Clustering

September 28, 2021

1 Data Clustering

1.1 What is data clustering

Data clustering means to aggregate data into different separated groups based on the data characteristics. The issues is: how to characterize data, and what is the criterior to put them into different groups?
Data clustering can happen in very different ways. Sometimes obvious, the data can have natural criterior, for example human related data can be grouped in gender, or age, or location, maybe for business data can group in different industries, different states or locations, etc. But also in many cases, data seems don't have an obvious way to differentiate them or group them. Then we need some special technique to analyze and find unseen rules which is hidden from us, among all the data.

Let's first study a data clustering based on geographic location.

1.2 John Snow's cholera data - Distance on data clustering

In 1854, there was an outbreak of cholera in North-western London, in the neighborhood around Broad Street. People did not know much about the disease at that time, and dont know how to control it. John Snow, a physician at that time, collected many data, including the location of those people who dead because of the disease. When he map all location on the map, it leaded him to connect it to the water pump on the map, and thus comes the idea that this disease maybe spread by water.

[7]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn')
plt.rcParams.update({'figure.figsize': (12,6),\
                     'figure.titlesize':16,\
                     'axes.titlesize':16,\
                     'axes.labelsize':14,\
                     'xtick.labelsize':14,\
                     'ytick.labelsize':14,
                    })
```

[8]:
```
import pathlib
```

```python
file_dir = pathlib.Path('D:/Edu/data_resource/data-Snow-Cholera/')
file_1 = file_dir/'snow_cholera_pumps.csv'
file_2 = file_dir/'snow_cholera_deaths.csv'

df_pumps = pd.read_csv(file_1)
df_pumps
```

[8]:

```
       x     y
0    8.7  17.9
1   11.0  18.5
2   13.4  17.4
3   14.9  17.8
4    8.7  14.9
5    8.9  12.8
6   12.6  11.7
7   10.7   7.4
8   13.5   8.0
9   16.4   9.3
10  18.9   9.7
11  16.0   5.0
12   9.0   5.1
```

[9]:

```python
df_deaths = pd.read_csv(file_2)
df_deaths
```

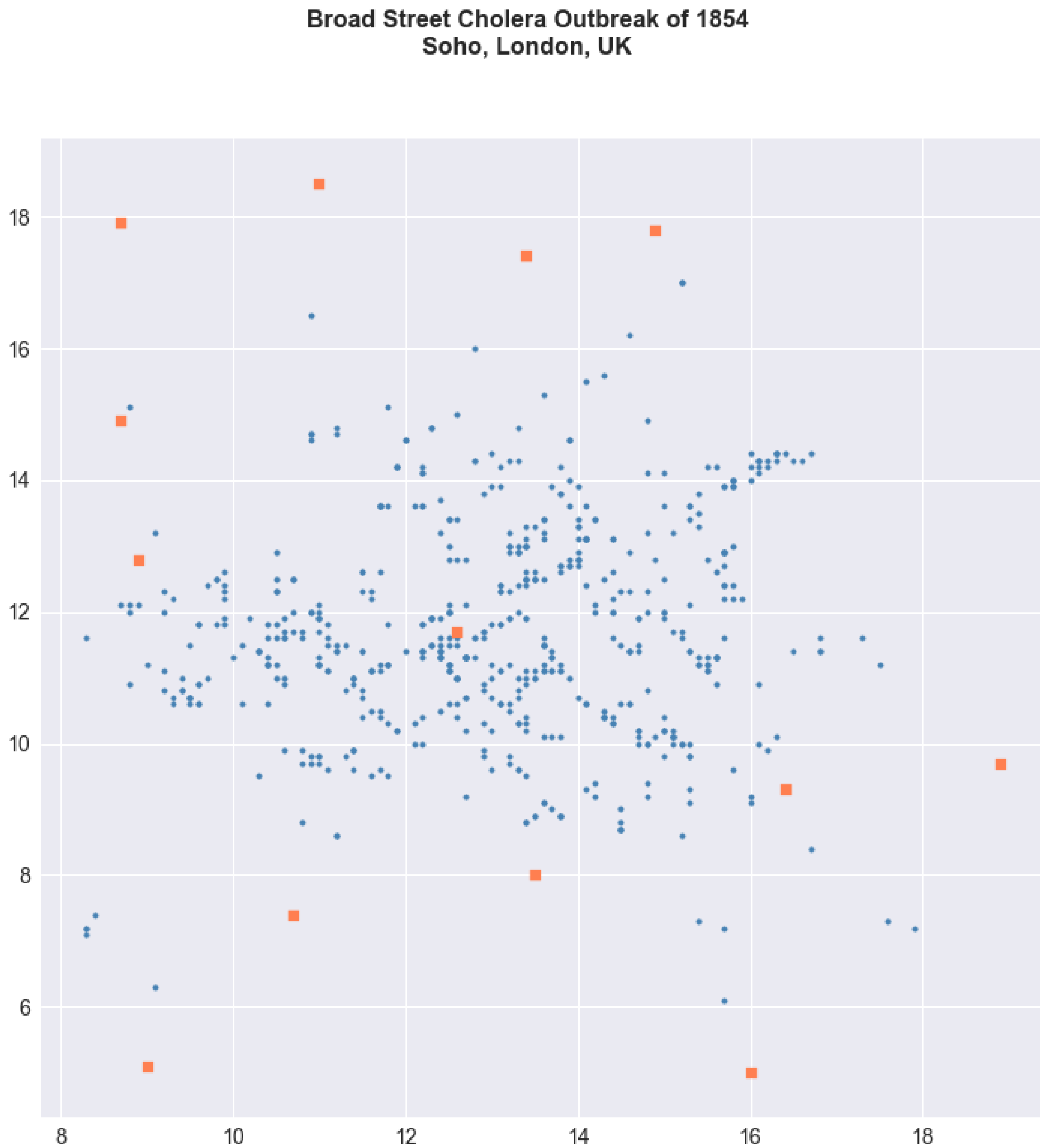
[9]:

```
        x     y
0    13.6  11.1
1     9.9  12.6
2    14.7  10.2
3    15.2  10.0
4    13.2  13.0
..    …     …
573  12.4  11.5
574  15.1  10.2
575  17.3  11.6
576  12.4  11.9
577  15.0  12.5

[578 rows x 2 columns]
```

[10]:

```python
fig, ax = plt.subplots()

fig.suptitle('Broad Street Cholera Outbreak of 1854'+'\n'+'Soho, London,␣
↪UK',fontweight='bold')
fig.set_size_inches((12,12))
ax.scatter(df_deaths['x'], df_deaths['y'], marker='.', color='steelblue')
ax.scatter(df_pumps['x'], df_pumps['y'], marker='s', color='coral')
```

```
plt.show()
```



We can group the death location point according to distance to the pump location. Then draw it on the map with different color.

[11]:
```
deaths_tmp = df_deaths.to_numpy()
idx_arr = np.array([-999 for _ in range(len(df_deaths))], dtype='int')
for i in range(len(df_deaths)):
    idx_arr[i] = (df_pumps - deaths_tmp[i]).apply(lambda x:x**2).sum(axis=1).
 ↪idxmin()
```

```
df_deaths['Pump'] = idx_arr
df_deaths
```

[11]:

```
        x     y  Pump
0    13.6  11.1     6
1     9.9  12.6     5
2    14.7  10.2     9
3    15.2  10.0     9
4    13.2  13.0     6
..    …     …     …
573  12.4  11.5     6
574  15.1  10.2     9
575  17.3  11.6     9
576  12.4  11.9     6
577  15.0  12.5     6

[578 rows x 3 columns]
```

[12]:

```
import matplotlib.patches as mpatches

fig = plt.figure(figsize=(12,12))
fig.suptitle('Broad Street Cholera Outbreak of 1854'+'\n'+'Soho, London,␣
↪UK',fontweight='bold')
ax = fig.add_subplot(111)

ax.scatter(df_deaths['x'], df_deaths['y'], c=df_deaths['Pump'], vmin=0.,␣
↪vmax=12., cmap='jet', marker='.')
ax.scatter(df_pumps['x'], df_pumps['y'], marker='s', color='coral')

for i in df_pumps.index:
    ax.text(df_pumps[['x']].loc[i], df_pumps[['y']].loc[i]+0.2, s=f'{i}',␣
↪ha='center', transform=ax.transData)

