Capstone-hw5

July 18, 2020

1. Introduction/Business Problem section

i) Background

Korean food(Asian food) is getting more and more famous in Seattle lately. We can see the Korean dining is everywhere in United States, especially in Los angela. We are going to find the best location to open in Seattle and do more research about the business opportuntiy in Seattle.

This report will try to gather data about other restaurant localization, competitors and best localization.

ii) Problem

As the goal of this is to create a business plan in the end, we need to make sure data from api are correct. We also need to check that customer could be interested in this specific business.

In order to do so, a survey in Seattle and Los Angeles will be done in addition to data gathering. I'll go in the cities and check at different hours if restaurants are working, if streets are full and so on, and what kind of restaurant works well. This survey will allow to validate the data analysis done here.

2. Data

This notebook is highly inspirated by the template given in the course. I will keep the idea of clustering the city by area and then plot heatmap to find better area.

I will change some data:

Country/City: United States Goal: Open a restaurant/little shop for workers in weekday and maybe saturday So, I will cross data from working days, and localisations.

I will use the following API:

Foursquare API: to find restaurant/venues Google API: reverse geolocalisation

[1]:
```
# Others imports:
from IPython.display import Image
import pickle
import json
import requests
import folium
import pandas as pd

# !pip install shapely
import shapely.geometry
```

```
# !pip install pyproj
import pyproj

import math
from geopy.geocoders import Nominatim

import warnings
warnings.simplefilter("ignore")
```

[2]:
```
CLIENT_ID = 'JBREGZ4UNA53HX43WMAD4TQ2X2XJWMX5DPHEZEIZHQA0ACNP' # your␣
 ↪Foursquare ID
CLIENT_SECRET = 'VNS40KF3V4MGSWWAV0IGQINZIGIT1EQKNCWBFPOS3QF1JMOJ' # your␣
 ↪Foursquare Secret
VERSION = '20180605' # Foursquare API version
GOOGLE_API_KEY='AIzaSyD2R_vyfevRHSv41bvf6AhnVyXcsjnZsWc'
```

[3]:
```
def get_coordinates(address):
    geolocator = Nominatim(user_agent="ny_explorer")
    location = geolocator.geocode(address)
    latitude = location.latitude
    longitude = location.longitude
    return [latitude, longitude]



address = 'Seattle, WA'
    #address = '925 4th Ave #3300, Seattle, WA 98104, USA'
Seattle_center = get_coordinates(address)
print('Coordinate of {}: {}'.format(address, Seattle_center))
```

```
Coordinate of Seattle, WA: [47.6038321, -122.3300624]
```

I created a grid of area candidates, equaly spaced, centered around city center and within ~1.5km from Prefecture. Our neighborhoods will be defined as circular areas with a radius of 100 meters, so our neighborhood centers will be 200 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).

[4]:
```
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('Seattle center longitude={}, latitude={}'.format(Seattle_center[1],␣
 ↪Seattle_center[0]))
x, y = lonlat_to_xy(Seattle_center[1], Seattle_center[0])
print('Seattle center UTM X={}, Y={}'.format(x, y))
lo, la = xy_to_lonlat(x, y)
print('Seattle center longitude={}, latitude={}'.format(lo, la))
```

```
Coordinate transformation check
-------------------------------
Seattle center longitude=-122.3300624, latitude=47.6038321
Seattle center UTM X=-2652222.8719972586, Y=13773159.99848766
Seattle center longitude=-122.33006239999997, latitude=47.60383210000003
```

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.

[5]:
```
Seattle_center_x, Seattle_center_y = lonlat_to_xy(Seattle_center[1],␣
 ↪Seattle_center[0]) # City center in Cartesian coordinates
nb_k = 10
radius = 100
k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = Seattle_center_x - radius*10
x_step = radius*2
y_min = Seattle_center_y - radius*2 - (int(nb_k/k)*k*radius*2 - radius*10)/2
y_step = radius*2 * k

latitudes = []
longitudes = []
distances_from_center = []
xs = []
ys = []
for i in range(0, int(nb_k/k)):
    y = y_min + i * y_step
    x_offset = radius if i%2==0 else 0
    for j in range(0, nb_k):
        x = x_min + j * x_step + x_offset
```

```
        distance_from_center = calc_xy_distance(Seattle_center_x,␣
↪Seattle_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)

map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
for lat, lon in zip(latitudes, longitudes):
    folium.Circle([lat, lon], radius=radius, color='blue', fill=False).
↪add_to(map_seattle)
map_seattle
```

