Spatial Network Analysis

Using Osmnx and Networkx

Zhejun Qiu

Apr 18th 2023

How do we know the government’s capacity to combat drug crimes?

1. The distance/traveling time from police office to the borderlines
   - Shortest path + Shortest time ← Spatial Network Analysis

2. Content:
   - OSMnx: Retrieve transportation network
   - Networkx: Centrality + Shortest Path
   - Networkx + Matplitlob: Visualizing Spatial Network

Installing and importing libraries

```
# OSMnx requires additional libraries such as geopandas and matplotlib
```

```
pip install geopandas
pip install matplotlib
pip install network
pip install osmnx
```

```
# Here I also import shapely.geometry to get linestrings and points
```

```
import geopandas as gpd
import osmnx as ox
import networkx as nx
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
from shapely.geometry import LineString, Point
```

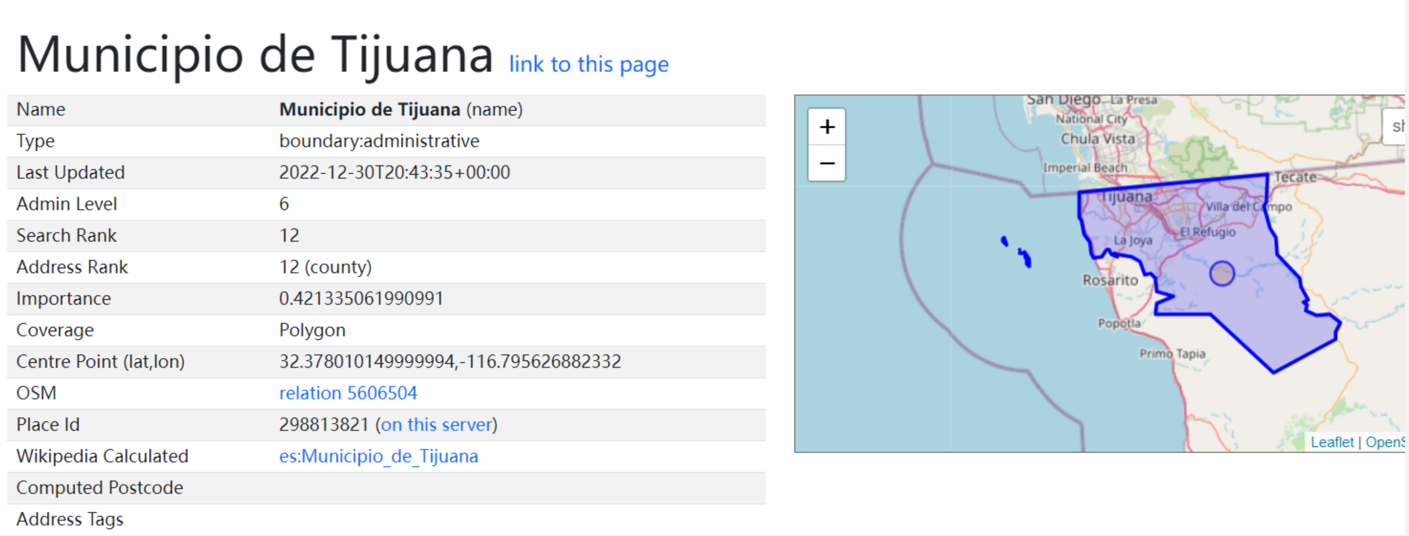
OSMnx: Retrieving Open Street Maps

# Not sure about the placename? see https://nominatim.openstreetmap.org/

```
place_name = "Municipio de Tijuana, Baja California, Mexico"
graph = ox.graph_from_place(place_name, network_type="drive")
graph_proj = ox.project_graph(graph)
```

# project graph network (for more accurate calculation)

Originally using (WGS84, EPSG:4326), now converted to Cartesian coordinates (x, y)



Municipio de Tijuana link to this page

Name	**Municipio de Tijuana** (name)
Type	boundary:administrative
Last Updated	2022-12-30T20:43:35+00:00
Admin Level	6
Search Rank	12
Address Rank	12 (county)
Importance	0.421335061990991
Coverage	Polygon
Centre Point (lat,lon)	32.378010149999994,-116.795626882332
OSM	relation 5606504
Place Id	298813821 (on this server)
Wikipedia Calculated	es:Municipio_de_Tijuana
Computed Postcode	
Address Tags	

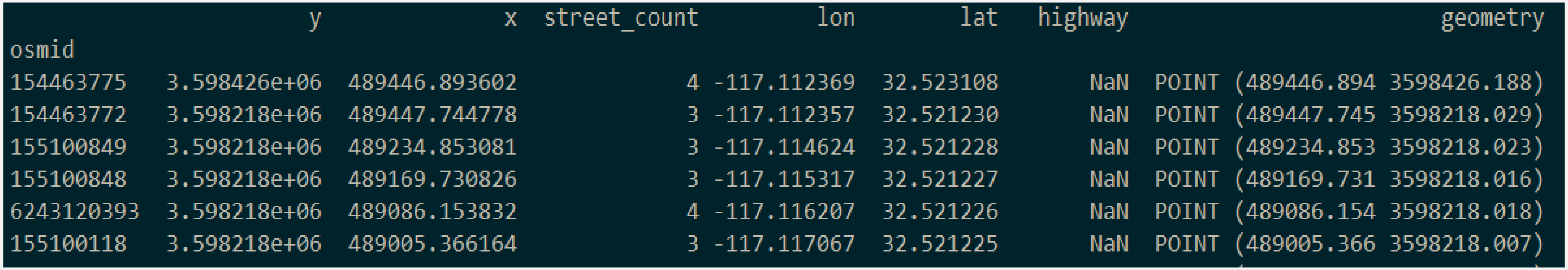
OSMnx: Retrieving Open Street Maps

# Not sure about the placename? see https://nominatim.openstreetmap.org/

```
place_name = "Municipio de Tijuana, Baja California, Mexico"
graph = ox.graph_from_place(place_name, network_type="drive")
graph_proj = ox.project_graph(graph)
```