ellipse = mpatches.Ellipse(xy=(df_deaths['x'].mean(), df_deaths['y'].mean()),␣
↪width=df_deaths['x'].std(), height=df_deaths['y'].std(), zorder=32,␣
↪fc='None', ec='IndianRed', lw=2)
ax.add_artist(ellipse)
plt.show()
```

This clustering is based on 'Euclidean distance'. If combined with actual street data can do clustering study of 'walking distance'.

1.3 k-means clustering

k-means clustering is an algorithm that for a set of n observation data, to cluster them into k clusters in which each observation belongs to the cluster with the nearest mean (cluster centers or cluster centroid).
We use the data which we created for the WHO suicide rate, shunshine duration, and GDP, for different countries.

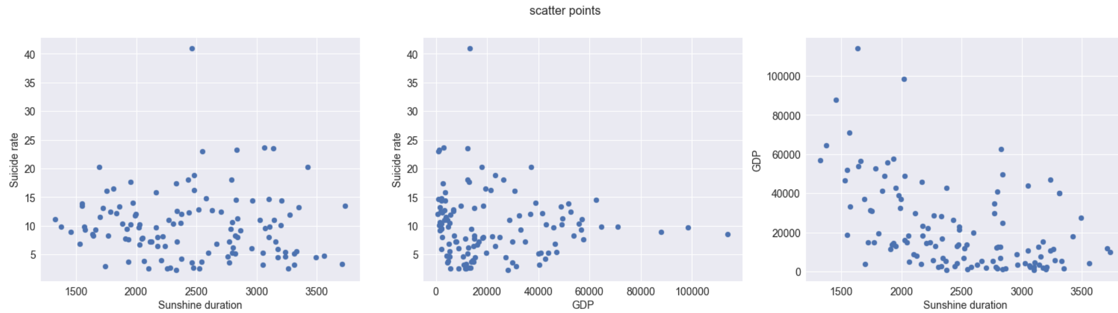
Here's a brief overview of how K-means works:

1. Decide the number of clusters (k)
2. Select k random points from the data as centroids
3. Assign all the points to the nearest cluster centroid
4. Calculate the centroid of newly formed clusters
5. Repeat steps 3 and 4

[13]:
```
# here we use the result data from the study of WHO data
dir = pathlib.Path.cwd()
file_path = dir/'..'/'UnitB1'/'final_data.csv'

df = pd.read_csv(file_path,index_col=0)
df
```

[13]:
```
             Both sexes         Year  NY.GDP.PCAP.PP.KD
Afghanistan         6.0  3175.100000        2065.036235
Albania             3.7  2544.000000       13671.488422
Algeria             2.6  3266.500000       11510.557088
Angola             12.6  2341.000000        6670.331458
Argentina           8.1  2220.300000       22063.904372
…                     …           …                  …
Uruguay            18.8  2481.400000       23032.734044
Uzbekistan          8.3  2823.900000        7014.324699
Vietnam             7.2  2123.500000        8041.178384
Zambia             14.4  2965.466667        3470.448024
Zimbabwe           23.6  3065.400000        3027.656038

[119 rows x 3 columns]
```

[14]:
```
fig, axes = plt.subplots(1,3, figsize = (26, 6))

fig.suptitle('scatter points')

axes[0].scatter(df['Year'], df['Both sexes'])
axes[0].set_ylabel('Suicide rate')
axes[0].set_xlabel('Sunshine duration')

axes[1].scatter(df['NY.GDP.PCAP.PP.KD'], df['Both sexes'])
axes[1].set_ylabel('Suicide rate')
axes[1].set_xlabel('GDP')

axes[2].scatter(df['Year'], df['NY.GDP.PCAP.PP.KD'])
axes[2].set_ylabel('GDP')
axes[2].set_xlabel('Sunshine duration')

plt.show()
```



Normally due different feature (column data) has different unit and variance, we need “whiten” it before use K-means algorithm.

1.3.1 K-means using SciPy stats

[15]:
```python
import scipy.stats as st
import scipy.cluster as sc

df_w = sc.vq.whiten(df)
df_w
```

[15]:
```
array([[1.04331888, 5.45664131, 0.09468723],
       [0.64337998, 4.37204985, 0.62687298],
       [0.45210485, 5.61371889, 0.52778871],
       [2.19096965, 4.02317952, 0.30585189],
       [1.40848049, 3.81574776, 1.01168688],
       [0.4694935 , 4.25174974, 0.62606024],
       [1.96491722, 4.88687706, 2.26766391],
       [1.80841939, 3.23779164, 2.56010143],
       [0.69554592, 3.79357816, 0.66207918],
       [0.67815727, 3.55057193, 0.21797062],
       [2.86912692, 3.10546151, 0.88418062],
       [2.41702207, 2.65691394, 2.37253736],
       [2.2083583 , 4.52064914, 0.15073161],
       [1.44325778, 3.04015574, 0.68305622],
       [3.5125069 , 5.88897701, 0.81511248],
       [1.11287347, 3.84544782, 0.67696157],
       [1.13026212, 3.74133354, 1.06339347],
       [2.50396531, 5.51490094, 0.09988172],
       [2.10402641, 4.07886129, 0.03446574],
       [2.76479503, 3.72092549, 0.16700775],
       [1.79103074, 3.48534254, 2.24708548],
       [3.99938904, 4.37634629, 0.0433372 ],
       [2.29530154, 5.76271917, 0.07242993],
       [1.39109184, 4.88380654, 1.1448279 ],
```

```
[1.16503942, 3.48626293, 0.73787347],
[0.64337998, 3.31817789, 0.66877371],
[2.01708317, 2.91928276, 0.17620223],
[1.91275128, 3.90459798, 1.31842283],
[0.55643674, 5.69552296, 1.84450289],
[1.32153725, 3.32028314, 2.64468949],
[2.06924911, 5.63520105, 0.25378329],
[1.3389259 , 3.28170118, 0.52137147],
[0.59121403, 6.37707467, 0.53937548],
[1.06070753, 5.08182053, 0.40332212],
[2.34746748, 2.66198374, 0.84839025],
[2.08663776, 3.42769396, 1.68875385],
[1.65192156, 4.78596817, 0.10185713],
[1.65192156, 3.30309741, 0.62746321],
[2.33007883, 3.19310874, 2.23252688],
[1.68669886, 3.72357495, 2.10161468],
[2.27791289, 2.95628937, 0.68549957],
[1.66931021, 5.27601928, 0.10192476],
[1.3389259 , 3.51620047, 0.68729612],
[1.44325778, 2.8253341 , 2.45949835],
[1.82580804, 4.06872171, 0.24746022],
[0.62599133, 4.76560308, 1.36288633],
[2.1388037 , 4.19331826, 0.11770512],
[2.15619235, 4.65217726, 0.08892142],
[7.11195704, 4.22872086, 0.59985271],
[0.45210485, 3.88363139, 0.26301871],
[2.05186046, 3.41652324, 1.49266296],
[1.94752858, 2.27882787, 2.60965161],
[2.24313559, 4.32568275, 0.3078511 ],
[0.45210485, 4.33363975, 0.54160952],
[0.88682105, 4.84929759, 0.56807782],
[0.81726646, 5.56955156, 0.48444383],
[1.54758967, 2.49708665, 4.02522155],
[0.9042097 , 5.69019539, 1.8344387 ],
[0.74771186, 4.07688493, 1.95618661],
[2.12141506, 3.15722493, 1.89738394],
[3.14734529, 4.17871039, 1.20829815],
[1.91275128, 5.20731932, 0.19853579],
[2.79957233, 3.01437714, 1.41495323],
[0.78248916, 5.46179703, 0.69577446],
[3.5125069 , 2.90610704, 1.69941513],
[1.49542373, 2.80814837, 5.22445355],
[1.59975562, 4.92273795, 0.07421388],
[1.84319669, 4.80490112, 0.06945119],
[1.39109184, 5.32734733, 0.10646589],
[0.92159834, 5.24852211, 2.01524774],
[0.95637564, 5.72714471, 0.23829916],
```

```
[0.92159834, 4.4618453 , 0.90335711],
[3.12995664, 4.79739669, 0.56475663],
[2.81696098, 4.26377975, 0.98738707],
[1.2693713 , 5.42289999, 0.34561313],
[4.03416634, 4.87731033, 0.05876035],
[2.34746748, 6.4231668 , 0.45142839],
[1.61714426, 2.85626842, 2.59659068],
[1.79103074, 3.35796286, 1.96606419],
[0.81726646, 4.74308978, 0.24997486],
[1.75625345, 5.50493321, 0.05614695],
[1.19981671, 4.58700897, 0.23547589],
[1.25198266, 4.01974238, 0.76115937],
[1.72147615, 2.36017366, 2.95532758],
[0.78248916, 6.00349126, 1.25152652],
[1.7040875 , 5.34221299, 0.21507082],
[0.50427079, 2.99633212, 1.44124341],
[0.62599133, 4.23284544, 0.19944991],
[1.07809618, 4.81716028, 0.5784545 ],
[0.4694935 , 3.92118221, 0.58937491],
[0.4347162 , 3.61433099, 0.40876306],
[1.61714426, 2.69987827, 1.51866111],
[1.25198266, 4.76044737, 1.59932442],
[1.2693713 , 3.63478201, 1.36905004],
[0.93898699, 5.5621617 , 2.15333551],
[1.91275128, 5.40663083, 0.15413168],
[1.37370319, 3.53166763, 0.83874337],
[1.68669886, 3.47564215, 4.51242468],
[2.43441072, 3.39246321, 1.78391964],
[3.02562475, 4.00427522, 0.12201278],
[2.55613126, 5.33273219, 0.03974328],
[4.08633228, 5.39858218, 0.57232332],
[0.92159834, 4.81234827, 1.87105646],
[0.8346551 , 6.11734672, 0.19192058],
[2.15619235, 3.06335647, 2.40868974],
[1.7040875 , 2.69128541, 3.25187968],
[1.39109184, 3.73793936, 0.84613915],
[2.57351991, 4.43383256, 0.09729458],
[0.55643674, 5.24156189, 0.49317073],
[0.3999389 , 4.0108058 , 1.29299919],
[1.80841939, 3.95959232, 0.10030092],
[3.0777907 , 3.35981032, 0.58731645],
[1.19981671, 2.62941678, 2.12785588],
[2.52135396, 4.859469  , 2.87178696],
[3.26906583, 4.26446718, 1.05611022],
[1.44325778, 4.85307845, 0.32162487],
[1.25198266, 3.64938988, 0.36870875],
[2.50396531, 5.09637111, 0.15912899],
```

