NB04-Cholera-Case-Study-OpenStreetMaps-Networkx-Part-2

October 4, 2018

```
&lt;IPython.core.display.HTML object&gt;
```

1 Notebook 3: Analyzing the John Snow Cholera Outbreak Using OpenStreetMaps and Networkx - Part 2

1.0.1 Summary of steps

We will carry out the following steps:

Step 1 - Read the street network graph, `G`, of Soho district using OSMnx using a set of coordinates in the middle of Soho district. **(We saved this graph in Notebook 2 and we will read it from the graphml file `soho.graphml`.)**

Step 2 - Load the original data sets from Notebook 1 (pumps and deaths) into `pumps_df` and `deaths_df`.

Step 3 - To represent coordnates from the pumps and deaths from the Notebook 1 in OSMnx graph format, we have to find the nearest OSMnx nodes to those points. We will add new columns to the pumps and deaths dataframes to accomodate new information coming from OSMnx. We will also store the short distances between original points to the nearest OSMnx points and store it in the respective dataframes.

Step 4 - To calculate mean distances from death coordinates to pump coordinates we will create a nested loop through records of both dataframes, `pumps_df` and `deaths_df`, for pairwise distance calculations between each pump and death coordinates. We will add the short distances from #1 to the pump point to death point distance and store this in a new dataframe called `routes_df`.

Step 5 - We will then create the map representation pump-to-death-points mean distances using `folium` and superimpose this on the markers generated in Notebook 1.

```
'0.7.1'
```

Let's read our street network graph from file using the dot function, `load_graphml()`.

1.1 Step 2. Load pumps and deaths data sets

Let's read the data set from a CSV file using the dot function `read_csv()`.

```
FID  DEATHS       LON        LAT
     0    0       3 -0.137930  51.513418
     1    1       2 -0.137883  51.513361
```

```
    2    2        1 -0.137853  51.513317
    3    3        1 -0.137812  51.513262
    4    4        4 -0.137767  51.513204
```

Since we "pickled" this dataframe, we can also **read** from the **pickled** file with the dot function called `read_pickle()`.

```
FID  DEATHS       LON        LAT
    0    0        3 -0.137930  51.513418
    1    1        2 -0.137883  51.513361
    2    2        1 -0.137853  51.513317
    3    3        1 -0.137812  51.513262
    4    4        4 -0.137767  51.513204
```

```
FID       LON        LAT
    0  250 -0.136668  51.513341
    1  251 -0.139586  51.513876
    2  252 -0.139671  51.514906
    3  253 -0.131630  51.512354
    4  254 -0.133594  51.512139
    5  255 -0.135919  51.511542
    6  256 -0.133962  51.510019
    7  257 -0.138199  51.511295
```

1.2 Step 3: Set up `pumps_df` and `deaths_df` dataframes to store additional `osmnx` information

1.2.1 Set up `pumps_df` dataframe for analysis

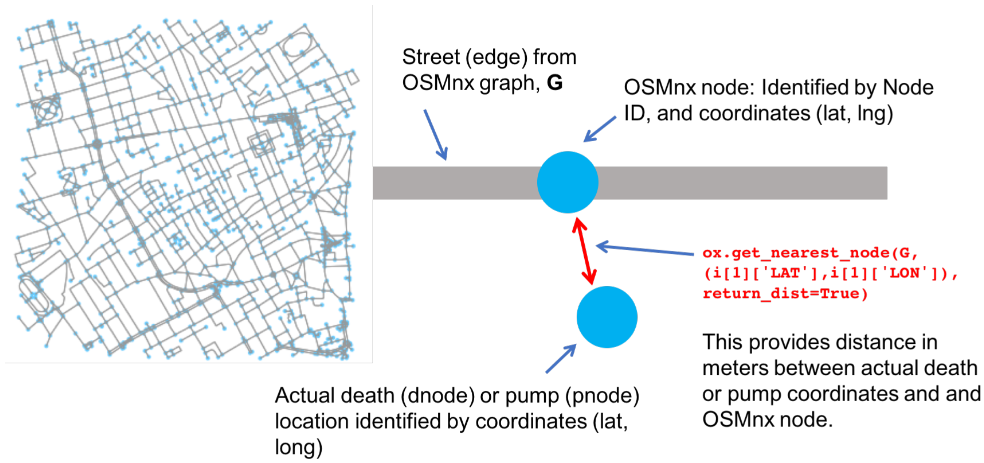
We retain the `LON` and `LAT` columns.

```
pumps_df = pumps_df[['LON', 'LAT']]
```

