CLUSTERING ALGORITHM FOR IRIS DATASET

NAME: SOORAJ ARUN

OVERVIEW

150 flower samples from three species make up the Iris dataset, which has four features: petal length, petal width, sepal length, and sepal width. The objective of a clustering project is to group the flowers according to these characteristics using unsupervised learning approaches such as K-Means or DBSCAN, and then evaluate if the algorithm can correctly identify the three species without the need of the target labels. Data pretreatment for the project includes visual exploration, feature scaling, and selecting a suitable clustering technique to assess the outcomes.

OBJECTIVE

The goal of this project is to identify flower groups based on their attributes (sepal length, sepal width, petal length, and petal width) by using hierarchical clustering and K-Means clustering techniques on the Iris dataset. Comparing how successfully the two unsupervised learning techniques cluster the flowers into different groups and assessing how closely these clusters match the dataset's real species are the objectives.

DATASET: IRIS DATA FROM SKLEARN

IMPORTING LIBRARIES

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.cluster import KMeans

from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch


from sklearn.metrics import silhouette_score

from sklearn.preprocessing import MinMaxScaler


import warnings
warnings.filterwarnings("ignore")
```

```python
from sklearn.datasets import load_iris
iris = load_iris()

iris

{'data': array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2],
        [4.7, 3.2, 1.3, 0.2],
        [4.6, 3.1, 1.5, 0.2],
        [5. , 3.6, 1.4, 0.2],
        [5.4, 3.9, 1.7, 0.4],
        [4.6, 3.4, 1.4, 0.3],
        [5. , 3.4, 1.5, 0.2],
        [4.4, 2.9, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.1],
        [5.4, 3.7, 1.5, 0.2],
        [4.8, 3.4, 1.6, 0.2],
        [4.8, 3. , 1.4, 0.1],
        [4.3, 3. , 1.1, 0.1],
        [5.8, 4. , 1.2, 0.2],
        [5.7, 4.4, 1.5, 0.4],
        [5.4, 3.9, 1.3, 0.4],
        [5.1, 3.5, 1.4, 0.3],
        [5.7, 3.8, 1.7, 0.3],
        [5.1, 3.8, 1.5, 0.3],
        [5.4, 3.4, 1.7, 0.2],
        [5.1, 3.7, 1.5, 0.4],
        [4.6, 3.6, 1. , 0.2],
        [5.1, 3.3, 1.7, 0.5],
        [4.8, 3.4, 1.9, 0.2],
        [5. , 3. , 1.6, 0.2],
        [5. , 3.4, 1.6, 0.4],
        [5.2, 3.5, 1.5, 0.2],
        [5.2, 3.4, 1.4, 0.2],
        [4.7, 3.2, 1.6, 0.2],
        [4.8, 3.1, 1.6, 0.2],
        [5.4, 3.4, 1.5, 0.4],
        [5.2, 4.1, 1.5, 0.1],
        [5.5, 4.2, 1.4, 0.2],
        [4.9, 3.1, 1.5, 0.2],
        [5. , 3.2, 1.2, 0.2],
        [5.5, 3.5, 1.3, 0.2],
        [4.9, 3.6, 1.4, 0.1],
        [4.4, 3. , 1.3, 0.2],
        [5.1, 3.4, 1.5, 0.2],
        [5. , 3.5, 1.3, 0.3],
        [4.5, 2.3, 1.3, 0.3],
        [4.4, 3.2, 1.3, 0.2],
        [5. , 3.5, 1.6, 0.6],
        [5.1, 3.8, 1.9, 0.4],
```

