In [65]:
```
## Loading the necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.decomposition import PCA
```

In [66]:
```
# Loading the dataset
data = load_iris()
df = pd.DataFrame(data.data, columns = data.feature_names)
df.head()
```

Out[66]:

	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

In [67]:
```
# Checking for null values
df.isnull().sum()
```

Out[67]:
```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
dtype: int64
```

In [68]:
```
# Getting the basic info
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
```

In [69]:
```
# Checking fo duplicates
df.drop_duplicates(inplace = True)
df.head()
```

Out[69]:

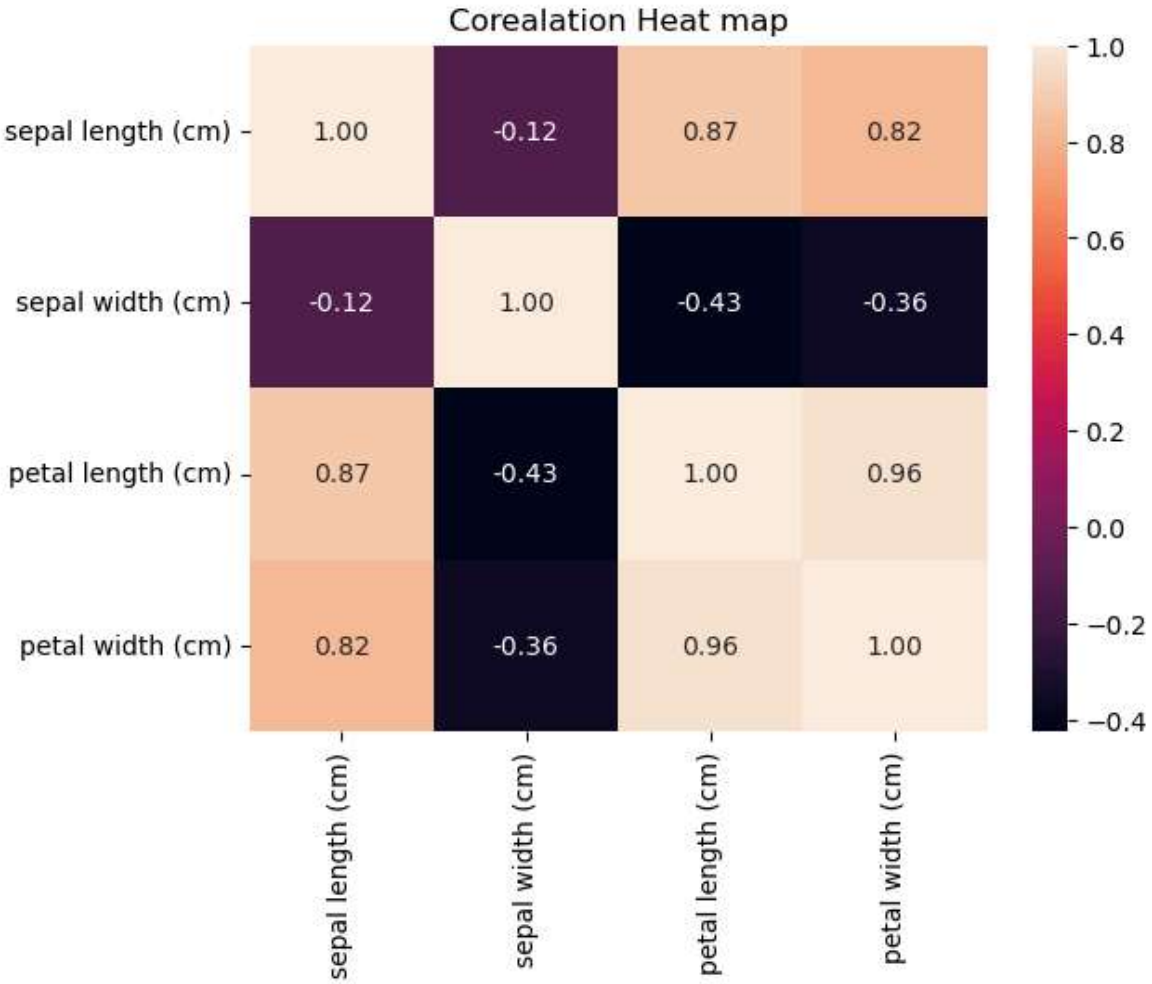
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [70]:
```
# Descriptive Analysis
df.describe()
```

Out[70]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
count	149.000000	149.000000	149.000000	149.000000
mean	5.843624	3.059732	3.748993	1.194631
std	0.830851	0.436342	1.767791	0.762622
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.300000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [71]:
```
# Plotting the heat map

corealtion_matrix = df.corr()

sns.heatmap(corealtion_matrix,annot=True,cmap='rocket',fmt='.2f')
plt.title('Corealation Heat map')
plt.show()
```



Clustering Algorithm

A) KMeans Clustering

K-Means is an unsupervised machine learning algorithm used to group data into a specified number of clusters (k). The process works as follows:

- Initialization: Randomly select k points as initial cluster centroids.
- Assignment: Assign each data point to the nearest centroid based on a distance metric (e.g., Euclidean distance).
- Update: Recalculate the centroids by averaging the data points assigned to each cluster.
- Repeat: Iterate steps 2 and 3 until the centroids stabilize (i.e., no significant change in their positions) or a maximum number of iterations is reached.
- Output: The algorithm outputs cluster assignments and the final centroid positions.
- K-Means works best with numerical data and assumes clusters are spherical and evenly sized. It may require preprocessing (e.g., scaling) for optimal results.

