HW2 0416246 王彥茹

Spatial clustering

Import the location file by pandas.read_csv()

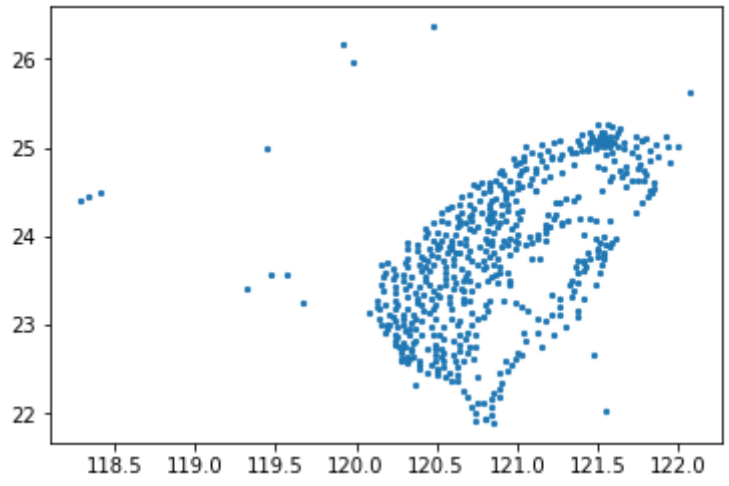
```
import numpy as np
import pandas as pd

file = 'location.csv'

raw_data = pd.read_csv(file,header=0, delimiter=',')
raw_data.columns = ['id', 'name', 'altitude', 'x', 'y','city', 'address' , 'A', 'B', 'C', 'D', 'E', 'F']

data = raw_data[['x', 'y']].copy()
```

	站號	站名	海拔高度(m)	經度	緯度	城市	地址	資料起始日期	撤站日期	備註	原站號	新站號	Unnamed: 12
0	466850	五分山雷達站	756.0	121.7812	25.0712	新北市	瑞芳區靜安路四段1巷1號	1988/07/01	NaN	本站只有雷達觀測資料。	NaN	NaN	NaN
1	466880	板橋	9.7	121.4420	24.9976	新北市	板橋區大觀路二段265巷62號	1972/03/01	NaN	原為探空站，自2002年開始進行氣象觀測。	NaN	NaN	NaN
2	466900	淡水	19.0	121.4489	25.1649	新北市	淡水區中正東路42巷6號	1942/01/01	NaN	NaN	NaN	NaN	NaN
3	466910	鞍部	825.8	121.5297	25.1826	臺北市	北投區陽明山竹子湖路111號	1937/01/01	NaN	NaN	NaN	NaN	NaN

extract only latitude and longitude data to do the clustering

Original data

```
import matplotlib.pyplot as plt

plt.scatter(raw_data['x'], raw_data['y'], s=5)
plt.show()
```



	x	y
0	121.7812	25.0712
1	121.4420	24.9976
2	121.4489	25.1649
3	121.5297	25.1826
4	121.5149	25.0377
5	121.5445	25.1621
6	121.7405	25.1333
7	122.0797	25.6280
8	121.6133	23.9751
9	121.0475	25.0067
10	121.8574	24.5967
11	121.7565	24.7640
12	118.2893	24.4073

DBSCAN

```
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

###DBSCAN###
dbsc = DBSCAN(eps = 0.1, min_samples = 8).fit(data) #(0.1, 8)
core_samples_mask = np.zeros_like(dbsc.labels_, dtype=bool)
core_samples_mask[dbsc.core_sample_indices_] = True
labels = dbsc.labels_

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

####draw figure####
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = data[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

    xy = data[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=3)
```

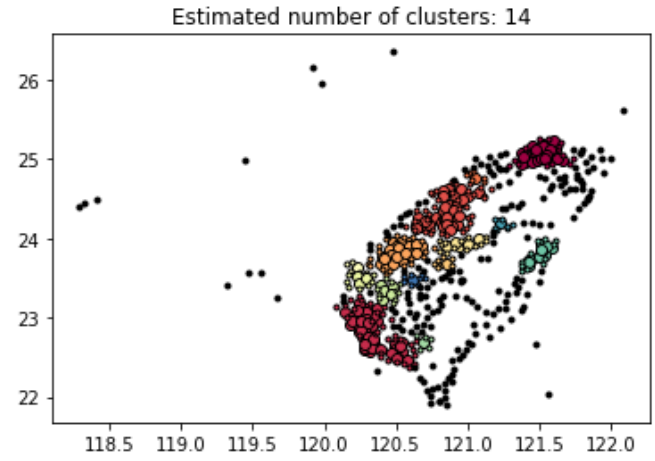
DBSCAN(eps = 0.1, min_samples = 8).fit(data)
eps: The maximum distance between two samples for them to be considered as in the same neighborhood
min_samples: The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself

observation:

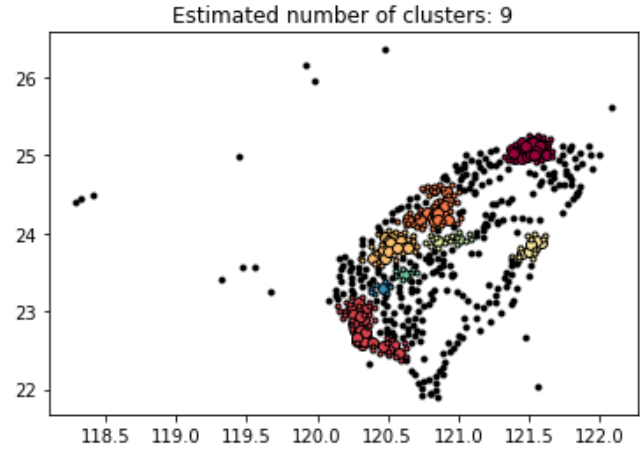
1. Since the data points are pretty close to each other, if I set the **eps** too small, it will cause the cluster can't contain too many data and therefore too many clusters.(一個群的範圍太小，分成太多群)

2. min_smaples cannot set too big, or the condition to become a cluster will be too hard to accomplish. Not only too hard to form a cluster but also consider too many data points to be noises.

