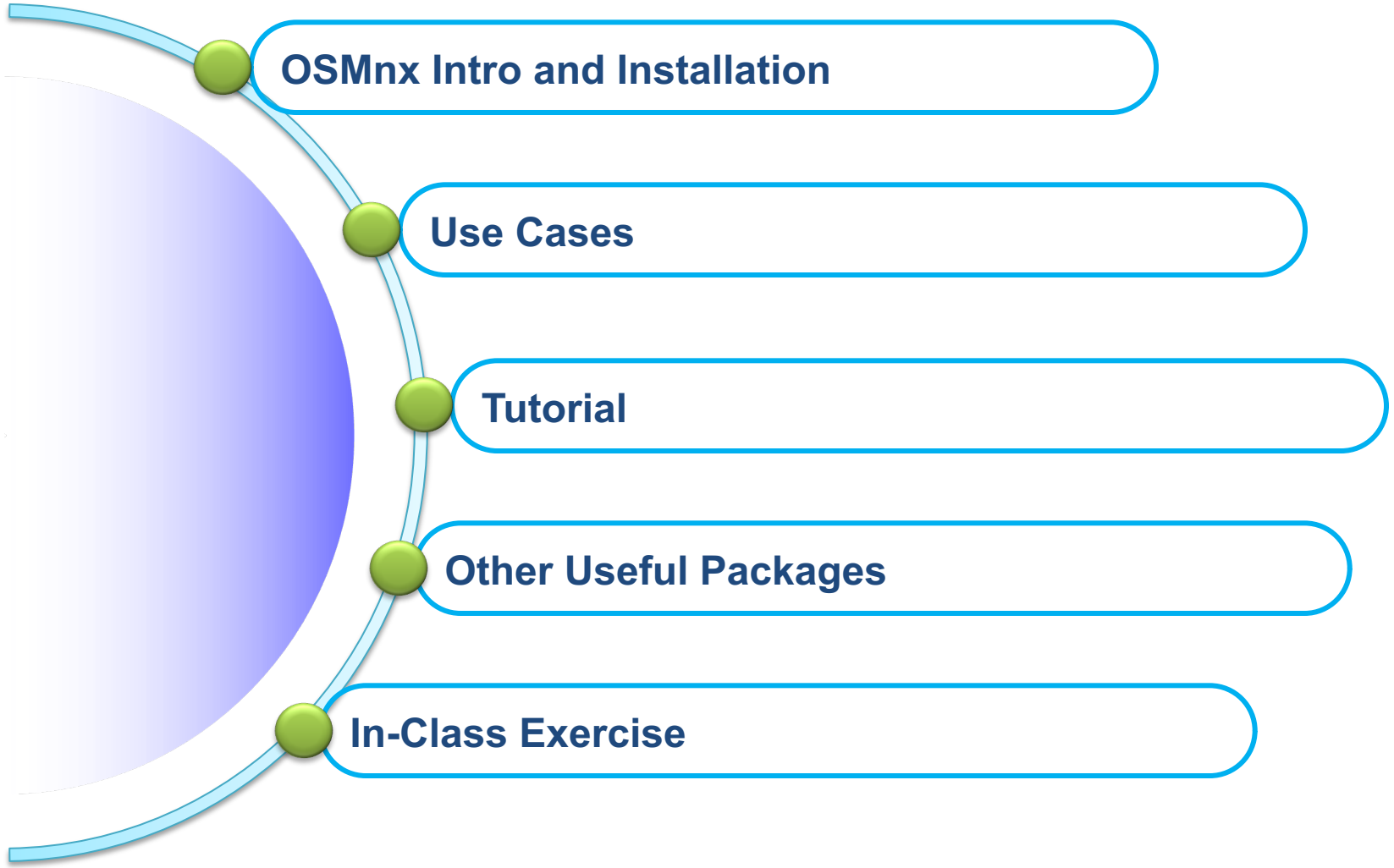
Outline


OSMnx Intro and Installation
Use Cases
Tutorial
Other Useful Packages
In-Class Exercise

What is OSMnx?

OSMnx is a Python package that lets you download geospatial data from OpenStreetMap and model, project, visualize, and analyze real-world street networks and any other geospatial geometries.

What can OSMnx be used for?

- OSMnx contains geodata that can be easily called by **name**, **coordinates** or **polygon** extraction.
- The data is extracted from OpenStreetMap and allows the user to **model**, **analyze** and **visualize** street networks.
- Extract Network analysis metrics such as: degree, closeness, eccentricity, etc.

Installing OSMnx

The package installation can be difficult.

For windows users:

- Make sure that numpy version 1.21 is installed. If you have another version use: '**conda install -c conda-forge numpy=1.21**'
- Follow: python - Fiona installation error on windows using pip - Stack Overflow
- Then: **pip install osmnx**

Recommendation: Do this in a virtual environment and install any needed packages.

Geographical Data for Urbana, IL

Drive Network

Bike Network

Walk Network

Data Extraction

In the OSMnx package, there are different ways to extract data:

Extract transportation data (network):

```
place_name = "Urbana, IL, USA"
graph = ox.graph_from_place(place_name, network_type='drive')
fig, ax = ox.plot_graph(graph)
```

Extract Shapely file :

```
city = ox.geocode_to_gdf('Urbana, IL, USA')
ax = ox.project_gdf(city).plot()
_ = ax.axis('off')
```

Network Analysis

The benefit of the package is that it can easily extract network metrics:

```
# calculate and print extended network stats
more_stats = ox.extended_stats(G, ecc=True, bc=True, cc=True)
for key in sorted(more_stats.keys()):
    print(key)
```

```
avg_neighbor_degree
avg_neighbor_degree_avg
avg_weighted_neighbor_degree
avg_weighted_neighbor_degree_avg
betweenness_centrality
betweenness_centrality_avg
center
closeness_centrality
closeness_centrality_avg
clustering_coefficient
clustering_coefficient_avg
clustering_coefficient_weighted
clustering_coefficient_weighted_avg
degree_centrality
degree_centrality_avg
diameter
eccentricity
pagerank
pagerank_max
pagerank_max_node
pagerank_min
pagerank_min_node
periphery
radius
```

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Transportation Road Network Analysis

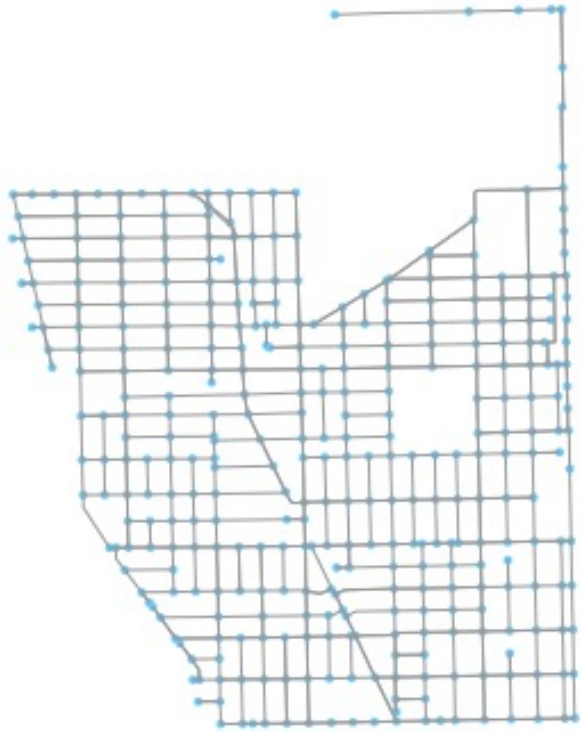
OSMnx

Uploading the Transportation Network

Using the OSMnx Package to Retrieve the Transportation Network Data

```
1  place_name = "Chicago, USA"
2  graph = ox.graph_from_place(place_name, network_type=
3  fig, ax = ox.plot_graph(graph)
```

```
1  place_name = "Lincoln Square, Chicago, USA"
2  graph = ox.graph_from_place(place_name, network_type='drive')
3  fig, ax = ox.plot_graph(graph)
```



Only displays routes that cane be travelled on using cars (roads, bridges, etc)

Embedded Information

```
edges = ox.graph_to_gdfs(graph, nodes=False, edges=True)
```

```
edges.columns
```

```
Index(['u', 'v', 'key', 'osmid', 'lanes', 'name', 'highway', 'oneway',
       'length', 'geometry', 'ref', 'maxspeed', 'access', 'tunnel'],
      dtype='object')
```

```
edges['highway'].value_counts()
```

```
residential     534
secondary       234
tertiary         94
primary          43
Name: highway, dtype: int64
```

Column	Description	Data type
bridge	Bridge feature	boolean
geometry	Geometry of the feature	Shapely.geometry
highway	Tag for roads, paths (road type)	str (list if multiple)
lanes	Number of lanes	int (or nan)
lenght	The length of a feature in meters	float
maxspeed	maximum legal speed limit	int (list if multiple)
name	Name of the (street) element	str (or nan)
oneway	Street is usable only in one direction	boolean
osmid	Unique node ids of the element	list
u	The first node of networkx edge tuple	int
v	The last node of networkx edge tuple	int

Projecting the data into the metric system:

```
graph_proj = ox.project_graph(graph)

fig, ax = ox.plot_graph(graph_proj)

plt.tight_layout()
```

Checking the Data in the Network

```
1 nodes_proj, edges_proj = ox.graph_to_gdfs(graph_proj, nodes=True, edges=True)
2 print("Coordinate system:", edges_proj.crs)
3 edges_proj.head()
```

```
Coordinate system: +proj=utm +zone=16 +ellps=WGS84 +datum=WGS84 +units=m +no_defs
```

	u	v	key	osmid	lanes	name	highway	oneway	length	geometry	ref	maxspeed	access	tunnel
0	26794572	257168668	0	[435653425, 435653423]	4	North Damen Avenue	secondary	False	202.249	LINESTRING (443735.9155334011 4645720.18171444...	NaN	NaN	NaN	NaN
1	26794572	26794604	0	[435653433, 435653429]	3	West Montrose Avenue	secondary	False	122.434	LINESTRING (443735.9155334011 4645720.18171444...	NaN	NaN	NaN	NaN
2	26794572	305908470	0	[27862070, 435653431]	3	West Montrose Avenue	secondary	False	134.360	LINESTRING (443735.9155334011 4645720.18171444...	NaN	NaN	NaN	NaN
3	26794593	26794631	0	30787531	NaN	West Montrose Avenue	secondary	False	45.205	LINESTRING (444150.1212910318 4645723.95141240...	NaN	NaN	NaN	NaN
4	26794593	257172684	0	34074402	NaN	North Ravenswood Avenue	residential	True	202.901	LINESTRING (444150.1212910318 4645723.95141240...	NaN	NaN	NaN	NaN

Summary Statistics for the Network

```
1 stats = ox.basic_stats(graph_proj)
2 stats
```

```
{'n': 359,
 'm': 905,
 'k_avg': 5.041782729805014,
 'intersection_count': 338,
 'streets_per_node_avg': 3.403899721448468,
 'streets_per_node_counts': {0: 0, 1: 21, 2: 1, 3: 152, 4: 182, 5: 3},
 'streets_per_node_proportion': {0: 0.0,
  1: 0.0584958217270195,
  2: 0.002785515320334262,
  3: 0.4233983286908078,
  4: 0.5069637883008357,
  5: 0.008356545961002786},
 'edge_length_total': 115325.19599999982,
 'edge_length_avg': 127.43115580110478,
 'street_length_total': 77176.12899999999,
 'street_length_avg': 132.83326850258175,
 'street_segments_count': 581,
 'node_density_km': None,
 'intersection_density_km': None,
 'edge_density_km': None,
 'street_density_km': None,
 'circuity_avg': 1.2996868885953827e-05,
 'self_loop_proportion': 0.0,
 'clean_intersection_count': None,
 'clean_intersection_density_km': None}
```

Edge/node level stats

Summary Statistics for the Network

```
area = edges_proj.unary_union.convex_hull.area
stats = ox.basic_stats(graph_proj, area=area)
extended_stats = ox.extended_stats(graph_proj, ecc=True, bc=True, cc=True)
for key, value in extended_stats.items():
    stats[key] = value


pd.Series(stats)
```

```
n
m
k_avg
intersection_count
streets_per_node_avg
streets_per_node_counts
streets_per_node_proportion
edge_length_total
edge_length_avg
street_length_total
street_length_avg
street_segments_count
node_density_km
intersection_density_km
edge_density_km
street_density_km
circuity_avg
self_loop_proportion
clean_intersection_count
clean_intersection_density_km
avg_neighbor_degree
avg_neighbor_degree_avg
avg_weighted_neighbor_degree
avg_weighted_neighbor_degree_avg
degree_centrality
degree_centrality_avg
clustering_coefficient
clustering_coefficient_avg
clustering_coefficient_weighted
clustering_coefficient_weighted_avg
pagerank
pagerank_max_node
pagerank_max
pagerank_min_node
pagerank_min
eccentricity
diameter
radius
center
periphery
closeness_centrality
closeness_centrality_avg
betweenness_centrality
betweenness_centrality_avg
dtype: object
```

We can for example see that the average node density in our network is **153 nodes/km** and that the total edge length of our network is **19787.7 meters**. We can also see that the degree centrality of our network is on **average 0.0334533**. Degree is a simple centrality measure that counts how many neighbors a node has. Another interesting measure is the PageRank that measures the importance of specific node in the graph. Here, we can see that the most important node in our graph seem to a node with **osmid 25416262**.

