ECT_HW4_108403523

一. python 實作

1. 載入資料並刪除特定欄位

- 載入資料

```
In [1]:
import pandas as pd
song = pd.read_csv('wu_songs.csv')
song
```

Out[1]:

	energy	liveness	tempo	speechiness	acousticness	instrumentalness	time_signature	danceability	key	duration_ms	loudness	valence	mode	
0	0.979	0.2720	128.876	0.1220	0.007620	0.015200	4	0.410	3	294740	-3.481	0.080	0	audio_
1	0.861	0.0747	100.080	0.0493	0.001890	0.000539	4	0.518	6	231762	-6.998	0.317	0	audio_
2	0.963	0.2030	195.979	0.1590	0.000090	0.000304	4	0.356	9	217959	-4.385	0.379	1	audio_
3	0.968	0.1060	158.089	0.1370	0.000047	0.000006	4	0.365	2	198987	-4.267	0.386	1	audio_
4	0.327	0.0922	75.122	0.0489	0.605000	0.000012	4	0.434	6	216100	-10.161	0.260	0	audio_
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1918	0.379	0.0986	107.989	0.0476	0.607000	0.000464	4	0.614	4	258987	-10.480	0.201	1	audio_
1919	0.733	0.3290	93.019	0.0720	0.006200	0.000000	4	0.748	11	312240	-4.421	0.291	0	audio_
1920	0.437	0.1160	103.921	0.0363	0.440000	0.000000	4	0.769	1	186443	-6.406	0.737	1	audio_
1921	0.841	0.1330	100.003	0.0391	0.015600	0.000000	4	0.743	0	226800	-5.189	0.814	1	audio_
1922	0.495	0.1920	120.047	0.0328	0.114000	0.002950	4	0.585	10	197227	-9.428	0.379	0	audio_

1923 rows × 15 columns

- 刪除題目指定不需要的欄位

```
In [2]:
song = song.drop(["time_signature","key","duration_ms","type","uri","mode"],axis=1)
song
```

Out[2]:

	energy	liveness	tempo	speechiness	acousticness	instrumentalness	danceability	loudness	valence
0	0.979	0.2720	128.876	0.1220	0.007620	0.015200	0.410	-3.481	0.080
1	0.861	0.0747	100.080	0.0493	0.001890	0.000539	0.518	-6.998	0.317
2	0.963	0.2030	195.979	0.1590	0.000090	0.000304	0.356	-4.385	0.379
3	0.968	0.1060	158.089	0.1370	0.000047	0.000006	0.365	-4.267	0.386
4	0.327	0.0922	75.122	0.0489	0.605000	0.000012	0.434	-10.161	0.260
...	...	...	...	...	...	...	...	...	...
1918	0.379	0.0986	107.989	0.0476	0.607000	0.000464	0.614	-10.480	0.201
1919	0.733	0.3290	93.019	0.0720	0.006200	0.000000	0.748	-4.421	0.291
1920	0.437	0.1160	103.921	0.0363	0.440000	0.000000	0.769	-6.406	0.737
1921	0.841	0.1330	100.003	0.0391	0.015600	0.000000	0.743	-5.189	0.814
1922	0.495	0.1920	120.047	0.0328	0.114000	0.002950	0.585	-9.428	0.379

1923 rows × 9 columns

---

2. 將剩下的欄位做特徵篩選

- 產生所有欄位可能的組合 (共84組)

產生所有欄位可能的組合

In [3]:
```
from itertools import combinations
column = ["energy","liveness","tempo","speechiness","acousticness","instrumentalness","danceability","loudness","valence"]
column_combinations = list(combinations(column,3))
for i in range(len(column_combinations)):
    column_combinations[i] = list(column_combinations[i])
len(column_combinations)
```

Out[3]: 84

- 測試取得特定欄位 (成功)

In [4]:
```
song[column_combinations[0]]
```

Out[4]:

	energy	liveness	tempo
0	0.979	0.2720	128.876
1	0.861	0.0747	100.080
2	0.963	0.2030	195.979
3	0.968	0.1060	158.089
4	0.327	0.0922	75.122
...	...	...	...
1918	0.379	0.0986	107.989
1919	0.733	0.3290	93.019
1920	0.437	0.1160	103.921
1921	0.841	0.1330	100.003
1922	0.495	0.1920	120.047

1923 rows × 3 columns

- 資料標準化 (StandardScaler)

In [5]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler().fit(song)
X_scaled = scaler.transform(song)
```

In [6]:
```
X_scaled
```

Out[6]:
```
array([[ 0.99938374,  0.47846684,  0.02072052, ..., -0.46694458,
         0.87140431, -1.36455913],
       [ 0.43068195, -0.85661954, -0.98363471, ...,  0.29458568,
        -0.33314735, -0.12710912],
       [ 0.92227163,  0.01155878,  2.36115837, ..., -0.84770971,
         0.56178967,  0.19661198],
       ...,
       [-1.6127889 , -0.57715138, -0.8496672 , ...,  2.06443841,
        -0.13039086,  2.06584027],
       [ 0.33429181, -0.46211607, -0.98632033, ...,  1.88110705,
         0.28642443,  2.46788099],
       [-1.33325751, -0.06287584, -0.2872199 , ...,  0.76701649,
        -1.16540796,  0.19661198]])
```

- 計算各種欄位組合及分群數的 silhouette score

計算各種欄位組合及分群數的 silhouette score

In [10]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
silhouette_avg = []
temp = []

for i in range(2,13):
    for j in range(84):
        scaler = StandardScaler().fit(song[column_combinations[j]])
        X_scaled = scaler.transform(song[column_combinations[j]])
        kmeans_fit = KMeans(n_clusters = i,random_state=15).fit(X_scaled)
        temp.append(silhouette_score(X_scaled, kmeans_fit.labels_))

    silhouette_avg.append(temp)
    temp = []
```

