# Walking Areas of London for Foodies on a Budget

Applied Data Science Capstone Project Report

Author: Mita Naicker
Date: March 30, 2019

## 1.0 Introduction

London, UK, is arguably one of the greatest cities in the world. It is a tourist hotspot. Those especially interested in "gastronomic tourism" have a plethora of restaurants to choose from, boasting cuisines from all over the world. Choosing which areas of London/Greater to visit for the purpose of sampling great food could be challenging, especially on a budget.

**Scenario**: A budget-minded tourist has put 3 days of their trip aside for self-created foodie walking tours. Each day will be devoted to one area of London/Greater London. Each day, restaurants with very good ratings within that area will be highlighted. Overall the restaurants in each area together should have a low overall price range.  The goal is to visit at least 3 restaurants per day.

To summarize, the **question** I want answered is: What areas of London offer the best foodie experience for people on a budget?

**Audience**: anyone interested in a sampling food from well-rated restaurants in London without spending too much money. Additionally, the results will appeal to those interested in walking as a mode of transportation between restaurants.

In terms of area, my project will focus on a 7.5 km radius limit from London's city centre. The hope is to find clusters of restaurants in three different areas which meet my criteria of high ratings and low price.

## 2.0 Data

### 2.1 Data Sources

Two datasets were used to solve this problem: 1) Google Places API location data and 2) Foursquare API location data.

**a) Google Places API location data** The Places API "place search" was used to query restaurants within 7.5 km of London.

Data of interest:
Restaurant name
Address

Latitude/longitude
Rating
Price

**b) Foursquare Places API location data**

The Foursquare Places API was also used to query restaurants within 7.5 km of London's city center. Both the venue search and venue details requests were used to retrieve results.

Data of interest:
Restaurant name
Address
Latitude/longitude
Rating
Price

*** Note that it would have been nice to also have a restaurant category (e.g. "Chinese"), but that information was not available with the Google results.*

## 2.2 Data Acquisition

### 2.2.1 Google Place API

A "text search" request was set up to query for restaurants using the Google Places API. The following parameters were used:

searchtype = 'restaurant'
location = '51.5074,-0.1278' #London lat/long
radius = 3000

This was the final URL:
*'https://maps.googleapis.com/maps/api/place/textsearch/json?type={}&location={}&radius={}&key={}'.format(searchtype, location, radius, API_KEY)*

A radius of 3000 (3 km) around London was used instead of 7500 because there is a limit of 60 results per query. Additional requests were then made using 4 latitude/longitude points around London to gather more data. The Google Maps 'measure distance' tool was used to gather the latitude/longitude values about 4.5 km away in each of the 4 cardinal directions from London's latitude/longitude. This was not an exact/precise measurement in any way, but it did not need to be.

The following latitude/longitude values were used:

a) 51.5074,-0.1278 (London)
b) 51.507394, -0.062715 (approximately 4.5 km east of London)
c) 51.547920, -0.127121 (approximately 4.5 km north of London)
d) 51.507865, -0.192860 (approximately 4.5 km west of London)
e) 51.466985, -0.128849 (approximately 4.5 km south of London)

To be clear, the only difference between the 5 requests was the lat/long point used in the URL. The radius (3000), type (restaurant), and of course API key remained the same.

By using a radius of 3 km for each query, we end up with a final radius of 7.5 km around London. Ideally we would have also used the intermediate NW, SW, NE, SE points 4.5 km from London as well, but I deemed that 60x5 or 300 results was enough for this data source.

JSON Results

The JSON results from the initial request for each lat/long contained 20 results/restaurants. All the data I needed was contained in the JSON results (restaurant name, address, latitude, longitude, rating, price level).