   (不容易形成 cluster，且把太多 data 視作 noise)



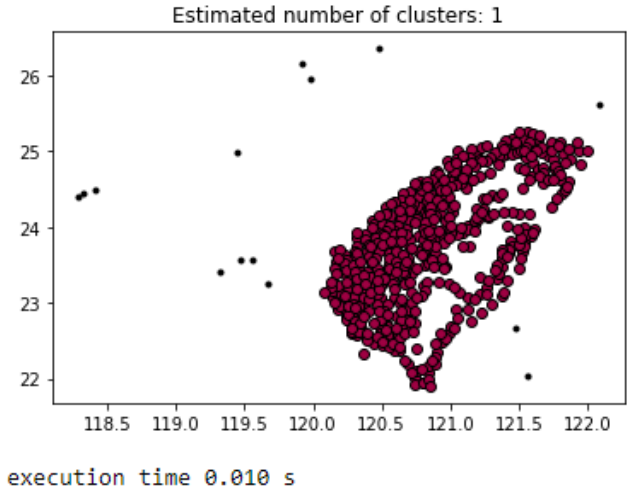
(eps=0.1, min_samples = 8)



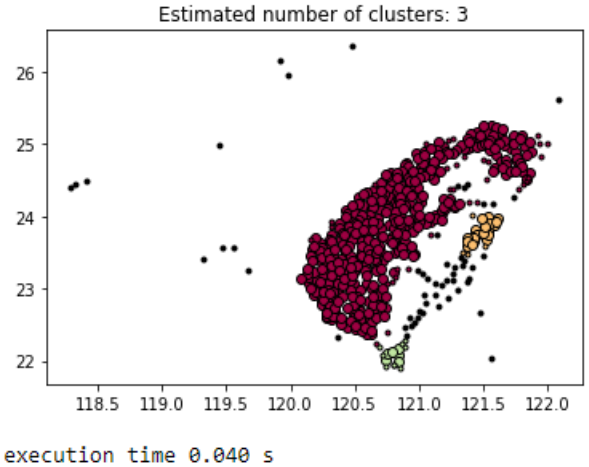
(eps=0.1, min_samples = 9)

本來打算參考計算出來一個縣市大約的距離作為 eps (大概 0.3~5)，

套用結果發現會讓整個台灣西部都變成同一個 cluster，效果不佳。



(eps=0.3, min_samples = 8)



(eps=0.15, min_samples = 8)

右圖是我在測試大概要多少 eps 才不會讓整個台灣幾乎只有一個 cluster，測試數值的時候發現只要 eps 稍微設大一點，整個台灣西部就會只剩一個 cluster

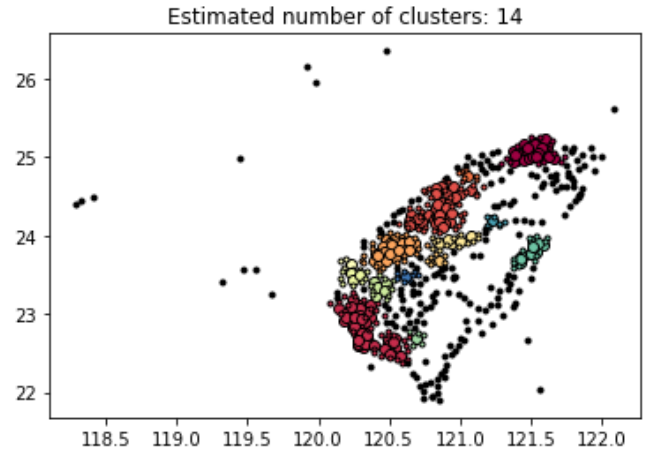
最後使用 eps=0.1,

min_samples = 8，得出的結

果最讓我滿意，大致上能分

出台灣的北中南部以及東部

地區。



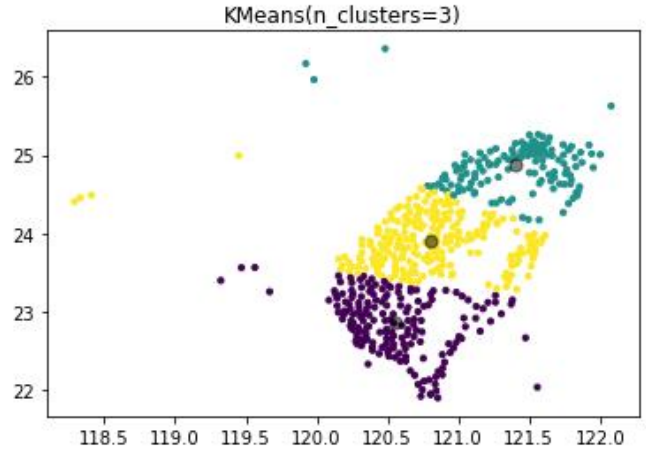
execution time 0.047 s

K-means

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=10)
kmeans.fit(data)

