# Capstone Project - The Battle of the Neighborhoods (Week 3-4)

### Applied Data Science Capstone by IBM/Coursera

## Introduction: Business Problem

In this project we will try to find an optimal location for a central kitchen. Specifically, this report will be targeted to stakeholders interested in providing ingredients and pre cooked meals to **Italian restaurant** in **Singapore**.

we will try to idenitfy the **locations that have highest number of italian restaurants**. We are also particularly interested in **areas with few traffic for delivery**. We would also prefer locations **as close to city center as possible**, assuming that first two conditions are met.

Advantages of each area will then be clearly expressed so that best possible final location can be chosen by stakeholders.

## Data

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

- number of Italian restaurants across the city grid
- Clustering of the restaurants locations
- travel time between each centroids

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 the city and travel time (traffic index) between centroids 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**

### 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. 6x6 killometers centered around Singapore city center.

Now let's create a grid of area candidates, equaly spaced, centered around city center. Our neighborhoods will be defined as circular areas with a radius of 800 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 [12]:

```
import requests

def get_coordinates(api_key, address, verbose=False):
    try:
        url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&address={}'.
        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 co
        lat = geographical_data['lat']
        lon = geographical_data['lng']
        return [lat, lon]
    except:
        return [None, None]

address = 'Center city square, Singapore, Singapore'
singapore_center = get_coordinates(google_api_key, address)
print('Coordinate of {}: {}'.format(address, singapore_center))
```

```
Coordinate of Center city square, Singapore, Singapore: [1.3111877, 10
3.8566675]
```

In [235]:

```
#!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='48N', 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='48N', 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('singapore_center longitude={}, latitude={}'.format(singapore_center[1], singa
x, y = lonlat_to_xy(singapore_center[1], singapore_center[0])
print('singapore_center UTM X={}, Y={}'.format(x, y))
lo, la = xy_to_lonlat(x, y)
print('singapore_center longitude={}, latitude={}'.format(lo, la))
```

```
Coordinate transformation check
-------------------------------
singapore_center longitude=103.8566675, latitude=1.3111877
singapore_center UTM X=372800.3177483371, Y=144954.93480977515
singapore_center longitude=103.8566675, latitude=1.3111877
```

In [136]:

```
# create the grid for the foursquare api call
import numpy as np
from scipy.interpolate import Rbf
# Build a regular grid with 500-metre cells.
extent = x_min, x_max, y_min, y_max = [x-11000, x+11000,
                                       y-6000, y+8000]
grid_x, grid_y = np.mgrid[x_min:x_max:800, y_min:y_max:800]
```

In [148]:

```
li=[]
li_X=[]
li_Y=[]
li_lat=[]
li_lon=[]
li_dist=[]
for j,k in zip(grid_x,grid_y):
    for xv,yv in zip(j,k):
        li_X.append(xv)
        li_Y.append(yv)
        dist=calc_xy_distance(x, y, xv, yv)
        li_dist.append(dist)
        lo, la = xy_to_lonlat(xv, yv)
        li_lon.append(lo)
        li_lat.append(la)
        li.append((la,lo))
```

In [138]:

```
#number of tiles in the grid
len(li)
```

Out[138]:

```
504
```

In [ ]:

```

```

In [139]:

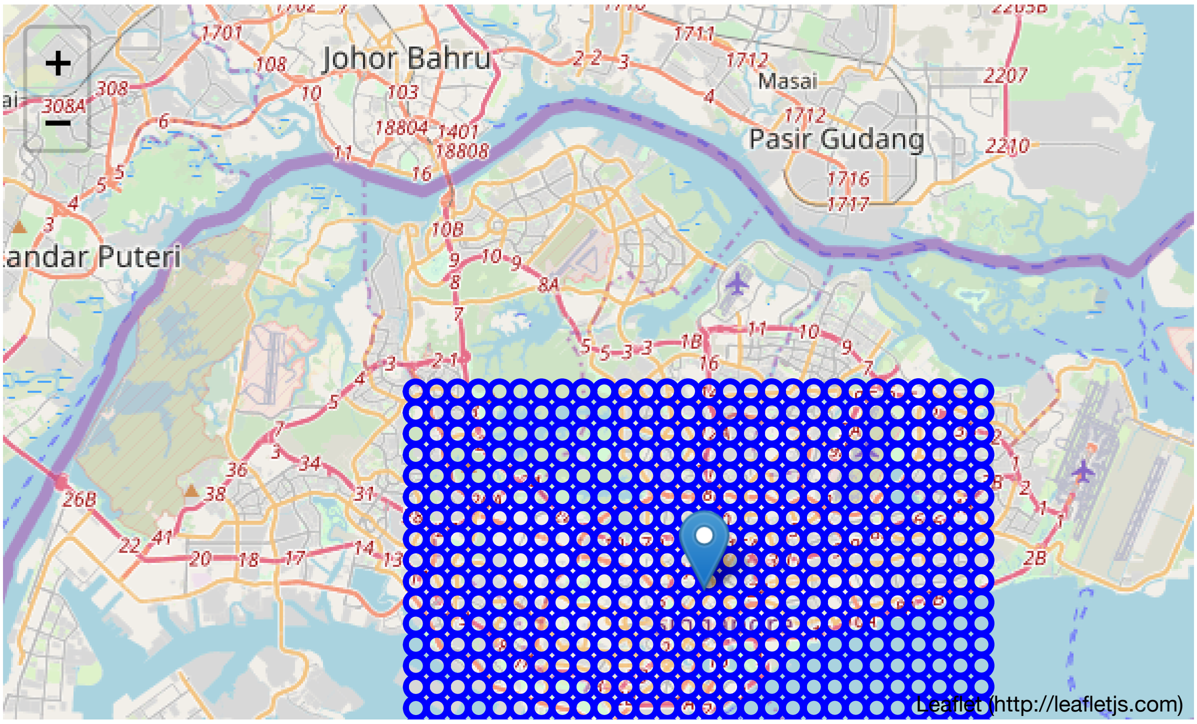
```
#!pip install folium

import folium
```