[5]:
```
<folium.folium.Map at 0x7fe19e3e0588>
```

get approximate addresses of those locations

[6]:
```
from geopy.point import Point
def get_address(latitude, longitude):
    geolocator = Nominatim(user_agent="ny_explorer")
    location = geolocator.reverse(Point(latitude,longitude))
    return (location.address)

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

```
Reverse geocoding check
-----------------------
Address of [47.6038321, -122.3300624] is: Seattle City Hall, 600, 4th Avenue,
West Edge, International District/Chinatown, Seattle, King County, Washington,
98104, United States of America
```

[7]:
```
addresses = []
compteur = 0

df_locations = pd.DataFrame()
loaded = False
try:
    with open('locations.pkl', 'rb') as f:
        df_locations = pickle.load(f)
    print('Location data loaded from pickle.')
    loaded = True
```

```
except:
    pass


if not loaded:
    print('Obtaining location addresses: ', end='')
    for lat, lon in zip(latitudes, longitudes):
        compteur = compteur + 1
        address = get_address(lat, lon)
        if address is None:
            address = 'NO ADDRESS'
        address = address.replace(', United States of America', '') # We don't␣
 ↪need country part of address
        addresses.append(address)
        if compteur > 500:
            print("Urgency exit")
            break
    #     print(compteur)
        print(' .', end='')
    print(' done.')
```

```
Location data loaded from pickle.
```

[8]:
```
if not loaded:
    addresses
```

[9]:
```
if not loaded:
    df_locations = pd.DataFrame({'Address': addresses,
                                 'Latitude': latitudes,
                                 'Longitude': longitudes,
                                 'X': xs,
                                 'Y': ys,
                                 'Distance from center': distances_from_center})

df_locations.head(10)
```

[9]:
```
                                             Address   Latitude   Longitude  \
0  217, 12th Avenue, Central Business District, Y...  47.604092 -122.316915
1  300, 10th Avenue, Central Business District, Y...  47.604993 -122.318870
2  412, Broadway, Central Business District, Yesl...  47.605895 -122.320825
3  Swedish First Hill Medical Center, 747, Broadw...  47.606796 -122.322781
4  O'Dea High School, 802, Terry Avenue, Central ...  47.607697 -122.324736
5  Bloodworks Northwest, 921, Terry Avenue, Centr...  47.608598 -122.326692
6  Virginia Mason Hospital & Seattle Medical Cent...  47.609500 -122.328648
7  1215, 8th Avenue, Central Business District, F...  47.610401 -122.330604
8  Washington State Convention Center, 800, Conve...  47.611302 -122.332560
9  Hotel Theodore, 1531, 7th Avenue, Central Busi...  47.612203 -122.334516
```

```
                X             Y  Distance from center
0 -2.653123e+06  1.377251e+07           1111.720843
1 -2.652923e+06  1.377251e+07            957.038784
2 -2.652723e+06  1.377251e+07            822.145506
3 -2.652523e+06  1.377251e+07            718.277964
4 -2.652323e+06  1.377251e+07            660.244828
5 -2.652123e+06  1.377251e+07            660.244828
6 -2.651923e+06  1.377251e+07            718.277964
7 -2.651723e+06  1.377251e+07            822.145506
8 -2.651523e+06  1.377251e+07            957.038784
9 -2.651323e+06  1.377251e+07           1111.720843
```

[10]:
```
df_locations.to_pickle('./locations.pkl')
```

3. Methodology

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 the ones who can be competitors, this mean food truck, quick food, take away, healthy, not restaurant taking too long.

[11]:
```
foursquare_client_id = CLIENT_ID
foursquare_client_secret = CLIENT_SECRET
```

[12]:
```
# Category IDs corresponding to Italian restaurants were taken from Foursquare␣
 ↪web site
# (https://developer.foursquare.com/docs/resources/categories):

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

# We will add some asian categories, and also take away food, and healthy food.␣
 ↪These category are the one that
# may be competitor
asian_restaurant_categories =␣
 ↪['4bf58dd8d48988d142941735','4bf58dd8d48988d145941735',␣
 ↪'4bf58dd8d48988d111941735',
                                '4bf58dd8d48988d1d2941735',␣
 ↪'4bf58dd8d48988d1d1941735', '4bf58dd8d48988d14a941735',
                              '56aa371be4b08b9a8d57350b',␣
 ↪'4bf58dd8d48988d1cb941735', '4bf58dd8d48988d1e0931735',
                              '4bf58dd8d48988d16c941735',␣
 ↪'4bf58dd8d48988d16f941735', '5283c7b4e4b094cb91ec88d7',
                              '4bf58dd8d48988d1bd941735',␣
 ↪'4bf58dd8d48988d1c5941735', '4bf58dd8d48988d1c7941735']

def is_restaurant(categories, specific_filter=None):
    restaurant_words = ['Restaurant', 'restaurant', 'diner', 'taverna',␣
 ↪'steakhouse', 'Brasserie', 'Creperie', 'Café',
```