labels = kmeans.predict(data)
centroids = kmeans.cluster_centers_

plt.scatter(data[:, 0], data[:, 1], c=labels, s=10, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=50, alpha=0.5)
```

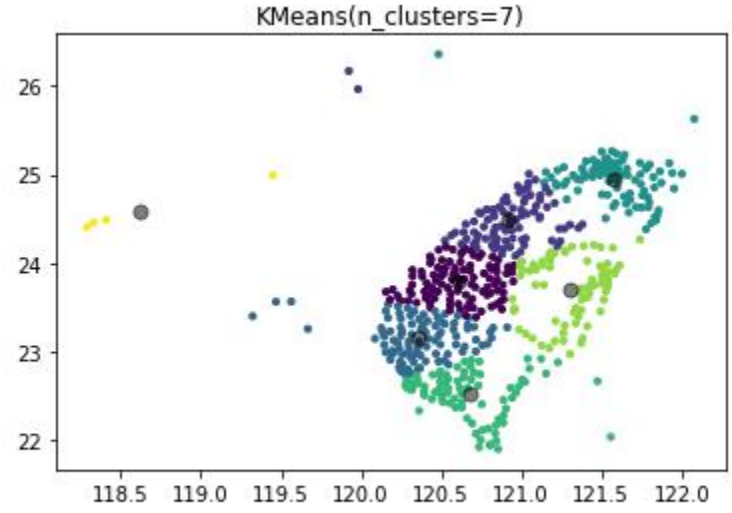
KMenas(n_clusters = N) 單純依據想要分幾個 cluster 設 N 值

observation:



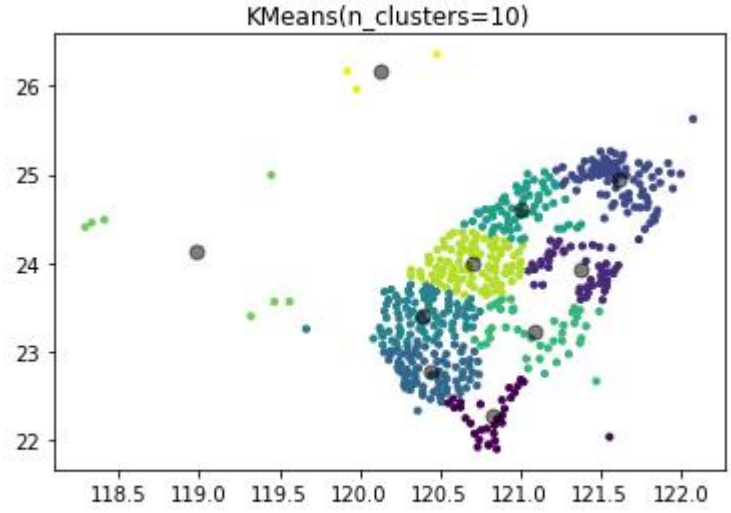
execution time 0.264 s

設 N = 3 就可以簡單分成北中南



execution time 0.300 s

N = 7，即可分出北北基/桃竹苗/彰化台中/雲嘉南/高屏/東部地區



execution time 0.408 s

N = 10， 還可分出花東地區，亦區分出金門/澎湖

Comparison:

由於 DBSACN 是 density-based clustering，當面對這種資料遍布平均又密集時，用 density 來做 clustering 時，就要特別注意參數的設定，才能得到比較合理又能讓人接受的結果。

另外，因為 k-means 會視相鄰的資料集為同一個 cluster，以這類型的 data 來講，用 k-means 來做 clustering 會比較方便，只要設想要分成幾個 cluster，就會在適當的地方有 centroids，將台灣分成不同地區。

不過就 execution time 來看，DBSCAN 確實比 k-means 來的快。

Temporal clustering

Do clustering with north power usage and Taipei's temperature
Extract csv file from MySQL (use inner join)
Use temperature and taipower data from 2016/10/01 to 2017/06/30
Choose 12 hours(8:00 to 19:00) in one day as a data point
Choose TAIPEI to represent north temperature

```
select north_temp , north_power_usage,  w_Time, w_date, w_hour
from
(select w.obsTime as w_Time, date(w.obsTime) as w_date, hour(w.obsTime) as w_hour, w.value as north_temp
 from weather w
 where date(w.obsTime) != '0000-00-00'
 && date(w.obsTime) >= date'2016-10-01'
 && date(w.obsTime) <= date'2017-06-30'
 && hour(w.obsTime) >= 8
 && hour(w.obsTime) <= 19
 && w.locationName='TAIPEI,臺北') t1

inner join

 (select date(create_date) as p_date, hour(p.create_time) as p_hour, p.north_usage as north_power_usage
  from taipower p
  where date(p.create_date) >= date'2016-10-01'
  && date(p.create_date) <= date'2017-06-30'
  && hour(p.create_time) >= 8
  && hour(p.create_time) <= 19) t2
  on t1.w_date = t2.p_date && t1.w_hour = t2.p_hour;
```

Import data by pandas.read_csv()

因為套用 sklearn.cluster 需要 float type 的 data，將 DATE 改成跟 2016-10-01 的天數差距，方便接下來做 clustering

```
import numpy as np
import pandas as pd

file = 'q2.csv'