- 找到分幾群的 silhouette score 比較高 (index+2群)

(分兩群的 score 比較高)

找到分幾群的 silhouette score 比較高 (index+2群)

In [11]:
```
print(max(silhouette_avg[0]))
print(max(silhouette_avg[1]))
print(max(silhouette_avg[2]))
print(max(silhouette_avg[3]))
print(max(silhouette_avg[4]))
print(max(silhouette_avg[5]))
print(max(silhouette_avg[6]))
print(max(silhouette_avg[7]))
print(max(silhouette_avg[8]))
print(max(silhouette_avg[9]))
print(max(silhouette_avg[10]))
```

```
0.632258638620883
0.6241754174289839
0.6016158450111958
0.5539769196815084
0.5650290074636424
0.5458667242749337
0.5484267350362002
0.5034128982063273
0.4924290279573839
0.5025242090684943
0.4950767994714273
```

- 找到是哪一種欄位組合

找到是哪一種欄位組合

In [12]:
```
for i in range(84):
    if silhouette_avg[0][i] == 0.632258638620883:
        print(i)
        break
```

```
20
```

- 檢視找出的資料欄位

檢視找出的資料欄位

In [14]:
```
1 song[column_combinations[20]]
```

Out[14]:

	energy	acousticness	loudness
0	0.979	0.007620	-3.481
1	0.861	0.001890	-6.998
2	0.963	0.000090	-4.385
3	0.968	0.000047	-4.267
4	0.327	0.605000	-10.161
...	...	...	...
1918	0.379	0.607000	-10.480
1919	0.733	0.006200	-4.421
1920	0.437	0.440000	-6.406
1921	0.841	0.015600	-5.189
1922	0.495	0.114000	-9.428

1923 rows × 3 columns

- 該欄位組合分不同群時的 silhouette_score

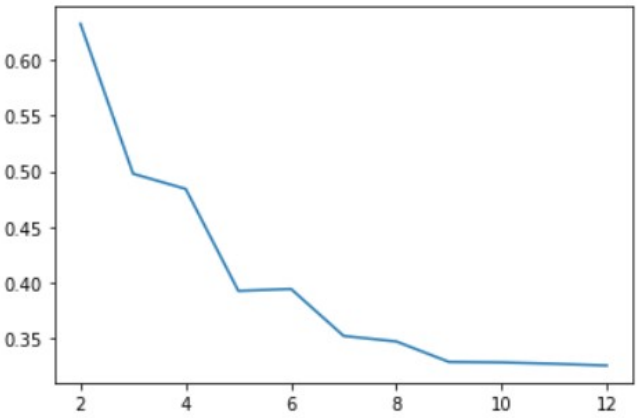
該欄位組合分不同群時的 **silhouette_score**

In [14]:
```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
silhouette_avg = []

for i in range(2,13):
    scaler = StandardScaler().fit(song[column_combinations[20]])
    X_scaled = scaler.transform(song[column_combinations[20]])
    kmeans_fit = KMeans(n_clusters = i,random_state=15).fit(X_scaled)
    silhouette_avg.append(silhouette_score(X_scaled, kmeans_fit.labels_))

plt.plot(range(2,13), silhouette_avg)
```

Out[14]: [<matplotlib.lines.Line2D at 0x2996731ac48>]



結論: 欄位是energy、acousticness、loudness，分2群。

---

3. 使用剛剛找出來的欄位用 k-means 做分群

- 設定模型參數進行預測

設定模型參數進行預測

In [12]:
```
kmeans = KMeans(n_clusters=4, random_state=15)
scaler = StandardScaler().fit(song[column_combinations[20]])
X_scaled = scaler.transform(song[column_combinations[20]])
bb = kmeans.fit_predict(X_scaled)
```

- 將用來預測的資料欄位轉換為 DataFrame (後續也會重複使用到xx)

將用來預測的資料欄位轉換為 DataFrame (後續也會重複使用到xx)

In [13]:
```
xx = pd.DataFrame(X_scaled,columns=["energy","acousticness","loudness"])
xx
```

Out[13]:

	energy	acousticness	loudness
0	0.999384	-0.472349	0.871404
1	0.430682	-0.496959	-0.333147
2	0.922272	-0.504691	0.561790
3	0.946369	-0.504872	0.602204
4	-2.142935	2.093298	-1.416456
...	...	...	...
1918	-1.892320	2.101887	-1.525712
1919	-0.186215	-0.478448	0.549460
1920	-1.612789	1.384650	-0.130391
1921	0.334292	-0.438076	0.286424
1922	-1.333258	-0.015465	-1.165408

1923 rows × 3 columns

- 將預測結果 bb 轉換成 DataFrame dd

將預測結果 bb 轉換成 DataFrame dd

In [14]:
```
1 dd = pd.DataFrame({"kmeans":bb})
2 dd
```

Out[14]:

	kmeans
0	0
1	0
2	0
3	0
4	1
...	...
1918	1
1919	0
1920	1
1921	0
1922	2

1923 rows × 1 columns

- 將 xx 與 dd 合併成新的 DataFrame

將 xx 與 dd 合併成新的 DataFrame

In [15]:
```
fff = pd.concat([xx,dd],axis=1)
fff
```

Out[15]:

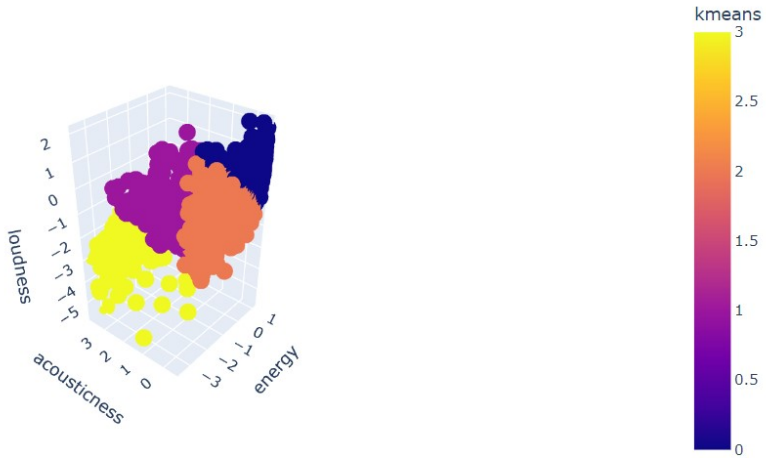
	energy	acousticness	loudness	kmeans
0	0.999384	-0.472349	0.871404	0
1	0.430682	-0.496959	-0.333147	0
2	0.922272	-0.504691	0.561790	0
3	0.946369	-0.504872	0.602204	0
4	-2.142935	2.093298	-1.416456	1
...	...	...	...	...
1918	-1.892320	2.101887	-1.525712	1
1919	-0.186215	-0.478448	0.549460	0
1920	-1.612789	1.384650	-0.130391	1
1921	0.334292	-0.438076	0.286424	0
1922	-1.333258	-0.015465	-1.165408	2

1923 rows × 4 columns

- 繪製出 3d 圖形

繪製出 3d 圖形

In [16]:
```
import plotly.express as px
# Use directly Columns as argument. You can use tab completion for this!
import plotly.express as px
fig = px.scatter_3d(fff, x='energy', y='acousticness', z='loudness',
                    color='kmeans')
fig.show()
```



4. 使用剛剛找出來的欄位用 Meanshift 做分群

- 找出 bandwidth 並做 Meanshift 分群

找出 bandwidth 並做 Meanshift 分群

In [17]:
```
from sklearn.cluster import MeanShift, estimate_bandwidth
import numpy as np
bandwidth = estimate_bandwidth(xx,quantile=0.32,n_samples=1000,random_state=15)

ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
labels = ms.fit_predict(xx)

labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)

print("number of estimated clusters : %d" % n_clusters_)
```

```
number of estimated clusters : 4
```

- 將分群結果 labels 存成 DataFrame

將分群結果 labels 存成 DataFrame

In [18]:
```
jj = pd.DataFrame(labels,columns=["meanshift"])
jj
```

Out[18]:

	meanshift
0	0
1	0
2	0
3	0
4	1
...	...
1918	1
1919	0
1920	1
1921	0
1922	1

1923 rows × 1 columns

- 將 xx 與 jj 合併成新的 DataFrame fff

將 xx 與 jj 合併成新的 DataFrame fff

In [19]:
```
fff = pd.concat([xx,jj],axis=1)
fff
```

Out[19]:

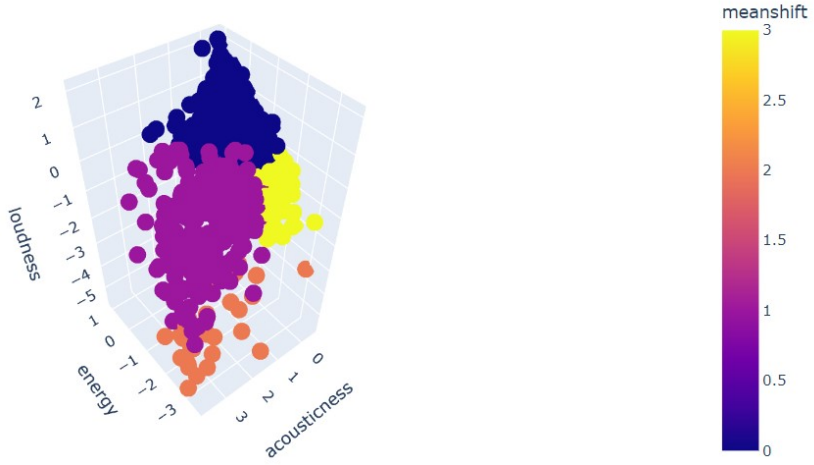
	energy	acousticness	loudness	meanshift
0	0.999384	-0.472349	0.871404	0
1	0.430682	-0.496959	-0.333147	0
2	0.922272	-0.504691	0.561790	0
3	0.946369	-0.504872	0.602204	0
4	-2.142935	2.093298	-1.416456	1
...	...	...	...	...
1918	-1.892320	2.101887	-1.525712	1
1919	-0.186215	-0.478448	0.549460	0
1920	-1.612789	1.384650	-0.130391	1
1921	0.334292	-0.438076	0.286424	0
1922	-1.333258	-0.015465	-1.165408	1

1923 rows × 4 columns

- 繪製出 Meanshift 圖形

繪製出 Meanshift 圖形

In [20]:
```
import plotly.express as px
# Use directly Columns as argument. You can use tab completion for this!
import plotly.express as px
fig = px.scatter_3d(fff, x='energy', y='acousticness', z='loudness',
                    color='meanshift')
fig.show()
```



---

5. 使用剛剛找出來的欄位用 k-prototypes 做分群

- 做 k-prototypes 分群

做 k-prototypes 分群

In [21]:
```
from kmodes.kprototypes import KPrototypes

kp = KPrototypes(n_clusters=4,random_state=15,init='Huang',verbose=0)
bbcall = kp.fit_predict(xx, categorical=[0,2])
```

- 將分群結果 bbcall 轉換成 DataFrame

將分群結果 bbcall 轉換成 DataFrame

In [22]:
```
bbcall_column = pd.DataFrame(bbcall,columns=["kprototypes"])
bbcall_column
```

Out[22]:

	kprototypes
0	0
1	0
2	0
3	0
4	1
...	...
1918	1
1919	0
1920	1
1921	0
1922	2

1923 rows × 1 columns

- 將分群結果 bbcall_column 與原資料 xx 合併成新的 DataFrame

將分群結果 bbcall_column 與原資料 xx 合併成新的 DataFrame

In [23]:
```
ffff = pd.concat([xx,bbcall_column],axis=1)
ffff
```

Out[23]:

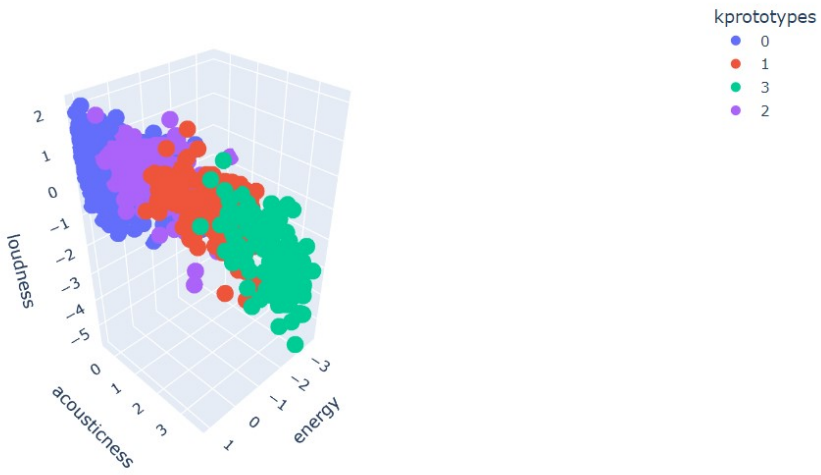
	energy	acousticness	loudness	kprototypes
0	0.999384	-0.472349	0.871404	0
1	0.430682	-0.496959	-0.333147	0
2	0.922272	-0.504691	0.561790	0
3	0.946369	-0.504872	0.602204	0
4	-2.142935	2.093298	-1.416456	1
...	...	...	...	...
1918	-1.892320	2.101887	-1.525712	1
1919	-0.186215	-0.478448	0.549460	0
1920	-1.612789	1.384650	-0.130391	1
1921	0.334292	-0.438076	0.286424	0
1922	-1.333258	-0.015465	-1.165408	2

1923 rows × 4 columns

- 繪製出 k-prototypes 3d 圖形

繪製出 k-prototypes 3d 圖形

In [24]:
```
import plotly.express as px
fig = px.scatter_3d(ffff, x='energy', y='acousticness', z='loudness',
                    color='kprototypes')
fig.show()
```



---

6. 使用剛剛找出來的欄位用 k-modes 做分群

- 用 k-modes 做分群

用 k-modes 做分群

In [32]:
```
from kmodes.kmodes import KModes
km = KModes(n_clusters=4,random_state=15,init='Huang',verbose=0)
gg = km.fit_predict(xx)
# Print the cluster centroids
print(gg)
```

```
[0 0 0 ... 0 0 0]
```

- 將分類結果 gg 存成 DataFrame bbcall_column2 並與原資料 xx 合併成新 DataFrame fffff

將分類結果 gg 存成 DataFrame bbcall_column2 並與原資料 xx 合併成新 DataFrame fffff

In [33]:
```
bbcall_column2 = pd.DataFrame(gg,columns=["kmodes"])
fffff = pd.concat([xx,bbcall_column2],axis=1)
fffff
```

Out[33]:

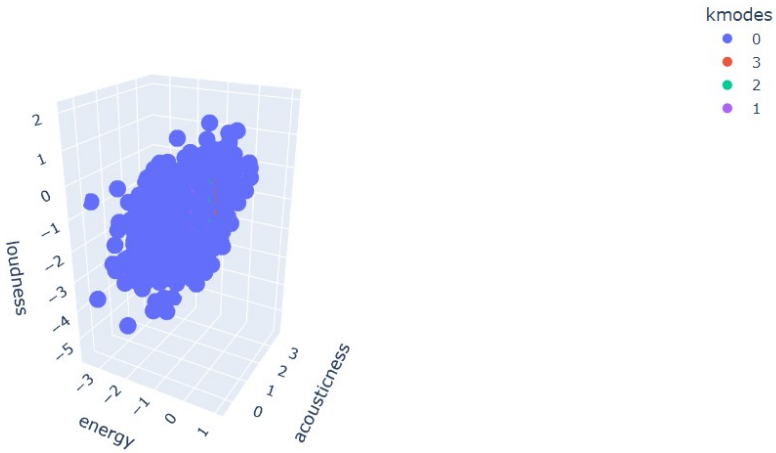
	energy	acousticness	loudness	kmodes
0	0.999384	-0.472349	0.871404	0
1	0.430682	-0.496959	-0.333147	0
2	0.922272	-0.504691	0.561790	0
3	0.946369	-0.504872	0.602204	3
4	-2.142935	2.093298	-1.416456	0
...	...	...	...	...
1918	-1.892320	2.101887	-1.525712	0
1919	-0.186215	-0.478448	0.549460	0
1920	-1.612789	1.384650	-0.130391	0
1921	0.334292	-0.438076	0.286424	0
1922	-1.333258	-0.015465	-1.165408	0

1923 rows × 4 columns

- 繪製出 k-modes 3d 圖形

繪製出 k-modes 3d 圖形

In [34]:
```
import plotly.express as px
fig = px.scatter_3d(fffff, x='energy', y='acousticness', z='loudness',
                    color='kmodes')
fig.show()
```



---

7. 比較說明上述四種分群法的差異

1. K-means : 分類時，必須通通使用 numeric 的 dtype，所以若有 categorical 的資料則需先做轉換才能做分群。給定初始群心後，它會去計算各筆資料和各群心的 Euclidean distance，利用計算出的距離遠近，判斷該筆資料要被分到哪一群。之後，新的群心會由被分類到該群的資料欄位數值平均決定，計算完後再次根據離群心之遠近，決定該筆

資料歸屬何群。最後，反覆進行新群心的計算和離群心遠近的分群判斷，直到資料分群不再變動。

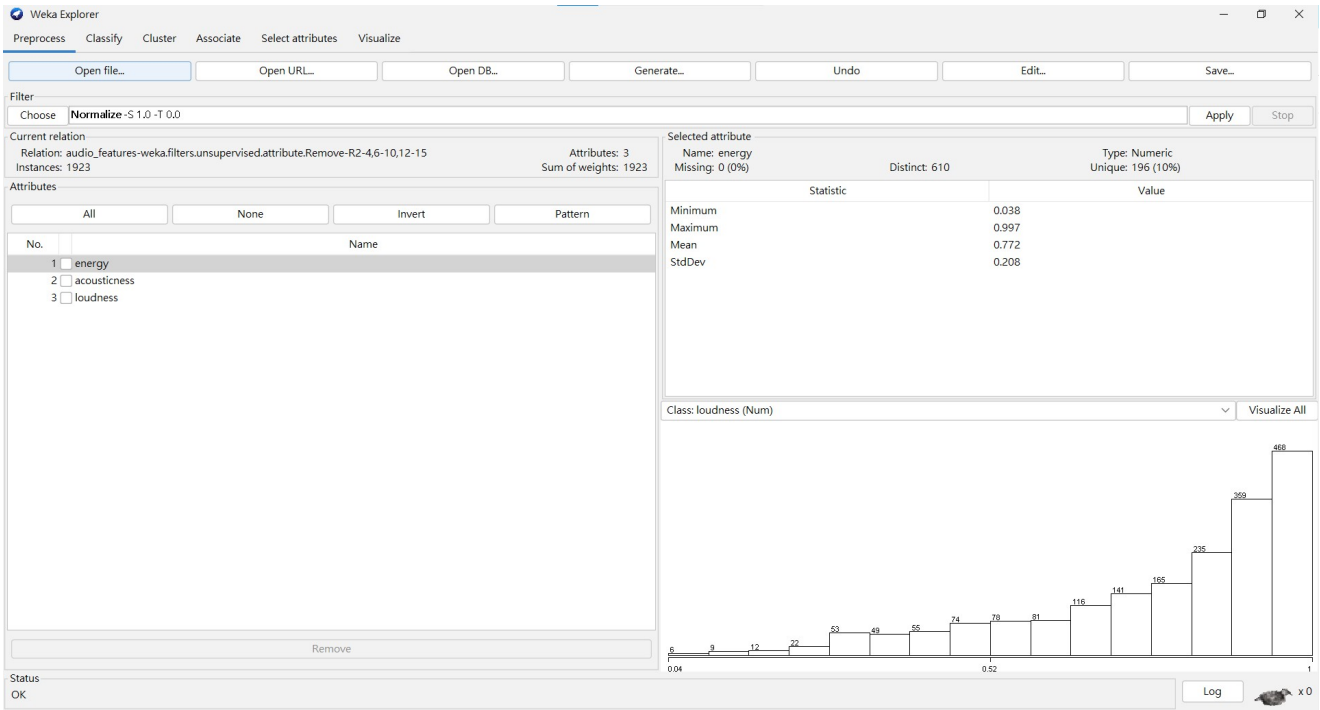
2. Meanshift : 在資料集中選定一個中心點，然後以這個中心點為圓心，r 為半徑，畫一個圓，求出這個圓心到所有點的向量的平均值。之後，利用圓心與向量均值的和算出新的圓心，然後反覆這個過程，直到找到涵蓋最多點的圓為止，也就是找到真正的分群中心。bandwidth 參數代表的就是圓的半徑 r。

3. K-prototypes : 兼具 K-means 和 K-modes 的特性。分群時，可以使用 categorical 和 numeric 資料，但不可以只有單一種資料，資料集必須同時具備兩種類型的資料。分群過程跟 K-means 大致相同，只是它混用與群心的 Euclidean distance 和 類別資料欄位的相似性，藉此決定該筆資料的分群歸屬。找新群心時，必須混和計算 numeric 資料平均和 categorical 資料的出現次數來擇定新群心。

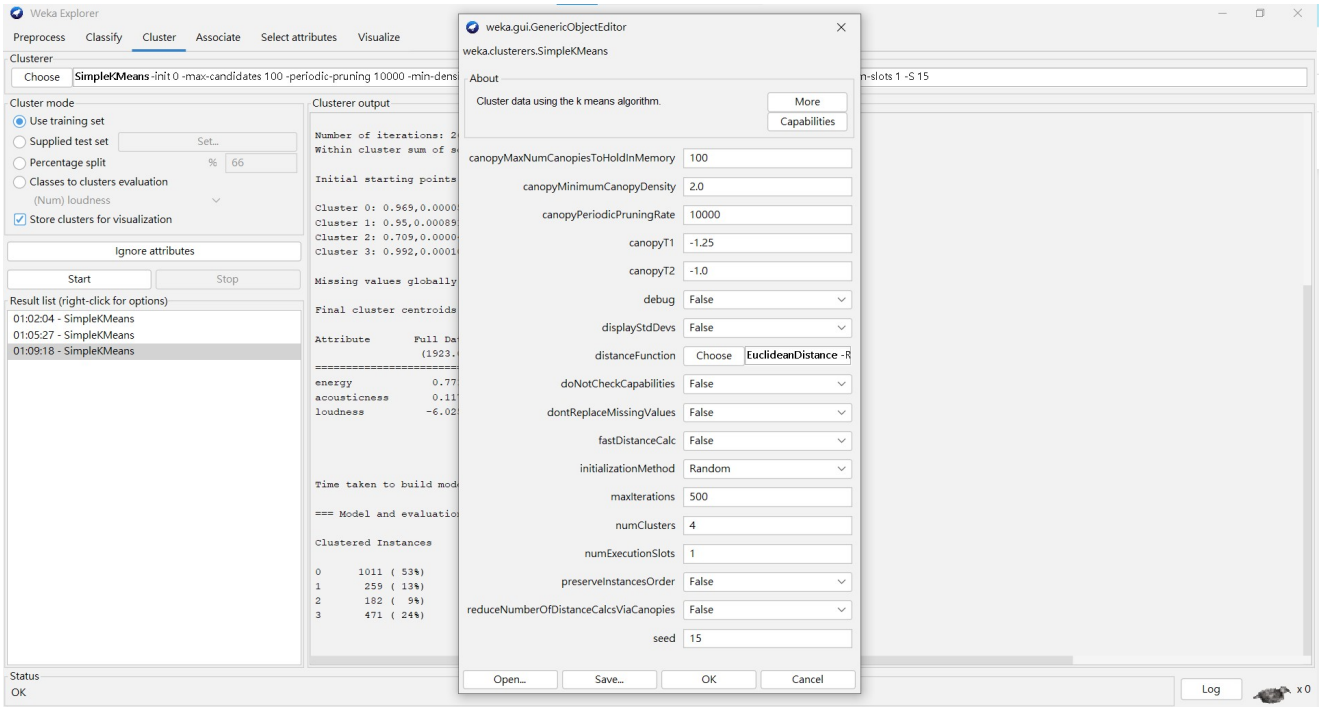
4. K-modes : 分群時，必須通通使用 categorical 的 dtype，所以經過標準化的 numeric 資料還要再經過後續轉換才能做分群。給定起始群心後，它會比較各筆資料類別欄位值跟群心類別欄位值的相似程度決定資料要被分到哪一群，後續過程和 k-means 相同，群心會在每次分類後被重新計算，新群心的類別欄位值會是被分到該群資料欄位值出現次數最頻繁的值，結束計算後開啟下一輪的分群直至資料分群不再變動。

二. 使用 **weka** 做分群

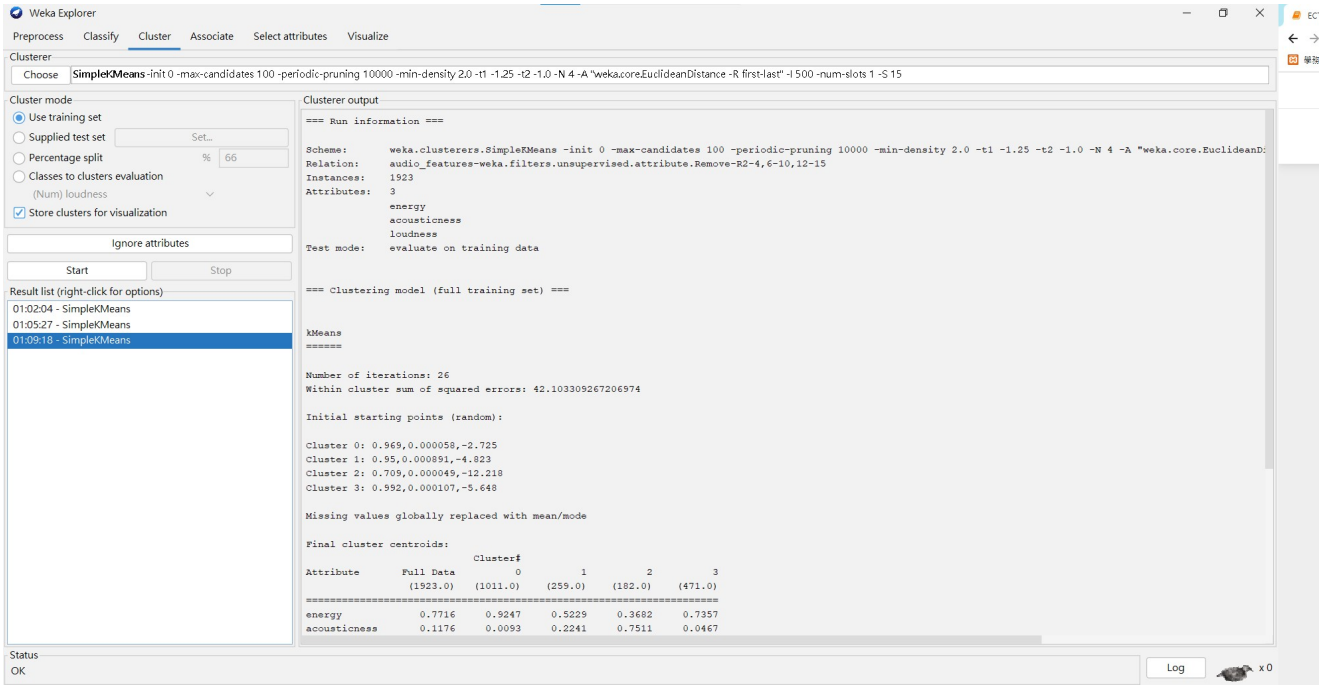
- 載入資料、保留找出的三個欄位



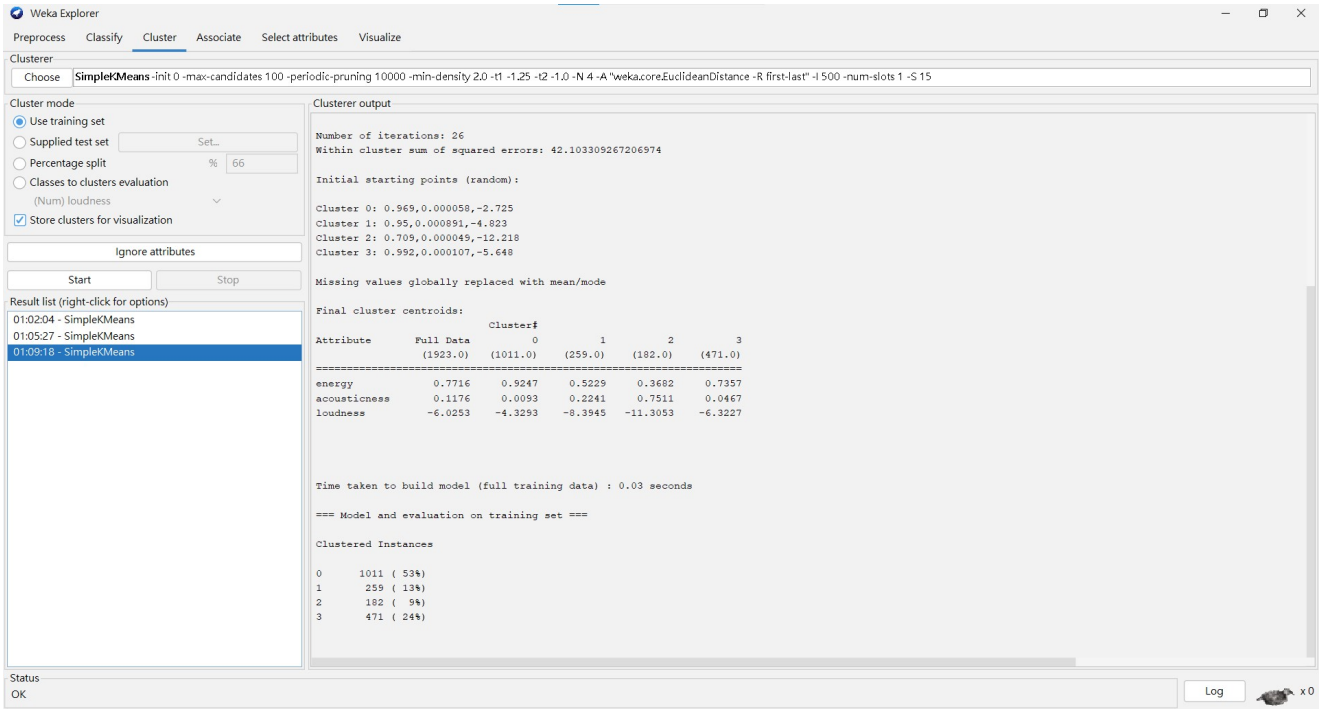
- 設定分群參數


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-dens
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Num) loudness
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
01:02:04 - SimpleKMeans
01:05:27 - SimpleKMeans
01:09:18 - SimpleKMeans
Clusterer output
Number of iterations: 2
Within cluster sum of s
Initial starting points
Cluster 0: 0.969,0.0000
Cluster 1: 0.95,0.00089
Cluster 2: 0.709,0.0000
Cluster 3: 0.992,0.0001
Missing values globally
Final cluster centroids
Attribute
Full Da
(1923.
energy 0.77
acousticness 0.11
loudness -6.02
Time taken to build mod
=== Model and evaluatio
Clustered Instances
0 1011 (53%)
1 259 (13%)
2 182 (9%)
3 471 (24%)
Status
OK
Log
x 0
weka.gui.GenericObjectEditor
weka.clusterers.SimpleKMeans
About
Cluster data using the k means algorithm.
More
Capabilities
canopyMaxNumCanopiesToHoldInMemory 100
canopyMinimumCanopyDensity 2.0
canopyPeriodicPruningRate 10000
canopyT1 -1.25
canopyT2 -1.0
debug False
displayStdDevs False
distanceFunction Choose EuclideanDistance -R
doNotCheckCapabilities False
dontReplaceMissingValues False
fastDistanceCalc False
initializationMethod Random
maxIterations 500
numClusters 4
numExecutionSlots 1
preserveInstancesOrder False
reduceNumberOfDistanceCalcsViaCanopies False
seed 15
Open...
Save...
OK
Cancel