Shortest Path Analysis

Let us set the starting point to be the centroid of the graph

We first need to extract the bounding boxes for all the features to create the union of them. This will allow us to find the overall bounding box.

```
1 edges_proj.bounds.head()
```

	minx	miny	maxx	maxy
0	443732.899696	4.645720e+06	443735.915533	4.645922e+06
1	443735.915533	4.645720e+06	443858.623886	4.645721e+06
2	443601.260250	4.645719e+06	443735.915533	4.645720e+06
3	444104.818240	4.645723e+06	444150.121291	4.645724e+06
4	444146.817006	4.645724e+06	444150.121291	4.645927e+06

```
1 #Creating a bounding box out of these
2 from shapely.geometry import box
3 bbox = box(*edges_proj.unary_union.bounds)
4 print(bbox)
```

```
POLYGON ((444150.1212910318 4645710.388891731, 444150.1212
88.25752804 4645710.388891731, 444150.1212910318 4645710.3
```

We can then define the origin point, to be the centroid of the bounding box

```
1 orig_point = bbox.centroid
2 print(orig_point)
```

```
POINT (442869.1894095359 4647337.779714304)
```

Shortest Path Analysis

We can define the end point

Let us select the **easternmost point** to be the terminal point

```
#Finding the easternmost node
nodes_proj['x'] = nodes_proj.x.astype(float)
maxx = nodes_proj['x'].max()
```

```
target_loc = nodes_proj.loc[nodes_proj['x']==maxx, :]
print(target_loc)
```

```
                      y              x     osmid          highway        lon  \
26794593   4.645724e+06   444150.121291  26794593  traffic_signals -87.673964

                lat                                    geometry
26794593  41.961524  POINT (444150.1212910318 4645723.951412406)
```

```
target_point = target_loc.geometry.values[0]
print(target_point)
```

```
POINT (444150.1212910318 4645723.951412406)
```

```
orig_xy = (orig_point.y, orig_point.x)
target_xy = (target_point.y, target_point.x)
orig_node = ox.get_nearest_node(graph_proj, orig_xy, method='euclidean')
target_node = ox.get_nearest_node(graph_proj, target_xy, method='euclidean')
o_closest = nodes_proj.loc[orig_node]
t_closest = nodes_proj.loc[target_node]

print(orig_node)
print(target_node)
```

```
158733110
26794593
```

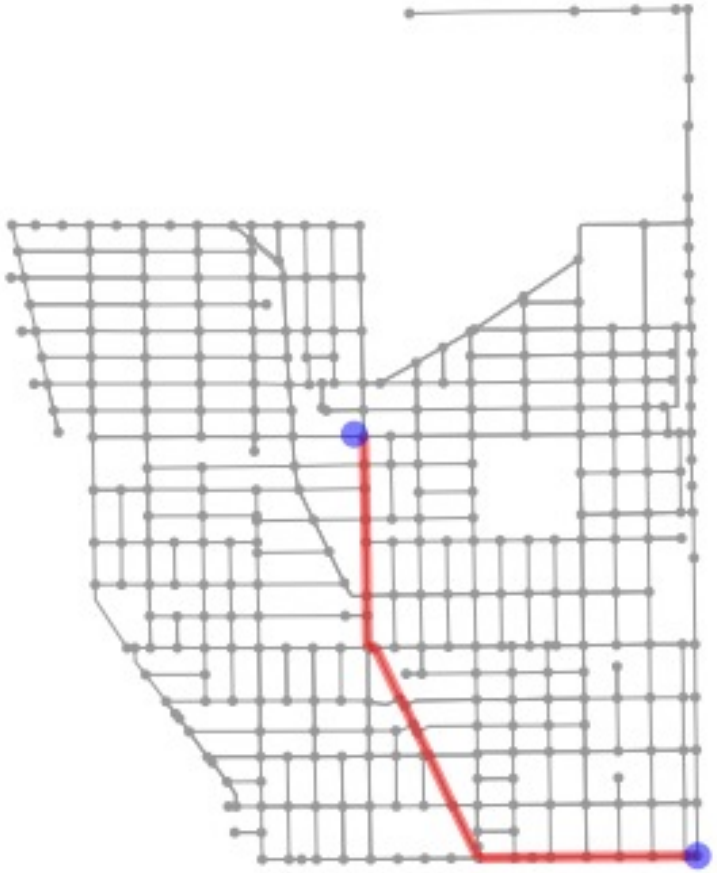
Shortest Path Analysis

```
#Making a GeoDataframe
od_nodes = gpd.GeoDataFrame([o_closest, t_closest], geometry='geometry', crs=nodes_proj.crs)
```

```
route = nx.shortest_path(G=graph_                                    ight='length')
print(route)
```

```
[158733110, 305896090, 305896089, 293                                6645, 305896646, 260227530, 262077533, 1
299592858, 263907187, 257110952, 1530                                08470, 26794572, 26794604, 26794603, 267
94631, 26794593]
```

```
fig, ax = ox.plot_graph_route(gra                                    on_point=target_xy)
plt.tight_layout()
```



```
<Figure size 432x288 with 0 Axes>
```

Shortest Path Analysis

```
#Saving the shortest path
#Find nodes in the shortest path
route_nodes = nodes_proj.loc[route]
print(route_nodes)
```

```
                    y             x       osmid          highway  \
158733110   4.647330e+06  442904.507856  158733110  traffic_signals
305896090   4.647222e+06  442906.186993  305896090              NaN
305896089   4.647130e+06  442908.493840  305896089              NaN
293816175   4.647012e+06  442911.034302  293816175  traffic_signals
293816188   4.646925e+06  442912.717786  293816188              NaN
293816189   4.646720e+06  442915.226642  293816189  traffic signals
344810034   4.646644e+06  442916.487248         34
305896645   4.646521e+06  442919.343454         30
305896646   4.646521e+06  442947.542586         30
260227530   4.646322e+06  443035.991597         26
262077533   4.646299e+06  443056.392129         26
1299592858  4.646226e+06  443089.945799        129
263907187   4.646198e+06  443104.144799         26
257110952   4.646109e+06  443147.043546         25
153011804   4.645917e+06  443235.428453         15
2758167240  4.645716e+06  443332.228308        275
1299592863  4.645717e+06  443467.084179        129
305908466   4.645718e+06  443534.060355         30
```

```
from shapely.geometry import LineString, Point
route_line = LineString(list(route_nodes.geometry.values))
print(route_line)
```

```
LINESTRING (442904.5078559591 4647330.037964049, 442906.1869932694 464
442911.0343016215 4647012.4546554, 442912.7177861492 4646925.066596619
961 4646643.692415647, 442919.343453755 4646520.741162321, 442947.5425
70581351, 443056.3921291734 4646298.556274774, 443089.9457985762 46462
3147.04354564 4646108.568601502, 443235.428452735 4645917.445573983, 4
4645717.287020091, 443534.0603553199 4645718.053220749, 443601.2602498
4444, 443858.6238859641 4645720.942626915, 443982.4270853776 4645721.8
0.1212910318 4645723.951412406)
```

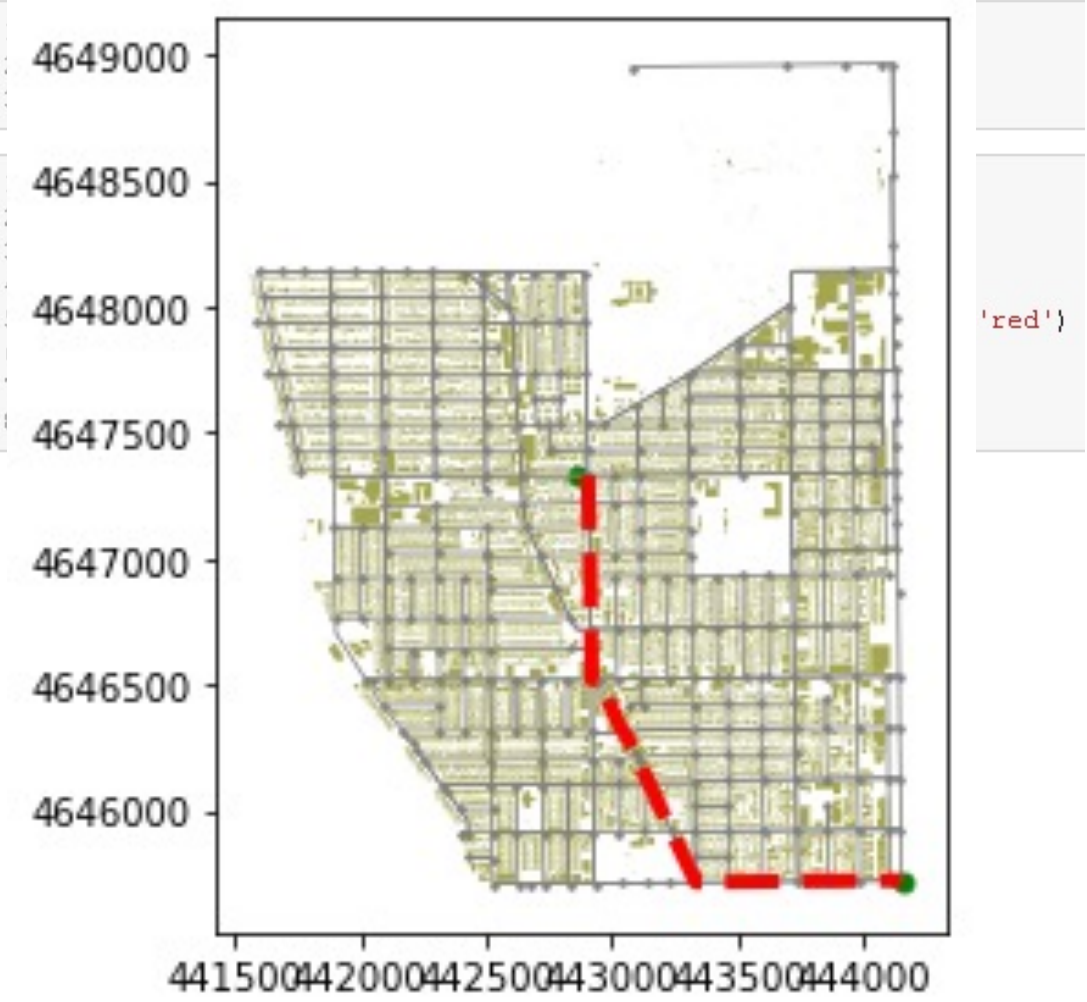
Shortest Path Analysis

```
#Creating a GeoDataframe with important information
route_geom = gpd.GeoDataFrame(crs=edges_proj.crs)
route_geom['geometry'] = None
route_geom['osmids'] = None
```

```
#Adding Geometry
route_geom.loc[0, 'geometry'] = route_line
route_geom.loc[0, 'osmids'] = str(list(route_nodes['osmid'].values))
route_geom['length_m'] = route_geom.length
```

```
od_points = gpd.GeoDataFrame(crs=edges_proj.crs)
od_points['geometry'] = None
od_points['type'] = None
od_points.loc[0, ['geometry', 'type']] = orig_point, 'Origin'
od_points.loc[1, ['geometry', 'type']] = target_point, 'Target'
od_points.head()
```

	geometry	type
0	POINT (442869.1894095359 4647337.779714304)	Origin
1	POINT (444150.1212910318 4645723.951412406)	Target

