0316323 薛世恩

Data mining hw2

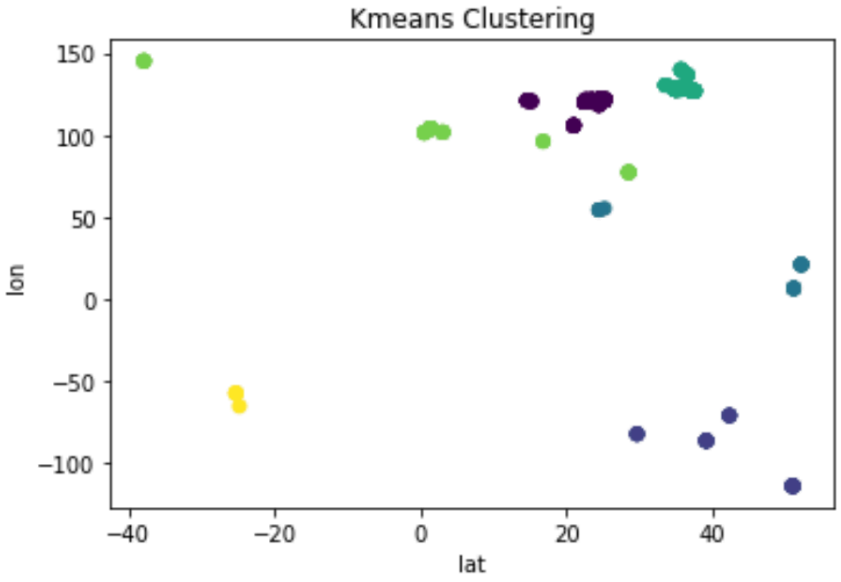
1.Spatial Clustering

a.

將資料對空間資訊做分群，如果資料在圖中某群，則代表在空間中與那一群相似。

Kmeans

```
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import KMeans, DBSCAN
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('201703_Taiwan.csv')
df1 = df.take(np.random.permutation(len(df))[:10000])
mat = df1[[' lat',' lon']]
matrix = mat.as_matrix()
km = KMeans(n_clusters = 6)
y_pre = km.fit_predict(matrix)
km.labels_
#plt.scatter(matrix[:,0],matrix[:,1], c=[plt.cm.spectra
plt.scatter(matrix[:, 0], matrix[:, 1], c = y_pre)
plt.title('Kmeans Clustering')
plt.xlabel('lat')
plt.ylabel('lon')
plt.show()
```



首先我使用 Kmeans 來做 geometric clustering，Kmeans 的分群會滿足所有資料點到其對應群中心的距離總和最小。

1.隨機選取資料組中的k筆資料當作初始群中心計算每個資料對應到最短距離的群中心

2.利用目前得到的分類重新計算群中心

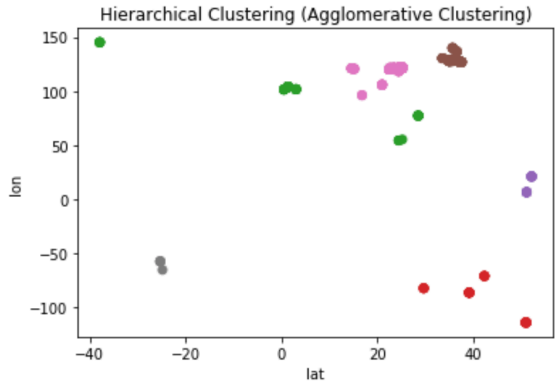
3.利用目前得到的分類重新計算群中心

4.重複step 2,3直到收斂(達到最大疊代次數 or 群心中移動距離很小)

一開始的群中心是不固定的，因此跑很多次可能都會得到不同的結果。初始群中心設定的不好可能導致不會的結果。

Hierarchical Clustering (Agglomerative Clustering)

