<u>Coursera Capstone Project: Applied Data Science – The Battle of the Neighbourhoods</u>

Gabriel Leow

1. Introduction

In this project we will identify the **best neighbourhood to open a supermarket in Edinburgh**. With COVID-19 imposing lockdowns throughout the world, and more people remaining at home, the demand for supermarkets is back on the rise, as one of the few physical activities that people can leave their homes to do. As such, supermarket chains should try to quickly expand in the near future to capture this added demand.

Specifically, this report will target **large supermarket chains such as Tesco and ASDA** by offering them a data-driven solution to the most profitable location to open their next outlet which will be best received by customers. Given that there are around **160** neighbourhoods in Edinburgh, this will greatly help supermarkets to limit the range of possible options and allow them to focus their efforts on researching areas within a potential cluster.

By clustering each neighbourhood by its facilities, we will **identify which clusters has "supermarket" and "grocery stores" listed as their top few locations** as it is a strong indication of profitability. We will use our data science powers to first, generate a list of Edinburgh's 160 neighbourhoods. Second, to merge this list with location data from Foursquare. We will then use K-Means Clustering to cluster the 160 neighbourhoods.

2. Data

The data in this project is obtained from Wikipedia, Geocoders, and Foursquare. We first need to identify our list of neighbourhoods in Edinburgh cities. To do so:

- we need to scrape webpages of Edinburgh's neighbourhoods (Wikipedia)
- the GPS coordinates (latitude and longitude) of each neighbourhood (Geocoders)
- List of facilities around each neighbourhood (foursquare API)

2.1 Neighbourhoods - List

The data of Edinburgh neighbourhoods was taken from the Wikipedia webpage: https://en.wikipedia.org/wiki/Category:Areas_of_Edinburgh. It was transferred into an excel CSV file and then cleaned up using Python. There are a total of 163 neighbourhoods in Edinburgh. See <u>Figures 1 and 2</u>.

Figure 1: Before Clean Up

```
[239]: df = pd.read_csv('Sg.csv')
       df
```

[239]:

	Unnamed: 0	Neighbourhood
0	1	*[[Abbeyhill]]
1	2	*[[Alnwickhill]]
2	3	*[[Ardmillan]]
3	4	*[[Baberton]]
4	5	*[[Balerno]]
...	...	...
158	159	*[[West Pilton]]
159	160	*[[Wester Broom]]
160	161	*[[Wester Hailes]]
161	162	*[[Western Harbour, Edinburgh|Western Harbour]]
162	163	*[[Westfield, Edinburgh|Westfield]]

163 rows × 2 columns

Figure 2: After Clean Up

```
[240]: df['Neighbourhood'] = [x.lstrip('*[').rstrip(']') for x in df['Neighbourhood']]
       df[['First','Last']] = df.Neighbourhood.str.split("|",expand=True)
       df=df.drop(['Unnamed: 0', 'Neighbourhood', 'Last'], axis=1)
       df.rename(columns={'First':'Neighbourhood'}, inplace=True)
       df
```

[240]:

	Neighbourhood
0	Abbeyhill
1	Alnwickhill
2	Ardmillan
3	Baberton
4	Balerno
...	...
158	West Pilton
159	Wester Broom
160	Wester Hailes
161	Western Harbour, Edinburgh
162	Westfield, Edinburgh

2.2 Neighbourhoods – GPS Coordinates (Geocode)

Before using the foursquare API, we had to append the GPS coordinates to the above dataframe. The dataframe was then saved as a CSV to check for errors. See Figure 3.

Figure 3: Append Dataframe with GPS Coordinates

```
[39]: #Code for appending Lat-Long

      location = [x for x in df['Neighbourhood'].unique().tolist()
                  if type(x) == str]
      latitude = []
      longitude =  []
      for i in range(0, len(location)):
          # remove things that does not seem usefull here
          try:
              address = location[i]
              geolocator = Nominatim(user_agent="ny_explorer")
              loc = geolocator.geocode(address)
              latitude.append(loc.latitude)
              longitude.append(loc.longitude)
              print('The geographical coordinate of location are {}, {}.'.format(loc.latitude, loc.longitude))
          except:
              # in the case the geolocator does not work, then add nan element to list
              # to keep the right size
              latitude.append(np.nan)
              longitude.append(np.nan)
      # create a dataframe with the location, latitude and longitude
      df1 = pd.DataFrame({'Neighbourhood':location,
                          'location_latitude': latitude,
                          'location_longitude':longitude})
      # merge on Neighbourhood with df to get the column
      df2 = df.merge(df1, on='Neighbourhood', how='left')
      df2.to_csv('Sg0.csv')
```

As there were a number of errors in the coordinates obtained from Geocoders, we had to manually edit the csv file using “editor – on Jupyter Notebook”, and reupload. We then plot the neighbourhoods on a folium map to visually see if there are outliers. See Figures 4 and 5.

