Lecture April 20

- Data Science Project Example
- GeoSpatial Analysis

Data Science Project Example

Arrests vs. Commuting networks Analysis

CrimevsCommuntinNetworks/*comparing the commuting flowsMG.ipynb*

Note: Here you find important Guidelines for Project Analysis and Presentation.

Geospatial Analysis

1-OSMNx: Library to analyze Road Networks and more

osmnx-features-demo_wKey.ipynb ***(needs google API key)***

2-Reading geospatial data from the Web

Reading_CensusShapefiles_wKey.ipynb ***(needs Census API key)***

3-Reading Commuters from the Bureau of Labor Statistics

FourSteps_PartI_wKey.ipynb

Note: See this part as useful, and advanced material. Links provided
For future references.

[illegible]

[illegible]

CrimevsCommutingNetworks.zip

flows.csv → [illegible]

tract_coords.csv

crimes_with_residences2.csv → [illegible]

ICERM

Predictive Policing

Aug 8 - 12, 2016

In [1]:
```
import pandas as pd
import networkx as nx # for later
from matplotlib import pyplot as plt
from matplotlib import pylab
%matplotlib inline
```

In [2]:
```
#Read original data
file_path = 'crimes_with_residences2.csv'

crime = pd.read_csv(file_path, dtype = {'blockgroup_residence' : str,
                                        'blockgroup' : str })
##eliminates the last digit to go from block to tract
crime['tract_residence'] = crime.blockgroup_residence.str[:-1]
crime['tract_crime'] = crime.blockgroup.str[:-1]
```

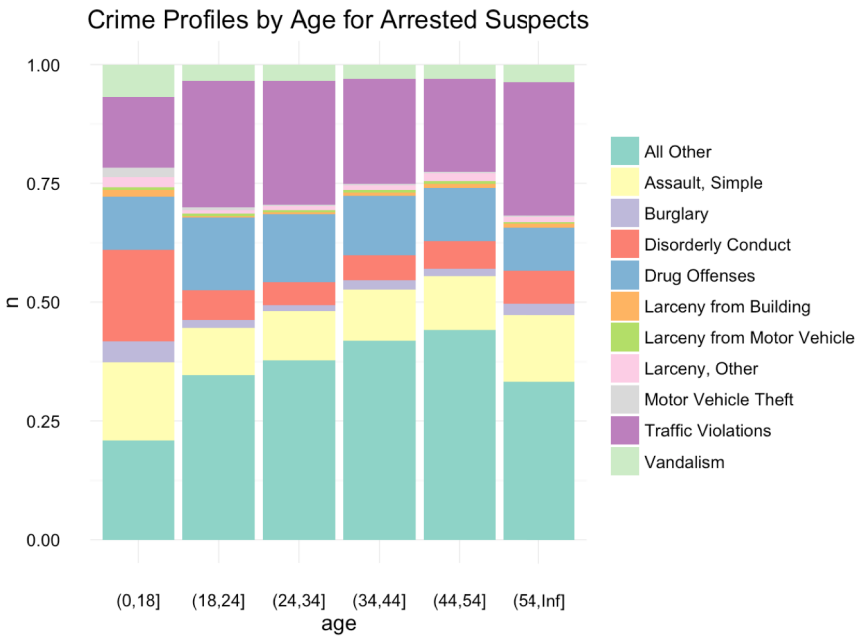
In [3]:
```
crime
```

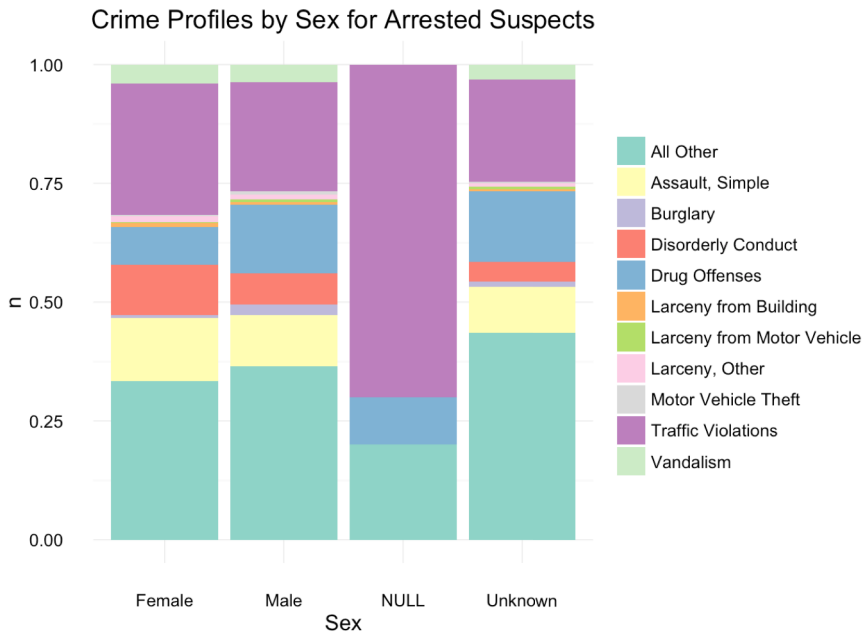
Out[3]:

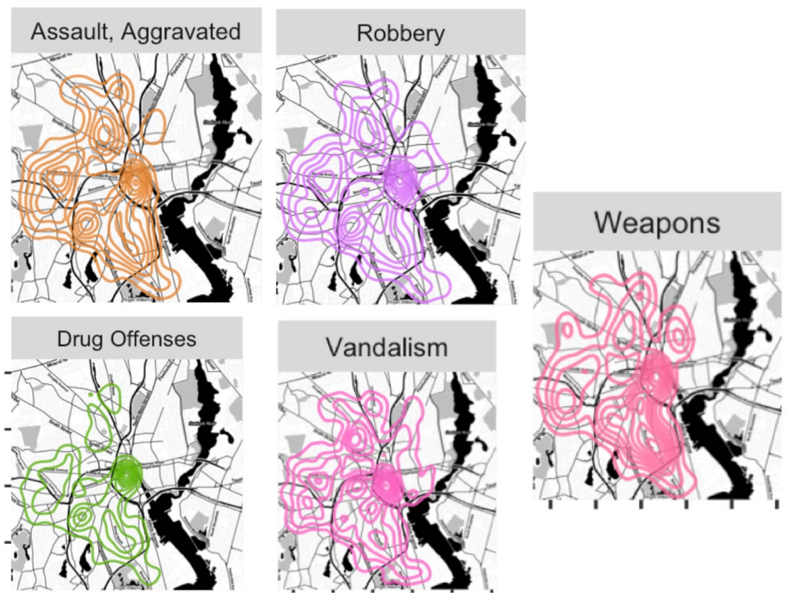
	Unnamed: 0	X	CaseID	CaseNumber	Location	ReportedDate	OccurredFromDate	OccurredThroughDate	OffenseID	IBRCode	.
0	1	154	673371	2010-00118035	50 EATON ST	11/6/10 1:09	11/6/10 1:09	11/6/10 1:09	587077	90Z	.
1	2	155	673371	2010-00118035	50 EATON ST	11/6/10 1:09	11/6/10 1:09	11/6/10 1:09	576015	120	.
2	3	156	623467	2009-00035148	50 EATON ST	4/8/09 20:02	4/8/09 20:02	4/8/09 20:02	519446	23H	.
3	4	157	608066	2008-00116479	50 EATON ST	10/7/08 17:37	10/3/08 12:00	10/7/08 17:30	501469	250	.
4	5	158	754943	2013-00084097	255 ATWELLS AVE	9/4/13 17:19	9/4/13 13:30	9/4/13 13:30	667426	23F	.
5	6	159	640785	2009-00115047	35 OPPER ST	10/23/09 18:41	10/10/09 12:00	10/10/09 12:00	539296	26A	.
6	7	160	629733	2009-00063976	53 LANGDON ST	6/18/09 20:00	10/2/08 12:00	6/18/09 12:00	526670	23H	.
7	8	161	640382	2009-00112156	100 BROAD ST	10/15/09 15:50	10/10/09 10:00	10/10/09 10:30	538867	23D	.
8	9	162	717369	2012-00041141	25 FLORENCE	5/10/12 17:38	5/10/12 17:35	5/10/12 17:35	625004	290	.

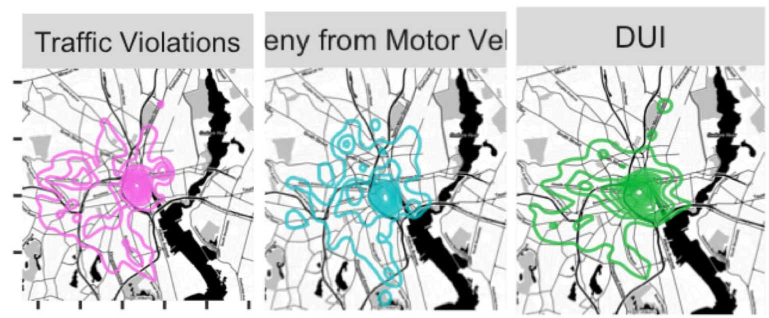
4]:

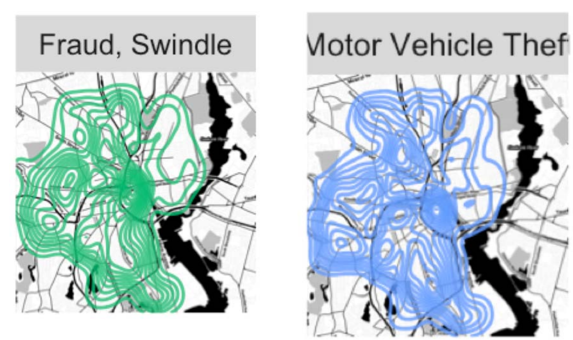
le	...	Race	Ethnicity	GangAffiliation	YearOfBirth	CurrentAge	lat_residence	lon_residence	blockgroup_residence	tract_residence	tract_crim
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700230
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700230
	...	White	NonHispanic	Plain Street Group	1989.0	27.0	41.813327	-71.416567	440070007002	44007000700	4400700230
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700230
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700090
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700290
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700290
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700080
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700190
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700190
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700290


Crime Profiles by Age for Arrested Suspects
1.00
0.75
0.50
0.25
0.00
n
(0,18]
(18,24]
(24,34]
(34,44]
(44,54]
(54,Inf]
age
All Other
Assault, Simple
Burglary
Disorderly Conduct
Drug Offenses
Larceny from Building
Larceny from Motor Vehicle
Larceny, Other
Motor Vehicle Theft
Traffic Violations
Vandalism


Crime Profiles by Sex for Arrested Suspects
1.00
0.75
0.50
0.25
0.00
n
Female
Male
NULL
Unknown
Sex
All Other
Assault, Simple
Burglary
Disorderly Conduct
Drug Offenses
Larceny from Building
Larceny from Motor Vehicle
Larceny, Other
Motor Vehicle Theft
Traffic Violations
Vandalism

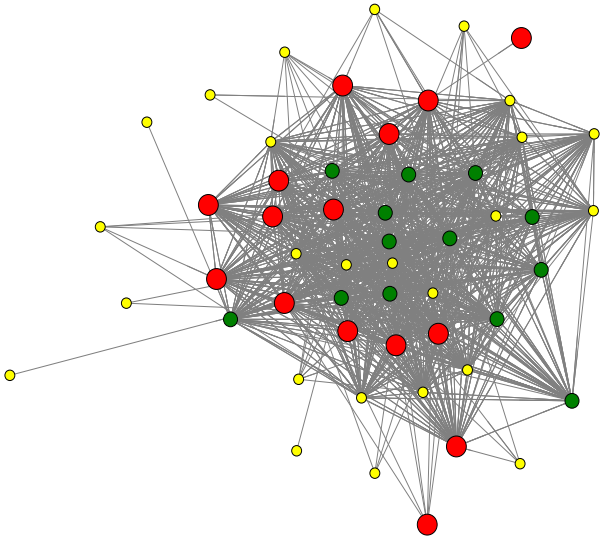

Assault, Aggravated
Robbery
Weapons
Drug Offenses
Vandalism


Traffic Violations
eny from Motor Ve
DUI


Fraud, Swindle
Motor Vehicle Thef

Constructing a Spatial Crime Network

- Each Node is a Census Tract or Blockgroup
- Node *A* has a directed edge to node *B* iff a suspect who lives in *A* committed a crime in *B.*
- We study the topological, spatial, and temporal structures of this network.

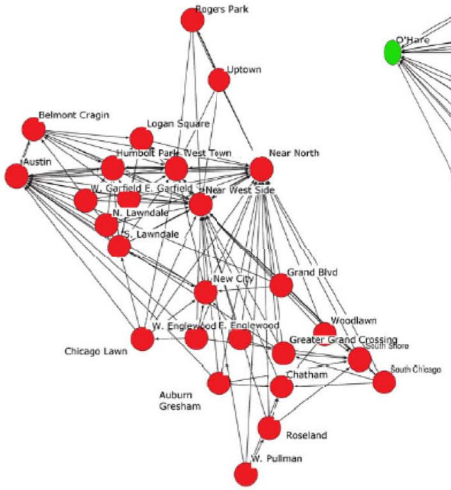


COMPARING CRIME AND COMMUTING NETWORKS

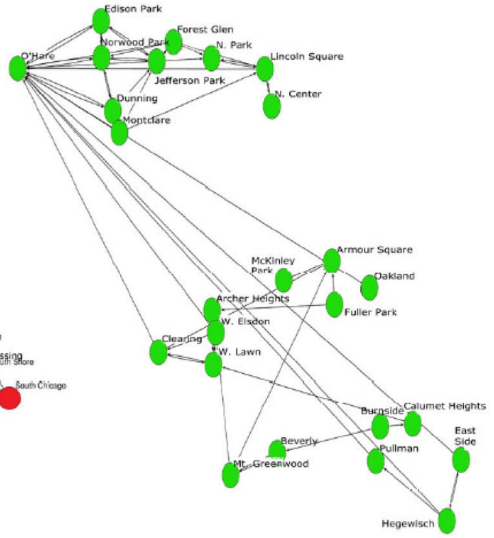
Inter-Neighborhood Networks and Exposure to Violence

Community Areas in Chicago

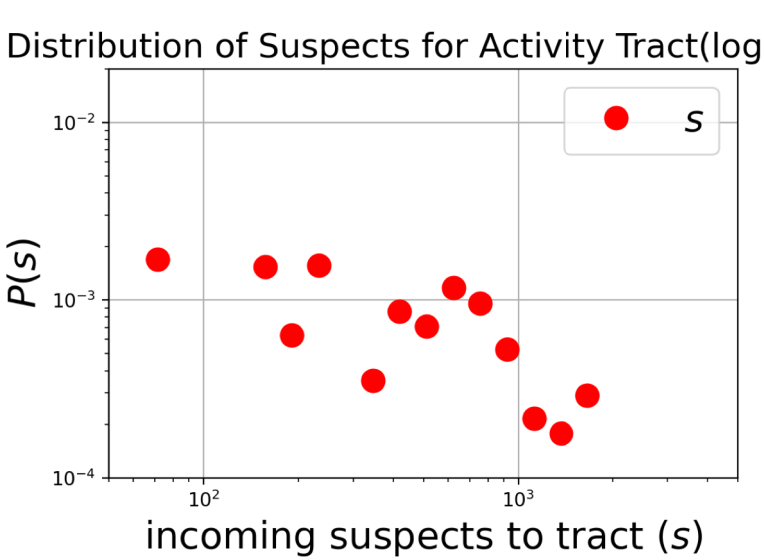
High-Violence Communities Commuting Ties

