The Battle of Neighborhoods Project

April 8, 2019

1 Capstone Project - The Battle of the Neighborhoods (Week 2)

1.0.1 Applied Data Science Capstone by IBM/Coursera

1.1 Introduction: Business Problem

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 **restaurant and school** in **New York**, Unite States.

Since there are lots of restaurants 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 Italy restaurant 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.Firstly, we want to explore the center cluster

1.2 Data

Based on definition of our problem, factors that will influence our decision are: * number of existing restaurants in the neighborhood (any type of restaurant) * number of and distance to Italian restaurants in the neighborhood, if any * distance of neighborhood from city center * number of school in the neighborhood (any type of school)

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 **Google Maps API reverse geocoding** * number of restaurants and their type and location in every neighborhood will be obtained using **Foursquare API** * number of schools and their type and location in every neighborhood will be obtained using **Foursquare API** * coordinate of New York center will be obtained using **MapBox API** of well known New York Queens location

1.3 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. 12x12 killometers centered around Berlin city center.

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

```
In [1]: import pandas as pd
        import numpy as np
        from bs4 import BeautifulSoup
        from matplotlib import  pyplot as plt
        import requests
        import folium
        from pandas.io.json import json_normalize
        import matplotlib.cm as cm
        import matplotlib.colors as colors
        from sklearn.cluster import KMeans
        from geopy.geocoders import Nominatim
        import json
        import seaborn as sns
        %matplotlib inline
```

```
In [2]: if True:
            df = pd.read_csv("./data/data.csv")
        #     toronto = pd.read_csv("./data/toronto.csv")
```

1.4 create a geolocator object for each city

```
In [3]: # The code was removed by Watson Studio for sharing.
        google_api_key = ""
```

```
In [4]: def get_coordinates(api_key, address, verbose=False):
            try:
                url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&address={}'.form
                response = requests.get(url).json()
                if verbose:
                    print('Google Maps API JSON result =>', response)
                results = response['results']
                geographical_data = results[0]['geometry']['location'] # get geographical coordi
                lat = geographical_data['lat']
                lon = geographical_data['lng']
                return [lat, lon]
            except:
                return [None, None]

        address = "Queens, New York, United States"
        center = get_coordinates(google_api_key, address)
        print('Coordinate of {}: {}'.format(address, center))
```

```
Coordinate of Queens, New York, United States: [40.7282239, -73.7948516]
```

```
In [ ]: # !pip install shapely
        # !pip install pyproj
```

```
In [5]: #!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)

        print('Coordinate transformation check')
        print('-------------------------------')
        print('Queens center longitude={}, latitude={}'.format(center[1], center[0]))
        x, y = lonlat_to_xy(center[1], center[0])
        print('Queens center UTM X={}, Y={}'.format(x, y))
        lo, la = xy_to_lonlat(x, y)
        print('Queens center longitude={}, latitude={}'.format(lo, la))
```

```
Coordinate transformation check
-------------------------------
Queens center longitude=-73.7948516, latitude=40.7282239
Queens center UTM X=-5818864.983873131, Y=9842433.386218188
Queens center longitude=-73.79485159999955, latitude=40.728223899998895
```

Next step, 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. Besides, let's visualize the data we have so far: city center location and candidate neighborhood centers.

Now let's create a grid of area candidates, same spaced, centered around city center and within ~6km from Queens. Our neighborhoods will be defined as circular areas with a radius of 300 meters, so our neighborhood centers will be 600 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 [6]: center_x, center_y = lonlat_to_xy(center[1], center[0]) # City center in Cartesian coord

        k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
        x_min = center_x - 6000
        x_step = 600
        y_min = center_y - 6000 - (int(21/k)*k*600 - 12000)/2
        y_step = 600 * k

        latitudes = []
        longitudes = []
        distances_from_center = []
        xs = []
        ys = []
        for i in range(0, int(21/k)):
            y = y_min + i * y_step
            x_offset = 300 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(center_x, center_y, x, y)
                if (distance_from_center <= 6001):
                    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.
```

```
In [7]: #!pip install folium

        import folium
```

```
In [8]: map_init = folium.Map(location=center, zoom_start=13)
        folium.Marker(center, popup='Queens').add_to(map_init)
        for lat, lon in zip(latitudes, longitudes):
            #folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue
            folium.Circle([lat, lon], radius=300, color='blue', fill=False).add_to(map_init)
            #folium.Marker([lat, lon]).add_to(map_berlin)
        map_init
```

```
Out[8]: <folium.folium.Map at 0x7f4ad7a50c18>
```

```
In [9]: def get_address(api_key, latitude, longitude, verbose=False):
            try:
                url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&latlng={},{}'.fo
                response = requests.get(url).json()
                if verbose:
                    print('Google Maps API JSON result =>', response)
                results = response['results']
                address = results[0]['formatted_address']
                return address
            except:
                return None

        addr = get_address(google_api_key, center[0], center[1])
        print('Reverse geocoding check')
        print('-----------------------')
        print('Address of [{}, {}] is: {}'.format(center[0], center[1], addr))

Reverse geocoding check
-----------------------
Address of [40.7282239, -73.7948516] is: Virginia Cheriton, Fresh Meadows, NY 11366, USA


In [ ]: print('Obtaining location addresses: ', end='')
        addresses = []
        for lat, lon in zip(latitudes, longitudes):
            address = get_address(google_api_key, lat, lon)
            if address is None:
                address = 'NO ADDRESS'
            address = address.replace(', USA', '') # We don't need country part of address
            addresses.append(address)
            print(' .', end='')
        print(' done.')

Obtaining location addresses:  . . . . . .