Why KMeans clustering might be suitable for the Iris dataset?

K-Means clustering is suitable for the Iris dataset because it is an unsupervised algorithm that can effectively group data points based on their features, and the Iris dataset contains well-defined numerical features (sepal length, sepal width, petal length, and petal width). These features provide clear separations between species in a multi-dimensional space, making it ideal for K-Means, which works by minimizing intra-cluster distances. Additionally, the Iris dataset is small, balanced, and has roughly spherical clusters, aligning well with the assumptions of K-Means. This allows the algorithm to group the data into three clusters that can correspond to the three species (setosa, versicolor, virginica) without relying on labeled data.

In [76]:
```
scaler=StandardScaler()
scaled_data=scaler.fit_transform(df)
X = pd.DataFrame(scaled_data)
```

In [77]:
```
X.head()
```

Out[77]:

	0	1	2	3
0	-0.898033	1.012401	-1.333255	-1.308624
1	-1.139562	-0.137353	-1.333255	-1.308624
2	-1.381091	0.322549	-1.390014	-1.308624
3	-1.501855	0.092598	-1.276496	-1.308624
4	-1.018798	1.242352	-1.333255	-1.308624

In [78]:
```
import warnings
warnings.filterwarnings("ignore")
```

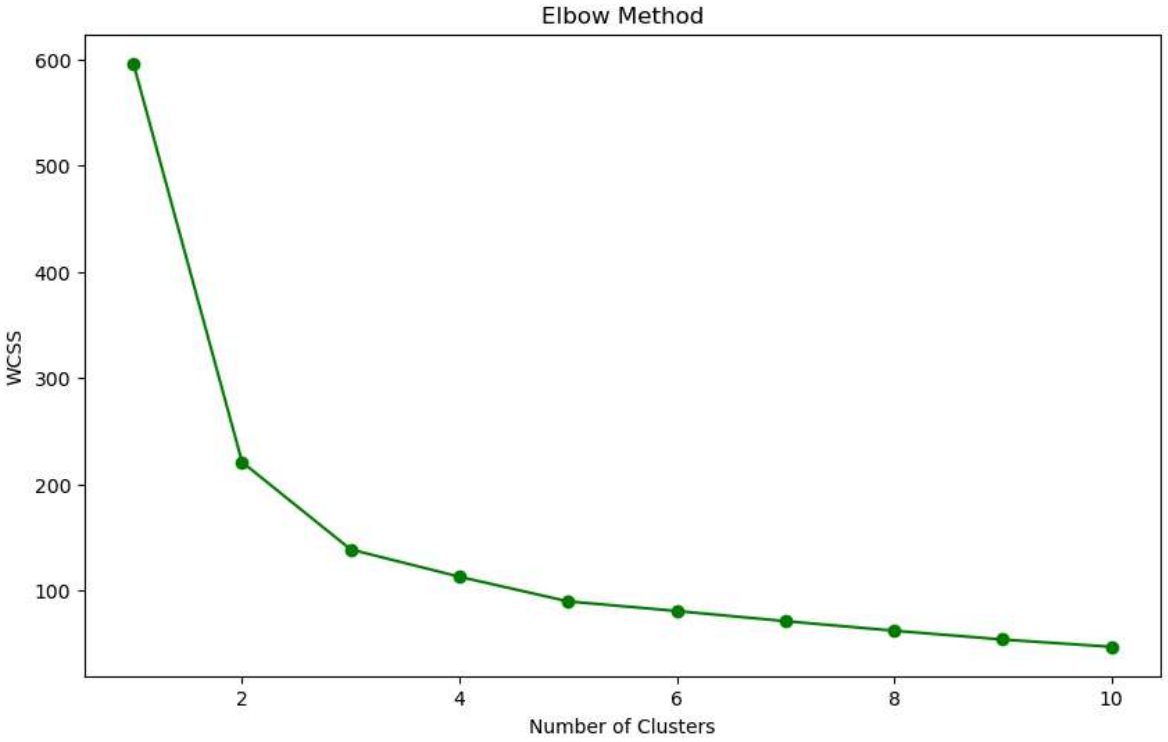
In [79]:
```
wcss = []

for i in range(1,11):
    kmeans = KMeans(n_clusters=i, n_init=10,random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

Elbow method :

The Elbow Method is a technique used in KMeans clustering to determine the optimal number of clusters (K) for a dataset. It works by plotting the within-cluster sum of squares (WCSS) against different values of K and looking for the "elbow point" in the graph. The elbow point is where the rate of decrease in WCSS slows significantly, indicating that adding more clusters does not provide substantial improvement in clustering performance. This optimal K balances model simplicity and accuracy, avoiding both underfitting and overfitting.

In [81]:
```
# Plotting the elbow method

plt.figure(figsize = (10,6))
plt.plot(range(1,11), wcss, marker = 'o',color = 'g')
plt.title('Elbow Method')
```

