree','closeness','betweenness'])
```

[14]:
```python
node_sts
```

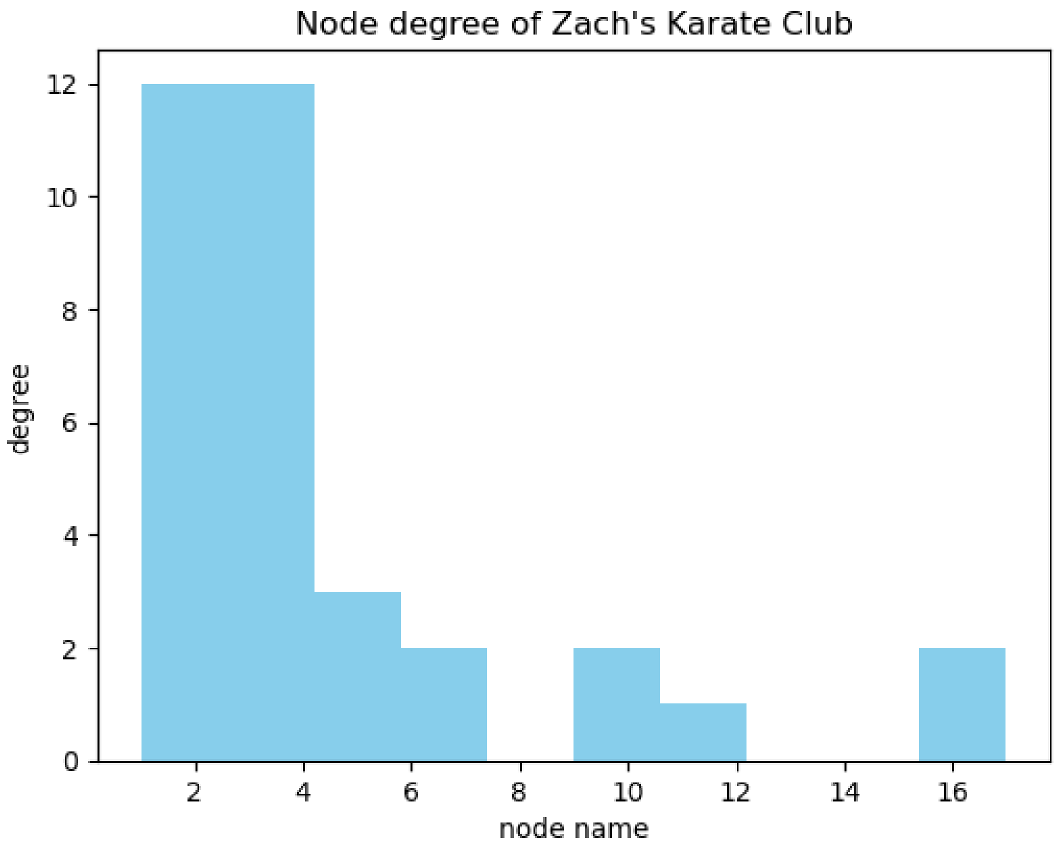
[14]:

	nodes	degree	closeness	betweenness
0	0	16	0.568966	0.437635
1	1	9	0.485294	0.053937
2	2	10	0.559322	0.143657
3	3	6	0.464789	0.011909
4	4	3	0.379310	0.000631
5	5	4	0.383721	0.029987
6	6	4	0.383721	0.029987
7	7	4	0.440000	0.000000
8	8	5	0.515625	0.055927
9	9	2	0.434211	0.000848
10	10	3	0.379310	0.000631
11	11	1	0.366667	0.000000
12	12	2	0.370787	0.000000
13	13	5	0.515625	0.045863
14	14	2	0.370787	0.000000
15	15	2	0.370787	0.000000
16	16	2	0.284483	0.000000
17	17	2	0.375000	0.000000
18	18	2	0.370787	0.000000
19	19	3	0.500000	0.032475
20	20	2	0.370787	0.000000
21	21	2	0.375000	0.000000
22	22	2	0.370787	0.000000
23	23	5	0.392857	0.017614
24	24	3	0.375000	0.002210
25	25	3	0.375000	0.003840
26	26	2	0.362637	0.000000
27	27	4	0.458333	0.022333
28	28	3	0.452055	0.001795
29	29	4	0.383721	0.002922
30	30	4	0.458333	0.014412
31	31	6	0.540984	0.138276
32	32	12	0.515625	0.145247
33	33	17	0.550000	0.304075

[15]:
```python
# lets make a quick histogram of node degrees. Change the default colour to␣
 ↪something nicer and include axis titles
plt.hist(degree, color='skyblue')
plt.ylabel('degree')
```

