2EVRP

October 5, 2019

1 2-Echelon Vehicle Routing Problem

## Intro

This notebook researches the solution of the 2 echelon VRP in the city of Kyiv for delivery to the post offices of the Nova Poshta company.

All the data is initialized, computed and retrieved from scratch using Google Maps API and other open-source solutions.

At the moment, all the code is reproducible if the `data` folder is present. However, in the case of the need of any modifications, you will need to input your own API key for Google Maps API.

[2]:
```
api_key = "<API_KEY>"
api_key = "AIzaSyAVHwvxtT7W5-LR7p2BFQHJOa_lUc-K2rQ"
```

## Initial setup

Google Maps API only returns up to 60 results, so in order to obtain a minimum of 100 points, we would have to define two points within the city and input the same search query in an attempt to obtain the desired result.

Instead, let's define 4 points -- the borders of the researched rectangular area.

[3]:
```
bl = (50.381347, 30.442942)
tl = (50.533611, 30.442942)
tr = (50.533611, 30.65)
br = (50.381347, 30.65)

borders = {"bl": bl,
           "tl": tl,
           "tr": tr,
           "br": br}
```

The distances between the two diagonals (`bl` <-> `tr` & `br` <-> `tl`) is approximately 22 km.

In order to get the most results, we need to specify circles halfway to the center with the diameter of half the diagonal. This will result in some overlapping, but the whole area will be covered.

Moreover, we will drop the results with coordinates above or below the defined rectangle to ensure we only consider customers within the rectangle.

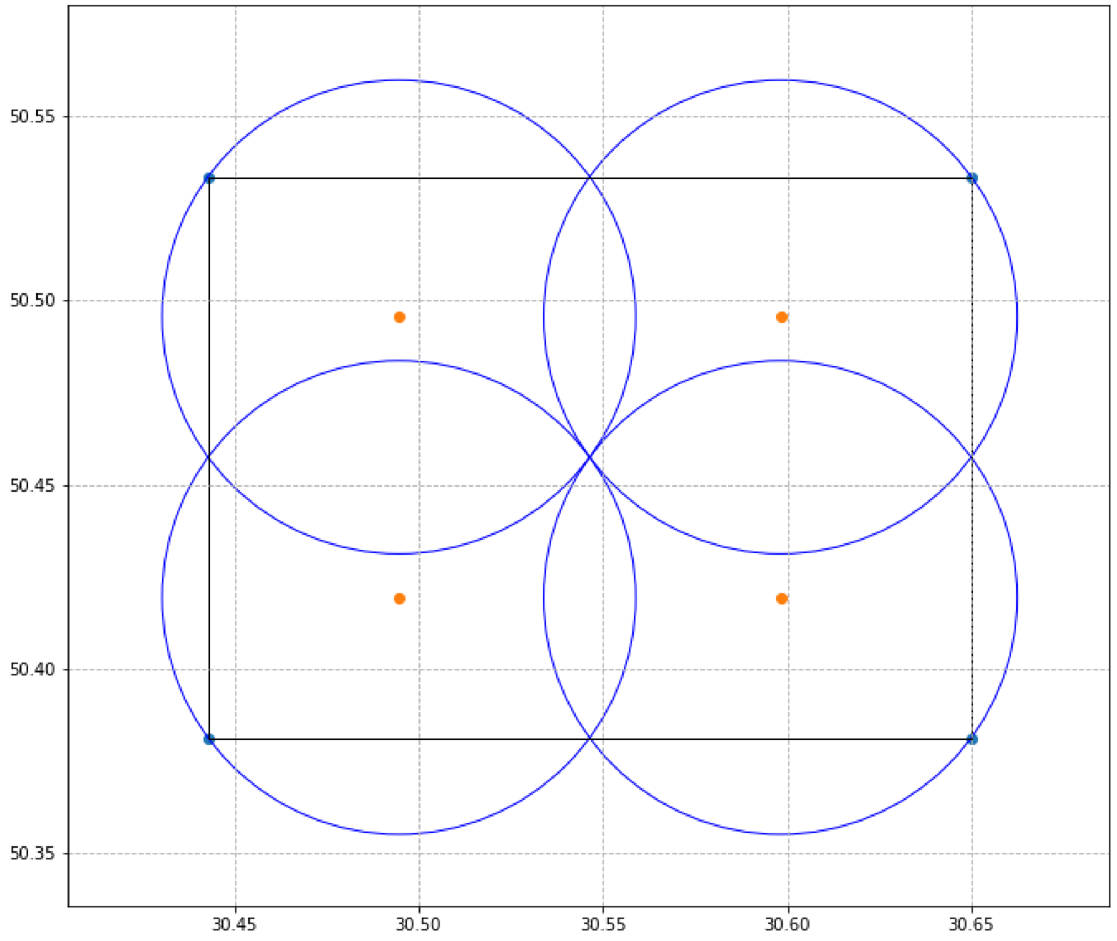
[4]:
```
import numpy as np
```

[5]:
```
c1 = (bl[0] + (tr[0] - bl[0]) / 4, bl[1] + (tr[1] - bl[1]) / 4)
c2 = (bl[0] + (tr[0] - bl[0]) / 4 * 3, bl[1] + (tr[1] - bl[1]) / 4 * 3)
c3 = (tl[0] + (br[0] - tl[0]) / 4, br[1] + (tl[1] - br[1]) / 4 * 3)
c4 = (tl[0] + (br[0] - tl[0]) / 4 * 3, br[1] + (tl[1] - br[1]) / 4)

r1 = np.linalg.norm(np.array(bl) - np.array(tr)) / 4
r2 = np.linalg.norm(np.array(br) - np.array(tl)) / 4

centers = {"c1": c1,
           "c2": c2,
           "c3": c3,
           "c4": c4}
```

[6]:
```
import matplotlib.pyplot as plt
```

[7]:
```
plt.figure(figsize=(20,10))
ax = plt.gca()
ax.cla()
plt.grid(linestyle='--')
plt.xlim(centers['c1'][1] - r1*1.4, centers['c1'][1] + r1*3)
plt.ylim(centers['c1'][0] - r1*1.3, centers['c1'][0] + r1*2.5)
circle1 = plt.Circle((centers['c1'][1], centers['c1'][0]), r1, color = 'blue',␣
 ↪fill = False)
circle2 = plt.Circle((centers['c2'][1], centers['c2'][0]), r1, color = 'blue',␣
 ↪fill = False)
```

```
circle3 = plt.Circle((centers['c3'][1], centers['c3'][0]), r2, color = 'blue',␣
 ↪fill = False)
circle4 = plt.Circle((centers['c4'][1], centers['c4'][0]), r2, color = 'blue',␣
 ↪fill = False)
rectangle = plt.Rectangle((bl[1], bl[0]), width = tr[1] - tl[1], height = tr[0]␣
 ↪- br[0], fill = False, color = "black")
ax.set_aspect(1)
ax.scatter(*zip(*[(x, y) for (y, x) in borders.values()]))
ax.scatter(*zip(*[(x, y) for (y, x) in centers.values()]))

ax.add_artist(circle1)
ax.add_artist(circle2)
ax.add_artist(circle3)
ax.add_artist(circle4)
ax.add_artist(rectangle)
plt.show()
```



[8]:
```python
from math import sin, cos, sqrt, atan2, radians

# approximate radius of earth in km
R = 6373.0

lat1 = radians(bl[0])
lon1 = radians(bl[1])
lat2 = radians(tr[0])
lon2 = radians(tr[1])

dlon = lon2 - lon1
dlat = lat2 - lat1

a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
c = 2 * atan2(sqrt(a), sqrt(1 - a))

distance = R * c

print(f"The diagonals of the rectagle are {distance:.2f} km.")
```

```
The diagonals of the rectagle are 22.40 km.
```

[9]:
```python
radius = distance / 4 * 1e3
print(f"The radius of each circle is {radius:.0f} m.")
```

```
The radius of each circle is 5600 m.
```

## Parsing Google Places API

[10]:
```python
from time import time, sleep
import pandas as pd
import googlemaps
```

[11]:
```python
loc_parsed_df = 'data/parsed_np.csv'
```

[12]:
```python
try:
    df = pd.read_csv(loc_parsed_df)
except:
    gmaps = googlemaps.Client(key = api_key)
    del df
    for center in centers.keys():
        print(centers[center])
        has_next_page = True
        first_page = True
        while has_next_page:
            if first_page:
                response = gmaps.places(query = "nova poshta", location =␣
 ↪centers[center], radius = radius)
```

```
                first_page = False
            else:
                response = gmaps.places(query = "nova poshta", location =␣
↪centers[center], radius = radius, page_token = next_page_token)
            try:
                next_page_token = response["next_page_token"]
                print(next_page_token)
            except:
                has_next_page = False
            try:
                df = pd.concat([df, pd.DataFrame(response['results'])])
            except:
                df = pd.DataFrame(response['results'])
            print(df.shape)
            sleep(5)
    df.drop_duplicates(subset = 'formatted_address', inplace = True)
df.head()
```

[12]:
```
                                     formatted_address  \
0          Saksahanskoho St, 76, Kyiv, Ukraine, 01032
1     ,       -      ,   .        , …
2         Preobrazhenska St, 8 , Kyiv, Ukraine, 03037
3     ."    , Lesi Ukrainky Blvd, 5, Kyiv, Ukra…
4     ,         , 31  ( . "       , K…

                                              geometry  \
0  {'location': {'lat': 50.44027639999999, 'lng':…
1  {'location': {'lat': 50.450083, 'lng': 30.4962…
2  {'location': {'lat': 50.4242448, 'lng': 30.472…
3  {'location': {'lat': 50.4352113, 'lng': 30.530…
4  {'location': {'lat': 50.4372615, 'lng': 30.518…

                                                  icon  \
0  https://maps.gstatic.com/mapfiles/place_api/ic…
1  https://maps.gstatic.com/mapfiles/place_api/ic…
2  https://maps.gstatic.com/mapfiles/place_api/ic…
3  https://maps.gstatic.com/mapfiles/place_api/ic…
4  https://maps.gstatic.com/mapfiles/place_api/ic…

                                         id               name  \
0  1d972a04859d4fab983732eccad4c6dd411a9c7d        Nova Poshta
1  1b10a884a4808f76fd0784247e54989c458a656f        Nova Poshta
2  e3d7c7406b127e42cec679d222035d549118d596   Nova Poshta №240
3  c0e0477b2ad6ba0fea0f41a79352459471502f3e        Nova Poshta
4  a28abba3956efb2bc79f6701bfaaf68776906d16        Nova Poshta

        opening_hours                                             photos  \
0  {'open_now': True}  [{'height': 4160, 'html_attributions': ['<a hr…
1  {'open_now': True}  [{'height': 3456, 'html_attributions': ['<a hr…
2  {'open_now': True}  [{'height': 1080, 'html_attributions': ['<a hr…
3  {'open_now': True}  [{'height': 2448, 'html_attributions': ['<a hr…
4  {'open_now': True}  [{'height': 3264, 'html_attributions': ['<a hr…

                      place_id  \
0  ChIJHw_UA_HO1EARC2d-Q4YGvBg
1  ChIJT1wtIGDO1EARDscyDRzUTPA
2  ChIJTw2tusfO1EARWh7P_e6if-o
3  ChIJk2oxTQHP1EARKlmkVD6n2PU
4  ChIJm9qlhf7O1EARZQ5CXQ2Cgy0

                                          plus_code  rating  \
0  {'compound_code': 'CGR2+4G Kyiv, Kyiv city', '…     3.8
1  {'compound_code': 'FF2W+2F Kyiv, Kyiv city', '…     4.3
2  {'compound_code': 'CFFC+MX Kyiv, Kyiv city', '…     4.2
3  {'compound_code': 'CGPJ+33 Kyiv, Kyiv city', '…     4.0
4  {'compound_code': 'CGP9+WG Kyiv, Kyiv city', '…     4.2

                     reference  \
0  ChIJHw_UA_HO1EARC2d-Q4YGvBg
1  ChIJT1wtIGDO1EARDscyDRzUTPA
2  ChIJTw2tusfO1EARWh7P_e6if-o
3  ChIJk2oxTQHP1EARKlmkVD6n2PU
4  ChIJm9qlhf7O1EARZQ5CXQ2Cgy0

                                              types  user_ratings_total
0  ['post_office', 'finance', 'point_of_interest'…                417
1           ['point_of_interest', 'establishment']                  192
2           ['point_of_interest', 'establishment']                  595
3           ['point_of_interest', 'establishment']                  115
4           ['point_of_interest', 'establishment']                  146
```

[13]:
```
print(f"A total of {len(df['formatted_address'].unique())} post offices were␣
 ↪found within the 4 circles.")
```

```
A total of 135 post offices were found within the 4 circles.
```

[14]:
```
df.reset_index(drop = True, inplace = True)
df.to_csv(loc_parsed_df, header = True, index = False)
```

[15]:
```
import ast

df['lat'] = df['geometry'].apply(lambda x: ast.
 ↪literal_eval(x)['location']['lat'])
```