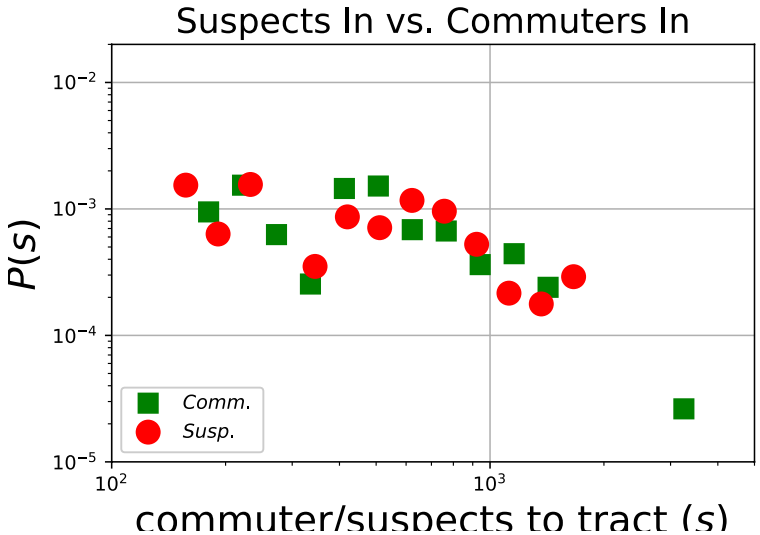


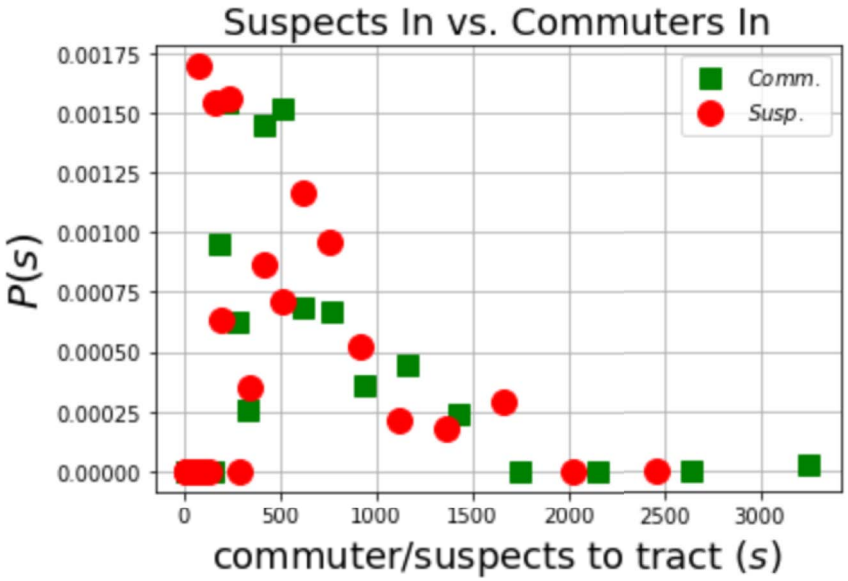
Low-Violence Communities Commuting Ties



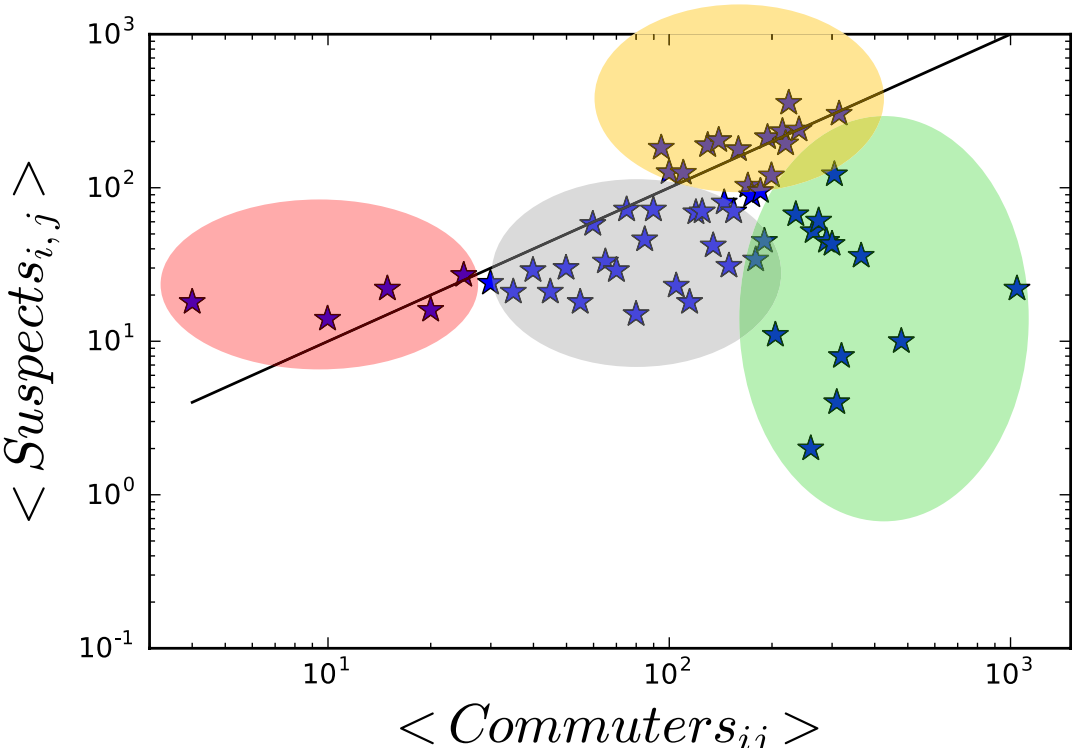
Source: Adapted from Graif (2013).

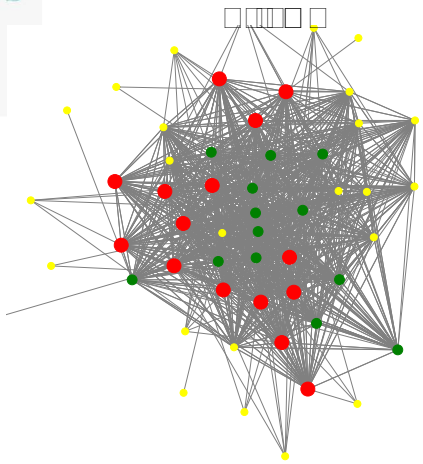

Distribution of Suspects for Activity Tract(log-log)
s
P(s)
incoming suspects to tract (s)

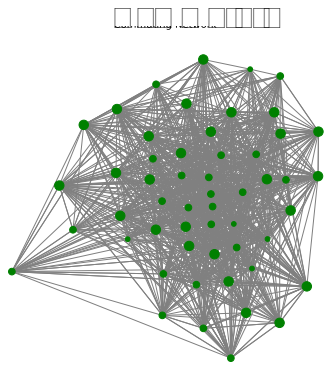

Suspects In vs. Commuters In
Comm.
Susp.
P(s)
commuter/suspects to tract (s)


Suspects In vs. Commuters In
Comm.
Susp.
P(s)
commuter/suspects to tract (s)

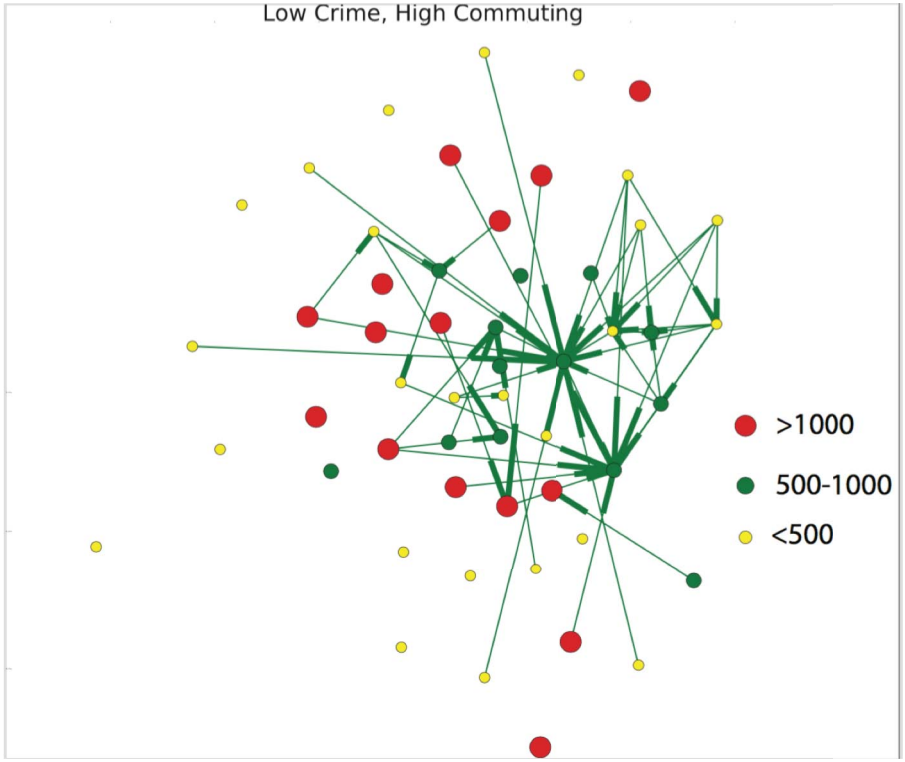
```
eI = [] # commuters>= 100 && suspects <=100
eII = [] # commuters> 100 && suspects >= commuters
eIII = [] # commuters<100 && suspects>commuters
eIV= []  # commuters<100 && suspects<=commuters
```

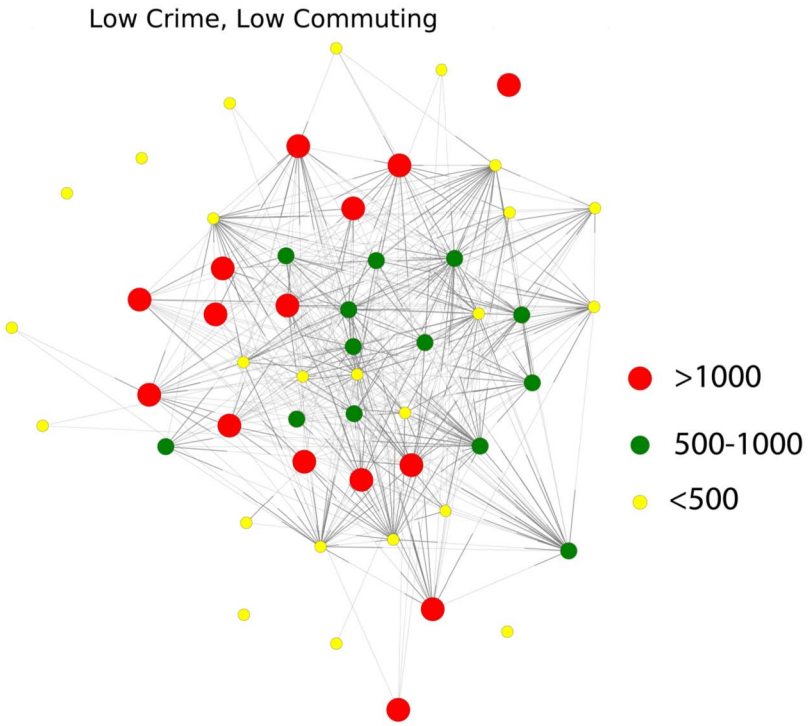


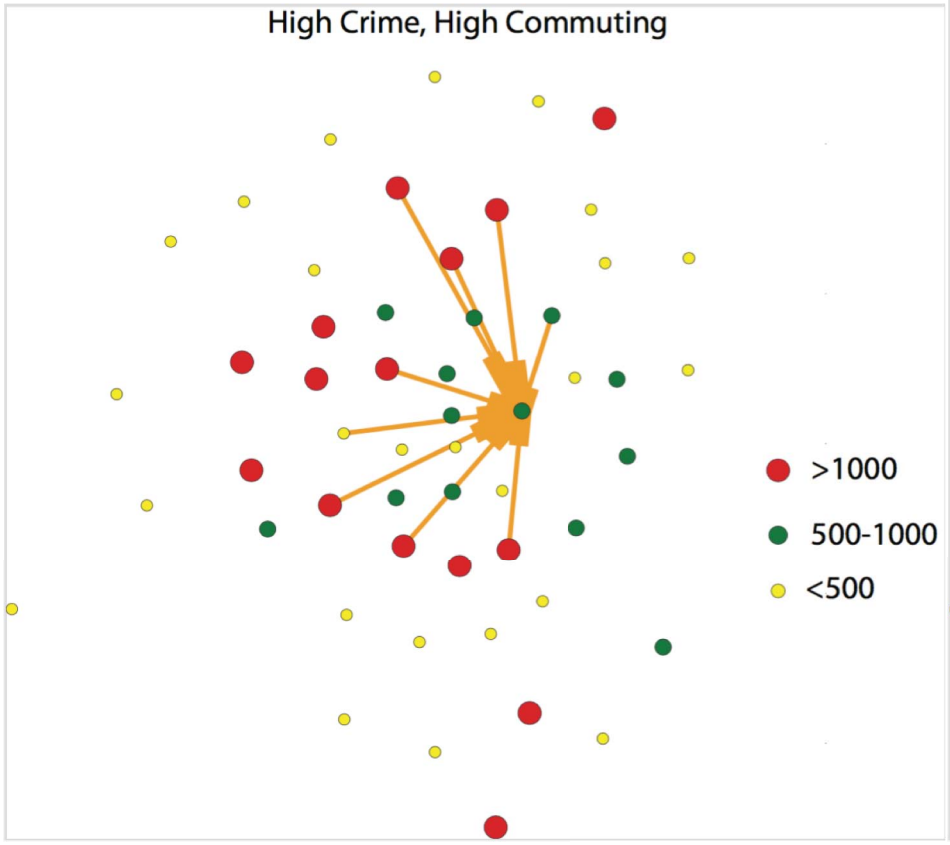


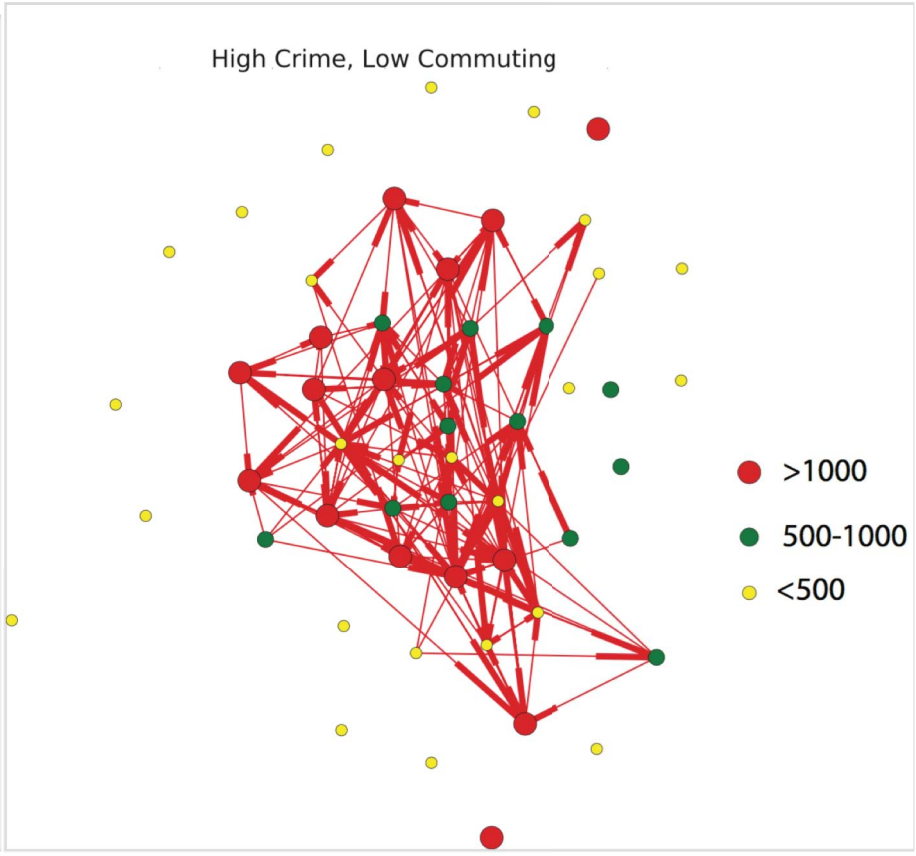


[illegible]