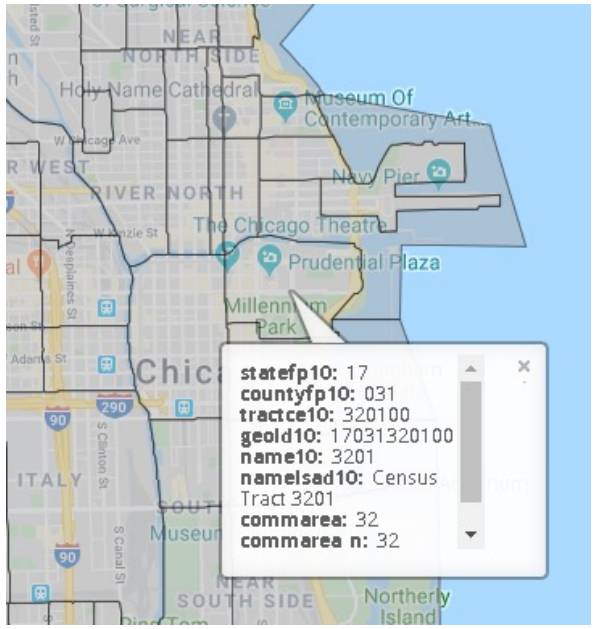


ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

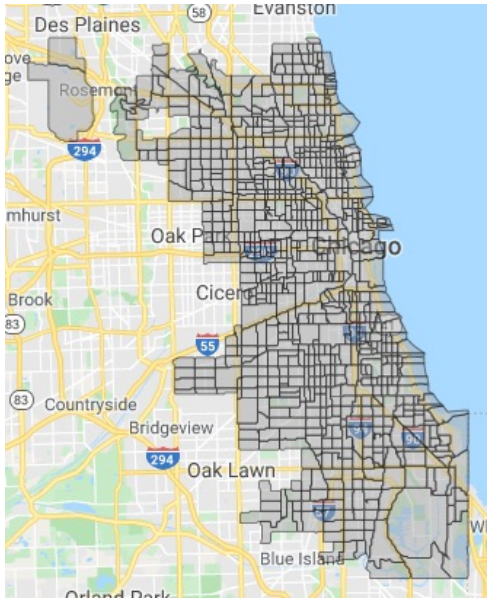
Chicago Traffic Analysis

OSMnx and Taxi + Train Data

Extracting the Location



Chicago Census Tracts

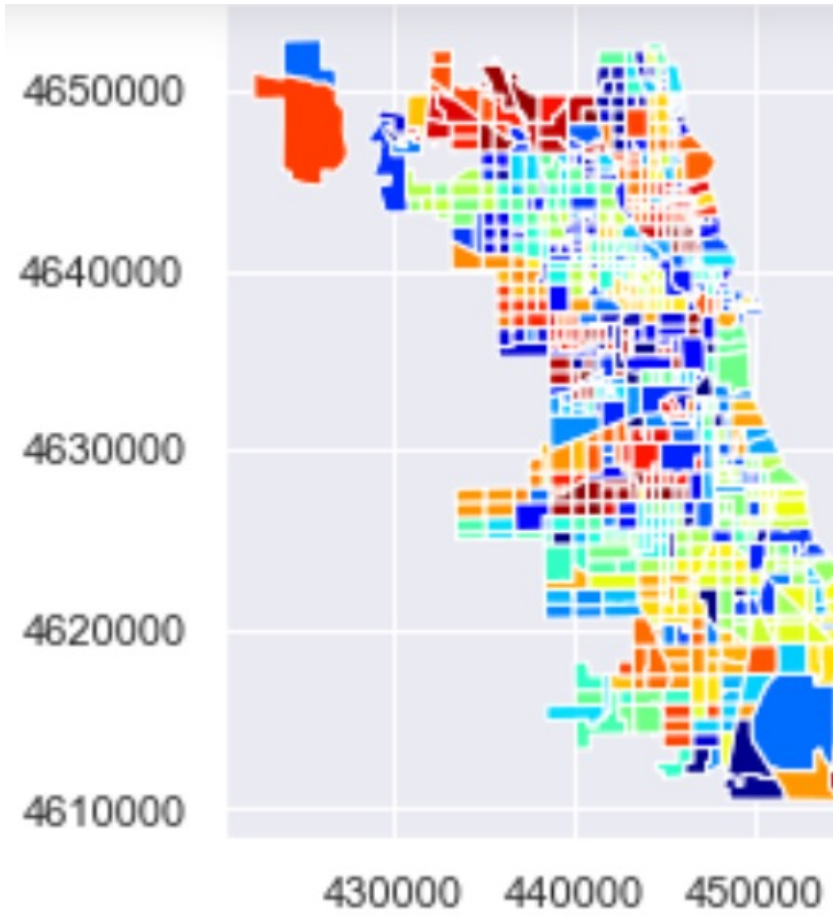


Focus Areas

- 13 census tracts
- Downtown Chicago area

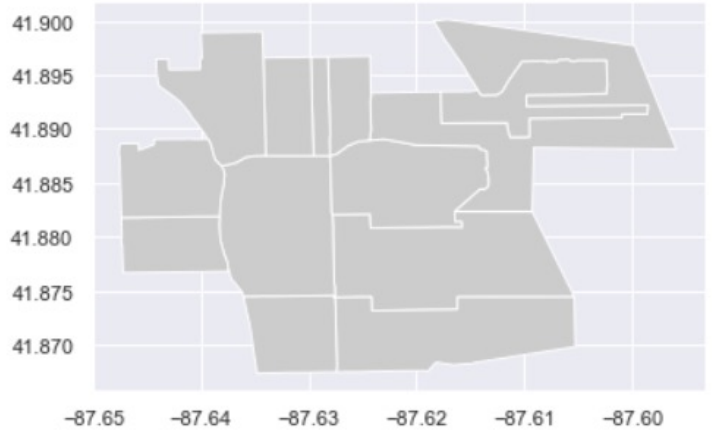
- The Chicago census tract data was extracted as JSON from the city of Chicago's data portal, http://data.cityofchicago.org
- In order to filter out the focus area, the tract names of the areas of interest were extracted and only those areas were considered.

Extracting the Polygons



The polygons can be extracted for each individual census tract and combined into a multipolygon through using the cascaded union function.

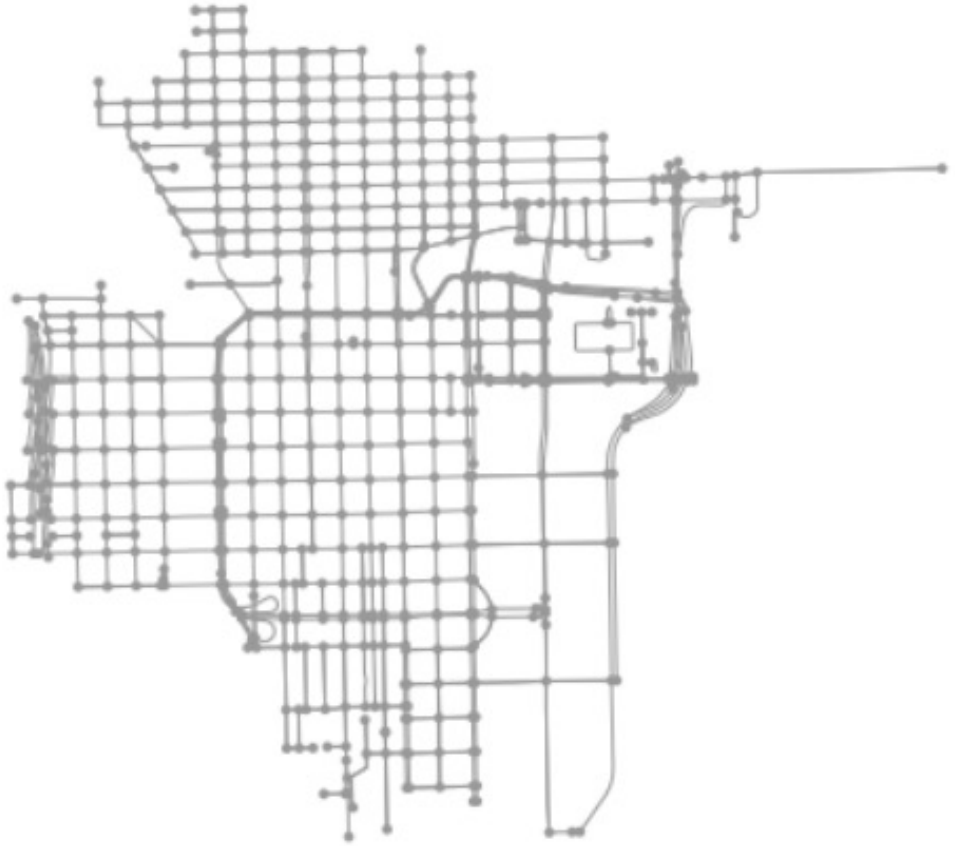
```
1 polygons = [places['geometry'][i] for i in range(15)]
2 from shapely.ops import cascaded_union
3 u = cascaded_union(polygons)
4 print(u)
```



Extracting the Street Data

The street data for the relevant census tracts can be extracted in one of two ways:

Bounding Box Method



Polygon Method

Extracting the Street Data

The street data contains information on the nodes and streets in the network:

	u	v	key	osmid	highway	name	oneway	length	geometry	lanes	maxspeed	ref	bridge	tunnel	junction	ac
0	261885955	5970344149	0	23701604	residential	South Sangamon Street	False	7.330	LINESTRING (-87.65086669999999 41.8817542, -87...	NaN	NaN	NaN	NaN	NaN	NaN	
1	261885955	288070377	0	24102347	residential	North Sangamon Street	False	148.948	LINESTRING (-87.65086669999999 41.8817542, -87...	NaN	NaN	NaN	NaN	NaN	NaN	
2	261885955	258955476	0	[633300070, 633300071]	secondary	West Madison Street	True	96.354	LINESTRING (-87.65086669999999 41.8817542, -87...	[1, 2]	NaN	NaN	NaN	NaN	NaN	
3	258957316	258957340	0	23810302	tertiary	West Lake Street	True	112.859	LINESTRING (-87.6339309 41.8857296, -87.632567...	NaN	NaN	NaN	NaN	NaN	NaN	
4	258957316	27477592	0	48785101	secondary	North Wells Street	True	140.435	LINESTRING (-87.6339309 41.8857296, -87.633913...	NaN	NaN	NaN	NaN	NaN	NaN	

Incorporating Taxi Data

- Which taxi provided the trip
- What times the trip started and ended
- Length of the trip in both time and distance
- Starting and ending Community Area — plus Census Tract for many trips
- Fare amount and other components of the trip cost
- Type of payment — such as cash or credit card.
- Taxi company

Trip ID	Taxi ID	Trip Start Timestamp	Trip End Timestamp	Trip Seconds	Trip Miles	Pickup Census Tract	Dropoff Censu
a24d365565d8...	f4ae9b2c8c8f5d...	01/01/2017 12:00:00 AM	01/01/2017 12:00:00 AM	4	0	17031081500	17031081500
25383fa1ec8fb3...	9dd0d41dfdbc1...	01/01/2017 12:00:00 AM	01/01/2017 12:00:00 AM	1	0	17031081800	17031081800
50e08ddc853ca...	368f191bda532...	01/01/2017 12:00:00 AM	01/01/2017 12:00:00 AM	185	0.1	17031081700	17031081600
ed600efc3cf6ed...	75cebaf9a78e2...	01/01/2017 12:00:00 AM	01/01/2017 12:00:00 AM	23	0	17031081600	17031081600
341c9a6f83e9a...	3e197e5471261...	01/01/2017 12:00:00 AM	01/01/2017 12:00:00 AM	252	0.6	17031839100	17031081800
f15eb2c50f4767...	a0df6e84237fd1...	01/01/2017 12:00:00 AM	01/01/2017 12:00:00 AM	214	0.44	17031280100	17031281900

Edge Betweenness Centrality

- We can use the extracted graph to run some analysis in terms of edge and node betweenness.
- Edge (Node) Betweenness: Centrality metric that is based on the number of shortest paths that go through a particular edge (node). The algorithm used is based on one proposed by **Ulkrik Brandes**.

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

betweenness centrality

(Freeman, 1977; Anthonisse, 1971)

"A faster algorithm for betweenness centrality" Ulkrik Brandes 2001

Weighted Edge Betweenness Centrality

- In order to consider the geographical importance of an edge in addition to real operational data, we utilize the taxi data to find the shortest route from each trip start and end location.
- We can then simulate the paths that these vehicles took.
- The weights are then used as inputs to the edge betweenness function

Results

The results of the case study highlight the important routes in the network based on the Weighted Edge Betweenness Centrality metric.