*Snippet of JSON results:*


```
{'html_attributions': [],
 'next_page_token': 'CrQCIwEAADz5CnkD-7dy8r9lRPlnknyGqZyeffqcc48WLnjZZMj9687Ego3ev5sL28oGT1VB3t7DnUQA9v5-RPCCQnhszZGAoj5aAEZOtt-lF61RDZJfCY0l
xF71yA68Ti-sMKXBVY5d6hRvd2Xf_P16VcfoQ9PgmCLld_M0Q38_efWHZHEapuADdAz9rAzYYp_x3iVjPzN6tyKv5KvL2NwN7SOCDxkhWP_xA23urelzbZd40lmMc8xSfApWmmE6pZe_K
VGrG-2OhgKJkhJ16DRYRqramGEEliNnTbwqwjbyhv6SEuaVw6O8C5QHm6UJZFEfE0MgIMzCrVTiumodkrjAgXV_Tg_dblQtqQx8LYXD2rn3ulf9V54-D-AT6KFy4jHZZPEC5jKNHBnltA
IvVqvAh3X_ocUSEGEZ5TaTH4SocwOHbmI-xxwaFD6uvohalq3AuSXReAj3ipDslQkq',
 'results': [{'formatted_address': 'Old Quebec St, Marylebone, London W1H 7AF',
              'geometry': {'location': {'lat': 51.5143591, 'lng': -0.158102},
                           'viewport': {'northeast': {'lat': 51.51571322989273,
                                                      'lng': -0.1567020201072778},
                                        'southwest': {'lat': 51.51301357010728,
                                                      'lng': -0.1594016798927222}}},
              'icon': 'https://maps.gstatic.com/mapfiles/place_api/icons/restaurant-71.png',
              'id': '1c804b7795e51825493374700c71f7ef258795e5',
              'name': 'Zaman Restaurant',
              'opening_hours': {'open_now': True},
              'photos': [{'height': 2281,
                          'html_attributions': ['<a '
                                                'href="https://maps.google.com/maps/contrib/109921090613939177174/photos">A '
```


To get the next 20 results, the ‘next_page_token’ shown above had to be sent in a subsequent request. The results from that request again had a ‘next_page_token’, so one more URL was created to gather the last 20 results, for a total of 60 results per lat/long point (which is the maximum allowed per request).

The data was extracted into a pandas dataframe for further cleaning. Along with the required fields from the JSON Rresults, a “source” column was added to hold which API the results were from. 232 rows were added to the dataframe (note that venues/restaurants that did not have all of the required fields were skipped/not added to the dataframe).

Snapshot of googleresults_df:

| | Name | Rating | Price_Level | Latitude | Longitude | Address | Source |
|---|---|---|---|---|---|---|---|
| 185 | 222 Vegan Cuisine | 8.8 | 2 | 51.486023 | -0.202981 | 222 North End Rd, Hammersmith, London W14 9NU | googlemaps |
| 20 | 34 Mayfair | 8.8 | 4 | 51.510302 | -0.152263 | 34 Grosvenor Square Entrance on, S Audley St, ... | googlemaps |

### 2.2.2 Foursquare Places API

Three request options were explored with the Foursquare Places API:
GET https://api.foursquare.com/v2/venues/explore (Get venue recommendations)
GET https://api.foursquare.com/v2/venues/search (Search for venues)
GET https://api.foursquare.com/v2/venues/VENUE_ID (Get venue details)

Price and rating were only accessible with the "get venue details" request, so I had to use one of the first two in combination with it.

The "get venue recommendations" or "explore" option could be set up to limit the results to recommended 'food' venues (recommended meant generally better ratings), but returned bakeries, fast food venues, cafes, etc, so returned some results I was not interested in.

The "search for venues" option could be set up to only search for venues in specific categories (a list of restaurant categories), which was more appealing. Even though the ratings were probably not as high as the recommended/explore venues, at least all of my results were restaurants.

I created a "search for venues" request:

*'https://api.foursquare.com/v2/venues/search?client_id={}&client_secret={}&ll={},{}&v={}&radius={}&intent={}\&limit={}&categoryId={}'.format(CLIENT_ID, CLIENT_SECRET, london_lat, london_long, VERSION, radius, intent, LIMIT, categoryId)*

The results limit with this API was 50. As with the Google Places API, 5 requests were made using the same lat/long values as before, so there were potentially 250 results across the 5 requests. Only the venue_id was extracted. Once all of the venue_id's were collected, "get venue details" requests were made for each venue_id to retrieve the required details for each restaurant. Example of "Get venue" URL:

*'https://api.foursquare.com/v2/venues/{}?&client_id={}&client_secret={}&v={}'.format(venue_id, CLIENT_ID, CLIENT_SECRET, VERSION)*

The data was extracted into a pandas dataframe for further cleaning. Along with the required fields from the JSON results, a “source” column was added to hold which API the results were from. 216 rows were added to the dataframe (note that venues/restaurants that did not have all of the required fields were skipped/not added to the dataframe).

