Capstone Final Project - Indian Restaurant

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 **Indian 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 Indian 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 promissing 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 Indian 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]:
```
# Import Libraries

import requests # library to handle requests
import pandas as pd # library for data analsysis
import numpy as np # library to handle data in a vectorized manner


!conda install -c conda-forge geopy --yes
from geopy.geocoders import Nominatim # module to convert an address into lati
tude and longitude values

!conda install -c conda-forge geocoder --yes
import geocoder # to get coordinates

import json # library to handle JSON files

from pandas.io.json import json_normalize # tranform JSON file into a pandas d
ataframe

# Matplotlib and associated plotting modules
import matplotlib.cm as cm
import matplotlib.colors as colors

# import k-means from clustering stage
from sklearn.cluster import KMeans

!conda install -c conda-forge folium=0.5.0 --yes
import folium # plotting library

!conda install -c conda-forge shapely --yes
import shapely.geometry

#!conda install -c conda-forge pyproj --yes

#import pyproj
!conda install -c conda-forge/label/cf202003 pyproj --yes
import pyproj

import math

print("Libraries imported.")
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.4

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: /srv/conda/envs/notebook

  added / updated specs:
    - geopy


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    geographiclib-1.50         |             py_0          34 KB  conda-forge
    geopy-2.0.0                |     pyh9f0ad1d_0          63 KB  conda-forge
    openssl-1.1.1g             |       h516909a_1         2.1 MB  conda-forge
    ------------------------------------------------------------
                                           Total:         2.2 MB

The following NEW packages will be INSTALLED:

  geographiclib      conda-forge/noarch::geographiclib-1.50-py_0
  geopy              conda-forge/noarch::geopy-2.0.0-pyh9f0ad1d_0

The following packages will be UPDATED:

  openssl                                1.1.1g-h516909a_0 --> 1.1.1g-h51690
9a_1



Downloading and Extracting Packages
geographiclib-1.50   | 34 KB     | ##################################### | 10
0%
openssl-1.1.1g       | 2.1 MB    | ##################################### | 10
0%
geopy-2.0.0          | 63 KB     | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: done
```

```
==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.4

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: /srv/conda/envs/notebook

  added / updated specs:
    - geocoder


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    future-0.18.2              |   py37hc8dfbb8_1         712 KB  conda-forge
    geocoder-1.38.1            |             py_1          53 KB  conda-forge
    ratelim-0.1.6              |             py_2           6 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         771 KB

The following NEW packages will be INSTALLED:

  future             conda-forge/linux-64::future-0.18.2-py37hc8dfbb8_1
  geocoder           conda-forge/noarch::geocoder-1.38.1-py_1
  ratelim            conda-forge/noarch::ratelim-0.1.6-py_2



Downloading and Extracting Packages
future-0.18.2        | 712 KB    | ##################################### | 10
0%
geocoder-1.38.1      | 53 KB     | ##################################### | 10
0%
ratelim-0.1.6        | 6 KB      | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible
solve.
Collecting package metadata (repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.4
```

```
Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: /srv/conda/envs/notebook

  added / updated specs:
    - folium=0.5.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    folium-0.5.0               |             py_0          45 KB  conda-forge
    vincent-0.4.4              |             py_1          28 KB  conda-forge
    ------------------------------------------------------------
                                           Total:          73 KB

The following NEW packages will be INSTALLED:

  folium             conda-forge/noarch::folium-0.5.0-py_0
  vincent            conda-forge/noarch::vincent-0.4.4-py_1



Downloading and Extracting Packages
folium-0.5.0         | 45 KB     | ##################################### | 10
0%
vincent-0.4.4        | 28 KB     | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.4

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: /srv/conda/envs/notebook
```

```
  added / updated specs:
    - shapely


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    geos-3.8.1                 |       he1b5a44_0         1.0 MB  conda-forge
    shapely-1.7.0              |   py37hc88ce51_3         435 KB  conda-forge
    ------------------------------------------------------------
                                           Total:         1.4 MB

The following NEW packages will be INSTALLED:

  geos               conda-forge/linux-64::geos-3.8.1-he1b5a44_0
  shapely            conda-forge/linux-64::shapely-1.7.0-py37hc88ce51_3



Downloading and Extracting Packages
shapely-1.7.0        | 435 KB    | ##################################### | 10
0%
geos-3.8.1           | 1.0 MB    | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.8.2
  latest version: 4.8.4

Please update conda by running

    $ conda update -n base conda



## Package Plan ##

  environment location: /srv/conda/envs/notebook

  added / updated specs:
    - pyproj


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2019.11.28 |       hecc5488_0         145 KB  conda-forg
e/label/cf202003
    certifi-2019.11.28         |   py37hc8dfbb8_1         149 KB  conda-forg
```

```
e/label/cf202003
    proj-6.3.1                 |       hc80f0dc_1        10.4 MB  conda-forg
e/label/cf202003
    pyproj-2.6.0               |   py37heba2c01_0         411 KB  conda-forg
e/label/cf202003
    ------------------------------------------------------------
                                           Total:        11.1 MB

The following NEW packages will be INSTALLED:

  proj               conda-forge/label/cf202003/linux-64::proj-6.3.1-hc80f0dc
_1
  pyproj             conda-forge/label/cf202003/linux-64::pyproj-2.6.0-py37he
ba2c01_0

The following packages will be SUPERSEDED by a higher-priority channel:

  ca-certificates    conda-forge::ca-certificates-2020.6.2~ --> conda-forge/l
abel/cf202003::ca-certificates-2019.11.28-hecc5488_0
  certifi            conda-forge::certifi-2020.6.20-py37hc~ --> conda-forge/l
abel/cf202003::certifi-2019.11.28-py37hc8dfbb8_1



Downloading and Extracting Packages
pyproj-2.6.0         | 411 KB    | ##################################### | 10
0%
proj-6.3.1           | 10.4 MB   | ##################################### | 10
0%
certifi-2019.11.28   | 149 KB    | ##################################### | 10
0%
ca-certificates-2019 | 145 KB    | ##################################### | 10
0%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
Libraries imported.
```

In [2]:

```
def get_latlng(neighborhood):
    # initialize your variable to None
    lat_lng_coords = None
    # loop until you get the coordinates
    while(lat_lng_coords is None):
        g = geocoder.geocodefarm('{}'.format(neighborhood))
        lat_lng_coords = g.latlng
    return lat_lng_coords

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