In [11]: df = pd.DataFrame({'Address': addresses,
                                    'Latitude': latitudes,
                                    'Longitude': longitudes,
                                    'X': xs,
                                    'Y': ys,
                                    'Distance from center': distances_from_center})

         df.head(10)

Out[11]:                                        Address   Latitude  Longitude  \
         0      93-46 210th Pl, Queens Village, NY 11428  40.716520 -73.751049
         1             211-30 90th Ct, Jamaica, NY 11428  40.720054 -73.750895
         2      89-28 213th St, Queens Village, NY 11427  40.723589 -73.750740
         3   214-46 Whitehall Terrace, Jamaica, NY 11427  40.727124 -73.750586
```

```
4  218-17 Grand Central Pkwy, Jamaica, NY 11427  40.730659 -73.750431
5        220-24 Hartland Ave, Jamaica, NY 11427  40.734194 -73.750276
6           220-72 77th Ave, Flushing, NY 11364  40.737730 -73.750122
7           204-12 100th Ave, Jamaica, NY 11423  40.711321 -73.755302
8   93-52 Francis Lewis Blvd, Jamaica, NY 11428  40.714855 -73.755149
9             90-21 208th St, Jamaica, NY 11428  40.718389 -73.754995

                X             Y  Distance from center
0 -5.820665e+06  9.836718e+06           5992.495307
1 -5.820065e+06  9.836718e+06           5840.376700
2 -5.819465e+06  9.836718e+06           5747.173218
3 -5.818865e+06  9.836718e+06           5715.767665
4 -5.818265e+06  9.836718e+06           5747.173218
5 -5.817665e+06  9.836718e+06           5840.376700
6 -5.817065e+06  9.836718e+06           5992.495307
7 -5.821565e+06  9.837237e+06           5855.766389
8 -5.820965e+06  9.837237e+06           5604.462508
9 -5.820365e+06  9.837237e+06           5408.326913
```

1.5 Get Food Category And School Information

Get the food category and the school information about Queens center by using FourSquare API.

```
In [12]: LIMIT = 500 # limit of number of venues returned by Foursquare API
         radius = 2000 # define radius
         CLIENT_ID = ""
         CLIENT_SECRET = ""
         VERSION = '20181020'
         categoryFood = "4d4b7105d754a06374d81259"
         categorySchool = "4bf58dd8d48988d13d941735"
```

```
In [13]: def getNearbyVenues(name, latitude, longitude, radius=2000, category=""):
             url = 'https://api.foursquare.com/v2/venues/search?&radius={}&'.format(radius)
             expand_infor = "client_id={}&client_secret={}&v={}&ll={},{}&limit={}".format(
                 CLIENT_ID, CLIENT_SECRET, VERSION, latitude, longitude, LIMIT
             )

             if category:
                 category_infor = "&categoryId={}".format(category)
             else:
                 category_infor = ""
             # merge the url
             url = url + expand_infor + category_infor


             # extract infromation
             response = requests.get(url).json()

             return response
```

```
In [ ]: # use the api to get category food
        df["CategoryFood"] = df.apply(lambda x: getNearbyVenues(
                x["Address"], x["Latitude"], x["Longitude"], category=categoryFood
            ), axis=1)

In [ ]: # use the api to get school infor
        df["CategorySchool"] = df.apply(lambda x: getNearbyVenues(
                x["Address"], x["Latitude"], x["Longitude"], category=categorySchool
            ), axis=1)
```

1.6 Parse School Information & Food Information

Next step, we want get the number of food category and the number of school

```
In [14]: def get_category(x, target="category"):
             """
             parameters:
             ----------
             target: string, default category
                 Choose target information. Like category, name, location
             """
             information = dict(
                 Name = [],
                 Location = [],
                 Category = []
                 )
             if "response" not in x or "venues" not in x["response"]:
                 return np.nan
             for item in x["response"]["venues"]:
                 name = item["name"]
                 location = (item["location"]["lat"], item["location"]["lng"])
                 category = item["categories"][0]["name"]

                 if name and location and category:
                     information["Name"].append(name)
                     information["Location"].append(location)
                     information["Category"].append(category)
             if target == "category":
                 return information["Category"]
             elif target == "name":
                 return information["Name"]
             elif target == "location":
                 return information["Location"]
             elif target in ["all", ""]:
                 return information
```

```
In [ ]: df["FoodInformation"] = df.CategoryFood.apply(get_category, target="all")
        df["SchoolInformation"] = df.CategorySchool.apply(get_category, target="all")
```

```
In [15]: df.head(2)
```

Out[15]:

	Address	Latitude	Longitude	X	Y	Distance from center	CategoryFood	CategorySchool	FoodInformation	SchoolInformation	...	African Restaurant	Vegetarian / Vegan Restaurant	New American Restaurant	Chinese Restaurant	Tapas Restaurant	Indian Restaurant	Dumpling Restaurant	Halal Restaurant	High School	Elementary School
0	93-46 210th Pl, Queens Village, NY 11428	40.716520	-73.751049	-5.820665e+06	9.836718e+06	5992.495307	{"meta": {"code": 200, "requestId": "5caa2df5f...	{"meta": {"code": 200, "requestId": "5caa2f4cd...	{"Name": ["Dunkin'", "Le Bon Pain", "Papa John...	{"Name": ["Queens Satellite Highschool", "Path...	...	0	0	0	5	0	0	0	0	8	0
1	211-30 90th Ct, Jamaica, NY 11428	40.720054	-73.750895	-5.820665e+06	9.836718e+06	5840.376700	{"meta": {"code": 200, "requestId": "5caa2df64...	{"meta": {"code": 200, "requestId": "5caa2f4dd...	{"Name": ["Baskin Robbins", "Sushi You", "Hot ...	{"Name": ["Queens Satellite Highschool", "Path...	...	0	0	0	4	0	0	1	0	7	0

[2 rows x 50 columns]