```
plt.xlabel("Number of Clusters")
plt.ylabel('WCSS')
plt.show()
```



Taking K = 5 from the graph

In [83]:
```
# applying kmeans clustering with optimal number of clusters
kmeans = KMeans(n_clusters=5, n_init = 10, random_state=0)
y_kmeans = kmeans.fit_predict(X)
```

In [84]:
```
df['Cluster'] = y_kmeans
df.head()
```

Out[84]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Cluster
0	5.1	3.5	1.4	0.2	4
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	4

In [85]:
```
# Applying PCA for comparison

pca = PCA(n_components =2)
df_pca = pca.fit_transform(X)
```

In [86]:
```
df_pca
```

Out[86]:
```
array([[-2.25269158e+00,  4.79299423e-01],
       [-2.07100441e+00, -6.74233952e-01],
       [-2.35314718e+00, -3.40442064e-01],
       [-2.28876231e+00, -5.95440778e-01],
       [-2.37729413e+00,  6.46885300e-01],
       [-2.06233579e+00,  1.48735169e+00],
       [-2.43205441e+00,  5.01784240e-02],
       [-2.22124438e+00,  2.22920844e-01],
       [-2.32468093e+00, -1.11233754e+00],
       [-2.17393647e+00, -4.69164509e-01],
       [-2.15372644e+00,  1.04128247e+00],
       [-2.31439973e+00,  1.34128141e-01],
       [-2.20836461e+00, -7.28302820e-01],
       [-2.62241020e+00, -9.57873840e-01],
       [-2.18502153e+00,  1.85550449e+00],
       [-2.24685180e+00,  2.68321162e+00],
       [-2.19408649e+00,  1.48183222e+00],
       [-2.17837789e+00,  4.88282008e-01],
       [-1.88595391e+00,  1.40095580e+00],
       [-2.33010833e+00,  1.12767836e+00],
       [-1.90318298e+00,  4.06025713e-01],
       [-2.19423860e+00,  9.23988780e-01],
       [-2.76123087e+00,  4.61020699e-01],
       [-1.80782540e+00,  8.50424513e-02],
       [-2.21558671e+00,  1.38267738e-01],
       [-1.94208255e+00, -6.26387935e-01],
       [-2.03967932e+00,  2.42265878e-01],
       [-2.15670739e+00,  5.25765574e-01],
       [-2.12808903e+00,  3.11713547e-01],
       [-2.25433417e+00, -3.36302466e-01],
       [-2.12973161e+00, -5.03888343e-01],
       [-1.82043094e+00,  4.21231150e-01],
       [-2.60035732e+00,  1.79281596e+00],
       [-2.43139780e+00,  2.14834969e+00],
       [-2.09962277e+00, -4.60181924e-01],
       [-2.19694532e+00, -2.06563077e-01],
       [-2.03344320e+00,  6.58264695e-01],
       [-2.51465433e+00,  5.92816432e-01],
       [-2.41917464e+00, -9.01045240e-01],
       [-2.15819787e+00,  2.68007128e-01],
       [-2.27436207e+00,  4.41815857e-01],
       [-1.85092217e+00, -2.33568150e+00],
       [-2.54228672e+00, -4.75700917e-01],
       [-1.95260798e+00,  4.72903207e-01],
       [-2.12404395e+00,  1.14218041e+00],
       [-2.05973723e+00, -7.10337652e-01],
       [-2.37148435e+00,  1.12007564e+00],
       [-2.38325602e+00, -3.84148483e-01],
       [-2.21677296e+00,  9.96196181e-01],
       [-2.19262601e+00,  8.86881654e-03],
       [ 1.10856780e+00,  8.51248930e-01],
       [ 7.38727067e-01,  5.86954076e-01],
       [ 1.24726636e+00,  6.05232801e-01],
       [ 4.11997054e-01, -1.75773643e+00],
       [ 1.08093541e+00, -2.17268419e-01],
       [ 3.94998250e-01, -5.97303729e-01],
       [ 7.54313553e-01,  7.66282269e-01],
       [-4.81342849e-01, -1.85218879e+00],
       [ 9.33798500e-01,  2.25248587e-02],
       [ 1.80093777e-02, -1.03470392e+00],
```

```
[-1.06196834e-01, -2.65503142e+00],
[ 4.47793561e-01, -6.79612668e-02],
[ 5.65844582e-01, -1.77192492e+00],
[ 7.25817300e-01, -1.92544114e-01],
[-2.60433605e-02, -4.42136645e-01],
[ 8.82171279e-01,  4.99178318e-01],
[ 3.57467043e-01, -1.99080523e-01],
[ 1.64909034e-01, -7.97356822e-01],
[ 1.22819443e+00, -1.62994011e+00],
[ 1.70366432e-01, -1.30665086e+00],
[ 7.45248597e-01,  3.92610004e-01],
[ 4.82495935e-01, -4.23857920e-01],
[ 1.23832352e+00, -9.41317874e-01],
[ 6.38745956e-01, -4.23181443e-01],
[ 7.08892454e-01, -7.17873080e-02],
[ 8.80680806e-01,  2.41419872e-01],
[ 1.26163660e+00, -8.82324175e-02],
[ 1.36429443e+00,  3.21733105e-01],
[ 6.71209133e-01, -2.31407546e-01],
[-3.42074786e-02, -1.06339446e+00],
[ 1.35938285e-01, -1.56578917e+00],
[ 2.86869203e-02, -1.57615162e+00],
[ 2.47661071e-01, -7.82151385e-01],
[ 1.06626094e+00, -6.39490089e-01],
[ 2.31374017e-01, -2.89253091e-01],
[ 4.37742629e-01,  8.40935844e-01],
[ 1.05529799e+00,  5.12300500e-01],
[ 1.04811985e+00, -1.39152669e+00],
[ 7.70889665e-02, -2.22565155e-01],
[ 2.88884976e-01, -1.33239211e+00],
[ 2.84765938e-01, -1.12318307e+00],
[ 6.31323588e-01,  1.87481817e-02],
[ 3.42154783e-01, -9.93443680e-01],
[-3.56740297e-01, -2.01977467e+00],
[ 2.94694757e-01, -8.59201772e-01],
[ 9.87594609e-02, -1.85081588e-01],
[ 2.34629192e-01, -3.88771165e-01],
[ 5.82799429e-01, -1.61959877e-01],
[-4.41305191e-01, -1.54450108e+00],
[ 2.63247557e-01, -6.02823192e-01],
[ 1.85132653e+00,  8.65063783e-01],
[ 1.16310899e+00, -7.02714905e-01],
[ 2.21017431e+00,  5.50427373e-01],
[ 1.44560415e+00, -5.40224147e-02],
[ 1.87327125e+00,  2.87512384e-01],
[ 2.75597059e+00,  7.85517856e-01],
[ 3.72549032e-01, -1.56008015e+00],
[ 2.30663299e+00,  4.06499492e-01],
[ 2.00988971e+00, -7.21606190e-01],
[ 2.26701470e+00,  1.91023669e+00],
[ 1.37096808e+00,  6.85232477e-01],
[ 1.60726341e+00, -4.29437466e-01],
[ 1.88928408e+00,  4.09649055e-01],
[ 1.26455057e+00, -1.16554279e+00],
[ 1.47312141e+00, -4.45129823e-01],
[ 1.59673799e+00,  6.69853676e-01],
[ 1.47720346e+00,  2.47442450e-01],
[ 2.43382015e+00,  2.54234388e+00],
[ 3.31268166e+00,  2.02026219e-03],
[ 1.26678977e+00, -1.71321334e+00],
```