In [140]:

```
p_singapore = folium.Map(location=singapore_center, zoom_start=13)
lium.Marker(singapore_center, popup='center').add_to(map_singapore)
r lat, lon in li:
  #folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='bl
  folium.Circle([lat, lon], radius=400, color='blue', fill=False).add_to(map_singapo
  #folium.Marker([lat, lon]).add_to(map_berlin)
p_singapore
```

Out[140]:




# centers are defined

# for each lat/lon, count the number of italian restaurants

OK, we now have the coordinates of centers of neighborhoods/areas to be evaluated, equally spaced (distance from every point to its neighbors is exactly the same)

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

In [141]:

```
def get_address(api_key, latitude, longitude, verbose=False):
    try:
        url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&latlng={},{}
        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, singapore_center[0], singapore_center[1])
print('Reverse geocoding check')
print('-----------------------')
print('Address of [{}, {}] is: {}'.format(singapore_center[0], singapore_center[1],
```

```
Reverse geocoding check
-----------------------
Address of [1.3111877, 103.8566675] is: 180 Kitchener Rd, Singapore 20
8539
```

In [142]:

```
print('Obtaining location addresses: ', end='')

addresses = []
for lat, lon in li:
    address = get_address(google_api_key, lat, lon)
    if address is None:
        address = 'NO ADDRESS'
    address = address.replace(', Singapore', '')
    addresses.append(address)
    print(' .', end='')
print(' done.')
```

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

## lets check the result of the adresses call

In [143]:

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

Out[143]:

```
['5D Ridley Park 248480',
 'Tanglin, Swan Lake',
 'Heliconia Walk',
 'Botanic Gardens MRT Station, Trellis Garden',
 '15C Camden Park 299809',
 '6 Adam Dr 289966',
 'Unnamed Road',
 'Central Water Catchment',
 'Island Club Rd',
 'Island Club Rd',
 'Central Water Catchment',
 'Old Upper Thomson Rd',
 '8 Sentosa Gateway 098269',
 '1 Harbour Front Walk 098585',
 '41 Telok Blangah Rise, Block 41 090041',
 '18 Telok Blangah Cres, Block 18 090018',
 '114 Bukit Merah View, Block 114 150114',
 '40B Jervois Rd 249039',
 '10A Chatsworth Rd 249792',
 '24 Nassim Hill 258469']
```

In [149]:

```
import pandas as pd

df_locations = pd.DataFrame({'Address': addresses,
                             'Latitude': li_lat,
                             'Longitude': li_lon,
                             'X': li_X,
                             'Y': li_Y,
                             'Distance from center': li_dist})

df_locations.head(10)
```