Low Crime, High Commuting
>1000
500-1000
<500


Low Crime, Low Commuting
>1000
500-1000
<500


High Crime, High Commuting
>1000
500-1000
<500


High Crime, Low Commuting
>1000
500-1000
<500

In [1]:
```
import pandas as pd
import networkx as nx # for later
from matplotlib import pyplot as plt
from matplotlib import pylab
%matplotlib inline
```

In [2]:
```
#Read original data
file_path = 'crimes_with_residences2.csv'

crime = pd.read_csv(file_path, dtype = {'blockgroup_residence' : str,
                                        'blockgroup' : str })
##eliminates the last digit to go from block to tract
crime['tract_residence'] = crime.blockgroup_residence.str[:-1]
crime['tract_crime'] = crime.blockgroup.str[:-1]
```

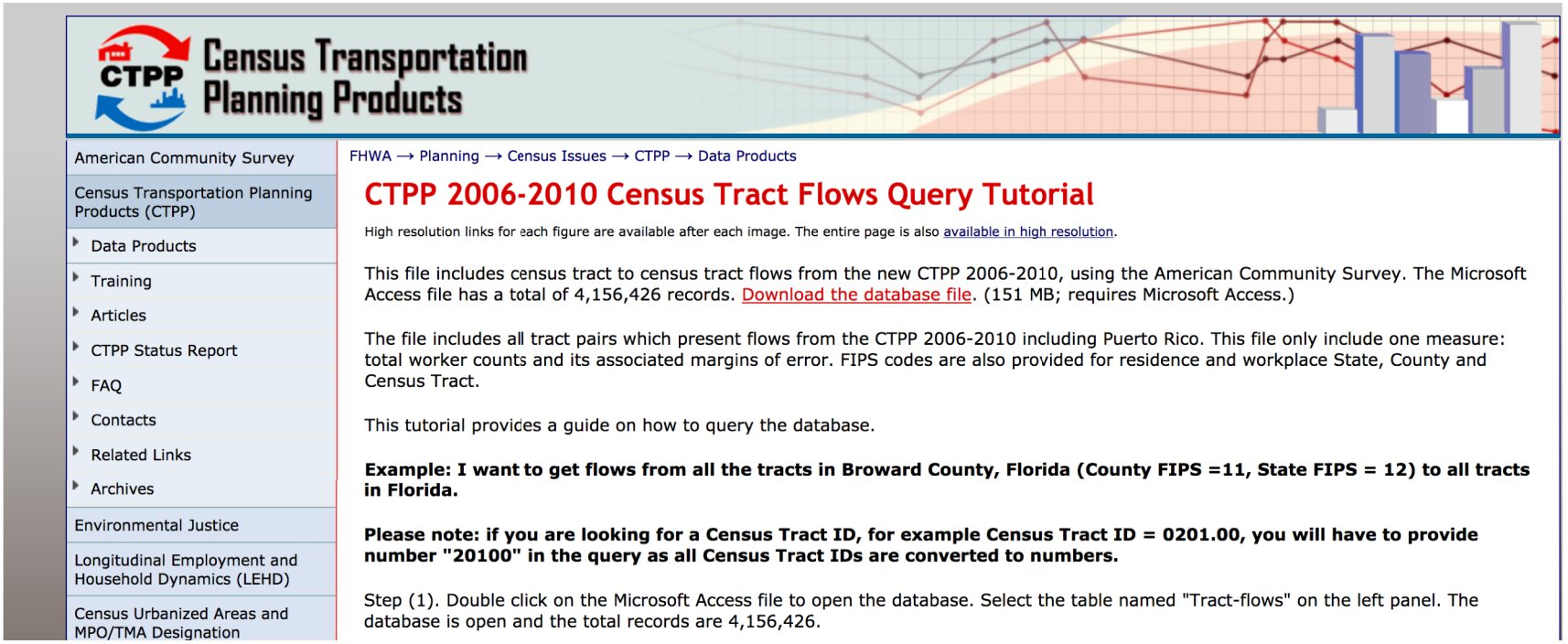
In [3]:
```
crime
```

Out[3]:

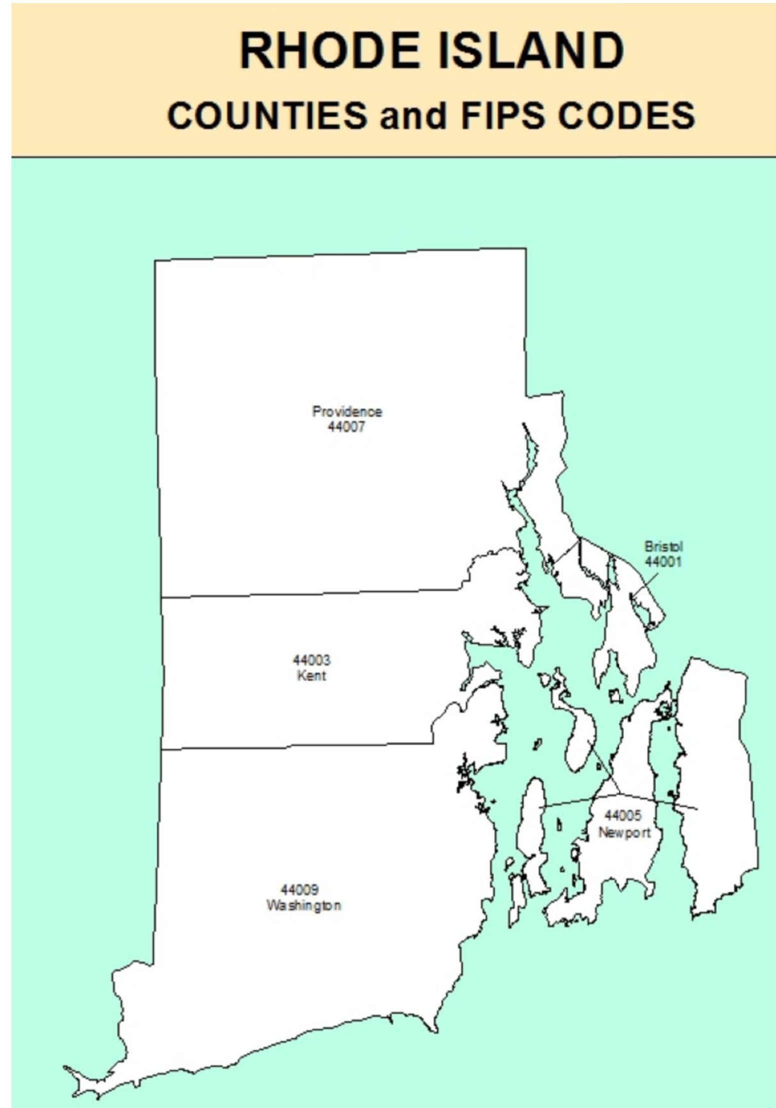
	Unnamed: 0	X	CaseID	CaseNumber	Location	ReportedDate	OccurredFromDate	OccurredThroughDate	OffenseID	IBRCode	.
0	1	154	673371	2010-00118035	50 EATON ST	11/6/10 1:09	11/6/10 1:09	11/6/10 1:09	587077	90Z	.
1	2	155	673371	2010-00118035	50 EATON ST	11/6/10 1:09	11/6/10 1:09	11/6/10 1:09	576015	120	.
2	3	156	623467	2009-00035148	50 EATON ST	4/8/09 20:02	4/8/09 20:02	4/8/09 20:02	519446	23H	.
3	4	157	608066	2008-00116479	50 EATON ST	10/7/08 17:37	10/3/08 12:00	10/7/08 17:30	501469	250	.
4	5	158	754943	2013-00084097	255 ATWELLS AVE	9/4/13 17:19	9/4/13 13:30	9/4/13 13:30	667426	23F	.
5	6	159	640785	2009-00115047	35 OPPER ST	10/23/09 18:41	10/10/09 12:00	10/10/09 12:00	539296	26A	.
6	7	160	629733	2009-00063976	53 LANGDON ST	6/18/09 20:00	10/2/08 12:00	6/18/09 12:00	526670	23H	.
7	8	161	640382	2009-00112156	100 BROAD ST	10/15/09 15:50	10/10/09 10:00	10/10/09 10:30	538867	23D	.
8	9	162	717369	2012-00041141	25 FLORENCE	5/10/12 17:38	5/10/12 17:35	5/10/12 17:35	625004	290	.

4]:

le	...	Race	Ethnicity	GangAffiliation	YearOfBirth	CurrentAge	lat_residence	lon_residence	blockgroup_residence	tract_residence	tract_crim
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700230
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700230
	...	White	NonHispanic	Plain Street Group	1989.0	27.0	41.813327	-71.416567	440070007002	44007000700	4400700230
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700230
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700090
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700290
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700290
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700080
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700190
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700190
	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4400700290


Census Transportation Planning Products
American Community Survey
Census Transportation Planning Products (CTPP)
Data Products
Training
Articles
CTPP Status Report
FAQ
Contacts
Related Links
Archives
Environmental Justice
Longitudinal Employment and Household Dynamics (LEHD)
Census Urbanized Areas and MPO/TMA Designation
FHWA → Planning → Census Issues → CTPP → Data Products
CTPP 2006-2010 Census Tract Flows Query Tutorial
High resolution links for each figure are available after each image. The entire page is also available in high resolution.
This file includes census tract to census tract flows from the new CTPP 2006-2010, using the American Community Survey. The Microsoft Access file has a total of 4,156,426 records. Download the database file. (151 MB; requires Microsoft Access.)
The file includes all tract pairs which present flows from the CTPP 2006-2010 including Puerto Rico. This file only include one measure: total worker counts and its associated margins of error. FIPS codes are also provided for residence and workplace State, County and Census Tract.
This tutorial provides a guide on how to query the database.
Example: I want to get flows from all the tracts in Broward County, Florida (County FIPS =11, State FIPS = 12) to all tracts in Florida.
Please note: if you are looking for a Census Tract ID, for example Census Tract ID = 0201.00, you will have to provide number "20100" in the query as all Census Tract IDs are converted to numbers.
Step (1). Double click on the Microsoft Access file to open the database. Select the table named "Tract-flows" on the left panel. The database is open and the total records are 4,156,426.

```
o,d,flow
44007003602,44007003602,1050
44009050600,44009050600,1020
44009051400,44009051400,940
44005041200,44005041200,855
44005041300,44005041300,775
44005040400,44005040400,755
44005040900,44005040900,705
44001030800,44001030800,680
44001030902,44001030902,605
44009050700,44009050700,595
44009050500,44009050500,565
44005040600,44005040600,555
44007011600,44007011600,550
44009041500,44009041500,540
44009050801,44009050801,535
44009051504,44009051202,525
```



```
GEOID,lon,lat
44007013101,-71.7336512995,41.893438796
44007013102,-71.6268318802,41.8934042838
44007018100,-71.5039655123,42.0140817544
44007018200,-71.4958408666,42.0109868607
44007016300,-71.4107442775,41.8656530502
44007013900,-71.4507705722,41.7592733125
44007010800,-71.3871992349,41.8863167695
44007002000,-71.4630189576,41.8308970773
44007018500,-71.4836897139,41.9958657653
44007001500,-71.4317177411,41.7934010037
44007000200,-71.4191492177,41.7944781293
44007014100,-71.4445650231,41.7970099984
44007017700,-71.5018455943,41.9881941926
44007015200,-71.3855811801,41.8793625053
44007016700,-71.3781412515,41.8722161591
44007016800,-71.3597169573,41.8750218494
44007003200,-71.3988945628,41.844056131
44007017800,-71.5077455866,41.9969508043
44007010400,-71.3790958279,41.8088117981
44007002500,-71.4268535825,41.829374321
44007014400,-71.4717416929,41.7527556458
44007012102,-71.4755859929,41.8469849707
```

In [5]:
```
#Using pandas to generate the edge_list
crime_edge_list = crime.groupby(['tract_residence', 'tract_crime']).CaseID.count()
crime_edge_list = crime_edge_list.reset_index(level=['tract_residence', 'tract_crime'])
```

In [6]:
```
crime_edge_list.head()
```

Out[6]:

	tract_residence	tract_crime	CaseID
0	44007000101	44007000101	225
1	44007000101	44007000102	47
2	44007000101	44007000200	16
3	44007000101	44007000300	40
4	44007000101	44007000400	18

[illegible]

```
#Using pandas to generate the edge_list
crime_edge_list = crime.groupby(['tract_residence', 'tract_crime']).CaseID.count()
```

```
crime_edge_list.tail()
```

```
tract_residence  tract_crime
44007016300      44007000800    2
                 44007002500    5
                 44007003100    2
                 44007003200    1
                 44007003500    1
Name: CaseID, dtype: int64
```

Note: try to understand the steps until here to be able to resuse the codes to create edge list from tables

```
#this step is to add a colum as index and keep the first two as table indexes.
crime_edge_list = crime_edge_list.reset_index(level=['tract_residence', 'tract_crime'])
crime_edge_list.head()
```

```
def prep_G(G,nod_w_coord):
    G_sub = G.copy()
    for n in G.nodes():
        #print(n)
        #list(nod_w_coord)
        if str(n) not in list(nod_w_coord):
            G_sub.remove_node(n)
    return G_sub
```

□ □□ □□□□□□□□□

□ □□□□□□□□□□□□□□□□□□□□

□□□□□□□ □□□ □□□ □□□□

```
crime_nx = nx.DiGraph()    #directed graph
for i in range(len(crime_edge_list)):
    crime_nx.add_edge(crime_edge_list.tract_residence[i],
                      crime_edge_list.tract_crime[i],
                      weight=crime_edge_list.CaseID[i])
```

```
nx.number_of_nodes(crime_nx)
```

```
53
```