```
Coordinate of Alexanderplatz, Berlin, Germany: [52.521671295186, 13.413330078
1219]
```

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 [3]:
```
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], berli
n_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=13.4133300781219, latitude=52.521671295186
Berlin center UTM X=392348.5035816965, Y=5820245.58778626
Berlin center longitude=13.4133300781219, latitude=52.521671295186
```

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**.

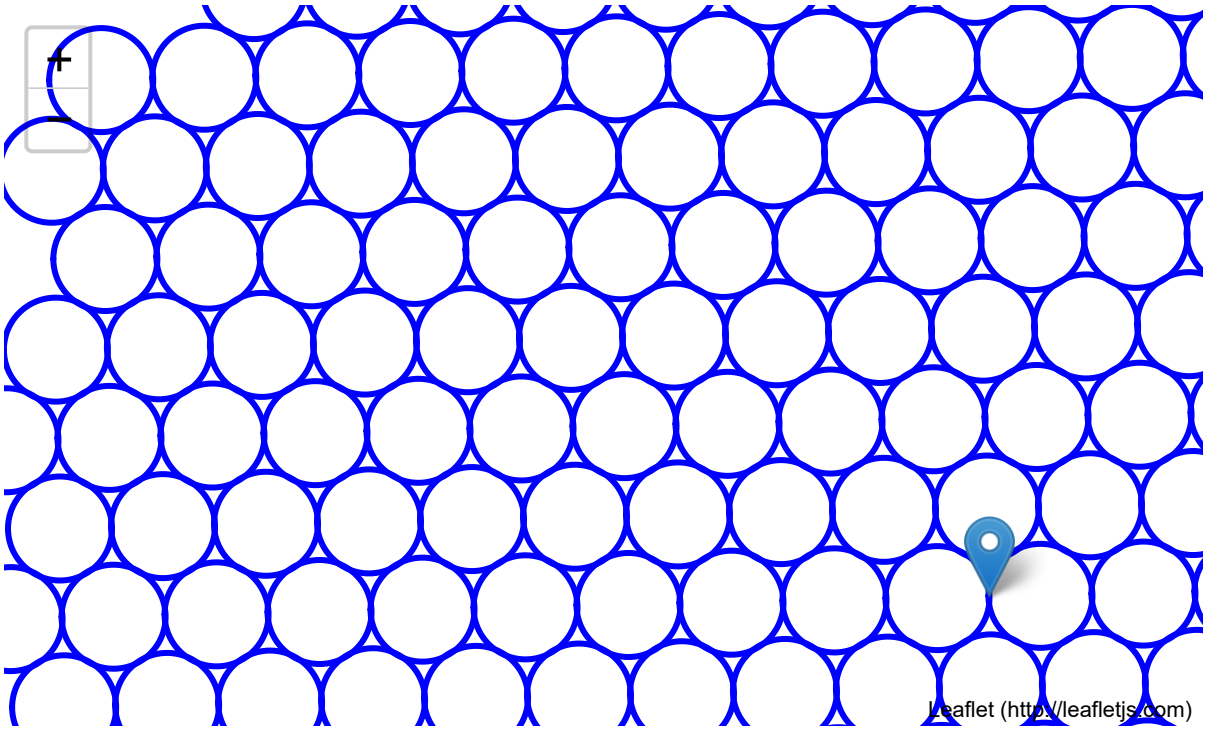
In [4]:
```
berlin_center_x, berlin_center_y = lonlat_to_xy(berlin_center[1], berlin_cente
r[0]) # City center in Cartesian coordinates

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_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.
```

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

In [5]:
```
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, fill_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[5]:



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 [11]:
```
def get_address(coord):
    try:
        location = geocoder.geocodefarm(coord, method='reverse')
        return location.json
    except:
        return None

coord = [] + [berlin_center[0]] + [berlin_center[1]]
addr = get_address(coord)['raw']['formatted_address']
print('Reverse geocoding check')
print('-----------------------')
print('Address of [{}, {}] is: {}'.format(berlin_center[0], berlin_center[1],
str(addr)))
```

```
Reverse geocoding check
-----------------------
Address of [52.521671295186, 13.4133300781219] is: Alexanderplatz 16, 10178 B
erlin, Germany
```

In [17]:
```
print('Obtaining location addresses: ', end='')
addresses = []
for lat, lon in zip(latitudes, longitudes):
    coordinate = [] + [lat] + [lon]
    addre = get_address(coordinate)
    if addre is None:
        addre = 'NO ADDRESS'
    else:
        address = addre['raw']['formatted_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 [21]:
```
addresses[100:120]
```

Out[21]:
```
['Potsdamer Straße 79, 10785 Berlin',
 'Möckernstraße 142, 10963 Berlin',
 'Friedrichstraße 19, 10969 Berlin',
 'Franz-Künstler Straße 6, 10969 Berlin',
 'Prinzenstraße 85E, 10969 Berlin',
 'Prinzenstraße 85E, 10969 Berlin',
 'Muskauer Straße 24, 10997 Berlin',
 'Zeughofstraße 1, 10997 Berlin',
 'Mühlenstraße 8, 10243 Berlin',
 'Lehmbruckstraße 5, 10245 Berlin',
 'Markgrafendamm 24B, 10245 Berlin',
 'Markgrafendamm 24B, 10245 Berlin',
 'Kaskelstraße 38, 10317 Berlin',
 'Rupprechtstraße 13, 10317 Berlin',
 'Rupprechtstraße 13, 10317 Berlin',
 'Budapester Straße 12, 10787 Berlin',
 'Klingelhöferstraße 10, 10785 Berlin',
 'Reichpietschufer 76, 10785 Berlin',
 'Reichpietschufer 76, 10785 Berlin',
 'Stresemannstraße 119, 10963 Berlin']
```

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

In [22]:
```
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[22]:

	Address	Latitude	Longitude	X	Y	Distance from center
0	A100, 12101 Berlin	52.469947	13.388690	390548.503582	5.814530e+06	5992.495307
1	Berlin, Berlin	52.470067	13.397520	391148.503582	5.814530e+06	5840.376700
2	Berlin, Berlin	52.470186	13.406349	391748.503582	5.814530e+06	5747.173218
3	Oderstraße 174, 12101 Berlin	52.470305	13.415178	392348.503582	5.814530e+06	5715.767665
4	Warthestraße 23, 12051 Berlin	52.470423	13.424008	392948.503582	5.814530e+06	5747.173218
5	Schierker Straße 19, 12053 Berlin	52.470540	13.432837	393548.503582	5.814530e+06	5840.376700
6	Karl-Marx Straße 213, 12055 Berlin	52.470657	13.441667	394148.503582	5.814530e+06	5992.495307
7	Hessenring 36, 12101 Berlin	52.474436	13.375275	389648.503582	5.815049e+06	5855.766389
8	Kleineweg 133, 12101 Berlin	52.474557	13.384105	390248.503582	5.815049e+06	5604.462508
9	Berlin, Berlin	52.474677	13.392935	390848.503582	5.815049e+06	5408.326913

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

In [24]:
```
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 'Indian restaurant' category, as we need info on Indian restaurants in the neighborhood.

Foursquare credentials are defined in hidden cell bellow.

In [28]:
```
foursquare_client_id = 'CIU1H3BNY2FI1YUCVBYBX2CQBQIDDDU5ZOM2Z2QGILVXHHZI' # yo
ur Foursquare ID
foursquare_client_secret = '54H4BXM00HEVG2MQW1OBTFASRABYWAE10PCW4QPO1TPBP23O'
# your Foursquare Secret
VERSION = '20180604'
```

In [29]:
```
# Category IDs corresponding to Indian restaurants were taken from Foursquare
 web site (https://developer.foursquare.com/docs/resources/categories):

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

indian_restaurant_categories = ['4bf58dd8d48988d10f941735','54135bf5e4b08f3d24
29dfe5','54135bf5e4b08f3d2429dff3',
                                '54135bf5e4b08f3d2429dff5','54135bf5e4b08f3d2
429dfe2','54135bf5e4b08f3d2429dff2',
                                '54135bf5e4b08f3d2429dfe1','54135bf5e4b08f3d2
429dfe3','54135bf5e4b08f3d2429dfe8',
                                '54135bf5e4b08f3d2429dfe9','54135bf5e4b08f3d2
429dfe6','54135bf5e4b08f3d2429dfdf',
                                '54135bf5e4b08f3d2429dfe4','54135bf5e4b08f3d2
429dfe7','54135bf5e4b08f3d2429dfea',
                                '54135bf5e4b08f3d2429dfeb','54135bf5e4b08f3d2
429dfed','54135bf5e4b08f3d2429dfee',
                                '54135bf5e4b08f3d2429dff4','54135bf5e4b08f3d2
429dfe0','54135bf5e4b08f3d2429dfdd',
                                '54135bf5e4b08f3d2429dff6','54135bf5e4b08f3d2
429dfef','54135bf5e4b08f3d2429dff0',
                                '54135bf5e4b08f3d2429dff1','54135bf5e4b08f3d2
429dfde','54135bf5e4b08f3d2429dfec']

def is_restaurant(categories, specific_filter=None):
    restaurant_words = ['restaurant', 'diner', 'tandoor', 'spicy']
    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, rad
ius=500, limit=100):
    version = '20180724'
    url = 'https://api.foursquare.com/v2/venues/explore?client_id={}&client_se
cret={}&v={}&ll={},{}&categoryId={}&radius={}&limit={}'.format(
```