raw_data = pd.read_csv(file,header=0, delimiter=',')
raw_data[['north_temp', 'north_power_usage']] = raw_data[['north_temp', 'north_power_usage']].apply(pd.to_numeric)
raw_data['w_Time'] = raw_data['w_Time'].apply(pd.to_datetime)
raw_data['w_date'] = raw_data['w_date'].apply(pd.to_datetime)
#raw_data['w_date'] = raw_data['w_date'].apply(pd.to_timedelta)
raw_data['time_delta'] = raw_data['w_date'] - pd.datetime(2016,10,1)
raw_data['index'] = raw_data['time_delta'].dt.total_seconds() / (24 * 60 * 60)
raw_data
```

	north_temp	north_power_usage	w_Time	w_date	w_hour	time_delta	index
0	29.0	852.0	2016-10-01 08:00:00	2016-10-01	8	0 days	0.0
1	30.5	942.5	2016-10-01 09:00:00	2016-10-01	9	0 days	0.0
2	31.4	1008.1	2016-10-01 10:00:00	2016-10-01	10	0 days	0.0
3	32.2	1062.0	2016-10-01 11:00:00	2016-10-01	11	0 days	0.0
4	32.6	1088.3	2016-10-01 12:00:00	2016-10-01	12	0 days	0.0
5	31.8	1080.8	2016-10-01 13:00:00	2016-10-01	13	0 days	0.0
6	31.9	1099.4	2016-10-01 14:00:00	2016-10-01	14	0 days	0.0
7	31.8	1084.5	2016-10-01 15:00:00	2016-10-01	15	0 days	0.0

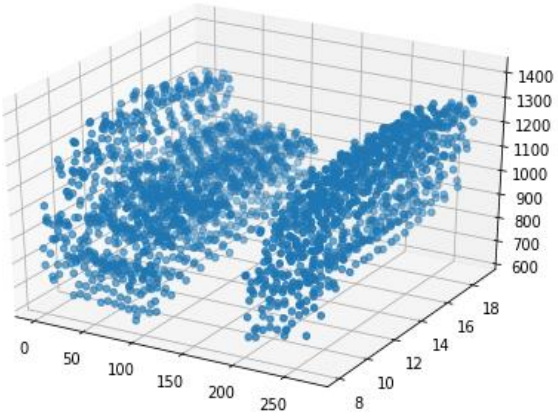
Original north power usage figure

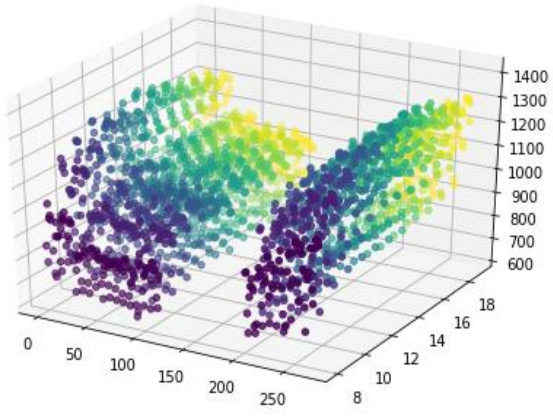
X 軸為日期(0 ~ 272 days represent from 2016-10-30 to 2017-06-30)

Y 軸為時間(早上 8 點至晚上 7 點 共計 12 個小時)

Z 軸為 power usage

右圖為視覺上方便利用顏色區分出不同時間(一個小時為一個顏色)





```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
power_data = raw_data[['index','w_hour','north_power_usage']].as_matrix()

fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(raw_data['index'], raw_data['w_hour'], raw_data['north_power_usage'], c=raw_data['w_hour'])

plt.show()
```

Original Taipei's temperature figure

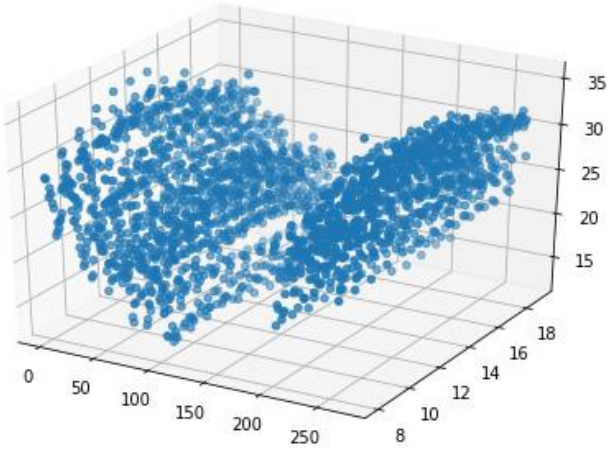
X 軸為日期(0 ~ 272 days represent from 2016-10-30 to 2017-06-30)

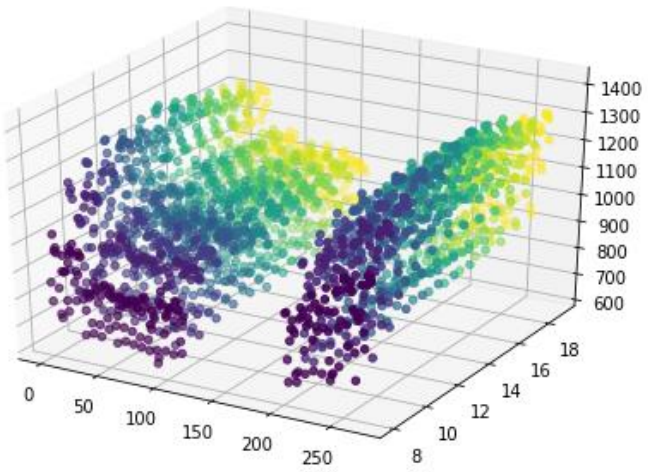
Y 軸為時間(早上 8 點至晚上 7 點 共計 12 個小時)

Z 軸為 Taipei's temperature

右圖為視覺上方便利用顏色區分出不同時間(一個小時為一個顏色)

```
power_data = raw_data[['index','w_hour','north_temp']].as_matrix()

fig = plt.figure()
ax = Axes3D(fig)

