Capstone Project-The battle of neighborhoods

Table of contents

Introduction: Business Problem

Problem Description

In this project we will try to find an optimal location for a restaurant. Specifically, this report will be targeted to stakeholders interested in opening an **Chinese Restaurant** in **New York**, The USA.

Since there are lots of cafes in New York we will try to detect **locations that are not already crowded with restaurants**. We are also particularly interested in **areas with no Chinese restaurants in vicinity**. We would also prefer locations **as close to city center as possible**, assuming that first two conditions are met.

We will use our data science powers to generate a few most promising neighborhoods based on this criteria. Advantages of each area will then be clearly expressed so that best possible final location can be chosen by stakeholders.

Background Discusssion

New York, which is the financial center of the United States, is crowded with people. In NY, you can find people from various countries and different cultural backgrounds.

As a Chinese/Asian food lover, I decided to do a primary research to discover whether there are some places in the center of NewYork to operate a Chinese restaurant. I hope this report could help the stakeholders to find a business solution and introduce this kind of delicious food to the lovely people in NY.

Data

Based on definition of our problem, factors that will influence our decission are:

- number of existing restaurants in the neighborhood (any type of restaurant)
- number of and distance to Chinese restaurants in the neighborhood, if any
- distance of neighborhood from city center

We decided to use regularly spaced grid of locations, centered around city center, to define our neighborhoods.

Following data sources will be needed to extract/generate the required information:

- centers of candidate areas will be generated algorithmically and approximate addresses of centers of those areas will be obtained using geopy
- number of restaurants and their type and location in every neighborhood will be obtained using **Foursquare API**
- coordinate of NY center will be obtained using geopy and geolocator of well known New York location (United Nations Headquarters)

Neighborhood Candidates

Let's create latitude & longitude coordinates for centroids of our candidate neighborhoods. We will create a grid of cells covering our area of interest which is aprox. 16x16 killometers centered around New York city center.

Let's first find the latitude & longitude of Berlin city center, using specific, well known address and Google Maps geocoding API.

In [6]:

```
#!pip install geocoder
import geocoder
```

In [7]:

```
#!pip install geopy # uncomment this line if you haven't completed the Foursquare API lab
from geopy.geocoders import Nominatim
```

In [8]:

```
address = 'United nations headquarters, New York City, NY'

geolocator = Nominatim(user_agent="ny_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of New York City are {}, {}.'.format(latitude, longitude))
```

```
The geograpical coordinate of New York City are 40.7496292, -73.967388998324597.
```

Now let's create a grid of area candidates, equaly spaced, centered around city center and within ~8km from City Hall Park. Our neighborhoods will be defined as circular areas with a radius of 400 meters, so our neighborhood centers will be 400 meters apart.

To accurately calculate distances we need to create our grid of locations in Cartesian 2D coordinate system which allows us to calculate distances in meters (not in latitude/longitude degrees). Then we'll project those coordinates back to latitude/longitude degrees to be shown on Folium map. So let's create functions to convert between WGS84 spherical coordinate system (latitude/longitude degrees) and UTM Cartesian coordinate system (X/Y coordinates in meters).

In [9]:

```
!pip install shapely
import shapely.geometry

!pip install pyproj
import pyproj

import math

def lonlat_to_xy(lon, lat):
    proj_latlon = pyproj.Proj(proj='latlong',datum='WGS84')
    proj_xy = pyproj.Proj(proj="utm", zone=33, datum='WGS84')
    xy = pyproj.transform(proj_latlon, proj_xy, lon, lat)
    return xy[0], xy[1]

def xy_to_lonlat(x, y):
    proj_latlon = pyproj.Proj(proj='latlong',datum='WGS84')
    proj_xy = pyproj.Proj(proj="utm", zone=33, datum='WGS84')
    lonlat = pyproj.transform(proj_xy, proj_latlon, x, y)
    return lonlat[0], lonlat[1]

def calc_xy_distance(x1, y1, x2, y2):
    dx = x2 - x1
    dy = y2 - y1
    return math.sqrt(dx*dx + dy*dy)
```

```
Collecting shapely
  Downloading https://files.pythonhosted.org/packages/20/fa/c96d3461fda99ed8e82ff0
b219ac2c8384694b4e640a611a1a8390ecd415/Shapely-1.7.0-cp36-cp36m-manylinux1_x86_64.
whl (1.8MB)
     |████████████████████████████████| 1.8MB 7.4M
B/s eta 0:00:01
Installing collected packages: shapely
Successfully installed shapely-1.7.0
Collecting pyproj
  Downloading https://files.pythonhosted.org/packages/e5/c3/071e080230ac4b6c64f1a2
e2f9161c9737a2bc7b683d2c90b024825000c0/pyproj-2.6.1.post1-cp36-cp36m-manylinux2010
_x86_64.whl (10.9MB)
     |████████████████████████████████| 10.9MB 14.
6MB/s eta 0:00:01
Installing collected packages: pyproj
Successfully installed pyproj-2.6.1.post1
```

In [12]:

```
import warnings
warnings.filterwarnings("ignore", category=Warning)    # To avoid warnings that influence beaut
y
```

In [13]:

```
print('Coordinate transformation check')
print('-------------------------------')
print('New York center longitude={}, latitude={}'.format(longitude, latitude))
x, y = lonlat_to_xy(longitude, latitude)
print('New York center UTM X={}, Y={}'.format(x, y))
lo, la = xy_to_lonlat(x, y)
print('New York center longitude={}, latitude={}'.format(lo, la))
```

```
Coordinate transformation check
-------------------------------
New York center longitude=-73.96738998324597, latitude=40.7496292
New York center UTM X=-5815898.467310907, Y=9864790.714015638
New York center longitude=-73.96738998324561, latitude=40.74962919999888
```

Let's create a **hexagonal grid of cells**: we offset every other row, and adjust vertical row spacing so that **every cell center is equally distant from all it's neighbors**.