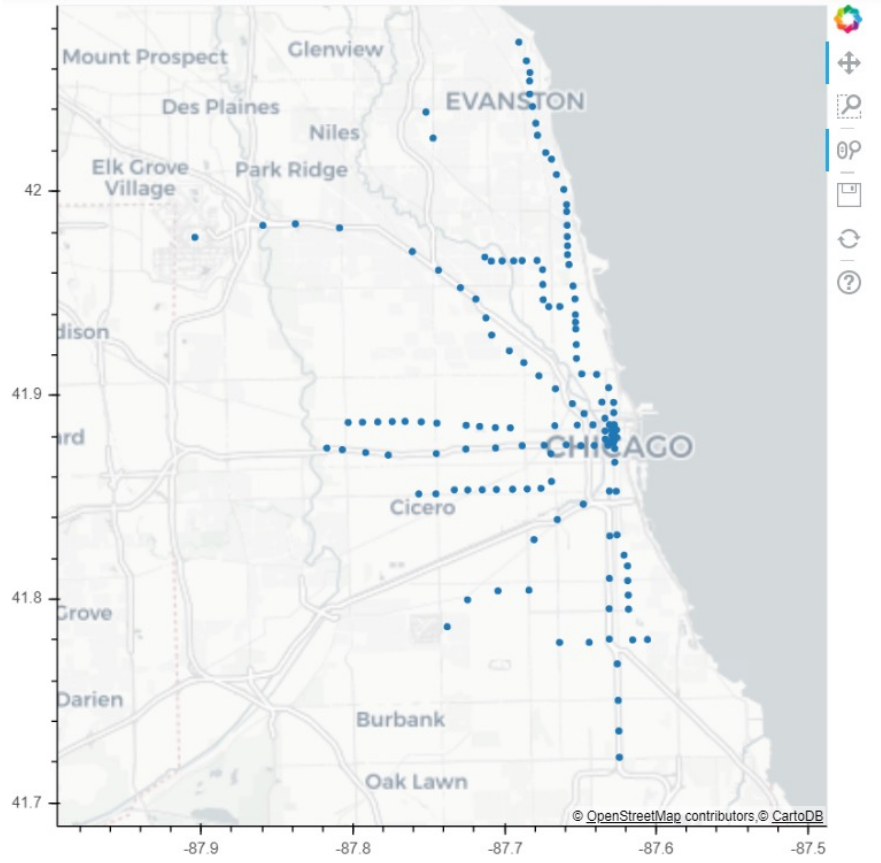
Clearly, the most important edges (roads) in the network are the edges that are central to the network but also contain main roads/highways that have points of interest along them.

Chicago Train Data

Chicago Train Data was also extracted from the Chicago city data portal. The data was preprocessed and only the trips that were taken in **2017** were recorded.

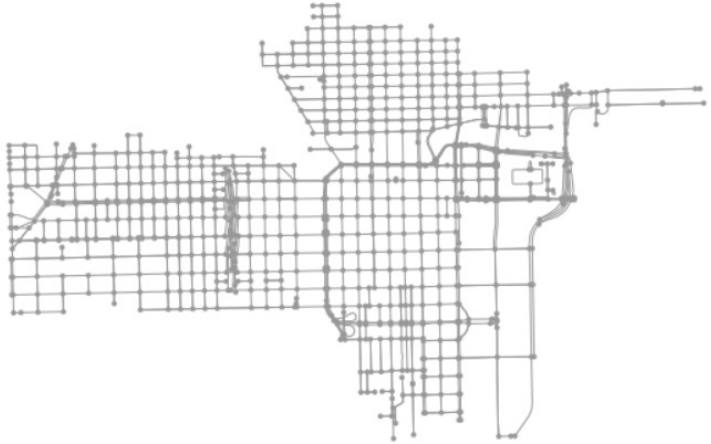
The **Bokeh** package was used to create the interactive plots

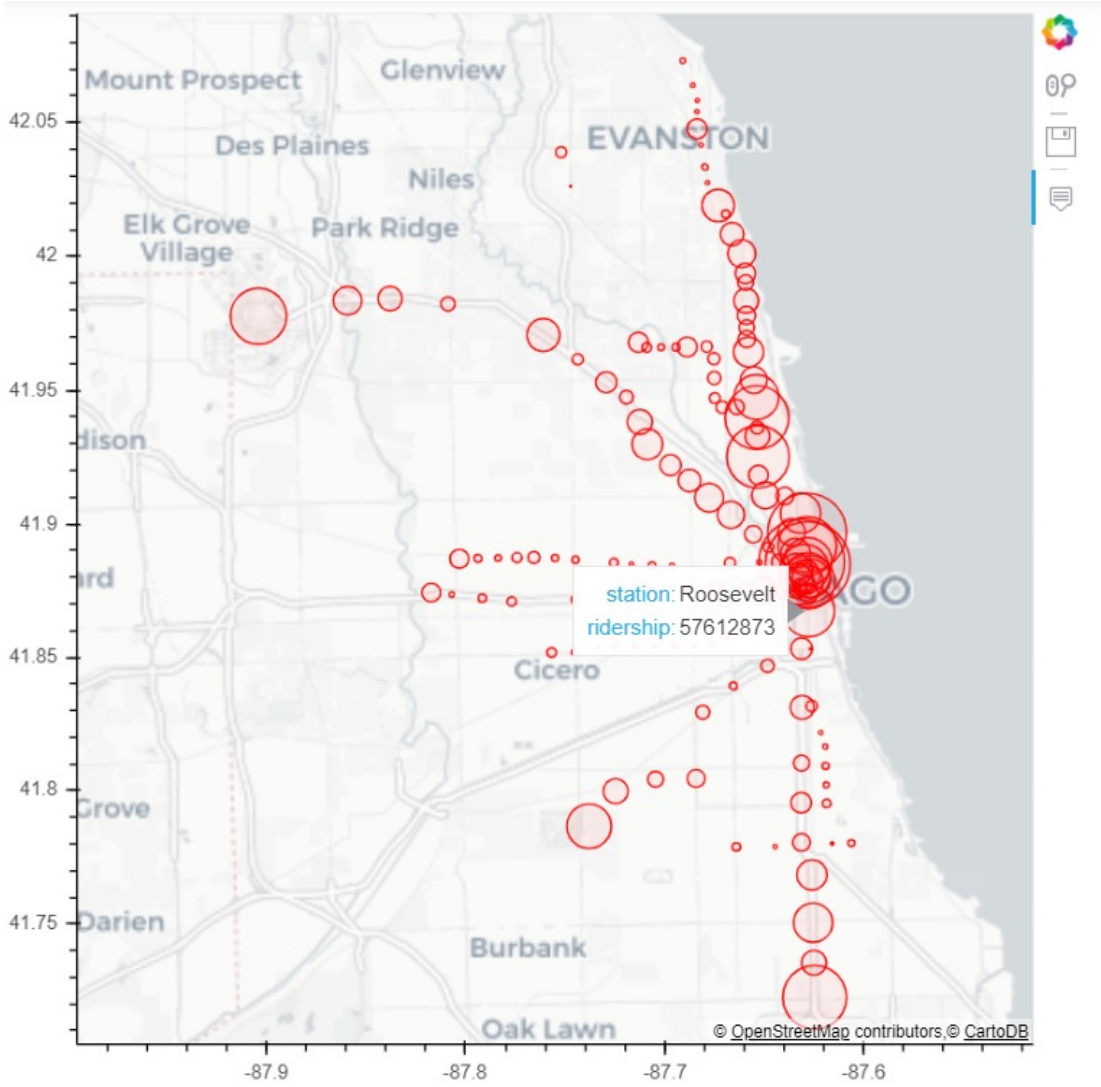
	station_id	stationame	month_beginning	avg_weekday_rides	avg_saturday_rides	avg_sunday-holiday_rides	monthtotal
0	40900	Howard	01/01/2001	6233.9	3814.5	2408.6	164447
1	41190	Jarvis	01/01/2001	1489.1	1054.0	718.0	40567
2	40100	Morse	01/01/2001	4412.5	3064.5	2087.8	119772
3	41300	Loyola	01/01/2001	4664.5	3156.0	1952.8	125008
4	40760	Granville	01/01/2001	3109.8	2126.0	1453.8	84189



Chicago Train Data

- The most popular stations are highlighted.
- Clearly, the most popular stations are the stations that are downtown and the stations on the outskirts





ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Mapping Natural Disasters

GeoPandas

Mapping Natural Disasters

Importing the map of the United States

```
country = geopandas.read_file("gz_2010_us_040_00_5m.json")
country.head()
```

	GEO_ID	STATE	NAME	LSAD	CENSUSAREA	geometry
0	0400000US01	01	Alabama		50645.326	(POLYGON ((-88.124658 30.28364, -88.0868119999...
1	0400000US02	02	Alaska		570640.950	(POLYGON ((-166.10574 53.988606, -166.075283 5...
2	0400000US04	04	Arizona		113594.084	POLYGON ((-112.538593 37.000674, -112.534545 3...
3	0400000US05	05	Arkansas		52035.477	POLYGON ((-94.042964 33.019219, -94.043036 33....
4	0400000US06	06	California		155779.220	(POLYGON ((-122.421439 37.869969, -122.421341 ...

```
type(country)
```

```
geopandas.geodataframe.GeoDataFrame
```

```
type(country.geometry)
```

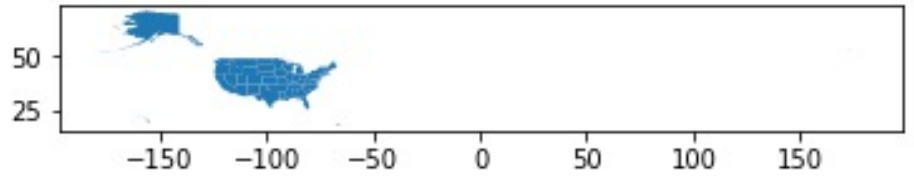
```
geopandas.geoseries.GeoSeries
```

```
type(country.geometry[0])
```

```
shapely.geometry.multipolygon.MultiPolygon
```

```
country.plot()
```

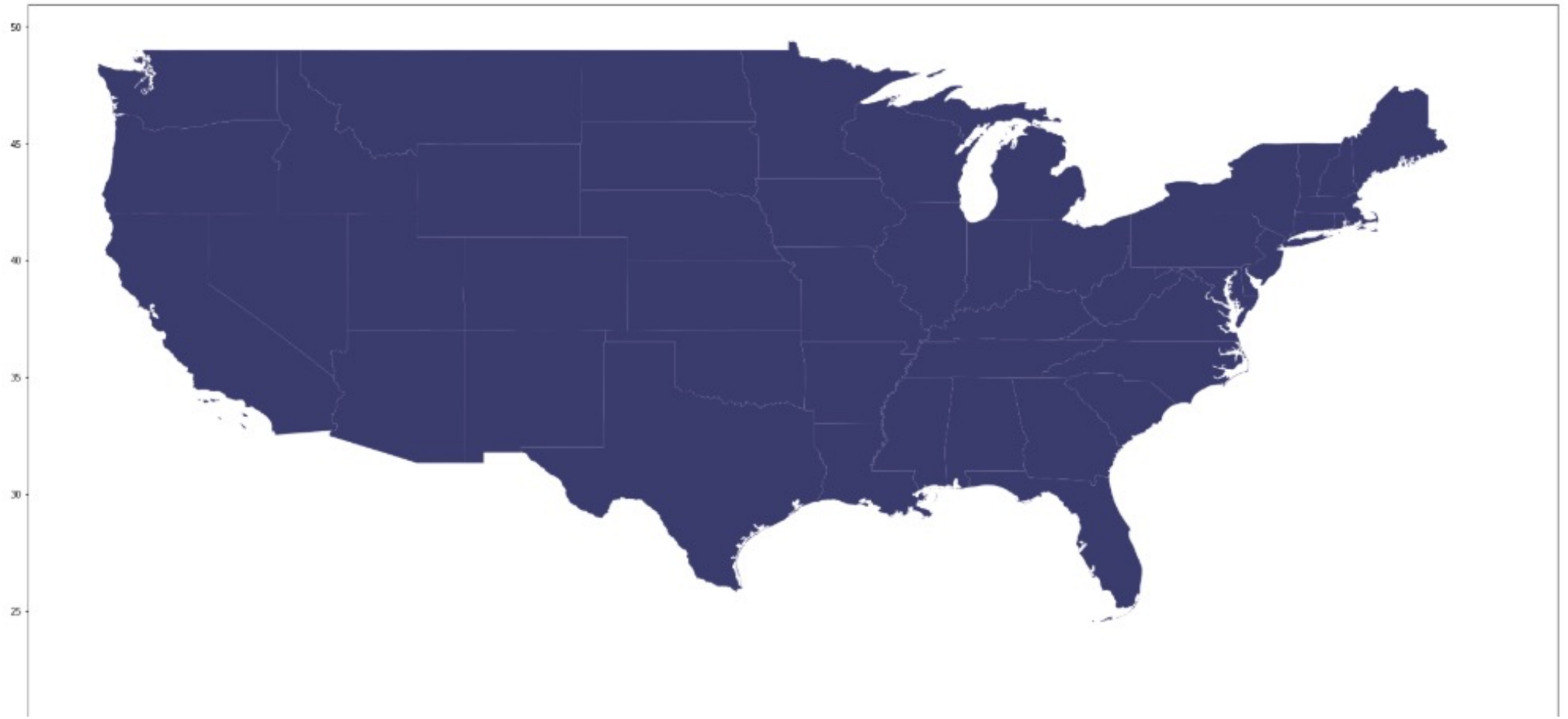
```
<matplotlib.axes._subplots.AxesSubplot at 0x189dbc4c9e8>
```