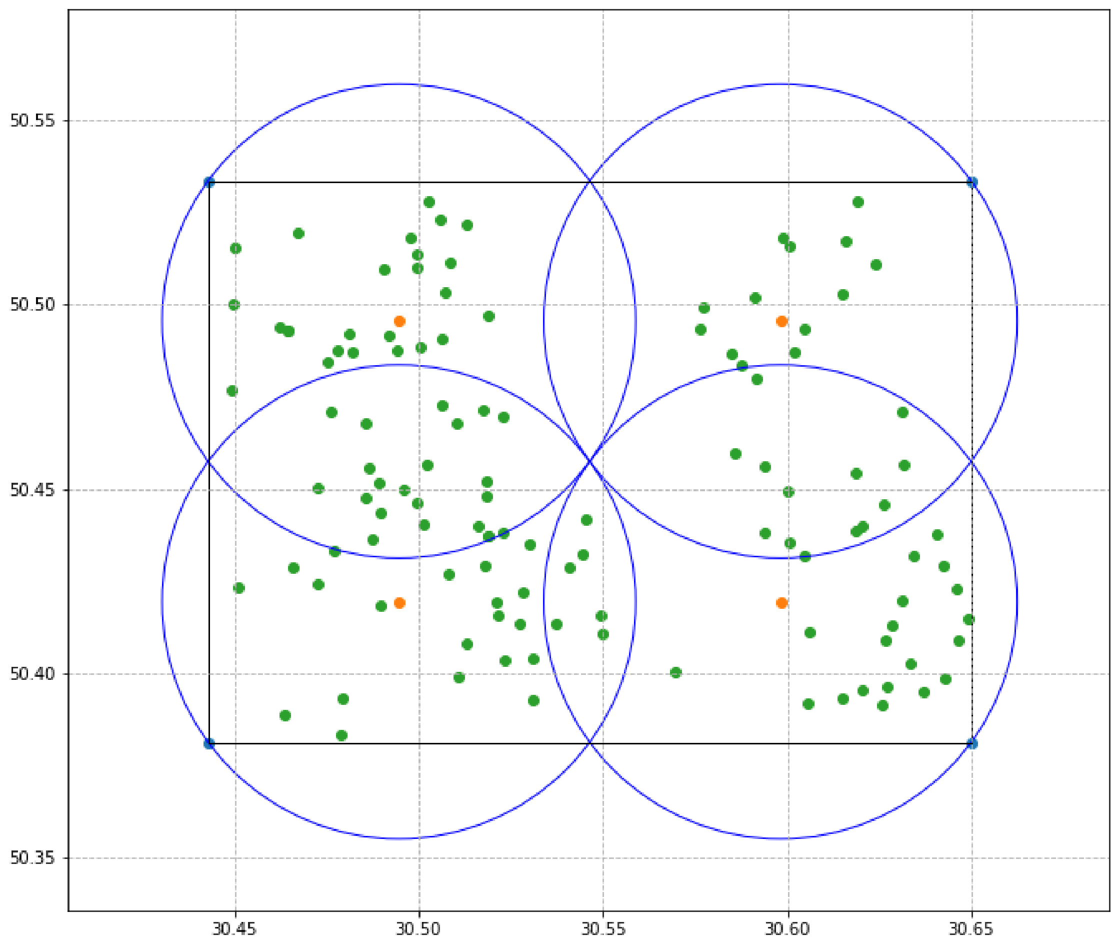
```
df['lon'] = df['geometry'].apply(lambda x: ast.
 ↪literal_eval(x)['location']['lng'])
```

```
[16]: df['in_square'] = ((df['lat'] > bl[0]) & (df['lat'] < tr[0]) & (df['lon'] >␣
 ↪bl[1]) & (df['lon'] < tr[1]))
```

```
[17]: print(f"A total of {len(df['formatted_address'][df['in_square']].unique())}␣
 ↪post offices found are within the considered rectangle.")
```

```
A total of 118 post offices found are within the considered rectangle.
```

```
[18]: plt.figure(figsize=(20,10))
ax = plt.gca()
ax.cla()
plt.grid(linestyle='--')
plt.xlim(centers['c1'][1] - r1*1.4, centers['c1'][1] + r1*3)
plt.ylim(centers['c1'][0] - r1*1.3, centers['c1'][0] + r1*2.5)
circle1 = plt.Circle((centers['c1'][1], centers['c1'][0]), r1, color = 'blue',␣
 ↪fill = False)
circle2 = plt.Circle((centers['c2'][1], centers['c2'][0]), r1, color = 'blue',␣
 ↪fill = False)
circle3 = plt.Circle((centers['c3'][1], centers['c3'][0]), r2, color = 'blue',␣
 ↪fill = False)
circle4 = plt.Circle((centers['c4'][1], centers['c4'][0]), r2, color = 'blue',␣
 ↪fill = False)
rectangle = plt.Rectangle((bl[1], bl[0]), width = tr[1] - tl[1], height = tr[0]␣
 ↪- br[0], fill = False, color = "black")
ax.set_aspect(1)
ax.scatter(*zip(*[(x, y) for (y, x) in borders.values()]))
ax.scatter(*zip(*[(x, y) for (y, x) in centers.values()]))

ax.scatter(df['lon'][df['in_square']], df['lat'][df['in_square']])

ax.add_artist(circle1)
ax.add_artist(circle2)
ax.add_artist(circle3)
ax.add_artist(circle4)
ax.add_artist(rectangle)
plt.show()
```



## Clustering

Let's analyze what amount of clusters is best fit for the given data points.

We will use K-means algorithm, which uses the Euclidean metric for the pairwise distance, thus there will most likely be non-optimal cluster assignments.

Let's perform the Silhouette analysis for cluster numbers from 2 to 6. 1 is redundant and everything above 6 is not viable for our considered task.

[19]:
```
clusters_df = df[['lat', 'lon', 'user_ratings_total',␣
↪'rating']][df['in_square']]
clusters_df['user_ratings_total_scaled'] = clusters_df['user_ratings_total'] /␣
↪clusters_df['user_ratings_total'].max()
clusters_df['weighted_ratings'] = clusters_df['rating'] *␣
↪clusters_df['user_ratings_total'] / (clusters_df['rating'] *␣
↪clusters_df['user_ratings_total']).max()
clusters_df.head()
```

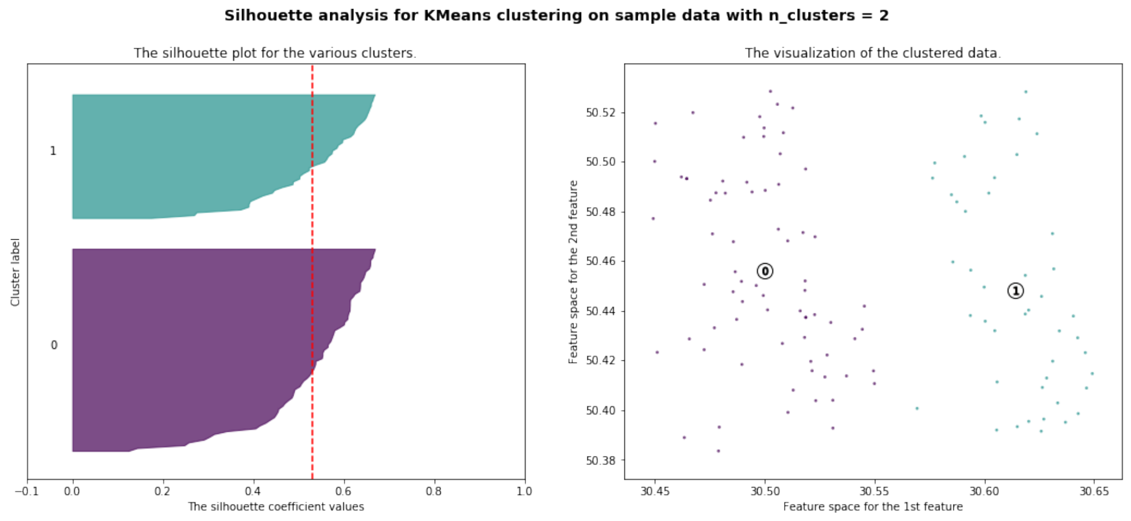
[19]:
```
         lat        lon  user_ratings_total  rating  \
0  50.440276  30.501372                 417     3.8
1  50.450083  30.496220                 192     4.3
2  50.424245  30.472491                 595     4.2
3  50.435211  30.530175                 115     4.0
4  50.437261  30.518789                 146     4.2

   user_ratings_total_scaled  weighted_ratings
0                   0.319540          0.296159
1                   0.147126          0.154303
2                   0.455939          0.467059
3                   0.088123          0.085973
4                   0.111877          0.114606
```

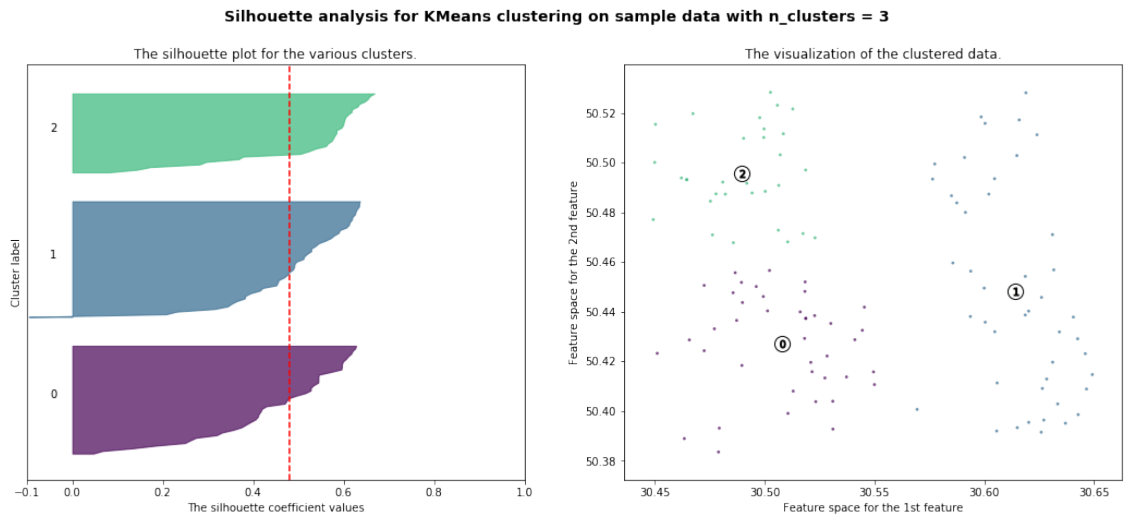
### Silhouette analysis for KMeans (Euclidean distances)

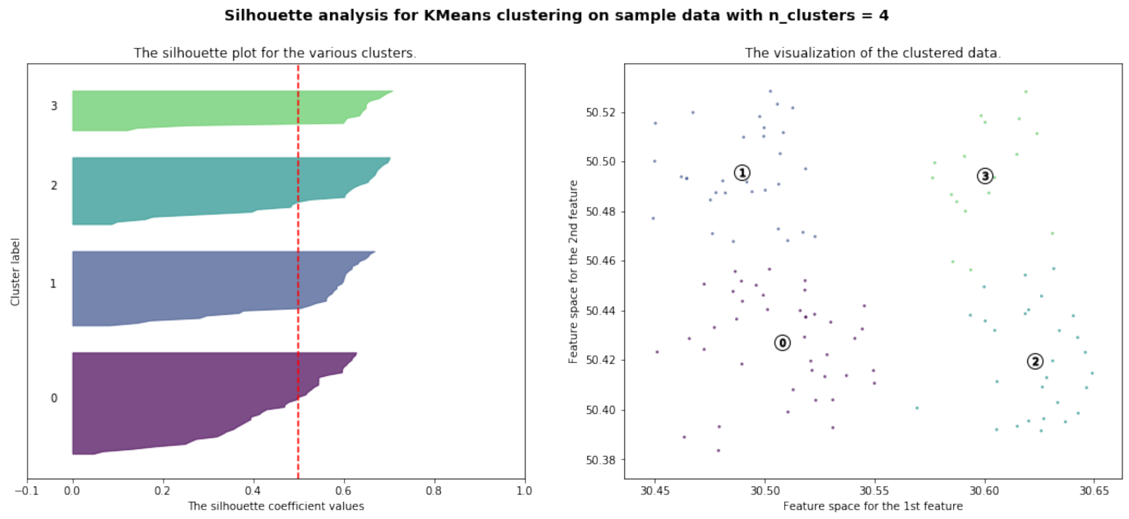
[20]:
```
X = clusters_df[['lat', 'lon']]
```

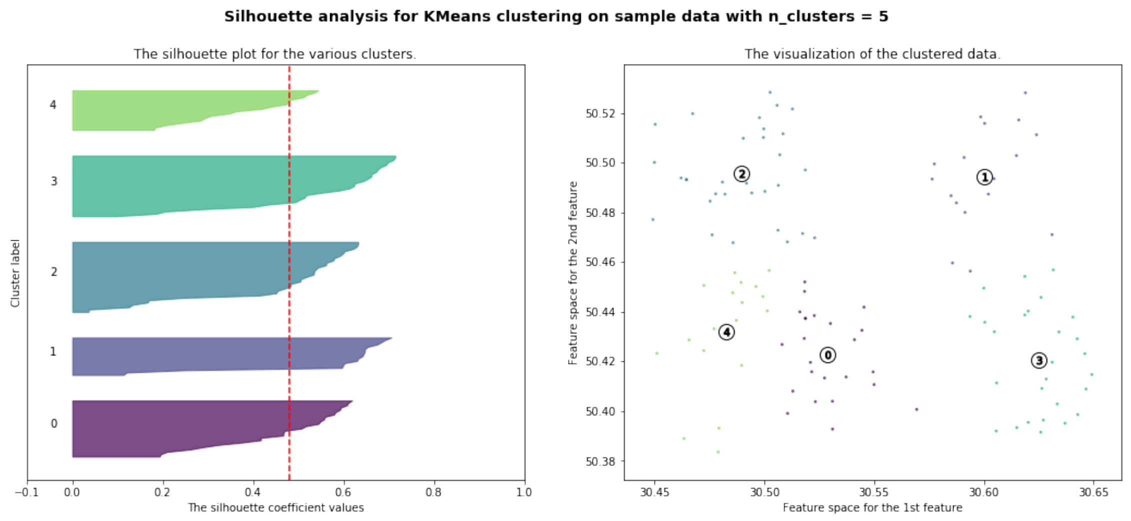
[21]:
```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

# Defining a range of clusters to search in
range_n_clusters = range(2, 7, 1)

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(clusters_df) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.viridis(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.viridis(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(X.iloc[:, 1], X.iloc[:, 0], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')
    # Labeling the clusters
    center_points = clusterer.cluster_centers_
    # Draw white circles at cluster centers
    ax2.scatter(center_points[:, 1], center_points[:, 0], marker='o',
                c="white", alpha=1, s=200, edgecolor='k')

    for i, c in enumerate(center_points):
        ax2.scatter(c[1], c[0], marker='$%d$' % i, alpha=1,
                    s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for KMeans clustering on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

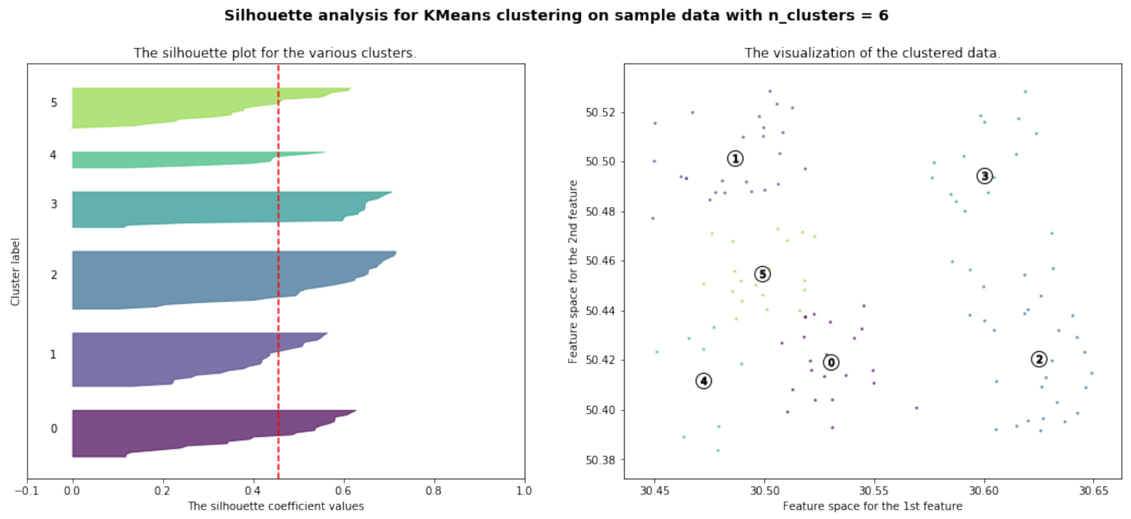
```
For n_clusters = 2 The average silhouette_score is : 0.5313525718118249
For n_clusters = 3 The average silhouette_score is : 0.47991360883988454
For n_clusters = 4 The average silhouette_score is : 0.4984763926878794
For n_clusters = 5 The average silhouette_score is : 0.47953602643476456
For n_clusters = 6 The average silhouette_score is : 0.45717502982804253
```




Silhouette analysis for KMeans clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
The visualization of the clustered data.
Cluster label
The silhouette coefficient values
Feature space for the 2nd feature
Feature space for the 1st feature
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
30.45
30.50
30.55
30.60
30.65
50.38
50.40
50.42
50.44
50.46
50.48
50.50
50.52
0
1
2


Silhouette analysis for KMeans clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
The visualization of the clustered data.
Cluster label
The silhouette coefficient values
Feature space for the 2nd feature
Feature space for the 1st feature
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
30.45
30.50
30.55
30.60
30.65
50.38
50.40
50.42
50.44
50.46
50.48
50.50
50.52
0
1
2
3





As Silhouette analysis demonstrates, 4 is the optimal number of clusters to break down the data points. We will not be using 2 as another oversimplistic version.

### KMeans with 4 clusters