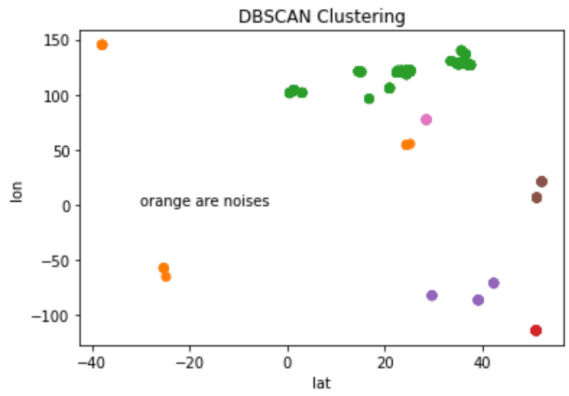
```
from sklearn.cluster import AgglomerativeClustering
def set_colors(labels, colors={-1:'C1',0:'C2',1:'C3',2:'C4',3:'C5',4:'C6',5:'C7',6:'C8',7:'C9',8:'C10'}):
    colored_labels = []
    for label in labels:
        colored_labels.append(colors[label])
    return colored_labels

estimator2 = AgglomerativeClustering(n_clusters=6)
X = df1[[' lat',' lon']]
estimator2.fit(X)
labels = estimator2.labels_

colors = set_colors(labels)
plt.scatter(df1[' lat'], df1[' lon'], c=colors)
plt.title('Hierarchical Clustering (Agglomerative Clustering)')
plt.xlabel("lat")
plt.ylabel("lon")
plt.show()
```

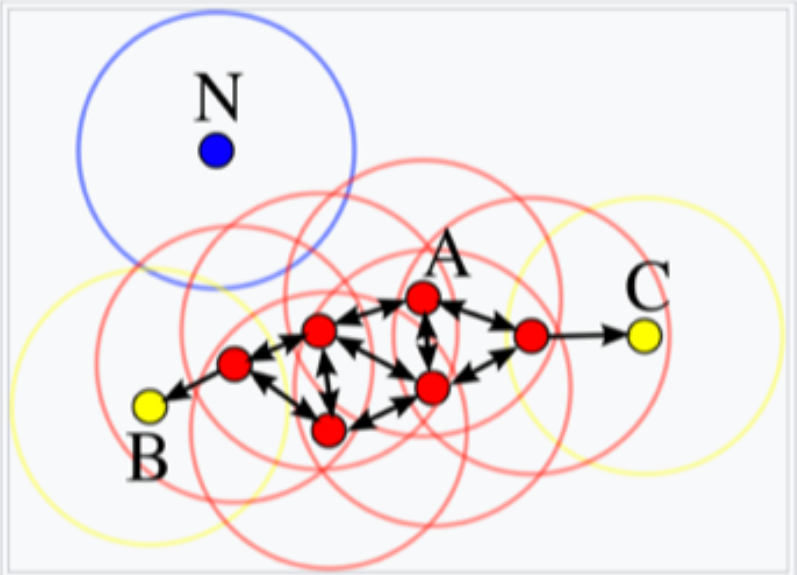


聚合式階層分群法（agglomerative hierarchical clustering）由樹狀結構的底部開始層層聚合。
一開始我們將每一筆資料視為一個群聚(cluster)，假設我們現在擁有n筆資料，則將這n筆資料視為n個群聚，亦即每個群聚包含一筆資料。然後找出所有群聚間，距離最接近的兩個群聚 ，將其合併成為一個新的群聚，直到群聚數目已將降到我們所要求的數目。

DBSCAN

```
def set_colors(labels, colors={-1:'C1',0:'C2',1:'C3',2:'C4',3:'C5',4:'C6',5:'C7',6:'C8',7:'C9',8:'C10'}):
    colored_labels = []
    for label in labels:
        colored_labels.append(colors[label])
    return colored_labels

mat1 = df1[[' lat',' lon']]
#matrix1 = mat1.as_matrix()
estimator = DBSCAN(eps=20, min_samples=10)
estimator.fit(mat1)
labels = estimator.labels_
#colors = estimator.fit_predict(mat1)
colors = set_colors(labels)
plt.scatter(df1[' lat'], df1[' lon'], c=colors)
plt.xlabel("lat")
plt.ylabel("lon")
#c1 is orange
plt.title('DBSCAN Clustering')
plt.text(-30, .025, 'orange are noises')
plt.show()
set(labels)
```





在這幅圖裡，minPts = 4，點 A 和其他紅色點是核心點，因為它們的 ε–鄰域（圖中紅色圓圈）裡包含最少 4 個點（包括自己），由於它們之間相互相可達，它們形成了一個聚類。點 B 和點 C 不是核心點，但它們可由 A 經其他核心點可達，所以也屬於同一個聚類。點 N 是局外點，它既不是核心點，又不由其他點可達。

Noisy sample，也就是局外點會被label成-1，並被標示成橘色。
參數有兩個分別為 eps 和形成高密度區域所需要的最少點數 minPts，它由一個任意未被訪問的點開始，然後探索這個點的 eps鄰域，如果 eps鄰域裡有足夠的點，則建立一個新的聚類，否則這個點被標籤為雜音。注意這個點之後可能被發現在其它點的 eps鄰域裡，而該 eps鄰域可能有足夠的點，屆時這個點會被加入該聚類中。
如果一個點位於一個聚類的密集區域裡，它的 eps鄰域裡的點也屬於該聚類，當這些新的點被加進聚類後，如果它們也在密集區域裡，它們的 ε–鄰域裡的點也會被加進聚類裡。這個過程將一直重覆，直至不能再加進更多的點為止，這樣，一個密度連結的聚類被完整地找出來。然後，一個未曾被訪問的點將被探索，從而發現一個新的聚類或雜音。

b.
首先最不同的是DBSCAN，因為它可以找出noisy samples，而其他兩種則是這些沒有達到標準的noisy sample也會被分群進去。
DBSCAN是以密度為基礎經由探訪做出連結，而不是像Kmeans或是agglomerative hierarchical clustering是以計算資料距離的方式做分群。由DBSCAN圖上綠色的分群為最明顯差異。
DBSCAN也不需給定目標分群數目，因為它會自動依照eps以及minPts將資料分成數群。

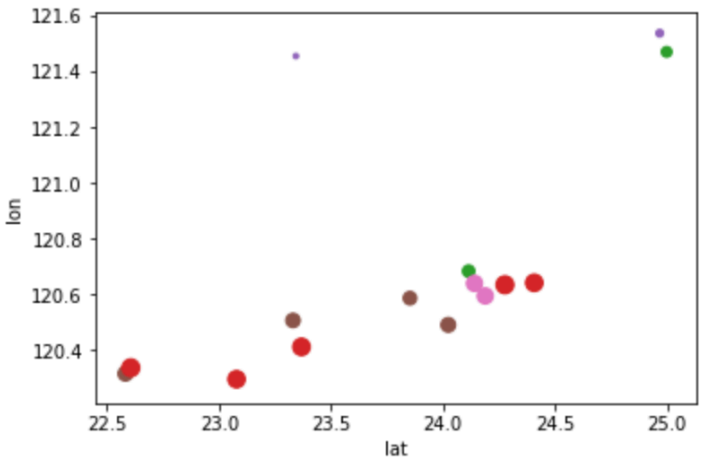
再來是Kmeans每次初始值都會不相同，且他的初始值會影響其結果，因此不一定每次執行結果都相同。而DBSCAN及agglomerative hierarchical clustering因演算法的緣故，每次執行結果都會相同。
Kmeans易受雜訊或是異常點干擾，而DBSCAN則不會。且Kmeans不易處理非球型或是不同大小的群，而DBSCAN可以。
Kmeans可以用於稀疏的高維數據，DBSCAN在這類數據上效能很差。

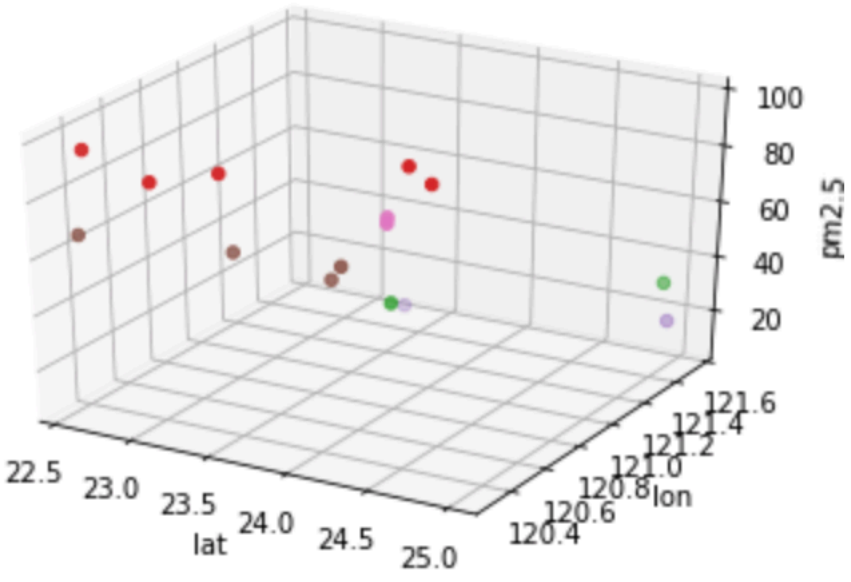
agglomerative hierarchical clustering的群數會逐漸減少，而kmeans分群的過程中，群的數目不變。

2.Spatial + PM2.5 Clustering

a.
我所選擇的資料是在2017年3月10日的0點0分40秒至50秒的資料。

```
from mpl_toolkits.mplot3d import Axes3D
df2 = df[(df['Date']=='2017-03-10')&(df['Time']>'00:00:40')&(df['Time']<'00:00:50')]
mat2 = df2[[' lat',' lon','PM2.5']]
matrix2 = mat2.as_matrix()
km2 = KMeans(n_clusters = 5)
km2.fit_predict(matrix2)

y_pre2 = set_colors(km2.labels_)
#plt.scatter(matrix[:,0],matrix[:,1], c=[plt.cm.spectral(float(i) /4) for i in km.lab
plt.scatter(matrix2[:,0], matrix2[:,1],matrix2[:,2], c = y_pre2)
plt.xlabel('lat')
plt.ylabel('lon')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('lat')
ax.set_ylabel('lon')
ax.set_zlabel('pm2.5')
ax.scatter(matrix2[:,0], matrix2[:,1],matrix2[:,2], c = y_pre2)
plt.show()
```

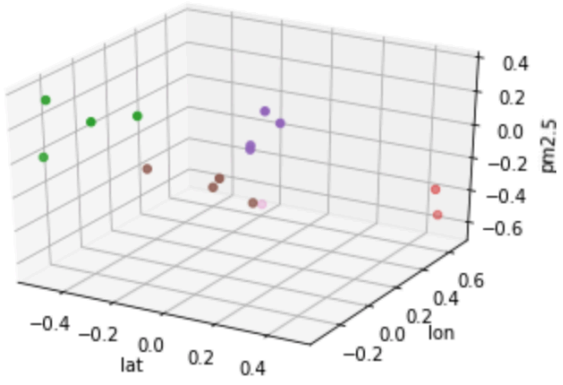




若點屬於某群，則其空間資訊及其pm2.5值與群中其他資料相近。

b. Normalize

```
df3 = df[(df['Date']=='2017-03-10')&(df['Time']>'00:00:40')&(df['Time']<'00:00:50')]

mat3 = df3[[' lat',' lon','PM2.5']]
#print(mat3)
mat3_norm = (mat3 - mat3.mean()) / (mat3.max() - mat3.min())
matrix3 = mat3_norm.as_matrix()
km3 = KMeans(n_clusters = 5)
km3.fit_predict(matrix3)

y_pre3 = set_colors(km3.labels_)
#plt.scatter(matrix[:,0],matrix[:,1], c=[plt.cm.spectral(float(i) /4) for i in km.lab
#plt.scatter(matrix3[:,0], matrix3[:,1],matrix3[:,2], c = y_pre3,marker='o')
#plt.xlabel('lat')
#plt.ylabel('lon')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('lat')
ax.set_ylabel('lon')
ax.set_zlabel('pm2.5')
ax.scatter(matrix3[:,0], matrix3[:,1],matrix3[:,2], c = y_pre3)
plt.show()
```



c. Compare the differences between clustering before normalization and clustering after normalization.

在normalize之前，因為每一個軸的所代表的意義不同(lat,lon,pm2.5)，很明顯的數值差異相當的大，因此在此例子中，分群很容易受到pm2.5影響，可以從normalize前的圖看出分群幾乎是由z軸(pm2.5)所決定，有很明顯的分層，因為lat與lon的距離的影響力被pm2.5完全的掩蓋。因此透過normalization來使三個軸的影響力不會特別偏某軸，因此可以從normalize後的圖看出分群不再只是受到pm2.5的影響，也沒有特別根據z軸分層的情況，分群比較平均。最明顯的例子就是圖的左上角在normalize前被分為紅色跟褐色兩群，而在normalize後都被分到綠色的群。

d. comparisons between Task 1 and Task 2
最明顯的就是維度上的提升，使得分群受到影響的因素變多，資料的分群不再只受到lat跟lon的影響，在三維時也會受到pm2.5的影響，然而由於pm2.5影響過大，因此要做normalization，使得每個軸對於資料分群的影響力較為平均。維度的上升可以讓資料的分群更加貼近現實，三維的分群較二維分群更能展現真實資料的狀況，因為我們擁有較多的資訊量，然而上升的維度也會讓計算較為複雜，需要更強大的計算能力。

3.Temporal Clustering

a.

```
adf = pd.read_csv('201703_Taiwan.csv')
adf = adf[(adf['Date']=='2017-03-03')&(adf['PM2.5'] != 0)&(adf['PM1'] != 0)&(adf['PM10'] != 0)]
adf['Date'] = pd.to_datetime(adf['Date'] + ' ' + adf['Time'])
adf = adf[["Date", "device_id", "PM2.5"]]
adf.head()
```

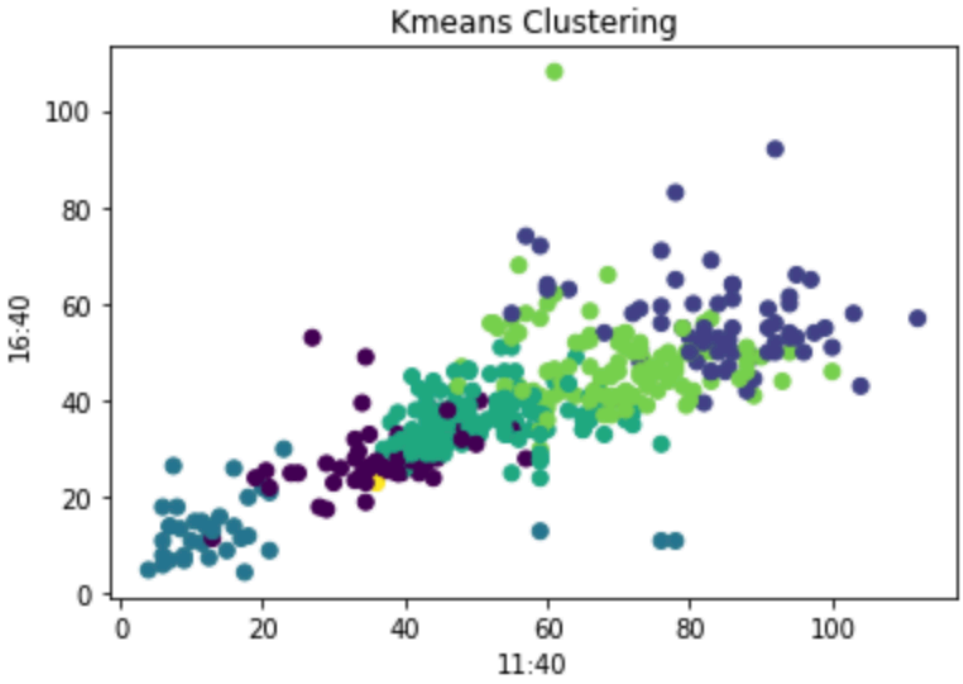
	Date	device_id	PM2.5
175168	2017-03-03 00:00:00	74DA38A86916	9
175169	2017-03-03 00:00:02	74DA3895C57E	59
175170	2017-03-03 00:00:05	74DA3895C2BE	50
175171	2017-03-03 00:00:06	74DA3895C58E	44
175172	2017-03-03 00:00:06	74DA3895C314	41

```
min_count = adf.groupby('device_id').size().reset_index(name='counts').mean()/2
min_count_adf = adf.groupby("device_id").filter(lambda x: len(x) > min_count)
grouper = min_count_adf.set_index("Date").groupby([pd.TimeGrouper('10T'), 'device_id'])
grouper = grouper['PM2.5'].mean().unstack()
ans = grouper.fillna(method='ffill').transpose().dropna()
ans
```

Date	2017-03-03 00:00:00	2017-03-03 00:10:00	2017-03-03 00:20:00	2017-03-03 00:30:00	2017-03-03 00:40:00	2017-03-03 00:50:00	2017-03-03 01:00:00	2017-03-03 01:10:00	2017-03-03 01:20:00	2017-03-03 01:30:00	...	2017-03-03 22:20:00	2017-03-03 22:30:00
device_id													
28C2DDDD436E	37.0	37.0	37.0	37.5	37.0	37.0	37.5	37.5	37.5	37.0	...	29.0	29.0
28C2DDDD4539	27.0	24.0	24.0	27.0	25.0	23.5	24.5	24.0	22.5	21.0	...	16.5	17.0
28C2DDDD47A8	36.0	37.0	36.0	36.0	36.5	36.0	35.0	32.0	32.0	32.0	...	21.5	19.0
28C2DDDD47BB	32.5	33.0	31.0	32.5	33.0	33.0	33.0	33.0	33.0	33.0	...	24.0	28.5
74DA388FF5F6	38.0	38.0	38.0	39.0	41.0	42.0	42.0	42.0	42.0	42.0	...	97.0	86.0
74DA388FF606	34.0	34.0	35.0	35.0	34.0	33.0	35.0	34.0	34.0	34.0	...	26.0	26.0
74DA388FF60A	36.5	36.5	36.5	37.0	38.0	38.0	39.0	38.0	38.0	38.0	...	31.0	31.0

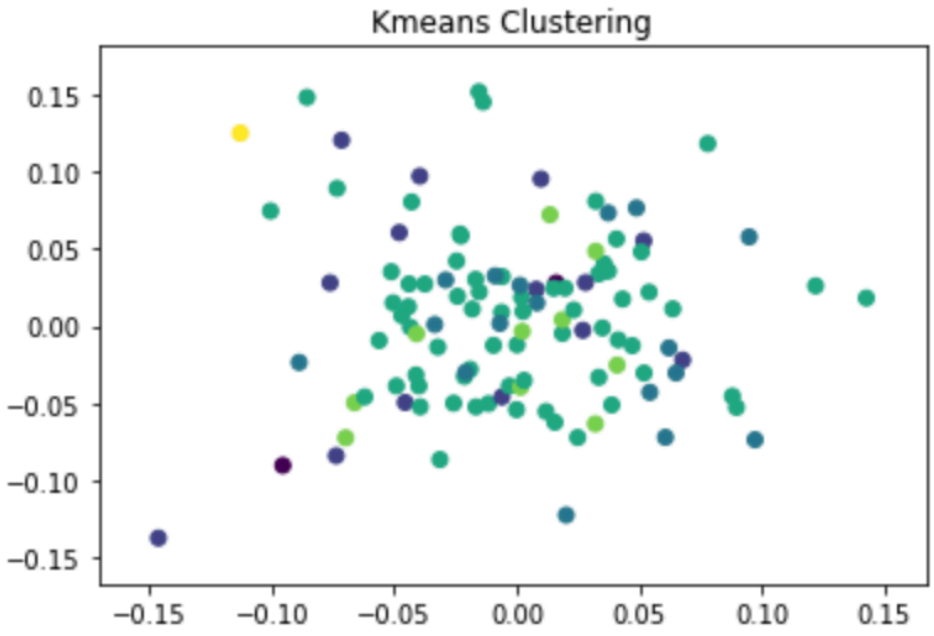
首先先將資料讀出並過濾到只剩所需要的資訊。
再過濾掉資訊量過少的device。因為如果資訊過少，做起來會有資料的不完整。
然候每十分鐘為間隔取樣並照device做group。
column為device，row為時間。
先將資料做填補。
然後做transpose將column轉為時間，row轉為device。
將還有缺漏資料的device給剔除。
最後就是做Kmeans。
並將label圖形化呈現。

```
anss=ans
matrix9 = ans.as_matrix()
km9 = KMeans(n_clusters = 6)
y_pre9 = km9.fit_predict(matrix9)
km9.labels_
#plt.scatter(matrix[:,0],matrix[:,1], c=[plt.cm.spectral(
plt.scatter(matrix9[:, 70], matrix9[:, 100], c = y_pre9)
plt.title('Kmeans Clustering')
plt.xlabel('11:40')
plt.ylabel('16:40')
plt.show()
```



b.PCA

```
from sklearn.decomposition import PCA
pca =PCA(n_components='mle')
pca.fit(anss)
matrix8 = pca.components_

km8 = KMeans(n_clusters = 6)
y_pre8 = km8.fit_predict(matrix8)
km8.labels_
#plt.scatter(matrix[:,0],matrix[:,1], c=[plt.cm.spectr
plt.scatter(matrix8[:, 3], matrix8[:, 8], c = y_pre8)
plt.title('Kmeans Clustering')

plt.show()
```



c.
在做PCA之前所產生144維的資料為，某裝置在幾點幾分時pm2.5值的對應值，如果我們將其做clustering也就是分群出出pm2.5值在3月三號那天pm2.5曲線長得很像的sensor，但是礙於維度不好將144維全部印出來也不好表示，所以特別印出11:40以及16:40維度的圖，也就是在這兩個時間點device的pm2.5值分佈圖。點的走勢大致由左下往右上，這代表的是，device如果在11:40時pm2.5值偏高，那麼在16:40時pm2.5直也會相對偏高，在11:40時pm2.5值偏低，那麼在16:40時pm2.5直也會相對偏低。在同一群的sensor曲線就會比較像，所以如果在某維度，如果分群可以分得非常清楚，就可以利用那個時間點的pm2.5值大概推測走勢會跟相近的群較為相似。

做完PCA後首先的影響是降維度，將多維度的資訊塞到低維度中間，因此將某些維度的圖印出來時原本的走勢就不見了。再來資訊量可能會稍微遺失，如果將維度大量降低，可能會造成資訊量嚴重缺漏。且維度就不再具有代表性，例如原本維度是指幾點幾分時的pm2.5值。

