



NEW YORK UNIVERSITY

SPATIAL INFORMATION RETRIEVAL IN THE CITY

Anastasios Noulas

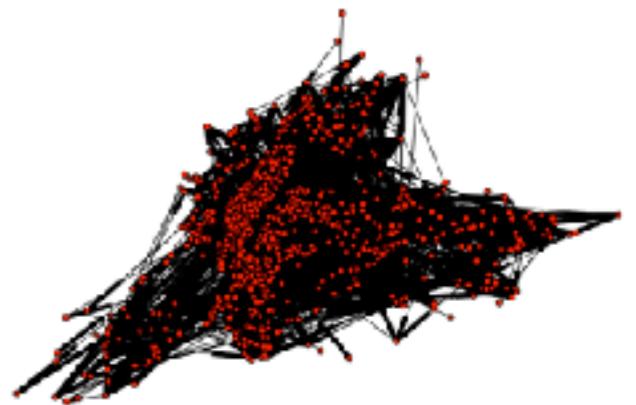
Data Science Institute

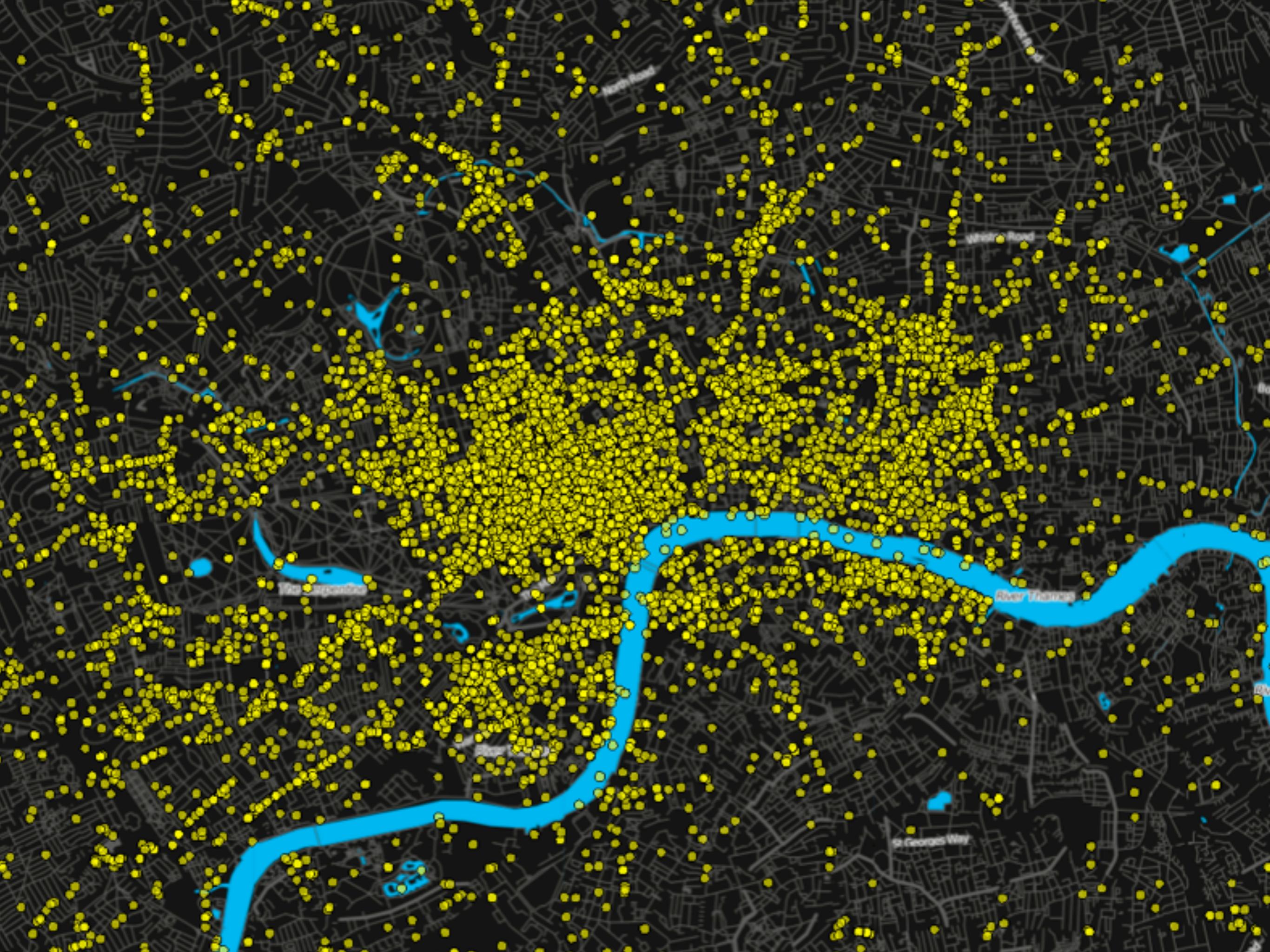
School of Computing & Communications

Lancaster University

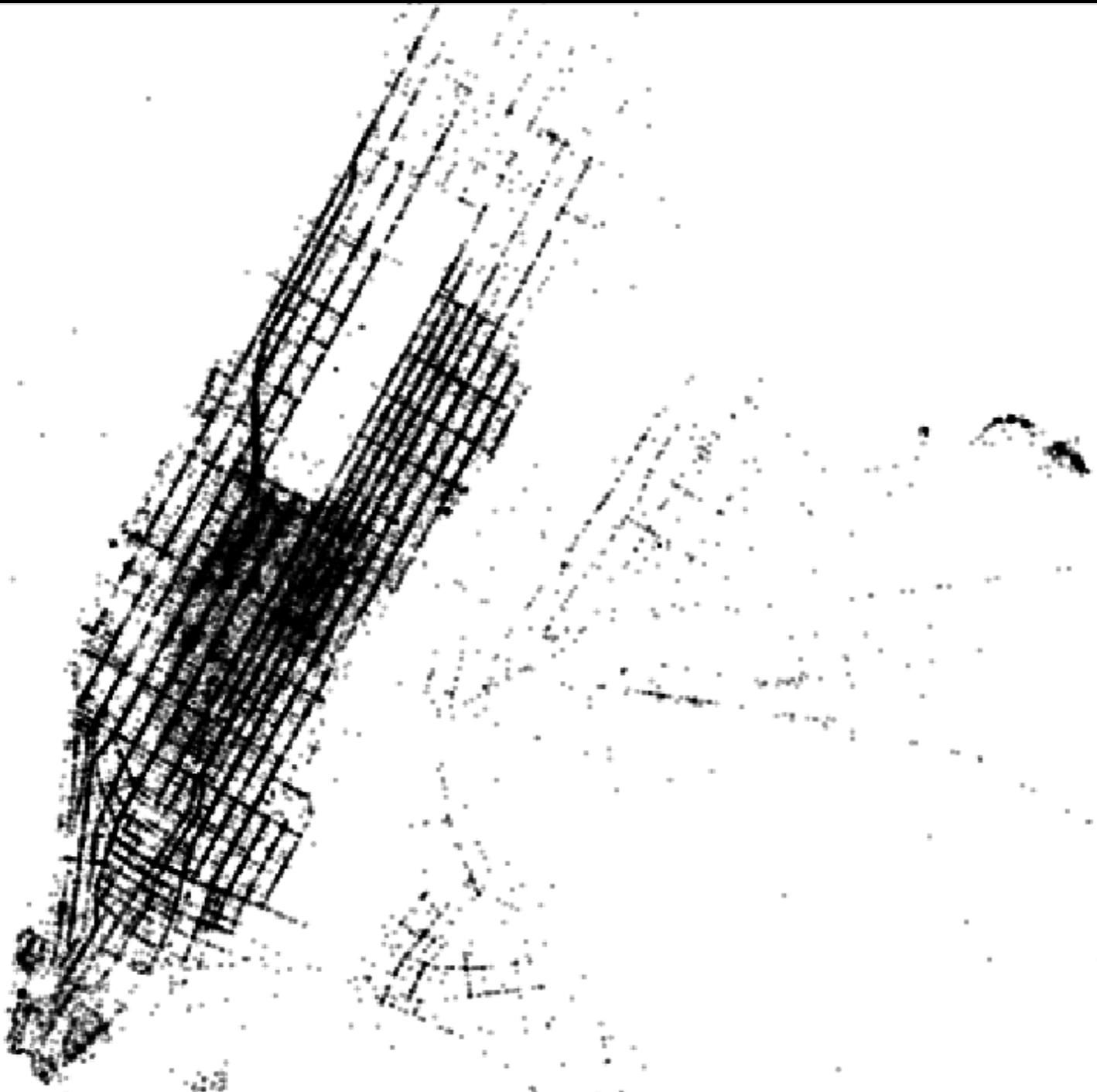
Practical: Outline

- Introduce basic spatial retrieval methods.
- Define and explore place networks using networks.
- Showcase Uber's deck.gl visualisation framework.
- Practical examples with Bokeh and Datashader





THE NEW YORK CITY TAXI DATASET



FOILing NYC's Taxi Trip Data

Freedom of Information Law

2013 Trip Data, 11GB, zipped!

2013 Fare Data, 7.7GB

**Idea: Uber Vs Yellow Taxi
Price Comparison.**

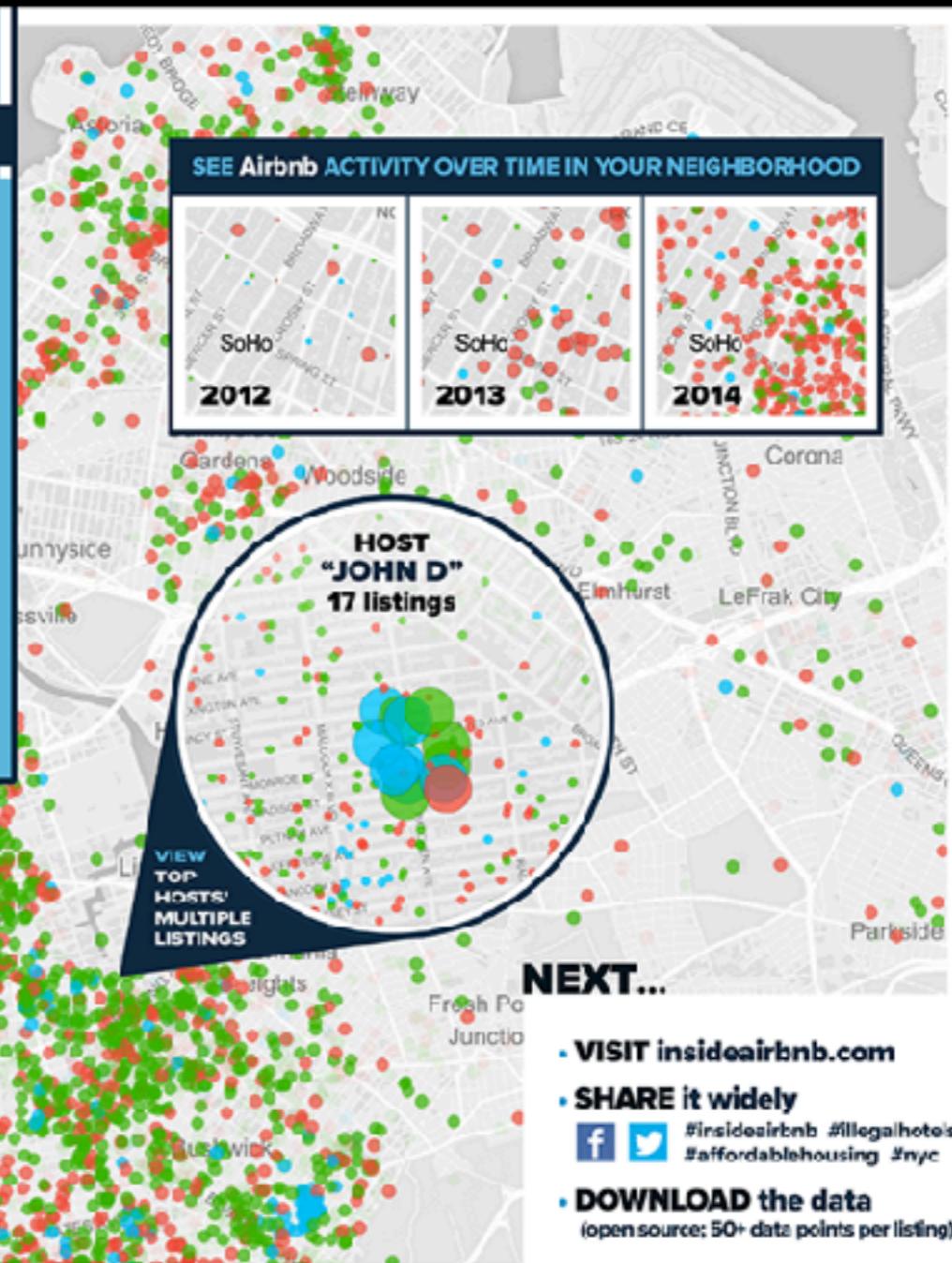
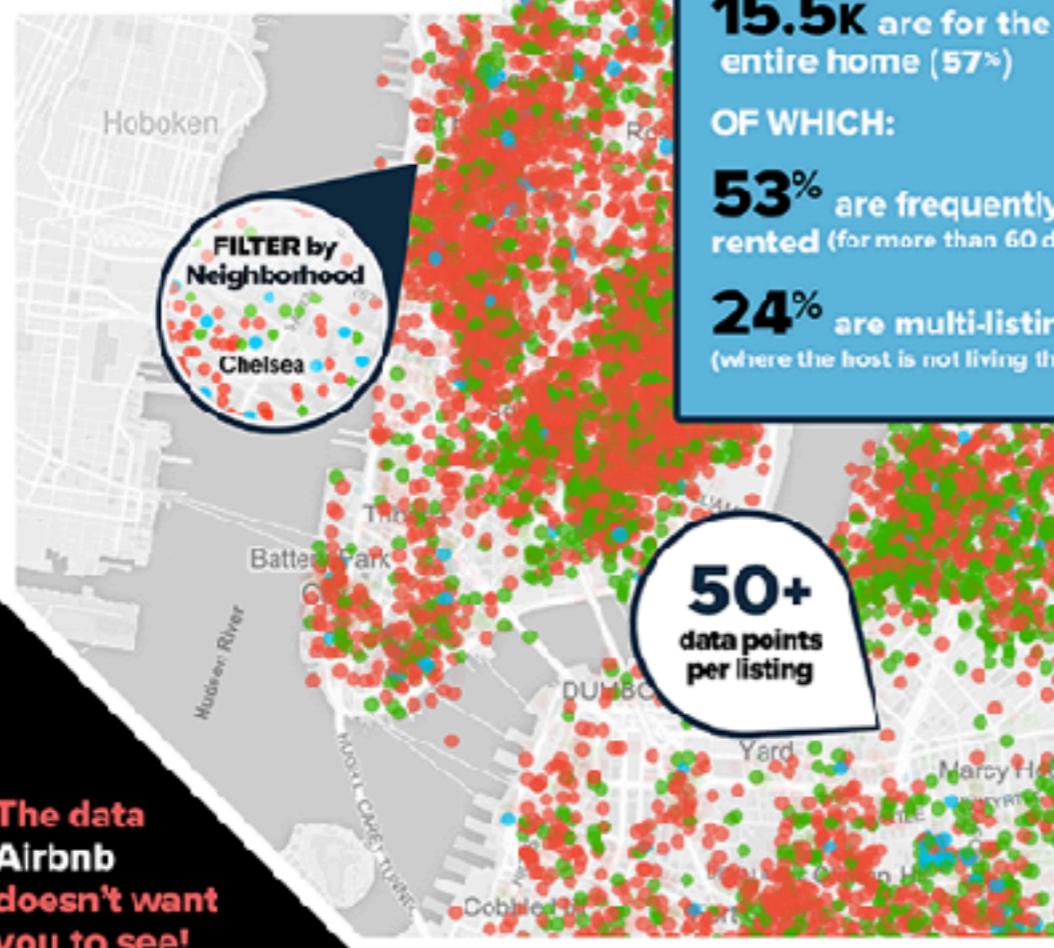
http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

Inside Airbnb

Adding data to the debate

INDEPENDENT, NON-COMMERCIAL,
OPEN SOURCE DATA TOOL

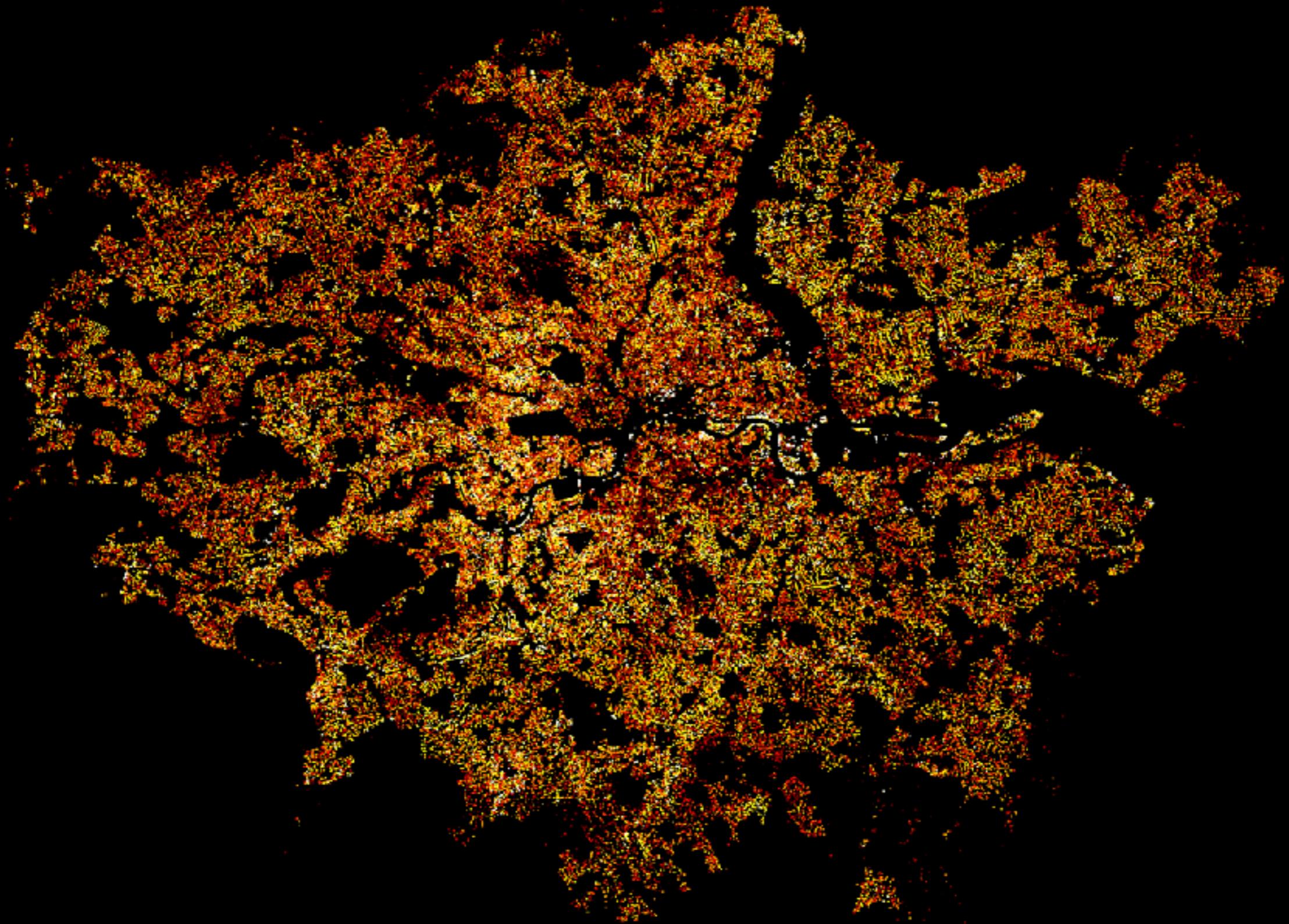
How is Airbnb really
being used in and affecting
your neighborhood?