```
[22]: from matplotlib.cm import viridis
      from matplotlib.colors import to_hex
```

```
[23]: def calculate_color(clust):
          """
          Convert cluster number to a color
          """
          # transform the gini coefficient to a matplotlib color
```

```
    mpl_color = viridis(clust / num_clusters)

    # transform from a matplotlib color to a valid CSS color
    gmaps_color = to_hex(mpl_color, keep_alpha = False)

    return gmaps_color
```

[24]:
```
num_clusters = 4
kmeans = KMeans(n_clusters = num_clusters).fit(clusters_df[['lat', 'lon']])
centroids = kmeans.cluster_centers_
print(centroids)
```

```
[[50.41978153 30.62298615]
 [50.49543372 30.48969494]
 [50.42705734 30.50797268]
 [50.4944713  30.60000928]]
```

[25]:
```
clusters_df['cluster'] = kmeans.labels_
clusters_df['color'] = clusters_df['cluster'].apply(calculate_color)
clusters_df.head()
```

[25]:
```
         lat        lon  user_ratings_total  rating  \
0  50.440276  30.501372                 417     3.8
1  50.450083  30.496220                 192     4.3
2  50.424245  30.472491                 595     4.2
3  50.435211  30.530175                 115     4.0
4  50.437261  30.518789                 146     4.2

   user_ratings_total_scaled  weighted_ratings  cluster    color
0                   0.319540          0.296159        2  #21918c
1                   0.147126          0.154303        2  #21918c
2                   0.455939          0.467059        2  #21918c
3                   0.088123          0.085973        2  #21918c
4                   0.111877          0.114606        2  #21918c
```

### Plot of KMeans clustering on Google Maps

[26]:
```
import gmaps
```

[27]:
```
gmaps.configure(api_key = api_key)
```

[28]:
```
lat = bl[0] + (tr[0] - bl[0]) / 2
lon = bl[1] + (tr[1] - bl[1]) / 2
print(f"Plotting the map centered at ({lat:.4f}, {lon:.4f}).")
```

```
Plotting the map centered at (50.4575, 30.5465).
```

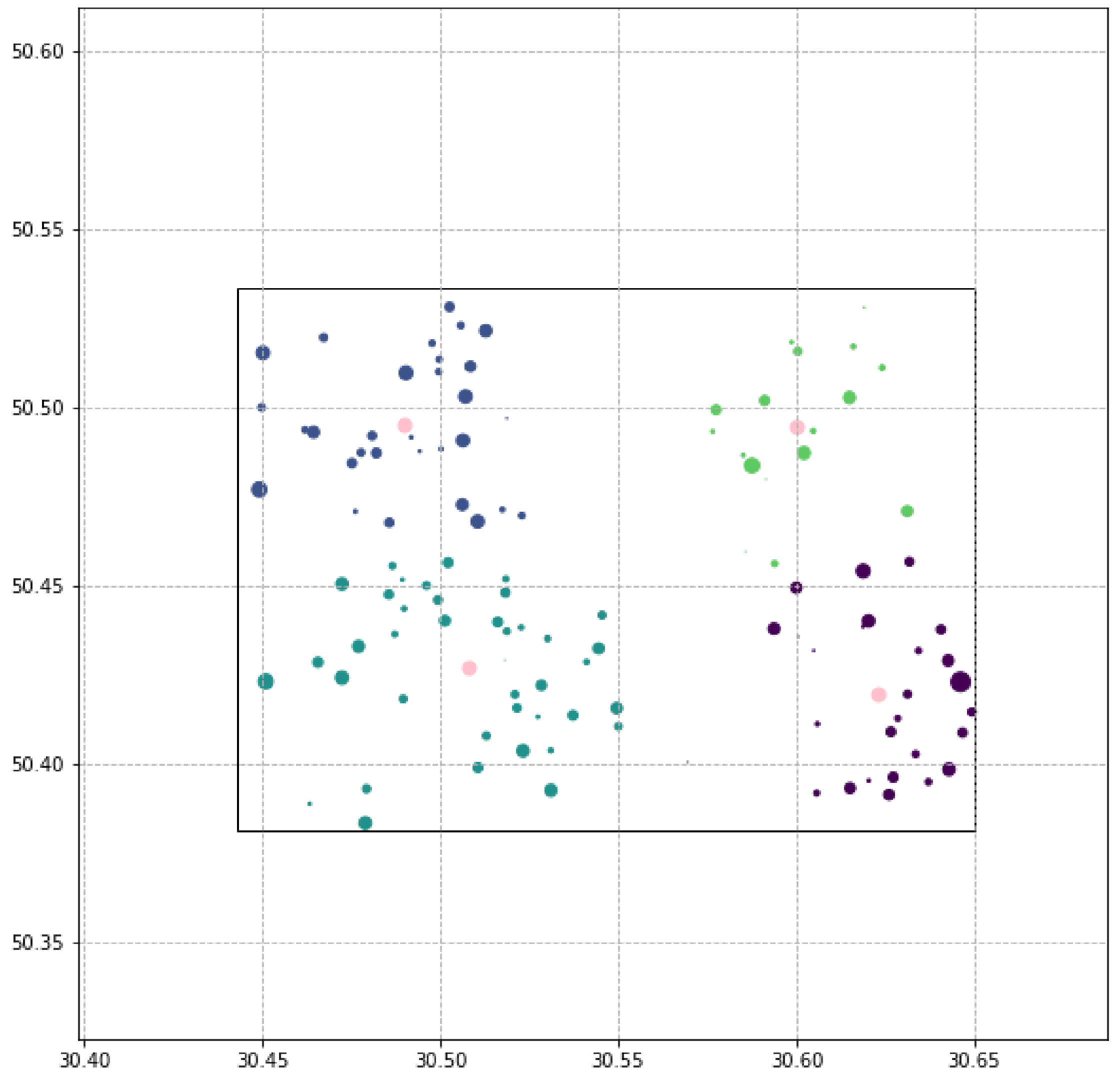
[29]:
```
figure_layout = {
    'width': '800px',
    'height': '800px',
    'border': '1px solid black',
    'padding': '1px'
}
zoom = 12
```

[30]:
```
fig = gmaps.figure(center = (lat, lon), zoom_level = zoom, layout=figure_layout)
for clust in clusters_df['cluster'].unique():
    cluster_layer = gmaps.symbol_layer(
        clusters_df[['lat', 'lon']][clusters_df['cluster'] == clust],
        fill_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        stroke_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        scale = 3
    )
    fig.add_layer(cluster_layer)

centers_layer = gmaps.symbol_layer(
        kmeans.cluster_centers_,
        fill_color = "#ff007f",
        stroke_color = "#ff007f",
        scale = 5)
fig.add_layer(centers_layer)

fig
```

```
Figure(layout=FigureLayout(border='1px solid black', height='800px', padding='1px', width='800
```

[31]:
```
print(f"Plotting weighted ratings of the post offices colored in the colors of␣
 ↪one of the {n_clusters} assigned.")
plt.figure(figsize=(20,10))
ax = plt.gca()
ax.cla()
plt.grid(linestyle='--')
plt.ylim(centers['c1'][0] - r1*1.5, centers['c1'][0] + r1*3)
plt.xlim(centers['c1'][1] - r1*1.5, centers['c1'][1] + r1*3)
rectangle = plt.Rectangle((bl[1], bl[0]), width = tr[1] - tl[1], height = tr[0]␣
 ↪- br[0], fill = False, color = "black")
ax.set_aspect(1)
size_series = clusters_df['weighted_ratings'] * 100
ax.scatter(clusters_df['lon'], clusters_df['lat'], s = size_series, c =␣
 ↪clusters_df['color'])
ax.scatter(centroids[:, 1], centroids[:, 0], c='pink', s=50)
ax.add_artist(rectangle)
plt.show()
```

```
Plotting weighted ratings of the post offices colored in the colors of one of
the 6 assigned.
```



## Finding distance matrix

[32]:
```
from tqdm import tqdm_notebook as tqdm
```

### Parsing Google Distance Matrix API

Parsing GDM to obtain driving distances

```
[33]: loc_dist_matrix = "data/distances_driving.npy"
```

```
[34]: try:
          matrix_driving = np.load(loc_dist_matrix)
      except:
          matrix_driving = np.empty((len(clusters_df), len(clusters_df)))
          columns = len(clusters_df)
          gmaps_api = googlemaps.Client(key = api_key)
          for row in tqdm(range(len(clusters_df))):
              for column in tqdm(range(columns)):
                  origins = (clusters_df['lat'].iloc[row], clusters_df['lon'].
       ↪iloc[row])
                  destination = (clusters_df['lat'].iloc[column], clusters_df['lon'].
       ↪iloc[column])
                  distance = gmaps_api.distance_matrix(origins, destination,␣
       ↪mode='driving')["rows"][0]["elements"][0]["distance"]["value"]
                  matrix_driving[row, column] = distance
              sleep(5)
          np.save(loc_dist_matrix, matrix_driving)
```

```
[35]: matrix_driving
```

```
[35]: array([[    0.,  1418.,  3594., …, 12426.,  5902., 13264.],
             [ 2089.,     0.,  4617., …, 13224.,  6975., 14506.],
             [ 4017.,  4679.,     0., …, 13393.,  6868., 14231.],
             …,
             [12197., 11940., 13259., …,     0.,  9991.,  7167.],
             [ 6578.,  7397.,  9117., …,  6652.,     0.,  7491.],
             [13976., 12427., 14584., …,  8400., 11491.,     0.]])
```

Parsing GDM to obtain walking distances

```
[36]: loc_dist_matrix_walking = "data/distances_walking.npy"
```

```
[37]: try:
          matrix_walking = np.load(loc_dist_matrix_walking)
      except:
          matrix_walking = np.empty((len(clusters_df), len(clusters_df)))
          columns = len(clusters_df)
          gmaps_api = googlemaps.Client(key = api_key)
          for row in tqdm(range(len(clusters_df))):
              for column in tqdm(range(columns)):
                  origins = (clusters_df['lat'].iloc[row], clusters_df['lon'].
       ↪iloc[row])
                  destination = (clusters_df['lat'].iloc[column], clusters_df['lon'].
       ↪iloc[column])
```

```
            distance = gmaps_api.distance_matrix(origins, destination,␣
↪mode='walking')["rows"][0]["elements"][0]["distance"]["value"]
            matrix_walking[row, column] = distance
        sleep(5)
    np.save(loc_dist_matrix_walking, matrix_walking)
```

[38]:
```
matrix_walking
```

[38]:
```
array([[    0.,  1418.,  3245., …,  9535.,  4962., 10791.],
       [ 1418.,     0.,  4228., …, 10231.,  6380., 11488.],
       [ 3245.,  4256.,     0., …, 10958.,  6634., 13177.],
       …,
       [ 9699., 10262., 10780., …,     0.,  5343.,  9507.],
       [ 4934.,  6352.,  6607., …,  5343.,     0.,  7467.],
       [10973., 11536., 13195., …,  9507.,  7467.,     0.]])
```

## Clustering. Part 2

Now that we have obtained the true distance matrices for driving and walking instead of Euclidean distance metric, we can now compute a more accurate clustering distribution.

However, `sklearn` does not have KMeans clustering for custom distribution matrices, and thus we ought to seek some other clustering technique.

Let's first take a look at a fairly popular clustering algorithm, OPTICS:

1.0.1 OPTICS

[39]:
```
from sklearn.cluster import OPTICS
```

[40]:
```
X = matrix_walking
```

[41]:
```
clust = OPTICS(min_samples = 10, xi=.05, min_cluster_size=.05)
clust.fit(X)
```

[41]:
```
OPTICS(algorithm='auto', cluster_method='xi', eps=None, leaf_size=30,
       max_eps=inf, metric='minkowski', metric_params=None,
       min_cluster_size=0.05, min_samples=10, n_jobs=None, p=2,
       predecessor_correction=True, xi=0.05)
```

[42]:
```
set(clust.labels_)
```

[42]:
```
{-1, 0, 1, 2, 3, 4}
```

[43]:
```
clusters_df['cluster'] = clust.labels_
clusters_df['color'] = clusters_df['cluster'].apply(calculate_color)
clusters_df.head()
```

```
[43]:          lat        lon  user_ratings_total  rating  \
      0  50.440276  30.501372                 417     3.8
      1  50.450083  30.496220                 192     4.3
      2  50.424245  30.472491                 595     4.2
      3  50.435211  30.530175                 115     4.0
      4  50.437261  30.518789                 146     4.2

         user_ratings_total_scaled  weighted_ratings  cluster    color
      0                   0.319540          0.296159       -1  #440154
      1                   0.147126          0.154303       -1  #440154
      2                   0.455939          0.467059       -1  #440154
      3                   0.088123          0.085973       -1  #440154
      4                   0.111877          0.114606       -1  #440154
```