Out[149]:

| | Address | Latitude | Longitude | X | Y | Distance from center |
|---|---|---|---|---|---|---|
| 0 | Pasir Panjang Drive 5 | 1.256870 | 103.757835 | 361800.317748 | 138954.93481 | 12529.964086 |
| 1 | Pasir Panjang Drive 5 | 1.264106 | 103.757832 | 361800.317748 | 139754.93481 | 12167.168939 |
| 2 | Pasir Panjang Drive 5 | 1.271342 | 103.757828 | 361800.317748 | 140554.93481 | 11847.362576 |
| 3 | Pasir Panjang Drive 5 | 1.278578 | 103.757825 | 361800.317748 | 141354.93481 | 11574.109037 |
| 4 | Pasir Panjang Drive 3 | 1.285814 | 103.757822 | 361800.317748 | 142154.93481 | 11350.770899 |
| 5 | Singapore | 1.293050 | 103.757818 | 361800.317748 | 142954.93481 | 11180.339887 |
| 6 | 9 Pandan Cres 128465 | 1.300286 | 103.757814 | 361800.317748 | 143754.93481 | 11065.260955 |
| 7 | 25 Pandan Cres 128477 | 1.307522 | 103.757811 | 361800.317748 | 144554.93481 | 11007.270325 |
| 8 | 41 W Coast Pl 127593 | 1.314758 | 103.757807 | 361800.317748 | 145354.93481 | 11007.270325 |
| 9 | 12 Faber Heights 129163 | 1.321994 | 103.757804 | 361800.317748 | 146154.93481 | 11065.260955 |

In [428]:

```
df_locations.describe()
```

Out[428]:

| | Latitude | Longitude | X | Y | Distance from center | Restaurants in area |
|---|---|---|---|---|---|---|
| count | 504.000000 | 504.000000 | 504.000000 | 504.000000 | 504.000000 | 504.000000 |
| mean | 1.318423 | 103.854867 | 372600.317748 | 145754.934810 | 7172.352910 | 5.212302 |
| std | 0.037580 | 0.058135 | 6468.618235 | 4154.625677 | 2870.604525 | 9.225072 |
| min | 1.256870 | 103.757774 | 361800.317748 | 138954.934810 | 447.213595 | 0.000000 |
| 25% | 1.285860 | 103.806320 | 367200.317748 | 142154.934810 | 5095.022648 | 0.000000 |
| 50% | 1.318421 | 103.854866 | 372600.317748 | 145754.934810 | 7246.992704 | 2.000000 |
| 75% | 1.350986 | 103.903411 | 378000.317748 | 149354.934810 | 9438.194549 | 6.250000 |
| max | 1.379977 | 103.951956 | 383400.317748 | 152554.934810 | 13370.115931 | 69.000000 |

In [150]:

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

In [170]:

```
latitudes=df_locations.Latitude
longitudes = df_locations.Longitude
```

## 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. 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 'Italian restaurant' category, as we need info on Italian restaurants in the neighborhood.

In [156]:

```

```

In [157]:

```
# Category IDs corresponding to Italian restaurants were taken from Foursquare web s

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

italian_restaurant_categories = ['4bf58dd8d48988d110941735','55a5a1ebe4b013909087cbb
                                 '55a5a1ebe4b013909087cba7','55a5a1ebe4b013909087cba
                                 '55a5a1ebe4b013909087cb95','55a5a1ebe4b013909087cb8
                                 '55a5a1ebe4b013909087cb98','55a5a1ebe4b013909087cbb
                                 '55a5a1ebe4b013909087cbb0','55a5a1ebe4b013909087cbb
                                 '55a5a1ebe4b013909087cbaa','55a5a1ebe4b013909087cb8
                                 '55a5a1ebe4b013909087cb92','55a5a1ebe4b013909087cb8
                                 '55a5a1ebe4b013909087cbb9','55a5a1ebe4b013909087cb7
                                 '55a5a1ebe4b013909087cb9e','55a5a1ebe4b013909087cbc

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(', Singapore', '')
    address = address.replace(', Singapore', '')
    return address

def get_venues_near_location(lat, lon, category, client_id, client_secret, radius=50
    version = '20180724'
    url = 'https://api.foursquare.com/v2/venues/explore?client_id={}&client_secret=
        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 [430]:

```
# Let's now go over our neighborhood locations and get nearby restaurants; we'll als

import pickle

def get_restaurants(lats, lons):
    restaurants = {}
    italian_restaurants = {}
    location_restaurants = []
    ratio_restaurants = []
    print('Obtaining venues around candidate locations:', end='')
    for lat, lon in zip(lats, lons):
        # Using radius=350 to meke sure we have overlaps/full coverage so we don't r
        venues = get_venues_near_location(lat, lon, food_category, foursquare_client
        area_restaurants = []
        nbresto=0
        nbrestoital=0
        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_italian = is_restaurant(venue_categories, specific_filter=ita
            if is_res:
                nbresto+=1
                x, y = lonlat_to_xy(venue_latlon[1], venue_latlon[0])
                restaurant = (venue_id, venue_name, venue_latlon[0], venue_latlon[1
                if venue_distance<=400:
                    area_restaurants.append(restaurant)
                restaurants[venue_id] = restaurant
                if is_italian:
                    nbrestoital+=1
                    italian_restaurants[venue_id] = restaurant
        location_restaurants.append(area_restaurants)
        ratio_restaurants.append([lat,lon,nbresto,nbrestoital])
        print(' .', end='')
    print(' done.')
    return restaurants, italian_restaurants, location_restaurants, ratio_restaurants