# project graph network (for more accurate calculation)

Originally using (WGS84, EPSG:4326), now converted to Cartesian coordinates (x, y)



```
                       y              x  street_count         lon        lat  highway                      geometry
osmid
154463775   3.598426e+06  489446.893602             4 -117.112369  32.523108      NaN  POINT (489446.894 3598426.188)
154463772   3.598218e+06  489447.744778             3 -117.112357  32.521230      NaN  POINT (489447.745 3598218.029)
155100849   3.598218e+06  489234.853081             3 -117.114624  32.521228      NaN  POINT (489234.853 3598218.023)
155100848   3.598218e+06  489169.730826             3 -117.115317  32.521227      NaN  POINT (489169.731 3598218.016)
6243120393  3.598218e+06  489086.153832             4 -117.116207  32.521226      NaN  POINT (489086.154 3598218.018)
155100118   3.598218e+06  489005.366164             3 -117.117067  32.521225      NaN  POINT (489005.366 3598218.007)
```

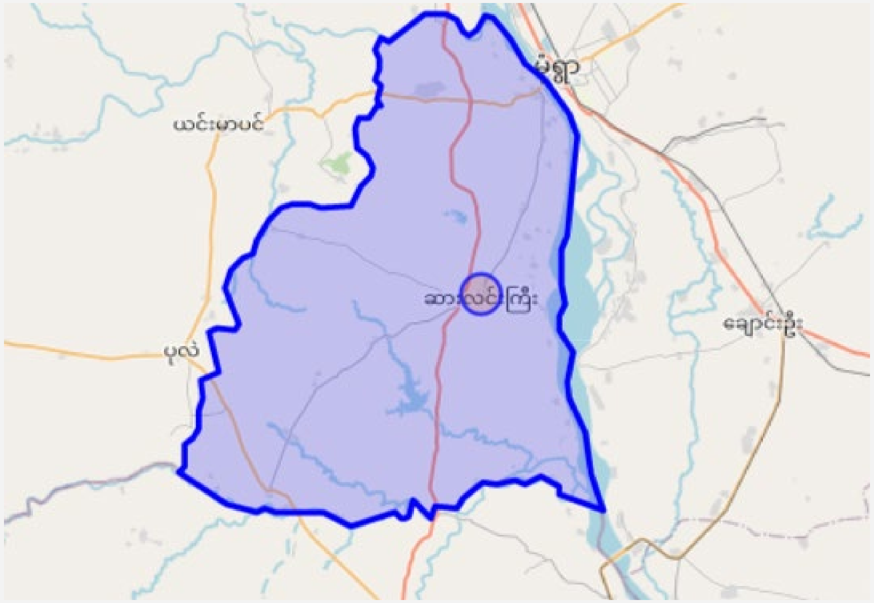
OSMnx: Retrieving Open Street Maps

# Not sure about the placename? see https://nominatim.openstreetmap.org/

```
place_name2 = "Salingyi, Sagaing, Myanmar“
graph2 = ox.graph_from_place(place_name2)
graph_proj2 = ox.project_graph(graph2)
```

# project graph network (for more accurate calculation)

Originally using (WGS84, EPSG:4326), now converted to Cartesian coordinates (x, y)



OSMnx: Retrieving Open Street Maps

# Get region area and police station location

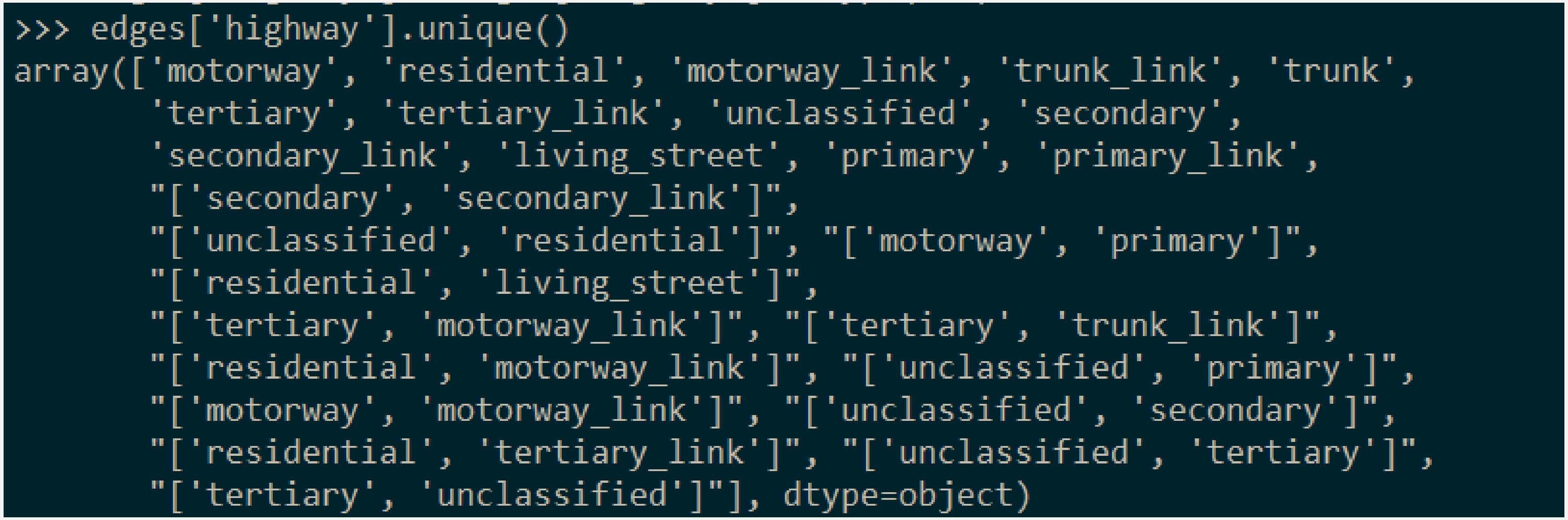
```
# get areas gdf
area = ox.geocode_to_gdf(place_name)
area_proj = ox.project_gdf(area)

# retreve buildings:
# tags = {'building': True}
tags = {"amenity": "police"}
police_station = ox.geometries_from_place(place_name, tags=tags)
police_station_proj = ox.project_gdf(police_station)
```