The data
Airbnb
doesn't want
you to see!

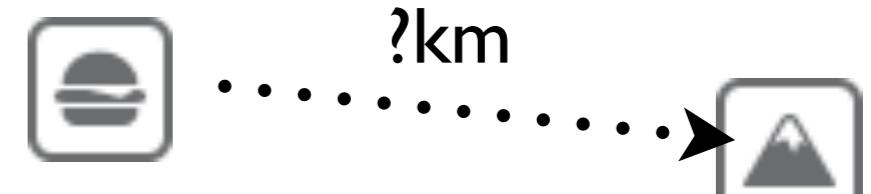
- VISIT insideairbnb.com
- SHARE it widely
 - f
 - t
 - #insideairbnb #illegalhotels #affordablehousing #nyc
- DOWNLOAD the data
(open source: 50+ data points per listing)

<http://insideairbnb.com/about.html>



Property sales in London 1995 - 2012

We will learn ..



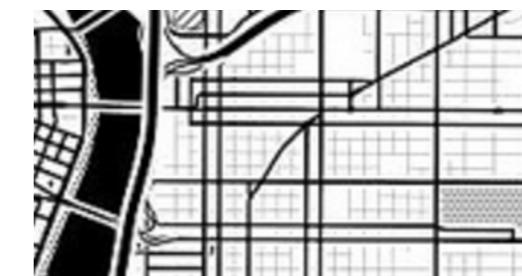
To measure the geographic distance between pairs of venues.



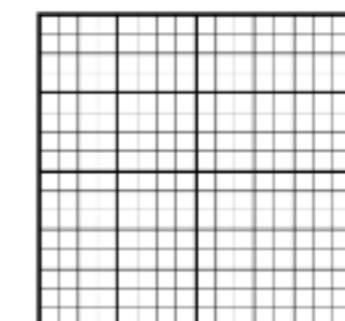
To infer the geographic center of a city.



To get all places within a radius given a pair of geographic coordinates.



Create our own Maps



Apply a grid over a city.

amongst others :)

What's the “true” centre of Europe?

https://en.wikipedia.org/wiki/Geographical_midpoint_of_Europe



the Centre of Europe, a sculpture park containing the world's largest sculpture made of TV sets.^[3]

Belarus [edit]

In 2000 Belarusian scientists Alexey Solomonov and Valery Anoshko published a report that stated the geographic centre of Europe was located near Lake Sho ($55^{\circ}10'55''N\ 28^{\circ}15'30''E$; Belarusian: Шо) in Vitsebsk Voblast.^[4]

Scientists from the Russian Central Research Institute of Geodesy, Aerial Survey and Cartography (Russian: ЦНИИГАиК) confirmed the calculations of Belarusian geodesists that the geographical centre of Europe is located in Polotsk $55^{\circ}30'0''N\ 28^{\circ}48'0''E$. A small monument to the Geographical Centre of Europe was set up in Polotsk on May 31, 2008.^[5]

See also: [Geographic center of Belarus](#)

Hungary [edit]

It is claimed that a 1992 survey found that the geometric centre of Europe is in the village of Tályá, Hungary $48.23610^{\circ}N\ 21.22574^{\circ}E$.^{[6][7]} In 2000, a sculpture was erected in the village, with a table on it declaring the place the "Geometric Centre of Europe".^[8]

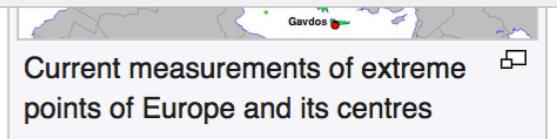
Estonia [edit]

It is claimed that if all the islands of Europe – from the Azores to Franz Joseph Land and from Crete to Iceland – are taken into consideration then the centre of Europe lies at $58^{\circ}18'14''N\ 22^{\circ}16'44''E$ in the village of Mõnnuste, on Saaremaa island in western Estonia. Again, no author and no method of calculation have been disclosed. The local Kärla Parish is seeking to verify the location and to turn it into a tourist location.^[9]

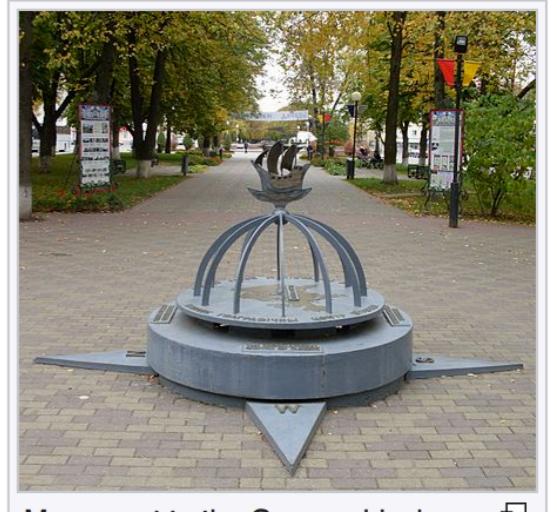
Other claimants [edit]

Locations currently vying for the distinction of being the centre of Europe include:

- the village of Kremnické Bane or the neighbouring village Krahule, near Kremnica, in central Slovakia^[10]
- the small town of Rakhiv, or the village of Dilove near Rakhiv, in western Ukraine^[11]
- the village of Bernotai, or Purnuškės, near Vilnius, in Lithuania^{[12][13]}
- a point on the island of Saaremaa in Estonia^[14]
- a point near Polotsk, or in Vitebsk, or near Babruysk, or near lake Sho in Belarus^[15]
- a point near the town of Tályá, in north-eastern Hungary^[16]



Monument in Purnuškės, Lithuania



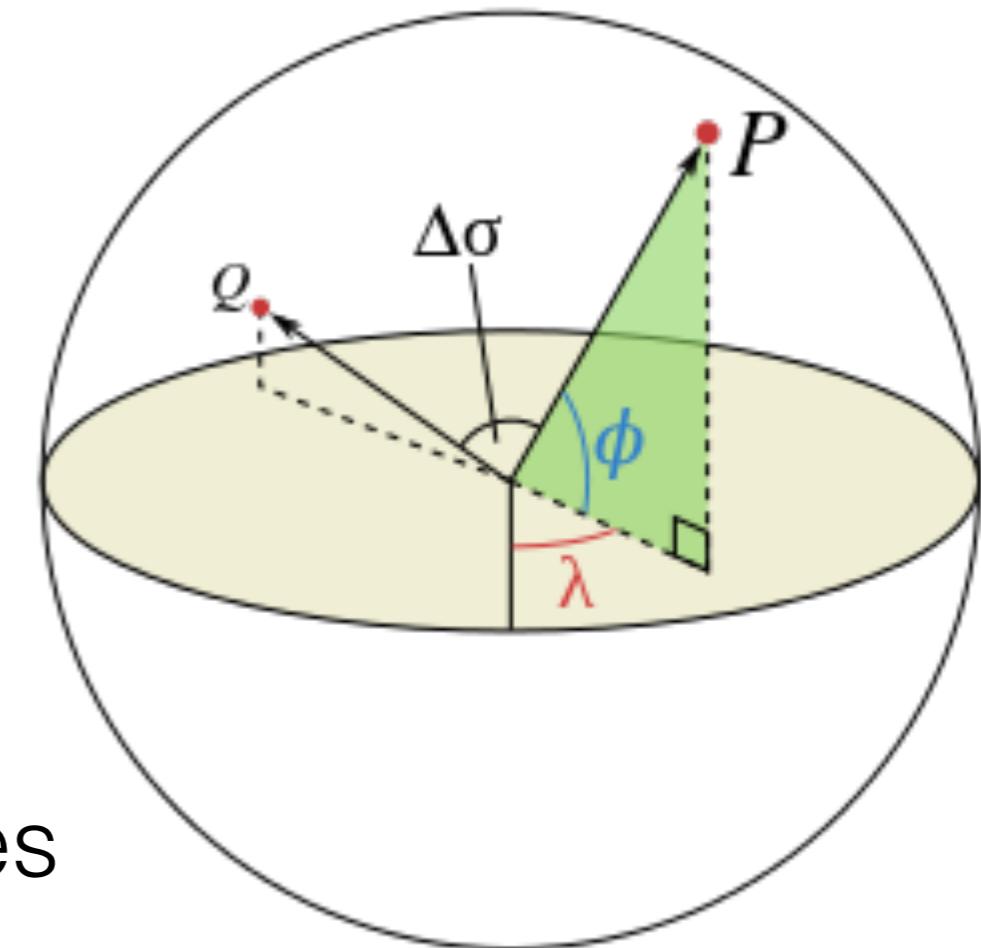
Monument to the Geographical

Calculating geographic distances

The Earth is not flat :)

So we use a spherical object as an - abstraction - of it.

The so-call haversine formula provides a satisfactory, non-perfect, way to measure geographic distances between two points on the planet.



$$\Delta\sigma = 2 \arcsin \sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos \phi_1 \cdot \cos \phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}.$$

Download <https://www.dropbox.com/s/e288u70hdckxatb/geolocator.py?dl=0>

source:
wikipedia

Harvesine formula in Python

geolocator.py

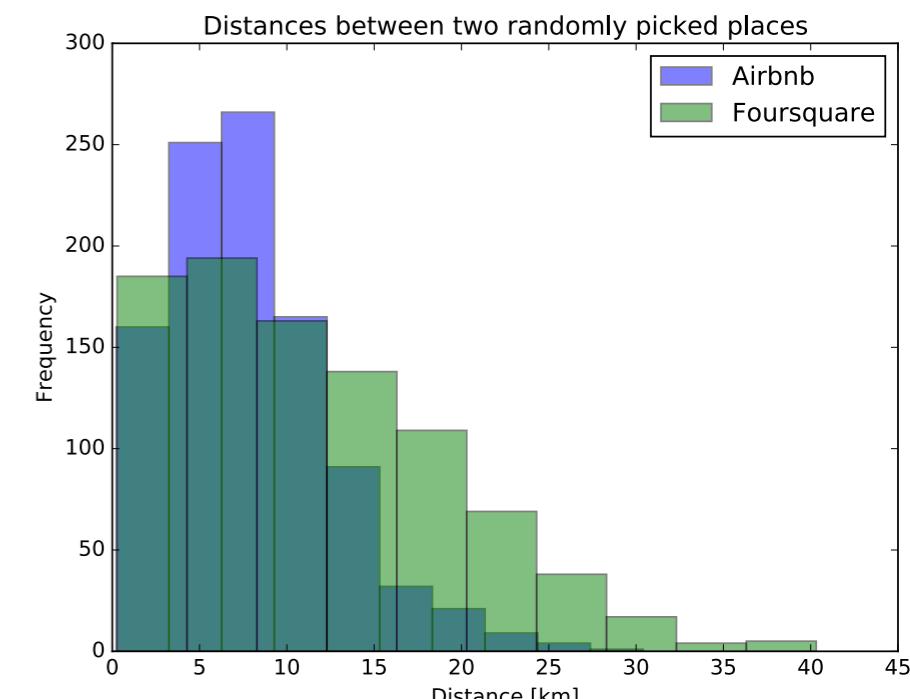
```
20 # earth's mean radius = 6,371km
21 earthradius = 6371.0
22
23 ▼ def getDistance(loc1, loc2):
24     "aliased default algorithm; args are (lat_decimal,lon_decimal) tuples"
25
26     return getDistanceByHaversine(loc1, loc2)
27
28 ▼ def getDistanceByHaversine(loc1, loc2):
29     "Haversine formula – give coordinates as (lat_decimal,lon_decimal) tuples"
30
31     lat1, lon1 = loc1
32     lat2, lon2 = loc2
33     # convert to radians
34     lon1 = lon1 * pi / 180.0
35     lon2 = lon2 * pi / 180.0
36     lat1 = lat1 * pi / 180.0
37     lat2 = lat2 * pi / 180.0
38
39     # haversine formula
40     dlon = lon2 - lon1
41     dlat = lat2 - lat1
42     a = (sin(dlat/2))**2 + cos(lat1) * cos(lat2) * (sin(dlon/2.0))**2
43     c = 2.0 * atan2(sqrt(a), sqrt(1.0-a))
44     km = earthradius * c
45     return km
```

```
1 #####  
2 # Reading CSV file with Airbnb Listings #  
3 # Data from insideairbnb.com/ #  
4 #####  
5  
6 #read a csv file with location information  
7 #let's start with Airbnb listings  
8 import csv  
9 import random  
10  
11 input_file = csv.DictReader(open('london_listings.csv'))  
12  
13 #CSV headers: id,name,host_id,host_name,neighbourhood_group,neighbourhood,latitude,longitude,room_type  
14  
15 airbnb_listings = {} # dictionary key is listing id, values are listing info in a python <list>  
16 for row in input_file:  
17  
18     if random.random() > 1.0: #control sample size: 100% here  
19         continue  
20  
21  
22     listing_id = row['id']  
23     latitude = row['latitude']  
24     longitude = row['longitude']  
25     num_reviews = row['number_of_reviews']  
26     price = row['price']  
27  
28     airbnb_listings[listing_id] = [float(latitude), float(longitude), num_reviews, float(price)]  
29  
30  
31 num_listings = len(airbnb_listings)  
32 print 'Number of Airbnb listings in London: ' + str(num_listings)
```

```

125 #####
126 # Geographic Analysis: Measure Distances      #
127 #####
128 |
129 import sys
130 sys.path.append('./location_central/') #adding directory to python path
131 import geolocator
132
133 num_samples = 1000
134 airbnb_distances = []
135 foursquare_distances = []
136 ▼ for i in xrange(0, num_samples):
137
138     #sample two airbnb venues
139     listing1 = random.choice(airbnb_listings.keys())
140     listing2 = random.choice(airbnb_listings.keys())
141
142     #sample two foursquare venues
143     venue1 = random.choice(foursquare_venues.keys())
144     venue2 = random.choice(foursquare_venues.keys())
145
146     air_distance = geolocator.getDistance(airbnb_listings[listing1][0:2], airbnb_listings[listing2][0:2])
147     venue_distance= geolocator.getDistance(foursquare_venues[venue1][0:2], foursquare_venues[venue2][0:2])
148
149     airbnb_distances.append(air_distance)
150     foursquare_distances.append(venue_distance)
151
152 #filtering out extreme values
153 airbnb_distances = [d for d in airbnb_distances if d < 50]
154 foursquare_distances = [d for d in foursquare_distances if d < 50]
155
156 pylab.title('Distances between two randomly picked places')
157 pylab.ylabel('Frequency')
158 pylab.xlabel('Distance [km]')
159 pylab.hist(airbnb_distances, alpha=0.5, label='Airbnb')
160 pylab.hist(foursquare_distances, alpha=0.5, label='Foursquare')
161 pylab.legend()
162 pylab.savefig('./graphs/distances.pdf')

```



Spatial Retrieval (1)

SpatialRetriever.py

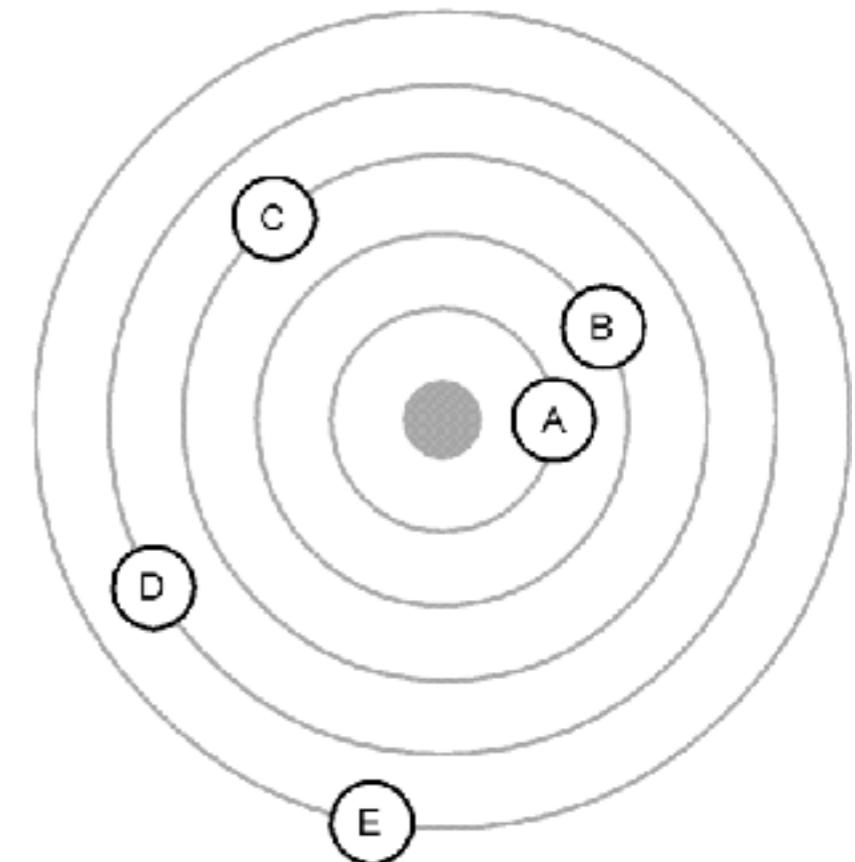
```
1 import sys
2 import numpy as np
3 import math
4 from math import *
5 from sets import Set
6 import geolocator
7 import random
8
9
10 class SpatialRetriever:
11     '''The Spatial Retriever is a module optimized for spatial queries.'''
12     def __init__(self, spatialElements):
13         self.spatialElements = spatialElements
14         self.load_location_data()
15
16
17     def load_location_data(self):
18         allLatLongs = []
19         allLatsDict = {}
20         allLongsDict = {}
21         allLats = []
22         allLongs = []
23         locationData = {}
24
25         for spatialElementId in self.spatialElements:
26             [latFloat, longFloat] = self.spatialElements[spatialElementId][0:2]
27             try:
28                 locationData[spatialElementId] = [latFloat, longFloat]
29             except:
30                 print sys.exc_info()
```

Spatial Retrieval (2)

```

66
67 def binary_search_optimized(self, sec, item):
68     left = 0
69     right = len(sec) - 1
70     while True:
71         mid = (left+right)/2
72         if item < sec[mid]:
73             right = mid
74         else:
75             left = mid
76
77         if right == left+1:
78             return sec[left:left+2]
79
80
81
82 def get_nearby_locations(self, latitude, longitude, distance, area_shape='square'):
83     #returns nearby locations given a point
84     ([lat_up, lat_down], [long_left, long_right]) = self.get_rectangle_coordinates(latitude,
85     #get lat list values
86     lat_left_limit_value = self.binary_search_optimized(self.allLats, lat_down)[1]
87     lat_right_limit_value = self.binary_search_optimized(self.allLats, lat_up)[0]
88
89     #...and indices
90     lat_left_limit_index = self.allLats.index(lat_left_limit_value)
91     lat_right_limit_index = self.allLats.index(lat_right_limit_value)
92
93     #get long list values
94     long_left_limit_value = self.binary_search_optimized(self.allLongs, long_left)[1]
95     long_right_limit_value = self.binary_search_optimized(self.allLongs, long_right)[0]
96
97     #...and indices

```



Spatial Retrieval (2++)