Mapping Natural Disasters

US State Boundaries

```
country[country['NAME'].isin(['Alaska','Hawaii']) == False].plot(figsize=(30,20), color='#3B3C6E');
```



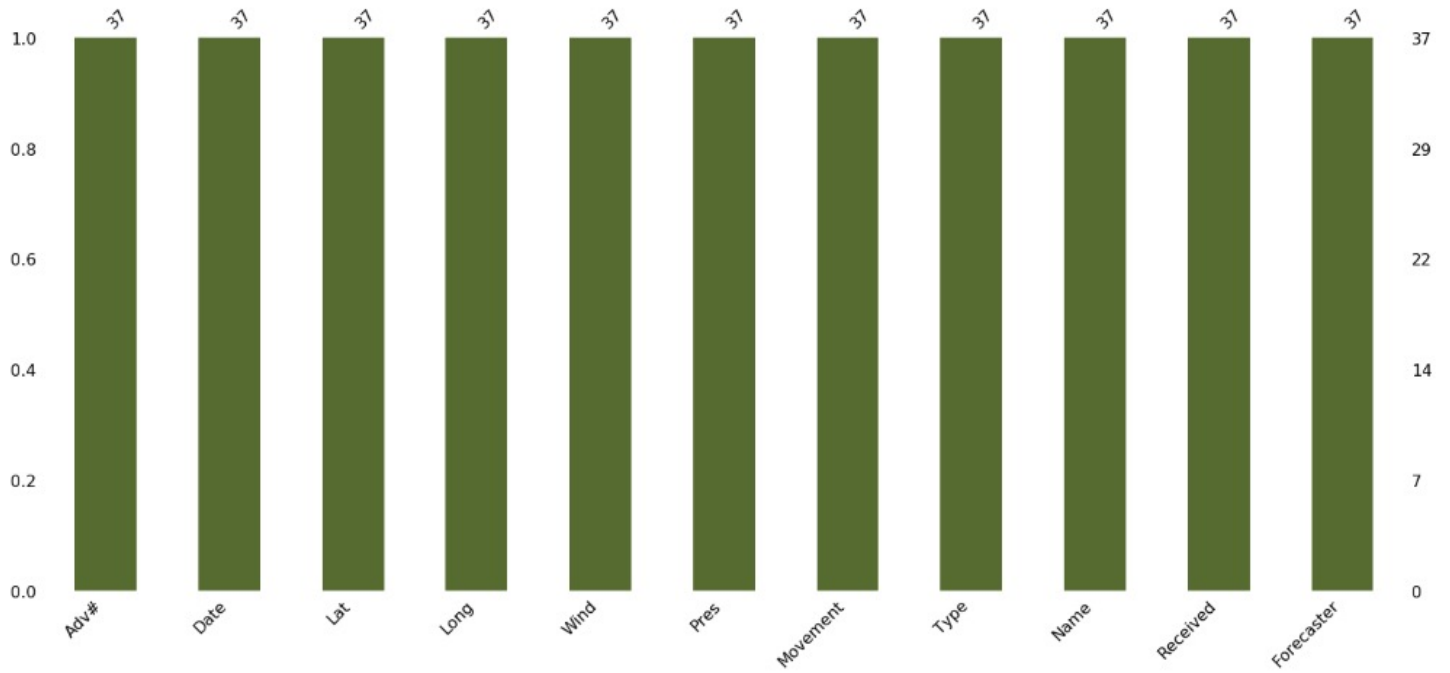
Hurricane Alberto Data

```
1 alberto = pd.read_csv('Alberto.csv')
2 alberto.head()
```

	Adv#	Date	Lat	Long	Wind	Pres	Movement	Type	Name	Received	Forecaster
0	1	5/25/2019 11:00	19.7	86.8	40	1005	Nne at 6 MPH (20 deg)	subTS	Alberto	5/25/2019 10:44	Stewart
1	1A	5/25/2019 14:00	19.5	86.5	40	1005	Stationary	subTS	Alberto	5/25/2019 13:36	Stewart
2	2	5/25/2019 17:00	19.4	86.3	40	1005	E at 2 MPH (90 deg)	subTS	Alberto	5/25/2019 16:50	Stewart
3	2A	5/25/2019 20:00	19.2	86.0	40	1006	E at 2 MPH (90 deg)	subTS	Alberto	5/25/2019 19:53	Beven
4	3	5/25/2019 23:00	19.4	85.7	40	1006	E at 5 MPH (90 deg)	subTS	Alberto	5/25/2019 22:44	Beven

```
1 alberto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 37 entries, 0 to 36
Data columns (total 11 columns):
Adv#          37 non-null object
Date          37 non-null object
Lat           37 non-null float64
Long          37 non-null float64
Wind          37 non-null int64
Pres          37 non-null int64
Movement      37 non-null object
Type          37 non-null object
Name          37 non-null object
Received      37 non-null object
Forecaster    37 non-null object
dtypes: float64(2), int64(2), object(7)
memory usage: 3.3+ KB
```



Preprocessed Data

```
1 alberto.describe()
```

	Lat	Long	Wind	Pres
count	37.000000	37.000000	37.000000	37.000000
mean	27.902703	85.727027	43.243243	998.081081
std	6.932551	0.970008	10.879148	5.182119
min	19.100000	83.900000	30.000000	990.000000
25%	22.800000	85.100000	40.000000	994.000000
50%	28.000000	85.800000	40.000000	998.000000
75%	30.900000	86.200000	50.000000	1002.000000
max	44.900000	87.900000	65.000000	1006.000000

We also have to convert the Lat and Log values from the tables into coordinates, we do this by removing the N,E,W,S values and replacing by respective negative values.

Dropping the columns that are not important

```
1 alberto = alberto.drop(['Adv#', 'Forecaster', 'Received'], axis=1)
2 alberto.head()
```

	Date	Lat	Long	Wind	Pres	Movement	Type	Name
0	5/25/2019 11:00	19.7	86.8	40	1005	Nne at 6 MPH (20 deg)	subTS	Alberto
1	5/25/2019 14:00	19.5	86.5	40	1005	Stationary	subTS	Alberto
2	5/25/2019 17:00	19.4	86.3	40	1005	E at 2 MPH (90 deg)	subTS	Alberto
3	5/25/2019 20:00	19.2	86.0	40	1006	E at 2 MPH (90 deg)	subTS	Alberto
4	5/25/2019 23:00	19.4	85.7	40	1006	E at 5 MPH (90 deg)	subTS	Alberto

```
1 alberto['Long'] = 0 - alberto['Long']
2 alberto.head()
```

	Date	Lat	Long	Wind	Pres	Movement	Type	Name
0	5/25/2019 11:00	19.7	-86.8	40	1005	Nne at 6 MPH (20 deg)	subTS	Alberto
1	5/25/2019 14:00	19.5	-86.5	40	1005	Stationary	subTS	Alberto
2	5/25/2019 17:00	19.4	-86.3	40	1005	E at 2 MPH (90 deg)	subTS	Alberto
3	5/25/2019 20:00	19.2	-86.0	40	1006	E at 2 MPH (90 deg)	subTS	Alberto
4	5/25/2019 23:00	19.4	-85.7	40	1006	E at 5 MPH (90 deg)	subTS	Alberto

Preprocessed Data

Creating coordinates from the updated latitude and longitude values

```
alberto['coordinates'] = alberto[['Long', 'Lat']].values.tolist()
alberto.head()
```

	Date	Lat	Long	Wind	Pres	Movement	Type	Name	coordinates
0	5/25/2019 11:00	19.7	-86.8	40	1005	Nne at 6 MPH (20 deg)	subTS	Alberto	[-86.8, 19.7]
1	5/25/2019 14:00	19.5	-86.5	40	1005	Stationary	subTS	Alberto	[-86.5, 19.5]
2	5/25/2019 17:00	19.4	-86.3	40	1005	E at 2 MPH (90 deg)	subTS	Alberto	[-86.3, 19.4]
3	5/25/2019 20:00	19.2	-86.0	40	1006	E at 2 MPH (90 deg)	subTS	Alberto	[-86.0, 19.2]
4	5/25/2019 23:00	19.4	-85.7	40	1006	E at 5 MPH (90 deg)	subTS	Alberto	[-85.7, 19.4]

Converting them to POINT types

```
alberto['coordinates'] = alberto['coordinates'].apply(Point)
alberto.head()
```

	Date	Lat	Long	Wind	Pres	Movement	Type	Name	coordinates
0	5/25/2019 11:00	19.7	-86.8	40	1005	Nne at 6 MPH (20 deg)	subTS	Alberto	POINT (-86.8 19.7)
1	5/25/2019 14:00	19.5	-86.5	40	1005	Stationary	subTS	Alberto	POINT (-86.5 19.5)
2	5/25/2019 17:00	19.4	-86.3	40	1005	E at 2 MPH (90 deg)	subTS	Alberto	POINT (-86.3 19.4)
3	5/25/2019 20:00	19.2	-86.0	40	1006	E at 2 MPH (90 deg)	subTS	Alberto	POINT (-86 19.2)
4	5/25/2019 23:00	19.4	-85.7	40	1006	E at 5 MPH (90 deg)	subTS	Alberto	POINT (-85.7 19.4)

Transforming the Data into GeoPandas

```
1 alberto = geopandas.GeoDataFrame(alberto, geometry='coordinates')
2 alberto.head()
```

	Date	Lat	Long	Wind	Pres	Movement	Type	Name	coordinates
0	5/25/2019 11:00	19.7	-86.8	40	1005	Nne at 6 MPH (20 deg)	subTS	Alberto	POINT (-86.8 19.7)
1	5/25/2019 14:00	19.5	-86.5	40	1005	Stationary	subTS	Alberto	POINT (-86.5 19.5)
2	5/25/2019 17:00	19.4	-86.3	40	1005	E at 2 MPH (90 deg)	subTS	Alberto	POINT (-86.3 19.4)
3	5/25/2019 20:00	19.2	-86.0	40	1006	E at 2 MPH (90 deg)	subTS	Alberto	POINT (-86 19.2)
4	5/25/2019 23:00	19.4	-85.7	40	1006	E at 5 MPH (90 deg)	subTS	Alberto	POINT (-85.7 19.4)

```
1 alberto[alberto['Name']=='Six']
```

Date	Lat	Long	Wind	Pres	Movement	Type	Name	coordinates

```
1 alberto.groupby('Name').Type.count()
```

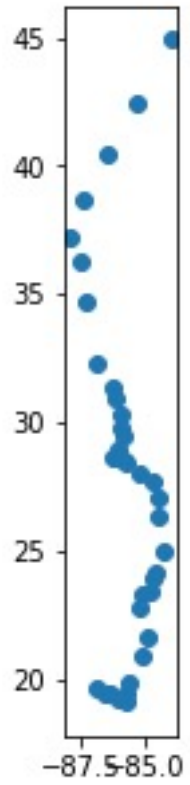
```
Name
ALBERTO      5
Alberto     29
LBERTO       3
Name: Type, dtype: int64
```

```
1 print("Mean wind speed of Hurricane alberto is {} mph and it can go up to {} mph maximum".format(round(alberto.Wind.mean(
2                                                                                                    alberto.Wind.max()))
```