[illegible]

```
commuting_nx = nx.DiGraph()    #directed graph
for i in range(len(commuting_edge_list)):
    commuting_nx.add_edge(commuting_edge_list.o[i],
                commuting_edge_list.d[i],
                weight=commuting_edge_list.flow[i])
```

```
commuting_nx = prep_G(commuting_nx,nod_w_coord)
```

```
nx.number_of_nodes(commuting_nx)
```

```
141
```

```
nx.number_of_edges(commuting_nx)
```

```
6007
```

```
commuting_nx_copy=commuting_nx.copy()
for n in commuting_nx_copy.nodes(): # only include
    #print(n)
    if n not in crime_nx.nodes():
        commuting_nx.remove_node(n);
```

```
coord_dict = coords.to_dict(orient = 'index')
pos = {coord_dict[key]['GEOID'] : (coord_dict[key]['lon'],coord_dict[key]['lat']) for key in coord_dict} ###ne
```

```
pos
```

```
{'44007013101': (-71.7336512995, 41.893438796),
 '44007013102': (-71.6268318802, 41.8934042838),
 '44007018100': (-71.5039655123, 42.0140817544),
 '44007018200': (-71.4958408666, 42.0109868607),
 '44007016300': (-71.4107442775, 41.8656530502),
 '44007013900': (-71.4507705722, 41.7592733125),
 '44007010800': (-71.3871992349, 41.8863167695),
```

```
coords = pd.read_csv('tract_coords.csv', dtype = {'GEOID' : str})
coords = coords.set_index(['GEOID'])
coord_dict = coords.to_dict(orient = 'index')
pos = {key : (coord_dict[key]['lon'],coord_dict[key]['lat']) for key in coord_dict}
```

```
len(commuting_edge_list)
```

```
19216
```

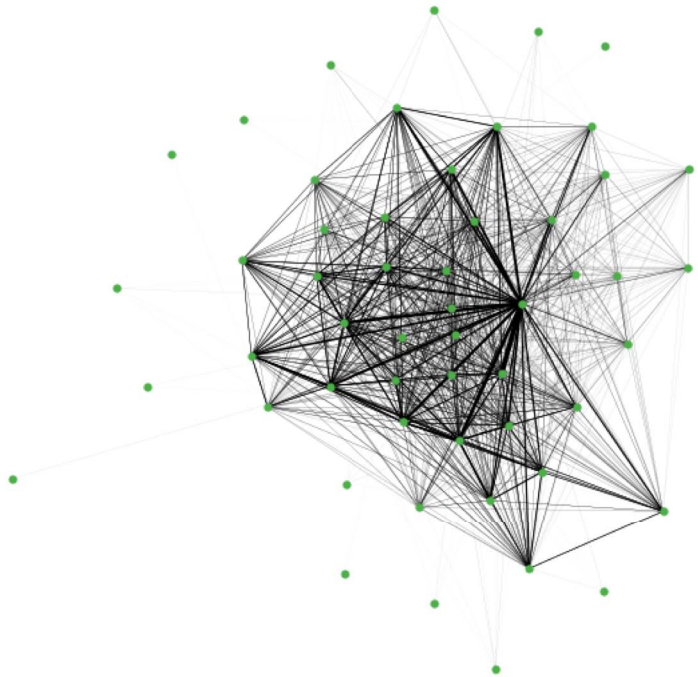
```
len(crime_edge_list)
```

```
1479
```

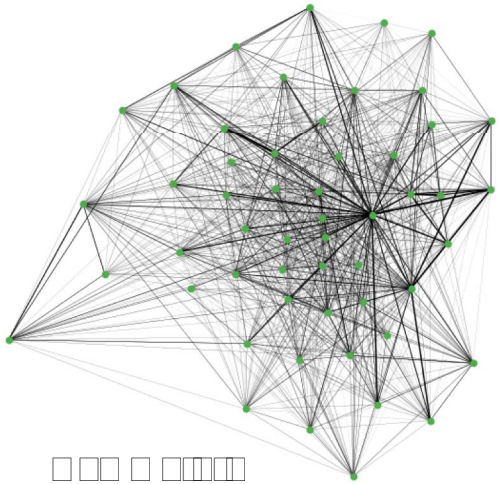
```
len(pos.keys())
```

```
141
```

In [310]:
```
#check information here https://networkx.github.io/documentation/networkx-1.10/reference/
# and also the layouts here https://networkx.github.io/documentation/development/_modules/
edgewidth = [ .01 * w['weight'] for (u,v,w) in crime_nx.edges(data=True)]

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111)

nx.draw(crime_nx,
        ax=ax,
        pos=pos,
        node_color="#4daf4a",
        node_size=40,
        font_size=500,
        linewidths=0,
        font_color='b',
        width = edgewidth,
        alpha = 1,
        arrows = False)

# ax.set_ylim([41.75, 41.86])
# ax.set_xlim([-71.48, -71.36])
```

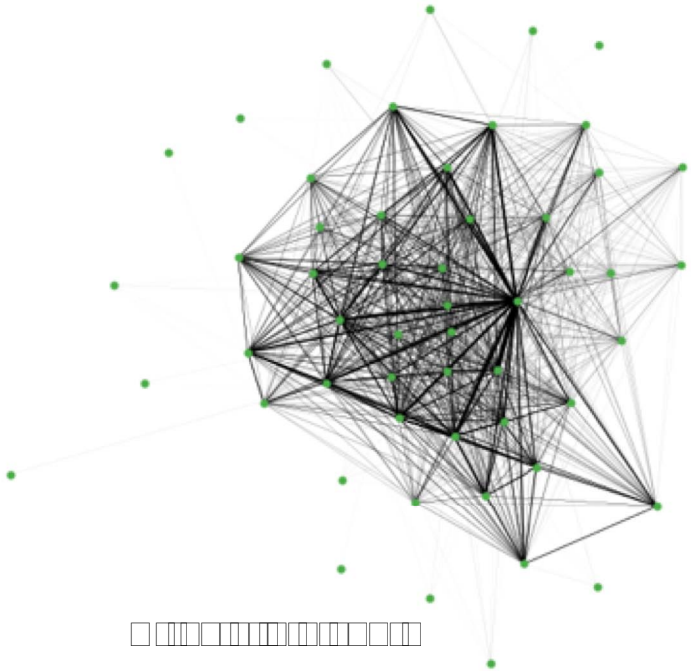


```
edgewidth = [ .005 * w['weight'] for (u,v,w) in commuting_nx.edges(data=True)]

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111)

nx.draw(commuting_nx,
        ax = ax,
        pos = pos,
        node_color="#4daf4a",
        node_size=60,
        font_size=11,
        linewidths=0,
        font_color='w',
        width = edgewidth,
        alpha = 1,
        arrows = False)

# ax.set_ylim([41.75, 41.86])
# ax.set_xlim([-71.48, -71.36])
```



```
print(len(commuting_nx.node))
print(len(commuting_nx.edges()))
print(len(crime_nx.node))
print(len(crime_nx.edges()))
```

```
53
1198
53
1479
```



```
crime_out = list(dict(crime_nx.out_degree(weight='weight')).values());
crime_in = list(dict(crime_nx.in_degree(weight='weight')).values());
```

```
max(crime_out)
```

```
2145
```

```
min(crime_in)
```

```
0
```

```
comm_out = list(dict(commuting_nx.out_degree(weight='weight')).values()
comm_in = list(dict(commuting_nx.in_degree(weight='weight')).values())
```

```
max(comm_out)
```

```
2044
```

```
min(comm_in)
```

```
204
```

Edges properties

```
]: wij = []
Cwij = []
g = nx.Graph();
i=0;
for edge in commuting_nx.edges(data=True):
    i=i+1;
    if crime_nx.has_edge(edge[0], edge[1]):
        wij.append(edge[2]['weight'])
        Cwij.append(crime_nx.get_edge_data(edge[0], edge[1])['weight'])
```

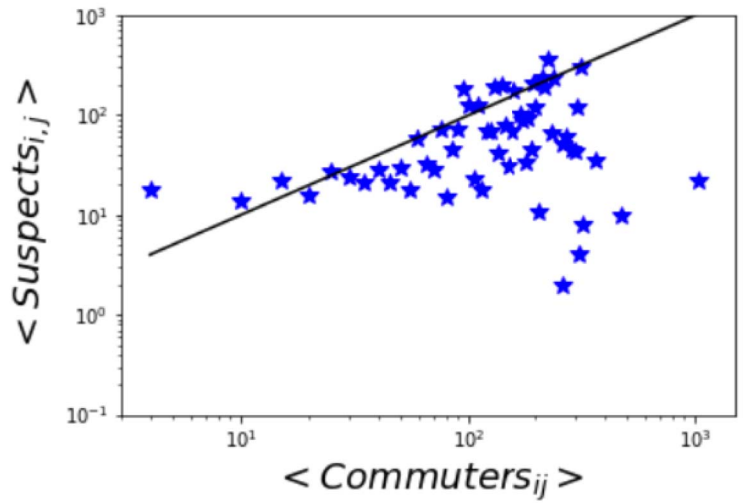
]:
```
len(wij_logBins)
```

]: 1050

]:
```
np.mean(wij)
```

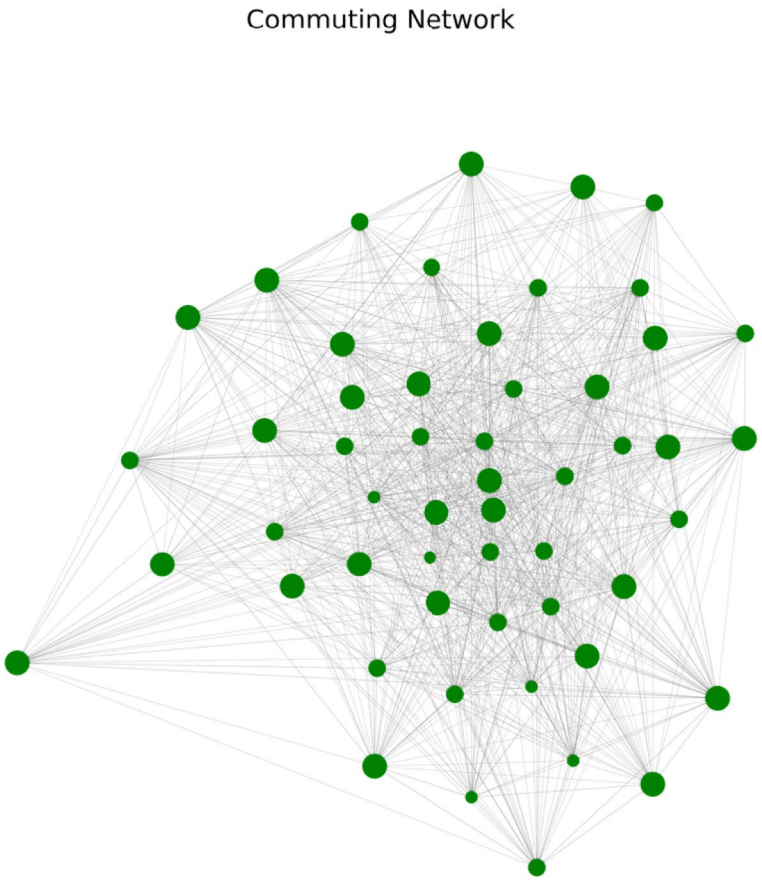
]: 52.92242595204513

]:
```
plt.loglog(bins[:-1],avg_Cw,'*',color='b',markersize=10)
#plt.loglog(out_logBins[:-1],out_logBinDensity,'-o', markersize=8,label=r'$s$',color='b')
plt.xlabel('$<Commuters_{ij}>$',fontsize=20)
plt.ylabel('$<Suspects_{i,j}>$',fontsize=20)
plt.minorticks_off()
plt.loglog(bins[:-1],bins[:-1],'k-')
plt.ylim(0.1,1000)
plt.xlim(3,1500)
plt.savefig("commutes.eps",format='eps', dpi=1000)
```



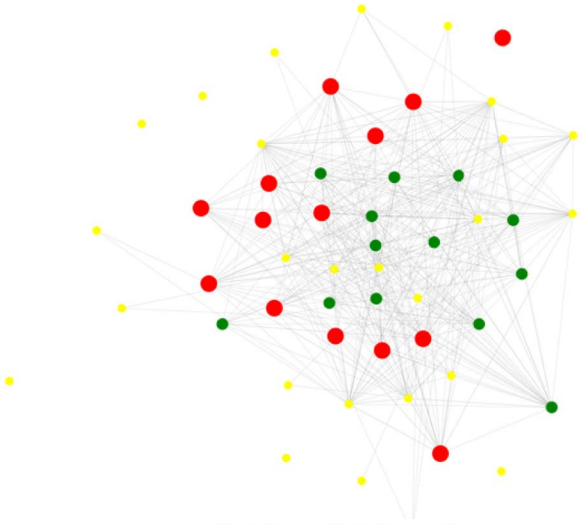
]:
```
import matplotlib.colors as colors
plt.figure(3)


nsc=[]
ncc=[]
comm_out = dict(commuting_nx.out_degree(weight='weight')).values()

for co in comm_out:
   #print co
   if(co>=1000):
      nsc.append(20000)
      ncc.append("green")
   if(co>=500 and co<1000):
      nsc.append(10000)
      ncc.append("green")
   if(co<500):
      nsc.append(5000)
      ncc.append("green")

plt.figure(figsize=(100,100))

nx.draw_networkx(commuting_nx, pos, node_size=nsc, node_color = ncc,edge_color='gray',with_labels=False)
plt.title('Commuting Network',fontsize=150)
plt.axis('off')
plt.savefig("Commuting.eps",format='eps', dpi=1000)
#plt.show() # or display
```