#ax.scatter(raw_data['index'], raw_data['w_hour'], raw_data['north_power_usage'], c=raw_data['w_hour'])
ax.scatter(raw_data['index'], raw_data['w_hour'], raw_data['north_temp'])

plt.show()
```





K means

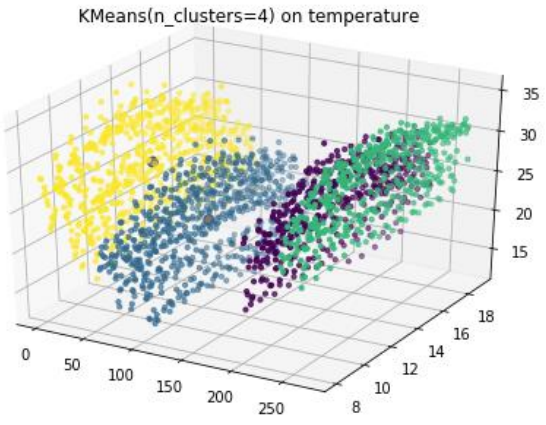
k-means on north power usage

```
import time
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
power_data = raw_data[['index','w_hour','north_power_usage']].as_matrix()

start = time.time()
kmeans = KMeans(n_clusters=4)
kmeans.fit(power_data)
end = time.time()

labels = kmeans.predict(power_data)
centroids = kmeans.cluster_centers_

fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(power_data[:, 0], power_data[:, 1], power_data[:, 2], c=labels, s=10, cmap='viridis')

centers = kmeans.cluster_centers_
ax.scatter(centers[:, 0], centers[:, 1], centers[:, 2], c='black', s=50, alpha=0.5)


plt.title('KMeans(n_clusters=4) on power data')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```

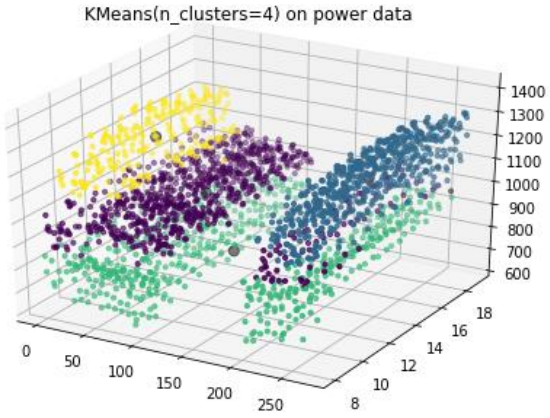
k-means on Taipei's temperature

```
temp_data = raw_data[['index','w_hour','north_temp']].as_matrix()

start = time.time()
kmeans = KMeans(n_clusters=4)
kmeans.fit(temp_data)
end = time.time()

labels = kmeans.predict(temp_data)
centroids = kmeans.cluster_centers_

fig = plt.figure()
ax = Axes3D(fig)

ax.scatter(temp_data[:, 0], temp_data[:, 1], temp_data[:, 2], c=labels, s=10, cmap='viridis')

centers = kmeans.cluster_centers_
ax.scatter(centers[:, 0], centers[:, 1], centers[:, 2], c='black', s=50, alpha=0.5)


plt.title('KMeans(n_clusters=4) on temperature')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```

result figure(left is temperature, right is north usage)



execution time 0.331 s



execution time 0.379 s

observation :

I don’t know which clustering alogorithm to choose, I decided to use k-means to take a look at its cluster result since k-means is easy to implement.

We can tell by the figure above. The clustering result of temperature and power usage is not quite similar. (temperature data 偏直向分群，power usage data 偏橫向分群)

Hence, I try to seek for clustering algorithm that can make the result of two clustering more similar.

Gaussian Mixture

There are four kind of covariance type to choose from when implementing Gaussian Mixture algorithm.

1. full: each component has its own general covariance matrix
2. diag: each component has its own diagonal covariance matrix
3. spherical: each component has its own single variance
4. tied: all component share the same general covariance matrix

I try implementing different covariance type to compare the effects

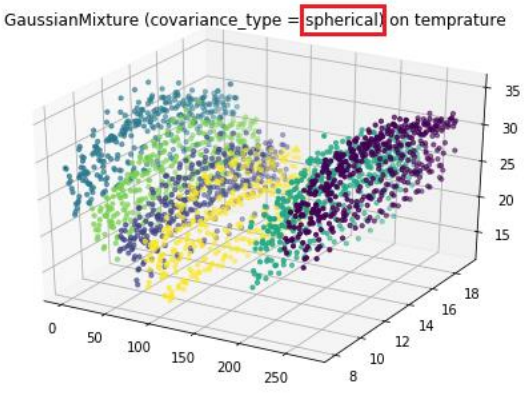
Gaussian Mixture on temperature data

```
from sklearn.mixture import GaussianMixture
start = time.time()
gmm = GaussianMixture(n_components=4, covariance_type='spherical').fit(temp_data)
end = time.time()

fig = plt.figure()
ax = Axes3D(fig)

labels = gmm.predict(temp_data)
ax.scatter(temp_data[:, 0], temp_data[:, 1], temp_data[:, 2], c=labels, s=10, cmap='viridis')

plt.title('GaussianMixture (covariance_type = spherical) on temprature')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```