In [14]:

```
ny_center_x, ny_center_y = lonlat_to_xy(longitude, latitude) # City center in Cartesian coordina
tes

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = ny_center_x - 8000
x_step = 800
y_min = ny_center_y - 8000 - (int(21/k)*k*800 - 16000)/2
y_step = 800 * k

latitudes = []
longitudes = []
distances_from_center = []
xs = []
ys = []
for i in range(0, int(21/k)):
    y = y_min + i * y_step
    x_offset = 400 if i%2==0 else 0
    for j in range(0, 21):
        x = x_min + j * x_step + x_offset
        distance_from_center = calc_xy_distance(ny_center_x, ny_center_y, x, y)
        if (distance_from_center <= 8001):
            lon, lat = xy_to_lonlat(x, y)
            latitudes.append(lat)
            longitudes.append(lon)
            distances_from_center.append(distance_from_center)
            xs.append(x)
            ys.append(y)

print(len(latitudes), 'candidate neighborhood centers generated.')
```

```
364 candidate neighborhood centers generated.
```

Let's visualize the data we have so far: city center location and candidate neighborhood centers:

In [15]:

```
!pip install folium

import folium
```

```
Collecting folium
  Downloading https://files.pythonhosted.org/packages/a4/f0/44e69d50519880287cc41e
7c8a6acc58daa9a9acf5f6afc52bcc70f69a6d/folium-0.11.0-py2.py3-none-any.whl (93kB)
     |████████████████████████████████| 102kB 7.1M
B/s ta 0:00:011
Requirement already satisfied: numpy in /opt/conda/envs/Python36/lib/python3.6/sit
e-packages (from folium) (1.15.4)
Requirement already satisfied: requests in /opt/conda/envs/Python36/lib/python3.6/
site-packages (from folium) (2.21.0)
Requirement already satisfied: jinja2>=2.9 in /opt/conda/envs/Python36/lib/python
3.6/site-packages (from folium) (2.10)
Collecting branca>=0.3.0 (from folium)
  Downloading https://files.pythonhosted.org/packages/13/fb/9eacc24ba3216510c6b59a
4ea1cd53d87f25ba76237d7f4393abeaf4c94e/branca-0.4.1-py3-none-any.whl
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /opt/conda/envs/Python36/l
ib/python3.6/site-packages (from requests->folium) (3.0.4)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /opt/conda/envs/Python36/l
ib/python3.6/site-packages (from requests->folium) (1.24.1)
Requirement already satisfied: idna<2.9,>=2.5 in /opt/conda/envs/Python36/lib/pyth
on3.6/site-packages (from requests->folium) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/envs/Python36/lib/
python3.6/site-packages (from requests->folium) (2020.4.5.1)
Requirement already satisfied: MarkupSafe>=0.23 in /opt/conda/envs/Python36/lib/py
thon3.6/site-packages (from jinja2>=2.9->folium) (1.1.0)
Installing collected packages: branca, folium
Successfully installed branca-0.4.1 folium-0.11.0
```

In [16]:

```
ny_center = [latitude, longitude]

map_ny = folium.Map(location=ny_center, zoom_start=13)
folium.Marker(ny_center, popup='United Nations Headquarters').add_to(map_ny)
for lat, lon in zip(latitudes, longitudes):
    #folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_opacity=1).add_to(map_berlin)
    folium.Circle([lat, lon], radius=300, color='blue', fill=False).add_to(map_ny)
    #folium.Marker([lat, lon]).add_to(map_berlin)
map_ny
```

Out[16]:

Make this Notebook Trusted to load map: File -> Trust Notebook

OK, we now have the coordinates of centers of neighborhoods/areas to be evaluated, equally spaced (distance from every point to it's neighbors is exactly the same) and within ~8km from United nations headquarters.

Let's now use GEopy API to get approximate addresses of those locations.

In [17]:

```
g = geolocator.reverse([latitude, longitude])
g.address
```

Out[17]:

```
'United Nations Headquarters, 405, FDR Drive, United Nations, Manhattan, Tudor City, Manhattan Community Board 6, New York, New York County, New York, 10017, United States of America'
```

In [33]:

```
print('Obtaining location addresses: ', end='')
addresses = []
for lat, lon in zip(latitudes, longitudes):
    address = geolocator.geocode([lat, lon]).address
    if address is None:
        address = 'NO ADDRESS'
    address = address.replace(', United States of America', '')
    address = address.replace(', New York County, New York', '')
    address = address.replace(', Kings County, New York', '')
    address = address.replace(', Queens County, New York', '')# We don't need country part of ad
dress
    addresses.append(address)
    print(' .', end='')
print(' done.')
```

```
Obtaining location addresses:  . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . done.
```

In [34]:

```
addresses[150:170]
```

Out[34]:

```
['56, East 93rd Street, Carnegie Hill, Manhattan, Manhattan Community Board 8, 101
28',
 '1165, 5th Avenue, Manhattan, Manhattan Community Board 11, 10029',
 'East Drive, Manhattan, New York, 10026',
 '161, Clymer Street, Williamsburg, New York, Brooklyn, 11211',
 '370, Bedford Avenue, Williamsburg, New York, Brooklyn, 11249',
 '100, Metropolitan Avenue, Williamsburg, New York, Brooklyn, 11249',
 'The Edge North Tower, North 7th Street, Williamsburg, New York, Brooklyn, 1121
1',
 'North 12th Street, Greenpoint, New York, Brooklyn, 11211',
 'India Street/Greenpoint, India Street, Greenpoint, New York, Brooklyn, 11222',
 '52, Freeman Street, Greenpoint, New York, Brooklyn, 11222',
 'Center Boulevard, Long Island City, Queens, New York, 11109',
 'Queens-Midtown Tunnel, Tudor City, New York, 10017-6927',
 'Manhattan, Manhattan Community Board 8',
 'FDR Drive, Turtle Bay, Manhattan, New York, 10155',
 '409, East 58th Street, Midtown East, Manhattan, Manhattan Community Board 6, 100
22',
 '304, East 64th Street, Lenox Hill, Manhattan, Manhattan Community Board 8, 1006
5',
 '201, East 70th Street, Lenox Hill, Manhattan, Manhattan Community Board 8, 1002
1',
 '122, East 76th Street, Upper East Side, Manhattan, Manhattan Community Board 8,
10021',
 '1082, Madison Avenue, Upper East Side, Manhattan, Manhattan Community Board 8, 1
0028',
 'Reservoir Running Track, Manhattan, New York']
```

Looking good. Let's now place all this into a Pandas dataframe.

In [35]:

```
import pandas as pd

df_locations = pd.DataFrame({'Address': addresses,
                             'Latitude': latitudes,
                             'Longitude': longitudes,
                             'X': xs,
                             'Y': ys,
                             'Distance from center(m)': distances_from_center})

df_locations.head(10)
```

Out[35]:

	Address	Latitude	Longitude	X	Y	Distance from center(m)
0	52-35, 58th Street, Maspeth, Queens, New York,...	40.734215	-73.908861	-5.818298e+06	9.857170e+06	7989.993742
1	47-63, 58th Street, Woodside, Queens, New York...	40.738930	-73.908681	-5.817498e+06	9.857170e+06	7787.168934
2	41-47, 56th Street, Woodside, Queens, New York...	40.743645	-73.908501	-5.816698e+06	9.857170e+06	7662.897624
3	54-18, 39th Avenue, Woodside, Queens, New York...	40.748360	-73.908321	-5.815898e+06	9.857170e+06	7621.023553
4	McDonald's, Northern Boulevard, Woodside, Quee...	40.753076	-73.908141	-5.815098e+06	9.857170e+06	7662.897624
5	Public School 151, 31st Avenue, Woodside House...	40.757792	-73.907961	-5.814298e+06	9.857170e+06	7787.168934
6	47-12, 28th Avenue, Steinway, Queens, New York...	40.762508	-73.907781	-5.813498e+06	9.857170e+06	7989.993742
7	55-64, 56th Street, Maspeth, Queens, New York,...	40.727263	-73.914497	-5.819498e+06	9.857863e+06	7807.688518
8	Long Island Expressway, Maspeth, Queens, New Y...	40.731976	-73.914318	-5.818698e+06	9.857863e+06	7472.616677
9	Brooklyn Queens Expressway, Woodside, Queens, ...	40.736691	-73.914139	-5.817898e+06	9.857863e+06	7211.102551

...and let's now save/persist this data into local file.