```
       [4.8, 3. , 1.4, 0.3],
       [5.1, 3.8, 1.6, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [5.5, 2.3, 4. , 1.3],
       [6.5, 2.8, 4.6, 1.5],
       [5.7, 2.8, 4.5, 1.3],
       [6.3, 3.3, 4.7, 1.6],
       [4.9, 2.4, 3.3, 1. ],
       [6.6, 2.9, 4.6, 1.3],
       [5.2, 2.7, 3.9, 1.4],
       [5. , 2. , 3.5, 1. ],
       [5.9, 3. , 4.2, 1.5],
       [6. , 2.2, 4. , 1. ],
       [6.1, 2.9, 4.7, 1.4],
       [5.6, 2.9, 3.6, 1.3],
       [6.7, 3.1, 4.4, 1.4],
       [5.6, 3. , 4.5, 1.5],
       [5.8, 2.7, 4.1, 1. ],
       [6.2, 2.2, 4.5, 1.5],
       [5.6, 2.5, 3.9, 1.1],
       [5.9, 3.2, 4.8, 1.8],
       [6.1, 2.8, 4. , 1.3],
       [6.3, 2.5, 4.9, 1.5],
       [6.1, 2.8, 4.7, 1.2],
       [6.4, 2.9, 4.3, 1.3],
       [6.6, 3. , 4.4, 1.4],
       [6.8, 2.8, 4.8, 1.4],
       [6.7, 3. , 5. , 1.7],
       [6. , 2.9, 4.5, 1.5],
       [5.7, 2.6, 3.5, 1. ],
       [5.5, 2.4, 3.8, 1.1],
       [5.5, 2.4, 3.7, 1. ],
       [5.8, 2.7, 3.9, 1.2],
       [6. , 2.7, 5.1, 1.6],
       [5.4, 3. , 4.5, 1.5],
       [6. , 3.4, 4.5, 1.6],
       [6.7, 3.1, 4.7, 1.5],
       [6.3, 2.3, 4.4, 1.3],
       [5.6, 3. , 4.1, 1.3],
       [5.5, 2.5, 4. , 1.3],
       [5.5, 2.6, 4.4, 1.2],
       [6.1, 3. , 4.6, 1.4],
       [5.8, 2.6, 4. , 1.2],
       [5. , 2.3, 3.3, 1. ],
```

```
          [5.6, 2.7, 4.2, 1.3],
          [5.7, 3. , 4.2, 1.2],
          [5.7, 2.9, 4.2, 1.3],
          [6.2, 2.9, 4.3, 1.3],
          [5.1, 2.5, 3. , 1.1],
          [5.7, 2.8, 4.1, 1.3],
          [6.3, 3.3, 6. , 2.5],
          [5.8, 2.7, 5.1, 1.9],
          [7.1, 3. , 5.9, 2.1],
          [6.3, 2.9, 5.6, 1.8],
          [6.5, 3. , 5.8, 2.2],
          [7.6, 3. , 6.6, 2.1],
          [4.9, 2.5, 4.5, 1.7],
          [7.3, 2.9, 6.3, 1.8],
          [6.7, 2.5, 5.8, 1.8],
          [7.2, 3.6, 6.1, 2.5],
          [6.5, 3.2, 5.1, 2. ],
          [6.4, 2.7, 5.3, 1.9],
          [6.8, 3. , 5.5, 2.1],
          [5.7, 2.5, 5. , 2. ],
          [5.8, 2.8, 5.1, 2.4],
          [6.4, 3.2, 5.3, 2.3],
          [6.5, 3. , 5.5, 1.8],
          [7.7, 3.8, 6.7, 2.2],
          [7.7, 2.6, 6.9, 2.3],
          [6. , 2.2, 5. , 1.5],
          [6.9, 3.2, 5.7, 2.3],
          [5.6, 2.8, 4.9, 2. ],
          [7.7, 2.8, 6.7, 2. ],
          [6.3, 2.7, 4.9, 1.8],
          [6.7, 3.3, 5.7, 2.1],
          [7.2, 3.2, 6. , 1.8],
          [6.2, 2.8, 4.8, 1.8],
          [6.1, 3. , 4.9, 1.8],
          [6.4, 2.8, 5.6, 2.1],
          [7.2, 3. , 5.8, 1.6],
          [7.4, 2.8, 6.1, 1.9],
          [7.9, 3.8, 6.4, 2. ],
          [6.4, 2.8, 5.6, 2.2],
          [6.3, 2.8, 5.1, 1.5],
          [6.1, 2.6, 5.6, 1.4],
          [7.7, 3. , 6.1, 2.3],
          [6.3, 3.4, 5.6, 2.4],
          [6.4, 3.1, 5.5, 1.8],
          [6. , 3. , 4.8, 1.8],
          [6.9, 3.1, 5.4, 2.1],
          [6.7, 3.1, 5.6, 2.4],
          [6.9, 3.1, 5.1, 2.3],
          [5.8, 2.7, 5.1, 1.9],
```