# building_from_place() & gdf_from_place() no longer works

OSMnx: Creating Transportation networks

# We can use method graph_to_gdfs()

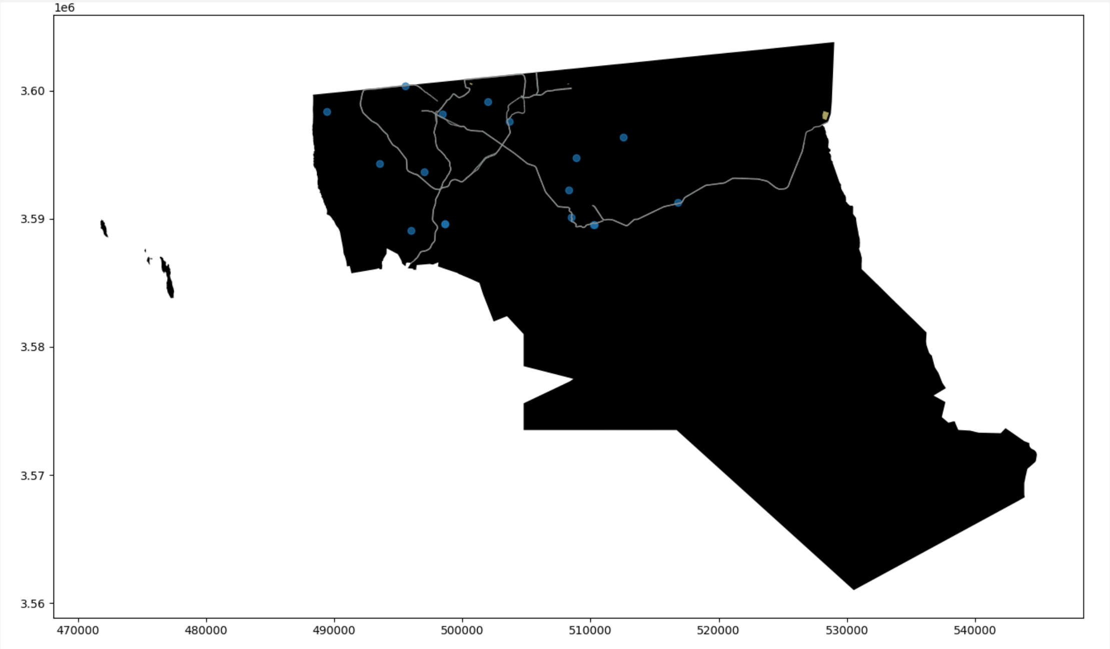
```
nodes, edges = ox.graph_to_gdfs(graph_proj)
edges['highway'] = edges['highway'].astype(str)
trunkway = edges.loc[edges['highway'] == "trunk", :]
```



```
>>> edges['highway'].unique()
array(['motorway', 'residential', 'motorway_link', 'trunk_link', 'trunk',
       'tertiary', 'tertiary_link', 'unclassified', 'secondary',
       'secondary_link', 'living_street', 'primary', 'primary_link',
       "['secondary', 'secondary_link']",
       "['unclassified', 'residential']", "['motorway', 'primary']",
       "['residential', 'living_street']",
       "['tertiary', 'motorway_link']", "['tertiary', 'trunk_link']",
       "['residential', 'motorway_link']", "['unclassified', 'primary']",
       "['motorway', 'motorway_link']", "['unclassified', 'secondary']",
       "['residential', 'tertiary_link']", "['unclassified', 'tertiary']",
       "['tertiary', 'unclassified']"], dtype=object)
```

OSMnx: Visualizing Spatial Networks

```
fig, ax = plt.subplots()
area_proj.plot(ax=ax, facecolor="black")
# edges.plot(ax=ax, linewidth=1, edgecolor='#8C8F8F')
trunkway.plot(ax=ax, linewidth=1, edgecolor='#8C8F8F')
police_station_proj.plot(ax=ax, facecolor='khaki', alpha=0.7)
plt.show()
```



Tips: Pryosm

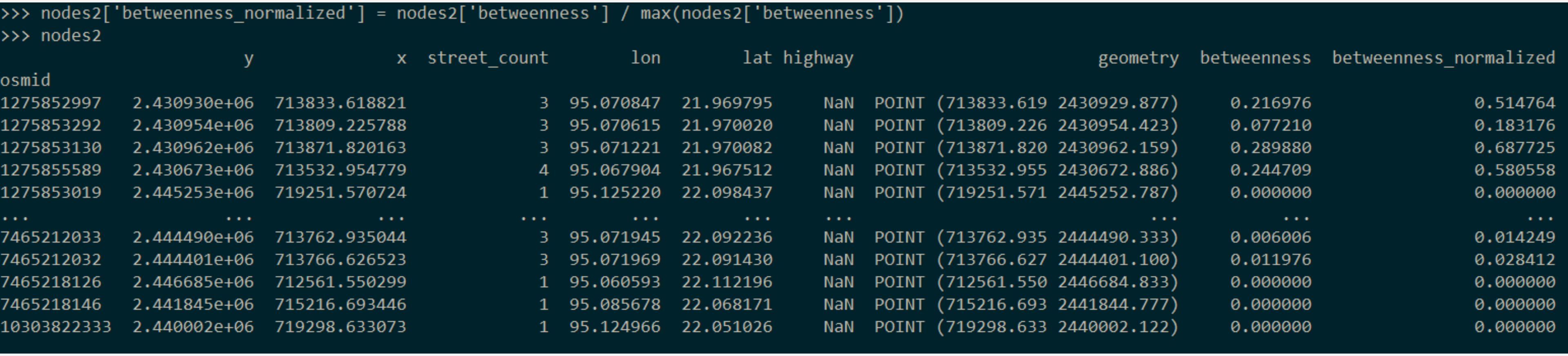
```
from pyrosm import OSM, get_data
osm = OSM(get_data("helsinki_pbf"))
roads = osm.get_network(network_type="driving")
```