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

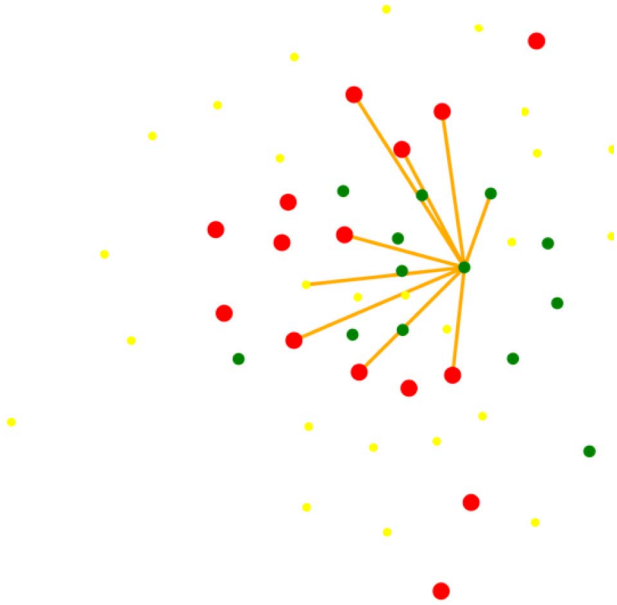


```
In [90]: #here we show the original edgelist
         plt.figure(4)
         plt.figure(figsize=(100,100))
         nx.draw_networkx_edges(crime_nx, pos, edgelist = eIV, width = 1 , alpha = 0.9, edge_color='gray')
         nx.draw_networkx_nodes(crime_nx, pos,node_size=ns,node_color=nc,linewidth=3)
         plt.title('Low Crime, Low Commuting',fontsize=150)
         plt.savefig("lowcrime_lowcomm.eps",format='eps', dpi=1000)

Out[90]: (-71.51063673483586, -71.3728322336259, 41.7582411426665, 41.87943161203349)

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

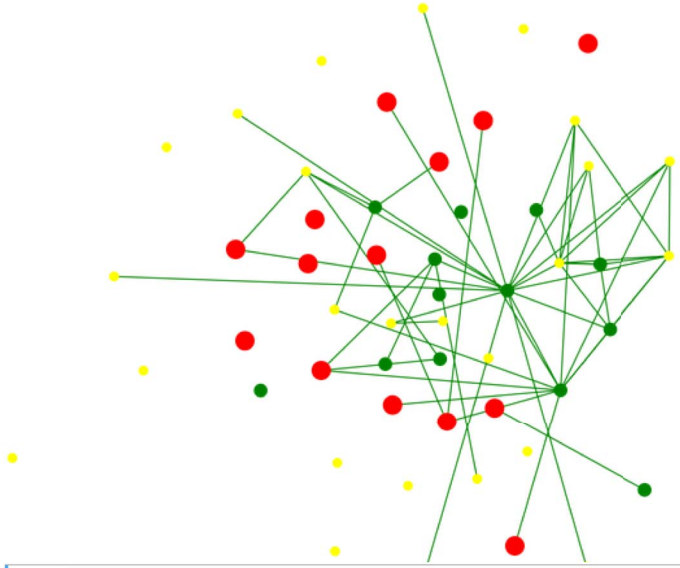
Low Crime, Low Commuting



High Crime, High Commuting

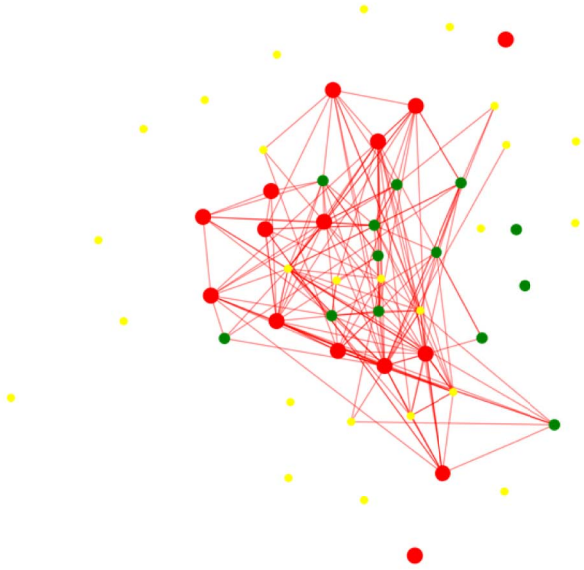


Low Crime, High Commuting

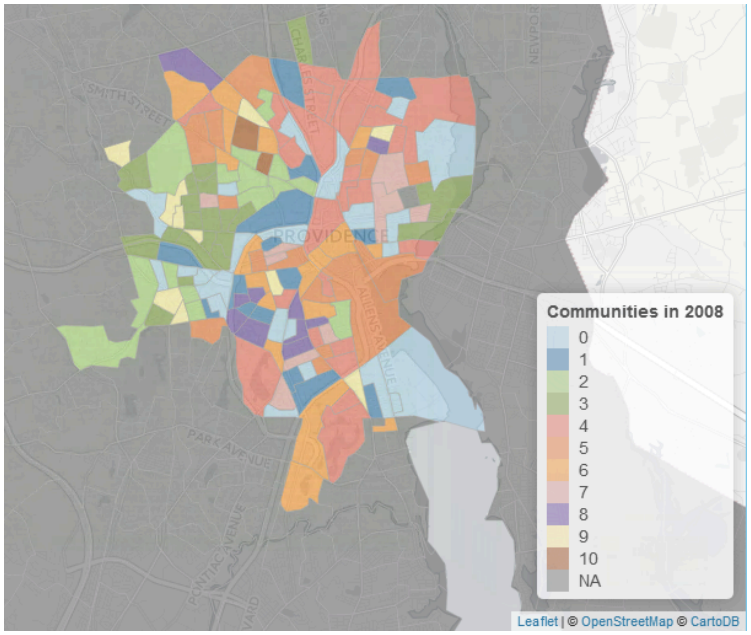


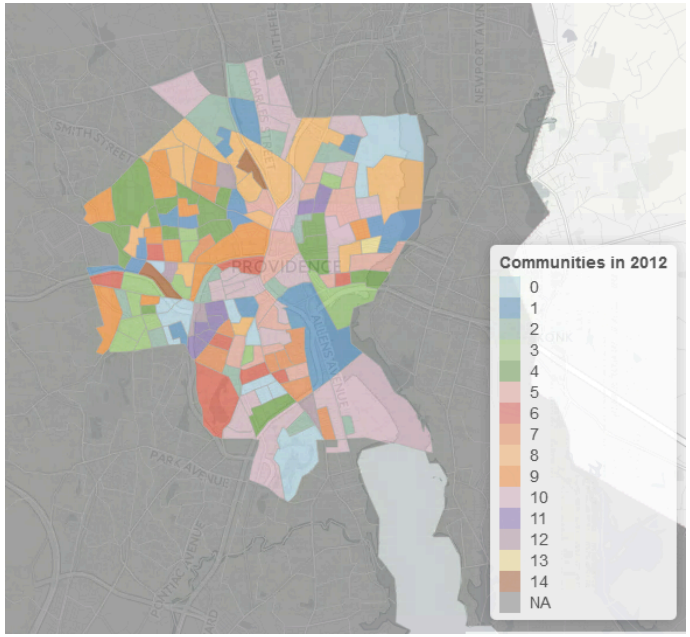
```
In [93]: #here we show the edgelist II, note we can save the files
         plt.figure(7)
         plt.figure(figsize=(100,100))
         nx.draw_networkx_edges(crime_nx, pos, edgelist = eIII, width = 10 , alpha = 0.5, edge_color = 'red') # and others
         nx.draw_networkx_nodes(crime_nx, pos, node_size=ns, node_color = nc, linewidth = 3)
         plt.title('High Crime, Low Commuting',fontsize=150)
         plt.savefig('highcrime_lowcomm.eps',format='eps', dpi=1000)

Out[93]: (-71.51019841904393, -71.37754510835603, 41.762562679512406, 41.87455792595382)
```

High Crime, Low Commuting



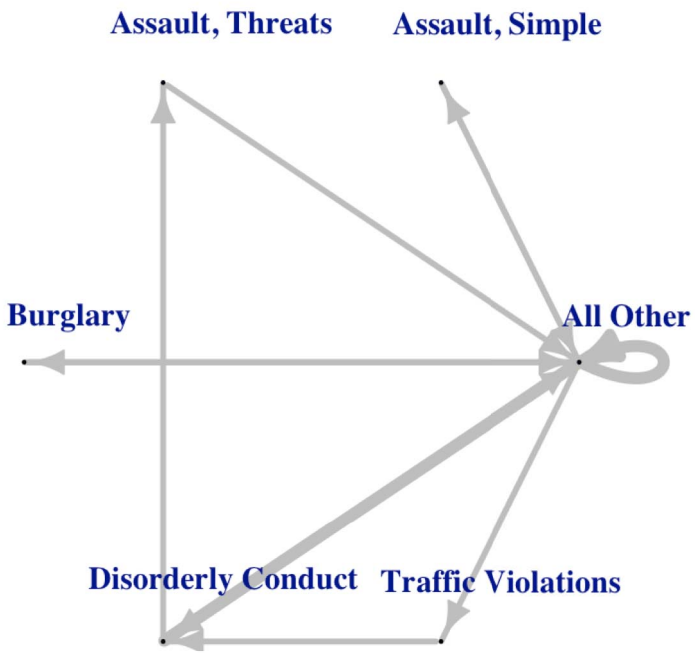
Other posibilities:
Spatial communities vs. Income





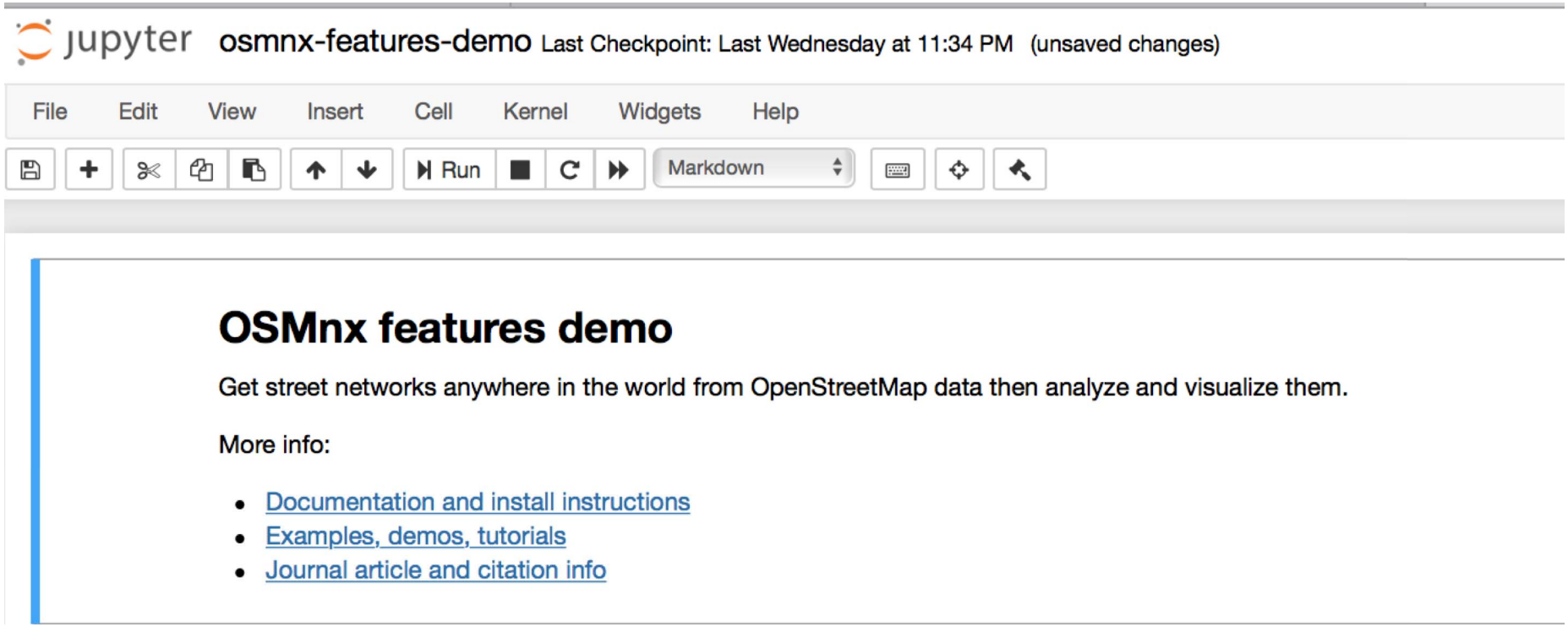
[illegible]

- [illegible]
  - [illegible]
  - [illegible]
- [illegible]
  - [illegible]
  - [illegible]



$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}.$$

□□□□□


jupyter osmnx-features-demo Last Checkpoint: Last Wednesday at 11:34 PM (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Markdown
OSMnx features demo
Get street networks anywhere in the world from OpenStreetMap data then analyze and visualize them.
More info:
Documentation and install instructions
Examples, demos, tutorials
Journal article and citation info

□□ □□□□□□□□□□□□□□□ □□□ □ □□□□□□□□□

In [2]:
```
# get the walking network for piedmont
G = ox.graph_from_place('Piedmont, California, USA', network_type='walk')
fig, ax = ox.plot_graph(G)
```

In [6]:
```
# what sized area does our network cover in square meters?
G_proj = ox.project_graph(G)
nodes_proj = ox.graph_to_gdfs(G_proj, edges=False)
graph_area_m = nodes_proj.unary_union.convex_hull.area
graph_area_m
```

Out[6]:
```
4263712.40555074
```

In [7]:
```
# show some basic stats about the network
ox.basic_stats(G_proj, area=graph_area_m, clean_intersects=True, circuity_dist='euclidean')
```

Out[7]:
```
{'n': 540,
 'm': 1506,
 'k_avg': 5.577777777777775,
 'intersection_count': 476,
 'streets_per_node_avg': 2.8944444444444444,
 'streets_per_node_counts': {0: 0, 1: 64, 2: 0, 3: 409, 4: 64, 5: 2, 6: 1},
 'streets_per_node_proportion': {0: 0.0,
  1: 0.11851851851851852,
  2: 0.0,
  3: 0.7574074074074074,
  4: 0.11851851851851852,
  5: 0.003703703703703704,
  6: 0.001851851851851852},
 'edge_length_total': 137796.308,
 'edge_length_avg': 91.49821248339973,
 'street_length_total': 69283.18899999997,
 'street_length_avg': 91.76581324503307,
 'street_segments_count': 755,
 'node_density_km': 126.65019322058349,
 'intersection_density_km': 111.63979994999582,
 'edge_density_km': 32318.387098672283,
 'street_density_km': 16249.498655163334,
 'circuity_avg': 1.0876239293554333,
 'self_loop_proportion': 0.0026560424966799467,
 'clean_intersection_count': 343,
 'clean_intersection_density_km': 80.44632643455581}
```

---

stats documentation: https://osmnx.readthedocs.io/en/stable/osmnx.html#module-osmnx.stats

stats documentation: https://osmnx.readthedocs.io/en/stable/osmnx.html#module-osmnx.stats

```
# see more stats (mostly topological stuff) with extended_stats
more_stats = ox.extended_stats(G, ecc=True, bc=True, cc=True) #use arguments to turn other toplogical analyses or
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
```

```
# pull up some stat's value
more_stats['radius']
```

```
2507.4629999999997
```

```
import networkx as nx
import matplotlib.pyplot as plt
```

```
# save graph to disk as shapefile (for GIS) or graphml file (for gephi etc)
ox.save_graph_shapefile(G, filename='mynetwork_shapefile')
ox.save_graphml(G, filename='mynetwork.graphml')
```

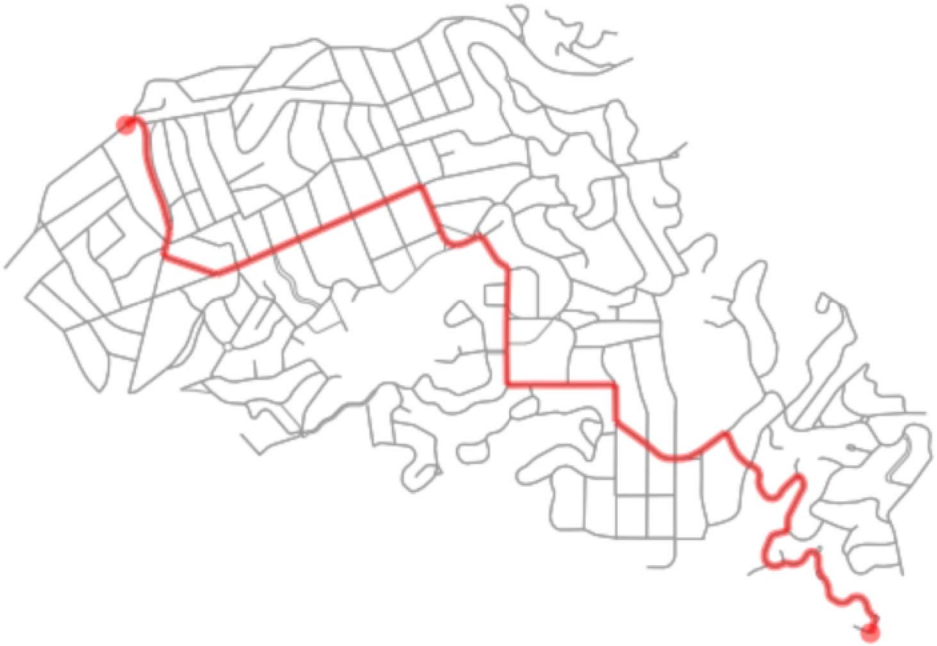
Visualize street centrality

```
# edge closeness centrality: convert graph to line graph so edges become nodes and vice versa
# use nx <> 2.1 to avoid a networkx bug
edge_centrality = nx.closeness_centrality(nx.line_graph(G))
```