```
[4.10372093, 5.26811385, 0.13882583]])
```

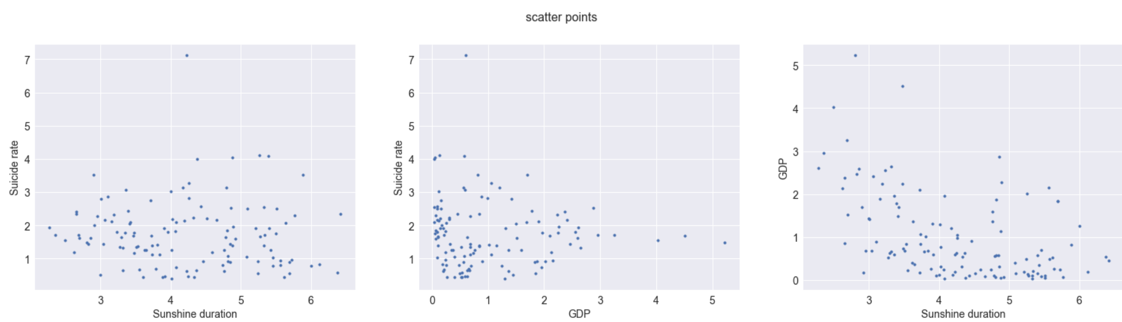
[16]:
```
df_w.mean(axis=0)
```

[16]:
```
array([1.73608846, 4.23325818, 1.01685545])
```

[17]:
```
df_w.std(axis=0)
```

[17]:
```
array([1., 1., 1.])
```

[18]:
```
fig, axes = plt.subplots(1,3, figsize = (26, 6))

fig.suptitle('scatter points')

# column 0 is suicide rate.
# column 1 is yearly sunshine duration
# column 2 is GDP PPP

axes[0].scatter(df_w.T[1], df_w.T[0], marker='.')
axes[0].set_ylabel('Suicide rate')
axes[0].set_xlabel('Sunshine duration')

axes[1].scatter(df_w.T[2], df_w.T[0], marker='.')
axes[1].set_ylabel('Suicide rate')
axes[1].set_xlabel('GDP')

axes[2].scatter(df_w.T[1], df_w.T[2], marker='.')
axes[2].set_ylabel('GDP')
axes[2].set_xlabel('Sunshine duration')

plt.show()
```



[19]:
```
centroid, label = sc.vq.kmeans2(df_w, 2)
```

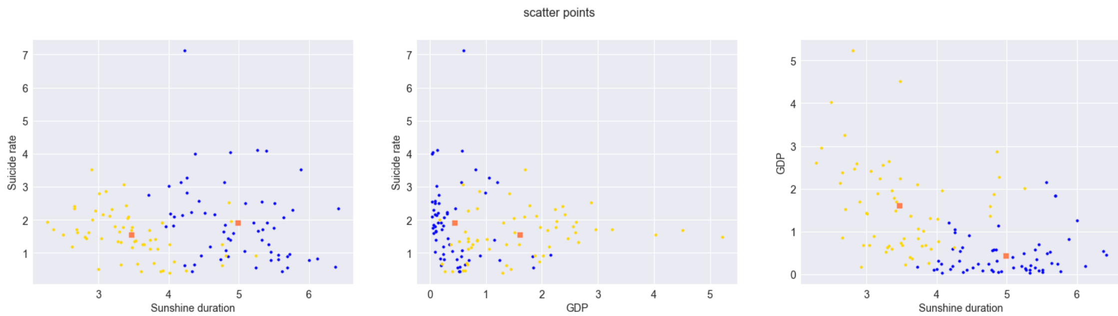
[20]:
```
centroid
```

[20]: array([[1.91622901, 4.9836922 , 0.44010809],
       [1.55289468, 3.47010494, 1.60337819]])

[21]:
```
label
```

[21]: array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0])

[22]:
```
s0 = (label==0)
s1 = (label==1)

fig, axes = plt.subplots(1,3, figsize = (26, 6))

fig.suptitle('scatter points')

# column 0 is suicide rate.
# column 1 is yearly sunshine duration
# column 2 is GDP PPP

axes[0].scatter(df_w.T[1][s0], df_w.T[0][s0], marker='.', color = 'blue')
axes[0].scatter(df_w.T[1][s1], df_w.T[0][s1], marker='.', color = 'gold')
axes[0].scatter(centroid.T[1], centroid.T[0], marker='s', color = 'coral')
axes[0].set_ylabel('Suicide rate')
axes[0].set_xlabel('Sunshine duration')

axes[1].scatter(df_w.T[2][s0], df_w.T[0][s0], marker='.', color = 'blue')
axes[1].scatter(df_w.T[2][s1], df_w.T[0][s1], marker='.', color = 'gold')
axes[1].scatter(centroid.T[2], centroid.T[0], marker='s', color = 'coral')
axes[1].set_ylabel('Suicide rate')
axes[1].set_xlabel('GDP')

axes[2].scatter(df_w.T[1][s0], df_w.T[2][s0], marker='.', color = 'blue')
axes[2].scatter(df_w.T[1][s1], df_w.T[2][s1], marker='.', color = 'gold')
axes[2].scatter(centroid.T[1], centroid.T[2], marker='s', color = 'coral')
axes[2].set_ylabel('GDP')
axes[2].set_xlabel('Sunshine duration')

plt.show()
```



[23]:
```python
centroid2, label2 = sc.vq.kmeans2(df_w, 2)
```

[24]:
```python
centroid2
```

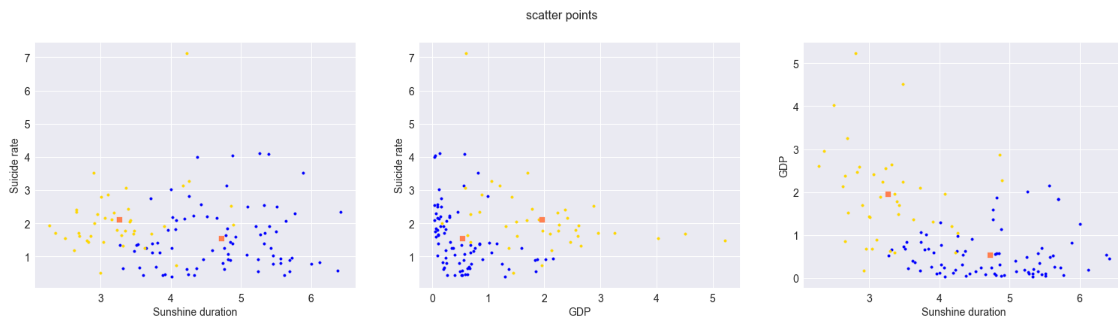
[24]:
```
array([[1.54648913, 4.72032931, 0.53965965],
       [2.11054715, 3.2712927 , 1.95931715]])
```

[25]:
```python
label2
```

[25]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 0])
```

[26]:
```python
s0 = (label2==0)
s1 = (label2==1)

fig, axes = plt.subplots(1,3, figsize = (26, 6))

fig.suptitle('scatter points')

# column 0 is suicide rate.
# column 1 is yearly sunshine duration
# column 2 is GDP PPP

axes[0].scatter(df_w.T[1][s0], df_w.T[0][s0], marker='.', color = 'blue')
axes[0].scatter(df_w.T[1][s1], df_w.T[0][s1], marker='.', color = 'gold')
axes[0].scatter(centroid2.T[1], centroid2.T[0], marker='s', color = 'coral')
axes[0].set_ylabel('Suicide rate')
axes[0].set_xlabel('Sunshine duration')

axes[1].scatter(df_w.T[2][s0], df_w.T[0][s0], marker='.', color = 'blue')
axes[1].scatter(df_w.T[2][s1], df_w.T[0][s1], marker='.', color = 'gold')
```

```
axes[1].scatter(centroid2.T[2], centroid2.T[0], marker='s', color = 'coral')
axes[1].set_ylabel('Suicide rate')
axes[1].set_xlabel('GDP')

axes[2].scatter(df_w.T[1][s0], df_w.T[2][s0], marker='.', color = 'blue')
axes[2].scatter(df_w.T[1][s1], df_w.T[2][s1], marker='.', color = 'gold')
axes[2].scatter(centroid2.T[1], centroid2.T[2], marker='s', color = 'coral')
axes[2].set_ylabel('GDP')
axes[2].set_xlabel('Sunshine duration')

plt.show()
```



