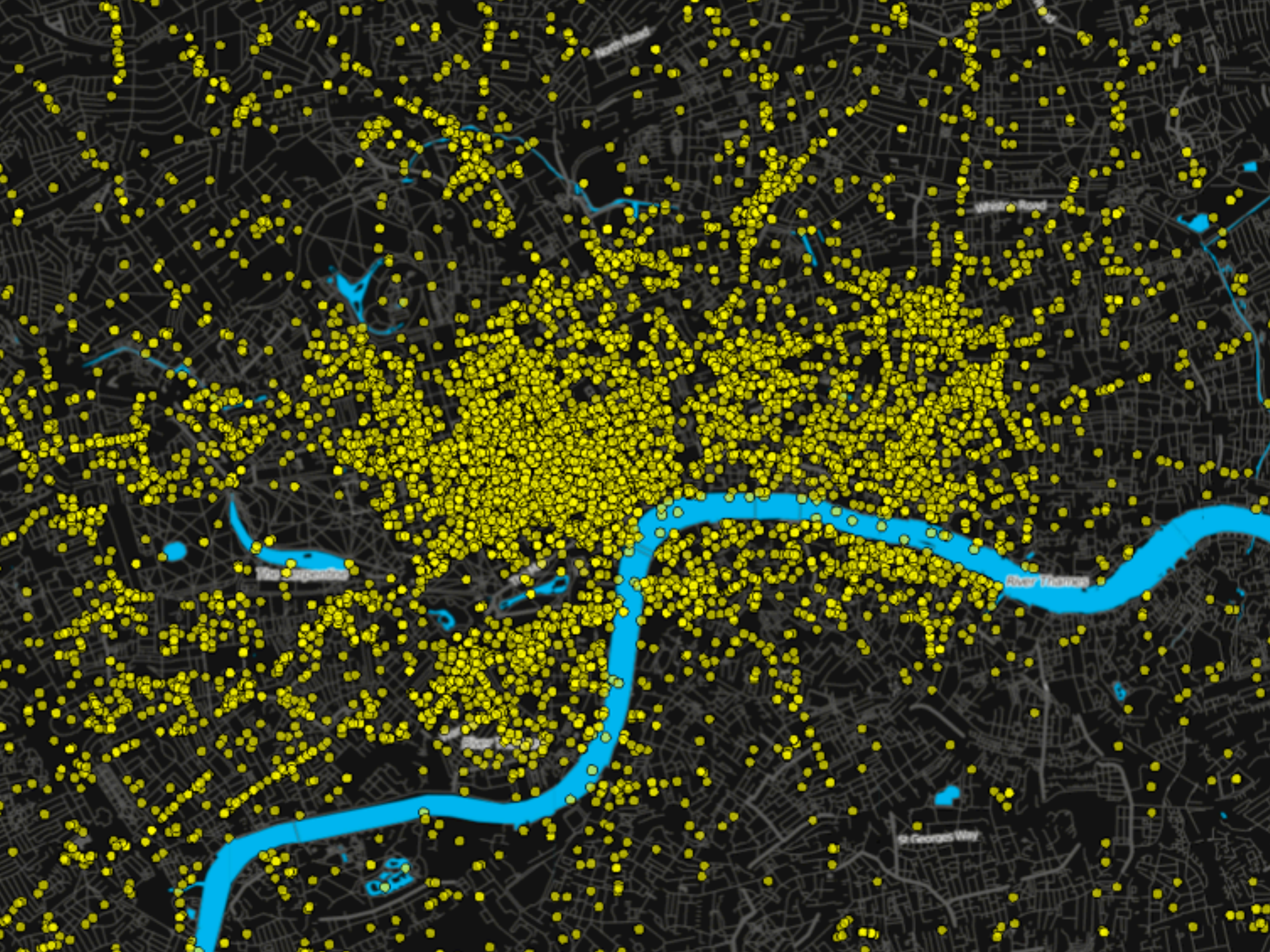
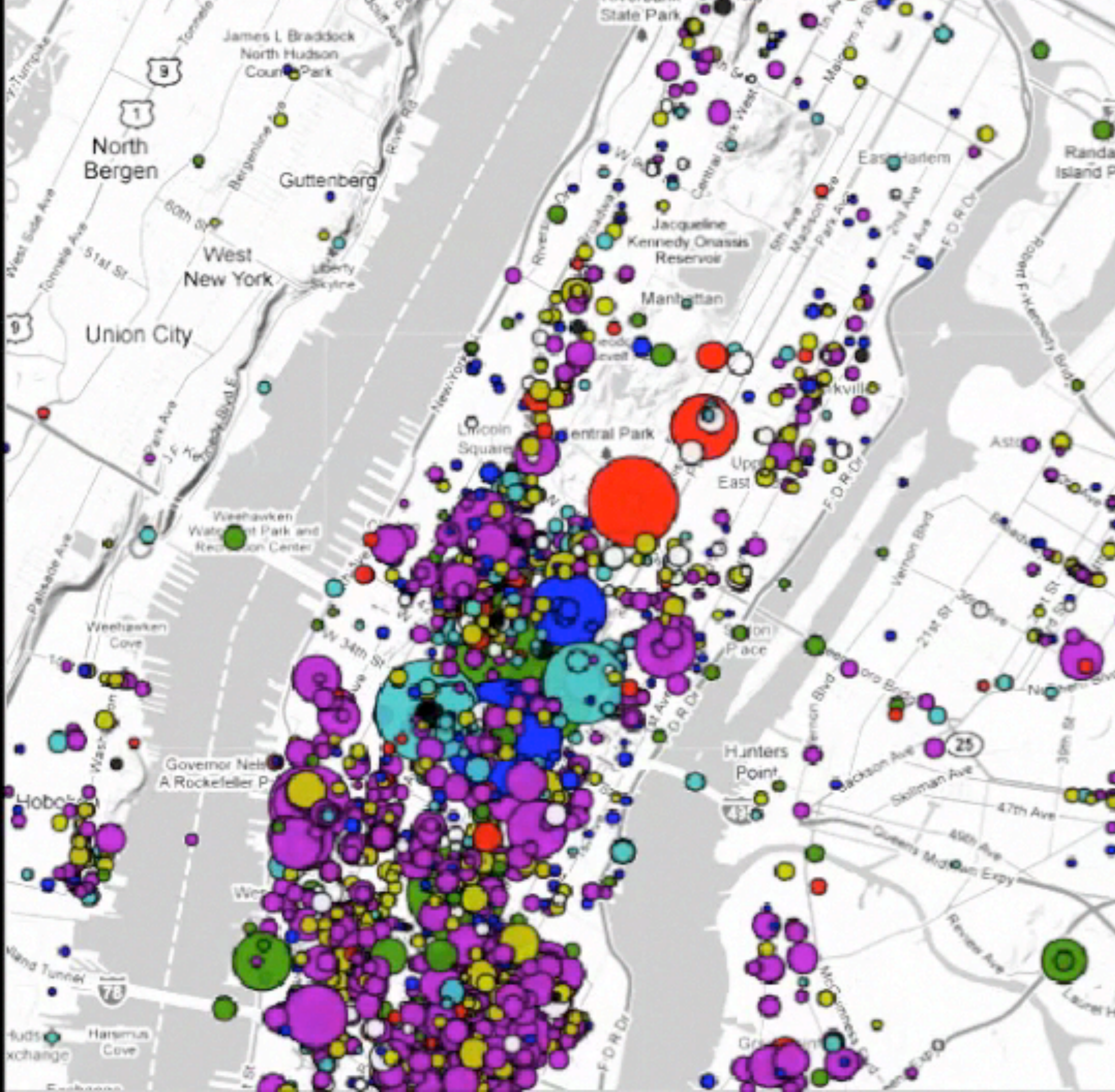
Data Science | Lancaster University

Mining Geo-referenced Data: Location-based Services and the Sharing Economy [Part 2]

