ETC_HW4_107403020 李泳輝

一、Python

1. 載入資料並刪除除了"energy", "speechiness", acousticness","instrumentalness", "loudness", "tempo","danceability",'valence', "liveness"以外之欄位(使用 pandas dataframe) (2%)

In [4]:
```
df = pd.read_csv("wu_songs.csv")
df.head()
```

Out[4]:

	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_feat
1	0.861	0.0747	100.080	0.0493	0.001890	0.000539	4	0.518	6	231762	-6.998	0.317	0	audio_feat
2	0.963	0.2030	195.979	0.1590	0.000090	0.000304	4	0.356	9	217959	-4.385	0.379	1	audio_feat
3	0.968	0.1060	158.089	0.1370	0.000047	0.000006	4	0.365	2	198987	-4.267	0.386	1	audio_feat
4	0.327	0.0922	75.122	0.0489	0.605000	0.000012	4	0.434	6	216100	-10.161	0.260	0	audio_feat

In [5]:
```
df = df[["energy", "speechiness", "acousticness","instrumentalness", "loudness","tempo","danceability",'valence', "liveness"]]
df.head()
```

Out[5]:

	energy	speechiness	acousticness	instrumentalness	loudness	tempo	danceability	valence	liveness
0	0.979	0.1220	0.007620	0.015200	-3.481	128.876	0.410	0.080	0.2720
1	0.861	0.0493	0.001890	0.000539	-6.998	100.080	0.518	0.317	0.0747
2	0.963	0.1590	0.000090	0.000304	-4.385	195.979	0.356	0.379	0.2030
3	0.968	0.1370	0.000047	0.000006	-4.267	158.089	0.365	0.386	0.1060
4	0.327	0.0489	0.605000	0.000012	-10.161	75.122	0.434	0.260	0.0922

2. 將剩下的欄位做特徵篩選的動作，並使用 kmeans silhouette analysis 的方法，找出在哪三個欄位的情況下(需考慮所有組合)，分 X 群會有最高的 silhouettescore。請找出 X 與 silhouette score 還有是哪三個欄位。(20%)

✧ 建立相關 function

In [7]:
```
from itertools import combinations
candidate_list = list(combinations(df.columns, 3))
```

標準化

In [8]:
```
from sklearn.preprocessing import StandardScaler
```

In [9]:
```
def standard_data(df):
    scaler = StandardScaler()
    scaler.fit(df)
    return scaler.transform(df)
```

kmeans & silhouette

In [10]:
```
from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [11]:
```
def do_kmeans_silhouette_analysis(X):
    silhouette_avg = []
    for i in range(2,11):
        kmeans_fit = KMeans(n_clusters = i, random_state=15).fit(X)
        silhouette_avg.append(silhouette_score(X, kmeans_fit.labels_))
    plt.plot(range(2,11), silhouette_avg)
    plt.xlabel("K")
    plt.ylabel("silhouette_score")
```

In [12]:
```
def count_kmeans_silhouette_analysis(X):
    highest_siloutte = 0
    K = 2
    for i in range(2, 13):
        kmeans_fit = KMeans(n_clusters = i, random_state=15).fit(X)
        score = silhouette_score(X, kmeans_fit.labels_)
        if highest_siloutte < score:
            highest_siloutte = score
            K = i
    return highest_siloutte, K
```

Find Hiegest Siloutte

In [13]:
```
def find_hiegest_siloutte(df, candidate):
    strategy = ClusterStratge(0, 0, 0)
    for i in range(0, len(candidate)):
        columns = list(candidate[i])
        X = standard_data(df[columns])
        score, K = count_kmeans_silhouette_analysis(X)
        if strategy.getScore() < score:
            strategy.setStrategy(columns, K, score)
    return strategy
```

In [14]:
```
def draw_features_cluster(df, list):
    X = standard_data(df[list])
    do_kmeans_silhouette_analysis(X)
```

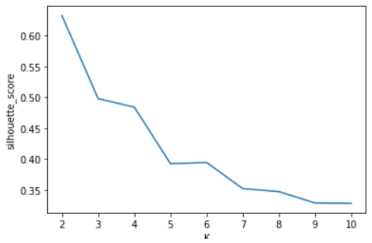
✧ 執行

In [15]:
```
strategy = find_hiegest_siloutte(df, candidate_list)
```

In [16]:
```
columns , K, score = strategy.getStratege()
print('columns: {}\nK={}\nsilhouette score: {}\n'.format(columns, K, score))
```