```
[44]: fig = gmaps.figure(center = (lat, lon), zoom_level = zoom, layout=figure_layout)
      for clust in clusters_df['cluster'].unique():
          cluster_layer = gmaps.symbol_layer(
              clusters_df[['lat', 'lon']][clusters_df['cluster'] == clust],
              fill_color = clusters_df['color'][clusters_df['cluster'] == clust].
       ↪iloc[0],
              stroke_color = clusters_df['color'][clusters_df['cluster'] == clust].
       ↪iloc[0],
              scale = 3
          )
          fig.add_layer(cluster_layer)

      fig
```

```
Figure(layout=FigureLayout(border='1px solid black', height='800px', padding='1px', width='800
```

As observed in the plotted image, the algorithm performs rather poorly on the data points provided. Thus, we continue our quest for a better clustering instrument to obtain clusters. Our next candidate is Agglomerative Clustering.

### Agglomerative Clustering

Unlike OPTICS, Agglomerative Clustering algorithm requires the number of clusters to be provided explicitly. Thus, we resort to Silhouette analysis once again to determine the optimal number of clusters.

Silhouette analysis for walking distances

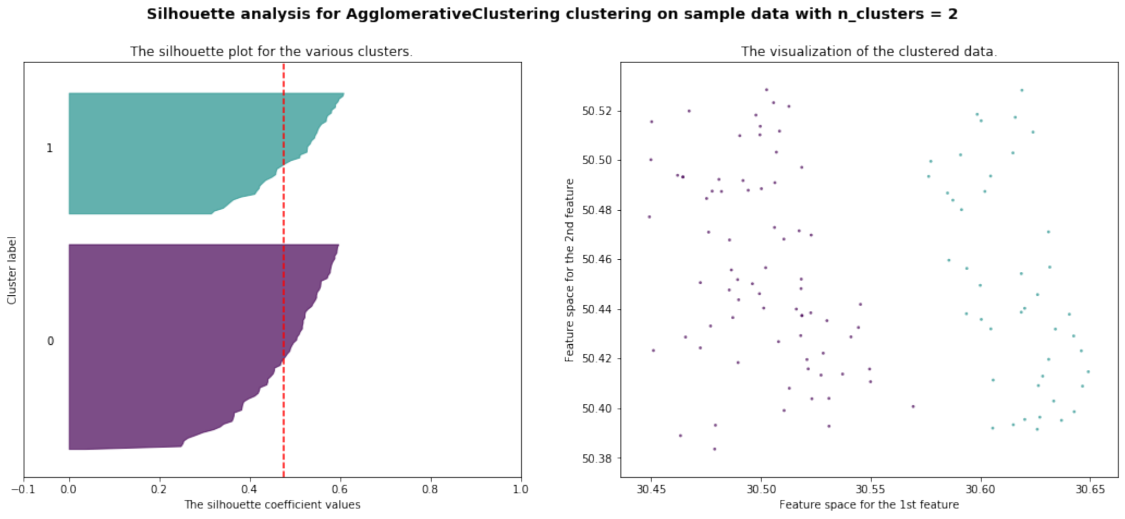
```
[45]: X = matrix_walking
```

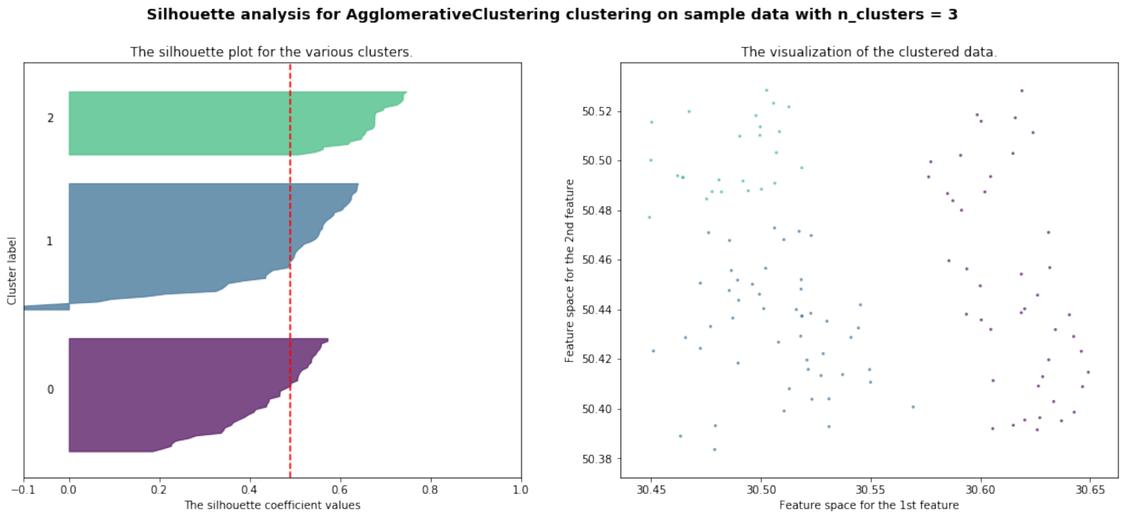
```
[46]: from sklearn.cluster import AgglomerativeClustering
      from sklearn.metrics import silhouette_samples, silhouette_score

      import matplotlib.pyplot as plt
```

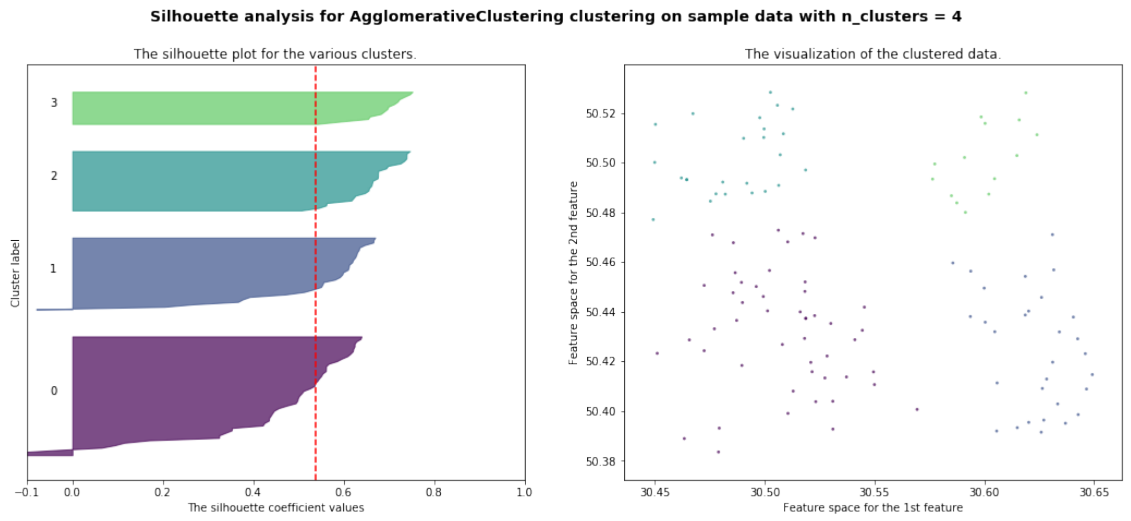
```python
import matplotlib.cm as cm
import numpy as np

# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and 3 clusters placed close
# together.

range_n_clusters = range(2, 11, 1)

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer = AgglomerativeClustering(n_clusters = n_clusters).fit(X)
    cluster_labels = clusterer.fit_predict(X)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(X, cluster_labels)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(X, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.viridis(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.viridis(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(clusters_df[['lat', 'lon']].iloc[:, 1], clusters_df[['lat',␣
↪'lon']].iloc[:, 0], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
#     centers = clusterer.cluster_centers_
#     # Draw white circles at cluster centers
#     ax2.scatter(centers[:, 1], centers[:, 0], marker='o',
#                 c="white", alpha=1, s=200, edgecolor='k')

#     for i, c in enumerate(centers):
#         ax2.scatter(c[1], c[0], marker='$%d$' % i, alpha=1,
#                     s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for AgglomerativeClustering clustering␣
↪on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')
```

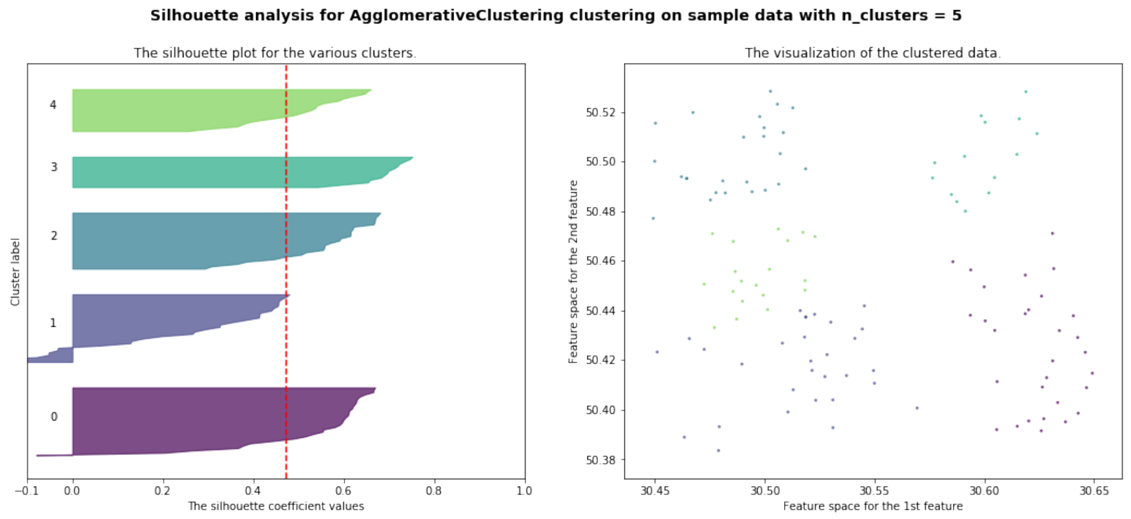
```
plt.show()
```

```
For n_clusters = 2 The average silhouette_score is : 0.47464123112016526
For n_clusters = 3 The average silhouette_score is : 0.4905534458244634
For n_clusters = 4 The average silhouette_score is : 0.5387711559407464
For n_clusters = 5 The average silhouette_score is : 0.47346922377244854
For n_clusters = 6 The average silhouette_score is : 0.4677147946916763
For n_clusters = 7 The average silhouette_score is : 0.4864971895572976
For n_clusters = 8 The average silhouette_score is : 0.45163784828291303
For n_clusters = 9 The average silhouette_score is : 0.4597822151527626
For n_clusters = 10 The average silhouette_score is : 0.44798708064541476
```

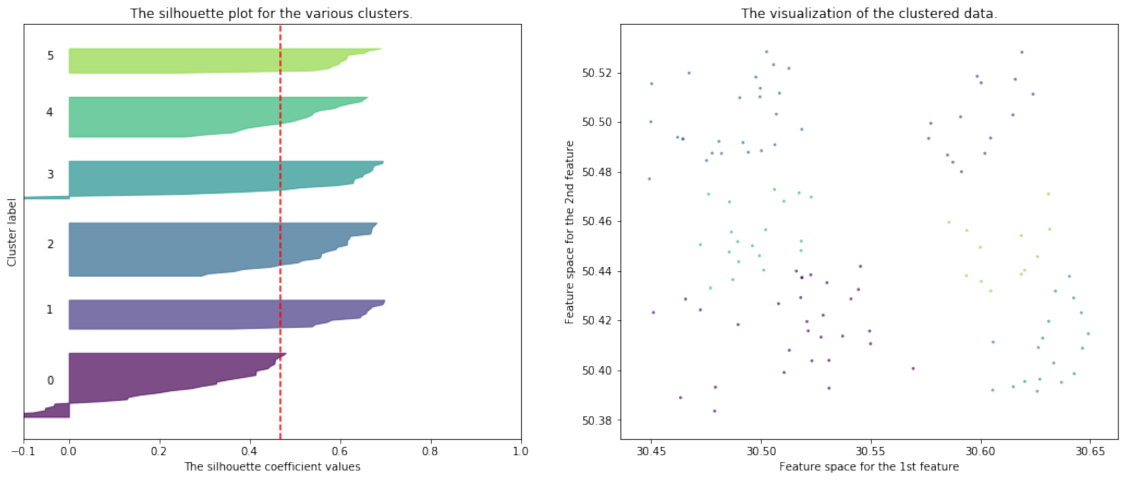




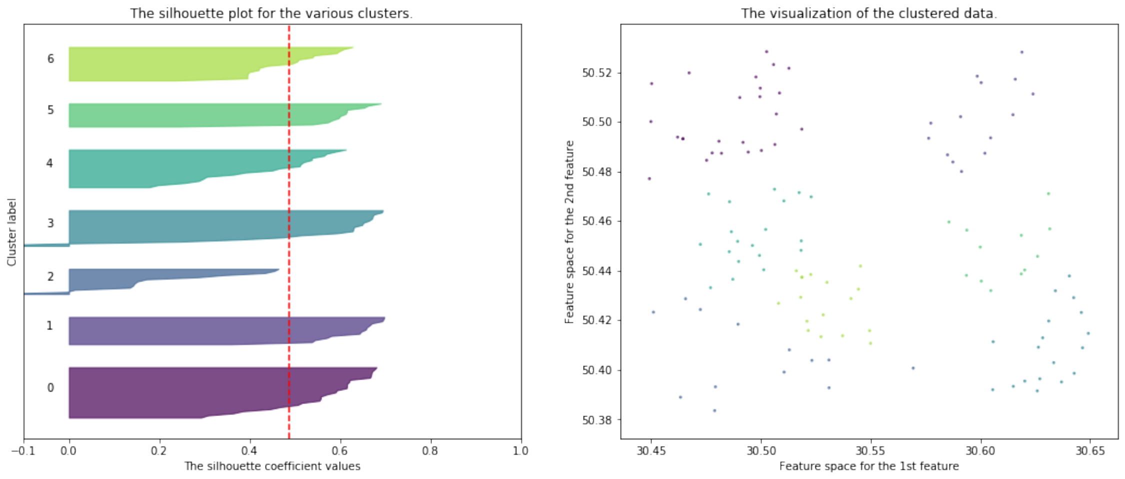

Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 4
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 1st feature
Feature space for the 2nd feature


Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 5
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 1st feature
Feature space for the 2nd feature

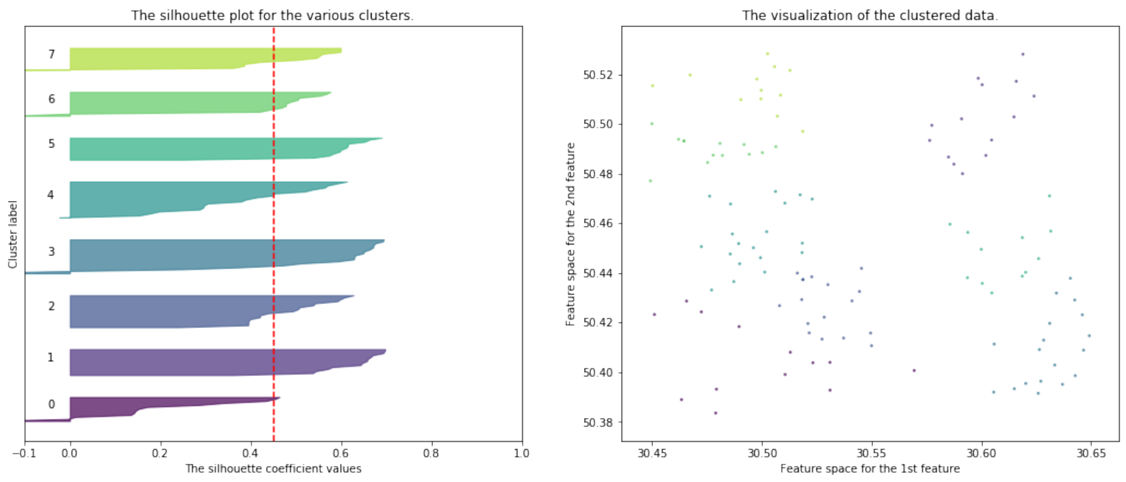
Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 6



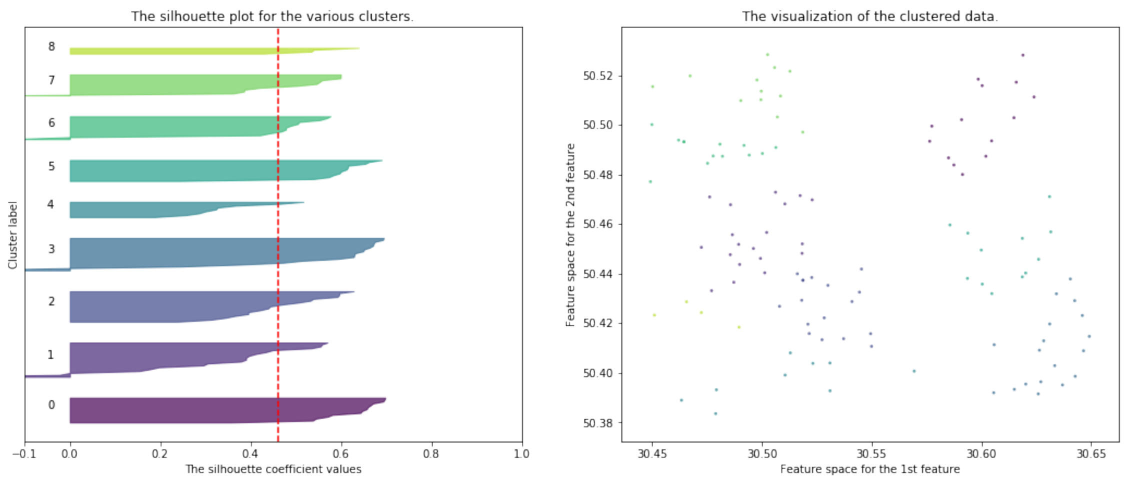
Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 7

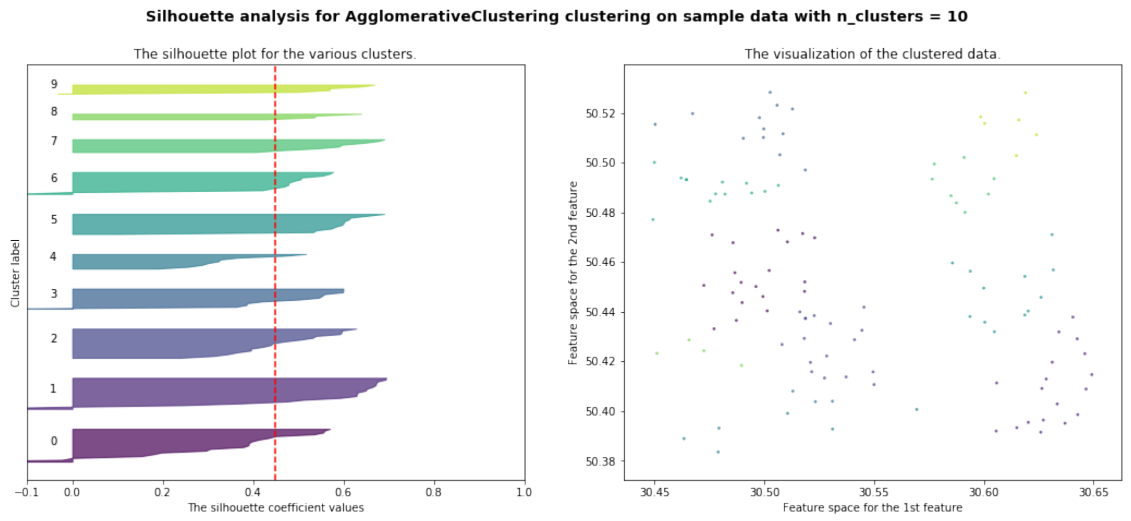


Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 8



Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 9





Silhouette analysis for driving distances

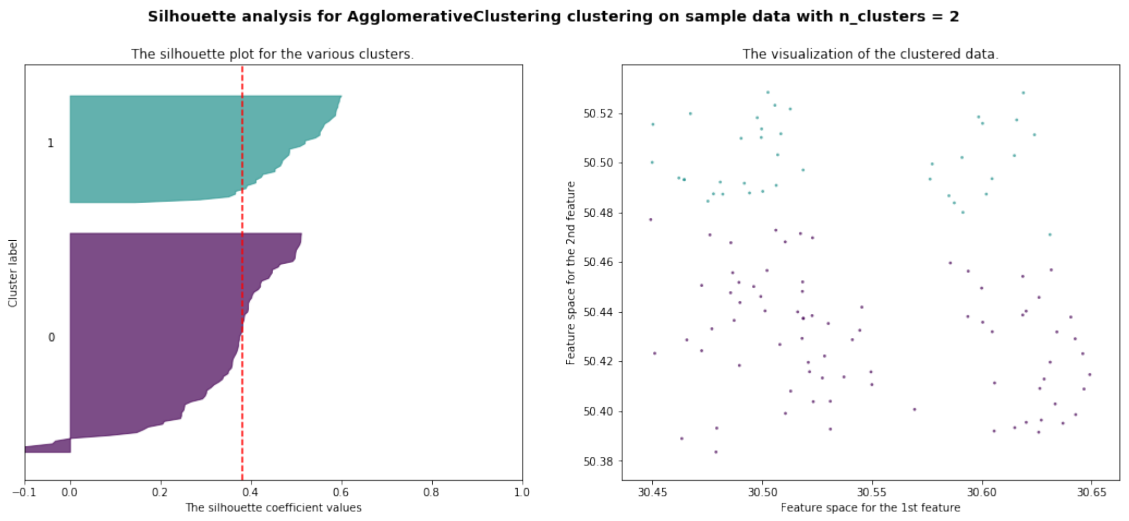
[47]:
```
X = matrix_driving
```

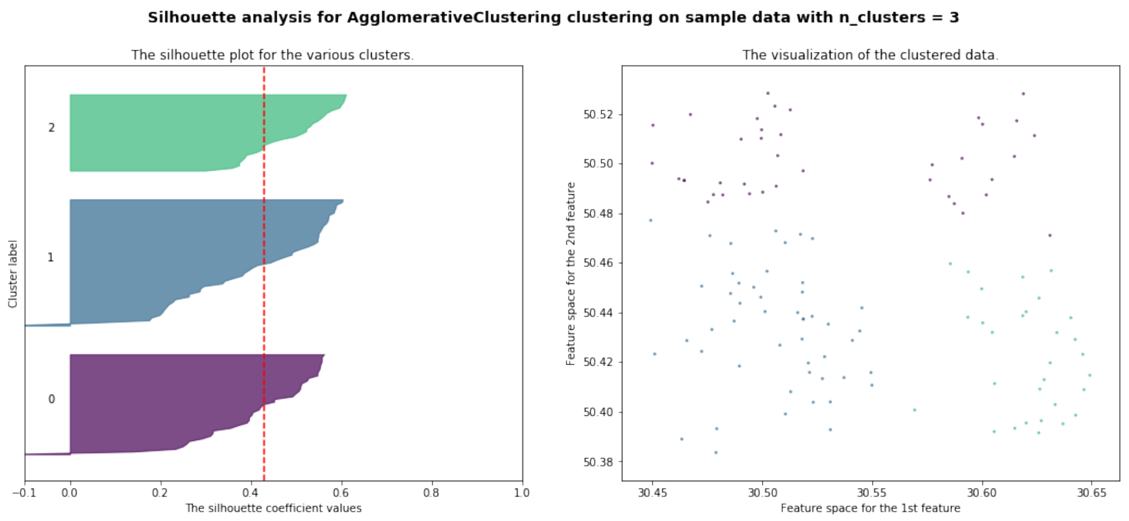
[48]:
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_samples, silhouette_score

import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and 3 clusters placed close
# together.

range_n_clusters = range(2, 11, 1)

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])
```

```python
# Initialize the clusterer with n_clusters value and a random generator
# seed of 10 for reproducibility.
clusterer = AgglomerativeClustering(n_clusters = n_clusters).fit(X)
cluster_labels = clusterer.fit_predict(X)

# The silhouette_score gives the average value for all the samples.
# This gives a perspective into the density and separation of the formed
# clusters
silhouette_avg = silhouette_score(X, cluster_labels)
print("For n_clusters =", n_clusters,
      "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(X, cluster_labels)

y_lower = 10
for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = cm.viridis(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
```