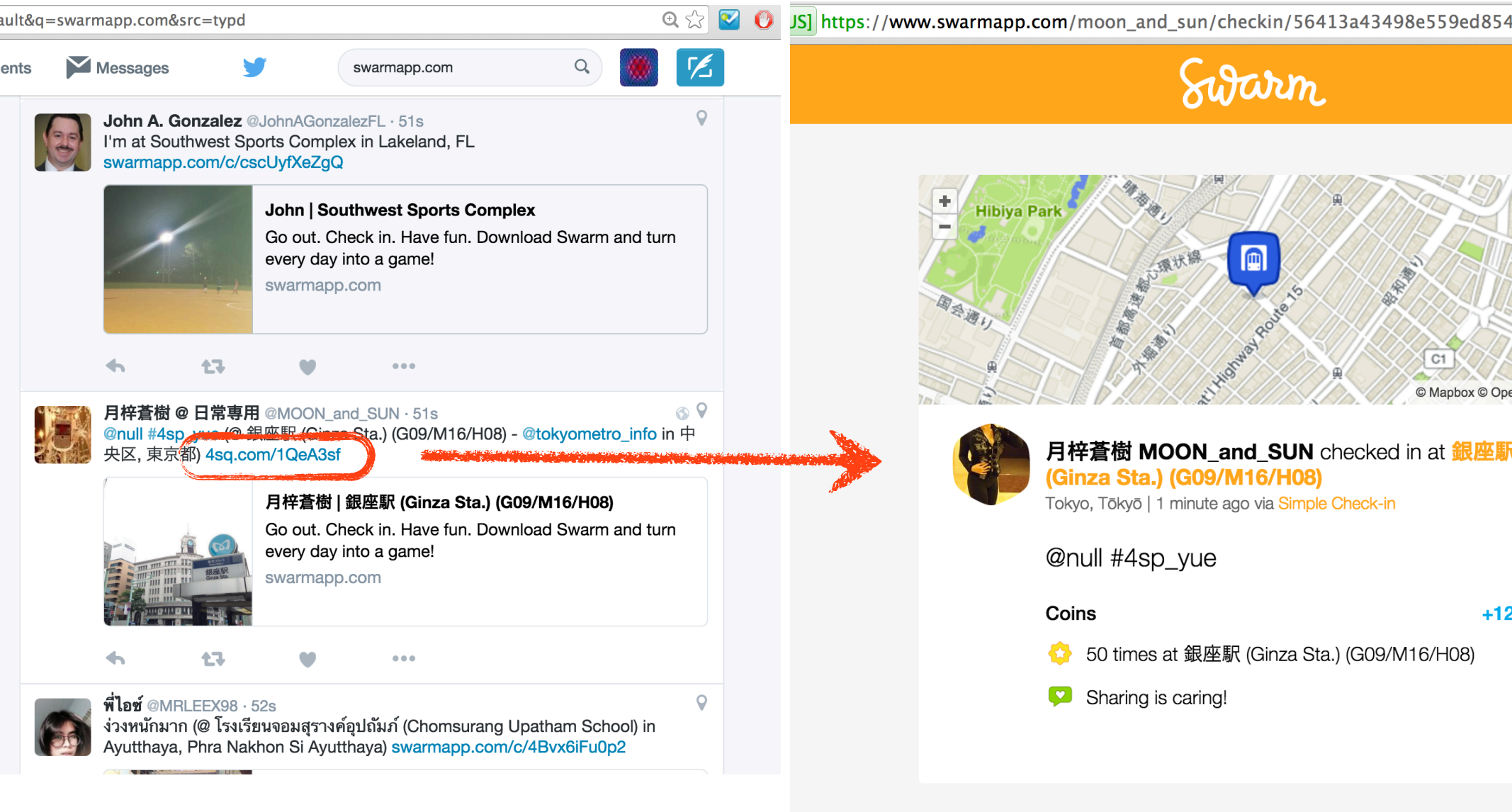
Anastasios Noulas
Data Science Institute, Lancaster University

November 2015
PyData, NYC, 2015


River Thames
St George's Way


James L Braddock North Hudson County Park
North Bergen
Guttenberg
West New York
Union City
Jacqueline Kennedy Onassis Reservoir
Manhattan
Central Park
Lincoln Square
East Harlem
Weehawken Cove
Harsimus Cove
Hunters Point
47th Ave
Skillman Ave
Jackson Ave
Vernon Blvd
Queens Midtown Expy
F D R Dr
Palisade Ave
Tonnele Ave
Park Ave
W 34th St
River Rd
5th Ave
Madison Ave
Park Ave
2nd Ave
1st Ave
21st St

Public check-ins



Working Directory:

https://github.com/bmtgoncalves/Mining-Georeferenced-Data/tree/master/PlaceNetworks_Live

Network Analysis in Python

https://networkx.github.io

NetworkX

NetworkX Home | Documentation | Download | Developer (Github)

High-productivity software for complex networks

NetworkX is a Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Documentation
all documentation

Examples
using the library

Reference
all functions and methods

Features

- Python language data structures for graphs, digraphs, and multigraphs.
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g. text, images, XML records)
- Edges can hold arbitrary data (e.g. weights, time-series)
- Open source BSD license
- Well tested: more than 1800 unit tests, >90% code coverage
- Additional benefits from Python: fast prototyping, easy to teach, multi-platform

Versions

Latest Release

1.10 – 2 August 2015
downloads | docs | pdf

Development

2.0dev
github | docs | pdf
build passing
coverage 93%

Contact

Mailing list
Issue tracker
Developer guide

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)
```

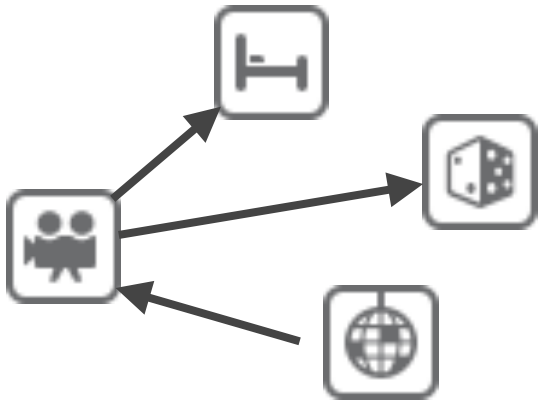
network_analysis.py

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
```

