Research analysis for London and Paris

1. Introduction

We are investigating the data of the two most popular capitals of Europe, London and Paris. We use the neighbourhoods data of these cities and we are trying find out out how they appear right now

2. Business Problem

The purpose of this research is the importance of the cities neighbourhoods to the life of the cities. As both of the cities are top tourist destinations,the neighbourhoods play significant role to the tourism. Tourists are looking what each neighbourhood offer. Also anyone that want to move to these cities, has the oppoortunity to decide which neighbourhood is better for living.

3. Data Description

We started with geographical location from both cities and postal codes are the first thing that we looked for. This helped us to find all the useful information that we wanted such as neighboroughs, boroughs

3.1 London

To fid out data for London solution, we use the following link
https://en.wikipedia.org/wiki/List_of_areas_of_London

1. *borough* : Name of Neighbourhood
2. *town* : Name of borough
3. *post_code* : Postal codes for London.

As this page does not contain any geograpcical location, the ArcGIS API was used

3.2 ArcGIS API

ArcGIS Online is a cloud-based mapping and analysis solution. Use it to make maps, analyze data, and to share and collaborate. Get access to workflow-specific apps, maps and data from around the globe, and tools for being mobile in the field

This tool helps us to find the latitude and the longtitude.

1. *latitude* : Latitude for Neighbourhood
2. *longitude* : Longitude for Neighbourhood

3.3 Paris

We use the following JSON data available at https://www.data.gouv.fr/fr/datasets/r/e88c6fda-1d09-42a0-a069-606d3259114e

1. *postal_code* : Postal codes for France
2. *nom_comm* : Name of Neighbourhoods in France
3. *nom_dept* : Name of the boroughs, equivalent to towns in France
4. *geo_point_2d* : Tuple containing the latitude and longitude of the Neighbourhoods.

3.4 Foursquare API Data

Foursquare City Guide, commonly known as Foursquare, is a local search-and-discovery mobile app developed by Foursquare Labs Inc. The app provides personalized recommendations of places to go near a user's current location based on users' previous browsing history and check-in history

We use this tool about different venues in different neighbourhoods of that specific borough

After finding the list of neighbourhoods, we then connect to the Foursquare API to gather information about venues inside each and every neighbourhood. For each neighbourhood, we have chosen the radius to be 500 meters.

The data retrieved from Foursquare contained information of venues within a specified distance of the longitude and latitude of the postcodes. The information obtained per venue as follows:

1. *Neighbourhood* : Name of the Neighbourhood
2. *Neighbourhood Latitude* : Latitude of the Neighbourhood
3. *Neighbourhood Longitude* : Longitude of the Neighbourhood
4. *Venue* : Name of the Venue
5. *Venue Latitude* : Latitude of Venue
6. *Venue Longitude* : Longitude of Venue
7. *Venue Category* : Category of Venue

After that our model is built

4. Methodology

We will be creating our model with the help of Python so we start off by importing all the required packages.

import pandas as pd import requests import numpy as np import matplotlib.cm as cm import matplotlib.colors as colors import folium from sklearn.cluster import KMeans

Package breakdown:

- *Pandas* : To collect and manipulate data in JSON and HTMl and then data analysis
- *requests* : Handle http requests

- *matplotlib* : Detailing the generated maps
- *folium* : Generating maps of London and Paris
- *sklearn* : To import Kmeans which is the machine learning model that we are using.

4.1 Data Collection

In [16]:
```
#In the data collection stage, we begin with collecting the required data for the ci

#To collect data for London, we scrape the List of areas of London wikipedia page to


url_london = "https://en.wikipedia.org/wiki/List_of_areas_of_London"
wiki_london_url = requests.get(url_london)
wiki_london_data = pd.read_html(wiki_london_url.text)
wiki_london_data = wiki_london_data[1]
wiki_london_data


#The data looks like this:
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-1074f01d6a61> in <module>
      5
      6 url_london = "https://en.wikipedia.org/wiki/List_of_areas_of_London"
----> 7 wiki_london_url = requests.get(url_london)
      8 wiki_london_data = pd.read_html(wiki_london_url.text)
      9 wiki_london_data = wiki_london_data[1]

NameError: name 'requests' is not defined
```

