Lab3_submission

January 26, 2022

1 Brooke Hunter Lab 3 Submission

Lab 3: Where is the nearest **Theatre**?

Objectives: * We will explore OpenStreetMap (OSM) data using osmnx. * Learn about OSM data stuctures * Compute walking distances using just a few lines of code * Visualize our data using `folium`

1.1 Question 1 (20 points):

Write a script that:

- Computes the Euclidean distance to another **amenity** of your choosing (HINT: use `gdf['amenity'].unique()` to list the different amenities). Feel free to download OSM buildings from another place and choose a different home' location.

- Makes an interactive map showing where your ten nearest amenities are using `folium`.

[1]:
```python
# Import modules
import osmnx as ox

import numpy as np
import pandas as pd
import geopandas as gpd

from shapely.geometry.polygon import Polygon
from shapely.geometry.multipolygon import MultiPolygon
from shapely.geometry import LineString, MultiLineString

# Specify type of data
tags = {'building': True}

# Download building geometries from OSM
gdf = ox.geometries_from_place('Eugene, Oregon, USA', tags)
```

```
C:\Users\brdeh\anaconda3\envs\lab3\lib\site-packages\osmnx\geometries.py:805:
ShapelyDeprecationWarning: __len__ for multi-part geometries is deprecated and
will be removed in Shapely 2.0. Check the length of the `geoms` property instead
to get the  number of parts of a multi-part geometry.
  for merged_outer_linestring in list(merged_outer_linestrings):
```

```
C:\Users\brdeh\anaconda3\envs\lab3\lib\site-packages\osmnx\geometries.py:805:
ShapelyDeprecationWarning: Iteration over multi-part geometries is deprecated
and will be removed in Shapely 2.0. Use the `geoms` property to access the
constituent parts of a multi-part geometry.
  for merged_outer_linestring in list(merged_outer_linestrings):
```

[2]:
```python
print(gdf.columns.tolist())
```

```
['addr:state', 'building', 'ele', 'gnis:county_id', 'gnis:created',
'gnis:feature_id', 'name', 'operator', 'geometry', 'access', 'wheelchair',
'source', 'ref', 'amenity', 'description', 'opening_hours', 'information',
'tourism', 'addr:city', 'addr:housenumber', 'addr:street', 'brand',
'brand:wikidata', 'brand:wikipedia', 'cuisine', 'takeaway', 'addr:postcode',
'bus', 'network', 'public_transport', 'fee', 'leisure', 'sport', 'material',
'emergency', 'nodes', 'building:levels', 'gnis:county_name', 'internet_access',
'shop', 'url', 'wikidata', 'wikipedia', 'name:ja', 'phone', 'website',
'air_conditioning', 'delivery', 'diet:vegan', 'denomination', 'religion',
'internet_access:fee', 'smoking', 'government', 'office', 'email', 'layer',
'location', 'man_made', 'payment:cash', 'payment:credit_cards',
'payment:debit_cards', 'drive_through', 'short_name', 'boundary', 'heritage',
'heritage:operator', 'nrhp:criteria', 'nrhp:inscription_date', 'nrhp:nhl',
'protection_title', 'ref:nrhp', 'name_1', 'alt_name', 'bar', 'switch',
'baseball:bullpen', 'baseball:dugout_fence', 'baseball:outfield_fence',
'baseball:safety_net', 'baseball:scoreboard', 'baseball:warning_track', 'lit',
'name:etymology', 'name:etymology:wikidata', 'name:etymology:wikipedia',
'start_date', 'height', 'loc_name', 'addr:unit', 'addr:housename',
'roof:levels', 'shelter_type', 'roof:shape', 'payment:mastercard',
'payment:visa', 'stroller', 'second_hand', 'addr:country', 'name:fa',
'operator:wikidata', 'operator:wikipedia', 'ref:walmart', 'note',
'source:position', 'gnis:edited', 'operator:type', 'atm', 'retreat',
'retreat:for', 'retreat:operator', 'retreat:operator:wikidata', 'studio',
'healthcare', 'social_facility', 'social_facility:for', 'official_name',
'craft', 'training', 'addr:county', 'healthcare:speciality', 'military',
'branch', 'outdoor_seating', 'diet:meat', 'diet:vegetarian', 'diet:gluten_free',
'historic', 'screen', 'image', 'opening_hours:covid19',
'opening_hours:drive_through', 'area', 'parking', 'internet_access:ssid',
'owner', 'old_name', 'source:name', 'nohousenumber', 'grades', 'content',
'service', 'fuel:diesel', 'fuel:octane_95', 'fuel:octane_98', 'self_service',
'organic', 'postal_code', 'disused:name', 'disused:shop', 'indoor', 'fax',
'stars', 'clothes', 'level', 'min_age', 'shop:herbs', 'shop:spices', 'shop:tea',
'building:min_level', 'microbrewery', 'wifi', 'service:vehicle:car_repair',
'service:vehicle:inspection', 'service:vehicle:oil_change', 'capacity',
'fuel:gasoline_87', 'fuel:gasoline_89', 'fuel:gasoline_91',
'health_facility:type', 'medical_system:western', 'toilets:disposal',
'toilets:handwashing', 'membership', 'bench', 'architect', 'club', 'bridge',
'tower:type', 'consulting', 'attraction', 'rooms', 'fuel:octane_87',
'fuel:octane_89', 'fuel:octane_92', 'fuel:biodiesel', 'building:material',
'number_of_apartments', 'fixme', 'recycling:cans', 'recycling:glass_bottles',
```

```
'recycling:plastic_bottles', 'recycling_type', 'beauty', 'payment:coins',
'fuel:ethanol', 'fuel:gasoline', 'payment:bitcoin', 'payment:bitcoincash',
'service:vehicle:diagnostics', 'contact:facebook', 'contact:fax',
'contact:linkedin', 'contact:phone', 'payment:american_express',
'payment:cheque', 'payment:discover_card', 'works', 'bicycle_parking',
'abandoned', 'tower:construction', 'building:flats', 'healthcare:counselling',
'dance:teaching', 'changing_table', 'unisex', 'bin', 'residential', 'elevation',
'mapillary', 'building:levels:roof', 'house:soliciting', 'soliciting',
'covered', 'facebook', 'opening_hours:url', 'yelp', 'surface', 'ways', 'type',
'contact:website']
```