```
       [ 2.04372124e+00,  9.00804563e-01],
       [ 9.83898272e-01, -5.73992460e-01],
       [ 2.90075316e+00,  3.97657100e-01],
       [ 1.33815252e+00, -4.89025799e-01],
       [ 1.70744479e+00,  1.00533899e+00],
       [ 1.96010534e+00,  9.95290093e-01],
       [ 1.18061229e+00, -3.22819788e-01],
       [ 1.02739137e+00,  5.88181161e-02],
       [ 1.79314778e+00, -1.94660539e-01],
       [ 1.86871469e+00,  5.49220870e-01],
       [ 2.43967389e+00,  2.45136467e-01],
       [ 2.31247277e+00,  2.61041168e+00],
       [ 1.86746147e+00, -1.85677955e-01],
       [ 1.11953075e+00, -3.00541658e-01],
       [ 1.20692448e+00, -8.18141804e-01],
       [ 2.80295612e+00,  8.41669980e-01],
       [ 1.58370611e+00,  1.06323390e+00],
       [ 1.35260091e+00,  4.15028326e-01],
       [ 9.31407188e-01,  1.23519657e-02],
       [ 1.85783688e+00,  6.66027635e-01],
       [ 2.02056027e+00,  6.05562551e-01],
       [ 1.90765124e+00,  6.79853206e-01],
       [ 2.04655008e+00,  8.58478010e-01],
       [ 2.00469956e+00,  1.04126932e+00],
       [ 1.87605193e+00,  3.78388342e-01],
       [ 1.56851596e+00, -9.04007671e-01],
       [ 1.52701783e+00,  2.61268020e-01],
       [ 1.38047056e+00,  1.00640530e+00],
       [ 9.67173695e-01, -2.85947210e-02]])
```

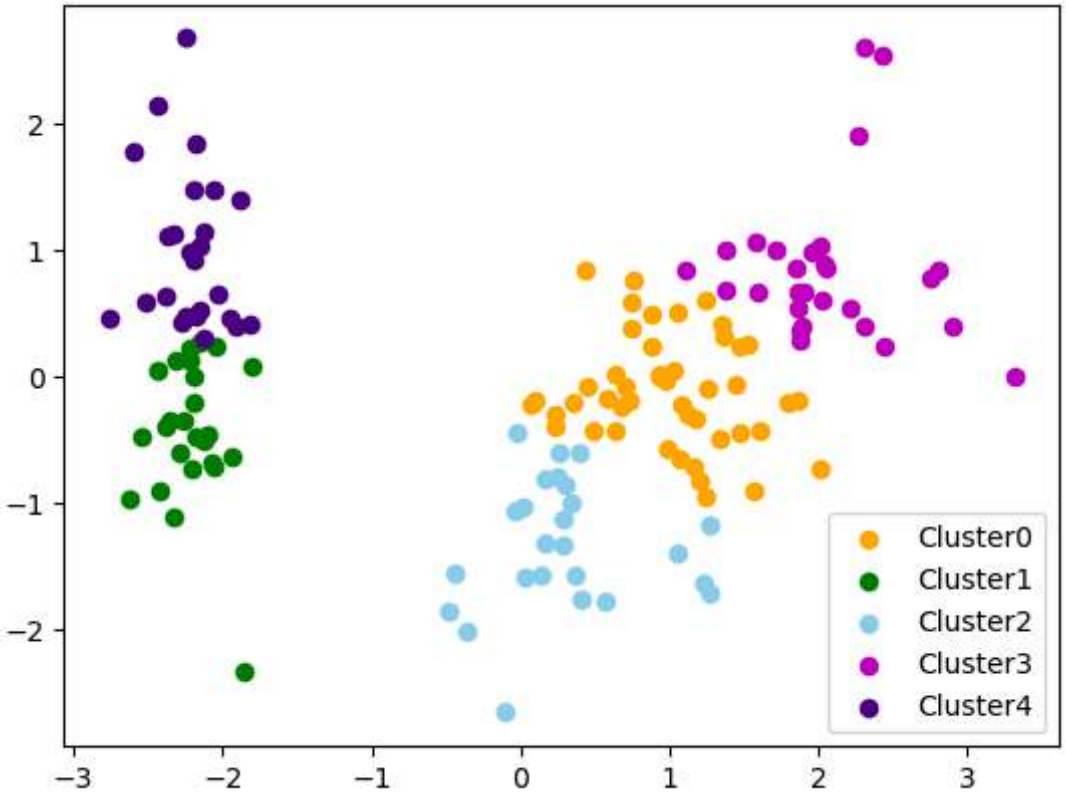
In [87]:
```
# Converting to dataframa
X_pca = pd.DataFrame(df_pca, columns = ['PC1','PC2']) # Reducing data using pca
X_pca.head()
```

Out[87]:

	PC1	PC2
0	-2.252692	0.479299
1	-2.071004	-0.674234
2	-2.353147	-0.340442
3	-2.288762	-0.595441
4	-2.377294	0.646885

Plotting the clusters plt.scatter(X_pca['PC1'],X_pca['PC2'], c = y_kmeans) plt.show()

In [88]:
```
# Using custom colours
colors = ['orange','g','skyblue','m','indigo']