```
In [16]: foodtype = set()
         for i in df.FoodInformation:
             if pd.notna(i):
                 i = json.loads(i)
                 foodtype.update(set(i["Category"]))
```

```
In [ ]: foodtype = pd.Series(list(foodtype))
        for column in foodtype[foodtype.str.contains("Restaurant", case=False)]:
            df[column] = df.FoodInformation.apply(lambda x: x["Category"].count(column) if pd.no
```

```
In [17]: schooltype = set()
         for i in df.SchoolInformation:
             if pd.notna(i):
                 i = json.loads(i)
                 schooltype.update(set(i["Category"]))

In [ ]: schooltype = pd.Series(list(schooltype))
        for column in schooltype[schooltype.str.contains("College|Elementary|School|University",
            df[column] = df.SchoolInformation.apply(lambda x: x["Category"].count(column) if pd.

In [ ]: schooltype

In [18]: df.head(2)

Out[18]:                                    Address   Latitude  Longitude  \
         0  93-46 210th Pl, Queens Village, NY 11428  40.716520 -73.751049
         1         211-30 90th Ct, Jamaica, NY 11428  40.720054 -73.750895

                       X             Y  Distance from center  \
         0 -5.820665e+06  9.836718e+06           5992.495307
         1 -5.820065e+06  9.836718e+06           5840.376700

                                              CategoryFood  \
         0  {"meta": {"code": 200, "requestId": "5caa2df5f...
         1  {"meta": {"code": 200, "requestId": "5caa2df64...

                                             CategorySchool  \
         0  {"meta": {"code": 200, "requestId": "5caa2f4cd...
         1  {"meta": {"code": 200, "requestId": "5caa2f4dd...

                                            FoodInformation  \
         0  {"Name": ["Dunkin'", "Le Bon Pain", "Papa John...
         1  {"Name": ["Baskin Robbins", "Sushi You", "Hot ...

                                         SchoolInformation  ...  African Restaurant  \
         0  {"Name": ["Queens Satellite Highschool", "Path...  ...                   0
         1  {"Name": ["Queens Satellite Highschool", "Path...  ...                   0

            Vegetarian / Vegan Restaurant  New American Restaurant  Chinese Restaurant  \
         0                              0                        0                   5
         1                              0                        0                   4

            Tapas Restaurant  Indian Restaurant  Dumpling Restaurant  Halal Restaurant  \
         0                 0                  0                    0                 0
         1                 0                  0                    0                 1

            High School  Elementary School
         0            8                  0
         1            7                  0
```

```
[2 rows x 50 columns]
```

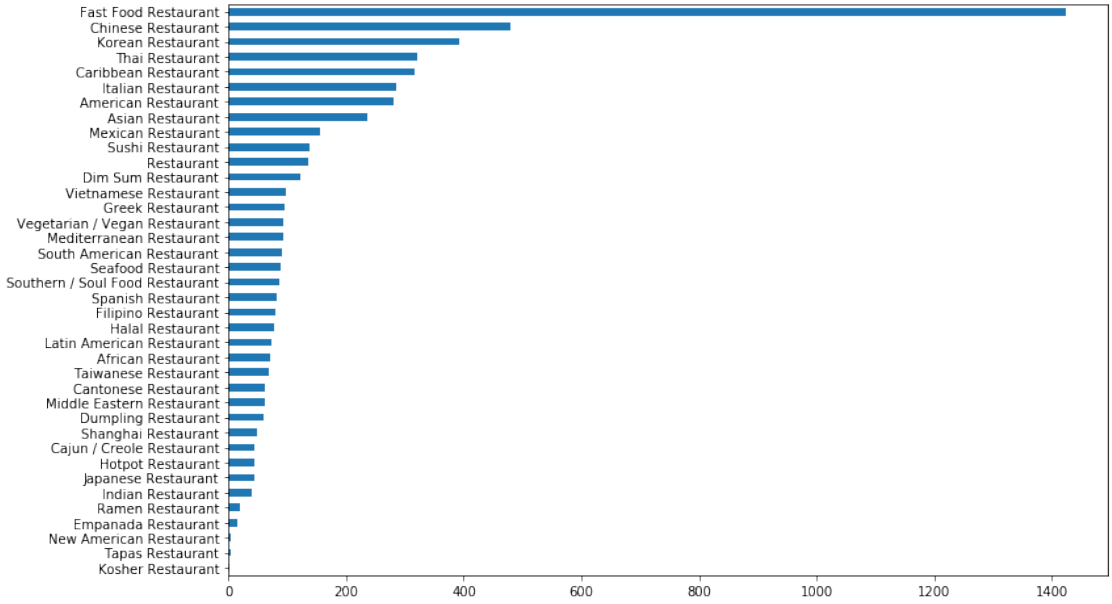
```
In [ ]: # store data
        if False:
            df.to_csv("./data/data.csv", index=False)
```

1.7 Display Information

Before we cluter the 364 location, we need to explore the school and the food category. We map the information on the map, so that we can explore the food and the school information clearly