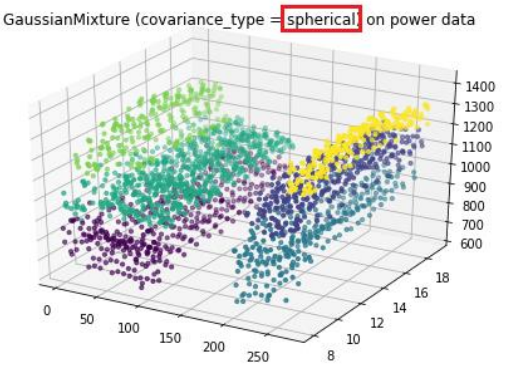
Gaussian Mixture on power data

```
from sklearn.mixture import GaussianMixture
start = time.time()
gmm = GaussianMixture(n_components=4, covariance_type='spherical').fit(power_data)
end = time.time()

fig = plt.figure()
ax = Axes3D(fig)

labels = gmm.predict(power_data)
ax.scatter(power_data[:, 0], power_data[:, 1], power_data[:, 2], c=labels, s=10, cmap='viridis')

plt.title('GaussianMixture (covariance_type = spherical) on power data')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```

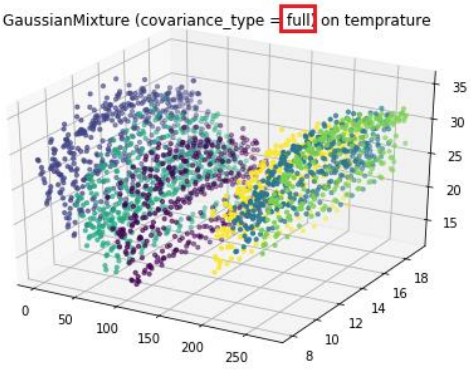
Originally, I set n_components to be 4 , but later on I realize that setting n_components to 6 can make the clustering result easier to compare.



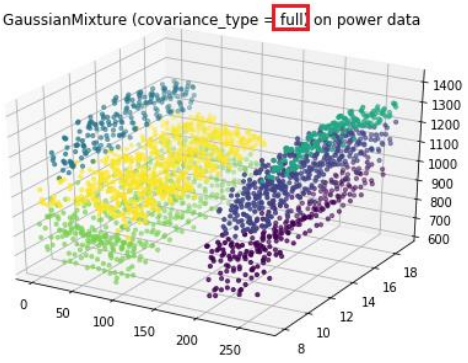
execution time 0.047 s



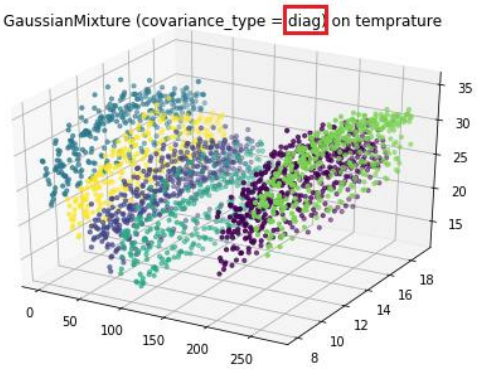
execution time 0.142 s



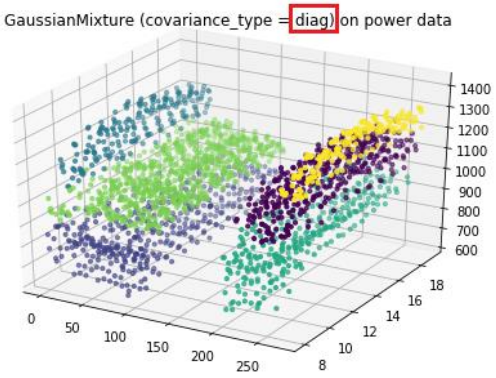
execution time 0.404 s



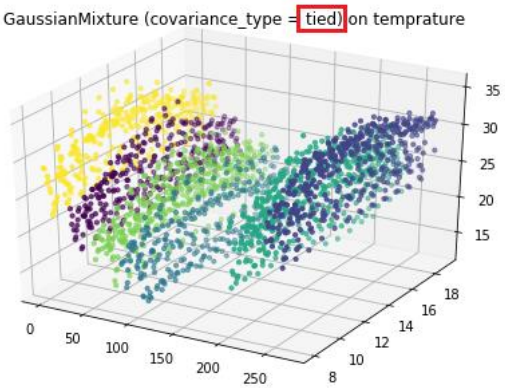
execution time 0.260 s



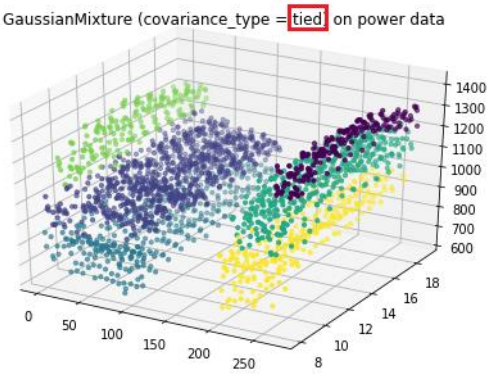
execution time 0.156 s



execution time 0.122 s



execution time 0.165 s



execution time 0.225 s

Observation:
The execution time depends on the chosen covariance type. For example, type **spherical** use only single variance rather than others using covariance matrix. Therefore, type **spherical** is the fastest. On the other hand, type **full** has the longest execution time since it assume that each component has its own general covariance matrix.

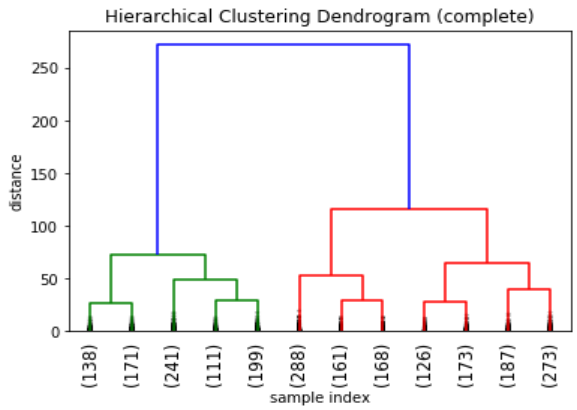
AgglomerativeClustering

Three kinds of linkage according to the distance between two points

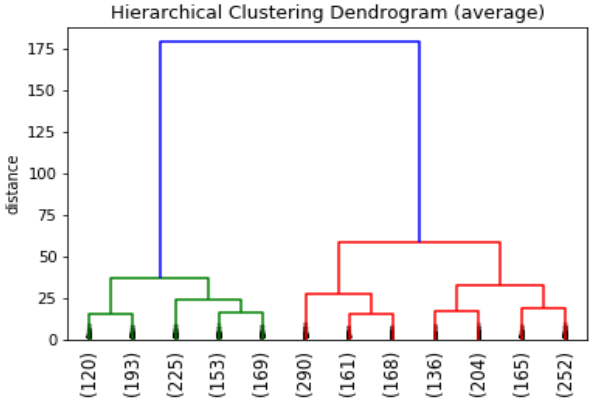
1. ward: smallest distance between two points
2. complete: largest distance between two points
3. average: average distance between two point

First, draw dendrogram to see the relationship between clusters
Dendrogram for temperature (left is ”complete”, right is “average”)