for i in range(5):
  cluster_data = X_pca[y_kmeans ==i]
  plt.scatter(cluster_data['PC1'],
              cluster_data['PC2'],
              c=colors[i],
              label = f'Cluster{i}')
  plt.legend()
```



B) Hierarchical Clustering

Hierarchical clustering is a clustering technique that groups data points into a hierarchy of clusters. It operates either in a divisive (top-down) or agglomerative (bottom-up) manner. Agglomerative clustering starts with each data point as its own cluster and iteratively merges the closest clusters based on a distance metric (e.g., Euclidean) until all points form a single cluster. Divisive clustering does the reverse, splitting one large cluster into smaller clusters. The results are visualized using a dendrogram, which shows the merging or splitting process and helps determine the optimal number of clusters by cutting the dendrogram at a specific level. Hierarchical clustering is particularly useful for understanding nested cluster relationships.

Why Hierarchical Clustering for Iris dataset?

Hierarchical clustering is important in the Iris dataset as it helps uncover natural groupings of the three flower species (Setosa, Versicolor, Virginica) without requiring predefined labels. It provides a dendrogram that visually represents the clustering process, offering insights into how data points and species are related at different levels. By leveraging distance-based metrics, hierarchical clustering identifies patterns in the petal and sepal measurements, making it a powerful tool for exploratory data analysis and understanding the dataset's structure.

In [92]:
```
X.head()
```

Out[92]:

	0	1	2	3
0	-0.898033	1.012401	-1.333255	-1.308624
1	-1.139562	-0.137353	-1.333255	-1.308624
2	-1.381091	0.322549	-1.390014	-1.308624
3	-1.501855	0.092598	-1.276496	-1.308624
4	-1.018798	1.242352	-1.333255	-1.308624

In [93]:
```
X_hc = df.values
```

In [94]:
```
X_hc
```

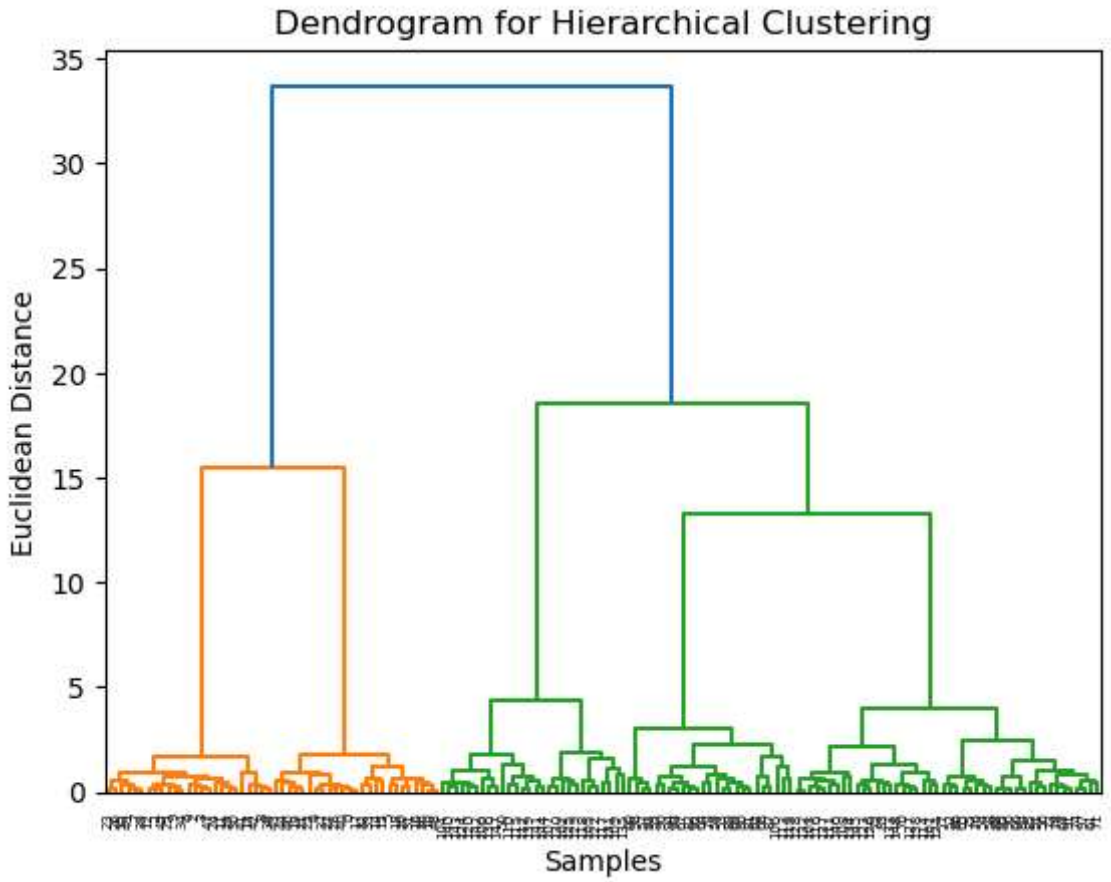
Out[94]:
```
array([[5.1, 3.5, 1.4, 0.2, 4. ],
       [4.9, 3. , 1.4, 0.2, 1. ],
       [4.7, 3.2, 1.3, 0.2, 1. ],
       [4.6, 3.1, 1.5, 0.2, 1. ],
       [5. , 3.6, 1.4, 0.2, 4. ],
       [5.4, 3.9, 1.7, 0.4, 4. ],
       [4.6, 3.4, 1.4, 0.3, 1. ],
       [5. , 3.4, 1.5, 0.2, 1. ],
       [4.4, 2.9, 1.4, 0.2, 1. ],
       [4.9, 3.1, 1.5, 0.1, 1. ],
       [5.4, 3.7, 1.5, 0.2, 4. ],
       [4.8, 3.4, 1.6, 0.2, 1. ],
       [4.8, 3. , 1.4, 0.1, 1. ],
       [4.3, 3. , 1.1, 0.1, 1. ],
       [5.8, 4. , 1.2, 0.2, 4. ],
       [5.7, 4.4, 1.5, 0.4, 4. ],
       [5.4, 3.9, 1.3, 0.4, 4. ],
       [5.1, 3.5, 1.4, 0.3, 4. ],
       [5.7, 3.8, 1.7, 0.3, 4. ],
       [5.1, 3.8, 1.5, 0.3, 4. ],
       [5.4, 3.4, 1.7, 0.2, 4. ],
       [5.1, 3.7, 1.5, 0.4, 4. ],
       [4.6, 3.6, 1. , 0.2, 4. ],
       [5.1, 3.3, 1.7, 0.5, 1. ],
       [4.8, 3.4, 1.9, 0.2, 1. ],
       [5. , 3. , 1.6, 0.2, 1. ],
       [5. , 3.4, 1.6, 0.4, 1. ],
       [5.2, 3.5, 1.5, 0.2, 4. ],
       [5.2, 3.4, 1.4, 0.2, 4. ],
       [4.7, 3.2, 1.6, 0.2, 1. ],
       [4.8, 3.1, 1.6, 0.2, 1. ],
       [5.4, 3.4, 1.5, 0.4, 4. ],
       [5.2, 4.1, 1.5, 0.1, 4. ],
       [5.5, 4.2, 1.4, 0.2, 4. ],
       [4.9, 3.1, 1.5, 0.2, 1. ],
       [5. , 3.2, 1.2, 0.2, 1. ],
       [5.5, 3.5, 1.3, 0.2, 4. ],
       [4.9, 3.6, 1.4, 0.1, 4. ],
       [4.4, 3. , 1.3, 0.2, 1. ],
       [5.1, 3.4, 1.5, 0.2, 1. ],
       [5. , 3.5, 1.3, 0.3, 4. ],
       [4.5, 2.3, 1.3, 0.3, 1. ],
       [4.4, 3.2, 1.3, 0.2, 1. ],
       [5. , 3.5, 1.6, 0.6, 4. ],
       [5.1, 3.8, 1.9, 0.4, 4. ],
       [4.8, 3. , 1.4, 0.3, 1. ],
       [5.1, 3.8, 1.6, 0.2, 4. ],
       [4.6, 3.2, 1.4, 0.2, 1. ],
       [5.3, 3.7, 1.5, 0.2, 4. ],
       [5. , 3.3, 1.4, 0.2, 1. ],
       [7. , 3.2, 4.7, 1.4, 3. ],
       [6.4, 3.2, 4.5, 1.5, 0. ],
       [6.9, 3.1, 4.9, 1.5, 0. ],
       [5.5, 2.3, 4. , 1.3, 2. ],
       [6.5, 2.8, 4.6, 1.5, 0. ],
       [5.7, 2.8, 4.5, 1.3, 2. ],
       [6.3, 3.3, 4.7, 1.6, 0. ],
       [4.9, 2.4, 3.3, 1. , 2. ],
       [6.6, 2.9, 4.6, 1.3, 0. ],
       [5.2, 2.7, 3.9, 1.4, 2. ],
```