We create five new columns: 1. Two (2),`pumps_df['GLON']` and `pumps_df['GLAT']`, to store coordinates from the OSMnx graph nodes 2. `pumps_df['DISTANCE']` to store distance between original coordinates and OSMnx graph coordinates 3. `pumps_df['NODE']` to store node ID of a coordinate in the OSMnx graph 4. `pumps_df['MEAN_DISTANCE']` to store the mean distance values between a pump and death coordinates.

We also store default values for these columns as below.

```
    LON        LAT  GLON  GLAT  DISTANCE  NODE  MEAN_DISTANCE
    0 -0.136668  51.513341   0.0   0.0       0.0     0             0.0
    1 -0.139586  51.513876   0.0   0.0       0.0     0             0.0
    2 -0.139671  51.514906   0.0   0.0       0.0     0             0.0
    3 -0.131630  51.512354   0.0   0.0       0.0     0             0.0
    4 -0.133594  51.512139   0.0   0.0       0.0     0             0.0
    5 -0.135919  51.511542   0.0   0.0       0.0     0             0.0
    6 -0.133962  51.510019   0.0   0.0       0.0     0             0.0
    7 -0.138199  51.511295   0.0   0.0       0.0     0             0.0
```



You can verify the data types for `pumps_df` columns with `dtypes` dataframe attribute like so.

```
LON                 float64
        LAT                 float64
        GLON                float64
        GLAT                float64
        DISTANCE            float64
        NODE                  int64
        MEAN_DISTANCE       float64
        dtype: object
```

We use a cell magic `%%time` to time the execution of each loop.

```
CPU times: user 60 ms, sys: 10 ms, total: 70 ms
Wall time: 74.9 ms
```

The code above obtains the distance between pump coordinates and OSMnx node coordinates using the `ox.get_nearest_node()` as shown in the diagram below.

To quickly obtain the mean of all values from the `DISTANCE` column, we use the `pandas` dot function, `mean()`.

```
12.01082334190247
```

You can make the print out more human-friendly by adding "meters".

```
12.01082334190247 meters
```

What does this value mean?

```
        LON          LAT        GLON         GLAT    DISTANCE         NODE   \
         0 -0.136668   51.513341 -0.136533   51.513391   10.882847     25473293
```

```
1 -0.139586  51.513876 -0.139462  51.513861   8.746434    21665926
2 -0.139671  51.514906 -0.139904  51.514855  17.076771  4684520654
3 -0.131630  51.512354 -0.131466  51.512196  20.870686      107807
4 -0.133594  51.512139 -0.133606  51.512189   5.593945   348875443
5 -0.135919  51.511542 -0.135762  51.511404  18.794818    25473300
6 -0.133962  51.510019 -0.133994  51.510125  11.952066  1663004187
7 -0.138199  51.511295 -0.138178  51.511281   2.169020    25257692

   MEAN_DISTANCE
0            0.0
1            0.0
2            0.0
3            0.0
4            0.0
5            0.0
6            0.0
7            0.0
```