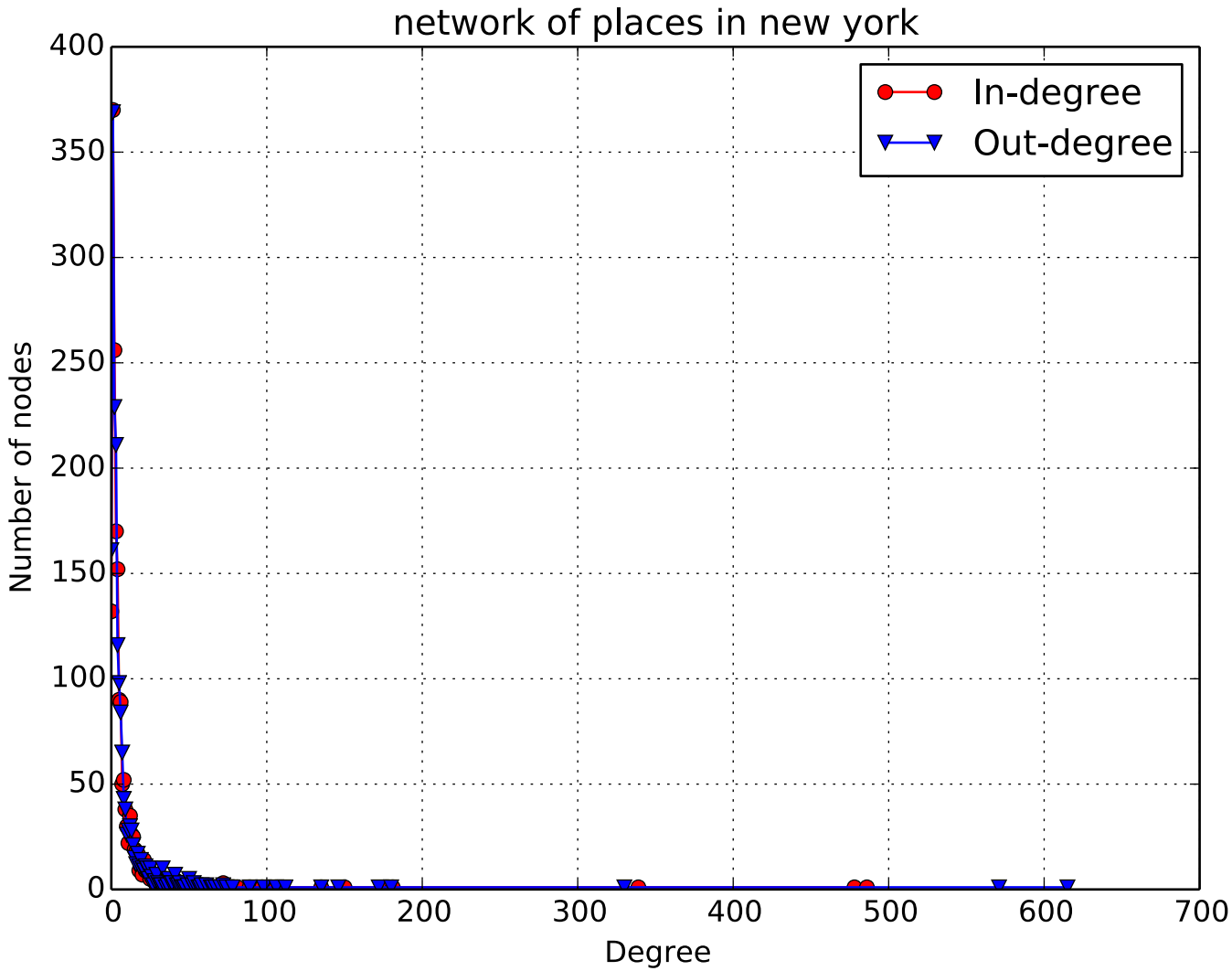
newyork_net.txt

network_analysis.py

Getting Degree Distributions

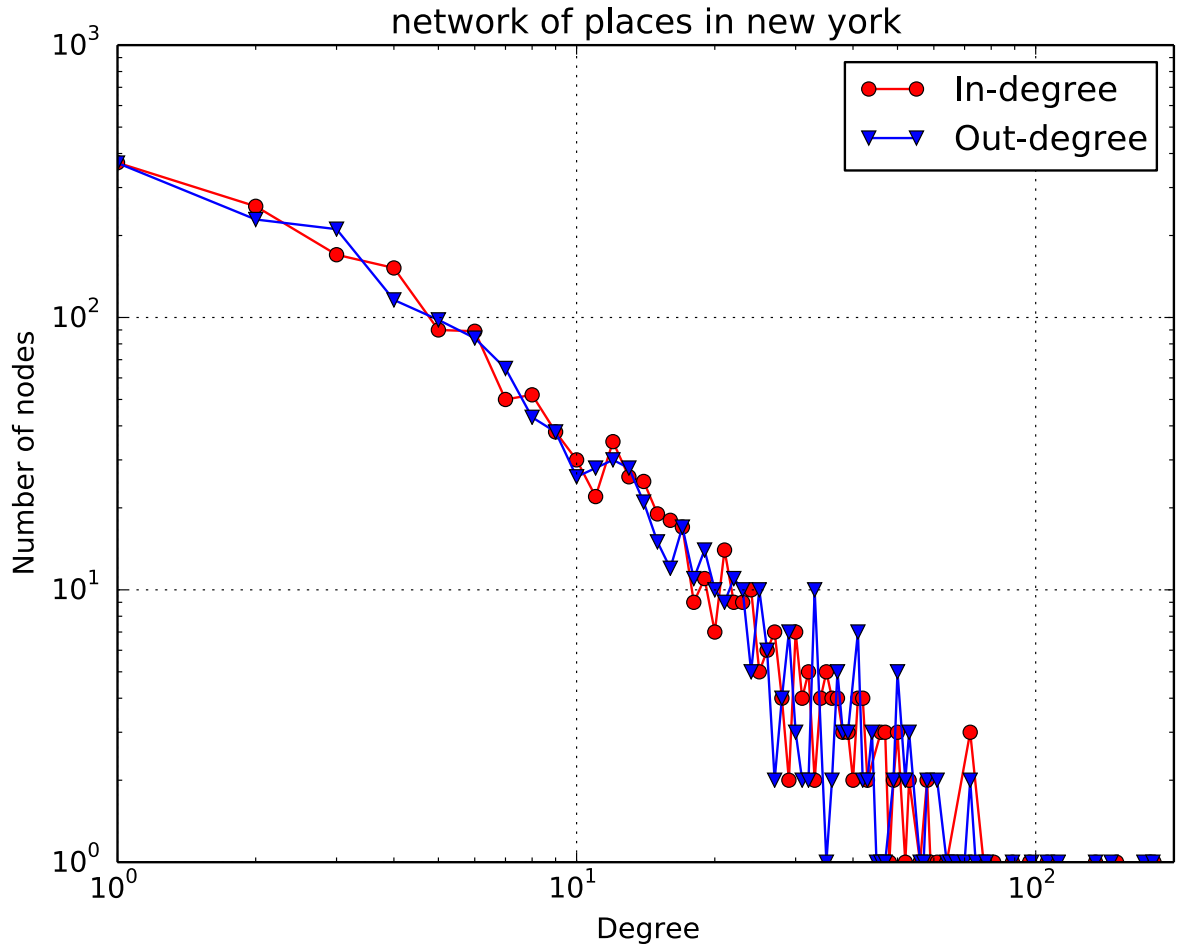
```
in_degrees  = nyc_net.in_degree() #Get degree distributions (in- and out-)
in_values = sorted(set(in_degrees.values()))
in_hist = [in_degrees.values().count(x) for x in in_values]
out_degrees  = nyc_net.out_degree()
out_values = sorted(set(out_degrees.values()))
out_hist = [out_degrees.values().count(x)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

Degree Distribution



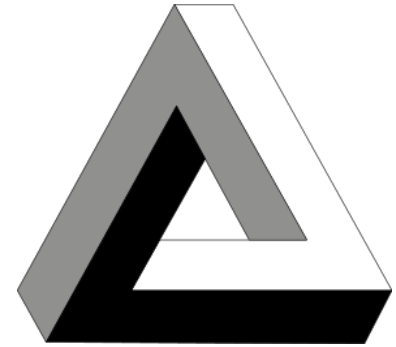
OOPS! We can't see much this way, can't we?

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



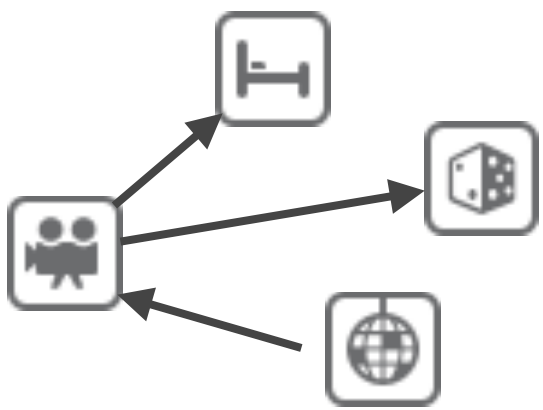
We can get the clustering coefficient of individual nodes or of all the nodes (but the first we convert the graph to an undirected one).

```
# Symmetrize the graph for simplcity
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
```

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)
```



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_cent = 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