```
[5. , 2. , 3.5, 1. , 2. ],
[5.9, 3. , 4.2, 1.5, 0. ],
[6. , 2.2, 4. , 1. , 2. ],
[6.1, 2.9, 4.7, 1.4, 0. ],
[5.6, 2.9, 3.6, 1.3, 2. ],
[6.7, 3.1, 4.4, 1.4, 0. ],
[5.6, 3. , 4.5, 1.5, 0. ],
[5.8, 2.7, 4.1, 1. , 2. ],
[6.2, 2.2, 4.5, 1.5, 2. ],
[5.6, 2.5, 3.9, 1.1, 2. ],
[5.9, 3.2, 4.8, 1.8, 0. ],
[6.1, 2.8, 4. , 1.3, 0. ],
[6.3, 2.5, 4.9, 1.5, 0. ],
[6.1, 2.8, 4.7, 1.2, 0. ],
[6.4, 2.9, 4.3, 1.3, 0. ],
[6.6, 3. , 4.4, 1.4, 0. ],
[6.8, 2.8, 4.8, 1.4, 0. ],
[6.7, 3. , 5. , 1.7, 0. ],
[6. , 2.9, 4.5, 1.5, 0. ],
[5.7, 2.6, 3.5, 1. , 2. ],
[5.5, 2.4, 3.8, 1.1, 2. ],
[5.5, 2.4, 3.7, 1. , 2. ],
[5.8, 2.7, 3.9, 1.2, 2. ],
[6. , 2.7, 5.1, 1.6, 0. ],
[5.4, 3. , 4.5, 1.5, 0. ],
[6. , 3.4, 4.5, 1.6, 0. ],
[6.7, 3.1, 4.7, 1.5, 0. ],
[6.3, 2.3, 4.4, 1.3, 2. ],
[5.6, 3. , 4.1, 1.3, 0. ],
[5.5, 2.5, 4. , 1.3, 2. ],
[5.5, 2.6, 4.4, 1.2, 2. ],
[6.1, 3. , 4.6, 1.4, 0. ],
[5.8, 2.6, 4. , 1.2, 2. ],
[5. , 2.3, 3.3, 1. , 2. ],
[5.6, 2.7, 4.2, 1.3, 2. ],
[5.7, 3. , 4.2, 1.2, 0. ],
[5.7, 2.9, 4.2, 1.3, 0. ],
[6.2, 2.9, 4.3, 1.3, 0. ],
[5.1, 2.5, 3. , 1.1, 2. ],
[5.7, 2.8, 4.1, 1.3, 2. ],
[6.3, 3.3, 6. , 2.5, 3. ],
[5.8, 2.7, 5.1, 1.9, 0. ],
[7.1, 3. , 5.9, 2.1, 3. ],
[6.3, 2.9, 5.6, 1.8, 0. ],
[6.5, 3. , 5.8, 2.2, 3. ],
[7.6, 3. , 6.6, 2.1, 3. ],
[4.9, 2.5, 4.5, 1.7, 2. ],
[7.3, 2.9, 6.3, 1.8, 3. ],
[6.7, 2.5, 5.8, 1.8, 0. ],
[7.2, 3.6, 6.1, 2.5, 3. ],
[6.5, 3.2, 5.1, 2. , 3. ],
[6.4, 2.7, 5.3, 1.9, 0. ],
[6.8, 3. , 5.5, 2.1, 3. ],
[5.7, 2.5, 5. , 2. , 2. ],
[5.8, 2.8, 5.1, 2.4, 0. ],
[6.4, 3.2, 5.3, 2.3, 3. ],
[6.5, 3. , 5.5, 1.8, 0. ],
[7.7, 3.8, 6.7, 2.2, 3. ],
[7.7, 2.6, 6.9, 2.3, 3. ],
[6. , 2.2, 5. , 1.5, 2. ],
```