```
        [6.8, 3.2, 5.9, 2.3],
        [6.7, 3.3, 5.7, 2.5],
        [6.7, 3. , 5.2, 2.3],
        [6.3, 2.5, 5. , 1.9],
        [6.5, 3. , 5.2, 2. ],
        [6.2, 3.4, 5.4, 2.3],
        [5.9, 3. , 5.1, 1.8]]),
 'target': 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, 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, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]),
 'frame': None,
 'target_names': array(['setosa', 'versicolor', 'virginica'],
dtype='<U10'),
 'DESCR': '.. _iris_dataset:\n\nIris plants dataset\
n--------------------\n\n**Data Set Characteristics:**\n\n:Number of
Instances: 150 (50 in each of three classes)\n:Number of Attributes: 4
numeric, predictive attributes and the class\n:Attribute Information:\
n    - sepal length in cm\n    - sepal width in cm\n    - petal length
in cm\n    - petal width in cm\n    - class:\n            - Iris-
Setosa\n            - Iris-Versicolour\n            - Iris-Virginica\
n\n:Summary Statistics:\n\n============== ==== ==== ======= =====
====================\n                Min  Max   Mean    SD   Class
Correlation\n============== ==== ==== ======= =====
====================\nsepal length:   4.3  7.9   5.84   0.83
0.7826\nsepal width:    2.0  4.4   3.05   0.43   -0.4194\npetal
length:   1.0  6.9   3.76   1.76    0.9490  (high!)\npetal width:
0.1  2.5   1.20   0.76    0.9565  (high!)\n============== ==== ====
======= ===== ====================\n\n:Missing Attribute Values: None\
n:Class Distribution: 33.3% for each of 3 classes.\n:Creator: R.A.
Fisher\n:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\
n:Date: July, 1988\n\nThe famous Iris database, first used by Sir R.A.
Fisher. The dataset is taken\nfrom Fisher\'s paper. Note that it\'s
the same as in R, but not as in the UCI\nMachine Learning Repository,
which has two wrong data points.\n\nThis is perhaps the best known
database to be found in the\npattern recognition literature.
Fisher\'s paper is a classic in the field and\nis referenced
frequently to this day.  (See Duda & Hart, for example.)  The\ndata
set contains 3 classes of 50 instances each, where each class refers
to a\ntype of iris plant.  One class is linearly separable from the
```

```
other 2; the\nlatter are NOT linearly separable from each other.\n\n|
details-start|\n**References**\n|details-split|\n\n- Fisher, R.A. "The
use of multiple measurements in taxonomic problems"\n  Annual
Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to\n
Mathematical Statistics" (John Wiley, NY, 1950).\n- Duda, R.O., &
Hart, P.E. (1973) Pattern Classification and Scene Analysis.\n
(Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.\n-
Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System\n
Structure and Classification Rule for Recognition in Partially
Exposed\n  Environments".  IEEE Transactions on Pattern Analysis and
Machine\n  Intelligence, Vol. PAMI-2, No. 1, 67-71.\n- Gates, G.W.
(1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions\n  on
Information Theory, May 1972, 431-433.\n- See also: 1988 MLC
Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II\n  conceptual
clustering system finds 3 classes in the data.\n- Many, many more ...\
n\n|details-end|\n',
 'feature_names': ['sepal length (cm)',
  'sepal width (cm)',
  'petal length (cm)',
  'petal width (cm)'],
 'filename': 'iris.csv',
 'data_module': 'sklearn.datasets.data'}
```