# Try to load from local file system in case we did this before
restaurants = {}
italian_restaurants = {}
location_restaurants = []
loaded = False
try:
    with open('restaurants_350.pkl', 'rb') as f:
        restaurants = pickle.load(f)
    with open('italian_restaurants_350.pkl', 'rb') as f:
        italian_restaurants = pickle.load(f)
    with open('location_restaurants_350.pkl', 'rb') as f:
        location_restaurants = pickle.load(f)
    print('Restaurant data loaded.')
    loaded = False
except:
    pass

# If load failed use the Foursquare API to get the data
if not loaded:
```

```
    restaurants, italian_restaurants, location_restaurants,ratio_restaurants = get_r

    # Let's persists this in local file system
    with open('restaurants_350.pkl', 'wb') as f:
        pickle.dump(restaurants, f)
    with open('italian_restaurants_350.pkl', 'wb') as f:
        pickle.dump(italian_restaurants, f)
    with open('location_restaurants_350.pkl', 'wb') as f:
        pickle.dump(location_restaurants, f)
```

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

In [433]:

```
with open('ratio_restaurants_350.pkl', 'wb') as f:
        pickle.dump(ratio_restaurants, f)
```

In [431]:

```
print('List of Italian restaurants')
print('---------------------------')
for r in list(italian_restaurants.values())[:2]:
    print(r)
print('...')
print('Total:', len(italian_restaurants))
```

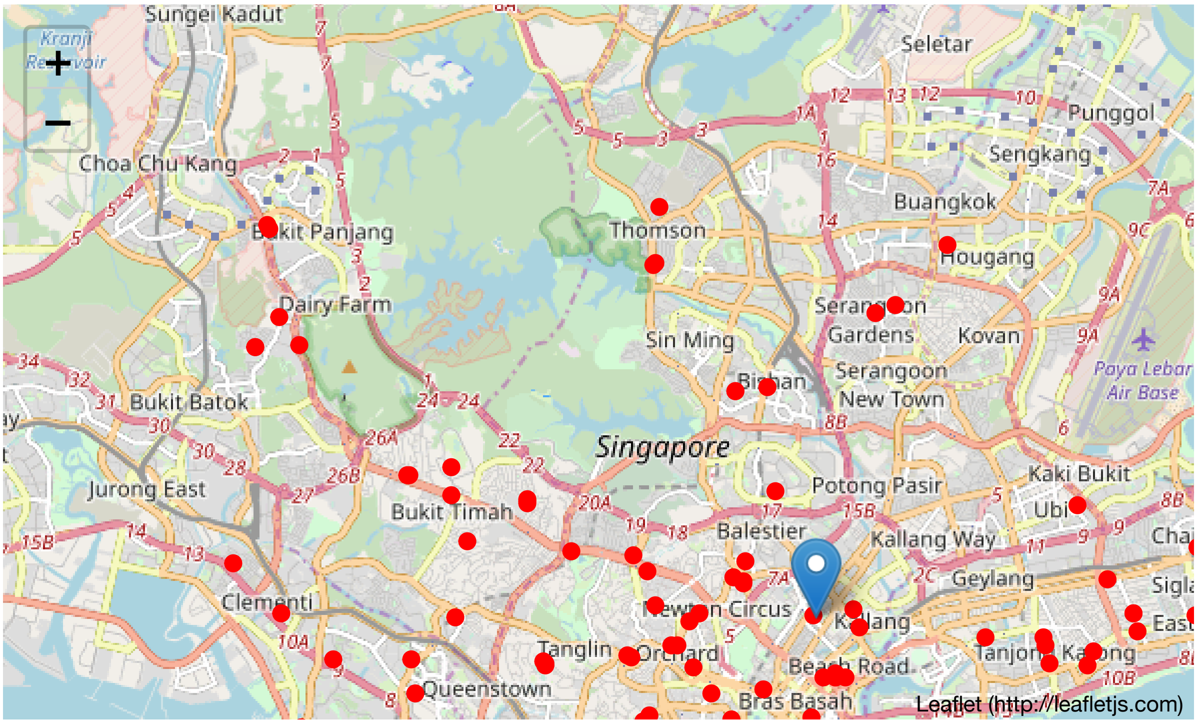
```
List of Italian restaurants
---------------------------
('4e426b68d4c0557c35b6663e', "Salvo's Restaurant & Bar", 1.32065343747
37622, 103.75663455862463, '104 Faber Dr (Faber Hills), 129412, Singap
ore', 198, True, 361670.1432846984, 146006.74216745418)
('4df70382d4c070d4df0f65a1', 'Cacio e Pepe', 1.3578013019914024, 103.7
604806031435, '3 Chu Lin Rd (off Jalan Remaja), 669599, Singapore', 30
2, True, 362100.1767231672, 150113.47256358247)
...
Total: 119
```

In [435]:

```
map_singapore = folium.Map(location=singapore_center, zoom_start=13)
folium.Marker(singapore_center, popup='center').add_to(map_singapore)
j=0
for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_italian = res[6]
    color = 'red' if is_italian else 'blue'
    if (color=='red'):
        folium.CircleMarker([lat, lon], radius=3, color=color, fill=True, fill_colo
    j+=1

map_singapore
```

Out[435]:




Looking good. So now we have all the restaurants in the area, and we know which ones are Italian restaurants.

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 Italian restaurant!

# Methodology

In this project we will direct our efforts on detecting areas of Singapore that have high italian restaurant density. 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 10km from Singapore center. We have also identified Italian restaurants (according to Foursquare categorization).

Second step in our analysis will exploration 'restaurant density' across different areas of Singapore - we will use kmeans to identify a few promising areas close with high number of italian restaurants in general.

```
We want to be able to select a few central areas which are surrounded by italian
restaurants to be intermediary kitchen to deliver precooked meals.
we will then select the most central of all to be the main supply provider.
```

## 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 italian restaurants in every area candidate**:

In [436]:

```
# Creating an empty Dataframe with column names only
dfObj = pd.DataFrame(columns=['X', 'Y'])
for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_italian = res[6]
    if is_italian==True:

        xv, yv = lonlat_to_xy(lon, lat)
        dfObj = dfObj.append({'lat': lat, 'lon':lon,'X':xv,'Y':yv }, ignore_index=Tr
```

In [437]:

```
dfObj
```

Out[437]:

| | X | Y | lat | lon |
|---|---|---|---|---|
| 0 | 361670.143285 | 146006.742167 | 1.320653 | 103.756635 |
| 1 | 362100.176723 | 150113.472564 | 1.357801 | 103.760481 |
| 2 | 362095.659130 | 150114.104833 | 1.357807 | 103.760440 |
| 3 | 362959.180230 | 142895.394638 | 1.292517 | 103.768233 |
| 4 | 362599.934627 | 145051.056789 | 1.312013 | 103.764995 |
| ... | ... | ... | ... | ... |
| 114 | 381339.981789 | 151775.368210 | 1.372917 | 103.933390 |
| 115 | 381674.738948 | 149679.098985 | 1.353956 | 103.936407 |
| 116 | 382616.137233 | 149729.124743 | 1.354413 | 103.944868 |
| 117 | 383105.793341 | 151714.992184 | 1.372378 | 103.949261 |
| 118 | 383393.369462 | 151893.548727 | 1.373995 | 103.951845 |

119 rows × 4 columns

In [438]:

```
good_xys = dfObj[['X','Y']].values
good_latitudes = dfObj['lat'].values
good_longitudes = dfObj['lon'].values
```

In [439]:

```
from sklearn.cluster import KMeans

# function returns WSS score for k values from 1 to kmax
def calculate_WSS(points, kmax):
  sse = []
  for k in range(1, kmax+1):
    kmeans = KMeans(n_clusters = k).fit(points)
    centroids = kmeans.cluster_centers_
    pred_clusters = kmeans.predict(points)
    curr_sse = 0

    # calculate square of Euclidean distance of each point from its cluster center a
    for i in range(len(points)):
      curr_center = centroids[pred_clusters[i]]
      curr_sse += (points[i, 0] - curr_center[0]) ** 2 + (points[i, 1] - curr_cente

    sse.append(curr_sse)
  return sse
```

In [440]:

```
re=calculate_WSS(good_xys, kmax=10)
```

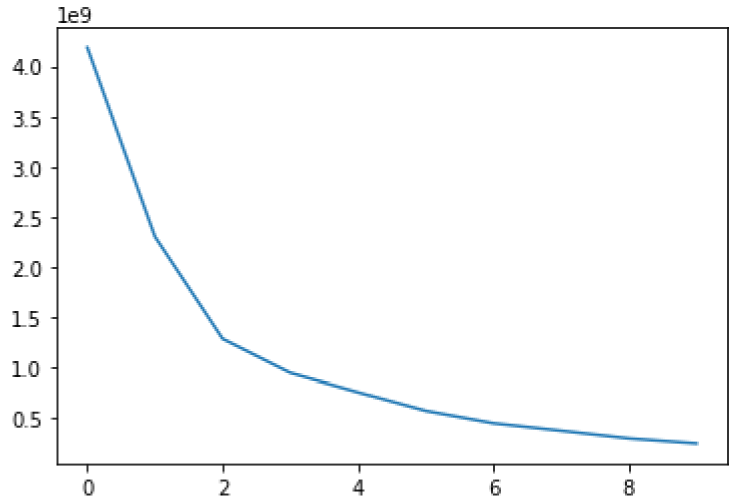
## lets define the number of spatial clusters of italian restaurants in singapore

In [332]:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(re)
```