```
In [20]: restaurant = df[
             df.columns[df.columns.str.contains("Restaurant", case=False)]
         ].sum()
```

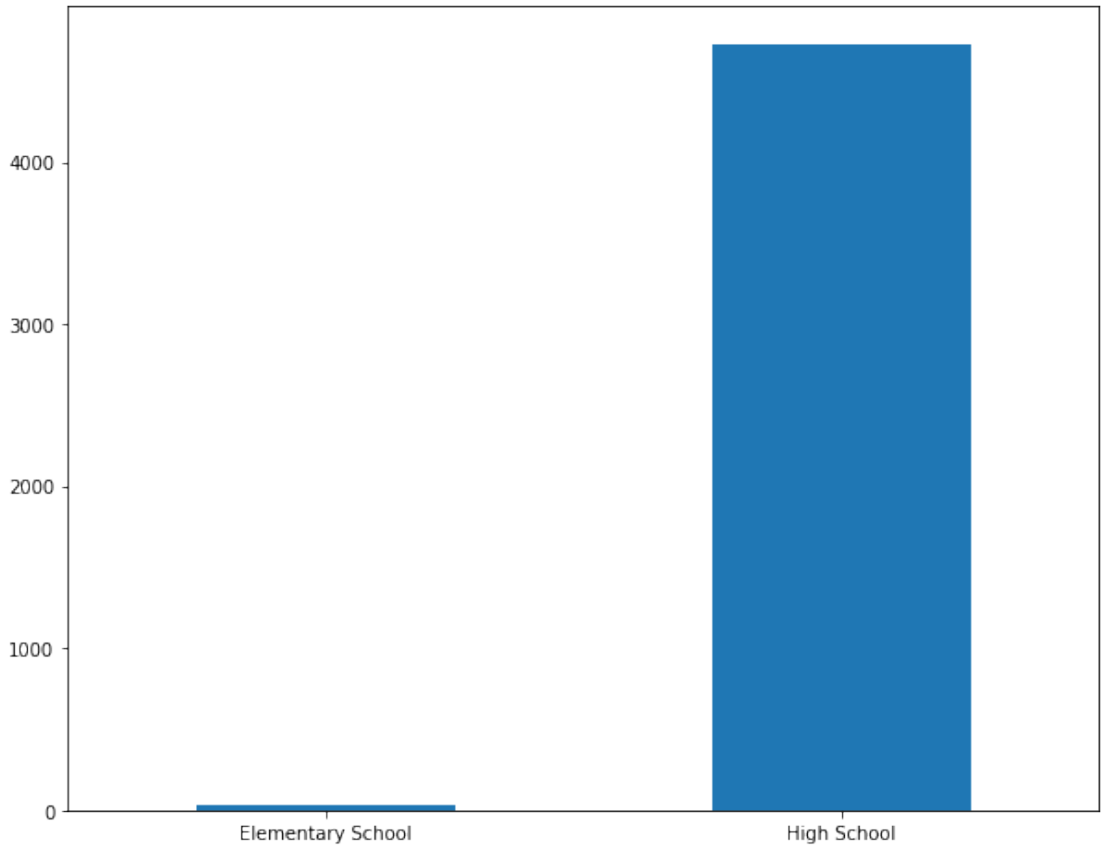
```
In [21]: restaurant.sort_values().plot(kind="barh", figsize=(12, 8));
```




```
In [22]: school = df[["High School", "Elementary School"]].sum()
```

```
In [ ]: school
```

```
In [23]: school.sort_values().plot(kind="bar", figsize=(10, 8), rot=0);
```



1.8 Map Information

```
In [26]: df["AllRestarant"] = df[
             df.columns[df.columns.str.contains("Restaurant", case=False)]
         ].sum(axis=1)

In [27]: df["AllSchool"] = df[["High School", "Elementary School"]].sum(axis=1)

In [29]: foodlocation = []
         for i in df["FoodInformation"]:
             if pd.notna(i):
                 i = json.loads(i)
                 foodlocation += i["Location"]

In [30]: center

Out[30]: [40.7282239, -73.7948516]

In [31]: map_init = folium.Map(location=center, zoom_start=13)
         folium.Marker(center, popup='Queens').add_to(map_init)
         # # add markers to map
```

```
        # label = '{}'.format(addr)
        # label = folium.Popup(label, parse_html=True)
        # # folium.Marker(center, popup=label).add_to(map_init)

        for lat, lon, sc, ar in zip(df["Latitude"], df["Longitude"], df["AllSchool"], df["AllRe
            if sc != 0:
                folium.CircleMarker([lat, lon], radius=sc * .2, color="blue", fill=True,
                                    fill_color="blue", fill_opacity=.3).add_to(map_init)
            if ar != 0:
                folium.CircleMarker([lat, lon], radius=ar * .2, color="red", fill=True,
                                    fill_color="red", fill_opacity=.3).add_to(map_init)
        #     label = 'School:{}\nRestaurant:{}'.format(sc, ar)
        #     label = folium.Popup(label, parse_html=True)
        #     folium.Marker([lat, lon], popup=label).add_to(map_init)