In [36]:

```
df_locations.to_pickle('./locations.pkl')
```

Foursquare

Now that we have our location candidates, let's use Foursquare API to get info on restaurants in each neighborhood.

We're interested in venues in 'food' category, but only those that are proper restaurants - coffe shops, pizza places, bakeries etc. are not direct competitors so we don't care about those. So we will include in out list only venues that have 'restaurant' in category name, and we'll make sure to detect and include all the subcategories of specific 'Chinese restaurant' category, as we need info on Chinese restaurants in the neighborhood.

In [37]:

```
# libraries for displaying images
from IPython.display import Image
from IPython.core.display import HTML
```

Foursquare credentials are defined in hidden cell bellow.

In [38]:

```
# @hidden_cell
import requests # library to handle requests
from pandas.io.json import json_normalize # tranforming json file into a pandas dataframe lib
rary

foursquare_client_id = '5KDV22J1PJQHC3HQFTNJWOCGYLZCWSMUFODSY3BRTMPYQOI1' # your Foursquare ID
foursquare_client_secret = 'SYUCDCORPTON5KF4RMECG3CNYFCCOWY4SKQUUTNFXUNL3NIV' # your Foursquare
 Secret

print('Your credentails:')
print('CLIENT_ID: ' + foursquare_client_id)
print('CLIENT_SECRET:' + foursquare_client_secret)
```

```
Your credentails:
CLIENT_ID: 5KDV22J1PJQHC3HQFTNJWOCGYLZCWSMUFODSY3BRTMPYQOI1
CLIENT_SECRET:SYUCDCORPTON5KF4RMECG3CNYFCCOWY4SKQUUTNFXUNL3NIV
```

In [39]:

```
# Category IDs corresponding to Chinese restaurants were taken from Foursquare web site (http
s://developer.foursquare.com/docs/resources/categories):

food_category = '4d4b7105d754a06374d81259' # 'Root' category for all food-related venues

chinese_restaurant_categories = ['4bf58dd8d48988d145941735','52af3a5e3cf9994f4e043bea','52af3a72
3cf9994f4e043bec',
                                 '52af3a7c3cf9994f4e043bed','58daa1558bbb0b01f18ec1d3','52af3a67
3cf9994f4e043beb',
                                 '52af3a903cf9994f4e043bee','4bf58dd8d48988d1f5931735','52af3a9f
3cf9994f4e043bef',
                                 '52af3aaa3cf9994f4e043bf0','52af3ab53cf9994f4e043bf1','52af3abe
3cf9994f4e043bf2',
                                 '52af3ac83cf9994f4e043bf3','52af3ad23cf9994f4e043bf4','52af3add
3cf9994f4e043bf5',
                                 '52af3af23cf9994f4e043bf7','52af3ae63cf9994f4e043bf6','52af3afc
3cf9994f4e043bf8',
                                 '52af3b053cf9994f4e043bf9','52af3b213cf9994f4e043bfa','52af3b29
3cf9994f4e043bfb',
                                 '52af3b343cf9994f4e043bfc','52af3b3b3cf9994f4e043bfd','52af3b46
3cf9994f4e043bfe',
                                 '52af3b633cf9994f4e043c01','52af3b513cf9994f4e043bff','52af3b59
3cf9994f4e043c00',
                                 '52af3b6e3cf9994f4e043c02','52af3b773cf9994f4e043c03','52af3b81
3cf9994f4e043c04',
                                 '52af3b893cf9994f4e043c05','52af3b913cf9994f4e043c06','52af3b9a
3cf9994f4e043c07',
                                 '52af3ba23cf9994f4e043c08']

def is_restaurant(categories, specific_filter=None):
    restaurant_words = ['restaurant', 'diner', 'taverna', 'steakhouse']
    restaurant = False
    specific = False
    for c in categories:
        category_name = c[0].lower()
        category_id = c[1]
        for r in restaurant_words:
            if r in category_name:
                restaurant = True
        if 'fast food' in category_name:
            restaurant = False
        if not(specific_filter is None) and (category_id in specific_filter):
            specific = True
            restaurant = True
    return restaurant, specific

def get_categories(categories):
    return [(cat['name'], cat['id']) for cat in categories]

def format_address(location):
    address = ', '.join(location['formattedAddress'])
    address = address.replace(', United States of America', '')
    address = address.replace(', New York County, New York', '')
    address = address.replace(', Kings County, New York', '')
    address = address.replace(', Queens County, New York', '')
    return address

def get_venues_near_location(lat, lon, category, client_id, client_secret, radius=500, limit=100
):
```

```
    version = '20180604'
    url = 'https://api.foursquare.com/v2/venues/explore?client_id={}&client_secret={}&v={}&ll=
{},{}&categoryId={}&radius={}&limit={}'.format(
        client_id, client_secret, version, lat, lon, category, radius, limit)
    try:
        results = requests.get(url).json()['response']['groups'][0]['items']
        venues = [(item['venue']['id'],
                   item['venue']['name'],
                   get_categories(item['venue']['categories']),
                   (item['venue']['location']['lat'], item['venue']['location']['lng']),
                   format_address(item['venue']['location']),
                   item['venue']['location']['distance']) for item in results]
    except:
        venues = []
    return venues
```

