41071102H 徐敏皓 HW8

Code:
https://colab.research.google.com/drive/1Oi1Zq3j2L38Wx6TIsdKesuySt
pLkFIn9?usp=sharing

Kmeans

1. Find optimal k by calculating minimum SSE

```
from sklearn.cluster import KMeans
# 5. 使用 Elbow Method 計算不同 n_clusters 的 SSE
sse = []  # 存放每個 n_clusters 對應的 SSE
cluster_range = range(1, 16)  # 測試 1 到 15 個聚類
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42)  # 設定隨機種子以確保重現性
    kmeans.fit(reduced_features)
    sse.append(kmeans.inertia_)  # KMeans 的 inertia_ 屬性即為 SSE
```

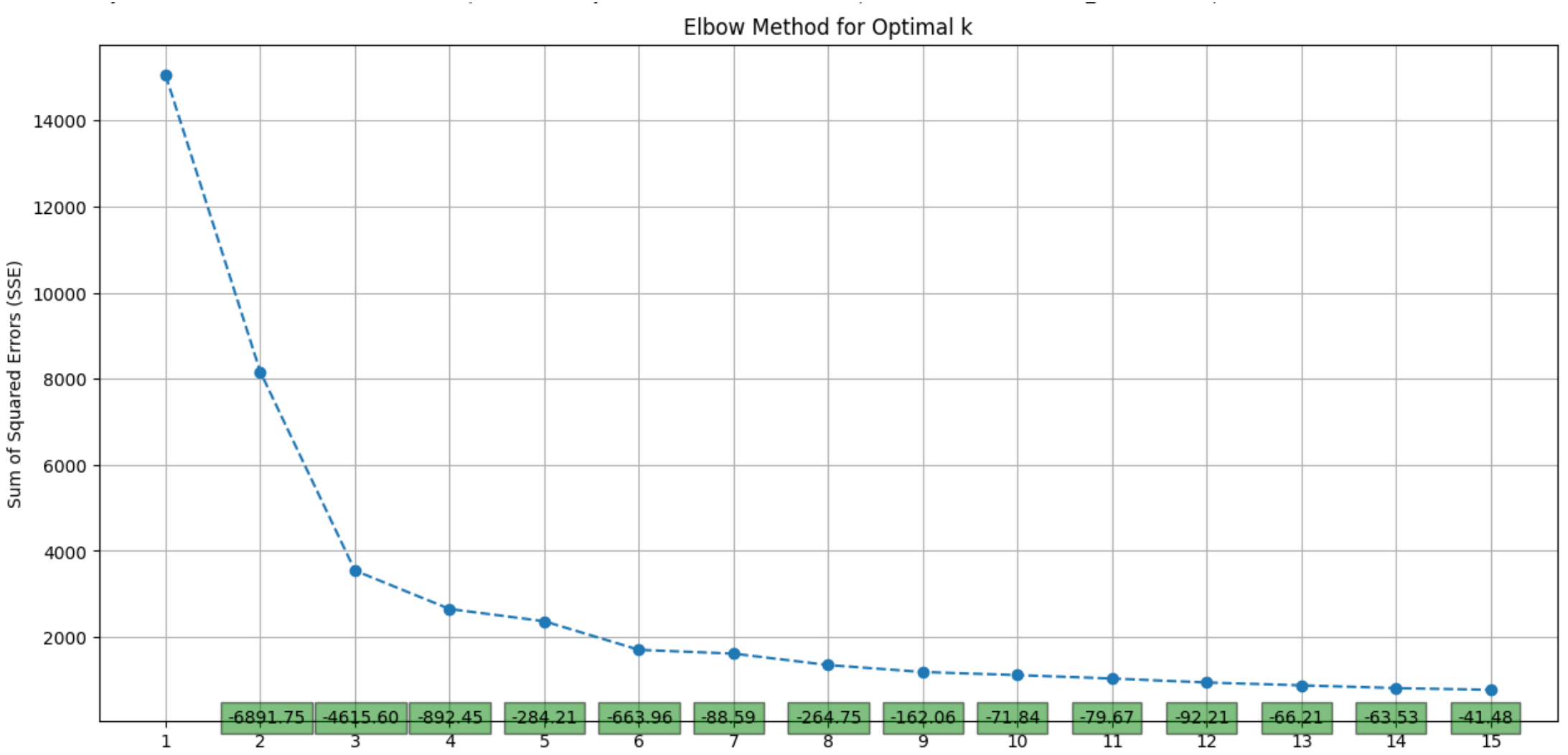
資料前處理:

1. 如果特徵的數值範圍差異很大，範圍較大的特徵會主導距離計算，導致分群結果偏向這些特徵。因此須將特徵標準化。

2. 為了視覺化有使用PCA降至二維。

Kmeans

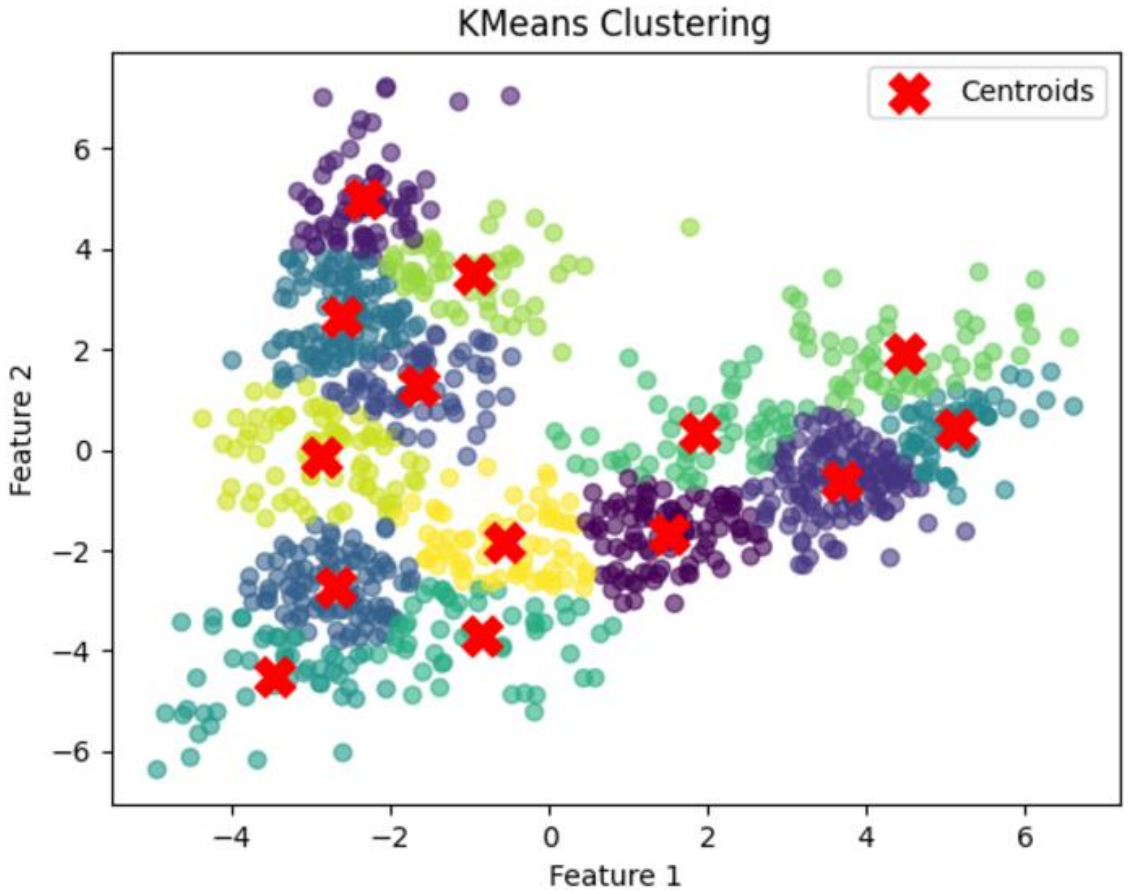
1. Find optimal k by calculating minimum SSE



由圖中可知最佳的k = 14

Kmeans

2. Using Kmeans to group and visualize the result



分群結果（共14群）

呼叫方式

```
from sklearn.cluster import KMeans

# 7. 設定 KMeans 模型
kmeans = KMeans(n_clusters=optimal_k)


# 8. 擬合模型
kmeans.fit(reduced_features)
```