Figure 4: re-read adjusted CSV, and clean-up

Figure 5: Plotting of Neighbourhoods of Folium Map (check for outliers)

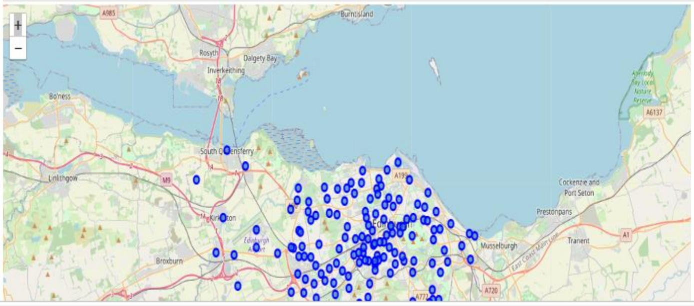
```
[241]: df2=pd.read_csv('Sg1.csv')
       df2=df2.drop(['Unnamed: 0'], axis=1)
       df2
```

[241]:

	Neighbourhood	location_latitude	location_longitude
0	Abbeyhill	55.955477	-3.170585
1	Alnwickhill	55.906087	-3.172548
2	Ardmillan	55.938466	-3.225230
3	Baberton	55.909804	-3.288869
4	Balerno	55.883914	-3.338761
...	...	...	...
158	West Pilton	55.971052	-3.245955
159	Wester Broom	55.936230	-3.291196
160	Wester Hailes	55.915225	-3.284488
161	Western Harbour, Edinburgh	55.984005	-3.180893
162	Westfield, Edinburgh	55.975494	-3.435117

```
[6]: map_edin = folium.Map(location=[55.9533,-3.1883], zoom_start=11)

     # add markers to map
     for lat, lng, Neighbourhood in zip(df2['location_latitude'], df2['location_longitude'], df2['Neighbourhood']):
         label = '{}'.format(Neighbourhood)
         label = folium.Popup(label, parse_html=True)
         folium.CircleMarker(
             [lat, lng],
             radius=5,
             popup=label,
             color='blue',
             fill=True,
             fill_color='#3186cc',
             fill_opacity=0.7,
             parse_html=False).add_to(map_edin)

     map_edin
```

[6]:



2.3 Neighbourhoods – Venue Data

The next step was to find the venue data for each neighbourhood. To do so, we used the Foursquare API, and appended the data to our original dataframe. See Figure 6.

Figure 6: Code to extract venue data from Foursquare

```
[58]: def getNearbyVenues(names, latitudes, longitudes, radius=1000, LIMIT=100):

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

              # create the API request URL for Arts and Entertainment
              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']['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 = ['Neighborhood',
                            'Neighborhood_Latitude',
                            'Neighborhood_Longitude',
                            'Venue',
                            'Venue_Latitude',
                            'Venue_Longitude',
                            'Venue_Category']

          return(nearby_venues)

[59]: edin_venues = getNearbyVenues(names=df2['Neighbourhood'],
                                        latitudes=df2['location_latitude'],
                                        longitudes=df2['location_longitude']
                                       )
```

3. Methodology

3.1 Exploratory Data Analysis

We first do a brief analysis on our obtained dataframe using the head, tail, describe and shape functions, to check if the rows and columns are as per our expectations. **We find that the data is in order, and there are sufficient (5,245) entries for us to perform data analysis on.** The 'head' of our dataframe is shown in Figure 7.

