1.IMPORTING THE DATA AND PRE-PROCESSING

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
```

In [2]:

```
iris=load_iris()
data=pd.DataFrame(iris.data,columns=iris.feature_names)
data['species']=iris.target
```

In [3]:

```
print(type(iris))
```

```
<class 'sklearn.utils._bunch.Bunch'>
```

In [4]:

```
iris.keys()
```

Out[4]:

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_
names', 'filename', 'data_module'])
```

In [5]:

```
iris['target_names']
data['target']=iris.target
```

In [6]:

```
data
```

Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	target
0	5.1	3.5	1.4	0.2	0	0
1	4.9	3.0	1.4	0.2	0	0
2	4.7	3.2	1.3	0.2	0	0
3	4.6	3.1	1.5	0.2	0	0
4	5.0	3.6	1.4	0.2	0	0
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2	2
146	6.3	2.5	5.0	1.9	2	2
147	6.5	3.0	5.2	2.0	2	2
148	6.2	3.4	5.4	2.3	2	2
149	5.9	3.0	5.1	1.8	2	2

150 rows × 6 columns

In [7]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 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
 4   species            150 non-null    int32  
 5   target             150 non-null    int32  
dtypes: float64(4), int32(2)
memory usage: 6.0 KB
```

In [8]:

```
data.isnull().sum()
```

Out[8]:

```
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
species              0
target               0
dtype: int64
```

In [9]:

```
data.describe()
```

Out[9]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	target
count	150.000000	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000	2.000000

2.CLUSTERNG ALGORITHM IMPLEMENTATION

A.KMeans Clustering

In [11]:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
```

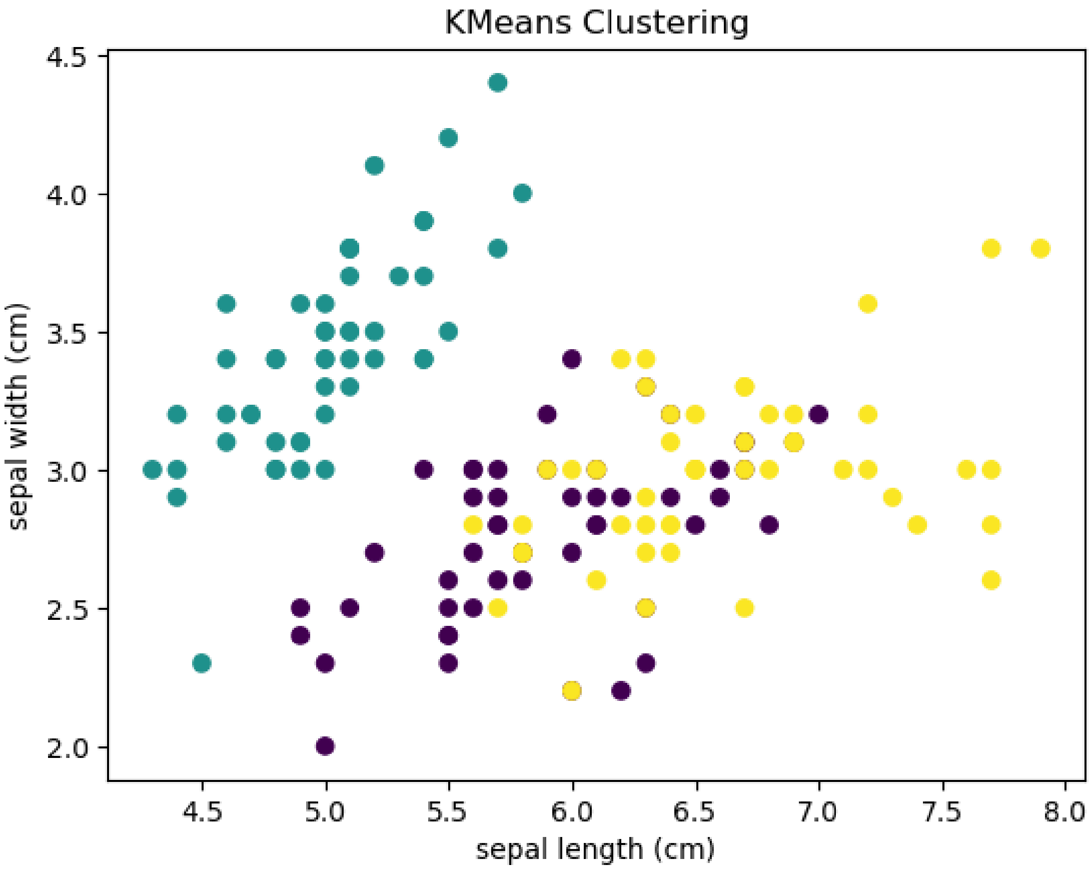
In [12]:

```
kmeans = KMeans(n_clusters=3, random_state=42)
data['Cluster'] = kmeans.fit_predict(data)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:14
12: FutureWarning: The default value of `n_init` will change from 10 to
'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warn
ing
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:14
36: UserWarning: KMeans is known to have a memory leak on Windows with M
KL, when there are less chunks than available threads. You can avoid it
by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [14]:

```
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=data['Cluster'], cmap='vi
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('KMeans Clustering')
plt.show()
```



B.Hierarchical Clustering

In [16]:

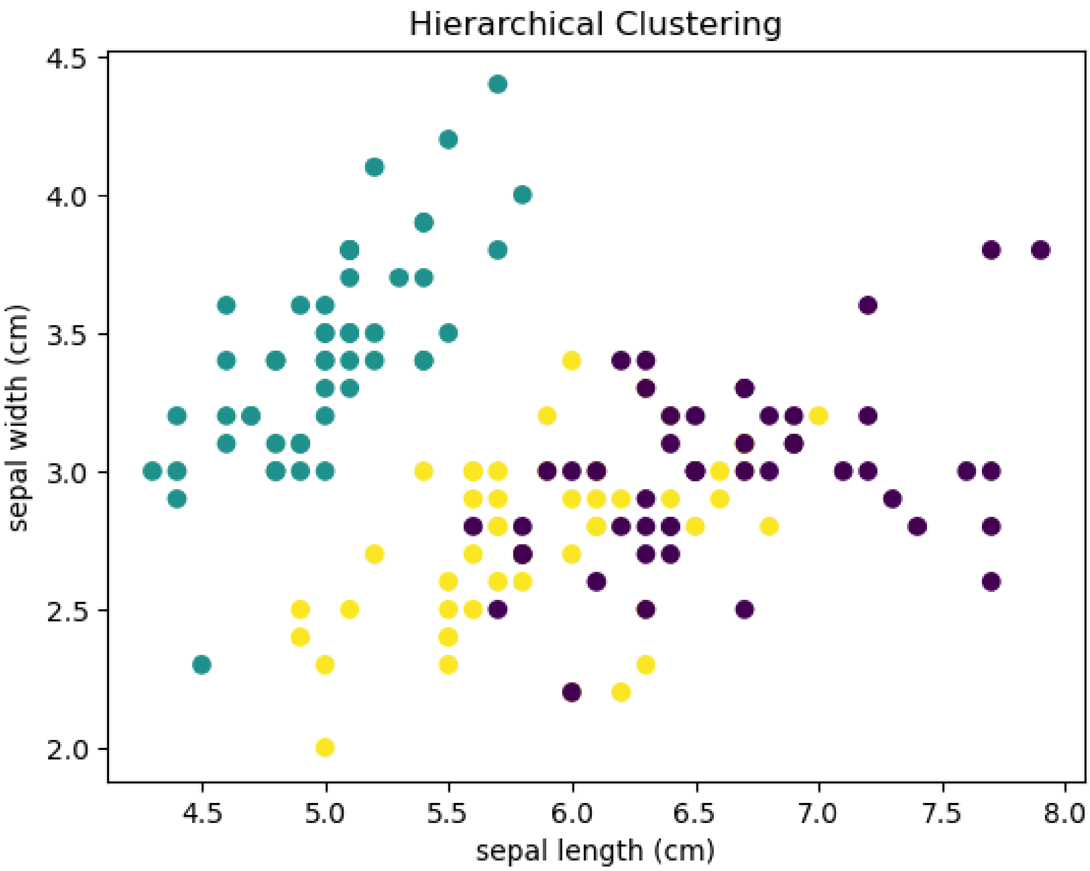
```
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram,linkage
```

In [17]:

```
hier_clust = AgglomerativeClustering(n_clusters=3)
data['HierCluster'] = hier_clust.fit_predict(data)
```

In [18]:

```
plt.scatter(data.iloc[:, 0], data.iloc[:, 1], c=data['HierCluster'], cmap=
plt.xlabel(iris.feature_names[0])
plt.ylabel(iris.feature_names[1])
plt.title('Hierarchical Clustering')
plt.show()
```


Hierarchical Clustering
sepal width (cm)
sepal length (cm)