```
columns: ['energy', 'acousticness', 'loudness']
K=2
silhouette score: 0.6322586386208744
```

In [17]:
```
draw_features_cluster(df, strategy.getColumns())
```



請解釋 silhouette 分析法 與 elbow 轉折判斷法的差別(3%)

✧ elbow 轉折判斷法是針對不同 K 的分群，計算樣本點到各自分群中心距離的總和。當分群的數目增加，距離的總和自然會下降。當 K 值大於真正的 K 值後，下降就不會那麼明顯。這樣正確的 K 值就會在這個轉折點上。但若遇到某些特例，像是下降幅度趨緩的圖，就不適合用 elbow 判斷方式。

✧ silhouette 分析法會衡量物件與所屬 cluster 的相似度，即內聚性。silhouette 的分數越高代表著物件與所屬的 cluster 有越密切的關聯。silhouette 的缺點是他的計算複雜度為 O(n^2)，若資料量上到百萬，計算算量會非常巨大。

3. 使用剛剛找出來的欄位用 k-means 做分群。超參數設定為 n_cluster=4 ,random_state=15。並使用 plotly 繪製出 3d 圖形如以下所示(15%)：

In [18]:
```
selected_df = df[columns]
X = standard_data(selected_df)
```

In [19]:
```
kmeans = KMeans(n_clusters=4, random_state=15).fit(X)
y_clusters = kmeans.fit_predict(X)
```

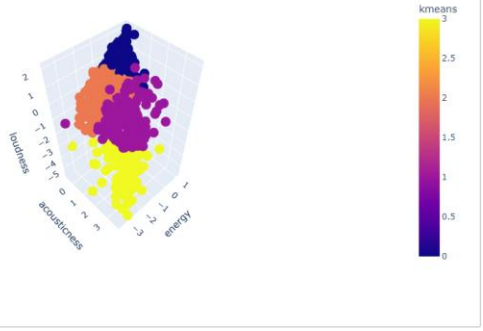
In [20]:
```
DF = pd.DataFrame(data = standard_data(selected_df), columns=['energy', 'acousticness', 'loudness'])
DF['kmeans'] = y_clusters
DF.head()
```

Out[20]:

	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

In [23]:
```
import plotly.express as px

fig = px.scatter_3d(DF, y='energy', z='loudness', x='acousticness',
                    color='kmeans')
fig.show()
```



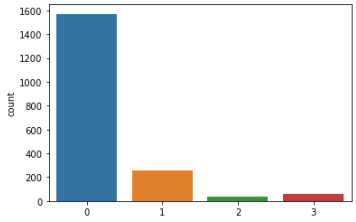
4. 使用剛剛找出來的欄位用 Meanshift 做分群(15%) 請找出最佳的 estimate_bandwidth. 超參數設定為 random_state=15,quantile=0.32, n_samples=1000

In [24]:
```
from sklearn.cluster import MeanShift
from sklearn.cluster import estimate_bandwidth
```

In [25]:
```
bandwidth = estimate_bandwidth(X, quantile=0.32, random_state=15, n_samples=1000)
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True).fit(X)
y_clusters = ms.predict(X)
```

In [26]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
sns.countplot(y_clusters)
```

Out[26]: <AxesSubplot:ylabel='count'>



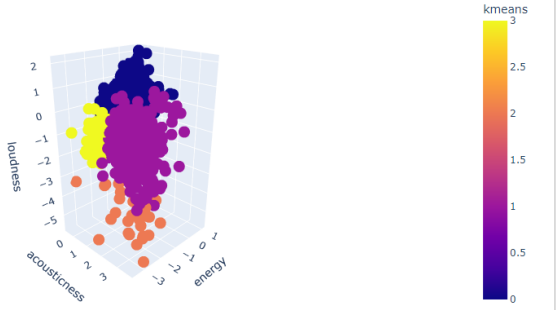
In [27]:
```
DF = pd.DataFrame(data = standard_data(selected_df), columns=['energy', 'acousticness', 'loudness'])
DF['kmeans'] = y_clusters
DF.head()
```