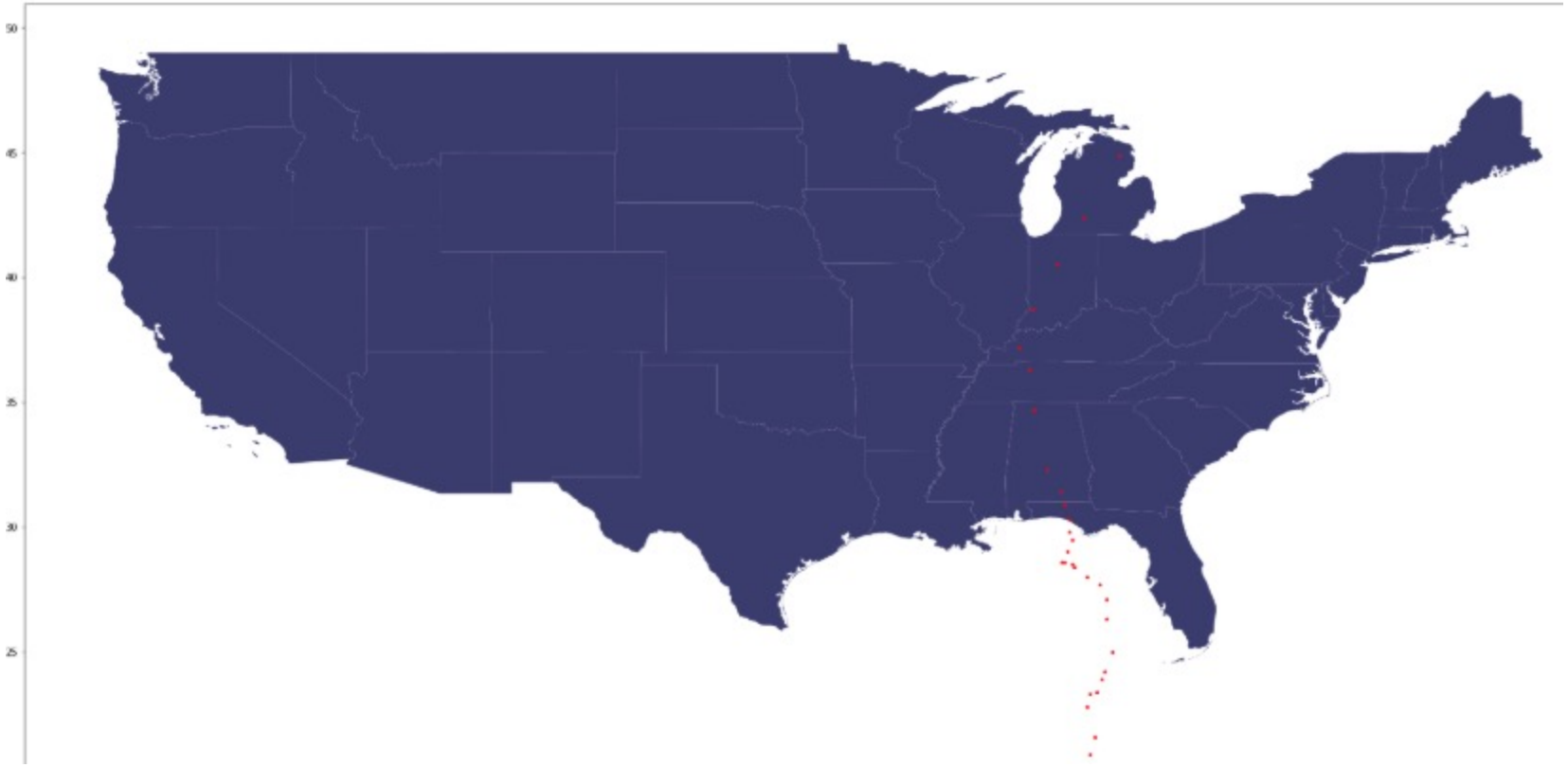
```
Mean wind speed of Hurricane alberto is 43.2432 mph and it can go up to 65 mph maximum
```

Plotting Hurricane Alberto Data

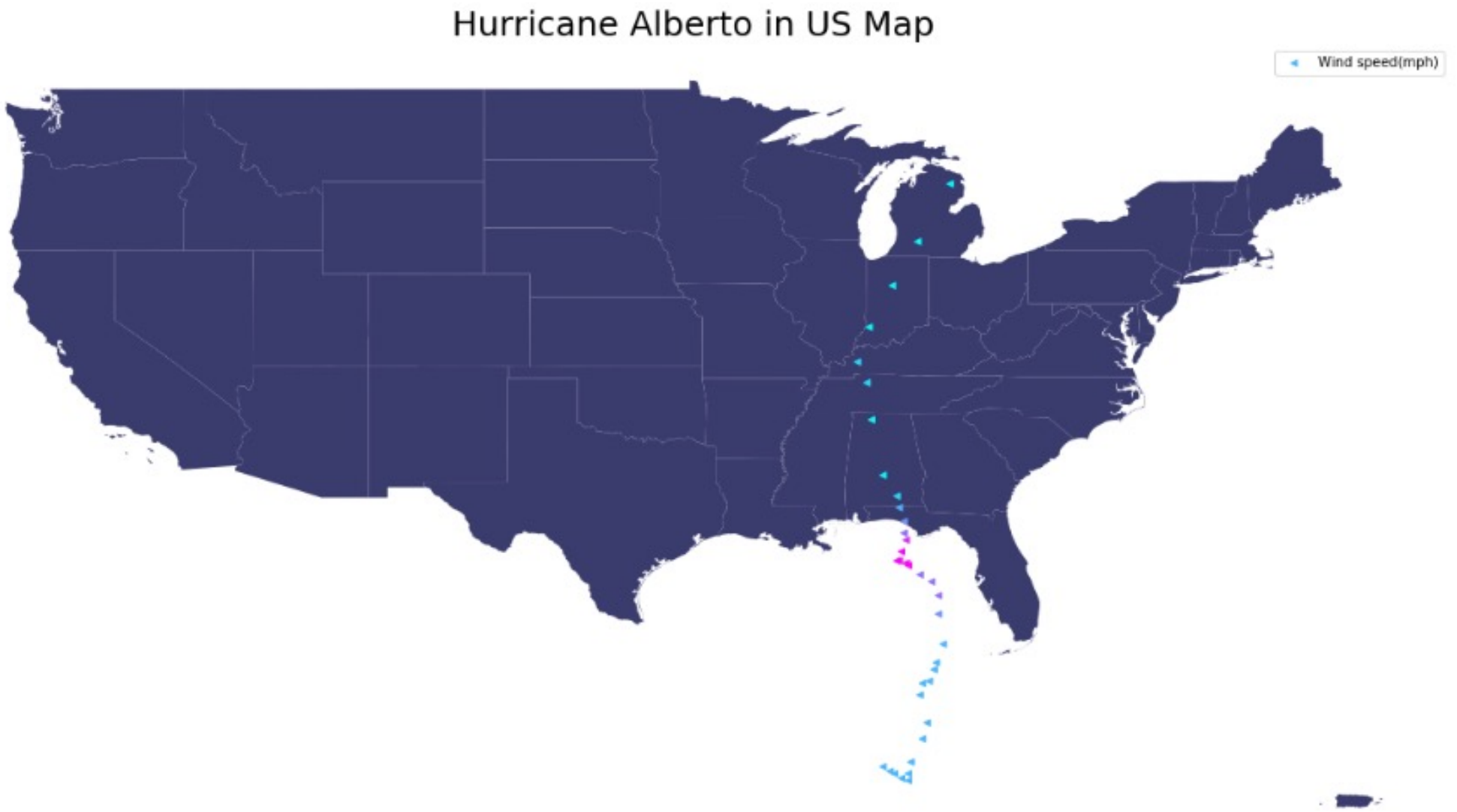
```
1  alberto.plot(figsize=(10,5));
```




```
1  # Plotting to see the hurricane overlay the US map:
2  fig, ax = plt.subplots(1, figsize=(30,20))
3  base = country[country['NAME'].isin(['Alaska','Hawaii']) == False].plot(ax=ax, color='#3B3C6E')
4
5  # plotting the hurricane position on top with red color to stand out:
6  alberto.plot(ax=base, color='red', marker="*", markersize=10);
```



Plotting Hurricane Data with Windspeed



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Interactive Geospatial Display

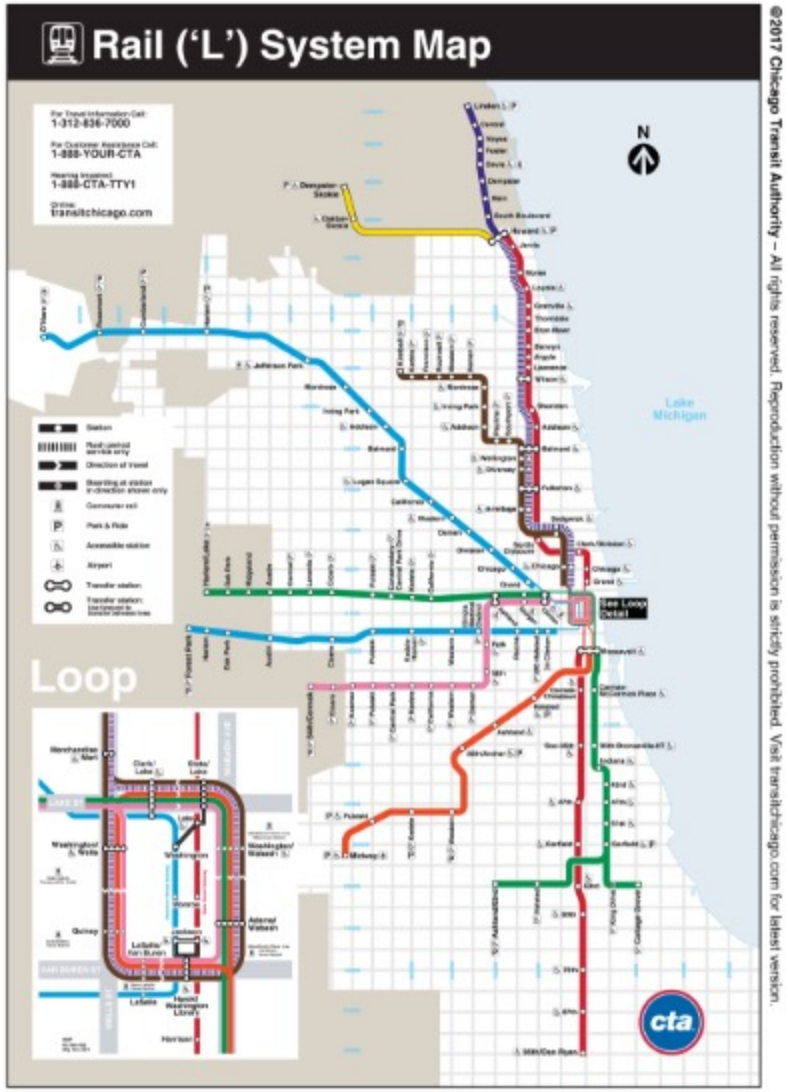
Bokeh

Chicago L-Train Data

Mainly used Pandas and Bokeh

- Panda: Used for storing and manipulating the data in a dataframe

- Bokeh: Visualization library that provides tools to overlay maps and features as well as create interactive media



Importing the Data

```
1 import pandas as pd
2 L_Rides = pd.read_csv('CTA_-_Ridership_-__L__Station_Entries_-_Monthly_Day-Type_Averages_
3 L_Rides.head()
```

	station_id	stationame	month_beginning	avg_weekday_rides	avg_saturday_rides	avg_sunday-holiday_rides	monthtotal
0	40900	Howard	01/01/2001	6233.9	3814.5	2408.6	164447
1	41190	Jarvis	01/01/2001	1489.1	1054.0	718.0	40567
2	40100	Morse	01/01/2001	4412.5	3064.5	2087.8	119772
3	41300	Loyola	01/01/2001	4664.5	3156.0	1952.8	125008
4	40760	Granville	01/01/2001	3109.8	2126.0	1453.8	84189

```
1 L_Rides.describe()
```

	station_id	avg_weekday_rides	avg_saturday_rides	avg_sunday-holiday_rides	monthtotal
count	31615.00000	31615.000000	31615.000000	31615.000000	31615.000000
mean	40762.74996	3911.504681	2270.955350	1653.016100	100968.823913
std	446.77332	3357.575472	2292.169994	1791.625929	87836.460042
min	40010.00000	0.000000	0.000000	0.000000	0.000000
25%	40370.00000	1468.650000	816.200000	557.000000	37948.500000
50%	40750.00000	2972.500000	1476.800000	1026.000000	75250.000000
75%	41150.00000	5124.100000	2912.300000	2053.000000	132575.000000
max	41700.00000	24041.400000	19171.300000	15982.000000	670496.000000

Data Analysis

Which of the two train stations next to a certain location is more popular?

