LOADING AND PREPROCESSING IRIS DATASET

```
import seaborn as sns
```

```
data=sns.load_dataset('iris')
data
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

```
# drop species column

features=data.drop(columns='species',axis=0)
features
```

	sepal_length	sepal_width	petal_length	petal_width
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
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Next steps: Generate code with `features` | View recommended plots | New interactive sheet

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
features.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Kmeans method

Brief description of how KMeans clustering work:

Initialization: Choose K initial centroids randomly from the dataset.

Assignment Step: Assign each data point to the nearest centroid, creating K clusters.

Update Step: Calculate the new centroids by averaging the points in each cluster.

Iteration: Repeat the assignment and update steps until the centroids no longer change significantly or a predefined number of iterations is reached.

The goal is to minimize the within-cluster variance, which means making the points in each cluster as similar as possible while maximizing the differences between clusters.

why KMeans clustering might be suitable for the Iris dataset?

KMeans clustering is particularly suitable for the Iris dataset due to its distinct clusters, as the three iris species show clear separability in their features. The dataset's low dimensionality, with only four dimensions, makes distance computations efficient. Additionally, KMeans operates as an unsupervised learning algorithm, allowing it to uncover patterns without requiring labeled data. Its simplicity makes it easy to implement and understand, while the clusters produced can be effectively visualized and interpreted. These characteristics make KMeans an ideal choice for analyzing the Iris dataset.

```
from sklearn.preprocessing import StandardScaler
standard=StandardScaler()
scaled_features=standard.fit_transform(features)
```

```
from sklearn.cluster import KMeans
```

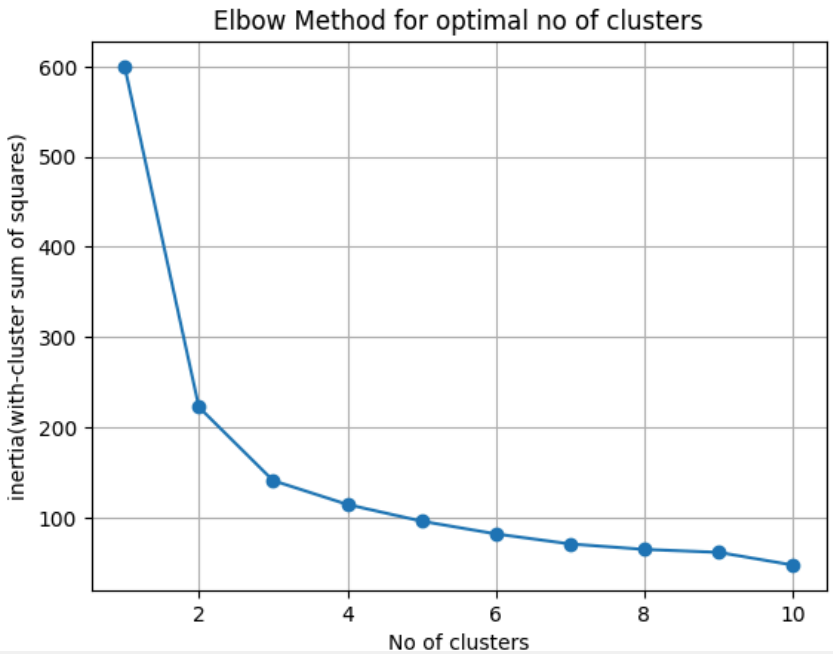
```
inertia=[]
k_values=range(1,11)

for k in k_values:
    kmeans=KMeans(n_clusters=k)
    kmeans.fit(scaled_features)
    inertia.append(kmeans.inertia_)
```

```
inertia
```

```
[599.9999999999999,
 222.36170496502294,
 140.90153181202442,
 114.41256181896091,
 95.91164442221368,
 81.77708925339263,
 70.65863985830325,
 64.58683391457959,
 61.33836091315736,
 47.24548065652756]
```