SpatialRetriever.py

....

```
all_lat_locs_cleared = []
for locs in all_locs_from_lat_list:
    for loc in locs:
        all_lat_locs_cleared.append(loc)

all_long_locs_cleared = []
for locs in all_locs_from_long_list:
    for loc in locs:
        all_long_locs_cleared.append(loc)

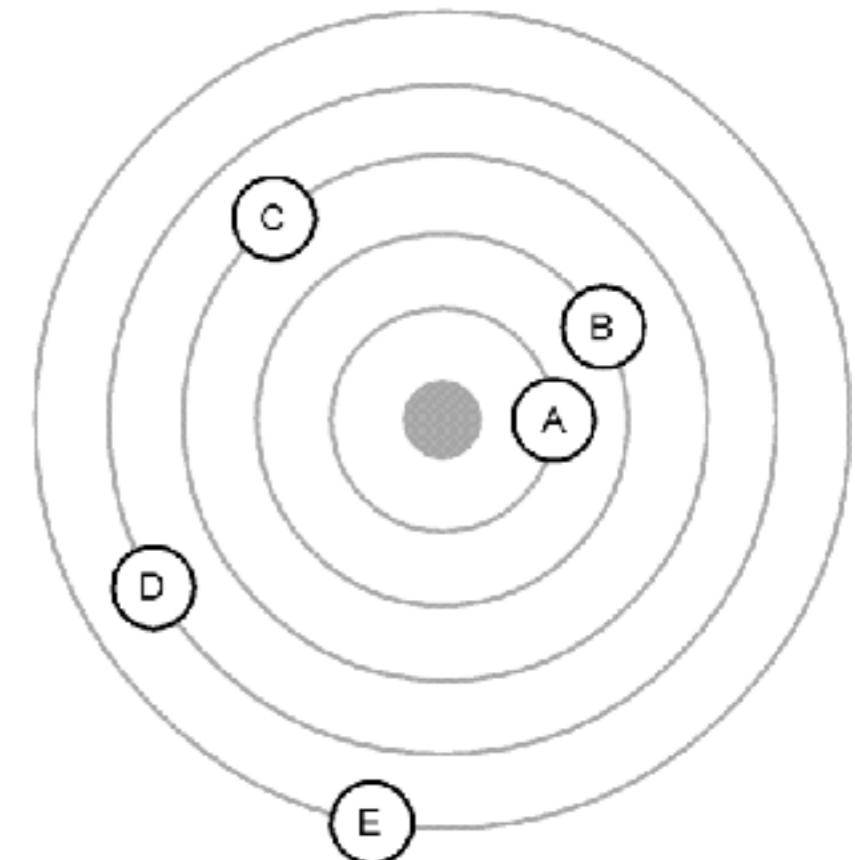
latSet = Set(all_lat_locs_cleared)
longSet = Set(all_long_locs_cleared)

all_locs_in_square_set = latSet.intersection(longSet)

location_list_insquare = list(all_locs_in_square_set)

if area_shape == 'square':
    return location_list_insquare

elif area_shape == 'circle':
    location_list_incircle = []
    for location in location_list_insquare:
        if geolocator.getDistance([latitude, longitude], self.locationData[location][0:2]) <= distance
            location_list_incircle.append(location)
    return location_list_incircle
```



Spatial Retrieval (3)

SpatialRetriever.py

```
def get_rectangle_coordinates(self,latitude,longitude,distance):
    d = distance                         #distance in km
    R = 6371.0                            #earth radius in km

    lat1 = latitude
    long1 = longitude

    #convert to radians
    long1 = long1 * pi / 180.0
    lat1 = lat1 * pi / 180.0
```

• • • •

Key method used by the previous method: afterall to get a grid of cell, you need to get a cell in the first place ...

The rest of the code is gibberish maths which i worked out one fine day during the first days of my phd ... it is the opposite process of the haversine formula for distance measurement..

Spatial Retrieval (3)

SpatialRetriever.py

• • • •

```
#convert to radians
long1 = long1 * pi / 180.0
lat1 = lat1 * pi / 180.0

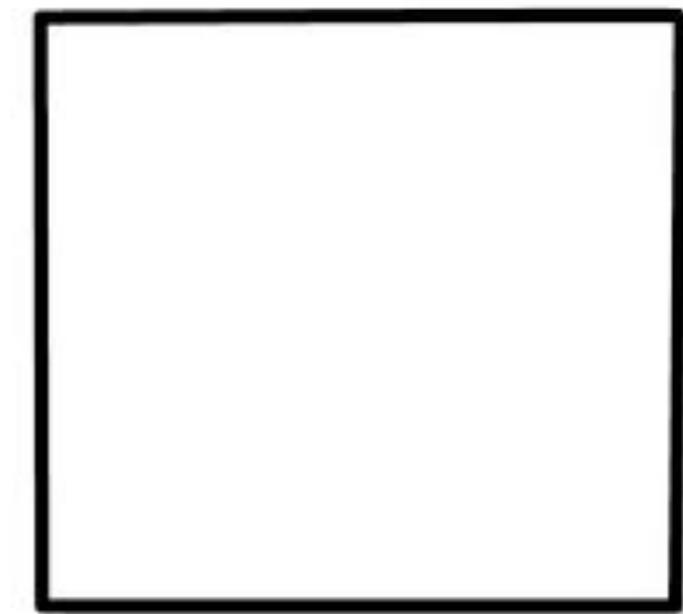
#case 1: lat1 == lat2
lat_up = lat1 + 2.0*asin(1.0/2.0 * sqrt(-sqrt(-4*pow(tan(d/(2.0*R)), 2) +1) +1)*sqrt(2)) #up
lat_down = lat1 - 2.0*asin(1.0/2.0 * sqrt(-sqrt(-4*pow(tan(d/(2.0*R)), 2) +1) +1)*sqrt(2)) #down
#case 2: long1 == long2
long_right = long1 + 2.0*asin(1.0/2.0 * sqrt(-sqrt(-4*pow(tan(d/(2.0*R)), 2) +1) +1)*sqrt(2)/pow(cos(lat1),1))
long_left = long1 - 2.0*asin(1.0/2.0 * sqrt(-sqrt(-4*pow(tan(d/(2.0*R)), 2) +1) +1)*sqrt(2)/pow(cos(lat1),1))

# convert from radians
lat_up = lat_up / pi * 180.0
lat_down = lat_down / pi * 180.0
long_left = long_left / pi * 180.0
long_right = long_right / pi * 180.0

lat2 = [lat_up,lat_down] #up,down
long2 = [long_left,long_right] #left,right

return (lat2,long2)
```

Reverse engineering of the harvesine formula
using maths can give you the coordinates
of rectangle for a given geographic center!



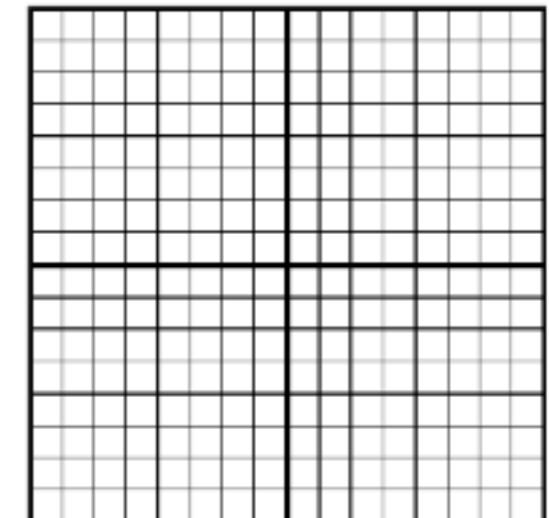
Spatial Retrieval (4)

SpatialRetriever.py

```
def area_recursive_splitter_v2(self, latitude, longitude, distance, number_of_squares):
    #split an area in squares of certain minimum size: number of squares better be power of 4
    ([lat_up, lat_down], [long_left, long_right]) = self.get_rectangle_coordinates(latitude, longitude, distance/2.0)
    if number_of_squares >= 4: #check
        new_dist = distance/2.0
        new_number_of_squares = number_of_squares/4

        #0.86*
        points_NW = self.area_recursive_splitter_v2(lat_up, long_left, new_dist, new_number_of_squares)
        points_NE = self.area_recursive_splitter_v2(lat_up, long_right, new_dist, new_number_of_squares)
        points_SE = self.area_recursive_splitter_v2(lat_down, long_right, new_dist, new_number_of_squares)
        points_SW = self.area_recursive_splitter_v2(lat_down, long_left, new_dist, new_number_of_squares)

        return points_SE + points_SW + points_NW + points_NE
    else:
        return [(latitude, longitude)]
```



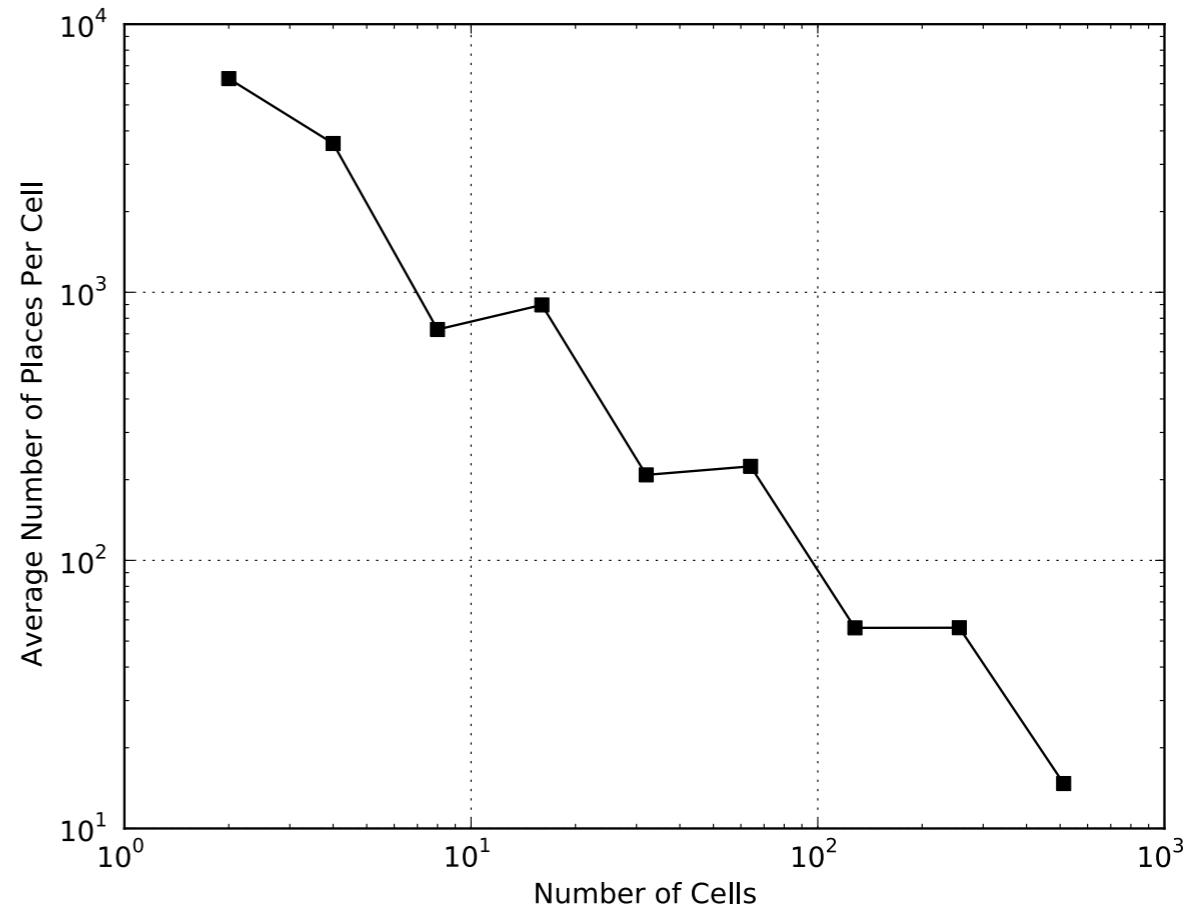
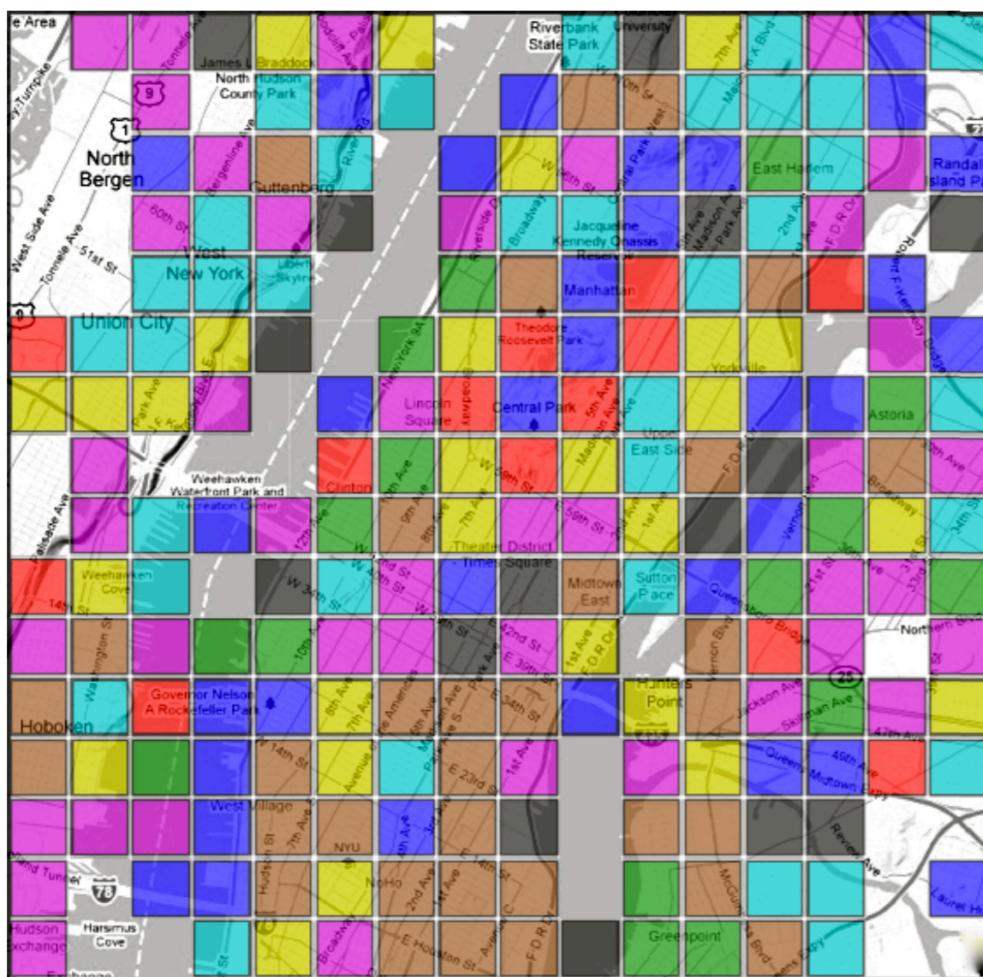
Given a central point (x,y) and a radius (distance), return a grid centered at (x,y).

The grid is represented as a list of tuples, where each tuple is the center of the grid's cell represented with geographic coordinates

The parameter `numberOfSquares` is the number of cells in the grid. This version is limited to `numberOfSquares` being a power of 2 (we will check an alternative grid make later)

why useful?

calculate the fractal dimension of a city



get nearby categories ...
and much more!

Spatial Retrieval (5)

SpatialRetriever.py

Gives you the top keys of a dictionary based on value number

```
171
172     def getTopDictionaryKeys(self,dictionary,number):
173         topList = []
174         a = dict(dictionary)
175         for i in range(0,number):
176             m = max(a, key=a.get)
177             topList.append( [m,a[m]] )
178             del a[m]
179
180     return topList
181
182
183     def get_city_center(self):
184         return (np.median(self.allLats), np.median(self.allLongs))
```

Gets “city centre” for the spatial object in question

Spatial Retrieval

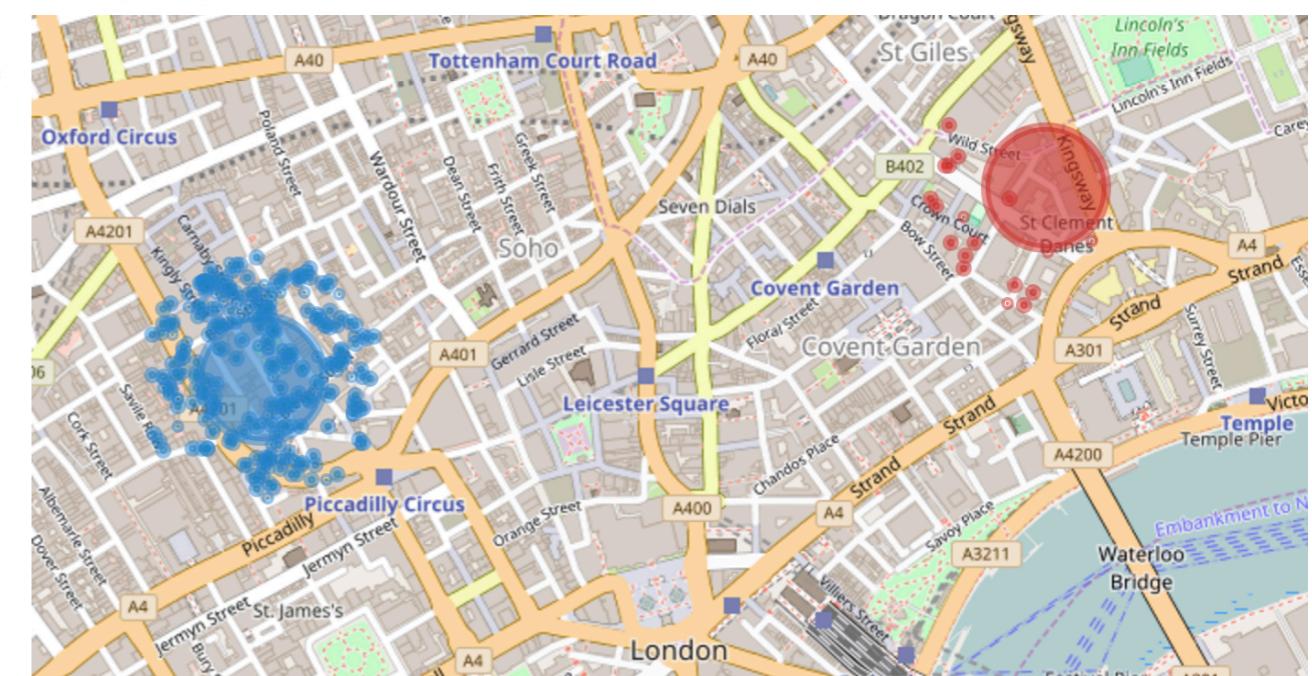
[example]

```
126 #####  
127 # Object Oriented Programming: Saves time and money!#  
128 #####  
129 import SpatialRetriever as SR  
130  
131 srAirbnb = SR.SpatialRetriever(airbnb_listings)  
132 srFoursquare = SR.SpatialRetriever(foursquare_venues)  
133  
134 print(srAirbnb.get_city_center())  
135 print(srFoursquare.get_city_center())  
136  
137 #Get the locations within 200 meters radius from the Foursquare city center  
138 fourquare_center = srFoursquare.get_city_center()  
139 nearby_locations = srFoursquare.get_nearby_locations(fourquare_center[0],  
140 ...     fourquare_center[1], 0.2, area_shape='circle')  
141  
142 fourquare_center = srAirbnb.get_city_center()  
143 nearby_locations = srAirbnb.get_nearby_locations(fourquare_center[0],  
144 ...     fourquare_center[1], 0.2, area_shape='circle')  
145
```

```

149 #####
150 # Visualise results using folium
151 #####
152 import folium
153
154 mymap = folium.Map(location=[city_center[0], city_center[1]], zoom_start=14)
155
156 #plot city center as a coloured circle
157 folium.CircleMarker(location=fourquare_center, radius=100,
158                      popup='Foursquare Centre', color='#3186cc',
159                      fill_color='#3186cc').add_to(mymap)
160
161 #plot city center as a coloured circle
162 folium.CircleMarker(location=airbnb_center, radius=100,
163                      popup='Airbnb Centre', color='#cc3131',
164                      fill_color='#cc3131').add_to(mymap)
165
166
167 for loc_id in nearby_locations_4sq:
168     folium.CircleMarker(srFoursquare.locationData[loc_id][0:2], radius = 5, color='#3186cc',
169                          fill_color='#2186ac').add_to(mymap)
170
171 for loc_id in nearby_locations_airbnb:
172     folium.CircleMarker(srAirbnb.locationData[loc_id][0:2], radius = 5, color='#cc3131',
173                          fill_color='#cc3131').add_to(mymap)
174
175 mymap.save('./graphs/MappingCityCenters.html')
176