```python
# Convert to DataFrame
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Drop the species column
features = df.drop(columns=['species'])

df.columns
```

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)', 'species'],
      dtype='object')
```

```python
# Dropping column
df = df.drop('species',axis=1)

df.columns
```

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',
       'petal width (cm)'],
      dtype='object')
```

```python
len(df.columns)
```

```
4
```

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
dtypes: float64(4)
memory usage: 4.8 KB
```

```
df.describe()
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000   
mean            5.843333          3.057333           3.758000   
std             0.828066          0.435866           1.765298   
min             4.300000          2.000000           1.000000   
25%             5.100000          2.800000           1.600000   
50%             5.800000          3.000000           4.350000   
75%             6.400000          3.300000           5.100000   
max             7.900000          4.400000           6.900000   

       petal width (cm)  
count        150.000000  
mean           1.199333  
std            0.762238  
min            0.100000  
25%            0.300000  
50%            1.300000  
75%            1.800000  
max            2.500000  
```

```
df.dtypes
```

```
sepal length (cm)    float64
sepal width (cm)     float64
petal length (cm)    float64
petal width (cm)     float64
dtype: object
```

```
df.shape
```

```
(150, 4)
```

```
df.head()
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width 
(cm)
0                5.1               3.5                1.4  
```

```
0.2
1                 4.9               3.0                1.4
0.2
2                 4.7               3.2                1.3
0.2
3                 4.6               3.1                1.5
0.2
4                 5.0               3.6                1.4
0.2
```

```
df.duplicated()
```

```
0      False
1      False
2      False
3      False
4      False
       ...
145    False
146    False
147    False
148    False
149    False
Length: 150, dtype: bool
```

```
df.duplicated().sum()
```

```
1
```

```
df = df.drop_duplicates()
```

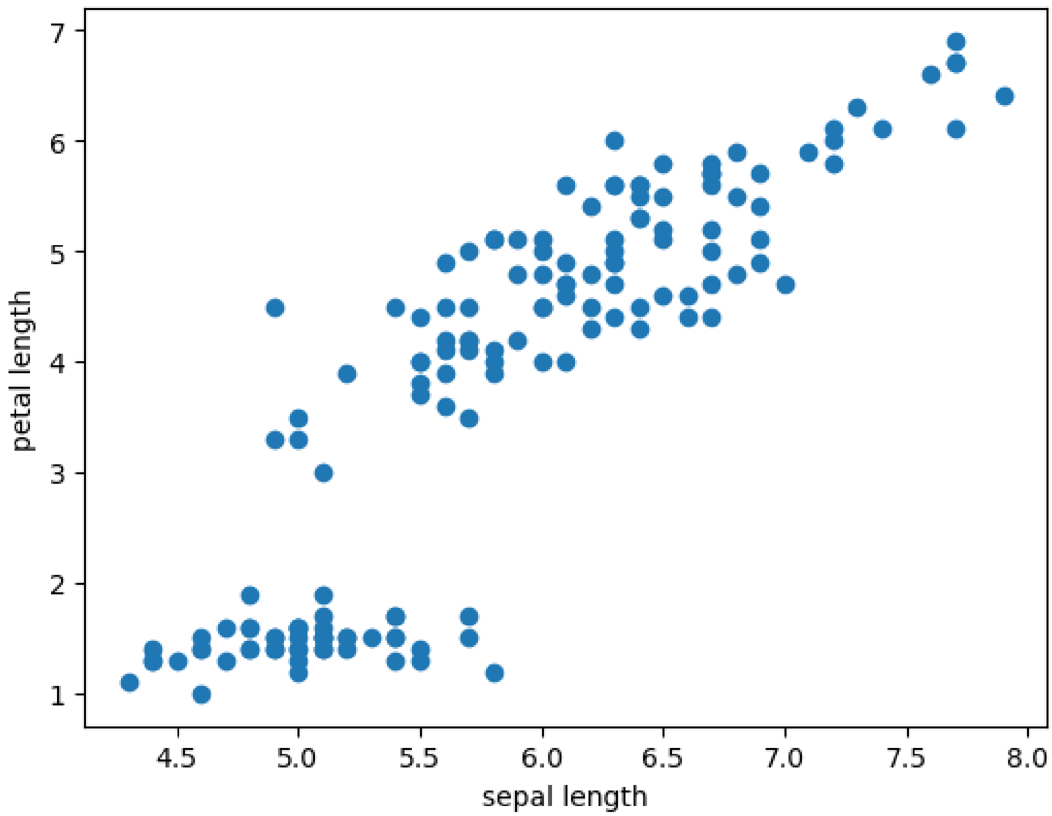
```
df.duplicated().sum()
```

```
0
```

