Capstone Project - The Battle of Neighborhoods (Week 2)

Author: Dávid Kóczi (Idea from example project)

Table of content

Introduction

In this project I will try to find an optimal location for a Chines restaurant in my country. Specifically, this report will be targeted to stakeholders interested in opening an **Chinese restaurant** in **Budapest**, Hungary.

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

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 decission are:

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

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

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

- centers of candidate areas will be generated algorithmically and approximate addresses of centers of those areas will be obtained using **Google Maps API reverse geocoding**
- number of restaurants and their type and location in every neighborhood will be obtained using **Foursquare API**
- coordinate of Budapest center will be obtained using **Google Maps API geocoding** of well known Budapest location

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. 10x10 killometers centered around Budapest city center.

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

In [1]:
```
pip install beautifulsoup4
```

```
Collecting beautifulsoup4
  Downloading https://files.pythonhosted.org/packages/66/25/ff030e2437265616a1e9b25ccc864e0371a0bc3adb7c5a404fd661c6f4f6/beauti
fulsoup4-4.9.1-py3-none-any.whl (115kB)
     |████████████████████████████████| 122kB 6.0MB/s eta 0:00:01
Collecting soupsieve>1.2 (from beautifulsoup4)
  Downloading https://files.pythonhosted.org/packages/6f/8f/457f4a5390eeae1cc3aeab89deb7724c965be841ffca6cfca9197482e470/soupsi
eve-2.0.1-py3-none-any.whl
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.9.1 soupsieve-2.0.1
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
pip install geocoder
```

```
Collecting geocoder
  Downloading https://files.pythonhosted.org/packages/4f/6b/13166c909ad2f2d76b929a4227c952630ebaf0d729f6317eb09cbceccbab/geocoder-1.38.1-py2.py3-none-any.whl (98kB)
     |████████████████████████████████| 102kB 5.7MB/s ta 0:00:011
Collecting click (from geocoder)
  Downloading https://files.pythonhosted.org/packages/d2/3d/fa76db83bf75c4f8d338c2fd15c8d33fdd7ad23a9b5e57eb6c5de26b430e/click-7.1.2-py2.py3-none-any.whl (82kB)
     |████████████████████████████████| 92kB 5.6MB/s  eta 0:00:01
Requirement already satisfied: six in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from geocoder) (1.14.0)
Requirement already satisfied: requests in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from geocoder) (2.23.0)
Collecting ratelim (from geocoder)
  Downloading https://files.pythonhosted.org/packages/f2/98/7e6d147fd16a10a5f821db6e25f192265d6ecca3d82957a4fdd592cad49c/ratelim-0.1.6-py2.py3-none-any.whl
Collecting future (from geocoder)
  Downloading https://files.pythonhosted.org/packages/45/0b/38b06fd9b92dc2b68d58b75f900e97884c45bedd2ff83203d933cf5851c9/future-0.18.2.tar.gz (829kB)
     |████████████████████████████████| 829kB 25.0MB/s eta 0:00:01
Requirement already satisfied: certifi>=2017.4.17 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests->geocoder) (2020.4.5.1)
Requirement already satisfied: chardet<4,>=3.0.2 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests->geocoder) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests->geocoder) (1.25.9)
Requirement already satisfied: idna<3,>=2.5 in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from requests->geocoder) (2.9)
Requirement already satisfied: decorator in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (from ratelim->geocoder) (4.4.2)
Building wheels for collected packages: future
  Building wheel for future (setup.py) ... done
  Stored in directory: /home/jupyterlab/.cache/pip/wheels/8b/99/a0/81daf51dcd359a9377b110a8a886b3895921802d2fc1b2397e
Successfully built future
Installing collected packages: click, ratelim, future, geocoder
Successfully installed click-7.1.2 future-0.18.2 geocoder-1.38.1 ratelim-0.1.6
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
pip install geopy
```

```
Collecting geopy
  Downloading https://files.pythonhosted.org/packages/ab/97/25def417bf5db4cc6b89b47a56961b893d4ee4fec0c335f5b9476a8ff153/geopy-1.22.0-py2.py3-none-any.whl (113kB)
     |████████████████████████████████| 122kB 6.1MB/s eta 0:00:01
Collecting geographiclib<2,>=1.49 (from geopy)
  Downloading https://files.pythonhosted.org/packages/8b/62/26ec95a98ba64299163199e95ad1b0e34ad3f4e176e221c40245f211e425/geographiclib-1.50-py3-none-any.whl
Installing collected packages: geographiclib, geopy
Successfully installed geographiclib-1.50 geopy-1.22.0
Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
pip install pyproj
```

```
Requirement already satisfied: pyproj in /home/jupyterlab/conda/envs/python/lib/python3.6/site-packages (1.9.6)
Note: you may need to restart the kernel to use updated packages.
```

In [5]:
```
pip install shapely
```

```
Collecting shapely
  Downloading https://files.pythonhosted.org/packages/20/fa/c96d3461fda99ed8e82ff0b219ac2c8384694b4e640a611a1a8390ecd415/Shapely-1.7.0-cp36-cp36m-manylinux1_x86_64.whl (1.8MB)
     |████████████████████████████████| 1.8MB 24.4MB/s eta 0:00:01
Installing collected packages: shapely
Successfully installed shapely-1.7.0
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import pandas as pd
import numpy as np
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
import requests
import sys
from bs4 import BeautifulSoup
import geocoder
import os
import folium # map rendering library
from geopy.geocoders import Nominatim
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.colors as colors
%matplotlib inline
import shapely.geometry
import pyproj
import pickle
import math

print('Libraries imported.')
```

```
Libraries imported.
```

In [3]:
```
foursquare_client_id = 'xxxxxxxxxxx'
foursquare_client_secret ='xxxxxxxxx'
google_api_key = 'xxxxxxxxxxx'
```

In [4]:
```python
import requests


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

address = 'Palota,Budapes,Hungary'

budapest_center = get_coordinates(google_api_key, address)
print('Coordinate of {}: {}'.format(address, budapest_center))
```

```
Coordinate of Palota,Budapes,Hungary: [47.4969189, 19.0361941]
```

In [5]:
```
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('Budapest center longitude={}, latitude={}'.format(budapest_center[1], budapest_center[0]))
x, y = lonlat_to_xy(budapest_center[1], budapest_center[0])
print('Budapest center UTM X={}, Y={}'.format(x, y))
lo, la = xy_to_lonlat(x, y)
print('Budapest center longitude={}, latitude={}'.format(lo, la))
```

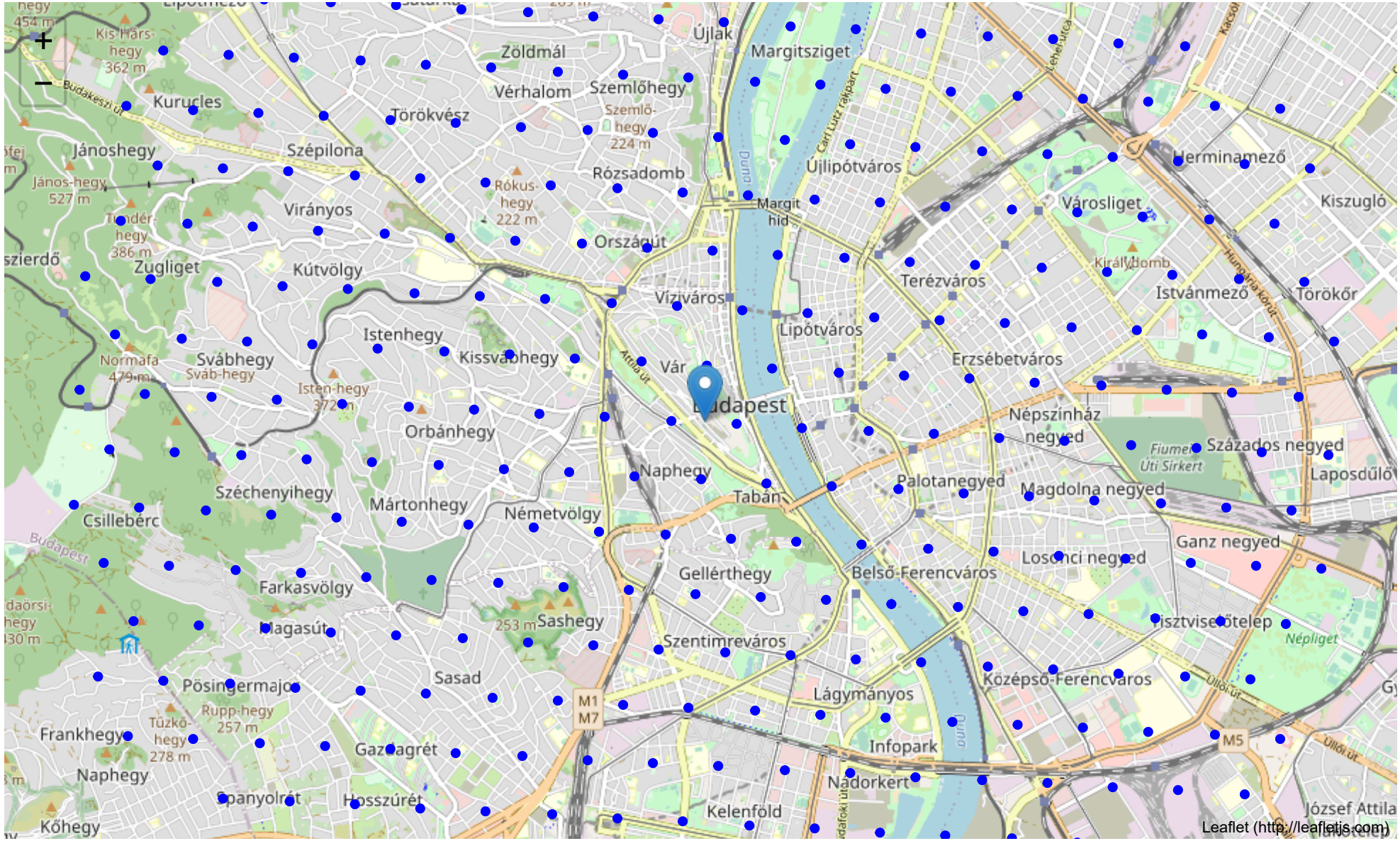
```
Coordinate transformation check
-------------------------------
Budapest center longitude=19.0361941, latitude=47.4969189
Budapest center UTM X=803975.6742963139, Y=5268287.058729421
Budapest center longitude=19.036194100000003, latitude=47.49691889999999
```