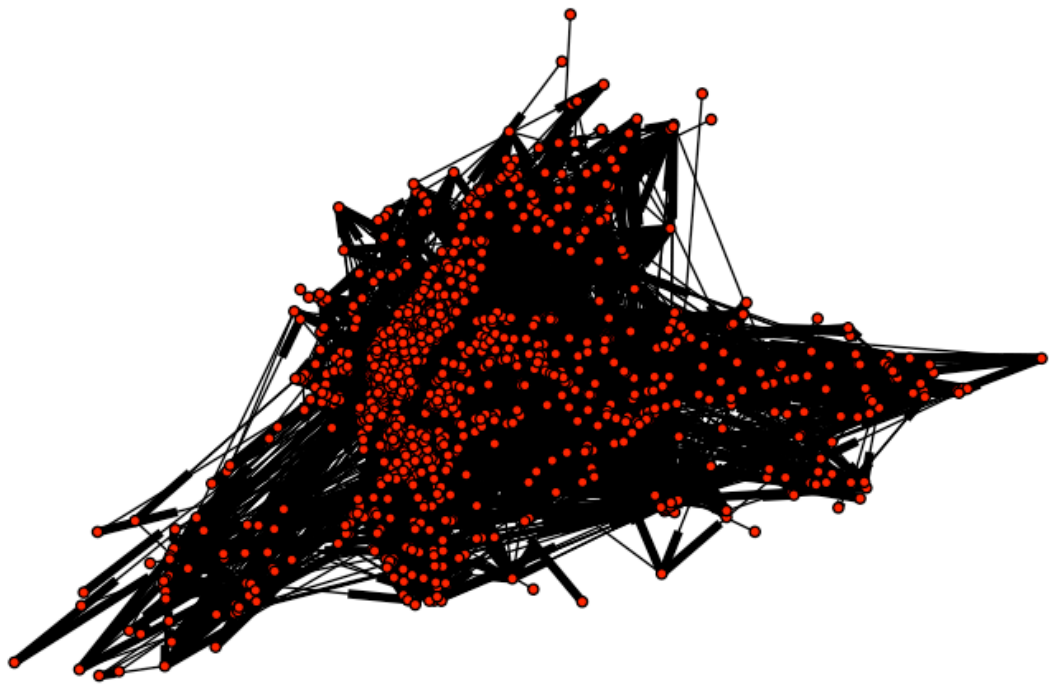
Eigenvector Centrality

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

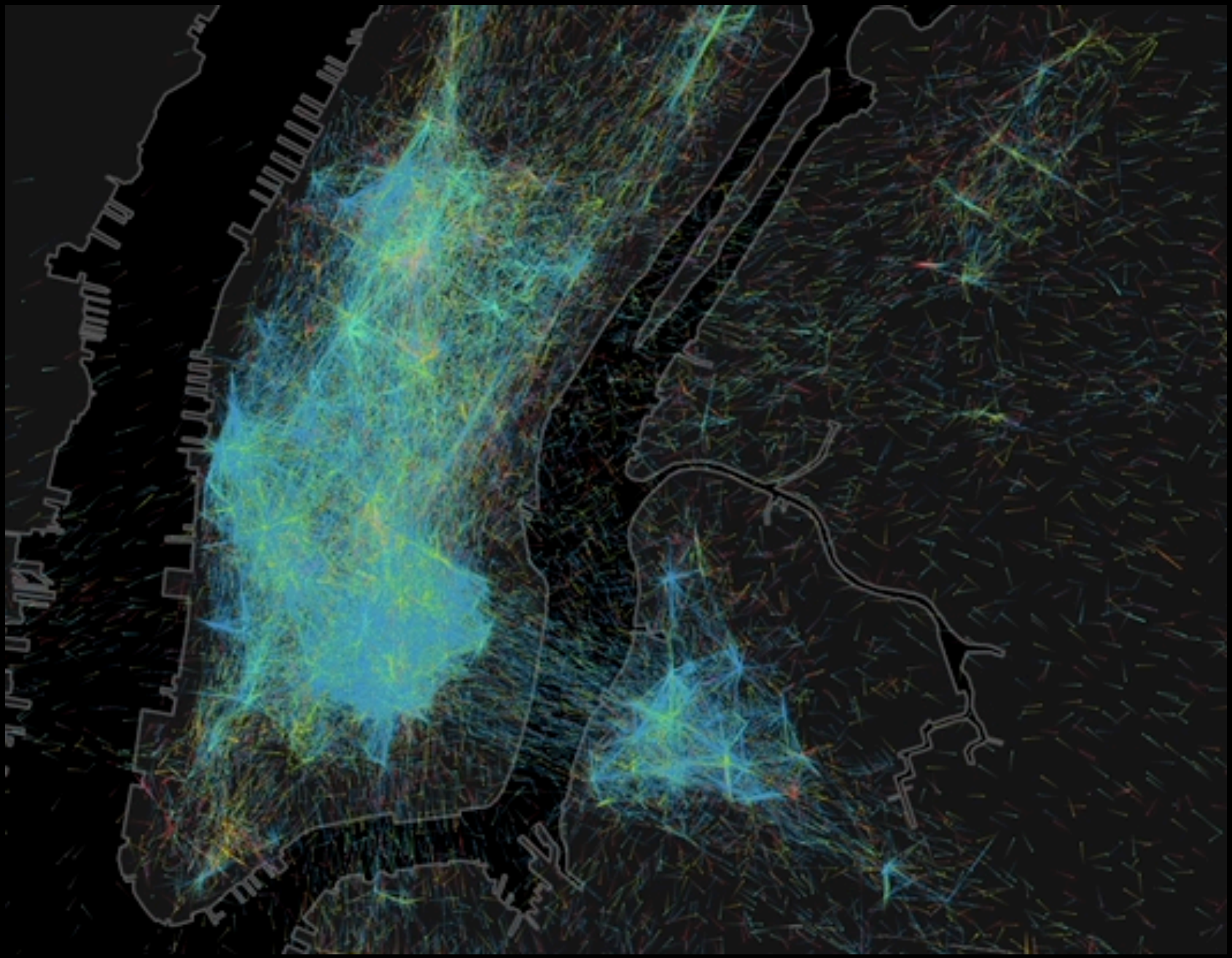
The ranking for the different centrality metrics does not change much, although this may well depend on the type of network under consideration. The results meet ones intuition (if they know New York), yet biases may also exist..