```
# list of edge values for the orginal graph
ev = [edge_centrality[edge + (0,)] for edge in G.edges()]

# color scale converted to list of colors for graph edges
norm = colors.Normalize(vmin=min(ev)*0.8, vmax=max(ev))
cmap = cm.ScalarMappable(norm=norm, cmap=cm.inferno)
ec = [cmap.to_rgba(cl) for cl in ev]

# color the edges in the original graph with closeness centralities in the line graph
fig, ax = ox.plot_graph(G, bgcolor='k', axis_off=True, node_size=0,
                        edge_color=ec, edge_linewidth=1.5, edge_alpha=1)
```

```
# find the shortest path between these nodes by time then plot it
route = nx.shortest_path(G, orig_node, dest_node, weight='time')
```

```
fig, ax = ox.plot_graph_route(G, route, node_size=0)
```



```
# how long is our route in minutes?
nx.shortest_path_length(G, orig_node, dest_node, weight='time')
```

```
6.776646575614848
```

Get networks other ways

make queries less ambiguous to help the geocoder out if it's not finding what you're looking for

```
]: # OSM doesn't geocode this simple query to what you might expect...
   place = 'San Francisco, California, USA'
   G = ox.graph_from_place(place, network_type='drive')
```

```
]: # let's see what it matches that query string to...
   url = 'https://nominatim.openstreetmap.org/search?format=json&limit=1&dedupe=0&polygon_geojson=1&q={}'
   url = url.format(place)
   response = requests.get(url)
   response.json()
```


```
        [-123.141495, 37.72616],
        [-123.132597, 37.719393],
        [-123.126114, 37.71673],
        [-123.119645, 37.715441],
        [-123.115575, 37.714045],
        [-123.104921, 37.712648],
        [-123.099647, 37.713591],
        [-123.094801, 37.714067],
        [-123.09285, 37.713688],
        [-123.0918, 37.711872],
        [-123.0899301, 37.7081025],
        [-123.088784, 37.705792],
        [-123.086057, 37.70075],
        [-123.08188, 37.69684],
        [-123.077155, 37.692324],
        [-123.074921, 37.690066],
        [-123.073063, 37.682583],
        [-123.068589, 37.673041],
        [-123.063578, 37.667316],
        [-123.05358, 37.658648],
        [-123.040696, 37.650328],
```

```
# make query an unambiguous dict to help the geocoder find specifically what you're looking for
place = {'city' : 'San Francisco',
         'state' : 'California',
         'country' : 'USA'}
G = ox.graph_from_place(place, network_type='drive')
fig, ax = ox.plot_graph(G, fig_height=12, node_size=0, edge_linewidth=0.5)
```

[illegible]

In [27]:
```
import googlemaps
from datetime import datetime
my_gkey='AIzaSyC34x3dNYkcg4W_yXKbXr9eUb2eZs1-ldE'

gmaps = googlemaps.Client(key=my_gkey)
```

In [31]:
```
# Geocoding an address
geocode_result = gmaps.geocode('1600 Amphitheatre Parkway, Mountain View, CA')
```

In [32]:
```
geocode_result
```

Out[32]:
```
[{'address_components': [{'long_name': '1600',
    'short_name': '1600',
    'types': ['street_number']},
   {'long_name': 'Amphitheatre Parkway',
    'short_name': 'Amphitheatre Pkwy',
    'types': ['route']},
   {'long_name': 'Mountain View',
    'short_name': 'Mountain View',
    'types': ['locality', 'political']},
   {'long_name': 'Santa Clara County',
    'short_name': 'Santa Clara County',
    'types': ['administrative_area_level_2', 'political']},
   {'long_name': 'California',
    'short_name': 'CA',
    'types': ['administrative_area_level_1', 'political']},
   {'long_name': 'United States',
    'short_name': 'US',
    'types': ['country', 'political']},
   {'long_name': '94043', 'short_name': '94043', 'types': ['postal_code']}],
  'formatted_address': '1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA',
  'geometry': {'location': {'lat': 37.4213117, 'lng': -122.0839677},
   'location_type': 'ROOFTOP',
   'viewport': {'northeast': {'lat': 37.4226606802915,
     'lng': -122.0826187197085},
    'southwest': {'lat': 37.4199627197085, 'lng': -122.0853166802915}}},
  'place_id': 'ChIJtYuu0V25j4ARwu5e4wwRYgE',
  'plus_code': {'compound_code': 'CWC8+GC Mountain View, California, United States',
   'global_code': '849VCWC8+GC'},
  'types': ['street_address']}]
```

[illegible]

[illegible]

```
yadd=geocode_result[0]['geometry']['location'] ['lat']
```

```
xadd=geocode_result[0]['geometry']['location'] ['lng']
```

```
address_coord=(yadd,xadd)
```

```
address_coord
```

```
(37.4213117, -122.0839677)
```

See a short tutorial here: https://towardsdatascience.com/how-to-generate-lat-and-long-coordinates-from-an-address-column-using-pandas-and-googlemaps-api-d66b2720248d

```
# Look up an address with reverse geocoding
reverse_geocode_result = gmaps.reverse_geocode((40.714224, -73.961452))
```

```
reverse_geocode_result
```

```
[{'address_components': [{'long_name': '279',
     'short_name': '279',
     'types': ['street_number']},
    {'long_name': 'Bedford Avenue',
     'short_name': 'Bedford Ave',
     'types': ['route']},
    {'long_name': 'Williamsburg',
     'short_name': 'Williamsburg',
     'types': ['neighborhood', 'political']},
    {'long_name': 'Brooklyn',
     'short_name': 'Brooklyn',
     'types': ['political', 'sublocality', 'sublocality_level_1']},
    {'long_name': 'Kings County',
     'short_name': 'Kings County',
     'types': ['administrative_area_level_2', 'political']},
    {'long_name': 'New York',
     'short_name': 'NY',
     'types': ['administrative_area_level_1', 'political']},
    {'long_name': 'United States',
     'short_name': 'US',
     'types': ['country', 'political']},
```

```
# Request directions via public transit
now = datetime.now()
```

```
now
```

```
datetime.datetime(2020, 4, 20, 9, 33, 7, 302620)
```

```
directions_result = gmaps.directions("Sydney Town Hall",
                                     "Parramatta, NSW",
                                     mode="transit",
                                     departure_time=now)
```

```
directions_result
```

```
     'value': 1587389675},
    'distance': {'text': '25.3 km', 'value': 25317},
    'duration': {'text': '46 mins', 'value': 2743},
    'end_address': 'Parramatta NSW 2150, Australia',
    'end_location': {'lat': -33.8135864, 'lng': 151.0034152},
    'start_address': '483 George St, Sydney NSW 2000, Australia',
    'start_location': {'lat': -33.8732103, 'lng': 151.2070107},
    'steps': [{'distance': {'text': '0.1 km', 'value': 104},
```

Run:

jupyter nbextension enable --py gmaps from the command line

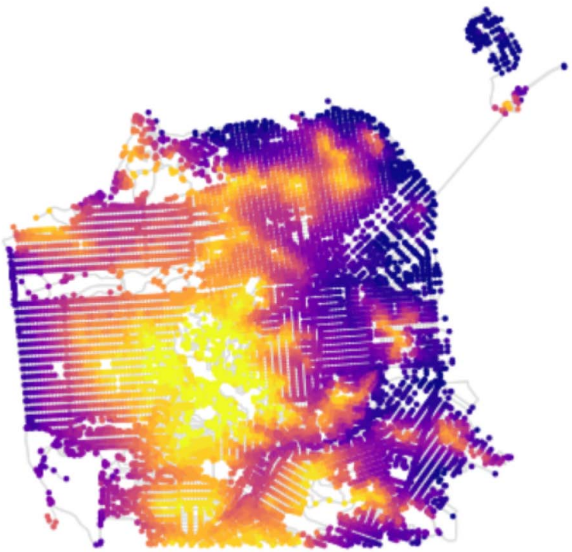
see a tutorial here:https://medium.com/future-vision/google-maps-in-python-part-2-393f96196eaf

```
import gmaps
gmaps.configure(api_key=my_gkey)
```

```
gmaps.figure(center=address_coord, zoom_level=12)
```

```python
# add elevation to nodes automatically, calculate edge grades, plot network
#from keys import google_elevation_api_key
googe_elevation_api_key='AIzaSyBKhBN3i0al6SHmKDqDgTzcUX1JvxcSTS8'
G = ox.add_node_elevations(G,api_key=googe_elevation_api_key)
G = ox.add_edge_grades(G)
```

```python
nc =ox.get_node_colors_by_attr(G, 'elevation', cmap='plasma', num_bins=20)
fig, ax = ox.plot_graph(G, fig_height=6, node_color=nc, node_size=12, node_zorder=2, edge_color='#dddddd')
```

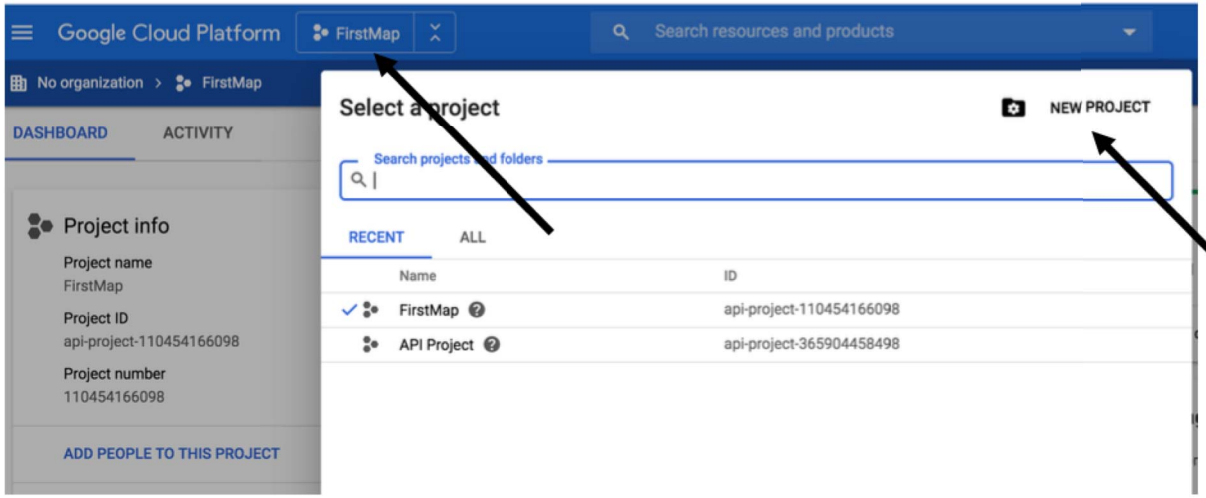


example: create impedance functions to route around hills: https://github.com/gboeing/osmnx-examples/blob/master/notebooks/12-node-elevations-edge-grades.ipynb

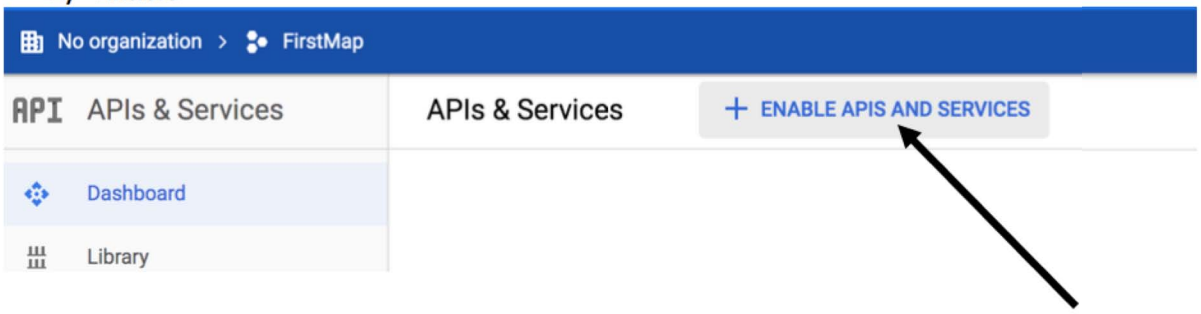
Lecture 11 needs that you install
omsnx
geopandas

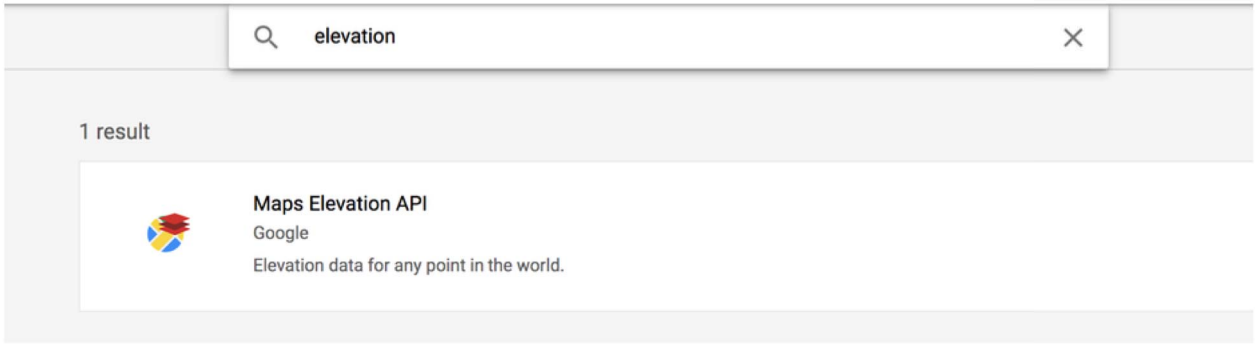
And google keys
Instructions: https://cloud.google.com/apis/docs/getting-started

1) Create and name a project or use a existing one



2) Enable



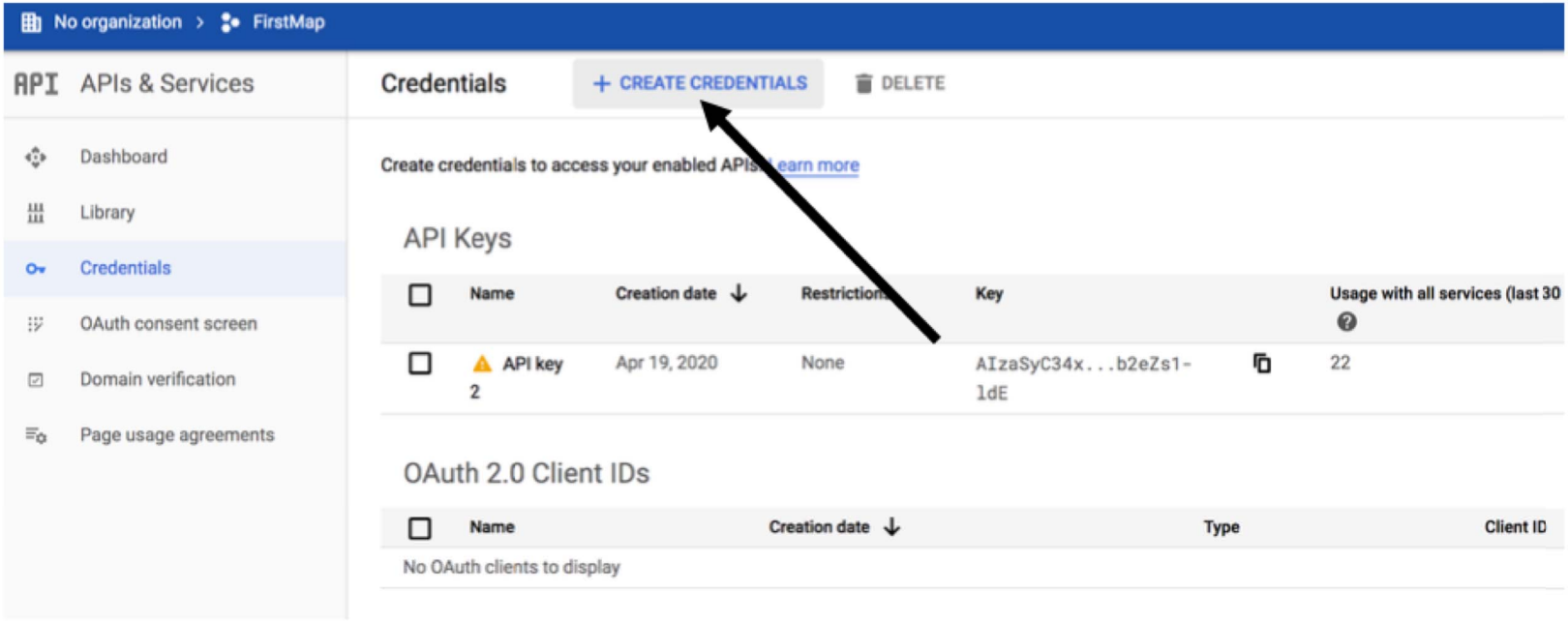


Activate various mapping APIS

Filter

Name	Requests	Errors (%)	Latency, median (ms)	Latency, 95% (ms)
Compute Engine API	39	0	221	537
Maps Elevation API	29	3	120	397
Geocoding API	10	30	65	393
Directions API	8	50	65	978
Maps JavaScript API	4	0	87	126
Cloud OS Login API				
Distance Matrix API				
Geolocation API				
Google Picker API				
Maps Embed API				
Maps Static API				
Roads API				
Street View Static API				
Time Zone API				


Instructions_Geocoding.pdf


No organization
FirstMap
APIs & Services
Dashboard
Library
Credentials
OAuth consent screen
Domain verification
Page usage agreements
Credentials
+ CREATE CREDENTIALS
DELETE
Create credentials to access your enabled APIs
Learn more
API Keys
Name
Creation date
Restrictions
Key
Usage with all services (last 30
API key 2
Apr 19, 2020
None
AIzaSyC34x...b2eZs1-ldE
22
OAuth 2.0 Client IDs
Name
Creation date
Type
Client ID
No OAuth clients to display