[27]:
```
centroid3, label3 = sc.vq.kmeans2(df_w, 2)
```

[28]:
```
centroid3
```

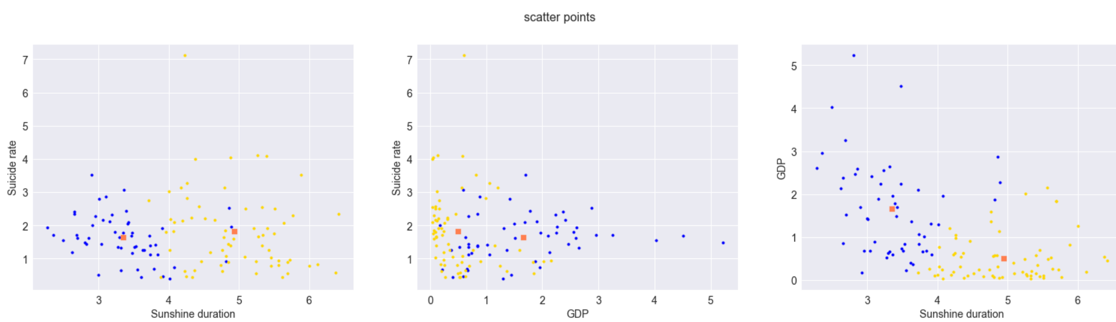
[28]:
```
array([[1.634861  , 3.35483953, 1.65984182],
       [1.81737718, 4.93865497, 0.50051791]])
```

[29]:
```
label3
```

[29]:
```
array([1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 1])
```

[30]:
```
s0 = (label3==0)
s1 = (label3==1)

fig, axes = plt.subplots(1,3, figsize = (26, 6))

fig.suptitle('scatter points')
```

```
# column 0 is suicide rate.
# column 1 is yearly sunshine duration
# column 2 is GDP PPP

axes[0].scatter(df_w.T[1][s0], df_w.T[0][s0], marker='.', color = 'blue')
axes[0].scatter(df_w.T[1][s1], df_w.T[0][s1], marker='.', color = 'gold')
axes[0].scatter(centroid3.T[1], centroid3.T[0], marker='s', color = 'coral')
axes[0].set_ylabel('Suicide rate')
axes[0].set_xlabel('Sunshine duration')

axes[1].scatter(df_w.T[2][s0], df_w.T[0][s0], marker='.', color = 'blue')
axes[1].scatter(df_w.T[2][s1], df_w.T[0][s1], marker='.', color = 'gold')
axes[1].scatter(centroid3.T[2], centroid3.T[0], marker='s', color = 'coral')
axes[1].set_ylabel('Suicide rate')
axes[1].set_xlabel('GDP')

axes[2].scatter(df_w.T[1][s0], df_w.T[2][s0], marker='.', color = 'blue')
axes[2].scatter(df_w.T[1][s1], df_w.T[2][s1], marker='.', color = 'gold')
axes[2].scatter(centroid3.T[1], centroid3.T[2], marker='s', color = 'coral')
axes[2].set_ylabel('GDP')
axes[2].set_xlabel('Sunshine duration')

plt.show()
```



1.3.2 K-means clustering using SciKit Learn

[31]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(df)
scaler
```

[31]: StandardScaler()

[32]:
```
df_scaled = scaler.transform(df)
df_scaled
```

```
[32]: array([[-6.92769582e-01,  1.22338313e+00, -9.22168215e-01],
       [-1.09270849e+00,  1.38791674e-01, -3.89982468e-01],
       [-1.28398361e+00,  1.38046071e+00, -4.89066739e-01],
       [ 4.54881187e-01, -2.10078656e-01, -7.11003559e-01],
       [-3.27607973e-01, -4.17510424e-01, -5.16856618e-03],
       [-1.26659497e+00,  1.84915604e-02, -3.90795207e-01],
       [ 2.28828763e-01,  6.53618876e-01,  1.25080846e+00],
       [ 7.23309308e-02, -9.95466543e-01,  1.54324598e+00],
       [-1.04054254e+00, -4.39680016e-01, -3.54776264e-01],
       [-1.05793119e+00, -6.82686246e-01, -7.98884832e-01],
       [ 1.13303846e+00, -1.12779667e+00, -1.32674828e-01],
       [ 6.80933611e-01, -1.57634424e+00,  1.35568192e+00],
       [ 4.72269835e-01,  2.87390958e-01, -8.66123834e-01],
       [-2.92830677e-01, -1.19310244e+00, -3.33799231e-01],
       [ 1.77641844e+00,  1.65571883e+00, -2.01742970e-01],
       [-6.23214990e-01, -3.87810357e-01, -3.39893879e-01],
       [-6.05826342e-01, -4.91924637e-01,  4.65380195e-02],
       [ 7.67876851e-01,  1.28164276e+00, -9.16973730e-01],
       [ 3.67937947e-01, -1.54396889e-01, -9.82389710e-01],
       [ 1.02870657e+00, -5.12332692e-01, -8.49847694e-01],
       [ 5.49422828e-02, -7.47915642e-01,  1.23023003e+00],
       [ 2.26330058e+00,  1.43088107e-01, -9.73518250e-01],
       [ 5.59213075e-01,  1.52946099e+00, -9.44425520e-01],
       [-3.44996621e-01,  6.50548359e-01,  1.27972455e-01],
       [-5.71049045e-01, -7.46995250e-01, -2.78981982e-01],
       [-1.09270849e+00, -9.15080288e-01, -3.48081738e-01],
       [ 2.80994707e-01, -1.31397542e+00, -8.40653217e-01],
       [ 1.76662819e-01, -3.28660197e-01,  3.01567381e-01],
       [-1.17965173e+00,  1.46226478e+00,  8.27647446e-01],
       [-4.14551213e-01, -9.12975036e-01,  1.62783404e+00],
       [ 3.33160651e-01,  1.40194287e+00, -7.63072154e-01],
       [-3.97162565e-01, -9.51557001e-01, -4.95483981e-01],
       [-1.14487443e+00,  2.14381649e+00, -4.77479967e-01],
       [-6.75380934e-01,  8.48562346e-01, -6.13533332e-01],
       [ 6.11379019e-01, -1.57127445e+00, -1.68465203e-01],
       [ 3.50549299e-01, -8.05564220e-01,  6.71898405e-01],
       [-8.41669013e-02,  5.52709995e-01, -9.14998319e-01],
       [-8.41669013e-02, -9.30160766e-01, -3.89392235e-01],
       [ 5.93990371e-01, -1.04014944e+00,  1.21567143e+00],
       [-4.93896053e-02, -5.09683225e-01,  1.08475924e+00],
       [ 5.41824427e-01, -1.27696881e+00, -3.31355883e-01],
       [-6.67782533e-02,  1.04276110e+00, -9.14930690e-01],
       [-3.97162565e-01, -7.17057708e-01, -3.29559329e-01],
       [-2.92830677e-01, -1.40792408e+00,  1.44264290e+00],
       [ 8.97195788e-02, -1.64536470e-01, -7.69395229e-01],
       [-1.11009713e+00,  5.32344904e-01,  3.46030886e-01],
       [ 4.02715243e-01, -3.99399235e-02, -8.99150324e-01],
```

```
[ 4.20103891e-01,  4.18919082e-01, -9.27934026e-01],
[ 5.37586857e+00, -4.53731854e-03, -4.17002741e-01],
[-1.28398361e+00, -3.49626788e-01, -7.53836736e-01],
[ 3.15772003e-01, -8.16734945e-01,  4.75807516e-01],
[ 2.11440115e-01, -1.95443031e+00,  1.59279616e+00],
[ 5.07047131e-01,  9.24245733e-02, -7.09004352e-01],
[-1.28398361e+00,  1.00381567e-01, -4.75245925e-01],
[-8.49267414e-01,  6.16039412e-01, -4.48777626e-01],
[-9.18822006e-01,  1.33629338e+00, -5.32411623e-01],
[-1.88498789e-01, -1.73617153e+00,  3.00836610e+00],
[-8.31878766e-01,  1.45693721e+00,  8.17583249e-01],
[-9.88376598e-01, -1.56373248e-01,  9.39331159e-01],
[ 3.85326595e-01, -1.07603325e+00,  8.80528494e-01],
[ 1.41125683e+00, -5.45477945e-02,  1.91442704e-01],
[ 1.76662819e-01,  9.74061144e-01, -8.18319663e-01],
[ 1.06348387e+00, -1.21888104e+00,  3.98097777e-01],
[-9.53599302e-01,  1.22853885e+00, -3.21080988e-01],
[ 1.77641844e+00, -1.32715114e+00,  6.82559678e-01],
[-2.40664733e-01, -1.42510981e+00,  4.20759810e+00],
[-1.36332845e-01,  6.89479767e-01, -9.42641570e-01],
[ 1.07108227e-01,  5.71642941e-01, -9.47404261e-01],
[-3.44996621e-01,  1.09408915e+00, -9.10389562e-01],
[-8.14490118e-01,  1.01526393e+00,  9.98392291e-01],
[-7.79712822e-01,  1.49388653e+00, -7.78556291e-01],
[-8.14490118e-01,  2.28587117e-01, -1.13498339e-01],
[ 1.39386818e+00,  5.64138506e-01, -4.52098819e-01],
[ 1.08087252e+00,  3.05215718e-02, -2.94683751e-02],
[-4.66717157e-01,  1.18964181e+00, -6.71242322e-01],
[ 2.29807788e+00,  6.44052153e-01, -9.58095094e-01],
[ 6.11379019e-01,  2.18990862e+00, -5.65427055e-01],
[-1.18944197e-01, -1.37698976e+00,  1.57973524e+00],
[ 5.49422828e-02, -8.75295322e-01,  9.49208746e-01],
[-9.18822006e-01,  5.09831597e-01, -7.66880593e-01],
[ 2.01649868e-02,  1.27167503e+00, -9.60708497e-01],
[-5.36271749e-01,  3.53750792e-01, -7.81379562e-01],
[-4.84105805e-01, -2.13515802e-01, -2.55696077e-01],
[-1.46123093e-02, -1.87308452e+00,  1.93847213e+00],
[-9.53599302e-01,  1.77023308e+00,  2.34671071e-01],
[-3.20009573e-02,  1.10895481e+00, -8.01784632e-01],
[-1.23181767e+00, -1.23692606e+00,  4.24387957e-01],
[-1.11009713e+00, -4.12743209e-04, -8.17405538e-01],
[-6.57992286e-01,  5.83902096e-01, -4.38400945e-01],
[-1.26659497e+00, -3.12075967e-01, -4.27480538e-01],
[-1.30137226e+00, -6.18927186e-01, -6.08092386e-01],
[-1.18944197e-01, -1.53337991e+00,  5.01805659e-01],
[-4.84105805e-01,  5.27189185e-01,  5.82468968e-01],
[-4.66717157e-01, -5.98476167e-01,  3.52194596e-01],
```