Snapshot of foursquare_df:

| | Name | Rating | Price_Level | Latitude | Longitude | Address | Source |
|---|---|---|---|---|---|---|---|
| 0 | Mezzah Lounge | 8.6 | 3 | 51.498668 | -0.162918 | 87–135 Brompton Rd (4th Fl.), London, Greater ... | foursquare |
| 1 | North Audley Cantine (NAC) | 9.2 | 2 | 51.512462 | -0.153264 | 41 North Audley St (at Lees Pl), London, Great... | foursquare |

## 2.3 Data Cleaning and Preparation

The google and foursquare dataframes were cleaned separately before they were concatenated.

### 2.3.1 Drop Duplicates

The name, latitude and longitude uniquely defined each row, so they were used to query for duplicates in both dataframes. I did expect duplicates in each dataframe as the search radiuses for each request did have some overlap.

```
#remove duplicate rows
df.drop_duplicates(['Name', 'Latitude', 'Longitude'], keep='last', inplace=True)
```

12 rows were dropped from googleresults_df.
20 rows were droppedfrom foursquare_df.

### 2.3.2 Cast Types

Initial exploration of both dataframes with ***df.info()*** revealed Price_Level was cast as an object (string). It needed to be numeric for statistical purposes.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 241 entries, 0 to 240
Data columns (total 9 columns):
Name                241 non-null object
Rating              241 non-null float64
Price_Level         241 non-null object
Latitude            241 non-null float64
Longitude           241 non-null float64
Address             241 non-null object
Postcode District   241 non-null object
Post Town           241 non-null object
API                 241 non-null object
dtypes: float64(3), object(6)
memory usage: 17.0+ KB
```

The price_level was changed from an object to a numeric column.

```
#cast price_level from object to numeric type
df[["Price_Level"]] = df[["Price_Level"]].apply(pd.to_numeric)
```

### 2.3.3 Adjust Google Ratings Values

The Foursquare ratings were out of 10.0, and the Google ratings out of 5.0. The Google ratings were multiplied by 2.0 to reflect the same range.