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()
```



New York City



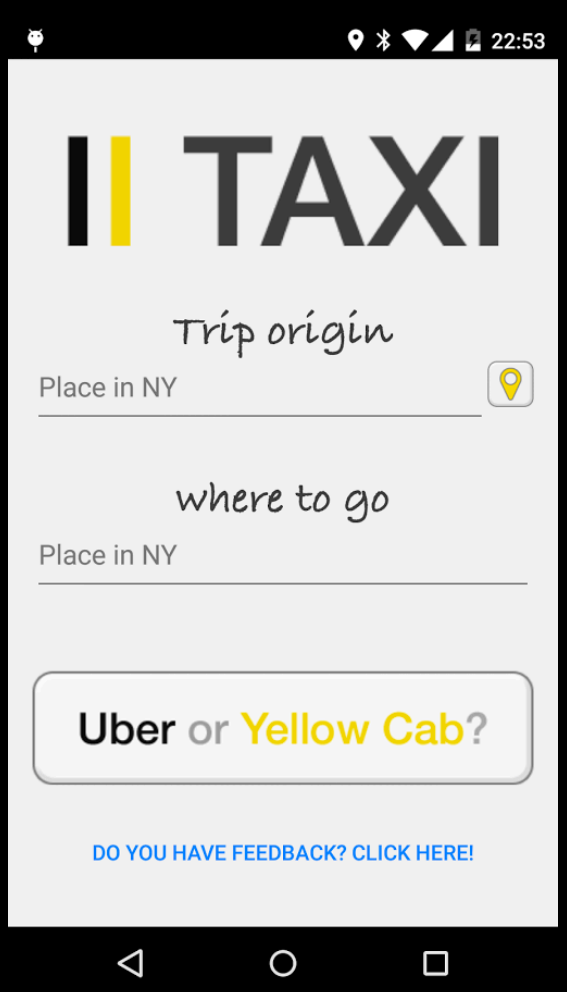
http://arxiv.org/pdf/1502.07979.pdf

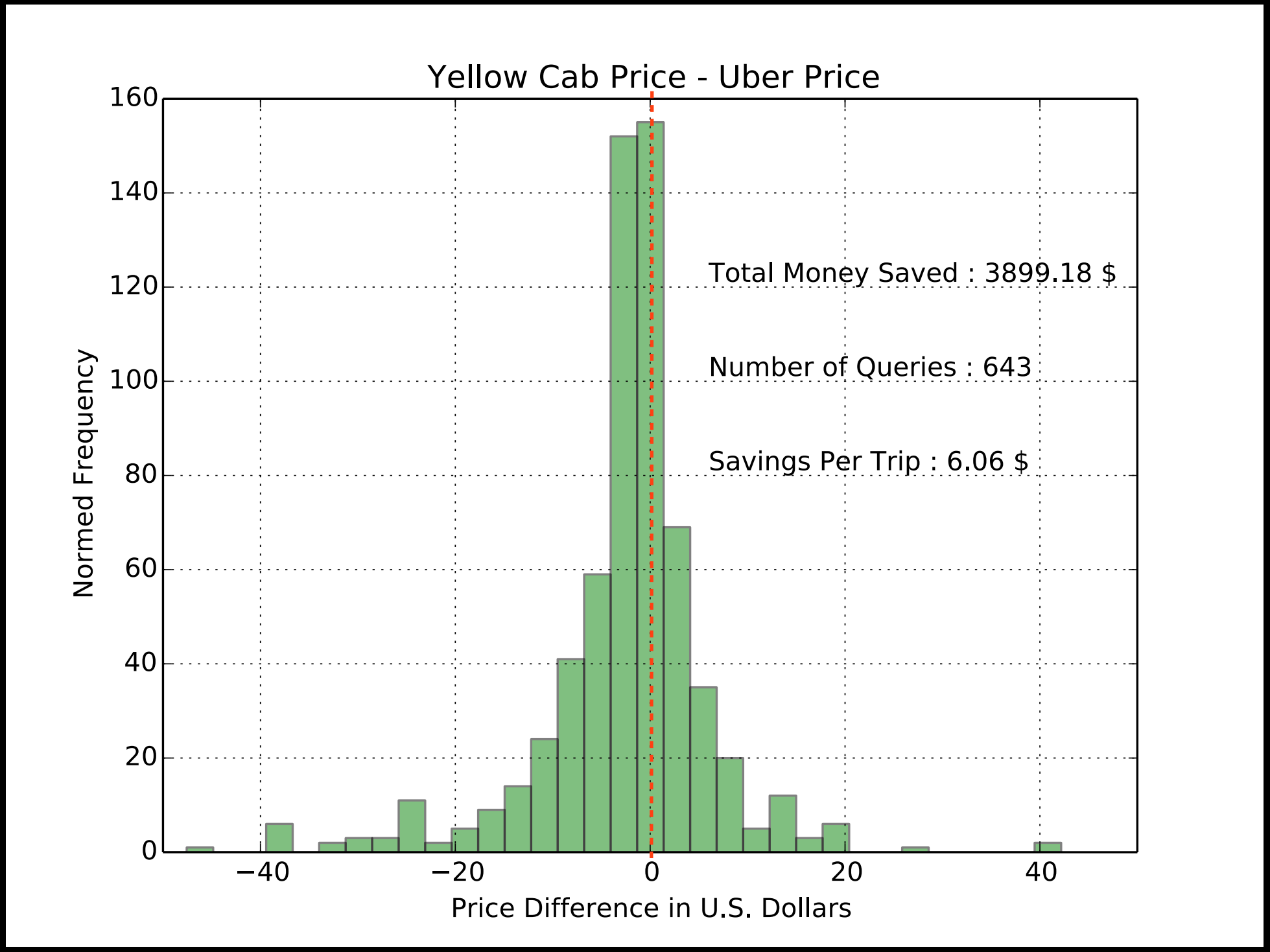
11-12PM

OpenStreetCab.com

Download on the App Store

GET IT ON Google play




Yellow Cab Price - Uber Price
Total Money Saved : 3899.18 $
Number of Queries : 643
Savings Per Trip : 6.06 $
Normed Frequency
Price Difference in U.S. Dollars
160
140
120
100
80
60
40
20
0
−40
−20
0
20
40

Home Notifications Messages Discover Search Twitter

Tim O'Reilly
@timoreilly

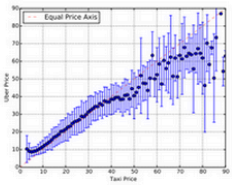
Following

Super-interesting. Data Mining Reveals When A Yellow Taxi Is Cheaper Than Uber bit.ly/1BQOI58 Nice work, @openstreetcab!

MIT Tech Review

Data Mining Reveals When a Yellow Taxi Is Cheaper Than Uber | MIT...

Computer scientists have compared a vast dataset of Yellow Taxi fares in New York City against Uber prices for the first time.



View on web

RETWEETS	FAVORITES
46	35

10:04 AM - 17 Mar 2015

Data Combos: Predicting the tendency of Geographic Areas to Surge

0.42

0.43

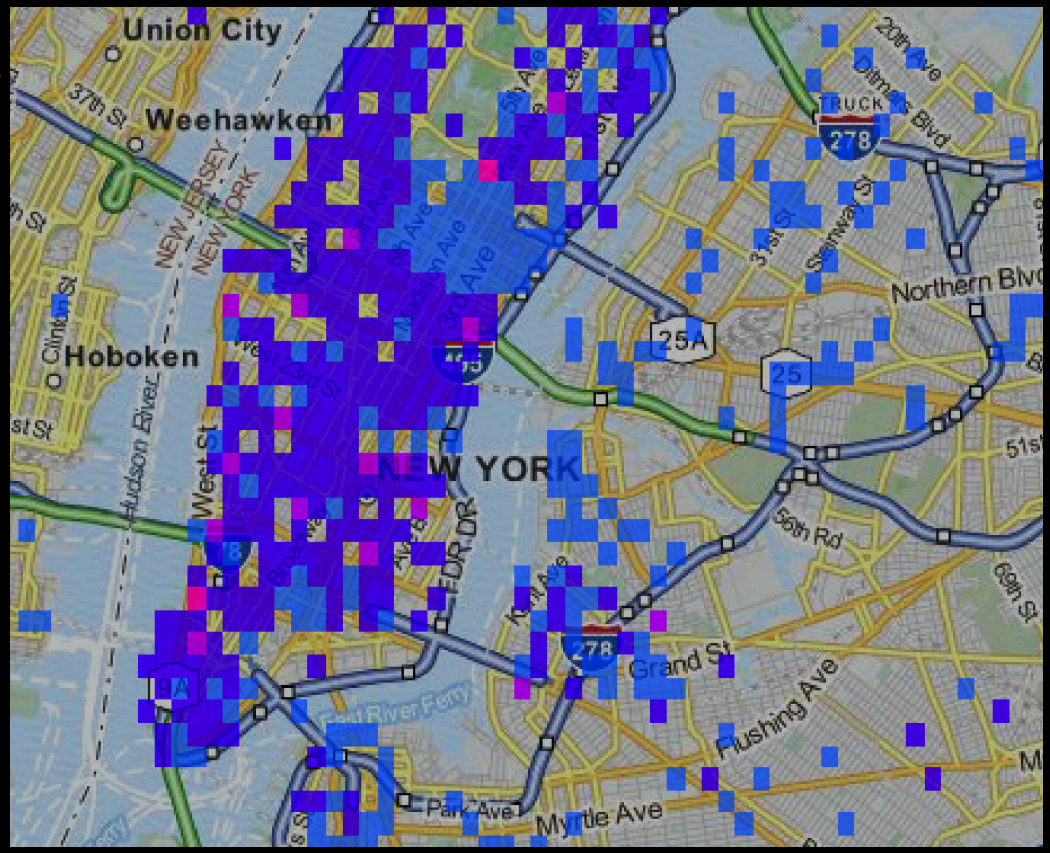
FOURSQUARE

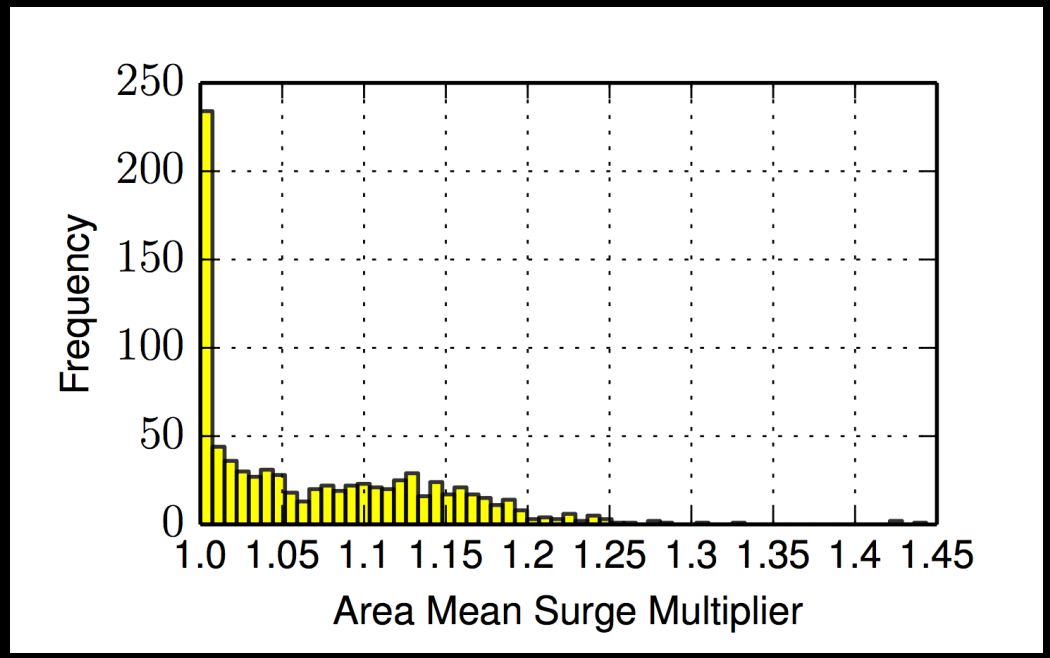
+

=

UBER

Combined Pearson's r= 0.8

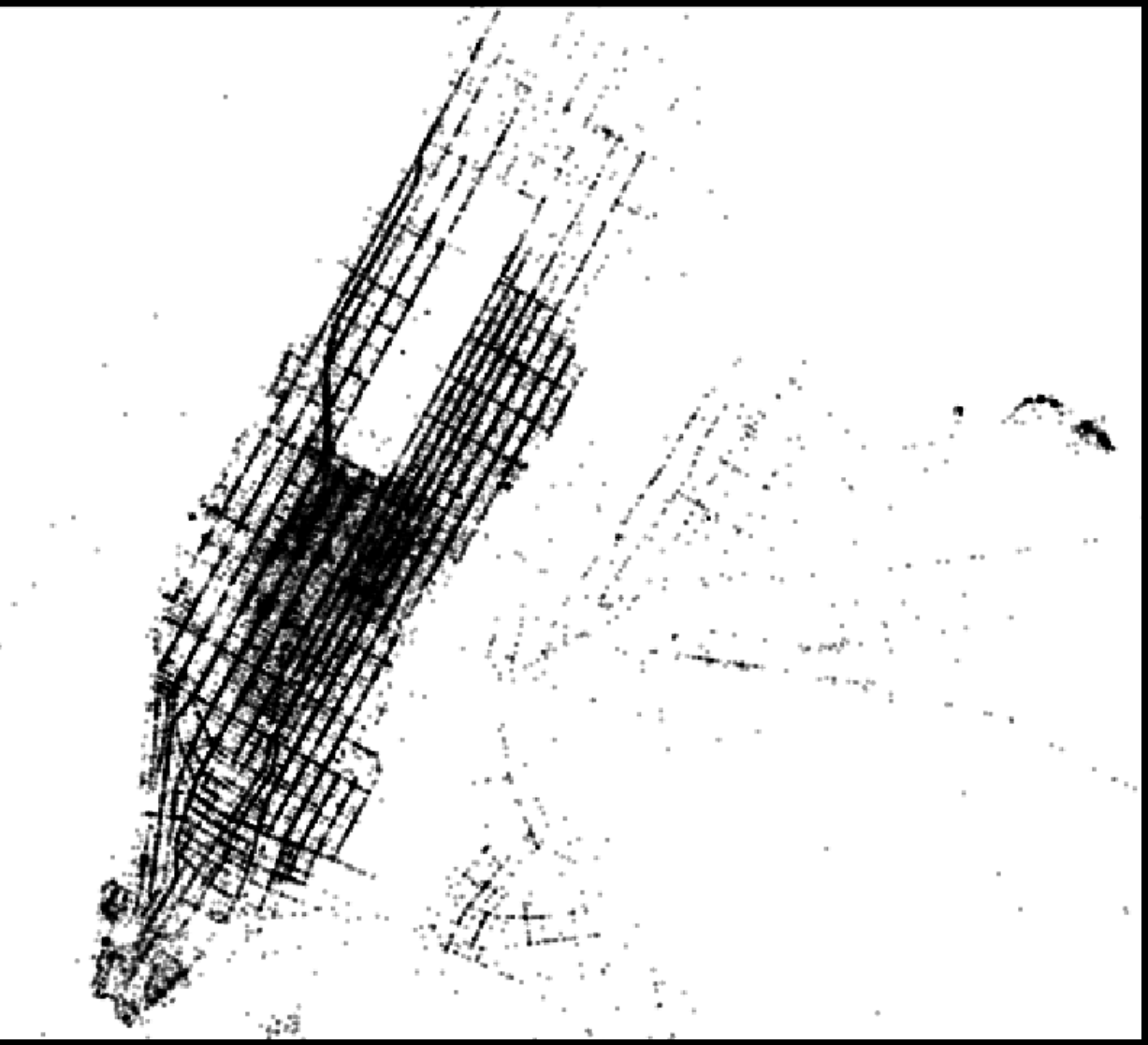




Working Directory:

https://github.com/bmtgoncalves/Mining-Georeferenced-Data/tree/master/PlotTaxiJourneys_Live

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

Plotting taxi journey positions