[3]:
```
# Count number of non-NaNs in each column
gdf.count()
```

[3]:
```
addr:state           565
building           55533
ele                   24
gnis:county_id        16
gnis:created          17
                   …
yelp                   1
surface                1
ways                  77
type                  77
contact:website        1
Length: 231, dtype: int64
```

[4]:
```
gdf['amenity'].unique()
```

[4]:
```
array([nan, 'restaurant', 'fuel', 'fire_station', 'cafe',
       'place_of_worship', 'fast_food', 'library', 'theatre', 'shelter',
       'school', 'bank', 'studio', 'dentist', 'social_facility',
       'training', 'pub', 'college', 'cinema', 'conference_centre',
       'community_centre', 'police', 'parking', 'doctors', 'post_office',
       'clinic', 'bus_station', 'prison', 'courthouse', 'veterinary',
       'music_school', 'bar', 'nightclub', 'car_wash', 'animal_shelter',
       'toilets', 'biergarten', 'childcare', 'recycling', 'marketplace',
       'bicycle_parking', 'arts_centre', 'events_venue', 'social_centre',
       'ice_cream'], dtype=object)
```

1.1.1 Filter Theatres below

[5]:
```
# Filter theatres
theatres = gdf[gdf['amenity'] == 'theatre'].reset_index()
theatres
```

[5]:
```
  element_type      osmid addr:state building  ele gnis:county_id  \
0          way  203427041         OR     roof  NaN            NaN
```

```
1           way  311045614        NaN       yes  NaN            NaN
2           way  315741025        NaN       yes  NaN            NaN
3           way  412267069        NaN       yes  NaN            NaN
4           way  420119848        NaN       yes  NaN            NaN

  gnis:created gnis:feature_id                              name operator  \
0          NaN             NaN               Cuthbert Amphitheater      NaN
1          NaN             NaN                Upstart Crow Studios      NaN
2          NaN             NaN                  Very Little Theater      NaN
3          NaN             NaN  Hult Center for the Performing Arts      NaN
4          NaN             NaN            Oregon Contemporary Theatre      NaN

   … house:soliciting soliciting covered facebook opening_hours:url yelp  \
0  …              NaN        NaN     NaN      NaN               NaN  NaN
1  …              NaN        NaN     NaN      NaN               NaN  NaN
2  …              NaN        NaN     NaN      NaN               NaN  NaN
3  …              NaN        NaN     NaN      NaN               NaN  NaN
4  …              NaN        NaN     NaN      NaN               NaN  NaN

  surface ways type contact:website
0     NaN  NaN  NaN             NaN
1     NaN  NaN  NaN             NaN
2     NaN  NaN  NaN             NaN
3     NaN  NaN  NaN             NaN
4     NaN  NaN  NaN             NaN

[5 rows x 233 columns]
```

1.1.2 Reproject to UTM and get centroids of Theatres and Cascade Hall

[6]:
```
# Reproject to UTM Zone 10N
gdf = gdf.to_crs('EPSG:32610')
theatres = theatres.to_crs('EPSG:32610')
# Get coordinates of Cascade Hall
cascade_hall = gdf[gdf['name'] == 'Cascade Hall'].reset_index()

# Get Theatre and Cascade Hall centroids
theatres['centroid'] = theatres['geometry'].apply(
  lambda x: x.centroid if type(x) == Polygon else (
  x.centroid if type(x) == MultiPolygon else x))

cascade_hall['centroid'] = cascade_hall['geometry'].apply(
  lambda x: x.centroid if type(x) == Polygon else (
  x.centroid if type(x) == MultiPolygon else x))
```