In [6]:
```
budapest_center_x, budapest_center_y = lonlat_to_xy(budapest_center[1], budapest_center[0]) # City center

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = budapest_center_x - 6000
x_step = 600
y_min = budapest_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(budapest_center_x, budapest_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]:
```
map_budapest = folium.Map(location=budapest_center, zoom_start=13)
folium.Marker(budapest_center, popup='Palota').add_to(map_budapest)
for lat, lon in zip(latitudes, longitudes):
    folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_opacity=1).add_to(map_budapest)
map_budapest
```

Out[7]:



In [8]:
```
def get_address(api_key, latitude, longitude, verbose=False):
    try:
        url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&latlng={},{}'.format(api_key, latitude, longitude)
        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, budapest_center[0], budapest_center[1])
print('Reverse geocoding check')
print('-----------------------')
print('Address of [{}, {}] is: {}'.format(budapest_center[0], budapest_center[1], addr))
```

```
Reverse geocoding check
-----------------------
Address of [47.4969189, 19.0361941] is: Budapest, 1014 Budapest, Budavári Palota F épület, 1013 Hungary
```

In [9]:
```
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(', Hungary', '') # We don't need country part of address
    addresses.append(address)
    print(' .', end='')
print(' done.')
```

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

In [10]:
```
addresses[150:170]
```

Out[10]:
```
['Budapest, Vajda Péter u. 2, 1089 Hungary',
 'Budapest, Vajda Péter u. 8, 1089 Hungary',
 'Budapest, Kőbányai út 41, 1101 Hungary',
 'Budapest, Sötétvágás utca, 1121 Hungary',
 'Budapest, Hegyhát út 16, 1121 Hungary',
 'Budapest, Agancs út 17, 1121 Hungary',
 'Budapest, Széchenyi-emlék út 12b, 1121 Hungary',
 'Budapest, Hangya u. 42, 1121 Hungary',
 'Budapest, Tamási Áron u. 30, 1124 Hungary',
 'Budapest, Németvölgyi út 53c, 1124 Hungary',
 'Budapest, Kiss János altábornagy u. 33a, 1126 Hungary',
 'Budapest, Avar u. 5a, 1123 Hungary',
 'Budapest, Lisznyai u. 27, 1016 Hungary',
 'Budapest, Attila út 11, 1013 Hungary',
 'Budapest, Március 15. tér, 1056 Hungary',
 'Budapest, Henszlmann Imre u. 5, 1053 Hungary',
 'Budapest, Horánszky u. 21, 1085 Hungary',
 'Budapest, Őr u. 3, 1084 Hungary',
 'Budapest, Magdolna u. 22, 1086 Hungary',
 'Budapest, Orczy tér 1., 1087 Hungary']
```

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

df_locations.head(10)
```

Out[11]:

	Address	Latitude	Longitude	X	Y	Distance from center
0	Budapest, Bedő u. 9, 1112 Hungary	47.446452	19.008445	802175.674296	5.262571e+06	5992.495307
1	Budapest, Murányi u. 14, 1221 Hungary	47.446174	19.016384	802775.674296	5.262571e+06	5840.376700
2	Budapest, Kubikos u. 3, 1116 Hungary	47.445895	19.024323	803375.674296	5.262571e+06	5747.173218
3	Budapest, Verbéna u. 53, 1116 Hungary	47.445616	19.032262	803975.674296	5.262571e+06	5715.767665
4	Budapest, Gyékényes u. 15, 1116 Hungary	47.445336	19.040200	804575.674296	5.262571e+06	5747.173218
5	Budapest, Hunyadi János út 8, 1117 Hungary	47.445056	19.048139	805175.674296	5.262571e+06	5840.376700
6	Budapest, Szikratávíró u. 12-14, 1211 Hungary	47.444774	19.056077	805775.674296	5.262571e+06	5992.495307
7	Budapest, Egér út, 1112 Hungary	47.451533	18.996890	801275.674296	5.263091e+06	5855.766389
8	Budapest, Repülőtéri út 6, 1112 Hungary	47.451255	19.004830	801875.674296	5.263091e+06	5604.462508
9	Budapest, Ütköző sor 12, 1112 Hungary	47.450977	19.012770	802475.674296	5.263091e+06	5408.326913

In [12]:
```
df_locations.to_pickle('./locations.pkl')
```

In [13]:

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

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

Chines_restaurant_categories = ['4bf58dd8d48988d145941735','52af3a5e3cf9994f4e043bea','52af3a723cf9994f4e043bec',
                                '52af3a7c3cf9994f4e043bed','58daa1558bbb0b01f18ec1d3','52af3a673cf9994f4e043beb',
                                '52af3a903cf9994f4e043bee','4bf58dd8d48988d1f5931735','52af3a9f3cf9994f4e043bef',
                                '52af3aaa3cf9994f4e043bf0','52af3ab53cf9994f4e043bf1','52af3abe3cf9994f4e043bf2',
                                '52af3ac83cf9994f4e043bf3','52af3ad23cf9994f4e043bf4','52af3add3cf9994f4e043bf5',
                                '52af3af23cf9994f4e043bf7','52af3ae63cf9994f4e043bf6','52af3afc3cf9994f4e043bf8',
                                '52af3b053cf9994f4e043bf9','52af3b213cf9994f4e043bfa','52af3b293cf9994f4e043bfb',
                                '52af3b343cf9994f4e043bfc','52af3b3b3cf9994f4e043bfd','52af3b463cf9994f4e043bfe',
                                '52af3b6e3cf9994f4e043c02','52af3b773cf9994f4e043c03','52af3b813cf9994f4e043c04',
                                '52af3b893cf9994f4e043c05','52af3b913cf9994f4e043c06','52af3b9a3cf9994f4e043c07',
                                '52af3b633cf9994f4e043c01','52af3b513cf9994f4e043bff','52af3b593cf9994f4e043c00',
                                '52af3ba23cf9994f4e043c08']

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

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

def format_address(location):
    address = ', '.join(location['formattedAddress'])
    address = address.replace(', Hungary', '')
    address = address.replace(', Budapest', '')
    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={}&lim
it={}'.format(
        client_id, client_secret, version, lat, lon, category, radius, limit)
    try:
        results = requests.get(url).json()['response']['groups'][0]['items']
        venues = [(item['venue']['id'],
                   item['venue']['name'],
                   get_categories(item['venue']['categories']),
                   (item['venue']['location']['lat'], item['venue']['location']['lng']),
                   format_address(item['venue']['location']),
                   item['venue']['location']['distance']) for item in results]
    except:
        venues = []
    return venues
```

In [14]:

```
def get_restaurants(lats, lons):
    restaurants = {}
    Chines_restaurants = {}
    location_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 miss any restaurant (we're using dictionarie
s 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:
            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_Chines = is_restaurant(venue_categories, specific_filter=Chines_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_Chines,
x, y)
                if venue_distance<=300:
                    area_restaurants.append(restaurant)
                restaurants[venue_id] = restaurant
                if is_Chines:
                    Chines_restaurants[venue_id] = restaurant
        location_restaurants.append(area_restaurants)
        print(' .', end='')
    print(' done.')
    return restaurants, Chines_restaurants, location_restaurants

# Try to load from local file system in case we did this before
restaurants = {}
Chines_restaurants = {}
location_restaurants = []
loaded = False
try:
    with open('restaurants_350.pkl', 'rb') as f:
        restaurants = pickle.load(f)
    with open('Chines_restaurants_350.pkl', 'rb') as f:
        Chines_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, Chines_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('Chines_restaurants_350.pkl', 'wb') as f:
        pickle.dump(Chines_restaurants, f)
    with open('location_restaurants_350.pkl', 'wb') as f:
        pickle.dump(location_restaurants, f)
```

```
Restaurant data loaded.
```

In [15]:
```
print('Total number of restaurants:', len(restaurants))
print('Total number of Chines restaurants:', len(Chines_restaurants))
print('Percentage of Chines restaurants: {:.2f}%'.format(len(Chines_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: 1195
Total number of Chines restaurants: 107
Percentage of Chines restaurants: 8.95%
Average number of restaurants in neighborhood: 2.9148351648351647
```

In [16]:
```
print('List of all restaurants')
print('----------------------')
for r in list(restaurants.values())[:10]:
    print(r)
print('...')
print('Total:', len(restaurants))
```