- n_clusters: 指定要分的群數
- random_state: 隨機種子

AgglomerativeClustering

1. Find optimal n_clusters by calculating minimum SSE

```
from sklearn.cluster import AgglomerativeClustering
sse = []
cluster_range = range(1, 16)
for n in cluster_range:    # 測試簇數從 1 到 15
        clustering = AgglomerativeClustering(n_clusters=n, linkage='ward')
        labels = clustering.fit_predict(reduced_features)

        # 計算 SSE
        cluster_centers = np.array([reduced_features[labels == i].mean(axis=0) for i in range(n)])
        distances = np.min(cdist(reduced_features, cluster_centers, 'euclidean'), axis=1)
        sse.append(np.sum(distances**2))
```

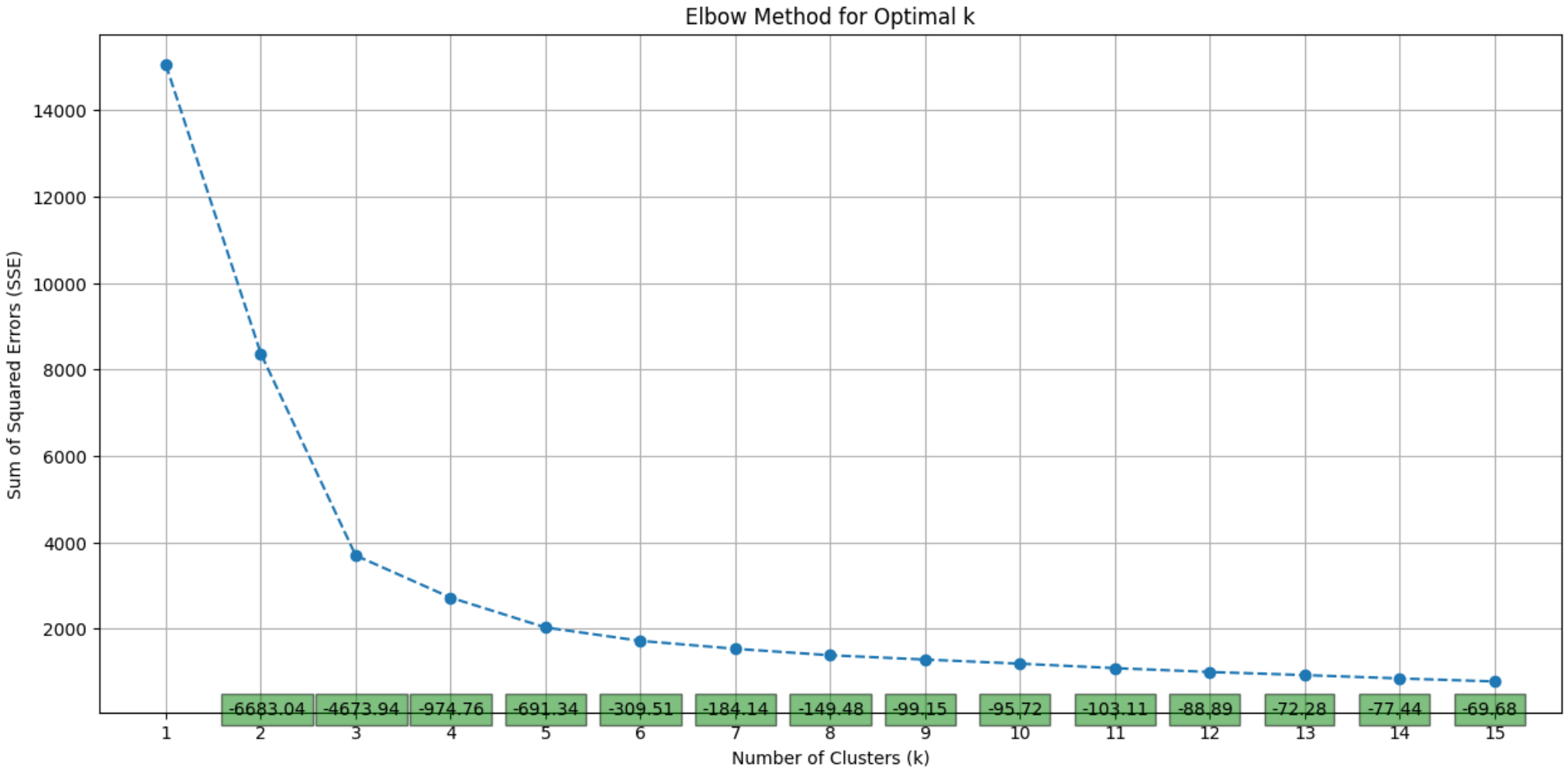
資料前處理:

1. 如果特徵的數值範圍差異很大，範圍較大的特徵會主導距離計算，導致分群結果偏向這些特徵。因此須將特徵標準化。

2. 為了視覺化有使用PCA降至二維。

AgglomerativeClustering

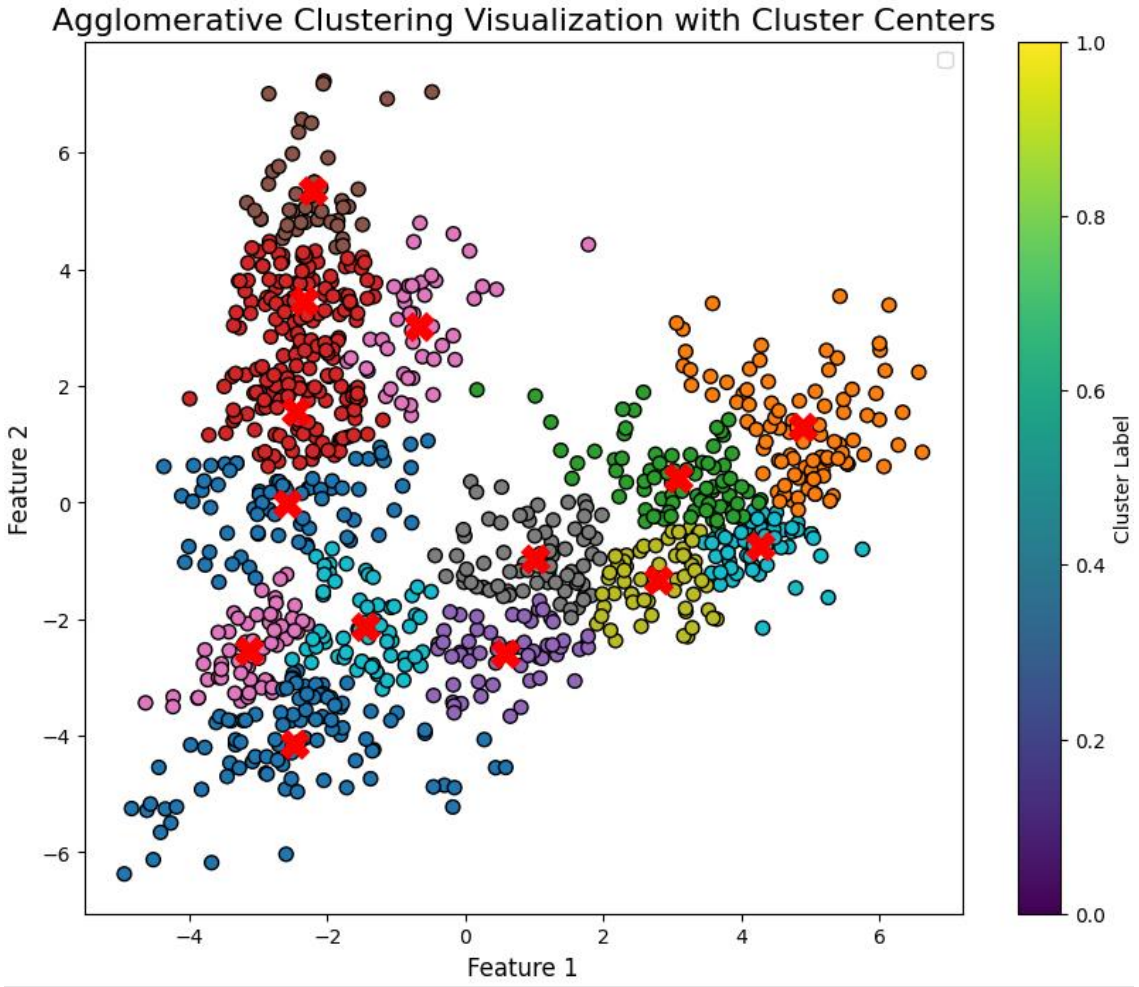
1. Find optimal n_clusters by calculating minimum SSE



由圖中可知最佳的k = 14

AgglomerativeClustering

2. Using AgglomerativeClustering to group and visualize the result



分群結果（共14群）

呼叫方式

```
from sklearn.cluster import AgglomerativeClustering
# 5. 使用 AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=optimal_k, linkage='ward')

# 6. 擬合模型
clustering.fit(reduced_features)
```

- n_clusters: 指定要分的群數
- linkage:用於合併cluster的方式

DBSCAN

1. Determining EPS and Minpts

```
from  sklearn.neighbors  import  NearestNeighbors
#  計算每個點的  k  邊界距離（min_samples  設為  4）
min_samples  =  4
neighbors  =  NearestNeighbors(n_neighbors=min_samples)
neighbors_fit  =  neighbors.fit(reduced_features)
distances,  indices  =  neighbors_fit.kneighbors(reduced_features)
```

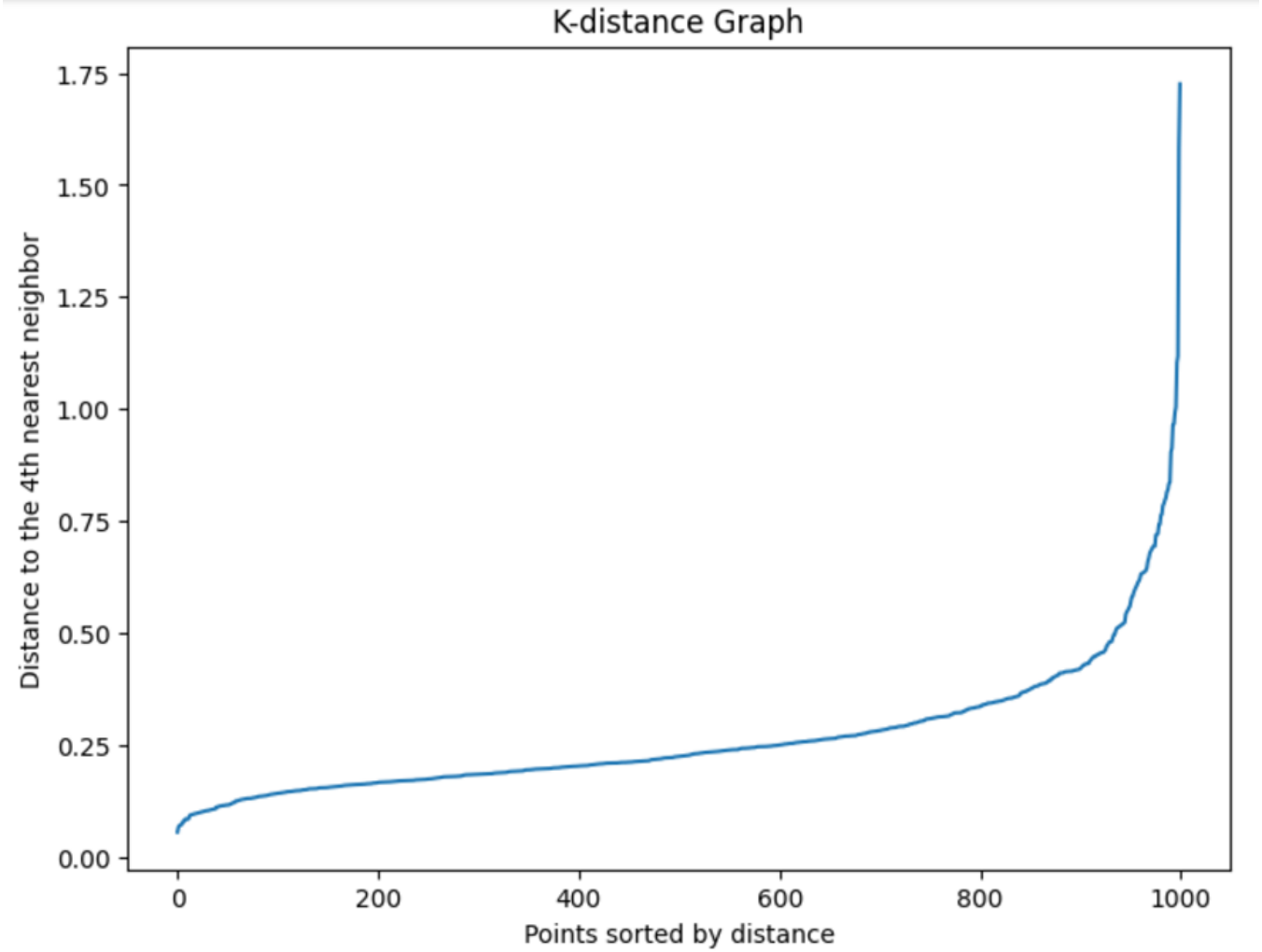
資料前處理:

1. 如果特徵的數值範圍差異很大，範圍較大的特徵會主導距離計算，導致分群結果偏向這些特徵。因此須將特徵標準化。

2. 為了視覺化有使用PCA降至二維。

DBSCAN

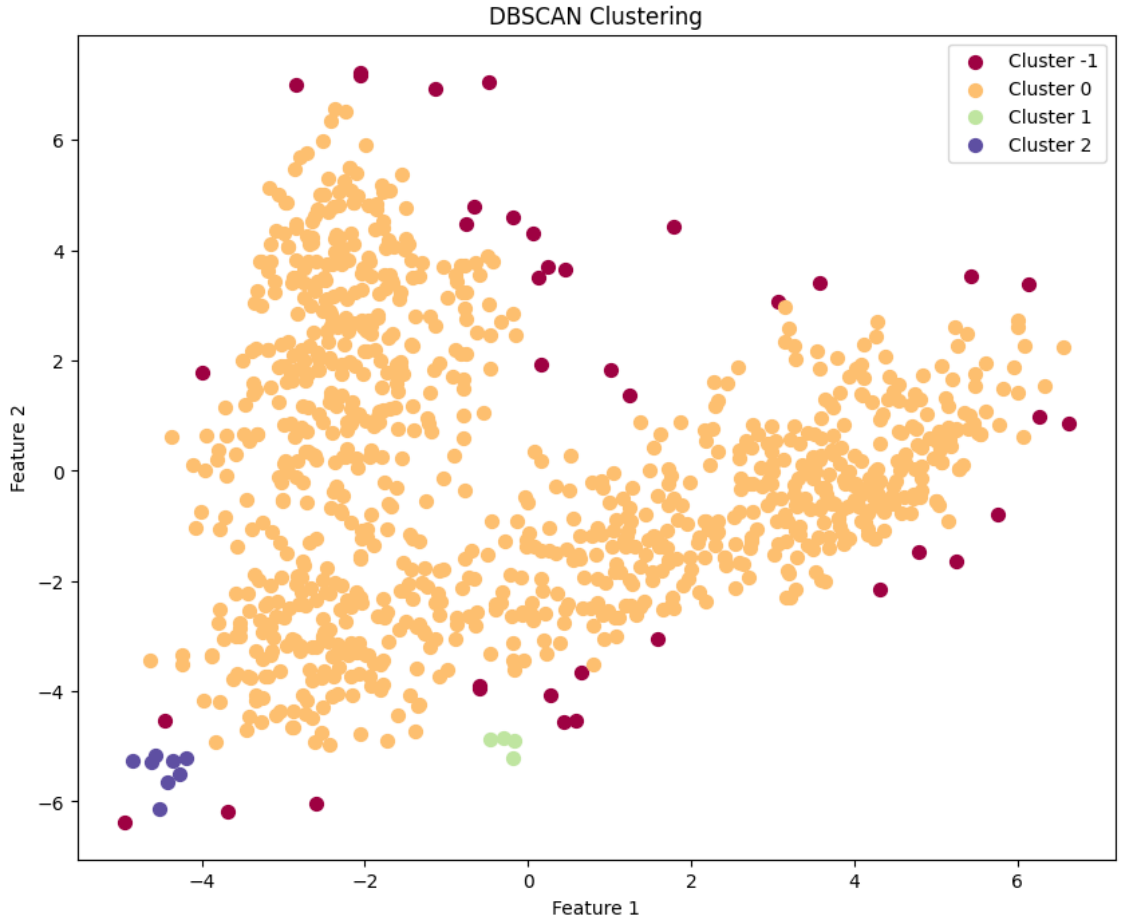
1. Determining EPS and Minpts



由圖中可知最佳的EPS落在0.25~0.5之間

DBSCAN

2. Using DBSCAN to group and visualize the result



分群結果

呼叫方式

```
from sklearn.cluster import DBSCAN
# 應用 DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=min_samples)
labels = dbscan.fit_predict(reduced_features)
```

- eps:兩個樣本被認為是鄰居的最大距離（即鄰域的半徑）
- min_samples:形成一個核心點所需的最少鄰居數量

normalized_mutual_info_score

呼叫方式

```
from  sklearn.metrics.cluster  import  normalized_mutual_info_score
#  計算正規化互信息分數
nmi_score  =  normalized_mutual_info_score(labels_true  =  true_labels,  labels_pred  =  predicted_labels,  average_method='arithmetic')
```

- labels_true:類別標籤的真實值
- labels_pred:類別標籤的預測值
- average_method:指定正規化的計算方式

得到的結果會介於[0, 1]。

1表示完全一致（分群結果與真實標籤完全匹配）。

0表示完全無關（分群結果與真實標籤沒有任何關聯）。

silhouette_score

呼叫方式

```
from  sklearn.metrics  import  silhouette_score
#  計算正規化互信息分數
score    =  silhouette_score(X  =  features,  labels  =  predicted_labels,  metric='euclidean')
```

- X: 特徵數據或距離矩陣
- labels: 群集標籤
- metric: 計算相似度的度量方式

得到的結果會介於[-1, 1]。

越接近1表示群內緊密，群間分離良好，分群效果好。

越接近0表示數據點位於兩群之間的邊界，分群效果中等。

越接近-1表示數據被錯誤分群，效果差。

分群效果評估（normalized_mutual_info_score）

實作片段，其中兩者分群的數量皆為14群

```