```
from scipy.cluster.hierarchy import dendrogram, linkage
start = time.time()
Z = linkage(temp_data, 'complete')
end = time.time()

plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=12,  # show only the last p merged clusters
    show_leaf_counts=True,  # otherwise numbers in brackets are counts
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
)
#max_d = 10000
#plt.axhline(y=max_d, c='k')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```



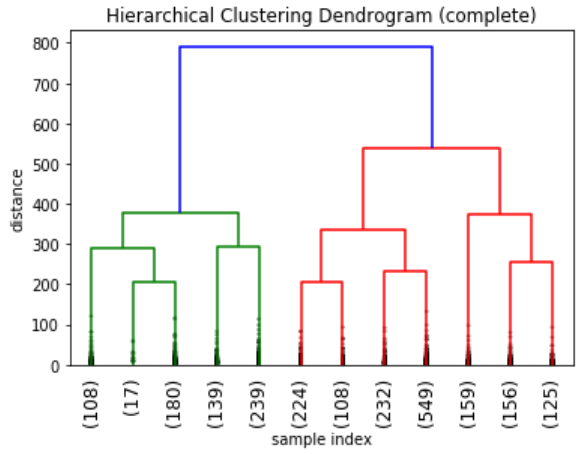
execution time 0.262 s



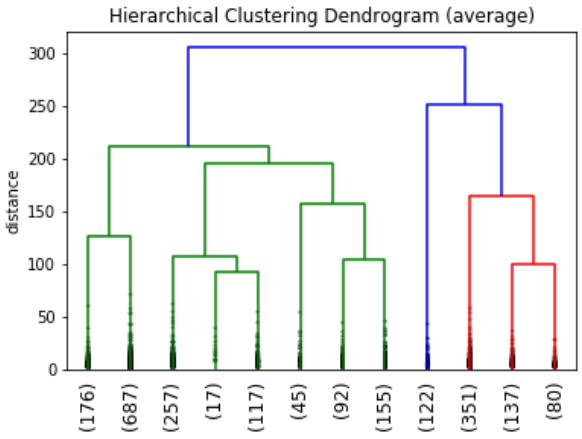
execution time 0.191 s

Dendrogram for power usage (left is ”complete”, right is “average”)

```
start = time.time()
Z = linkage(power_data, 'complete')
end = time.time()

plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    truncate_mode='lastp',  # show only the last p merged clusters
    p=12,  # show only the last p merged clusters
    show_leaf_counts=True,  # otherwise numbers in brackets are counts
    leaf_rotation=90.,
    leaf_font_size=12.,
    show_contracted=True,  # to get a distribution impression in truncated branches
)
plt.show()
print('execution time {:.3f} s'.format(end-start))
```



execution time 0.249 s



execution time 0.310 s

I didn’t show the figure of linkage type of “ward” since I think that the result of “complete” and “average” is enough to show the difference.

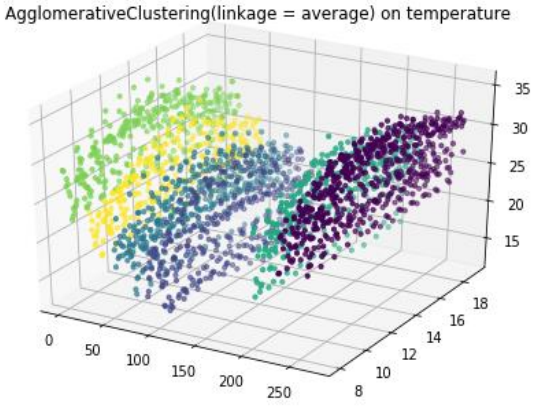
Then, do the clustering and show the result

Agglomerative clustering on temperature

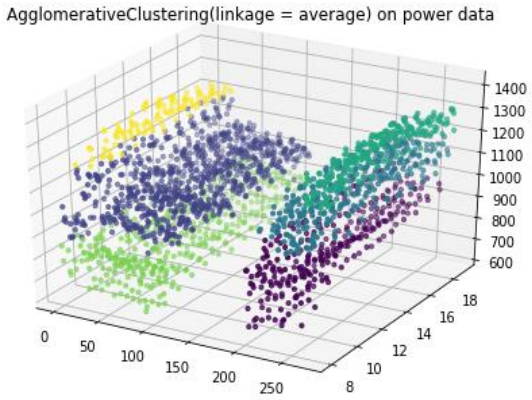
```
from sklearn.cluster import AgglomerativeClustering
start = time.time()
hclust = AgglomerativeClustering(linkage='average', affinity='euclidean', n_clusters=6)
hclust.fit(temp_data)
end = time.time()


fig = plt.figure()
ax = Axes3D(fig)

labels = hclust.labels_
ax.scatter(temp_data[:, 0], temp_data[:, 1], temp_data[:, 2], c=labels, s=10, cmap='viridis')

plt.title('AgglomerativeClustering(linkage = average) on temperature')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```

Agglomerative clustering on power data