```
       [-7.97101470e-01,  1.32890352e+00,  1.13648006e+00],
       [ 1.76662819e-01,  1.17337265e+00, -8.62723773e-01],
       [-3.62385269e-01, -7.01590550e-01, -1.78112078e-01],
       [-4.93896053e-02, -7.57616032e-01,  3.49556923e+00],
       [ 6.98322259e-01, -8.40794968e-01,  7.67064195e-01],
       [ 1.28953629e+00, -2.28982960e-01, -8.94842671e-01],
       [ 8.20042795e-01,  1.09947401e+00, -9.77112163e-01],
       [ 2.35024382e+00,  1.16532400e+00, -4.44532129e-01],
       [-8.14490118e-01,  5.79090091e-01,  8.54201016e-01],
       [-9.01433358e-01,  1.88408854e+00, -8.24934871e-01],
       [ 4.20103891e-01, -1.16990171e+00,  1.39183429e+00],
       [-3.20009573e-02, -1.54197277e+00,  2.23502423e+00],
       [-3.44996621e-01, -4.95318819e-01, -1.70716296e-01],
       [ 8.37431443e-01,  2.00574376e-01, -9.19560868e-01],
       [-1.17965173e+00,  1.00830371e+00, -5.23684721e-01],
       [-1.33614956e+00, -2.22452382e-01,  2.76143746e-01],
       [ 7.23309308e-02, -2.73665859e-01, -9.16554530e-01],
       [ 1.34170224e+00, -8.73447856e-01, -4.29539001e-01],
       [-5.36271749e-01, -1.60384140e+00,  1.11100043e+00],
       [ 7.85265499e-01,  6.26210818e-01,  1.85493151e+00],
       [ 1.53297736e+00,  3.12090010e-02,  3.92547713e-02],
       [-2.92830677e-01,  6.19820273e-01, -6.95230582e-01],
       [-4.84105805e-01, -5.83868296e-01, -6.48146694e-01],
       [ 7.67876851e-01,  8.63112932e-01, -8.57726462e-01],
       [ 2.36763247e+00,  1.03485567e+00, -8.78029616e-01]])
```

[33]:
```python
df_scaled.std(axis=0)
```

[33]:
```
array([1., 1., 1.])
```

[34]:
```python
df_scaled.mean(axis=0)
```

[34]:
```
array([-3.13474736e-16, -1.26696039e-15,  3.93709342e-16])
```

[35]:
```python
from sklearn.cluster import KMeans

kmeans = KMeans(init="random", n_clusters=2, n_init=10, max_iter=300)
kmeans.fit(df_scaled)
```

[35]:
```
KMeans(init='random', n_clusters=2)
```

[36]:
```python
kmeans.cluster_centers_
```

[36]:
```
array([[-0.1824879 ,  0.4473393 , -0.48059893],
       [ 0.38898736, -0.95353903,  1.02443457]])
```

[37]:
```python
kmeans.inertia_
```

[37]: 239.20390789131238

[38]:
```
kmeans.n_iter_
```

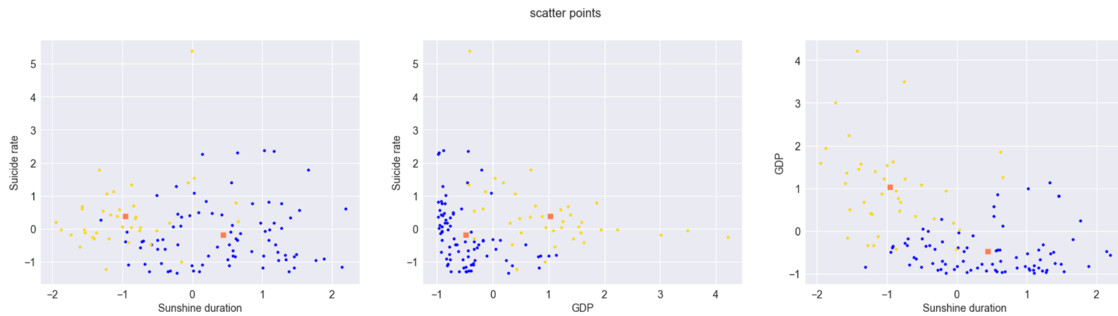
[38]: 5

[39]:
```
kmeans.labels_
```

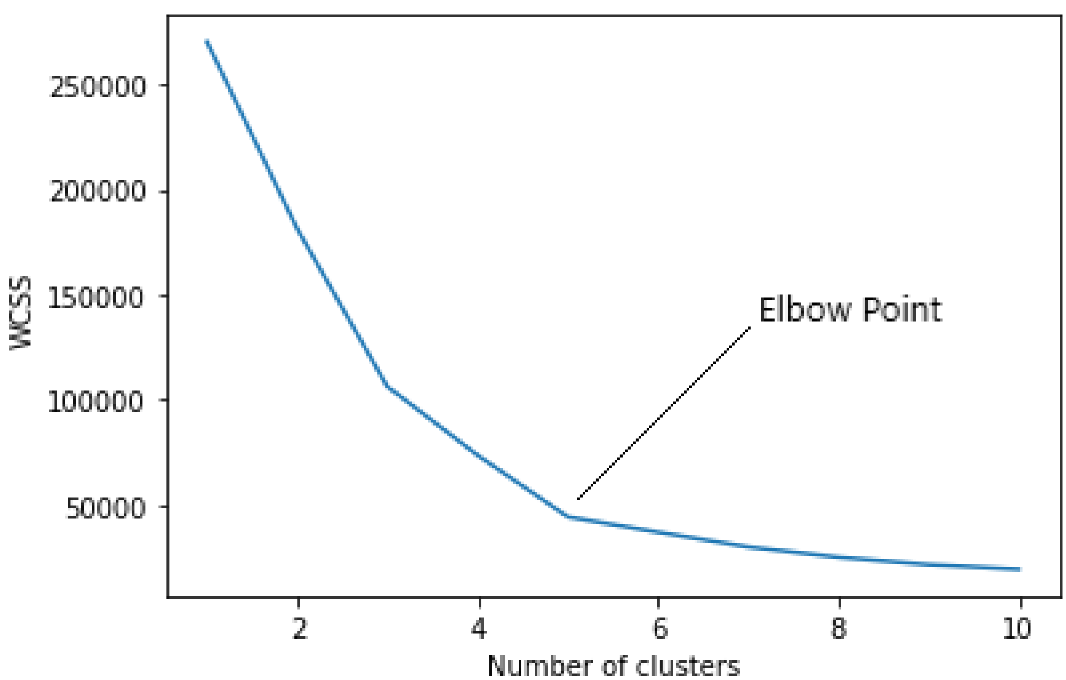
[39]:
```
array([0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 0, 0])
```

[40]:
```
s0 = (kmeans.labels_==0)
s1 = (kmeans.labels_==1)

fig, axes = plt.subplots(1,3, figsize = (26, 6))

fig.suptitle('scatter points')

# column 0 is suicide rate.
# column 1 is yearly sunshine duration
# column 2 is GDP PPP

axes[0].scatter(df_scaled.T[1][s0], df_scaled.T[0][s0], marker='.', color =␣
 ↪'blue')
axes[0].scatter(df_scaled.T[1][s1], df_scaled.T[0][s1], marker='.', color =␣
 ↪'gold')
axes[0].scatter(kmeans.cluster_centers_.T[1], kmeans.cluster_centers_.T[0],␣
 ↪marker='s', color = 'coral')
axes[0].set_ylabel('Suicide rate')
axes[0].set_xlabel('Sunshine duration')

axes[1].scatter(df_scaled.T[2][s0], df_scaled.T[0][s0], marker='.', color =␣
 ↪'blue')
axes[1].scatter(df_scaled.T[2][s1], df_scaled.T[0][s1], marker='.', color =␣
 ↪'gold')
axes[1].scatter(kmeans.cluster_centers_.T[2], kmeans.cluster_centers_.T[0],␣
 ↪marker='s', color = 'coral')
axes[1].set_ylabel('Suicide rate')
axes[1].set_xlabel('GDP')

axes[2].scatter(df_scaled.T[1][s0], df_scaled.T[2][s0], marker='.', color =␣
 ↪'blue')
```

