LAB 06 - Clustering Methods - Comparison ¶

SUBMITTED BY:

Name : Kavitha.S
Reg no : 21122033
Class : 2MSDS

LAB OVERVIEW:

Illustrate KMeans and Agglomerative Hierarchical Clustering on Iris Dataset, considering only two features - Sepal Length and Petal Width.
Use Elbow Method as a way to find optimum number of clusters

PROBLEM DEFINITION:

To perform K-Means and Agglomerative Hierarchial Clustering on the in-built iris dataset. To find the optimum number of clusters.

APPROACH:

Imported the necessary packages. Analyzed the dataset, drop the unneseccary columns. performed K-Means clustering on 2 features with various number of clusters. Found the optimum number of clusters using Elbow method. For Agglomerative hierarchial clustering, found the optimum number of clusters using silhouette score.

In [97]:
```python
from sklearn import datasets
import pandas as pd
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
import seaborn as sns
```

In [3]:
```python
iris = datasets.load_iris()
```

In [11]:
```python
print("Features:",iris.feature_names)
print("\nTarget:",iris.target_names)
```

```
Features: ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']

Target: ['setosa' 'versicolor' 'virginica']
```

In [56]:
```
iris_df=pd.DataFrame(iris.data)
iris_df['Species']=iris.target
iris_df.head()
```

Out[56]:

	0	1	2	3	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [57]:
```
x = iris_df.iloc[:, [0, 1, 2, 3]].values
```

In [59]:
```
iris_df.columns=['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', '
iris_df.head()
```

Out[59]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [60]:
```
iris_df.describe()
```

Out[60]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [61]:
```
iris_df.shape
```

Out[61]: (150, 5)

In [62]:
```
iris_df = iris_df.drop(['SepalWidth','PetalLength','Species'],axis=1)
```

In [63]:
```
iris_df.head()
```

Out[63]:

	SepalLength	PetalWidth
0	5.1	0.2
1	4.9	0.2
2	4.7	0.2
3	4.6	0.2
4	5.0	0.2

In [64]:
```
iris_df.SepalLength = iris_df.SepalLength.reshape(1, -1)
```

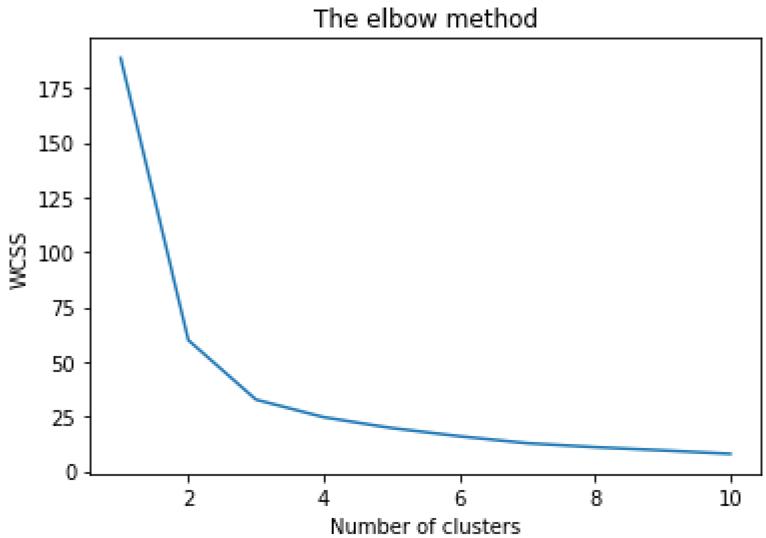
...