In [40]:

```
# Let's now go over our neighborhood locations and get nearby restaurants; we'll also maintain a dictionary of all found restaurants and all found italian restaurants

import pickle

def get_restaurants(lats, lons):
    restaurants = {}
    chinese_restaurants = {}
    location_restaurants = []

    print('Obtaining venues around candidate locations:', end='')
    for lat, lon in zip(lats, lons):
        # Using radius=450 to meke sure we have overlaps/full coverage so we don't miss any restaurant
        # we're using dictionaries to remove any duplicates resulting from area overlaps
        venues = get_venues_near_location(lat, lon, food_category, foursquare_client_id, foursquare_client_secret, radius=4500, limit=100)

        area_restaurants = []
        for venue in venues:
            venue_id = venue[0]
            venue_name = venue[1]
            venue_categories = venue[2]
            venue_latlon = venue[3]
            venue_address = venue[4]
            venue_distance = venue[5]
            is_res, is_chinese = is_restaurant(venue_categories, specific_filter=chinese_restaurant_categories)
            if is_res:
                x, y = lonlat_to_xy(venue_latlon[1], venue_latlon[0])
                restaurant = (venue_id, venue_name, venue_latlon[0], venue_latlon[1], venue_address, venue_distance, is_chinese, x, y)
                if venue_distance<=400:
                    area_restaurants.append(restaurant)
                restaurants[venue_id] = restaurant
                if is_chinese:
                    chinese_restaurants[venue_id] = restaurant
        location_restaurants.append(area_restaurants)
        print(' .', end='')
    print(' done.')
    return restaurants, chinese_restaurants, location_restaurants

# Try to load from local file system in case we did this before
restaurants = {}
chinese_restaurants = {}
location_restaurants = []
loaded = False

try:
    with open('restaurants_450.pkl', 'rb') as f:
        restaurants = pickle.load(f)
    with open('chinese_restaurants_450.pkl', 'rb') as f:
        chinese_restaurants = pickle.load(f)
    with open('location_restaurants_450.pkl', 'rb') as f:
        location_restaurants = pickle.load(f)
    print('Restaurant data loaded.')
    loaded = True
except:
    pass
```

```
# If load failed use the Foursquare API to get the data
if not loaded:
    restaurants, chinese_restaurants, location_restaurants = get_restaurants(latitudes, longitud
es)

    # Let's persists this in local file system
    with open('restaurants_450.pkl', 'wb') as f:
        pickle.dump(restaurants, f)
    with open('chinese_restaurants_450.pkl', 'wb') as f:
        pickle.dump(chinese_restaurants, f)
    with open('location_restaurants_450.pkl', 'wb') as f:
        pickle.dump(location_restaurants, f)
```

```
Obtaining venues around candidate locations: . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . done.
```

In [41]:

```
import numpy as np

print('Total number of restaurants:', len(restaurants))
print('Total number of Chinese restaurants:', len(chinese_restaurants))
print('Percentage of Chinese restaurants: {:.2f}%'.format(len(chinese_restaurants) / len(restau
rants) * 100))
print('Average number of restaurants in neighborhood:', np.array([len(r) for r in location_rest
aurants]).mean())
```

```
Total number of restaurants: 562
Total number of Chinese restaurants: 14
Percentage of Chinese restaurants: 2.49%
Average number of restaurants in neighborhood: 1.2197802197802199
```

In [42]:

```
print('List of all restaurants')
print('-----------------------')
for r in list(restaurants.values())[:10]:
    print(r)
print('...')
print('Total:', len(restaurants))
```

```
List of all restaurants
-----------------------
('4aecab47f964a52026ca21e3', "Tito Rad's Grill & Restaurant", 40.74247745309511, -73.9155837893486, '49-10 Queens Blvd (btwn 49th & 50th St), Woodside, NY 11377, United States', 3034, False, -5816922.752591292, 9858077.193497935)
('56217bcc498efe46cdf1b333', 'Cemitas el Tigre', 40.739321413562735, -73.91982682320209, '45-14 48th Ave (46th St.), Woodside, NY 11377, United States', 2259, False, -5817473.574118812, 9858608.922301093)
('49dbfd83f964a520395f1fe3', 'La Flor', 40.74443789878033, -73.91156538057702, '53-02 Roosevelt Ave (at 53rd St), Woodside, NY 11377, United States', 2353, False, -5816575.426040203, 9857568.67707435)
('4da0dca3b3e7236a4768fd78', 'I Love Paraguay', 40.741087267057395, -73.92149020353071, '43-16 Greenpoint Ave, Sunnyside, NY 11104, United States', 3045, False, -5817180.352263195, 9858832.018964184)
('4a6cf6eef964a52034d21fe3', 'De Mole', 40.739319819813794, -73.92041391089583, '4502 48th Ave (at 45th St), Woodside, NY 11377, United States', 2224, False, -5817476.019836334, 9858684.623701487)
('567a40b1498ef19172a0b3cf', 'The Alcove', 40.74592873256032, -73.91526311066848, '41-11 49th St (Skillman), Sunnyside, NY, United States', 3060, False, -5816336.4804803105, 9858052.730780352)
('4c0d756e2466a593409e7721', 'The Haab Mexican Cafe', 40.73905580430105, -73.91780627164373, '4722 48th St, Woodside, NY 11377, United States', 1921, False, -5817511.109814014, 9858347.061335834)
('3fd66200f964a5204ef11ee3', 'SriPraPhai', 40.7463421117063, -73.89924799709517, '64-13 39th Ave (btwn 64th & 65th St), Woodside, NY 11377, United States', 3072, False, -5816206.376269922, 9855990.02821158)
('555d29f7498eb6112f44ecac', 'SoleLuna', 40.7440962823303, -73.92412726169988, '40-01 Queens Blvd, Sunnyside, NY 11104, United States', 3252, False, -5816679.973289107, 9859186.669762047)
('5004a603e4b03a9a7558faa1', 'Takesushi', 40.74409055369966, -73.92235338862658, '4346 42nd St, Sunnyside, NY 11104, United States', 2707, False, -5816674.3918748405, 9858957.925595906)
...
Total: 562
```

In [43]:

```
print('List of Chinese restaurants')
print('---------------------------')
for r in list(chinese_restaurants.values())[:10]:
    print(r)
print('...')
print('Total:', len(chinese_restaurants))
```

```
List of Chinese restaurants
---------------------------
('5a7f87cc356b49777208f5f4', 'Shanghai Zhen Gong Fu', 40.736321901750145, -73.8771
457262649, '86-16 Queens Blvd, Elmhurst, NY 11373, United States', 2675, True, -58
17820.783591679, 9853089.838536188)
('54820ad7498e5a6cf6ff64e2', "Xi'an Famous Foods", 40.72425868174162, -73.95108670
563869, '648 Manhattan Ave (at Bedford Ave), Brooklyn, NY 11222, United States', 2
576, True, -5820142.0155828465, 9862568.879268475)
('5685f10238fa7274dab2566c', 'Win Son', 40.70743, -73.943359, '159 Graham Ave (at
Montrose Ave), Brooklyn, NY 11206, United States', 2267, True, -5822968.658407411,
9861491.481283415)
('559d79fa498e43f2849245fa', 'Kings County Imperial', 40.71545608222545, -73.95116
830185611, '20 Skillman Ave (btwn Union Ave & Lorimer St), Brooklyn, NY 11211, Uni
ted States', 2383, True, -5821635.354983379, 9862537.722597387)
('58df00318cfe546addb99246', 'Birds of a Feather', 40.71426224260337, -73.96057226
233967, '191 Grand St (btwn Bedford & Driggs Ave), Brooklyn, NY 11211, United Stat
es', 2055, True, -5821871.60814275, 9863745.889702829)
('5b380f649deb7d00399fdf9d', 'Kings County Imperial', 40.71781670552335, -73.98556
881621373, '168 1/2 Delancey St (btw Clinton & Attorney), New York, NY 10002, Unit
ed States', 2192, True, -5821356.862129496, 9866988.681299336)
('4e3484038877beb5e9a22a0b', 'Café China', 40.7499796904135, -73.98223427700086,
'13 E 37th St (btwn 5th Ave & Madison Ave), New York, NY 10016, United States', 32
32, True, -5815891.246618496, 9866706.033176707)
('5a7e4674da2e00425ee2921d', 'Málà Project', 40.75685008448552, -73.9808552750597
2, '41 W 46th St (btwn 5th & 6th Ave), New York, NY 10036, United States', 3209, T
rue, -5814721.848275769, 9866559.827573612)
('529e3657498efb17e9c800b7', "Xi'an Famous Foods", 40.798353, -73.969378, '2675 Br
oadway (at 102nd St), New York, NY 10025, United States', 2586, True, -5807649.932
593138, 9865273.251514561)
('5647ee82498e8bfc0ddef53d', 'Málà Project', 40.727126, -73.98545, '122 1st Ave (b
twn 7th St & St Marks Pl), New York, NY 10009, United States', 1765, True, -581977
7.479782567, 9867015.986913405)
...
Total: 14
```