```
        client_id, client_secret, version, lat, lon, category, radius, limit)
    try:
        results = requests.get(url).json()['response']['groups'][0]['items']
        venues = [(item['venue']['id'],
                   item['venue']['name'],
                   get_categories(item['venue']['categories']),
                   (item['venue']['location']['lat'], item['venue']['location'
]['lng']),
                   format_address(item['venue']['location']),
                   item['venue']['location']['distance']) for item in results]
    except:
        venues = []
    return venues
```

In [30]:
```
# Let's now go over our neighborhood locations and get nearby restaurants; w
e'll also maintain a dictionary of all found restaurants and all found indian
 restaurants

import pickle

def get_restaurants(lats, lons):
    restaurants = {}
    indian_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 d
on't miss any restaurant (we're using dictionaries to remove any duplicates re
sulting 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_indian = is_restaurant(venue_categories, specific_filte
r=indian_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_lat
lon[1], venue_address, venue_distance, is_indian, x, y)
                if venue_distance<=300:
                    area_restaurants.append(restaurant)
                restaurants[venue_id] = restaurant
                if is_indian:
                    indian_restaurants[venue_id] = restaurant
        location_restaurants.append(area_restaurants)
        print(' .', end='')
    print(' done.')
    return restaurants, indian_restaurants, location_restaurants

# Try to load from local file system in case we did this before
restaurants = {}
indian_restaurants = {}
location_restaurants = []
loaded = False
try:
    with open('restaurants_350.pkl', 'rb') as f:
        restaurants = pickle.load(f)
    with open('indian_restaurants_350.pkl', 'rb') as f:
        indian_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, indian_restaurants, location_restaurants = get_restaurants(la
titudes, longitudes)

    # Let's persists this in local file system
    with open('restaurants_350.pkl', 'wb') as f:
        pickle.dump(restaurants, f)
    with open('indian_restaurants_350.pkl', 'wb') as f:
        pickle.dump(indian_restaurants, f)
    with open('location_restaurants_350.pkl', 'wb') as f:
        pickle.dump(location_restaurants, f)
```

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

In [31]:
```
import numpy as np

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

```
Total number of restaurants: 2009
Total number of Indian restaurants: 79
Percentage of Indian restaurants: 3.93%
Average number of restaurants in neighborhood: 4.802197802197802
```

In [32]:
```
print('List of all restaurants')
print('----------------------')
for r in list(restaurants.values())[:10]:
    print(r)
print('...')
print('Total:', len(restaurants))
```

```
List of all restaurants
----------------------
('529c75dc498e5f28c08db637', 'Orient Food', 52.468418, 13.385631, 'Tempelhofe
r damm 124, 12099 Berlin', 268, False, 390336.8958508584, 5814364.380615842)
('52b5d5d0498edc50653df469', 'Yasaka-Sushi', 52.46877834428962, 13.3854589712
42336, 'Tempelhofer Damm 124, 12099 Berlin', 254, False, 390326.1066713873, 5
814404.719417979)
('4f53b76de4b0754d59050702', 'Yoko Sushi', 52.46798, 13.38516391, 'Tempelhofe
r Damm 128, 12099 Berlin', 324, False, 390304.07978736167, 5814316.375297558
5)
('53bfbf97498e767d04eba262', 'My Asia', 52.46863425153149, 13.38513817439509,
'Tempelhofer Damm 122 (Ringahnstraße), 12103 Berlin', 281, False, 390303.9585
045325, 5814389.180474727)
('4e68e3beae60186280aeda86', 'Mahatma', 52.47215311646854, 13.38533023091156,
'Tempelhofer Damm 102 (Manfred-von-Richthofen-Str.), 12101 Berlin', 280, Tru
e, 390325.75282294105, 5814780.258426609)
('53820ddf498e1b7adfc5c1d0', 'Korean BBQ', 52.46780776977539, 13.399324417114
258, 'Berlin', 279, False, 391265.5148455473, 5814275.810391908)
('50799684e4b0772056912f11', 'Tsukushiya', 52.46777229048404, 13.418218701427
607, 'Dresdener Str 16 (Oranienplatz), 10999 Berlin', 349, False, 392548.8398
433876, 5814243.591668439)
('51fba1b9498e7e29656ca673', 'Warthe-Mal', 52.469079958985226, 13.42228820095
4803, 'Warthestr.46, Berlin', 189, False, 392828.4416519639, 5814382.9862255
6)
('50ce405fe4b0aceb60754dbc', 'Merakli Köfteci', 52.471037770421816, 13.429444
253096332, 'Hermannstr 174, 12051 Berlin', 236, False, 393319.24466957763, 58
14590.143150846)
('51474e67e4b02de9576b8ec6', 'STAR Gemüse-Schawarma & Falafel', 52.4703494828
974, 13.429663296488343, 'Berlin', 216, False, 393332.45803737885, 5814513.26
7629099)
...
Total: 2009
```

In [33]:
```
print('List of Indian restaurants')
print('--------------------------')
for r in list(indian_restaurants.values())[:10]:
    print(r)
print('...')
print('Total:', len(indian_restaurants))
```

```
List of Indian restaurants
--------------------------
('4e68e3beae60186280aeda86', 'Mahatma', 52.47215311646854, 13.38533023091156,
'Tempelhofer Damm 102 (Manfred-von-Richthofen-Str.), 12101 Berlin', 280, Tru
e, 390325.75282294105, 5814780.258426609)
('4c422ab0520fa593d9cfcaac', 'Shaan', 52.47402032561738, 13.447161330814634,
'Richardplatz 20, 12055 Berlin', 173, True, 394529.7412089095, 5814895.848594
16)
('5d0e62f08be3b00023c2d833', 'Babu', 52.478177, 13.448091, 'Treptower Straße
95 (Sonnenallee), 12059 Berlin', 257, True, 394602.81495960173, 5815356.80255
9959)
('4e5a798faeb7d78d9f84bb10', 'Raamson', 52.485502, 13.362842700176621, 'Kolon
nenstr. 48/49, 10829 Berlin', 320, True, 388832.0797755071, 5816299.30619891
1)
('4dd80c3b8877f1150ff96fab', 'India Restaurant', 52.48603472205474, 13.450007
521844405, '10117 Berlin', 327, True, 394751.72338554746, 5816227.955295741)
('51841240e4b0e28130c3b9a3', 'ammAmma', 52.48511790858147, 13.35231297766884
5, 'Eisenacher Str. 59, 10823 Berlin', 337, True, 388116.15577082976, 581627
2.84816847)
('51e7ed71498e0888433d044e', 'Mela', 52.487047190504796, 13.358316415987348,
'Crellestr. 46, 10827 Berlin', 255, True, 388528.66002463724, 5816478.1397560
8)
('4db5588d1e7248d135b1b769', 'Swera', 52.48988058993677, 13.389086368461768,
'Bergmannstr. 103, 10961 Berlin', 157, True, 390624.84580401145, 5816746.2229
44492)
('4b83cb0bf964a520be1031e3', 'India', 52.489769031789045, 13.389660619461798,
'Bergmannstr. 100, 10961 Berlin', 169, True, 390663.55600623094, 5816732.9457
69632)
('4dfb390018386e743d97353a', 'Anantha Raja', 52.49108042390628, 13.3942825506
3214, 'Zossener Str. 16, 10961 Berlin', 288, True, 390980.591173515, 5816871.
812256747)
...
Total: 79
```