```
#multiple Google ratings by 2.0 to match Foursquare range
googleresults_df['Rating'] = 2.0*googleresults_df['Rating']
```

### 2.3.4 Missing Data

When initially moving data from the JSON to the dataframes, I came across errors due to missing price level and ratings fields. A **try/except** block was added to the code to handle these situations, so any results that had missing fields were not moved into the dataframe.

### 2.3.5 Concatenate DataFrames

Finally, both dataframes were concatenated to create 396 rows. The data was ready for further analysis.

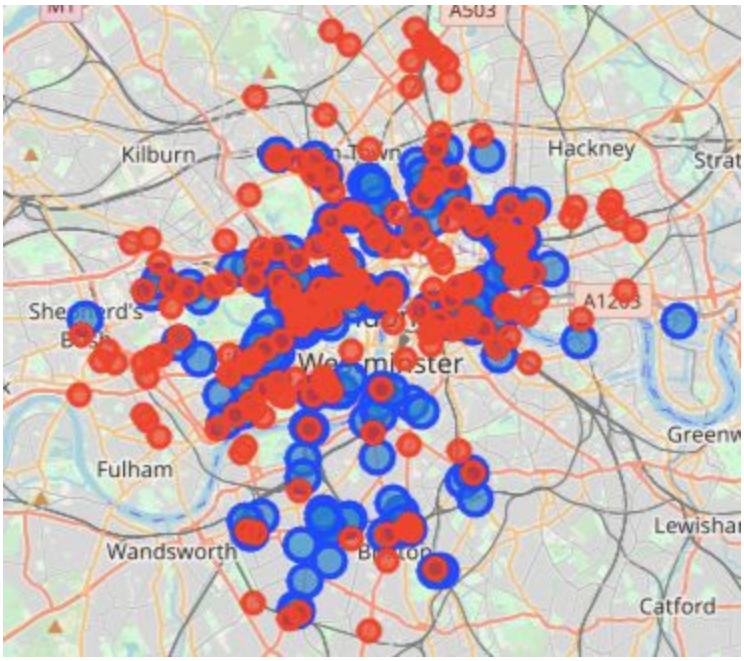
## 2.4 Methodology

### 2.4.1 Exploratory Data Analysis

#### 2.4.1.1 Visualize Data on Map

The Foursquare and Google data were plotted on a Folium map in different colours to visually compare the restaurant locations.  There did not seem to be much overlap between the two.

*Below: Map of Foursquare and Google Data (Foursquare data in larger blue markers and Google data in smaller red markers)*



Zooming into a denser area (below), I did see some duplicates, but not many.  Duplicates are ok to keep in the data as they do show the variance in the data (a restaurant could have different ratings between the 2 sources so it is important to keep this information).

*Below: Zooming into denser area reveals duplicates.*