1.1.3 Compute Euclidean Distances from Cascade hall

[7]:
```
# Compute distances
cascade_hall_x = cascade_hall['centroid'].x.values[0]
cascade_hall_y = cascade_hall['centroid'].y.values[0]
distances = np.sqrt(((cascade_hall_x - theatres['centroid'].x.values)**2)
                    + ((cascade_hall_y - theatres['centroid'].y.values)**2))

# Add to GeoDataFrame
theatres['euclidean_distance'] = distances
# There are only 5 theatres in the data so I changed it to only print 5
print(theatres.nsmallest(5, ['euclidean_distance'])[['name',␣
↪'euclidean_distance']])
```

```
                                 name  euclidean_distance
0                Cuthbert Amphitheater         1106.016895
2                  Very Little Theater         1534.108047
3   Hult Center for the Performing Arts         1717.247614
4          Oregon Contemporary Theatre         1777.116148
1                  Upstart Crow Studios         2919.983829
```

1.1.4 Import Folium and Plot

[8]:
```
# Make a new DataFrame containing only the three most relevant columns
nearest_theatres = theatres.nsmallest(5, ['euclidean_distance'])[['name',␣
↪'euclidean_distance', 'centroid']]

# Set column geometry
nearest_theatres = nearest_theatres.set_geometry('centroid')

# Convert back to WGS84
nearest_theatres = nearest_theatres.to_crs('EPSG:4326')

# Import package
import folium

# Define center of map (i.e. Cascade Hall) and initial zoom level
lat_lon = [44.0464, -123.0736]
m = folium.Map(location=lat_lon, zoom_start=12)

for i in range(0, nearest_theatres.shape[0]):
    my_string = 'name: {}, distance: {}'.format(nearest_theatres.
↪iloc[i]['name'], nearest_theatres.iloc[i]['euclidean_distance'])
    folium.Marker([nearest_theatres.iloc[i]['centroid'].y, nearest_theatres.
↪iloc[i]['centroid'].x],
                  popup=my_string).add_to(m)

# Display map
```

```
m
```

[8]: <folium.folium.Map at 0x17bef2bd730>

1.2 Question 2 (20 points):

Adapt the code above to compute the network distance between two points (either in Eugene or in a city of your choice) and show your results using an interactive map. Write a few sentences about what your map shows.

[9]:
```
# Import module
import networkx as nx
# Define coordinates of Cascade Hall
lat_lon = (44.0464, -123.0736)

# Import walkable street network data around Cascade Hall
g = ox.graph_from_point(lat_lon, dist=3500, network_type='walk')

# Plot map
fig, ax = ox.plot_graph(g, node_size=10)
```

[10]:
```
# Convert to graph
graph_proj = ox.project_graph(g)

# Get edges and nodes separately
nodes_proj, edges_proj = ox.graph_to_gdfs(graph_proj, nodes=True, edges=True)

# Check projection is UTM Zone 10N
print("Coordinate system:", edges_proj.crs)

# Convert the theatre dataset back to UTM Zone 10N
nearest_theatres = nearest_theatres.to_crs('EPSG:32610')
```

```
Coordinate system: +proj=utm +zone=10 +ellps=WGS84 +datum=WGS84 +units=m
```

```
+no_defs +type=crs
```

[11]:
```
# Get x and y coordinates of Cascade Hall
orig_xy = (cascade_hall['centroid'].y.values[0], cascade_hall['centroid'].x.
 ↪values[0])

# Get x and y coordinates of one of the theatres (the furthest of the ten)
target_xy = (nearest_theatres['centroid'].y.values[-1],␣
 ↪nearest_theatres['centroid'].x.values[-1])
```

[12]:
```
# Find the node in the graph that is closest to the origin point (here, we want␣
 ↪to get the node id)
orig_node = ox.distance.nearest_nodes(G=graph_proj, X=orig_xy[1], Y=orig_xy[0],␣
 ↪return_dist=False)

# Find the node in the graph that is closest to the target point (here, we want␣
 ↪to get the node id)
target_node = ox.distance.nearest_nodes(graph_proj, X=target_xy[1],␣
 ↪Y=target_xy[0], return_dist=False)
```

[13]:
```
# Calculate the shortest path
route = nx.shortest_path(G=graph_proj, source=orig_node, target=target_node,␣
 ↪weight='length')
```

[14]:
```
# Plot the shortest path using folium
m = ox.plot_route_folium(g, route, weight=5)
m
```

[14]:
```
<folium.folium.Map at 0x17becb58c40>
```

1.2.1 Write a few sentances about what your map shows

This map shows the shortest walking path to get from Cascade Hall to the Upstart Crow Studios Dance Center (which is the farthest theatre in the inventory).

