Location scouting with Uber H3 index and Foursquare Data

1. Introduction

Have you ever wondered how restaurant chains like McDonald's or Starbucks may always happen to be located in a perfect location? They use geographic and demographic data to build so-called geographic information systems (or "GIS"). These systems enable them to analyze locations based on data and choose the ones that benefit their businesses the most.

While the idea sounds simple, it is quite difficult to build a respective analysis tool as it requires vast amounts of data as well as interdisciplinary expertise to get the system to a point it provides some meaningful insights. GIS is therefore so far only accessible for larger brands that have the resources to build and customize them.

Smaller restaurant chains on the other hand still often heavily rely on manual location scouting which requires a lot of time and is often based on the experience of real estate agents.

While this can be successful, it bears a lot of risks as the agents most likely provide a subjective view only. As a result, it takes

smaller restaurant chains much longer to expand which makes it more difficult for them to compete. What if I tell you that it doesn't have to be this way?

In this story, we will explore a low-budget approach to build our GIS that can help smaller restaurant chains to make better-informed decisions and speed up their location scouting.

2. The business problem

As already indicated above, we want to help a smaller restaurant chain expand by building a customized GIS for them.

To make this a real-life scenario, we will use a small to a medium-sized Italian restaurant chain called L'Osteria. Their restaurants are extremely popular in Germany as they offer quality Italian food as well as good drinks for a reasonable price.

They also don't look like the typical Italian restaurant around the corner as their restaurants have a more modern design that is especially appealing to younger generations.

Founded in Nuremberg in 2015, they already managed to expand to over 100 restaurants within Germany. Overall it is an extremely successful concept.

However, now the chain looks to expand internationally. They have already opened a few restaurants in the Netherlands and United Kingdom. While location scouting itself is a stressful and

time-consuming task, it is even more complicated when you have to do it in a foreign country.

To reduce the stress this might put on their organisation, let's help them a little by building a GIS that learns from their existing restaurants and applies these insights to any other city on earth. Yes, you read right — **any** other city.

3. The Data

Excited already? Good! But unfortunately, we will have to do some groundwork first. So let us talk about data. To build our low-budget GIS, we have to get data that fulfils the following criteria:

- The data should be **meaningful** to our use case
- It should be **for free**
- And it should be **computable** in a reasonable amount of time with a standard laptop you can buy of the shelf

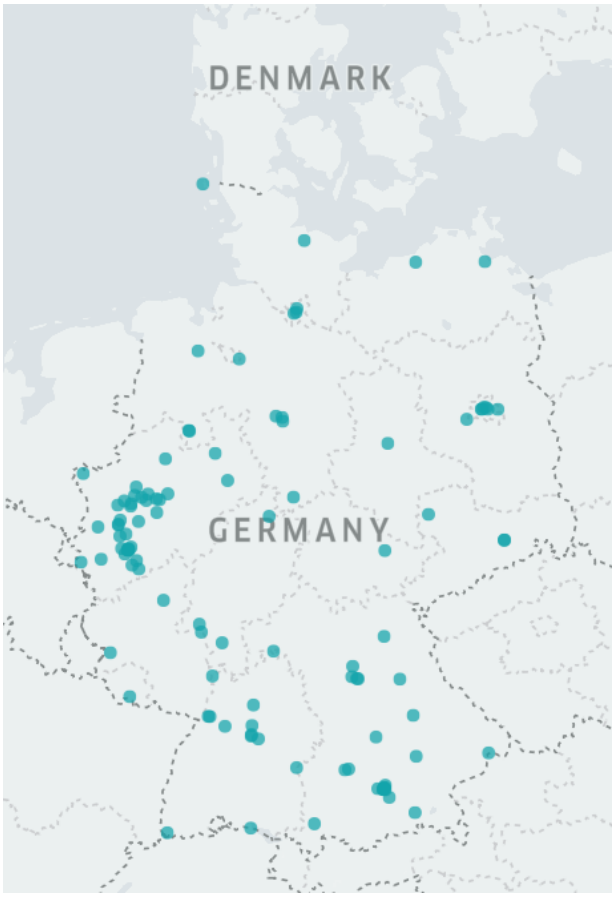
That doesn't leave us with a lot of options. Luckily though, we can be creative.

3.1 L'Osteria Data

At this point, I need to note that L'Osteria is not paying me anything to conduct this analysis nor do I have access to their data. That makes our task extremely difficult since we want to customize our GIS to their restaurants. So this is where our creativity comes into play — we will simply scrape data from their website!

Using the Beautifulsoup library, we can download the addresses of their Germany based restaurants from their website and transform this list to receive a Pandas dataframe.

As the data on their website does not contain any geographic coordinates, we have to get this information differently by reverse-engineering the addresses.



We can do this by using Nominatim API that we can access via the Geopy library. After scraping, reverse engineering and some further standardization we receive an initial dataset with 107 restaurants.

3.2 Uber H3 Index Data

I was lucky enough to travel a lot in my youth but it is still impossible for me to tell whether a restaurant is located in a trendy area or not. To draw any conclusions on the areas the restaurants are located in, I would have to google their locations and research the respective city as a whole.

However, we have over 100 restaurants in our dataset. Conducting a respective analysis would take me weeks.

Since this isn't resource-efficient, we have to find a way of speeding things up. To analyse the location of each restaurant,

we need to define a radius from which we want to conclude from.

One approach would be to get shapefiles for each neighbourhood in each city L'Osteria has a restaurant in and then download information related to these neighbourhoods. But good luck trying to get these shapefiles for Germany! Here, every federal state has its data service — one more confusing than the other. Collecting the necessary data would take weeks and provide us with a minimum granularity. So that is not an option.

This is where Uber's H3 hexagonal mapping framework comes into play. At its core, H3 is a geospatial analysis tool that provides a global index of hierarchically ordered hexagons.

Just imagine that you would map the entire earth (pole to pole) in hexagonal polygons. Now, each hexagon would receive a unique ID so that you could exactly tell what place on earth belongs to which hexagon.

Finally, think of different layers of hexagons. Each layer is much smaller and more precise, enabling you to drill up and down on your map. That is exactly what H3 does — it provides you unique hexagons with different resolutions, all the way down to a precise 0.0000009 km^2 area.

So how is this helping us? By knowing the GPS coordinates of the L'Osteria restaurants, we can assign them to a unique

hexagon. This hexagon will then serve as our designated radius for which we will download further data for our analysis.

The H3 framework also helps us to map any city on earth with the same size of hexagons, making it possible for us to scale our approach to any other city on earth.

To get there, we need to map the coordinates of our L'Osteria restaurants against the H3 index. To do so, we can use the H3 library by Uber. The library provides a function that does the mapping for us if we can provide longitude and latitude geolocations.