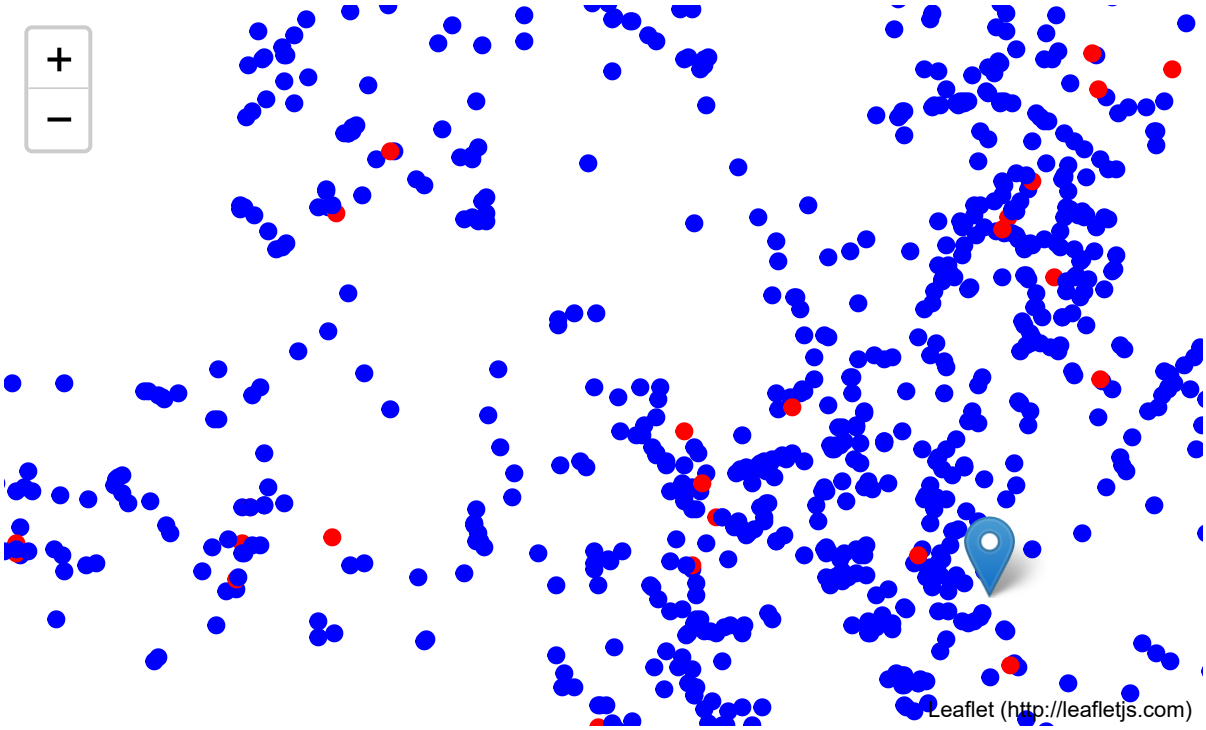
In [34]:
```
print('Restaurants around location')
print('---------------------------')
for i in range(100, 110):
    rs = location_restaurants[i][:8]
    names = ', '.join([r[1] for r in rs])
    print('Restaurants around location {}: {}'.format(i+1, names))
```

```
Restaurants around location
---------------------------
Restaurants around location 101: Mabuhay, Scandic Restaurant, Tischlein deck
dich
Restaurants around location 102: Solar, Mirami - Sushi & Asia fusion cuisin,
Layla, Ristorante Marinelli, Diomira, Cucina Italiana, Mexican, Restaurant Ho
f zwei
Restaurants around location 103: NaNum, Mama Cook, Café Nullpunkt, Paracas, O
rient Food, Delhi 6, Tumi, Dai An Asia
Restaurants around location 104:
Restaurants around location 105: Shishi, Pacifico
Restaurants around location 106: Die Henne, Santa Maria, Zur kleinen Markthal
le, Cevichería, Parantez, Habibi, Maroush, Chikogi
Restaurants around location 107: La Piadina, 3 Schwestern, Der Goldene Hahn,
Weltrestaurant Markthalle, Long March Canteen, Olive
Restaurants around location 108: Salumeria Lamuri, Restaurant Richard
Restaurants around location 109: Vincent Vegan, Scheers Schnitzel, Michelberg
er Restaurant, Seoulkitchen Korean BBQ & Sushi, Tony Roma's, Riogrande, Asia
Bistro, Nano Falafel
Restaurants around location 110: Cantalupo & Resteino, Ba Qué, Kotai Asia, Ar
eti, Opera Restaurant and Bar, Asia Food Store "We lunch", La Cesta, Sedici C
ucina e Delicatezze
```

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

In [35]:
```
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_indian = res[6]
    color = 'red' if is_indian else 'blue'
    folium.CircleMarker([lat, lon], radius=3, color=color, fill=True, fill_col
or=color, fill_opacity=1).add_to(map_berlin)
map_berlin
```

Out[35]:



Looking good. So now we have all the restaurants in area within few kilometers from Alexanderplatz, and we know which ones are Indian 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 Indian 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 Indian 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 Indian 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 Indian 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 Indian 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 [36]:
```
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.arra
y(location_restaurants_count).mean())

df_locations.head(10)
```

```
Average number of restaurants in every area with radius=300m: 4.8021978021978
02
```

Out[36]:

	Address	Latitude	Longitude	X	Y	Distance from center	Restaurants in area
0	A100, 12101 Berlin	52.469947	13.388690	390548.503582	5.814530e+06	5992.495307	3
1	Berlin, Berlin	52.470067	13.397520	391148.503582	5.814530e+06	5840.376700	1
2	Berlin, Berlin	52.470186	13.406349	391748.503582	5.814530e+06	5747.173218	0
3	Oderstraße 174, 12101 Berlin	52.470305	13.415178	392348.503582	5.814530e+06	5715.767665	0
4	Warthestraße 23, 12051 Berlin	52.470423	13.424008	392948.503582	5.814530e+06	5747.173218	1
5	Schierker Straße 19, 12053 Berlin	52.470540	13.432837	393548.503582	5.814530e+06	5840.376700	6
6	Karl-Marx Straße 213, 12055 Berlin	52.470657	13.441667	394148.503582	5.814530e+06	5992.495307	6
7	Hessenring 36, 12101 Berlin	52.474436	13.375275	389648.503582	5.815049e+06	5855.766389	0
8	Kleineweg 133, 12101 Berlin	52.474557	13.384105	390248.503582	5.815049e+06	5604.462508	1
9	Berlin, Berlin	52.474677	13.392935	390848.503582	5.815049e+06	5408.326913	0

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

In [37]:
```
distances_to_indian_restaurant = []

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

df_locations['Distance to Indian restaurant'] = distances_to_indian_restaurant
```

In [38]:
```
df_locations.head(10)
```

Out[38]:

	Address	Latitude	Longitude	X	Y	Distance from center	Restaurants in area	
0	A100, 12101 Berlin	52.469947	13.388690	390548.503582	5.814530e+06	5992.495307	3	
1	Berlin, Berlin	52.470067	13.397520	391148.503582	5.814530e+06	5840.376700	1	
2	Berlin, Berlin	52.470186	13.406349	391748.503582	5.814530e+06	5747.173218	0	1
3	Oderstraße 174, 12101 Berlin	52.470305	13.415178	392348.503582	5.814530e+06	5715.767665	0	2
4	Warthestraße 23, 12051 Berlin	52.470423	13.424008	392948.503582	5.814530e+06	5747.173218	1	1
5	Schierker Straße 19, 12053 Berlin	52.470540	13.432837	393548.503582	5.814530e+06	5840.376700	6	1
6	Karl-Marx Straße 213, 12055 Berlin	52.470657	13.441667	394148.503582	5.814530e+06	5992.495307	6	
7	Hessenring 36, 12101 Berlin	52.474436	13.375275	389648.503582	5.815049e+06	5855.766389	0	
8	Kleineweg 133, 12101 Berlin	52.474557	13.384105	390248.503582	5.815049e+06	5604.462508	1	
9	Berlin, Berlin	52.474677	13.392935	390848.503582	5.815049e+06	5408.326913	0	

In [39]:
```
print('Average distance to closest Indian restaurant from each area center:',
df_locations['Distance to Indian restaurant'].mean())
```

```
Average distance to closest Indian restaurant from each area center: 816.6883
659779518
```

OK, so **on average Indian restaurant can be found within ~816m** from every area center candidate. That's fairly close, so we need to filter our areas carefully!

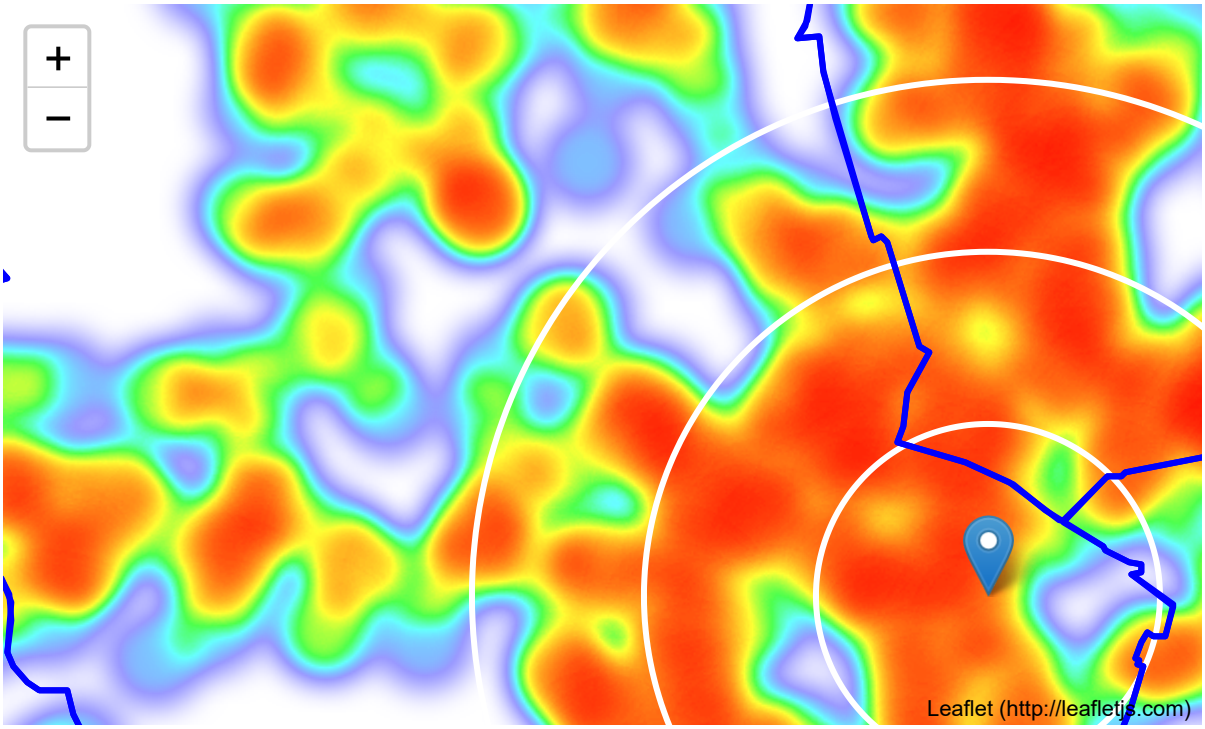
Let's crete a map showing **heatmap / density of restaurants** and try to extract some meaningfull info from that. Also, let's show **borders of Berlin boroughs** on our map and a few circles indicating distance of 1km, 2km and 3km from Alexanderplatz.

In [40]:
```
berlin_boroughs_url = 'https://raw.githubusercontent.com/m-hoerz/berlin-shapes/master/berliner-bezirke.geojson'
berlin_boroughs = requests.get(berlin_boroughs_url).json()

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

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

indian_latlons = [[res[2], res[3]] for res in indian_restaurants.values()]
```

In [42]:
```
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 cartod
bdark_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(ma
p_berlin)
folium.Circle(berlin_center, radius=2000, fill=False, color='white').add_to(ma
p_berlin)
folium.Circle(berlin_center, radius=3000, fill=False, color='white').add_to(ma
p_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[42]:

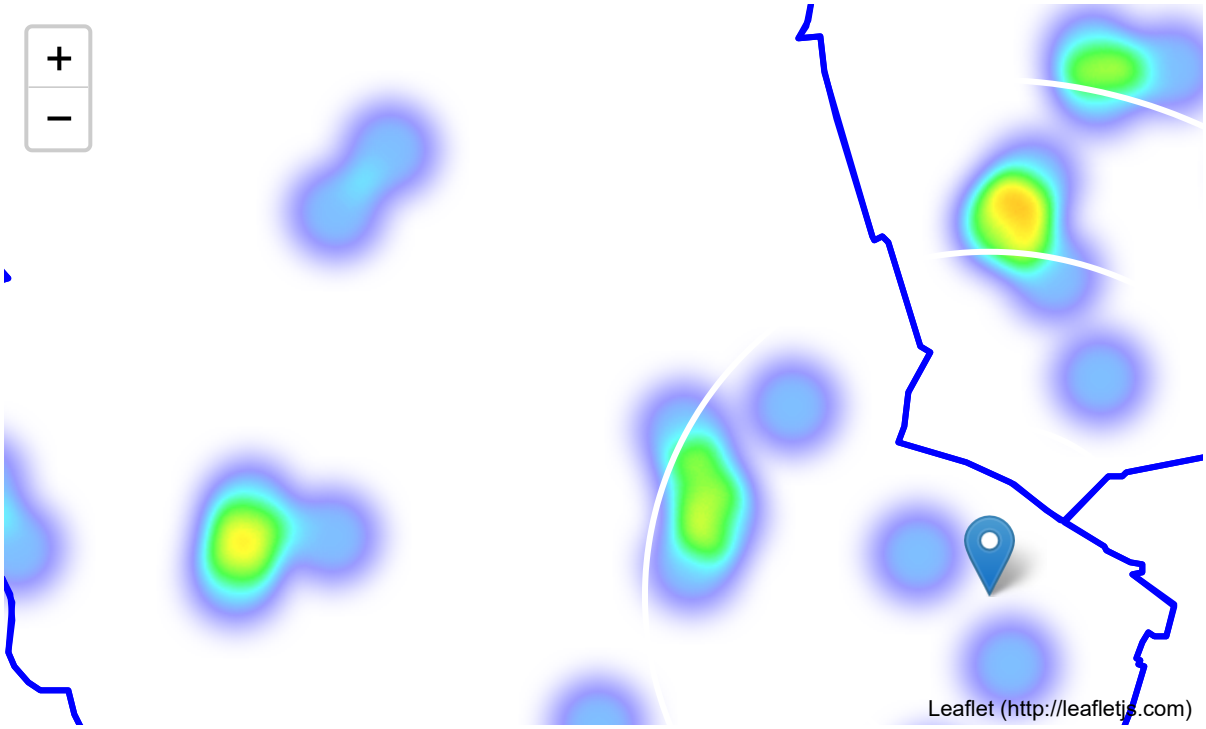


Looks like a few pockets of low restaurant density closest to city center can be found **south, south-east and east from Alexanderplatz**.

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

In [43]:
```
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_berlin) #cartodbpositron cartod
bdark_matter
HeatMap(indian_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(ma
p_berlin)
folium.Circle(berlin_center, radius=2000, fill=False, color='white').add_to(ma
p_berlin)
folium.Circle(berlin_center, radius=3000, fill=False, color='white').add_to(ma
p_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[43]:



This map is not so 'hot' (Indian restaurants represent a subset of ~15% of all restaurants in Berlin) but it also indicates higher density of existing Indian restaurants directly north and west from Alexanderplatz, with closest pockets of **low Indian 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)

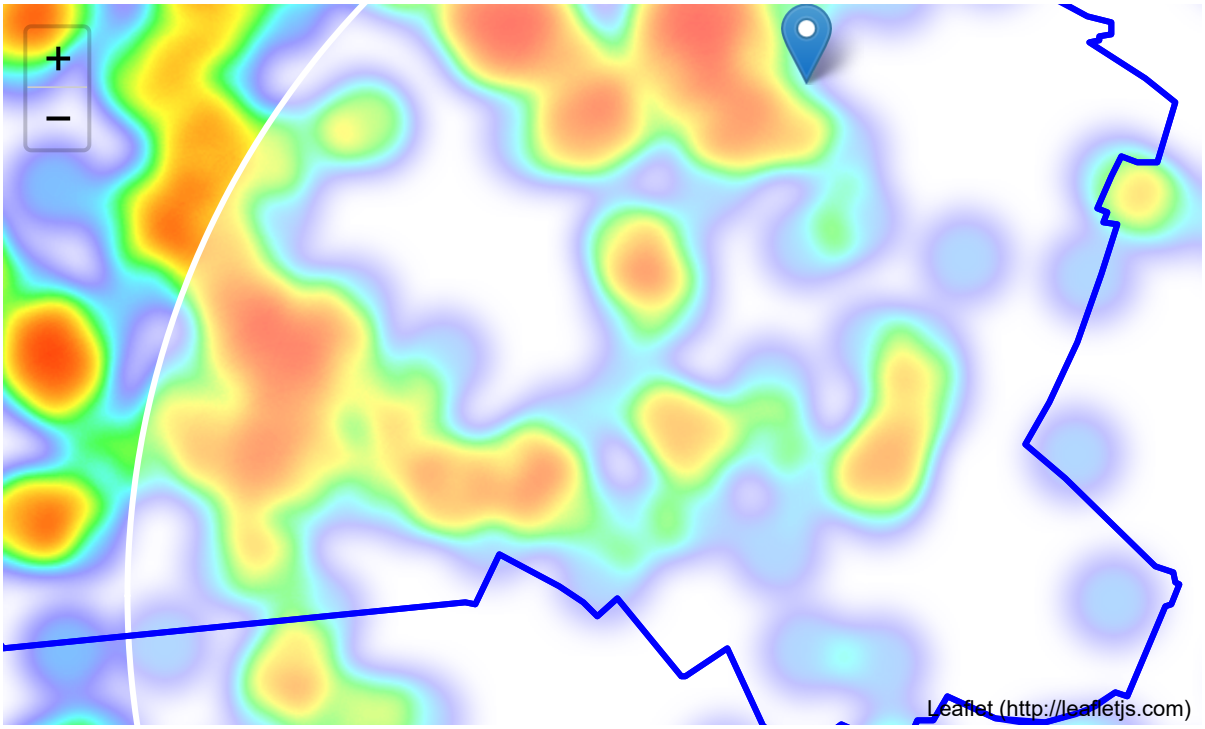
"As anyone from Kreuzberg will tell you, this district is not just the coolest in Berlin, but the hippest location in the entire universe. Kreuzberg has long been famed for its diverse cultural life, its experimental alternative lifestyles and the powerful spell it exercises on young people from across 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 [44]:
```
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_opacity=
0.4).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[44]:



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 [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.
```

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 Indian restaurant**.

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_indian_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, indian_restaurants)
    roi_indian_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 Indian restaurant':roi_indian_di
stances})

df_roi_locations.head(10)
```

Out[47]:

	Latitude	Longitude	X	Y	Restaurants nearby	Distance to Indian restaurant
0	52.485813	13.421249	392798.503582	5.816246e+06	6	476.494060
1	52.485833	13.422721	392898.503582	5.816246e+06	7	395.376008
2	52.486483	13.413124	392248.503582	5.816332e+06	0	481.775832
3	52.486503	13.414596	392348.503582	5.816332e+06	0	548.475319
4	52.486522	13.416068	392448.503582	5.816332e+06	0	624.133319
5	52.486542	13.417540	392548.503582	5.816332e+06	3	587.837226
6	52.486562	13.419012	392648.503582	5.816332e+06	5	556.979723
7	52.486581	13.420485	392748.503582	5.816332e+06	6	483.264420
8	52.486601	13.421957	392848.503582	5.816332e+06	7	390.922113
9	52.486621	13.423429	392948.503582	5.816332e+06	10	303.472728

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 Indian restaurants in radius of 400 meters**.

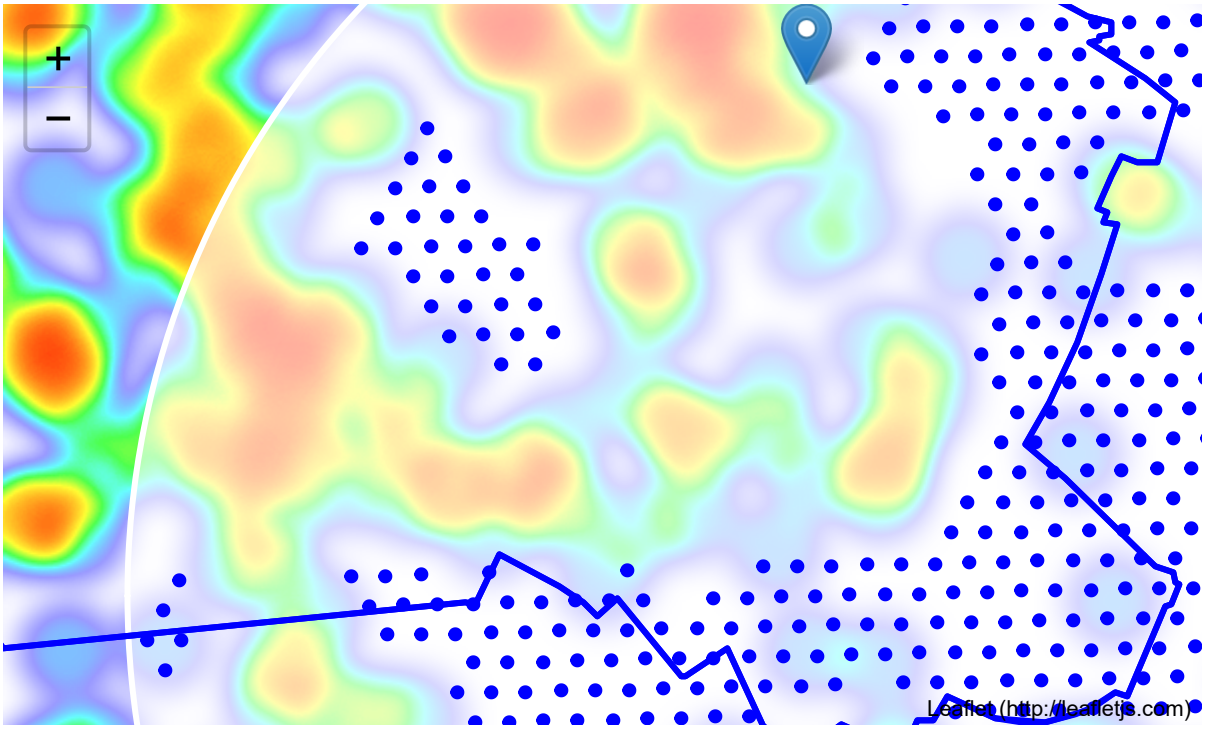
In [48]:
```
good_res_count = np.array((df_roi_locations['Restaurants nearby']<=2))
print('Locations with no more than two restaurants nearby:', good_res_count.su
m())