```



folium:

<https://github.com/python-visualization/folium>

bounding box
counts [alternative]

Initialize a Grid to hold data

```
lonMin = -74.1 #minimum longitude
```

```
lonMax = -73.7
```

```
lonStep = 0.0025 #defines cell size
```

```
latMin = 40.6 #minimum latitude
```

```
latMax = 41.0
```

```
latStep = 0.0025 #defines cell size
```

```
latLen = int ((latMax - latMin) / latStep) +1 #number of cells on the y-axis
```

```
lonLen = int ((lonMax - lonMin) / lonStep) +1 #number of cells on the x-axis
```

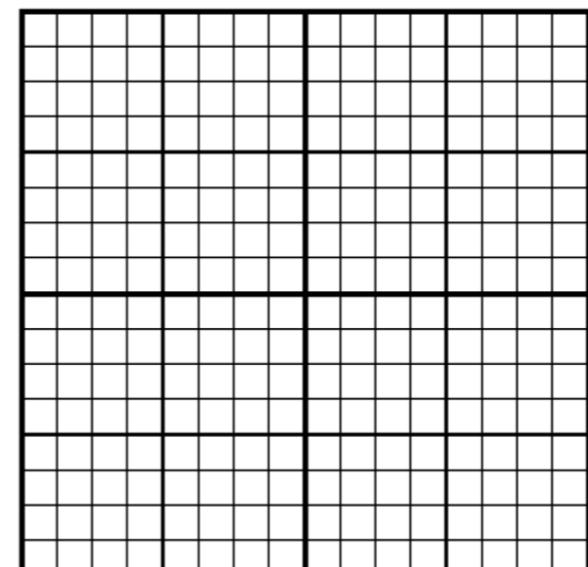
```
#Cell counts for each source of geo-data
```

```
FSQcellCount = np.zeros ((latLen, lonLen))
```

```
AIRcellCount = np.zeros ((latLen, lonLen))
```

```
TAXIcellCount = np.zeros ((latLen, lonLen))
```

GridCounts.py



Loading 3 geo-data layers

GridCounts.py

```
##### LOADING COORDS FROM 3 GEO LAYERS #####
airbnb_coords = []

with open('listings_sample.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        airbnb_coords.append([float(row['latitude']), float(row['longitude'])])

print 'Number of Airbnb listings: ' + str(len(airbnb_coords))

#reading Foursquare venue data #
foursquare_coords = []
with open('venue_data_4sq_newyork_anon.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        foursquare_coords.append([float(row['latitude']), float(row['longitude'])])

print 'Number of Foursquare listings: ' + str(len(foursquare_coords))

#reading Taxi journey coords #
taxi_coords = []
with open('taxi_trips.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        taxi_coords.append([float(row['latitude']), float(row['longitude'])])

print 'Number of Yellow Taxi journeys: ' + str(len(taxi_coords))
```

Count occurrences per cell

```
all_coords = [foursquare_coords, airbnb_coords, taxi_coords]
for i in range(0, 3):
    current_coords = all_coords[i]

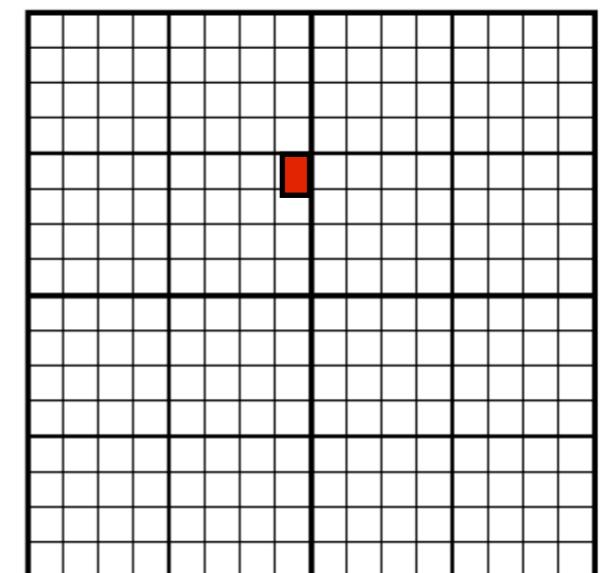
    for coords in current_coords:
        lat = coords[0]
        longit = coords[1]

        #if outside the grid then ignore point
        if (lat < latMin) or (lat > latMax) or (longit < lonMin) or (longit > lonMax):
            continue

        #identify cell where points belongs too
        cx = int ((longit - lonMin) / lonStep)
        cy = int((lat - latMin) / latStep)

        if i == 0:
            FSQcellCount[cy, cx] += 1
        elif i == 1:
            AIRcellCount[cy, cx] +=1
        else:
            TAXIcellCount[cy, cx] +=1
```

GridCounts.py

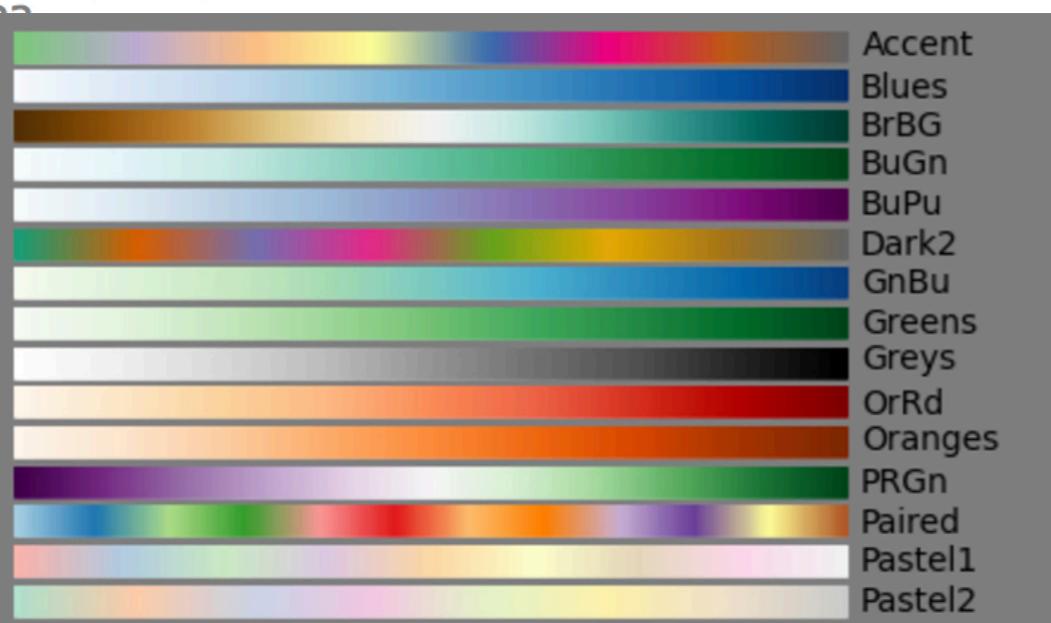


This is an archival dump of old wiki content --- see scipy.org for current material.

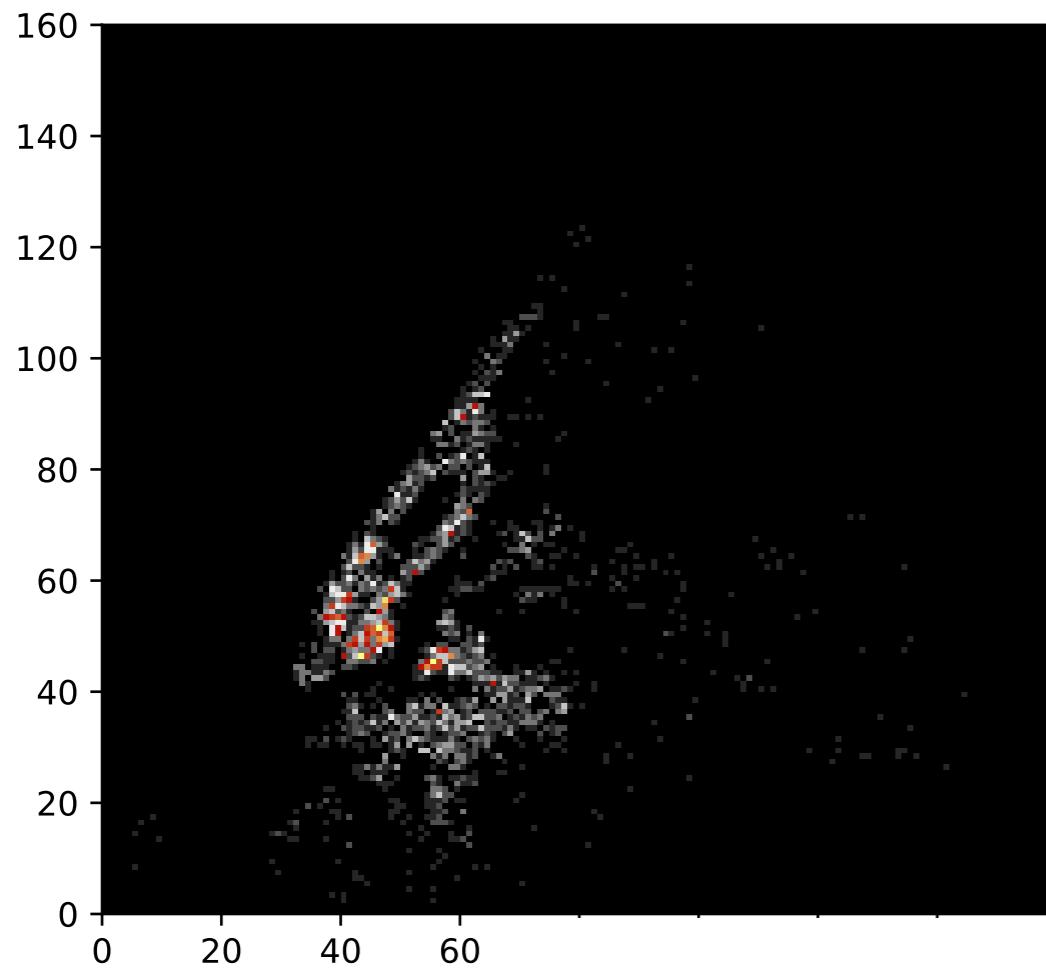
Please see <http://scipy-cookbook.readthedocs.org/>

SciPy: Cookbook/ Matplotlib/ Show_colormaps

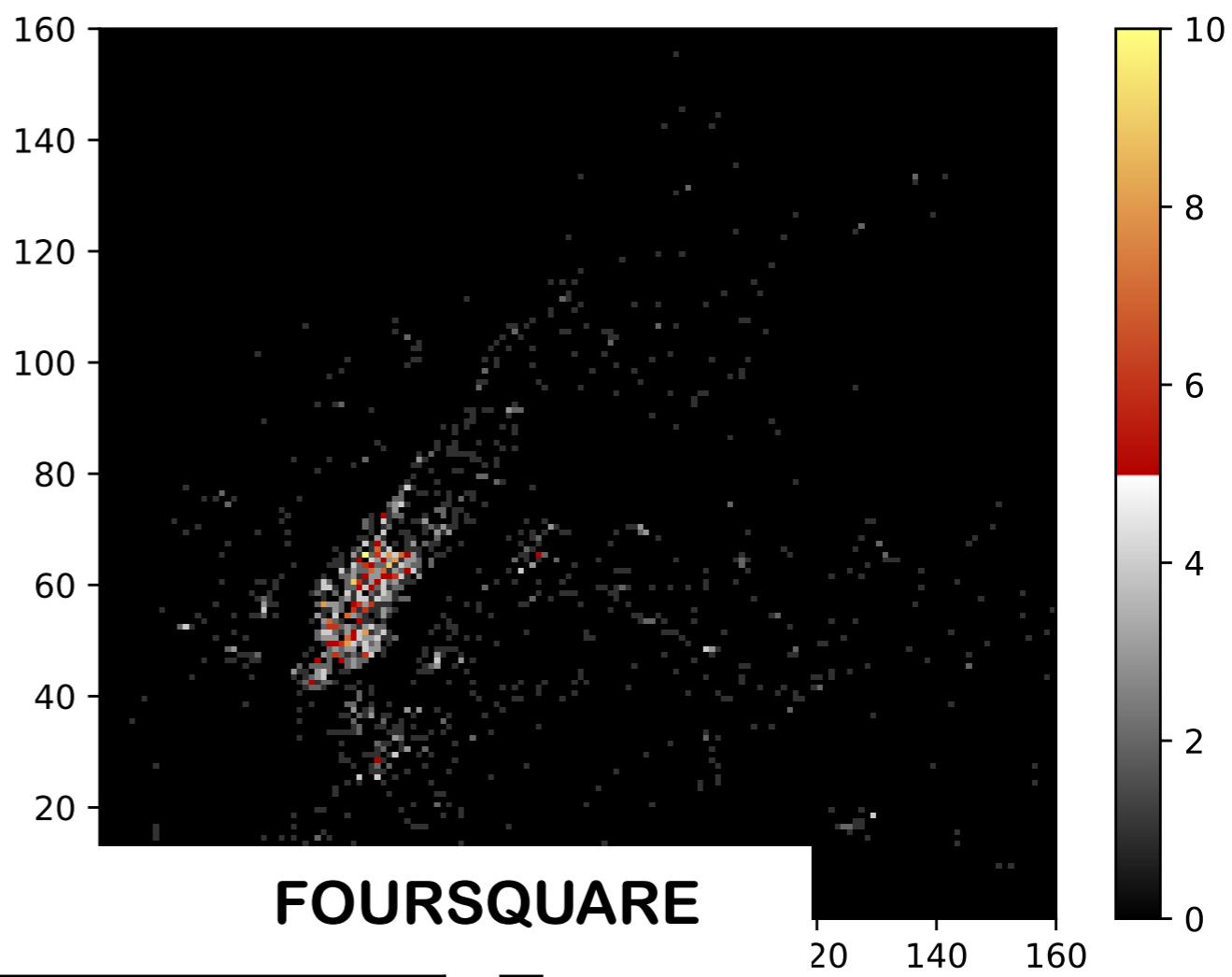
```
204 ##### PLOT MAPS #####
205
206 cdict = {'red': ((0.0, 0.0, 0.0),
207                 (0.5, 1.0, 0.7),
208                 (1.0, 1.0, 1.0)),
209             'green': ((0.0, 0.0, 0.0),
210                 (0.5, 1.0, 0.0),
211                 (1.0, 1.0, 1.0)),
212             'blue': ((0.0, 0.0, 0.0),
213                 (0.5, 1.0, 0.0),
214                 (1.0, 0.5, 1.0))}
215 mapIndex=0
216 for cellCount in [AIRcellCount,FSQcellCount,TAXIcellCount]:
217     my_cmap = plt.matplotlib.colors.LinearSegmentedColormap('my_colormap',cdict,256)
218     mapIndex +=1
219     plt.pcolor(cellCount,cmap=my_cmap)
220     plt.colorbar()
221     plt.savefig('map' + str(mapIndex) + '.pdf')
222     plt.close()
```



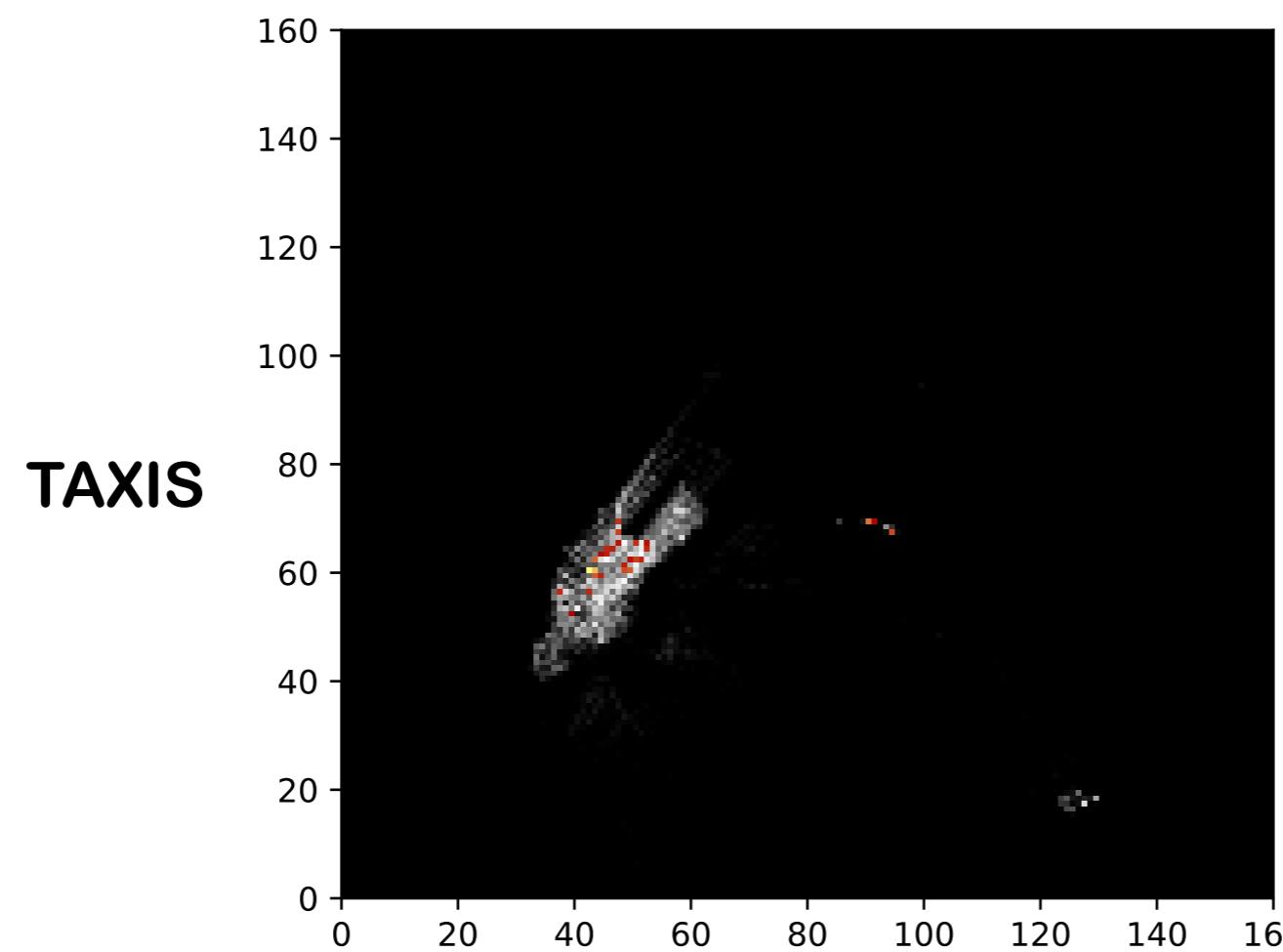
GridCounts.py



AIRBNB



FOURSQUARE



TAXIS

place networks

Loading Foursquare places data

```
### READ FOURSQUARE VENUES DATA ###
node_data = {}
with open('venue_data_4sq_newyork_anon.csv') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        latit = float(row['latitude'])
        longit = float(row['longitude'])
        place_title = row['title']
        node_id = int(row['vid'])
        node_data[node_id] = (place_title, latit, longit)
```

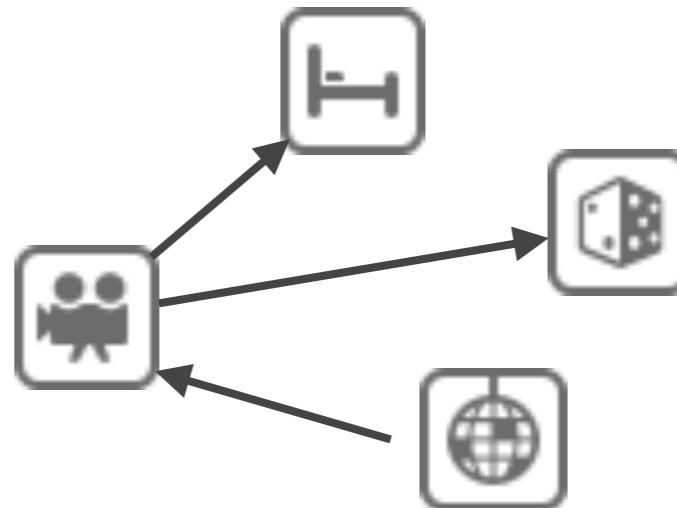
network_analysis.py

venue_data_4sq_newyork_anon.csv

Generating a Place Network

Let's load a New York's Network of Places : it's a directed graph with places as nodes.