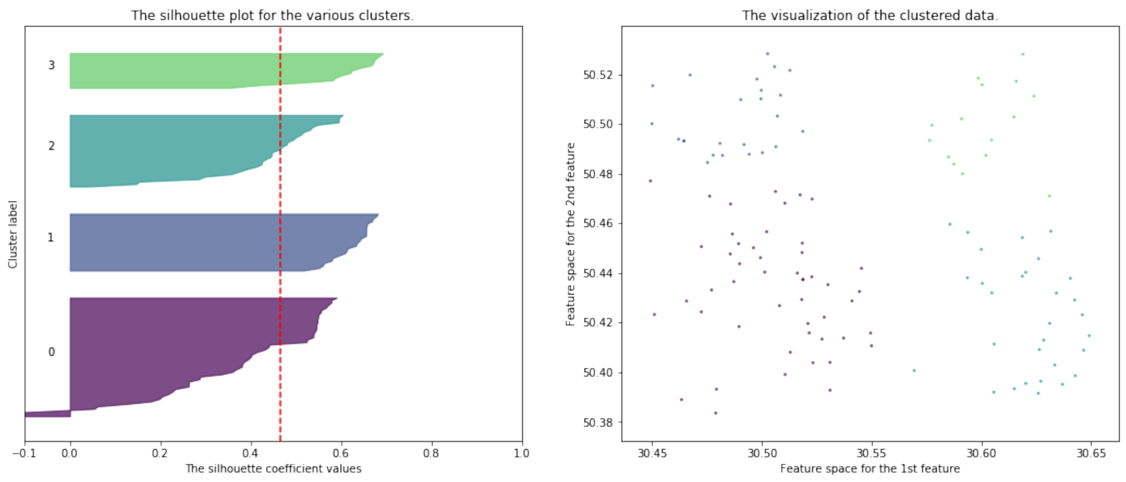
```
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    # 2nd Plot showing the actual clusters formed
    colors = cm.viridis(cluster_labels.astype(float) / n_clusters)
    ax2.scatter(clusters_df[['lat', 'lon']].iloc[:, 1], clusters_df[['lat',␣
↪'lon']].iloc[:, 0], marker='.', s=30, lw=0, alpha=0.7,
                c=colors, edgecolor='k')

    # Labeling the clusters
#     centers = clusterer.cluster_centers_
#     # Draw white circles at cluster centers
#     ax2.scatter(centers[:, 1], centers[:, 0], marker='o',
#                 c="white", alpha=1, s=200, edgecolor='k')

#     for i, c in enumerate(centers):
#         ax2.scatter(c[1], c[0], marker='$%d$' % i, alpha=1,
#                     s=50, edgecolor='k')

    ax2.set_title("The visualization of the clustered data.")
    ax2.set_xlabel("Feature space for the 1st feature")
    ax2.set_ylabel("Feature space for the 2nd feature")

    plt.suptitle(("Silhouette analysis for AgglomerativeClustering clustering␣
↪on sample data "
                  "with n_clusters = %d" % n_clusters),
                 fontsize=14, fontweight='bold')

plt.show()
```

```
For n_clusters = 2 The average silhouette_score is : 0.3821315755233003
For n_clusters = 3 The average silhouette_score is : 0.4294077237548883
For n_clusters = 4 The average silhouette_score is : 0.4646955777488318
For n_clusters = 5 The average silhouette_score is : 0.4316577616249155
For n_clusters = 6 The average silhouette_score is : 0.4511081139176247
For n_clusters = 7 The average silhouette_score is : 0.4239611414912252
For n_clusters = 8 The average silhouette_score is : 0.41383346250153585
For n_clusters = 9 The average silhouette_score is : 0.4091470620804474
For n_clusters = 10 The average silhouette_score is : 0.38438344857638174
```

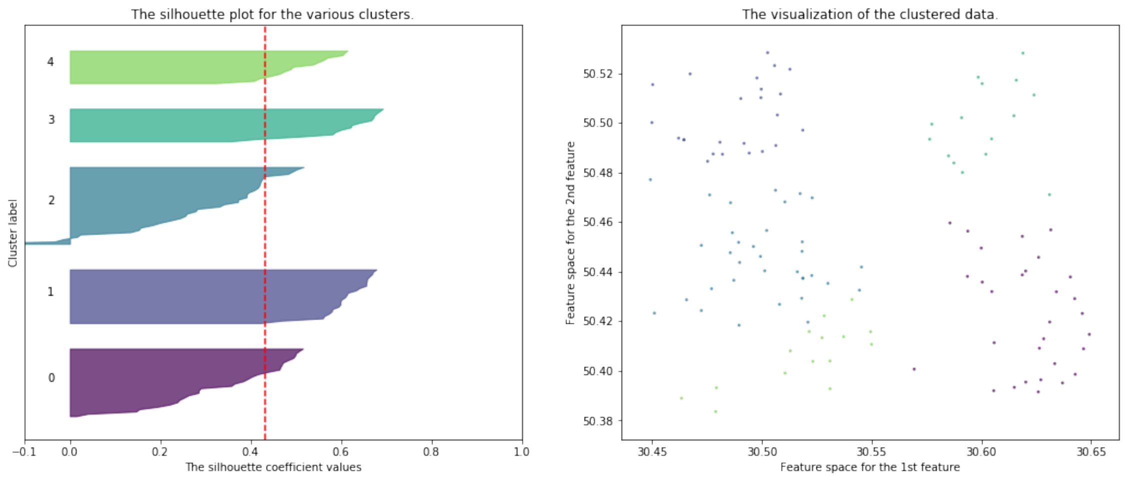

Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 2
The silhouette plot for the various clusters.
Cluster label
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
50.52
50.50
50.48
50.46
50.44
50.42
50.40
50.38
30.45
30.50
30.55
30.60
30.65
Feature space for the 1st feature

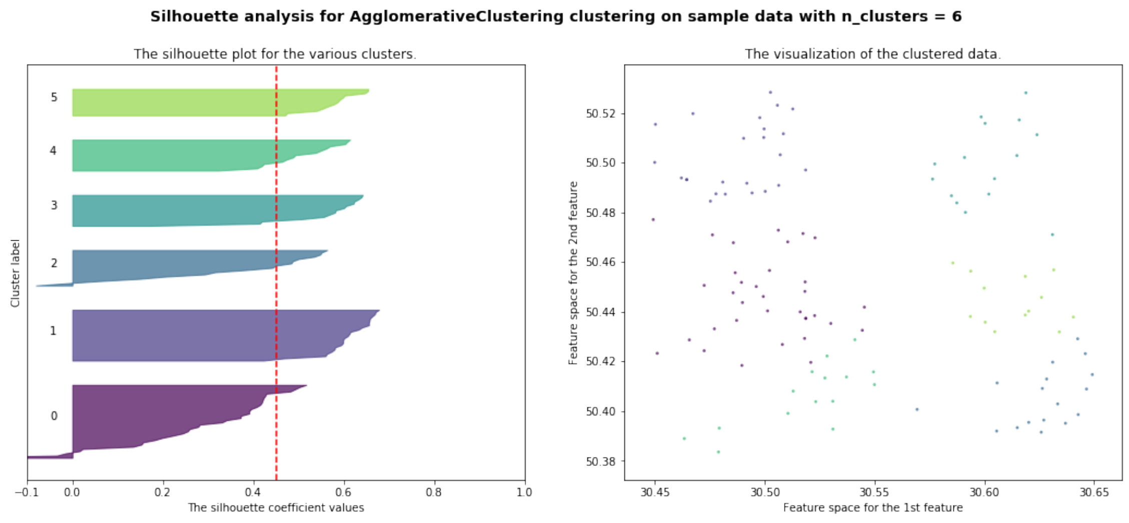

Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 3
The silhouette plot for the various clusters.
Cluster label
2
1
0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
50.52
50.50
50.48
50.46
50.44
50.42
50.40
50.38
30.45
30.50
30.55
30.60
30.65
Feature space for the 1st feature