FINDING NUMBER OF CLUSTERS

```
## Drawing scatterplot to find the number of clusters
plt.scatter(df['sepal length (cm)'],df['petal length (cm)'])
plt.xlabel('sepal length')
plt.ylabel('petal length')
```

```
Text(0, 0.5, 'petal length')
```

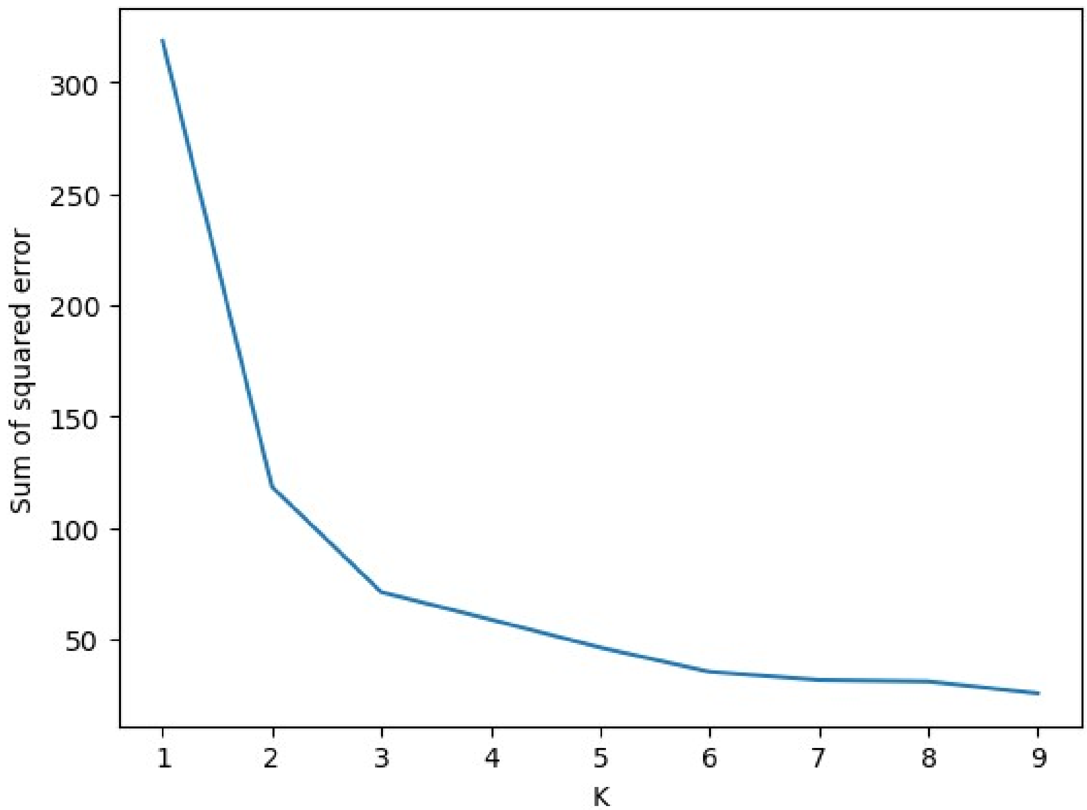


ELBOW METHOD

```
sse = []  #WCSS
k_rng = range(1,10)
for k in k_rng:
    km = KMeans(n_clusters=k)
    km.fit(df[['sepal length (cm)','sepal length (cm)','sepal width
(cm)','petal width (cm)']])
    sse.append(km.inertia_)

plt.xlabel('K')
plt.ylabel('Sum of squared error')
plt.plot(k_rng,sse)
```

```
[<matplotlib.lines.Line2D at 0x25edac94b00>]
```