        map_init
```

Out[31]: <folium.folium.Map at 0x7f4ad4ca6fd0>

```
In [32]: df.columns
```

```
Out[32]: Index(['Address', 'Latitude', 'Longitude', 'X', 'Y', 'Distance from center',
                'CategoryFood', 'CategorySchool', 'FoodInformation',
                'SchoolInformation', 'Thai Restaurant', 'Mediterranean Restaurant',
                'Cantonese Restaurant', 'Sushi Restaurant', 'Latin American Restaurant',
                'Mexican Restaurant', 'Southern / Soul Food Restaurant',
                'American Restaurant', 'Filipino Restaurant',
                'Cajun / Creole Restaurant', 'Korean Restaurant',
                'Fast Food Restaurant', 'Spanish Restaurant', 'Caribbean Restaurant',
                'Italian Restaurant', 'Vietnamese Restaurant', 'Taiwanese Restaurant',
                'Asian Restaurant', 'South American Restaurant', 'Ramen Restaurant',
                'Middle Eastern Restaurant', 'Seafood Restaurant', 'Hotpot Restaurant',
                'Shanghai Restaurant', 'Japanese Restaurant', 'Empanada Restaurant',
                'Greek Restaurant', 'Restaurant', 'Dim Sum Restaurant',
                'Kosher Restaurant', 'African Restaurant',
                'Vegetarian / Vegan Restaurant', 'New American Restaurant',
                'Chinese Restaurant', 'Tapas Restaurant', 'Indian Restaurant',
                'Dumpling Restaurant', 'Halal Restaurant', 'High School',
                'Elementary School', 'AllRestarant', 'AllSchool'],
               dtype='object')
```

```
In [33]: train_data = df.drop(["Address", "CategoryFood", "CategorySchool", "FoodInformation", "
```

1.9 Create Cluster

We want to explore the number of cluster. So we use the KMeans algorithm to create model. Now we must check out how many clusters in the 364 candidate locations.

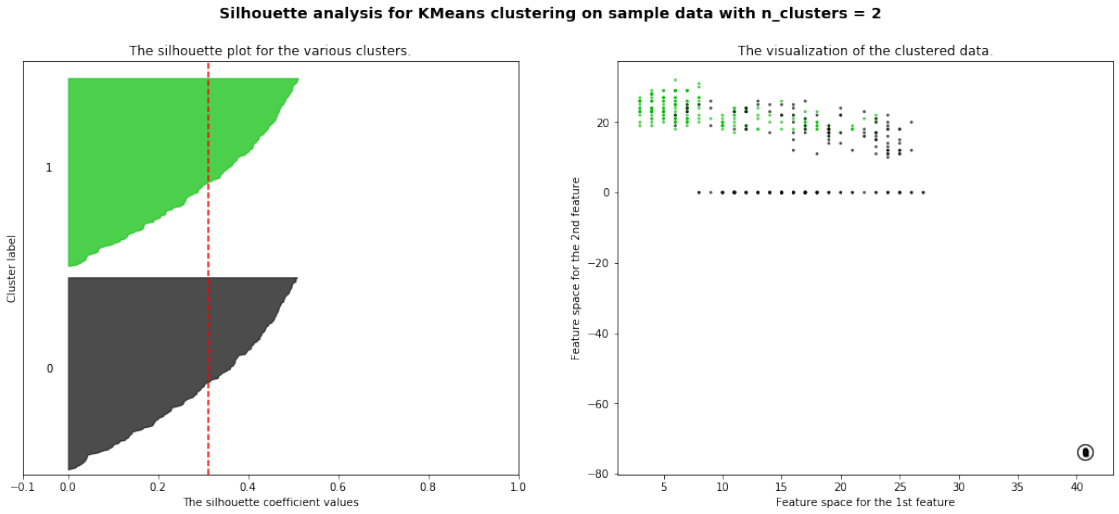
```
In [34]: from sklearn.metrics import silhouette_samples, silhouette_score
```

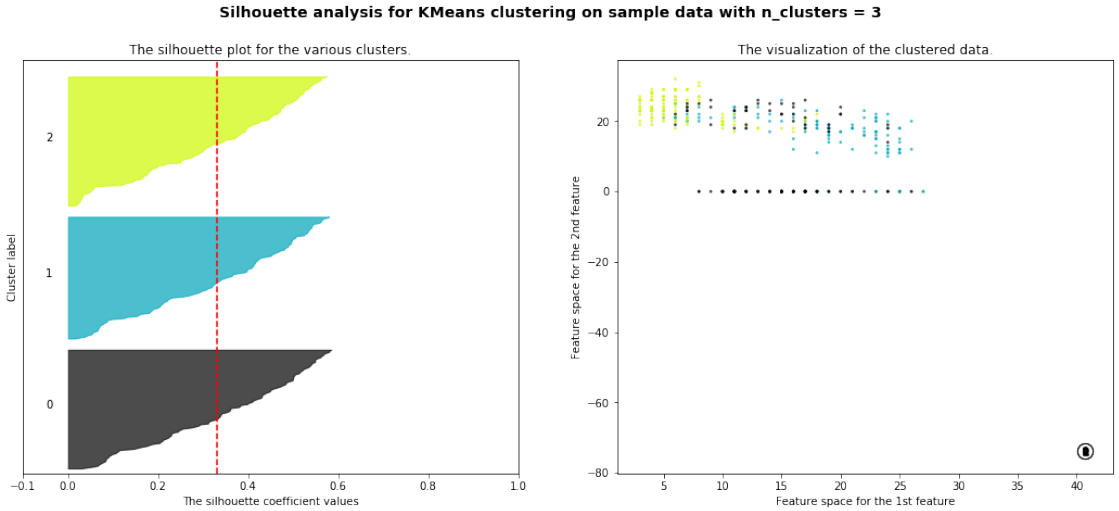
```
        import matplotlib.pyplot as plt
        #import matplotlib.cm as cm
        import numpy as np

In [35]: sse={}
        for n_clusters in range(2, 9):
            # Create a subplot with 1 row and 2 columns
            fig, (ax1, ax2) = plt.subplots(1, 2)
            fig.set_size_inches(18, 7)

            # The 1st subplot is the silhouette plot
            # The silhouette coefficient can range from -1, 1

            ax1.set_xlim([-0.1, 1])
            # The (n_clusters+1)*10 is for inserting blank space between silhouette
            # plots of individual clusters, to demarcate them clearly.
            ax1.set_ylim([0, len(train_data) + (n_clusters + 1) * 10])