Figure 7: First 5 rows of Data Frame

[243]:

	Neighborhood	Neighborhood_Latitude	Neighborhood_Longitude	Venue	Venue_Latitude	Venue_Longitude	Venue_Category
0	Abbeyhill	55.955477	-3.170585	Regent Bar	55.956316	-3.172168	Gay Bar
1	Abbeyhill	55.955477	-3.170585	Holyrood Abbey	55.953118	-3.171659	Historic Site
2	Abbeyhill	55.955477	-3.170585	The Manna House Bakery & Patisserie	55.958138	-3.171927	Bakery
3	Abbeyhill	55.955477	-3.170585	Century General Store & Cafe	55.956678	-3.171944	Café
4	Abbeyhill	55.955477	-3.170585	Palace of Holyroodhouse	55.952666	-3.171689	Palace

We then do a slightly deeper analysis, and are satisfied with the 248 unique venue categories. We subsequently group the venue data by neighbourhoods to get a summary of each neighbourhood's venues, and prepare the data for further analysis. See Figure 8. However, we find that there are now only 161 rows, versus 163 neighbourhoods as earlier described. This means that some neighbourhoods do not have venue data in Foursquare. **We will deal with this later.**

Figure 8: Further EDA using Groupby (neighbourhood)

[246]:
```
print('There are {} uniques categories.'.format(len(edin_venues['Venue_Category'].unique())))
edin_venues.groupby('Neighborhood').count()
```

There are 248 uniques categories.

[246]:

Neighborhood	Neighborhood_Latitude	Neighborhood_Longitude	Venue	Venue_Latitude	Venue_Longitude	Venue_Category
Abbeyhill	71	71	71	71	71	71
Alnwickhill	7	7	7	7	7	7
Ardmillan	61	61	61	61	61	61
Baberton	7	7	7	7	7	7
Balerno	4	4	4	4	4	4
...	...	...	...	...	...	...
West Pilton	14	14	14	14	14	14
Wester Broom	27	27	27	27	27	27
Wester Hailes	7	7	7	7	7	7
Western Harbour, Edinburgh	39	39	39	39	39	39
Westfield, Edinburgh	1	1	1	1	1	1

161 rows × 6 columns

3.2 One-Hot Encoding for Categorical Variables

Since the venue category is in text (categorical variables), we convert it to binary (1,0) by performing one-hot encoding. See Figure 9.

Figure 9: One-Hot Encoding

```
[62]: # one hot encoding
edin_onehot = pd.get_dummies(edin_venues[['Venue_Category']], prefix="", prefix_sep="")

# add neighborhood column back to dataframe
edin_onehot['Neighborhood'] = edin_venues['Neighborhood']

#Move neighborhood column to the front
N1 = edin_onehot['Neighborhood']
edin_onehot.drop(labels=['Neighborhood'], axis=1,inplace = True)
edin_onehot.insert(0, 'Neighborhood', N1)
edin_onehot.head()
```

[62]:

	Neighborhood	Accessories Store	African Restaurant	Airport	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Argentinian Restaurant	Art Gallery	...	Waterfront	Well	Whisky Bar	Wine Bar	Wine Shop	Wings Joint	Women's Store	Yoga Studio
0	Abbeyhill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	Abbeyhill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	Abbeyhill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	Abbeyhill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	Abbeyhill	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

3.3 Derive Top 10 Venue categories for each neighbourhood

The next step is to find the top 10 venue categories for each neighbourhood. We do this by first, grouping the dataframe by 'neighbourhood' once more, then defining a 'top10' function and finally, executing the code. See Figures 10 and 11.

Figure 10: Preparing Dataframe for Top 10 Analysis

```
[63]: edin_grouped = edin_onehot.groupby('Neighborhood').mean().reset_index()
edin_grouped
```

[63]:

	Neighborhood	Accessories Store	African Restaurant	Airport	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Argentinian Restaurant	Art Gallery
0	Abbeyhill	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.014085
1	Alnwickhill	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...
159	Western Harbour, Edinburgh	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
160	Westfield, Edinburgh	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

Figure 11: Deriving the Top 10 venue categories for each neighbourhood

```
[64]: 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]
```

```
215]: num_top_venues = 10

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

# create columns according to number of top venues
columns = ['Neighborhood']
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
neighborhoods_venues_sorted = pd.DataFrame(columns=columns)
neighborhoods_venues_sorted['Neighborhood'] = edin_grouped['Neighborhood']

for ind in np.arange(edin_grouped.shape[0]):
    neighborhoods_venues_sorted.iloc[ind, 1:] = return_most_common_venues(edin_grouped.iloc[ind, :], num_top_venues)

neighborhoods_venues_sorted.head()
```

215]:

	Neighborhood	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	Abbeyhill	Café	Bar	Grocery Store	Park	Bakery	Hotel	Pub	Liquor Store	Coffee Shop	Restaurant
1	Alnwickhill	Furniture / Home Store	Pub	English Restaurant	Golf Driving Range	Park	Sporting Goods Shop	Hotel	Food Court	Food & Drink Shop	Electronics Store
2	Ardmillan	Hotel	Pub	Coffee Shop	Grocery Store	Café	Chinese Restaurant	Supermarket	Italian Restaurant	Pizza Place	Convenience Store

3.4 K-Means Clustering

We chose K-Means as the Machine Learning algorithm as the variables of venue categories are large, and K-means is computationally able to handle the calculations.