```
plt.xlabel('node name')
plt.title("Node degree of Zach's Karate Club")
```

[15]:
```
Text(0.5, 1.0, "Node degree of Zach's Karate Club")
```

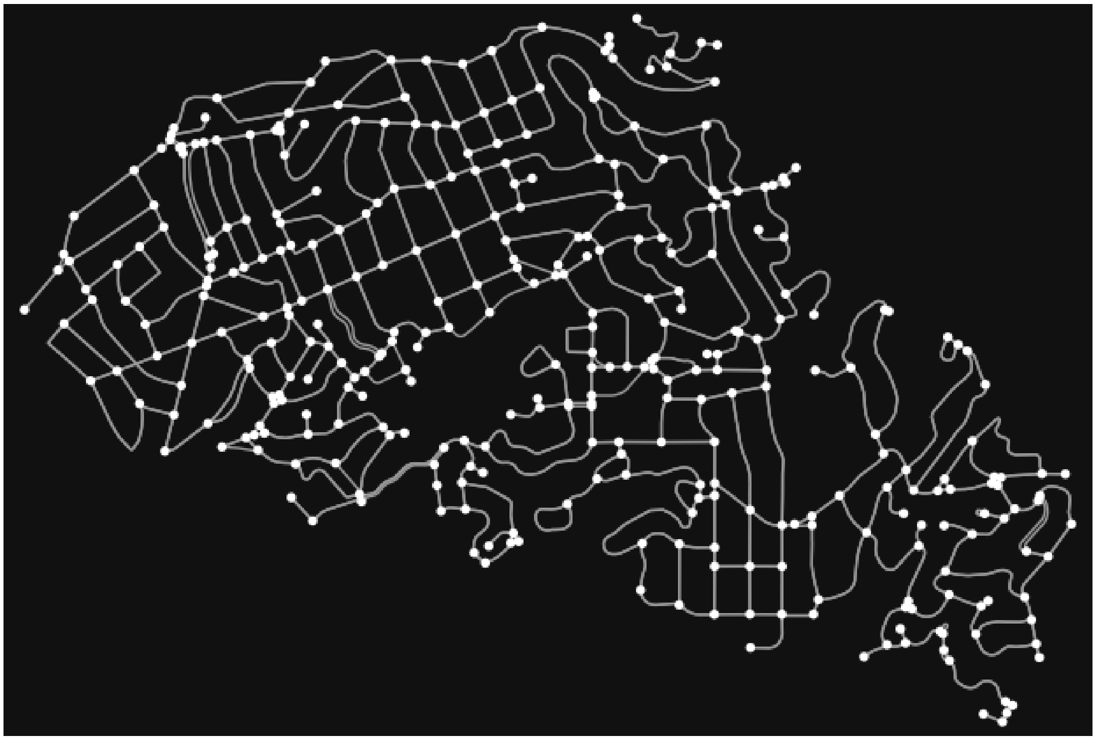


Now that we've done a bit of work with networks in Python, let's familiarise ourselves with Open Street Map (OSM). OSM is a publicly accessible database of street and building footprints across the world. The osmnx package lets us easily access this database and convert information from OSM to networks. We'll take a simple example for this tutorial of Piedmont, California, USA.

Similar to our network calculations above, we'll be referring to the docs to find the functions we need in the osmnx package - you'll find those here.

[16]:
```
# download the street network for Piedmont, Califonia
G = ox.graph_from_place("Piedmont, California, USA", network_type="drive")
```

[17]:
```
# visulise the street network
fig, ax = ox.plot_graph(G)
```



[18]:
```
# take a look at the data in the network. What are the edge attributes?
gdf_nodes, gdf_edges = ox.graph_to_gdfs(G)
gdf_edges.head()
```

[18]:
```
                                  osmid          name      highway  oneway   length  \
u        v        key
53017091 53064327 0       6345781   Rose Avenue  residential   False  231.335
         53075599 0       6345781   Rose Avenue  residential   False  121.114
53018397 53097980 0     196739937  Linda Avenue     tertiary   False  100.767
         53018399 0       6327298   Lake Avenue  residential   False  124.622
         53018411 0     196739937  Linda Avenue     tertiary   False   37.803