However, besides providing the coordinates, we also need to define the resolution of the hexagon we want to assign our coordinates. The H3 index has 16 unique resolutions that we can choose from:

H3 Resolution	Average Hexagon Area (km2)	Average Hexagon Edge Length (km)	Number of unique indexes
0	4.250547e+06	1107.712591	122
1	6.072210e+05	418.676005	842
2	8.674585e+04	158.244656	5882
3	1.239226e+04	59.810858	41162
4	1.770324e+03	22.606379	288122
5	2.529034e+02	8.544408	2016842
6	3.612905e+01	3.229483	14117882
7	5.161293e+00	1.220630	98825162
8	7.373276e-01	0.461355	691776122
9	1.053325e-01	0.174376	4842432842
10	1.504750e-02	0.065908	33897029882
11	2.149600e-03	0.024911	237279209162
12	3.071000e-04	0.009416	1660954464122
13	4.390000e-05	0.003560	11626681248842
14	6.300000e-06	0.001349	81386768741882
15	9.000000e-07	0.000510	569707381193162

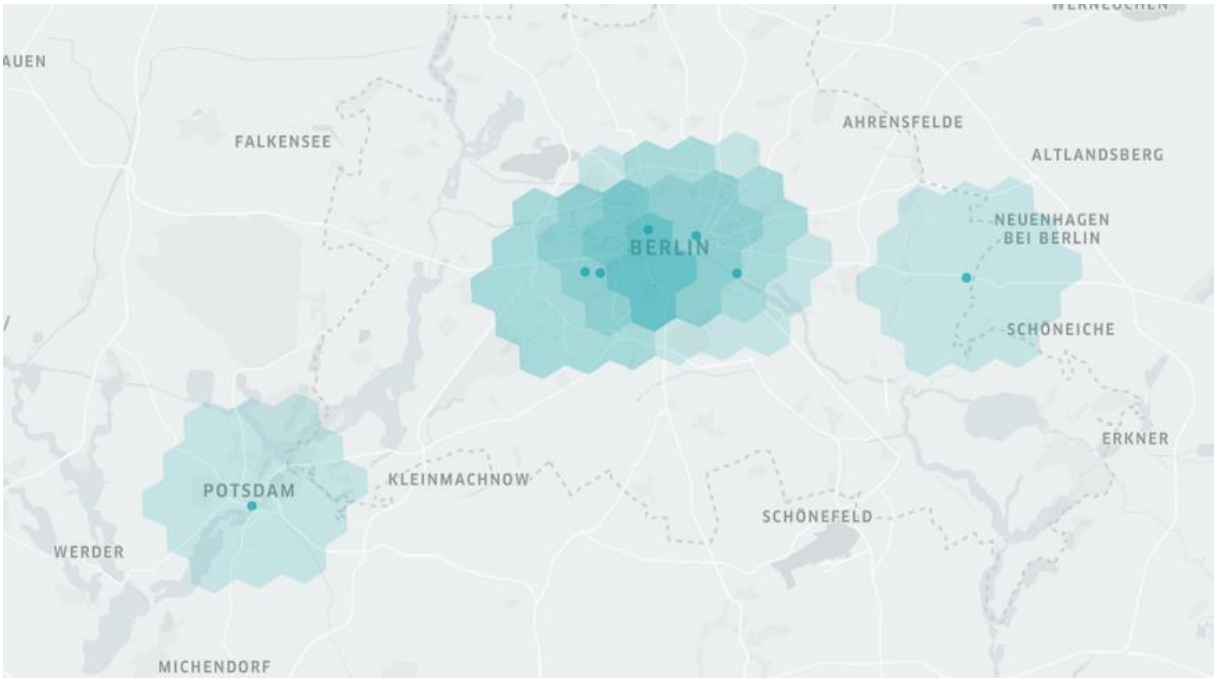
As described above, the resolution goes all the way down to 0.0000009 km² or 0.9 m² respectively. That is extremely small and would not serve our purpose well as we aim to analyze larger areas around our restaurants.

After visualising and mapping the hexagons against our locations, I have decided that resolution 9 seemed to be a good match in most cases. The following picture shows you how that would look like in the case of Berlin-based restaurants.

Did you notice that in one case the restaurant is not located in the centre of the H3 hexagon but pretty much at its edge? That is because the H3 index does not centre around our locations but merely assigns them to pre-defined hexagons.

Now that is an issue for us since it reflects our restaurants' catchment areas poorly. We, therefore, expand our mapping and

include the neighbouring hexagons with a degree of two (we are adding directly neighbouring hexagons and then their neighbours) as well. As a result, we receive hexagon clusters that mimic our catchment areas much better.



3.4 OpenStreetMap Data

The H3 index is covering the whole world. Downloading data for each hexagon individually and then analyzing them based on their coordinates would eat up tremendous resources. So we need to become creative again.

To reduce the hexagons to the area of our interest, we need to introduce some geographical boundaries. We can do that by downloading a respective shapefile from a free to use online service called GeoFabrik which is part of the international OpenStreetMap project. Here, you can find shapefiles of all larger cities, countries or regions on earth.

In our case, I have chosen London as our target city. To implement it in our GIS, we need to download its shapefile. However, before we can use it, we need to apply some more transformation since, in the case of London, there is another exception. London (as many other cities) has an area within its city that has special political rights.

In the case of London, this area is called the "City of London". As the GeoFabrik follows official guidelines and borders, it excludes these areas from its shapefiles. As a result, London's shapefile has a hole in the middle.

Since we want to analyze the entire city, we need to transform our shapefile to get rid of the doughnut hole. We can do that by using the exterior polygon function in the Shapely library. It allows us to focus on the exterior boundaries of our shape and create a new shape from it that does not contain the doughnut hole.

We can then use the polyfill function provided by the H3 Python library to fill up the shapefile with hexagons that lie within its boundaries. That, however, leaves us with another challenge. Our border does not fit 100% to the H3 index as well. Consequently, our H3 hexagons are not covering all areas near the shape's border.

While this might seem minor, the uncovered areas are summing up to quite a large amount. We will therefore append neighbouring hexagons with a degree of two (the neighbour of the neighbour) for each hexagon that is at the edge of our shapefile. As a result, we receive a dataset with hexagons that cover the entire area of London.

3.5 Foursquare Data

To add data to our hexagons, we will use Foursquare which is an app that lets its users rank all different kinds of venues all over the world. By doing so, Foursquare has built one of the most comprehensive geospatial data sets worldwide.

The data set covers an extremely large amount of venue and user data that allows you to access everything from a venue's name, location, or menu up to user ratings, comments or pictures of each venue. And the best part: Foursquare provides developers with an API that is (in limits) free to use!

Even though Foursquare provides an incredible variety of data, we will use the venue categories (e.g., sushi restaurant, park, sports club, Italian restaurant, etc.) for our analysis only. While it would most likely increase the success of our GIS to include more information such as rankings or types of user data, including this information would also require much more computing resources. Since we do not have these resources, we will not leave this data untouched for now.

We will use the Foursquare venue data to analyze the area of the hexagons (or neighbouring hexagons) in which L'Osteria has existing restaurants as well as the hexagons of London. This way we will be able to cluster the hexagons according to their characteristics at a later stage in this project.

To download data from the Foursquare API, we need to provide some settings:

- **Client credentials**
  We have to provide our client id, secret and access token that we have created within our Foursquare developer account. Foursquare requires this information to match the request with an existing account.

- **Latitude & longitude:**
  We need to provide the latitude and longitude from where we want to download data from.

- **Version:**
  We need to specify the version of the API we want to use. In our case, we will stick to the versioning '20180605'.

- **Radius:**
  The API requires us to state a radius from which we want to download data. Picking the right radius is crucial as a small radius will miss data and a very large radius can potentially double it. As a result, our algorithm would be biased. So what's the correct radius? According to the H3 documentation, a hexagon with a resolution of 9 has a radius (based on its centre) of 0.174375668km. Since the API requires the input in meters, we need to convert this figure to 174.375668m.