Out[332]:

```
[<matplotlib.lines.Line2D at 0x1a285ee250>]
```




**We can see using elbow method that the best number of clusters is 5**

In [412]:

```
from sklearn.cluster import KMeans
number_of_clusters = 5
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_]
```

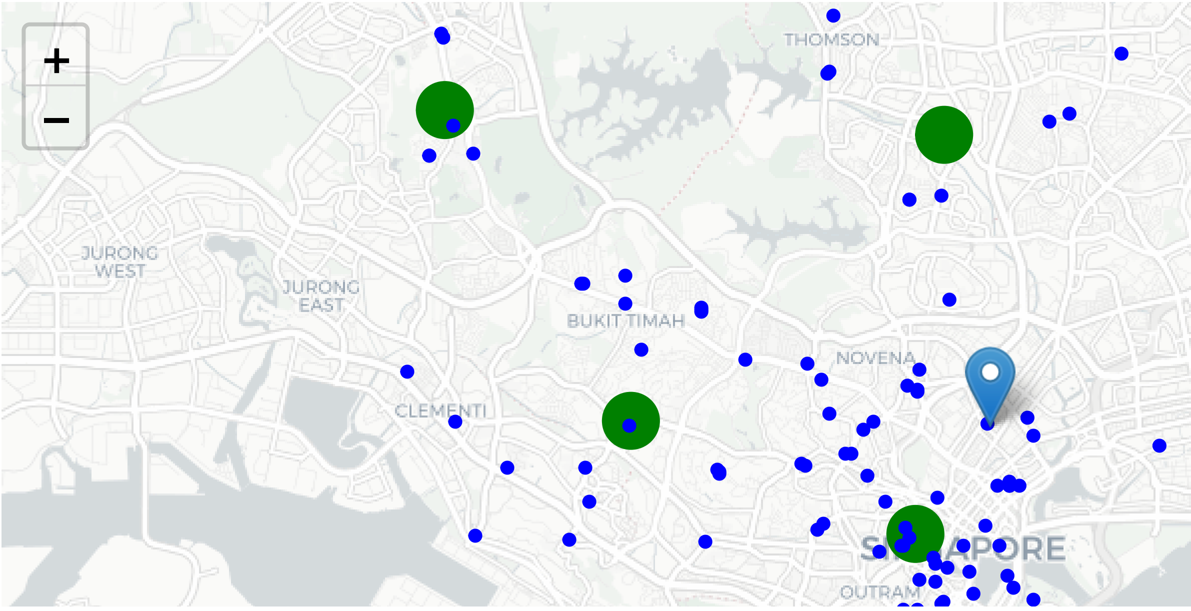
**we plot the centers of the clusters and the restaurants**

In [441]:

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

map_singapore = folium.Map(location=singapore_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_singapore)
HeatMap([]).add_to(map_singapore)
#folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opacity=0.4).
folium.Marker(singapore_center).add_to(map_singapore)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=True, fill_opacity=1).
for lat, lon in zip(good_latitudes, good_longitudes):
     folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color=
map_singapore
```

Out[441]:




**now we mesure the transport time between each cluster and will select the most central as the main supplier**

In [443]:

```
#distance between each cluster
cluster_centers
# importing googlemaps module
import googlemaps
from datetime import datetime

# Requires API key

gmaps = googlemaps.Client(key=google_api_key)

# Requires formating
list_centroids=[]
for k,v in cluster_centers:
    list_centroids.append(str(v)+','+str(k))

li_test={}
for j in range(len(list_centroids)):
    origins = list_centroids[j]
    for k in range(len(list_centroids)):
        destinations=list_centroids[k]
        my_dist = gmaps.distance_matrix(origins=origins,destinations=destinations)
        li_test[origins, destinations]=my_dist
```

**here we count the number of italian restaurants around each centroids**

In [444]:

```
# nombre de restau par centroide
# distribution des temps de parcours
res_k=kmeans.predict(good_xys)
```

In [445]:

```
time_per_cluser=[]
df_clusters = pd.DataFrame(columns=['cluster_id','avg_seconds','avg_dist','percent_r

for j in range(len(list_centroids)):
    origins = list_centroids[j]
    v=0
    w=0
    for k in range(len(list_centroids)):
        destinations=list_centroids[k]
        v+=li_test[origins,destinations]['rows'][0]['elements'][0]['duration']['valu
        w+=li_test[origins,destinations]['rows'][0]['elements'][0]['distance']['valu
    v=v/len(list_centroids)
    w=w/len(list_centroids)
    pt=sum(res_k==j)/len(res_k)*100

    df_clusters =df_clusters.append({'cluster_id':j,'avg_seconds':v,'avg_dist':w,'pe
```

In [446]:

```
df_clusters.sort_values(by=['avg_seconds','percent_restaurants'], ascending=False)
```

Out[446]:

| | cluster_id | avg_seconds | avg_dist | percent_restaurants | lon | lat |
|---|---|---|---|---|---|---|
| 2 | 2.0 | 1081.2 | 14940.6 | 17.647059 | 103.917859 | 1.324889 |
| 4 | 4.0 | 998.8 | 13238.2 | 5.042017 | 103.763051 | 1.365537 |
| 0 | 0.0 | 849.6 | 10739.2 | 8.403361 | 103.848730 | 1.361287 |
| 3 | 3.0 | 833.8 | 10922.8 | 48.739496 | 103.843801 | 1.292813 |
| 1 | 1.0 | 745.6 | 10065.0 | 20.168067 | 103.794981 | 1.312218 |

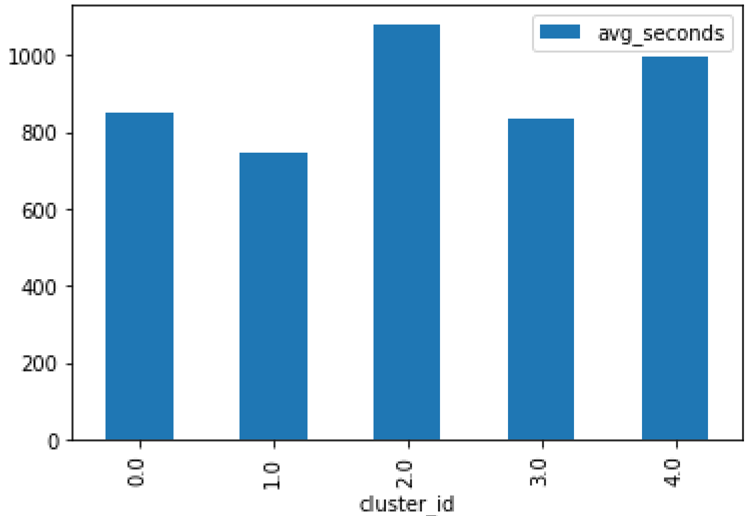
**we observe that cluster 1 and 3 are the most central in time**

In [426]:

```
df_clusters.plot.bar(x='cluster_id',y=['avg_seconds'])
```

Out[426]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a28dae2d0>



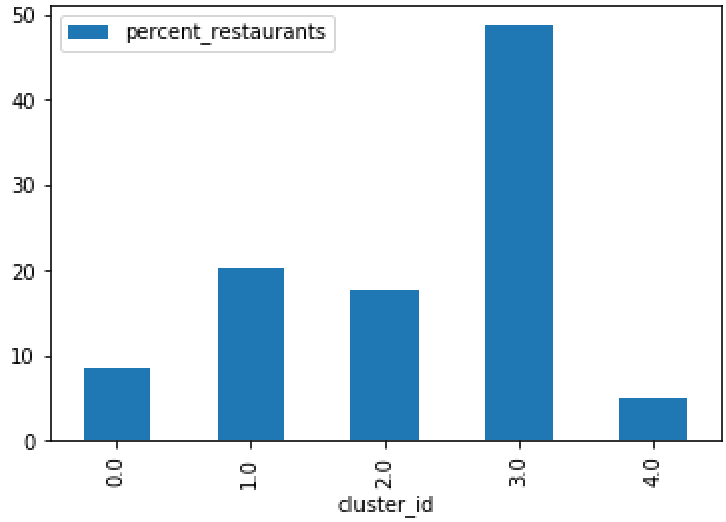

**we also observe that the cluster3 is surrounded by almost 50% of the total number of italian restaurants in singapore**

In [447]:

```
df_clusters.plot.bar(x='cluster_id',y=['percent_restaurants'])
```

Out[447]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a294e8210>
```




**we thus define the center of cluster 3 as our main supply provider, and the 4 others will serve as intermediary kitchen**