```
                        'Truck', 'Sandwich', 'Pizza']
    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'])
    return address
def get_venues_near_location(lat, lon, category, client_id, client_secret,␣
 ↪radius=500, limit=100):
    version = '20180724'
    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
```

[13]:
```
# 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

def get_restaurants(lats, lons):
```

```python
    from tqdm.autonotebook import tqdm
    tqdm.pandas()
    restaurants = {}
    asian_restaurants = {}
    location_restaurants = []

    print('Obtaining venues around candidate locations:', end='')
    pbar = tqdm(total=len(lats))
    for lat, lon in zip(lats, lons):
        # Using radius=350 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=350,␣
 ↪limit=100)
        area_restaurants = []
        for venue in venues:
#             with open('2. venue.txt', 'w') as outfile:
#                 json.dump(venue, outfile)
            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_asian = is_restaurant(venue_categories,␣
 ↪specific_filter=asian_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_asian, x, y)

                if venue_distance<=100:
                    area_restaurants.append(restaurant)
                restaurants[venue_id] = restaurant
                if is_asian:
                    asian_restaurants[venue_id] = restaurant
        pbar.update(1)

        location_restaurants.append(area_restaurants)
#         print(' .', end='')
    pbar.close()
#     print(' done.')
    return restaurants, asian_restaurants, location_restaurants
```

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


try:
    with open('restaurants_350.pkl', 'rb') as f:
        restaurants = pickle.load(f)
    with open('asian_restaurants_350.pkl', 'rb') as f:
        asian_restaurants = pickle.load(f)
    with open('location_restaurants_350.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, asian_restaurants, location_restaurants =␣
 ↪get_restaurants(latitudes, longitudes)


    # Let's persists this in local file system
    with open('restaurants_350.pkl', 'wb') as f:
        pickle.dump(restaurants, f)
    with open('asian_restaurants_350.pkl', 'wb') as f:
        pickle.dump(asian_restaurants, f)
    with open('location_restaurants_350.pkl', 'wb') as f:
        pickle.dump(location_restaurants, f)
```

```
Restaurant data loaded.
```

[14]:
```
import numpy as np

print('Total number of restaurants:', len(restaurants))
print('Total number of Asian restaurants:', len(asian_restaurants))
print('Percentage of Asian restaurants: {:.2f}%'.format(len(asian_restaurants) /
 ↪ len(restaurants) * 100))
print('Average number of restaurants in neighborhood:', np.array([len(r) for r␣
 ↪in location_restaurants]).mean())
```

```
Total number of restaurants: 309
Total number of Asian restaurants: 165
```

```
Percentage of Asian restaurants: 53.40%
Average number of restaurants in neighborhood: 1.4636363636363636
```

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

[15]:
```
map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_asian = res[6]
    color = 'red' if is_asian else 'blue'
    label = '{}'.format(res[1])
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
 ↪fill_color=color,
                        popup=label, fill_opacity=1, parse_html=False).
 ↪add_to(map_seattle)
map_seattle
```

[15]: <folium.folium.Map at 0x7fe19e6b2a58>

So, on this map we can see all potential competitors, take away restaurant, healthy restaurant. Asian restaurant are in red.

No we need to analyse companies/university localisation and cluster and cross both analysis.

We need to remember that the target are worker wanting to buy healthy fast food for breakfast and lunch, we don't aim the evening. Also, another target can be universities.