# I don’t recommend that

1. Need to know the pbf file such as “myanmar-latest.osm.pbf”
2. Takes much more time than osmnx.

Networkx: Spatial Network Analysis

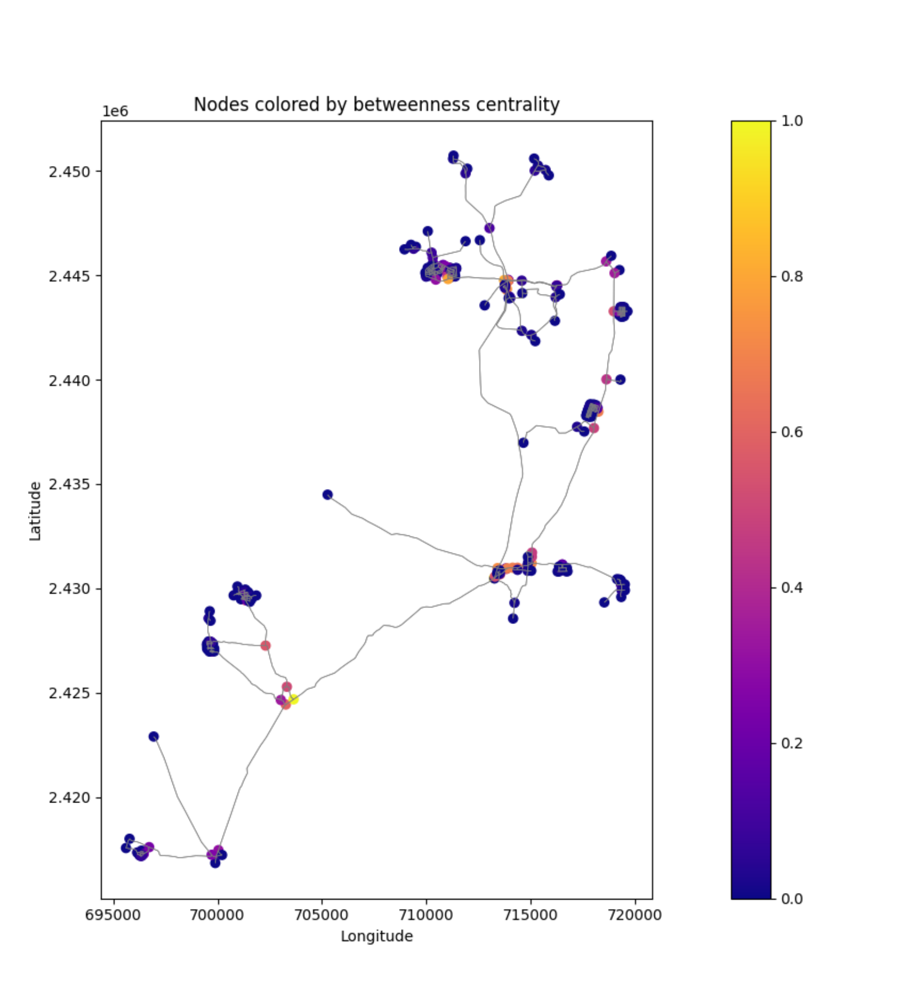
```
# Compute betweenness centrality and closeness centrality
nodes2, edges2 = ox.graph_to_gdfs(graph_proj2)
betweenness_centrality = nx.betweenness_centrality(graph_proj2, weight='length')
closeness_centrality = nx.closeness_centrality(graph_proj2, distance='length’)
Degree = nx.degree(graph_proj2, distance='length’)

# concate betweenness to nodes
nodes2['betweenness'] = nodes2.index.map(betweenness_centrality)
# Normalize betweenness centrality values to a 0-1 scale for better coloring
nodes2['betweenness_normalized'] = nodes2['betweenness'] / max(nodes2['betweenness'])
```



```
>>> nodes2['betweenness_normalized'] = nodes2['betweenness'] / max(nodes2['betweenness'])
>>> nodes2
                        y              x  street_count        lon        lat highway                        geometry  betweenness  betweenness_normalized
osmid
1275852997   2.430930e+06  713833.618821             3  95.070847  21.969795     NaN  POINT (713833.619 2430929.877)     0.216976                0.514764
1275853292   2.430954e+06  713809.225788             3  95.070615  21.970020     NaN  POINT (713809.226 2430954.423)     0.077210                0.183176
1275853130   2.430962e+06  713871.820163             3  95.071221  21.970082     NaN  POINT (713871.820 2430962.159)     0.289880                0.687725
1275855589   2.430673e+06  713532.954779             4  95.067904  21.967512     NaN  POINT (713532.955 2430672.886)     0.244709                0.580558
1275853019   2.445253e+06  719251.570724             1  95.125220  22.098437     NaN  POINT (719251.571 2445252.787)     0.000000                0.000000
...                   ...            ...           ...        ...        ...     ...                             ...          ...                     ...
7465212033   2.444490e+06  713762.935044             3  95.071945  22.092236     NaN  POINT (713762.935 2444490.333)     0.006006                0.014249
7465212032   2.444401e+06  713766.626523             3  95.071969  22.091430     NaN  POINT (713766.627 2444401.100)     0.011976                0.028412
7465218126   2.446685e+06  712561.550299             1  95.060593  22.112196     NaN  POINT (712561.550 2446684.833)     0.000000                0.000000
7465218146   2.441845e+06  715216.693446             1  95.085678  22.068171     NaN  POINT (715216.693 2441844.777)     0.000000                0.000000
10303822333  2.440002e+06  719298.633073             1  95.124966  22.051026     NaN  POINT (719298.633 2440002.122)     0.000000                0.000000
```

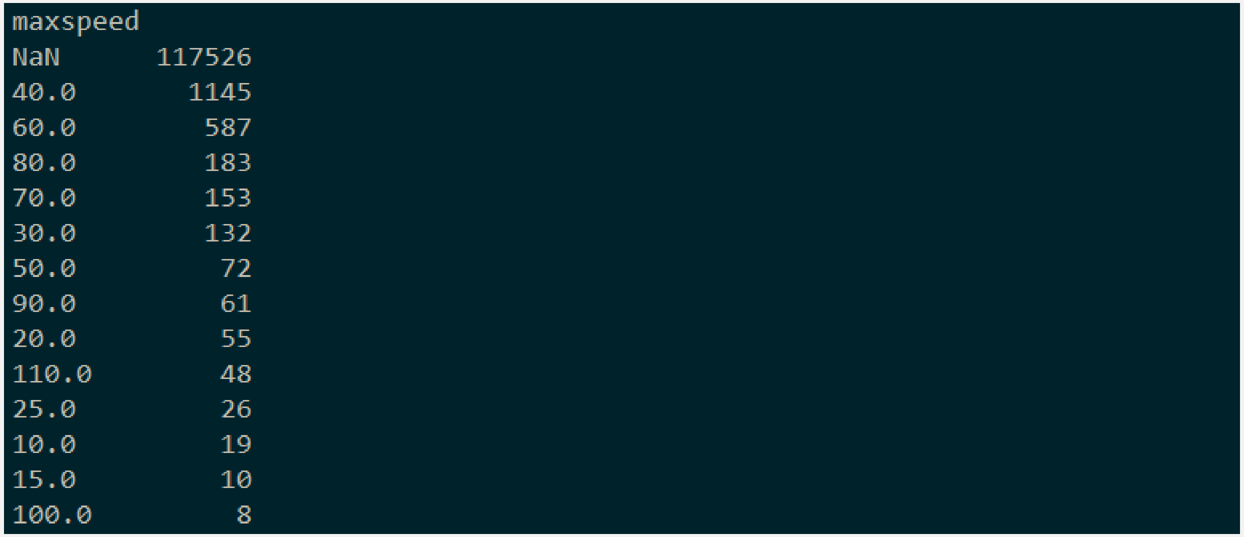
Networkx: Visualizing Centrality

```
fig, ax2 = plt.subplots(figsize=(10, 10))
ax2.set_aspect('equal')
edges2.plot(ax=ax2, edgecolor="gray",
linewidth=0.5)
nodes2.plot(ax=ax2,
column="betweenness_normalized",
cmap='plasma', legend=True)
ax2.set_title("Nodes colored by betweenness
centrality")
ax2.set_xlabel("Longitude")
ax2.set_ylabel("Latitude")
plt.show()
```



