London Real Estate - Rent Analysis

Find the flat that better fit your needs

Location based Clustering

30th November 2019

Federico Sciuca

[Contact me on LinkedIn (https://www.linkedin.com/in/federico-sciuca/)](https://www.linkedin.com/in/federico-sciuca/)

Table of Contents

Executive Summary

This is not the end of the study path but just the start!
As an analyst, I'm interested in finding new insight to understand a certain phenomenon but as a marketer, I also have a client-centric vision of the businesses and one of the powerful ways to make business is implementing solutions that permit the user to make his/her decision fast and easy.

Everything that saves time and efforts to the user has a great value on the market.

Empowering people to make an informed decision is a great way to improve the world we're living in.

API

https://opencagedata.com/ (https://opencagedata.com/)

https://it.foursquare.com/ (https://it.foursquare.com/)

Introduction and Business problem presentation

We could identify 3 main reasons why a flat doesn't fit the customer needs:

- The flat looks old and stale
- The neighbour hasn't the expected commodities nearby
- The price is too high for that particular flat or out of budget

Our goal with this Notebook is to have a systematic way to analyze the offers posted by RightMove.co.uk to produce a map of the best opportunities in the city.
If you are looking for a new flat and you like your actual neighbour, we can provide you with a list of all the opportunity on the market.

For this project, I'm going to create a simple software that scrape the website RightMove to collect an updated list of flat for rent, analyze each offer using Foursquare and cluster them to divide the housing market in 20 groups with similar neighbours.

Methodology

For this particular analysis, we are going to collect updated data from **RightMove.co.uk**.

To do so, I decided to spend time developing a **web scraping** application using ***Beautiful Soup 4***, but then I discovered a repository on **GitHub** offered by ***toby-p*** and available here (https://github.com/toby-p/rightmove_webscraper.py), that's provide a easy way to scrape RightMove!

This script collect the following data:

- price
- type
- address
- url
- agent_url
- postcode
- number_bedrooms
- search_date

A record will look like the following:

id	price	type	address	url	agent_url	postcode	nı
0	2210.0	2 bedroom terraced house	Lifford Street, SW15	http://www.rightmove.co.uk/property-to-rent/pr (http://www.rightmove.co.uk/property-to-rent/pr)...	http://www.rightmove.co.uk/estate-agents/agent (http://www.rightmove.co.uk/estate-agents/agent)...	SW15	

The address is in the format *"Street, City, Postcode"* and is an **unstructured field** but for our purpose, we can leave as it is. Instead, the **PostCode** present a *"limited"* format because we have the first two/three digits only. **This is not accurate enough to collect meaningful data about the venues around the flat**.

In order **to solve this problem**, we are going to use OpenCage Geocoder API (https://opencagedata.com/) to *look up coordinates from a postal address*. This is a case when an unstructured field becomes helpful.

To **associate each rent offer to a District**, we are going to **join the data table with a second dataset** presenting two columns:

- District Name
- PostCode

This dataset had been created **scraping a Wikipedia Table** (available here (https://en.wikipedia.org/wiki/London_postal_district)) with the data we need for the exercise.

When the data are collected and merged into a single data frame, we are going to **cluster** them using the **K-Means algorithm** to divide the market into *20 different clusters*. To have an idea of the distribution on the territory of the offers, I plotted **2 meaningful maps**:

- Cluster map: this map shows the distribution of the clusters using colours to identify each cluster.
- Heating map: this map shows the areas with a higher number of offers.

To better understand the market, I decided to develop some **bar plots** to easily identify the **average price for a studio flat, 1 bedroom flat and 2 bedroom flat grouped by the District Name**.
I've also plotted a **correlation matrix** to identify if there are strong unexpected correlations between features in the dataset.

Finally, **I decided to conclude the project by asking the user to insert some data:**

- Your address: this input is used to analyze the neighbourhood you are living in and to use this information to find the cluster you belong to.
- The number of bedrooms you are looking for: this input is used to filter the results of the cluster you belong to.

This part of the analysis has as output a data frame with a list of filtered results based on your preference.

Results

As expected, the **price of a flat can't be forecast based on the venues around it only** and there is, of course, a strong correlation between the number of bedrooms and price. *Nevertheless, is possible to develop a prediction model to take in consideration not only the characteristics of the flat but also the District the flat belongs to and the presence of some key venues near the flat.*
An example of **key factors** could be the presence of **supermarkets with high reputation**, **a public transport stop**, **schools or Universities**, **Hospital**. The **correlation** between price and these categories **is low but still important to the final users**.

The **goal of this notebook** is to provide to everyone a way to scrape the housing market and **identify the best offers that fit the user's personal needs**. Providing an ideal address and the number of bedrooms the user is looking for, he/she can easily understand which area of London is the best for his/her next step.

Data Exploration

We can take a look at the raw data and visualize some distributions to better understand how the market is divided.
We are going to **plot a bar chart** that shows the **number of offers listed by the number of bedrooms**.

In [113]:

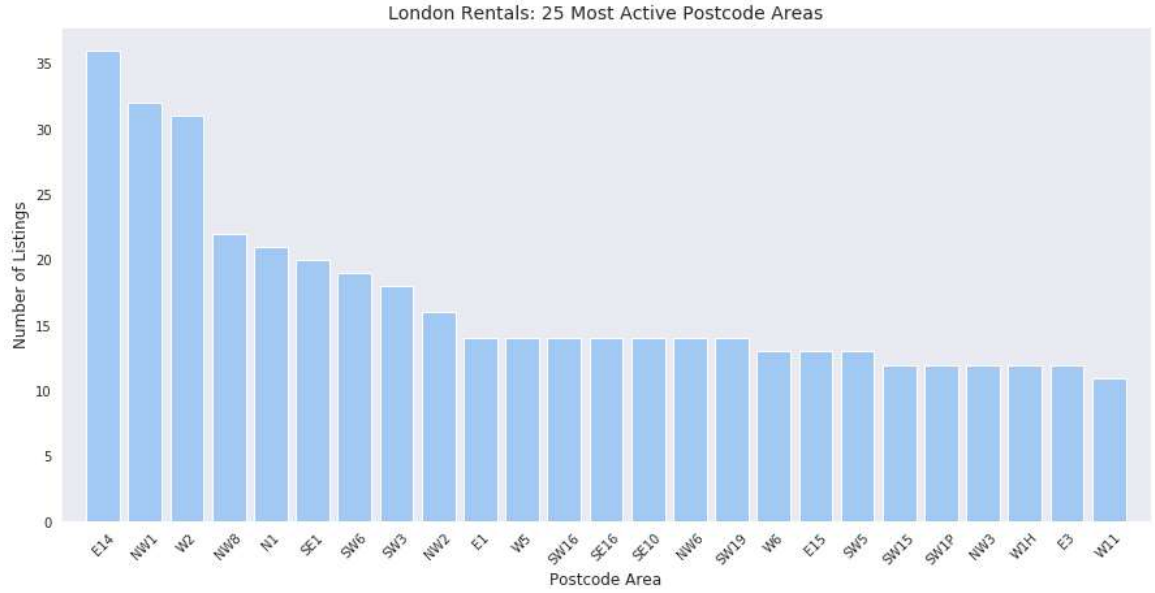
```
import seaborn as sns
def plot_by_type(rmd: RightmoveData):
    """Bar chart of count of results by number of bedrooms."""
    df = rmd.summary()
    labels = [f"{i}-bed" if i != 0 else "Studio" for i in df["number_bedrooms"]]
    x = df.index
    y = df["count"]
    sns.set_style("dark")
    sns.set_palette(sns.color_palette("pastel"))
    fig, ax = plt.subplots(figsize=(12, 8))
    plt.title("London Rentals: Number of Listings by Apartment Type", size = 14)
    plt.xlabel("Apartment Type", size = 12)
    plt.ylabel("Number of Listings", size = 12)
    plt.xticks(size = 12)
    plt.yticks(size = 12)
    plt.bar(x, y, tick_label=labels)
    plt.show()

plot_by_type(rmd)
```

As we can see, the **most frequent** offers on the market are about **2 bedrooms apartments** and **1 bedroom apartment**.

Where are the most offers located by postcode?
We can plot a new **bar chart to explore the distribution of offers by postcode**.

In [111]:

```
def plot_by_postcode(rmd: RightmoveData, number_to_plot: int = 25):
    """Plot count of results by postcode."""
    df = rmd.summary("postcode")
    df.sort_values(by="count", ascending=False, inplace=True)
    df = df.reset_index(drop=True)[:number_to_plot]
    x, y = df["postcode"], df["count"]
    ymax = ((df["count"].max() // 5) + 1) * 5
    sns.set_palette(sns.color_palette("pastel"))
    fig, ax = plt.subplots(figsize=(15, 7))
    ax.bar(x.index, height=y)
    ax.set_title(f"London Rentals: {number_to_plot} Most Active Postcode Areas", size=1
4)
    ax.set_xlabel("Postcode Area", size=12)
    ax.set_ylabel("Number of Listings", size=12)
    ax.set_xticks(x.index)
    ax.set_xlim(-1, x.index[-1]+1)
    ax.set_xticklabels(x.values, rotation=45)
    ax.set_yticks(range(0, ymax, 5))
    return fig

f = plot_by_postcode(rmd, number_to_plot=25)
```



Data Manipulation

In order to make the results easier to read and interpret, I decided to **join each postcode to the District Name**.
This step is of **primary importance to make the results enjoyable to the final user**.

First of all **I scraped the Wikipedia Table** using **Beautiful Soup 4** and I saved the result as a CSV. To make this Jupyter Notebook shorter, *I'll not include the code used to scrape the table* but I'm going to **open the CSV file as a DataFrame** and work directly on it.

To define the location of each District, we are going to identify latitude and longitude of them using OpenCageData.

In order to expand the information dataset relative to each flat on the market, we are going to use OpenCageData. This API permits to search and grab geographical information based on latitude and longitude or address. We are going to use the postcode of each District to expand our dataframe.

In [17]:

```
df_districts['Latitude'] = latitudes
df_districts['Longitude'] = longitudes
df_districts.head()
```

Out[17]:

	PostCode	District Name	Latitude	Longitude
0	E1	Eastern Head district	51.489334	-0.144055
1	E2	Bethnal Green	51.489334	-0.144055
2	E3	Bow	51.514947	-0.093046
3	E4	Chingford	51.507322	-0.127647
4	E5	Clapton	51.514947	-0.093046

Having the latitude and longitude of each district could be useful for future and more in-depth analysis.

However, for this project, it is essential to expand our database by **associating each apartment with its geographical coordinates**.

Using OpenCage to expand the record for each flat

We can do the same process to expand the information related to each flat. OpenCage Data can be used to collect geographical information searching by address.

In order to have enough data to work with, is better to collect the following geographic information:

- latitude
- longitude
- county
- complete postcode
- state district
- suburb

However, during the analysis, I discovered that not all the data are available for every address in the data frame. This fact limited a bit the analysis we could do but it doesn't prevent us to achieve our goal to develop a new way to cluster the housing market by similarity.

The dataset we have looks like the following:

In [22]:

```
print(df_info.shape)
df_info.head(3)
```

```
(1038, 8)
```

Out[22]:

	i	address	Latitude_a	Longitude_a	county	Postcode_complete	state_district	su
0	0	Drayton Gardens, Chelsea, London, SW10	51.488547	-0.181068	Royal Borough of Kensington and Chelsea	SW10	Greater London	Bror
1	1	Earls Court , London, SW5	51.491612	-0.193903	Royal Borough of Kensington and Chelsea	SW5 9LY	Greater London	
2	2	Blackheath Road, London, SE10	51.474230	-0.021016	Royal Borough of Greenwich	SE10 8GA	Greater London	

In order to properly match the new DataFrame to the original set of data, I defined the column "i" that is equal to the index values of each record from the original DataSet.

Now it's time to **merge the data frames**. I'll merge the *df_districts and df_rm on Postcode*. In this way I'll have a match between postcode and district name.

Then I'll merge the *df_merged with the new df_info*.
In this way I'll have available for each flat on the market as many location information as possible.

I'll drop the "state district" column because it doesn't add any further information.

After the merging operation, the new data frame looks like the following:

In [25]:

```
print(df_merged.shape)
df_merged.head(3)
```

```
(1050, 11)
```

Out[25]:

	price	type	address	url	ager
0	35750.0	6 bedroom house	Drayton Gardens, Chelsea, London, SW10	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.uk/e agents/ag
1	2881.0	2 bedroom apartment	Earls Court , London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.uk/e agents/ag
2	1250.0	1 bedroom flat	Blackheath Road, London, SE10	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.uk/e agents/ag

To be sure that each address has been merged with the correct series of new information, I used the 'i' column as double check.
This solution permitted me to double-check that the previous index matched the new index, this means that I dropped the records that have not now information.

In [29]:

```
print(df_location.shape)
df_location.tail(3)
```

```
(1050, 19)
```

Out[29]:

	i	price	type	address_x	url	
1047	1047	2900.0	2 bedroom apartment	Gordon Road, Nunhead, London, SE15	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmo
1048	1048	2600.0	2 bedroom flat	Boardwalk Place, E14	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmo
1049	1049	3640.0	4 bedroom apartment	Melcombe Street, Marylebone, London, NW1	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmo

As you can see, now we have the original DataFrame expanded.

What we have added:

- **District Name**
- **Latitude** *and* **Longitude**: lat and lng of the "District Name"
- **Address_y**: a copy of the original address that we'll drop. I used it to be sure that the DataFrames have been merged correctly
- **Latitude_a** *and* **Longitude_a**: lat and lng of the flat.
- **County**
- **Postcode_complete**: an extension of the original PostCode
- **State_district**
- **suburb**

Foursquare API - Find the most common venues near the flat

Collect meaningful information about the most common venues around each flat.

This step is crucial to develop a good **clustering K-Means model**.

We are going to use the **Foursquare API** to collect the **first 100 venues** in a **radius of 500 meters around each flat posted on RightMove**.

In [31]:

```
df_clustering.T.apply(lambda x: x.nunique(), axis=1)
```

Out[31]:

```
i                    1050
price                 351
type                   80
address_x             957
url                  1042
agent_url             556
PostCode              136
number_bedrooms         9
search_date             1
District Name         105
Latitude               26
Longitude              26
address_y             947
Latitude_a            750
Longitude_a           748
county                 67
Postcode_complete     638
state_district          7
suburb                222
dtype: int64
```

To be sure that the code run smoothly during the analysis, is better to **drop** all the **rows that have a Null value** in one of the columns **"Latitude_a" or "Longitude_a" or both** using the following code.

In [32]:

```
# Drop rows with Longitude_a or Latitude_a = np.nan
df_clustering.dropna(subset=['Latitude_a', 'Longitude_a'], inplace=True)
print(df_clustering.shape)
df_clustering.rename(columns={"level_0" : "i-match"}, inplace=True)
df_clustering.reset_index()
df_clustering.head(3)

#Note: I'm saving the columns level_0, i, i_match, address_x and address_y to have a wa
y to double-check all the data manipulation. After I'm sure that everything is correct,
I'll drop them.
```

```
(1038, 19)
```

Out[32]:

	i	price	type	address_x	url	a
0	0	35750.0	6 bedroom house	Drayton Gardens, Chelsea, London, SW10	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.u agent
1	1	2881.0	2 bedroom apartment	Earls Court , London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.u agent
2	2	1250.0	1 bedroom flat	Blackheath Road, London, SE10	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.u agent

Now it's time to define the setting variables for **the Foursquare API** in order to **collect the information** we need to develop a **clustering analysis**.

Address and Venues

The process of collecting the venues around each flat is fundamental to develop a clustering analysis.
the following are the number of venues I collected for the first three record of the dataset:

In [38]:

```
df_address_venues.groupby(['address_x', 'i', 'price', 'type']).count().head(3)
```

Out[38]:

address_x	i	price	type	address Latitude	address Longitude	Venue	Venue Latitude	Venue Longitude	Venu Categor
Latitude House, Oval Road, Camden, NW1	186	3445.0	2 bedroom apartment	93	93	93	93	93	9
Reminder Lane, North Greenwich, SE10	928	3000.0	2 bedroom flat	13	13	13	13	13	1
Walton Street, Chelsea, SW3	44	6000.0	4 bedroom terraced house	100	100	100	100	100	10

In [39]:

```
print("There are " , len(df_address_venues["Venue Category"].unique()), " unique categories in the dataset")
```

```
There are  425  unique categories in the dataset
```

Working with categorical variables don't permit to explore mean or develop other distribution analysis because it doesn't make sense. This means that in order to find the most frequent venue category, is important to transform the venues in dummies variables.
After that, is better to reorganize the dataset. The new dataset have a record for each venue category related to each flat (this means that each row has one "1" only out of 425 venues category). For this reason the shape of the table is the following:

In [41]:

```
df_venues_dummies.shape
```

Out[41]:

```
(42728, 429)
```

In the direction of creating a more useful dataset, we need to group the data by 'address', 'i', 'price' and 'type'. In this way, we can see that the resulting shape of the table is:

In [43]:

```
df_dcategories.shape
```

Out[43]:

```
(1017, 429)
```

To make the analysis faster and tinier, we are going to find the top 20 most common venues that will be used to develop the clustering analysis.

Finally we have a dataset ready to be used four our purpose!

In [45]:

```
num_top_venues = 20

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['address', 'i', 'price', 'type']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
address_venues_sorted = pd.DataFrame(columns=columns)
address_venues_sorted['address'] = df_dcategories['address']
address_venues_sorted['i'] = df_dcategories['i']
address_venues_sorted['price'] = df_dcategories['price']
address_venues_sorted['type'] = df_dcategories['type']

for ind in np.arange(df_dcategories.shape[0]):
    address_venues_sorted.iloc[ind, 4:] = return_most_common_venues(df_dcategories.iloc
[ind, 3:], num_top_venues)

address_venues_sorted.head(3)
```

Out[45]:

	address	i	price	type	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue
0	Latitude House, Oval Road, Camden, NW1	186	3445.0	2 bedroom apartment	Coffee Shop	Bar	Burger Joint	Music Venue	Pub
1	Reminder Lane, North Greenwich, SE10	928	3000.0	2 bedroom flat	Park	Spa	Waterfront	Harbor / Marina	Pub
2	Walton Street, Chelsea, SW3	44	6000.0	4 bedroom terraced house	Café	Italian Restaurant	Hotel	Coffee Shop	Middle Eastern Restaurant

Clustering using K-means

Now that we have all the information we need to divide the market by geographic similarity, it's time to split it in Clusters.

We are going to use the **K-Means algorithm** for this purpose.
K-Means clustering is a method of **vector quantization**, originally from signal processing, that is **popular for cluster analysis in data mining**. K-Means clustering aims **to partition n observations into k clusters** in which **each observation belongs to the cluster with the nearest mean**, serving as a prototype of the cluster.

Initially I need to tidy the dataset from the columns that don't contain independents variables.
For this reason I drop the columns "address", "i", "price" and "type". Some of these columns could be used for future analysis and in favor of developing a pricing model but in this particular situation, we are just going to perform clustering analysis and location selection.

The table now is ready to be used to train the K-Means algorithm and looks like the following:

In [49]:

```
df_dcategories_nn = df_dcategories.drop(columns={'address', 'i', 'price', 'type'})
df_dcategories_nn.head(3)
```

Out[49]:

	ATM	Accessories Store	Acupuncturist	Adult Boutique	Afghan Restaurant	African Restaurant	American Restaurant	Animal Shelter
0	0.0	0.0	0.0	0.0	0.0	0.0	0.010753	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0

There is not a right answer to the question: how many clusters I need?
For this particular paper I decided to split the dataset in 20 clusters to be sure that small differences between two locations influence the cluster the flat belongs to.

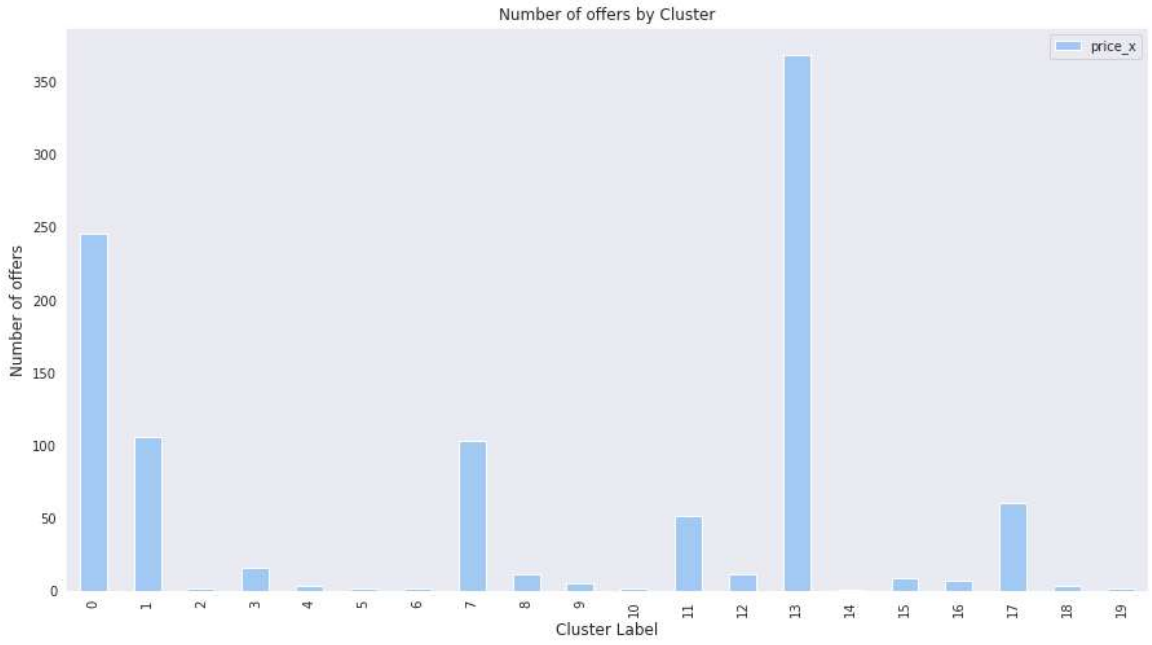
Clustering distribution

At the first impression, the number of clusters looks too high but after testing other values of k, I decided that the best division of the market is achieved using an high number of clusters because in other case the distribution of the flats are even more concentrated in a single cluster. For example, if we decide to set k = 8, one of the clusters will contain almost 50% of the market, making the entire project useless.

We can plot a bar chart to see how the flats are distributed by clusters.

In [114]:

```
df_plot1 = df_final_clean.groupby(['Cluster Label']).count()

df_plot1.reset_index().plot(x="Cluster Label", y="price_x", kind="bar", figsize=(15,8))
plt.xlabel("Cluster Label", size = 12)
plt.ylabel("Number of offers", size = 12)
plt.title("Number of offers by Cluster")

plt.show()
```



London Clusters Distribution Map

To show that the clusters are not selected by density, we can plot the map relative the dataset.
Each colour is related to a particular cluster. Is easy to understand that the green/yellow cluster is the one with an high presence of coffee shops, restaurants and shops because are located in the areas with an high intensity of tourists and shopping areas.

In [61]:

```
# London latitude and longitude
latitude = 51.509865
longitude = -0.118092

# create map
map_clusters = folium.Map(location=[latitude, longitude], zoom_start=12)

# set color scheme for the clusters
x = np.arange(k_clusters)
ys = [i + x + (i*x)**2 for i in range(k_clusters)]
colors_array = cm.rainbow(np.linspace(0, 1, len(ys)))
rainbow = [colors.rgb2hex(i) for i in colors_array]

# add markers to the map
markers_colors = []
for lat, lon, poi, cluster in zip(df_final_clean['Latitude_a'], df_final_clean['Longitu
de_a'], df_final_clean['address_x'], df_final_clean['Cluster Label']):
    label = folium.Popup(str(poi) + ' Cluster ' + str(cluster), parse_html=True)
    folium.CircleMarker(
        [lat, lon],
        radius=5,
        popup=label,
        color=rainbow[cluster-1],
        fill=True,
        fill_color=rainbow[cluster-1],
        fill_opacity=0.7).add_to(map_clusters)

map_clusters
```

Out[61]:

Housing market - heating map

Is possible and helpful to plot a heating map to identify the areas that present an high volume of offers. This particular view change a lot during the day because in London, on average, each flat has been rented in 7 days more or less.
The constant changing of this map suggest that the housing market is very active.
A long term analysis would be able to identify if there are or not any phenomenon of seasonality or peak of activity related to the expiring of the tenancy agreements.

Another interesting analysis would be interesting to conduct is querying the relation between the house market and the Brexit announcements. Did the Brexit accelerate the cycles into the housing market promoting a major number of short term tenancy agreements?

In [64]:

```
base_map
```

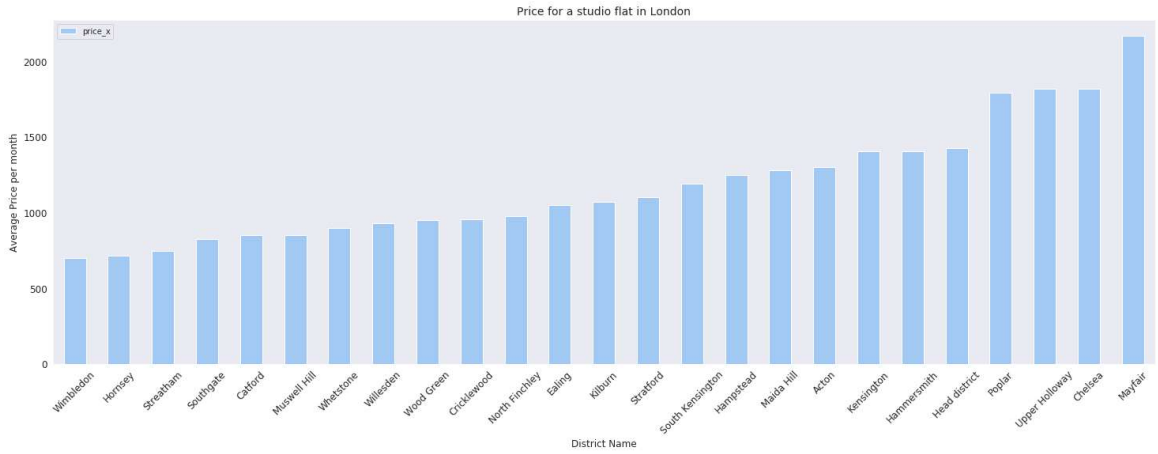
Out[64]:

Average price for a studio flat by district

After a briefly analysis of the total market, I think could be helpful for the reader to have a landscape view of the average price of the 5 most common type of flat in relation with the district. The standard deviation of the price of a studio flat in London is **390.872148** and the average price is **1179.176667**.
We can define the coefficient of variation to be able to compare the type of flats and the price.
The coefficient of variation (CV), also known as relative standard deviation (RSD), is a standardized measure of dispersion of a probability distribution or frequency distribution. It is often expressed as a percentage, and is defined as the ratio of the standard deviation to the mean (or its absolute value).
In this case the coefficient of variation is equal to **0.33147886906076307**.

In [118]:

```
import matplotlib.pyplot as plt
df_studio_flat.plot('District Name', 'price_x', kind='bar', figsize=(25,8))

plt.xlabel('District Name', size = 12)
plt.ylabel("Average Price per month", size = 12)
plt.title("Price for a studio flat in London", size = 14)
plt.xticks(size = 12, rotation = 45)
plt.yticks(size = 12)

plt.show()
```

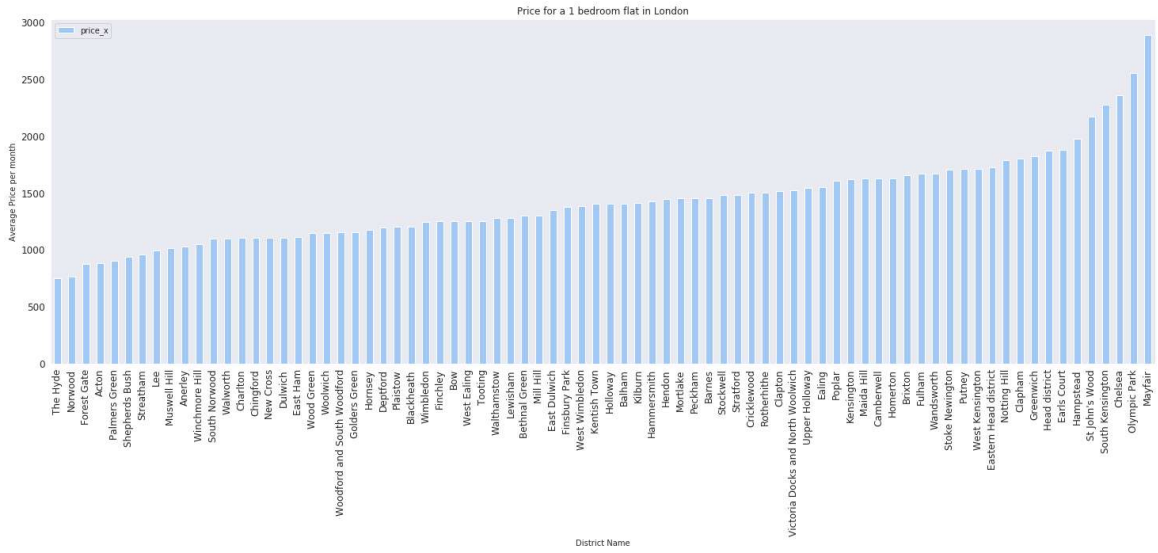


Average price for a 1 bedroom flat by district

Being the 1 bedroom flat more common than the studio flat, is understandable that it is more diffuse geographically speaking between districts.
The standard deviation of the price of a 1 bedroom flat in London is **391.457509** and the price is **1421.509474** on average. The **coefficient of variation** is equal to **0.27538156879002273**.

In [120]:

```
import matplotlib.pyplot as plt
df_1bed.plot('District Name', 'price_x', kind='bar', figsize=(25,8))

plt.xlabel('District Name')
plt.ylabel("Average Price per month")
plt.title("Price for a 1 bedroom flat in London")
plt.xticks(size = 12)
plt.yticks(size = 12)

plt.show()
```

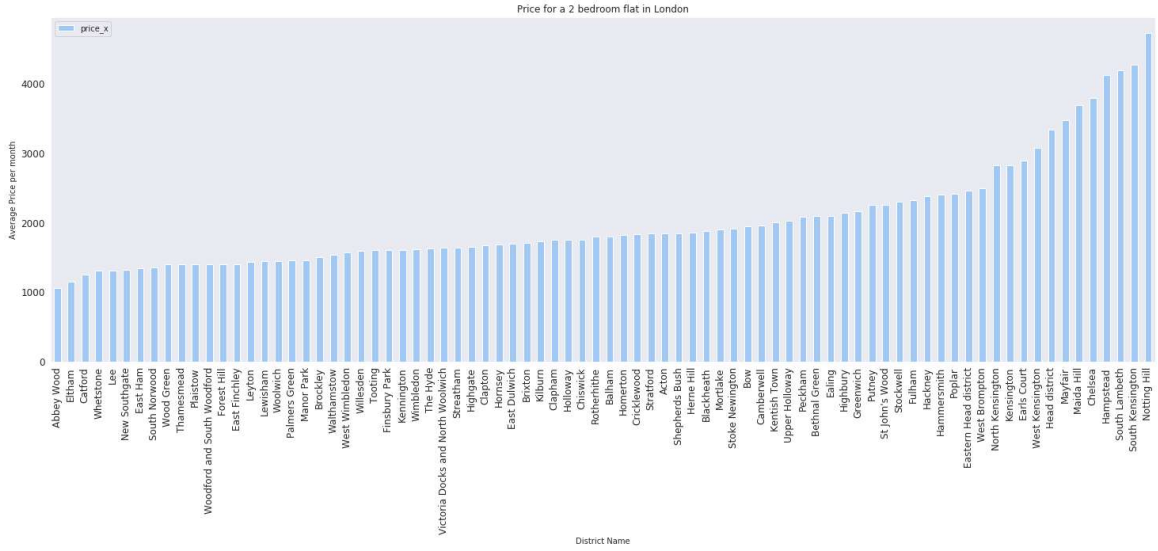


Average price for a 2 bedroom flat by district

We can say the same for the 2 bedroom flat. Is interesting to notice that, in the time of writing, the most expensive area is valuated more than twice the average price of the market for a 2 bedroom flat. This would be interesting to analyze in detail in order to define a pricing model that take in consideration not only the number of bedrooms and bathrooms each flat has but also the location, the venues near the flat, the criminal rate and other variables.
The standard deviation of the price of a 2 bedroom flat in London is **775.400588** and the price is **2021.686994** on average.
The **coefficient of variation** is equal to **0.3835413643661201**.

In [121]:

```
import matplotlib.pyplot as plt
df_2bed.plot('District Name', 'price_x', kind='bar', figsize=(25,8))

plt.xlabel('District Name')
plt.ylabel("Average Price per month")
plt.title("Price for a 2 bedroom flat in London")
plt.xticks(size = 12)
plt.yticks(size = 12)
plt.show()
```

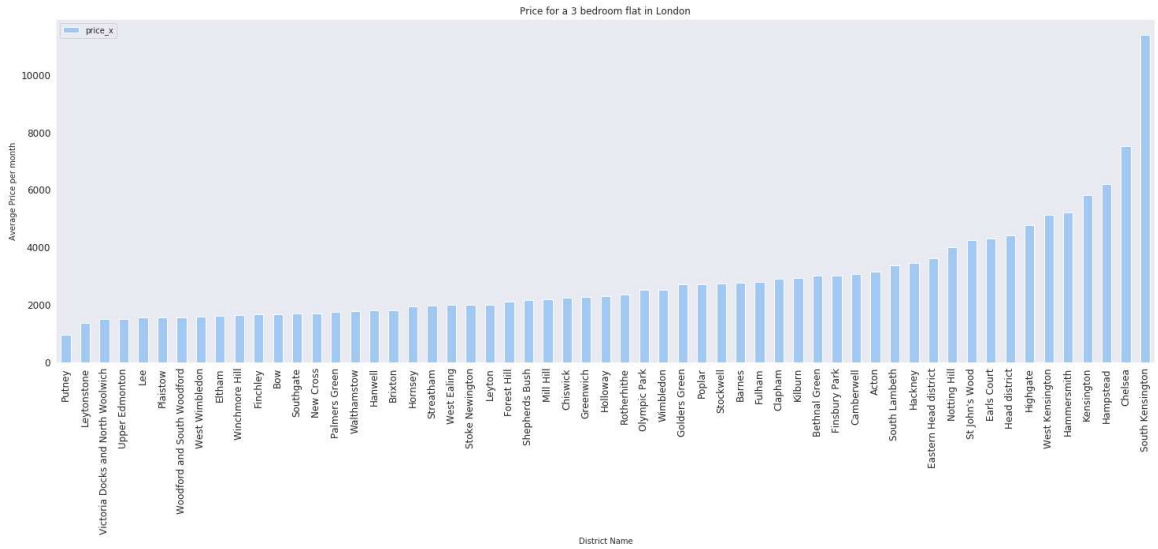


Average price for a 3 bedroom flat by district

If you are looking for a 3 bedroom flat, the distribution is different.
The standard deviation of the price of a 3 bedroom flat in London is **1760.560347** and the price is **2880.458934** on average.
The **coefficient of variation** is equal to **0.6112082787291006**.

In [128]:

```
import matplotlib.pyplot as plt
df_2bed.plot('District Name', 'price_x', kind='bar', figsize=(25,8))

plt.xlabel('District Name')
plt.ylabel("Average Price per month")
plt.title("Price for a 3 bedroom flat in London")
plt.xticks(size = 12)
plt.yticks(size = 12)
plt.show()
```

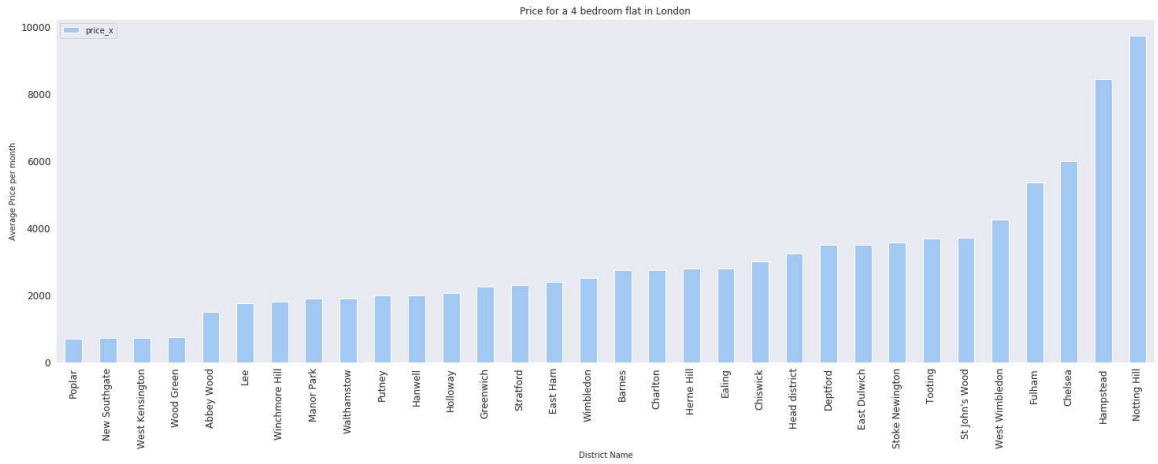


Average price for a 4 bedroom flat by district

Finally, some data about the market of the 4 bedrooms flats.
The standard deviation of the price of a 4 bedroom flat in London is **2019.186818** and the price is **3010.632161** on average.
The **coefficient of variation** is equal to **0.670685327871245**.

In [133]:

```
import matplotlib.pyplot as plt
df_2bed.plot('District Name', 'price_x', kind='bar', figsize=(25,8))

plt.xlabel('District Name')
plt.ylabel("Average Price per month")
plt.title("Price for a 4 bedroom flat in London")
plt.xticks(size = 12)
plt.yticks(size = 12)
plt.show()
```



Summary of the previous analysis

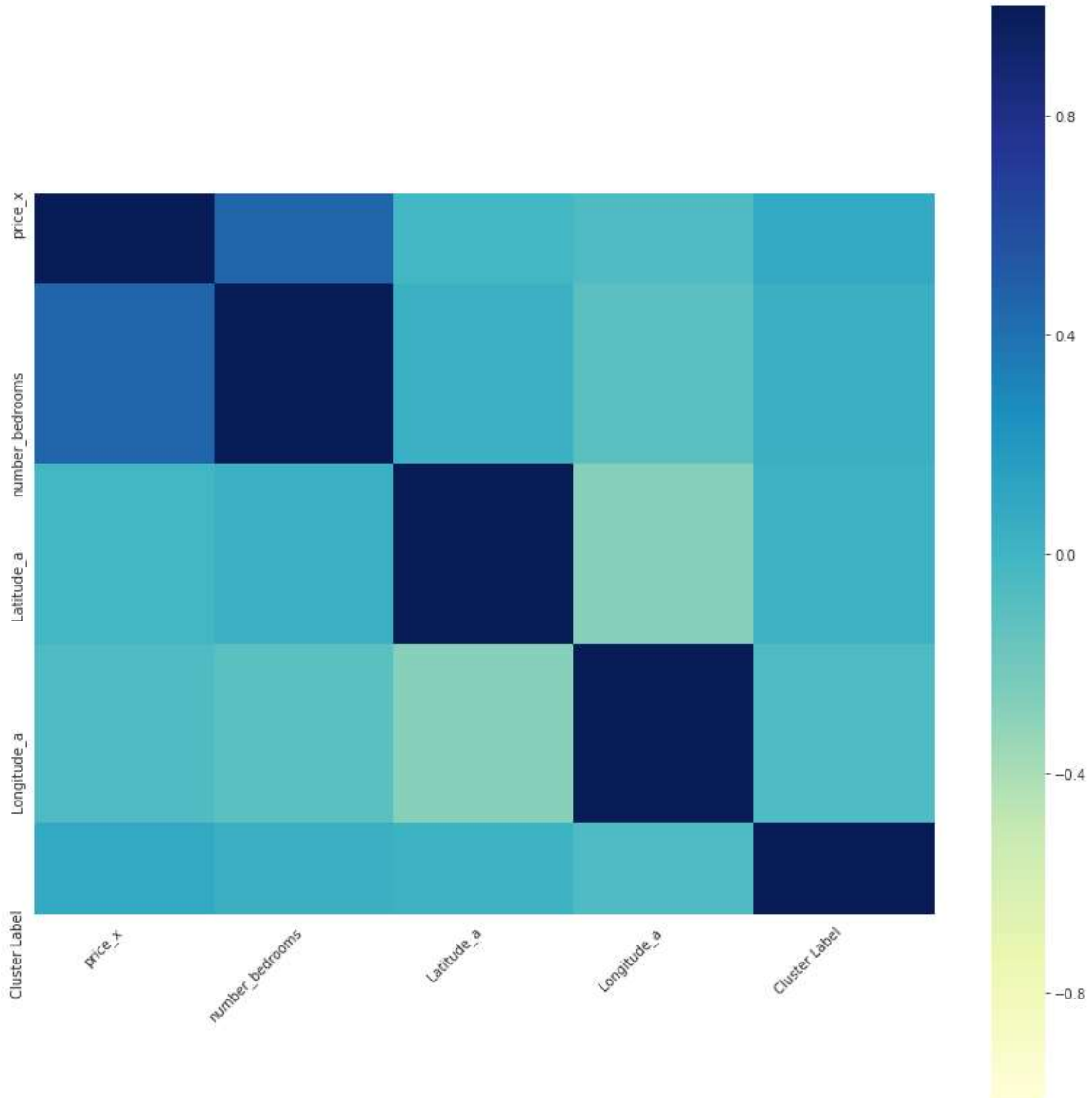
As we can see, some of the districts in London are very expensive compare to others for the same category of flat.
Following the data, the cheapest solution is to live with other people and share the flat but valuating the convenience of a flat based on the coefficient of variation, the best solution is to rent a 1 bedroom flat because the location of the building doesn't make a lot of difference in term of price.

Correlation matrix

After the previous analysis we could wonder if there are any correlations between some of the variables.

In [187]:

```
from matplotlib import pyplot

corr = df_final_clean.corr()

plt.figure(figsize=(15,15))

ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    #cmap=sns.diverging_palette(20, 220, n=200),
    cmap="YlGnBu",
    square=True

)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
```



As expected, the only thing we can understand using the correlation matrix is that there is a correlation between price and the number of bedrooms.

The Final Input: find your best offer!

Find the flats available on the market with the neighbourhood you desire

As a starting point for this last section of the Notebook, we are going to load the K-Means model (it's necessary if it has been developed in a different Notebook), ask to the user to insert **a location** he would love to live in, **the range of price** and the **number of rooms** he/she is looking for.

With the data submitted by the user, we are going to **analyze the neighbor he/she is looking for** and predict the cluster it belongs to.
Based on these information, we can **provide a list of the best opportunities on the market**.

In [87]:

```
k_clusters_fit = pickle.load(open("k_clusters_fit.pkl", "rb"))
df_dcategories_nn = pd.read_csv('df_dcategries_nn.csv')
```

In [92]:

```
address = input('Insert your address')
bedroom = input('How many badrooms are you looking for?')
price = input("What's your budget?")
```

Find the venues around the address as a sample using OpenCage Geocode.

In [93]:

```
key = 'REPLACE WITH YOUR PERSONAL KEY'
geocoder = OpenCageGeocode(key)

query = u'{}'.format(address)
results = geocoder.geocode(query)

latitude = results[0]['geometry']['lat']
longitude = results[0]['geometry']['lng']

print(u'%f;%f;%s;%s' % (results[0]['geometry']['lat'],
                        results[0]['geometry']['lng'],
                        results[0]['components']['country_code'],
                        results[0]['annotations']['timezone']['name']))
```

```
51.512070;-0.157574;gb;Europe/London
```

In [94]:

```
df_your_location = pd.DataFrame(columns = ['address', 'latitude', 'longitude', 'number
 bedrooms'])
df_your_location = df_your_location.append(pd.Series([address, latitude, longitude, int
(bedroom)], index=df_your_location.columns), ignore_index = True)
```

In [95]:

```
columns_list = df_dcategories_nn.columns
columns_list = columns_list.tolist()
```

Set up the FourSquare Key to collect the data we need about the address.

In [ ]:

```
CLIENT_ID = 'REPLACE WITH YOUR CLIENT ID' # your Foursquare ID
CLIENT_SECRET = 'REPLACE WITH YOUR SECRET KEY' # your Foursquare Secret
VERSION = '20180605' # Foursquare API version

print('Your credentails:')
print('CLIENT_ID: ' + CLIENT_ID)
print('CLIENT_SECRET:' + CLIENT_SECRET)
```

In [97]:

```
def getNearbyVenuesExp(names, latitudes, longitudes, bedrooms, radius = 500, LIMIT = 100):

    venues_list=[]
    for name, lat, lng, bedroom in zip(names, latitudes, longitudes, bedrooms):
        print(name)

        # create the API request URL
        url = 'https://api.foursquare.com/v2/venues/explore?&client_id={}&client_secret={}&v={}&ll={},{}&radius={}&limit={}'.format(
            CLIENT_ID,
            CLIENT_SECRET,
            VERSION,
            lat,
            lng,
            radius,
            LIMIT)

        if (lat != np.nan) and (lng != np.nan):

            # make the GET request
            results = requests.get(url).json()["response"]['groups'][0]['items']

            # return only relevant information for each nearby venue
            venues_list.append([(
                name,
                lat,
                lng,
                bedroom,
                v['venue']['name'],
                v['venue']['location']['lat'],
                v['venue']['location']['lng'],
                v['venue']['categories'][0]['name']) for v in results])

            nearby_venues = pd.DataFrame([item for venue_list in venues_list for item in venue_list])
            nearby_venues.columns = ['address',
                                     'latitude',
                                     'longitude',
                                     'bedroom',
                                     'Venue',
                                     'Venue Latitude',
                                     'Venue Longitude',
                                     'Venue Category']
        else:
            continue

    return(nearby_venues)
```

In [98]:

```
df_address_venues = getNearbyVenuesExp(names = df_your_location['address'],
                                       latitudes = df_your_location['latitude'],
                                       longitudes = df_your_location['longitude'],
                                       bedrooms = df_your_location['number bedrooms']
                                      )
```

```
45 Park Lane, London, W1K 1PM
```

In [99]:

```
df_address_venues.groupby(['address', 'bedroom']).count()
```

Out[99]:

		latitude	longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
address	bedroom						
45 Park Lane, London, W1K 1PM	2	100	100	100	100	100	100

In [101]:

```
# one hot encoding
df_venues_dummies = pd.get_dummies(df_address_venues[['Venue Category']], prefix="", pr
efix_sep="")

# add neighbourhood column back to dataframe
df_venues_dummies['address'] = df_address_venues['address']
df_venues_dummies['bedroom'] = df_address_venues['bedroom']

# move address column to the first column
temp_address = df_venues_dummies['address']
df_venues_dummies.drop(labels=['address'], axis=1,inplace = True)
df_venues_dummies.insert(0, 'address', temp_address)

# move bedroom column to the second column
temp_i = df_venues_dummies['bedroom']
df_venues_dummies.drop(labels=['bedroom'], axis=1,inplace = True)
df_venues_dummies.insert(1, 'bedroom', temp_i)


df_dcategories = df_venues_dummies.groupby(['address', 'bedroom']).mean().reset_index()
df_dcategories.head()
```

Out[101]:

	address	bedroom	Accessories Store	American Restaurant	Bakery	Bar	Boutique	Café	Clothing Store	Cl Hou
0	45 Park Lane, London, W1K 1PM	2	0.01	0.02	0.03	0.01	0.02	0.05	0.04	0.

In [102]:

```
# Create a dataframe with the top 20 venues

def return_most_common_venues(row, num_top_venues):
    row_categories = row.iloc[1:]
    row_categories_sorted = row_categories.sort_values(ascending=False)

    return row_categories_sorted.index.values[0:num_top_venues]
```

In [103]:

```
num_top_venues = 20

if num_top_venues <= (len(df_dcategories.columns) - 2):
    num_top_venues = num_top_venues
else:
    num_top_venues = (len(df_dcategories.columns) - 2)

indicators = ['st', 'nd', 'rd']

# create columns according to number of top venues
columns = ['address', 'bedroom']
for ind in np.arange(num_top_venues):
    try:
        columns.append('{}{} Most Common Venue'.format(ind+1, indicators[ind]))
    except:
        columns.append('{}th Most Common Venue'.format(ind+1))

# create a new dataframe
address_venues_sorted = pd.DataFrame(columns=columns)
address_venues_sorted['address'] = df_dcategories['address']
address_venues_sorted['bedroom'] = df_dcategories['bedroom']

for ind in np.arange(df_dcategories.shape[0]):
    address_venues_sorted.iloc[ind, 2:] = return_most_common_venues(df_dcategories.iloc
[ind, 1:], num_top_venues)

address_venues_sorted.head()
```

Out[103]:

	address	bedroom	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4th Most Common Venue	5th Most Common Venue	6th Most Common Venue	7th M Comm Ver
0	45 Park Lane, London, W1K 1PM	2	Hotel	Coffee Shop	Hotel Bar	Italian Restaurant	Café	Restaurant	Cloth St

In [104]:

```
df_prediction = df_dcategories.drop(columns = ['address', 'bedroom'])
```

In order to predict the cluster the address belongs to, we need to have the same number of features.
The following code take the list of columns of the dataset used to train the K-Means algorithm and update the records using the values we collected in relation of the address that the user has submitted.

In [105]:

```
df_col = {}

for col in columns_list[1:]:
    df_col[col] = 0

df_columns_df = pd.DataFrame.from_dict(df_col, orient = 'index')
df_columns_df = df_columns_df.transpose()

df_columns_df.update(df_prediction)
df_columns_df
```

Out[105]:

	ATM	Accessories Store	Acupuncturist	Adult Boutique	Afghan Restaurant	African Restaurant	American Restaurant	Animal Shelter
0	0	0.01	0	0	0	0	0.02	0

Now we can finally predict the cluster!

In [106]:

```
cluster_target = k_clusters_fit.predict(df_columns_df)[0]
```

In [107]:

```
cluster_target
```

Out[107]:

7

And show a list of offers based on geographical preferences of the user.

In [108]:

```
df_final_clean.head(2)
```

Out[108]:

	price_x	type_x	address_x	url	ager
0	35750.0	6 bedroom house	Drayton Gardens, Chelsea, London, SW10	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.uk/e agents/ag
1	2881.0	2 bedroom apartment	Earls Court , London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.co.uk/e agents/ag

In [109]:

```
df_final_clean[
    (df_final_clean.number_bedrooms == float(bedroom))
    & (df_final_clean['Cluster Label'] == cluster_target)
    & (df_final_clean.price_x <= float(price))
].sort_values(by='price_x', ascending = True).drop(columns=["District Name", "Latitude"
, "Longitude", "address", "price_y", "type_y"])
```

Out[109]:

	price_x	type_x	address_x	url	
128	1300.0	2 bedroom flat	Rochdale Road - Walthamstow - E17	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
477	1350.0	2 bedroom flat	South Vale, London	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
730	1350.0	2 bedroom maisonette	Connell Crescent, London, W5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
788	1400.0	2 bedroom apartment	Bowes Lyon Hall, LONDON, E16	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
837	1400.0	2 bedroom flat	Helmsley, Cleveland Road, E18	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
207	1550.0	2 bedroom flat	Metro Building, 148 Major Road, London, E15	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
720	1650.0	2 bedroom apartment	Azura Court, Stratford, London	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
667	1700.0	2 bedroom flat	Greystoke House, Brunswick Road, Ealing, Londo...	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
437	1820.0	2 bedroom flat	Norman Road, Greenwich, London, SE10	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
501	2058.0	2 bedroom flat	Nevern Square, Earls Court	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
35	2080.0	2 bedroom flat	Elsham Road, London, W14	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
238	2249.0	2 bedroom flat	Uxbridge Road, Ealing, W5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
823	2249.0	2 bedroom apartment	Nevern Square, Earls Court, London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag

	price_x	type_x	address_x	url	
483	2249.0	2 bedroom apartment	Philbeach Gardens, Earls Court, London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
388	2250.0	2 bedroom apartment	Legacy Tower, 88 Great Eastern Road, London, E15	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
882	2253.0	2 bedroom flat	Belgrave Road London SW1V	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
448	2384.0	2 bedroom flat	Courtfield Gardens, South Kensington, London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
271	2392.0	2 bedroom flat	Uxbridge Road, Ealing, W5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
436	2492.0	2 bedroom flat	Cleveland Terrace, London, W2	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
1015	2600.0	2 bedroom flat	Boardwalk Place, E14	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
319	2750.0	2 bedroom flat	Block B, London, E14	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
1010	2817.0	2 bedroom flat	Lexham Gardens, London, W8	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
230	2817.0	2 bedroom flat	Hogarth Road, Earl's Court, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
1	2881.0	2 bedroom apartment	Earls Court , London, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
195	2950.0	2 bedroom flat	St Georges Road, London, SE1	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
389	2968.0	2 bedroom apartment	Elvaston Place, South Kensington, London, SW7	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag

	price_x	type_x	address_x	url	
735	3250.0	2 bedroom maisonette	Wetherby Place South Kensington SW7	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
715	3358.0	2 bedroom flat	Montagu Mews South, Marylebone	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
755	3358.0	2 bedroom mews house	Montagu Mews South, Marylebone, London, W1H	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
562	3358.0	2 bedroom mews house	Montagu Mews South London W1H	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
890	3445.0	2 bedroom flat	Hyde Park Gardens, London, W2	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
833	3684.0	2 bedroom flat	Randolph Avenue, Little Venice, London, W9	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag
263	3900.0	2 bedroom flat	Hogarth Road, Earl's Court, SW5	http://www.rightmove.co.uk/property-to-rent/pr...	http://www.rightmove.c ag

Conclusion

In this project, we went through the identification of a business problem and the data we need to collect in order to solve it. In the first section of the paper, we focus on collecting the data we need. In the second section we focuse on data cleaning and we expand the dataset using multiple sources and API. After that, we developed a machine learning model to divide the market into 20 clusters by geographical similarity.

The last section is dedicated to the user, allowing him to input the address of a street where he would like to live, the number of bedrooms he needs and the price range. This allows each user to identify the cluster that best suits their needs and empower people to filter an huge amount of data quickly.

Developing this project had been a great experiance and gave me a lot of insight of how the project could be expanded to define when a flat is overpriced or underpriced. We could also study the real estate value based on criminality rate, location, geographical information and so on.

Thank You!

Federico Sciuca

[Contact me on LinkedIn (https://www.linkedin.com/in/federico-sciuca/)](https://www.linkedin.com/in/federico-sciuca/)