```
import csv
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

#reading Taxi journey geographic coordinates #
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'])])

y = [i[0] for i in taxi_coords]
x = [i[1] for i in taxi_coords]
colors = ['k' for i in range(0, len(y))]
popularities = [0.1 for i in range(0, len(y))]

m = Basemap(projection='merc',resolution='l',llcrnrlon=-74.0616,urcrnrlat=40.82,
urcrnrlon=-73.8563,llcrnrlat=40.699) #center map to NYC

# maps geocoordinates to pixel positions
x1,y1=m(x,y)
m.scatter(x1,y1,s=popularities,c=colors, marker="o",alpha=0.7)
plt.savefig('taxis.png')
plt.close()
```

taxi_trips.csv

PlotTaxiPos.py

Working Directory:

https://github.com/bmtgoncalves/Mining-Georeferenced-Data/tree/master/SurgePrediction_Live

Loading Airbnb listings in NYC

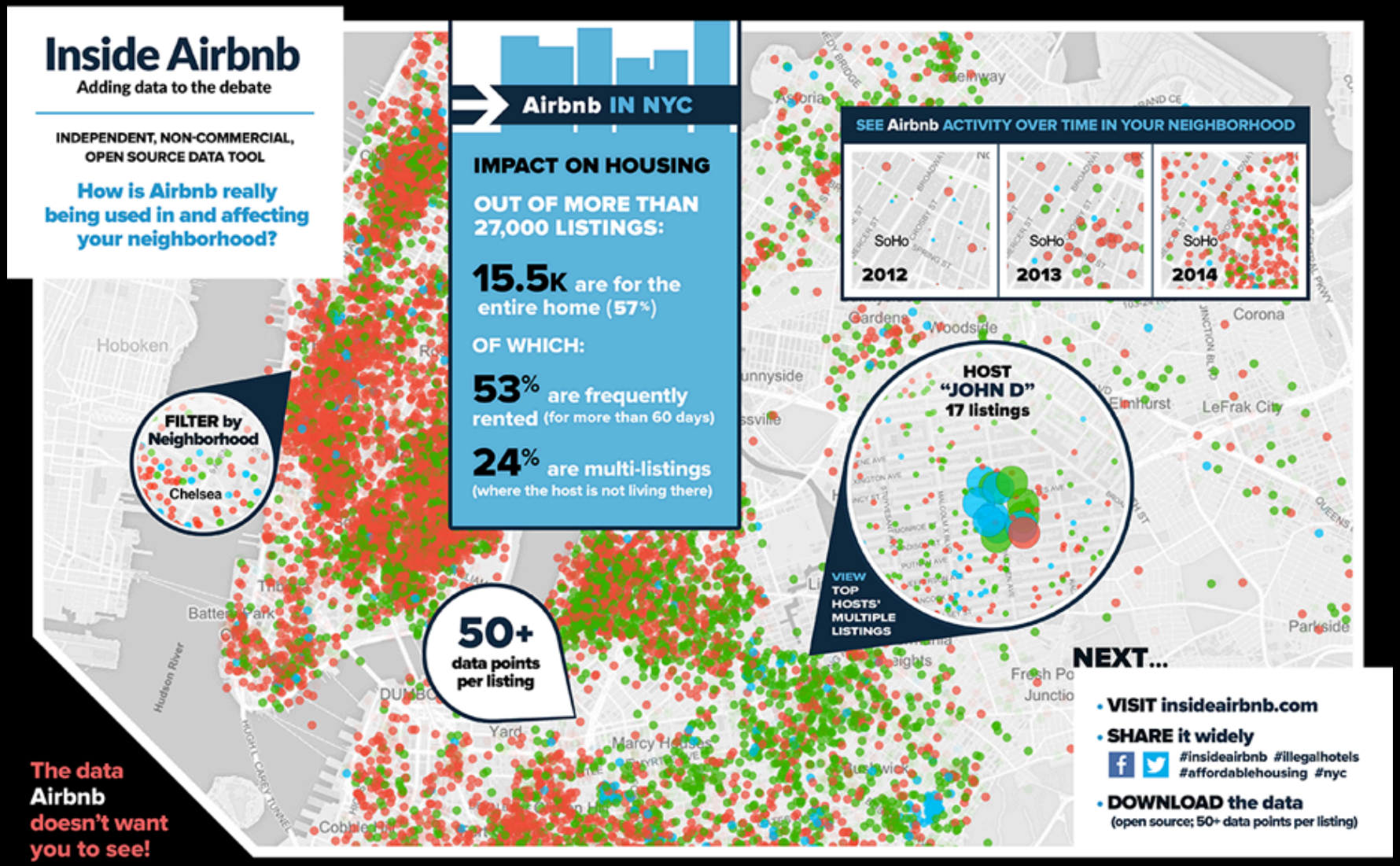
```
airbnb_coords = []