Bryn Mawr and Berwyn Red Line

```
# Find all the data where the Berwyn stop is the 'stationame'
L_Rides[L_Rides['stationame'] == 'Berwyn']
```

	station_id	stationame	month_beginning	avg_weekday_rides	avg_saturday_rides	avg_sunday-holiday_rides	monthtotal
7	40340	Berwyn	01/01/2001	3329.6	2050.8	1445.6	88683
149	40340	Berwyn	02/01/2001	3420.1	2073.8	1564.3	82954
290	40340	Berwyn	03/01/2001	3402.8	2226.8	1579.3	92312
431	40340	Berwyn	04/01/2001	3343.7	2223.0	1560.4	86911
573	40340	Berwyn	05/01/2001	3388.9	2177.3	1690.4	91716

```
L_Rides.dtypes
```

```
station_id                int64
stationame               object
month_beginning          object
avg_weekday_rides       float64
avg_saturday_rides      float64
avg_sunday-holiday_rides float64
monthtotal                int64
dtype: object
```

Converting the date stamp into a date format in Pandas

```
L_Rides['date_time'] = pd.to_datetime(L_Rides['month_beginning'])
L_Rides=L_Rides.drop(['month_beginning'], axis=1)
L_Rides.head()
```

	station_id	stationame	avg_weekday_rides	avg_saturday_rides	avg_sunday-holiday_rides	monthtotal	date_time
0	40900	Howard	6233.9	3814.5	2408.6	164447	2001-01-01
1	41190	Jarvis	1489.1	1054.0	718.0	40567	2001-01-01
2	40100	Morse	4412.5	3064.5	2087.8	119772	2001-01-01
3	41300	Loyola	4664.5	3156.0	1952.8	125008	2001-01-01
4	40760	Granville	3109.8	2126.0	1453.8	84189	2001-01-01

Data Analysis

Limiting the analysis to 2017

```
Berwyn_Rides = L_Rides[(L_Rides['stationname'] == 'Berwyn') &
                       (L_Rides['date_time'] >= '01/01/2017') &
                       (L_Rides['date_time'] < '12/31/2017')]
```

```
Berwyn_Rides['monthtotal'].sum()
```

```
1094141
```

```
BrynMawr_Rides = L_Rides[(L_Rides['stationname'] == 'Bryn Mawr') &
                         (L_Rides['date_time'] >= '01/01/2017') &
                         (L_Rides['date_time'] < '12/31/2017')]
BrynMawr_Rides['monthtotal'].sum()
```

```
1509613
```

Here, we can see that the traffic at BrynMawr in 2017 is approximately 50% greater than the traffic at Berwyn

Finding the Most and Least Popular Stations

```
1 # What are the most popular stations and the least popular?
2 Station_Group = L_Rides.groupby(['stationame'])
```

```
1 Station_Group.groups
```

```
{'18th': Int64Index([   76,   218,   359,   501,   643,   784,   926,  1069,  1211,
             1353,
            ...
            30256, 30400, 30544, 30688, 30832, 30976, 31119, 31262, 31405,
            31548],
           dtype='int64', length=222),
 '35-Bronzeville-IIT': Int64Index([   98,   240,   381,   523,   665,   807,   950,  1092,  1234,
             1376,
            ...
            30281, 30425, 30569, 30713, 30857, 31001, 31144, 31287, 31430,
            31573],
           dtype='int64', length=222),
```

```
1 Station_Group['monthtotal'].sum()
```

```
stationame
18th                    8601327
35-Bronzeville-IIT     12274103
35th/Archer            15275681
43rd                    5633804
47th-Dan Ryan          17599600
                          ...
Western-Cermak          5369566
Western-Forest Park     8266947
Western-Orange         19636487
Western/Milwaukee      25044551
Wilson                 33472920
Name: monthtotal, Length: 148, dtype: int64
```

Station Popularity

```
1 Station_Totals = pd.DataFrame(Station_Group['monthtotal'].sum())
2 Station_Totals.sort_values(ascending=False,by='monthtotal',inplace=True)
3 Station_Totals
```

stationname	monthtotal
Clark/Lake	94794425
Lake/State	94243224
Chicago/State	87346160
95th/Dan Ryan	70730114
Belmont-North Main	70676134
...	...
King Drive	3751424
Kostner	2294002
Cermak-McCormick Place	2139792
Oakton-Skokie	1899165
Homan	27

148 rows × 1 columns

Top 5 Stations

```
1 Station_Totals.head(5)
```

stationname	monthtotal
Clark/Lake	94794425
Lake/State	94243224
Chicago/State	87346160
95th/Dan Ryan	70730114
Belmont-North Main	70676134

Last 5 Stations

```
1 Station_Totals.tail(5)
```

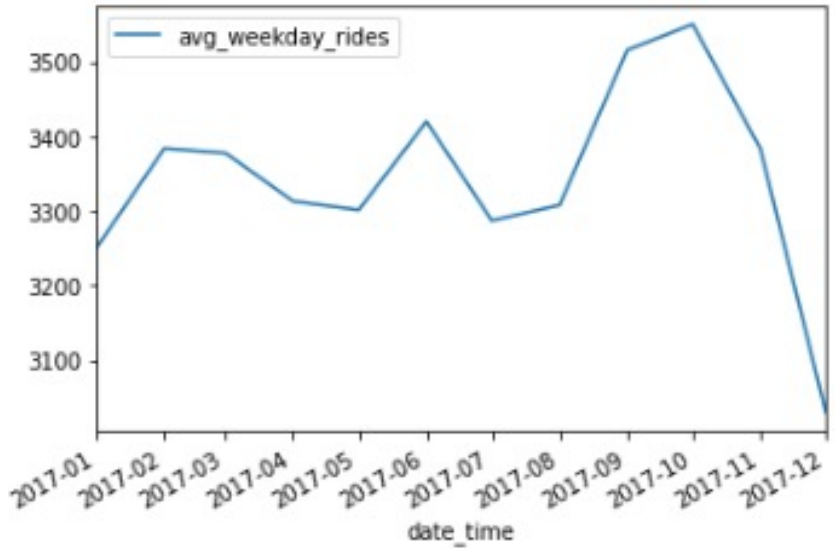
stationname	monthtotal
King Drive	3751424
Kostner	2294002
Cermak-McCormick Place	2139792
Oakton-Skokie	1899165
Homan	27

Plotting Results

Average Weekday Rides

```
1 Berwyn_Rides.plot(x ='date_time', y='avg_weekday_rides')
```

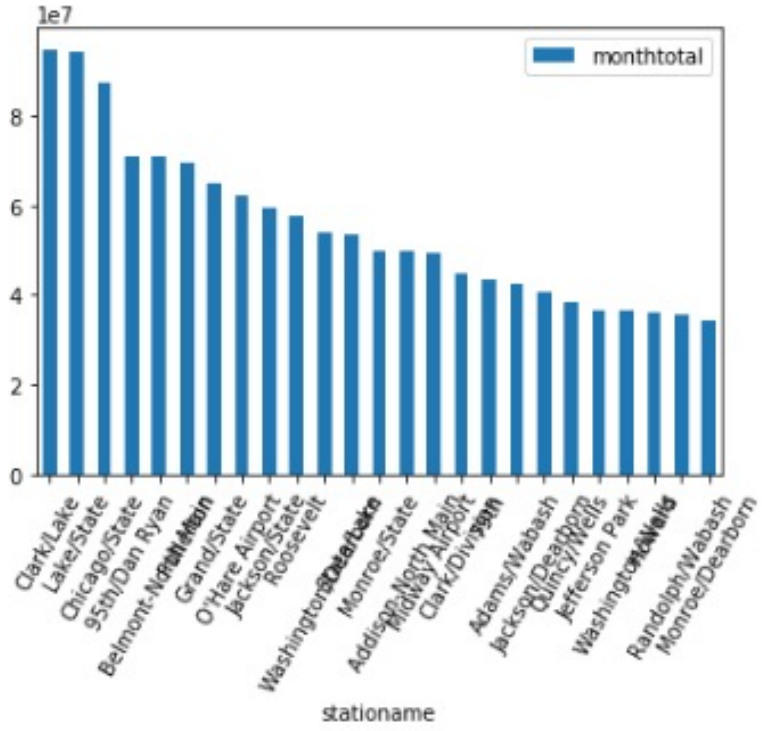
```
<matplotlib.axes._subplots.AxesSubplot at 0x235db612a20>
```



Top 25 Stations

```
1 Station_Totals.head(25).plot(kind='bar',rot=60)
```

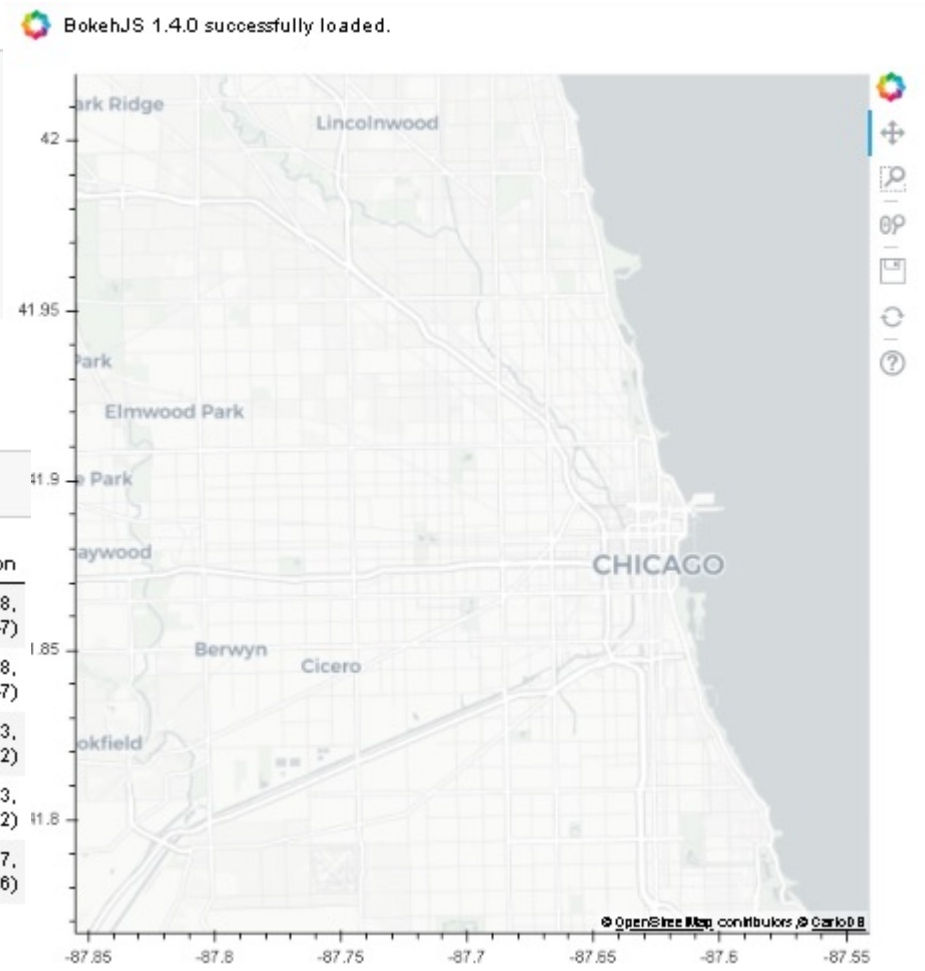
```
<matplotlib.axes._subplots.AxesSubplot at 0x235db57b6a0>
```



Creating Interactive Plots

What part of the city has the most train ridership?