	Location	London borough	Post town	Postcode district	Dial code	OS grid ref
0	Abbey Wood	Bexley, Greenwich [7]	LONDON	SE2	020	TQ465785
1	Acton	Ealing, Hammersmith and Fulham[8]	LONDON	W3, W4	020	TQ205805
2	Addington	Croydon[8]	CROYDON	CR0	020	TQ375645
3	Addiscombe	Croydon[8]	CROYDON	CR0	020	TQ345665
4	Albany Park	Bexley	BEXLEY, SIDCUP	DA5, DA14	020	TQ478728
...	...	...	...	...	...	...
528	Woolwich	Greenwich	LONDON	SE18	020	TQ435795
529	Worcester Park	Sutton, Kingston upon Thames	WORCESTER PARK	KT4	020	TQ225655
530	Wormwood Scrubs	Hammersmith and Fulham	LONDON	W12	020	TQ225815
531	Yeading	Hillingdon	HAYES	UB4	020	TQ115825
532	Yiewsley	Hillingdon	WEST DRAYTON	UB7	020	TQ063804

To collect data for Paris, we download the JSON file containg all the postal codes of France from https://www.data.gouv.fr/fr/datasets/r/e88c6fda-1d09-42a0-a069-606d3259114e

Using Pandas we load the table after reading the JSON file:

```
!wget -q -O 'france-data.json'
https://www.data.gouv.fr/fr/datasets/r/e88c6fda-1d09-42a0-a069-606d3259114e
print("Data Downloaded!")
paris_raw = pd.read_json('france-data.json')
paris_raw.head()
```

	datasetid	recordid	fields	geometry	record_timestamp
0	correspondances-code-insee-code-postal	21e809b1d4480333c8b6fe7addd8f3b06f343e2c	{'code_comm': '003', 'nom_dept': 'VAL-DE-MARNE...	{'type': 'Point', 'coordinates': [2.3335102498...	2016-09-21T00:29:06.175+02:00
1	correspondances-code-insee-code-postal	c38925e974a8875071da3eb1391a6935d9c97e07	{'code_comm': '430', 'nom_dept': 'SEINE-ET-MAR...	{'type': 'Point', 'coordinates': [2.7879422114...	2016-09-21T00:29:06.175+02:00
2	correspondances-code-insee-code-postal	7c0aa8ba7a7b4320a9cf5abf12288eb76e3eead8	{'code_comm': '412', 'nom_dept': 'SEINE-ET-MAR...	{'type': 'Point', 'coordinates': [2.5107818983...	2016-09-21T00:29:06.175+02:00
3	correspondances-code-insee-code-postal	b123405b4d069c33725418aab20ca0b741f8a5d8	{'code_comm': '598', 'nom_dept': 'VAL-D'OISE',...	{'type': 'Point', 'coordinates': [2.3004997834...	2016-09-21T00:29:06.175+02:00
4	correspondances-code-insee-code-postal	33dea89ab43606076200134a51f2b9d2d7d62256	{'code_comm': '040', 'nom_dept': 'SEINE-ET-MAR...	{'type': 'Point', 'coordinates': [2.5699190953...	2016-09-21T00:29:06.175+02:00

4.2 Data Preprocessing

For London, We replace the spaces with underscores in the title.The *borough* column has numbers within square brackets that we remove using:

```
wiki_london_data.rename(columns=lambda x: x.strip().replace(" ", "_"),
inplace=True)
wiki_london_data['borough'] = wiki_london_data['borough'].map(lambda x:
x.rstrip(']').rstrip('0123456789').rstrip('['))
```

For Paris, we break down each of the nested fields and create the dataframe that we need:

```
paris_field_data = pd.DataFrame()
for f in paris_raw.fields:
    dict_new = f
    paris_field_data = paris_field_data.append(dict_new, ignore_index=True)

paris_field_data.head()
```

4.3 Feature Selection

For both of our datasets, we need only the borough, neighbourhood, postal codes and geolocations (latitude and longitude). So we end up selecting the columns that we need by:

```
df1 = wiki_london_data.drop( [ wiki_london_data.columns[0],
wiki_london_data.columns[4], wiki_london_data.columns[5] ], axis=1)

df_2 =
paris_field_data[['postal_code','nom_comm','nom_dept','geo_point_2d']]
```

4.4 Feature Engineering

Both of our Datasets actually contain information related to all the cities in the country. We can narrow down and further process the data by selecting only the neighbourhoods pertaining to 'London' and 'Paris'

```
df1 = df1[df1['town'].str.contains('LONDON')]

df_paris =
df_2[df_2['nom_dept'].str.contains('PARIS')].reset_index(drop=True)
```