### 2.4.1.2 Describe() Results

A describe() on each dataframe revealed that the Foursquare ratings had a larger range between ratings, and a much lower minimum value for rating than google.  This could mean that the Google API returned higher rated results by default, which might help explain the lack of overlap between the two sources.

*Below: foursquare_df.describe()*

| | Rating | Price_Level |
|---|---|---|
| **count** | 196.000000 | 196.000000 |
| **mean** | 8.319898 | 2.178571 |
| **std** | 0.822418 | 0.739889 |
| **min** | 5.800000 | 1.000000 |
| **25%** | 7.800000 | 2.000000 |
| **50%** | 8.500000 | 2.000000 |
| **75%** | 9.000000 | 3.000000 |
| **max** | 9.600000 | 4.000000 |

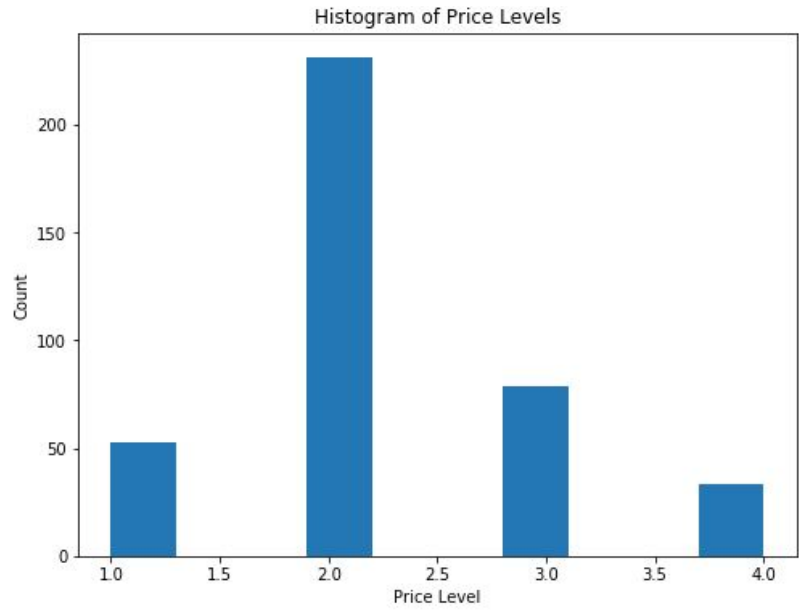
*googleresults_df.describe()*

| | Rating | Price_Level |
|---|---|---|
| **count** | 200.000000 | 200.000000 |
| **mean** | 8.893000 | 2.285000 |
| **std** | 0.339599 | 0.822898 |
| **min** | 8.000000 | 1.000000 |
| **25%** | 8.600000 | 2.000000 |
| **50%** | 9.000000 | 2.000000 |
| **75%** | 9.200000 | 3.000000 |
| **max** | 9.800000 | 4.000000 |

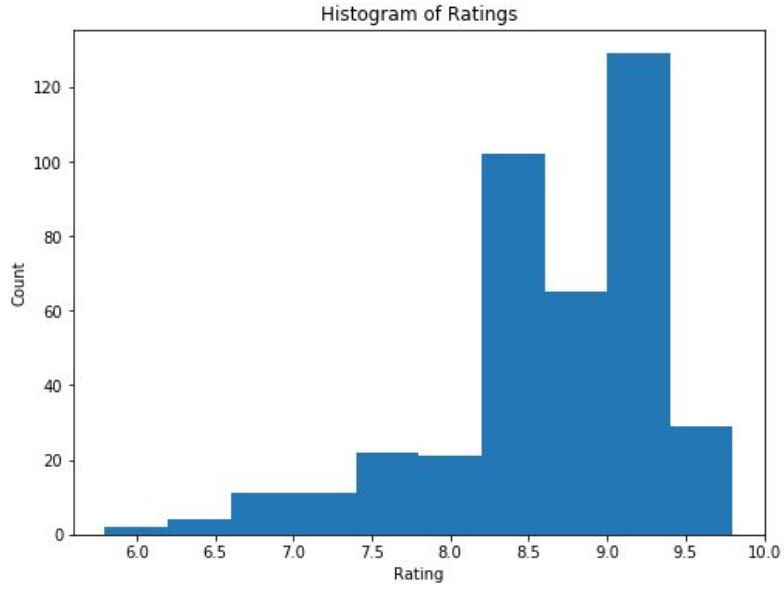
## 2.4.2 Inferential Statistical Testing

### 2.4.2.1 Histograms

The Google and Foursquare results dataframes were finally concatenated, into a new dataframe called df_all_rows.  Histograms of price_level and ratings were plotted.



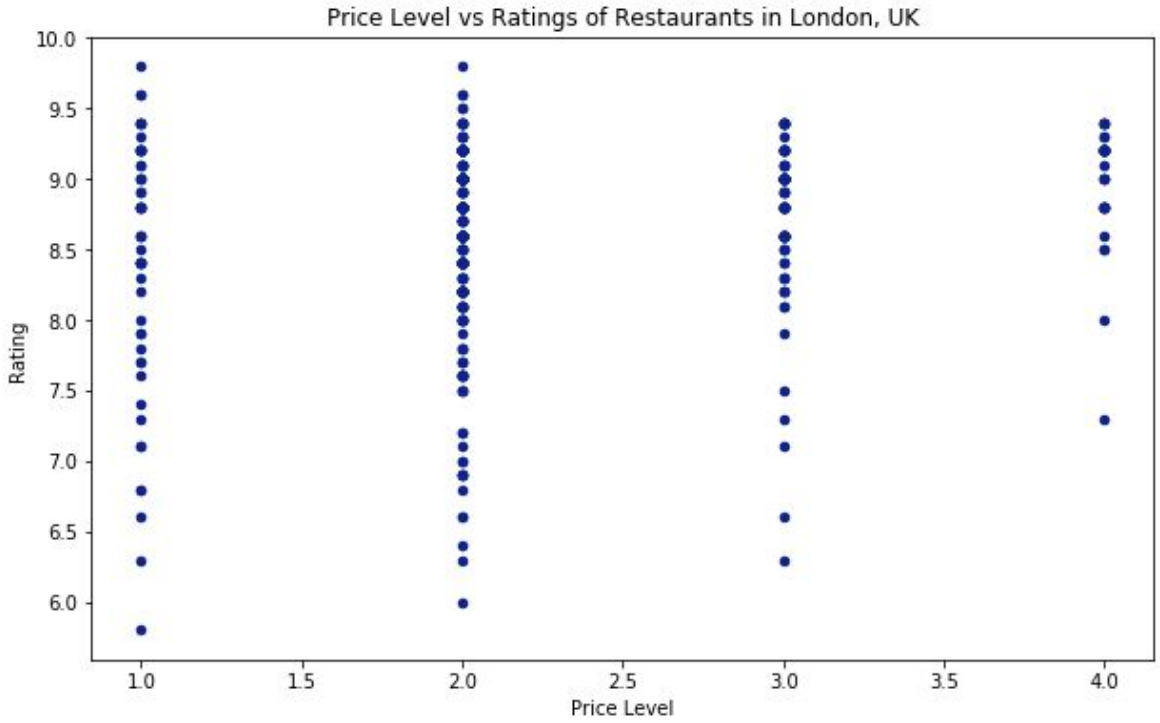

*Above: Histogram shows the highest count of price_level is 2.0.  This is encouraging.*




*Left: Histogram shows highest count of ratings is between about 9.0 and 9.4. This is also encouraging.*