```
# you can get networks anywhere in the world
G = ox.graph_from_place('Modena, Italy', network_type='drive_service')
fig, ax = ox.plot_graph(G, fig_height=8, node_size=0, edge_linewidth=0.5)
```

```
# or get network by coordinates, bounding box, or any custom polygon shape
# useful when OSM just doesn't have a polygon for the place you want
davis_hall = (37.8745656,-122.2580841)
one_mile = 1609 #meters
G = ox.graph_from_point(davis_hall, distance=one_mile, network_type='walk')
fig, ax = ox.plot_graph(G, fig_height=8, node_size=0)
```

examples of getting networks by coordinates, bounding box, or any custom polygon shape: https://github.com/gboeing/osmnx-examples/blob/master/notebooks/01-overview-osmnx.ipynb

Get other infrastructure types

like rail or electric grids

50]:
```
# get rail network
# note this is rail *infrastructure* and thus includes crossovers, sidings, spurs, yards, etc
# for station-based rail network, you should prob download a station adjacency matrix elsewhere
G = ox.graph_from_place('New York City, New York',
                        retain_all=False, truncate_by_edge=True, simplify=True,
                        network_type='none', infrastructure='way["railway"~"subway"]')

fig, ax = ox.plot_graph(G, fig_height=10, node_size=0)
```

Merge multiple graphs together

```
]: place1 = {'city':'Albany', 'state':'California', 'country':'USA'}
   place2 = {'city':'El Cerrito', 'state':'California', 'country':'USA'}
   G1 = ox.graph_from_place(place1, network_type='drive', buffer_dist=100)
   G2 = ox.graph_from_place(place2, network_type='drive', buffer_dist=100)
   G = nx.compose(G1, G2)
   fig, ax = ox.plot_graph(G, node_size=0)
```

Geopandas and DataFrames

- Read Census information by geographic unit and color it in a map

Reading_CensusShapefiles_wKey.ipynb

Use conda install -c conda-forge gdal

The first step is to use `PyShp` to read in our Shapefile. If we were just reading a Shapefile from disk we would simply call:

```
r = shapefile.Reader("myshp.shp")
```

However, because we are reading our Shapefile from a zipfile, we will need to specify the individual components of this file. Unfortunately, we can't read the contents of a zipfile directly into Python from the URL. While we could have Python download the file to disk and then read it in from there, I want to demonstrate how this can be done all in memory without any disk activity. Therefore we need to take an additional step where the data from the URL is read into a `StringIO` object, which is in turn read by Python as a zip file. The `StringIO` object is just an intermediate data type between the data read in from the URL and the zip file in Python. This is done in one line of code below:

]:]:
```
import geopandas as gpd
import requests
import zipfile
import io
import matplotlib.pyplot as plt
%matplotlib inline
```

]:]:
```
print(gpd.__version__)
```

```
0.7.0
```

]:]:
```
url = 'http://www2.census.gov/geo/tiger/GENZ2015/shp/cb_2015_us_county_500k.zip'
local_path = 'tmp/'
print('Downloading shapefile...')
r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
print("Done")
z.extractall(path=local_path) # extract to folder
filenames = [y for y in sorted(z.namelist()) for ending in ['dbf', 'prj', 'shp', 'shx'] if y.endswith(ending)]
print(filenames)
```

```
Downloading shapefile...
Done
['cb_2015_us_county_500k.dbf', 'cb_2015_us_county_500k.prj', 'cb_2015_us_county_500k.shp', 'cb_2015_us_county_5
00k.shx']
```

]:]:
```
dbf, prj, shp, shx = [filename for filename in filenames]
usa = gpd.read_file(local_path + shp)
print("Shape of the dataframe: {}".format(usa.shape))
print("Projection of dataframe: {}".format(usa.crs))
usa.tail() #last 5 records in dataframe
```

```
Shape of the dataframe: (3233, 10)
Projection of dataframe: epsg:4269
```

	STATEFP	COUNTYFP	COUNTYNS	AFFGEOID	GEOID	NAME	LSAD	ALAND	AWATER	geometry
3228	45	019	01252740	0500000US45019	45019	Charleston	06	2372842394	1144346152	MULTIPOLYGON (((-79.50795 33.02008, -79.50713 ...
3229	45	077	01248015	0500000US45077	45077	Pickens	06	1285536060	40612589	MULTIPOLYGON (((-82.86687 34.61742, -82.86451 ...
3230	46	123	01265784	0500000US46123	46123	Tripp	06	4176233698	13272785	POLYGON ((-100.23091 43.49989, -100.23044 43.5...
3231	47	073	01639752	0500000US47073	47073	Hawkins	06	1261443215	32545400	POLYGON ((-83.28890 36.37879, -83.28250 36.382...
3232	48	011	01383791	0500000US48011	48011	Armstrong	06	2354581764	12219587	POLYGON ((-101.62940 34.75006, -101.62806 34.8...

Now this zipfile object can be treated as any other zipfile that was read in from disk. Below I identify the filenames of the 4 necessary components of the Shapefile. Note that the `prj` file is actually not 100% necessary, but in contains the coordinate reference system information which is really nice to have, and will be used below.

```
print(len(usa))
```

```
3233
```

```
nj = usa[usa.STATEFP=='44']
```

```
ax = nj.plot(color = 'green', figsize=(10,10),linewidth=2)
ax.set(xticks=[], yticks=[])
plt.savefig("NJ_Counties.png", bbox_inches='tight')
```

```
print(nj.head())
```

```
     STATEFP COUNTYFP  COUNTYNS        AFFGEOID  GEOID        NAME LSAD  \
645       44       009  01219782  0500000US44009  44009  Washington   06
724       44       005  01219779  0500000US44005  44005     Newport   06
871       44       007  01219781  0500000US44007  44007  Providence   06
1514      44       001  01219777  0500000US44001  44001     Bristol   06
2106      44       003  01219778  0500000US44003  44003        Kent   06

          ALAND     AWATER                                           geometry
645    852834829  604715929  MULTIPOLYGON (((-71.61313 41.16028, -71.61053 ...
724    265220482  546983257  MULTIPOLYGON (((-71.28802 41.64558, -71.28647 ...
871   1060637931   67704263  POLYGON ((-71.79924 42.00806, -71.76601 42.009...
1514    62550804   53350670  POLYGON ((-71.35390 41.75130, -71.34718 41.756...
2106   436515567   50720026  POLYGON ((-71.78967 41.72457, -71.75435 41.725...
```

Modify here to extract census tracts of CA

```
url = 'http://www2.census.gov/geo/tiger/GENZ2015/shp/cb_2015_06_tract_500k.zip'
local_path = 'tmp/'
print('Downloading shapefile...')
r = requests.get(url)
z = zipfile.ZipFile(io.BytesIO(r.content))
print("Done")
z.extractall(path=local_path) # extract to folder
filenames = [y for y in sorted(z.namelist()) for ending in ['dbf', 'prj', 'shp', 'shx'] if y.endswith(ending)]
print(filenames)
```

```
Downloading shapefile...
Done
['cb_2015_06_tract_500k.dbf', 'cb_2015_06_tract_500k.prj', 'cb_2015_06_tract_500k.shp', 'cb_2015_06_tract_500k.
shx']
```

```
dbf, prj, shp, shx = [filename for filename in filenames]
usa = gpd.read_file(local_path + shp)
print("Shape of the dataframe: {}".format(usa.shape))
print("Projection of dataframe: {}".format(usa.crs))
usa.tail() #last 5 records in dataframe
```

```
Shape of the dataframe: (8043, 10)
Projection of dataframe: epsg:4269
```

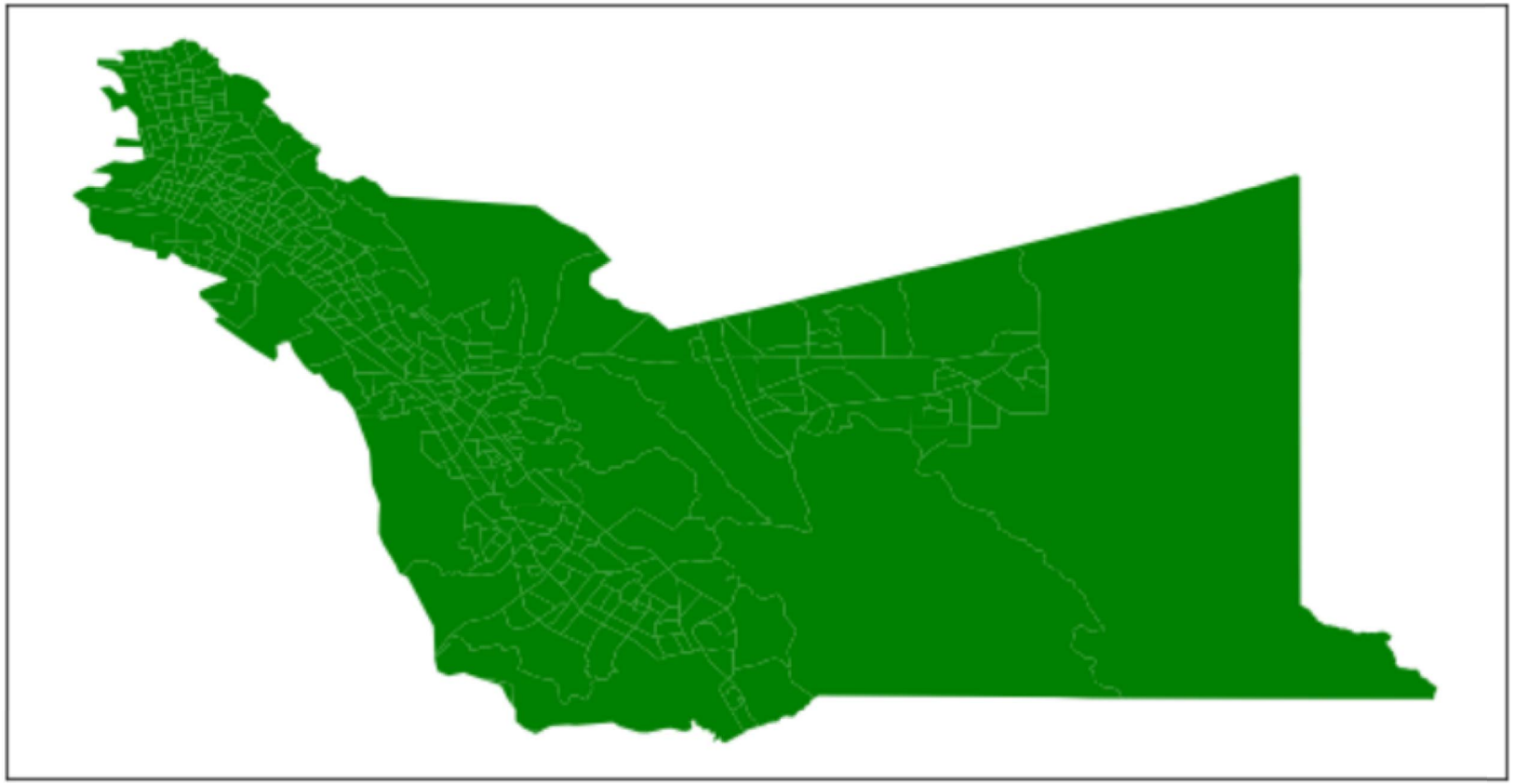
	STATEFP	COUNTYFP	TRACTCE	AFFGEOID	GEOID	NAME	LSAD	ALAND	AWATER	geometry
8038	06	067	007426	1400000US06067007426	06067007426	74.26	CT	868096	0	POLYGON ((-121.38701 38.71209, -121.38563 38.7...
8039	06	113	010203	1400000US06113010203	06113010203	102.03	CT	10247745	756079	POLYGON ((-121.58346 38.57860, -121.58277 38.5...
8040	06	073	018611	1400000US06073018611	06073018611	186.11	CT	24598731	92324	POLYGON ((-117.29814 33.26156, -117.29753 33.2...
8041	06	059	110301	1400000US06059110301	06059110301	1103.01	CT	1835530	0	POLYGON ((-118.03749 33.85865, -118.03315 33.8...
8042	06	061	021122	1400000US06061021122	06061021122	211.22	CT	3119506	0	POLYGON ((-121.25720 38.82472, -121.25450 38.8...