We first have to prepare the dataframe for K-Means clustering.

Figure 11: Preparing Dataframe for K-means Clustering

Methodology 6A - Prepare Dataframe by dropping neighbourhood

```
[66]: edin_grouped_clustering = edin_grouped.drop('Neighborhood', 1)
      edin_grouped_clustering.head()
```

[66]:

	Accessories Store	African Restaurant	Airport	Airport Lounge	Airport Service	Airport Terminal	American Restaurant	Argentinian Restaurant	Art Gallery	A Museu
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.014085	0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0

5 rows × 247 columns

Methodology 6B - Normalise Dataframe

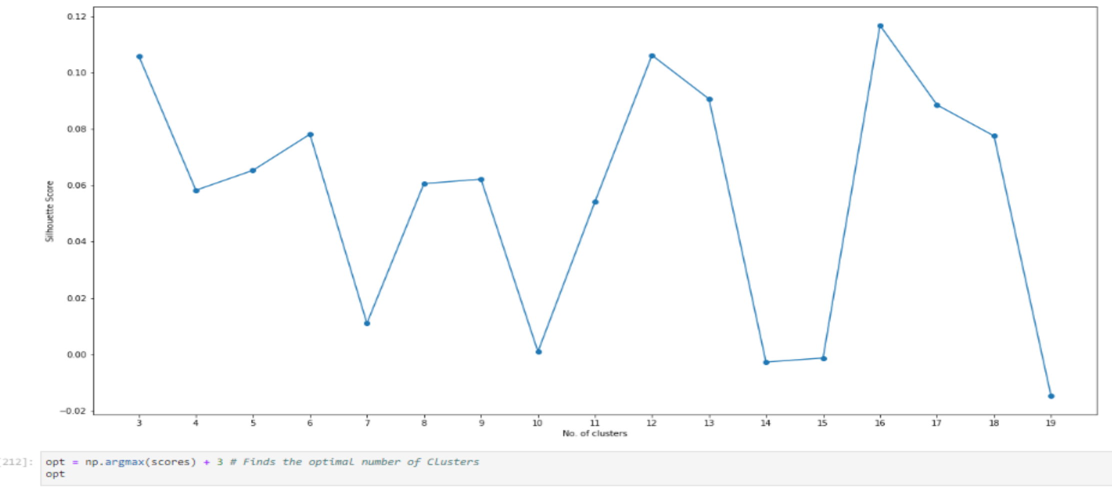
```
[79]: edin_grouped_clustering_norm = StandardScaler().fit_transform(edin_grouped_clustering)
      edin_grouped_clustering_norm

[79]: array([[-0.07905694, -0.32238937, -0.11113166, ..., -0.07905694,
              -0.11009771, -0.1017671 ],
             [-0.07905694, -0.32238937, -0.11113166, ..., -0.07905694,
              -0.11009771, -0.1017671 ],
             [-0.07905694, -0.32238937, -0.11113166, ..., -0.07905694,
              -0.11009771, -0.1017671 ],
```

Next, we have to find the optimal number of clusters. To do so, we use the silhouette score, which is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranged from -1 to +1, with +1 indicating that the model is good while -1 indicating that the model is poor. See Figures 12 and 13.

Figure 12: Code for executing Silhouette Score

```
[180]: import matplotlib.pyplot as plt
       %matplotlib inline

       def plot(x, y, xlabel, ylabel):
           plt.figure(figsize=(20,10))
           plt.plot(np.arange(3, x), y, 'o-')
           plt.xlabel(xlabel)
           plt.ylabel(ylabel)
           plt.xticks(np.arange(3, x))
           plt.show()

[181]: max_range = 20 # Maximum range of clusters

[182]: from sklearn.metrics import silhouette_samples, silhouette_score

       indices = []
       scores = []

       for kclusters in range(3, max_range) :

           # Run k-means clustering
           kmeans = KMeans(n_clusters = kclusters, init = 'k-means++', random_state = 0).fit_predict(edin_grouped_clustering_norm)

           # Gets the score for the clustering operation performed
           score = silhouette_score(edin_grouped_clustering_norm, kmeans)

           # Appending the index and score to the respective lists
           indices.append(kclusters)
           scores.append(score)

[187]: plot(max_range, scores, "No. of clusters", "Silhouette Score")
```