                                                                      geometry  \
u        v        key
53017091 53064327 0     LINESTRING (-122.24760 37.82625, -122.24551 37…
         53075599 0     LINESTRING (-122.24760 37.82625, -122.24770 37…
53018397 53097980 0     LINESTRING (-122.24719 37.82422, -122.24777 37…
         53018399 0     LINESTRING (-122.24719 37.82422, -122.24712 37…
         53018411 0     LINESTRING (-122.24719 37.82422, -122.24713 37…

                        lanes maxspeed bridge junction
u        v        key
53017091 53064327 0       NaN      NaN    NaN      NaN
```

```
          53075599 0     NaN      NaN     NaN      NaN
53018397  53097980 0     NaN      NaN     NaN      NaN
          53018399 0     NaN      NaN     NaN      NaN
          53018411 0     NaN      NaN     NaN      NaN
```

[19]:
```python
# calculate some basic network statistics using the in-built functions.
ox.basic_stats(G)
```

[19]:
```
{'n': 348,
 'm': 940,
 'k_avg': 5.402298850574713,
 'intersection_count': 314,
 'streets_per_node_avg': 2.9597701149425286,
 'streets_per_node_counts': {0: 0, 1: 34, 2: 0, 3: 264, 4: 47, 5: 2, 6: 1},
 'streets_per_node_proportion': {0: 0.0,
  1: 0.09770114942528736,
  2: 0.0,
  3: 0.7586206896551724,
  4: 0.13505747126436782,
  5: 0.005747126436781609,
  6: 0.0028735632183908046},
 'edge_length_total': 113088.96999999996,
 'edge_length_avg': 120.30741489361698,
 'street_length_total': 58500.43000000003,
 'street_length_avg': 118.90331300813014,
 'street_segments_count': 492,
 'node_density_km': None,
 'intersection_density_km': None,
 'edge_density_km': None,
 'street_density_km': None,
 'circuity_avg': 1.1132609164541127,
 'self_loop_proportion': 0.006382978723404255,
 'clean_intersection_count': None,
 'clean_intersection_density_km': None}
```

[20]:
```python
# what do the values above mean? Make some nice print statements to give some␣
↪of the stats highlights.
sts = ox.basic_stats(G)
print("There are {0} nodes and {1} edges in the Network.".
↪format(sts['n'],sts['m']))
print("The average node degree is {0:.2f} and there are {1:.0f} total km of␣