            # Initialize the clusterer with n_clusters value and a random generator
            # seed of 10 for reproducibility.
            clusterer = KMeans(n_clusters=n_clusters, random_state=10)
            cluster_labels = clusterer.fit_predict(train_data)

            # The silhouette_score gives the average value for all the samples.
            # This gives a perspective into the density and separation of the formed
            # clusters
            silhouette_avg = silhouette_score(train_data, cluster_labels)
            print("For", n_clusters, " Clusters  "
                  "the average silhouette_score is :", silhouette_avg)
            sse[n_clusters] = silhouette_avg
            # Compute the silhouette scores for each sample
            sample_silhouette_values = silhouette_samples(train_data, cluster_labels)

            y_lower = 5
            for i in range(n_clusters):
                # Aggregate the silhouette scores for samples belonging to
                # cluster i, and sort them
                ith_cluster_silhouette_values = \
                    sample_silhouette_values[cluster_labels == i]

                ith_cluster_silhouette_values.sort()

                size_cluster_i = ith_cluster_silhouette_values.shape[0]
                y_upper = y_lower + size_cluster_i

                color1 = cm.nipy_spectral(float(i) / n_clusters)
                ax1.fill_betweenx(np.arange(y_lower, y_upper),
                                  0, ith_cluster_silhouette_values,
```

```
                                    facecolor=color1, edgecolor=color1, alpha=0.7)

            # Label the silhouette plots with their cluster numbers at the middle
            ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

            # Compute the new y_lower for next plot
            y_lower = y_upper + 10  # 10 for the 0 samples

        ax1.set_title("The silhouette plot for the various clusters.")
        ax1.set_xlabel("The silhouette coefficient values")
        ax1.set_ylabel("Cluster label")

        # The vertical line for average silhouette score of all the values
        ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

        ax1.set_yticks([])  # Clear the yaxis labels / ticks
        ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

        # 2nd Plot showing the actual clusters formed
        colors2 = cm.nipy_spectral(cluster_labels.astype(float) / n_clusters)
        ax2.scatter(train_data.loc[:, "AllSchool"], train_data.loc[:, "AllRestarant"],
                    marker='.', s=30, lw=0, alpha=0.7, c=colors2, edgecolor='k')

        # Labeling the clusters
        centers = clusterer.cluster_centers_
        # Draw white circles at cluster centers
        ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                    c="white", alpha=1, s=200, edgecolor='k')

        for i, c in enumerate(centers):
            ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                        s=50, edgecolor='k')

        ax2.set_title("The visualization of the clustered data.")
        ax2.set_xlabel("Feature space for the 1st feature")
        ax2.set_ylabel("Feature space for the 2nd feature")

        plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                      "with n_clusters = %d" % n_clusters),
                     fontsize=14, fontweight='bold')

    plt.show()
```

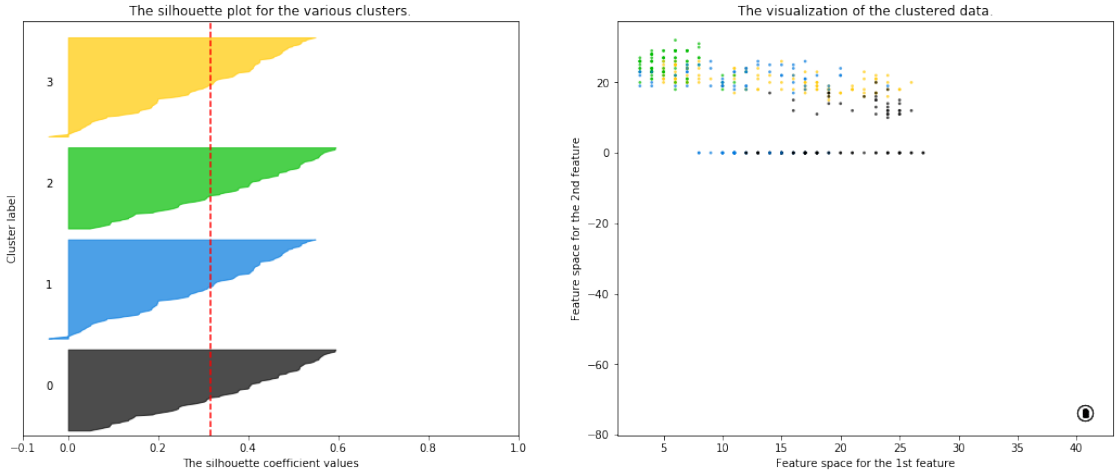
```
For 2  Clusters  the average silhouette_score is : 0.3117313248980779
For 3  Clusters  the average silhouette_score is : 0.33131850079862557
For 4  Clusters  the average silhouette_score is : 0.3151641991559406
For 5  Clusters  the average silhouette_score is : 0.3584715125165422
For 6  Clusters  the average silhouette_score is : 0.35448521619051715
```