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))
```

A lot of fields in there: we pick only coordinates for this task!

id,listing_url,scrape_id,last_scraped,name,summary,space,description,experiences_offered,neighborhood_overview,notes,transit,thumbnail_url,medium_url,picture_url,xl_picture_url,host_id,host_url,host_name,host_since,host_location,host_about,host_response_time,host_response_rate,host_acceptance_rate,host_is_superhost,host_thumbnail_url,host_picture_url,host_neighbourhood,host_listings_count,host_total_listings_count,host_verifications,host_has_profile_pic,host_identity_verified,street,neighbourhood,neighbourhood_cleansed,neighbourhood_group_cleansed,city,state,zipcode,market,smart_location,country_code,country,latitude,longitude,is_location_exact,property_type,room_type,accommodates,bathrooms,bedrooms,beds,bed_type,amenities,square_feet,price,weekly_price,monthly_price,security_deposit,cleaning_fee,guests_included,extra_people,minimum_nights,maximum_nights,calendar_updated,has_availability,availability_30,availability_60,availability_90,availability_365,calendar_last_scraped,number_of_reviews,first_review,last_review,review_scores_rating,review_scores_accuracy,review_scores_cleanliness,review_scores_checkin,review_scores_communication,review_scores_location,review_scores_value,requires_license,license,jurisdiction_names,instant_bookable,cancellation_policy,require_guest_profile_picture,require_guest_phone_verification,calculated_host_listings_count,reviews_per_month

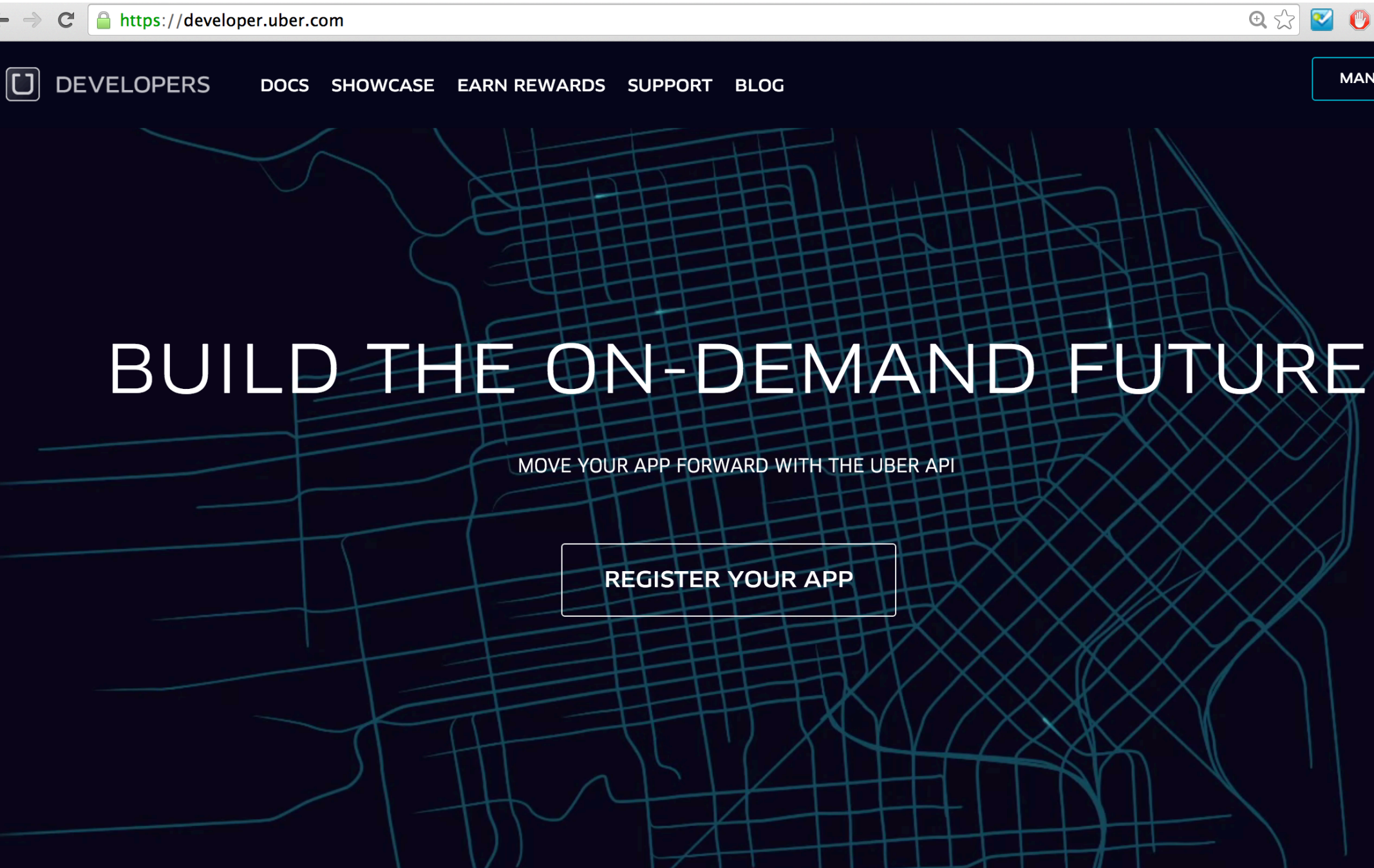
listings_sample.csv

MultiLayerSurgePrediction.py



http://insideairbnb.com/about.html

Uber's API: register (not now!) to get your keys...



Querying the Uber API

```
def uber_request(latitude, longitude, code='price', end_latitude=None, end_longitude=None):
        client_id = 'ADD'
        server_token = 'ADD'
        secret = 'ADD'


        price_parameters = {
                # 'client_id': client_id,
           'server_token': server_token,
           'client_secret': secret,
           'start_latitude': latitude,
           'start_longitude': longitude,
           'end_latitude': end_latitude,
           'end_longitude': end_longitude,
        }

        price_url = 'https://api.uber.com/v1/estimates/price'
        print 'Getting price quote...'
        req_url = price_url
        response_price = requests.get(price_url, params=price_parameters)
        data_price = response_price.json()
        return data_price
```

https://developer.uber.com/v1/tutorials/

Price Query output

(40.762211, -73.986389, {u'prices': [{u'localized_display_name': u'uberX', u'distance': 3.64, u'display_name': u'uberX', u'product_id': u'b8e5c464-5de2-4539-a35a-986d6e58f186', u'high_estimate': 23, u'surge_multiplier': 1.0, u'minimum': 8, u'low_estimate': 17, u'duration': 1205, u'estimate': u'$17-23', u'currency_code': u'USD'}, {u'localized_display_name': u'uberXL', u'distance': 3.64, u'display_name': u'uberXL', u'product_id': u'1e0ce2df-4a1e-4333-86dd-dc0c67aaabe1', u'high_estimate': 34, u'surge_multiplier': 1.0, u'minimum': 12, u'low_estimate': 25, u'duration': 1205, u'estimate': u'$25-34', u'currency_code': u'USD'}, {u'localized_display_name': u'uberFAMILY', u'distance': 3.64, u'display_name': u'uberFAMILY', u'product_id': u'd6d6d7ad-67f9-43ef-a8de-86bd6224613a', u'high_estimate': 33, u'surge_multiplier': 1.0, u'minimum': 18, u'low_estimate': 27, u'duration': 1205, u'estimate': u'$27-33', u'currency_code': u'USD'}, {u'localized_display_name': u'UberBLACK', u'distance': 3.64, u'display_name': u'UberBLACK', u'product_id': u'0e9d8dd3-ffec-4c2b-9714-537e6174bb88', u'high_estimate': 40, u'surge_multiplier': 1.0, u'minimum': 15, u'low_estimate': 31, u'duration': 1205, u'estimate': u'$31-40', u'currency_code': u'USD'}, {u'localized_display_name': u'UberSUV', u'distance': 3.64, u'display_name': u'UberSUV', u'product_id': u'56487469-0d3d-4f19-b662-234b7576a562', u'high_estimate': 53, u'surge_multiplier': 1.0, u'minimum': 25, u'low_estimate': 43, u'duration': 1205, u'estimate': u'$43-53', u'currency_code': u'USD'}, {u'localized_display_name': u'uberT', u'distance': 3.64, u'display_name': u'uberT', u'product_id': u'ebe413ab-cf49-465f-8564-a71119bfa449', u'high_estimate': None, u'surge_multiplier': 1.0, u'minimum': None, u'low_estimate': None, u'duration': 1205, u'estimate': u'Metered', u'currency_code': None}, {u'localized_display_name': u'Yellow WAV', u'distance': 3.64, u'display_name': u'Yellow WAV', u'product_id': u'1864554f-7796-4043-82d4-883ddde1070a', u'high_estimate': 0, u'surge_multiplier': 1.0, u'minimum': 0, u'low_estimate': 0, u'duration': 1205, u'estimate': u'$0', u'currency_code': u'USD'}]}, 40.757778, -73.94799)

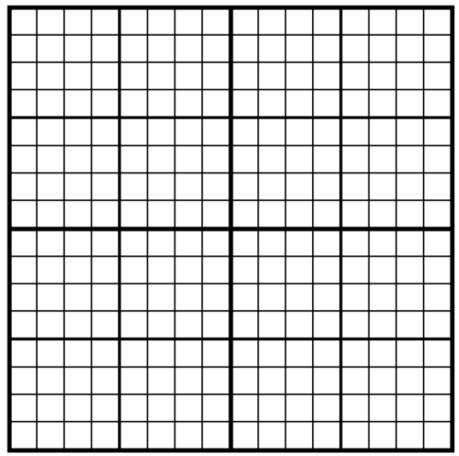
Querying the Uber API (2)