↪roads in the network".format(sts['k_avg'],sts['edge_length_total']/1000))
```

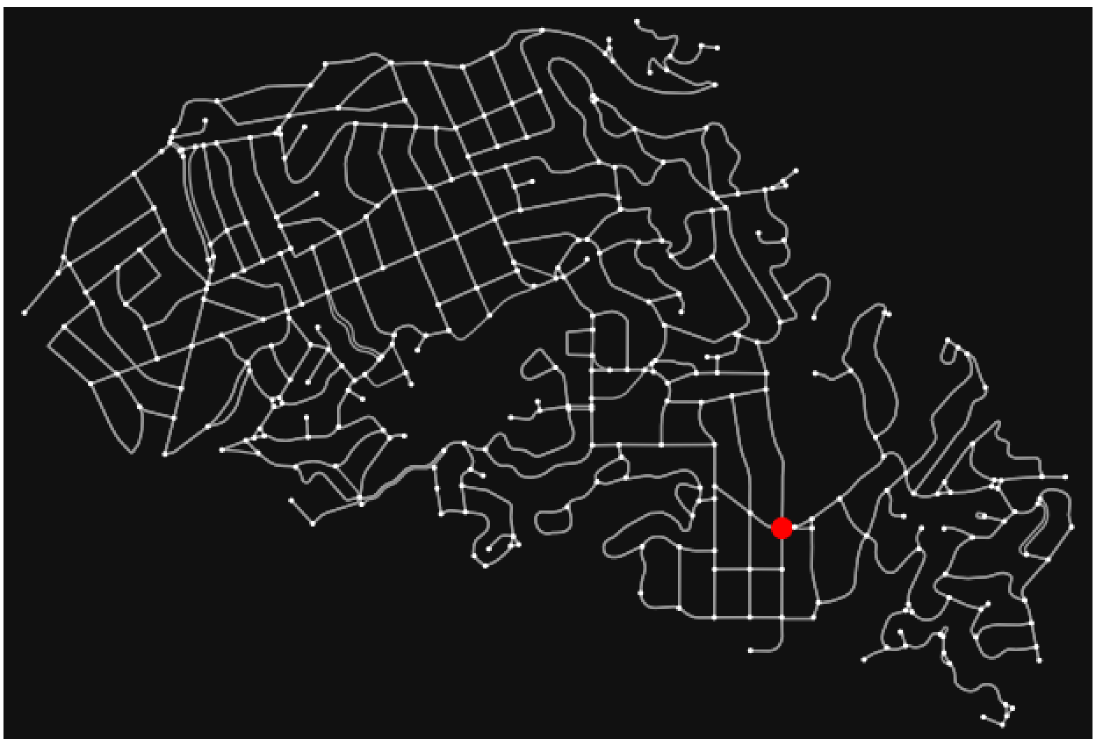
```
There are 348 nodes and 940 edges in the Network.
The average node degree is 5.40 and there are 113 total km of roads in the
network
```

[21]:
```
# find the most central node in terms of betweenness centrality. What␣
↪percentage of roads pass through this node?
# for this, we'll convert our graph to a digraph (no parallel links) and␣
↪calculate betweenness based on road length
bc = nx.betweenness_centrality(ox.get_digraph(G), weight="length")
max_node, max_bc = max(bc.items(), key=lambda x: x[1])
```

[22]:
```
print("The most central node in the network is {0}, and approximately {1:.0%}␣
↪of roads pass through it.".format(max_node, max_bc))
```

```
The most central node in the network is 53124805, and approximately 32% of roads
pass through it.
```

[23]:
```
# plot the street network with the most central node highlighted
nc = ["r" if node == max_node else "w" for node in G.nodes]
ns = [80 if node == max_node else 5 for node in G.nodes]
fig, ax = ox.plot_graph(G, node_size=ns, node_color=nc)
```



Now we'll do some fancy visualisation for closeness centrality by inverting our network (so that nodes become links and vice versa). We'll make use of a colourmap from matplotlib - more details here.

[24]:
```
# first, we convert our graph into a line graph, which accomplishes this "flip"␣
↪for us
G_line = nx.line_graph(G)

# then we can calculate the closeness centrality of our new "edges"
edge_cc = nx.closeness_centrality(G_line)

# and set this value as an edge attribute in our orignal (unflipped) network␣
↪with the name "edge closeness"
nx.set_edge_attributes(G, edge_cc, "edge_centrality")
```

[25]:
```
# to plot, we need to map our edge closeness attribute onto a colourmap
e_clrs = ox.plot.get_edge_colors_by_attr(G, "edge_centrality", cmap="inferno")
```

[26]:
```
# then we can plot, setting our edge colours to our mapped ones (above) and the␣
↪node size to zero (for a clean plot)
fig, ax = ox.plot_graph(G, edge_color=e_clrs, node_size=0)
```