```
axes[2].scatter(df_scaled.T[1][s1], df_scaled.T[2][s1], marker='.', color =␣
↪'gold')
axes[2].scatter(kmeans.cluster_centers_.T[1], kmeans.cluster_centers_.T[2],␣
↪marker='s', color = 'coral')
axes[2].set_ylabel('GDP')
axes[2].set_xlabel('Sunshine duration')

plt.show()
```



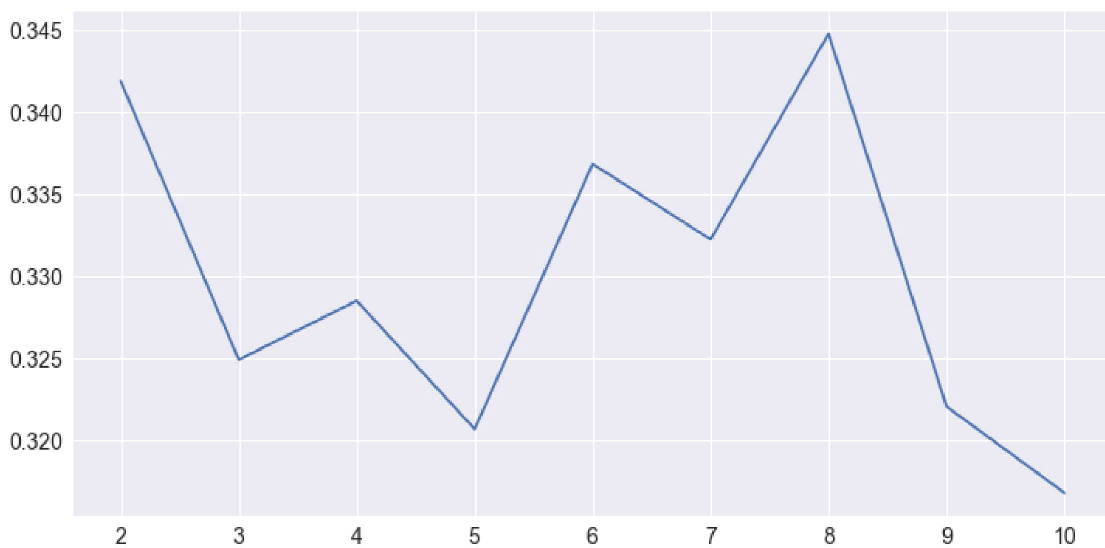
For a given group of data, if we don't have any previous knowledge about it, how many clusters best fit is not clear. There is a method called elbow method to decide how many clusters would possibly best fit. For this mothod, we try different amount of clusters, and select one as best. (WCSS is within cluster sum of squares)



Another way is to use silhouette_score which scikit learn has a function to give it. using different

clusters to run the K-means and get silhouette_score for each, then choose the highest score and that cluster number can be taken as the best choice.
scikit-learn silhouette_score is calcultion of average silhouette_coefficient of all the samples. For each specific sample point, the silhouette_coefficient is calculated as: $\frac{(b-a)}{max(a,b)}$, where a is mean intra-cluster distance for the sample, and b is mean nearest cluster distance for the sample.

[41]:
```
from sklearn.metrics import silhouette_score

silhouette_coefficients = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, init="random", n_init=10, max_iter=300)
    kmeans.fit(df_scaled)
    score = silhouette_score(df_scaled, kmeans.labels_)
    silhouette_coefficients.append(score)

plt.plot(range(2, 11), silhouette_coefficients)
plt.show()
```



For our data this shows that 2 clusters and 8 cluster is high possibly good choices. But all the values actual difference is not large, so this also an evidence that the whole dataset not showing clear clustering.
But if we have some knowledge and understandings of the data, we can set the clustering from our understandings. Silhouette score is not always good.
Sure there are also many other metrics to evaluate the clustering is good or not so good.

1.4 Hierarchical clustering

There are mainly two types of hierarchical clustering:

1. Agglomerative hierarchical clustering
2. Divisive Hierarchical clustering

The agglomerative method will take all data record (point) as separate clusters (thus each cluster has only one point). Then perform compare and make the most close 2 clusters into one cluster. Through doing this repetitively, finally makes the whole set into one cluster.
We can decide how many cluster with the decision of what is rule of disimilarity between clusters, together with the dendrogram.
The divisive method start from all points belong to one big cluster, through repetively divide with the disimilarity rules, until finally each point is one cluster.
For the agglomerative method, the importance is how we decide the distance between different clusters, and choose 2 nearest cluster to merge into 1.
For the divisive method, the importance is how we decide the distance inside one cluster, and find the cluster with the biggest internal distance and split it into 2.
In most case we use the agglomerative method to perform hierarchical clustering.

[42]:
```
file_path = pathlib.Path("D:/Edu/data_resource/data-UCI ml/Wholesale customers␣
↪data.csv")

df_b = pd.read_csv(file_path)
df_b.head()
```

[42]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

[43]:
```
df_b.shape
```

[43]:
```
(440, 8)
```

[44]:
```
df_b.dtypes
```

[44]:
```
Channel             int64
Region              int64
Fresh               int64
Milk                int64
Grocery             int64
Frozen              int64
Detergents_Paper    int64
Delicassen          int64
dtype: object
```

[45]:
```
df_b.isna().any()
```

[45]:
```
Channel             False
Region              False
Fresh               False
Milk                False
Grocery             False
Frozen              False
Detergents_Paper    False
Delicassen          False
dtype: bool
```

[46]:
```
df_c = df_b[['Fresh','Milk','Grocery','Frozen','Detergents_Paper','Delicassen']]
df_c
```

[46]:
```
     Fresh   Milk  Grocery  Frozen  Detergents_Paper  Delicassen
0    12669   9656     7561     214              2674        1338
1     7057   9810     9568    1762              3293        1776
2     6353   8808     7684    2405              3516        7844
3    13265   1196     4221    6404               507        1788
4    22615   5410     7198    3915              1777        5185
..     …      …       …       …                 …           …
435  29703  12051    16027   13135               182        2204
436  39228   1431      764    4510                93        2346
437  14531  15488    30243     437             14841        1867
438  10290   1981     2232    1038               168        2125
439   2787   1698     2510      65               477          52

[440 rows x 6 columns]
```

1.4.1 using SciPy

[47]:
```
df_cw = sc.vq.whiten(df_c)
```

[48]:
```
df_cw
```

[48]:
```
array([[1.00285375, 1.3098235 , 0.79653552, 0.04413141, 0.56147767,
        0.47499033],
       [0.55861859, 1.33071339, 1.00796876, 0.36336239, 0.69145325,
        0.63048044],
       [0.5028913 , 1.19479343, 0.80949331, 0.49596285, 0.73827805,
        2.78462194],
       …,
       [1.1502461 , 2.10092651, 3.18603672, 0.09011882, 3.11626408,
        0.66278546],
       [0.81453667, 0.26872   , 0.23513653, 0.21405798, 0.03527608,
        0.75437552],
       [0.22061358, 0.23033143, 0.26442324, 0.0134044 , 0.10015888,
        0.01846001]])
```

[49]:
```
df_cw.mean(axis=0)
```

[49]:
```
array([0.94992056, 0.78625573, 0.83765041, 0.63349857, 0.60504641,
       0.54132939])
```

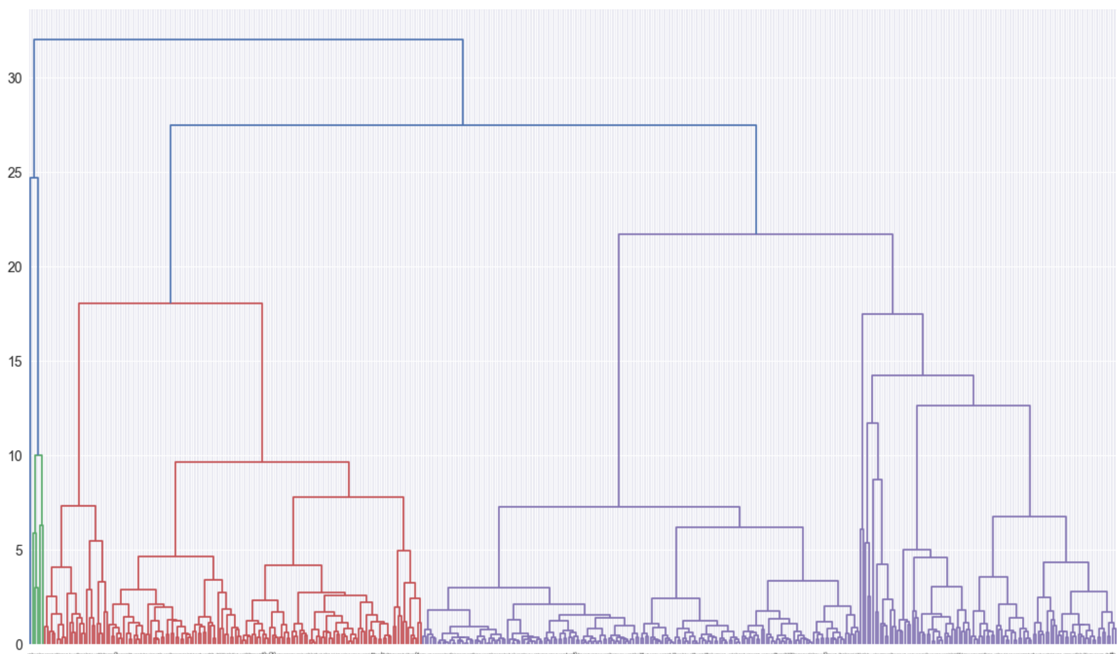
[50]:
```
df_cw.std(axis=0)
```

[50]:
```
array([1., 1., 1., 1., 1., 1.])
```

[51]:
```
import scipy.spatial.distance as ssd