Networkx: Calculating Traveling time

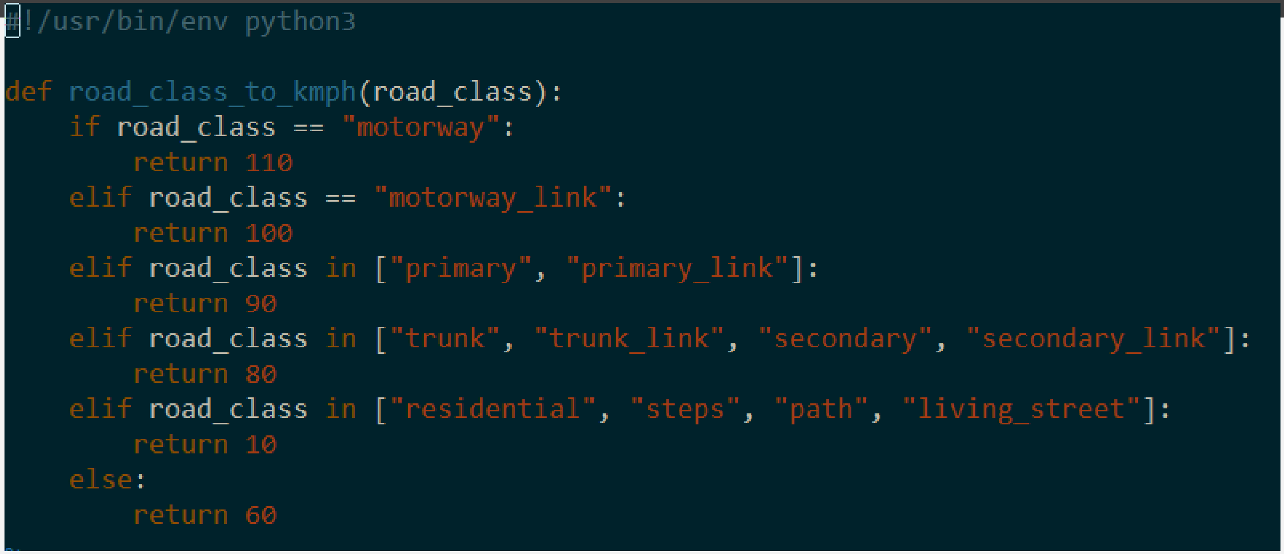
```
edges["maxspeed"].value_counts(dropna=False)
```



```
maxspeed
NaN       117526
40.0        1145
60.0         587
80.0         183
70.0         153
30.0         132
50.0          72
90.0          61
20.0          55
110.0         48
25.0          26
10.0          19
15.0          10
100.0          8
```

Networkx: Calculating Traveling time

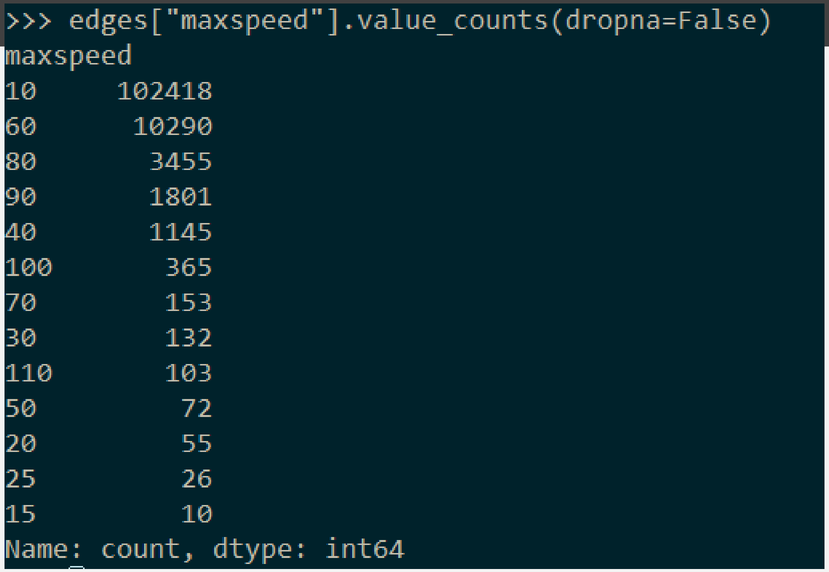
```
edges = edges.loc[~edges["highway"].isin(['cycleway', 'footway', 'pedestrian', 'trail',
'crossing'])].copy()
mask = edges["maxspeed"].isnull()
edges_without_maxspeed = edges.loc[mask].copy()
edges_with_maxspeed = edges.loc[~mask].copy()
edges_without_maxspeed["maxspeed"] =
edges_without_maxspeed["highway"].apply(speed_cal.road_class_to_kmph)
edges = pd.concat([edges_with_maxspeed, edges_without_maxspeed], ignore_index=True)
```



```
#!/usr/bin/env python3

def road_class_to_kmph(road_class):
    if road_class == "motorway":
        return 110
    elif road_class == "motorway_link":
        return 100
    elif road_class in ["primary", "primary_link"]:
        return 90
    elif road_class in ["trunk", "trunk_link", "secondary", "secondary_link"]:
        return 80
    elif road_class in ["residential", "steps", "path", "living_street"]:
        return 10
    else:
        return 60
```