```
List of all restaurants
----------------------
('51cb0f1f498ec6821e1b8ddf', 'Regős Vendéglő', 47.44873665124521, 19.00875657477026, 'Budapest, Kapolcs Utca 2/a, Magyarorszá
g', 255, False, 802186.0789610844, 5262826.310361378)
('5abe23db86f4cc283628c3bc', 'Rétisas Étterem', 47.447865, 19.025368, 'Budapest, 1116, Magyarország', 232, False, 803443.118081
6896, 5262794.231572624)
('4d7cd7fdcf3f37046c635840', 'Két Dió Vendéglő', 47.44807715976893, 19.025110176387784, 'Budapest, Hunyadi Mátyás út 56, 1116,
Magyarország', 250, False, 803422.4633907927, 5262816.797640705)
('4e143bb562e14518a923543e', 'Mazsola Konyhája', 47.443572805589206, 19.04454422207084, 'Budapest, Vegyész u. 17-25., 1116, Mag
yarország', 316, False, 804913.3094058295, 5262392.4263895685)
('5a02bd4c6e465055da6b9cfc', 'Péterhegyi Lejtő', 47.45127, 19.016428, 'Budapest, 1112, Magyarország', 328, False, 802749.721230
1941, 5263137.674752849)
('4bd17410caff9521d0aad0f0', 'VakVarjú Vendéglő és Sörkert', 47.449675522614854, 19.036447240466366, 'Budapest, Érem u. 2., 111
6, Magyarország', 52, False, 804267.7254642283, 5263038.751666099)
('4db160ab1e729fcc565002ca', 'Újbuda Terasz', 47.45082819301837, 19.03802614452911, 'Budapest, Fehérvári út 168-178 (Kondorosi
út), 1116, Magyarország', 132, False, 804380.0658363949, 5263173.014228045)
('4ca76b83931bb60cde229de2', 'Kondorosi Terasz', 47.450842, 19.038222833333336, 'Magyarország', 145, False, 804394.8100395815,
5263175.319308126)
('4bd564ce6f6495217ef76eec', 'Big Robi Konyhája', 47.45162583933072, 19.050085331558627, 'Budapest, Budafoki út 215., Magyarors
zág', 289, False, 805284.3159062241, 5263308.980096486)
('4bcc4a9468f976b0ee4d6283', 'Bazsalikom Vendéglő', 47.453783319282124, 19.021655283767256, 'Budapest, Gépész u. 32. (Bazsaliko
m u.), 1116, Magyarország', 300, False, 803129.2226407711, 5263437.326550361)
...
Total: 1195
```

In [17]:
```
print('List of Chines restaurants')
print('--------------------------')
for r in list(Chines_restaurants.values())[:10]:
    print(r)
print('...')
print('Total:', len(Chines_restaurants))
```

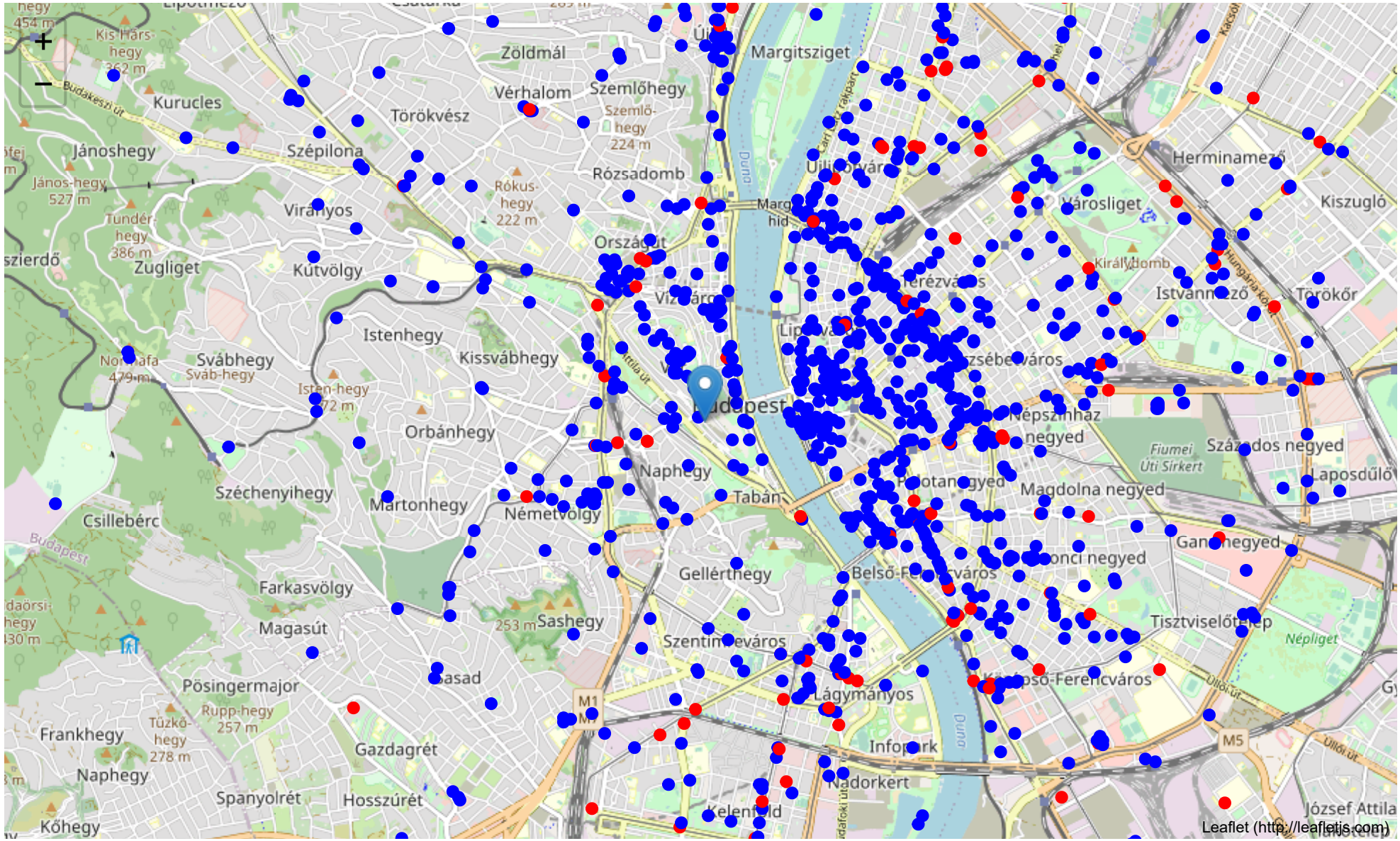
```
List of Chines restaurants
--------------------------
('4d1b1d5c6526a35d70d00c16', 'Renhe Kínai Gyors Büfé', 47.46020340279991, 19.05195526708575, 'Budapest, Újbuda Center (Budafoki út), 1117, Magyarország', 171, True, 805375.5050742112, 5264269.399003791)
('4e9ea27330f8fb677600b52d', 'Yu Cheng Kínai Gyorsbüfé', 47.465346722888484, 19.022654357379107, 'Budapest, Etele tér 5., 1115, Magyarország', 249, True, 803137.9426323414, 5264726.058479761)
('4ebd23f229c2a4fb0a2cc93f', 'Kínai Büfé', 47.463836, 19.033225, 'Magyarország', 59, True, 803943.1299851767, 5264599.526640626)
('4d3055cd789a8cfa530a2ec6', 'Kínai étterem', 47.46287984902365, 19.04245079055973, 'Budapest, Fehérvári út 88/b (Hengermalom út), 1119, Magyarország', 138, True, 804643.81710959, 5264529.445585362)
('507a910de4b0c101c6e15467', 'Hong Li Kínai Gyorsétterem', 47.46596957328054, 19.043284187397806, 'Magyarország', 347, True, 804688.7325836045, 5264876.020426936)
('4d9b02865e52224b2a6038e3', 'Kínai Büfé', 47.471367663056405, 19.030937973376734, 'Budapest, Bartók Béla út 124, 1113, Magyarország', 263, True, 803727.3474262815, 5265427.432985619)
('523c846911d20d24810670ff', 'Jinwei kínai gyorsétterem', 47.47023065027255, 19.045247634972085, 'Magyarország', 198, True, 804811.9966107197, 5265357.183419685)
('4f7d74e4e4b0ffb6a3885532', 'Csang Úr Gyorsétterem', 47.467598464150186, 19.04609648689286, 'Magyarország', 102, True, 804891.1940828533, 5265068.050085352)
('4cd808b37bb06dcb8f29a0b2', 'Peking Kínai Étterem', 47.46622298299074, 19.07946705407958, 'Budapest IX. kerület, Koppány u. 2-4. (Tesco Soroksári úti Hipermarket), 1097, Magyarország', 184, True, 807413.4806095813, 5265046.816128633)
('50530798e4b0588850dd5b0a', 'sherry peking', 47.46585346467214, 19.09925915762648, '1097 Budapest Koppány Utca 2-4, Magyarország', 188, True, 808906.8894641148, 5265084.321623923)
...
Total: 107
```

In [18]:
```
print('Restaurants around location')
print('--------------------------')
for i in range(100, 110):
    rs = location_restaurants[i][:8]
    names = ', '.join([r[1] for r in rs])
    print('Restaurants around location {}: {}'.format(i+1, names))
```

```
Restaurants around location
--------------------------
Restaurants around location 101: Talponálló kifőzde
Restaurants around location 102: Kakukkfészek
Restaurants around location 103:
Restaurants around location 104: Bécsiszelet Vendéglő
Restaurants around location 105: Mermel étterem, Kuriózum Café
Restaurants around location 106: Hemingway Étterem, Kisvigadó, Montenegroi Gurman, Family Restaurant Kifőzde, Gyros Büfé, Corne
r Falatozó
Restaurants around location 107: Deli's Vegán Bisztró, Eker Gyros Török Étterem, Las Vegans, Buono by Il Treno, Istanbul Kebabs
Non-Stop, Madárfészek gyorsétterem és kávézó, Madárfészek Kínai Gyorsétterem, Döner kebab
Restaurants around location 108: Gyros, El Vandom salátabár, Pho 74, Íz-lelő Étkezde, Ganesha Vega, Stoczek Menza, Bercsényi Ki
svendéglő, Rondo Gyorsétterem és Kávézó, Ji Li Kínai Büfé
Restaurants around location 109: A38 Étterem
Restaurants around location 110: Saloniki Gyros Taverna, Wok Express, Hange Restaurant (Hange Étterem), Lele Kínai Gyorsbüfé, A
dames Salátabár, Pho Ha Noi, Zorba Gyros, Haller Étterem
```

In [19]:
```
map_budapest = folium.Map(location=budapest_center, zoom_start=13)
folium.Marker(budapest_center, popup='Palota').add_to(map_budapest)
for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_budapest = res[6]
    color = 'red' if is_budapest else 'blue'
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True, fill_color=color, fill_opacity=1).add_to(map_budapest)
map_budapest
```

Out[19]:



Now we have all the restaurants in area within few kilometers from Budapest Palota , and we know which ones are Chines restaurants! We also know which restaurants exactly are in vicinity of every neighborhood candidate center.

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

Methodology

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

In first step we have collected the required **data: location and type (category) of every restaurant within 6km from Budapest center** (Budapest Palota). We have also **identified Chines restaurants** (according to Foursquare categorization).

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

In third and final step we will focus on most promising areas and within those create **clusters of locations that meet some basic requirements** established in discussion with stakeholders: we will take into consideration locations with **no more than two restaurants in radius of 250 meters**, and we want locations **without Chines restaurants in radius of 400 meters**. We will present map of all such locations but also create clusters (using **k-means clustering**) of those locations to identify general zones / neighborhoods / addresses which should be a starting point for final 'street level' exploration and search for optimal venue location by stakeholders.

Analysis

In [20]:
```
location_restaurants_count = [len(res) for res in location_restaurants]

df_locations['Restaurants in area'] = location_restaurants_count

print('Average number of restaurants in every area with radius=300m:', np.array(location_restaurants_count).mean())

df_locations.head(10)
```

```
Average number of restaurants in every area with radius=300m: 2.9148351648351647
```

Out[20]:

	Address	Latitude	Longitude	X	Y	Distance from center	Restaurants in area
0	Budapest, Bedő u. 9, 1112 Hungary	47.446452	19.008445	802175.674296	5.262571e+06	5992.495307	1
1	Budapest, Murányi u. 14, 1221 Hungary	47.446174	19.016384	802775.674296	5.262571e+06	5840.376700	0
2	Budapest, Kubikos u. 3, 1116 Hungary	47.445895	19.024323	803375.674296	5.262571e+06	5747.173218	2
3	Budapest, Verbéna u. 53, 1116 Hungary	47.445616	19.032262	803975.674296	5.262571e+06	5715.767665	0
4	Budapest, Gyékényes u. 15, 1116 Hungary	47.445336	19.040200	804575.674296	5.262571e+06	5747.173218	0
5	Budapest, Hunyadi János út 8, 1117 Hungary	47.445056	19.048139	805175.674296	5.262571e+06	5840.376700	0
6	Budapest, Szikratávíró u. 12-14, 1211 Hungary	47.444774	19.056077	805775.674296	5.262571e+06	5992.495307	0
7	Budapest, Egér út, 1112 Hungary	47.451533	18.996890	801275.674296	5.263091e+06	5855.766389	0
8	Budapest, Repülőtéri út 6, 1112 Hungary	47.451255	19.004830	801875.674296	5.263091e+06	5604.462508	0
9	Budapest, Ütköző sor 12, 1112 Hungary	47.450977	19.012770	802475.674296	5.263091e+06	5408.326913	0

In [29]:
```
distances_to_chines_restaurant = []

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

df_locations['Distance to Chines restaurant'] = distances_to_chines_restaurant
```

In [30]:
```
df_locations.head(10)
```

Out[30]:

	Address	Latitude	Longitude	X	Y	Distance from center	Restaurants in area	Distance to Chines restaurant
0	Budapest, Bedő u. 9, 1112 Hungary	47.446452	19.008445	802175.674296	5.262571e+06	5992.495307	1	2359.869269
1	Budapest, Murányi u. 14, 1221 Hungary	47.446174	19.016384	802775.674296	5.262571e+06	5840.376700	0	2185.008229
2	Budapest, Kubikos u. 3, 1116 Hungary	47.445895	19.024323	803375.674296	5.262571e+06	5747.173218	2	2106.120963
3	Budapest, Verbéna u. 53, 1116 Hungary	47.445616	19.032262	803975.674296	5.262571e+06	5715.767665	0	2028.496656
4	Budapest, Gyékényes u. 15, 1116 Hungary	47.445336	19.040200	804575.674296	5.262571e+06	5747.173218	0	1877.045510
5	Budapest, Hunyadi János út 8, 1117 Hungary	47.445056	19.048139	805175.674296	5.262571e+06	5840.376700	0	1709.825404
6	Budapest, Szikratávíró u. 12-14, 1211 Hungary	47.444774	19.056077	805775.674296	5.262571e+06	5992.495307	0	1744.622016
7	Budapest, Egér út, 1112 Hungary	47.451533	18.996890	801275.674296	5.263091e+06	5855.766389	0	2463.528471
8	Budapest, Repülőtéri út 6, 1112 Hungary	47.451255	19.004830	801875.674296	5.263091e+06	5604.462508	0	2065.682449
9	Budapest, Ütköző sor 12, 1112 Hungary	47.450977	19.012770	802475.674296	5.263091e+06	5408.326913	0	1764.177422

In [31]:
```
print('Average distance to closest Chines restaurant from each area center:', df_locations['Distance to Chines restaurant'].mean())
```

```
Average distance to closest Chines restaurant from each area center: 906.6730769768445
```

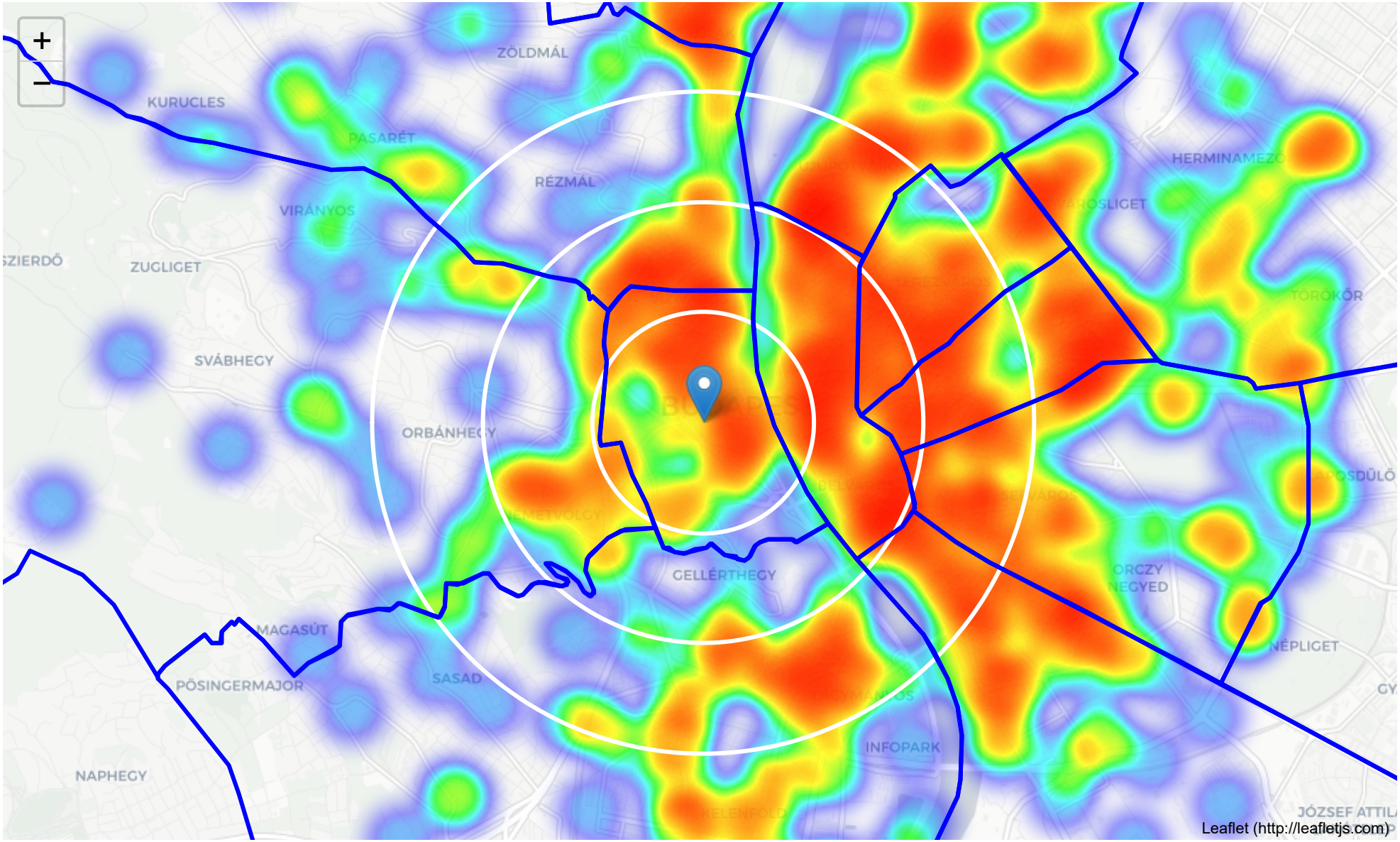
In [40]:
```
budapest_boroughs = 'budapest.geojson'


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

In [41]:
```
restaurant_latlons = [[res[2], res[3]] for res in restaurants.values()]

chines_latlons = [[res[2], res[3]] for res in Chines_restaurants.values()]
```

In [42]:

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

map_budapest = folium.Map(location=budapest_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_budapest) #cartodbpositron cartodbdark_matter
HeatMap(restaurant_latlons).add_to(map_budapest)
folium.Marker(budapest_center).add_to(map_budapest)
folium.Circle(budapest_center, radius=1000, fill=False, color='white').add_to(map_budapest)
folium.Circle(budapest_center, radius=2000, fill=False, color='white').add_to(map_budapest)
folium.Circle(budapest_center, radius=3000, fill=False, color='white').add_to(map_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[42]:

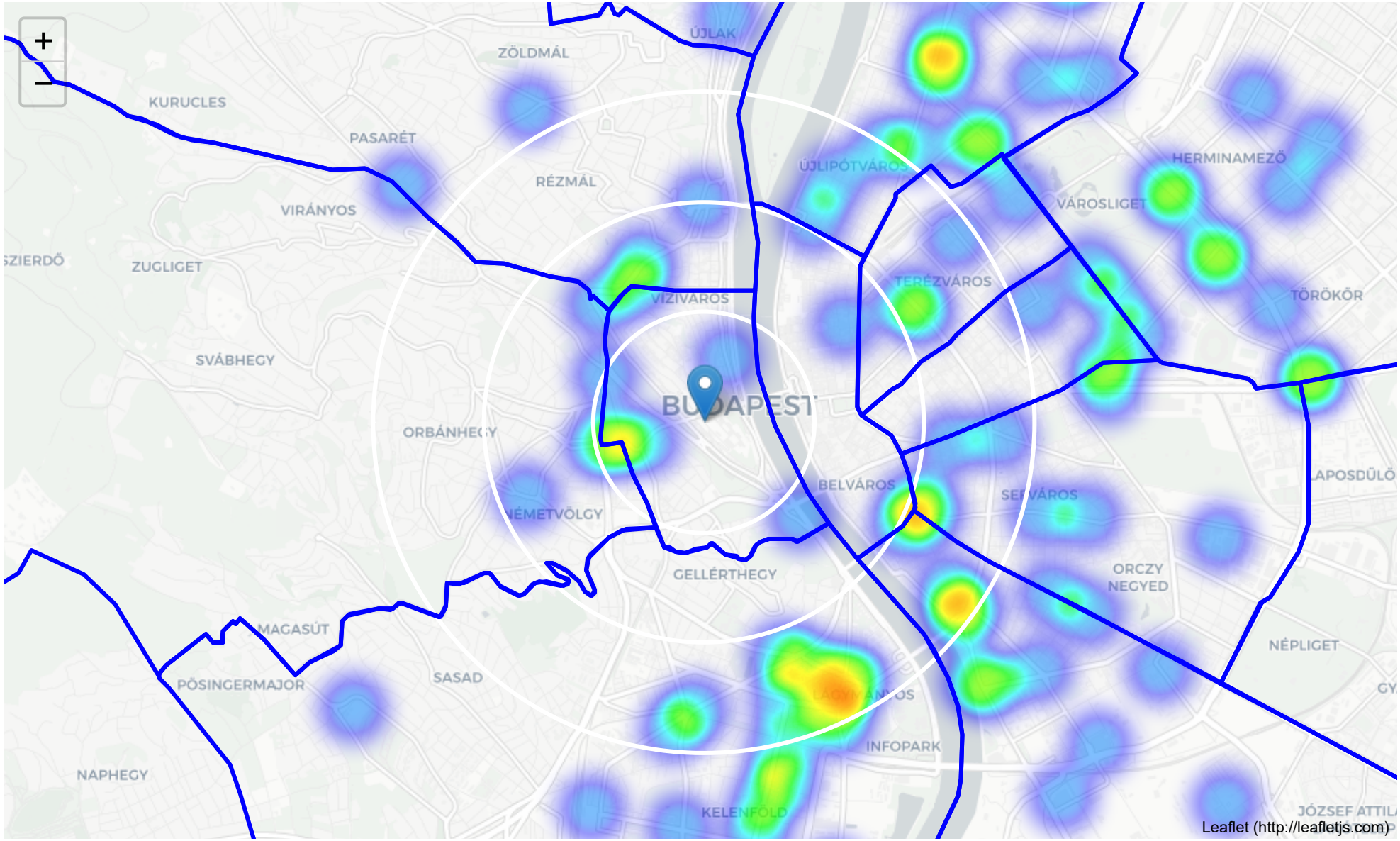


Looks like a few pockets of low restaurant density closest to city center can be found **south, south-east and east from Budapest palota**.

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

In [43]:
```
map_budapest = folium.Map(location=budapest_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_budapest) #cartodbpositron cartodbdark_matter
HeatMap(chines_latlons).add_to(map_budapest)
folium.Marker(budapest_center).add_to(map_budapest)
folium.Circle(budapest_center, radius=1000, fill=False, color='white').add_to(map_budapest)
folium.Circle(budapest_center, radius=2000, fill=False, color='white').add_to(map_budapest)
folium.Circle(budapest_center, radius=3000, fill=False, color='white').add_to(map_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[43]:



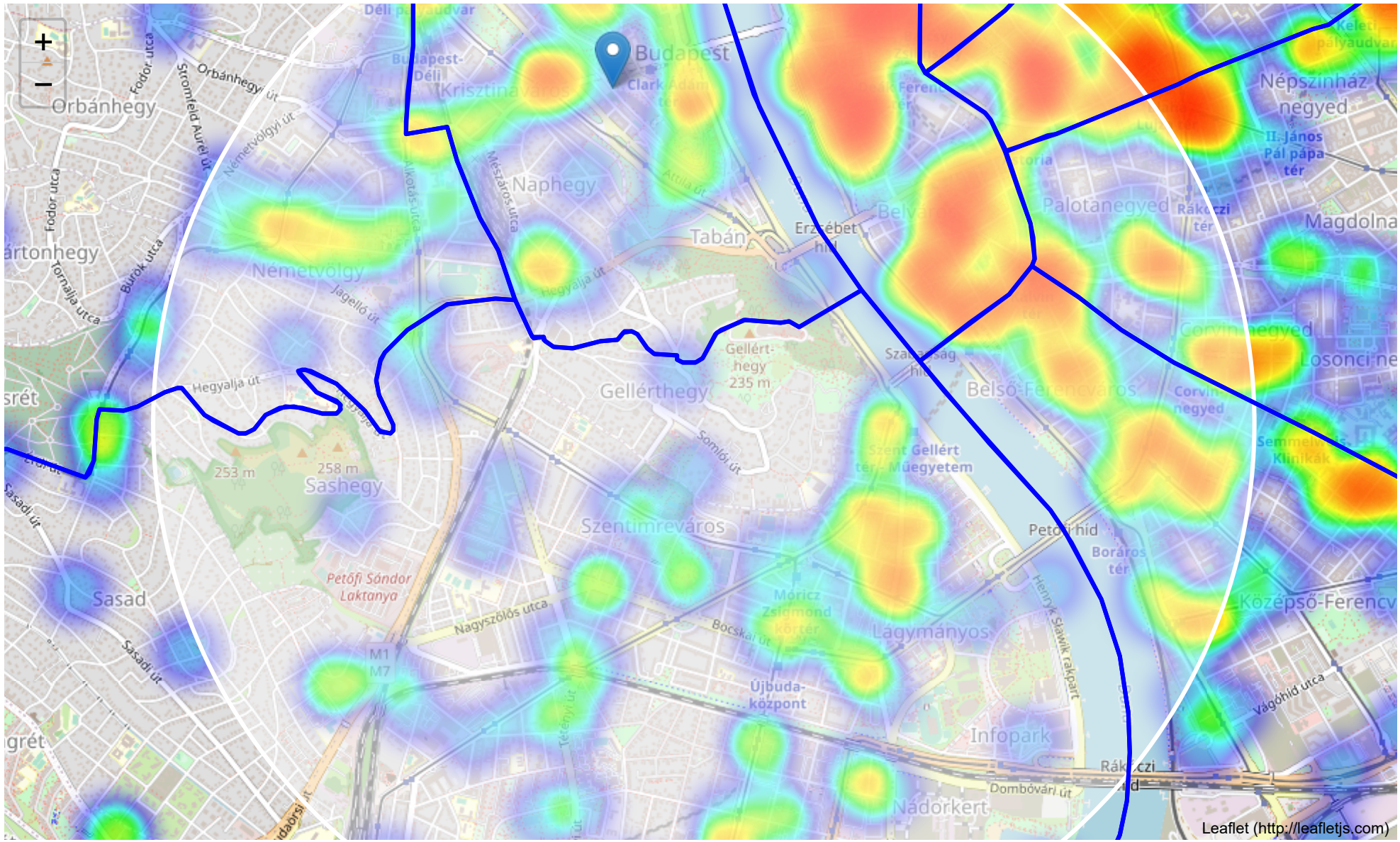
This map is not so 'hot' (Chines restaurants represent a subset of ~15% of all restaurants in Budapest) but it also indicates higher density of existing Chines restaurants directly north and west from Budapest Palota, with closest pockets of **low Italian restaurant density positioned east, south-east and south from city center**.

Based on this we will now focus our analysis on areas *south-west, south, south-east and east from Budapest center* - we will move the center of our area of interest and reduce it's size to have a radius of **2.5km**.

In [44]:

```
roi_x_min = budapest_center_x - 2000
roi_y_max = budapest_center_y + 1000
roi_width = 5000
roi_height = 5000
roi_center_x = roi_x_min + 2500
roi_center_y = roi_y_max - 2500
roi_center_lon, roi_center_lat = xy_to_lonlat(roi_center_x, roi_center_y)
roi_center = [roi_center_lat, roi_center_lon]

map_budapest = folium.Map(location=roi_center, zoom_start=14)
HeatMap(restaurant_latlons).add_to(map_budapest)
folium.Marker(budapest_center).add_to(map_budapest)
folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opacity=0.4).add_to(map_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[44]:



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

In [45]:
```
k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_step = 100
y_step = 100 * k
roi_y_min = roi_center_y - 2500

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

```
2261 candidate neighborhood centers generated.
```

In [46]:
```
def count_restaurants_nearby(x, y, restaurants, radius=250):
    count = 0
    for res in restaurants.values():
        res_x = res[7]; res_y = res[8]
        d = calc_xy_distance(x, y, res_x, res_y)
        if d<=radius:
            count += 1
    return count

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

roi_restaurant_counts = []
roi_chines_distances = []

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

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

In [47]:
```
# 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 Italian restaurant':roi_chines_distances})

df_roi_locations.head(10)
```

Out[47]:

	Latitude	Longitude	X	Y	Restaurants nearby	Distance to Italian restaurant
0	47.460806	19.039397	804425.674296	5.264287e+06	1	326.094580
1	47.460759	19.040720	804525.674296	5.264287e+06	1	269.646272
2	47.461840	19.032177	803875.674296	5.264374e+06	1	235.723218
3	47.461794	19.033500	803975.674296	5.264374e+06	1	228.197936
4	47.461747	19.034824	804075.674296	5.264374e+06	0	261.883868
5	47.461700	19.036147	804175.674296	5.264374e+06	0	324.178997
6	47.461654	19.037471	804275.674296	5.264374e+06	0	399.747275
7	47.461607	19.038795	804375.674296	5.264374e+06	0	310.111788
8	47.461560	19.040118	804475.674296	5.264374e+06	2	229.217710
9	47.461513	19.041442	804575.674296	5.264374e+06	4	170.035867

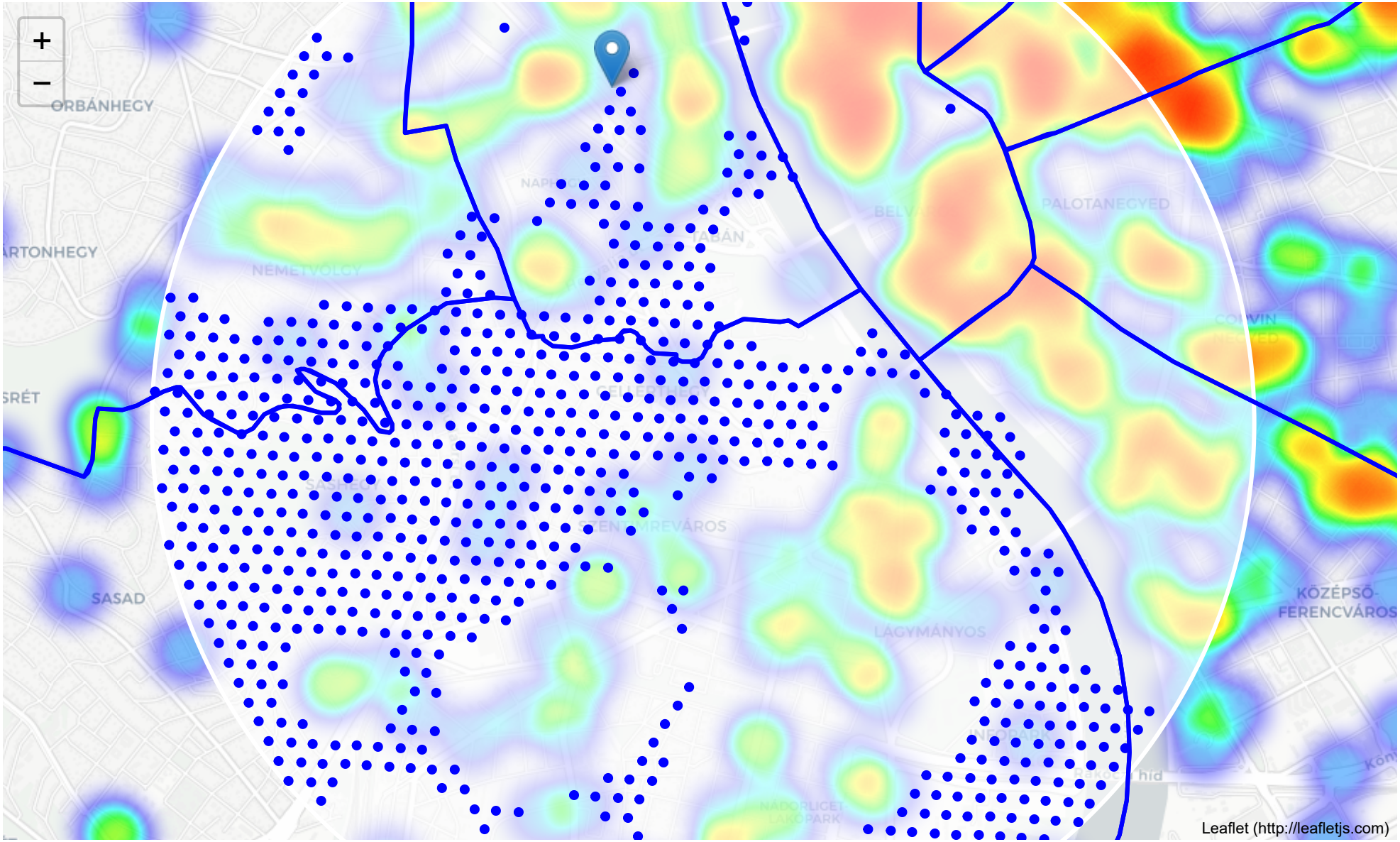
In [64]:
```
good_res_count = np.array((df_roi_locations['Restaurants nearby']<=2))
print('Locations with no more than two restaurants nearby:', good_res_count.sum())

good_ita_distance = np.array(df_roi_locations['Distance to Italian restaurant']>=400)
print('Locations with no Chines restaurants within 400m:', good_ita_distance.sum())

good_locations = np.logical_and(good_res_count, good_ita_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: 1180
Locations with no Chines restaurants within 400m: 1151
Locations with both conditions met: 755
```

In [66]:

```
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_budapest = folium.Map(location=roi_center, zoom_start=14)
folium.TileLayer('cartodbpositron').add_to(map_budapest)
HeatMap(restaurant_latlons).add_to(map_budapest)
folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opacity=0.6).add_to(map_budapest)
folium.Marker(budapest_center).add_to(map_budapest)
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_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[66]:

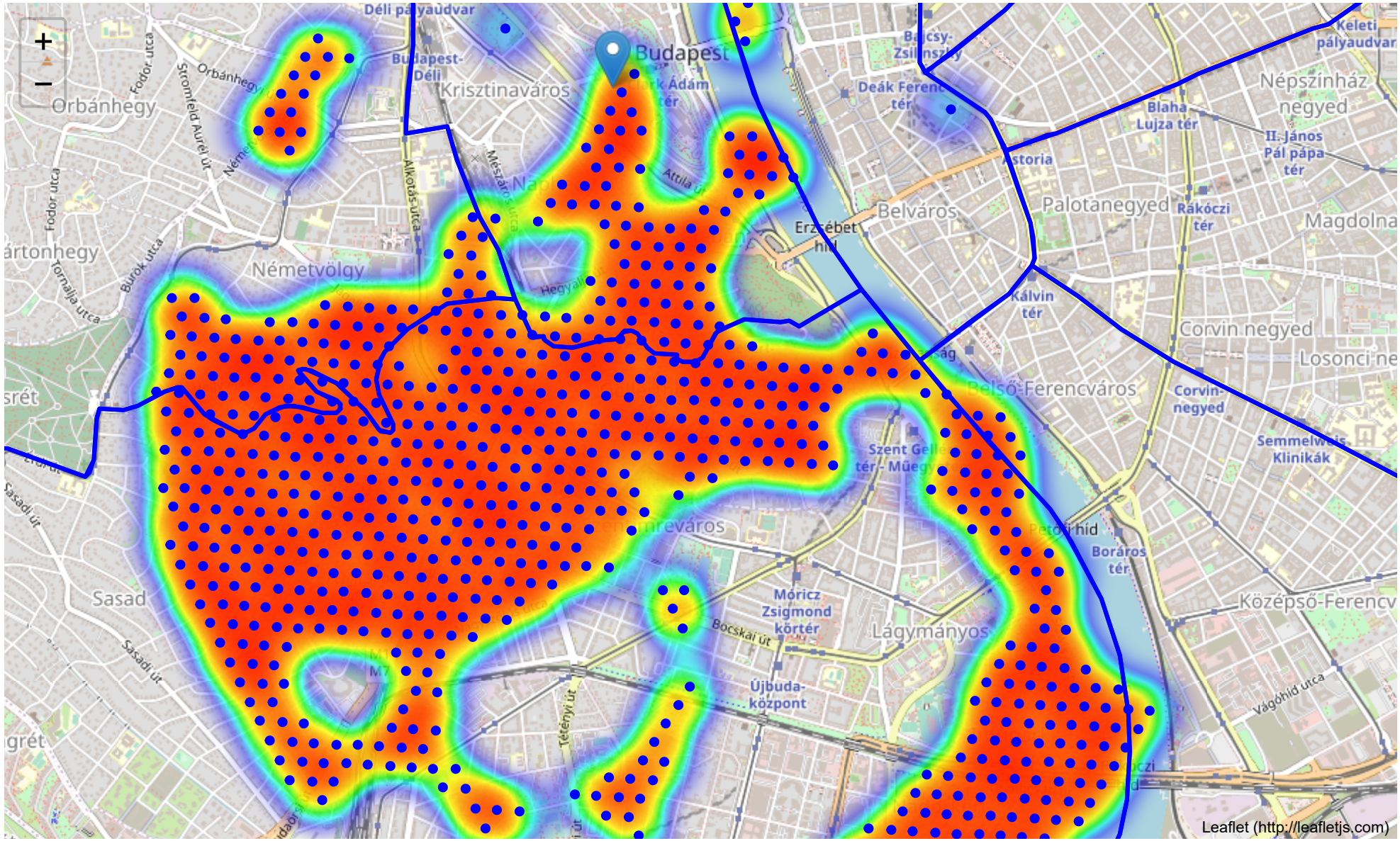


Looking good. We now have a bunch of locations fairly close to Budapest Palota , and we know that each of those locations has no more than two restaurants in radius of 250m, and no Chines restaurant closer than 400m. Any of those locations is a potential candidate for a new Chines restaurant, at least based on nearby competition.

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

In [67]:
```
map_budapest = folium.Map(location=roi_center, zoom_start=14)
HeatMap(good_locations, radius=25).add_to(map_budapest)
folium.Marker(budapest_center).add_to(map_budapest)
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_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[67]:



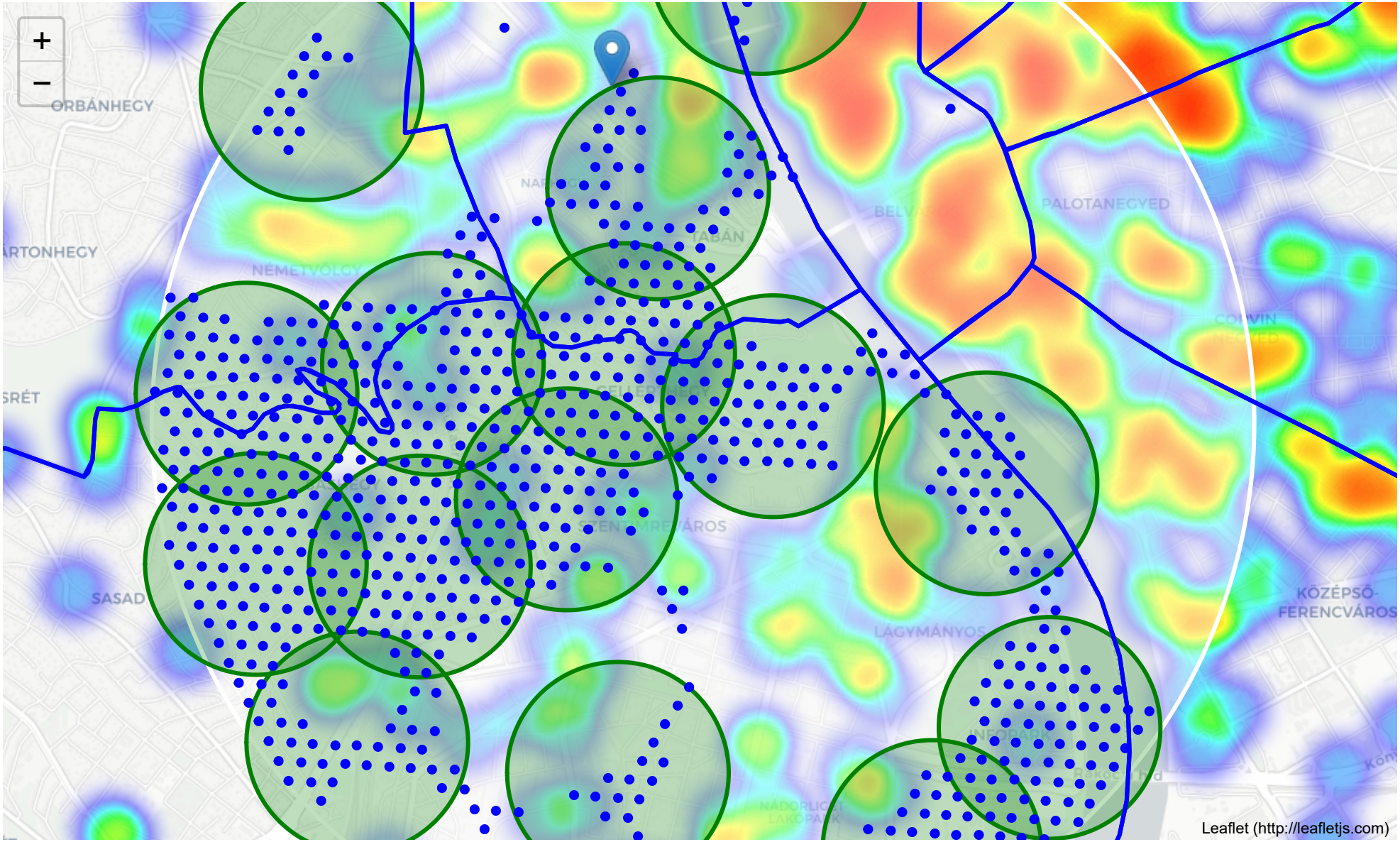
Looking good. What we have now is a clear indication of zones with low number of restaurants in vicinity, and *no* Chines restaurants at all nearby.

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

In [69]:

```
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_budapest = folium.Map(location=roi_center, zoom_start=14)
folium.TileLayer('cartodbpositron').add_to(map_budapest)
HeatMap(restaurant_latlons).add_to(map_budapest)
folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opacity=0.4).add_to(map_budapest)
folium.Marker(budapest_center).add_to(map_budapest)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=True, fill_opacity=0.25).add_to(map_budapest)
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_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[69]:



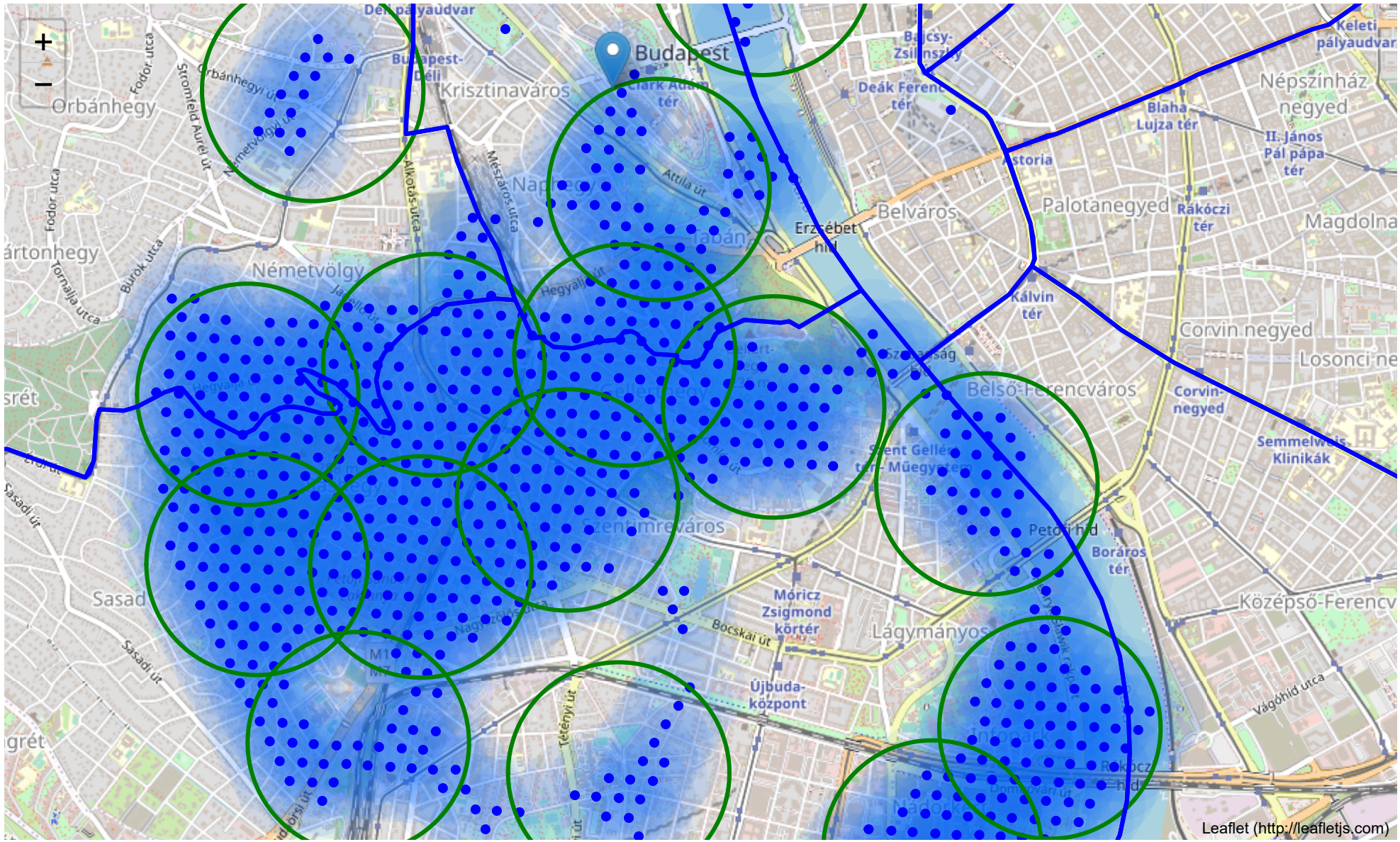
My clusters represent groupings of most of the candidate locations and cluster centers are placed nicely in the middle of the zones 'rich' with location candidates.

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

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

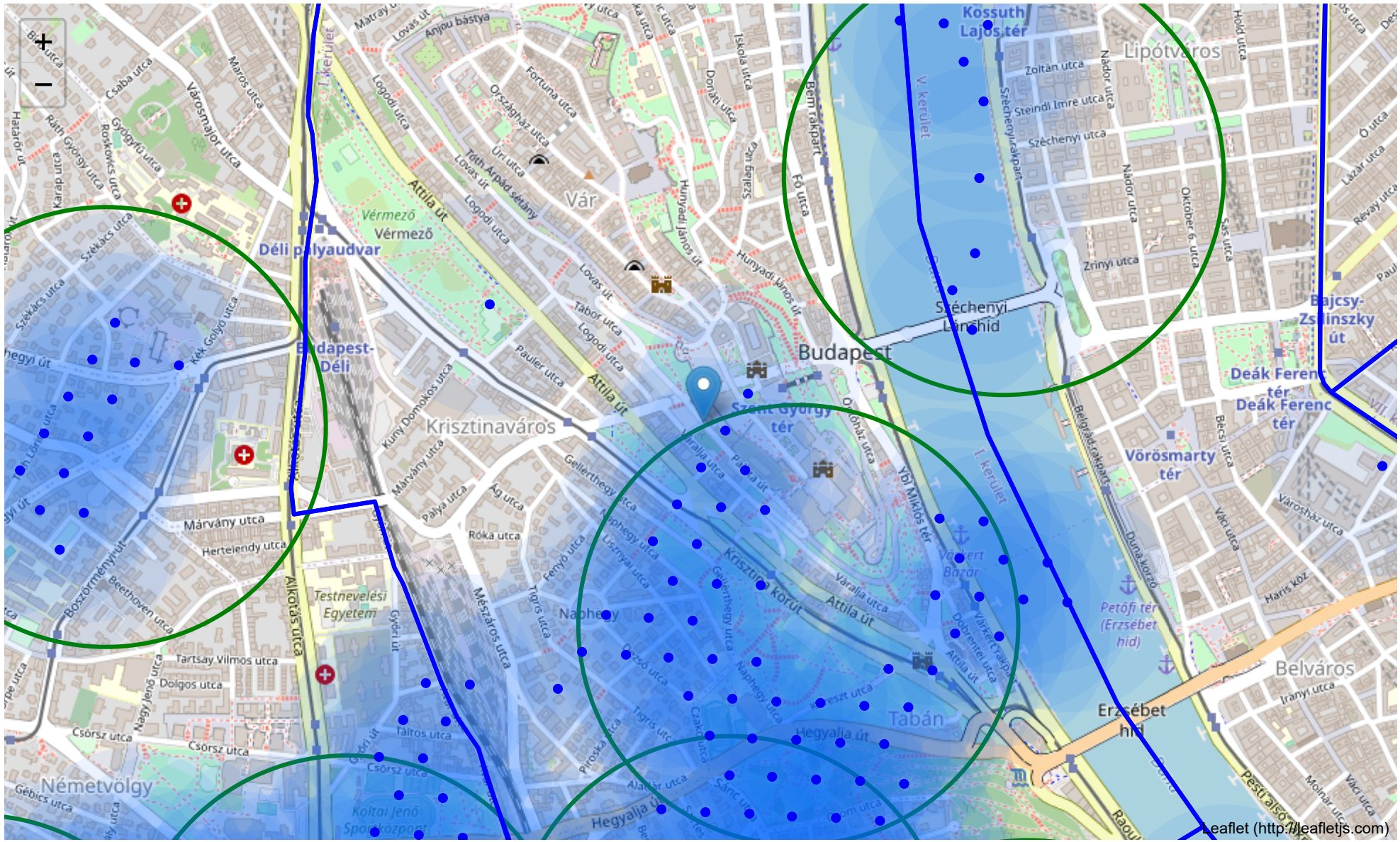
In [70]:
```
map_budapest = folium.Map(location=roi_center, zoom_start=14)
folium.Marker(budapest_center).add_to(map_budapest)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#00000000', fill=True, fill_color='#0066ff', fill_opacity=0.07).add_to(map_bud
apest)
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_budapest)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=False).add_to(map_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[70]:


Budapest
Krisztinaváros
Orbánhegy
Németvölgy
Belváros
Palotanegyed
Magdolna
Corvin negyed
Belső-Ferencváros
Lágymányos
Középső-Ferencv
Losonci ne
Népszínház negyed
Sasad
Móricz Zsigmond körtér
Újbuda-központ
Clark Ádám tér
Deák Ferenc tér
Blaha Lujza tér
II. János Pál pápa tér
Rákóczi tér
Kálvin tér
Erzsébet híd
Petőfi híd
Boráros tér
Corvin-negyed
Semmelweis Klinikák
Szent Gellért tér - Műegyetem
Bocskai út
Attila út
Alkotás utca
Mészáros utca
Fodor utca
Stromfeld Aurél út
Orbánhegyi út
Hegyalja út
Tétényi út
Vágóhíd utca
Leaflet (http://leafletjs.com)

In [73]:
```
map_budapest = folium.Map(location=[47.4969189, 19.0361941], zoom_start=15)
folium.Marker(budapest_center).add_to(map_budapest)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=False).add_to(map_budapest)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#0000ff00', fill=True, fill_color='#0066ff', fill_opacity=0.07).add_to(map_bud
apest)
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_budapest)
folium.GeoJson(budapest_boroughs, style_function=boroughs_style, name='geojson').add_to(map_budapest)
map_budapest
```

Out[73]:



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

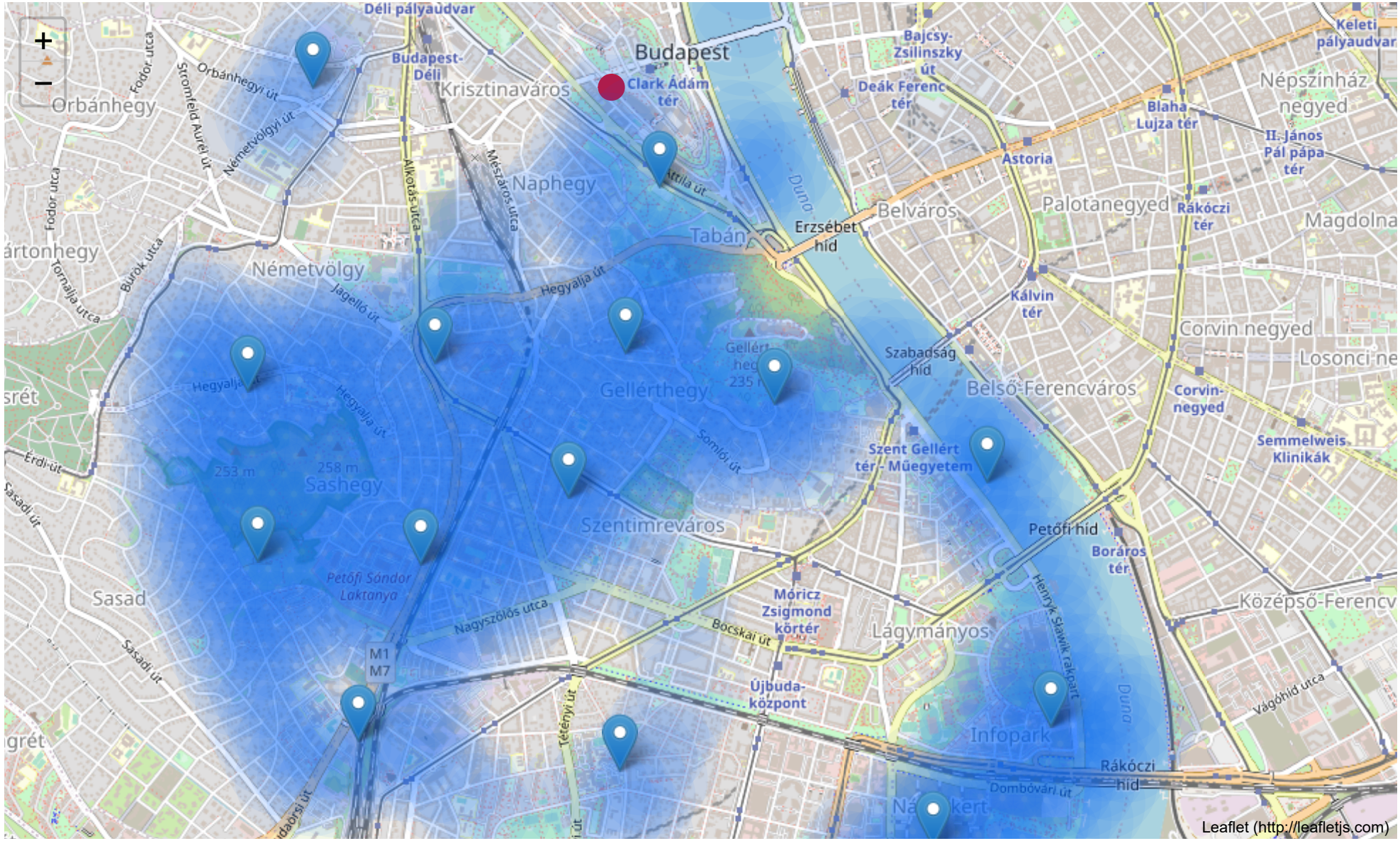
In [74]:
```
candidate_area_addresses = []
print('============================================================')
print('Addresses of centers of areas recommended for further analysis')
print('============================================================\n')
for lon, lat in cluster_centers:
    addr = get_address(google_api_key, lat, lon).replace(', Hungary', '')
    candidate_area_addresses.append(addr)
    x, y = lonlat_to_xy(lon, lat)
    d = calc_xy_distance(x, y, budapest_center_x, budapest_center_y)
    print('{}{} => {:.1f}km from Palota'.format(addr, ' '*(50-len(addr)), d/1000))
```

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

Budapest, Krisztina krt. 101, 1016 Hungary         => 0.5km from Palota
Budapest, Budaörsi út 22, fsz 4, 1118 Hungary      => 1.5km from Palota
Budapest, Pázmány Péter stny. 37, 1117 Hungary     => 3.5km from Palota
Budapest, Budaörsi út 62, 1118 Hungary             => 3.2km from Palota
Budapest, 1644+600 FKM Jobb parti szelvény XI/121. sz. raszter, 1111 Hungary => 2.5km from Palota
Budapest, Dayka Gábor u. 29, 1112 Hungary          => 2.7km from Palota
Budapest, Sasfiók u. 3, 1124 Hungary               => 2.2km from Palota
Budapest, Tarcali u. 14, 1113 Hungary              => 1.9km from Palota
Budapest, Számadó u. 10, 1118 Hungary              => 1.2km from Palota
Budapest, Fejér Lipót u. 25, 1119 Hungary          => 3.1km from Palota
Budapest, Németvölgyi út 13, 1126 Hungary          => 1.4km from Palota
Budapest, Hauszmann Alajos u. 11, 1116 Hungary     => 3.8km from Palota
Budapest, Rezeda u. 9, 1118 Hungary                => 1.6km from Palota
Budapest, Budaörsi út 51, 1118 Hungary             => 2.3km from Palota
Budapest, Akadémia u. 6, 1054 Hungary              => 0.9km from Palota
```

This concludes our analysis. We have created 15 addresses representing centers of zones containing locations with low number of restaurants and no Chines restaurants nearby, all zones being fairly close to city center (all less than 4km from Budapest Palota, and about half of those less than 2km from Palota). Although zones are shown on map with a radius of ~500 meters (green circles), their shape is actually very irregular and their centers/addresses should be considered only as a starting point for exploring area neighborhoods in search for potential restaurant locations. Most of the zones are located in, which we have identified as interesting due to being popular with tourists, fairly close to city center and well connected by public transport.

In [75]:
```
map_budapest = folium.Map(location=roi_center, zoom_start=14)
folium.Circle(budapest_center, radius=50, color='red', fill=True, fill_color='red', fill_opacity=1).add_to(map_budapest)
for lonlat, addr in zip(cluster_centers, candidate_area_addresses):
    folium.Marker([lonlat[1], lonlat[0]], popup=addr).add_to(map_budapest)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#0000ff00', fill=True, fill_color='#0066ff', fill_opacity=0.05).add_to(map_bud
apest)
map_budapest
```

Out[75]:



Results

Our analysis shows that although there is a great number of restaurants in Budapest (~2000 in our initial area of interest which was 12x12km around Palota), there are pockets of low restaurant density fairly close to city center. Highest concentration of restaurants was detected north and west from Palota, so we focused our attention to areas south, south-east and east, corresponding to boroughs Vár, kastély and south-east corner of central Palace borough. Another borough was identified as potentially interesting (Kalrk Ádám square, north-east from Palota), but our attention was focused on Vár and Palota which offer a combination of popularity among tourists, closeness to city center, strong socio-economic dynamics *and* a number of pockets of low restaurant density.

After directing our attention to this more narrow area of interest (covering approx. 5x5km south-east from Palota) we first created a dense grid of location candidates (spaced 100m appart); those locations were then filtered so that those with more than two restaurants in radius of 250m and those with an Chines restaurant closer than 400m were removed.

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

Result of all this is 15 zones containing largest number of potential new restaurant locations based on number of and distance to existing venues - both restaurants in general and Chines restaurants particularly. This, of course, does not imply that those zones are actually optimal locations for a new restaurant! Purpose of this analysis was to only provide info on areas close to Budapest center but not crowded with existing restaurants (particularly Chines) - it is entirely possible that there is a very good reason for small number of restaurants in any of those areas, reasons which would make them unsuitable for a new restaurant regardless of lack of competition in the area. Recommended zones should therefore be considered only as a starting point for more detailed analysis which could eventually result in location which has not only no nearby competition but also other factors taken into account and all other relevant conditions met.

Conclusion

Purpose of this project was to identify Budapest areas close to center with low number of restaurants (particularly Italian restaurants) in order to aid stakeholders in narrowing down the search for optimal location for a new Chines restaurant. By calculating restaurant density distribution from Foursquare data we have first identified general boroughs that justify further analysis (Vár and Kastély), and then generated extensive collection of locations which satisfy some basic requirements regarding existing nearby restaurants. Clustering of those locations was then performed in order to create major zones of interest (containing greatest number of potential locations) and addresses of those zone centers were created to be used as starting points for final exploration by stakeholders.

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

This project prepared by based on the example project. Thank you for your review!