Place Network



Connect two places if a user transition has been observed between them!

```
### LOAD NETWORK FROM FILE ###
nyc_net = nx.read_edgelist('newyork_net.txt',create_using=nx.DiGraph(), nodetype = int)

# some simple stats
N,K = nyc_net.order(), nyc_net.size()
avg_deg = float(K)/N

print "Loaded New York City Place network."
print "Nodes: ", N
print "Edges: ", K
print "Average degree: ", avg_deg
```

newyork_net.txt

network_analysis.py

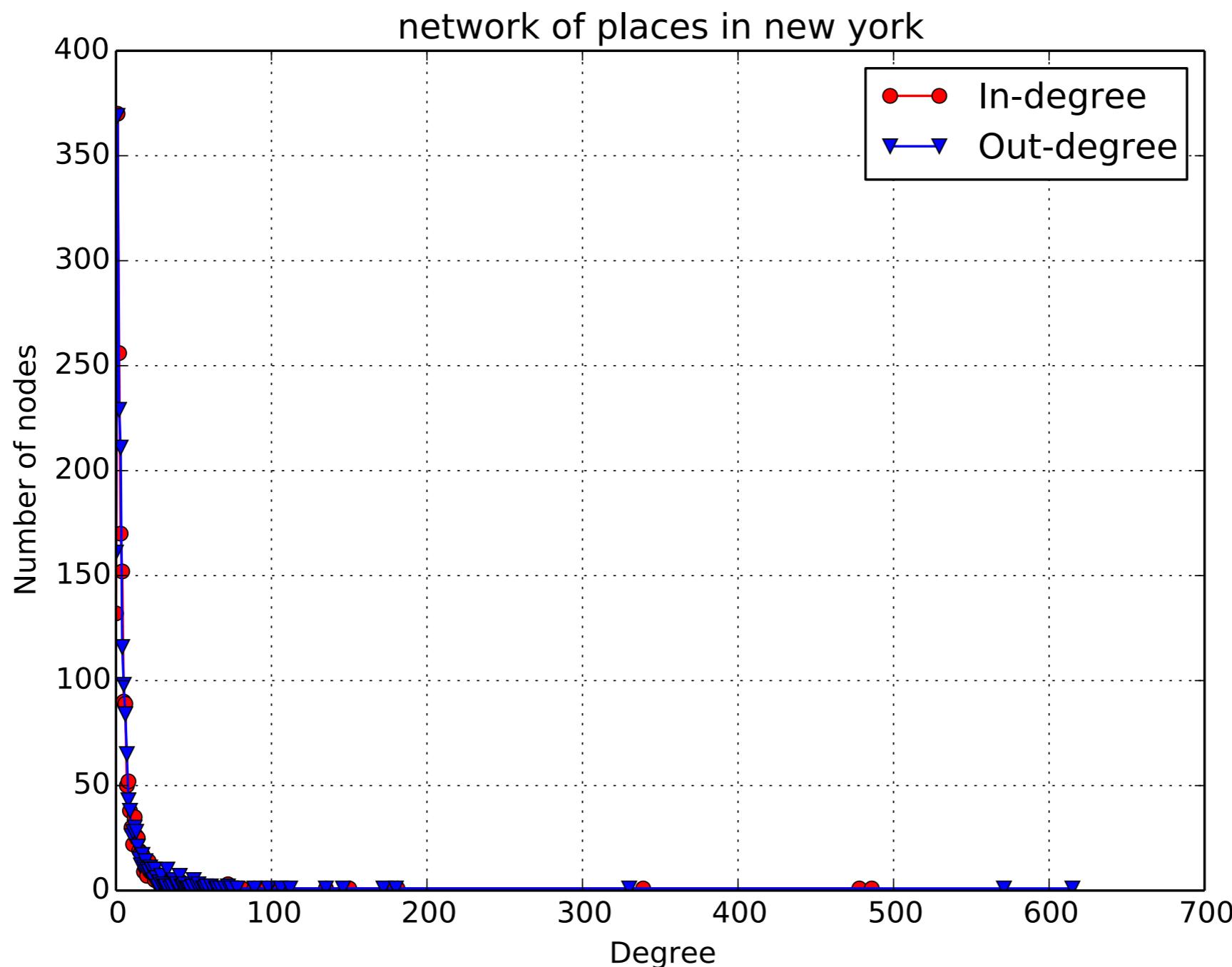
Getting Degree Distributions

```
in_degrees = nyc_net.in_degree() #Get degree distributions (in- and out-)
in_values = sorted(set(in_degrees.values()))
in_hist = [in_degrees.values().count(x) for x in in_values]
out_degrees = nyc_net.out_degree()
out_values = sorted(set(out_degrees.values()))
out_hist = [out_degrees.values().count(x) for x in out_values]

# plot degree distributions
logScale = True #set this True to plot data in logarithmic scale
plt.figure()
if logScale:
    plt.loglog(in_values,in_hist,'ro-') # red color with marker 'o'
    plt.loglog(out_values,out_hist,'bv-') # blue color with marker 'v'
else:
    plt.plot(in_values,in_hist,'ro-') # red color with marker 'o'
    plt.plot(out_values,out_hist,'bv-') # blue color with marker 'v'
plt.legend(['In-degree','Out-degree'])
plt.xlabel('Degree')
plt.ylabel('Number of nodes')
plt.title('network of places in new york')
plt.grid(True)
if logScale:
    plt.xlim([0,2*10**2])
    plt.savefig('nyc_net_degree_distribution_loglog.pdf')
else:
    plt.savefig('nyc_net_degree_distribution.pdf')
plt.close()
```

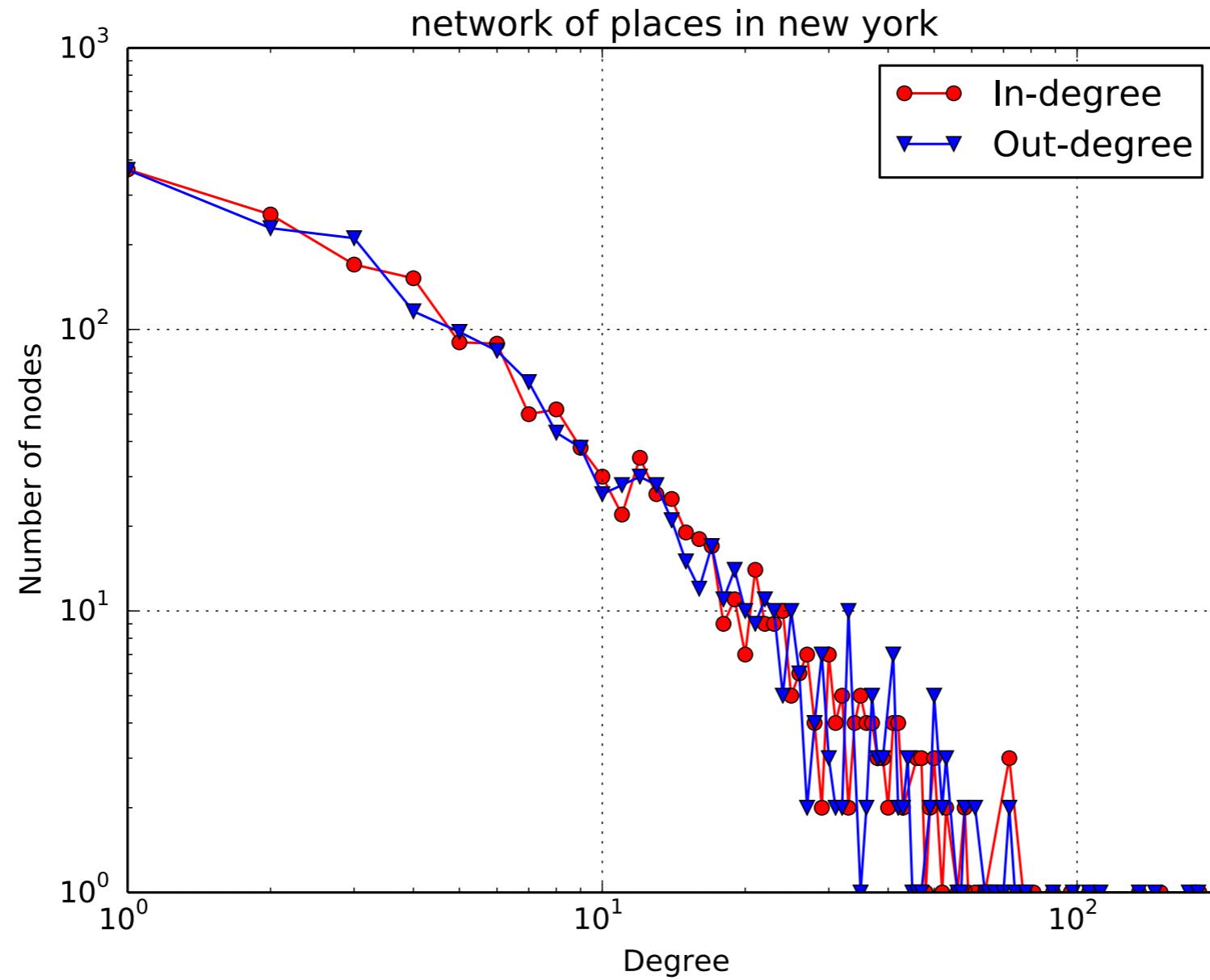
network_analysis.py

Degree Distribution



OOPS! We can't see much this way, can't we?

Logarithmic beauty!



Change scale of x,y axis by replacing
plt.plot(in_values,in_hist,'ro-') # in-degree
with
plt.loglog(in_values,in_hist,'ro-') # in-degree

Triadic Formations in Networks



We can get the clustering coefficient of individual nodes or of all the nodes (but the first we convert the graph to an undirected one).

```
# Symmetrize the graph for simplicity
nyc_net_ud = nyc_net.to_undirected()

# We are interested in the largest connected component
nyc_net_components = nx.connected_component_subgraphs(nyc_net_ud)
nyc_net_mc = nyc_net_components[0]

# Graph statistics for the main component
N_mc, K_mc = nyc_net_mc.order(), nyc_net_mc.size()
avg_deg_mc = float(2*K_mc)/N_mc
avg_clust = nx.average_clustering(nyc_net_mc)

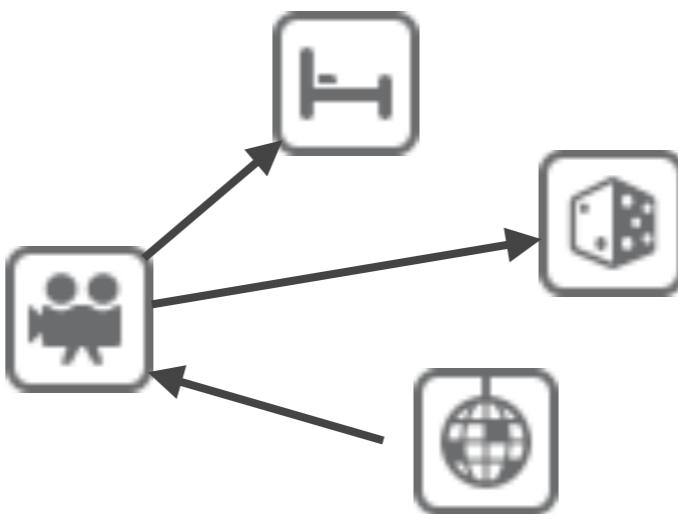
print ""
print "New York Place Network graph main component."
print "Nodes: ", N_mc
print "Edges: ", K_mc
print "Average degree: ", avg_deg_mc
print "Average clustering coefficient: ", avg_clust
```

network_analysis.py

Node Centralities

Now, we will extract the main connected component; then we will compute node centrality measures.

```
# Betweenness centrality  
bet_cen = nx.betweenness_centrality(nyc_net_mc)  
# Degree centrality  
deg_cen = nx.degree_centrality(nyc_net_mc)  
# Eigenvector centrality  
eig_cen = nx.eigenvector_centrality(nyc_net_mc)
```



network_analysis.py

Most Central Nodes

First we introduce a utility method: given a dictionary and a threshold parameter, the top-K elements of the dictionary are returned according to element values.

```
def getTopDictionaryKeys(dictionary,number):
    topList = []
    a = dict(dictionary)
    for i in range(0,number):
        m = max(a, key=a.get)
        topList.append([m,a[m]])
        del a[m]

    return topList
```

We can then apply the method on the various centrality methods available. Below we extract the top-10 most central nodes for each case.

```
top_bet_cen = getTopDictionaryKeys(bet_cen,10)

top_deg_cen = getTopDictionaryKeys(deg_cen,10)

top_eig_cen = getTopDictionaryKeys(eig_cen,10)
```

Interpretability Matters

```
print 'Top-10 places for betweenness centrality.'  
for [node_id, value] in top_bet_cen:  
    title = node_data[node_id][0]  
    print title  
    print '-----'  
  
print 'Top-10 places for degree centrality.'  
for [node_id, value] in top_deg_cen:  
    title = node_data[node_id][0]  
    print title  
    print '-----'  
  
print 'Top-10 places for eigenvector centrality.'  
for [node_id, value] in top_eig_cent:  
    title = node_data[node_id][0]  
    print title  
    print '-----'
```

Most Central Nodes

Betweenness Centrality

Top - 10
JFK Airport
LaGuardia Airport (LGA)
Madison Square Garden
Empire State Building
Frying Pan
The Garden at Studio Square
Chelsea Clearview Cinemas
230 Fifth Rooftop Lounge
33rd St PATH Station
Tiffany & Co.

Degree Centrality

Top - 10
JFK Airport
LaGuardia Airport (LGA)
Madison Square Garden
Empire State Building
Frying Pan
230 Fifth Rooftop Lounge
Chelsea Clearview Cinemas
The Garden at Studio Square
Tiffany & Co.
Central Park West

Eigenvector Centrality

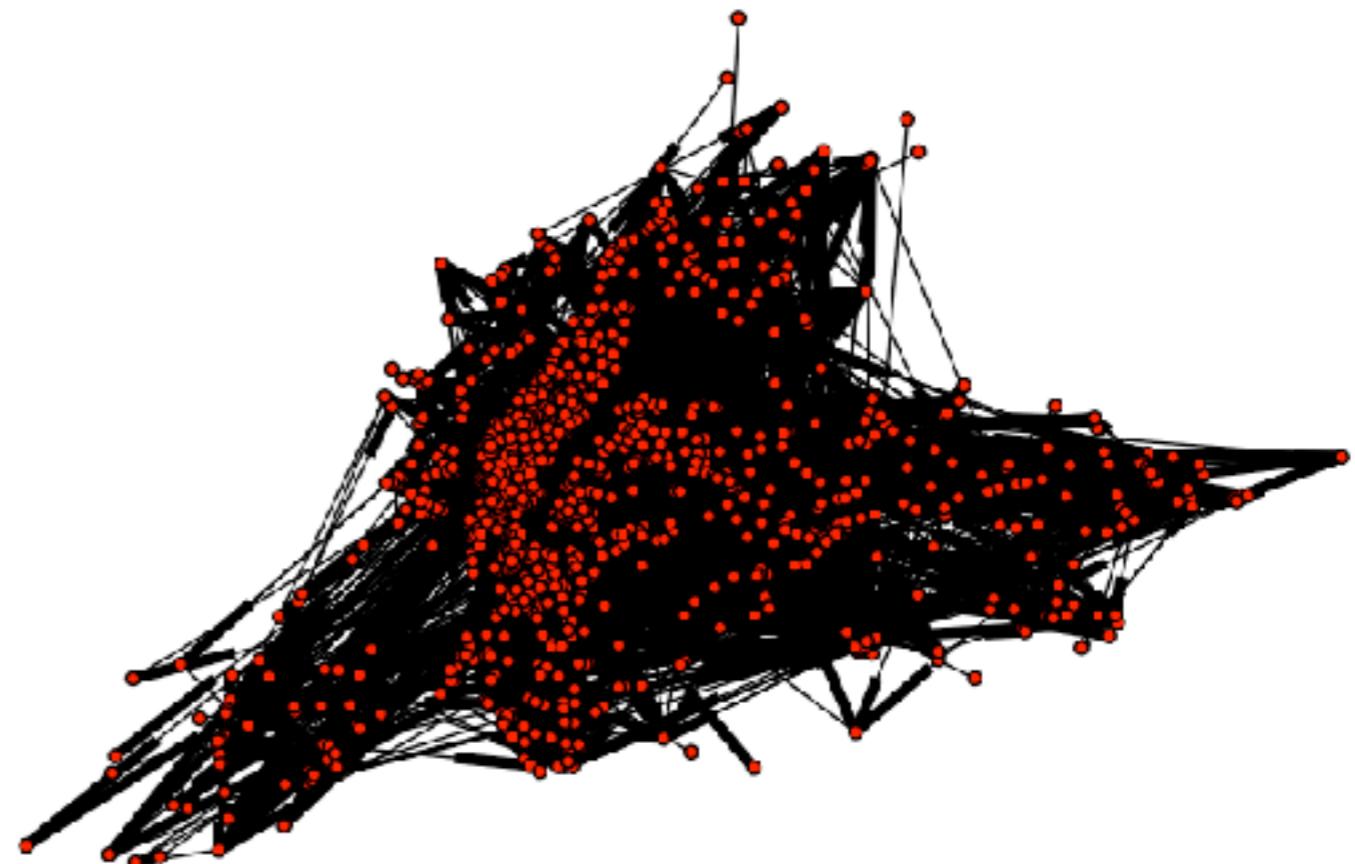
Top - 10
JFK Airport
LaGuardia Airport (LGA)
Madison Square Garden
Frying Pan
Empire State Building
230 Fifth Rooftop Lounge
Chelsea Clearview Cinemas
Tiffany & Co.
Central Park West
The Garden at Studio Square