[16]:
```
# 'Root' category for all universities venues
univ_category = ['4d4b7105d754a06372d81259']

def get_meta_venues(lats, lons, meta_category):
    from tqdm.autonotebook import tqdm
    tqdm.pandas()
    meta_venues = {}
    neighborhoods_venues = []

#     print('Obtaining venues around candidate locations:', end='')
    pbar = tqdm(total=len(lats), desc = 'Obtaining venues', unit= ' coord')
    for lat, lon in zip(lats, lons):
        # Using radius=350 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)
        area_meta_venues = []
        for i, category in enumerate(meta_category):
            venues = get_venues_near_location(lat, lon, category,␣
 ↪foursquare_client_id,
                                              foursquare_client_secret,␣
 ↪radius=350, limit=100)
            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]

                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_asian, x, y)
                if venue_distance<=100:
                    area_meta_venues.append(restaurant)
                meta_venues[venue_id] = restaurant

            neighborhoods_venues.append(area_meta_venues)
        pbar.update(1)
    pbar.close()
#     print(' done.')
    return meta_venues, neighborhoods_venues



# plantage = plantage # Just to make sure we won't go after this point

# Try to load from local file system in case we did this before
meta_univ = {}
neighborhoods_univ = []
loaded = False
try:
    with open('meta_univ_350.pkl', 'rb') as f:
        meta_univ = pickle.load(f)
    with open('neighborhoods_univ_350.pkl', 'rb') as f:
        neighborhoods_univ = pickle.load(f)
    print('Universities data loaded.')
    loaded = True
except:
    pass

if not loaded:
    meta_univ, neighborhoods_univ = get_meta_venues(latitudes, longitudes,␣
↪univ_category)

    # Let's persists this in local file system
    with open('meta_univ_350.pkl', 'wb') as f:
        pickle.dump(meta_univ, f)
    with open('neighborhoods_univ_350.pkl', 'wb') as f:
```

```
        pickle.dump(neighborhoods_univ, f)

print('Total number of universities:', len(meta_univ))
print('Average number of universities in neighborhood:', np.array([len(r) for r␣
 ↪in neighborhoods_univ]).mean())

map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
for res in meta_univ.values():
    lat = res[2]; lon = res[3]
    is_univ = res[6]
    color = 'blue'
    label = '{}'.format(res[1])
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
 ↪fill_color=color,
                            popup=label, fill_opacity=1, parse_html=False).
 ↪add_to(map_seattle)
map_seattle
```

```
Universities data loaded.
Total number of universities: 67
Average number of universities in neighborhood: 0.41818181818181815
```

[16]: <folium.folium.Map at 0x7fe19ea22b00>

Companies visualization

[17]:
```
# 'Root' category for all companies venues
companies_category = ['4d4b7105d754a06375d81259', '4d4b7105d754a06378d81259',␣
 ↪'4d4b7105d754a06379d81259',
                      '4d4b7104d754a06370d81259']

# Try to load from local file system in case we did this before
meta_company = {}
neighborhoods_company = []
loaded = False
try:
    with open('meta_company_350.pkl', 'rb') as f:
        meta_company = pickle.load(f)
    with open('neighborhoods_company_350.pkl', 'rb') as f:
        neighborhoods_company = pickle.load(f)
    print('Companies data loaded.')
    loaded = True
except:
    pass

if not loaded:
```

```
    meta_company, neighborhoods_company = get_meta_venues(latitudes,␣
↪longitudes, companies_category)

    # Let's persists this in local file system
    with open('meta_company_350.pkl', 'wb') as f:
        pickle.dump(meta_company, f)
    with open('neighborhoods_company_350.pkl', 'wb') as f:
        pickle.dump(neighborhoods_company, f)

print('Total number of companies:', len(meta_company))
print('Average number of companies in neighborhood:', np.array([len(r) for r in␣
↪neighborhoods_company]).mean())

map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
for res in meta_company.values():
    lat = res[2]; lon = res[3]
    is_company = res[6]
    color = 'red'
    label = '{}'.format(res[1])
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
↪fill_color=color,
                        popup=label, fill_opacity=1, parse_html=False).
↪add_to(map_seattle)
map_seattle
```

```
Companies data loaded.
Total number of companies: 2752
Average number of companies in neighborhood: 16.318181818181817
```

[17]: `<folium.folium.Map at 0x7fe19edca630>`

Global map of worker/students versus Restaurants

[18]:
```
map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
for res in meta_company.values():
    lat = res[2]; lon = res[3]
    color = 'green'
    label = '{}'.format(res[1])
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
↪fill_color=color,
                        popup=label, fill_opacity=1, parse_html=False).
↪add_to(map_seattle)
for res in meta_univ.values():
    lat = res[2]; lon = res[3]
```

```
    color = 'yellow'
    label = '{}'.format(res[1])
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
↪fill_color=color,
                        popup=label, fill_opacity=1, parse_html=False).
↪add_to(map_seattle)

for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_asian = res[6]
    color = 'red' if is_asian else 'blue'
    label = '{}'.format(res[1])
    label = folium.Popup(label, parse_html=True)
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
↪fill_color=color,
                        popup=label, fill_opacity=1, parse_html=False).
↪add_to(map_seattle)
map_seattle
```

[18]: `<folium.folium.Map at 0x7fe19ede64a8>`

green:company yellow:university red:asian restaurant bllue:non-asian restaurant
Creation and visualization of customer group:

[19]:
```
# 'Root' category for all companies venues
customer_category = ['4d4b7105d754a06375d81259', '4d4b7105d754a06378d81259',␣
↪'4d4b7105d754a06379d81259',
                     '4d4b7104d754a06370d81259', '4d4b7105d754a06372d81259']

# Try to load from local file system in case we did this before
meta_customers = []
neighborhoods_customers = []
loaded = False
try:
    with open('meta_customers_350.pkl', 'rb') as f:
        meta_customers = pickle.load(f)
    with open('neighborhoods_customers_350.pkl', 'rb') as f:
        neighborhoods_customers = pickle.load(f)
    print('Companies data loaded.')
    loaded = True
except:
    pass

if not loaded:
    from tqdm.autonotebook import tqdm
    pbar1 = tqdm(total=len(customer_category), desc = 'Cycling categories',␣
↪unit= ' categories')
```

```
    for category in customer_category:
        meta_customer, neighborhoods_customer = get_meta_venues(latitudes,␣
↪longitudes, [category])
        meta_customers.append(meta_customer)
        neighborhoods_customers.append(neighborhoods_customer)
        pbar1.update(1)
    pbar1.close()

    # Let's persists this in local file system
    with open('meta_customers_350.pkl', 'wb') as f:
        pickle.dump(meta_customers, f)
    with open('neighborhoods_customers_350.pkl', 'wb') as f:
        pickle.dump(neighborhoods_customers, f)

print('Total number of customers:', len(meta_customers))
print('Average number of customers in neighborhood:', np.array([len(r) for r in␣
↪neighborhoods_customers]).mean())

map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
for meta_customer in meta_customers:
    for res in meta_customer.values():
        lat = res[2]; lon = res[3]
        color = 'red'
        label = '{}'.format(res[1])
        label = folium.Popup(label, parse_html=True)
        folium.CircleMarker([lat, lon], radius=3, color=color, fill=True,␣
↪fill_color=color,
                            popup=label, fill_opacity=1, parse_html=False).
↪add_to(map_seattle)
map_seattle
```

```
Companies data loaded.
Total number of customers: 5
Average number of customers in neighborhood: 110.0
```

[19]:
```
<folium.folium.Map at 0x7fe1a5394cf8>
```

4. 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 business in every area candidate:

[20]:
```
counts = []
for i, neighborhoods_customer in enumerate(neighborhoods_customers):
    counts.append([len(res) for res in neighborhoods_customers[i]])
final_count= []
len(counts[0])
```

```
for m in range(len(counts[0])):
    final_count.append(0)
    for n in range(len(counts)):
        final_count[m] = final_count[m] + counts[n][m]
len(final_count)
```

[20]: 110

[21]:
```
# location_customers_count = [len(res) for res in neighborhoods_customers]
counts = []
for i, neighborhoods_customer in enumerate(neighborhoods_customers):
    counts.append([len(res) for res in neighborhoods_customers[i]])
location_customers_count= []
len(counts[0])
for m in range(len(counts[0])):
    location_customers_count.append(0)
    for n in range(len(counts)):
        location_customers_count[m] = location_customers_count[m] + counts[n][m]

df_locations['Customers in area'] = location_customers_count

print('Average number of customers in every area with radius=100m:', np.
 ↪array(location_customers_count).mean())

df_locations.head(5)
```

```
Average number of customers in every area with radius=100m: 16.490909090909092
```

[21]:
```
                                              Address   Latitude   Longitude  \
0  217, 12th Avenue, Central Business District, Y...  47.604092 -122.316915
1  300, 10th Avenue, Central Business District, Y...  47.604993 -122.318870
2  412, Broadway, Central Business District, Yesl...  47.605895 -122.320825
3  Swedish First Hill Medical Center, 747, Broadw...  47.606796 -122.322781
4  O'Dea High School, 802, Terry Avenue, Central ...  47.607697 -122.324736

              X             Y  Distance from center  Customers in area
0 -2.653123e+06  1.377251e+07           1111.720843                  5
1 -2.652923e+06  1.377251e+07            957.038784                  2
2 -2.652723e+06  1.377251e+07            822.145506                 23
3 -2.652523e+06  1.377251e+07            718.277964                  9
4 -2.652323e+06  1.377251e+07            660.244828                  9
```

top 10 area

[22]:
```
df_locations.sort_values(by='Customers in area', ascending=False).head(10)
```