```
from sklearn.cluster import AgglomerativeClustering
start = time.time()
hclust = AgglomerativeClustering(linkage='average', affinity='euclidean', n_clusters=6)
hclust.fit(power_data)
end = time.time()


fig = plt.figure()
ax = Axes3D(fig)

labels = hclust.labels_
ax.scatter(power_data[:, 0], power_data[:, 1], power_data[:, 2], c=labels, s=10, cmap='viridis')

plt.title('AgglomerativeClustering(linkage = average) on power data')
plt.show()
print('execution time {:.3f} s'.format(end-start))
```

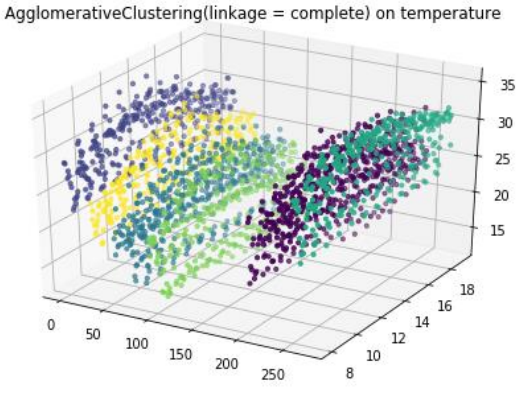


execution time 0.242 s

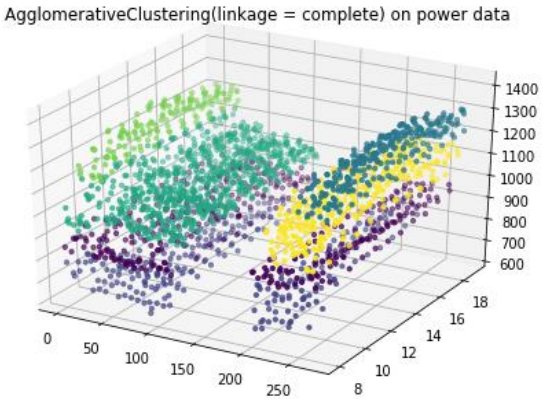


execution time 0.463 s

linkage type = average



execution time 0.285 s



execution time 0.239 s

linkage type = complete

Observation:
As we can see from the figure above, the clustering result on power usage data are a bit different between linkage type “complete” and “average”. Type “complete”，把 power usage 值較低的部分歸類在同一群。另外，type “average”，把 power usage 值較低的部分歸類成左半部及右半部不同的 cluster。

Due to the chosen linkage criterion, we can see the different result of the clustering.

Explanation:
I’ve use three kind of clustering algorithm on temperature and power usage data, but the result doesn’t show much difference and the clustering result on temperature and power usage data are not similar on my opinion.
The only similarity between temperature and power usage clustering result is that three algorithms all separate value from low to high.

三種 clustering 都把 temperature 及 power usage 從數值的高低作分群，不過 temperature 方面，相近的日子較容易被分成一群(直向分群)，而在 power usage 方面，較容易把同樣的用電量分成一群，日期間的分割不明顯(橫向分群)。我認為兩者的 clustering 結果，並沒有太多的相似之處。

I think that it might be the missing datas which cause the clustering result of power usage and temperature aren’t so similar. Therefore, I should fix the missing datas the next time I do this kind of work.

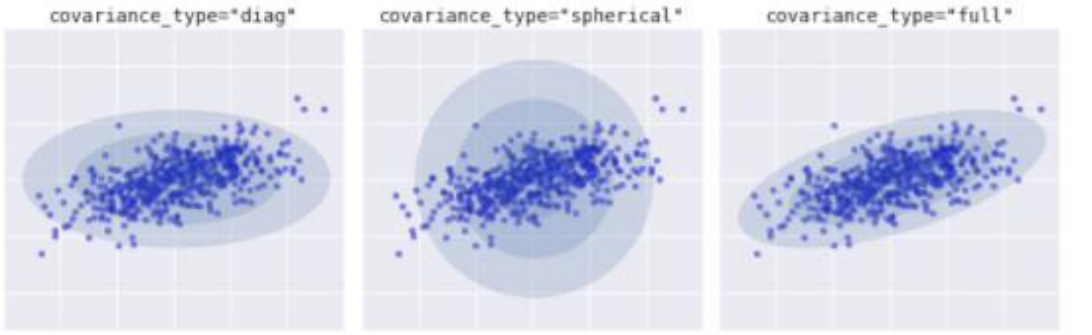
Conclusion:

K – means :
Requires prior knowledge of the data about how many clusters we want to divide the data into. Hence, for the first task, it’s easy for me to do k

means because I've already know what kind of clustering result I expected to see.
On the other hand, on the second task, I know nothing about what the clustering result it should be so it's hard for me to specify the clusters.

DBSCAN:
It does not require to specify the number of clusters at the first place, also it can find arbitrarily shaped clusters and eliminates noise. How to choose the min points and the epsilon is critical to the quality of the clustering result.

Gaussian Mixture:
It can be viewed as an extension of the ideas behind *k*-means. GMM is *a lot* more flexible in terms of cluster covariance. It attempts to find a mixture of multi-dimensional Gaussian probability distributions that best model any input dataset. Instead of spherical cluster in k-means, it also provides ellipse, elongated or rotated distribution of points in a cluster.



Hierarchy clustering:
Don't need any prior knowledge to the data and we can stop at whatever level (or clusters) we wish to find the appropriate number of clusters.
It can't not handle big data well like k-means does since it's an O(n^2) algorithm but the k-means is O(n).