```
len(usa)
```

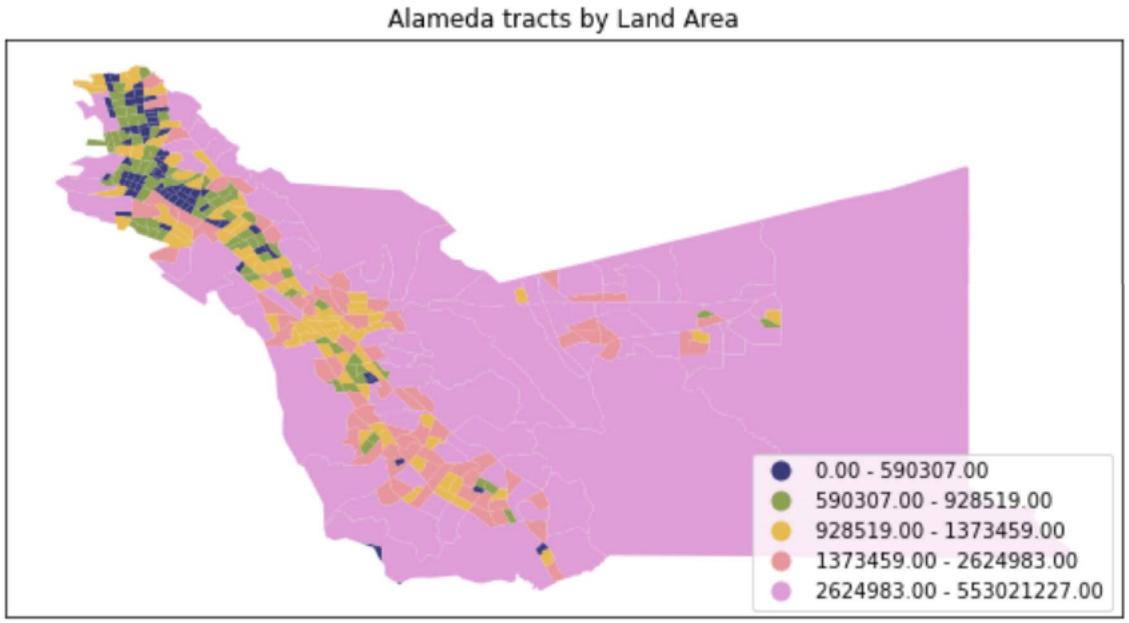
```
8043
```

```
alameda = usa[usa.COUNTYFP=='001']
```

```
ax = alameda.plot(color = 'green', figsize=(10,10),linewidth=2)
ax.set(xticks=[], yticks=[])
plt.savefig("ALAMEDA_tracts.png", bbox_inches='tight')
```



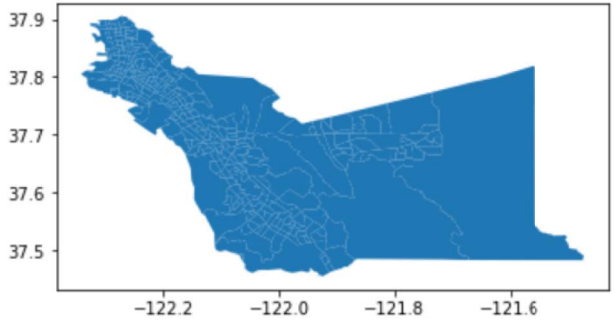
```
ax = alameda.plot(figsize=(10,10), column='ALAND', cmap="tab20b", scheme='quantiles', legend=True)
ax.set(xticks=[], yticks=[]) #removes axes
ax.set_title("Alameda tracts by Land Area", fontsize='large')
#add the legend and specify its location
leg = ax.get_legend()
leg.set_bbox_to_anchor((1.0,0.3))
plt.savefig("Alameda_tracts.png", bbox_inches='tight')
```



```
alameda_tract_geo = alameda[alameda.COUNTYFP=='001']
#alameda_tract_geo = alameda.set_index("GEOID")['geometry'].to_crs(epsg=3310)
```

```
alameda_tract_geo.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1269fe750>
```




```
alameda_tract_geo.head()
```

	STATEFP	COUNTYFP	TRACTCE	AFFGEOID	GEOID	NAME	LSAD	ALAND	AWATER	geometry
0	06	001	400600	1400000US06001400600	06001400600	4006	CT	297856	0	POLYGON ((-122.26807 37.84414, -122.26514 37.8...
1	06	001	400900	1400000US06001400900	06001400900	4009	CT	420877	0	POLYGON ((-122.28558 37.83978, -122.28319 37.8...
2	06	001	401400	1400000US06001401400	06001401400	4014	CT	758241	0	POLYGON ((-122.27861 37.82688, -122.26856 37.8...
3	06	001	403000	1400000US06001403000	06001403000	4030	CT	352394	0	POLYGON ((-122.27476 37.79883, -122.27127 37.8...
4	06	001	405902	1400000US06001405902	06001405902	4059.02	CT	487280	0	POLYGON ((-122.24717 37.78991, -122.24351 37.7...

```
#current coordinate dataframe
print(alameda_tract_geo.crs)
```

```
epsg:4269
```

```
alameda_tract_geo['geometry'].head()
```

```
0    POLYGON ((-122.26807 37.84414, -122.26514 37.8...
1    POLYGON ((-122.28558 37.83978, -122.28319 37.8...
2    POLYGON ((-122.27861 37.82688, -122.26856 37.8...
3    POLYGON ((-122.27476 37.79883, -122.27127 37.8...
4    POLYGON ((-122.24717 37.78991, -122.24351 37.7...
Name: geometry, dtype: geometry
```

Create an empty column alameda_tract_geo['income'] = None

```
['income'] = None
```

Get a census API KEY here: https://api.census.gov/data/key_signup.html

Tutorial in Geopandas dataframes here https://github.com/Automating-GIS-processes/Lesson-2-Geo-DataFrames/blob/master/Lesson/pandas-geopandas.md

Here we work with the Census Tracts of Alameda County

```
from census import Census
from us import states
import csv
```

```
MY_API_KEY='a13903b6e0409abe8cdebe647bb39a8bcf10bb28'
c = Census(MY_API_KEY)
c.acs5.get(('NAME', 'B19013_001E'),
           {'for': 'state:{}'.format(states.CA.fips)})
```

```
[{'NAME': 'California', 'B19013_001E': 67169.0, 'state': '06'}]
```

```
request = c.acs5.state_county_tract('B19013_001E', '06', '001', Census.ALL)
```

```
data = json.loads(str(request).replace("'",'"'))
f = csv.writer(open("income.csv", "w"))
```

```
alameda_tract_geo['income'] = None
```

[illegible]

```
for index, row in alameda_tract_geo.iterrows():
    # Update the value in 'area' column with area information at index
    poly_area = row['geometry'].area
    # Print information for the user
    temp2=row['GEOID']
    #print("Polygon area at index {0} is: {1:.6f}".format(index, poly_area),'',temp2)
    for line in request:
        temp=line["state"]+line["county"]+line["tract"];
        if(temp==temp2):
            print(temp,' ',temp2,' ',line["B19013_001E"]);
            if line["B19013_001E"] > 0:
                alameda_tract_geo.loc[index, 'income']=line["B19013_001E"]
            else:
                alameda_tract_geo.loc[index, 'income']=0.0
```

```
06001400600   06001400600   86458.0
06001400900   06001400900   57011.0
06001401400   06001401400   30479.0
06001403000   06001403000   22711.0
06001405902   06001405902   36000.0
06001407102   06001407102   52383.0
06001409500   06001409500   33963.0
06001421400   06001421400   146774.0
06001422900   06001422900   22399.0
06001424001   06001424001   71392.0
06001427900   06001427900   93071.0
06001430800   06001430800   81133.0
06001432600   06001432600   56404.0
06001433900   06001433900   42363.0
06001436100   06001436100   78870.0
06001437300   06001437300   73625.0
06001440304   06001440304   85417.0
06001440335   06001440335   137265.0
06001441602   06001441602   103588.0
06001442100   06001442100   175417.0
06001443322   06001443322   150132.0
```

[illegible]

The file Tract_flows.csv has the flows between FIPS in the US

It was extracted from the CTPP community survey https://www.fhwa.dot.gov/planning/census_issues/ctpp/data_products/2006-2010_tract_flows/index.cfm

This link: https://davidabailey.com/articles/Creating-a-Four-step-Transportation-Model-in-Python makes a flow model in Python using population and employment data. Note the model tries to resemble the data of the link above. It has a GitHub repositoru with the code

I am sharing the file here FourStepModel.ipynb here. I modified it to be able to run with Networkx 2.0 The time consuming part is to have all the libraries installed. My lesson: Do not run the code without having able to install and import every single required package.

[illegible]

```
gdf = geopandas.GeoDataFrame
```

Query example from the US Census API: https://www.census.gov/data/developers/guidance/api-user-guide/query-examples.html

```
url ='https://api.census.gov/data/2015/acs/acs5?get=NAME,B08015_001E&for=county&in=state:06&key=a13903b6e0409abe8cdebe6
r = requests.get(url)
```

```
Production = pandas.DataFrame(r.json()[1:], columns = r.json()[0], dtype='int')
nameSplit = lambda x: x.split(',')[0]
Production['NAME'] = Production['NAME'].apply(nameSplit)
zones = pandas.merge(zones, Production)
zones['Production'] = zones['B08015_001E']
```

```
zones.head()
```

	geometry	NAME	centroid	B08015_001E	state	county	Production
0	POLYGON ((-120.07333 38.70109, -120.07332 38.7...	Alpine County	(38.59725063317207, -119.82065303034453)	270	06	003	270
1	POLYGON ((-121.11866 38.71707, -121.11176 38.7...	El Dorado County	(38.778737042440646, -120.52465107178368)	62135	06	017	62135
2	POLYGON ((-121.59768 39.12779, -121.59773 39.1...	Yuba County	(39.26900765367862, -121.3512627386866)	21565	06	115	21565
3	POLYGON ((-122.16494 38.64246, -122.16502 38.6...	Yolo County	(38.68666040139593, -121.90161702501356)	65060	06	113	65060
4	POLYGON ((-121.48300 36.76505, -121.48295 36.7...	San Benito County	(36.60570589668711, -121.07499552455516)	21420	06	069	21420

```
prod=pandas.to_numeric(pandas.Series(zones['Production']), errors ='ignore')
```

```
prod.sum()
```

```
13191930
```

```
def getEmployment(state,county):
    prefix = 'EN'
    seasonal_adjustment = 'U'
    #area = format(state, "02d") + format(county, "03d")
    area = str(state)+str(county);
    #print(area)
    data_type = '1'
    size = '0'
    ownership = '0'
    industry = '10'
    seriesid = prefix + seasonal_adjustment + area + data_type + size + ownership + industry
    headers = {'Content-type': 'application/json'}
    data = json.dumps({"seriesid": [seriesid],"startyear":"2015", "endyear":"2015", "registrationKey":"e819346a50f44b57
    p = requests.post('https://api.bls.gov/publicAPI/v2/timeseries/data/',data=data, headers=headers)
    employment = p.json()['Results']['series'][0]['data'][0]['value']
    print(area,'',employment)
    return(employment)
```

Read me Bureau of labor statistics: https://www.bls.gov/help/hlpforma.html

LBS API example: https://www.bls.gov/developers/api_python.htm#python1

Get your API: https://data.bls.gov/registrationEngine/

```
int(getEmployment('06','003'))
```

```
06003  531

531
```

```
#employment = lambda row: int(getEmployment('06','003'))
employment = lambda row: int(getEmployment(row['state'], row['county']))
```

This part takes 1 min for each of the 58 CA counties it prints the area code and #employments as it runs

```
zones['Attraction'] = zones.transpose().apply(employment)
```

```
06003
06017
06115
06113
06069
06019
06015
06023
06029
06077
```

```
attract=pandas.to_numeric(pandas.Series(zones['Attraction']), errors ='ignore')
```

```
attract.sum()
```

```
16222937
```

```
zones['Production'] = prod*attract.sum()/prod.sum()
zones.index = zones.NAME
zones.sort_index(inplace=True)
```

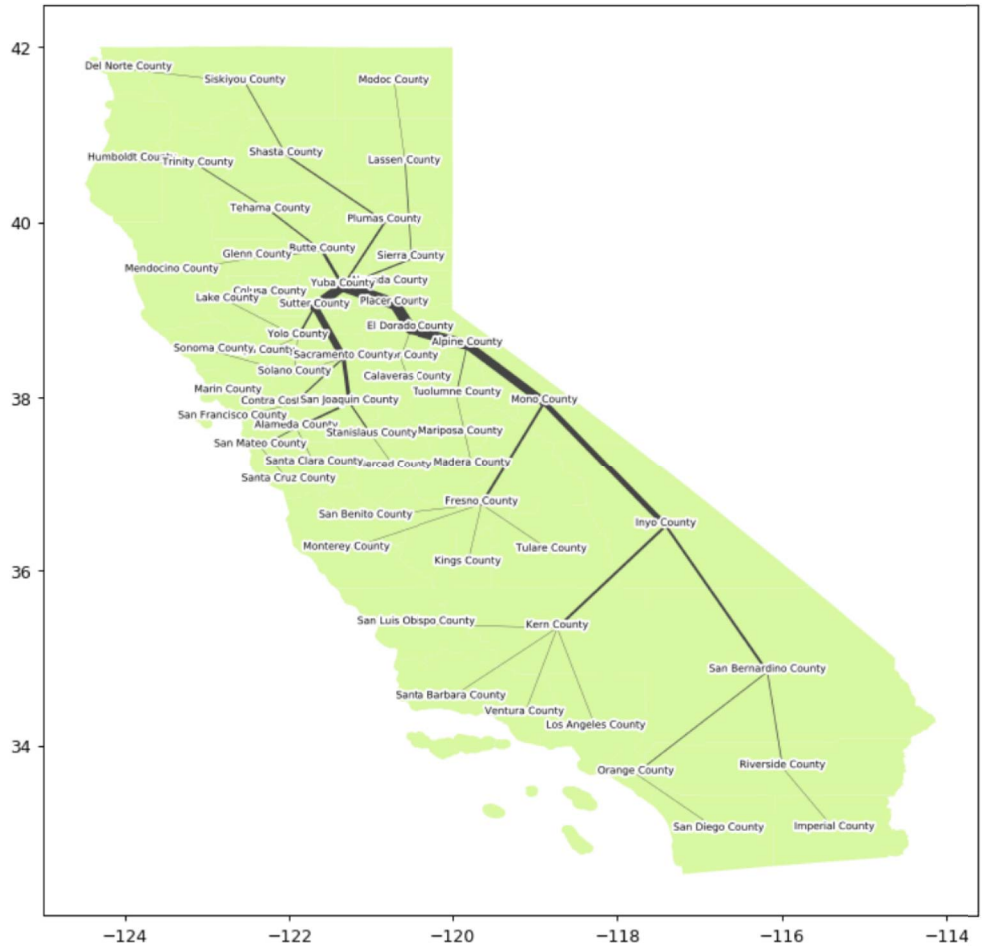
```
attract.sum()/prod.sum()
```

```
1.2297622106848656
```

```
pandas.set_option('display.float_format', lambda x: '%.0f' % x)
zones[['Production', 'Attraction']].head()
```

NAME	Production	Attraction
Alameda County	601083	741843
Alpine County	332	531
Amador County	12267	11500
Butte County	84977	79361
Calaveras County	16141	8888

In [57]:
```
visualize(G, zones.transpose())
```



For playing with parameters

In [58]:
```
# Trip Distribution
beta = 0.001
costMatrix = costMatrixGenerator(zones.transpose(), costFunction, beta)
trips = tripDistribution(zones, costMatrix)
# Mode Choice
modes = {'walking': .05, 'cycling': .05, 'driving': .05}
probabilityMatrix = probabilityMatrixGenerator(zones.transpose(), modeChoiceFunction, modes)
drivingTrips = trips * probabilityMatrix['driving']
# Route Assignment
G = routeAssignment(zones.transpose(), drivingTrips)
visualize(G, zones.transpose())
```

```
ipfn converged: convergence_rate below threshold
```

Registation Key

To make it work, one needs a API key here:

data = json.dumps({"seriesid": [seriesid],"startyear":"2015", "endyear":"2015", "registrationKey":"83df8f76b8dd4dfd856209d5f4d7923d"})

which everyone registers here: https://data.bls.gov/registrationEngine/ to receive by email.