from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics.cluster import normalized_mutual_info_score
```

```
# 5. 設定 KMeans 模型
optimal_k = 14
kmeans = KMeans(n_clusters=optimal_k)
```

```
# 7. 計算正規化互信息分數
nmi_Kmeans = normalized_mutual_info_score(labels_true = true_labels, labels_pred = predicted_labels, average_method='arithmetic')
```

```
# 8. 使用 AgglomerativeClustering
clustering = AgglomerativeClustering(n_clusters=optimal_k, linkage='ward')
```

```
# 10. 計算正規化互信息分數
nmi_AgglomerativeClustering = normalized_mutual_info_score(true_labels, predicted_labels, average_method='arithmetic')
```

下圖為執行結果

兩者分群效果差不多，但都不佳

```
KMeans NMI: 0.291, AgglomerativeClustering NMI: 0.286
```

分群效果評估
（silhouette_score）

實作片段，其中Kmeans分群的數量皆為14群，DBSCAN的eps為0.5，Minpts為4

```
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
    # 5. 設定 KMeans 模型
    optimal_k = 14
    kmeans = KMeans(n_clusters=optimal_k)
# 7. 計算正規化互信息分數
silhouette_Kmeans = silhouette_score(X = reduced_features, labels = predicted_labels, metric='euclidean')
                    # 8. 應用 DBSCAN
                    min_samples = 4
                    dbscan = DBSCAN(eps=0.5, min_samples=min_samples)
# 10. 計算正規化互信息分數
silhouette_DBSCAN = silhouette_score(X = reduced_features, labels = predicted_labels, metric='euclidean')
```

下圖為執行結果

Kmeans分群效果比DBSCAN好

```
KMeans Silhouette: 0.345, DBSCAN Silhouette: -0.002
```