In [44]:

```
print('Restaurants around location')
print('---------------------------')
for i in range(100, 110):
    rs = location_restaurants[i][:8]
    names = ', '.join([r[1] for r in rs])
    print('Restaurants around location {}: {}'.format(i+1, names))
```

```
Restaurants around location
---------------------------
Restaurants around location 101:
Restaurants around location 102:
Restaurants around location 103:
Restaurants around location 104:
Restaurants around location 105: LIC Market
Restaurants around location 106:
Restaurants around location 107:
Restaurants around location 108:
Restaurants around location 109:
Restaurants around location 110:
```

Let's now see all the collected restaurants in our area of interest on map, and let's also show Chinese restaurants in different color.

In [45]:

```
map_ny = folium.Map(location=ny_center, zoom_start=13)
folium.Marker(ny_center, popup='United Nations Headquarters').add_to(map_ny)
for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_chinese = res[6]
    color = 'red' if is_chinese else 'blue'
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True, fill_color=color, fill_op
acity=1).add_to(map_ny)

map_ny
```

Out[45]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Looking good. So now we have all the restaurants in area within few kilometers from United Nations Headquarters, and we know which ones are Chinese restaurants! We also know which restaurants exactly are in vicinity of every neighborhood candidate center.

This concludes the data gathering phase - we're now ready to use this data for analysis to produce the report on optimal locations for a new Chinese restaurant!

Methodology

In this project we will direct our efforts on detecting areas of New York that have low restaurant density, particularly those with low number of Italian restaurants. We will limit our analysis to area ~6km around city center.

In first step we have collected the required **data: location and type (category) of every restaurant within 8km from New York center** (United Nations Headquarters). We have also **identified Chinese restaurants** (according to Foursquare categorization).

Second step in our analysis will be calculation and exploration of '**restaurant density**' across different areas of NY - we will use **heatmaps** to identify a few promising areas close to center with low number of restaurants in general (*and* no Chinese restaurants in vicinity) and focus our attention on those areas.

In third and final step we will focus on most promising areas and within those create **clusters of locations that meet some basic requirements** established in discussion with stakeholders: we will take into consideration locations with **no restaurant in radius of 250 meters**, and we want locations **without Chinese restaurants in radius of 1000 meters**. We will present map of all such locations but also create clusters (using **k-means clustering**) of those locations to identify general zones / neighborhoods / addresses which should be a starting point for final 'street level' exploration and search for optimal venue location by stakeholders.

Analysis

Let's perform some basic explanatory data analysis and derive some additional info from our raw data. First let's count the **number of restaurants in every area candidate**:

In [46]:

```
location_restaurants_count = [len(res) for res in location_restaurants]

df_locations['Restaurants in area'] = location_restaurants_count

print('Average number of restaurants in every area with radius=400m:', np.array(location_restaur
ants_count).mean())

df_locations.head(10)
```

```
Average number of restaurants in every area with radius=400m: 1.2197802197802199
```

Out[46]:

	Address	Latitude	Longitude	X	Y	Distance from center(m)	Restaurants in area
0	52-35, 58th Street, Maspeth, Queens, New York,...	40.734215	-73.908861	-5.818298e+06	9.857170e+06	7989.993742	(
1	47-63, 58th Street, Woodside, Queens, New York...	40.738930	-73.908681	-5.817498e+06	9.857170e+06	7787.168934	(
2	41-47, 56th Street, Woodside, Queens, New York...	40.743645	-73.908501	-5.816698e+06	9.857170e+06	7662.897624	1
3	54-18, 39th Avenue, Woodside, Queens, New York...	40.748360	-73.908321	-5.815898e+06	9.857170e+06	7621.023553	(
4	McDonald's, Northern Boulevard, Woodside, Quee...	40.753076	-73.908141	-5.815098e+06	9.857170e+06	7662.897624	(
5	Public School 151, 31st Avenue, Woodside House...	40.757792	-73.907961	-5.814298e+06	9.857170e+06	7787.168934	(
6	47-12, 28th Avenue, Steinway, Queens, New York...	40.762508	-73.907781	-5.813498e+06	9.857170e+06	7989.993742	(
7	55-64, 56th Street, Maspeth, Queens, New York,...	40.727263	-73.914497	-5.819498e+06	9.857863e+06	7807.688518	(
8	Long Island Expressway, Maspeth, Queens, New Y...	40.731976	-73.914318	-5.818698e+06	9.857863e+06	7472.616677	(
9	Brooklyn Queens Expressway, Woodside, Queens, ...	40.736691	-73.914139	-5.817898e+06	9.857863e+06	7211.102551	(

OK, now let's calculate the **distance to nearest Chinese restaurant from every area candidate center** (not only those within 300m - we want distance to closest one, regardless of how distant it is).

In [47]:

```
distances_to_chinese_restaurant = []

for area_x, area_y in zip(xs, ys):
    min_distance = 10000
    for res in chinese_restaurants.values():
        res_x = res[7]
        res_y = res[8]
        d = calc_xy_distance(area_x, area_y, res_x, res_y)
        if d<min_distance:
            min_distance = d
    distances_to_chinese_restaurant.append(min_distance)

df_locations['Distance to Chinese restaurant'] = distances_to_chinese_restaurant
```

In [48]:

```
df_locations.head(10)
```

Out[48]:

	Address	Latitude	Longitude	X	Y	Distance from center(m)	Restaurants in area
0	52-35, 58th Street, Maspeth, Queens, New York,...	40.734215	-73.908861	-5.818298e+06	9.857170e+06	7989.993742	(
1	47-63, 58th Street, Woodside, Queens, New York...	40.738930	-73.908681	-5.817498e+06	9.857170e+06	7787.168934	(
2	41-47, 56th Street, Woodside, Queens, New York...	40.743645	-73.908501	-5.816698e+06	9.857170e+06	7662.897624	1
3	54-18, 39th Avenue, Woodside, Queens, New York...	40.748360	-73.908321	-5.815898e+06	9.857170e+06	7621.023553	(
4	McDonald's, Northern Boulevard, Woodside, Quee...	40.753076	-73.908141	-5.815098e+06	9.857170e+06	7662.897624	(
5	Public School 151, 31st Avenue, Woodside House...	40.757792	-73.907961	-5.814298e+06	9.857170e+06	7787.168934	(
6	47-12, 28th Avenue, Steinway, Queens, New York...	40.762508	-73.907781	-5.813498e+06	9.857170e+06	7989.993742	(
7	55-64, 56th Street, Maspeth, Queens, New York,...	40.727263	-73.914497	-5.819498e+06	9.857863e+06	7807.688518	(
8	Long Island Expressway, Maspeth, Queens, New Y...	40.731976	-73.914318	-5.818698e+06	9.857863e+06	7472.616677	(
9	Brooklyn Queens Expressway, Woodside, Queens, ...	40.736691	-73.914139	-5.817898e+06	9.857863e+06	7211.102551	(

In [49]:

```
print('Average distance to closest Chinese restaurant from each area center:', df_locations['Dis
tance to Chinese restaurant'].mean())
```

```
Average distance to closest Chinese restaurant from each area center: 2679.4775283
434897
```

OK, so **on average Italian restaurant can be found within ~2700m** from every area center candidate.

Let's crete a map showing **heatmap / density of restaurants** and try to extract some meaningfull info from that. Also, let's show **borders of New York boroughs** on our map and a few circles indicating distance of 1km, 2km and 3km from United Nations Headquarters.

In [50]:

```
import requests
bronx_boroughs_url = 'https://raw.githubusercontent.com/utisz/compound-cities/master/new-york-ci
ty/bronx.geo.json'
bronx_boroughs = requests.get(bronx_boroughs_url).json()

brooklyn_boroughs_url = 'https://raw.githubusercontent.com/utisz/compound-cities/master/new-york
-city/brooklyn.geo.json'
brooklyn_boroughs = requests.get(brooklyn_boroughs_url).json()

manhattan_boroughs_url = 'https://raw.githubusercontent.com/utisz/compound-cities/master/new-yor
k-city/manhattan.geo.json'
manhattan_boroughs = requests.get(manhattan_boroughs_url).json()

staten_island_boroughs_url = 'https://raw.githubusercontent.com/utisz/compound-cities/master/new
-york-city/staten-island.geo.json'
staten_island_boroughs = requests.get(staten_island_boroughs_url).json()

queens_boroughs_url = 'https://raw.githubusercontent.com/utisz/compound-cities/master/new-york-c
ity/queens.geo.json'
queens_boroughs = requests.get(queens_boroughs_url).json()

def boroughs_style(feature):
    return { 'color': 'blue', 'fill': False }
```

In [51]:

```
restaurant_latlons = [[res[2], res[3]] for res in restaurants.values()]

chinese_latlons = [[res[2], res[3]] for res in chinese_restaurants.values()]
```

In [67]:

```
from folium import plugins
from folium.plugins import HeatMap

map_ny = folium.Map(location=ny_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_ny)  # cartodbpositron cartodbdark_matter
HeatMap(restaurant_latlons).add_to(map_ny)
folium.Marker(ny_center).add_to(map_ny)
folium.Circle(ny_center, radius=1000, fill=False, color='white').add_to(map_ny)
folium.Circle(ny_center, radius=2000, fill=False, color='white').add_to(map_ny)
folium.Circle(ny_center, radius=3000, fill=False, color='white').add_to(map_ny)

folium.GeoJson(bronx_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(staten_island_boroughs, style_function=boroughs_style, name='geojson').add_to(map
_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[67]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Looks like a few pockets of low restaurant density closest to city center can be found **north, north-east and south from United Nations Headquarters**.

Let's create another heatmap map showing **heatmap/density of Chinese restaurants** only.

In [54]:

```
map_ny = folium.Map(location=ny_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_ny) #cartodbpositron cartodbdark_matter
HeatMap(chinese_latlons).add_to(map_ny)
folium.Marker(ny_center).add_to(map_ny)
folium.Circle(ny_center, radius=1000, fill=False, color='white').add_to(map_ny)
folium.Circle(ny_center, radius=2000, fill=False, color='white').add_to(map_ny)
folium.Circle(ny_center, radius=3000, fill=False, color='white').add_to(map_ny)

folium.GeoJson(bronx_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(staten_island_boroughs, style_function=boroughs_style, name='geojson').add_to(map
_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[54]:

Make this Notebook Trusted to load map: File -> Trust Notebook

This map is not so 'hot' (Chinese restaurants represent a subset of ~2.5% of all restaurants in New York) but it also indicates higher density of existing Chinese restaurants directly north and west from Alexanderplatz, with closest pockets of **low Chinese restaurant density positioned east, south-east and south from city center**.

Based on this we will now focus our analysis on areas *south-west, south, south-east and east from Berlin center* - we will move the center of our area of interest and reduce it's size to have a radius of **1.5km**. This places our location candidates mostly in boroughs **Manhattan** (another potentially interesting borough is **Brooklyn** with large low restaurant density north-east from city center, however this borough is less interesting to stakeholders as it's mostly residental and less popular with tourists).

Manhattan, brooklyn and Queens

Analysis of popular travel guides and web sites often mention Manhattan, brooklyn and Queens as beautifull, interesting, rich with culture, 'hip' and 'cool' New York neighborhoods popular with tourists and loved by Berliners.

Popular with tourists, alternative and bohemian but booming and trendy, relatively close to city center and well connected, those boroughs appear to justify further analysis.

Let's define new, more narrow region of interest, which will include low-restaurant-count parts of Manhattan, brooklyn and Queens closest to United Nations Headquarters.

As we all know, manhattan middle city is the most busy part of New York and this research will mainly discover the situation in this part.