```
       [6.9, 3.2, 5.7, 2.3, 3. ],
       [5.6, 2.8, 4.9, 2. , 0. ],
       [7.7, 2.8, 6.7, 2. , 3. ],
       [6.3, 2.7, 4.9, 1.8, 0. ],
       [6.7, 3.3, 5.7, 2.1, 3. ],
       [7.2, 3.2, 6. , 1.8, 3. ],
       [6.2, 2.8, 4.8, 1.8, 0. ],
       [6.1, 3. , 4.9, 1.8, 0. ],
       [6.4, 2.8, 5.6, 2.1, 0. ],
       [7.2, 3. , 5.8, 1.6, 3. ],
       [7.4, 2.8, 6.1, 1.9, 3. ],
       [7.9, 3.8, 6.4, 2. , 3. ],
       [6.4, 2.8, 5.6, 2.2, 0. ],
       [6.3, 2.8, 5.1, 1.5, 0. ],
       [6.1, 2.6, 5.6, 1.4, 0. ],
       [7.7, 3. , 6.1, 2.3, 3. ],
       [6.3, 3.4, 5.6, 2.4, 3. ],
       [6.4, 3.1, 5.5, 1.8, 0. ],
       [6. , 3. , 4.8, 1.8, 0. ],
       [6.9, 3.1, 5.4, 2.1, 3. ],
       [6.7, 3.1, 5.6, 2.4, 3. ],
       [6.9, 3.1, 5.1, 2.3, 3. ],
       [6.8, 3.2, 5.9, 2.3, 3. ],
       [6.7, 3.3, 5.7, 2.5, 3. ],
       [6.7, 3. , 5.2, 2.3, 3. ],
       [6.3, 2.5, 5. , 1.9, 0. ],
       [6.5, 3. , 5.2, 2. , 0. ],
       [6.2, 3.4, 5.4, 2.3, 3. ],
       [5.9, 3. , 5.1, 1.8, 0. ]])
```

In [95]:

```
# Using the dendrogram to find the optimal number of clusters

import scipy.cluster.hierarchy as sch
dendrogram = sch.dendrogram(sch.linkage(X_hc, method = 'ward'))


plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Euclidean Distance')
plt.show()
```



In [96]:
```
from sklearn.cluster import AgglomerativeClustering
hc = AgglomerativeClustering(n_clusters = 5, metric = 'euclidean', linkage = 'wa
```

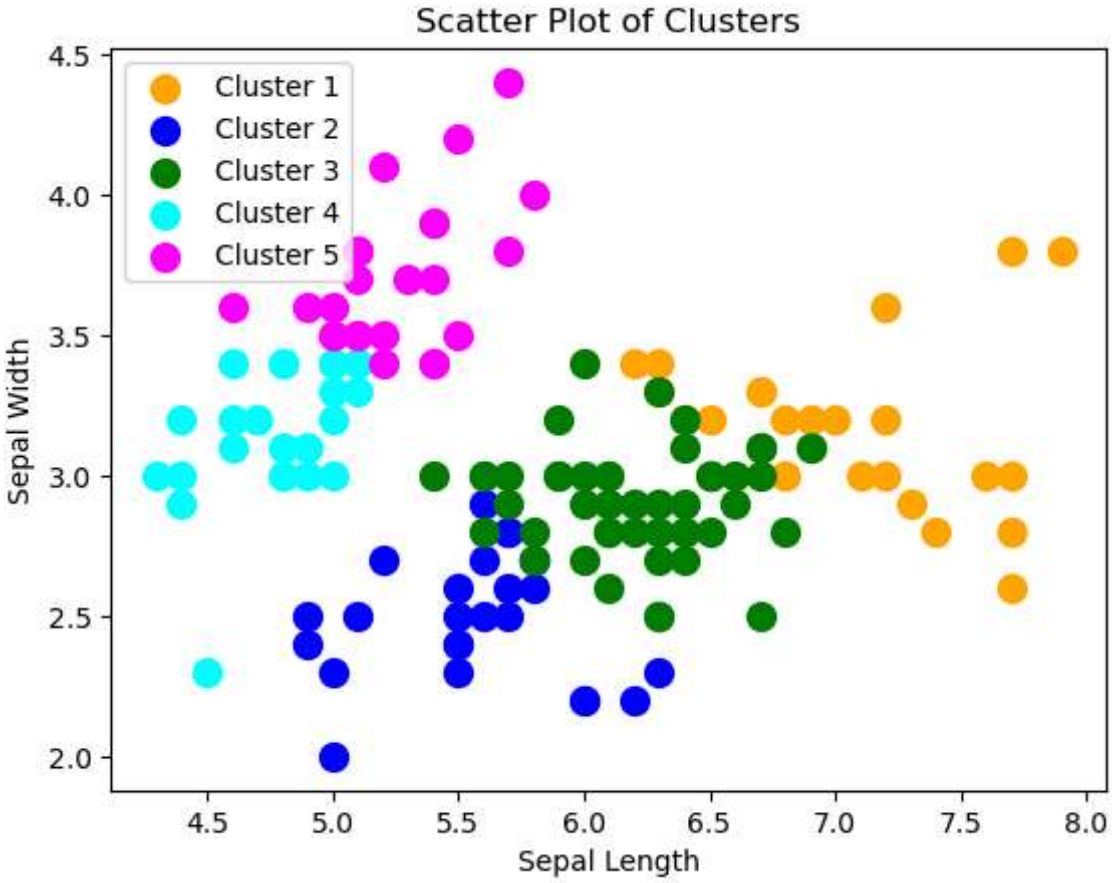
In [97]:
```
y_hc = hc.fit_predict(X_hc)
```

In [98]:
```
y_hc
```

Out[98]:
```
array([4, 3, 3, 3, 4, 4, 3, 3, 3, 3, 4, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 3, 3, 3, 3, 4, 4, 3, 3, 4, 4, 4, 3, 3, 4, 4, 3, 3, 4, 3, 3, 4,
       4, 3, 4, 3, 4, 3, 0, 2, 2, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 2, 1, 2,
       2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2, 1,
       2, 1, 1, 2, 1, 1, 1, 2, 2, 2, 1, 1, 0, 2, 0, 2, 0, 0, 1, 0, 2, 0,
       0, 2, 0, 1, 2, 0, 2, 0, 0, 1, 0, 2, 0, 2, 0, 0, 2, 2, 2, 0, 0, 0,
       2, 2, 2, 0, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 2, 0, 2], dtype=int64)
```

In [99]:
```
# Plotting data points from Cluster 1
plt.scatter(X_hc[y_hc == 0, 0], X_hc[y_hc == 0, 1], s = 100, c = 'orange', label

# X_hc[y_hc == 0, 0] represents the x-coordinates of the data points in cluster
# X_hc[y_hc == 0, 1] represents the y-coordinates of the data points in cluster

# Plotting data points from Cluster 2
plt.scatter(X_hc[y_hc == 1, 0], X_hc[y_hc == 1, 1], s = 100, c = 'blue', label =

# Plotting data points from Cluster 3
plt.scatter(X_hc[y_hc == 2, 0], X_hc[y_hc == 2, 1], s = 100, c = 'green', label

# Plotting data points from Cluster 4
plt.scatter(X_hc[y_hc == 3, 0], X_hc[y_hc == 3, 1], s = 100, c = 'cyan', label =

# Plotting data points from Cluster 5
plt.scatter(X_hc[y_hc == 4, 0], X_hc[y_hc == 4, 1], s = 100, c = 'magenta', labe
```

```
plt.title('Scatter Plot of Clusters')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.legend()
plt.show()
```



In [100…

```
# silhouette score
from sklearn.metrics import silhouette_score

sil_sc = silhouette_score(X_hc, y_hc)
sil_sc
```

Out[100…

```
0.7011420828859187
```