#### 2.4.2.2 Scatterplot

A scatterplot of the results shows pleasing results, as we see the highest rated restaurants fall in the lower price ranges (below).




### 2.4.3 Machine Learning

#### 2.4.3.1 DBScan or K-Means?

I chose the DBSCAN clustering algorithm to generate my answer.  I chose DBCSAN over the other clustering algorithms, k-means in particular, because I was interested in the spatial distance between restaurants (recall the restaurants in each recommended area should be within walking distance).  DBSCAN is superior to k-means for spatial data because it clusters based on a physical distance between each point and a minimum cluster size. K-means minimizes for variance, not geodetic distance.

#### 2.4.3.2 DBScan Parameters

After working with various values for eps and min_samples, DBScan was run with the following parameters:

**Eps = 0.15** (The maximum distance between two samples for them to be considered as in the same neighborhood.) This values is in decimal degrees, and equates to roughly 3.5 km.

**Min_samples = 3** (The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself.)

Lower eps values resulted in more noise and insignificant clusters, while higher eps values resulted in fewer and larger cluster even though they had less noise. Larger min_sample values created more noise.

*Code for DBScan set up and execution*

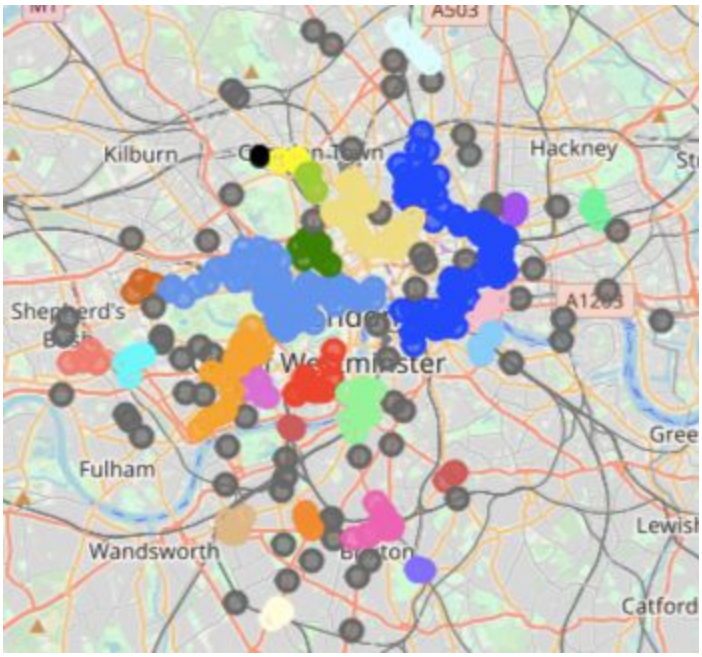
```
# Set up and run DBSCAN clustering