In [82]:

```
roi_x_min = ny_center_x -1000
roi_y_max = ny_center_y + 4000
roi_width = 5000
roi_height = 5000
roi_center_x = roi_x_min + 1500
roi_center_y = roi_y_max - 1500
roi_center_lon, roi_center_lat = xy_to_lonlat(roi_center_x, roi_center_y)
roi_center = [roi_center_lat, roi_center_lon]

map_ny = folium.Map(location=roi_center, zoom_start=14)
HeatMap(restaurant_latlons).add_to(map_ny)
folium.Marker(ny_center).add_to(map_ny)
folium.Circle(roi_center, radius=1500, color='white', fill=True, fill_opacity=0.4).add_to(map_n
y)


folium.GeoJson(bronx_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(staten_island_boroughs, style_function=boroughs_style, name='geojson').add_to(map
_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[82]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Not bad - this nicely covers all the pockets of low restaurant density in Manhattan and Queens closest to United Nations Headquarters.

Let's also create new, more dense grid of location candidates restricted to our new region of interest (let's make our location candidates 100m appart).

In [91]:

```
k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_step = 100
y_step = 100 * k
roi_y_min = roi_center_y - 1500

roi_latitudes = []
roi_longitudes = []
roi_xs = []
roi_ys = []
for i in range(0, int(51/k)):
    y = roi_y_min + i * y_step
    x_offset = 50 if i%2==0 else 0
    for j in range(0, 51):
        x = roi_x_min + j * x_step + x_offset
        d = calc_xy_distance(roi_center_x, roi_center_y, x, y)
        if (d <= 1501):
            lon, lat = xy_to_lonlat(x, y)
            roi_latitudes.append(lat)
            roi_longitudes.append(lon)
            roi_xs.append(x)
            roi_ys.append(y)

print(len(roi_latitudes), 'candidate neighborhood centers generated.')
```

```
821 candidate neighborhood centers generated.
```

OK. Now let's calculate two most important things for each location candidate: **number of restaurants in vicinity** (we'll use radius of **250 meters**) and **distance to closest Chinese restaurant**.

In [92]:

```
def count_restaurants_nearby(x, y, restaurants, radius=250):
    count = 0
    for res in restaurants.values():
        res_x = res[7]; res_y = res[8]
        d = calc_xy_distance(x, y, res_x, res_y)
        if d<=radius:
            count += 1
    return count

def find_nearest_restaurant(x, y, restaurants):
    d_min = 100000
    for res in restaurants.values():
        res_x = res[7]; res_y = res[8]
        d = calc_xy_distance(x, y, res_x, res_y)
        if d<=d_min:
            d_min = d
    return d_min

roi_restaurant_counts = []
roi_chinese_distances = []

print('Generating data on location candidates... ', end='')
for x, y in zip(roi_xs, roi_ys):
    count = count_restaurants_nearby(x, y, restaurants, radius=250)
    roi_restaurant_counts.append(count)
    distance = find_nearest_restaurant(x, y, chinese_restaurants)
    roi_chinese_distances.append(distance)
print('done.')
```

```
Generating data on location candidates... done.
```

In [93]:

```
# Let's put this into dataframe
df_roi_locations = pd.DataFrame({'Latitude':roi_latitudes,
                                 'Longitude':roi_longitudes,
                                 'X':roi_xs,
                                 'Y':roi_ys,
                                 'Restaurants nearby':roi_restaurant_counts,
                                 'Distance to Chinese restaurant':roi_chinese_distances})

df_roi_locations.head(10)
```

Out[93]:

	Latitude	Longitude	X	Y	Restaurants nearby	Distance to Chinese restaurant
0	40.752443	-73.975046	-5.815448e+06	9.865791e+06	1	1016.790382
1	40.753033	-73.975025	-5.815348e+06	9.865791e+06	0	992.062034
2	40.749804	-73.975812	-5.815898e+06	9.865877e+06	1	828.748077
3	40.750394	-73.975791	-5.815798e+06	9.865877e+06	0	833.894020
4	40.750983	-73.975770	-5.815698e+06	9.865877e+06	1	850.843757
5	40.751573	-73.975749	-5.815598e+06	9.865877e+06	1	878.914649
6	40.752162	-73.975728	-5.815498e+06	9.865877e+06	1	917.086049
7	40.752752	-73.975707	-5.815398e+06	9.865877e+06	0	961.059107
8	40.753341	-73.975686	-5.815298e+06	9.865877e+06	0	893.482401
9	40.753931	-73.975664	-5.815198e+06	9.865877e+06	0	832.458404

OK. Let us now **filter** those locations: we're interested only in **locations with no restaurant in radius of 250 meters**, and **no Chinese restaurants in radius of 1000 meters**.

In [102]:

```
good_res_count = np.array((df_roi_locations['Restaurants nearby']<=0))
print('Locations with no restaurant nearby:', good_res_count.sum())

good_chi_distance = np.array(df_roi_locations['Distance to Chinese restaurant']>=1000)
print('Locations with no Chinese restaurants within 1000m:', good_chi_distance.sum())

good_locations = np.logical_and(good_res_count, good_chi_distance)
print('Locations with both conditions met:', good_locations.sum())

df_good_locations = df_roi_locations[good_locations]
```

```
Locations with no restaurant nearby: 398
Locations with no Chinese restaurants within 1000m: 334
Locations with both conditions met: 199
```

Let's see how this looks on a map.

In [103]:

```
good_latitudes = df_good_locations['Latitude'].values
good_longitudes = df_good_locations['Longitude'].values

good_locations = [[lat, lon] for lat, lon in zip(good_latitudes, good_longitudes)]

map_ny = folium.Map(location=roi_center, zoom_start=15)
folium.TileLayer('cartodbpositron').add_to(map_ny)
HeatMap(restaurant_latlons).add_to(map_ny)
folium.Circle(roi_center, radius=1500, color='white', fill=True, fill_opacity=0.6).add_to(map_n
y)
folium.Marker(ny_center).add_to(map_ny)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_
opacity=1).add_to(map_ny)

folium.GeoJson(bronx_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(staten_island_boroughs, style_function=boroughs_style, name='geojson').add_to(map
_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[103]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Looking good. We now have a bunch of locations fairly close to United Nations Headquarters (mostly in Manhattan Middle City), and we know that each of those locations has no restaurant in radius of 250m, and no Chinese restaurant closer than 1000m. Any of those locations is a potential candidate for a new Chinese restaurant, at least based on nearby competition.

Let's now show those good locations in a form of heatmap:

In [118]:

```
map_ny = folium.Map(location=roi_center, zoom_start=15)
HeatMap(good_locations, radius=25).add_to(map_ny)
folium.Marker(ny_center).add_to(map_ny)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_
opacity=1).add_to(map_ny)

folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[118]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Looking good. What we have now is a clear indication of zones with no restaurant in vicinity, and *no* Chinese restaurants at all nearby.

Let us now **cluster** those locations to create **centers of zones containing good locations**. Those zones, their centers and addresses will be the final result of our analysis.

In [110]:

```
from sklearn.cluster import KMeans

number_of_clusters = 8

good_xys = df_good_locations[['X', 'Y']].values
kmeans = KMeans(n_clusters=number_of_clusters, random_state=0).fit(good_xys)

cluster_centers = [xy_to_lonlat(cc[0], cc[1]) for cc in kmeans.cluster_centers_]

map_ny = folium.Map(location=roi_center, zoom_start=15)
folium.TileLayer('cartodbpositron').add_to(map_ny)
HeatMap(restaurant_latlons).add_to(map_ny)
folium.Circle(roi_center, radius=1500, color='white', fill=True, fill_opacity=0.3).add_to(map_n
y)
folium.Marker(ny_center).add_to(map_ny)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=250, color='green', fill=True, fill_opacity=0.25).add_to(m
ap_ny)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_
opacity=1).add_to(map_ny)

folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[110]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Not bad - our clusters represent groupings of most of the candidate locations and cluster centers are placed nicely in the middle of the zones 'rich' with location candidates.