Looking over our London dataset, we can see that we don't have the geolocation data. We need to extrapolate the missing data for our neighbourhoods. We perform this by leveraging the

`ArcGIS API` . With the Help of `ArcGIS API` we can get the latitude and longitude of our London neighbourhood data.

```
from arcgis.geocoding import geocode
from arcgis.gis import GIS
gis = GIS()
```

Defining London arcgis geocode function to return latitude and longitude

```
def get_x_y_uk(address1):
    lat_coords = 0
    lng_coords = 0
    g = geocode(address='{}, London, England, GBR'.format(address1))[0]
    lng_coords = g['location']['x']
    lat_coords = g['location']['y']
    return str(lat_coords) +","+ str(lng_coords)
```

Passing postal codes of london to get the geographical co-ordinates

```
coordinates_latlng_uk = geo_coordinates_uk.apply(lambda x: get_x_y_uk(x))
```

We proceed with Merging our source data with the geographical co-ordinates to make our dataset ready for the next stage

```
london_merged = pd.concat([df1,lat_uk.astype(float), lng_uk.astype(float)],
axis=1)
london_merged.columns=
['borough','town','post_code','latitude','longitude']
london_merged
```

	borough	town	post_code	latitude	longitude
0	Bexley, Greenwich	LONDON	SE2	51.49245	0.12127
1	Ealing, Hammersmith and Fulham	LONDON	W3, W4	51.51324	-0.26746
6	City	LONDON	EC3	51.51200	-0.08058
7	Westminster	LONDON	WC2	51.51651	-0.11968
9	Bromley	LONDON	SE20	51.41009	-0.05683
...	...	...	...	...	...
523	Redbridge	LONDON	IG8, E18	51.58977	0.03052
524	Redbridge, Waltham Forest	LONDON, WOODFORD GREEN	IG8	51.50642	-0.12721
527	Barnet	LONDON	N12	51.61592	-0.17674
528	Greenwich	LONDON	SE18	51.48207	0.07143
530	Hammersmith and Fulham	LONDON	W12	51.50645	-0.23691

As for our Paris dataset, we don't need to get the geo coordinates using an external data source or collect it with the `ArcGIS` API call since we already have it stored in the *geo_point_2d* column as a tuple in the df_paris dataframe.

We just need to extract the latitude and longitude for the column:

```
paris_lat = paris_latlng.apply(lambda x: x.split(',')[0])
paris_lat = paris_lat.apply(lambda x: x.lstrip('['))

paris_lng = paris_latlng.apply(lambda x: x.split(',')[1])
paris_lng = paris_lng.apply(lambda x: x.rstrip(']'))
```

```
paris_geo_lat  = pd.DataFrame(paris_lat.astype(float))
paris_geo_lat.columns=['Latitude']

paris_geo_lng = pd.DataFrame(paris_lng.astype(float))
paris_geo_lng.columns=['Longitude']
```

We then create our Paris dataset with the required information:

```
paris_combined_data = pd.concat([df_paris.drop('geo_point_2d', axis=1),
paris_geo_lat, paris_geo_lng], axis=1)
paris_combined_data
```