[22]:
```
                                               Address   Latitude   Longitude  \
36  Daniels Recital Hall, 801, 5th Avenue, Central...  47.605613 -122.331132
75  Grand Central, 107, Occidental Avenue South, W...  47.600130 -122.333791
6   Virginia Mason Hospital & Seattle Medical Cent...  47.609500 -122.328648
```

```
65  The Halal Guys, 105, Yesler Way, West Edge, In...  47.601726 -122.333616
16  782, Madison Street, Central Business District...  47.607904 -122.328824
45  701, 4th Avenue, Central Business District, Fi...  47.604017 -122.331308
22  Harborview Medical Center, 325, 9th Avenue, Ce...  47.603604 -122.323133
37  Central Library, 1000, 4th Avenue, Central Bus...  47.606514 -122.333088
19  U.S. Bank Centre, 1420, 5th Avenue, Central Bu...  47.610607 -122.334692
9   Hotel Theodore, 1531, 7th Avenue, Central Busi...  47.612203 -122.334516

               X             Y  Distance from center  Customers in area
36 -2.652023e+06  1.377303e+07            240.192379                 62
75 -2.652223e+06  1.377372e+07            559.807621                 56
6  -2.651923e+06  1.377251e+07            718.277964                 56
65 -2.652123e+06  1.377355e+07            399.326338                 53
16 -2.652023e+06  1.377268e+07            519.467306                 52
45 -2.652123e+06  1.377320e+07            107.774892                 51
22 -2.652723e+06  1.377285e+07            586.318455                 46
37 -2.651823e+06  1.377303e+07            421.535739                 43
19 -2.651423e+06  1.377268e+07            932.655500                 42
9  -2.651323e+06  1.377251e+07           1111.720843                 41
```

[23]:
```
restaurant_latlons = [[res[2], res[3]] for res in restaurants.values()]

customers_latlons = []
for meta_customer in meta_customers:
    customers_latlons.append([[res[2], res[3]] for res in meta_customer.
 ↪values()])
```

[24]:
```
from folium import plugins
from folium.plugins import HeatMap

map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.TileLayer('cartodbpositron').add_to(map_seattle) #cartodbpositron␣
 ↪cartodbdark_matter
for customers_latlon in customers_latlons:
    HeatMap(customers_latlon).add_to(map_seattle)
folium.Circle(Seattle_center, radius=100, fill=False, color='white').
 ↪add_to(map_seattle)
folium.Circle(Seattle_center, radius=300, fill=False, color='white').
 ↪add_to(map_seattle)
folium.Circle(Seattle_center, radius=500, fill=False, color='white').
 ↪add_to(map_seattle)
map_seattle
```

[24]:
```
<folium.folium.Map at 0x7fe19e578a58>
```

We can clearly identify two big cluster, on northwest, on southwest.
Let's create another heatmap map showing heatmap/density of restaurants.

[25]:
```
map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.TileLayer('cartodbpositron').add_to(map_seattle) #cartodbpositron␣
↪cartodbdark_matter
HeatMap(restaurant_latlons).add_to(map_seattle)
folium.Marker(Seattle_center, popup='Prefecture').add_to(map_seattle)
folium.Marker(Seattle_center, popup='Center',
    icon=folium.Icon(color='red', icon='info-sign')).add_to(map_seattle)
folium.Circle(Seattle_center, radius=100, fill=False, color='white').
↪add_to(map_seattle)
folium.Circle(Seattle_center, radius=300, fill=False, color='white').
↪add_to(map_seattle)
folium.Circle(Seattle_center, radius=500, fill=False, color='white').
↪add_to(map_seattle)
map_seattle
```

[25]: <folium.folium.Map at 0x7fe1888cfcf8>

As expected, the restaurants are also in this area, but if we look at the first map, we can see that we almost have no competitors in the central area.

K mean clustering

Let's do the kmean clustering to see what will be the result.

[26]:
```
# We plot the area where we'll search for good localisation
roi_x_min = Seattle_center_x -1000
roi_y_max = Seattle_center_y
roi_width = 1500
roi_height = 1500
roi_center_x = roi_x_min
roi_center_y = roi_y_max
roi_center_lon, roi_center_lat = xy_to_lonlat(roi_center_x, roi_center_y)
roi_center = [roi_center_lat, roi_center_lon]

map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
for customers_latlon in customers_latlons:
    HeatMap(customers_latlon).add_to(map_seattle)
folium.Marker(Seattle_center).add_to(map_seattle)
folium.Circle(Seattle_center, radius=700, color='white', fill=True,␣
↪fill_opacity=0.4).add_to(map_seattle)
map_seattle
```

