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

Applied Data Science Capstone by IBM/Coursera

Table of contents

Introduction: Business Problem

In this project we will try to find an optimal location for a restaurant. Specifically, this report will be targeted to stakeholders interested in opening an **Italian restaurant** in **Berlin**, Germany.
Since there are lots of restaurants in Berlin we will try to detect **locations that are not already crowded with restaurants**. We are also particularly interested in **areas with no Italian restaurants in vicinity**. We would also prefer locations **as close to city center as possible**, assuming that first two conditions are met.

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

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 Italian 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 Berlin center will be obtained using **Google Maps API geocoding** of well known Berlin location (Alexanderplatz)

Neighborhood Candidates

Let's create latitude & longitude coordinates for centroids of our candidate neighborhoods. We will create a grid of cells covering our area of interest which is aprox. 12x12 killometers centered around Berlin city center.

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

In [1]:

```
# The code was removed by Watson Studio for sharing.
```

In [2]:

```
import requests

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

address = 'Alexanderplatz, Berlin, Germany'
berlin_center = get_coordinates(google_api_key, address)
print('Coordinate of {}: {}'.format(address, berlin_center))
```

```
Coordinate of Alexanderplatz, Berlin, Germany: [52.5219184, 13.4132147]
```

Now let's create a grid of area candidates, equaly spaced, centered around city center and within ~6km from Alexanderplatz. Our neighborhoods will be defined as circular areas with a radius of 300 meters, so our neighborhood centers will be 600 meters apart.

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

In [5]:

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

#!pip install pyproj
import pyproj

import math

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

def xy_to_lonlat(x, y):
    proj_latlon = pyproj.Proj(proj='latlong',datum='WGS84')
```

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

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

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

```
Coordinate transformation check
-------------------------------
Berlin center longitude=13.4132147, latitude=52.5219184
Berlin center UTM X=392341.28017572395, Y=5820273.243274779
Berlin center longitude=13.413214700000001, latitude=52.52191839999997
```

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

In [6]:

```
berlin_center_x, berlin_center_y = lonlat_to_xy(berlin_center[1], berlin_ce
nter[0]) # City center in Cartesian coordinates

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = berlin_center_x - 6000
x_step = 600
y_min = berlin_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(berlin_center_x, berlin_cen
ter_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.
```

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

In [9]:

```
#!pip install folium

import folium
```

In [30]:

```
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.Marker(berlin_center, popup='Alexanderplatz').add_to(map_berlin)
for lat, lon in zip(latitudes, longitudes):
    #folium.CircleMarker([lat, lon], radius=2, color='blue', fill=True, fil
l_color='blue', fill_opacity=1).add_to(map_berlin)
    folium.Circle([lat, lon], radius=300, color='blue', fill=False).add_to(
map_berlin)
    #folium.Marker([lat, lon]).add_to(map_berlin)
map_berlin
```

Out[30]:

OK, we now have the coordinates of centers of neighborhoods/areas to be evaluated, equally spaced (distance from every point to it's neighbors is exactly the same) and within ~6km from Alexanderplatz.

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

In [12]:

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

```
Reverse geocoding check
-----------------------
Address of [52.5219184, 13.4132147] is: Alexanderpl. 5, 10178 Berlin, Germa
ny
```

In [13]:

```python
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(', Germany', '') # We don't need country part
of address
    addresses.append(address)
    print(' .', end='')
print(' done.')
```

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

In [14]:

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

Out[14]:

```
['Frankfurter Allee 147-149, 10365 Berlin',
 'Magdalenenstraße 12, 10365 Berlin',
 'Siegfriedstraße 207, 10365 Berlin',
 'Englische Str. 3, 10587 Berlin',
 'Händelallee 51, 10557 Berlin',
 'Spreeweg, 10557 Berlin',
 'John-Foster-Dulles-Allee 10, 10557 Berlin',
 'B96, 10557 Berlin',
 'Pariser Platz 6A, 10117 Berlin',
 'Unter den Linden 38, 10117 Berlin',
 'Unter den Linden 5, 10117 Berlin',
 'Spreeufer 6, 10178 Berlin',
 'Parochialstraße, 10179 Berlin',
 'Neue Blumenstraße 1, 10179 Berlin',
 'Blumenstraße 41, 10243 Berlin',
 'B5 85, 10243 Berlin',
 'Weidenweg 27, 10249 Berlin',
```

```
 'Rigaer Str. 96, 10247 Berlin',
 'Bänschstraße 58, 10247 Berlin',
 'Parkaue 30, 10367 Berlin']
```

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

In [15]:

```
import pandas as pd

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[15]:

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

In [16]:

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

Foursquare

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

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

In [17]:

```
# The code was removed by Watson Studio for sharing.
```

In [18]:

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

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

italian_restaurant_categories = ['4bf58dd8d48988d110941735','55a5a1ebe4b013
909087cbb6','55a5a1ebe4b013909087cb7c',
                                 '55a5a1ebe4b013909087cba7','55a5a1ebe4b013
909087cba1','55a5a1ebe4b013909087cba4',
                                 '55a5a1ebe4b013909087cb95','55a5a1ebe4b013
909087cb89','55a5a1ebe4b013909087cb9b',
                                 '55a5a1ebe4b013909087cb98','55a5a1ebe4b013
909087cbbf','55a5a1ebe4b013909087cb79',
```

```
                                  '55a5a1ebe4b013909087cbb0','55a5a1ebe4b013
909087cbb3','55a5a1ebe4b013909087cb74',
                                  '55a5a1ebe4b013909087cbaa','55a5a1ebe4b013
909087cb83','55a5a1ebe4b013909087cb8c',
                                  '55a5a1ebe4b013909087cb92','55a5a1ebe4b013
909087cb8f','55a5a1ebe4b013909087cb86',
                                  '55a5a1ebe4b013909087cbb9','55a5a1ebe4b013
909087cb7f','55a5a1ebe4b013909087cbbc',
                                  '55a5a1ebe4b013909087cb9e','55a5a1ebe4b013
909087cbc2','55a5a1ebe4b013909087cbad']

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(', Deutschland', '')
    address = address.replace(', Germany', '')
    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, limi
t)
    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 [19]:

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

import pickle

def get_restaurants(lats, lons):
    restaurants = {}
    italian_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 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:
            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=italian_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_italian, x, y)
                if venue_distance<=300:
                    area_restaurants.append(restaurant)
                restaurants[venue_id] = restaurant
```

```
                if is_italian:
                    italian_restaurants[venue_id] = restaurant
        location_restaurants.append(area_restaurants)
        print(' .', end='')
    print(' done.')
    return restaurants, italian_restaurants, location_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 = True
except:
    pass

# If load failed use the Foursquare API to get the data
if not loaded:
    restaurants, italian_restaurants, location_restaurants = get_restaurant
s(latitudes, longitudes)

    # 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 [20]:

```
import numpy as np

print('Total number of restaurants:', len(restaurants))
print('Total number of Italian restaurants:', len(italian_restaurants))
print('Percentage of Italian restaurants: {:.2f}%'.format(len(italian_resta
urants) / len(restaurants) * 100))
print('Average number of restaurants in neighborhood:', np.array([len(r) fo
r r in location_restaurants]).mean())
```

```
Total number of restaurants: 2031
Total number of Italian restaurants: 312
Percentage of Italian restaurants: 15.36%
Average number of restaurants in neighborhood: 4.91208791209
```

In [21]:

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

```
List of all restaurants
-----------------------
('5546072a498e349bf0e737e1', 'Shaam Restaurant', 52.474363806181806, 13.440
120220184326, 'Karl-Marx-Straße 177, 10247 Berlin', 249, False, 394052.3577
5333317, 5814944.355430137)
('4fce25c6e4b0f39fffdd0447', 'Wursterei', 52.5058278495275, 13.333072532529
153, 'Hardenbergplatz 27d, 10623 Berlin', 133, False, 386862.9315917266, 58
18606.191572046)
('57ffdde438fa512462a6b490', 'Einstein Kaffeehaus & Restaurant', 52.516953,
13.385849, 'Unter der Linden 42, 10117 Berlin', 69, False, 390472.374173701
33, 5819762.151308152)
('514316eae4b080a105a5b4f5', 'Allee Bistro', 52.534855836549994, 13.4972411
38618675, 'Berlin', 279, False, 398071.8391866421, 5821590.182515125)
('4c3a05951a38ef3b86079321', 'Louis', 52.474274260971214, 13.44509717979576
5, 'Richardplatz 5, 12055 Berlin', 158, False, 394390.1589999274, 5814927.1
0762019)
('4b62bc3df964a520b4502ae3', 'Kaplan Döner', 52.556723244788124, 13.3736550
87007442, 'Osloer Str. 84, Berlin', 248, False, 389744.70399348286, 5824204
.024383282)
('507eb672e4b032f203a43bee', 'Vino e Cucina', 52.490001421043665, 13.385260
10531851, 'Kreuzbergstr. 77, 10965 Berlin', 218, True, 390365.3747334052, 5
816765.463894998)
('4bbc5dde51b89c744f4f872a', 'Thai Tasty', 52.523448363244846, 13.379427543
998961, 'Luisenstr. 14, 10117 Berlin', 318, False, 390052.8981128248, 58204
94.335557022)
```

```
('51f02bc1498ed5e8bd0f1672', 'Lecker Song', 52.544179118842244, 13.42020477
7148115, 'Schliemannstr. 19, 10437 Berlin', 150, False, 392869.7024230916,
5822738.722407024)
('4c655634e0c4be9a73d18758', 'Marjan Grill', 52.52019090974542, 13.34699250
7015973, 'Stadtbahnbogen 411 (Bartningallee), 10557 Berlin', 91, False, 387
844.2151326508, 5820181.937990252)
...
Total: 2031
```

In [22]:

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

```
List of Italian restaurants
---------------------------
('4b4f6063f964a520e10327e3', 'Salumeria Culinario', 52.526394678482745, 13.
393537136029817, 'Tucholskystr. 34 (Auguststr.), 10117 Berlin', 123, True,
391017.3861974631, 5820800.631260842)
('56d5838e498eda2c7124a8f0', 'Pascarella', 52.53224028238963, 13.3809829056
63293, 'Berlin', 168, True, 390180.3491140888, 5821469.816676741)
('4f1ff655e4b0ec749c54b273', 'Agata Torrisi', 52.53651019364004, 13.3777808
74234496, 'Wöhlertstr. 5, 10115 Berlin', 121, True, 389973.84217276424, 582
1949.599236318)
('551ecd4e498e52f76b5f4310', "Antonello's Cevicheria & Street Food", 52.490
42751718539, 13.390365472982154, 'Nostitzstr. 22, Berlin', 240, True, 39071
3.04239021626, 5816805.116119504)
('4bf2dd126991c9b6629829e9', 'Al Contadino Sotto Le Stelle', 52.52780835209
213, 13.401225263437848, 'Auguststr. 36 (Joachimstr.), 10115 Berlin', 250,
True, 391542.39733332, 5820946.282045525)
('4afc5179f964a5207e2122e3', 'Boccondivino', 52.522249160938536, 13.3843169
85590111, 'Albrechtstr. 18, 10117 Berlin', 161, True, 390381.61197342863, 5
820353.522465905)
('52c863ca498e73da17e59cb9', 'Caligari', 52.475843, 13.423658, 'Kienitzerst
r. 110, Berlin', 307, True, 392937.9001831622, 5815133.146853393)
('507eb672e4b032f203a43bee', 'Vino e Cucina', 52.490001421043665, 13.385260
10531851, 'Kreuzbergstr. 77, 10965 Berlin', 218, True, 390365.3747334052, 5
816765.463894998)
('4b3a4c4df964a520056425e3', 'Fratelli La Bionda', 52.48876483018588, 13.39
7204875946045, 'Bergmannstr. 31 (Heimstr.), 10961 Berlin', 41, True, 391173
.2712361226, 5816609.86209068)
('5454eda7498ee281f036eddf', 'Pastificio Tosatti', 52.5435, 13.419621, 'Sch
liemannstr. 14a, 10437 Berlin', 236, True, 392828.4632061965, 5822664.05614
4068)
...
Total: 312
```

In [23]:

```
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: Mabuhay, Scandic Restaurant
Restaurants around location 102: Solar, THE POST Brasserie & Bar, Ristorant
e Marinelli, Diomira, Mexican, Cucina Italiana, Restaurant Hof zwei, Morélo
s Steakhaus & Cocktailbar
Restaurants around location 103: Paracas, Nobelhart & Schmutzig, Mama Cook,
Trattoria da Vinci, Steakhaus Asador, Tumi, Delhi 6, Deutsche Küche By Kaes
e-koenig.de
Restaurants around location 104:
Restaurants around location 105: Pacifico, food bag 2, TAT Imbiss
Restaurants around location 106: Santa Maria, Die Henne, Zur kleinen Markth
alle, Parantez, Habibi, Maroush, Sol y Sombra, Chez Michel
Restaurants around location 107: La Piadina, 3 Schwestern, Trattoria Marech
iaro, Goldener Hahn, Weltrestaurant Markthalle, Long March Canteen, Olive
Restaurants around location 108: Salumeria Lamuri, Restaurant Richard
Restaurants around location 109: Scheers Schnitzel, Seoulkitchen Korean BBQ
& Sushi, Michelberger Restaurant, Rio Grande, Nano Falafel, Bistro Istanbul
, Asia Bistro
Restaurants around location 110: Dînette, Areti, Kotai Asia, Opera Restaura
nt and Bar, Asia Food Store "We lunch", Ba Qué, La Cesta, Sedici Cucina e D
elicatezze
```

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

In [29]:

```
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.Marker(berlin_center, popup='Alexanderplatz').add_to(map_berlin)
for res in restaurants.values():
    lat = res[2]; lon = res[3]
    is_italian = res[6]
    color = 'red' if is_italian else 'blue'
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True, fill_
color=color, fill_opacity=1).add_to(map_berlin)
map_berlin
```

Out[29]:

Looking good. So now we have all the restaurants in area within few kilometers from Alexanderplatz, and we know which ones are Italian 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 Italian restaurant!

Methodology

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

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

Second step in our analysis will be calculation and exploration of '**restaurant density**' across different areas of Berlin - we will use **heatmaps** to identify a few promising areas close to center with low number of restaurants in general (*and* no Italian 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 Italian 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

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

In [31]:

```
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.a
rray(location_restaurants_count).mean())

df_locations.head(10)
```

```
Average number of restaurants in every area with radius=300m: 4.91208791209
```

Out[31]:

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

In [32]:

```
distances_to_italian_restaurant = []

for area_x, area_y in zip(xs, ys):
    min_distance = 10000
    for res in italian_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_italian_restaurant.append(min_distance)

df_locations['Distance to Italian restaurant'] = distances_to_italian_resta
urant
```

In [33]:

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

Out[33]:

In [35]:

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

```
Average distance to closest Italian restaurant from each area center: 495.2
099580523902
```

OK, so **on average Italian restaurant can be found within ~500m** from every area center candidate. That's fairly close, so we need to filter our areas carefully!

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

In [36]:

```
berlin_boroughs_url = 'https://raw.githubusercontent.com/m-hoerz/berlin-sha
pes/master/berliner-bezirke.geojson'
berlin_boroughs = requests.get(berlin_boroughs_url).json()

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

In [37]:

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

italian_latlons = [[res[2], res[3]] for res in italian_restaurants.values()
]
```

In [38]:

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

map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_berlin) #cartodbpositron car
todbdark_matter
HeatMap(restaurant_latlons).add_to(map_berlin)
folium.Marker(berlin_center).add_to(map_berlin)
folium.Circle(berlin_center, radius=1000, fill=False, color='white').add_to
(map_berlin)
folium.Circle(berlin_center, radius=2000, fill=False, color='white').add_to
(map_berlin)
folium.Circle(berlin_center, radius=3000, fill=False, color='white').add_to
(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[38]:

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

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

In [39]:

```
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_berlin) #cartodbpositron car
todbdark_matter
HeatMap(italian_latlons).add_to(map_berlin)
folium.Marker(berlin_center).add_to(map_berlin)
folium.Circle(berlin_center, radius=1000, fill=False, color='white').add_to
(map_berlin)
folium.Circle(berlin_center, radius=2000, fill=False, color='white').add_to
(map_berlin)
folium.Circle(berlin_center, radius=3000, fill=False, color='white').add_to
(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[39]:

This map is not so 'hot' (Italian restaurants represent a subset of ~15% of all restaurants in Berlin) but it also indicates higher density of existing Italian restaurants directly north and west from Alexanderplatz, 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 Berlin center* - we will move the center of our area of interest and reduce it's size to have a radius of **2.5km**. This places our location candidates mostly in boroughs **Kreuzberg and Friedrichshain** (another potentially interesting borough is **Prenzlauer Berg** with large low restaurant density north-east from city center, however this borough is less interesting to stakeholders as it's mostly residental and less popular with tourists).

Kreuzberg and Friedrichshain

Analysis of popular travel guides and web sites often mention Kreuzberg and Friedrichshain as beautifull, interesting, rich with culture, 'hip' and 'cool' Berlin neighborhoods popular with tourists and loved by Berliners.

"Bold and brazen, Kreuzberg's creative people, places, and spaces might challenge your paradigm." Tags: Nightlife, Artsy, Dining, Trendy, Loved by Berliners, Great Transit (airbnb.com)

"Kreuzberg has long been revered for its diverse cultural life and as a part of Berlin where alternative lifestyles have flourished. Envisioning the glamorous yet gritty nature of Berlin often conjures up scenes from this neighbourhood, where cultures, movements and artistic flare adorn the walls of building and fills the air. Brimming with nightclubs, street food, and art galleries, Kreuzberg is the place to be for Berlin's young and trendy." (theculturetrip.com)

"Imagine an art gallery turned inside out and you'll begin to envision Friedrichshain. Single walls aren't canvases for creative works, entire buildings are canvases. This zealously expressive east Berlin neighborhood forgoes social norms" Tags: Artsy, Nightlife, Trendy, Dining, Touristy, Shopping, Great Transit, Loved by Berliners (airbnb.com)

"As anyone from Kreuzberg will tell you, this district is not just the coolest in Berlin, but the hippest location in the entire universe. Kreuzberg has long been famed for its diverse cultural life, its experimental alternative lifestyles and the powerful spell it exercises on young people from across Germany. In 2001, Kreuzberg and Friedrichshain were merged to form one administrative borough. When it comes to club culture, Friedrichshain is now out in front – with southern Friedrichshain particularly ranked as home to the highest density of clubs in the city." (visitberlin.de)

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

Let's define new, more narrow region of interest, which will include low-restaurant-count parts of Kreuzberg and Friedrichshain closest to Alexanderplatz.

In [40]:

```
roi_x_min = berlin_center_x - 2000
roi_y_max = berlin_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_berlin = folium.Map(location=roi_center, zoom_start=14)
HeatMap(restaurant_latlons).add_to(map_berlin)
folium.Marker(berlin_center).add_to(map_berlin)
folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opaci
ty=0.4).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[40]:

Not bad - this nicely covers all the pockets of low restaurant density in Kreuzberg and Friedrichshain closest to Berlin center.

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 [41]:

```
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.
```

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

In [94]:

```python
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_italian_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, italian_restaurants)
    roi_italian_distances.append(distance)
print('done.')
```

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

In [95]:

```python
# 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_itali
an_distances})

df_roi_locations.head(10)
```

Out[95]:

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 Italian restaurants in radius of 400 meters**.

In [98]:

```
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 restaura
nt']>=400)
print('Locations with no Italian restaurants within 400m:', good_ita_distan
ce.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: 798
Locations with no Italian restaurants within 400m: 380
Locations with both conditions met: 319
```

Let's see how this looks on a map.

In [99]:

```
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_longi
tudes)]

map_berlin = folium.Map(location=roi_center, zoom_start=14)
folium.TileLayer('cartodbpositron').add_to(map_berlin)
HeatMap(restaurant_latlons).add_to(map_berlin)
folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opaci
ty=0.6).add_to(map_berlin)
folium.Marker(berlin_center).add_to(map_berlin)
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_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[99]:

Looking good. We now have a bunch of locations fairly close to Alexanderplatz (mostly in Kreuzberg, Friedrichshain and south-east corner of Mitte boroughs), and we know that each of those locations has no more than two restaurants in radius of 250m, and no Italian restaurant closer than 400m. Any of those locations is a potential candidate for a new Italian restaurant, at least based on nearby competition.

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

In [100]:

```
map_berlin = folium.Map(location=roi_center, zoom_start=14)
```

```
HeatMap(good_locations, radius=25).add_to(map_berlin)
folium.Marker(berlin_center).add_to(map_berlin)
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_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[100]:

Looking good. What we have now is a clear indication of zones with low number of restaurants in vicinity, and *no* Italian 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 [101]:

```
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_cent
ers_]

map_berlin = folium.Map(location=roi_center, zoom_start=14)
folium.TileLayer('cartodbpositron').add_to(map_berlin)
HeatMap(restaurant_latlons).add_to(map_berlin)
folium.Circle(roi_center, radius=2500, color='white', fill=True, fill_opaci
ty=0.4).add_to(map_berlin)
folium.Marker(berlin_center).add_to(map_berlin)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=True, fill_op
acity=0.25).add_to(map_berlin)
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_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[101]:

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

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

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

In [104]:

```
map_berlin = folium.Map(location=roi_center, zoom_start=14)
folium.Marker(berlin_center).add_to(map_berlin)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#00000000', fill=True, fil
l_color='#0066ff', fill_opacity=0.07).add_to(map_berlin)
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_berlin)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=False).add_to
(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[104]:

Let's zoom in on candidate areas in **Kreuzberg**:

In [81]:

```
map_berlin = folium.Map(location=[52.498972, 13.409591], zoom_start=15)
folium.Marker(berlin_center).add_to(map_berlin)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=False).add_to
(map_berlin)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#0000ff00', fill=True, fil
l_color='#0066ff', fill_opacity=0.07).add_to(map_berlin)
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_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
map_berlin
```

Out[81]:

...and candidate areas in **Friedrichshain**:

In [82]:

```
map_berlin = folium.Map(location=[52.516347, 13.428403], zoom_start=15)
folium.Marker(berlin_center).add_to(map_berlin)
for lon, lat in cluster_centers:
    folium.Circle([lat, lon], radius=500, color='green', fill=False).add_to
(map_berlin)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#0000ff00', fill=True, fil
l_color='#0066ff', fill_opacity=0.07).add_to(map_berlin)
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_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojso
n').add_to(map_berlin)
```

```
map_berlin
```

Out[82]:

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

In [86]:

```
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(', Germany', '')
    candidate_area_addresses.append(addr)
    x, y = lonlat_to_xy(lon, lat)
    d = calc_xy_distance(x, y, berlin_center_x, berlin_center_y)
    print('{}{} => {:.1f}km from Alexanderplatz'.format(addr, ' '*(50-len(a
ddr)), d/1000))
```

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

Michaelkirchpl. 15, 10179 Berlin                   => 1.7km from Alexanderp
latz
Lohmühlenstraße 1, 12435 Berlin                    => 3.7km from Alexanderp
latz
Berolinastraße 12, 10178 Berlin                    => 0.5km from Alexanderp
latz
Neuenburger Str. 15, 10969 Berlin                  => 2.7km from Alexanderp
latz
Landsberger Allee 37, 10249 Berlin                 => 2.0km from Alexanderp
latz
Bona-Peiser-Weg 4, 10179 Berlin                    => 1.7km from Alexanderp
latz
Gitschiner Str. 33, 10969 Berlin                   => 2.7km from Alexanderp
latz
Stallschreiberstraße 48, 10969 Berlin              => 1.8km from Alexanderp
latz
An der Ostbahn 5, 10243 Berlin                     => 2.5km from Alexanderp
latz
Hasenheide 81, 10967 Berlin                        => 3.9km from Alexanderp
latz
Ifflandstraße 9, 10179 Berlin                      => 0.9km from Alexanderp
latz
Reichenberger Str. 92, 10999 Berlin                => 3.8km from Alexanderp
latz
Platz der Vereinten Nationen 29, 10249 Berlin      => 1.1km from Alexanderp
latz
```

```
Lobeckstraße 62, 10969 Berlin                    => 2.3km from Alexanderp
latz
Unterwasserstraße 9, 10117 Berlin                => 1.1km from Alexanderp
latz
```

This concludes our analysis. We have created 15 addresses representing centers of zones containing locations with low number of restaurants and no Italian restaurants nearby, all zones being fairly close to city center (all less than 4km from Alexanderplazt, and about half of those less than 2km from Alexanderplatz). 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 Kreuzberg and Friedrichshain boroughs, which we have identified as interesting due to being popular with tourists, fairly close to city center and well connected by public transport.

In [87]:

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

Out[87]:

Results and Discussion

Our analysis shows that although there is a great number of restaurants in Berlin (~2000 in our initial area of interest which was 12x12km around Alexanderplatz), there are pockets of low restaurant density fairly close to city center. Highest concentration of restaurants was detected north and west from Alexanderplatz, so we focused our attention to areas south, south-east and east, corresponding to boroughs Kreuzberg, Friedrichshain and south-east corner of central Mitte borough. Another borough was identified as potentially interesting (Prenzlauer Berg, north-east from Alexanderplatz), but our attention was focused on Kreuzberg and Friedrichshain 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 Alexanderplatz) 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 Italian 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 Italian 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 Berlin center but not crowded with existing restaurants (particularly Italian) - 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 Berlin 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 Italian restaurant. By calculating restaurant density distribution from Foursquare data we have first identified general boroughs that justify further analysis (Kreuzberg and Friedrichshain), 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.