```
import matplotlib.pyplot as plt
plt.plot(k_values,inertia,marker="o")
plt.xlabel('No of clusters')
plt.ylabel('inertia(with-cluster sum of squares)')
plt.title('Elbow Method for optimal no of clusters')
plt.grid(True)
plt.show()
```

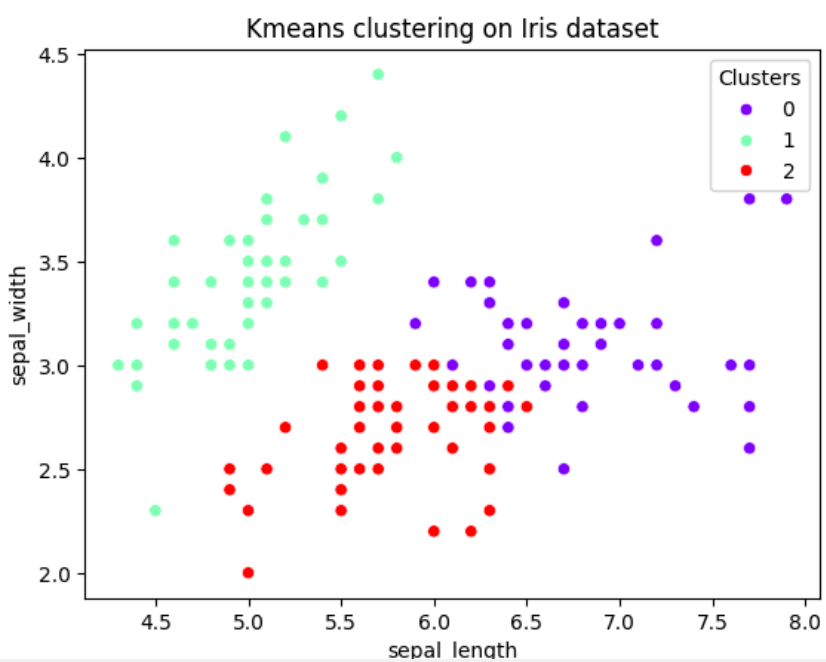



```
from sklearn.cluster import KMeans
kmeans=KMeans(n_clusters=3)
kmeans.fit_predict(scaled_features)
data['Clusters']=kmeans.labels_
```

```
list(data)
```

```
['sepal_length',
 'sepal_width',
 'petal_length',
 'petal_width',
 'species',
 'Clusters']
```

```
import seaborn as sns
sns.scatterplot(x=features['sepal_length'],y=features['sepal_width'],hue=data['Clusters'],palette="rainbow")
plt.title('Kmeans clustering on Iris dataset')
plt.show()
```



Hierarchial Clustering

** Brief description of how Hierarchical clustering works**

Hierarchical clustering is an unsupervised learning method that groups data points into a hierarchy of clusters. It typically uses an agglomerative approach, starting with each data point as an individual cluster and iteratively merging the closest clusters based on a distance metric until one cluster remains or a specified number of clusters is achieved. Alternatively, a divisive approach can start with one cluster and recursively split it. The results are often visualized in a dendrogram, a tree-like diagram that shows the relationships between clusters at various levels of granularity.

why Hierarchical clustering might be suitable for the Iris data SET?

Hierarchical clustering is well-suited for the Iris dataset because it can effectively reveal the natural groupings of the three distinct iris species. It produces a dendrogram for easy visualization of cluster relationships, allowing users to determine the optimal number of clusters without needing to specify it in advance. This method provides interpretability of how clusters relate to one another and is robust against variations in cluster shapes and sizes, making it a good choice for exploring the diverse patterns in the Iris dataset.