Addresses of those cluster centers will be a good starting point for exploring the neighborhoods to find the best possible location based on neighborhood specifics.

Let's see those zones on a city map without heatmap, using shaded areas to indicate our clusters:

In [112]:

```
map_ny = folium.Map(location=roi_center, zoom_start=15)
folium.Marker(ny_center).add_to(map_ny)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=150, color='#00000000', fill=True, fill_color='#0066ff', fill_opacity=0.07).add_to(map_ny)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_opacity=1).add_to(map_ny)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=300, color='green', fill=False).add_to(map_ny)

folium.GeoJson(brooklyn_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(manhattan_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)
folium.GeoJson(queens_boroughs, style_function=boroughs_style, name='geojson').add_to(map_ny)

map_ny
```

Out[112]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Finaly, let's **reverse geocode those candidate area centers to get the addresses** which can be presented to stakeholders.

In [116]:

```
candidate_area_addresses = []
print('==============================================================')
print('Addresses of centers of areas recommended for further analysis')
print('==============================================================\n')
for lon, lat in cluster_centers:
    addr = geolocator.reverse( [lat, lon]).address.replace(', United States of America', '')
    addr = addr.replace(', New York County, New York', '')
    addr = addr.replace(', Kings County, New York', '')
    addr = addr.replace(', Queens County, New York', '')# We don't need country part of address

    candidate_area_addresses.append(addr)
    x, y = lonlat_to_xy(lon, lat)
    d = calc_xy_distance(x, y, ny_center_x, ny_center_y)
    print('{}{} => {:.1f}km from United Nations Headquarters'.format(addr, ' '*(50-len(addr)), d/1000))
```

```
==============================================================
Addresses of centers of areas recommended for further analysis
==============================================================

330, West 38th Street, Garment District, Manhattan, Manhattan Community Board 4, 1
0018 => 3.5km from United Nations Headquarters
215, West 28th Street, Flower District, Manhattan, Manhattan Community Board 4, 10
001 => 3.5km from United Nations Headquarters
403, West 41st Street, Hell's Kitchen, Manhattan, Manhattan Community Board 4, 100
36 => 3.7km from United Nations Headquarters
Penn Station (Upper Level), 8th Avenue, Chelsea, Manhattan, Manhattan Community Bo
ard 4, 10017 => 3.2km from United Nations Headquarters
320, West 31st Street, Hudson Yards, Manhattan, Manhattan Community Board 4, 10001
=> 3.7km from United Nations Headquarters
Hyatt House, 815, 6th Avenue, Flower District, Manhattan, Manhattan Community Boar
d 5, 10001 => 3.1km from United Nations Headquarters
42nd Street - Times Square (A,C,E), West 43rd Street, Theater District, Manhattan,
Manhattan Community Board 5, 10036 => 3.2km from United Nations Headquarters
237, West 35th Street, Garment District, Herald Square, Manhattan Community Board
5, 10018 => 3.1km from United Nations Headquarters
```

This concludes our analysis. We have created 8 addresses representing centers of zones containing locations with no restaurant and no Chinese restaurants nearby, all zones being fairly close to city center (all less than 4km from United Nations Headquarters, and all of those are more than 3km from that place). Although zones are shown on map with a radius of ~300 meters (green circles), their shape is actually very irregular and their centers/addresses should be considered only as a starting point for exploring area neighborhoods in search for potential restaurant locations. All of the zones are located in Manhattan mid city, which we have identified as interesting due to being popular with tourists, fairly close to Manhatton mid city center and well connected by public transport.

In [119]:

```
map_ny = folium.Map(location=roi_center, zoom_start=15)
folium.Circle(ny_center, radius=50, color='red', fill=True, fill_color='red', fill_opacity=1).a
dd_to(map_ny)

for lonlat, addr in zip(cluster_centers, candidate_area_addresses):
    folium.Marker([lonlat[1], lonlat[0]], popup=addr).add_to(map_ny)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=150, color='#0000ff00', fill=True, fill_color='#0066ff', f
ill_opacity=0.05).add_to(map_ny)

map_ny
```

Out[119]:

Make this Notebook Trusted to load map: File -> Trust Notebook

Results and Discussion

Our analysis shows that although there is a great number of restaurants in New York (561 in our initial area of interest which was 16x16km around United Nations Headquarters), there are pockets of low restaurant density fairly close to city center. Highest concentration of restaurants was in Manhattan, so we focused our attention to this borough. Another borough was identified as potentially interesting (Brooklyn and Queens), but our attention was focused on Manhattan which offer a combination of popularity among tourists, closeness to city center, strong socio-economic dynamics, high density of well-known corporations *and* a number of pockets of low restaurant density.

After directing our attention to this more narrow area of interest (mainly in Manhattan mid city) we first created a dense grid of location candidates (spaced 100m appart); those locations were then filtered so that those with more than two restaurants in radius of 250m and those with an Chinese restaurant closer than 1000m were removed.

Those location candidates were then clustered to create zones of interest which contain greatest number of location candidates. Addresses of centers of those zones were also generated using reverse geocoding to be used as markers/starting points for more detailed local analysis based on other factors.

Result of all this is 8 zones containing largest number of potential new restaurant locations based on number of and distance to existing venues - both restaurants in general and Chinese restaurants particularly. This, of course, does not imply that those zones are actually optimal locations for a new restaurant! Purpose of this analysis was to only provide info on areas close to New York city center but not crowded with existing restaurants (particularly Chinese) - it is entirely possible that there is a very good reason for small number of restaurants in any of those areas, reasons which would make them unsuitable for a new restaurant regardless of lack of competition in the area. Recommended zones should therefore be considered only as a starting point for more detailed analysis which could eventually result in location which has not only no nearby competition but also other factors taken into account and all other relevant conditions met.

Conclusion

Purpose of this project was to identify New York city close to center with low number of restaurants (particularly Chinese restaurants) in order to aid stakeholders in narrowing down the search for optimal location for a new Chinese/ Asian restaurant. By calculating restaurant density distribution from Foursquare data we have first identified general boroughs that justify further analysis (Manhattan, Brooklyn and Queens), and then generated extensive collection of locations which satisfy some basic requirements regarding existing nearby restaurants. In this research, focus is mainly on Manhattan mid city, which is thought as the heart of New York and has low density of both restaurants and Chinese/ Asian restaurants than lower city. Clustering of those locations was then performed in order to create major zones of interest (containing greatest number of potential locations) and addresses of those zone centers were created to be used as starting points for final exploration by stakeholders.

Final decission on optimal restaurant location will be made by stakeholders based on specific characteristics of neighborhoods and locations in every recommended zone, taking into consideration additional factors like attractiveness of each location (proximity to park or water), levels of noise / proximity to major roads, real estate availability, prices, social and economic dynamics of every neighborhood etc.