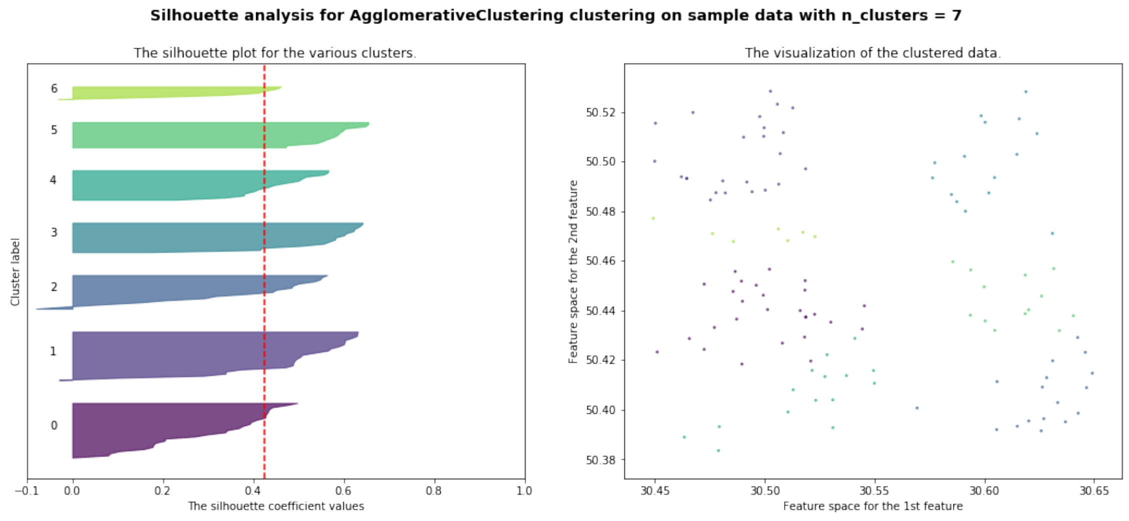
Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 4



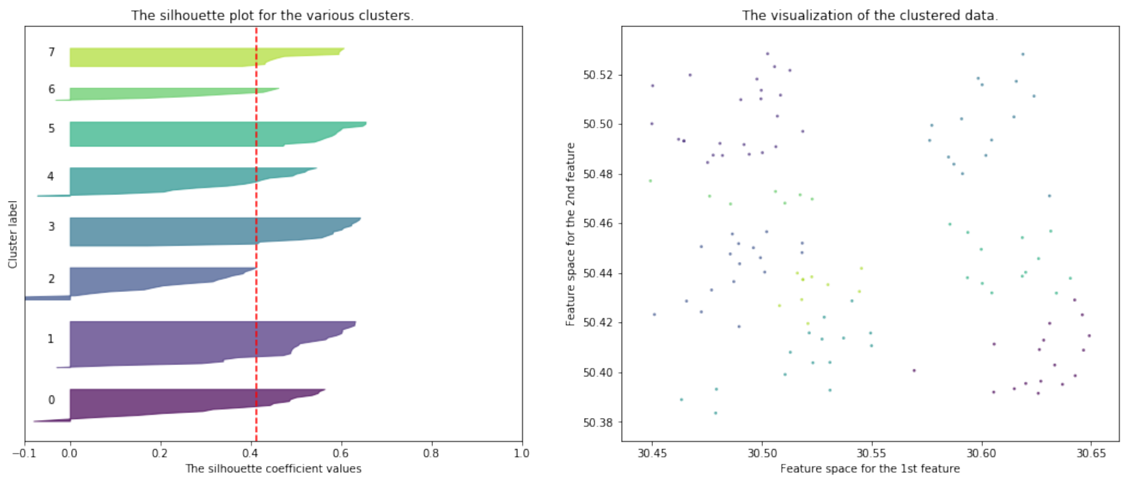
Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 5



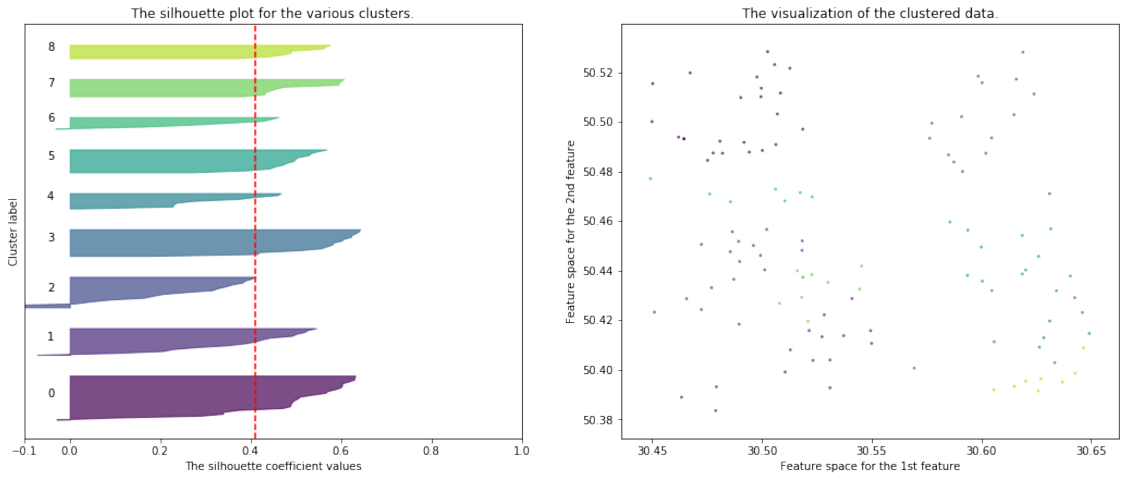

Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 6
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

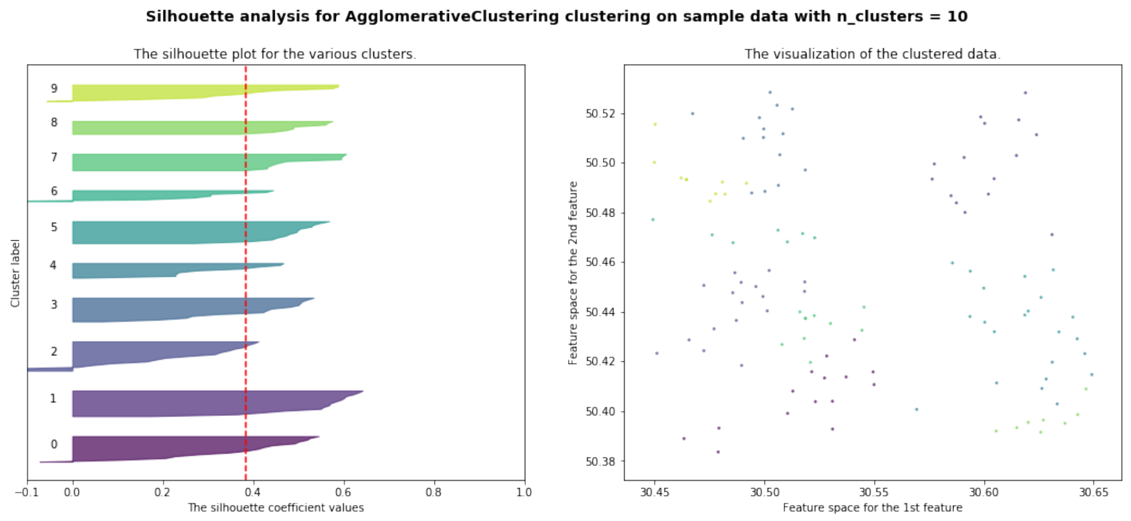

Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 7
The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
The visualization of the clustered data.
Feature space for the 2nd feature
Feature space for the 1st feature

Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 8



Silhouette analysis for AgglomerativeClustering clustering on sample data with n_clusters = 9





The Silhouette analysis of Agglomerative Clustering for both driving and walking distances hints us that the optimal number of clusters is 4.

[49]:
```
n_clusters = 4
```

### Agglomerative Clustering for walking distances

[50]:
```
X = matrix_walking
```

[51]:
```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters = n_clusters).fit(X)
clustering
```

[51]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=4,
                        pooling_func='deprecated')
```

[52]:
```
clusters_df['cluster'] = clustering.labels_
clusters_df['color'] = clusters_df['cluster'].apply(calculate_color)
clusters_df.head()
```

[52]:
```
         lat        lon  user_ratings_total  rating  \
0  50.440276  30.501372                 417     3.8
1  50.450083  30.496220                 192     4.3
2  50.424245  30.472491                 595     4.2
3  50.435211  30.530175                 115     4.0
4  50.437261  30.518789                 146     4.2

   user_ratings_total_scaled  weighted_ratings  cluster    color
```

```
0                    0.319540          0.296159        0  #440154
1                    0.147126          0.154303        0  #440154
2                    0.455939          0.467059        0  #440154
3                    0.088123          0.085973        0  #440154
4                    0.111877          0.114606        0  #440154
```

[53]:
```
fig = gmaps.figure(center = (lat, lon), zoom_level = zoom, layout=figure_layout)
for clust in clusters_df['cluster'].unique():
    cluster_layer = gmaps.symbol_layer(
        clusters_df[['lat', 'lon']][clusters_df['cluster'] == clust],
        fill_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        stroke_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        scale = 3
    )
    fig.add_layer(cluster_layer)

# centers_layer = gmaps.symbol_layer(
#         kmeans.cluster_centers_,
#         fill_color = "#ff007f",
#         stroke_color = "#ff007f",
#         scale = 5)
# fig.add_layer(centers_layer)

fig
```

```
Figure(layout=FigureLayout(border='1px solid black', height='800px', padding='1px', width='800
```

### Agglomerative Clustering for driving distances

[54]:
```
X = matrix_driving
```

[55]:
```
from sklearn.cluster import AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters = n_clusters).fit(X)
clustering
```

[55]:
```
AgglomerativeClustering(affinity='euclidean', compute_full_tree='auto',
                        connectivity=None, distance_threshold=None,
                        linkage='ward', memory=None, n_clusters=4,
                        pooling_func='deprecated')
```

[56]:
```
clusters_df['cluster'] = clustering.labels_
clusters_df['color'] = clusters_df['cluster'].apply(calculate_color)
clusters_df.head()
```

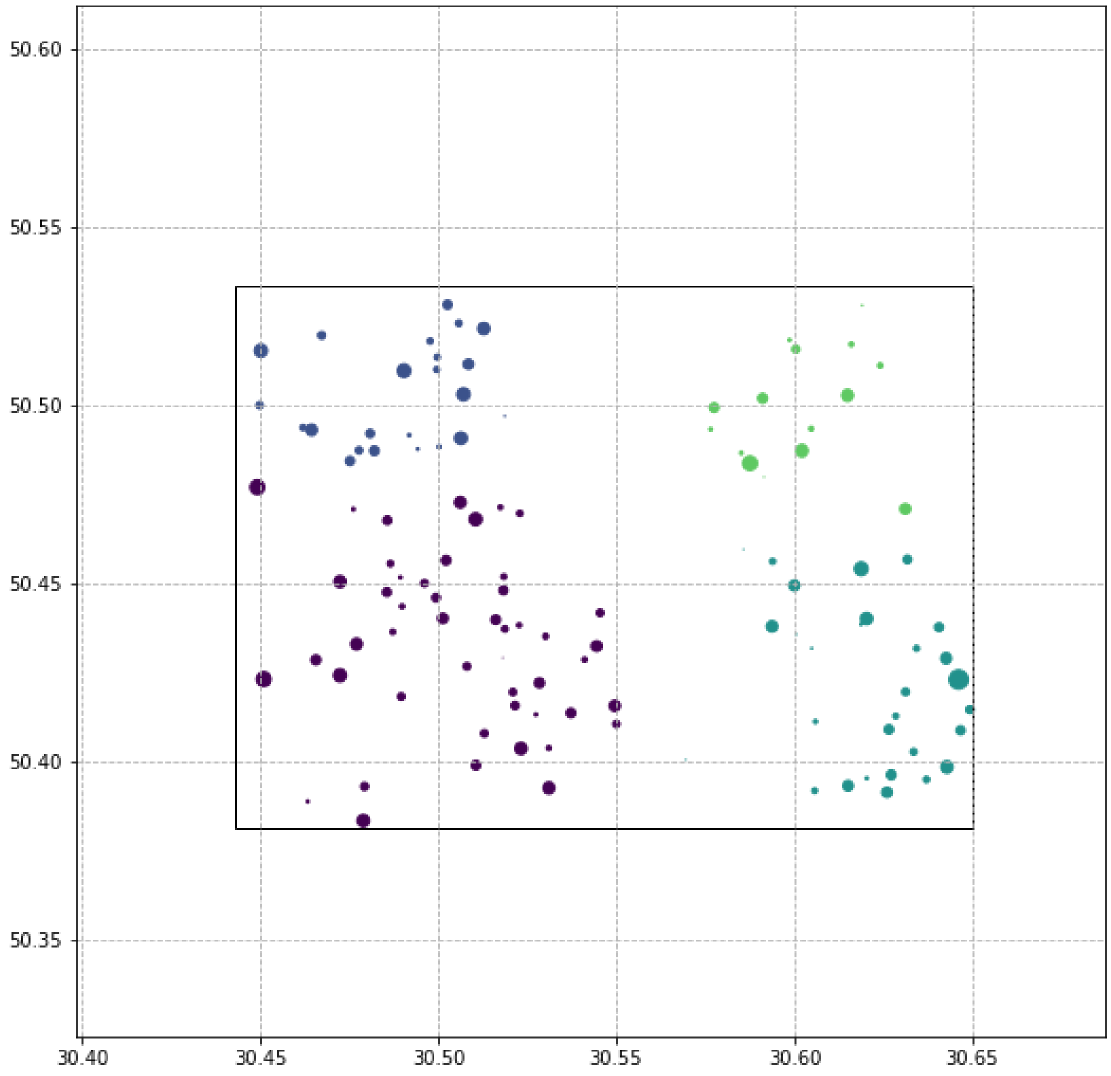
[56]:
```
         lat        lon  user_ratings_total  rating  \
0  50.440276  30.501372                 417     3.8
1  50.450083  30.496220                 192     4.3
2  50.424245  30.472491                 595     4.2
3  50.435211  30.530175                 115     4.0
4  50.437261  30.518789                 146     4.2

   user_ratings_total_scaled  weighted_ratings  cluster    color
0                   0.319540          0.296159        0  #440154
1                   0.147126          0.154303        0  #440154
2                   0.455939          0.467059        0  #440154
3                   0.088123          0.085973        0  #440154
4                   0.111877          0.114606        0  #440154
```

[57]:
```python
fig = gmaps.figure(center = (lat, lon), zoom_level = zoom, layout=figure_layout)
for clust in clusters_df['cluster'].unique():
    cluster_layer = gmaps.symbol_layer(
        clusters_df[['lat', 'lon']][clusters_df['cluster'] == clust],
        fill_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        stroke_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        scale = 3
    )
    fig.add_layer(cluster_layer)

# centers_layer = gmaps.symbol_layer(
#         kmeans.cluster_centers_,
#         fill_color = "#ff007f",
#         stroke_color = "#ff007f",
#         scale = 5)
# fig.add_layer(centers_layer)

fig
```

```
Figure(layout=FigureLayout(border='1px solid black', height='800px', padding='1px', width='800p
```

[58]:
```python
plt.figure(figsize=(20,10))
ax = plt.gca()
ax.cla()
plt.grid(linestyle='--')
plt.ylim(centers['c1'][0] - r1*1.5, centers['c1'][0] + r1*3)
plt.xlim(centers['c1'][1] - r1*1.5, centers['c1'][1] + r1*3)
rectangle = plt.Rectangle((bl[1], bl[0]), width = tr[1] - tl[1], height = tr[0]␣
 ↪- br[0], fill = False, color = "black")
ax.set_aspect(1)
```

