Problem statement

Background

Consider you are living in a nice Toronto city (to be more explicit in green Crescent Town) surrounded by parks, but you have to relocate to Berlin (where I actually live), because you want to enlarge a footprint of your fast growing startup, that has just raised some millions through series “C” funding round. You might also have a family and they surely will go with you, so you are sort of picky regarding new neighborhood.

Of course, you want that in Berlin you settle down in the same nice and pretty area, have all you need to feel happy and everything you have got used too. You want to get some Berlin insights upfront, but you cannot waste your time surfing the web in search of right option, because you are obviously very busy with marketing and growth strategy. So finally you would like to know which neighborhoods (called Ortsteile) will full fill your needs the most.

Generalization

Relocation outside your home city is one of the most pressing questions, that you need to answer once you decided to leave your domicile for whatever reason. The majority of people would like to have comparable life quality, surroundings and environment in the new place. But it is not really straightforward to figure out which area is the best match, if you are not familiar with a new city.
The objective of this project is to utilize Foursquare location data, leveraged by clustering of venues to determine what might be the ‘best’ neighborhood in Berlin to relocate.

Stakeholders

This piece of analysis can be used to compare areas in different capitals in terms of life quality, accessibility and diversity. This can be utilized by relocation agencies and real-estate agents, in order to provide better service and advice for their clients, as well as business owners who are willing to expand beyond their domiciles and need to obtain insights into new hoods upfront. Either way it can support state authorities, giving a better understanding regarding how life quality and diversity vary in the cities within the country.

Data

To achieve the goal stated above, I used the following sources:

- Wiki pages to pull out hoods vs boroughs. To parse both html files, I used BeautifulSoup, that allows to parse HTML webpages and navigate over tags, so that finally data can be packed into a data frame.
- To located coordinates of each hood I used Geopy Nominatim geocoder for OpenStreetMap data. Foursquare API was used to enrich hoods with venues.
- Parsed data frames, containing hoods, boroughs and their geo location are saved under names Toronto_data.csv and Berlin_data.csv for further reference.

Finally, to retrieve the venues and venues’ details I used Foursquare RESTful API calls. Foursquare is the biggest location data provider, claiming itself to be the most trusted, independent geodata and technology platform, more information can be found here.

Toronto hoods

List of Toronto neighborhoods vs Boroughs is pulled out from here: https://en.wikipedia.org/wiki/List_of_city-designated_neighbourhoods_in_Toronto

For administrative purposes, the City of Toronto is divided into 140 neighborhoods. These divisions are used for internal planning purposes. The boundaries and names often do not conform to the usage of the general population or designated business improvement areas.

	Borough	Neighborhood	Latitude	Longitude
0	Scarborough	Agincourt and Brimwood	43.785353	-79.278549
1	Scarborough	Agincourt and Malvern	43.781969	-79.257689
2	Etobicoke	Alderwood	43.601717	-79.545232
3	Old City of Toronto	The Annex and Seaton Village	43.663173	-79.410783
4	North York	Don Mills	43.775347	-79.345944

Berlin hoods

List of Berlin neighborhoods vs Boroughs data is pulled out form here.

In January 1, 2001 an administrative reform divided Berlin into 12 boroughs (Bezirke), which function as administrative districts according to the principles of self-government. The boroughs are divided into 96 neighborhoods (Ortsteile).

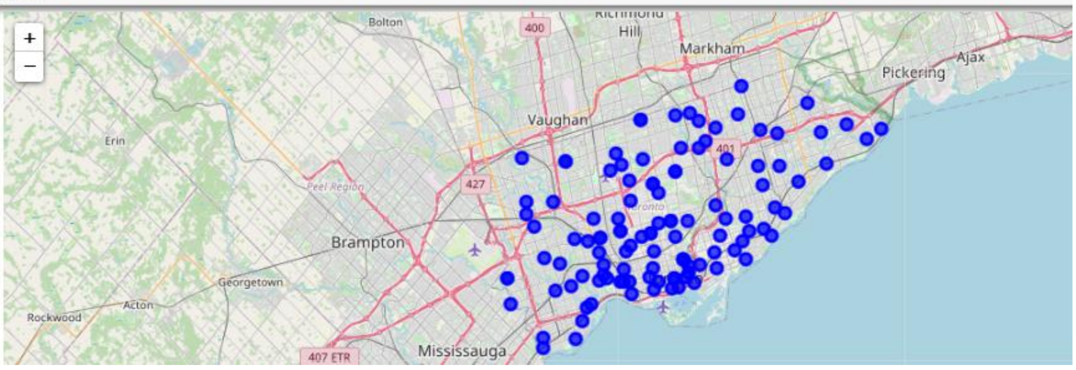
	Neighborhood	Borough	Latitude	Longitude
0	Mitte	Mitte	52.517690	13.402376
1	Moabit	Mitte	52.530102	13.342542
2	Hansaviertel	Mitte	52.519123	13.341873
3	Tiergarten	Mitte	52.509778	13.357260
4	Wedding	Mitte	52.550123	13.341970

Lets plot our neighborhoods. The maps were generated using python 'folium' library.

Toronto

```
In [289]: #plot neibourhoods in Totonto
          adress = 'Toronto'
          zoom = 10
          c = 'blue'
          latitude = LAT_TORO
          longitude = LONG_TORO

          map_n = folium.Map(location=[latitude, longitude], zoom_start=zoom)
          map_n = hf.plot_markers(tz[:], map_n, c, 5)
          map_n
```

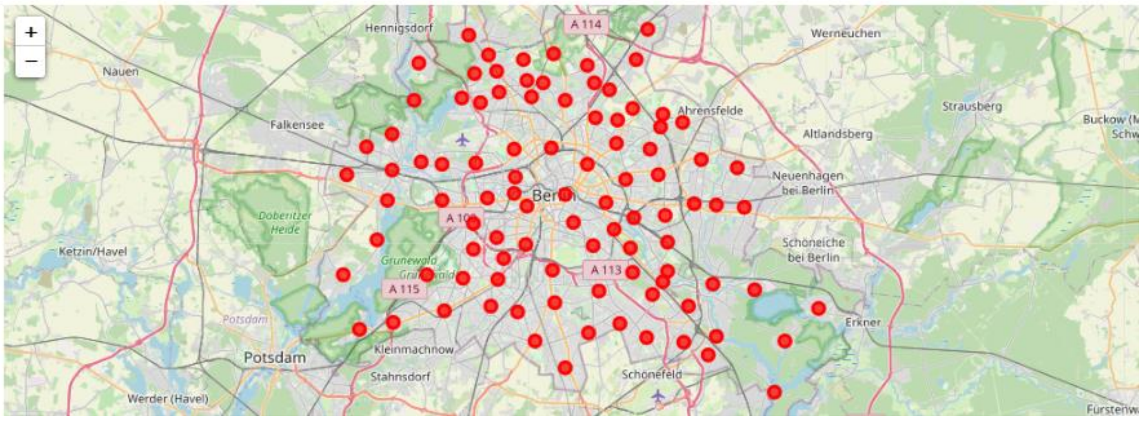
Out[289]:



Berlin

```
In [290]: #plot neibourhoods in Berlin
          adress = 'Berlin'
          c ='red'
          latitude = LAT_BERL
          longitude = LONG_BERL

          map_n = folium.Map(location=[latitude, longitude], zoom_start=zoom)
          map_n = hf.plot_markers(br[:], map_n, c, 5)
          map_n
```

```
Out[290]:
```



Methodology

Foursquare requests

The Foursquare API is used to explore the neighborhoods and get to know what venues are in there, how they are rated, what is the price category, number of likes and so on. To access the API, the used needs to define 'CLIENT_ID', 'CLIENT_SECRET', 'CLIENT_TOKEN' and 'VERSION'.

For the purpose of the analysis we need to fetch all venues from all Toronto and Berlin hoods together with rating, likes, price category, as these parameters normally define the quality of a venue and the area. This task was addressed by creation of helper functions call one within the other:

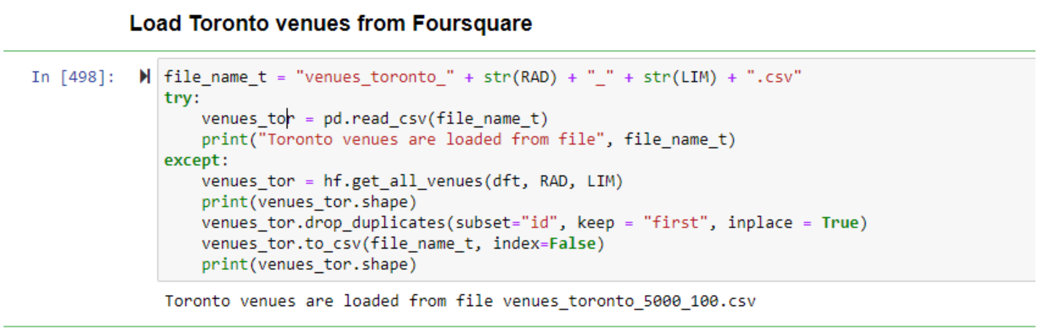
1. get_all_venues(dataframe,radius,limit) – a function wrapper that loops over all hoods and pulled out venues for each through *get_venues* call, and then runs *expore_venue* on each venue to assign category, price, rating and the number of tips and likes. Function returns a data frame, containing Venue ID, Name, Category, Address, Borough, Neighborhood, Latitude, Longitude, Rating, Price category, Tips, Likes.

	Name	Category	Address	Borough	Neighborhood	Latitude	Longitude	id	Rating	Price_cat	Tips	Likes
0	Tim Hortons	Coffee Shop	4200 Sheppard Ave East	Scarborough	Agincourt and Brimwood	43.785637	-79.279215	5058e50ce4b0a3bd556ccbdf	7	0	3	14.0
1	Bus Stop 57, 85 & 190	Bus Line	Midland & Shepard	Scarborough	Agincourt and Brimwood	43.785114	-79.278564	4f66441be4b0cded5e234b44	0	0	0	0.0
2	May Yan Seafood Restaurant 陸福海鮮酒家	Chinese Restaurant	4227 Sheppard Ave. E. Unit B2	Scarborough	Agincourt and Brimwood	43.784513	-79.277735	5d80fe5ba86ac4000795406a	0	0	0	0.0
3	Rainbow Food	Chinese Restaurant	4227 Sheppard Avenue East	Scarborough	Agincourt and Brimwood	43.784946	-79.277958	4c6dd5dd06ed6dcbd338a522	0	0	1	1.0
4	Agincourt Professional Building	Medical Center	4235 Sheppard Avenue East	Scarborough	Agincourt and Brimwood	43.785299	-79.277342	4d2b7d4cf7a9224bdffe0fa0	0	0	0	0.0

2. *get_venues(Neighborhood,latitude, longitude,radius,limit)* - extracts all possible venues given the radius (set to 1000km) and limited to limit (set to 1000). This function uses GET request to endpoint https://api.foursquare.com/v2/venues/search. More details regarding response fields can be found [here](https://developer.foursquare.com/docs/api-reference/venues/search/). The .json result, containing venues 'Name', 'Category', 'Address', 'Borough', 'Latitude', 'Longitude' and 'id' is parsed and passed on to *expore_venue* function .
3. *expore_venue(venue_id)* retrieves rating, price and the number of like and tips for given venue, based on the venue ID, sending GET request to https://api.foursquare.com/v2/venues/ endpoint.

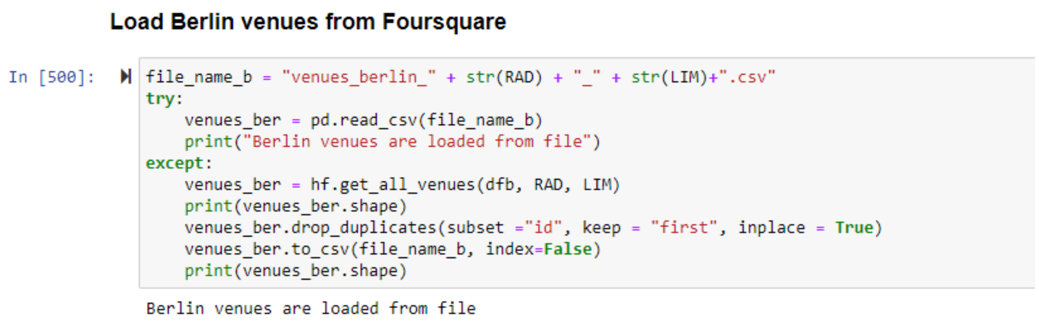
Note: the above functions were moved to the helper_functions.py file to assure better readability of the main code.

Final data frames are saved down in the script directory under the names venues_<city>_<radius>_<limit>.csv



Load Toronto venues from Foursquare

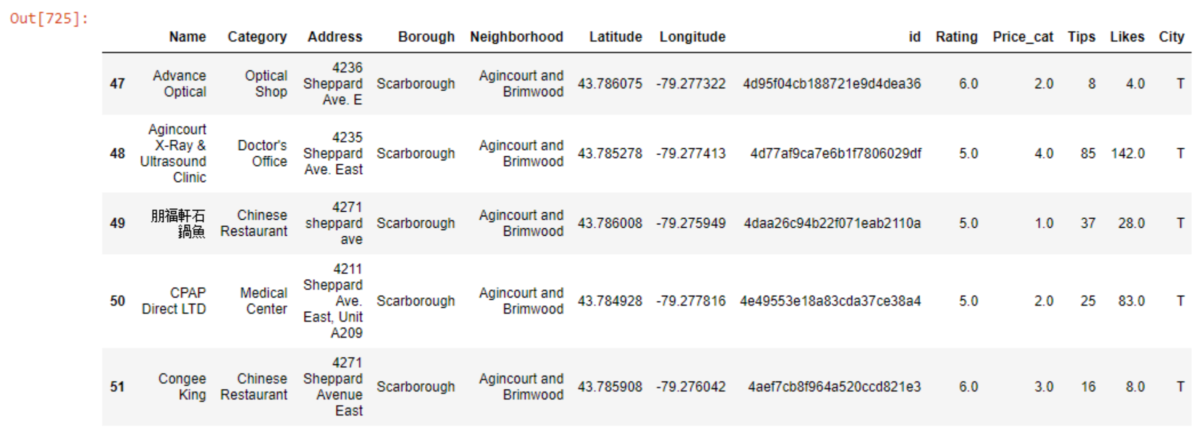
```
In [498]:  file_name_t = "venues_toronto_" + str(RAD) + "_" + str(LIM) + ".csv"
           try:
               venues_tor = pd.read_csv(file_name_t)
               print("Toronto venues are loaded from file", file_name_t)
           except:
               venues_tor = hf.get_all_venues(dft, RAD, LIM)
               print(venues_tor.shape)
               venues_tor.drop_duplicates(subset="id", keep = "first", inplace = True)
               venues_tor.to_csv(file_name_t, index=False)
               print(venues_tor.shape)

           Toronto venues are loaded from file venues_toronto_5000_100.csv
```



Load Berlin venues from Foursquare

```
In [500]:  file_name_b = "venues_berlin_" + str(RAD) + "_" + str(LIM)+".csv"
           try:
               venues_ber = pd.read_csv(file_name_b)
               print("Berlin venues are loaded from file")
           except:
               venues_ber = hf.get_all_venues(dfb, RAD, LIM)
               print(venues_ber.shape)
               venues_ber.drop_duplicates(subset ="id", keep = "first", inplace = True)
               venues_ber.to_csv(file_name_b, index=False)
               print(venues_ber.shape)

           Berlin venues are loaded from file
```

The processed data frame looks as follows:

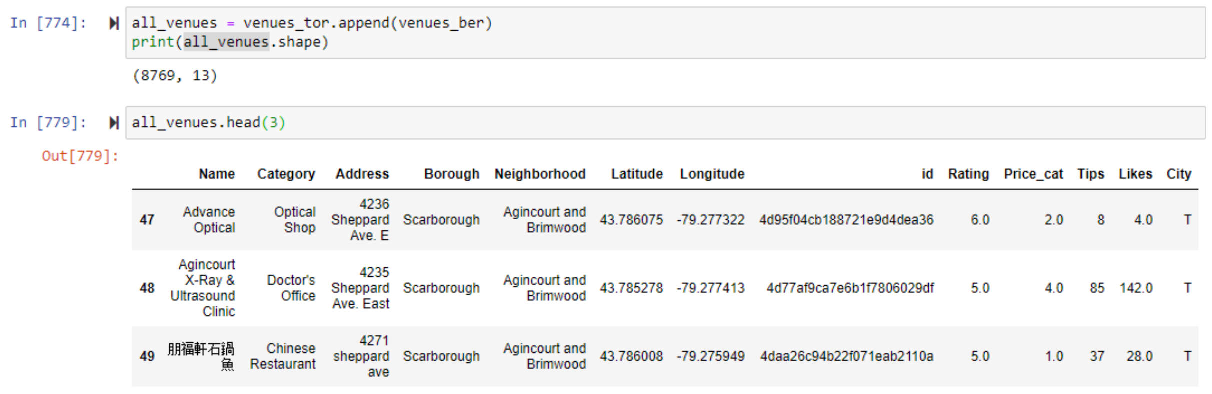


Out[725]:

	Name	Category	Address	Borough	Neighborhood	Latitude	Longitude	id	Rating	Price_cat	Tips	Likes	City
47	Advance Optical	Optical Shop	4236 Sheppard Ave. E	Scarborough	Agincourt and Brimwood	43.786075	-79.277322	4d95f04cb188721e9d4dea36	6.0	2.0	8	4.0	T
48	Agincourt X-Ray & Ultrasound Clinic	Doctor's Office	4235 Sheppard Ave. East	Scarborough	Agincourt and Brimwood	43.785278	-79.277413	4d77af9ca7e6b1f7806029df	5.0	4.0	85	142.0	T
49	朋福軒石鍋魚	Chinese Restaurant	4271 sheppard ave	Scarborough	Agincourt and Brimwood	43.786008	-79.275949	4daa26c94b22f071eab2110a	5.0	1.0	37	28.0	T
50	CPAP Direct LTD	Medical Center	4211 Sheppard Ave. East, Unit A209	Scarborough	Agincourt and Brimwood	43.784928	-79.277816	4e49553e18a83cda37ce38a4	5.0	2.0	25	83.0	T
51	Congee King	Chinese Restaurant	4271 Sheppard Avenue East	Scarborough	Agincourt and Brimwood	43.785908	-79.276042	4aef7cb8f964a520ccd821e3	6.0	3.0	16	8.0	T

Exploratory Data Analysis

As of now we have 2 data frames, one of each containing exhaustive information about venues in Toronto and Berlin. The number of Toronto venues is 4,187 and 4,582 for Berlin. After we merge two data sets in one big data frame, we get 8,769 rows in total.



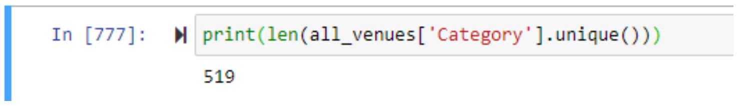
```
In [774]:   all_venues = venues_tor.append(venues_ber)
            print(all_venues.shape)

            (8769, 13)

In [779]:   all_venues.head(3)

Out[779]:
```

	Name	Category	Address	Borough	Neighborhood	Latitude	Longitude	id	Rating	Price_cat	Tips	Likes	City
47	Advance Optical	Optical Shop	4236 Sheppard Ave. E	Scarborough	Agincourt and Brimwood	43.786075	-79.277322	4d95f04cb188721e9d4dea36	6.0	2.0	8	4.0	T
48	Agincourt X-Ray & Ultrasound Clinic	Doctor's Office	4235 Sheppard Ave. East	Scarborough	Agincourt and Brimwood	43.785278	-79.277413	4d77af9ca7e6b1f7806029df	5.0	4.0	85	142.0	T
49	朋福軒石鍋魚	Chinese Restaurant	4271 sheppard ave	Scarborough	Agincourt and Brimwood	43.786008	-79.275949	4daa26c94b22f071eab2110a	5.0	1.0	37	28.0	T

Data Cleaning

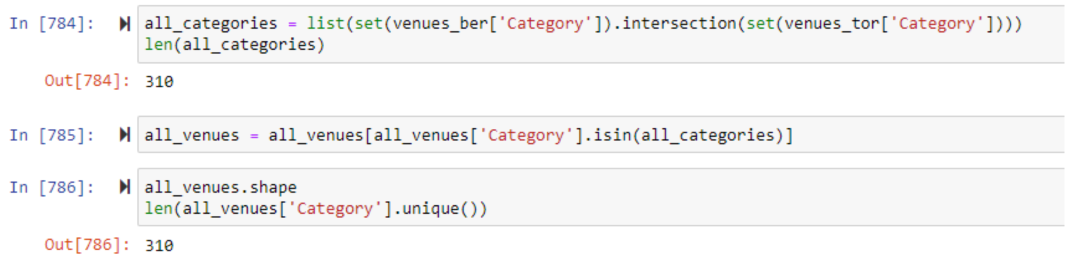
The crux of the project is to conclude what area in Berlin matches specific hood in Toronto, in terms of quality and variability the most, by clustering them using categories as well as ratings and prices information. Hence, we can throw away those venues that have category missing, or unrated or don't have a price. We also keep only those categories that present in both cities. Further more, some categories are redundant, for example 'Building' or 'Bus stop' do not add much value, as they don't help to distinguish between the hoods, but can potentially add noise. The latter were also dropped. Overall, there are 43 venues with missing Rating and 58, that don't have price category (1,15% of the entire population). After missing values are dropped the resulting data frame has 8711 rows.

Lets see how many unique categories we have. The overall number of categories for Toronto and Berlin altogether totals up to 519.



```
In [777]:   print(len(all_venues['Category'].unique()))

            519
```

Keep only those categories that present in both cities.



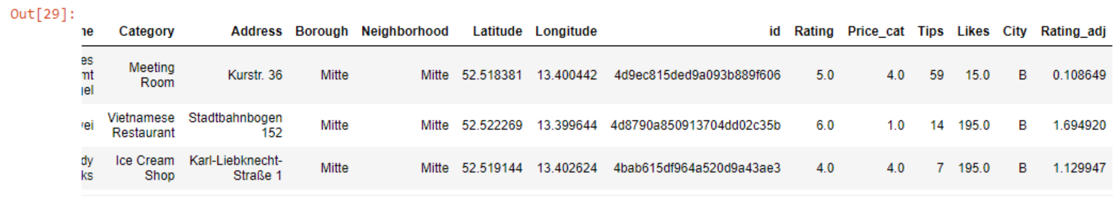
```
In [784]:   all_categories = list(set(venues_ber['Category']).intersection(set(venues_tor['Category'])))
            len(all_categories)

 Out[784]:  310

In [785]:   all_venues = all_venues[all_venues['Category'].isin(all_categories)]

In [786]:   all_venues.shape
            len(all_venues['Category'].unique())

 Out[786]:  310
```

Feature Engineering

Now let's create a new feature, called Likeness that will capture user preferences regarding the venue, as well as reflect its price.

1. Adjusted Rating

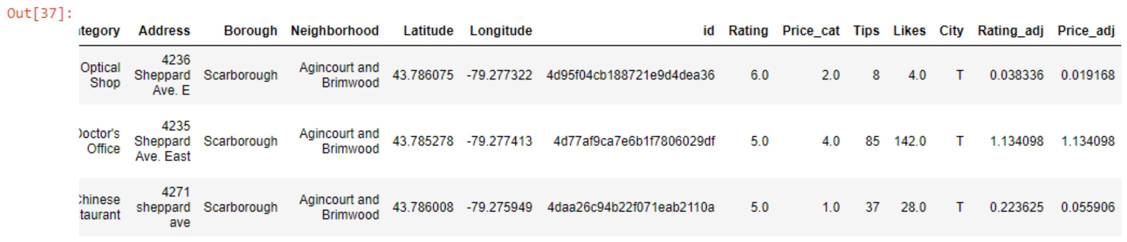
Although rating calculation algorithm, developed by foursquare is comprehensive, rating as a feature can be misleading. If we consider the case when a venue is highly rated by just a few people (e.g. owner, his wife and some employees), that can provide biased opinion. To account for that lets created likes and tips weighted rating feature that is equal to product of tips by likes divided by average number of tips and average number of likes per venue in the city. So in this case the venues that are liked, disliked, commented on more will get higher weights as their rating are more fair and reliable and vice versa.


Out[29]:

ıe	Category	Address	Borough	Neighborhood	Latitude	Longitude	id	Rating	Price_cat	Tips	Likes	City	Rating_adj
es mt ıel	Meeting Room	Kurstr. 36	Mitte	Mitte	52.518381	13.400442	4d9ec815ded9a093b889f606	5.0	4.0	59	15.0	B	0.108649
ıei	Vietnamese Restaurant	Stadtbahnbogen 152	Mitte	Mitte	52.522269	13.399644	4d8790a850913704dd02c35b	6.0	1.0	14	195.0	B	1.694920
dy ks	Ice Cream Shop	Karl-Liebknecht-Straße 1	Mitte	Mitte	52.519144	13.402624	4bab615df964a520d9a43ae3	4.0	4.0	7	195.0	B	1.129947

2. Adjusted price category

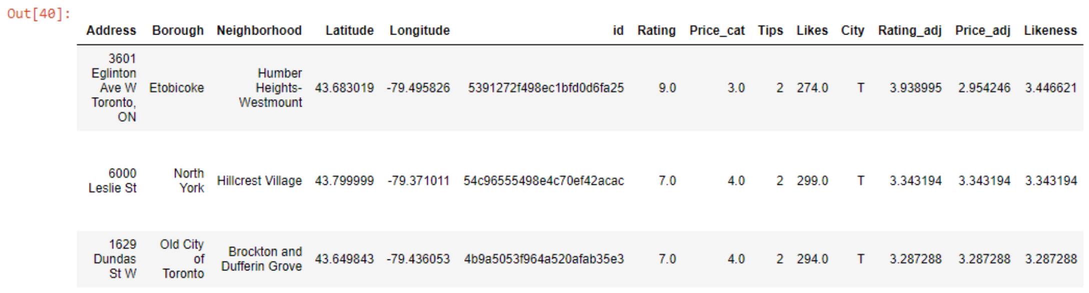
Adjusted price is one more feature, that will give us a better understanding of the price category. Price is ranging from 1 to 4, 1 being the least expensive, 4 being the most expensive. Grouping is made mainly based on a price per entree. But is also does not reflect the reality, as normally if you go to an expensive place, you expect very high quality if service, hence you tend to be more picky, compared to modest venues. In this case if a 4-category venue gets high rating it really stands out from the crowd. Let's account for this as well and construct a new feature that is equal to price divided by 4 times adjusted rating, that adds more weight to those venues that are expensive and highly and fairly rated, that means that they are really good.


Out[37]:

ıtegory	Address	Borough	Neighborhood	Latitude	Longitude	id	Rating	Price_cat	Tips	Likes	City	Rating_adj	Price_adj
Optical Shop	4236 Sheppard Ave. E	Scarborough	Agincourt and Brimwood	43.786075	-79.277322	4d95f04cb188721e9d4dea36	6.0	2.0	8	4.0	T	0.038336	0.019168
)octor's Office	4235 Sheppard Ave. East	Scarborough	Agincourt and Brimwood	43.785278	-79.277413	4d77af9ca7e6b1f7806029df	5.0	4.0	85	142.0	T	1.134098	1.134098
:hinese taurant	4271 sheppard ave	Scarborough	Agincourt and Brimwood	43.786008	-79.275949	4daa26c94b22f071eab2110a	5.0	1.0	37	28.0	T	0.223625	0.055906

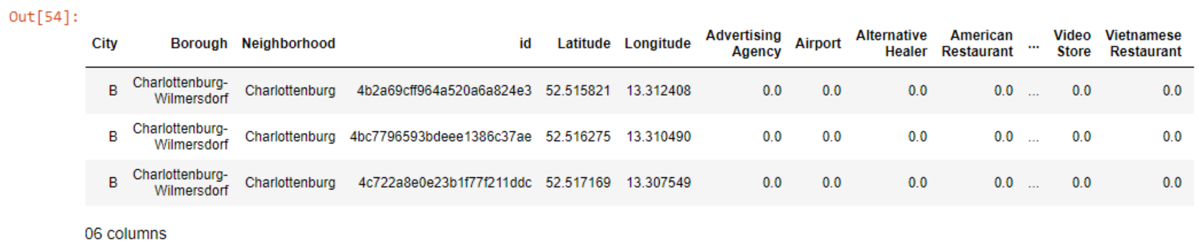
3. Likeness

This is the final feature that will capture the quality and attractiveness of the venue and call it Likeness, that is equal to the weighted sum of adjusted price and adjusted rating (for the purpose of the study I took equal wights at 0.5).


Out[40]:

Address	Borough	Neighborhood	Latitude	Longitude	id	Rating	Price_cat	Tips	Likes	City	Rating_adj	Price_adj	Likeness
3601 Eglinton Ave W Toronto, ON	Etobicoke	Humber Heights-Westmount	43.683019	-79.495826	5391272f498ec1bfd0d6fa25	9.0	3.0	2	274.0	T	3.938995	2.954246	3.446621
6000 Leslie St	North York	Hillcrest Village	43.799999	-79.371011	54c96555498e4c70ef42acac	7.0	4.0	2	299.0	T	3.343194	3.343194	3.343194
1629 Dundas St W	Old City of Toronto	Brockton and Dufferin Grove	43.649843	-79.436053	4b9a5053f964a520afab35e3	7.0	4.0	2	294.0	T	3.287288	3.287288	3.287288

On the final step the total data frame is pivoted so that venue categories become columns in the new data frame:



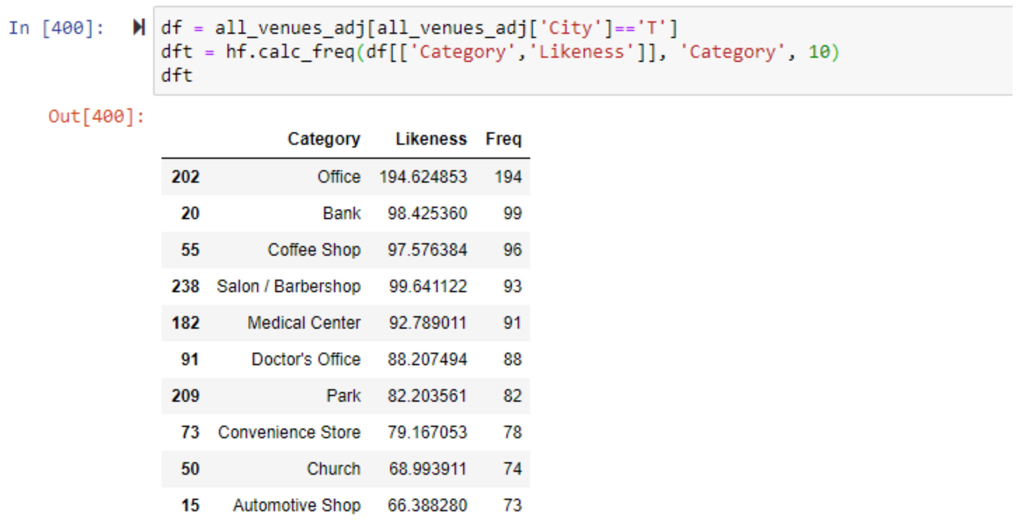
```
Out[54]:
```

City	Borough	Neighborhood	id	Latitude	Longitude	Advertising Agency	Airport	Alternative Healer	American Restaurant	...	Video Store	Vietnamese Restaurant
B	Charlottenburg-Wilmersdorf	Charlottenburg	4b2a69cff964a520a6a824e3	52.515821	13.312408	0.0	0.0	0.0	0.0	...	0.0	0.0
B	Charlottenburg-Wilmersdorf	Charlottenburg	4bc7796593bdeee1386c37ae	52.516275	13.310490	0.0	0.0	0.0	0.0	...	0.0	0.0
B	Charlottenburg-Wilmersdorf	Charlottenburg	4c722a8e0e23b1f77f211ddc	52.517169	13.307549	0.0	0.0	0.0	0.0	...	0.0	0.0

06 columns

Creation of these feature set by assigning a vector to the venue will allow to quantify how attractive the area is and also to analyze the frequency of the venue of that or this category within each capital as well as within each hoods. The above process is taken forth by using 'one hot encoding' function of python 'pandas' library. One hot encoding converts the categorical variables (which are 'Venue Category') into a form that could be provided to ML algorithms to do a better job in prediction.

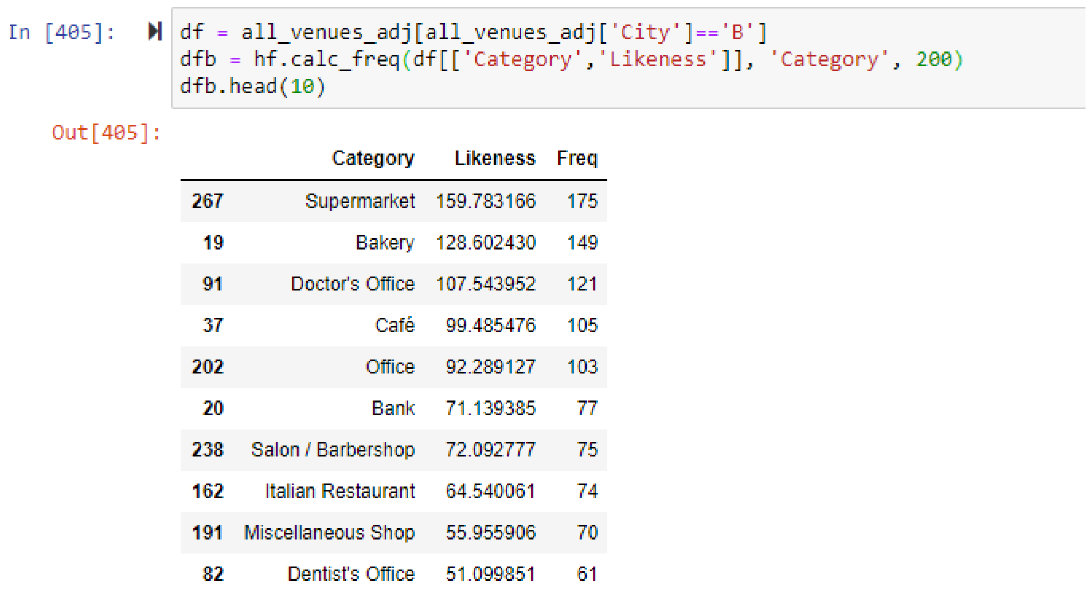
In Toronto the below venues are the most frequent:



```
In [400]:  df = all_venues_adj[all_venues_adj['City']=='T']
           dft = hf.calc_freq(df[['Category','Likeness']], 'Category', 10)
           dft

Out[400]:
```

	Category	Likeness	Freq
202	Office	194.624853	194
20	Bank	98.425360	99
55	Coffee Shop	97.576384	96
238	Salon / Barbershop	99.641122	93
182	Medical Center	92.789011	91
91	Doctor's Office	88.207494	88
209	Park	82.203561	82
73	Convenience Store	79.167053	78
50	Church	68.993911	74
15	Automotive Shop	66.388280	73

Whereas in Berlin these are the ones that can be seen the most:

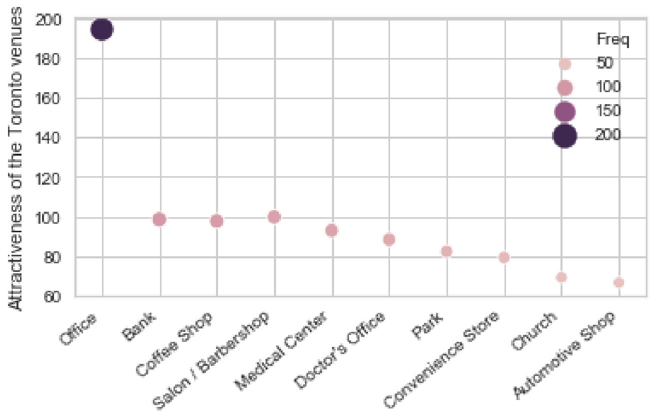


```
In [405]:  df = all_venues_adj[all_venues_adj['City']=='B']
           dfb = hf.calc_freq(df[['Category','Likeness']], 'Category', 200)
           dfb.head(10)

Out[405]:
```

	Category	Likeness	Freq
267	Supermarket	159.783166	175
19	Bakery	128.602430	149
91	Doctor's Office	107.543952	121
37	Café	99.485476	105
202	Office	92.289127	103
20	Bank	71.139385	77
238	Salon / Barbershop	72.092777	75
162	Italian Restaurant	64.540061	74
191	Miscellaneous Shop	55.955906	70
82	Dentist's Office	51.099851	61

Visualization

The below plot displays the relation between frequency and how much the category is like in average.



For example, office, bank and coffee shop are top 3 most enjoyed and common places in Toronto, one of each showing up in round 50 hoods.

```
In [458]:  freq=toronto_likeness.groupby('Neighborhood')[all_categories].sum().reset_index()
           freq=freq[freq[all_categories]!=0].count()
           freq.sort_values(ascending=False).head(10)

Out[458]:  Category
           Coffee Shop           57
           Office                53
           Salon / Barbershop    52
           Bank                  49
           Convenience Store     43
           Park                  43
           Church                43
           Pizza Place           43
           Medical Center        42
           Pharmacy              40
           dtype: int64
```

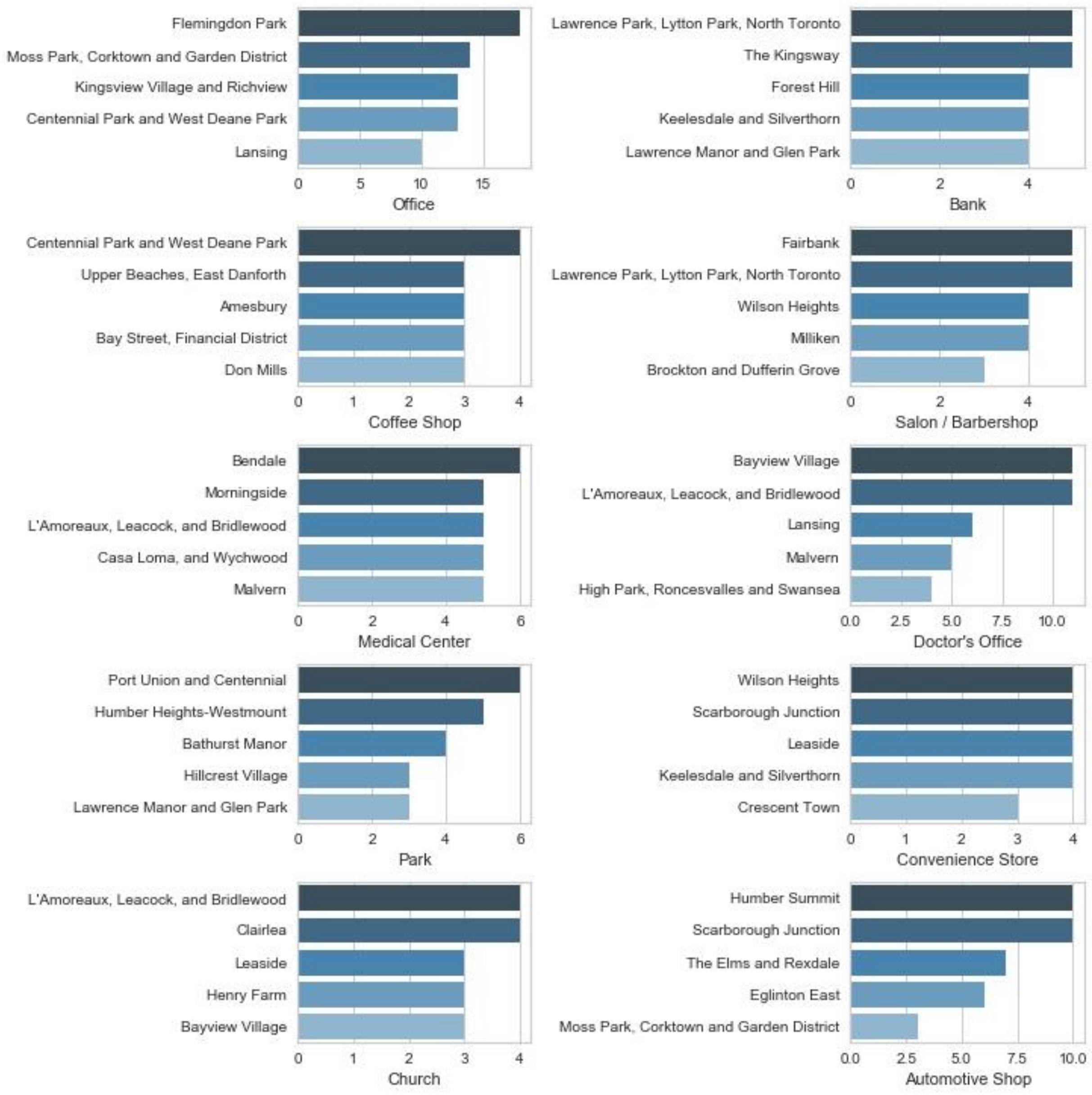
and in average they are also the most liked ones:

```
n [387]:  top_venues_toronto = venues_toronto_grouped.describe().transpose()
          top_venues_toronto = top_venues_toronto.sort_values('mean', ascending=False)[0:10]
          top_venues_toronto.reset_index()[['Category','mean']]
```

Out[387]:

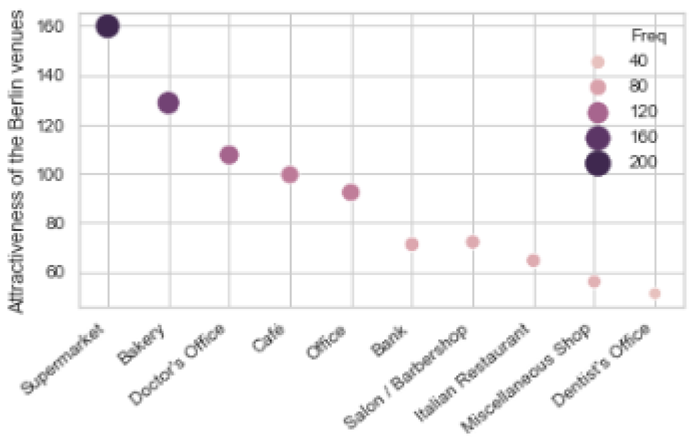
	Category	mean
0	Office	0.051451
1	Park	0.042969
2	Coffee Shop	0.028671
3	Salon / Barbershop	0.027792
4	Medical Center	0.026298
5	Bank	0.024629
6	Church	0.023744
7	Convenience Store	0.023720
8	Doctor's Office	0.021706
9	Dentist's Office	0.020206

These categories were analyzed individually, to see what hood contributes the most. As seen in the top left bar plot Flemingdon park is the main contributor, making up for ca. 25% of all offices in Toronto. And the majority of automotive shops are in Hummer Summit and Scarborough Junction. To visit a doctor office one should travel to Bayview village are L'Amoreaux.



The plotting was made using Seaborn library in a loop.

For Berlin the leading categories substantially differ, thus the most liked and frequent venues are supermarkets, bakeries and doctor's office.



That are equally distributed across the city and are seen in more than 50 hoods.

n [468]:

```
freq=berlin_likeness.groupby('Neighborhood')[all_categories].sum().reset_index()
freq=freq[freq[all_categories]!=0].count()
freq.sort_values(ascending=False).head(10)
```

Out[468]:

```
Category
Supermarket           69
Bakery                59
Café                  53
Office                51
Doctor's Office       51
Bank                  45
Italian Restaurant    45
Salon / Barbershop    44
Pharmacy              43
Dentist's Office      42
dtype: int64
```

They are the most likes ones.

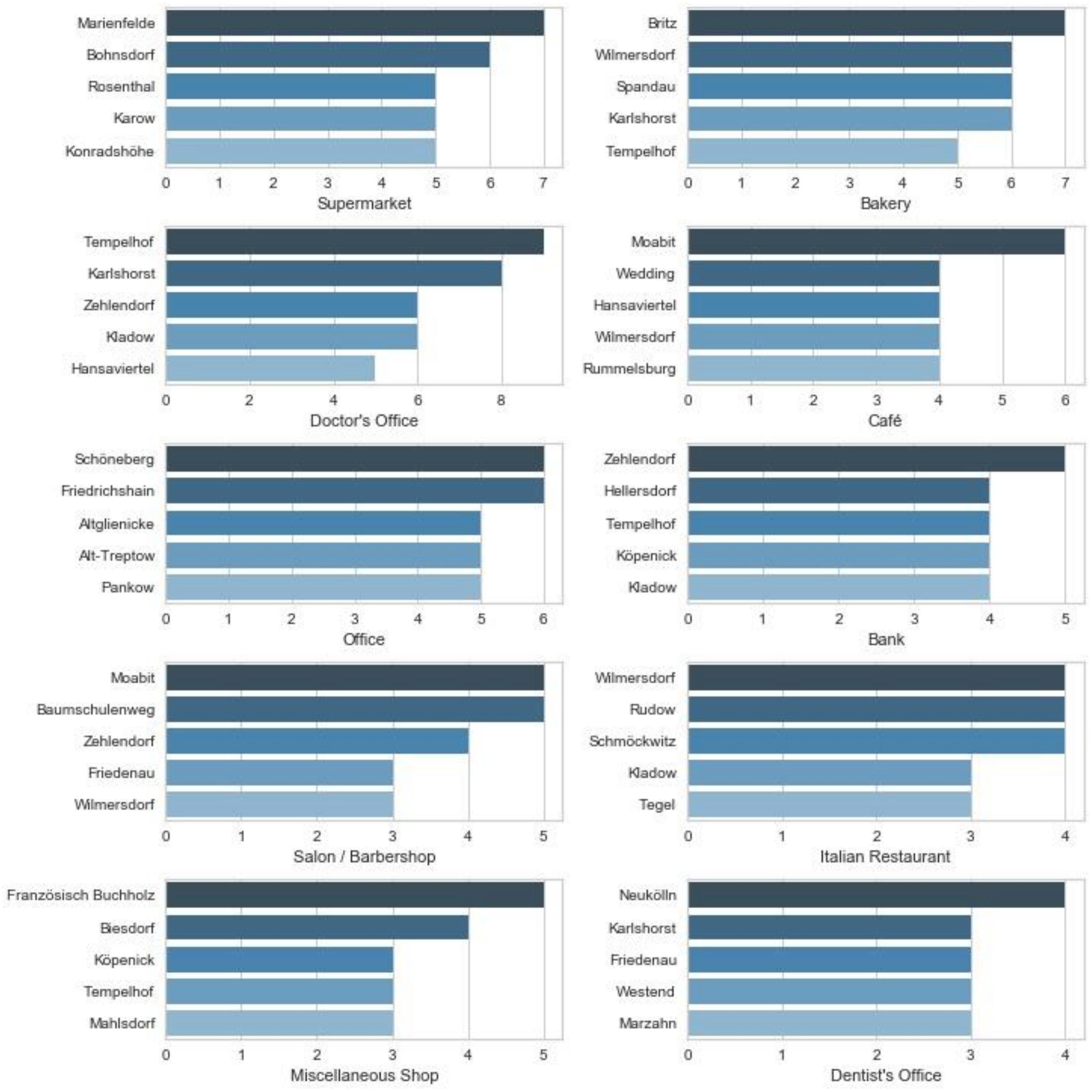
In [492]:

```
# group all venues in the hood adding likeness of the venues within the hood
top_venues_berlin = berlin_likeness.groupby('Neighborhood')[all_categories].mean().reset_index().describe().transpose()
top_venues_berlin = top_venues_berlin.sort_values('mean', ascending=False)[0:10]
top_venues_berlin.reset_index()[['Category','mean']]
```

Out[492]:

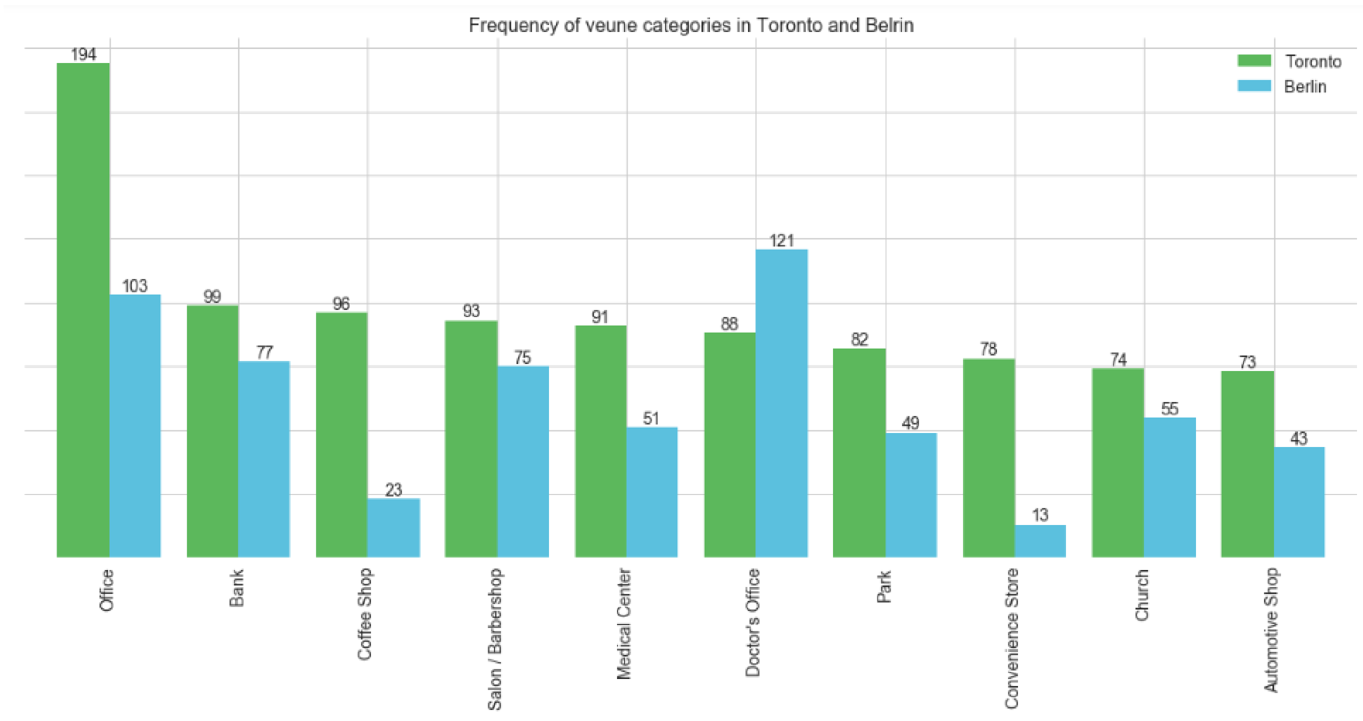
	Category	mean
0	Supermarket	0.055762
1	Bakery	0.031378
2	Café	0.026273
3	Bank	0.025196
4	Doctor's Office	0.025067
5	Office	0.024849
6	Italian Restaurant	0.020133
7	Gas Station	0.017541
8	Salon / Barbershop	0.017073
9	Pharmacy	0.014325

If we take a close look on top 10 most frequent venue categories, we will notice that each of 5 hoods contribute almost equally to the frequency of occurrence.



Although the number of venues that is return through GET call is limited by 100, the lifestyle of the hood is majorly defined by top 10 venues.

Finally let's compare how two cities vary in terms of venues populations and see how frequent the most common venues in Toronto will appear in Berlin. From the bar plot below it is seen that Berlin has 2 times less offices that Toronto, but more doctor offices. Toronto has more coffee shops and parks.



Machine Learning

K-means clustering

Clustering of hoods in each city was performed with the module sklearn.cluster, using K-means algorithm, that organizes (labels) data points into k disjoint clusters based on certain similarities. K-means is an unsupervised Machine Learning algorithm that requires to specify one parameter - the number of clusters. There are 2 most popular methods to find the optimal number of clusters, namely Elbow Method and Silhouette Method. Given that we what to have a certain degree of separation of data points as well as the size of cites, it is reasonable to set minimal number of cluster to 10, otherwise hoods will be clustered very broadly.

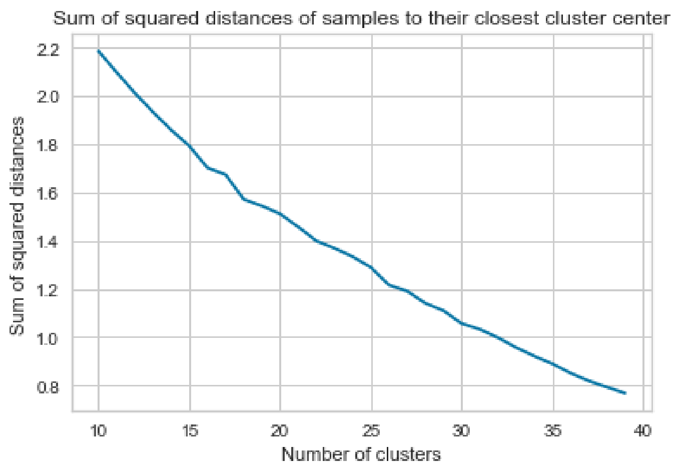
The Elbow Method

The Elbow Method calculates the sum of squared distances of samples to their closest cluster center for different values of k. The optimal number of clusters is the value after which there is no significant decrease in the sum of squared distances.

The graphical footprint of this method for Toronto is laid out on the plot below (looping for the k from 10 to 39):

```
In [68]: t_cl = venues_toronto_grouped[all_categories]
         SSE = []
         for k in range(K_min,K_max):
             kmeans = KMeans(n_clusters=k, n_init=50).fit(t_cl)
             SSE.append(kmeans.inertia_)
```

```
In [75]: plt.plot(range(K_min,K_max),SSE)
         plt.ylabel('Sum of squared distances')
         plt.xlabel('Number of clusters')
         plt.title('Sum of squared distances of samples to their closest cluster center')
         plt.show()
```



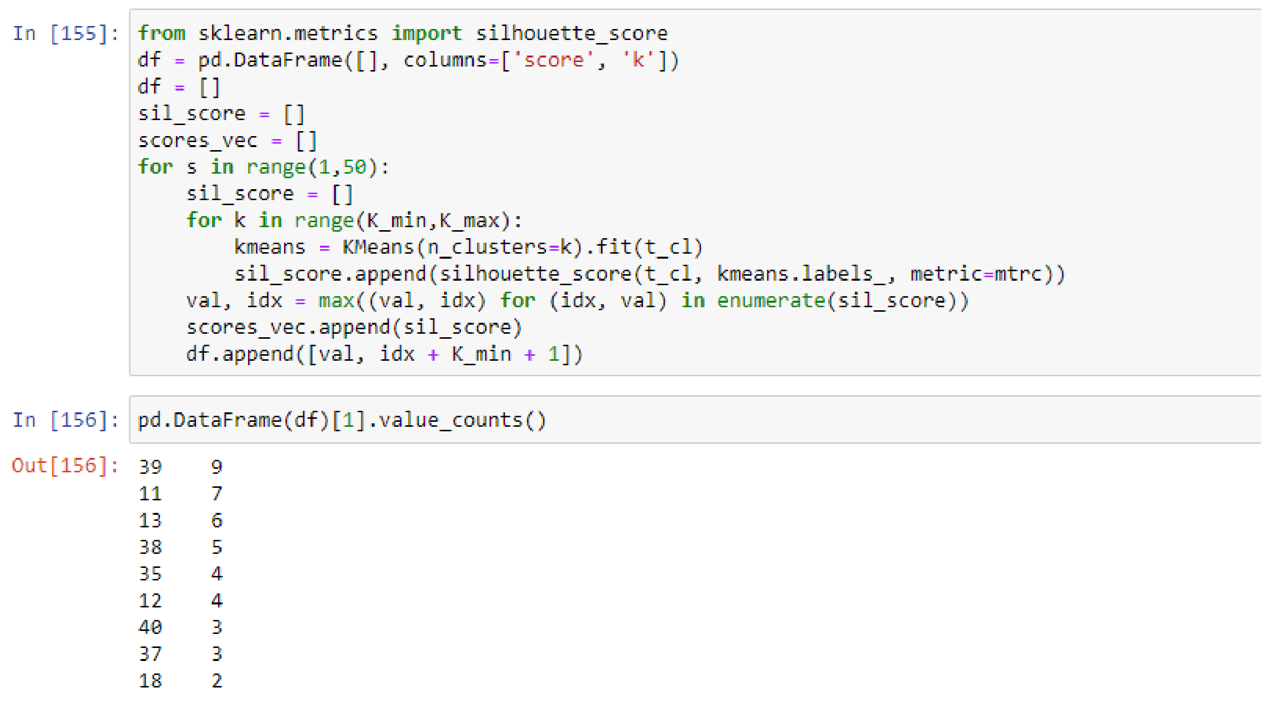
For this specific case Elbow method does not yield any sensible result as the sum of squared distances keeps decreasing with the increase of the number of clusters, hence its impossible to conclude optimal k.

The Silhouette Method

The method to measure how well data points were classified based on Silhouette Coefficient, calculated using similarity of points within the cluster (inter-cluster distance) and separation between the clusters (the distance between a sample cluster and the nearest cluster that the sample is not a part of) under the user-defined distance metric.

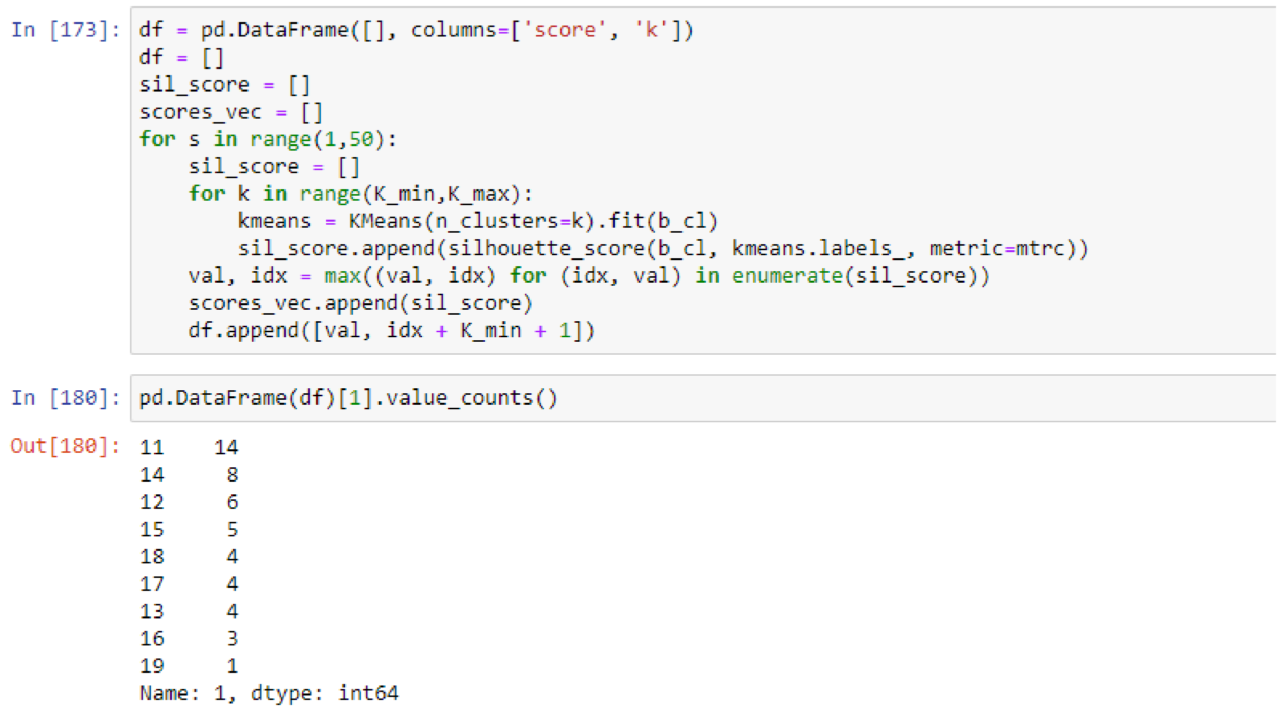
The Silhouette Method was implemented for k ranging from 10 to 39 under manhattan distance. It was noticed that the method is sensitive to initialization, meaning that for different runs we can get different optimal values. To account for that different optimal k-values were calculated for 40 runs of the method.

After the analysis of k frequencies, the value of k=13 was taken.



```
In [155]: from sklearn.metrics import silhouette_score
          df = pd.DataFrame([], columns=['score', 'k'])
          df = []
          sil_score = []
          scores_vec = []
          for s in range(1,50):
              sil_score = []
              for k in range(K_min,K_max):
                  kmeans = KMeans(n_clusters=k).fit(t_cl)
                  sil_score.append(silhouette_score(t_cl, kmeans.labels_, metric=mtrc))
              val, idx = max((val, idx) for (idx, val) in enumerate(sil_score))
              scores_vec.append(sil_score)
              df.append([val, idx + K_min + 1])

In [156]: pd.DataFrame(df)[1].value_counts()

Out[156]: 39    9
          11    7
          13    6
          38    5
          35    4
          12    4
          40    3
          37    3
          18    2
```

The same was performed for Berlin data set, resulting in k=14.



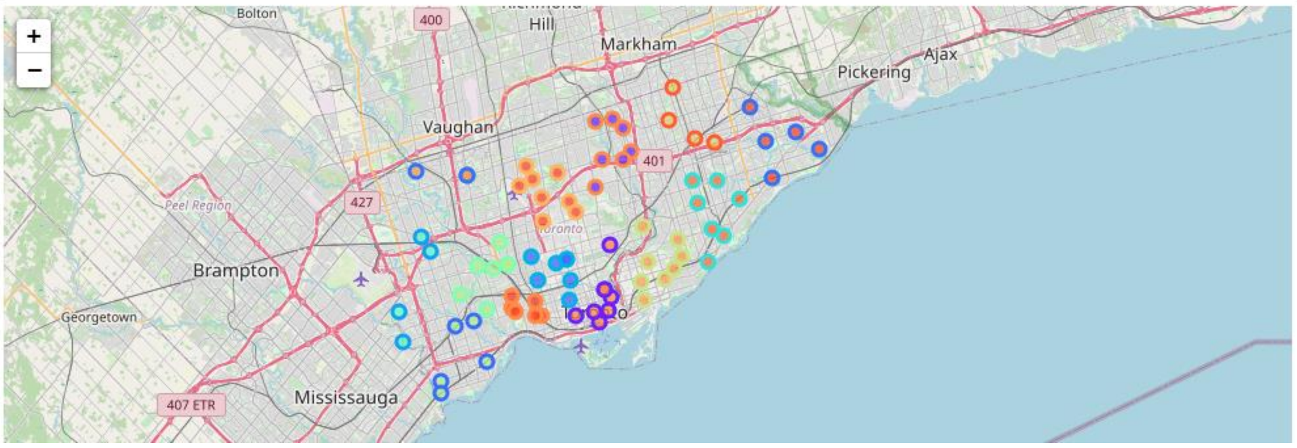
```
In [173]: df = pd.DataFrame([], columns=['score', 'k'])
          df = []
          sil_score = []
          scores_vec = []
          for s in range(1,50):
              sil_score = []
              for k in range(K_min,K_max):
                  kmeans = KMeans(n_clusters=k).fit(b_cl)
                  sil_score.append(silhouette_score(b_cl, kmeans.labels_, metric=mtrc))
              val, idx = max((val, idx) for (idx, val) in enumerate(sil_score))
              scores_vec.append(sil_score)
              df.append([val, idx + K_min + 1])

In [180]: pd.DataFrame(df)[1].value_counts()

Out[180]: 11    14
          14     8
          12     6
          15     5
          18     4
          17     4
          13     4
          16     3
          19     1
          Name: 1, dtype: int64
```

Finally lets generate clusters for the best k, found on previous step. And visualize results on the map using folium library.

Toronto

```
In [258]: #cluster anf plot data for the the best k for Toronto
          best_k_toronto = 13
          df = venues_toronto_grouped[:]
          df = df.merge(tz, on='Neighborhood')
          toronto_clustered = hf.cluster_data(df[:], best_k_toronto,cols)
          map_clusters = folium.Map(location=[latitude, longitude], zoom_start=zoom)
          [rainbow, map_clusters] = hf.plot_clusters(map_clusters, toronto_clustered, best_k_toronto, 5)
          map_clusters
```

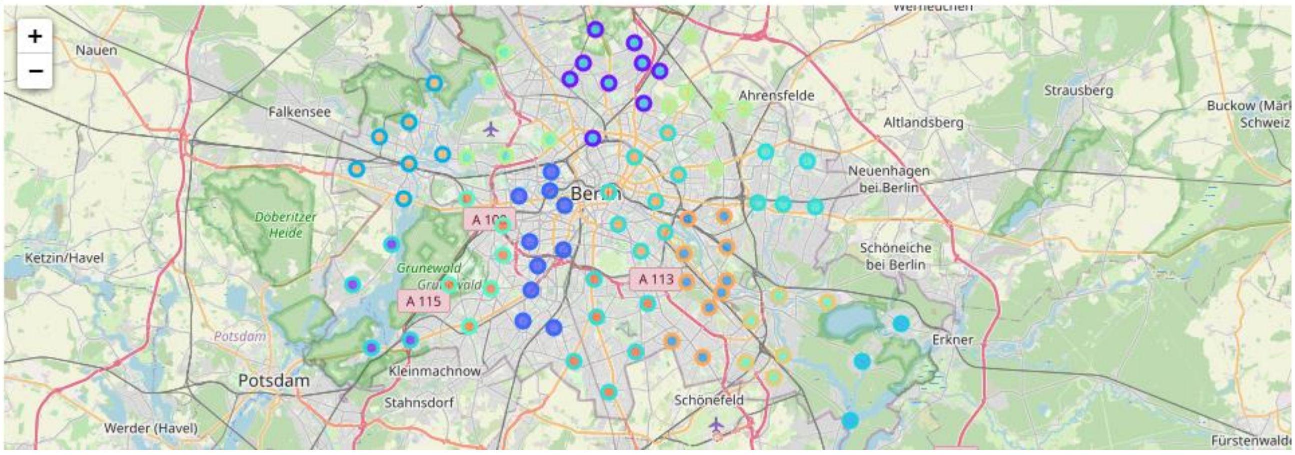
Out[258]:



Berlin

```
In [102]: #cluster anf plot data for the the best k for Toronto
          best_k_berlin = 14
          df = venues_berlin_grouped[:]
          df = df.merge(br, on='Neighborhood')
          berlin_clustered = hf.cluster_data(df[:],best_k_berlin,cols)
          map_clusters = folium.Map(location=[latitude, longitude], zoom_start=zoom)
          [rainbow, map_clusters] = hf.plot_clusters(map_clusters, berlin_clustered, best_k_berlin, 5)
          map_clusters
```

Out[102]:



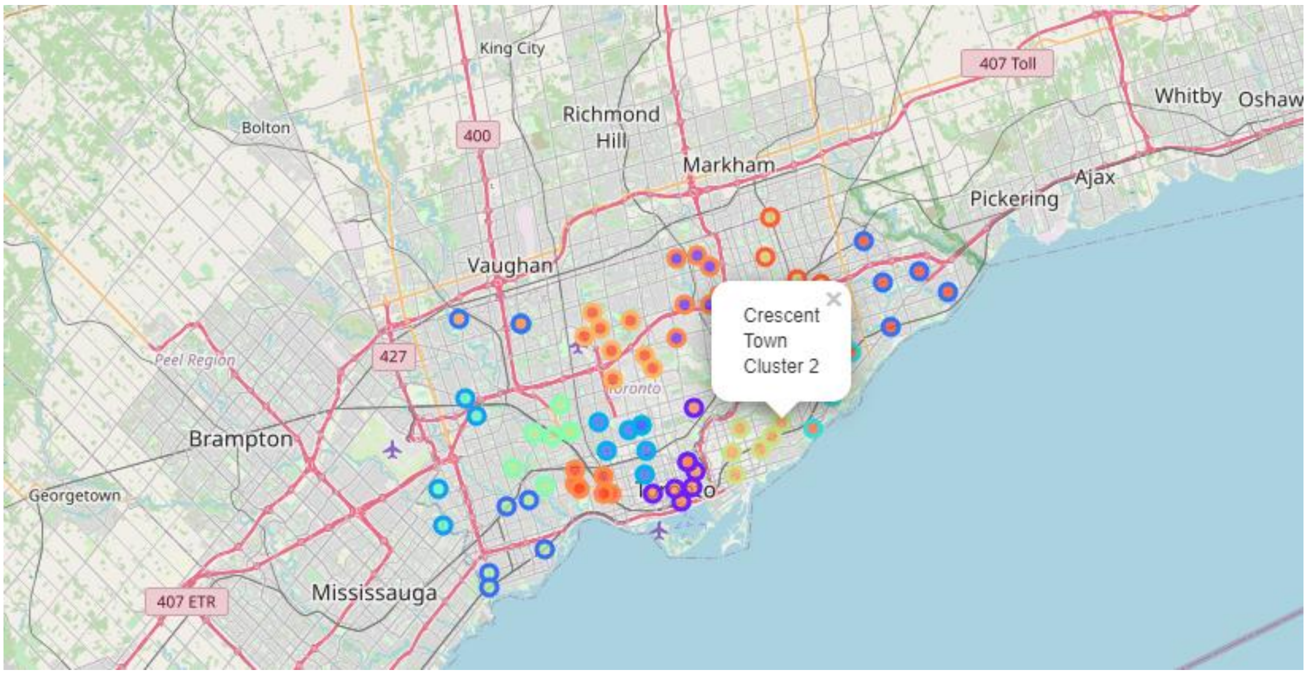
The clustering was isolated into separate function *cluster_data* that runs kmeans algorithm on the features for the best k and returns data frame that can be easily plotted on the map.

Finding best hood

On the previous steps we have generated clusters for Toronto and Berlin, in this section the best match for Crescent Town hood will be found.

Let’s identify the cluster for Crescent Town, this is cluster 2.

```
In [241]: my_hood='Crescent Town'

In [242]: my_cluster_num = toronto_clustered[toronto_clustered['Neighborhood'] == my_hood]['Cluster Labels'].values[0]

In [243]: my_cluster_num

Out[243]: 2
```



Looping over all Berlin clusters we calculate Euclidean distance between 2nd cluster center and all cluster center in Berlin.

```
In [262]: dist = pd.DataFrame(np.zeros((len(berlin_clustered_grouped['Cluster Labels']), 2)))
          dist.columns = ['Cluster Labels','dist']
          for i in  berlin_clustered_grouped['Cluster Labels']:
              berlin_cluster = berlin_clustered_grouped[berlin_clustered_grouped['Cluster Labels'] == i]
              d = berlin_cluster.values-my_cluster.values
              d = np.linalg.norm(d)
              dist.iloc[i,0] = i
              dist.iloc[i,1] = d

          dist=dist.sort_values(by='dist', ascending=True).reset_index()
          dist.head()
```

Out[262]:

	index	Cluster Labels	dist
0	2	2.0	0.109722
1	3	3.0	1.011079
2	1	1.0	1.011512
3	0	0.0	2.003530
4	4	4.0	2.004362

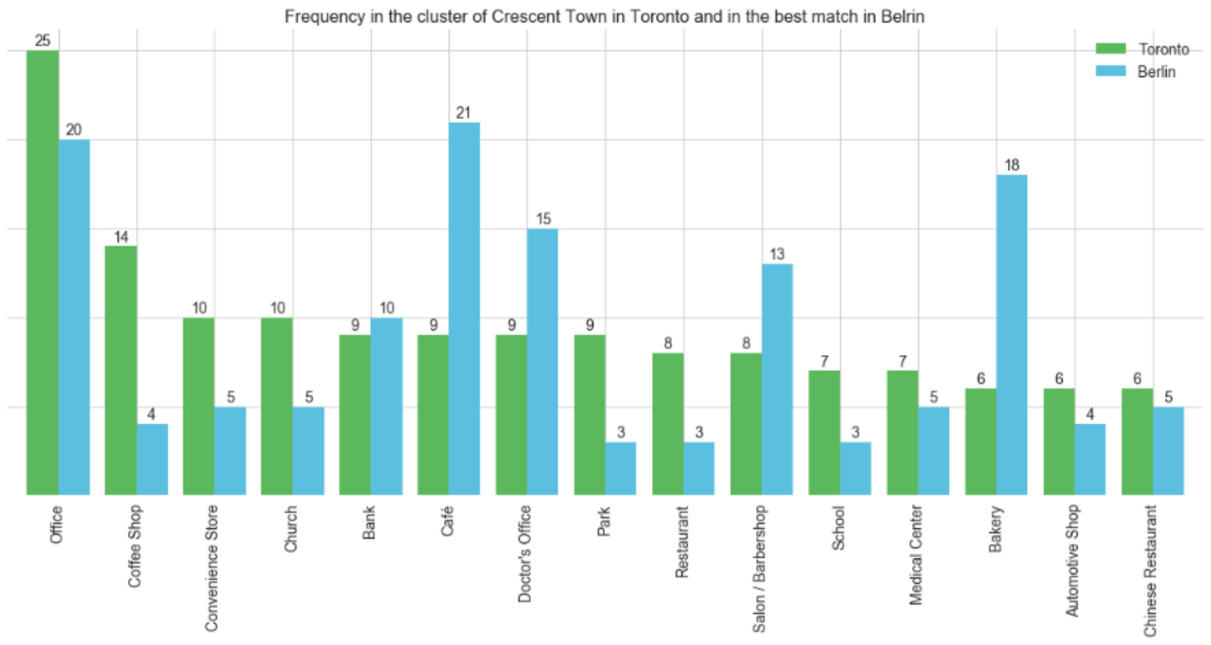
The cluster with a minimum distance would be the best match. In this case it is also cluster number 2.

```
In [260]: best_cluster_num = dist['Cluster Labels'][0]
          best_cluster_num

Out[260]: 2.0
```

Knowing cluster number we filter out the best Berlin hoods to relocate to.

```
In [231]: best_venues['Neighborhood'].unique()

Out[231]: array(['Hansaviertel', 'Wilmersdorf', 'Friedenau', 'Schöneberg',
                 'Steglitz', 'Charlottenburg', 'Moabit', 'Lichterfelde',
                 'Tiergarten', 'Lankwitz'], dtype=object)
```

Lets visualize our findings and examine the characteristics of both areas.

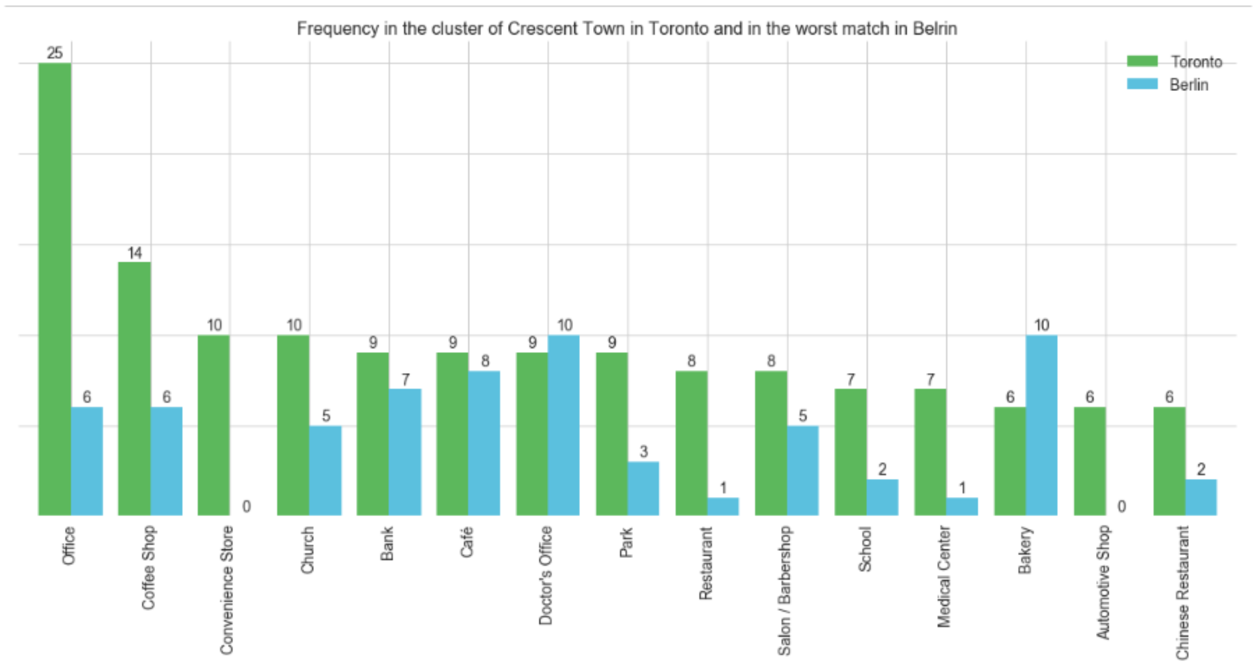


The above bar chart shows that the majority of loved Toronto spots have similar rate of occurrence in Berlin, especially with regards to most popular places such as offices, banks, barbershops, that have high weights.

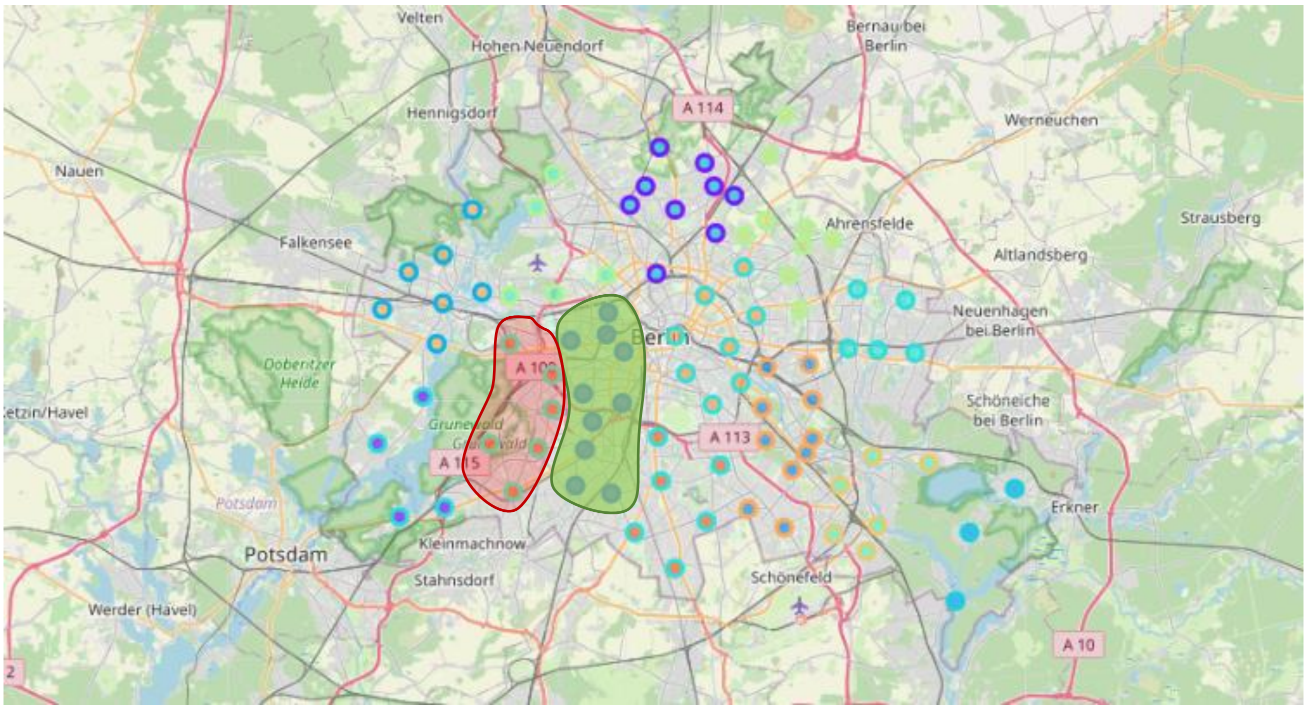
Now let's explore the worst area to move to. Follow the same steps this is cluster 12:

```
In [272]: worst_venues['Neighborhood'].unique()

Out[272]: array(['Grunewald', 'Zehlendorf', 'Halensee', 'Westend', 'Schmargendorf',
                 'Dahlem'], dtype=object)
```

This is how dissimilarity look like, the most likes offices and cafe are missing out, barbershops are not that common.



On the final step we locate Berlin clusters. The best (green) and the worst (red) areas to relocate are displayed on the map below.



Discussion

To understand what is the best Berlin match for ‘my hood’ in Toronto, four consecutive steps were made:

1. Calculation of ‘likeness’ vector, that defines my hood

2. Calculation of ‘likeness’ vectors for Berlin clusters

3. Calculation of distances between cluster center of my hood and clusters in Berlin,

4. Minimum distance points to the best match in terms of quality and habits of the area.

The below table contains Berlin clusters of hoods ordered from the best to the worst match:

Euclidean Centroid Distance	**Cluster Labels**	**Berlin Neighborhood**
0.11	**2**	Friedenau, Schoeneberg, Charlottenburg, Lichterfelde, Lankwitz, Steglitz, Moabit, Tiergarten, Hansaviertel, Wilmersdorf
1.01	**3**	Stadtrandsiedlung Malchow, Buch, Malchow, Neu-Hohenschoenhausen, Alt-Hohenschoenhausen, Falkenberg, Karow, Wartenberg, Lichtenberg
1.01	**1**	Planterwald, Rummelsburg, Rudow, Niederschoeneweide, Johannisthal, Karlshorst, Friedrichsfelde, Gropiusstadt, Oberschöneweide, Baumschulenweg
2.00	**0**	Tegel, Wedding, Siemensstadt, Reinickendorf, Charlottenburg-Nord
2.00	**4**	Nikolassee, Kladow, Gatow, Wannsee
3.00	**5**	Pankow, Gesundbrunnen, Heinersdorf, Blankenfelde, Wilhelmsruh, Franzoesisch Buchholz, Blankenburg, Niederschoenhausen
4.00	**6**	Haselhorst, Spandau, Hakenfelde, Wilhelmstadt, Staaken, Falkenhagener Feld, Konradshoehe
5.00	**7**	Rahnsdorf, Prenzlauer Berg, Weissensee
6.00	**8**	Weissensee, Neukoelln, Alt-Treptow, Friedrichshain, Fennpfuhl, Prenzlauer Berg, Kreuzberg, Mitte
7.00	**9**	Lichtenrade, Buckow, Mariendorf, Marienfelde, Tempelhof, Britz
8.00	**10**	Bohnsdorf, Koepenick, Adlershof, Friedrichshagen, Altglienicke, Gruenau
9.00	**11**	Kaulsdorf, Marzahn, Biesdorf, Mahlsdorf, Hellersdorf
10.00	**12**	Halensee, Westend, Grunewald, Schmargendorf, Zehlendorf, Dahlem

Conclusion

The main benefit of clustering algorithms, k-means in this case, is that they are able to group unlabeled high dimensional datasets, helping to understand and structure data. After all hoods in Toronto were clustered in 13 groups based on their attractiveness (likeness value) and habits (represented by frequency of the venue), the wrapping cluster for Crescent Town was found, and analyzed against the best match in Berlin, showing alignment with the preferences of the city of Toronto.

The results of the study can be improved if using broader range of location data providers to increase venue coverage. The same idea can be propagated on other cities, and will allow to make suggestions to business owners who are willing to grow abroad, as well as relocation service providers, real-estate companies and real-estate developers, or any other party willing to invest it the new area. The results can might be of interest for state authorities, giving a better understanding of life quality and similarity within the country.

References

Functions that pullout data from Foursquare were taken from Applied Data Science Capstone' course labs. Although slightly adopted for my own needs, originally they were crafted by Alex Aklson and Polong Lin.