1.2.2 Set up `deaths_df` dataframe for analysis

```
DEATHS        int64
         LON         float64
         LAT         float64
         GLON        float64
         GLAT        float64
         DISTANCE    float64
         NODE          int64
         dtype: object

     DEATHS       LON        LAT  GLON  GLAT  DISTANCE  NODE
         245       3 -0.137108  51.514526   0.0   0.0       0.0     0
         246       2 -0.137065  51.514706   0.0   0.0       0.0     0
         247       1 -0.138474  51.512311   0.0   0.0       0.0     0
         248       1 -0.138123  51.511998   0.0   0.0       0.0     0
         249       1 -0.137762  51.511856   0.0   0.0       0.0     0

CPU times: user 1.45 s, sys: 10 ms, total: 1.46 s
Wall time: 1.48 s
```

```
15.540870559932959
```

```
     DEATHS       LON        LAT      GLON       GLAT   DISTANCE        NODE
         0         3 -0.137930  51.513418 -0.137948  51.513408   1.692941    25501340
         1         2 -0.137883  51.513361 -0.137948  51.513408   6.886582    25501340
         2         1 -0.137853  51.513317 -0.137835  51.513236   9.138491   701600719
         3         1 -0.137812  51.513262 -0.137835  51.513236   3.332220   701600719
         4         4 -0.137767  51.513204 -0.137835  51.513236   5.861446   701600719
         5         2 -0.137537  51.513184 -0.137541  51.513317  14.780078   701600731
```

6	2	-0.138200	51.513359	-0.138377	51.513267	15.979920	25501330
7	2	-0.138045	51.513328	-0.137948	51.513408	11.117584	25501340
8	3	-0.138276	51.513323	-0.138377	51.513267	9.383252	25501330
9	2	-0.138223	51.513427	-0.137948	51.513408	19.135555	25501340
10	2	-0.138337	51.513381	-0.138377	51.513267	13.007540	25501330
11	1	-0.138563	51.513462	-0.138596	51.513496	4.461965	25501328
12	3	-0.138426	51.513216	-0.138377	51.513267	6.580060	25501330
13	1	-0.138378	51.513169	-0.138377	51.513267	10.864198	25501330
14	4	-0.138337	51.513116	-0.138204	51.513038	12.686940	2784682639
15	1	-0.138645	51.513240	-0.138775	51.513109	17.109988	21665930
16	1	-0.138698	51.513164	-0.138775	51.513109	8.097942	21665930
17	1	-0.137924	51.513178	-0.137835	51.513236	8.895100	701600719
18	4	-0.137865	51.513111	-0.137835	51.513236	14.011261	701600719
19	3	-0.137811	51.513055	-0.137662	51.513019	11.068022	108072
20	2	-0.138762	51.513441	-0.138596	51.513496	13.022775	25501328
21	1	-0.138799	51.513592	-0.138742	51.513646	7.213844	25501325
22	2	-0.139045	51.513402	-0.139007	51.513333	8.119061	21665931
23	2	-0.138970	51.513380	-0.139007	51.513333	5.832931	21665931
24	2	-0.138863	51.513411	-0.139007	51.513333	13.223316	21665931