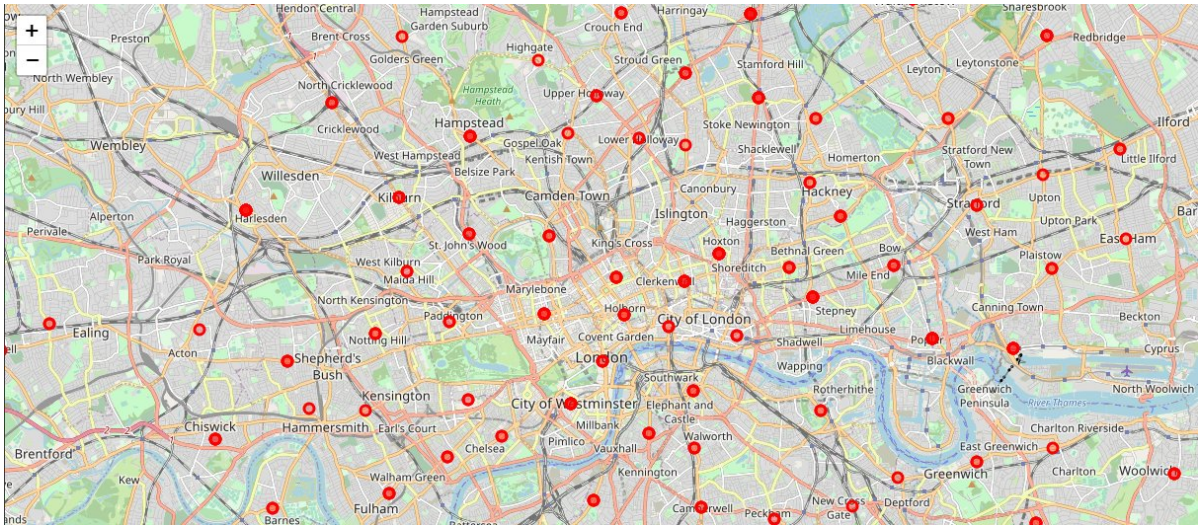
	postal_code	nom_comm	nom_dept	Latitude	Longitude
0	75010	PARIS-10E-ARRONDISSEMENT	PARIS	48.876029	2.361113
1	75016	PARIS-16E-ARRONDISSEMENT	PARIS	48.860399	2.262100
2	75009	PARIS-9E-ARRONDISSEMENT	PARIS	48.876896	2.337460
3	75015	PARIS-15E-ARRONDISSEMENT	PARIS	48.840155	2.293559
4	75002	PARIS-2E-ARRONDISSEMENT	PARIS	48.867903	2.344107
5	75011	PARIS-11E-ARRONDISSEMENT	PARIS	48.859415	2.378741
6	75005	PARIS-5E-ARRONDISSEMENT	PARIS	48.844509	2.349859
7	75019	PARIS-19E-ARRONDISSEMENT	PARIS	48.886869	2.384694
8	75020	PARIS-20E-ARRONDISSEMENT	PARIS	48.863187	2.400820
9	75003	PARIS-3E-ARRONDISSEMENT	PARIS	48.863054	2.359361
10	75006	PARIS-6E-ARRONDISSEMENT	PARIS	48.848968	2.332671
11	75018	PARIS-18E-ARRONDISSEMENT	PARIS	48.892735	2.348712
12	75008	PARIS-8E-ARRONDISSEMENT	PARIS	48.872527	2.312583
13	75013	PARIS-13E-ARRONDISSEMENT	PARIS	48.828718	2.362468
14	75012	PARIS-12E-ARRONDISSEMENT	PARIS	48.835156	2.419807
15	75007	PARIS-7E-ARRONDISSEMENT	PARIS	48.856083	2.312439

Note: Both the datasets have been properly processed and formatted. Since the same steps are applied to both the datasets of London and Paris, we will be discussing the code for only the London dataset for simplicity.

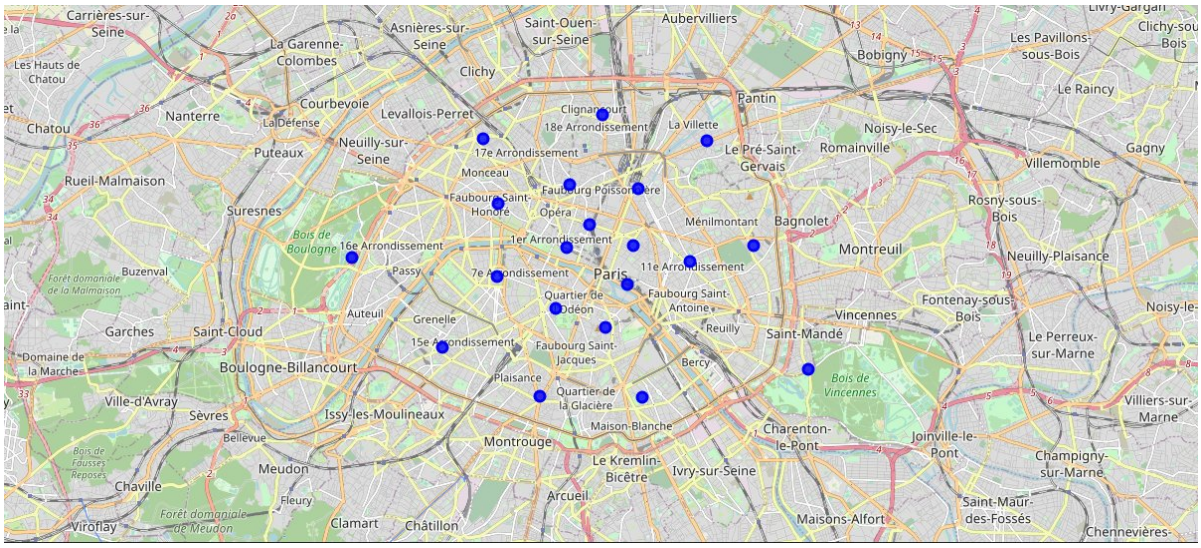
4.5 Visualizing the Neighbourhoods of London and Paris

Now that our datasets are ready, using the `Folium` package, we can visualize the maps of London and Paris with the neighbourhoods that we collected.