```
f_out = open('UberQuestOutput.txt', 'w')

for i in range(0,1000):
        coordsOrigin = random.choice(taxi_coords)
        coordsDestination = random.choice(taxi_coords)
        latitude_O = coordsOrigin[0]
        longitude_O = coordsOrigin[1]

        latitude_D = coordsDestination[0]
        longitude_D = coordsDestination[1]

        try:
                uber_response = uber_request(latitude_O, longitude_O, 'price', latitude_D, longitude_D)
        except:
                time.sleep(10.0)
                continue
        print >> f_out, str((latitude_O, longitude_O, uber_response, latitude_D, longitude_D))
        print 'Query retrieved, sleeping for a sec.'
        time.sleep(1.0)


f_out.close()
print 'Uber Quest is Over.'
```

averageSurgeInCell.csv

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))
```

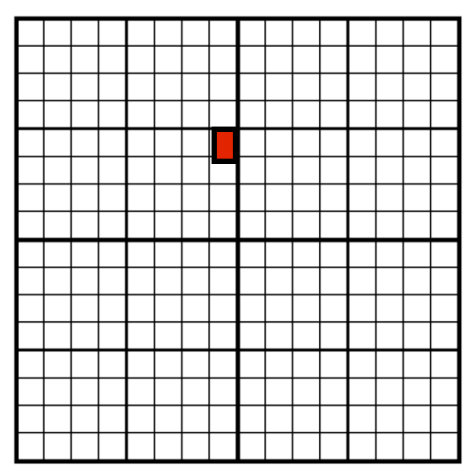


Loading 3 geo-data layers

```
######## 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))
```

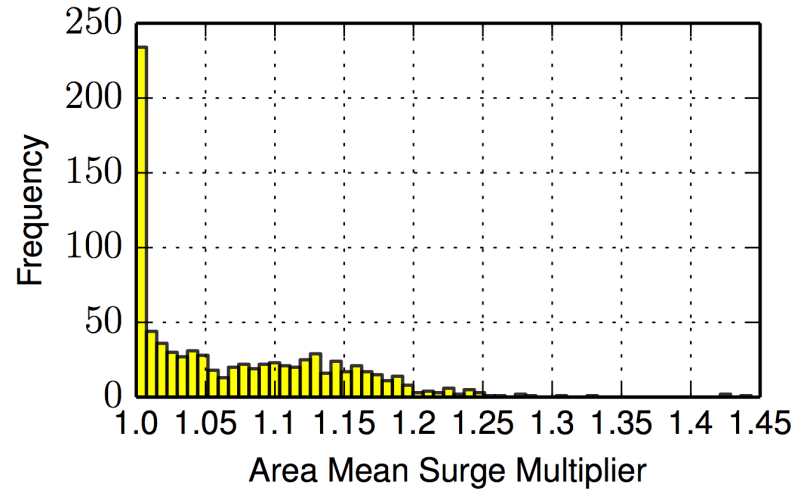
Count occurrences per cell

```
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 ingore point
                if (lat < latMin) or (lat > latMax) or (longit < lonMin) or (longit > lonMax):
                   continue

                #if outside the grid then ingore point
                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
```



Loading and plotting average surge per area

```
#Load Area Average Surge: assume you have queried previously the UBER API and
have collected info on pricing for an area
latLongSurge = {}
for l in open('averageSurgeInCell.csv', 'r'):
        splits = l.split(',')
        cy = int(splits[0])
        cx = int(splits[1])
        averageSurgeCoeff = float(splits[2])

        latLongSurge.setdefault(cy, {})
        latLongSurge[cy][cx] = averageSurgeCoeff
```




```
####################
import pylab as plt
surgeValues = [latLongSurge[cy][cx] for cy in range(0, latLen-1) for cx in range(0,lonLen-1) if
latLongSurge[cy][cx] !=0]

plt.hist(surgeValues , color='yellow', bins=60, label = 'Surge Multipliers',  alpha=0.8)
plt.ylabel('Frequency')
plt.xlabel('Area Mean Surge Multiplier')

vals = ['1.0','1.05','1.1','1.15','1.2','1.25','1.3','1.35','1.4','1.45']
plt.xticks([1.0,1.05,1.1,1.15,1.2,1.25,1.3,1.35,1.4,1.45], vals, fontsize=10)


plt.grid(True)
plt.savefig('surgeMultDistribution.pdf')
plt.close()
```

Surge Prediction

```
import scipy.stats as stats

yellows = [TAXIcellCount[cy][cx] for cy in range(0, latLen-1) for cx in range(0,lonLen-1) if
latLongSurge[cy][cx] !=0]

places = [FSQcellCount[cy][cx] for cy in range(0, latLen-1) for cx in range(0,lonLen-1) if
latLongSurge[cy][cx] !=0]

listings = [AIRcellCount[cy][cx] for cy in range(0, latLen-1) for cx in range(0,lonLen-1) if
latLongSurge[cy][cx] !=0]

cnt = 0
labels = ['Yellow Taxis', 'Foursquare Places', 'Airbnb Listings']
print len(surgeValues)
print len(yellows)
for predictor in [yellows, places, listings]:
        r, p_value = stats.pearsonr(surgeValues, predictor)
```

```
Yellow Taxis
0.432527851946
---
Foursquare Places
0.407164228291
---
Airbnb Listings
0.327540653569
```

Supervised learning for Surge prediction

```
from sklearn import tree
y_train = [] #a list for your training labels
X_train = [] #feature values go here
for i in range(0, len(surgeValues)):
        y_train.append(surgeValues[i])
        X_train.append([yellows[i], listings[i], places[i]])

print 'Training Decision Tree Regressor..'
###  Training and Testing: LEAVE ONE OUT ERROR ####
super_predictions = []
super_predictions2 = []
for i in range(0, len(surgeValues)):
        training_data_X = X_train[:i] + X_train[i+1:]
        label_data_Y = y_train[:i] + y_train[i+1:]
        NX_train = []
        ny_train = []
        for j in range(0, len(label_data_Y)):
                if label_data_Y[j] == 1.0:
                        continue
                else:
                        NX_train.append(X_train[j])
                        ny_train.append(y_train[j])

        clf = tree.DecisionTreeRegressor(max_depth=30).fit(NX_train, ny_train)
        super_predictions.append(clf.predict(X_train[i])[0])

r, p_value = stats.pearsonr(surgeValues, super_predictions)
print 'Decision Tree Supervised Learning Regressor, r :'  + str(r)
```

New VERSION

OF OPENSTREETCAB

COMING SOON

IN NYC + LONDON

http://arxiv.org/pdf/1508.07292.pdf

@OpenStreetCab

THANK YOU

@taslanous