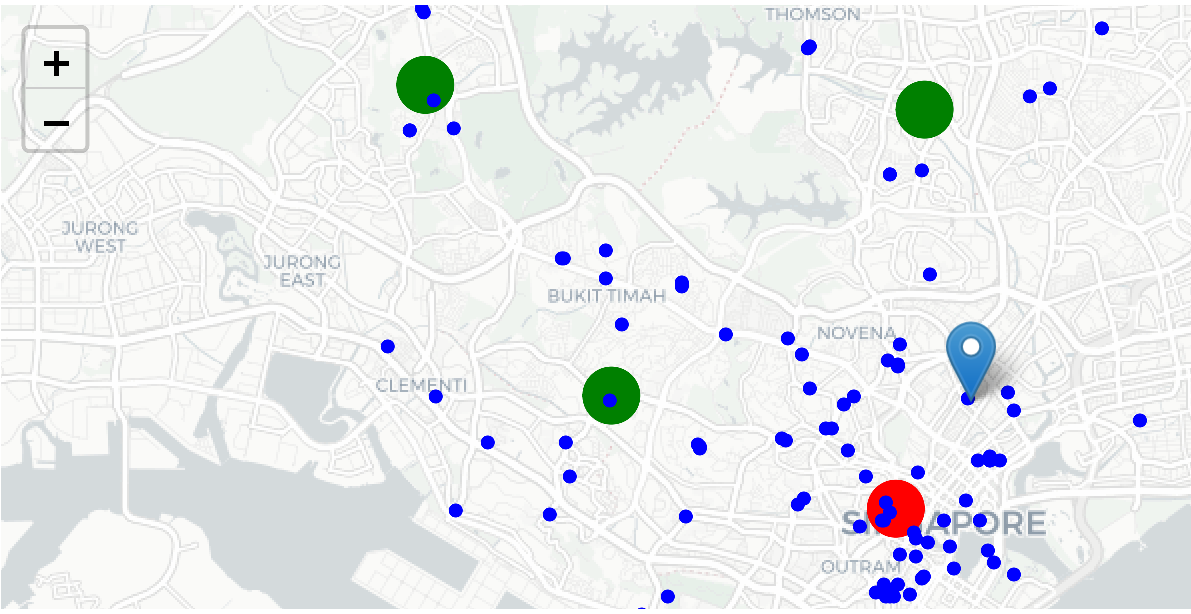
In [421]:

```
best_centroid=3
best_centroid_latlon=cluster_centers[best_centroid]
```

In [449]:

```
map_singapore = folium.Map(location=singapore_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_singapore)
HeatMap([]).add_to(map_singapore)
#folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opacity=0.4).
folium.Marker(singapore_center).add_to(map_singapore)
k=0
color_centroid='green'
for lon, lat in cluster_centers:
    if k==best_centroid:color_centroid='red'
    else:color_centroid='green'
    folium.Circle([lat, lon], radius=500, color=color_centroid, fill=True, fill_opac
    k+=1
for lat, lon in zip(good_latitudes, good_longitudes):
     folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color=
map_singapore
```

Out[449]:




**we could extend our study by exploring the ratio of italian restaurants per district**

In [451]:

```
def generateBaseMap(default_location=singapore_center, default_zoom_start=12):
    base_map = folium.Map(location=default_location, control_scale=True, zoom_start=
    return base_map
```

In [517]:

```
_ratio = pd.DataFrame(columns=['lat','lon','italian','all'])
tio_restaurants:
f j[2]==0 else j[3]/j[2]
ters_ratio =df_clusters_ratio.append({'lat':j[0],'lon':j[1],'italian':j[3],'all':j[2
```

In [514]:

```
from folium.plugins import HeatMap

df_clusters_ratio['count'] = 1
base_map = generateBaseMap()
HeatMap(data=df_clusters_ratio[['lat', 'lon', 'ratio']].groupby(['lat', 'lon']).sum(

base_map.add_child(folium.ClickForMarker(popup='Potential Location'))
```

Out[514]:

```
<folium.plugins.heat_map.HeatMap at 0x1a2853ed50>
```

# Results and Discussion

In this study we have been able to study the urban composition of singapore by collecting venues type per unit of surface. The first phase consisted at collecting the number of restaurants per urban unit and among them, how many are of italian kitchen. The second phase was about identifying the main areas where we could implement a centralized kitchen to distribute pre cooked meals and ingredients into all these italian restaurants in the optimal way. the third phase was about identifying, among all centroids, which one is the most central in terms of time and italian restaurant density, and not distance, to be the main supply for the other central kitchens.

# Conclusion

We have been able to find the ideal location for both a main central kitchen and secondary kitchens to sell precooked italian meals and ingredients to restaurants. by applying a systematic location analysis, we have collected precious informations about each district which could then use later to further deepen the study.

In [ ]:

```

```