```
data=sns.load_dataset('iris')
```

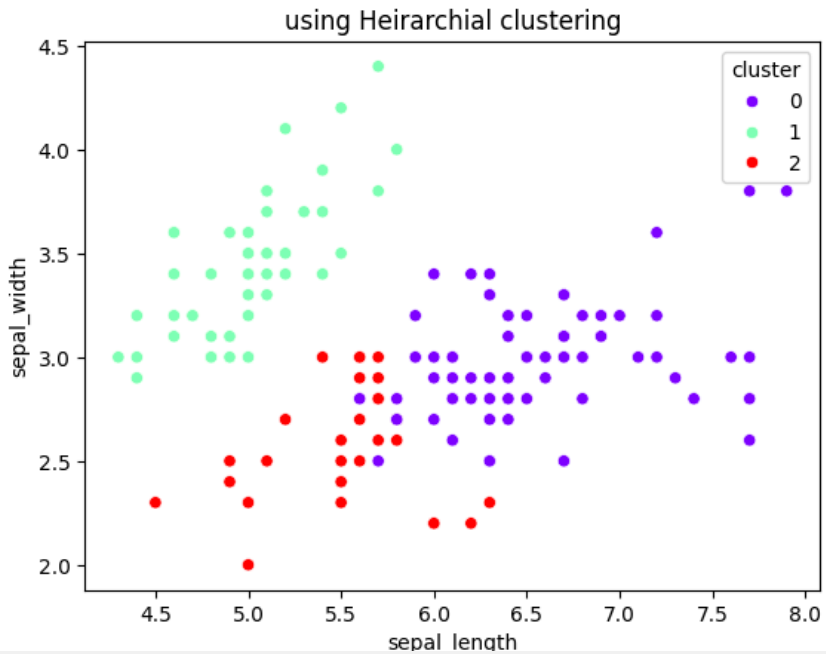
```
from sklearn.cluster import AgglomerativeClustering
hc=AgglomerativeClustering(n_clusters=3,linkage='ward')
data['cluster']=hc.fit_predict(scaled_features)
data
```

	sepal_length	sepal_width	petal_length	petal_width	species	cluster
0	5.1	3.5	1.4	0.2	setosa	1
1	4.9	3.0	1.4	0.2	setosa	1
2	4.7	3.2	1.3	0.2	setosa	1
3	4.6	3.1	1.5	0.2	setosa	1
4	5.0	3.6	1.4	0.2	setosa	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	virginica	0
146	6.3	2.5	5.0	1.9	virginica	0
147	6.5	3.0	5.2	2.0	virginica	0
148	6.2	3.4	5.4	2.3	virginica	0
149	5.9	3.0	5.1	1.8	virginica	0

150 rows × 6 columns

Next steps: Generate code with data | View recommended plots | New interactive sheet

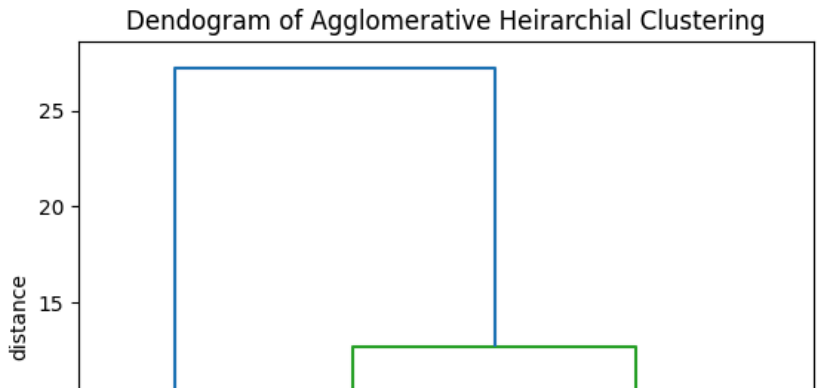
```
sns.scatterplot(x='sepal_length',y='sepal_width',data=data,hue='cluster',palette='rainbow')
plt.title('using Heirarchial clustering')
plt.show()
```




```
from scipy.cluster.hierarchy import dendrogram,linkage
z=linkage(scaled_features,method='ward')
```

```
lab=data['species'].tolist()
```

```
dendrogram(z,labels=lab,leaf_rotation=90)
plt.title('Dendogram of Agglomerative Heirarchial Clustering')
plt.xlabel('species')
plt.ylabel('distance')
plt.show()
```



Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.