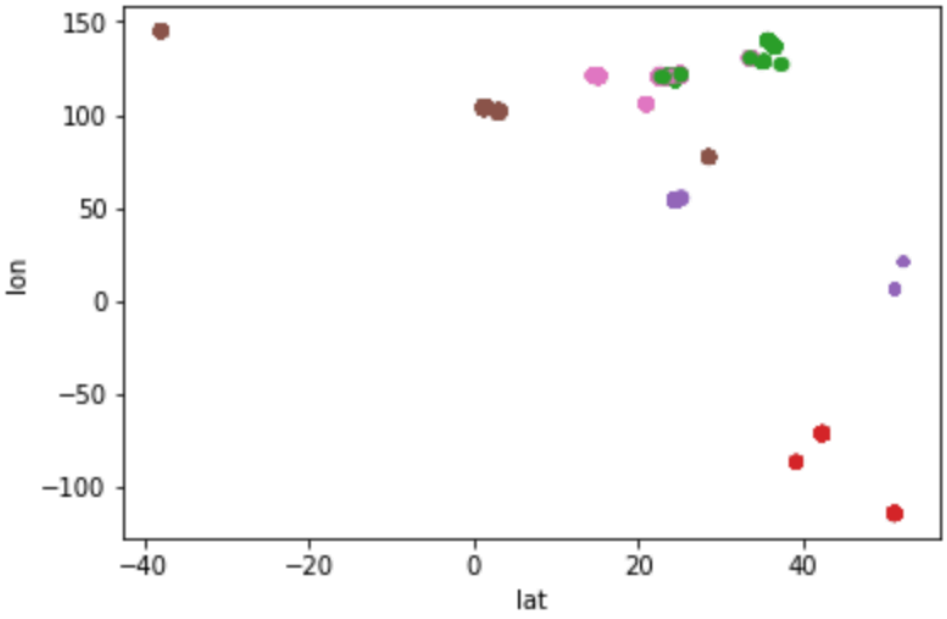
4.Other

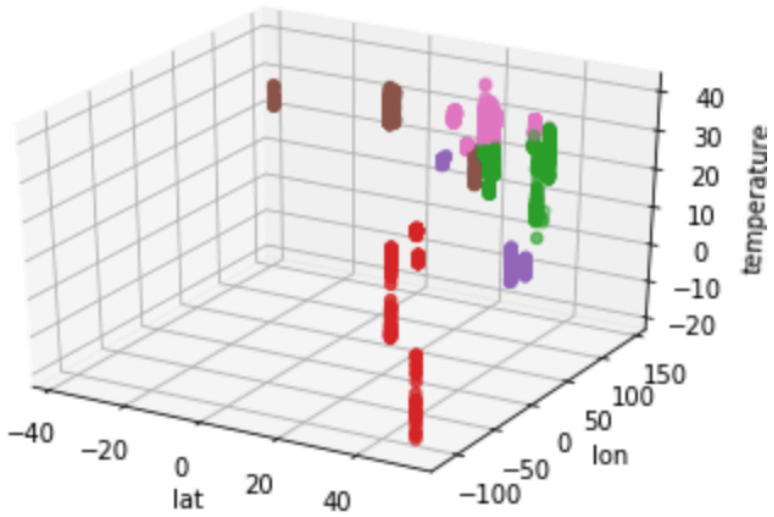
```
df5 = df[(df['Date']=='2017-03-10')]
mat5 = df5[[' lat',' lon','Temperature']]

mat5_norm = (mat5 - mat5.mean()) / (mat5.max() - mat5.min())
matrix5 = mat5_norm.as_matrix()
km5 = KMeans(n_clusters = 5)
km5.fit_predict(matrix5)

y_pre5 = set_colors(km5.labels_)
#plt.scatter(matrix[:,0],matrix[:,1], c=[plt.cm.spectral(float(i)
plt.scatter(matrix5[:,0], matrix5[:,1],matrix5[:,2], c = y_pre5)
plt.xlabel('lat')
plt.ylabel('lon')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('lat')
ax.set_ylabel('lon')
ax.set_zlabel('temperature')
ax.scatter(matrix5[:,0], matrix5[:,1],matrix5[:,2], c = y_pre5)
plt.show()
```

資料在空間資訊與溫度資訊與同群的資料相似。
空間資訊以及溫度的三維clustering。

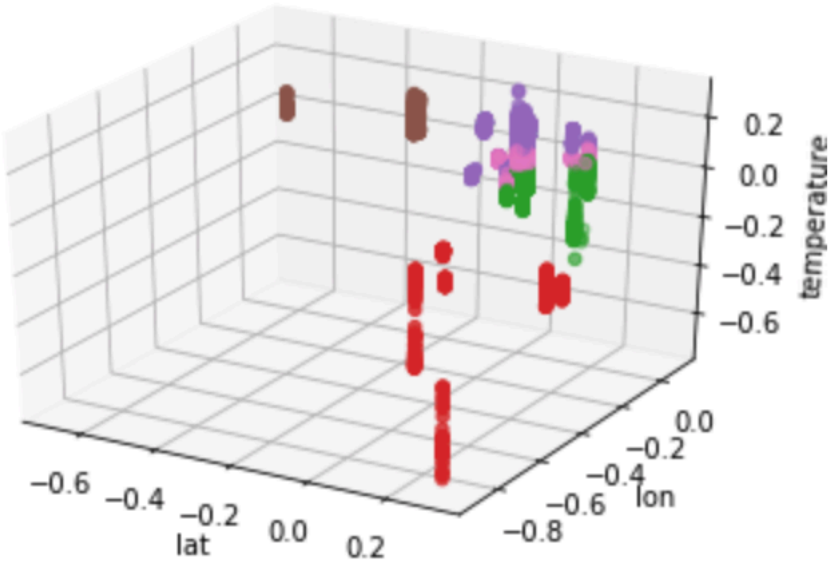
Normalization
mat5_norm = (mat5 - mat5.mean()) / (mat5.max() - mat5.min())

Before normalization





After normalization



因為device位置固定且一天之中溫度變化較大，因此溫度在3D空間中大致呈棒狀，在normalize後，除了紅色的群因為空間距離其他群較遠，因此沒有受到影響，而其他距離相近的群，則受到溫度的影響，而在溫度不同時被分配到不同的群，並跟群中的點較為相似。