```
size_series = clusters_df['weighted_ratings'] * 100
ax.scatter(clusters_df['lon'], clusters_df['lat'], s = size_series, c =␣
 ↪clusters_df['color'])
# ax.scatter(centroids[:, 1], centroids[:, 0], c='pink', s=50)
ax.add_artist(rectangle)
plt.show()
```



## Estimating demand of consumers

Since we do not have any meaningful way of estimating the demand of post office consumers directly, we will try to obtain it via a relatively relatable metric.

We will use the number of ratings as a relative metric of the demand scale. For instance, an office with 50 reviews will have 50 points of demand.

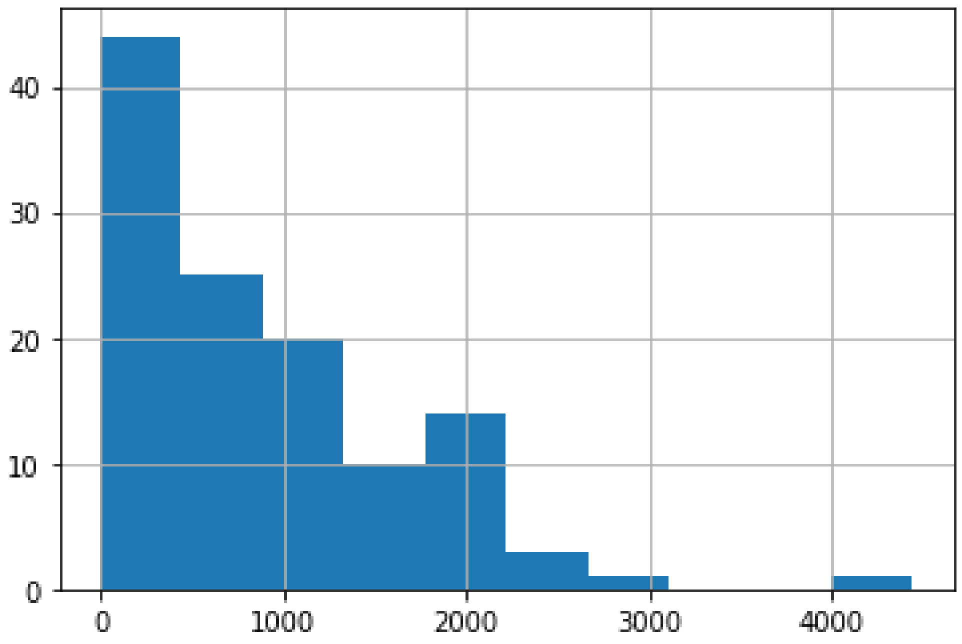
We then assume that 80% of the users is users who frequent the considered post office at least once a week and the other 20% use the post office's services at least once a month. So for an office with 50 points of demand will be considered to have `0.8 * 50 * 4 (weeks) = 160 + 0.2 * 50 = 170` visitors.

```
[59]: clusters_df['demand'] = clusters_df['user_ratings_total'].apply(lambda x: x *␣
       ↪(0.8 * 4 + 0.2))
      clusters_df.head()
```

```
[59]:          lat        lon  user_ratings_total  rating  \
      0  50.440276  30.501372                 417     3.8
      1  50.450083  30.496220                 192     4.3
      2  50.424245  30.472491                 595     4.2
      3  50.435211  30.530175                 115     4.0
      4  50.437261  30.518789                 146     4.2

         user_ratings_total_scaled  weighted_ratings  cluster    color  demand
      0                   0.319540          0.296159        0  #440154  1417.8
      1                   0.147126          0.154303        0  #440154   652.8
      2                   0.455939          0.467059        0  #440154  2023.0
      3                   0.088123          0.085973        0  #440154   391.0
      4                   0.111877          0.114606        0  #440154   496.4
```

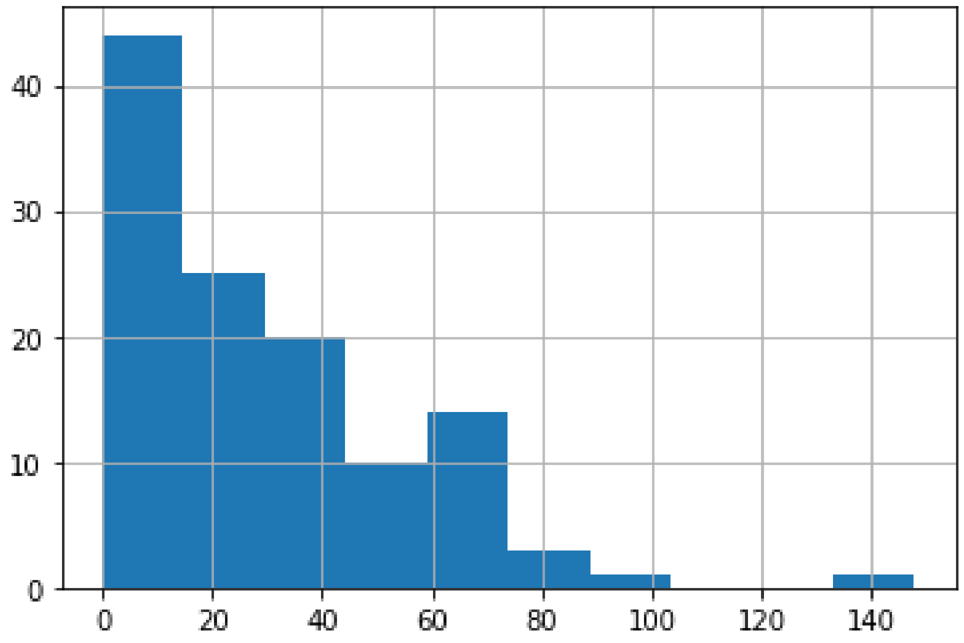
```
[60]: clusters_df['demand'].hist()
```

```
[60]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7fefead518>
```

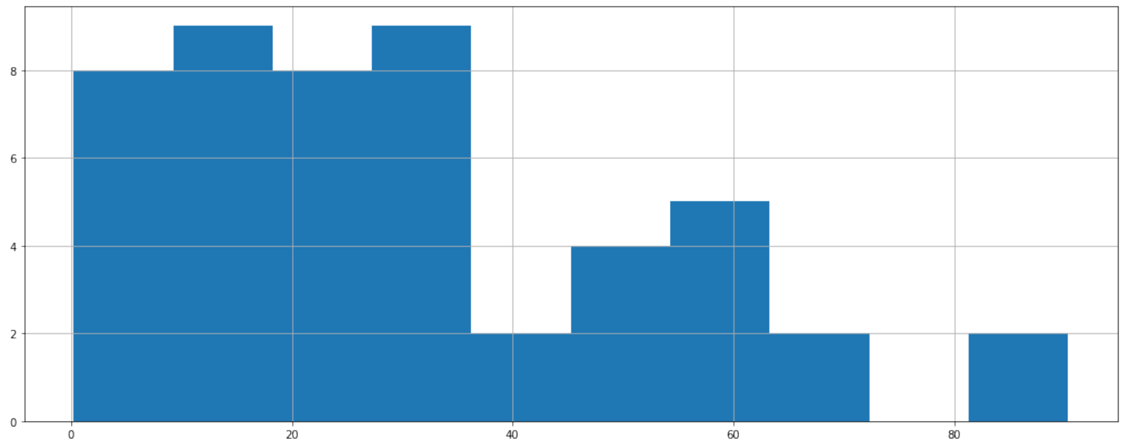


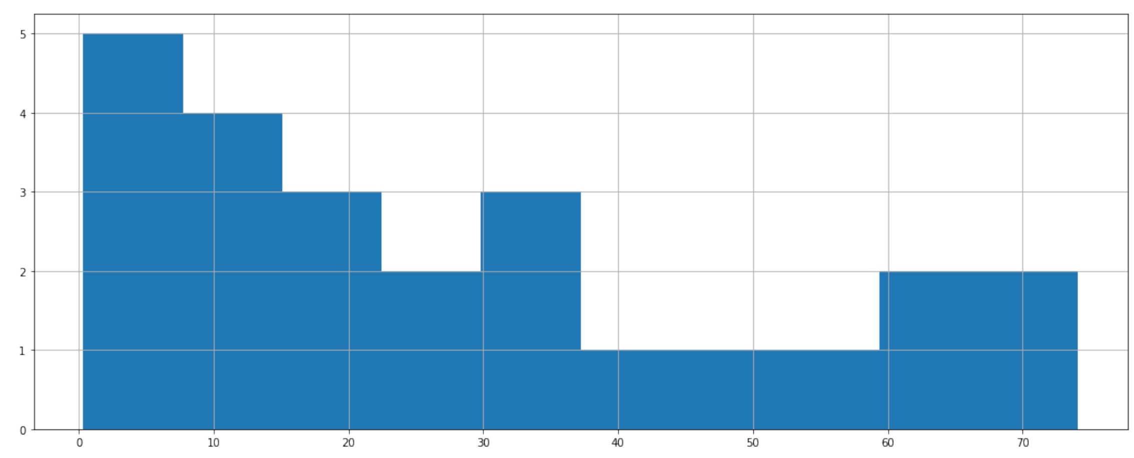
[61]:
```
clusters_df['demand_daily'] = clusters_df['demand'] / 30
clusters_df['demand_daily'].hist()
```

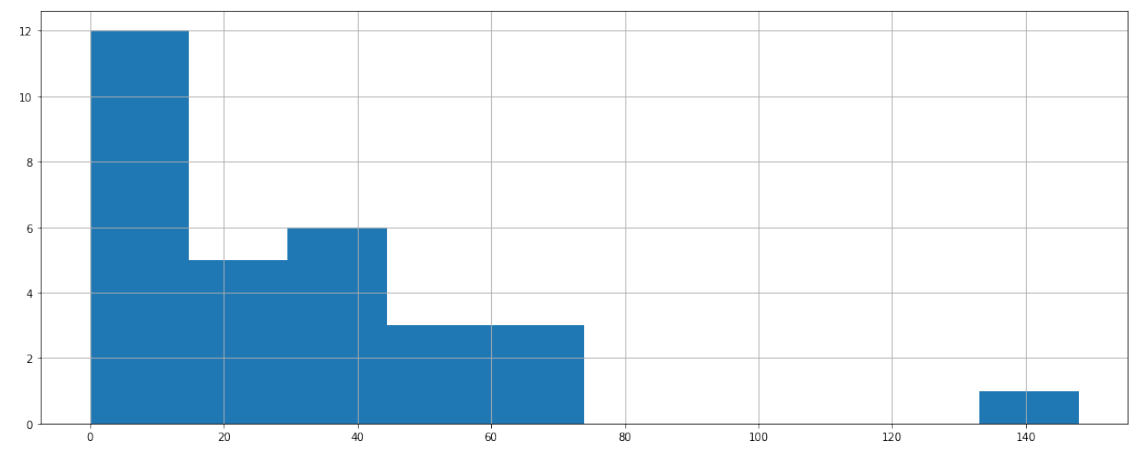
[61]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7fd3878ba8>
```

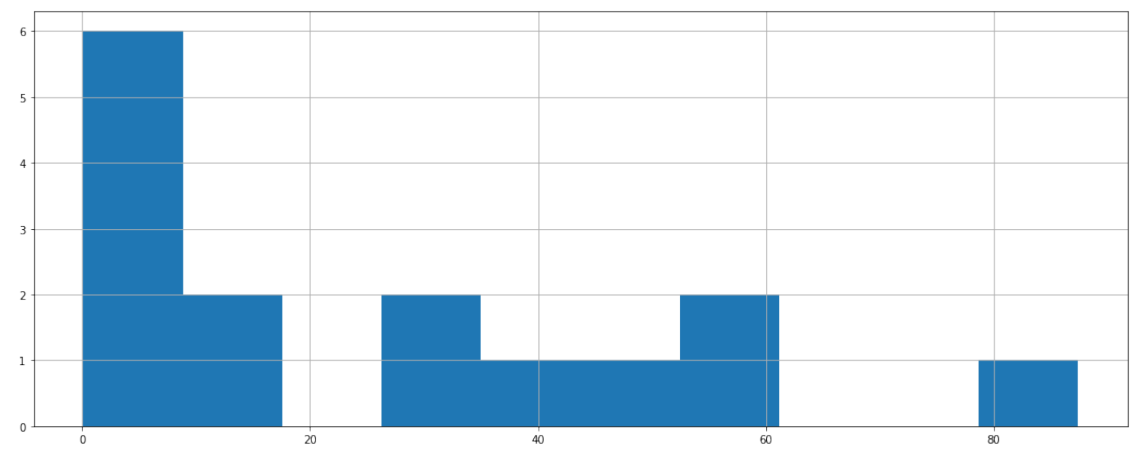


[62]:
```
for cluster in clusters_df['cluster'].unique():
    fig, ax1 = plt.subplots(1, 1)
    fig.set_size_inches(18, 7)
    clusters_df['demand_daily'][clusters_df['cluster'] == cluster].hist()
```









Let's further assume that each user's demand is a 0.7 kg parcel on average. Then the total daily average demand for each cluster, i.e. Satellites, is as follows:

```
[63]: assumed_weight = 0.7
```

```
[64]: clusters_df[['cluster', 'demand_daily']].groupby("cluster").sum() *␣
       ↪assumed_weight
```

```
[64]:          demand_daily
      cluster
      0         1049.104000
      1          492.501333
      2          617.292667
      3          268.226000
```

```
[65]: clusters_df['demand_daily'].sum() * assumed_weight
```

```
[65]: 2427.124
```

### Choosing vehicles for transportation

To address such demand, we have to decide on the types of vehicles that will deliver the parcels from the City Distribution Center to the Satellites, and from there to the individual consumers.

The vehicle for delivering between CDC and S needs to be able to carry up to 3,000 kg of cargo.

The vehicles delivering from Satellites need to be capable of delivering up to 1,100 kg of cargo. However, for smaller deliveries, a load capacity of as small as 300 kg, 650 kg, and 500 kg are required.

To ensure CSR and environmental sustainability, let's first consider fully electric vehicles in order to find those that can potentially fulfill the demand.

Mitsubishi Fuso Canter For deliveries from CDC to Satellites, Mitsubishi Fuso Canter can be considered. The vehicle is capable of carrying up to 3,000 kg of cargo, and can cover up to 100 km on a single charge. Although the charge is quite small, the Satellites can be equipped with quick chargers, which will be charging the truck during unloading. The maximum speed of the truck is 90 km/h, which is enough for routes within the city. It can be charged fully in 7 hours from a regular charger, and in just an hour from a quick charger.

Electric tricycles TailG For deliveries from Satellites to customers, compact and mobile all-electric tricycles seem like a perfect option. Their optimal speed is 35 km/h, and they can cover up to 60 km on a single stock battery, but additional compartments for batteries allow to increases this by a factor of 2 or 3. It charges fully within 5-8 hours. The maximum load of the tricycle is 500 kg, which makes it a perfect option for two of the four Satellites.

Volkswagen e-Caddy For deliveries at higher load Satellites, a small electric cargo vehicle e-Caddy can be considered. Not only its single charge capacity is enough to cover 250 km, but it also has a payload of 636 kg.