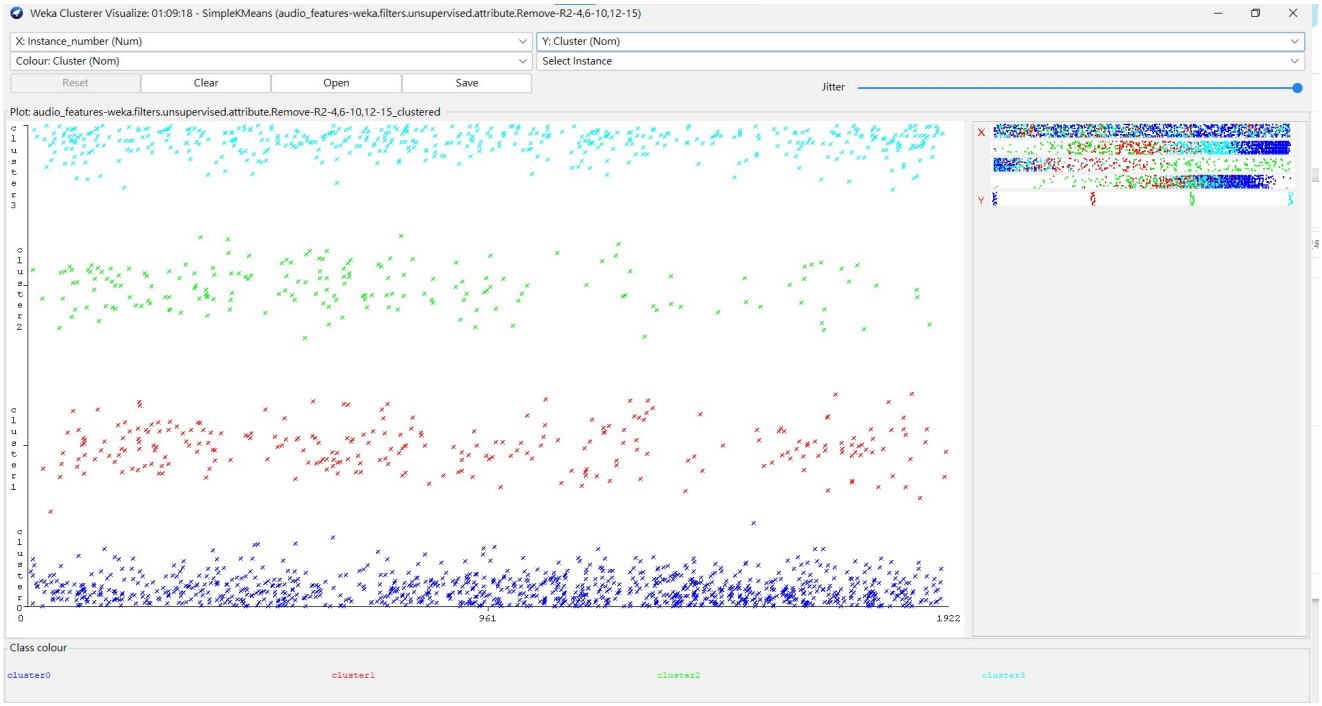
- 分群結果(1)


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 4 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 15
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Num) loudness
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
01:02:04 - SimpleKMeans
01:05:27 - SimpleKMeans
01:09:18 - SimpleKMeans
Clusterer output
=== Run information ===
Scheme: weka.clusterers.SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 4 -A "weka.core.EuclideanD
Relation: audio_features-weka.filters.unsupervised.attribute.Remove-R2-4,6-10,12-15
Instances: 1923
Attributes: 3
energy
acousticness
loudness
Test mode: evaluate on training data
=== Clustering model (full training set) ===
kMeans
======
Number of iterations: 26
Within cluster sum of squared errors: 42.103309267206974
Initial starting points (random):
Cluster 0: 0.969,0.000058,-2.725
Cluster 1: 0.95,0.000891,-4.823
Cluster 2: 0.709,0.000049,-12.218
Cluster 3: 0.952,0.000107,-5.648
Missing values globally replaced with mean/mode
Final cluster centroids:
Cluster#
Attribute Full Data 0 1 2 3
(1923.0) (1011.0) (259.0) (182.0) (471.0)
energy 0.7716 0.9247 0.5229 0.3682 0.7357
acousticness 0.1176 0.0093 0.2241 0.7511 0.0467
Status
OK
Log
x 0

- 分群結果(2)


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Clusterer
Choose
SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 4 -A "weka.core.EuclideanDistance -R first-last" -I 500 -num-slots 1 -S 15
Cluster mode
Use training set
Supplied test set
Set...
Percentage split
%
66
Classes to clusters evaluation
(Num) loudness
Store clusters for visualization
Ignore attributes
Start
Stop
Result list (right-click for options)
01:02:04 - SimpleKMeans
01:05:27 - SimpleKMeans
01:09:18 - SimpleKMeans
Clusterer output
Number of iterations: 26
Within cluster sum of squared errors: 42.103309267206974
Initial starting points (random):
Cluster 0: 0.969,0.000058,-2.725
Cluster 1: 0.95,0.000891,-4.823
Cluster 2: 0.709,0.000049,-12.218
Cluster 3: 0.992,0.000107,-5.648
Missing values globally replaced with mean/mode
Final cluster centroids:
Cluster#
Attribute Full Data 0 1 2 3
(1923.0) (1011.0) (259.0) (182.0) (471.0)
energy 0.7716 0.9247 0.5229 0.3682 0.7357
acousticness 0.1176 0.0093 0.2241 0.7511 0.0467
loudness -6.0253 -4.3293 -8.3945 -11.3053 -6.3227
Time taken to build model (full training data) : 0.03 seconds
=== Model and evaluation on training set ===
Clustered Instances
0 1011 (53%)
1 259 (13%)
2 182 (9%)
3 471 (24%)
Status
OK
Log
x 0

- 分群結果視覺化


Weka Clusterer Visualize: 01:09:18 - SimpleKMeans (audio_features-weka.filters.unsupervised.attribute.Remove-R2-4,6-10,12-15)
X: Instance_number (Num)
Y: Cluster (Nom)
Colour: Cluster (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: audio_features-weka.filters.unsupervised.attribute.Remove-R2-4,6-10,12-15_clustered
Cluster3
Cluster2
Cluster1
Cluster0
0
961
1922
X
Y
Class colour
cluster0
cluster1
cluster2
cluster3