NUMBER OF CLUSTERS : 3

SCALING

```
# Applying MinMax scaler

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df)
```

IMPLEMENTING CLUSTERING METHODS

K MEANS CLUSTERING

KMeans clustering is an unsupervised machine learning algorithm that partitions the data into k clusters based on feature similarity. It assigns each data point to the nearest cluster centroid and iterates to minimize the sum of squared distances between data points and their assigned cluster centers.

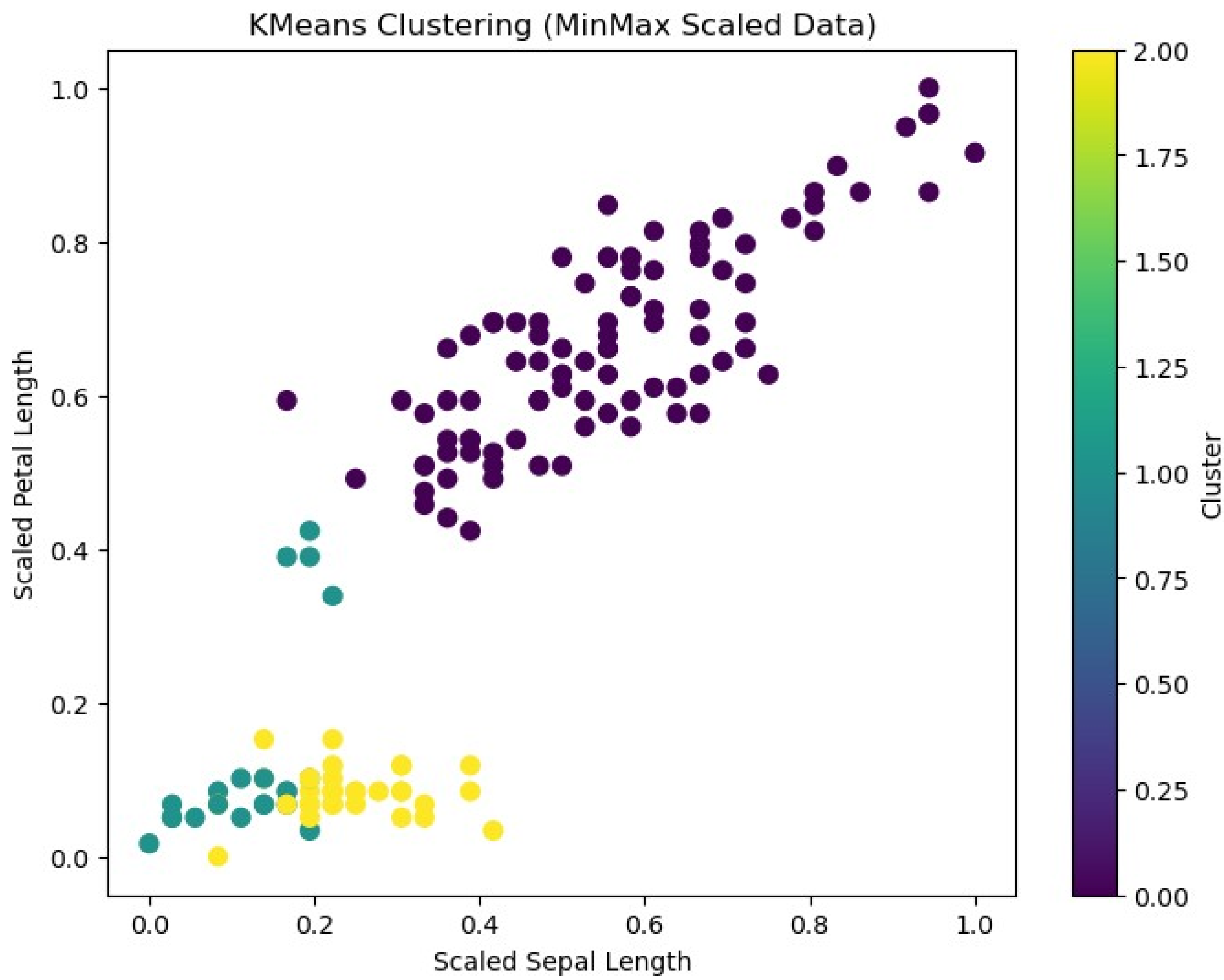
The steps of the KMeans algorithm are: Choose k initial centroids randomly. Assign each data point to the nearest centroid. Recompute the centroids based on the assigned data points. Repeat steps 2 and 3 until convergence (centroids don't change significantly).