25	1	-0.138752	51.513641	-0.138742	51.513646	0.900637	25501325
26	1	-0.138808	51.513693	-0.138742	51.513646	6.897698	25501325
27	3	-0.138856	51.513745	-0.138934	51.513841	12.000980	25501320
28	1	-0.138887	51.513676	-0.138742	51.513646	10.546810	25501325
29	1	-0.139239	51.513590	-0.139363	51.513684	13.549185	4233926316
..	...	...	...	...	...	...	...
220	3	-0.135679	51.513766	-0.135533	51.513668	14.911211	21666011
221	1	-0.135814	51.513726	-0.135533	51.513668	20.527139	21666011
222	5	-0.135905	51.513692	-0.135918	51.513542	16.702162	21666010
223	4	-0.135992	51.513672	-0.135918	51.513542	15.342576	21666010
224	4	-0.136217	51.513603	-0.136261	51.513469	15.262476	108073
225	1	-0.136579	51.513482	-0.136533	51.513391	10.599659	25473293
226	4	-0.136675	51.513458	-0.136533	51.513391	12.329916	25473293
227	1	-0.136764	51.513429	-0.136533	51.513391	16.537850	25473293
228	3	-0.136877	51.513404	-0.136987	51.513504	13.465111	701608618
229	2	-0.136953	51.513359	-0.137054	51.513242	14.774894	25501289
230	1	-0.137230	51.513378	-0.137193	51.513441	7.431692	701608613
231	2	-0.136651	51.513855	-0.136092	51.513963	40.480584	25501182
232	1	-0.136503	51.513875	-0.136092	51.513963	30.054525	25501182
233	1	-0.137367	51.513565	-0.137308	51.513723	18.078658	1324710127
234	2	-0.137422	51.513616	-0.137308	51.513723	14.308505	1324710127
235	3	-0.137472	51.513742	-0.137308	51.513723	11.529086	1324710127
236	1	-0.138300	51.513918	-0.138185	51.513714	23.975473	499350858
237	1	-0.137363	51.513772	-0.137308	51.513723	6.606033	1324710127
238	4	-0.137995	51.513502	-0.137948	51.513408	10.964118	25501340
239	2	-0.138139	51.513712	-0.138185	51.513714	3.230596	499350858
240	2	-0.138239	51.513644	-0.138185	51.513714	8.669845	499350858
241	1	-0.138272	51.513711	-0.138185	51.513714	5.998988	499350858
242	5	-0.138083	51.514061	-0.137992	51.513768	33.174874	2770153342

```
243       3 -0.137912  51.514748 -0.137840  51.514732   5.263568  4702603653
244       2 -0.137707  51.514794 -0.137741  51.514761   4.395828  4702603655
245       3 -0.137108  51.514526 -0.136870  51.514363  24.490766    21666019
246       2 -0.137065  51.514706 -0.137192  51.514831  16.487636     9521025
247       1 -0.138474  51.512311 -0.138203  51.512108  29.345953      108070
248       1 -0.138123  51.511998 -0.138203  51.512108  13.416832      108070
249       1 -0.137762  51.511856 -0.137775  51.511712  16.037627    25473286

[250 rows x 7 columns]
```

```
DEATHS        int64
        LON         float64
        LAT         float64
        GLON        float64
        GLAT        float64
        DISTANCE    float64
        NODE          int64
        dtype: object
```