sklearn.utils.check_random_state(1000)
Clus_dataSet = df_all_rows[['Latitude','Longitude']]
Clus_dataSet = np.nan_to_num(Clus_dataSet)
Clus_dataSet = StandardScaler().fit_transform(Clus_dataSet)

#eps of 0.15 is about 3.5 km
db = DBSCAN(eps=0.15, min_samples=3, metric ='euclidean').fit(Clus_dataSet)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_
df_all_rows["Clus_Db"]=labels
```

## 2.5 Results

### 2.5.1 DBScan Clusters

Using DBScan with an eps = 0.15 and min_samples = 3 resulted in the following clusters. Three clusters (13, 14 and 25) show high average ratings and lower price levels. There were 67 data points that were considered noise and assigned cluster# = -1.

```
Cluster 0, Avg Rating: 8.76, Avg Price: 2.58
Cluster 1, Avg Rating: 8.41, Avg Price: 2.08
Cluster 2, Avg Rating: 8.55, Avg Price: 2.17
Cluster 3, Avg Rating: 8.83, Avg Price: 2.38
Cluster 4, Avg Rating: 8.36, Avg Price: 1.96
Cluster 5, Avg Rating: 8.81, Avg Price: 2.67
Cluster 6, Avg Rating: 8.9, Avg Price: 2.25
Cluster 7, Avg Rating: 8.01, Avg Price: 1.71
Cluster 8, Avg Rating: 8.56, Avg Price: 2.0
Cluster 9, Avg Rating: 8.87, Avg Price: 2.33
Cluster 10, Avg Rating: 7.92, Avg Price: 1.67
Cluster 11, Avg Rating: 8.4, Avg Price: 1.85
Cluster 12, Avg Rating: 8.9, Avg Price: 2.4
Cluster 13, Avg Rating: 9.05, Avg Price: 2.0
Cluster 14, Avg Rating: 9.2, Avg Price: 1.6
Cluster 15, Avg Rating: 7.9, Avg Price: 2.33
Cluster 16, Avg Rating: 8.35, Avg Price: 2.0
Cluster 17, Avg Rating: 8.83, Avg Price: 2.67
Cluster 18, Avg Rating: 8.2, Avg Price: 2.33
Cluster 19, Avg Rating: 8.8, Avg Price: 3.0
Cluster 20, Avg Rating: 8.75, Avg Price: 2.25
Cluster 21, Avg Rating: 7.4, Avg Price: 2.0
Cluster 22, Avg Rating: 8.83, Avg Price: 2.33
Cluster 23, Avg Rating: 8.5, Avg Price: 1.33
Cluster 24, Avg Rating: 8.87, Avg Price: 2.0
Cluster 25, Avg Rating: 9.0, Avg Price: 1.67
[13, 14, 25]
```




Left: Map view of resulting clusters. Grey circles represent cluster# = -1 (noise).

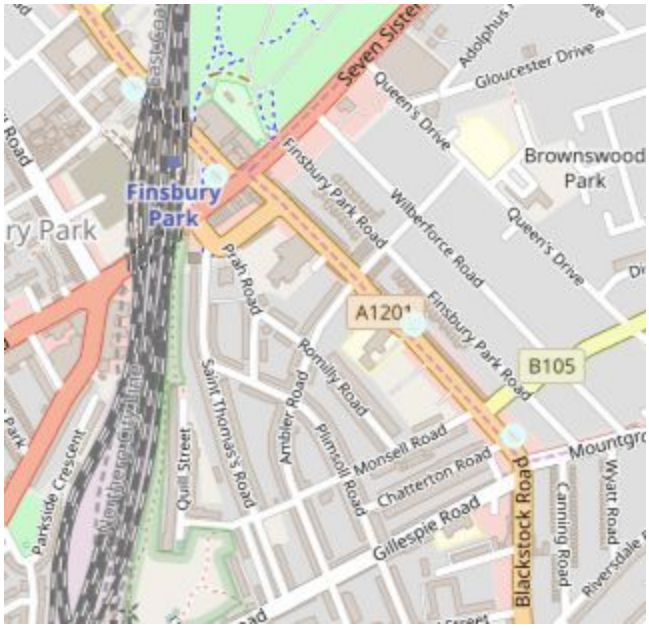
### 2.5.1 Clusters Meeting Criteria

The data for the clusters that met my criteria (price_level <= 2.0 and rating >= 9.0) were extracted into a new dataframe. The resulting three clusters were in 3 different areas of London, and contained between 3 to 5 recommended restaurants.

| | Name | Rating | Price_Level | Latitude | Longitude | Address | Source | Clus_Db |
|---|---|---|---|---|---|---|---|---|
| **37** | Chez Abir Lebanese Restaurant | 9.0 | 2 | 51.497064 | -0.212609 | 34 Blythe Rd, Hammersmith, London W14 0HA | googlemaps | 13 |
| **108** | Los Molinos | 8.8 | 2 | 51.497394 | -0.222459 | 127 Shepherds Bush Rd, Hammersmith, London W6 7LP | googlemaps | 13 |
| **140** | Pentolina | 9.6 | 2 | 51.497612 | -0.217451 | 71 Blythe Rd, Hammersmith, London W14 0HP | googlemaps | 13 |
| **173** | The Bird in Hand | 8.8 | 2 | 51.499733 | -0.215419 | 88 Masbro Road, Masbro Rd, London, Brook Green... | googlemaps | 13 |
| **55** | Dotori | 9.2 | 2 | 51.564676 | -0.105179 | 3 Stroud Green Rd, Finsbury Park, London N4 2DQ | googlemaps | 14 |
| **56** | E-Mono | 9.2 | 1 | 51.565849 | -0.106959 | 13 Stroud Green Rd, Finsbury Park, London N4 2AL | googlemaps | 14 |
| **90** | Il Cavaliere Italian Restaurant | 9.4 | 1 | 51.562691 | -0.100947 | 81 Blackstock Rd, Finsbury Park, London N4 2JW | googlemaps | 14 |
| **141** | Petek Restaurant | 9.2 | 2 | 51.568405 | -0.110101 | 96 Stroud Green Rd, Stroud Green, London N4 3EN | googlemaps | 14 |