```
For 7  Clusters  the average silhouette_score is : 0.3426151420642198
For 8  Clusters  the average silhouette_score is : 0.3234411632415881
```

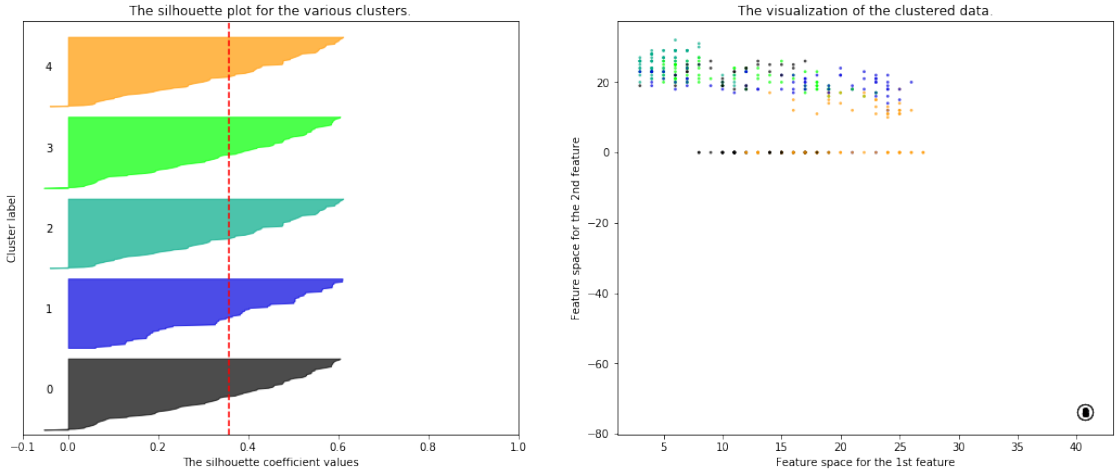




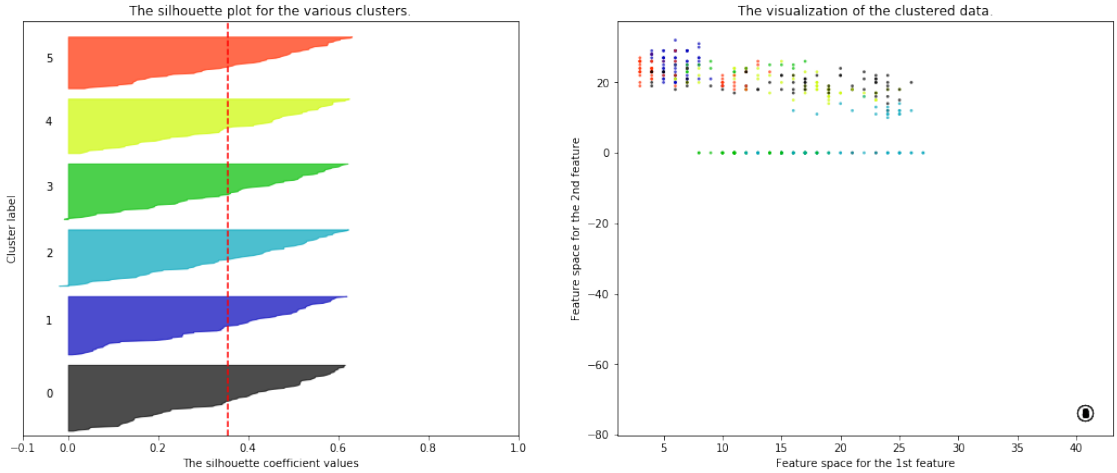
Silhouette analysis for KMeans clustering on sample data with n_clusters = 4



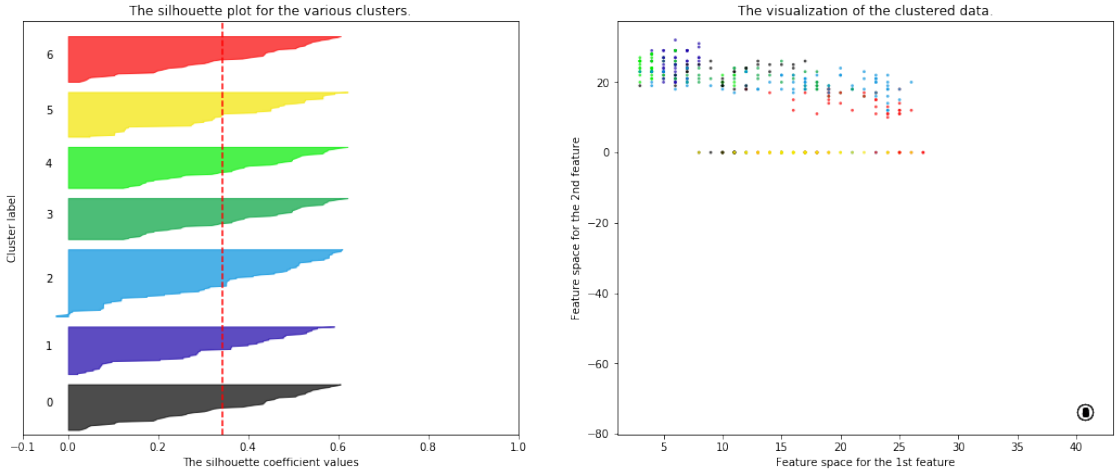
Silhouette analysis for KMeans clustering on sample data with n_clusters = 5

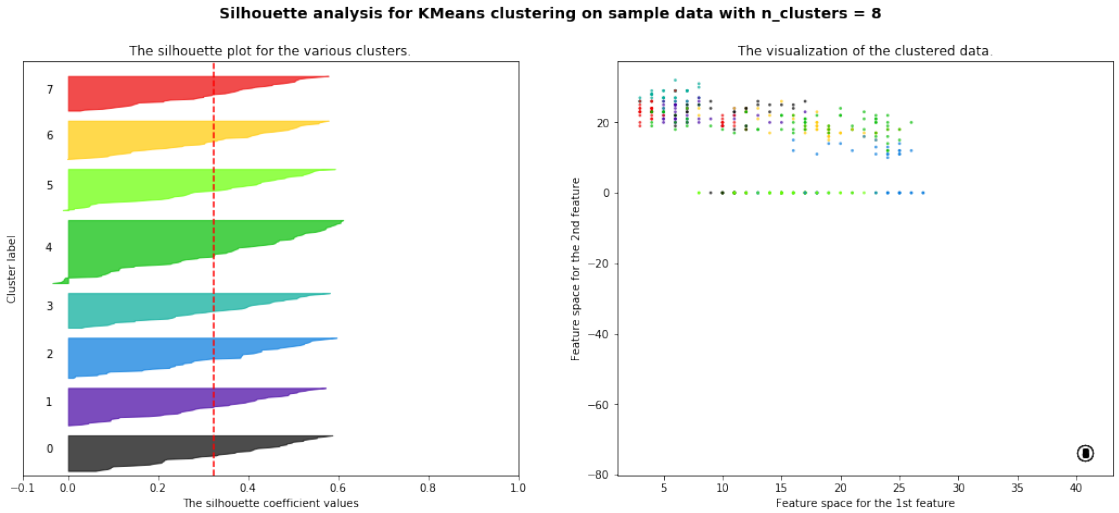


Silhouette analysis for KMeans clustering on sample data with n_clusters = 6

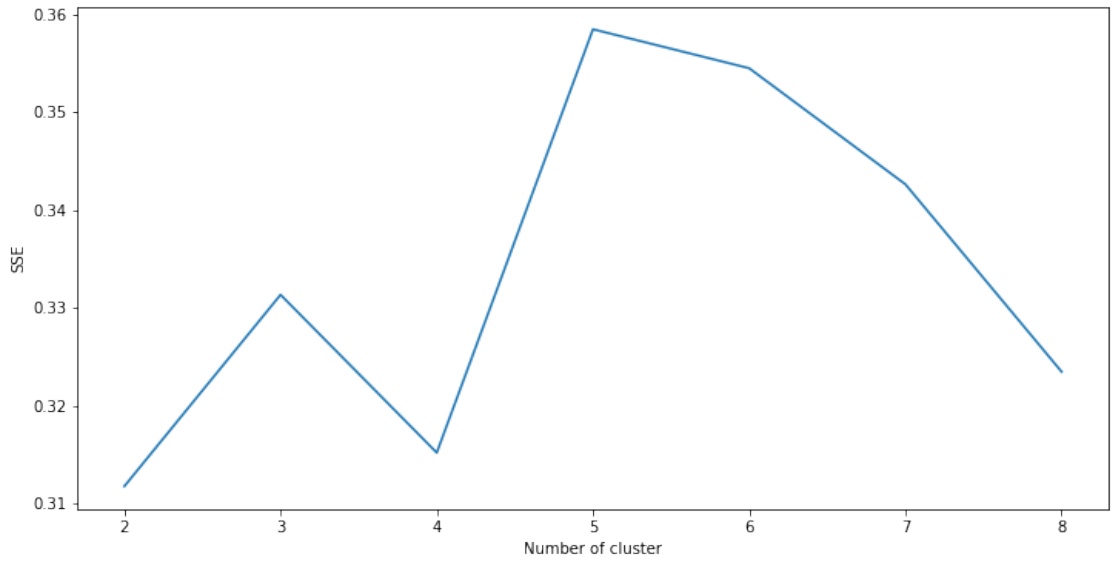


Silhouette analysis for KMeans clustering on sample data with n_clusters = 7





In [36]:
```
plt.figure(figsize=(12, 6))
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```