- **Limit:**
  The API downloads all venues within the given radius. The limit setting allows us to state the maximum of venues we want to download. As the London dataset includes more than 20k hexagons, a high limit (e.g. 100) would result in very large amounts of data. As this approach is supposed to be replicable with standard hardware, we will set the limit to 5 venues in our radius. That would give us approximately more than 100k data points to work with which should be enough for an algorithm to handle on our vintage laptops.

Finally, we need to talk about another limit — the API call limit. Foursquare data is generally for free but the API limits the calls for a regular user to 950 calls per day. A premium user with a personal subscription plan is allowed to make up to 99,500 calls per day.

Both, regular and premium users, are limited to a 500 calls per hour limit. Our venue data set has over 20,000 unique hexagons. With a regular user's call limit, we would need ~21 days to extract the information. Since that is way too long, I have upgraded my account to a personal non-commercial account. This plan is for free as well, however, you need to verify yourself and provide a credit card.

While this solves our problem with the daily limit, we still have the issue with the hourly limit. To overcome this, we need to partition our dataset so that it contains a maximum of 500 rows each.

We then need to install a timer that pauses our program for 1 hour before the next partition is downloaded. Since we have to wait one hour in between each download process, the program will run over 40 hours in total.

To avoid any data losses during the process, we will save the information for each partition as a CSV file. After all, partitions have been downloaded, the datasets are merged and we can continue. As a result, we receive the following dataset.

PS: If you do not want to download the venue data yourself, you can find it in my repository.

(101748, 6)

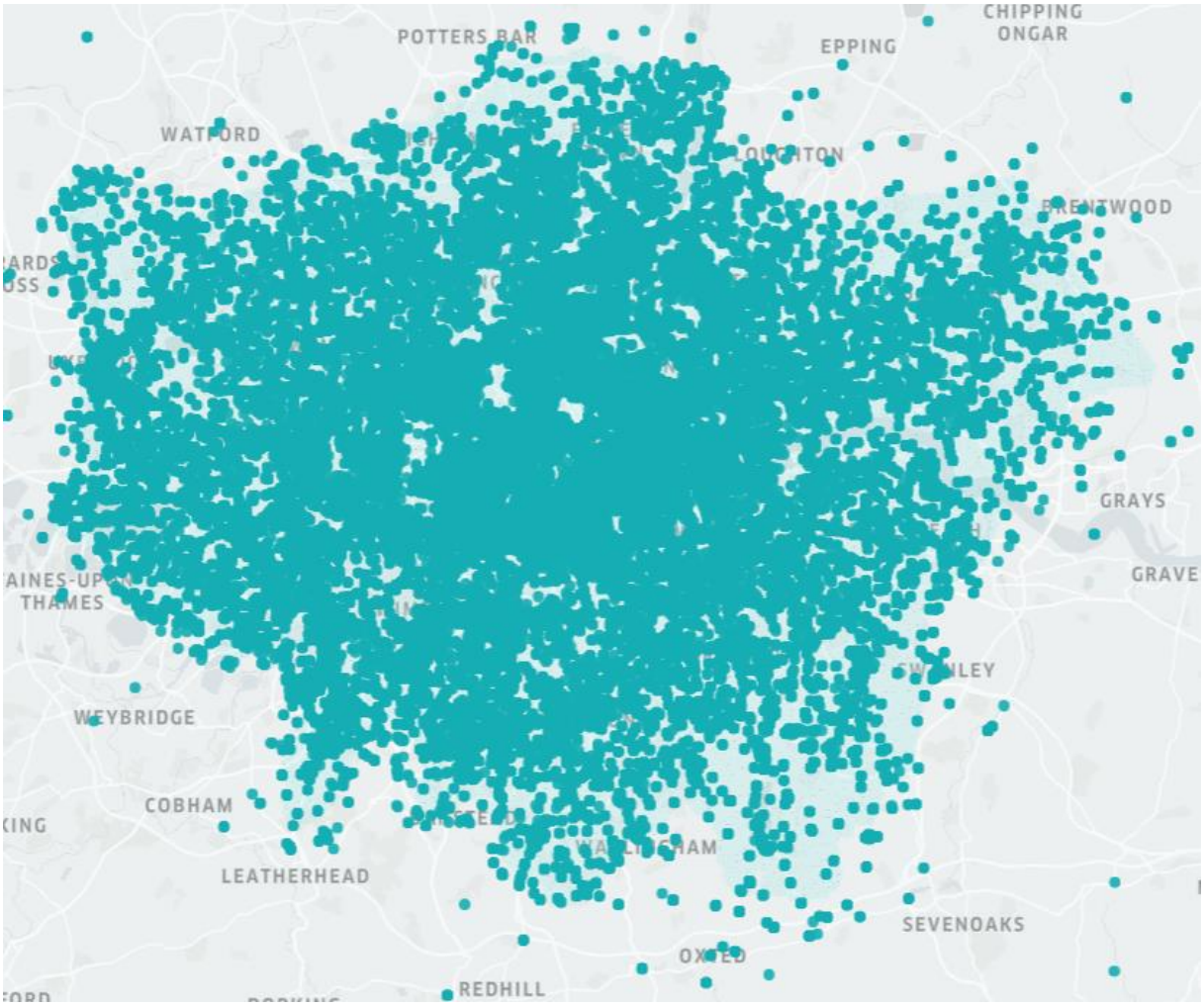
	Unnamed: 0	name	categories	lat	lng	H3_INDEX
0	0	Rackspace University	College Classroom	51.502441	-0.429112	89195d3648bffff
1	1	Rackspace Gym	Gym	51.501777	-0.428416	89195d3648bffff
2	2	The Hard Rack	Corporate Cafeteria	51.501697	-0.427958	89195d3648bffff
3	3	Rackspace	Office	51.501678	-0.427785	89195d3648bffff
4	4	BTC Group	Office	51.501377	-0.429892	89195d3648bffff

When we map our data points, we can observe that it seems like we covered most of the city's area with our venue data. We can see that there are some blank spots, especially in the east and south of the city. That makes sense as these areas are majorly inhabited and covered with farms or forests.

In addition, we can see that there seem to be some large uncovered areas directly within the city. One such area is St. John's Wood or Clerkenwell.

According to Google and Wikipedia, these districts are quite fancy/trendy so one would assume that Foursquare should have something there in the database. But apparently, nobody in these districts is a Foursquare user, nor anybody who is using Foursquare seems to go there.

Maybe there is an interesting relationship to be discovered here? But that is out of scope for this exercise, so we will accept the blank spots and move on.



When you have a look at the city's border, you will notice that we have venues in our dataset that are very far away from our hexagonal grid. That should not be possible as we have parsed a specific radius and GPS coordinates into the Foursquare API.

When we zoom out, we can see that this issue is much larger. Our dataset includes venue data up to Manchester. These outliers indicate, that the API is not working well. As the number of outliers seems to be pretty small, we will accept them for now and not conduct any further cleaning.

When zooming further into the map, we can see that there are a lot of hexagons that show no entries at all while there are some hexagons that have more than 5 venues assigned.

Now, the latter is a problem as this indicates that we might have double entries which could bias our algorithm greatly. This indicates that the API/database is indeed not entirely reliable.

However, it could also mean that we parsed the wrong parameters for the radius. Consequently, our radii would

overlap. However, this would not matter so much as long as we do not have any double entries in our dataset.

We can observe the same behaviour in our Germany-based venue data. This part of the dataset includes outliers and blank spots as well.

As with the London-based data, we can accept these issues as long as there are no double entries. Luckily for us, there are no double entries in our dataset and we can continue with the next step.

4. The Methodology

Alright, we have our dataset! Let us jump to the heart of our GIS — designing an algorithm that helps us to analyze the gathered data. For this exercise, we will use a simple k-means algorithm that will help us to cluster our dataset. The idea behind it is very simple:

We train our algorithm on both German and London venue data. We will then analyze the resulting clusters for common features between London and German-based data. All clusters that show few similarities can be excluded, leaving us with clusters that are potentially interesting for setting up new restaurants.

For our algorithm, we will use the k-means clustering library provided by scikit learn. This library requires a very specific form of input which means that we have to do some further data transformation. We can do so by applying one hot encoding which will transpose our existing dataframe and calculate how often each venue is appearing in each catchment area. After applying one hot encoding, our dataset will look like this:

CATCHMENT_ID	ATM	Accessories Store	Acupuncturist	Adult Boutique	Adult Education Center	Advertising Agency	Afghan Restaurant	African Restaurant	Airport	...
02e864a1-2d80-11ec-aa44-0433c21e09d2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
02e88b98-2d80-11ec-ae9b-0433c21e09d2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
02e88b99-2d80-11ec-b908-0433c21e09d2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
02e88b9a-2d80-11ec-94ec-0433c21e09d2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
02e88b9b-2d80-11ec-b0f8-0433c21e09d2	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

Now, before we jump to training our algorithm, we need to add another preprocessing step: Feature scaling.

Wikipedia defines feature scaling as “a method used to normalize the range of independent variables or features of data”. So what does that mean? The distance between our data points often differs widely which can cause some objective functions of machine learning algorithms to return false output. This is especially the case with classifiers that use the Euclidean distance which (in short) is the length of a line segment between two points.

Now, the wider the range of data points, within one feature, the higher the impact of that feature on the overall classification will be. Standard scaling now helps us to “normalize” these distances and thus adjust the impact of individual features. After scaling our features, each feature is supposed to contribute proportionally to the final result.

Since the k-means algorithm runs by default on the Euclidean distance, this step is very important for us. Luckily, scaling our features is very simple as we can use the StandardScaler library by Scikit. After feature scaling our dataset will look like this:

ATM	Accessories Store	Acupuncturist	Adult Boutique	Adult Education Center	Advertising Agency	Afghan Restaurant	African Restaurant	Airport	Airport Gate	...
-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	-0.071947	...
-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	-0.071947	...
-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	-0.071947	...
-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	-0.071947	...
10.679138	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	-0.071947	...

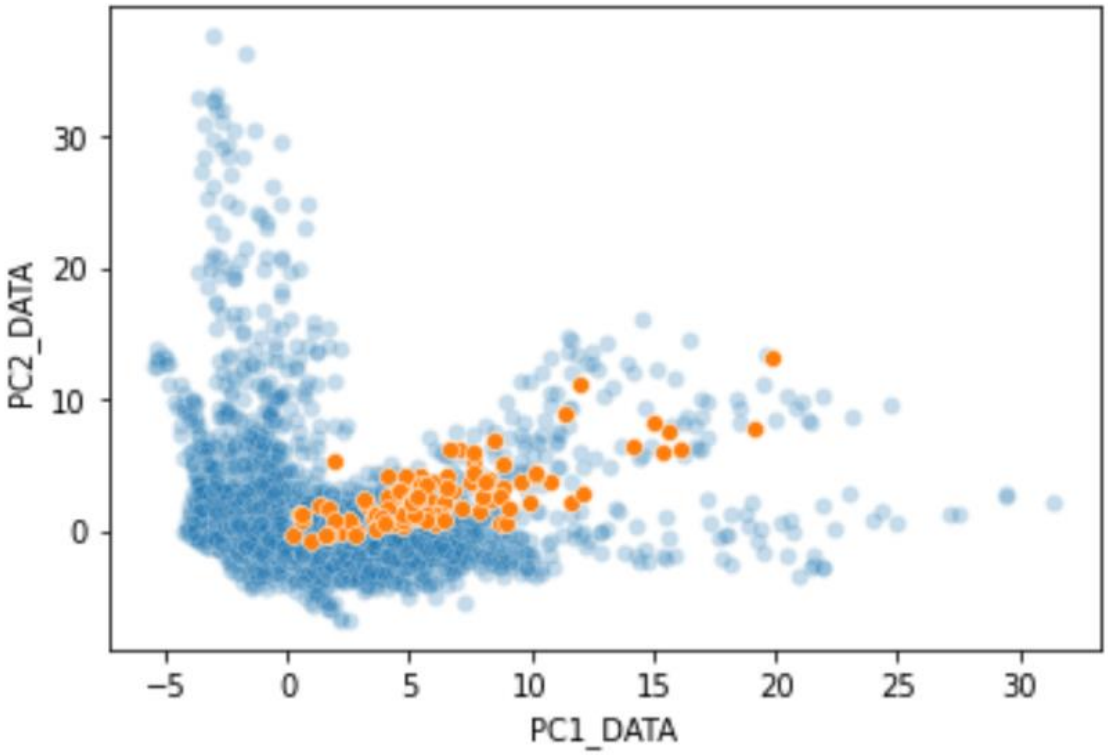
Besides reducing our prediction error, feature scaling is also helpful for data visualisation because it enables us to perform a so-called “Partial Component Analysis” or “PCA”.

PCA is something beautiful that is often used for data exploration as it helps us to reduce a multidimensional dataset into a shape that we can visualize.

Question mark over your head? No problem, let me help you with that: After one hot encoding, our dataset contains over 600 different features (e.g. zoos, parks, bars, restaurants, etc.). Each feature represents its dimension.

If we want to understand our dataset better, we would have to plot our data points for each dimension individually. This way we could have a look at how each feature behaves. However, with more than 600 different dimensions, this approach would take way too much time and would most likely be very error-prone.

So what we can do now is use PCA to reduce all of these dimensions down to two dimensions. This way, we can plot all data points on a 2D axis:



s you can see, we have all data points on one graph. As a result, it is much easier for us to interpret how our data behaves and what we are dealing with.

Now, ignore the orange dots for a moment and focus on the entire shape of our data. We can see that we have a lot of outliers on the bottom right and upper left. The majority of the data is concentrated in the left corner (10 to 10) of our graph.

It seems like we have only one large cluster in our data set that shares similar properties. That would not be surprising as we are analyzing venue categories only.

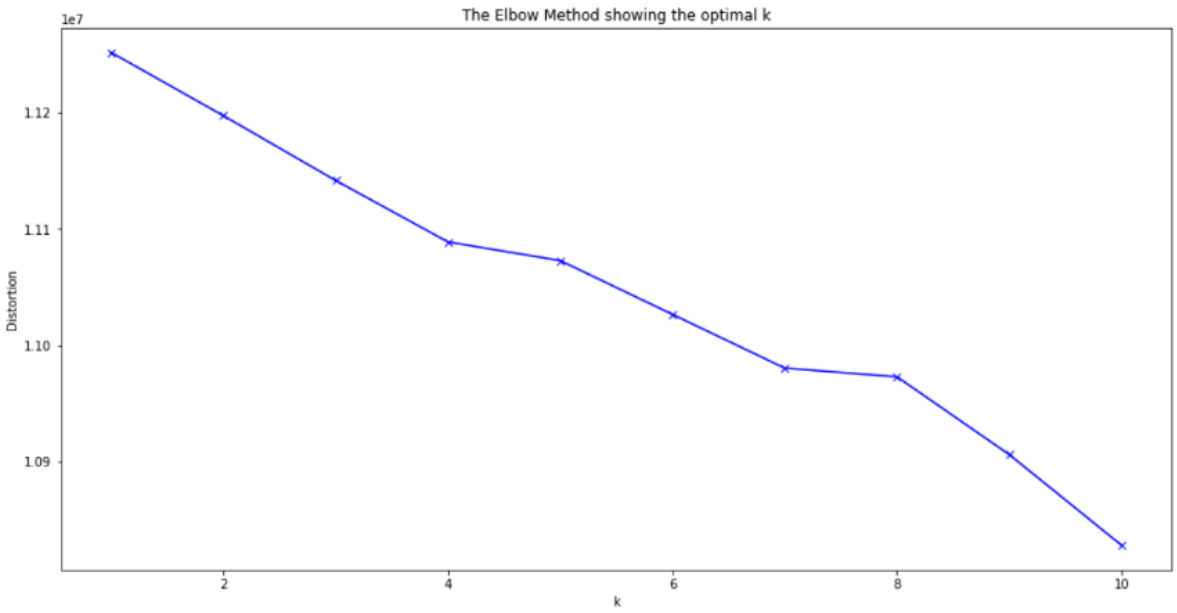
Let us now talk about the orange dots. These represent the data that originates from our existing L'Osteria restaurants. We can see that the majority of this data lies within the large data cloud in the bottom left and some of it belongs to the outliers on the bottom right side of the graph.

In summary, we can observe that there seems to be only one major cluster but at the same time we can see that the data points we are interested in (L'Osteria restaurants) have a clear indication to the right side of our graph.

So when we evaluate our algorithm, we should keep this in mind and see whether we managed to catch these data points in a distinguished cluster or not.

As our data is now ready to be fed into the algorithm, we need to determine how many clusters we want to divide our data into. To find the right amount of clusters for our dataset, we can use the so-called "Elbow" method.

The Elbow method is pretty simple but effective. With this approach, you set a range of cluster amounts. For each amount in the range, the algorithm will be trained again. Afterwards, we can analyze the distortions of each run and make our decision based on the distortion of each cluster:



As you can see, we have conducted the Elbow method in a range of 10. In other words, we have trained the algorithm 10 times, where each time the number of clusters was raised by one. As a result, we receive a linear plot that shows the distortion of each run.

Usually, the Elbow method returns a linear plot where the distortion falls sharply within the first couple of training runs. The amount of clusters for the final algorithm is then determined by choosing the number of clusters that showed the highest drop in distortion.

However, we can see that our Elbow method does not show a clear drop in distortion for any cluster amount. The distortion falls steadily until the amount of four then rises slightly just to keep decreasing with a steady pace again.

The method is not revealing the optimal amount of clusters. That is not surprising as our data is very condensed and does not show clear clusters straight away.

However, this does not necessarily mean that there are no clusters in our data at all. It simply means that the k-means algorithm has trouble identifying them.

There are several ways to improve the results (e.g. adding different data or further data transformation), however, we will refrain from doing so for now.

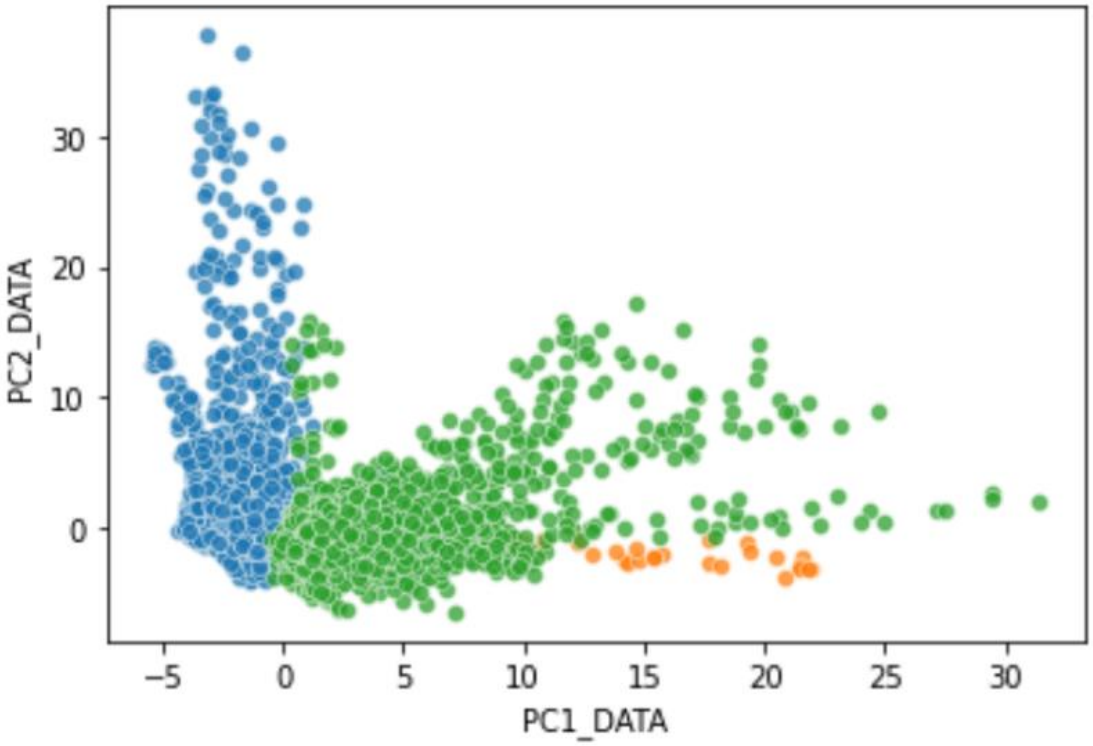
Our major objective is to build a model that identifies similarities between existing restaurants in Germany and locations in London to reduce the time needed for manual location scouting. Consequently, if our algorithm can rule out a significant amount of catchment areas in London, we can still consider our approach to be successful.

To keep things simple and save computing time, we will start with three clusters. As a result, we receive a new column in our dataset, containing the cluster IDs:

CLUSTER_LABELS	ATM	Accessories Store	Acupuncturist	Adult Boutique	Adult Education Center	Advertising Agency	Afghan Restaurant	African Restaurant	Airport	...
2	-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	...
2	-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	...
2	-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	...
2	-0.051770	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	...
2	10.679138	-0.10452	-0.033453	-0.03411	-0.063112	-0.026101	-0.062013	-0.121017	-0.177632	...

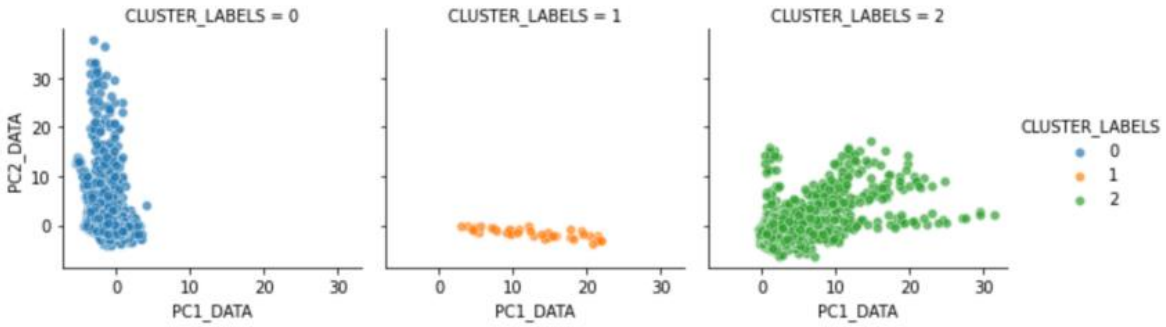
5. The Results

In this section, we will have a closer look at the results of our clustering. As we have over 600 dimensions to deal with, we have to reduce the dimensions to two by using PCA. This way we can plot our clusters in the same shape as before:



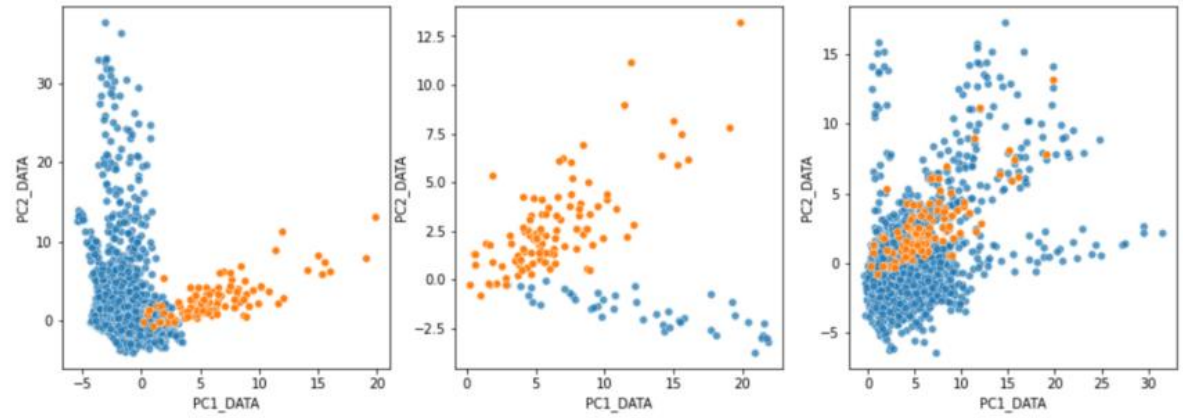
We can see that the algorithm has successfully divided our data into the desired amount of clusters. Blue is cluster 0, orange is cluster 1 and green is cluster 2.

We can see that the outliers as well as the large data cloud in the middle have been divided into their clusters. However, we can also see that there is some overlapping of the clusters. So let us look at each cluster individually:



We can see that we have two major clusters (0 and 2) as well as one small cluster (1). Cluster 0 and 2 are separated while cluster 1 seems to be rather random.

On a first look, we can assume that cluster 2 is a pretty good fit for our existing restaurants. To confirm this hypothesis, we can map the restaurant data onto the different clusters:



The above graph shows us how the data points of our existing restaurants are in orange. The blue points resemble the data points of each cluster. We can see that cluster 0 has only a few similarities as only a small amount of data points are

overlapping. Cluster 1 is a bad fit as it shows nearly no overlay at all.

Finally, cluster 2 is the best fit, covering all of our existing data points. However, we should note that cluster 2 is a little bit too large and shows additional outliers in a different direction than our existing restaurant data. This indicates that an additional clustering of this particular cluster could lead to even better results. However, for now, we will leave it be as time and resources are scarce.

The results of our experiment confirm that clusters 0 and 1 have only little to no overlay with data points from existing L'Osteria restaurants. Consequently, we will drop all catchment areas from these clusters and focus on cluster 2 in the next step of our analysis in which we will try to determine what makes this cluster so special.

To understand its properties better, we can visualize all features at once and analyze the most common features. To do so, we need to conduct two steps. First, we need to isolate our cluster 2 and get back the initial dataset before we applied PCA and transformation. This way we will be able to interpret the results better.

Second, we need to exclude all features that show a 0 in their column. As a result, we will exclude all venue categories that are not relevant for the chosen cluster and thus (hopefully) decrease the number of dimensions (it would be a little difficult to graph

over 600 features at once). After transformation, our dataset will look again like this:

(3974, 638)

	ATM	Accessories Store	Adult Boutique	Adult Education Center	Advertising Agency	Afghan Restaurant	African Restaurant	Airport	Airport Gate	Airport Lounge	...
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...
4	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...

As we can see, we unfortunately still have over 600 dimensions to deal with. To tackle this problem, we can visualize the features based on their amounts to get an initial overview. We can do so by generating a word cloud:

Before we jump to any conclusions, let us quickly talk about what word cloud does. It is a very simple algorithm that counts the unique entries within a string and visualizes them based on the number of their observations. Or in other words, the more often a word exists in a string, the larger its visualization.

Now, that leaves us with quite a large room for interpretation. Why? Because it all depends on how your string is structured and how you let word cloud interpret your string. This principle is also called regular expression operations. To make this clearer, here is a little example:

In our case, we create a string from our column names. The string includes each name exactly in the same amount as the type of venue that appears in cluster 2. If, for example, we have the venue type “College Soccer Field” three times in our dataset, our string variable will include “College Soccer Field College Soccer Field College Soccer Field”.

As you might guess already, now comes the tricky part. We know that the venue we look for consists out of three words but the algorithm doesn’t. WordCloud separates strings by default through a space which means that it will interpret each word individually.

For our analysis, this means that the generated word cloud will analyze our venues based on single words and not entire venues. While it would be possible to alter the regular expression operations in a way that WordCloud could interpret the venues

as a whole, we will refrain from doing so at this point. The reason for this decision is that it does us a favour as it e.g. filters out all sorts of duplications. Here is another example:

The data set includes a lot of different venues as e.g. “College Building”, “College Field” or “College Soccer Field”. The algorithm will now aggregate each word individually and will put weight on “College” as it is the most common word. And that is good because for us it is at this stage not important to know the details of the venue but to get an overall impression.

But enough of theory. Let’s have a look at what the cloud tells us. The first things that pop up are:

- Park
- Pub
- Groceries/Supermarket
- Caffès
- Bus/train
- Apartments/Office/Buildings
- Indian restaurants

We can also see that WordCloud was not able to interpret each word individually, indicating that the regex operator we chose is not perfect for our dataset. So it pays off to have a closer look at some of the smaller words as well.

We can find more food-related venues as well as more store, fitness and medical-related types. All in all, cluster 2 seems to include only catchment areas that are in urbanized areas with great infrastructure. It seems to be located close to a lot of buildings related to housing/working.

While there are other restaurants (mainly indian cuisine) in the area, it does not seem to be the largest factor (which makes sense as that would mean a lot of competition).

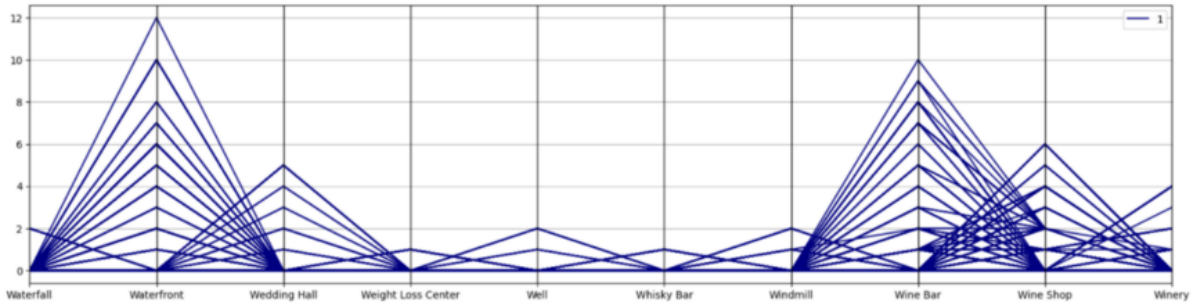
On the other hand, catchment areas seem to have a large number of coffee shops, pubs and supermarkets nearby. However, the most common feature seems to be parks which do not fit well with infrastructure and housing so we should keep that in mind when conducting further analysis.

In conclusion, it seems like an optimal catchment area lies within neighbourhoods where a lot of people live or work. While the catchment areas have nearby restaurants, these restaurants offer a different type of cuisine. Other foods and beverage venues are pubs or caffès which offer a different venue profile than L'Osteria restaurants.

Finally, the ideal catchment area has an outstanding infrastructure, making it easily accessible. All together that sounds like a pretty good fit for our restaurant chain. But we should have a more detailed look at our variables before we get too pleased.

To go into more detail, we will have to look at the features individually. Yes, individually. And to see whether we have outliers in our dataset that blur our outcome, we will use so-called parallel coordinate plots via pandas and matplotlib.

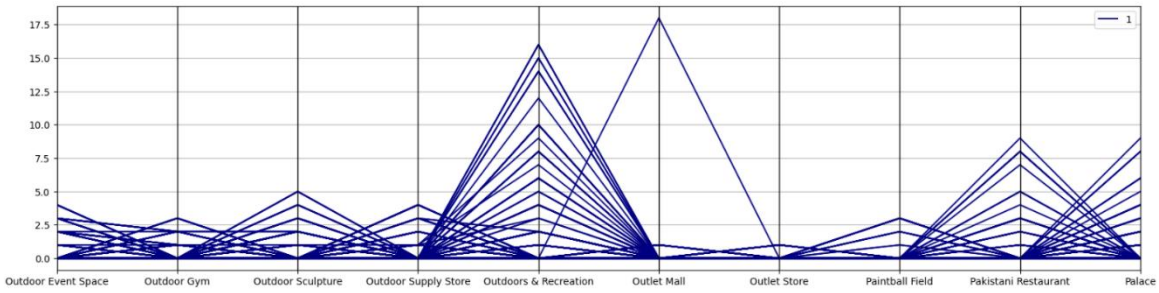
Since we have so many dimensions, we have to plot a lot of graphs. To spare you some time, we will only discuss the outcome of the analysis instead of going through each plot individually. To get you started, here is an example of how a graph looks like:

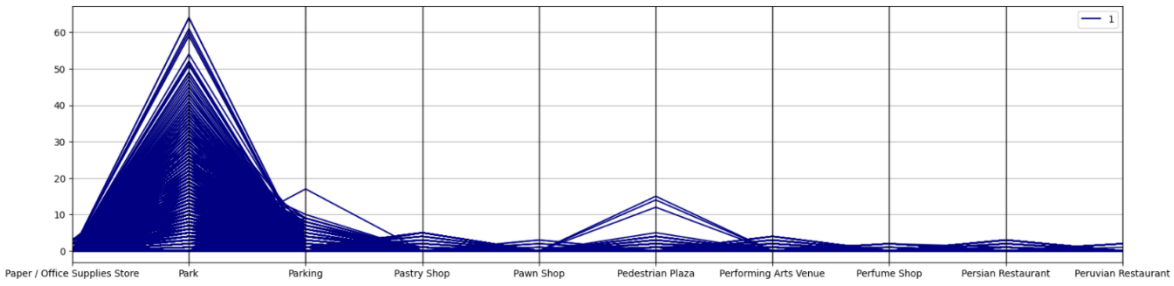


Parallel liner plots are drawing a line for each row/catchment area that has a respective entry. Consequently, features with a lot of entries will have a very high density of lines while others will only show a few lines. If you look at the above graph, you can see that “Waterfront” has somewhat 10 entries while “Weight Loss Center” and “Whisky Bar” have significantly fewer.

Another way of interpretation is evaluating the lines in terms of the y-axis. If the lines spike, it means there are a lot of venues for this feature in our cluster. However, if only one line is spiking, it shows that there is only one catchment area in our cluster that has a very high amount of a specific venue.

If, on the other hand, there are a lot of spiking lines, it is a clear indicator that this feature is very common in all catchment areas. Examples of this are the features “Outlet Mall” and “Park”. While “Outlet Mall” has only one large entry, the feature “Park” indeed seems to happen in larger amounts in all catchment areas.





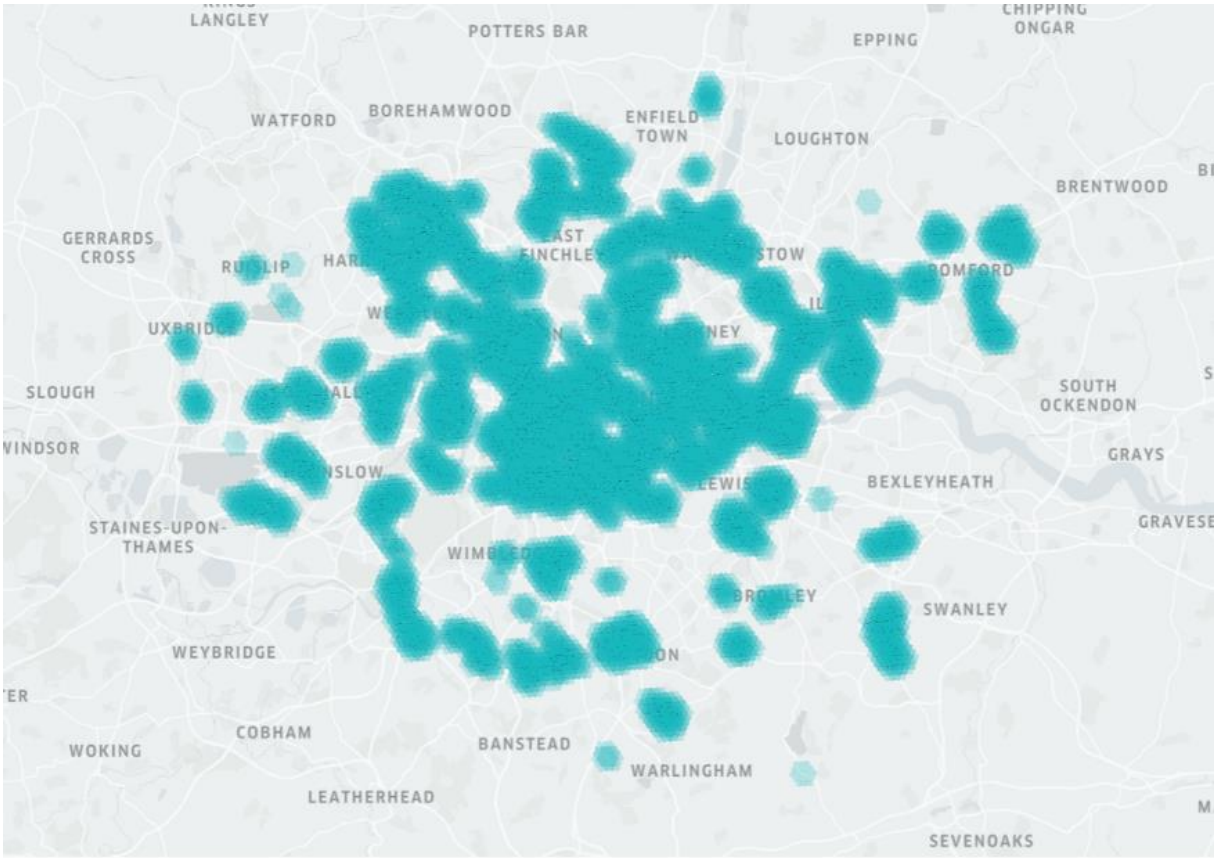
So when analyzing the graphs, we are interested in features (or groups of features) that appear often and in many catchment areas. Here is a summary of my findings:

Our word cloud turned out as a very good indicator. Cluster 2 seems to include catchment areas that are located in some kind of suburban area. They have a large share of parks or related venues such as fields, forests, gardens, etc. as well as large amounts of residential living and colleges/schools.

There is also a high concentration of stores, ranging from pharmacies and supermarkets to department stores and clothing shops. The area also seems to have a great infrastructure as there is a high amount of metro and bus stations.

Finally, all catchment areas have a large share of pubs (or other alcohol-related venues) while the amount of restaurants seems to be rather low. The only exception for the latter is Indian restaurants which seem to happen quite often.

So now we are left with the big reveal. Let us visualize our target cluster on a Kepler map and see what our actual outcome is:



We can see that we have successfully removed a large amount of hexagons/catchment areas within London. We can also see that

our algorithm has identified a lot of catchment areas in London's suburban districts. This confirms our first guess based on the word cloud and parallel line plots.

However, we can also see that there are large amounts of catchment areas within the city centre of London.

6. The Discussion

Let us have a quick recap! Our objective was to build a low-budget analysis tool, based on geographical information that enables us to scout for new, fitting restaurant locations all over the world.

Then, we had to build a clustering algorithm that is using Foursquare venue data to compare characteristics of existing restaurant locations with characteristics of given geographical boundaries.

After that, we visualized our data and applied further analysis to identify a target cluster that enables us to identify unique geographical areas that have similar characteristics as our existing restaurants.

Ok, everybody on the same page? Great! Then let us talk about the obvious drawbacks.

6.1 Not Enough Diverse Data

Our dataset (and thus our entire analysis) is based solely on venue data. While this data is important, it makes our analysis very homogenous.

To make our algorithm and our analysis more meaningful, we should use other data as well. In addition to our venue data, it would be highly interesting to include gender-based demographic data in our analysis.

We could also add income data and other behavioural information (e.g. such as movements between areas) to our analysis. As a result, our clustering would be much more precise and offer more insights that would allow us to scout for locations that meet the requirements of our desired target groups.

6.2 No Feature Selection

Every machine learning-based information system should include some kind of feature selection. Feature selection has the objective to reduce the number of input variables to reduce computation costs as well as improve a model's performance by selecting input variables with a proportionally large performance contribution.

Feature selection can be challenging as selecting the appropriate measure depends on the type of model (e.g. neural network), data types as well as input and output variables. In our case, we could introduce many different feature selection processes but here are a few suggestions:

- **WCSS minimization**
  When using euclidean distance, K-means aims to minimize the within-cluster sum of squares (WCSS). By analyzing the maximum absolute centroids dimensional movements for each dimension, we can determine which features have the most effects on clustering.
- **"Greedy algorithm"**
  The idea of this approach is rather simple. We choose a desired range of clusters and train our algorithm for

each feature individually within that range. For each iteration, we will save the results in form of a performance metric like silhouette score or Dunn index. We then choose the feature with the highest score and add it to our list of target variables. We then remove it from our training set and repeat the iteration. We again add the feature with the highest score to our final list. We repeat this process until certain pre-defined criteria are met. Such criteria could be a maximum amount of target features or a specific performance metric threshold. It is possible to extend this algorithm by one more step where the process is applied on the target list as well.

6.3 No Hyper-Parameter Tuning

Ok, granted. There is not so much we can do about this. However, we could play around a little more with the initial centroid placements. The standard setting of the algorithm is that the centroids are initialized randomly. If we set the initial coordinates ourselves, we might get better results.

In addition, we could try out more clusters. Yes, more. Even if the Elbow method indicates that there is no optimal k, increasing the amounts of clusters might give us a segmentation that fits even better with our existing restaurants' profiles.

6.4 No Model Comparison

Finally, we should note that we have tried one model approach only. To build a more sophisticated information

system, we should confirm our model's performance by comparing it to a different algorithm that was trained on the same data. Other clustering algorithms for such purpose could be:

- Mean shift clustering
- Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
- Expectation-Maximization (EM) Clustering using Gaussian Mixture Models (GMM)
- Agglomerative Hierarchical Clustering

A detailed description of all approaches can be found in this story.

7. The conclusion

It is time to wrap up and talk about what we have achieved. We wanted to build a low budget GIS that helps us to analyse existing restaurants and take these insights to propose new locations in any other city of our choice. Our objective then faced us with two challenges:

1. We needed to find a way to analyse our existing restaurant data and make it transferable to any other place on earth.

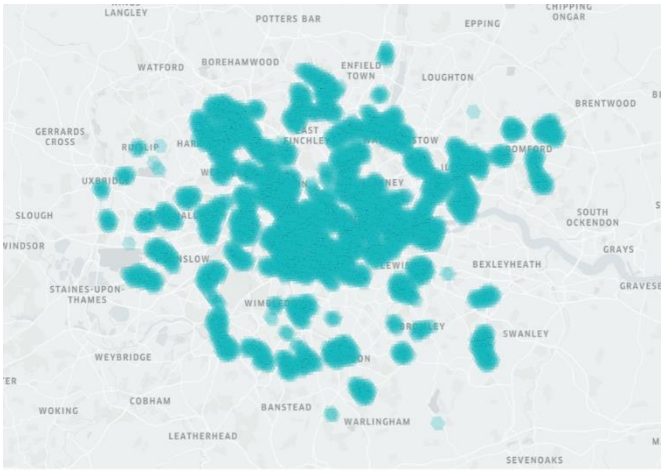
2. We had to find a way to analyse existing restaurants and new locations based on the same data so that we can learn from it.

To overcome these hurdles, we have applied a combination of Uber's H3 geographical index and OpenStreetMap data. As a result, we are now able to analyse the areas around existing restaurants in a standardized way which allows us to apply this analysis to any other city worldwide.

By choosing Foursquare as our common dataset, we can analyse existing areas and new locations based on the same data. This enabled us to draw conclusions from our existing restaurants and apply these to our target cities.

As a result, we managed to build a GIS that clusters new locations based on our existing restaurants and suggests only catchment areas that are similar to our existing network.

We tested our approach on a real-life example of an Italian restaurant chain, L'Osteria, and London as our target city. Our system suggested a total of 16,997 catchment areas within London.



After applying our model, we successfully reduced this amount to 3,974 catchment areas — a reduction of more than 75%. We can thus deem our approach as successful in the sense that it successfully decreases the area we would have to focus our scouting activities on.

However, it must also be pointed out that the suggested approach is yet not sophisticated enough to go into production. Both data and model have yet too much room for improvement as discussed in the previous section. It is therefore highly recommended to look into these issues before one can apply this approach in a real-life business case.

With that, we come to the end of this post. Thank you for reading till the end. I hope this post was informative and helped you in your endeavour.