Neighbourhood map of London:



Neighbourhood map of Paris:



Now that we have visualized the neighbourhoods, we need to find out what each neighbourhood is like and what are the common venue and venue categories within a 500m radius.

This is where `Foursquare` comes into play. With the help of `Foursquare` we define a function which collects information pertaining to each neighbourhood including that of the name of the neighbourhood, geo-coordinates, venue and venue categories.

```
LIMIT=100

def getNearbyVenues(names, latitudes, longitudes, radius=500):

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

            # 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,
                v['venue']['name'],
                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 = ['Neighbourhood',
                  'Neighbourhood Latitude',
                  'Neighbourhood Longitude',
                  'Venue',
                  'Venue Category']

    return(nearby_venues)
```

Resulting data looks like:

	Neighbourhood	Neighbourhood Latitude	Neighbourhood Longitude	Venue	Venue Category
0	Bexley, Greenwich	51.49245	0.12127	Sainsbury's	Supermarket
1	Bexley, Greenwich	51.49245	0.12127	Lesnes Abbey	Historic Site
2	Bexley, Greenwich	51.49245	0.12127	Lidl	Supermarket
3	Bexley, Greenwich	51.49245	0.12127	Abbey Wood Railway Station (ABW)	Train Station
4	Bexley, Greenwich	51.49245	0.12127	Bean @ Work	Coffee Shop

4.6 One Hot Encoding

Since we are trying to find out what are the different kinds of venue categories present in each neighbourhood and then calculate the top 10 common venues to base our similarity on, we use the One Hot Encoding to work with our categorical datatype of the venue categories. This helps to convert the categorical data into numeric data.

We won't be using label encoding in this situation since label encoding might cause our machine learning model to have a bias or a sort of ranking which we are trying to avoid by using One Hot Encoding.

We perform one hot encoding and then calculate the mean of the grouped venue categories for each of the neighbourhoods.

```
# One hot encoding
London_venue_cat = pd.get_dummies(venues_in_London[['Venue Category']],
prefix="", prefix_sep="")

# Adding neighbourhood to the mix
London_venue_cat['Neighbourhood'] = venues_in_London['Neighbourhood']
```

```
# moving neighborhood column to the first column
fixed_columns = [London_venue_cat.columns[-1]] +
list(London_venue_cat.columns[:-1])
London_venue_cat = London_venue_cat[fixed_columns]

# Grouping and calculating the mean
London_grouped =
London_venue_cat.groupby('Neighbourhood').mean().reset_index()
```

	Neighbourhood	Accessories Store	Adult Boutique	African Restaurant	American Restaurant	Antique Shop	Arcade	Arepa Restaurant	Argentinian Restaurant	Art Gallery	Art Museum	Arts & Crafts Store	Asian Restaurant	Athletics & Sports
0	Barnet	0.0	0.0	0.0	0.001887	0.0	0.0	0.0	0.007547	0.0	0.0	0.0	0.020755	0.0
1	Barnet, Brent, Camden	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0
2	Bexley	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0
3	Bexley, Greenwich	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0
4	Bexley, Greenwich	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.000000	0.0

4.7 Top Venues in the Neighbourhoods

In our next step, We need to rank and label the top venue categories in our neighborhood.

Let's define a function to get the top venue categories in the neighbourhood

```
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]
```

There are many categories, we will consider top 10 categories to avoid data skew.