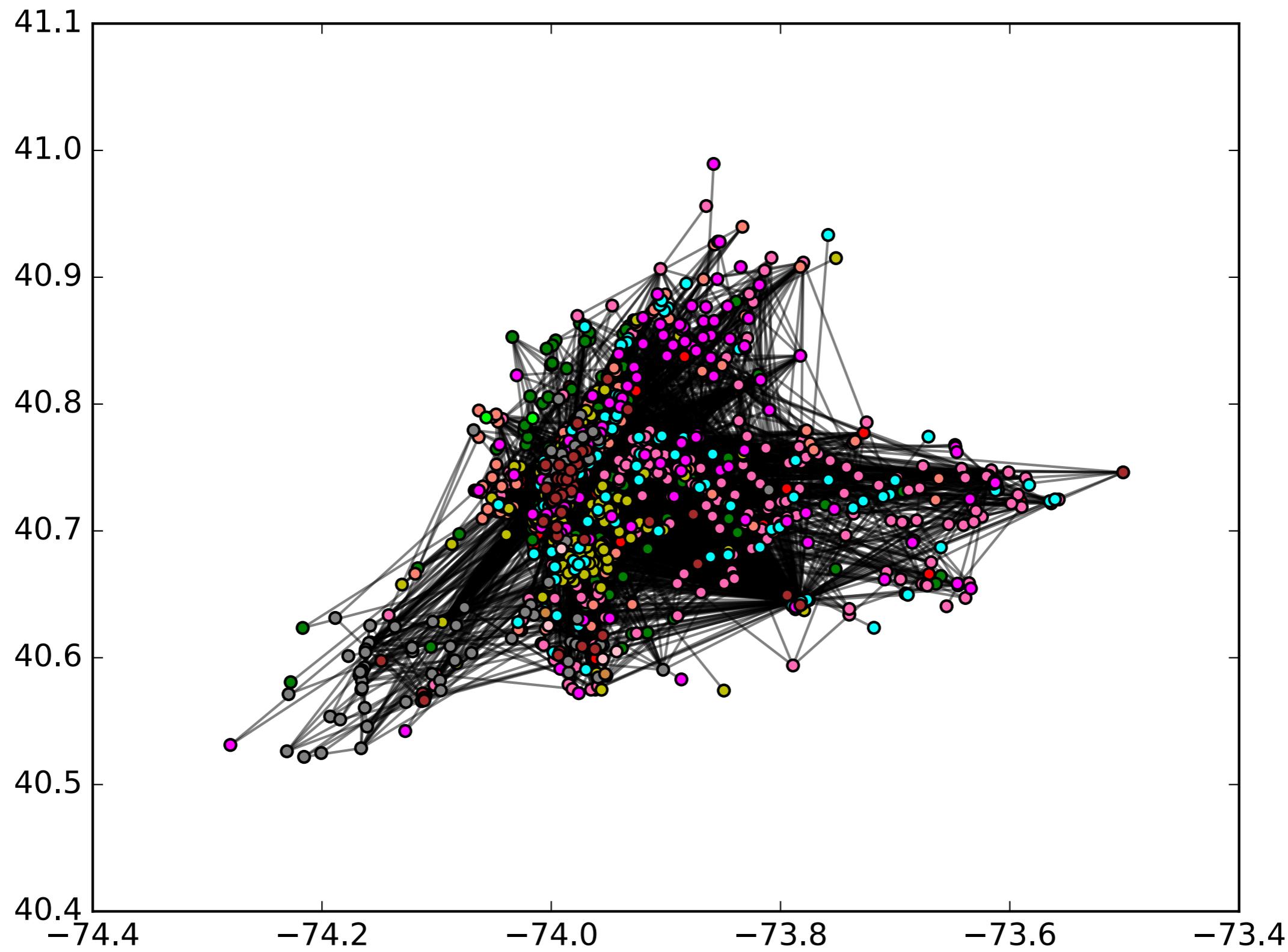
The ranking for the different centrality metrics does not change much, although this may well depend on the type of network under consideration. The results meet ones intuition (if they know New York), yet biases may also exist..

Drawing the Graph

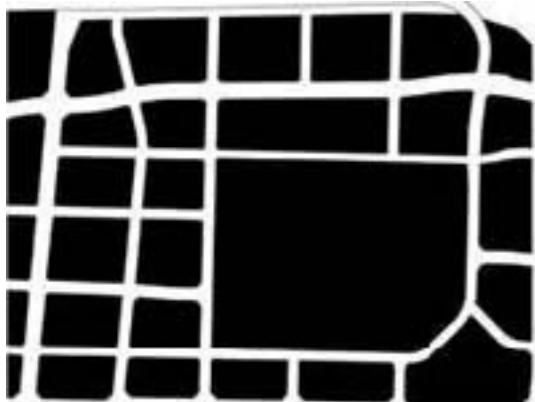
```
# draw the graph using information about nodes geographic positions
pos_dict = {}
for node_id, node_info in node_data.items():
    pos_dict[node_id] = (node_info[2], node_info[1])
nx.draw(nyc_net, pos=pos_dict, with_labels=False, node_size=20)
plt.savefig('nyc_net_graph.png')
plt.close()
```

network_analysis.py





OSMnx for Street Network Analysis



MISSISSAUGA



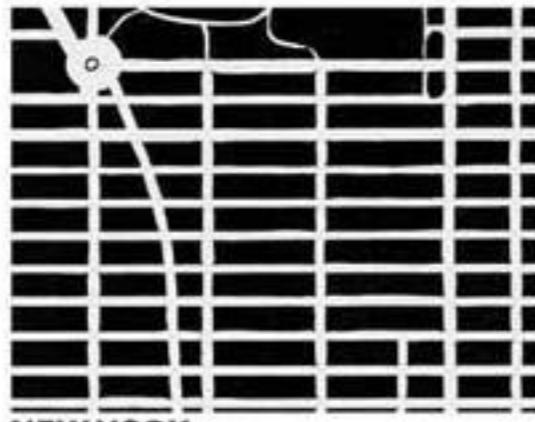
BARCELONA



COPENHAGEN



LONDON



NEW YORK



PARIS



ROME



SAN FRANCISCO



TORONTO

showcasing
deck.gl

Mapbox



Products Developers Industries Company Pricing



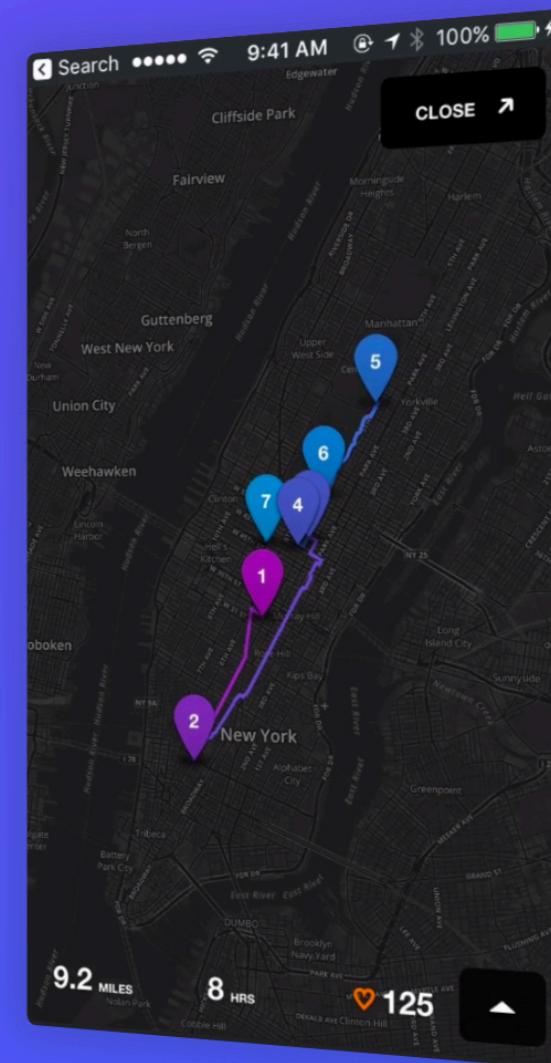
The location platform for developers and designers

APIs for maps, geocoding, driving directions, and more

[Get started](#)

or [contact sales](#)

Supports



Mapbox: Gallery



<https://www.mapbox.com/gallery/#map-6>

Mapbox: Studio

[Products](#)[Developers](#)[Industries](#)[Company](#)[Pricing](#)[Account](#)[Mapbox Studio](#)[Help](#)

tnoulas

[Sign out](#)

MAPBOX STUDIO

Design custom maps that fit seamlessly in your application.

[Get started →](#)

Design for your brand

Design on top of seven Mapbox template styles



Professional cartography

A point-and-click interface built for designers



Custom data

Import or create your own custom data layer

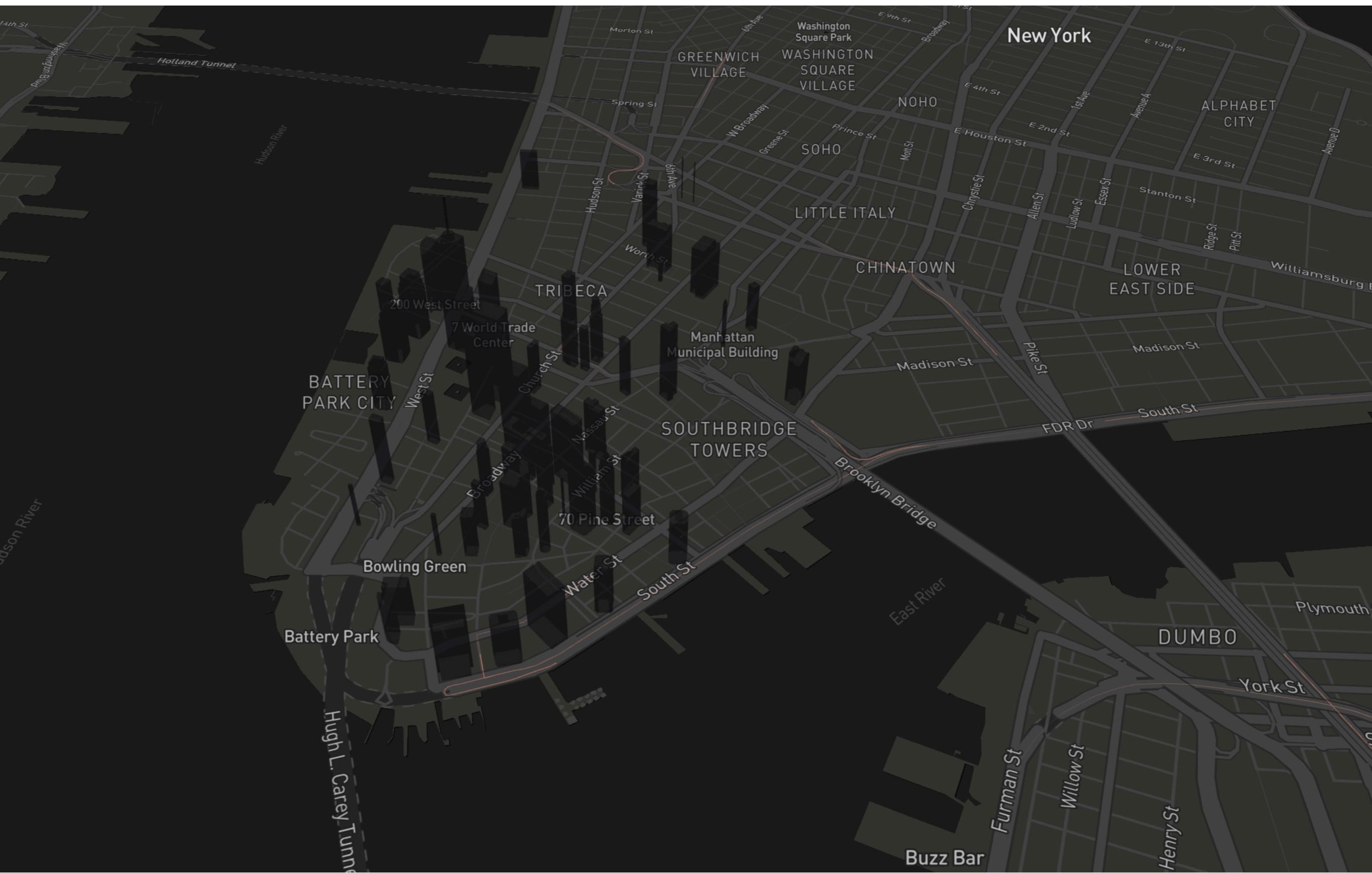


Cross-platform integration

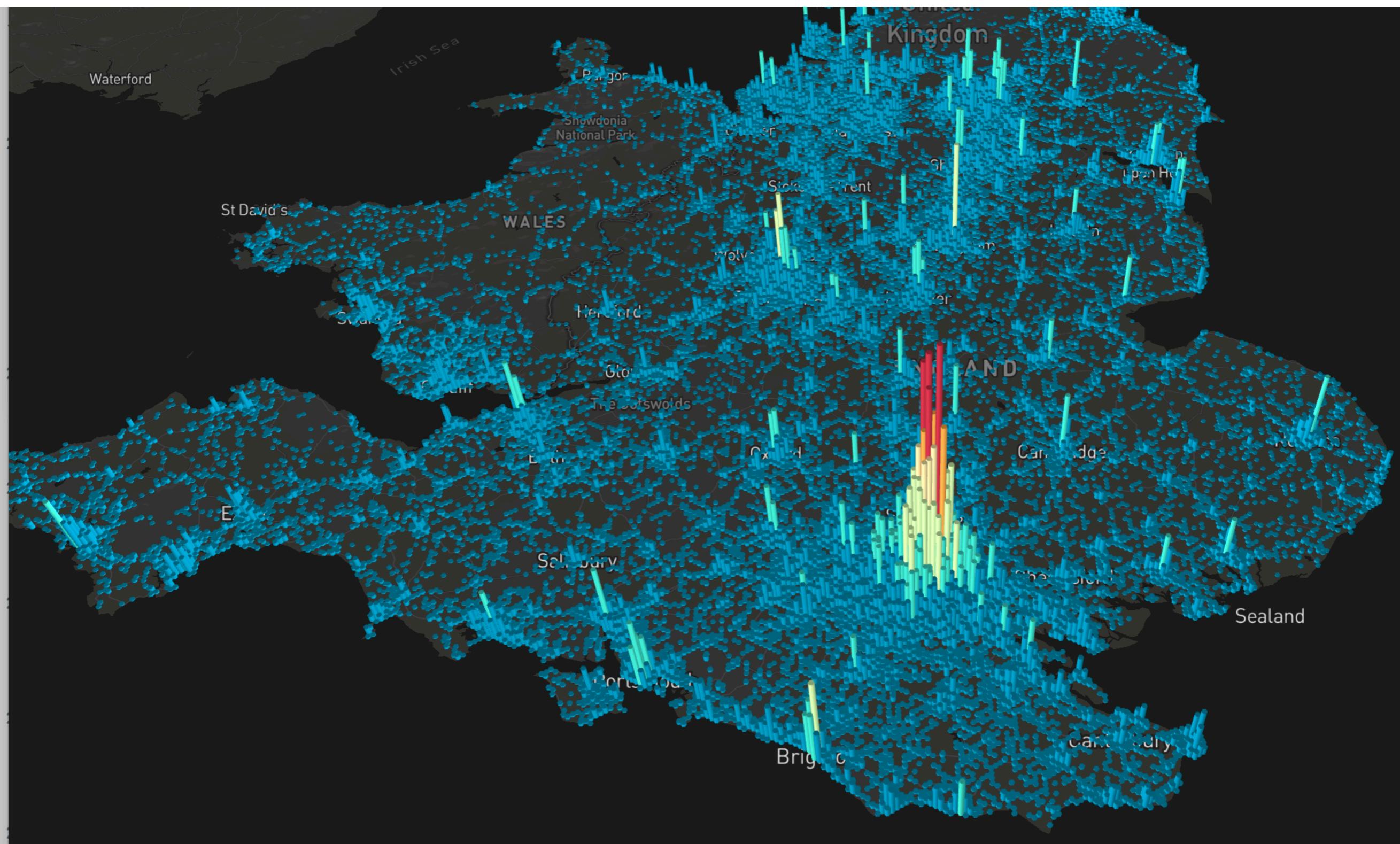
Integrate your map into mobile or web apps with our SDKs