[26]: <folium.folium.Map at 0x7fe19c20f630>

[27]:
```
k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
nb_k = 20 #51 a la base
x_step = 100
y_step = 100 * k
roi_y_min = roi_center_y - 700

roi_latitudes = []
```

```
roi_longitudes = []
roi_xs = []
roi_ys = []
for i in range(0, int(nb_k/k)):
    y = roi_y_min + i * y_step
    x_offset = (nb_k-1) 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 <= 2501):
            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.')
```

```
563 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 150 meters) and number of customers.

[29]:
```
def count_restaurants_nearby(x, y, restaurants, radius=150):
    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


def count_customers_nearby(x, y, customers, radius=150):
    count = 0
    for meta_customer in meta_customers:
        for res in meta_customer.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

roi_restaurant_counts = []
roi_asian_restaurants = []
roi_customer = []

print('Generating data on location candidates... ', end='')
for x, y in zip(roi_xs, roi_ys):
    count = count_restaurants_nearby(x, y, restaurants, radius=100)
    roi_restaurant_counts.append(count)

    distance = find_nearest_restaurant(x, y, asian_restaurants)
    roi_asian_restaurants.append(distance)

    custom = count_customers_nearby(x, y, meta_customers)
    roi_customer.append(custom)
print('done.')
```

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

[30]:
```
# 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 Asian restaurant':
 ↪roi_asian_restaurants,
                                 'Customers':roi_customer})

df_roi_locations.sort_values(by='Customers', ascending=False).head(10)
```

[30]:
```
      Latitude   Longitude             X             Y  Restaurants nearby  \
186  47.605353 -122.331393 -2.652023e+06  1.377307e+07                   0
311  47.602575 -122.334464 -2.652004e+06  1.377350e+07                   3
185  47.604903 -122.330415 -2.652123e+06  1.377307e+07                   5
335  47.601466 -122.333877 -2.652123e+06  1.377359e+07                  14
384  47.599870 -122.334053 -2.652223e+06  1.377376e+07                   6
310  47.602125 -122.333486 -2.652104e+06  1.377350e+07                   8
210  47.604416 -122.331178 -2.652104e+06  1.377315e+07                   4
408  47.598933 -122.333837 -2.652304e+06  1.377385e+07                   4
289  47.604414 -122.336635 -2.651723e+06  1.377341e+07                   3
211  47.604866 -122.332156 -2.652004e+06  1.377315e+07                   2

     Distance to Asian restaurant  Customers
186                    113.335435        153
```

```
311                    91.689117        143
185                    30.687867        125
335                    14.976102        122
384                    72.177583        122
310                    39.721710        120
210                    26.981858        119
408                   176.485481        118
289                   104.479281        117
211                    56.481524        116
```

OK. Let us now filter those locations: we're interested only in locations with no more than two restaurants in radius of 250 meters, and no asian restaurants in radius of 100 meters, and more than 10 customers.

[32]:
```
good_res_count = np.array((df_roi_locations['Restaurants nearby']<=2))
print('Locations with no more than two restaurants nearby:', good_res_count.
 ↪sum())

good_asi_distance = np.array(df_roi_locations['Distance to Asian␣
 ↪restaurant']>=100)
print('Locations with no Asian restaurants within 400m:', good_asi_distance.
 ↪sum())

good_custmer_count = np.array(df_roi_locations['Customers']>=10)
print('Locations with more than 10 customers:', good_custmer_count.sum())

good_locations = np.logical_and(good_custmer_count, good_res_count,␣
 ↪good_asi_distance)
print('Locations with both conditions met:', good_locations.sum())

df_good_locations = df_roi_locations[good_locations]
```

```
Locations with no more than two restaurants nearby: 430
Locations with no Asian restaurants within 400m: 362
Locations with more than 10 customers: 351
Locations with both conditions met: 220
```

Let's see how this looks on a map.

[34]:
```
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_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.TileLayer('cartodbpositron').add_to(map_seattle)
HeatMap(restaurant_latlons).add_to(map_seattle)
```

```
folium.Circle(Seattle_center, radius=700, color='white', fill=True,␣
↪fill_opacity=0.6).add_to(map_seattle)
folium.Marker(Seattle_center).add_to(map_seattle)
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_seattle)
map_seattle
```

[34]: <folium.folium.Map at 0x7fe1888cf278>

We can identify two mains areas, one south, the other one north as expected.
Let's see heatmap:

[35]:
```
map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
HeatMap(good_locations, radius=35).add_to(map_seattle)
folium.Marker(Seattle_center).add_to(map_seattle)
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_seattle)
map_seattle
```

[35]: <folium.folium.Map at 0x7fe1888cf518>

Looking good. What we have now is a clear indication of zones with low number of restaurants in vicinity, and no Asian restaurants at all nearby, and good numbers of customers.

5. Result

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.

[36]:
```
from sklearn.cluster import KMeans