Networkx: Calculating Traveling time

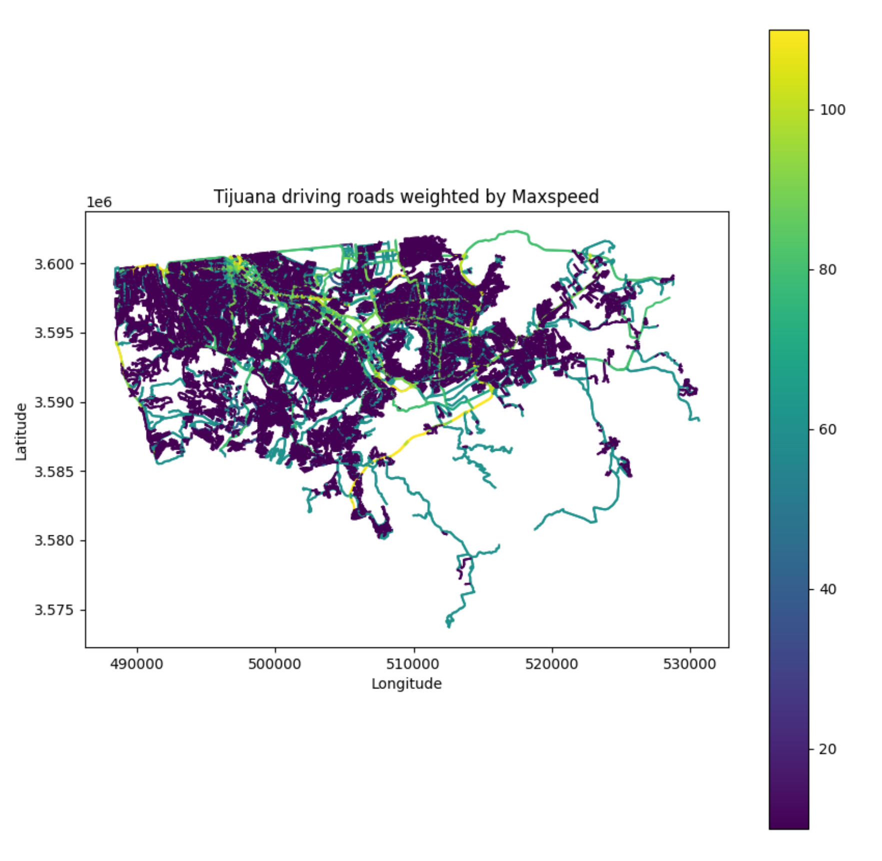
```
edges = edges.loc[~edges["highway"].isin(['cycleway', 'footway', 'pedestrian', 'trail',
'crossing'])].copy()
mask = edges["maxspeed"].isnull()
edges_without_maxspeed = edges.loc[mask].copy()
edges_with_maxspeed = edges.loc[~mask].copy()
edges_without_maxspeed["maxspeed"] =
edges_without_maxspeed["highway"].apply(speed_cal.road_class_to_kmph)
edges = pd.concat([edges_with_maxspeed, edges_without_maxspeed], ignore_index=True)
```

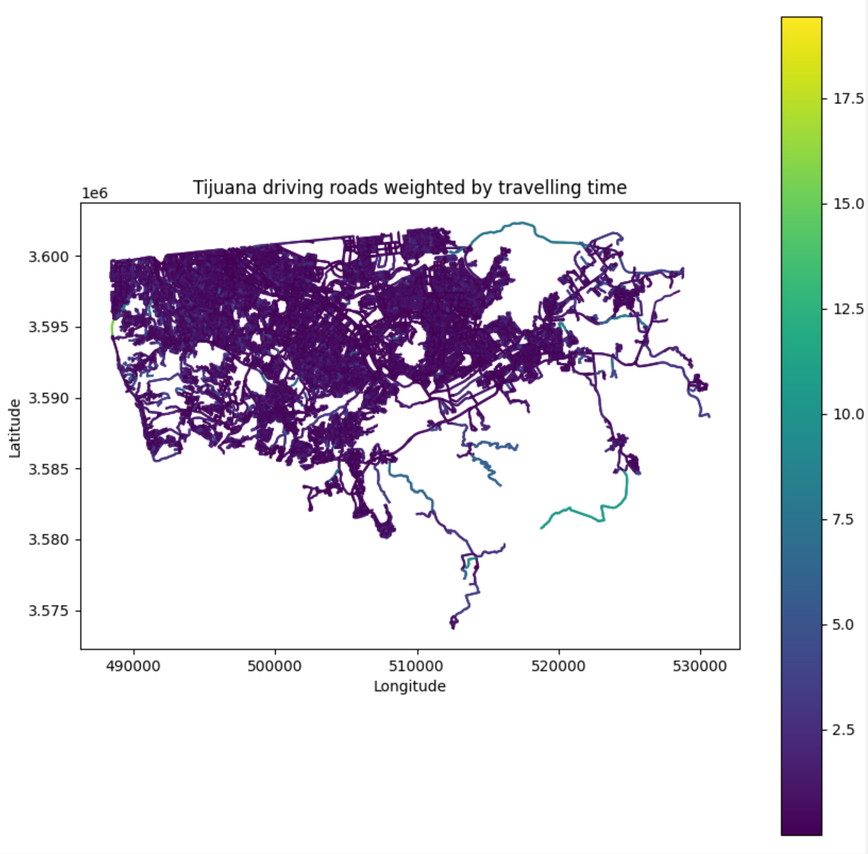


```
>>> edges["maxspeed"].value_counts(dropna=False)
maxspeed
10      102418
60       10290
80        3455
90        1801
40        1145
100        365
70         153
30         132
110        103
50          72
20          55
25          26
15          10
Name: count, dtype: int64
```

Networkx: Calculating Traveling time

```
edges["maxspeed"] = edges["maxspeed"].astype(int)
edges["travel_time_minutes"] = (edges["length"] / (edges["maxspeed"]/3.6))/60
edges.loc[0:10, ["maxspeed", "highway", "travel_time_minutes"]]
```