Defining a function to label them accurately

```
num_top_venues = 10

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

# create columns according to number of top venues
columns = ['Neighbourhood']
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))
```

Getting the top venue categories in the neighbourhoods of London

```
# create a new dataframe for London
neighborhoods_venues_sorted_london = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted_london['Neighbourhood'] =
London_grouped['Neighbourhood']

for ind in np.arange(London_grouped.shape[0]):
    neighborhoods_venues_sorted_london.iloc[ind, 1:] =
return_most_common_venues(London_grouped.iloc[ind, :], num_top_venues)
```

```
neighborhoods_venues_sorted_london.head()
```

	Neighbourhood	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 Most Common Venue	8th Most Common Venue	9th Most Common Venue	10th Most Common Venue
0	Barnet	Coffee Shop	Café	Grocery Store	Pub	Italian Restaurant	Supermarket	Pharmacy	Chinese Restaurant	Turkish Restaurant	Pizza Place
1	Barnet, Brent, Camden	Gym / Fitness Center	Music Venue	Clothing Store	Supermarket	Zoo Exhibit	Film Studio	Event Space	Exhibit	Falafel Restaurant	Farmers Market
2	Bexley	Supermarket	Historic Site	Train Station	Platform	Convenience Store	Coffee Shop	Bus Stop	Golf Course	Construction & Landscaping	Park
3	Bexley, Greenwich	Park	Construction & Landscaping	Sports Club	Bus Stop	Golf Course	Historic Site	Food Service	Convenience Store	Department Store	Cycle Studio
4	Bexley, Greenwich	Supermarket	Platform	Convenience Store	Historic Site	Train Station	Coffee Shop	Zoo Exhibit	Film Studio	Event Space	Exhibit

4.8 Model Building - KMeans

Moving on to the most exicitng part - **Model Building!** We will be using KMeans Clustering Machine learning algorithm to cluster similar neighbourhoods together. We will be going with the number of clusters as 5.

```
# set number of clusters
k_num_clusters = 5

London_grouped_clustering = London_grouped.drop('Neighbourhood', 1)

# run k-means clustering
kmeans_london = KMeans(n_clusters=k_num_clusters,
random_state=0).fit(London_grouped_clustering)
```

Our model has labelled each of the neighbourhoods, we add the label into our dataset.

```
neighborhoods_venues_sorted_london.insert(0, 'Cluster Labels',
kmeans_london.labels_ +1)
```

We then join London_merged with our neighbourhood venues sorted to add latitude & longitude for each of the neighborhood to prepare it for visualization.

```
london_data = london_merged

london_data =
london_data.join(neighborhoods_venues_sorted_london.set_index('Neighbourhood'),
 on='borough')

london_data.head()
```

	borough	town	post_code	latitude	longitude	Cluster Labels	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 Most Common Venue	8th M Comr Venu
0	Bexley, Greenwich	LONDON	SE2	51.49245	0.12127	4	Supermarket	Platform	Convenience Store	Historic Site	Train Station	Coffee Shop	Zoo Exhibit	Film Studi
1	Ealing, Hammersmith and Fulham	LONDON	W3, W4	51.51324	-0.26746	1	Grocery Store	Train Station	Breakfast Spot	Park	Indian Restaurant	Deli / Bodega	Fish Market	Exhib
6	City	LONDON	EC3	51.51200	-0.08058	2	Coffee Shop	Italian Restaurant	Hotel	Pub	Gym / Fitness Center	Food Truck	Sandwich Place	Beer
7	Westminster	LONDON	WC2	51.51651	-0.11968	2	Hotel	Coffee Shop	Pub	Sandwich Place	Café	Italian Restaurant	Restaurant	Theat
9	Bromley	LONDON	SE20	51.41009	-0.05683	2	Supermarket	Grocery Store	Convenience Store	Hotel	Fast Food Restaurant	Park	Italian Restaurant	Gym / Fitnes Cente