1.3 Question 3 (10 points):

- a) Calculate the average difference between the Euclidean and network distances for you amenities
- b) Describe some situations where it would not be advisable to use Euclidean distances?

[15]:
```
# Get the nodes along the shortest path
route_nodes = nodes_proj.loc[route]

# Create a geometry for the shortest path
route_line = LineString(list(route_nodes['geometry'].values))

# Create a GeoDataFrame
```

```
route_geom = gpd.GeoDataFrame([[route_line]], geometry='geometry',␣
↪crs=edges_proj.crs, columns=['geometry'])

# Print length of route
print('Walking distance to %s = %.1f km' % (nearest_theatres['name'].iloc[-1],␣
↪route_geom['geometry'].length / 1000))
```

```
Walking distance to Upstart Crow Studios = 3.4 km
```

[16]:
```
# Get x and y coordinates of all ten of the nearest theatres
target_xy = (nearest_theatres['centroid'].y.values,␣
↪nearest_theatres['centroid'].x.values)
```

[17]:
```
routes = []
distances = []
for i in range(len(target_xy[0])):

    # Find the node in the graph that is closest to the target point (here, we␣
↪want to get the node id)
    target_node = ox.distance.nearest_nodes(graph_proj, X=target_xy[1][i],␣
↪Y=target_xy[0][i], return_dist=False)

    # Calculate the shortest path
    route = nx.shortest_path(G=graph_proj, source=orig_node,␣
↪target=target_node, weight='length')

    # Append route to list
    routes.append(route)

    # Get the nodes along the shortest path
    route_nodes = nodes_proj.loc[route]

    # Create a geometry for the shortest path
    route_line = LineString(list(route_nodes['geometry'].values))

    # Create a GeoDataFrame
    route_geom = gpd.GeoDataFrame([[route_line]], geometry='geometry',␣
↪crs=edges_proj.crs, columns=['geometry'])

    # Print length of route
    print('Walking distance to %s = %.1f km' % (nearest_theatres['name'].
↪iloc[i], route_geom['geometry'].length / 1000))

    # Append distances to list
    distances.append(route_geom['geometry'].length[0])
```

```
Walking distance to Cuthbert Amphitheater = 3.4 km
Walking distance to Very Little Theater = 1.9 km
```

```
Walking distance to Hult Center for the Performing Arts = 1.9 km
Walking distance to Oregon Contemporary Theatre = 2.0 km
Walking distance to Upstart Crow Studios = 3.4 km
```

[18]:
```
nearest_theatres['network_distance'] = distances
nearest_theatres
```

[18]:
```
                                  name  euclidean_distance  \
0                 Cuthbert Amphitheater         1106.016895
2                   Very Little Theater         1534.108047
3   Hult Center for the Performing Arts         1717.247614
4          Oregon Contemporary Theatre         1777.116148
1                   Upstart Crow Studios         2919.983829

                         centroid  network_distance
0  POINT (493910.795 4878110.410)       3394.638547
2  POINT (493383.625 4875662.662)       1924.529999
3  POINT (492533.201 4877727.879)       1940.752486
4  POINT (492359.531 4877389.351)       1962.904129
1  POINT (491552.823 4878451.708)       3421.982324
```

- a) Calculate the average difference between the Euclidean and network distances for you amenities

[20]:
```
differences = nearest_theatres['network_distance'] -␣
 ↪nearest_theatres['euclidean_distance']
print(differences)
average = np.mean(differences)

print('The average difference between Euclidean and network distances is about␣
 ↪{:.2f} meters'.format(average))
```

```
0    2288.621652
2     390.421951
3     223.504872
4     185.787981
1     501.998494
dtype: float64
The average difference between Euclidean and network distances is about 718.07
meters
```

The average difference between the Euclidean network distance is 718.07 meters (with euclidean underestimating the distance).

- b) Describe some situations where it would not be advisable to use Euclidean distances?

Using the Euclidean distance would be bad if you had a lot of buildings in between (which you already provided this example). But also if you were in a rural area, the euclidean distance might be short, but the actual infrastucutre (roads, sidewalks, etc) may not exist to get their easily. Thus the network distance would be a lot longer. Similarly there could be two "close" objects/places on either side of a large mountain or river based on the euclidean distance. So if you just looked at the euclidean distance, you may think it is an easy path to get there to your destination… when in reality you would need to climb/swim… which isn't ideal probably. Thus the network distance that provides an feasible path to your destination would be ideal.

1.4 Remember to submit your answers to Questions 1, 2 and 3 by Friday 11:59pm

[ ]:
```

```