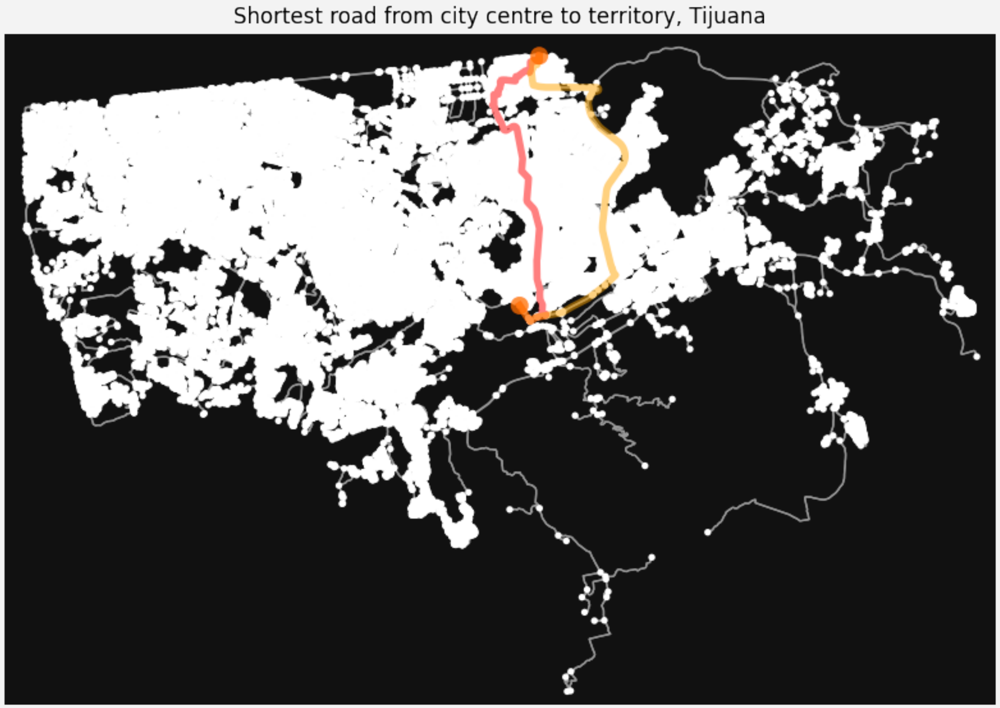


Networkx: Shortest Path

```
centroid = edges.unary_union.convex_hull.centroid
nodes["y"] = nodes["y"].astype(float)

maxy = nodes["y"].max()
target_loc = nodes.loc[nodes["y"] == maxy, :]
target_point = target_loc.geometry.values[0]


st_node_id, dist_to_st = ox.distance.nearest_nodes(graph_proj, centroid.x, centroid.y,
return_dist=True)
ed_node_id, dist_to_ed = ox.distance.nearest_nodes(graph_proj, target_point.x,
target_point.y, return_dist=True)

rt1 = nx.shortest_path(graph_proj, source=st_node_id, target=ed_node_id, weight='length')
rt2 = nx.shortest_path(graph_proj, source=st_node_id, target=ed_node_id,
weight='travel_time_minutes')
```

Networkx: Shortest Path

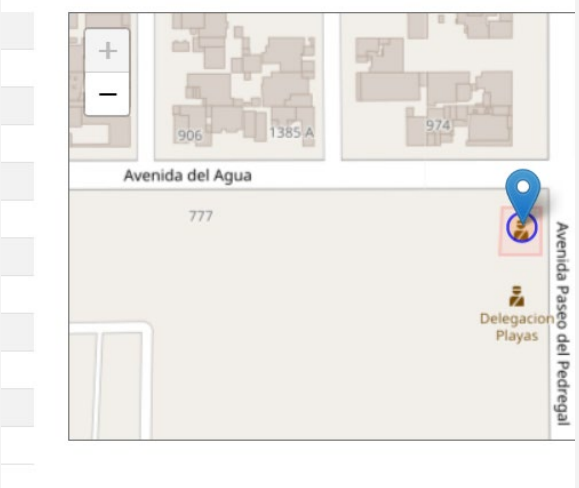
```
fig, ax5 = ox.plot_graph(graph_proj, figsize=(10, 10), close=False, show=False)
ox.plot_graph_route(graph_proj, rt1, ax=ax5, close=False, show=False, route_color='red')
ox.plot_graph_route(graph_proj, rt2, ax=ax5, close=False, show=False, route_color='orange')
ax5.set_aspect('equal')
ax5.set_title("Shortest road from city centre to territory, Tijuana")
plt.show()
```



Networkx: Shortest Path

```
# Destination
dest_address = "Delfin del Pacifico, Municipio de Tijuana"
dest_y, dest_x = ox.geocode(dest_address)  # In an order (y, x)
dest = Point(dest_x, dest_y)
dest = ox.projection.project_geometry(dest)
# Origin
orig_address = "Delegacion Playas, Municipio de Tijuana"
orig_y, orig_x = ox.geocode(orig_address)
orig = Point(orig_x, orig_y)
orig = ox.projection.project_geometry(orig)
```

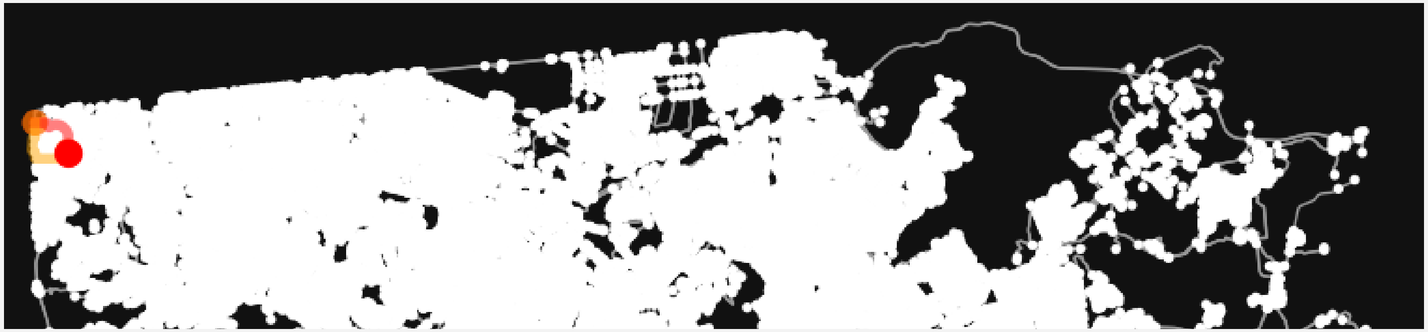
show map bounds
Vista del Pacifico
San Diego
Autopista Tijuan
San Diego County
Bunker Hill
Autopista Tijuana-Ensenada
Los Laureles
Imperial Beach
International Friendship Park
Playas de Tijuana Secc Terrazas
Tijuana
Playas de Tijuana Secc Monumental
Playas de Tijuana Secc Coronado
Puesta del Sol
La Cuspide
Playas de Tijuana Secc Costa
Imperial Beach
MEX 1D