4.9 Visualizing the clustered Neighbourhoods

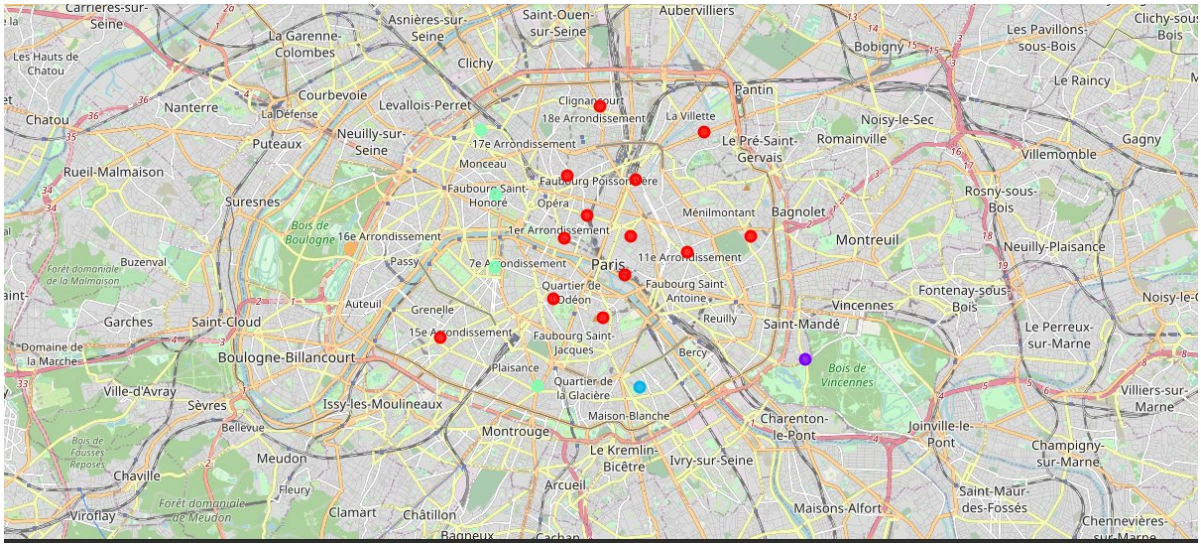
Our data is processed, missing data is collected and compiled. The Model is built. All that's remaining is to see the clustered neighbourhoods on the map. Again, we use `Folium` package to do so.

We drop all the NaN values to prevent data skew

```
london_data_nonan = london_data.dropna(subset=['Cluster Labels'])
```

Map of clustered neighbourhoods of London:

Map of clustered neighbourhoods of Paris



4.9.1 Examining our Clusters

We could examine our clusters by expanding on our code using the `Cluster Labels` column:

Cluster 1

```
london_data_nonan.loc[london_data_nonan['Cluster Labels'] == 1,
london_data_nonan.columns[[1] + list(range(5,
london_data_nonan.shape[1]))]]
```

Cluster 2

```
london_data_nonan.loc[london_data_nonan['Cluster Labels'] == 2,
london_data_nonan.columns[[1] + list(range(5,
london_data_nonan.shape[1]))]]
```

Cluster 3

```
london_data_nonan.loc[london_data_nonan['Cluster Labels'] == 3,
london_data_nonan.columns[[1] + list(range(5,
london_data_nonan.shape[1]))]]
```

Cluster 4

```
london_data_nonan.loc[london_data_nonan['Cluster Labels'] == 4,
london_data_nonan.columns[[1] + list(range(5,
london_data_nonan.shape[1]))]]
```

Cluster 5

```
london_data_nonan.loc[london_data_nonan['Cluster Labels'] == 5,
london_data_nonan.columns[[1] + list(range(5,
london_data_nonan.shape[1]))]]
```

5. Results and Discussion

London is a multicultural capital. Anyone can find different cuisines such as Chinese and Indian. It has many pubs, many restaurants, coffee shops etc. There are many stores. One can travel by bus or train. It has a zoo and many parks. In general, London is a city that you csn find anything that you want

Paris is smaller in size. It also has many different cuisines such as French and Chinese. There are many restaurants here, as well as many small bistros. In Paris, many use public transport and bicycles. It has many fashion stores and many museums. In general, Paris is a more alternative and relaxing option

6. Conclusion

The purpose of this project was to explore the cities of London and Paris and see how attractive it is to potential tourists and migrants. We explored both the cities based on their postal codes and then extrapolated the common venues present in each of the neighbourhoods finally concluding with clustering similar neighbourhoods together.

We could see that each of the neighbourhoods in both the cities have a wide variety of experiences to offer which is unique in it's own way. The cultural diversity is quite evident which also gives the feeling of a sense of inclusion.

Both Paris and London seem to offer a vacation stay or a romantic getaway with a lot of places to explore, beautiful landscapes, amazing food and a wide variety of culture. Overall, it's upto the stakeholders to decide which experience they would prefer more and which would more to their liking.

The detailed code is available on https://github.com/giangalis/Coursera_Capstone-Week-4---The-Battle-of-Neighborhoods/blob/main/Coursera_Capstone%20Week%204%20DS.ipynb

7. References

1. The Battle of Neighbourhood — My London's Perspective by Dayo John
2. The Battle of neighborhoods! What is the best place where can I start my restaurant business in Paris? by Zakaria BOUZIANE
3. Foursquare API
4. ArcGIS API