Get an account - Create your own maps

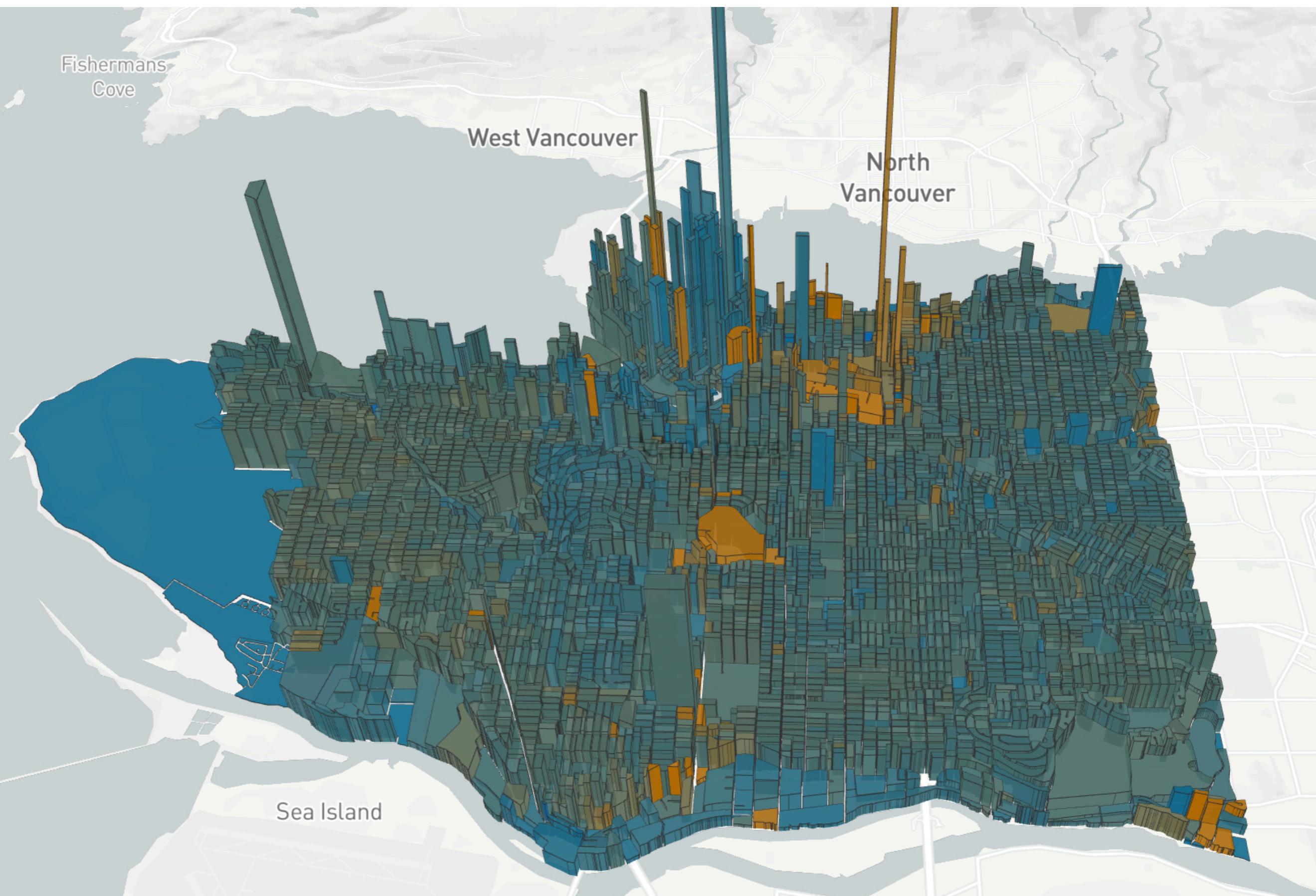
deck.gl :: trips



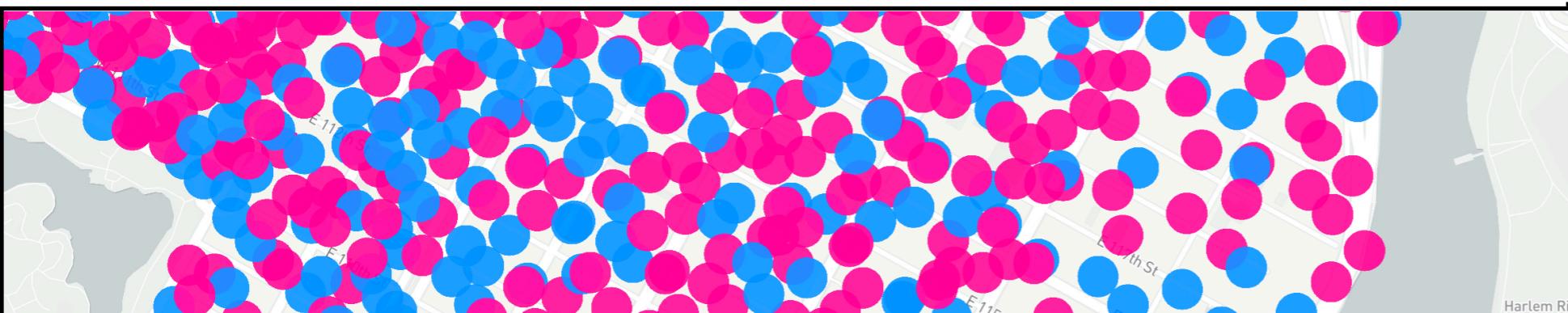
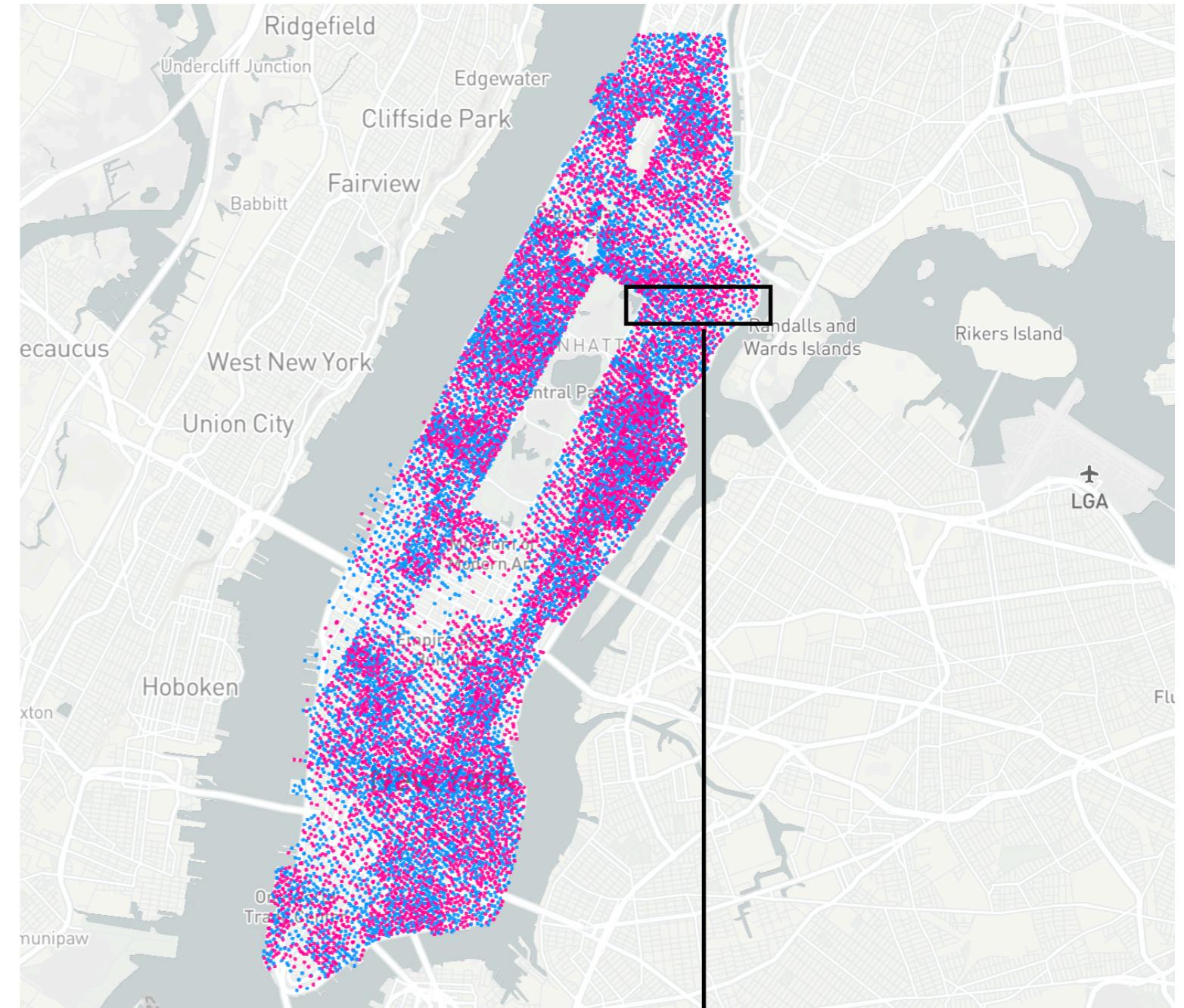
deck.gl :: 3D - heatmaps



deck.gl :: 3D - geojsons



deck.gl :: scatter plot



Visualising real estate data with Bokeh and Datashader

Welcome to Bokeh

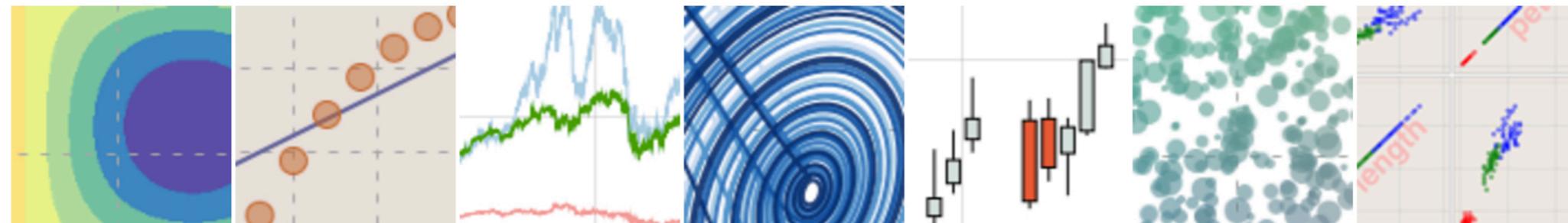
Bokeh is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, and to extend this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications.

To get started using Bokeh to make your visualizations, see the [User Guide](#).

To see examples of how you might use Bokeh with your own data, check out the [Gallery](#).

A complete API reference of Bokeh is at [Reference Guide](#).

If you are interested in contributing to Bokeh, or extending the library, see the [Developer Guide](#).

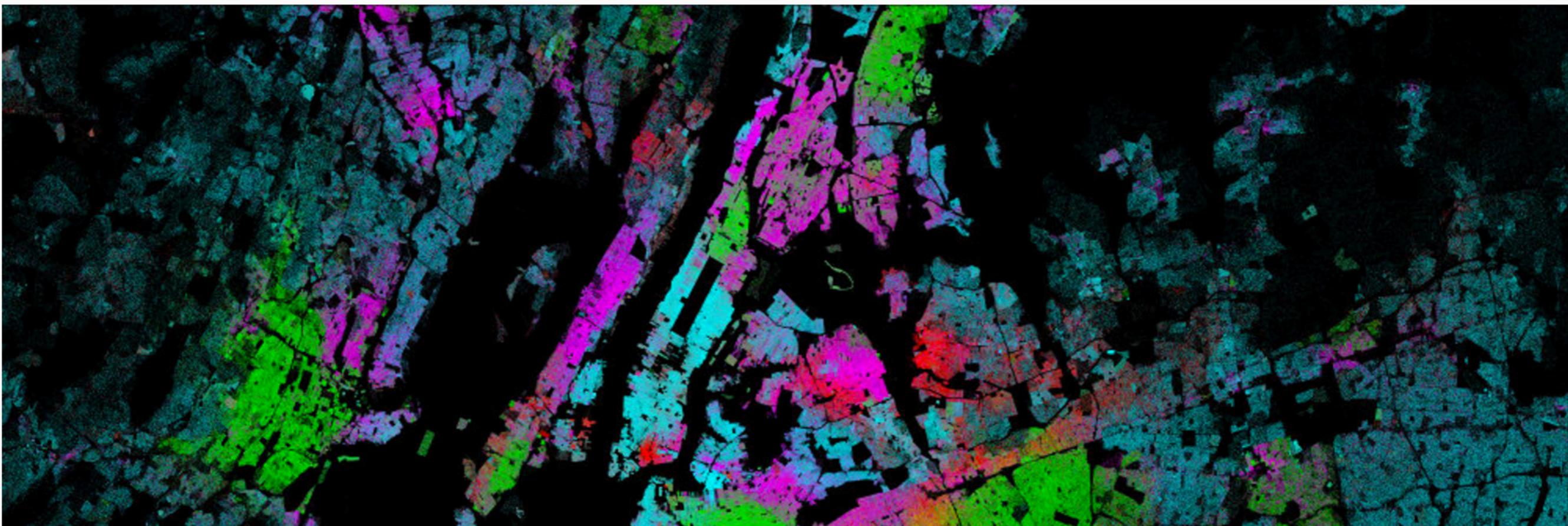


Datashader

Datashader is a graphics pipeline system for creating meaningful representations of large datasets quickly and flexibly. Datashader breaks the creation of images into a series of explicit steps that allow computations to be done on intermediate representations. This approach allows accurate and effective visualizations to be produced automatically, and also makes it simple for data scientists to focus on particular data and relationships of interest in a principled way. Using highly optimized rendering routines written in Python but compiled to machine code using [Numba](#), datashader makes it practical to work with extremely large datasets even on standard hardware.

To make it concrete, here's an example of what datashader code looks like:

```
>>> import datashader as ds
>>> import datashader.transfer_functions as tf
>>> import pandas as pd
>>> df = pd.read_csv('user_data.csv')
```



London sales dataset

London Year_2013-2014.csv *

```
1 transaction_id,price,date_processed,post_code,postcode,property_type,whether_newbuild,tenure,address2,address4,town,local_authority,co  
unty,record_status,year,month,quarter,house_flat,statsward,oa11,lsoa11,msoa11,inner_outer,year_month,Postcode_sector,Postcode_district  
,Ward14,ward_code,borough_code,borough_name  
2 {0FF377DF-DC87-4D54-90DC-5E200B155704},215000,01-Jan-13,HA0 4EE,HA04EE,F,N,L, , ,WEMBLEY,BRENT,GREATER  
LONDON,A,2013,1,1,2,00AEGJ,E00002335,E01000474,E02000118,Outer,2013/1,HA04,HA0,E05000085,E05000085,E09000005,Brent  
3 {76742B18-E591-4C1C-AACA-9D9BA3C067AB},295000,01-Jan-13,N16 7BQ,N167BQ,F,N,L, , ,LONDON,HACKNEY,GREATER  
LONDON,A,2013,1,1,2,00AMGH,E00008824,E01001764,E02000354,Inner,2013/1,N167,N16,E05009373,E05000238,E09000012,Hackney  
4 {4D71DE37-9898-4983-AC71-7A464FBB0878},740000,01-Jan-13,SW17 7HH,SW177HH,F,N,L,FLAT 1, ,LONDON,WANDSWORTH,GREATER  
LONDON,A,2013,1,1,2,00BJGH,E00022892,E01004549,E02000949,Inner,2013/1,SW177,SW17,E05000618,E05000618,E09000032,Wandsworth  
5 {F850DE7C-D297-4360-82B5-4E4DDC149E55},225000,02-Jan-13,BR1 2FE,BR12FE,F,N,L,FLAT 7, ,BROMLEY,BROMLEY,GREATER  
LONDON,A,2013,1,1,2,00AFGD,E00166105,E01000649,E02000133,Outer,2013/1,BR12,BR1,E05000106,E05000106,E09000006,Bromley  
6 {38152E09-DD38-4CC8-B58B-E3AE45EFA543},380000,02-Jan-13,BR2 8QW,BR28QW,S,N,F, , ,BROMLEY,BROMLEY,GREATER  
LONDON,A,2013,1,1,1,00AFGF,E00003224,E01000667,E02000159,Outer,2013/1,BR28,BR2,E05000108,E05000108,E09000006,Bromley  
7 {42AD92B3-D870-42A0-B476-DE4C8A0C1149},184000,02-Jan-13,BR5 4BX,BR54BX,T,N,F, , ,ORPINGTON,BROMLEY,GREATER  
LONDON,A,2013,1,1,1,00AFGM,E00003521,E01000724,E02000145,Outer,2013/1,BR54,BR5,E05000114,E05000114,E09000006,Bromley  
8 {0382446A-C5C7-4827-A0A8-F1B2903C5754},315000,02-Jan-13,BR6 7EY,BR67EY,D,N,F, , ,ORPINGTON,BROMLEY,GREATER  
LONDON,A,2013,1,1,1,00AFGQ,E00003657,E01000749,E02000162,Outer,2013/1,BR67,BR6,E05000117,E05000117,E09000006,Bromley  
9 {20D7852C-BE68-4911-8D54-B0C420200A75},225000,02-Jan-13,E10 5DL,E105DL,T,N,F, , ,LONDON,WALTHAM FOREST,GREATER  
LONDON,A,2013,1,1,1,00BHGM,E00022294,E01004432,E02000917,Outer,2013/1,E105,E10,E05000604,E05000604,E09000031,Waltham Forest  
10 {33BE2D6F-FA4C-4BE7-A48D-F35763B1C132},1375000,02-Jan-13,E14 9HL,E149HL,F,Y,L,FLAT 3701, ,LONDON,TOWER HAMLETS,GREATER  
LONDON,A,2013,1,1,2,00BGGG,E00167152,E01032771,E02006854,Inner,2013/1,E149,E14,E05009323,E05000583,E09000030,Tower Hamlets  
11 {24F4F8B4-FC3B-49E8-ACDF-7FC71B3197EE},275000,02-Jan-13,E14 9LL,E149LL,F,N,L,FLAT 47, ,LONDON,TOWER HAMLETS,GREATER  
LONDON,A,2013,1,1,2,00BGGG,E00167112,E01032770,E02006853,Inner,2013/1,E149,E14,E05009323,E05000583,E09000030,Tower Hamlets  
12 {1079ABFD-4D88-4E81-AEE8-6AD40B433331},245000,02-Jan-13,E16 1FH,E161FH,F,Y,L,FLAT M07, ,LONDON,NEWHAM,GREATER  
LONDON,A,2013,1,1,2,00BBGE,E00175044,E01033576,E02000747,Inner,2013/1,E161,E16,E05000478,E05000478,E09000025,Newham  
13 {EE004A0B-961F-4C42-A9E5-A0F44DF3090E},170000,02-Jan-13,IG1 1FS,IG11FS,F,Y,L,FLAT 7, ,ILFORD,REDBRIDGE,GREATER  
LONDON,A,2013,1,1,2,00BCGE,E00018572,E01033591,E02000780,Outer,2013/1,IG11,IG1,E05000501,E05000501,E09000026,Redbridge  
14 {7106CC07-5310-4780-8A9C-00EE230829FF},205000,02-Jan-13,IG2 7SA,IG27SA,T,N,F, , ,ILFORD,REDBRIDGE,GREATER  
LONDON,A,2013,1,1,1,00BCFY,E00018350,E01003640,E02000763,Outer,2013/1,IG27,IG2,E05000495,E05000495,E09000026,Redbridge
```

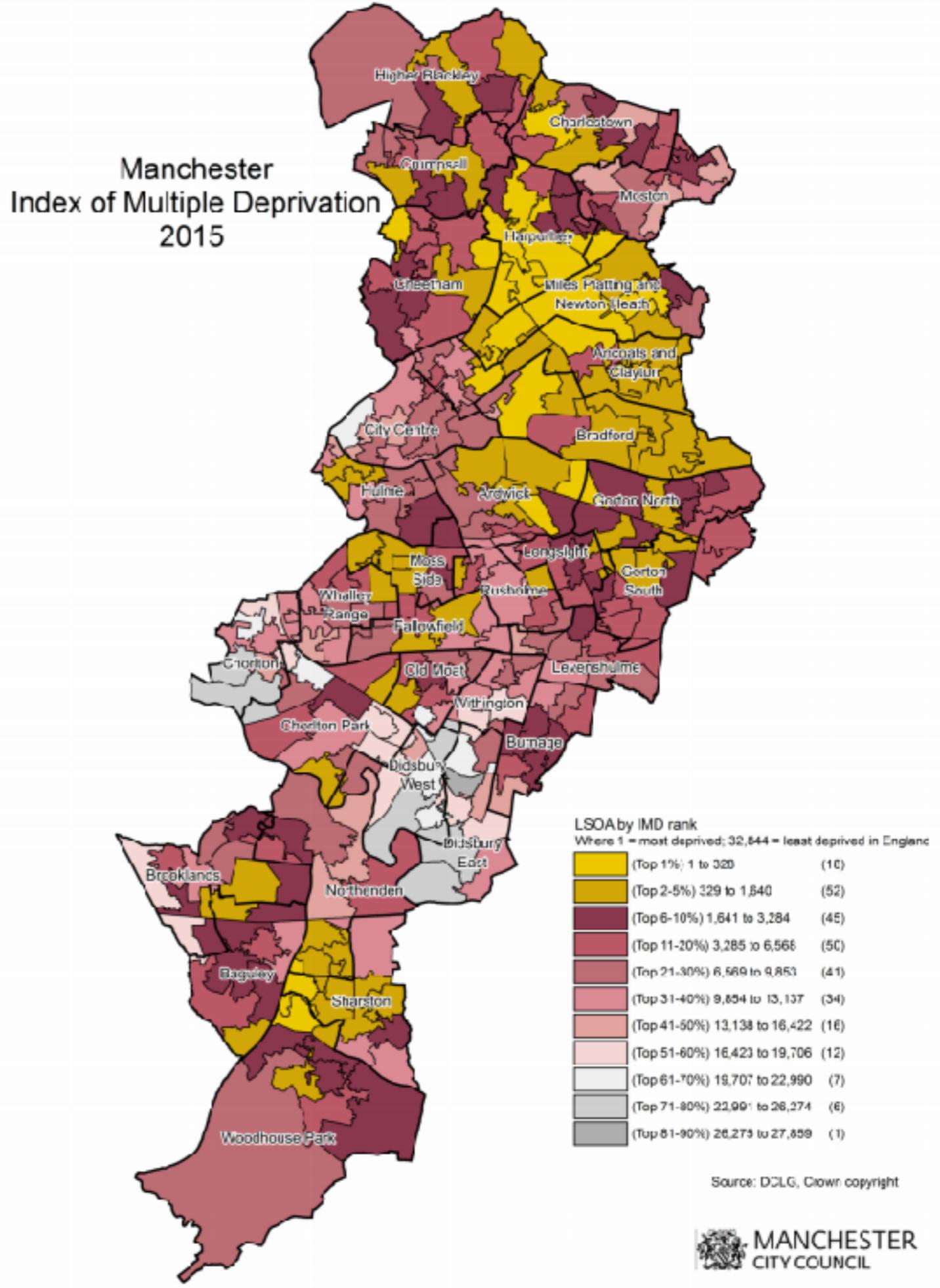
<https://data.london.gov.uk/dataset/average-house-prices-borough/resource/5507b84e-bf36-4388-b770-897915550e80>