Figure 13: Silhouette Score Graph and Optimal Number of Clusters



The optimal number of clusters is 16. *Note that 2 clusters would have given a higher silouhette score than 16. However, as 2 clusters would not have given us meaningful information to assess the best choice of location for a new supermarket, we have excluded K=2 from the computation*.

We then perform the K-Means clustering based on number of clusters = 16. The following lines of code show the execution of K-Means, adding of cluster labels into the existing dataframe. Remember that we started off with 163 neighbourhoods, but there is only venue data for 161 of them. As such, we also drop the NA values so that the folium map can be subsequently generated. See Figure 14.

Figure 14: Running K-Means Clustering for Cluster=16, and preparing data for Folium Map

Methodology 6D - Perform K-means clustering for 16 clusters

```
#Cluster Neighborhoods
# set number of clusters
kclusters = 16

# run k-means clustering
kmeans = KMeans(n_clusters=kclusters, random_state=0).fit(edin_grouped_clustering_norm)

# check cluster labels generated for each row in the dataframe
kmeans.labels_[0:162]
```

```
array([ 7,  2,  2,  2,  2, 11,  2,  2,  7,  2, 12,  2,  2,  2,  2,  2,  7,
        9,  0,  2,  2,  2,  2,  7, 15,  0,  2,  2,  2, 13,  2,  2,  1,  2,
        2,  9, 13, 15,  2,  2,  2,  2,  2,  2,  2,  2,  8,  2,  3,  2,  2,
        2,  5,  2,  2,  2,  9,  2,  2,  2,  2,  2,  2,  8,  2,  2,  9,  2,
       11,  2,  1,  2,  2,  0,  8,  2,  0,  2,  5,  2,  4,  2,  2,  2,  2,
        2, 15,  2,  1,  7,  2,  2,  2,  8, 15,  0,  9,  7,  7,  7,  2,  2,
        2,  2,  2, 11,  6,  7,  9, 10,  7,  9,  2,  7,  2,  2,  2,  2,  7,
        2,  7,  2,  7,  2,  7,  2, 11,  2,  2,  2, 11,  2,  7,  2,  2,  2,
        2, 15,  9,  2,  2, 13,  2,  2,  5,  7,  1,  1,  2, 10,  4,  7,  2,
        8,  9,  3,  2,  2,  2, 14,  2], dtype=int32)
```

```
# add clustering labels into Dataframe
neighborhoods_venues_sorted.insert(0, 'Cluster Labels', kmeans.labels_)
```

```
edin_merged = df2

# merge toronto_grouped with toronto_data to add latitude/longitude for each neighborhood
edin_merged = edin_merged.join(neighborhoods_venues_sorted.set_index('Neighborhood'), on='Neighbourhood')
```

Methodology 6D - (A) Drop NA Values

```
edin_merged.dropna(inplace = True)
edin_merged = edin_merged.astype({"Cluster Labels": int})
```