number_of_clusters = 15

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_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.TileLayer('cartodbpositron').add_to(map_seattle)
for customers_latlon in customers_latlons:
    HeatMap(customers_latlon).add_to(map_seattle)
folium.Circle(Seattle_center, radius=700, color='white', fill=True,␣
↪fill_opacity=0.4).add_to(map_seattle)
folium.Marker(Seattle_center).add_to(map_seattle)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=80, color='green', fill=True,␣
↪fill_opacity=0.25).add_to(map_seattle)
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_seattle)
map_seattle
```

[36]: <folium.folium.Map at 0x7fe18a55fcc0>

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:

[37]:
```
map_seattle = folium.Map(location=Seattle_center, zoom_start=15)
folium.Marker(Seattle_center).add_to(map_seattle)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#00000000', fill=True,␣
 ↪fill_color='#0066ff',
                  fill_opacity=0.07).add_to(map_seattle)
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_seattle)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=80, color='green', fill=False).
 ↪add_to(map_seattle)
map_seattle
```

[37]: <folium.folium.Map at 0x7fe188d6c588>

Finaly, let's reverse geocode those candidate area centers to get the addresses

[43]:
```
candidate_area_addresses = []
print('==============================================================')
print('Addresses of centers of areas recommended for further analysis')
print('==============================================================\n')
for lon, lat in cluster_centers:
    addr = get_address(lat, lon).replace(', United States of America', '')
    candidate_area_addresses.append(addr)
    x, y = lonlat_to_xy(lon, lat)
    d = calc_xy_distance(x, y, Seattle_center_x, Seattle_center_y)
    print('{}{} => {:.1f}km from Prefecture'.format(addr, ' '*(50-len(addr)), d/
 ↪1000))
```

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


Hambach Building, South King Street, West Edge, International
District/Chinatown, Seattle, King County, Washington, 98104 => 0.8km from
Prefecture
5th and Madison Condos, 909, 5th Avenue, Central Business District, First Hill,
```

```
Seattle, King County, Washington, 98164 => 0.3km from Prefecture
1086, Jefferson Street, Central Business District, First Hill, Seattle, King
County, Washington, 98104 => 0.7km from Prefecture
Third and Stewart Garage, 3rd Avenue, Central Business District, Belltown,
Seattle, King County, Washington, 98181 => 1.2km from Prefecture
Seattle King Street, 301, South Jackson Street, West Edge, International
District/Chinatown, Seattle, King County, Washington, 98104 => 0.7km from
Prefecture
741, James Street, Central Business District, First Hill, Seattle, King County,
Washington, 98104 => 0.3km from Prefecture
M Street Medical Building, 910, 8th Avenue, Central Business District, First
Hill, Seattle, King County, Washington, 98104 => 0.5km from Prefecture
61, Columbia Street, West Edge, International District/Chinatown, Seattle, King
County, Washington, 98104 => 0.6km from Prefecture
398, Dilling Way, West Edge, International District/Chinatown, Seattle, King
County, Washington, 98104 => 0.2km from Prefecture
Two Union Square, 601, Union Street, Central Business District, First Hill,
Seattle, King County, Washington, 98101 => 0.8km from Prefecture
House of Hong, 409, 8th Avenue South, West Edge, International
District/Chinatown, Seattle, King County, Washington, 98104 => 0.9km from
Prefecture
Benaroya Hall, 200, University Street, West Edge, Belltown, Seattle, King
County, Washington, 98101 => 0.7km from Prefecture
814, 6th Avenue South, West Edge, International District/Chinatown, Seattle,
King County, Washington, 98134 => 1.1km from Prefecture
Coastal Environmental Systems, 820, 1st Avenue South, West Edge, International
District/Chinatown, Seattle, King County, Washington, 98134 => 1.1km from
Prefecture
Waterfront Park, 1301, Alaskan Way, Pike Place Market Area, Belltown, Seattle,
King County, Washington, 98101 => 1.0km from Prefecture
```

6. Discussion

From the above result, we can know the different address of our choice of the best location of restaurant. Seattle is a large city, so the number and density of restaurant is quite high, in this analysis I tried to corrolate the number of restaurant and quantity of potential customer.

the Prefecture area is distributed near the center of Seattle.

We must just take care of one thing, I thing the api didn't return all data, we are missing a lot of companies, the map is still good, and the result can be trusted, but we should cross check data with other data source.

In order to be more accurate, it could be possible to give a weight to customers for example, a university with 1000 student would then weight more than a hair cut company with two employees.

7.Conclusion

The idea of this project came from the homework requirement and one data case from kaggle. I applied my own data source and methdology to it.Also, I have created/modify a huge quantity of function in order to adapt.

It's very far from being perfect, a lot of work can be done, other source of data can be found, but in the end the result seams to correlate with the real world, when we know the city, the area predicted seams correct.

[ ]:
```

```