| **199** | Yildiz Restaurant | 9.0 | 2 | 51.561209 | -0.098796 | 163 Blackstock Rd, Finsbury Park, London N4 2JS | googlemaps | 14 |
| **13** | Asakusa | 8.8 | 2 | 51.534021 | -0.138289 | 265 Eversholt St, Kings Cross, London NW1 1BA | googlemaps | 25 |
| **50** | Daphne | 9.6 | 1 | 51.537759 | -0.140337 | 83 Bayham St, Camden Town, London NW1 0AG | googlemaps | 25 |
| **353** | The Blues Kitchen | 8.6 | 2 | 51.537182 | -0.141099 | 111-113 Camden High St, Camden Town, Greater L... | foursquare | 25 |

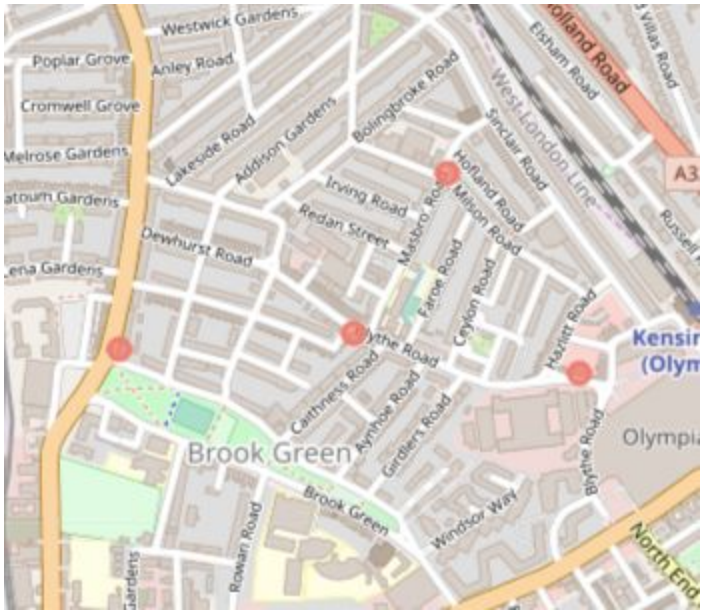
*Below: The 3 clusters plotted on a map.*


*Below: 4 restaurants in cluster #14 (Finsbury Park area)*



*Below: 3 restaurants in cluster #25 (Camden Town area)*


*Below: 4 restaurants in cluster #13 (Hammersmith area)*



## 2.6 Discussion

### 2.6.1 DBScan ‘metric’ parameter

DBScan was run using euclidean distance (straight line between 2 points) for the ‘metric’ parameter. The earth is more of a sphere, and the ‘great circle’ or orthodromic distance is the shortest distance between 2 points on a sphere. I did not see a specific option for this as a valid value for the metric parameter. However, it could be worth exploring if any of the valid options shown below are in fact orthodromic distances, and if so, if this would make a difference in the results.

Options for metric parameter:

metric : *str or function, optional*

The distance metric to use. The distance function can be ‘braycurtis’, ‘canberra’, ‘chebyshev’, ‘cityblock’, ‘correlation’, ‘cosine’, ‘dice’, ‘euclidean’, ‘hamming’, ‘jaccard’, ‘jensenshannon’, ‘kulsinski’, ‘mahalanobis’, ‘matching’, ‘minkowski’, ‘rogerstanimoto’, ‘russellrao’, ‘seuclidean’, ‘sokalmichener’, ‘sokalsneath’, ‘sqeuclidean’, ‘yule’.

### 2.6.2 DBScan vs K-Means

My reasons for selecting DBScan was outlined. However, it would still be interesting to see what K-Means produces. The postcode district could easily be extracted from the address field, so the results could be grouped by postcode district to still get the spatial aspect required for this project.

### 2.6.2 Future Direction

I gathered data from two of the larger user-rated location-based API’s in order to analyze data from almost 400 restaurants. TripAdvisor is another popular source for user-based ratings, but it did not have a free API available, and I could not scrape data from the search results past the first page of results. If there is a way to gather data from TripAdvisor it would add more credence to the results.

Also, I did not consider any other venues of interest as my focus was on restaurants only. However, for tourists in particular, other venues of interest could extracted for each area chosen to help rank which ones are more appealing overall.

## 2.7 Conclusion

To recap, the question asked was: **What areas of London offer the best foodie experience for people on a budget?**

The main criteria was to find three areas that offered: high rated restaurants, low prices, and walking distance between restaurants.  To goal was to visit at least 3 restaurants in each area.

The statistical testing and exploratory data analysis showed promise that the question would be answered.  The clustering algorithm, DBScan, did in fact answer the question.  The Hammersmith, Finsbury Park and Camden Town areas of Greater London offer up some fantastic cheap eats! Plus, calories can be burned walking in between the restaurants, so there no need to feel guilty by indulging a little extra.