4. Results and Discussion

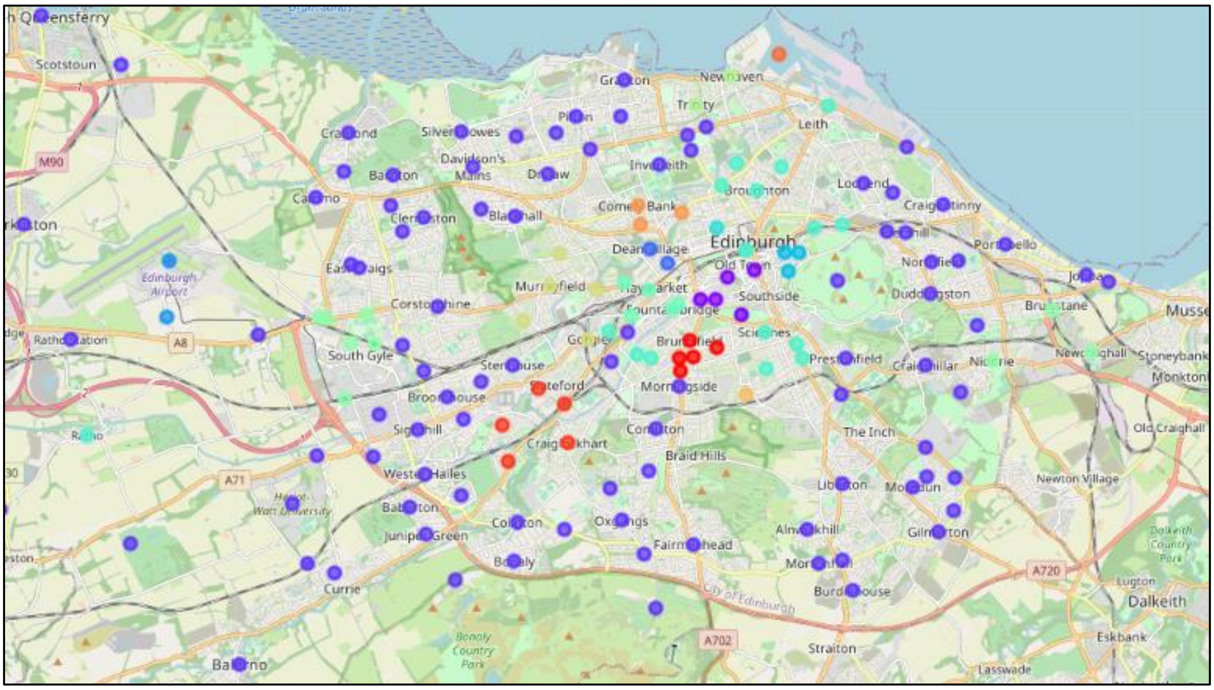
4.1 Visualisation - Folium Mapping

The neighbourhoods are divided into 16 clusters, and visualised using different colours on Folium to make them distinguishable. See Figures 15 and 16 for code and map.

Figure 15: Code for Visualisation on Folium

```
[219]: #Create map of clusters
address = 'Edinburgh, United Kingdom'

geolocator = Nominatim(user_agent="tr_explorer")
location = geolocator.geocode(address)
latitude = location.latitude
longitude = location.longitude
print('The geograpical coordinate of Edinburgh are {}, {}.'.format(latitude, longitude))

The geograpical coordinate of Edinburgh are 55.9533456, -3.1883749.

[242]: map_clusters = folium.Map(location=[latitude, longitude], zoom_start=12)

# set color scheme for the clusters
x = np.arange(kclusters)
ys = [i + x + (i*x)**2 for i in range(kclusters)]
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(edin_merged['location_latitude'], edin_merged['location_longitude'], edin_merged['Neighbourhood'], edin_merged['Cluster Labels']):
    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
```

Figure 16: Visualisation of Clusters on Folium



4.2 Cluster Analysis and Discussion

We then examine each cluster, and combine the counts of the top 10 venue categories for each Cluster into a dataframe, to see which Cluster has the highest percentage of supermarkets and grocery stores, and is therefore most suitable for the opening of a supermarket. See Figure 17 for an example of code for examining 1 specific cluster, Figure 18 for the code for combining the results of all clusters into a dataframe, and Figure 19 for what the dataframe looks like.

Figure 17: Example Code for Examining 1 Cluster

Cluster 1

```
[248]: C1 = edin_merged.loc[edin_merged['Cluster Labels'] == 0, edin_merged.columns[[0] + list(range(4, edin_merged.shape[1]))]]
C1
```

[248]:

	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
20	Bruntsfield	Coffee Shop	Pub	Grocery Store	Café	Hotel	Restaurant	Italian Restaurant	Park	Movie Theater	Bakery
29	Church Hill, Edinburgh	Coffee Shop	Pub	Café	Grocery Store	Italian Restaurant	Bakery	Restaurant	Fish & Chips Shop	Park	Sandwich Place
78	Greenhill, Edinburgh	Coffee Shop	Pub	Grocery Store	Café	Bar	Indian Restaurant	Restaurant	Italian Restaurant	Bakery	Sandwich Place
81	Holy Corner	Coffee Shop	Pub	Grocery Store	Italian Restaurant	Café	Pizza Place	Indian Restaurant	Restaurant	Hotel	Bakery
100	Marchmont	Coffee Shop	Café	Italian Restaurant	Bakery	Pub	Restaurant	Grocery Store	Hotel	Movie Theater	Park

