

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

Applied Data Science Capstone by IBM/Coursera

### Table of contents

- [Introduction: Business Problem](#)
- [Data](#)
- [Methodology](#)
- [Analysis](#)
- [Results and Discussion](#)
- [Conclusion](#)

## ▼ 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.

Saving...



generate a few most promising neighborhoods based on this criteria. Advantages of each area will possible final location can be chosen by stakeholders.

## ▼ Data

Based on definition of our problem, factors that will influence our decision 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 approx. 12x12 kilometers 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.

# The code was removed by Watson Studio for sharing.

```
import requests
```

```
def get_coordinates(api_key, address, verbose=False):
```

Saving...



```
url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&address={}'.format(api_key, address)
response = requests.get(url).json()
```

```
    if verbose:
```

```
        print('Google Maps API JSON result =>', response)
```

```
    results = response['results']
```

```
    geographical_data = results[0]['geometry']['location'] # get geographical coordinates
```

```


lat = geographical_data['lat']
lon = geographical_data['lng']
return [lat, lon]
except:
    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, equally 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).

```

#!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_latlon = pyproj.Proj(proj=projlong, 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], berlin_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=52.521918399999997

```

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 its neighbors**.

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

```

k = math.sqrt(3) / 2 # Vertical offset for hexagonal grid cells
x_min = berlin_center_x - 6000

```

Saving...



```
t(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:

```

#!pip install folium

```

```

import folium

```

```

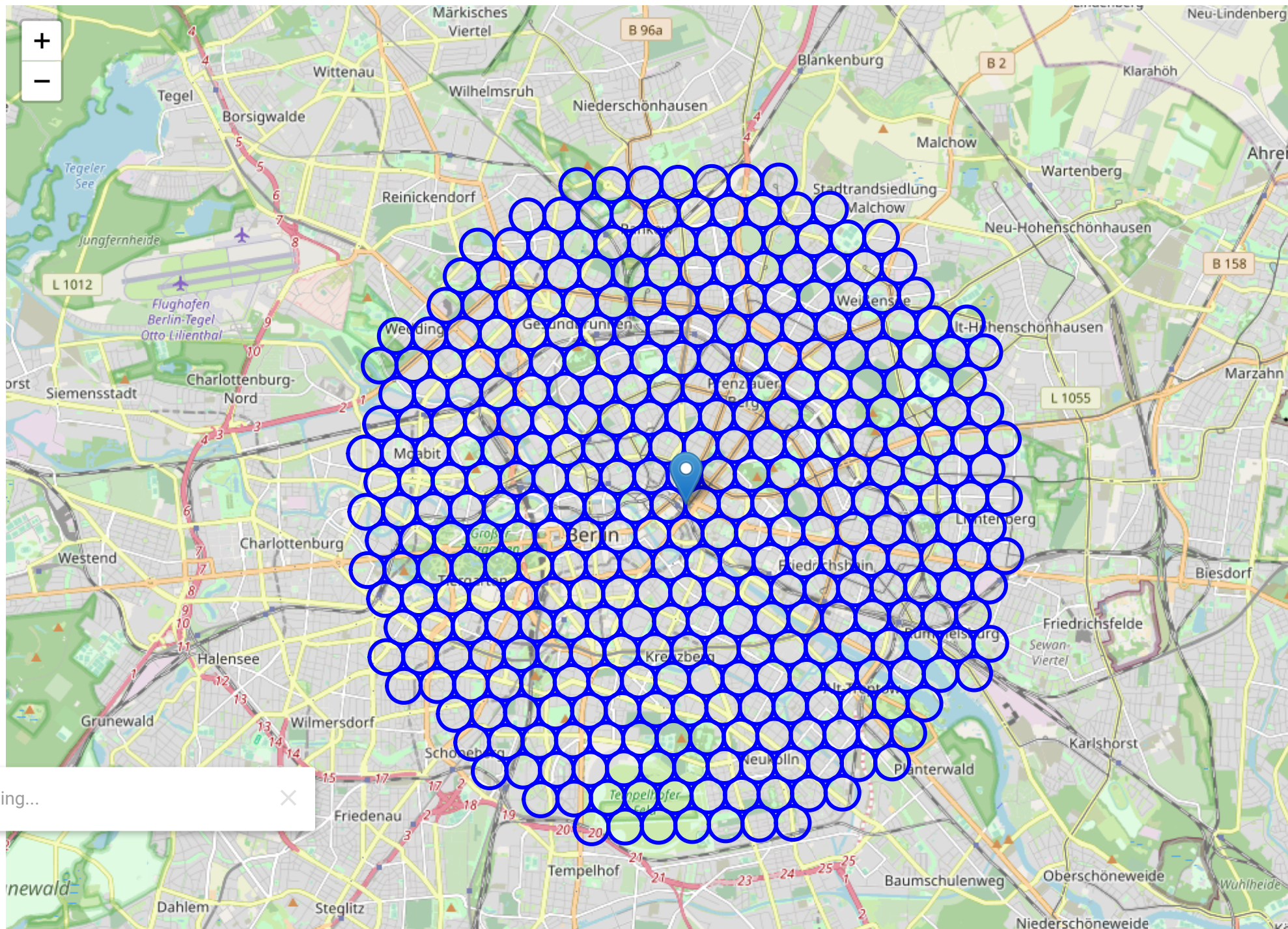
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.Marker(berlin_center, popup='Alexanderplatz').add_to(map_berlin)
for lat, lon in zip(latitudes, longitudes):
    folium.Marker([lat, lon], radius=2, color='blue', fill=True, fill_color='blue', fill_opacity=1).add_to(map_berlin)
  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

```

Saving...







Saving...




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.

```
def get_address(api_key, latitude, longitude, verbose=False):
    try:
        url = 'https://maps.googleapis.com/maps/api/geocode/json?key={}&latlng={},{}'.format(api_key, latitude, longitude)
        response = requests.get(url).json()
        if verbose:
            print('Google Maps API JSON result =>', response)
        results = response['results']
        address = results[0]['formatted_address']
        return address
    except:
        return None
```

```
addr = get_address(google_api_key, 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

Saving...



[147] is: Alexanderpl. 5, 10178 Berlin, Germany

```
print('Obtaining location addresses: ', end='')
addresses = []
for lat, lon in zip(latitudes, longitudes):
    address = get_address(google_api_key, lat, lon)
```


```

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: . . . . .

addresses[150:170]

 ['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']

Saving...



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

```
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)
```

	Address	Distance from center	Latitude	Longitude	X	Y
0	Bundesautobahn 100 & Tempelhofer Damm, 12099 B...	5992.495307	52.470194	13.388575	390541.280176	5.814557e+06
1	09R/27L, 12101 Berlin	5840.376700	52.470314	13.397404	391141.280176	5.814557e+06
2	09R/27L, 12049 Berlin	5747.173218	52.470434	13.406234	391741.280176	5.814557e+06
3	09R/27L, 12049 Berlin	5715.767665	52.470552	13.415063	392341.280176	5.814557e+06
4	Warthestraße 23, 12051 Berlin	5747.173218	52.470670	13.423893	392941.280176	5.814557e+06
5	Schierker Str. 19-20, 12051 Berlin	5840.376700	52.470788	13.432722	393541.280176	5.814557e+06
6	Karl-Marx-Straße 213, 12055 Berlin	5992.495307	52.470904	13.441552	394141.280176	5.814557e+06
7	Hessenring 34, 12101 Berlin	5855.766389	52.474683	13.375159	389641.280176	5.815077e+06
8	Thuyring 6, 12101 Berlin	5604.462508	52.474804	13.383989	390241.280176	5.815077e+06
9	09L/27R, 12101 Berlin	5408.326913	52.474924	13.392820	390841.280176	5.815077e+06

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

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

Saving...

## ► 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 - coffee shops, pizza places, bakeries etc. are not direct competitors so we don't care about those. So we will include in our 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.

↳ 11 cells hidden

## ▼ 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

Saving...



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**:

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

df_locations['Restaurants in area'] = location_restaurants_count

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

df_locations.head(10)
```

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

	Address	Distance from center	Latitude	Longitude	X	Y	Restau
0	Bundesautobahn 100 & Tempelhofer Damm, 12099 B...	5992.495307	52.470194	13.388575	390541.280176	5.814557e+06	
1	09R/27L, 12101 Berlin	5840.376700	52.470314	13.397404	391141.280176	5.814557e+06	
2	09R/27L, 12049 Berlin	5747.173218	52.470434	13.406234	391741.280176	5.814557e+06	
3	09R/27L, 12049 Berlin	5715.767665	52.470552	13.415063	392341.280176	5.814557e+06	
4	Warthestraße 23, 12051 Berlin	5747.173218	52.470670	13.423893	392941.280176	5.814557e+06	
5	Schierker Str. 19-20, 12051 Berlin	5840.376700	52.470788	13.432722	393541.280176	5.814557e+06	
6	Karl-Marx-Straße 213, 12055 Berlin	5992.495307	52.470904	13.441552	394141.280176	5.814557e+06	
7	Hessenring 34, 12101 Berlin	5855.766389	52.474683	13.375159	389641.280176	5.815077e+06	
8	Thuyring 6, 12101 Berlin	5604.462508	52.474804	13.383989	390241.280176	5.815077e+06	
9	09L/27R, 12101 Berlin	5408.326913	52.474924	13.392820	390841.280176	5.815077e+06	

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).

Saving...



```
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_restaurant

df_locations.head(10)

```



	Address	Distance from center	Latitude	Longitude	X	Y	Restaurant
0	Bundesautobahn 100 & Tempelhofer Damm, 12099 B...	5992.495307	52.470194	13.388575	390541.280176	5.814557e+06	
1	09R/27L, 12101 Berlin	5840.376700	52.470314	13.397404	391141.280176	5.814557e+06	
2	09R/27L, 12049 Berlin	5747.173218	52.470434	13.406234	391741.280176	5.814557e+06	
3	09R/27L, 12049 Berlin	5715.767665	52.470552	13.415063	392341.280176	5.814557e+06	
4	Warthestraße 23, 12051 Berlin	5747.173218	52.470670	13.423893	392941.280176	5.814557e+06	
5	Schierker Str. 19-20, 12051 Berlin	5840.376700	52.470788	13.432722	393541.280176	5.814557e+06	
6	Karl-Marx-Straße 213, 12055 Berlin	5992.495307	52.470904	13.441552	394141.280176	5.814557e+06	
7	Hessenring 34, 12101 Berlin	5855.766389	52.474683	13.375159	389641.280176	5.815077e+06	
8	Thuyring 6, 12101 Berlin	5604.462508	52.474804	13.383989	390241.280176	5.815077e+06	
9	09L/27R, 12101 Berlin	5408.326913	52.474924	13.392820	390841.280176	5.815077e+06	

Saving...



italian restaurant from each area center:', df\_locations['Distance to Italian restaurant'].mean())



Average distance to closest Italian restaurant from each area center: 495.2099580523902

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 create a map showing **heatmap / density of restaurants** and try to extract some meaningful 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.

```
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 }
```

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

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

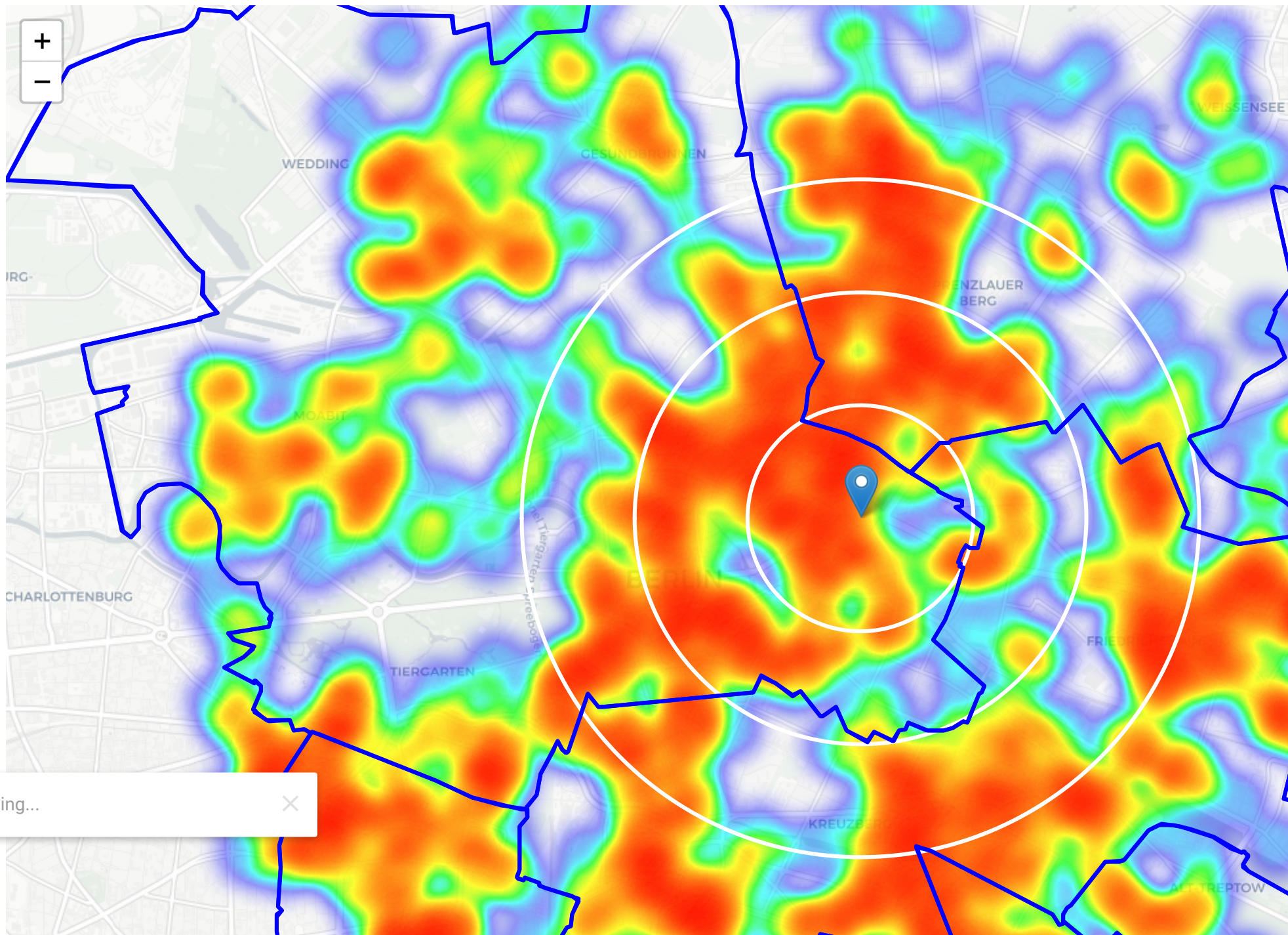
```
from folium import plugins
from folium.plugins import HeatMap
```