Thus, to cover the needs of our consumers, we would need: * 1 Fuso Canter for CDC * 3 TailG (1 each for Satellites 1 and 3, and 1 more as a backup for Satellite 0) * 3 e-Caddy (1 for Satellite 2 and 2 for Satellite 0)

### Satellites locations

In order to estimate the candidate locations for the CDC, let's first obtain the locations of Satellites. In order to do so, we perform a K-Means cluster centroid search for each of the clusters obtained by Agglomerative Clustering.

[66]:
```
satellites_centroids = {}
for cluster in clusters_df["cluster"].unique():
    num_clusters = 1
    kmeans = KMeans(n_clusters = num_clusters).fit(clusters_df[['lat',␣
 ↪'lon']][clusters_df["cluster"] == cluster])
    satellites_centroids[cluster] = kmeans.cluster_centers_[0]
```

[67]:
```
satellites_centroids
```

[67]:
```
{0: array([50.4333465 , 30.50619528]),
 1: array([50.50253647, 30.48799279]),
 2: array([50.42232394, 30.62077663]),
 3: array([50.49934511, 30.60136474])}
```

[68]:
```
fig = gmaps.figure(center = (lat, lon), zoom_level = zoom, layout=figure_layout)
for clust in clusters_df['cluster'].unique():
    cluster_layer = gmaps.symbol_layer(
        clusters_df[['lat', 'lon']][clusters_df['cluster'] == clust],
        fill_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        stroke_color = clusters_df['color'][clusters_df['cluster'] == clust].
 ↪iloc[0],
        scale = 3
    )
    fig.add_layer(cluster_layer)

centers_layer = gmaps.symbol_layer(
        satellites_centroids.values(),
        fill_color = "#ff007f",
        stroke_color = "#ff007f",
        scale = 5)
fig.add_layer(centers_layer)

fig
```

```
Figure(layout=FigureLayout(border='1px solid black', height='800px', padding='1px', width='800p
```

Altough it might seem a little counter-intuitive, the knowledge of the Kyiv city logistics hints us that

CDC should be placed either somewhere near Podil region, or alternatively near the Blockbuster Mall.

Let's define the corresponding points and compute the distances to be covered to the Satellites.

[69]:
```
podil_cdc = (50.471194, 30.523142)
mall_cdc = (50.486732, 30.519672)
```

[70]:
```
podil_sat_coords = [(x, y) for (x, y) in satellites_centroids.values()] +␣
 ↪[podil_cdc]
mall_sat_coords = [(x, y) for (x, y) in satellites_centroids.values()] +␣
 ↪[mall_cdc]
```

[71]:
```
loc_podil_matrix = "data/podil_cdc_matrix.npy"
loc_mall_matrix = "data/mall_cdc_matrix.npy"
```

[72]:
```
try:
    podil_driving = np.load(loc_podil_matrix)
except:
    podil_driving = np.empty((len(podil_sat_coords), len(podil_sat_coords)))
    columns = len(podil_sat_coords)
    gmaps_api = googlemaps.Client(key = api_key)
    for row in tqdm(range(len(podil_sat_coords))):
        for column in tqdm(range(columns)):
            origins = (podil_sat_coords[row][0], podil_sat_coords[row][1])
            destination = (podil_sat_coords[column][0],␣
 ↪podil_sat_coords[column][1])
    #         print(f"Distance from {origins} to {destination}")
            distance = gmaps_api.distance_matrix(origins, destination,␣
 ↪mode='driving')["rows"][0]["elements"][0]["distance"]["value"]
            podil_driving[row, column] = distance
        sleep(5)
    np.save(loc_podil_matrix, podil_driving)
```

[73]:
```
try:
    mall_driving = np.load(loc_mall_matrix)
except:
    mall_driving = np.empty((len(mall_sat_coords), len(mall_sat_coords)))
    columns = len(mall_sat_coords)
    gmaps_api = googlemaps.Client(key = api_key)
    for row in tqdm(range(len(mall_sat_coords))):
        for column in tqdm(range(columns)):
            origins = (mall_sat_coords[row][0], mall_sat_coords[row][1])
            destination = (mall_sat_coords[column][0],␣
 ↪mall_sat_coords[column][1])
    #         print(f"Distance from {origins} to {destination}")
            distance = gmaps_api.distance_matrix(origins, destination,␣
 ↪mode='driving')["rows"][0]["elements"][0]["distance"]["value"]
```

```
            mall_driving[row, column] = distance
        sleep(5)
    np.save(loc_mall_matrix, mall_driving)
```

## Finding optimal route for both candidate CDCs

```
[74]: from itertools import permutations
      l = list(permutations(range(0, 4)))
      routes = [(4, *l[i], 4) for i in range(len(l))]
      routes
```

```
[74]: [(4, 0, 1, 2, 3, 4),
       (4, 0, 1, 3, 2, 4),
       (4, 0, 2, 1, 3, 4),
       (4, 0, 2, 3, 1, 4),
       (4, 0, 3, 1, 2, 4),
       (4, 0, 3, 2, 1, 4),
       (4, 1, 0, 2, 3, 4),
       (4, 1, 0, 3, 2, 4),
       (4, 1, 2, 0, 3, 4),
       (4, 1, 2, 3, 0, 4),
       (4, 1, 3, 0, 2, 4),
       (4, 1, 3, 2, 0, 4),
       (4, 2, 0, 1, 3, 4),
       (4, 2, 0, 3, 1, 4),
       (4, 2, 1, 0, 3, 4),
       (4, 2, 1, 3, 0, 4),
       (4, 2, 3, 0, 1, 4),
       (4, 2, 3, 1, 0, 4),
       (4, 3, 0, 1, 2, 4),
       (4, 3, 0, 2, 1, 4),
       (4, 3, 1, 0, 2, 4),
       (4, 3, 1, 2, 0, 4),
       (4, 3, 2, 0, 1, 4),
       (4, 3, 2, 1, 0, 4)]
```

Searching both minimum paths for Podil and Mall CDCs.

```
[75]: # Initializing total distances to arbitrarily large numbers to find min via␣
       ↪comparison
      podil_total_dist = 1e9
      mall_total_dist = 1e9
      podil_route = ""
      mall_route = ""
      for route in routes:
          podil_dist = 0
          mall_dist = 0
          for i in range(len(route) - 1):
```

```
        podil_dist += podil_driving[route[i], route[i + 1]]
        mall_dist += mall_driving[route[i], route[i + 1]]
    if podil_dist < podil_total_dist:
        podil_total_dist = podil_dist
        podil_route = route
    if mall_dist < mall_total_dist:
        mall_total_dist = mall_dist
        mall_route = route
```

[76]:
```
print(f"For Podil CDC the optimal route is {podil_route}. The distance is␣
 ↪{podil_total_dist}")
print(f"For Mall CDC the optimal route is {mall_route}. The distance is␣
 ↪{mall_total_dist}")
```

```
For Podil CDC the optimal route is (4, 1, 3, 2, 0, 4). The distance is 47828.0
For Mall CDC the optimal route is (4, 3, 2, 0, 1, 4). The distance is 47324.0
```

As can be deduced from the computed results, the CDCs nearby the Blockbuster Mall has a 500 meter shorter path. For the purposes of further analysis, we assume that it is indeed a better choice for the CDC.

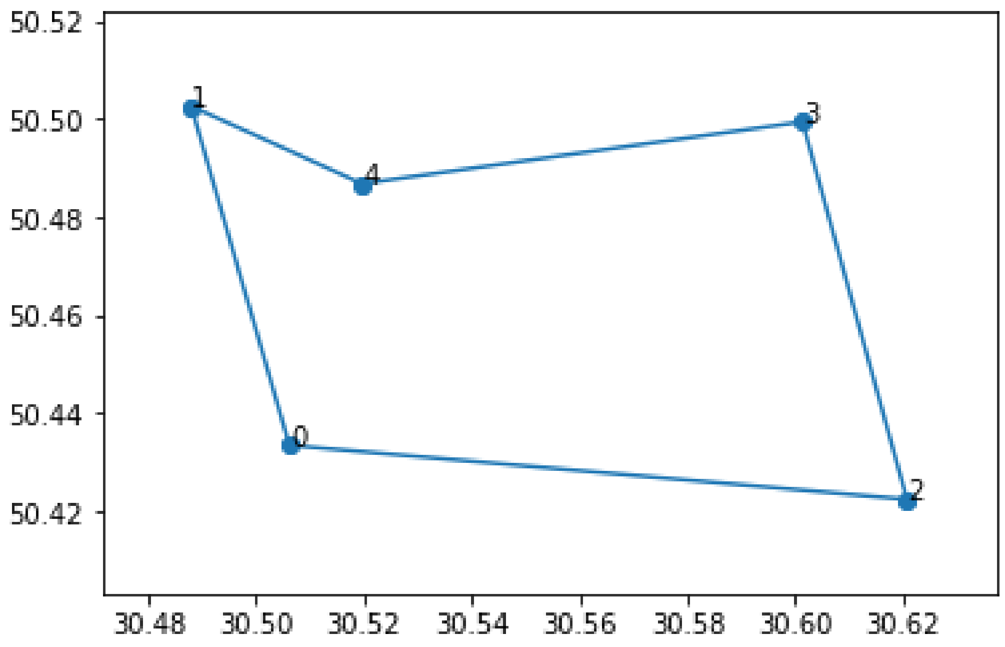
[77]:
```
positions = np.concatenate((np.array(mall_sat_coords), np.
 ↪array(mall_sat_coords)))
distances = np.concatenate((np.array(mall_driving), np.array(mall_driving)))
N = 5
```

[78]:
```
full_route = mall_route
full_route
```

[78]:
```
(4, 3, 2, 0, 1, 4)
```

[79]:
```
new_cities_order = (np.array([mall_sat_coords[full_route[i]] for i in␣
 ↪range(len(full_route))]))
# Plot the cities.
fig, ax = plt.subplots()
ax.scatter(positions[:,1],positions[:,0])
for i in range(len((mall_sat_coords))):
    ax.annotate(i, (positions[i][1], positions[i][0]))
# Plot the path.
ax.plot(new_cities_order[:,1], new_cities_order[:,0])
# plt.show()
# Print the route as row numbers and the total distance travelled by the path.
print("Route: " + str(full_route) + "\n\nDistance: " + str(mall_total_dist))
```

```
Route: (4, 3, 2, 0, 1, 4)

Distance: 47324.0
```



## Defining transportation cost

To recall, we have made a decision to promote all-electric parcel delivery. Thus we need to estimate the cost of 1 km driven.

The average cost of 1 km travelled by an electric vehicle in Ukraine is estimated at $0.05, which is the lowest price one can get for a cargo vehicle.

We will take the cost of 1 km as the only determinant of the variable costs. Since we have the payload capacity of the vehicles as given, the quantity of the total vehicles can be computed. Thus, all other expenses will be considered as either fixed costs or investments (e.g. the purchase of the vehicle).

[80]:
```
cost_of_km = 0.05
```

## Scenario 1. 2-Echelon Single Source Location Model

In this scenario we assume that the Satellites act as Pack Stations, i.e. the customers come to them to pick up their parcels.

The problem can then be simplified to a Traveling Salesman Problem, where the City Distribution Center acts as the starting and ending node, and the salesman has to visit each other node on the graph.

We have solved this problem earlier, when we were choosing the proper location for the CDC. To recap:

[81]:
```
# Initializing total distance to arbitrarily large numbers to find min via␣
↪comparison
```

```
mall_total_dist = 1e9
mall_route = ""
for route in routes:
    mall_dist = 0
    for i in range(len(route) - 1):
        mall_dist += mall_driving[route[i], route[i + 1]]
    if mall_dist < mall_total_dist:
        mall_total_dist = mall_dist
        mall_route = route
```

[82]:
```
print(f"For Mall CDC the optimal route is {mall_route}. The distance is␣
↪{mall_total_dist}")
```

```
For Mall CDC the optimal route is (4, 3, 2, 0, 1, 4). The distance is 47324.0
```

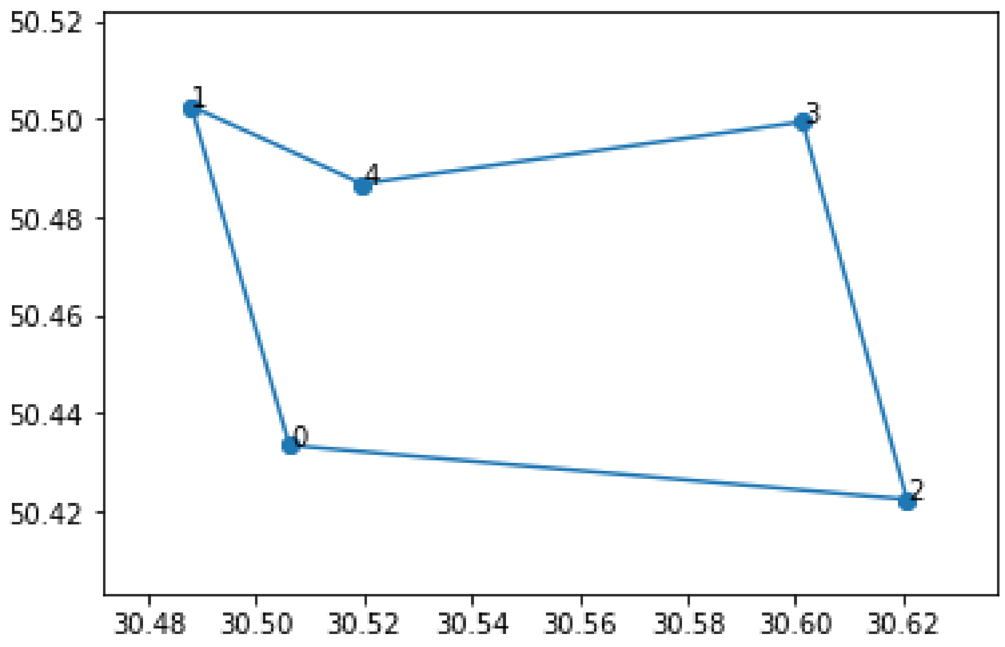
[83]:
```
positions = np.concatenate((np.array(mall_sat_coords), np.
↪array(mall_sat_coords)))
distances = np.concatenate((np.array(mall_driving), np.array(mall_driving)))
N = 5
```

[84]:
```
full_route = mall_route
full_route
```

[84]:
```
(4, 3, 2, 0, 1, 4)
```

[85]:
```
new_cities_order = (np.array([mall_sat_coords[full_route[i]] for i in␣
↪range(len(full_route))]))
# Plot the cities.
fig, ax = plt.subplots()
ax.scatter(positions[:,1],positions[:,0])
for i in range(len((mall_sat_coords))):
    ax.annotate(i, (positions[i][1], positions[i][0]))
# Plot the path.
ax.plot(new_cities_order[:,1], new_cities_order[:,0])
# plt.show()
# Print the route as row numbers and the total distance travelled by the path.
print("Route: " + str(full_route) + "\n\nDistance: " + str(mall_total_dist))
```

```
Route: (4, 3, 2, 0, 1, 4)

Distance: 47324.0
```


0
1
2
3
4
50.52
50.50
50.48
50.46
50.44
50.42
30.48
30.50
30.52
30.54
30.56
30.58
30.60
30.62