dist_cw = ssd.pdist(df_cw)
dist_cw
```

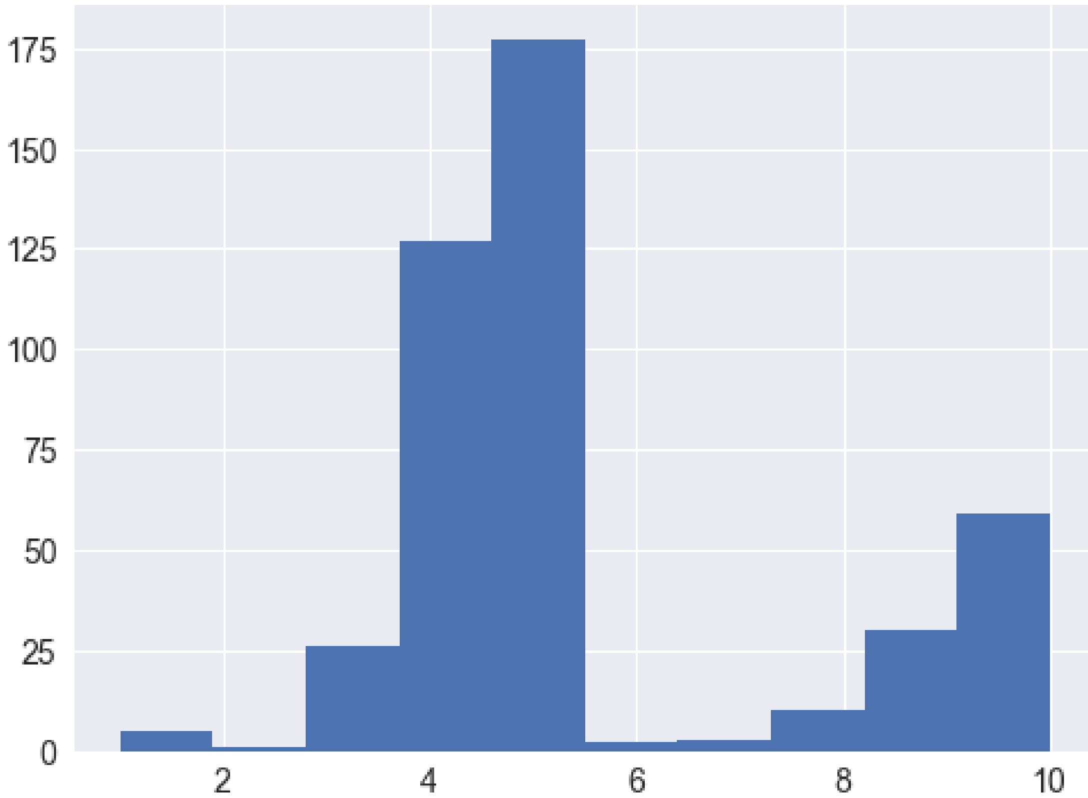
[51]:
```
array([0.62085789, 2.41519574, 1.81797539, …, 4.65765286, 4.73466988,
       0.97011304])
```

[52]:
```
hc_result = sc.hierarchy.linkage(dist_cw,method='ward')

fig, ax = plt.subplots(1, figsize=(20,12))
sc.hierarchy.dendrogram(hc_result)
plt.show()
```



[53]:
```
cw_cluster = sc.hierarchy.fcluster(hc_result, 10, criterion='maxclust')

fig, ax = plt.subplots(1, figsize=(8,6))
```

```
ax.hist(cw_cluster)
plt.show()
```



1.4.2 Using sklearn

[54]:
```
from sklearn.preprocessing import MinMaxScaler
# using MinMaxScaler is only for illustration purpose, show how to use this␣
↪function.
# normally unless you quite sure the range of data is more important and has␣
↪obvious meaning in your analysis then you use it.
# otherwise most of the case use standard scaler, which scales the standard␣
↪deviation to 1 for all features.
scaler = MinMaxScaler().fit(df_c)
scaler
```

[54]: MinMaxScaler()

[55]:
```
d_scaled = scaler.transform(df_c)
d_scaled
```

[55]: array([[0.11294004, 0.13072723, 0.08146416, 0.0031063 , 0.0654272 ,
         0.02784731],
        [0.06289903, 0.13282409, 0.10309667, 0.02854842, 0.08058985,
         0.03698373],
        [0.05662161, 0.11918086, 0.08278992, 0.03911643, 0.08605232,
         0.16355861],
        …,
        [0.1295431 , 0.21013575, 0.32594285, 0.00677142, 0.36346267,
         0.03888194],
        [0.091727 , 0.02622442, 0.02402535, 0.01664914, 0.00404174,
         0.04426366],
        [0.02482434, 0.02237109, 0.02702178, 0.00065742, 0.01161082,
         0.00102211]])

[56]:
```
d_scaled.mean(axis=0)
```

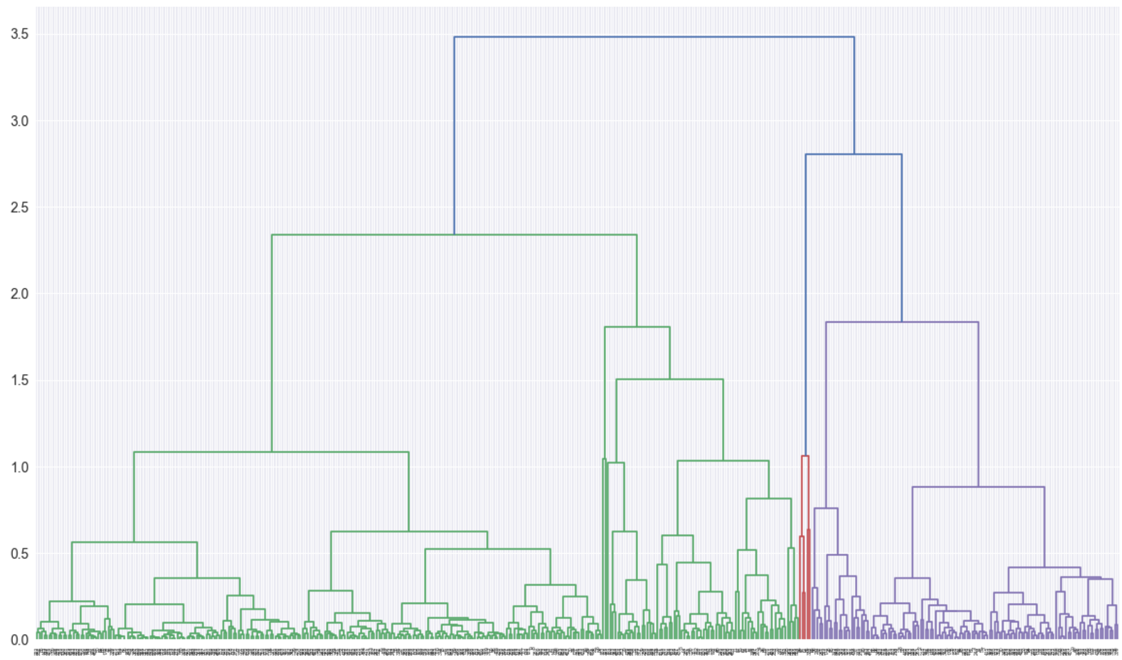
[56]: array([0.10697737, 0.07817309, 0.08567077, 0.05007777, 0.07050983,
       0.03174532])

[57]:
```
d_scaled.std(axis=0)
```

[57]: array([0.11264533, 0.10037697, 0.10231369, 0.07969814, 0.11665769,
       0.05875885])

[58]:
```
from sklearn.cluster import AgglomerativeClustering

def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    # Plot the corresponding dendrogram
    fig, ax = plt.subplots(1, figsize=(20,12))
    sc.hierarchy.dendrogram(linkage_matrix, **kwargs)
```