1.3 Step 4: Pairwise calculation of distance between `deaths_df` and `pumps_df` coordinates

Let's create the new dataframe, `routes_df` from the list.

```
  DNODE     PNODE    DISTANCE
0       25501340  25473293  149.065973
1       25501340  25473293  154.259614
2      701600719  25473293  135.818211
3      701600719  25473293  130.011941
4      701600719  25473293  132.541167
5      701600731  25473293  163.738076
6       25501330  25473293  197.222923
7       25501340  25473293  158.490615
8       25501330  25473293  190.626255
9       25501340  25473293  166.508586
10      25501330  25473293  194.250543
11      25501328  25473293  215.413276
12      25501330  25473293  187.823062
13      25501330  25473293  192.107201
14    2784682639  25473293  166.681201
15      21665930  25473293  228.755097
16      21665930  25473293  219.743051
17     701600719  25473293  135.574821
18     701600719  25473293  140.690981
19        108072  25473293  110.814174
20      25501328  25473293  223.974086
21      25501325  25473293  237.665549
22      21665931  25473293  249.403188
```

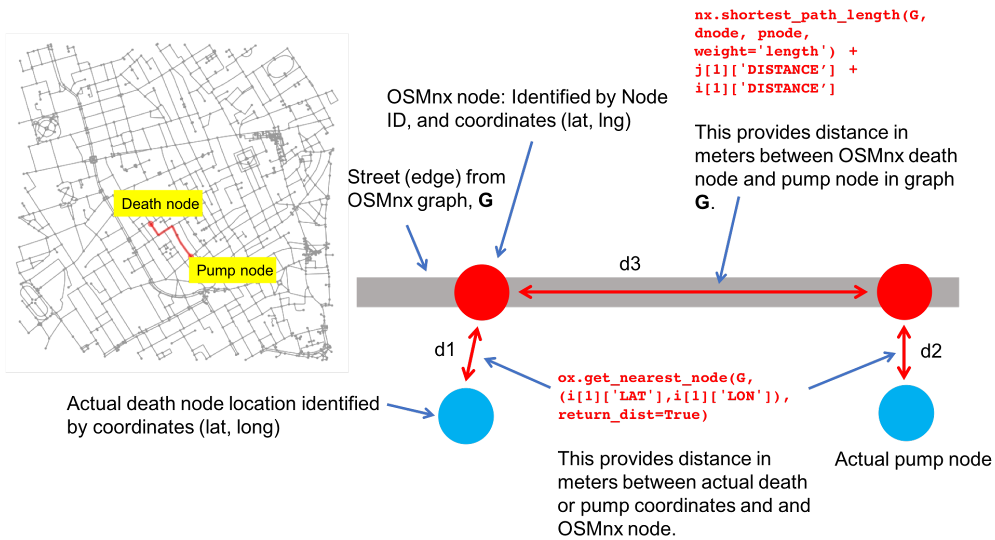
```
23        21665931  25473293  247.117058
24        21665931  25473293  254.507444
25        25501325  25473293  231.352342
26        25501325  25473293  237.349403
27        25501320  25473293  264.590145
28        25501325  25473293  240.998515
29      4233926316  25473293  301.042286
...            ...       ...         ...
1970      21666011  25257692  433.846088
1971      21666011  25257692  439.462016
1972      21666010  25257692  405.550145
1973      21666010  25257692  404.190559
1974        108073  25257692  385.700225
1975      25473293  25257692  360.346928
1976      25473293  25257692  362.077185
1977      25473293  25257692  366.285119
1978     701608618  25257692  372.897718
1979      25501289  25257692  334.217538
1980     701608613  25257692  350.942294
1981      25501182  25257692  590.058748
1982      25501182  25257692  579.632689
1983    1324710127  25257692  395.348883
1984    1324710127  25257692  391.578730
1985    1324710127  25257692  388.799311
1986     499350858  25257692  380.043690
1987    1324710127  25257692  383.876258
1988      25501340  25257692  329.180344
1989     499350858  25257692  359.298812
1990     499350858  25257692  364.738062
1991     499350858  25257692  362.067205
1992    2770153342  25257692  403.913917
1993    4702603653  25257692  538.819312
1994    4702603655  25257692  545.854271
1995      21666019  25257692  503.025579
1996       9521025  25257692  551.675164
1997        108070  25257692  152.336645
1998        108070  25257692  136.407524
1999      25473286  25257692  192.093064

[2000 rows x 3 columns]
```

This code snippet below obtains the total distance, `d1 + d2 + d3`.

`distance = \ nx.shortest_path_length(G, dnode, pnode, weight='length') + \ j[1]['DISTANCE'] + i[1]['DISTANCE']`

By this time we have obtained three sets of distances (see following figure): 1. `d1` = distance in meters between coordinates of an actual pump and an OSMnx node from graph, `G` from `j[1]['DISTANCE']` 2. `d2` = distance in meters between coordinates of a death location and an OSMnx node from graph, `G` from `i[1]['DISTANCE']` 3. `d3` = distance in meters between co-



ordinates of two OSMnx nodes (pump and death location) from `nx.shortest_path_length(G, dnode, pnode, weight='length')`

Let's assume that people in Soho district would prefer to walk 400 meters to fetch water from each pump by setting a filter of 400 meters or less as "walkable".

Let's inspect that filtered dataframe, `routes2_df`. How many rows did we end up with?

	DNODE	PNODE	DISTANCE
0	25501340	25473293	149.065973
1	25501340	25473293	154.259614
2	701600719	25473293	135.818211
3	701600719	25473293	130.011941
4	701600719	25473293	132.541167
5	701600731	25473293	163.738076
6	25501330	25473293	197.222923
7	25501340	25473293	158.490615
8	25501330	25473293	190.626255
9	25501340	25473293	166.508586
10	25501330	25473293	194.250543
11	25501328	25473293	215.413276
12	25501330	25473293	187.823062
13	25501330	25473293	192.107201
14	2784682639	25473293	166.681201
15	21665930	25473293	228.755097
16	21665930	25473293	219.743051
17	701600719	25473293	135.574821
18	701600719	25473293	140.690981
19	108072	25473293	110.814174
20	25501328	25473293	223.974086
21	25501325	25473293	237.665549
22	21665931	25473293	249.403188