Networkx: Shortest Path

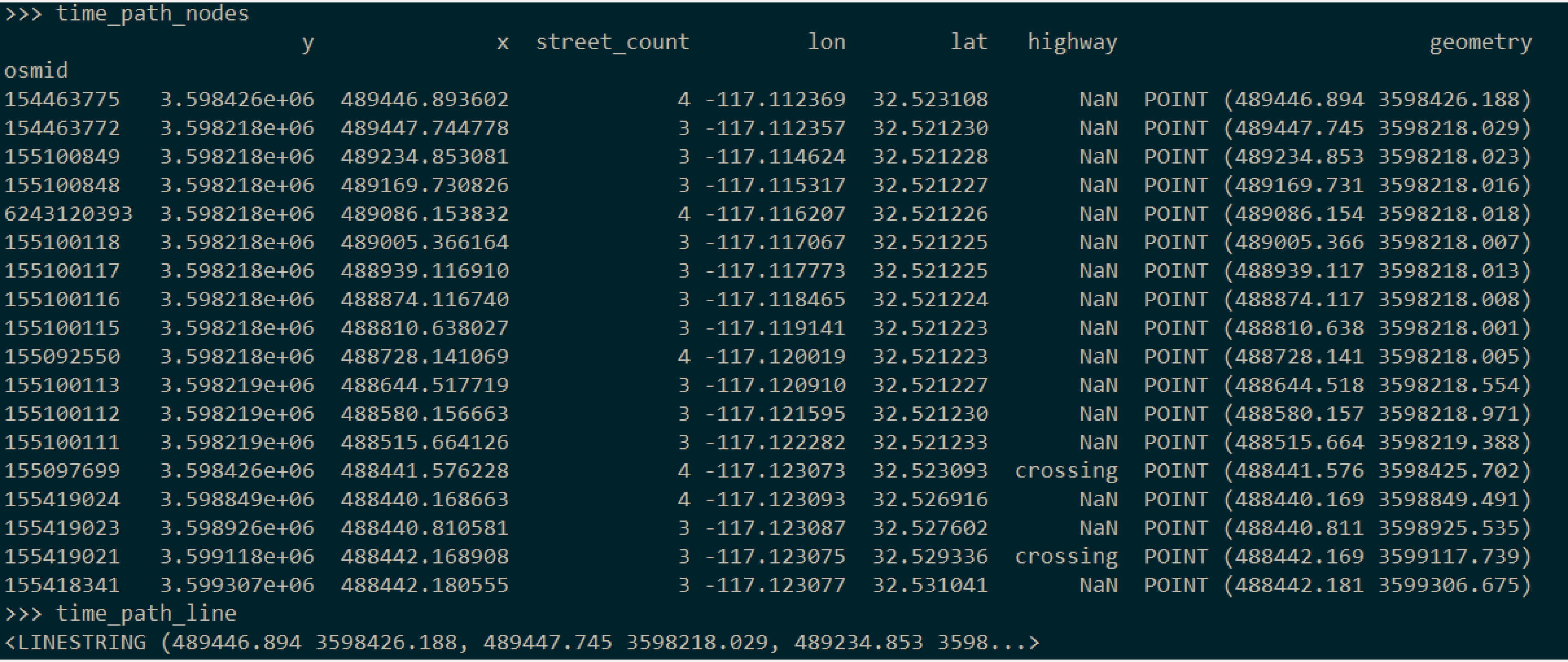
```
metric_path = nx.dijkstra_path(graph_proj, source=orig_node_id, target=dest_node_id,
weight='length')
time_path = nx.dijkstra_path(graph_proj, source=orig_node_id, target=dest_node_id,
weight='travel_time_minutes')

travel_length = nx.dijkstra_path_length(graph_proj, source=orig_node_id,
target=dest_node_id, weight='length')
travel_time = nx.dijkstra_path_length(graph_proj, source=orig_node_id, target=dest_node_id,
weight='travel_time_minutes')
```

Shortest path distance 1768.7 meters, travel time 17.0 minutes.



Networkx: Shortest Path

```
time_path_nodes = nodes.loc[time_path]
time_path_line = LineString(list(time_path_nodes.geometry.values))
```



```
>>> time_path_nodes
                      y              x  street_count         lon        lat   highway                        geometry
osmid
154463775   3.598426e+06  489446.893602             4 -117.112369  32.523108       NaN  POINT (489446.894 3598426.188)
154463772   3.598218e+06  489447.744778             3 -117.112357  32.521230       NaN  POINT (489447.745 3598218.029)
155100849   3.598218e+06  489234.853081             3 -117.114624  32.521228       NaN  POINT (489234.853 3598218.023)
155100848   3.598218e+06  489169.730826             3 -117.115317  32.521227       NaN  POINT (489169.731 3598218.016)
6243120393  3.598218e+06  489086.153832             4 -117.116207  32.521226       NaN  POINT (489086.154 3598218.018)
155100118   3.598218e+06  489005.366164             3 -117.117067  32.521225       NaN  POINT (489005.366 3598218.007)
155100117   3.598218e+06  488939.116910             3 -117.117773  32.521225       NaN  POINT (488939.117 3598218.013)
155100116   3.598218e+06  488874.116740             3 -117.118465  32.521224       NaN  POINT (488874.117 3598218.008)
155100115   3.598218e+06  488810.638027             3 -117.119141  32.521223       NaN  POINT (488810.638 3598218.001)
155092550   3.598218e+06  488728.141069             4 -117.120019  32.521223       NaN  POINT (488728.141 3598218.005)
155100113   3.598219e+06  488644.517719             3 -117.120910  32.521227       NaN  POINT (488644.518 3598218.554)
155100112   3.598219e+06  488580.156663             3 -117.121595  32.521230       NaN  POINT (488580.157 3598218.971)
155100111   3.598219e+06  488515.664126             3 -117.122282  32.521233       NaN  POINT (488515.664 3598219.388)
155097699   3.598426e+06  488441.576228             4 -117.123073  32.523093  crossing  POINT (488441.576 3598425.702)
155419024   3.598849e+06  488440.168663             4 -117.123093  32.526916       NaN  POINT (488440.169 3598849.491)
155419023   3.598926e+06  488440.810581             3 -117.123087  32.527602       NaN  POINT (488440.811 3598925.535)
155419021   3.599118e+06  488442.168908             3 -117.123075  32.529336  crossing  POINT (488442.169 3599117.739)
155418341   3.599307e+06  488442.180555             3 -117.123077  32.531041       NaN  POINT (488442.181 3599306.675)
>>> time_path_line
<LINESTRING (489446.894 3598426.188, 489447.745 3598218.029, 489234.853 3598...>
```

# For more complicated calculation? Try r5py!