Figure 18: Code for combining Cluster results into 1 Dataframe

Results 3 - Combining Clusters to see occurrence of categories in clusters

```
[290]: D1=C1.stack().value_counts()
D2=C2.stack().value_counts()
D3=C3.stack().value_counts()
D4=C4.stack().value_counts()
D5=C5.stack().value_counts()
D6=C6.stack().value_counts()
D7=C7.stack().value_counts()
D8=C8.stack().value_counts()
D9=C9.stack().value_counts()
D10=C10.stack().value_counts()
D11=C11.stack().value_counts()
D12=C12.stack().value_counts()
D13=C13.stack().value_counts()
D14=C14.stack().value_counts()
D15=C15.stack().value_counts()
D16=C16.stack().value_counts()
```

Results 3A - Combining Clusters and Cleaning Dataframe

```
[332]: Cluster_List = pd.DataFrame(
    {'Cluster 1': D1,
     'Cluster 2': D2,
     'Cluster 3': D3,
     'Cluster 4': D4,
     'Cluster 5': D5,
     'Cluster 6': D6,
     'Cluster 7': D7,
     'Cluster 8': D8,
     'Cluster 9': D9,
     'Cluster 10': D10,
     'Cluster 11': D11,
     'Cluster 12': D12,
     'Cluster 13': D13,
     'Cluster 14': D14,
     'Cluster 15': D15,
     'Cluster 16': D16
    })
pd.set_option("display.max_rows", 120)
Cluster_List=Cluster_List.replace(1.0,np.NaN)
Cluster_List.dropna(axis = 0, how = 'all', inplace=True)
Cluster_List=Cluster_List.replace(np.NaN,0.0)
Cluster_List.head()
```

Figure 19: Dataframe of Cluster_List

[332]:

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6	Cluster 7	Cluster 8	Cluster 9	Cluster 10	Cluster 11	Cluster 12	Cluster 13	Cluster 14	Cluster 15	Cluster 16
Airport Lounge	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	0.0	0.0
Airport Terminal	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	0.0	0.0
Art Gallery	0.0	0.0	3.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
Athletics & Sports	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Auto Garage	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

We then perform further analysis on the dataframe. Instead of having the number of counts for each cluster, for example, cluster 1 has a total of 50 venue categories, 20 are theatres, 10 are supermarkets, 10 are Cafes and 10 are hotels, we convert this to a ratio. For example, cluster 1 has 40% theatres, 20% supermarkets, 20% cafes and 20% hotels. This will tell us what the proportion of venue category in that cluster is. In addition, we remove irrelevant venue categories, reducing it to just grocery stores and supermarkets, and sort the clusters by their percentage of supermarkets. See Figure 20 and 21.

Figure 20: Convert absolute count to ratio

```
[364]: Cluster_List.loc['Total',:]= Cluster_List.sum(axis=0)
       Cluster2=Cluster_List.transpose()
       Cluster3 = Cluster2.loc[:,'Airport Lounge':'Zoo Exhibit'].div(Cluster2["Total"], axis=0)
       Cluster3
```

Figure 21: Remove unnecessary venue categories and sort by percentage of supermarkets in clusters

```
[365]: Cluster4=Cluster3[['Grocery Store','Supermarket']]
       Percentage_GC_SP=Cluster4*100
       Percentage_GC_SP.sort_values('Supermarket', ascending=False)
```

[365]:

	Grocery Store	Supermarket
Cluster 11	0.000000	5.555556
Cluster 16	3.418803	4.273504
Cluster 10	0.000000	1.754386
Cluster 12	3.508772	1.754386
Cluster 3	1.958610	1.589061
Cluster 8	2.123142	0.636943
Cluster 1	3.546099	0.000000
Cluster 2	0.000000	0.000000
Cluster 4	0.000000	0.000000
Cluster 5	0.000000	0.000000
Cluster 6	0.000000	0.000000
Cluster 9	1.587302	0.000000
Cluster 14	0.000000	0.000000
Cluster 7	NaN	NaN
Cluster 13	NaN	NaN
Cluster 15	NaN	NaN

From this table, we can see that Cluster 11 and 16 are the most probable clusters. We therefore pull out its individual records. See Figures 22 and 23.