In [65]:
```
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_in
    kmeans.fit(iris_df)
    wcss.append(kmeans.inertia_)
```

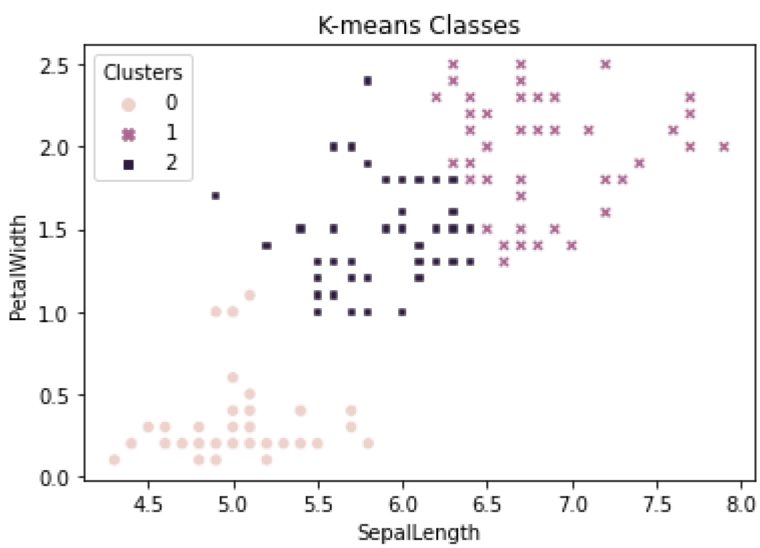
```
C:\Users\SRIDHAR\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:881: Us
erWarning: KMeans is known to have a memory leak on Windows with MKL, when ther
e are less chunks than available threads. You can avoid it by setting the envir
onment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [66]:
```
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



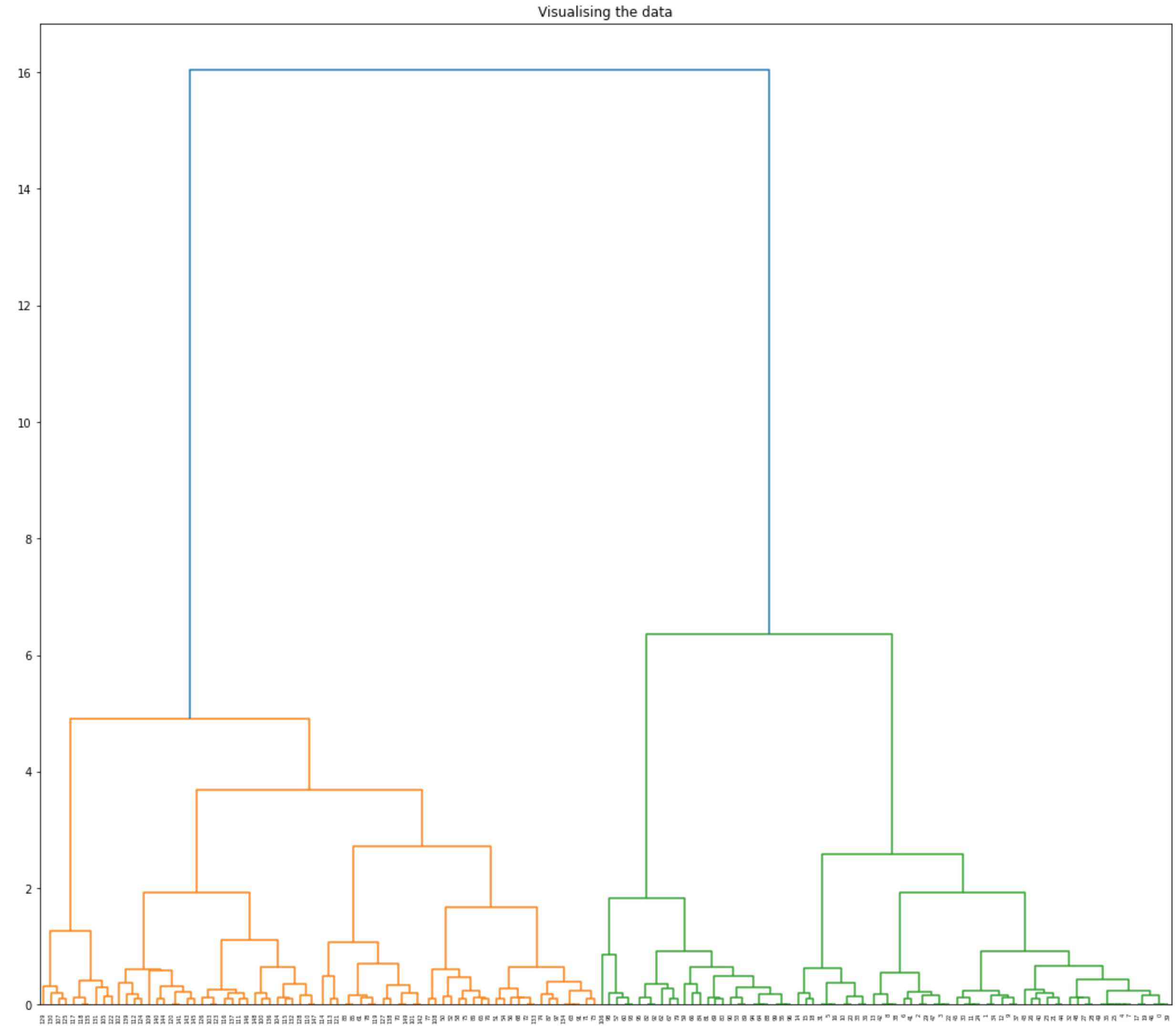
In [69]:
```
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init =
identified_clusters = kmeans.fit_predict(iris_df)
```

In [102]:
```
plt.title('K-means Classes')
sns.scatterplot(x=data_with_clusters['SepalLength'], y=data_with_clusters['P
plt.show()
```



In [74]:

```
plt.figure(figsize =(18, 16))
plt.title('Visualising the data')
Dendrogram = shc.dendrogram((shc.linkage(iris_df, method ='ward')))
```



In [ ]:

```
df_h = df.copy(deep=True)
df_h['labels_h'] = cluster2
```

In [80]:
```
ac2 = AgglomerativeClustering(n_clusters = 2)
ac2.fit_predict(iris_df)
```

Out[80]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], dtype=int64)
```

In [81]:
```
ac3 = AgglomerativeClustering(n_clusters = 3)
ac3.fit_predict(iris_df)
```

Out[81]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2, 0,
       2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 2, 0, 0, 0,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [82]:
```
ac4 = AgglomerativeClustering(n_clusters = 4)
ac4.fit_predict(iris_df)
```

Out[82]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2, 0,
       2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 2, 0, 0, 0,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 3, 2, 3, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 3, 0, 0, 3, 0, 0, 0, 3, 3, 3,
       0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

In [83]:
```
ac5 = AgglomerativeClustering(n_clusters = 5)
ac5.fit_predict(iris_df)
```

Out[83]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2, 0,
       2, 2, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 2, 0, 0, 0,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 4, 0, 4, 4, 4, 3, 2, 3, 0, 4,
       4, 4, 4, 0, 0, 4, 4, 3, 3, 0, 4, 0, 3, 4, 4, 3, 4, 0, 4, 3, 3, 3,
       4, 0, 0, 3, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4, 4, 4, 4, 0], dtype=int64)
```

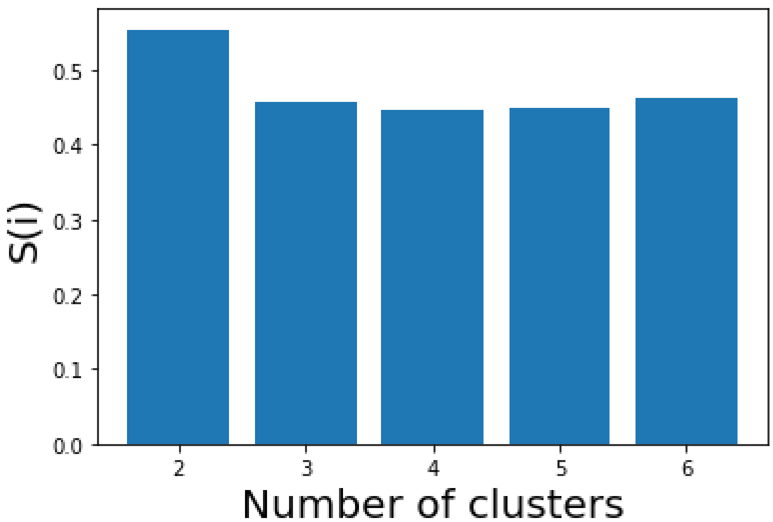
In [84]:
```
ac6 = AgglomerativeClustering(n_clusters = 6)
ac6.fit_predict(iris_df)
```