Why suitable: The Iris dataset has well-separated flower species based on features like sepal length, sepal width, petal length, and petal width. Since the dataset has inherent groupings (species of flowers), KMeans can effectively group similar data points together.The Iris dataset has well-separated flower species based on features like sepal length, sepal width, petal length, and petal width. Since the dataset has inherent groupings (species of flowers), KMeans can effectively group similar data points together.

```python
# KMeans Clustering
# Apply KMeans Clustering
kmeans = KMeans(n_clusters=3, random_state=42)  # Assuming 3 clusters
kmeans_labels = kmeans.fit_predict(scaled_data)


# Add cluster labels to the DataFrame
df['kmeans_labels'] = kmeans_labels


# Scatter plot to visualize clusters
plt.figure(figsize=(8, 6))
plt.scatter(scaled_data[:, 0], scaled_data[:, 2], c=kmeans_labels,
cmap='viridis', s=50)
plt.title('KMeans Clustering (MinMax Scaled Data)')
plt.xlabel('Scaled Sepal Length')
plt.ylabel('Scaled Petal Length')
plt.colorbar(label='Cluster')
plt.show()
```



HIERARCHICAL CLUSTERING

Hierarchical clustering is an unsupervised machine learning algorithm that builds a hierarchy of clusters either by:

Agglomerative (Bottom-Up): Starts with each data point as its own cluster and iteratively merges the closest clustersved.

Divisive (Top-Down): Starts with all data points in one cluster and recursively splits them.