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.tile_providers import CARTODBPOSITRON
p = figure(x_range=(-9780000, -9745000), y_range=(5130000, 5160000),
           x_axis_type="mercator", y_axis_type="mercator")
p.add_tile(CARTODBPOSITRON)
output_notebook()
show(p)
```



Train Stop Data

```
L_Map = pd.read_csv('CTA_-_System_Information_-_List_of__L__Stops_-_Map.csv',sep=',')
L_Map.head()
```

	STOP_ID	STOP_NAME	STATION_NAME	MAP_ID	ADA	RED	BLUE	G	BRN	P	Pexp	Y	Pnk	O	Location
0	30162	18th (54th/Cermak-bound)	18th	40830	True	False	False	False	False	False	False	False	True	False	(41.857908, -87.669147)
1	30161	18th (Loop-bound)	18th	40830	True	False	False	False	False	False	False	False	True	False	(41.857908, -87.669147)
2	30022	35th/Archer (Loop-bound)	35th/Archer	40120	True	False	False	False	False	False	False	False	False	True	(41.829353, -87.680622)
3	30023	35th/Archer (Midway-bound)	35th/Archer	40120	True	False	False	False	False	False	False	False	False	True	(41.829353, -87.680622)
4	30214	35-Bronzeville-IIT (63rd-bound)	35th-Bronzeville-IIT	41120	True	False	False	True	False	False	False	False	False	False	(41.831677, -87.625826)

Plotting Interactive Maps

```
import math
from ast import literal_eval
def merc(Coords):
    Coordinates = literal_eval(Coords)
    lat = Coordinates[0]
    lon = Coordinates[1]

    r_major = 6378137.000
    x = r_major * math.radians(lon)
    scale = x/lon
    y = 180.0/math.pi * math.log(math.tan(math.pi/4.0 +
        lat * (math.pi/180.0)/2.0)) * scale
    return (x, y)
```

```
L_Map['coords_x'] = L_Map['Location'].apply(lambda x: merc(x)[0])
L_Map['coords_y'] = L_Map['Location'].apply(lambda x: merc(x)[1])
```

```
L_Map[['Location','coords_x','coords_y']].head()
```

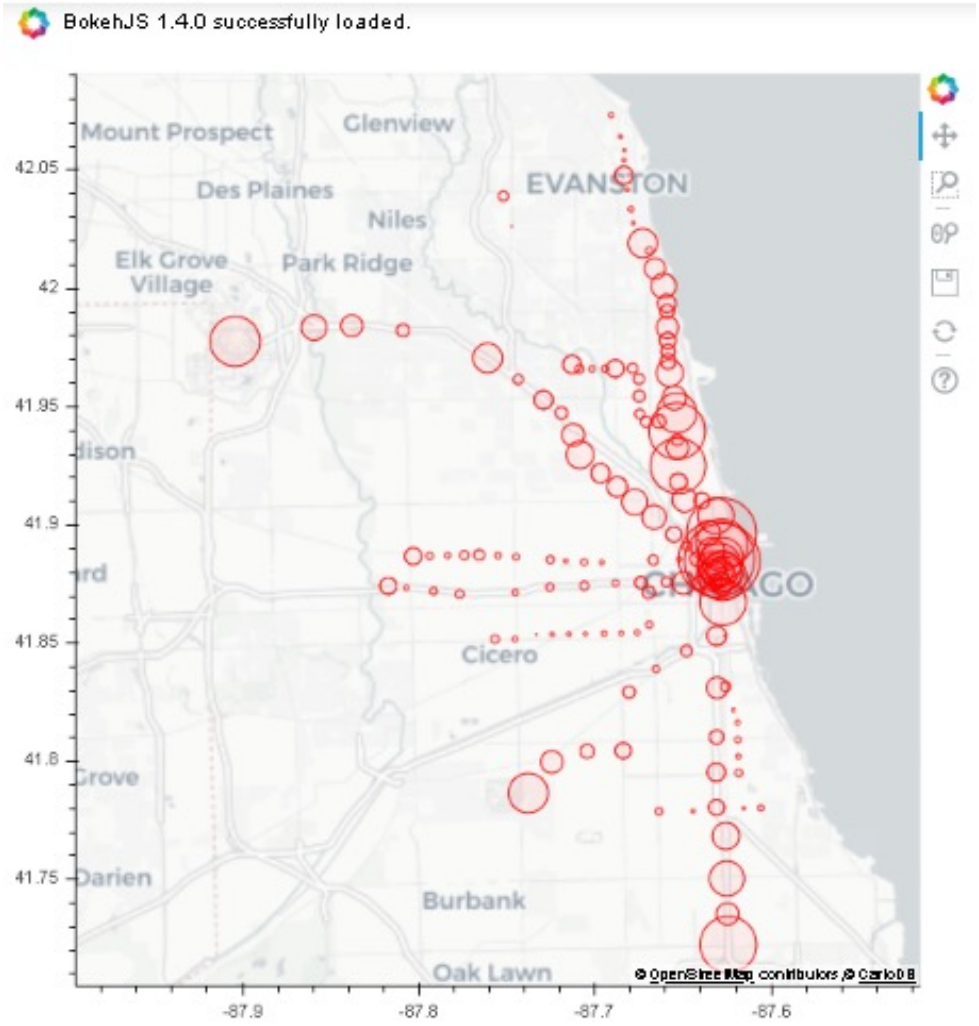
	Location	coords_x	coords_y
0	(41.857908, -87.669147)	-9.759285e+06	5.139718e+06
1	(41.857908, -87.669147)	-9.759285e+06	5.139718e+06
2	(41.829353, -87.680622)	-9.760562e+06	5.135452e+06
3	(41.829353, -87.680622)	-9.760562e+06	5.135452e+06
4	(41.831677, -87.625826)	-9.754462e+06	5.135799e+06

We also need to get rid of duplicate stops since they represent the same stop going in different directions

```
#To get rid of duplicates (two stops per station)
L_Map.drop_duplicates(subset='MAP_ID', keep="last", inplace=True)
```

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.tile_providers import CARTODBPOSITRON
p = figure(x_axis_type="mercator", y_axis_type="mercator")
p.add_tile(CARTODBPOSITRON)
p.circle(x = L_Map['coords_x'],
         y = L_Map['coords_y'])
output_notebook()
show(p)
```

Plotting Interactive Maps




```
# 1. Group ridership totals
Station_Group = L_Rides.groupby(['station_id'])
Station_Totals = pd.DataFrame(Station_Group['monthtotal'].sum())
# 2. Merge dataframes
Merged = pd.merge(L_Map, Station_Totals, left_on='MAP_ID', right_index=True)
```

```
Merged[['coords_x','coords_y','monthtotal']].head()
```

	coords_x	coords_y	monthtotal
1	-9.759285e+06	5.139718e+06	8601327
3	-9.760562e+06	5.135452e+06	15275681
5	-9.754462e+06	5.135799e+06	12274103
7	-9.753705e+06	5.133526e+06	5633804
9	-9.753683e+06	5.132443e+06	7463158

```
Merged['circle_sizes'] = Merged['monthtotal'] / 2000000
```

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.tile_providers import CARTODBPOSITRON
# range bounds supplied in web mercator coordinates
p = figure(x_axis_type="mercator", y_axis_type="mercator")
p.add_tile(CARTODBPOSITRON)
p.circle(x=Merged['coords_x'],
         y=Merged['coords_y'],
         size=Merged['circle_sizes'],
         line_color="#FF0000",
         fill_color="#FF0000",
         fill_alpha=0.05)

output_notebook()
show(p)
```

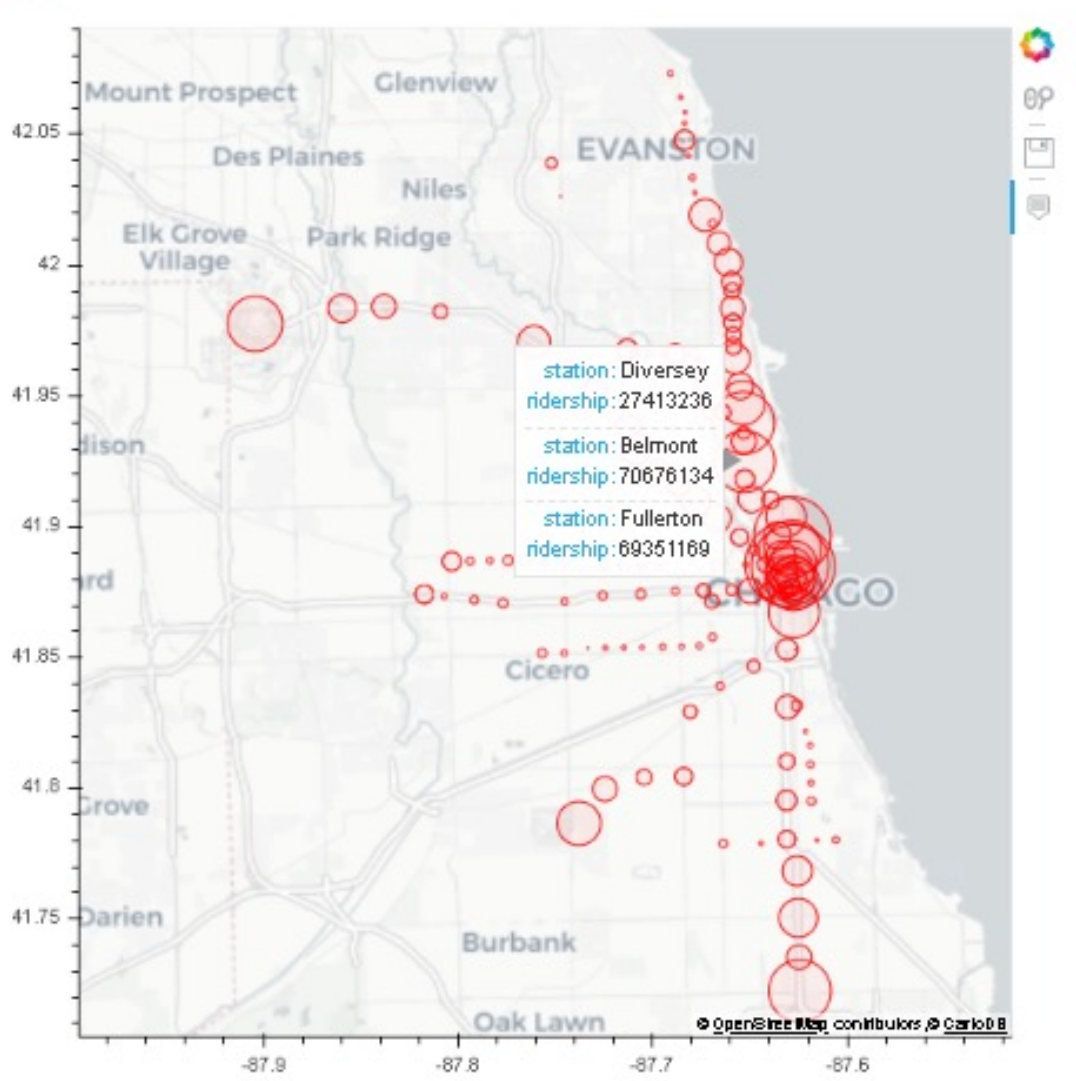
```
BokehDeprecationWarning: CARTODBPOSITRON was deprecated in Bokeh 1.1.0 and will be removed, use get_provid
er(Vendors.CARTODBPOSITRON) instead.
```

Creating Interactive Maps

```
from bokeh.plotting import figure, show, output_notebook
from bokeh.tile_providers import CARTODBPOSITRON
from bokeh.models import ColumnDataSource, HoverTool
source = ColumnDataSource(data=dict(
                        x=list(Merged['coords_x']),
                        y=list(Merged['coords_y']),
                        ridership=list(Merged['monthtotal']),
                        sizes=list(Merged['circle_sizes']),
                        stationname=list(Merged['STATION_NAME'])))
hover = HoverTool(tooltips=[
    ("station", "@stationname"),
    ("ridership","@ridership")

])
# range bounds supplied in web mercator coordinates
p = figure(x_axis_type="mercator",
           y_axis_type="mercator",
           tools=[hover, 'wheel_zoom','save'])
p.add_tile(CARTODBPOSITRON)
p.circle(x='x',
         y='y',
         source=source,
         size='sizes',
         line_color="#FF0000",
         fill_color="#FF0000",
         fill_alpha=0.05)

output_notebook()
show(p)
```



Exercise

In order to familiarize yourself with the package, download the OSMnx_Class_Exercise.ipynb file from Canvas.

Exercise: Import the transportation network for your hometown and find your home address and the address of a point of interest you visited frequently (school, mall, etc.) and find the shortest path between the two addresses.

Bonus Exercise: Try finding the fastest path versus the shortest path using the second part of the code.

Second part of the code: osmnx-examples/02-routing-speed-time.ipynb at v0.13.0 · gboeing/osmnx-examples · GitHub

Tools for more information

For more tutorials with OSMnx, check out the following useful links:

- OSMnx Documentation: User reference — OSMnx 1.1.2 documentation
- GitHub - gboeing/osmnx: OSMnx: Python for street networks. Retrieve, model, analyze, and visualize street networks and other spatial data from OpenStreetMap.
- Tutorials: GitHub - gboeing/osmnx-examples: Usage examples, demos, and tutorials for OSMnx.
- Features + Projects: OSMnx Features Round-Up – Geoff Boeing