Out[84]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 4, 4, 4, 2, 4, 2, 4, 2, 4, 2, 2, 5, 2, 4, 2, 4,
       2, 2, 4, 2, 5, 4, 4, 4, 4, 4, 4, 4, 5, 2, 2, 2, 2, 5, 2, 5, 4, 4,
       2, 2, 2, 4, 2, 2, 2, 2, 2, 4, 2, 2, 1, 5, 1, 1, 1, 3, 2, 3, 4, 1,
       1, 1, 1, 5, 5, 1, 1, 3, 3, 5, 1, 5, 3, 1, 1, 3, 1, 5, 1, 3, 3, 3,
       1, 4, 4, 3, 1, 1, 5, 1, 1, 1, 5, 1, 1, 1, 1, 1, 1, 5], dtype=int64)
```

In [89]:

```
k = [2, 3, 4, 5, 6]
silhouette_scores = []
silhouette_scores.append(
        silhouette_score(iris_df, ac2.fit_predict(iris_df)))
silhouette_scores.append(
        silhouette_score(iris_df, ac3.fit_predict(iris_df)))
silhouette_scores.append(
        silhouette_score(iris_df, ac4.fit_predict(iris_df)))
silhouette_scores.append(
        silhouette_score(iris_df, ac5.fit_predict(iris_df)))
silhouette_scores.append(
        silhouette_score(iris_df, ac6.fit_predict(iris_df)))

plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```



INFERENCE:

Thus, with the help of the silhouette scores, it is concluded that the optimal number of clusters for the given data and clustering technique is 2.

In [93]:

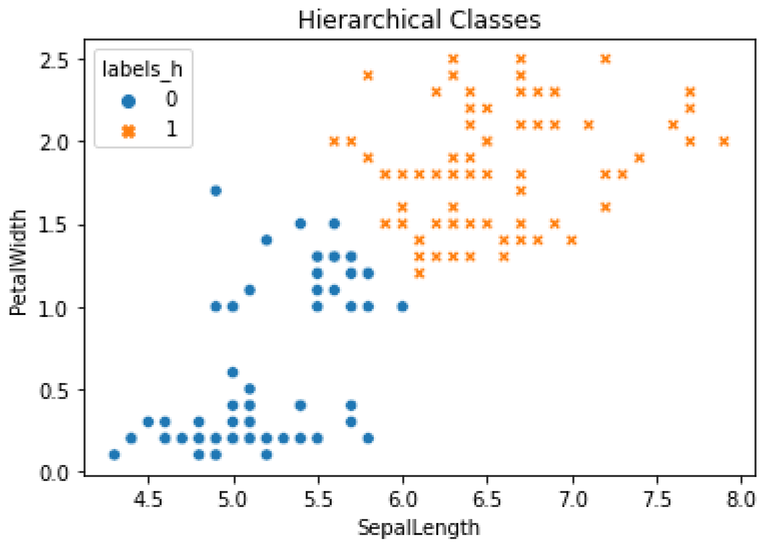
```
hie_clus = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linka
cluster2 = hie_clus.fit_predict(iris_df)
df_h = iris_df.copy(deep=True)
df_h['labels_h'] = cluster2
```

In [95]:
```
df_h.head()
```

Out[95]:

	SepalLength	PetalWidth	labels_h
0	5.1	0.2	0
1	4.9	0.2	0
2	4.7	0.2	0
3	4.6	0.2	0
4	5.0	0.2	0

In [103]:
```
plt.title('Hierarchical Classes')
sns.scatterplot(x=df_h['SepalLength'], y=df_h['PetalWidth'], hue='labels_h',
plt.show()
```



CONCLUSION:

For K-means, the optimum number of clusters was 3 and for Hierarchial, it was 2.

REFERENCES:

https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/
(https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/)
https://www.kaggle.com/code/khotijahs1/hierarchical-agglomerative-clustering
(https://www.kaggle.com/code/khotijahs1/hierarchical-agglomerative-clustering)
https://www.kaggle.com/code/khotijahs1/k-means-clustering-of-iris-dataset/notebook
(https://www.kaggle.com/code/khotijahs1/k-means-clustering-of-iris-dataset/notebook)