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

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: 818
Locations with no Indian restaurants within 400m: 1514
Locations with both conditions met: 742
```

Let's see how this looks on a map.

In [49]:
```
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_longitud
es)]

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_opacity=
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_co
lor='blue', fill_opacity=1).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[49]:

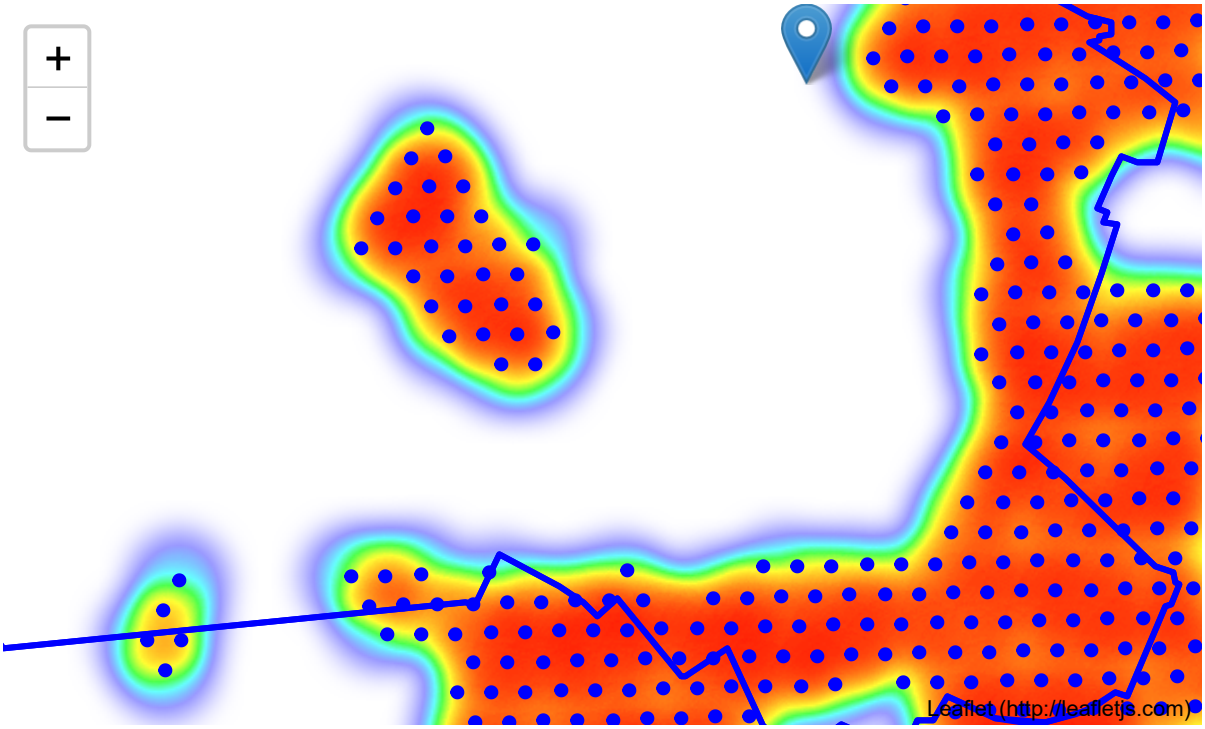


Looking good. We now have a bunch of locations fairly close to Alexanderplatz (mostly in Kreuzberg, Friedrichshain and south-east corner of Mitte boroughs), and we know that each of those locations has no more than two restaurants in radius of 250m, and no Indian restaurant closer than 400m. Any of those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

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

In [50]:
```
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_co
lor='blue', fill_opacity=1).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[50]:

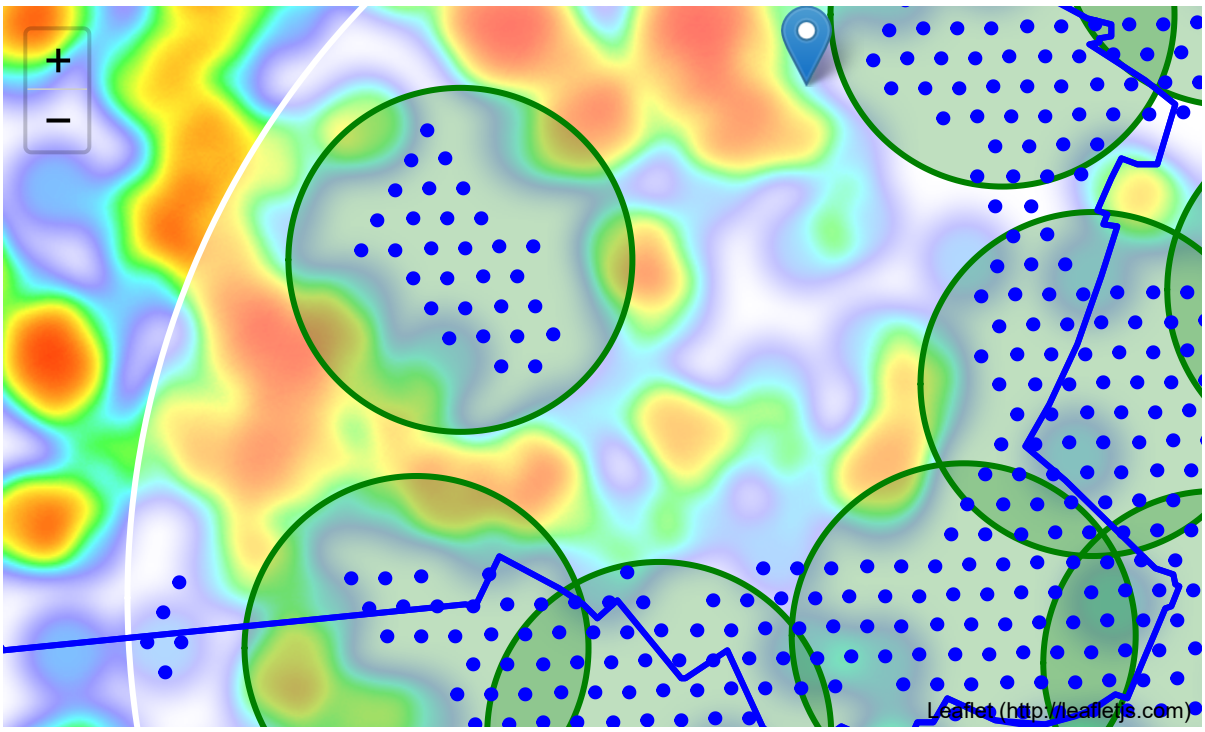


Looking good. What we have now is a clear indication of zones with low number of restaurants in vicinity, and *no* Indian restaurants at all nearby.

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 [51]:
```
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_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_opacity=
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_opaci
ty=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_co
lor='blue', fill_opacity=1).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[51]:



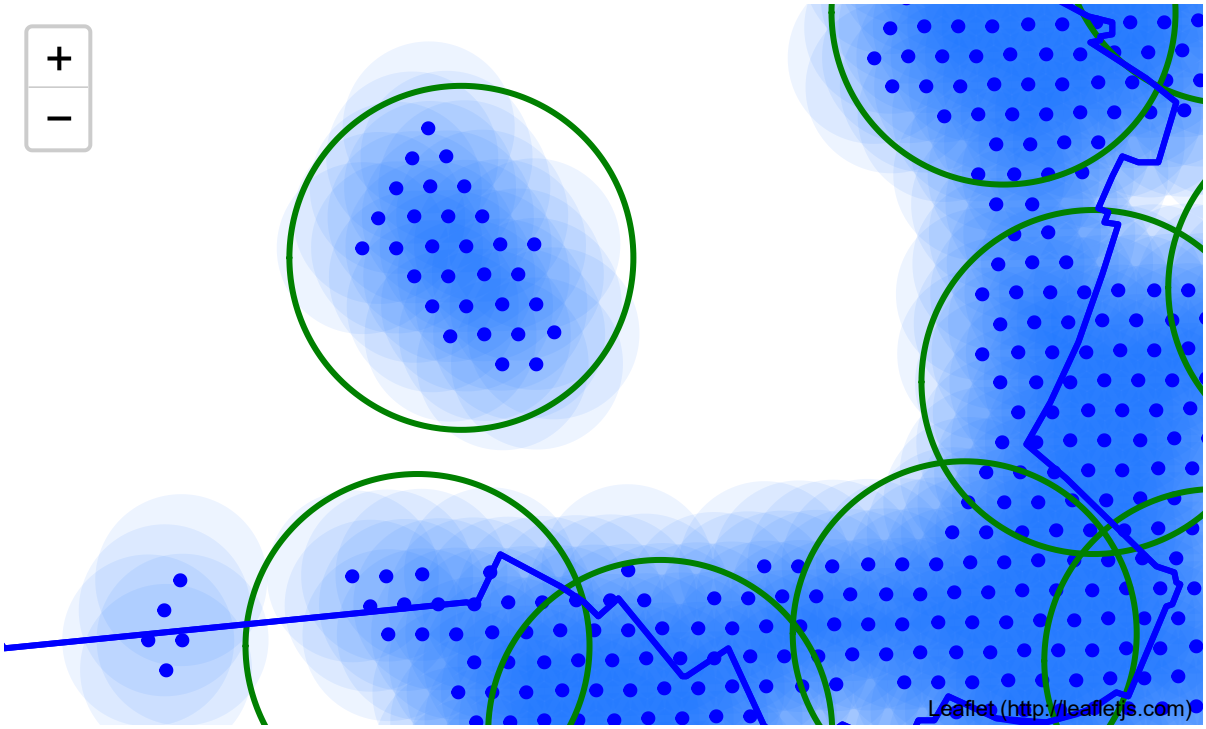
Not bad - our clusters represent groupings of most of the candidate locations and cluster centers are placed nicely in the middle of the zones 'rich' with location candidates.

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 [52]:
```
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, fill_c
olor='#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_co
lor='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(ma
p_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

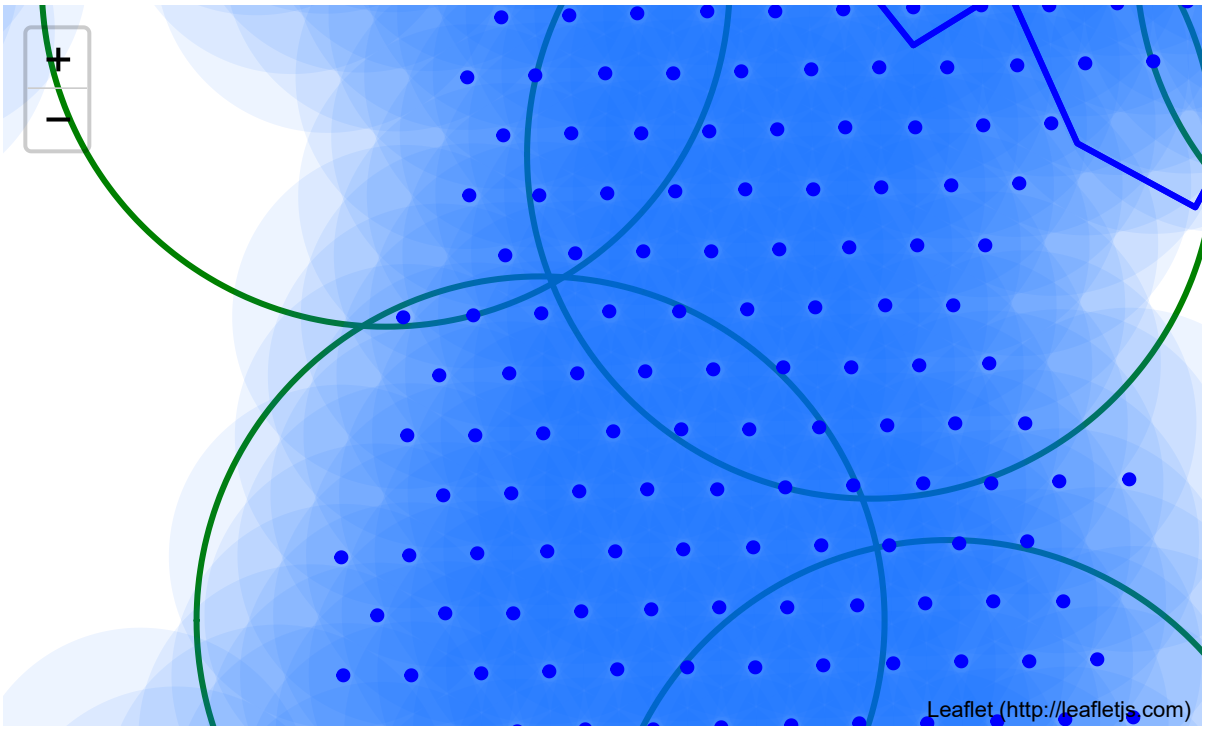
Out[52]:



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

In [53]:
```
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(ma
p_berlin)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#0000ff00', fill=True, fill_c
olor='#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_co
lor='blue', fill_opacity=1).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[53]:



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

In [54]:
```
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(ma
p_berlin)
for lat, lon in zip(good_latitudes, good_longitudes):
    folium.Circle([lat, lon], radius=250, color='#0000ff00', fill=True, fill_c
olor='#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_co
lor='blue', fill_opacity=1).add_to(map_berlin)
folium.GeoJson(berlin_boroughs, style_function=boroughs_style, name='geojson')
.add_to(map_berlin)
map_berlin
```

Out[54]:

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

In [ ]:
```
candidate_area_addresses = []
print('============================================================')
print('Addresses of centers of areas recommended for further analysis')
print('============================================================\n')
for lon, lat in cluster_centers:
    coord = [] + [lat] + [lon]
    addr = get_address(coord).json['raw']['formatted_address'].replace(', Germ
any', '')
    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(addr
)), d/1000))
```

This concludes our analysis. We have created 15 addresses representing centers of zones containing locations with low number of restaurants and no Indian 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, fill_color='#0066ff', fill_opacity=0.05).add_to(map_berlin)
map_berlin
```

Out[87]: