# IBM Data Science Professional Certificate

## Capstone Project: Recommendation of London Boroughs

## Introduction

A very common problem for a lot of people is choosing where to live in a big city. While rent is an important factor when deciding which area to live in, there are a lot of other subjective criteria based on personal preferences - e.g., what kind of venues would one like to have close to home (shopping, bars, parks, etc.), are there parks nearby, etc. There are various solutions to this problem - online search, asking friends, visiting the areas yourself - all of these are rather time consuming and subjective. However, this problem can be tackled using a data-driven approach by creating a recommendation engine that would suggest areas that match the user-defined criteria.

London, UK is one of the largest cities in the world and contains 33 areas (also known as boroughs). This presents a good use case for implementing such an algorithm and testing whether it makes sensible recommendations.

## Data

To build a recommender, we need to have some information about the rents in London, borough geography, and what characteristics of each borough. While rent and geographical data are quite objective, there are many ways to characterise boroughs. For this project, however, we are going to use information about the venues retrieved from Foursquare API.

### Data Sources

The following data sources are going to be used to recommend Greater London boroughs to the user:

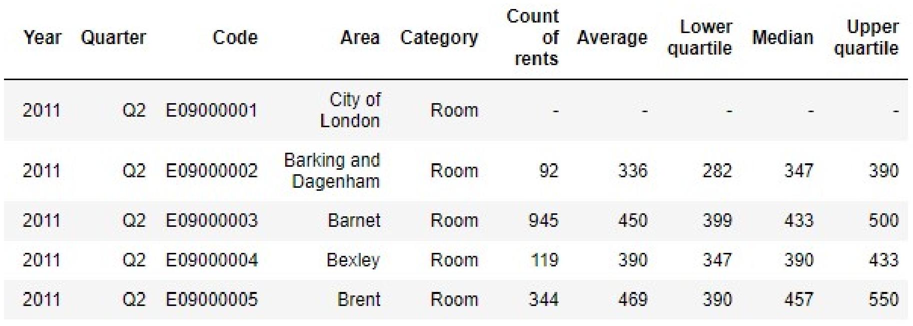
1. Rent data - available from London Datastore in *.xls format: Average Private Rents, Borough
2. List of London Boroughs and their geographical coordinates, available from Wikipedia (html table): List of London Boroughs
3. Greater London borough boundaries in GeoJSON format, available here
4. Information on venues in Greater London area. Available via free version of Foursquare API.

### Data Cleaning

**Rent Data** Rent data is retrieved from official London Datastore website which tracks average private rents per borough. The data is released quarterly. The rent information is provided for each borough and contains:

- Category - type of accommodation, e.g., Room, Studio, One Bedroom, Two Bedroom, etc.

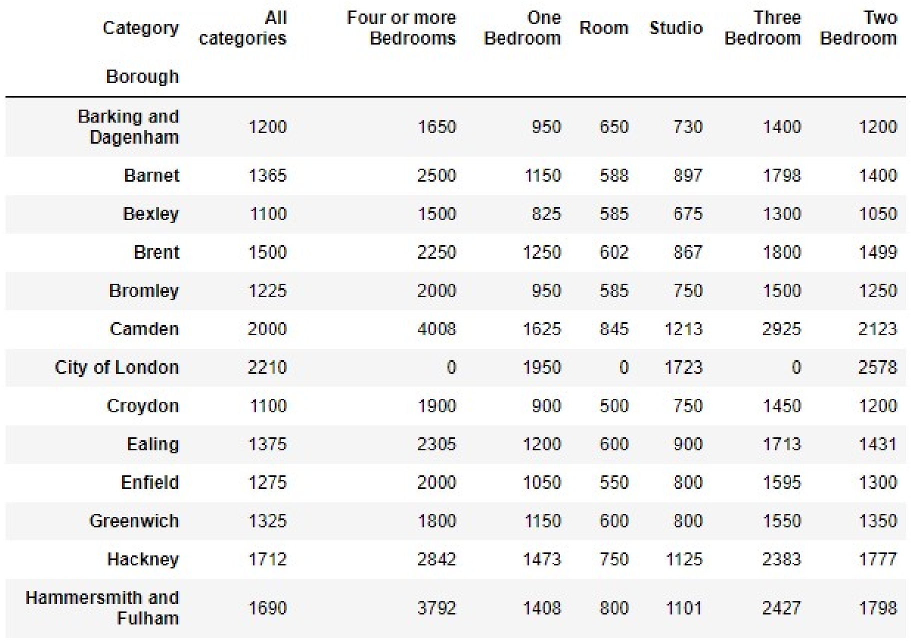
- Average
- Lower quartile
- Median
- Upper quartile



| Year | Quarter | Code | Area | Category | Count of rents | Average | Lower quartile | Median | Upper quartile |
|---|---|---|---|---|---|---|---|---|---|
| 2011 | Q2 | E09000001 | City of London | Room | - | - | - | - | - |
| 2011 | Q2 | E09000002 | Barking and Dagenham | Room | 92 | 336 | 282 | 347 | 390 |
| 2011 | Q2 | E09000003 | Barnet | Room | 945 | 450 | 399 | 433 | 500 |
| 2011 | Q2 | E09000004 | Bexley | Room | 119 | 390 | 347 | 390 | 433 |
| 2011 | Q2 | E09000005 | Brent | Room | 344 | 469 | 390 | 457 | 550 |

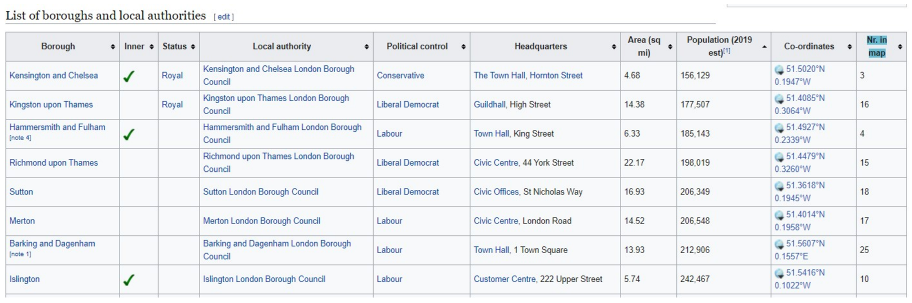
Figure 1: Rent data from London.gov website

Only latest available data is used, which at the time was first quarter of 2019. Also, we are going to use lower quartile, median, and upper quartile data instead of average to consider the skewed nature of averages. In addition, aggregate areas (e.g., 'North West') are excluded. Median rents for 2019 Q1 for the first few boroughs are shown below.



| Category | All categories | Four or more Bedrooms | One Bedroom | Room | Studio | Three Bedroom | Two Bedroom |
|---|---|---|---|---|---|---|---|
| Borough | | | | | | | |
| Barking and Dagenham | 1200 | 1650 | 950 | 650 | 730 | 1400 | 1200 |
| Barnet | 1365 | 2500 | 1150 | 588 | 897 | 1798 | 1400 |
| Bexley | 1100 | 1500 | 825 | 585 | 675 | 1300 | 1050 |
| Brent | 1500 | 2250 | 1250 | 602 | 867 | 1800 | 1499 |
| Bromley | 1225 | 2000 | 950 | 585 | 750 | 1500 | 1250 |
| Camden | 2000 | 4008 | 1625 | 845 | 1213 | 2925 | 2123 |
| City of London | 2210 | 0 | 1950 | 0 | 1723 | 0 | 2578 |
| Croydon | 1100 | 1900 | 900 | 500 | 750 | 1450 | 1200 |
| Ealing | 1375 | 2305 | 1200 | 600 | 900 | 1713 | 1431 |
| Enfield | 1275 | 2000 | 1050 | 550 | 800 | 1595 | 1300 |
| Greenwich | 1325 | 1800 | 1150 | 600 | 800 | 1550 | 1350 |
| Hackney | 1712 | 2842 | 1473 | 750 | 1125 | 2383 | 1777 |
| Hammersmith and Fulham | 1690 | 3792 | 1408 | 800 | 1101 | 2427 | 1798 |

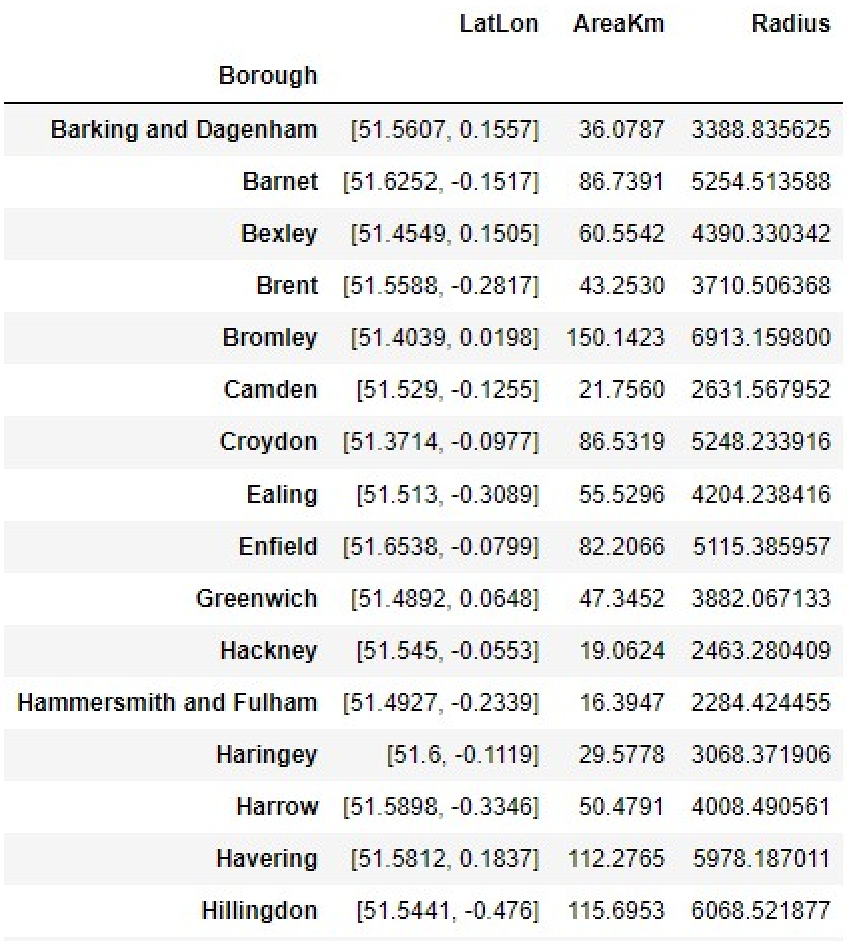
**Borough Data** In addition to rent data, we need to have some geographical data of the boroughs. Namely, we need to know the locations (i.e. center of the borough) as well as their boundaries. This will be required to determine which borough the venue belongs to.



List of boroughs and local authorities [ edit ]

| Borough | Inner | Status | Local authority | Political control | Headquarters | Area (sq mi) | Population (2019 est)[1] | Co-ordinates | Nr. in map |
|---|---|---|---|---|---|---|---|---|---|
| Kensington and Chelsea | ✓ | Royal | Kensington and Chelsea London Borough Council | Conservative | The Town Hall, Hornton Street | 4.68 | 156,129 | 51.5020°N 0.1947°W | 3 |
| Kingston upon Thames | | Royal | Kingston upon Thames London Borough Council | Liberal Democrat | Guildhall, High Street | 14.38 | 177,507 | 51.4085°N 0.3064°W | 16 |
| Hammersmith and Fulham [note 4] | ✓ | | Hammersmith and Fulham London Borough Council | Labour | Town Hall, King Street | 6.33 | 185,143 | 51.4927°N 0.2339°W | 4 |
| Richmond upon Thames | | | Richmond upon Thames London Borough Council | Liberal Democrat | Civic Centre, 44 York Street | 22.17 | 198,019 | 51.4479°N 0.3260°W | 15 |
| Sutton | | | Sutton London Borough Council | Liberal Democrat | Civic Offices, St Nicholas Way | 16.93 | 206,349 | 51.3618°N 0.1945°W | 18 |
| Merton | | | Merton London Borough Council | Labour | Civic Centre, London Road | 14.52 | 206,548 | 51.4014°N 0.1958°W | 17 |
| Barking and Dagenham [note 1] | | | Barking and Dagenham London Borough Council | Labour | Town Hall, 1 Town Square | 13.93 | 212,906 | 51.5607°N 0.1557°E | 25 |
| Islington | ✓ | | Islington London Borough Council | Labour | Customer Centre, 222 Upper Street | 5.74 | 242,467 | 51.5416°N 0.1022°W | 10 |

Information about boroughs and their geographical centers can be found on Wikipedia. The data from the provided table is retrieved using `BeautifulSoup` in Python and converted to a pandas DataFrame:



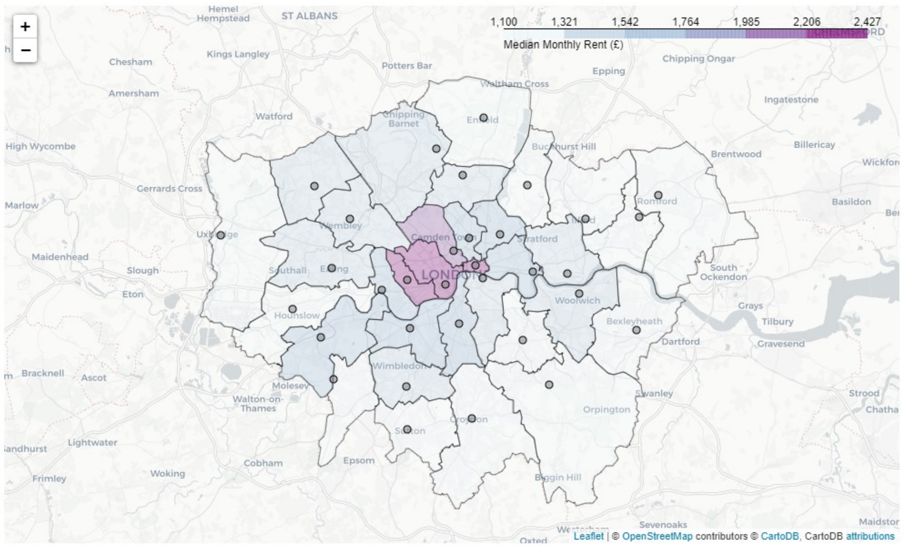
| | LatLon | AreaKm | Radius |
|---|---|---|---|
| **Borough** | | | |
| **Barking and Dagenham** | [51.5607, 0.1557] | 36.0787 | 3388.835625 |
| **Barnet** | [51.6252, -0.1517] | 86.7391 | 5254.513588 |
| **Bexley** | [51.4549, 0.1505] | 60.5542 | 4390.330342 |
| **Brent** | [51.5588, -0.2817] | 43.2530 | 3710.506368 |
| **Bromley** | [51.4039, 0.0198] | 150.1423 | 6913.159800 |
| **Camden** | [51.529, -0.1255] | 21.7560 | 2631.567952 |
| **Croydon** | [51.3714, -0.0977] | 86.5319 | 5248.233916 |
| **Ealing** | [51.513, -0.3089] | 55.5296 | 4204.238416 |
| **Enfield** | [51.6538, -0.0799] | 82.2066 | 5115.385957 |
| **Greenwich** | [51.4892, 0.0648] | 47.3452 | 3882.067133 |
| **Hackney** | [51.545, -0.0553] | 19.0624 | 2463.280409 |
| **Hammersmith and Fulham** | [51.4927, -0.2339] | 16.3947 | 2284.424455 |
| **Haringey** | [51.6, -0.1119] | 29.5778 | 3068.371906 |
| **Harrow** | [51.5898, -0.3346] | 50.4791 | 4008.490561 |
| **Havering** | [51.5812, 0.1837] | 112.2765 | 5978.187011 |
| **Hillingdon** | [51.5441, -0.476] | 115.6953 | 6068.521877 |

In addition, a theoretical radius of the borough is calculated using area:

$$r = \sqrt{\frac{S}{\pi}},$$

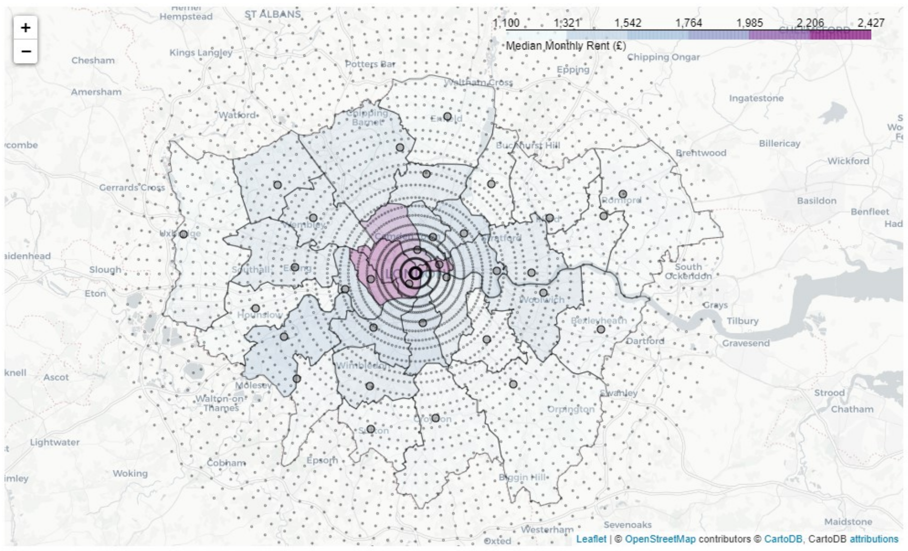
where $r$ is radius, $S$ is area, and $\pi$ is ~3.1459… This calculation of the radius approximates each borough as a circle and will be used in the next step when retrieving venue data.

The borough data combined with rent data can then be conveniently shown on a map:


1,100
1,321
1,542
1,764
1,985
2,206
2,427
Median Monthly Rent (£)
Leaflet | © OpenStreetMap contributors © CartoDB, CartoDB attributions

**Venue data** To characterize boroughs, we are going to use venue data available for free via Foursquare Places API. To retrieve the venues, we need to provide the API with the location (longitude and latitude) and the radius within which we want to get all the venues. Unfortunately, the free API version limits the results to 50 per request. Given that there are likely much more venues within each borough, a different method is required.

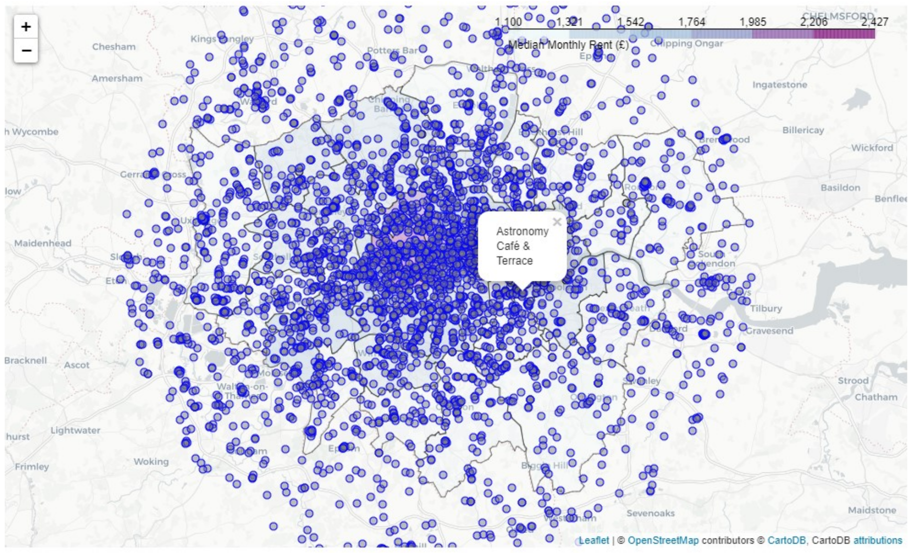
While this approach is not the most efficient, without a priori knowledge about the density of the venues the easiest solution was chosen - a grid scan of the entire London area. This was done by approximating the entire Greater London as a circle and dividing it into nine concentric rings. Each ring was divided into sub-rings depending on its location - more sub-rings in the inner rings, and less sub-rings in the outer rings. This was chosen assuming that the density of the venues is much larger in the central parts of London. The result of this procedure is a set of concentric rings that covers the entire Greater London area:


1,100
1,321
1,542
1,764
1,985
2,206
2,427
Median Monthly Rent (£)
Leaflet | © OpenStreetMap contributors © CartoDB, CartoDB attributions

We then create a point on each circle three degrees apart, which results in a grid of ~3500 locations that we then use to scan the entire area by iterating through each coordinate and sending it to Foursquare Places API's venues search endpoint.

At the end of the scan, we retrieve around 180,000 venues. However, as our search was not very efficient, a lot of them are duplicated. Dropping duplicate venues, we are left with 19,737 unique venues with the following information:

| CenterLatitude | CenterLongitude | Venue | Venue Latitude | Venue Longitude | Venue Category |
|---|---|---|---|---|---|
| 51.5074 | -0.119386 | Southbank Centre | 51.505843 | -0.116985 | Performing Arts Venue |
| 51.5074 | -0.119386 | Whitehall Gardens | 51.506354 | -0.122900 | Garden |
| 51.5074 | -0.119386 | Gordon's Wine Bar | 51.507911 | -0.123293 | Wine Bar |
| 51.5074 | -0.119386 | National Theatre | 51.507376 | -0.114793 | Theater |
| 51.5074 | -0.119386 | Royal Festival Hall | 51.505838 | -0.116667 | Concert Hall |



## Methodology

This section describes the methodology of the recommendation algorithm. The algorithm for this system is rather simple, but this is mainly because we do not have a large sample of user data to do, for example, collaborative filtering. Therefore, we are relying on explicit input from a single user.

**Data preparation** Each venue retrieved from the Foursquare API is mapped to a borough based on its coordinates and borough's boundaries. This is implemented using a computation geometry Python package `shapely`.

We then group each type of venue to nine large arbitrary groups:

1. **Going out** - bars, pubs, night clubs.
2. **Green spaces** - parks, gardens, trails, beaches, etc.
3. **Eating out** - restaurants, cafes, coffee shops, brasseries, etc.
4. **Entertainment** - theatres, museums, art galleries, zoos, bowling, etc.
5. **Shopping** - stores, malls, arcades, boutiques, shops.
6. **Health and Sports** - gyms, yoga studios, basketball courts, football courts, etc.
7. **Groceries** - supermarkets, bodegas, liquor stores, etc.
8. **Public Transport** - bus stops, tram stops, metro stops, train stations, etc.
9. **Other** - a catch-all category for everything else

We then create a representation of each borough by calculating normalized density vector $\boldsymbol{a}$ of venue groups, where each element is defined as:

$$a_i = \frac{n_i}{N},$$

where $n_i$ is the number of venues of the $i$th group in the borough, and N is total number of venues in the borough.

**Recommendation algorithm** The purpose of the algorithm is to recommend a list of boroughs to a user based on their specified preferences.

Let $W$ be a venue matrix of dimensions $n \times m$, where every element represents a normalized density of venue group $j$ in a borough $i$:

$$W = \begin{bmatrix} a_{11} & ... & a_{1m} \\ ... & a_{ij} & ... \\ a_{n1} & ... & a_{nm} \end{bmatrix}$$

The user-provided preferences can be formally written as a column vector $p$:

$$p = \begin{bmatrix} p_1 \\ ... \\ p_m \end{bmatrix}$$

Recommendation matrix $R$ then can be calculated as a matrix product:

$$R = W \times p = \begin{bmatrix} r_1 \\ ... \\ r_m \end{bmatrix}$$

This matrix is a column vector of length $n$ where each row represents the weight of each borough. The larger the weight, the closer the borough matches users preference.

Given a number $N$ of boroughs we want to recommend, $N$ largest weights are selected and the list of boroughs are returned in an order from the highest weigh $r$ to the lowest.

Implementation in Python:

```python
def create_preferences(ranking=None):
    "Converts `ranking` list to normalized pandas DataFrame"
    pref_dict = {
        'Eating out': 0,
        'Entertainment': 0,
        'Green spaces': 0,
        'Groceries': 0,
        'Health and Sports': 0,
        'Nightlife': 0,
        'Other': 0,
        'Public Transport': 0,
        'Shopping': 0,
    }

    N = len(ranking)

    for i, cat in enumerate(ranking):
        pref_dict[cat] = N - i

    df = pd.DataFrame.from_dict(pref_dict, orient='index', columns=['Preference'])
    df['Preference'] = df['Preference'] / df['Preference'].sum()

    return df

def recommend_boroughs(W=None, p=None, n_brghs=5):
    """
    Calculates matrix product of DataFrames W and p

    Inputs:
        W - pandas DataFrame containing venue group density for each borough
        p - pandas DataFrame with user group preferences
        n_brghs - int, number of boroughs to recommend

    Returns:
        rec_boroughs - `n_brghs` number of boroughs sorted by highest match value
        df_rec - resulting recommendation matrix as pandas DataFrame

    """
    W_ = W.copy()
    W_.set_index(['Borough'], inplace=True)
    df_rec = (p['Preference'] * W_).sum(axis=1).to_frame()
    df_rec.columns = ['Match']
    df_rec.sort_values(by='Match', ascending=False, inplace=True)
    rec_boroughs = df_rec.head(n_brghs).index.tolist()
    return rec_boroughs, df_rec
```

**Filtering venue matrix based** The venue density matrix $W$ is filtered in advance based on user's selection on upper and lower limits for rent: $m_{min}$ and $m_{max}$ and the accommodation type. The dataset for borough rent data contains first $q_1$ and third quartile $q_3$ information. Only the boroughs satisfying either of the below condition are kept in the $W$ matrix:

$$q_1 \leq m_{max} \wedge q_3 \geq m_{max} \tag{1}$$
$$q_1 \leq m_{min} \wedge q_3 \geq m_{min} \tag{2}$$
$$\tag{3}$$

Implementation in Python:

```python
def filter_rent_data(df=None, categories=None, rent_range=None):
    """
    Returns boroughs that satisfy the conditions for `categories` and `rent_range`

    Inputs:
        df - pandas DataFrame containing rent data
        categories - an iterable or a string specifying appropriate accommodation types
        rent_range - a list or a tuple with r_min and r_max rent ranges.

    Output:
        boroughs - a list of boroughs that match the condition

    """
    if isinstance(categories, str):
        cats = [categories]
    else:
        cats = categories

    cat_cond = df['Category'].isin(cats)
    rent_lower = rent_range[0]
    rent_higher = rent_range[1]

    # If invalid data provided
    if rent_lower > rent_higher:
        rent_higher = rent_lower

    rent_cond_1 = (df['Lower quartile'] <= rent_higher) & (df['Upper quartile'] >= rent_high
    rent_cond_2 = (df['Lower quartile'] <= rent_lower) & (df['Upper quartile'] >= rent_lower
    rent_cond = rent_cond_1 | rent_cond_2
    rent_cond = (df['Lower quartile'] <= rent_higher) & (df['Upper quartile'] >= rent_lower)

    not_null = ~df['Median'].isnull()

    df_filtered = df.loc[(cat_cond & rent_cond & not_null)]

    boroughs = df_filtered['Borough'].unique().tolist()

    return boroughs
```

Both of the above functions are then combined into one large function:

```
def recommend_and_plot(df_groups, df_rent, rent_range=None, acm_type=None, ranking=None, n=5
    """
    Recommends top n venues and plots the results on the map

    """
    available_boroughs = filter_rent_data(df=df_rent, categories=acm_type, rent_range=rent_r
    p = create_preferences(ranking=ranking)
    df_grp_filtered = df_groups.loc[df_groups['Borough'].isin(available_boroughs)]
    rec_boroughs, df_rec = recommend_boroughs(W=df_grp_filtered, p=p, n_brghs=n)

    # Printing and plotting of results
    # ...
```

## Results and discussion

We started with information about monthly rents for 33 London boroughs and combined it with data about venues within those boroughs retrieved from Foursquare API. Using grid-scanning we identified approx. 20,000 unique venues and grouped them into nine larger groups.

Plotting the boroughs and venues on the map showed that the distribution of the venues was not uniform and most of them were concentrated in the central parts of London. We assigned each venue to a borough based on its latitude and longitude, and boroughs geographical limits. By creating a normalized density vector of these venues for each borough, we effectively created a mathematical representation of the neighbourhood that could then be used in a recommendation algorithm.

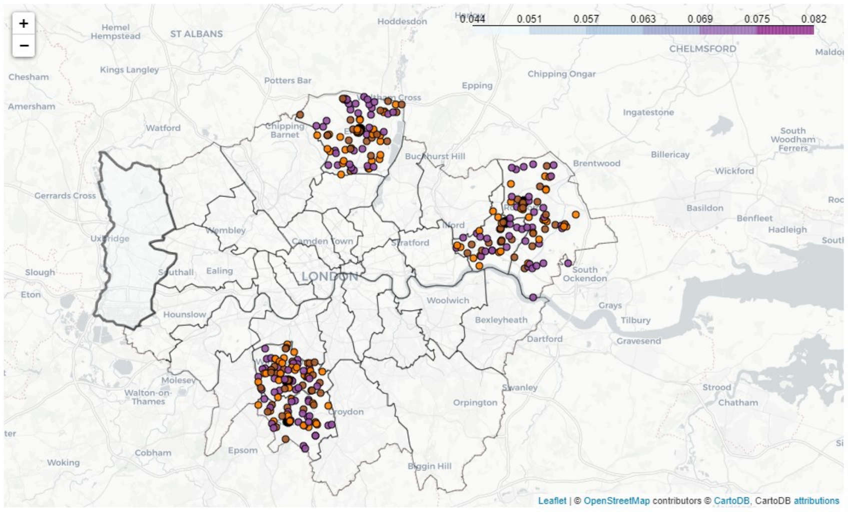
```
rent_range = [400, 800]
acm_type = ['Studio', 'One Bedroom']
group_rank = [
    'Public Transport', # 1
    'Groceries',        # 2
    'Green spaces'      # 3
]
```

```
recommend_and_plot(df_groups=df_groups_norm,
                   df_rent=df_rent,
                   rent_range=rent_range,
                   acm_type=acm_type,
                   ranking=group_rank,
                   n=5)
```

```
================================================================================
= Recommended boroughs based on your preferences
================================================================================
#1: Barking and Dagenham
#2: Sutton
#3: Merton
#4: Enfield
#5: Havering
================================================================================
```

Using accommodation and rent preferences, and combining this data with venue density matrices of the boroughs we created a recommendation function that displays an arbitrary, user-defined, number of recommended boroughs with preferred venues overlaid on a map.

Out[73]:


0.044
0.051
0.057
0.063
0.069
0.075
0.082
LONDON
Leaflet | © OpenStreetMap contributors © CartoDB, CartoDB attributions

This model is a good illustration of the general concept and could be extended to incorporate additional information. It could also be implemented as a user-friendly web application.

## Conclusion

The purpose of this project was to create an algorithm that recommends a London neighbourhood to a user based on their preferences. After cleaning data and performing exploratory data analysis, the process was synthesized to one function `recommend_and_plot` function which could be re-used in a dedicated application. This could be used to help people in the initial step of choosing where to live in London.

Admittedly, this is a simplistic model as there are way more important data points that could be used (e.g., crime rate, congestion, air quality, etc.). Nonetheless, the algorithm works and is a good demonstration of general principles that can be used for combining different types of information together to come up with a recommendation. While not perfect, this algorithm could serve as one of the steps in a person's decision-making process when looking for a new place to rent in London.