```
clustering = AgglomerativeClustering(distance_threshold=0, n_clusters=None,␣
 ↪linkage='ward').fit(d_scaled)
plot_dendrogram(clustering)
plt.show()
```



[59]:
```
clustering.labels_
```

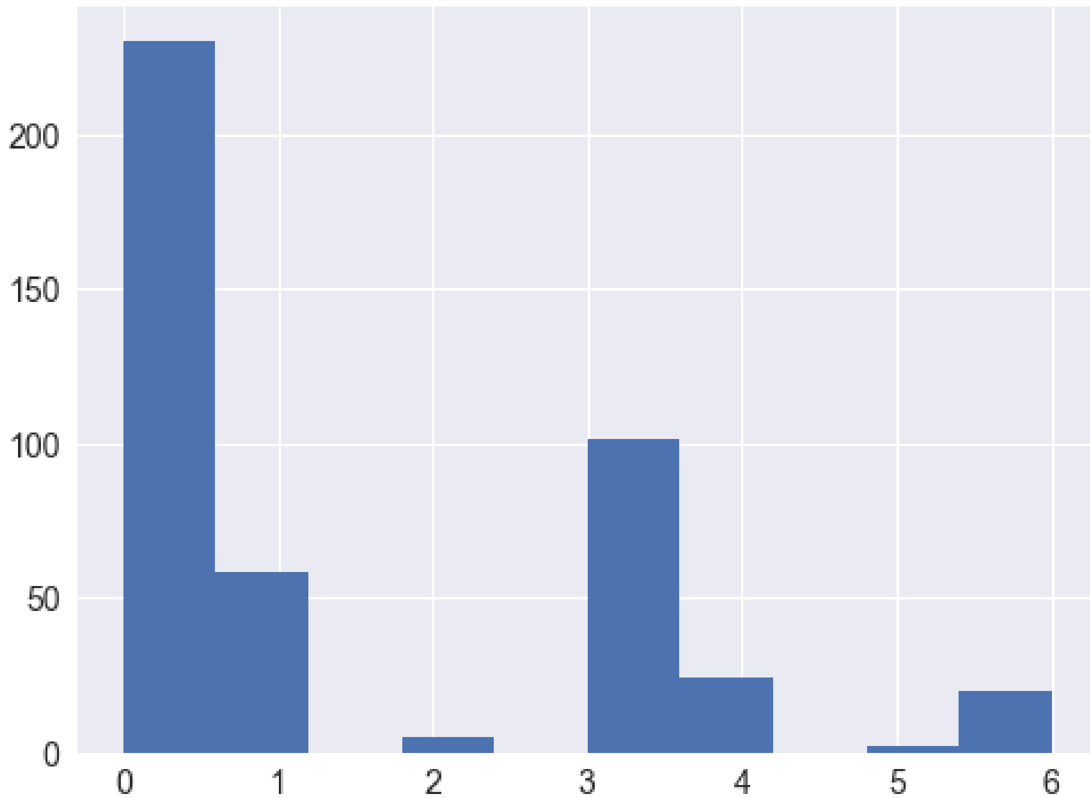
[59]:
```
array([236, 315, 407, 226, 291, 296, 325, 370, 386, 413, 237, 399, 221,
       429, 369, 305, 371, 229, 273, 247, 316, 402, 289, 351, 383, 425,
       417, 373, 381, 311, 418, 392, 372, 359, 424, 262, 323, 415, 287,
       295, 261, 244, 389, 404, 292, 299, 285, 343, 427, 423, 321, 306,
       319, 260, 365, 346, 263, 245, 431, 279, 270, 248, 393, 304, 387,
       297, 366, 230, 238, 380, 255, 322, 309, 223, 239, 320, 419, 337,
       385, 334, 433, 358, 256, 428, 395, 127, 327, 335, 241, 288, 374,
       280, 283, 432, 377, 341, 430, 439, 406, 349, 147, 336, 347, 219,
       376, 330, 298, 264, 375, 312, 408, 227, 211, 318, 436, 249, 410,
       155, 130, 354, 396, 344, 422, 326, 352, 139, 420, 290, 268, 246,
       357, 388, 251, 340, 438, 426, 233, 145, 250, 272, 391, 403, 152,
       187, 148, 367, 384, 171, 362, 379, 400, 203, 201, 434, 409, 437,
       267, 416, 414, 405, 144, 274, 314, 124, 252, 125, 368, 353, 216,
       397, 317, 435, 293, 277, 363, 282, 281, 182, 329, 235, 300, 193,
       331, 324, 275, 364, 160, 398, 269, 234, 158, 186, 209, 117, 412,
       257, 276, 350, 310, 212, 164, 185, 286, 355, 284, 217, 214, 183,
       258, 220, 313, 192, 161, 176, 345, 154, 339, 390, 243, 361, 194,
```

```
       254, 143, 162, 202, 159, 333, 105, 228, 184, 204, 195, 394, 196,
       122, 328, 169, 134, 207,  80, 303, 224, 109, 114, 121, 178,  96,
       136, 197,  73, 271, 215, 356, 157, 118, 307, 191, 301, 111, 253,
       222,  92, 294, 348, 135, 378, 180, 232, 225, 123,  91, 173,  79,
       140, 189, 360, 163, 100, 175,  95, 208, 172, 112,  55, 113, 131,
       181,  47, 167,  58, 199, 242, 240, 177, 110, 302, 198, 174, 308,
       218, 101, 142, 150,  54, 342, 259, 146, 188, 129,  88, 338,  90,
        61, 265,  60, 153, 266,  76, 126, 119,  97, 206, 132, 411,  81,
        71, 213, 107, 166, 179, 168, 205, 401, 382, 231, 141, 156, 421,
       128, 108,  53,  72, 115, 151, 102,  45, 103, 332, 170, 120,  35,
        85,  62,  65,  87,  56,  59,  77, 200,  89, 165, 210, 133,  70,
       190,  99,  27,  64,  93,  49,  75,  98,  83,  44,  94,  48,  67,
        69, 149,  39,  43,  37, 106, 137,  50, 104,  82, 116,  26,  24,
        63,  29,  23,  34,  74,  21,  86,  36,  78,  40,  52,  17,  84,
        46, 278,  31,  57,  41,  11, 138,  30,  51,  42,  38,  25,  28,
        18,   8,  22,  19,  12,  13,  10,  68,  33,  14,  66,  20,   9,
         4,  32,  15,   6,   5,  16,   7,   3,   1,   2,   0], dtype=int64)
```

[60]:
```
clustering = AgglomerativeClustering(distance_threshold=1.5, n_clusters=None).
 ↪fit(d_scaled)
clustering.labels_
```

[60]:
```
array([0, 3, 3, 1, 1, 0, 0, 3, 3, 3, 3, 0, 1, 1, 1, 0, 3, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 4, 6, 0, 0, 0, 1, 0, 3, 1, 3, 3, 6, 1, 0, 3, 3,
       3, 4, 3, 2, 3, 4, 0, 3, 6, 3, 0, 3, 4, 3, 0, 3, 0, 2, 3, 3, 0, 4,
       3, 1, 0, 0, 1, 1, 0, 1, 3, 0, 0, 4, 0, 0, 0, 3, 3, 0, 0, 2, 2, 1,
       0, 1, 0, 1, 4, 1, 3, 0, 3, 0, 0, 0, 3, 3, 3, 6, 0, 0, 3, 3, 0, 3,
       0, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 6, 1, 0, 0, 6, 0, 0,
       0, 0, 0, 0, 3, 0, 0, 0, 0, 1, 1, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 4, 3, 0, 3, 3, 3, 0, 0, 4, 3, 3, 3, 0, 0, 0, 3, 4, 3, 3, 0, 3,
       6, 0, 0, 0, 0, 6, 3, 5, 0, 0, 0, 3, 3, 3, 1, 0, 0, 3, 0, 1, 1, 3,
       0, 0, 3, 4, 1, 0, 0, 3, 0, 3, 3, 4, 0, 4, 0, 3, 3, 3, 4, 0, 3, 0,
       0, 3, 0, 0, 0, 0, 0, 0, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 6, 1, 0,
       0, 0, 3, 3, 0, 0, 0, 0, 0, 4, 0, 1, 1, 0, 0, 3, 6, 6, 0, 0, 0, 3,
       3, 1, 3, 0, 3, 0, 0, 0, 0, 6, 0, 0, 1, 1, 0, 0, 0, 0, 6, 1, 6, 6,
       0, 1, 0, 6, 0, 0, 0, 3, 0, 3, 0, 0, 3, 0, 0, 3, 3, 3, 4, 3, 3, 0,
       0, 3, 1, 1, 4, 0, 0, 3, 0, 0, 0, 4, 0, 0, 0, 0, 0, 5, 0, 0, 1, 0,
       0, 4, 0, 2, 1, 1, 0, 0, 1, 0, 3, 3, 3, 4, 0, 3, 3, 1, 0, 4, 0, 4,
       0, 3, 1, 0, 0, 3, 1, 0, 0, 0, 0, 3, 0, 3, 0, 0, 0, 0, 6, 1, 0, 0,
       0, 0, 3, 6, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       3, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 3, 0, 0, 3, 0, 3, 1, 0, 3, 3, 3,
       3, 0, 3, 0, 0, 0, 0, 1, 3, 1, 0, 0, 3, 1, 0, 0, 0, 1, 6, 4, 0, 0],
      dtype=int64)
```

[61]:
```
fig, ax = plt.subplots(1, figsize=(8,6))
ax.hist(clustering.labels_)
plt.show()
```


200
150
100
50
0
0
1
2
3
4
5
6

```
[ ]:
```