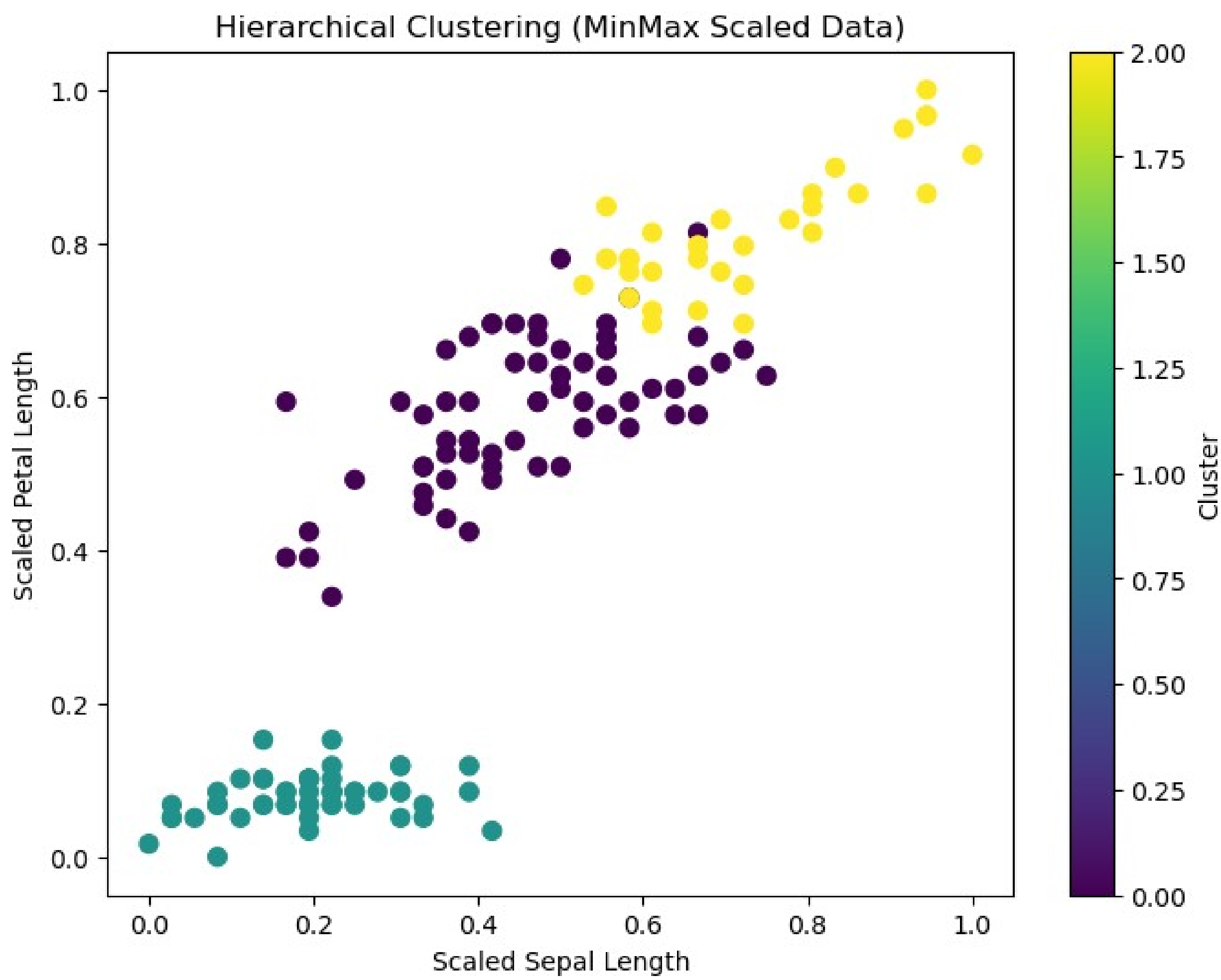
Agglomerative clustering is more commonly used. The process continues until all points are grouped into a single cluster or until the desired number of clusters is achieved.

Why Hierarchical Clustering Might Be Suitable for the Iris Dataset: Hierarchical clustering doesn't require the number of clusters to be pre-defined. It is useful for understanding the data structure and dendrograms can help visualize how clusters are formed. The Iris dataset, with its relatively small size and clear separation between classes, works well with hierarchical clustering.

```
# Apply Hierarchical Clustering
hc = AgglomerativeClustering(n_clusters=3, linkage='ward',
metric='euclidean')
hc_labels = hc.fit_predict(scaled_data)  # Fit and predict cluster
labels

# Add cluster labels to the DataFrame
df['hc_labels'] = hc_labels

# Scatter plot to visualize clusters
plt.figure(figsize=(8, 6))
plt.scatter(scaled_data[:, 0], scaled_data[:, 2], c=hc_labels,
cmap='viridis', s=50)
plt.title('Hierarchical Clustering (MinMax Scaled Data)')
plt.xlabel('Scaled Sepal Length')
plt.ylabel('Scaled Petal Length')
plt.colorbar(label='Cluster')
plt.show()
```



VALUATION

```
# Calculate Silhouette Score
silhouette = silhouette_score(scaled_data, kmeans_labels)
print(f"Silhouette Score (KMeans with MinMax Scaling): 
{silhouette:.4f}")
```

```
Silhouette Score (KMeans with MinMax Scaling): 0.4968
```

```
# Calculate Silhouette Score
silhouette = silhouette_score(scaled_data, hc_labels)
print(f"Silhouette Score (Hierarchical Clustering with MinMax 
Scaling): {silhouette:.4f}")
```

```
Silhouette Score (Hierarchical Clustering with MinMax Scaling): 0.5064
```