In [37]:
```
kclusters = 5
clusterer = KMeans(n_clusters=kclusters, random_state=10)
cluster_labels = clusterer.fit_predict(train_data)
train_data["Cluster"] = cluster_labels
```

In [38]:
```
train_data.Cluster.unique()
```

Out[38]:
```
array([3, 2, 1, 0, 4])
```

In [39]:
```
train_data.head(2)
```

Out[39]:

	Latitude	Longitude	X	Y	Distance from center	\
0	40.716520	-73.751049	-5.820665e+06	9.836718e+06	5992.495307	
1	40.720054	-73.750895	-5.820065e+06	9.836718e+06	5840.376700	

	Thai Restaurant	Mediterranean Restaurant	Cantonese Restaurant	\
0	0	0	0	
1	0	0	0	

	Sushi Restaurant	Latin American Restaurant	...	Chinese Restaurant	\
0	2	1	...	5	
1	2	1	...	4	

	Tapas Restaurant	Indian Restaurant	Dumpling Restaurant	Halal Restaurant	\
0	0	0	0	0	
1	0	0	0	1	

	High School	Elementary School	AllRestarant	AllSchool	Cluster
0	8	0	26	8	3
1	7	0	24	7	3

[2 rows x 48 columns]

In [40]:
```
map_init = folium.Map(location= center, zoom_start=13, tiles="CartoDB dark_matter")
import matplotlib.colors as colors
# set color scheme for the clusters
x = np.arange(kclusters)
colors_array = cm.rainbow(np.linspace(0, 1, kclusters))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, cluster in zip(train_data['Latitude'], train_data['Longitude'], train_dat
    folium.CircleMarker(
        [lat, lon],
        radius=3,
#         popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=0.7).add_to(map_init)

map_init
```

Out[40]:
```
<folium.folium.Map at 0x7f4acc1bfcc0>
```

1.10 Results and Discussion

Our analysis shows that although there is a great number of restaurants in New York City (~2000 in our initial area of interest which was 12x12km around Queens), the fast food restaurant is main type close to city center. The eastern Queens has much more food restaurant than the western Queens center, while the school is opposite. So, if we want to choose a appropriate location to open a new restaurant, we choose from the target.