Out[27]:

	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

In [28]:
```
import plotly.express as px

fig = px.scatter_3d(DF, y='energy', z='loudness', x='acousticness',
                    color='kmeans')
fig.show()
```

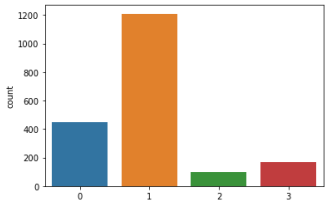


5. 使用剛剛找出來的欄位用 k-prototypes 做分群。超參數設定為 n_cluster=4, random_state=15,init=’ Huang’ ,verbose=0。並使用 plotly 繪製出 3d 圖形如第三題的圖(15%)

In [30]:
```
from kmodes.kprototypes import KPrototypes
kp = KPrototypes(n_clusters=4, random_state=15, init='Huang', verbose=0)
y_clusters = kp.fit_predict(X, categorical=[0])
```

In [31]:
```
warnings.simplefilter(action='ignore', category=FutureWarning)
sns.countplot(y_clusters)
```

Out[31]: <AxesSubplot:ylabel='count'>



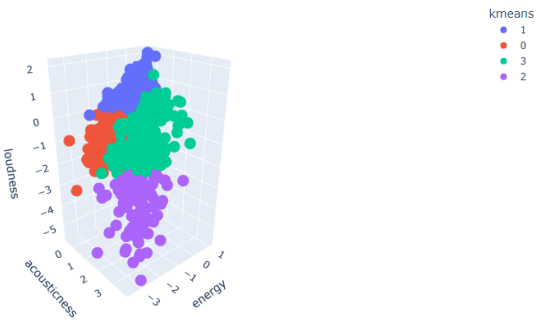
In [32]:
```
DF = pd.DataFrame(data = standard_data(selected_df), columns=['energy', 'acousticness', 'loudness'])
DF['kmeans'] = y_clusters
DF.head()
```

Out[32]:

	energy	acousticness	loudness	kmeans
0	0.999384	-0.472349	0.871404	1
1	0.430682	-0.496959	-0.333147	0
2	0.922272	-0.504691	0.561790	1
3	0.946369	-0.504872	0.602204	1
4	-2.142935	2.093298	-1.416456	3

In [33]:
```
import plotly.express as px

fig = px.scatter_3d(DF, y='energy', z='loudness', x='acousticness',
                    color='kmeans')
fig.show()
```

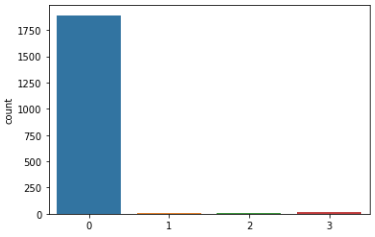


6. 使用剛剛找出來的欄位用 k-modes 做分群。超參數設定為 n_cluster=4,random_state=15,init=’ Huang’ ,verbose=0。並使用 plotly 繪製出 3d 圖形如第三題的圖(15%)

```
In [34]: from kmodes.kmodes import KModes
         kmode = KModes(n_clusters=4, random_state=15, init='Huang', verbose=0)
         y_clusters = kmode.fit_predict(X)

In [35]: warnings.simplefilter(action='ignore', category=FutureWarning)
         sns.countplot(y_clusters)

Out[35]: <AxesSubplot:ylabel='count'>
```



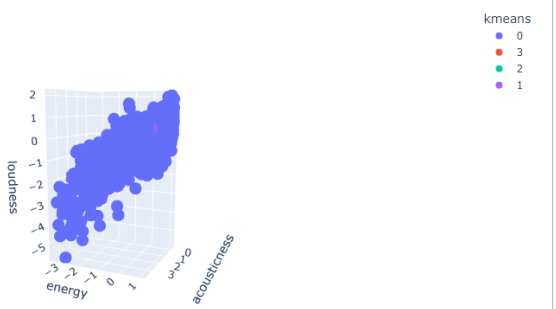

```
In [36]: DF = pd.DataFrame(data = standard_data(selected_df), columns=['energy', 'acousticness', 'loudness'])
         DF['kmeans'] = y_clusters
         DF.head()
```

Out[36]:

	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	3
4	-2.142935	2.093298	-1.416456	0

```
In [37]: import plotly.express as px

         fig = px.scatter_3d(DF, y='energy', z='loudness', x='acousticness',
                             color='kmeans')
         fig.show()
```

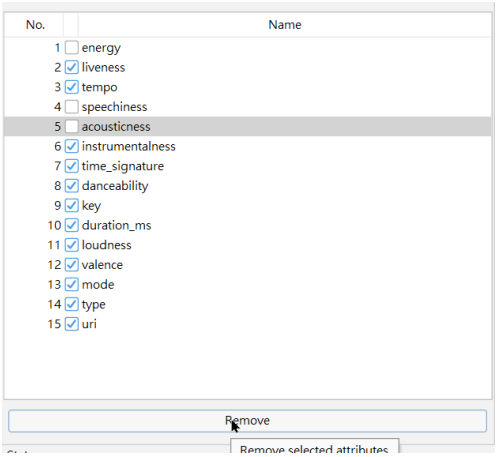


7. 請比較說明上述四種分群法的差異(5%)

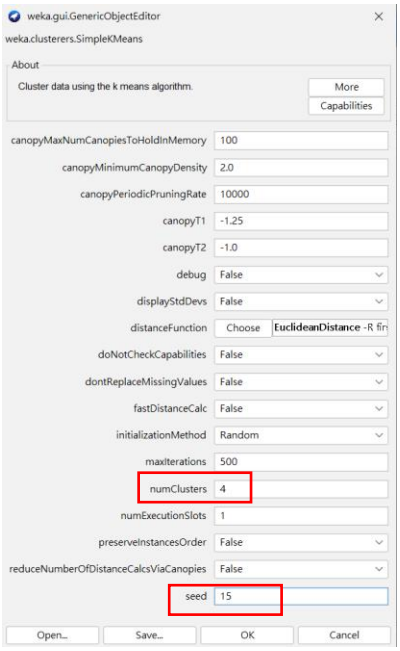
- ✧ K-Means 演算法中，最終的 cluster 結果容易受到初始中心的影響。
- ✧ Mean Shift 演算法與 K-Means 一樣是基於 Cluster 中心的演算法，但 Mean Shift 演算法不需要事先制定類別個數 K。
- ✧ K-modes 演算法可以看做 K-means 算法在非數值屬性集合的版本，將原本 K-Means 使用的歐基里德距離替換成相異程度的算法。
- ✧ K-prototypes 散與數值屬性兩種混合的數據進行 clustering，其結合了 K-Means 與 K-modes 算法。

二、Weka

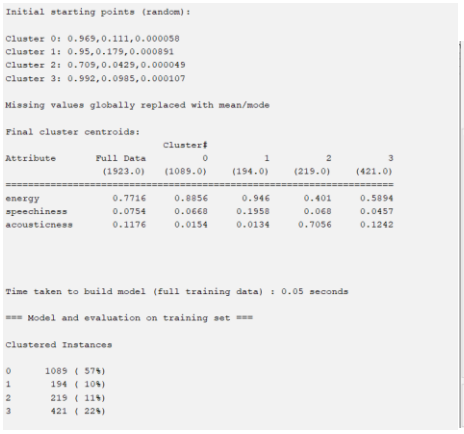
- 只留下要用的三個欄位



- 設定 SimpleKMeans 超參數



- 執行結果



```
Initial starting points (random):

Cluster 0: 0.969,0.111,0.000058
Cluster 1: 0.95,0.179,0.000891
Cluster 2: 0.709,0.0429,0.000049
Cluster 3: 0.992,0.0985,0.000107

Missing values globally replaced with mean/mode

Final cluster centroids:
                        Cluster#
Attribute       Full Data          0          1          2          3
                  (1923.0)   (1089.0)   (194.0)   (219.0)   (421.0)
=============================================================================
energy              0.7716     0.8856     0.946     0.401     0.5894
speechiness         0.0754     0.0668    0.1958     0.068     0.0457
acousticness        0.1176     0.0154    0.0134    0.7056     0.1242



Time taken to build model (full training data) : 0.05 seconds

=== Model and evaluation on training set ===

Clustered Instances

0      1089 ( 57%)
1       194 ( 10%)
2       219 ( 11%)
3       421 ( 22%)
```