Figure 22: Cluster 11's results

Cluster 11

```
[289]: C11 = edin_merged.loc[edin_merged['Cluster Labels'] == 10, edin_merged.columns[[0] + list(range(4, edin_merged.shape[1]))]]
       C11
```

[289]:

	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
115	Newhaven, Edinburgh	Hotel	Café	Park	Supermarket	Grocery Store	Clothing Store	Food & Drink Shop	Sandwich Place	Bike Trail	Street Food Gathering
151	Trinity, Edinburgh	Pub	Park	Hotel	Café	Food & Drink Shop	Rugby Pitch	Bike Trail	Supermarket	Climbing Gym	Trail

Figure 23: Cluster 16's results

Cluster 16

```
[266]: C16=edin_merged.loc[edin_merged['Cluster Labels'] == 15, edin_merged.columns[[0] + list(range(4, edin_merged.shape[1]))]]
C16
```

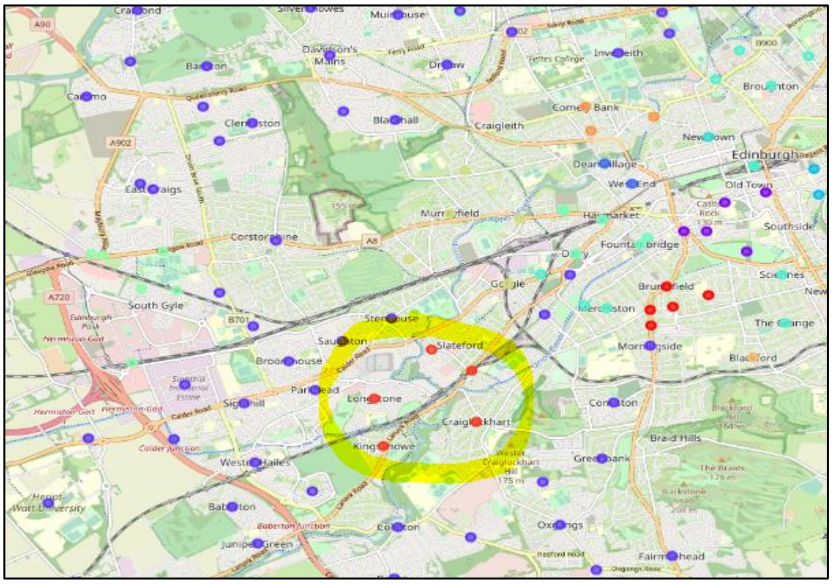
[266]:

	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
28	Chesser	Supermarket	Grocery Store	Gas Station	Auto Garage	Soccer Field	Bowling Alley	Fast Food Restaurant	Market	Tourist Information Center	Chinese Restaurant
41	Craiglockhart	Soccer Field	Stadium	Supermarket	Trail	Grocery Store	Gas Station	Tennis Court	Coffee Shop	Fast Food Restaurant	Market
91	Kingsknowe	Trail	Gas Station	Chinese Restaurant	Supermarket	Market	Train Station	Deli / Bodega	Department Store	Fountain	Forest
99	Longstone, Edinburgh	Supermarket	Museum	Fast Food Restaurant	Tourist Information Center	Market	Trail	Train Station	Chinese Restaurant	Grocery Store	Gym
143	Slateford	Grocery Store	Supermarket	Coffee Shop	Trail	Gym / Fitness Center	Bowling Alley	Soccer Field	Fast Food Restaurant	Nature Preserve	Gas Station

From the 2 results, we can see that first, Cluster 11 only has 2 neighbourhoods while Cluster 16 has 5 neighbourhoods. Second, the occurrence of Supermarkets and Neighbourhoods in Cluster 16 is more intensive than in Cluster 11. Third and most importantly, Supermarkets and grocery stores are more significant in Cluster 16 as it is often in the top 3 most common venues whereas for Cluster 11, they are in the 4-10th positions.

From Figure 24, we can see that Cluster 16 (red) are clustered within the same geographical area. It also sits along a main train track/line going to one of Scotland's biggest cities – Glasgow. This could be a strong explanation as to why there are so many supermarkets in that area, even more so than the suburb areas where they are supposed to be more households.

Figure 24: Visualisation of Cluster 16



5. Conclusion

Given that COVID-19 is here to stay until a vaccine is developed, the demand for supermarkets will continue to increase, especially since it is one of the few essential activities that people can perform and leave their homes for. For supermarket chains in Edinburgh that wish to open more supermarket outlets, Cluster 16 offers potential of high profitability, especially given the high footfall due to the train station serving that area.