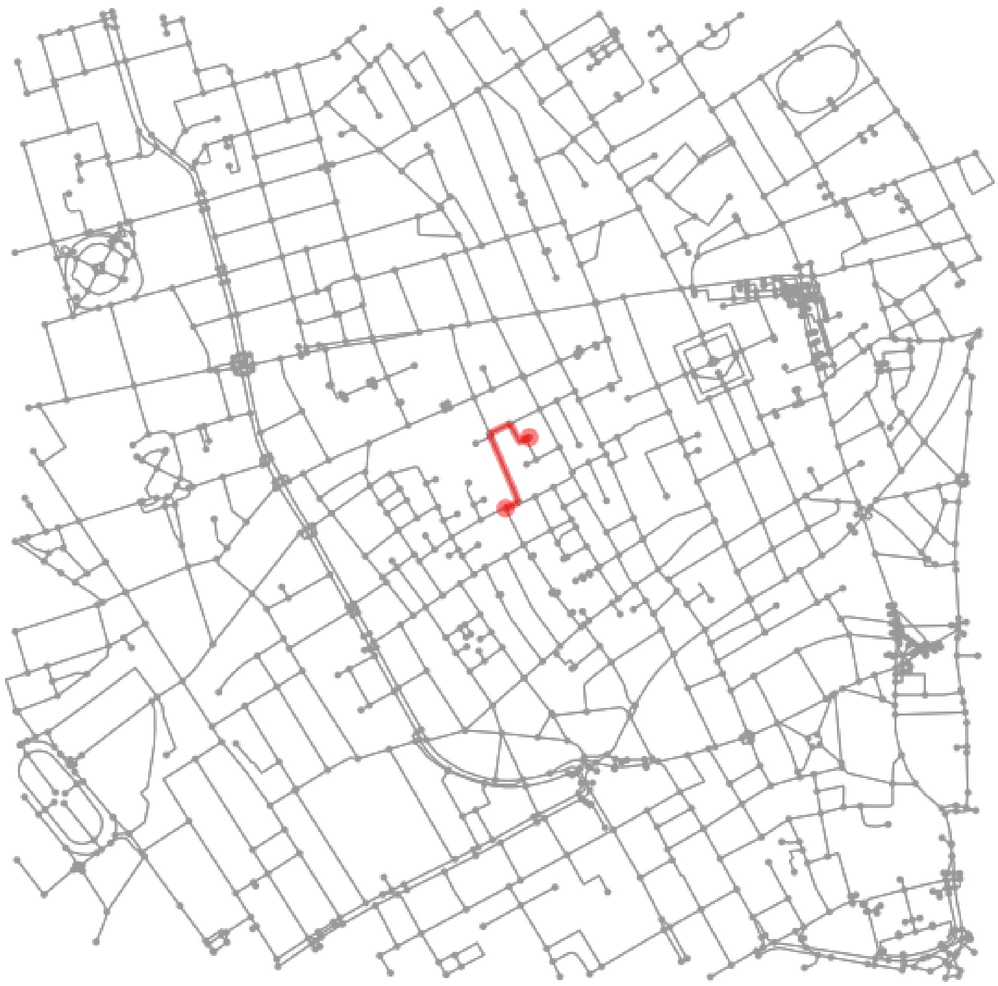
```
23      21665931  25473293  247.117058
24      21665931  25473293  254.507444
25      25501325  25473293  231.352342
26      25501325  25473293  237.349403
27      25501320  25473293  264.590145
28      25501325  25473293  240.998515
29    4233926316  25473293  301.042286
...          ...       ...         ...
1874    26845546  25257692  399.222870
1877    26845546  25257692  398.639337
1879    25473409  25257692  355.329287
1880    25473409  25257692  350.010342
1881    25473409  25257692  346.487569
1882    25473409  25257692  343.726625
1883    25473409  25257692  347.417581
1884  1330788638  25257692  364.496006
1885    25473409  25257692  360.196354
1887      108073  25257692  379.791659
1888      108073  25257692  378.176006
1974      108073  25257692  385.700225
1975    25473293  25257692  360.346928
1976    25473293  25257692  362.077185
1977    25473293  25257692  366.285119
1978   701608618  25257692  372.897718
1979    25501289  25257692  334.217538
1980   701608613  25257692  350.942294
1983  1324710127  25257692  395.348883
1984  1324710127  25257692  391.578730
1985  1324710127  25257692  388.799311
1986   499350858  25257692  380.043690
1987  1324710127  25257692  383.876258
1988    25501340  25257692  329.180344
1989   499350858  25257692  359.298812
1990   499350858  25257692  364.738062
1991   499350858  25257692  362.067205
1997      108070  25257692  152.336645
1998      108070  25257692  136.407524
1999    25473286  25257692  192.093064

[930 rows x 3 columns]

routes_df record count (unfiltered):  2000
 routes2_df record count (filtered, 400 meters):  930
```