LSOAs

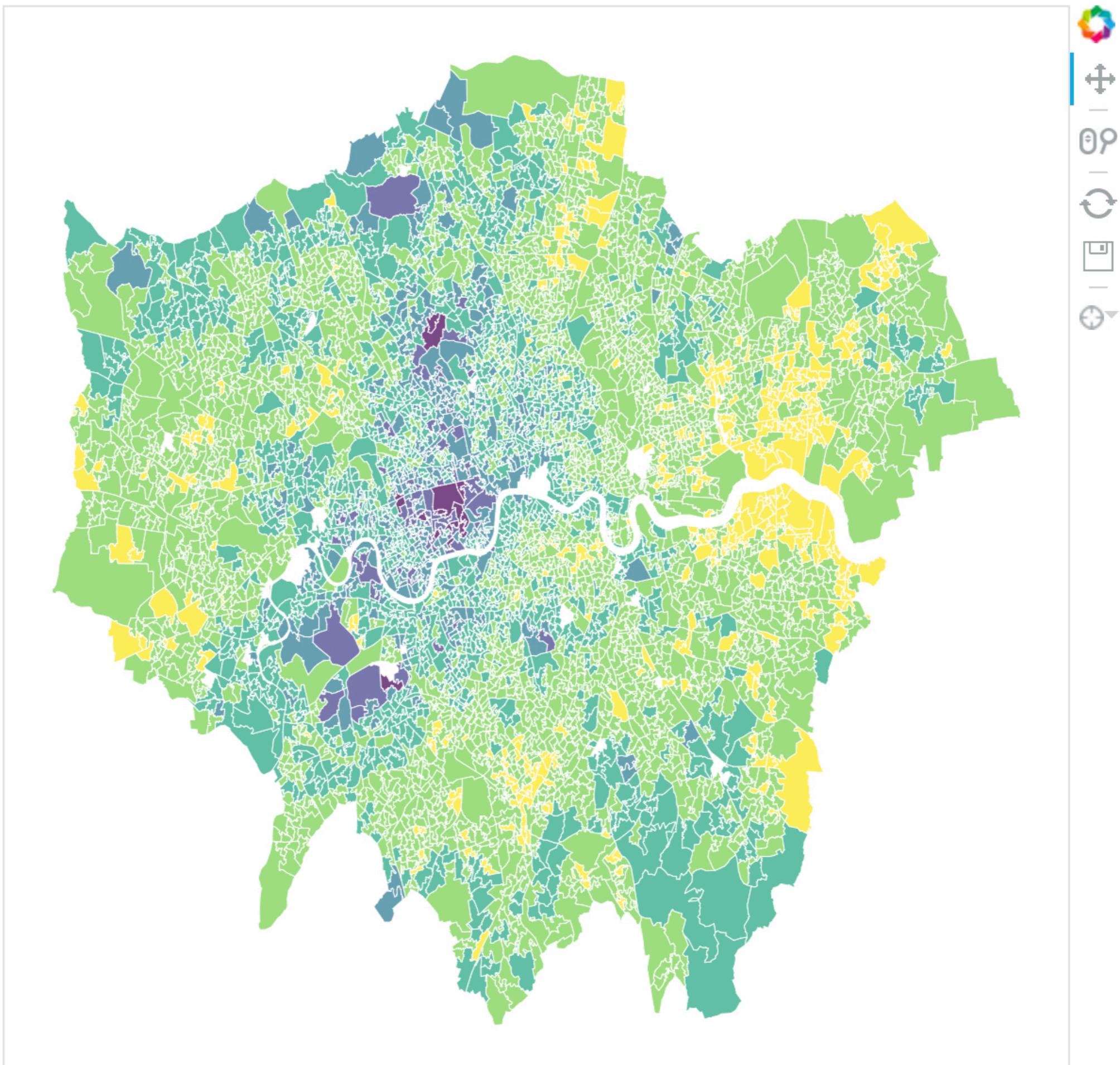
Manchester
Index of Multiple Deprivation
2015

Government based admin
boundaries

Approximate 1500 people
in each area.



London Median House Prices, 2013-2014



```
24
25 class LondonSalesMapper:
26
27     def __init__(self):
28         pass
29
30
31     def load_lsoa_polygons(self):
32         polygon_data = pygeoj.load(filepath="LondonLSOA.geojson")
33         spatial_entity_to_coordinates = {}
34
35         for feature in polygon_data:
36             coords = feature.geometry.coordinates
37             coords = coords[0][0]
38             lsoa_id = feature.properties['LSOA11CD']
39
40             xs = [i for i,j in coords]
41             ys = [j for i,j in coords]
42             spatial_entity_to_coordinates[lsoa_id] = [xs,ys]
43
44     return spatial_entity_to_coordinates
45
```

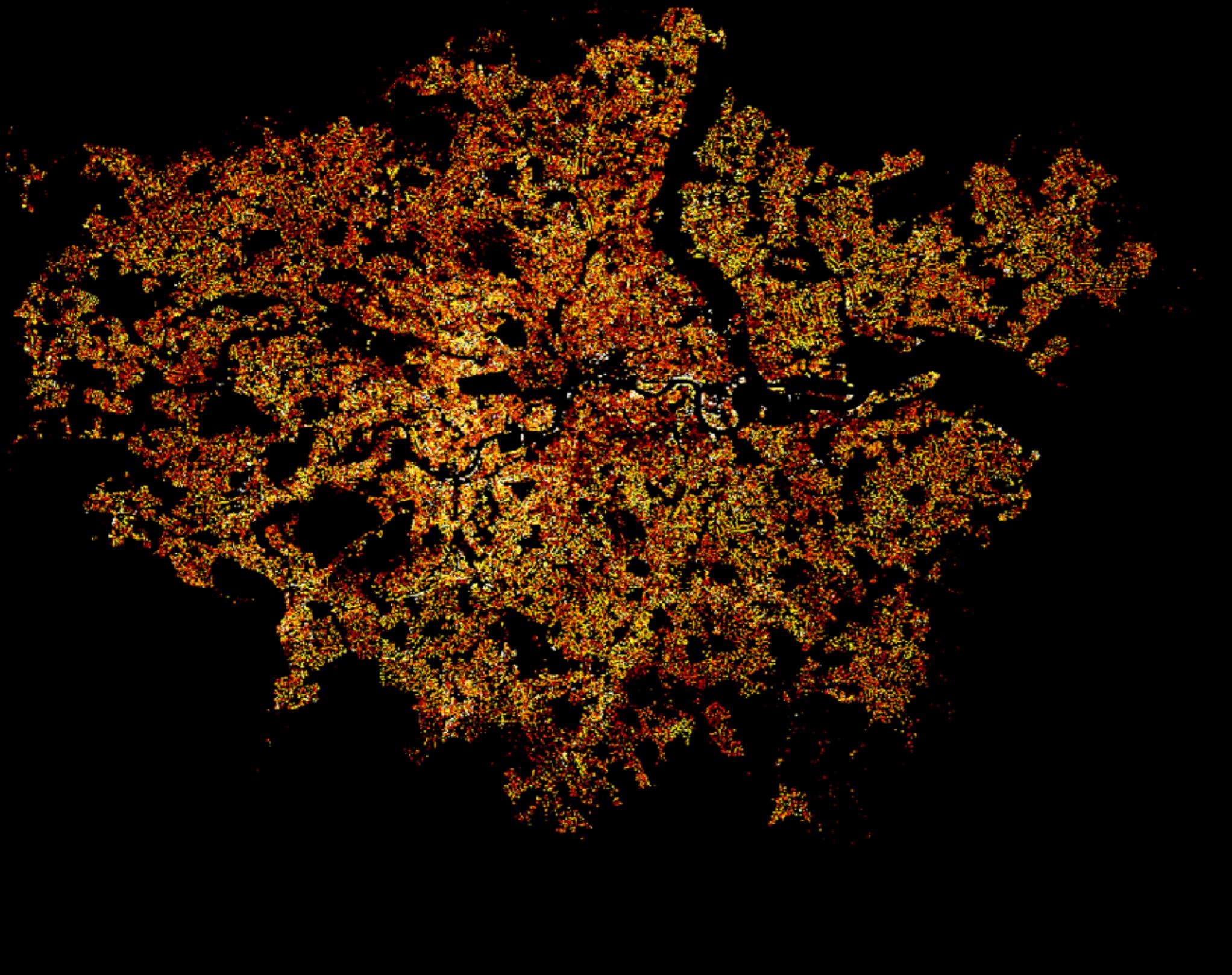
LondonSalesMapper.py

```
47
48     def load_lsoa_price_values(self):
49         entity_to_prices = {}
50
51         with open('london_sales_2013_2014.csv', 'rb') as csvfile:
52             reader = csv.DictReader(csvfile)
53             for row in reader:
54                 try:
55                     lsoa_id = row['lsoa11']
56                     price = float(row['price'])
57                     date = row['year_month']
58                     entity_to_prices.setdefault(lsoa_id, [])
59                     entity_to_prices[lsoa_id].append(price)
60                 except:
61                     continue
62
63         #average
64         for entity, all_prices in entity_to_prices.items():
65             entity_to_prices[entity] = np.median(all_prices)
66
67     return entity_to_prices
68
```

```
113
114 def plot_bokeh_intensity_map(self, spatial_entity_to_coordinates, spatial_entity_to_values):
115
116     entity_xs = []
117     entity_ys = []
118     entity_names = []
119     entity_rates = []
120     for name, coords in spatial_entity_to_coordinates.items():
121         xs = [i for i in coords[0]]
122         ys = [i for i in coords[1]]
123         try:
124             intensity_value = np.median(spatial_entity_to_values[name])
125         except:
126             intensity_value = 0.0
127
128         entity_xs.append(xs)
129         entity_ys.append(ys)
130         entity_names.append(name)
131         entity_rates.append(intensity_value)
132
133     palette.reverse()
134     color_mapper = LogColorMapper(palette=palette)
135
136
137     source = ColumnDataSource(data=dict(
138         x=entity_xs,
139         y=entity_ys,
140         name=entity_names,
141         rate=entity_rates,
142     ))
143
144     TOOLS = "pan,wheel_zoom,reset,hover,save"
```

LondonSalesMapper.py

```
143
144     TOOLS = "pan,wheel_zoom,reset,hover,save"
145
146     p = figure(
147         title="London Median House Prices, 2013–2014", tools=TOOLS,
148         x_axis_location=None, y_axis_location=None
149     )
150     p.grid.grid_line_color = None
151
152     p.patches('x', 'y', source=source,
153                fill_color={'field': 'rate', 'transform': color_mapper},
154                fill_alpha=0.7, line_color="white", line_width=0.5)
155
156     hover = p.select_one(HoverTool)
157     hover.point_policy = "follow_mouse"
158     hover.tooltips = [
159         ("Name", "@name"),
160         ("Price change rate)", "@rate%"),
161         ("(Long, Lat)", "($x, $y)"),
162     ]
163
164     show(p)
165
```

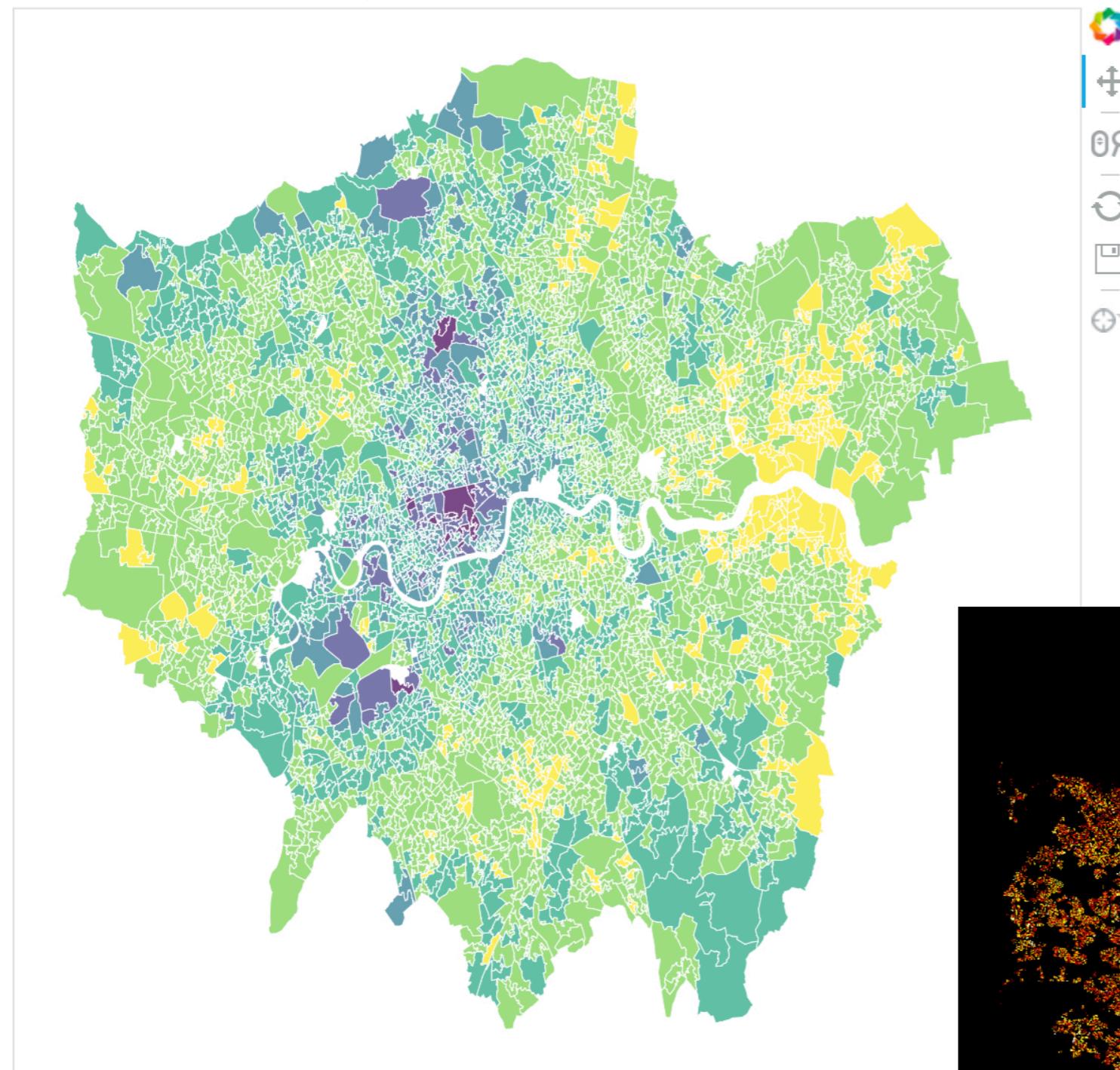


LondonSalesMapper.py

```
71
72     def plot_sales_mapstogram(self, year=None):
73
74         #define a bounding box around London
75         london = [-0.535583, 51.73, 0.334, 51.23 , 'london']
76
77         #read postcode frequency data: y_col,x_col,year: 51.464568,0.161226,1995
78         df = pd.read_csv('postcode_freq_data.csv')
79
80         city = london
81         maxLong = city[2]
82         minLong = city[0]
83         maxLat = city[1]
84         minLat = city[3]
85
86         #filter out "noisy" data
87         if year == None:
88             df = df.loc[(df['x_col'] < maxLong) & (df['x_col'] > minLong) & (df['y_col'] > minLat)]
89         else:
90             #consider annual filtering
91             df = df.loc[(df['x_col'] < maxLong) & (df['x_col'] > minLong) & (df['y_col'] > minLat)]
92
93         print (df.tail())
94
95         LDN = x_range, y_range = ((minLong,maxLong), (minLat, maxLat))
96
```

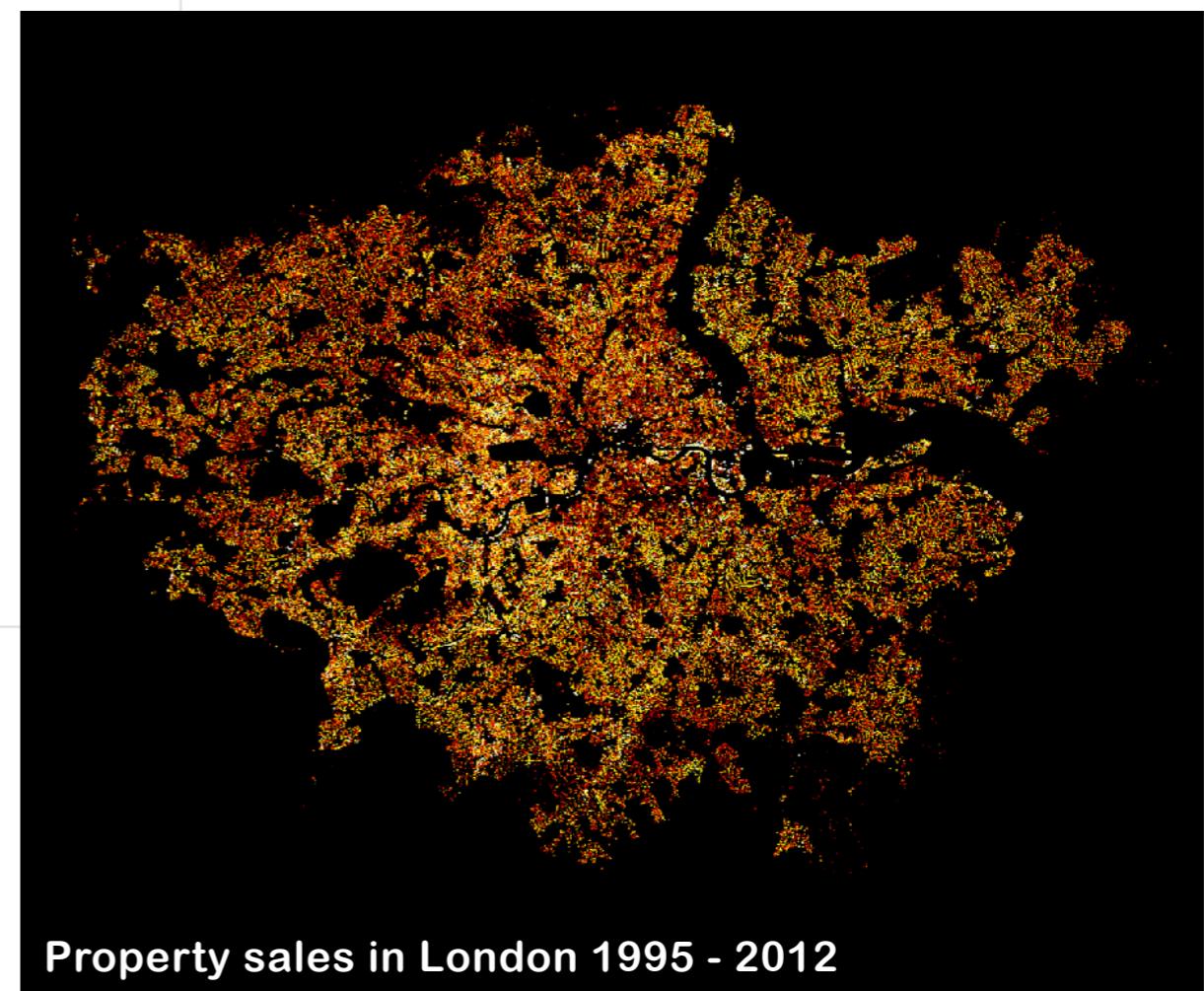
```
--  
97 #define resolution of image: may need to change for varying data volumes  
98 plot_width = int(1500)  
99 plot_height = int(plot_width//1.2)  
100 background = "black"  
101 export = partial(export_image, export_path="export", background=background)  
102  
103 def create_image(x_range, y_range, w=plot_width, h=plot_height):  
104     #set up canvas and populate with datapoints  
105     cvs = ds.Canvas(plot_width=w, plot_height=h, x_range=x_range, y_range=y_range)  
106     agg = cvs.points(df, 'x_col', 'y_col')  
107     img = tf.shade(agg, cmap=Hot, how='eq_hist')  
108     return tf.dynspread(img, threshold=0.5, max_px=4)  
109  
110 export(create_image(*LDN), city[4] + '_' + str(year) + '_hot')  
111
```

London Median House Prices, 2013-2014



LondonSalesMapper.py

Property sales in London 1995 - 2012



THANK YOU

@taslanous