```
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_berlin) #cartodbpositron cartodbdark_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='geojson').add_to(map_berlin)
```

Saving...



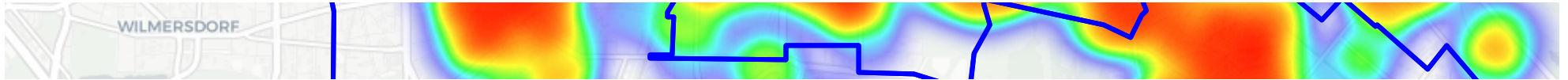




Saving...







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.

```
map_berlin = folium.Map(location=berlin_center, zoom_start=13)
folium.TileLayer('cartodbpositron').add_to(map_berlin) #cartodbpositron cartodbdark_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='geojson').add_to(map_berlin)
map_berlin
```

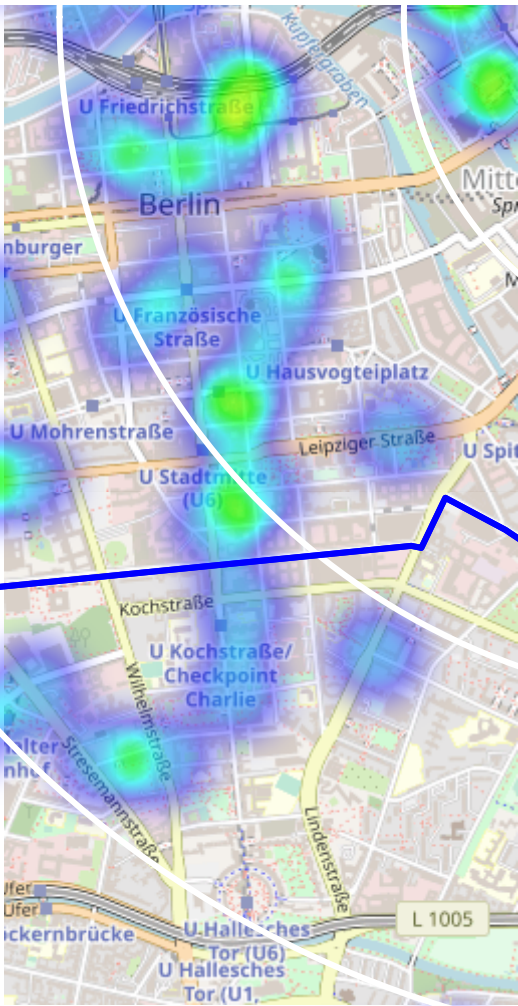


Saving...





Saving... ✕





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 residential and less popular with tourists).

## ► Kreuzberg and Friedrichshain

Analysis of popular travel guides and web sites often mention Kreuzberg and Friedrichshain as beautiful, 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)

Saving...



ut and you'll begin to envision Friedrichshain. Single walls aren't canvases for creative works, entire 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.

## ▼ 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 apart); 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.

These location candidates were then clustered to create zones of interest which contain greatest number of location candidates. Addresses of these zones were 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 decision 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.

Saving...



Saving...