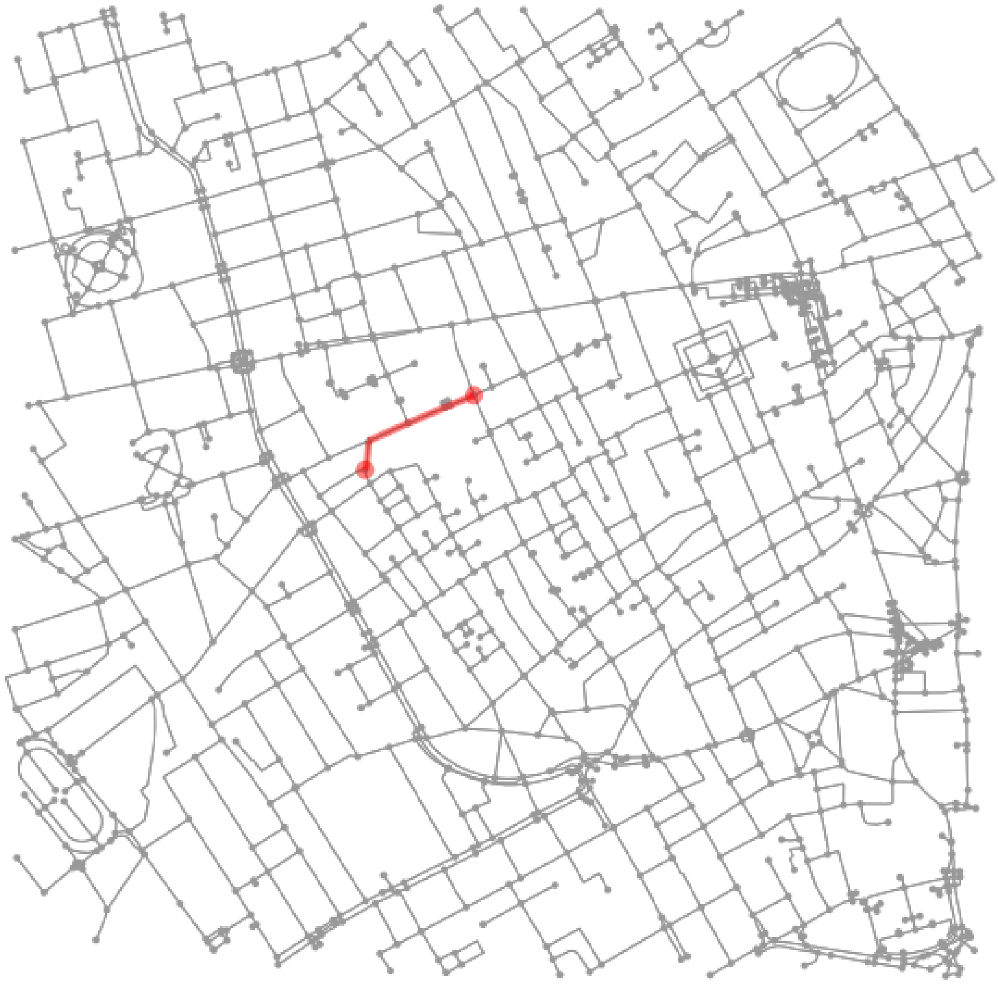
1.3.1 Visualizing node-to-node distances from computations above

Let's see what our graph looks like by plotting random routes between death and pump points. Let's look at three random samples. We will use a "random choice" selected from the python package, `numpy`.



```
Random pair number 1, death node=771982972, pump_node=25473293
```

```
Random pair number 2, death node=4233926316, pump_node=25473293
```



```
Random pair number 3, death node=9521025, pump_node=21665926
```

1.3.2 Updating `pumps_df` with mean distance values from `d1 + d2 + d3`

Let's store the mean distances for each pump from `routes_df` to `pumps_df`.

```
Pump 0:
  Node ID:  25473293
 Mean Distance:  183.9160322148311 meters
Pump 1:
  Node ID:  21665926
 Mean Distance:  258.0191657035271 meters
Pump 2:
  Node ID:  4684520654
```

```
 Mean Distance:  342.4008608219888 meters
Pump 3:
  Node ID:  107807
 Mean Distance:  354.6350938821946 meters
Pump 4:
  Node ID:  348875443
 Mean Distance:  289.5648468807347 meters
Pump 5:
  Node ID:  25473300
 Mean Distance:  286.0317983694777 meters
Pump 6:
  Node ID:  1663004187
 Mean Distance:  358.37847641438447 meters
Pump 7:
  Node ID:  25257692
 Mean Distance:  306.5233567622205 meters
```

```
        LON        LAT      GLON       GLAT   DISTANCE        NODE  \
0 -0.136668  51.513341 -0.136533  51.513391  10.882847    25473293
1 -0.139586  51.513876 -0.139462  51.513861   8.746434    21665926
2 -0.139671  51.514906 -0.139904  51.514855  17.076771  4684520654
3 -0.131630  51.512354 -0.131466  51.512196  20.870686      107807
4 -0.133594  51.512139 -0.133606  51.512189   5.593945   348875443
5 -0.135919  51.511542 -0.135762  51.511404  18.794818    25473300
6 -0.133962  51.510019 -0.133994  51.510125  11.952066  1663004187
7 -0.138199  51.511295 -0.138178  51.511281   2.169020    25257692

   MEAN_DISTANCE
0     183.916032
1     258.019166
2     342.400861
3     354.635094
4     289.564847
5     286.031798
6     358.378476
7     306.523357
```

What does the column `MEAN_DISTANCE` mean?

1.4 Step 5: Create `folium` map to show which pumps have mean shortest walkable distance values to most death points

1.4.1 Recreate Notebook 1 map and markers for pumps and deaths

```
<folium.folium.Map at 0x7f0f08b7ca58>
```

Let's now identify which pump has the shortest walkable distance (filter=400 meters) from death locations.

```
<folium.folium.Map at 0x7f0f08b7ca58>
```

1.5 Putting it All Together

Now that you have read all the narratives and code explanations, and seen the outputs of all the code, we can put all these together into one "program".

```
<folium.folium.Map at 0x7fdfa4b44cc0>
```

1.6 Congratulations!

You have just gained some expertise in: 1. The Cholera Outbreak in 1854 London 2. Use of a few Python packages for data analysis and visualization: `pandas`, `folium` 3. Use of the Open Street Maps - NetworkX package, `osmnx` for street network type analysis 4. Generating value out of information sources (mortality and street network data)

1.7 References

1. Boeing, Geoff. OSMnx: Python for Street Networks. URL: https://geoffboeing.com/2016/11/osmnx-python-street-networks/
2. Networkx. URL: https://networkx.github.io/
3. Shiode S. Revisiting John Snow's map: network-based spatial demarcation of cholera area. International Journal of Geographical Information Science Volume 26, 2012 - Issue 1. URL: https://www.tandfonline.com/doi/abs/10.1080/13658816.2011.577433.