52. Clustering in ML

Unsupervised learning is divided into:

1. **Clustering** - divide the data into different clusters, classify/categorize the data
2. **Association** - Arrangement of data

List of some popular unsupervised learning algorithm:

- K-means clustering
- Hierarchal clustering
- DBSCAN Clustering
- Apriori Algorithm / F Growth
- Principle Component Analysis

```
In [ ]:
```

53. K-Means Clustering

- K-Means Clustering is an unsupervised learning algorithm, which groups the unlabelled dataset into different clusters.
- K defines the number of pre-defined clusters that need to be created in the process.

K-Means algo:

- First decide the centriod, center in the dataset
- take two data point, and draw a line b/w them
- pass another line from middle of the line
- take neighbouring data points from the decided central data point

How K-Means work:

1. Take random sample point
2. Create groups
3. Search nearest point
4. Calculate mean (Move points)

No description has been provided for this image

Elbow Method:

- The Elbow method is one of the most popular ways to find the optimal number of clusters
- This method uses the concept of WCSS value. WCSS stands forWithin-Cluster Sum of Squares, which defines the total variations within a cluster.
- The formuls of **WCSS** is:

$$\text{WCSS} = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where:

- (K) = Number of clusters
- (C_i) = (i)-th cluster
- (x) = A data point in cluster (C_i)
- (\mu_i) = Centroid of cluster (C_i)
- (| x - \mu_i |) = Euclidean distance between data point (x) and centroid (\mu_i)

How does WCSS is calculated:

- Caclulate the distance from decided central data point and its neighbouring data points (x - u)
- Take square of the distance
- Sum of the distances from central point and all neighbouring data points

No description has been provided for this image

K-Means ++: To have best clustering in the data. It takes 2 decided points away from each other.

In [ ]:

54. K-Means Clustering (Practical)

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
dataset = pd.read_csv(r'Data/iris_raw.csv')
dataset.head(3)
```

Out[2]:

	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

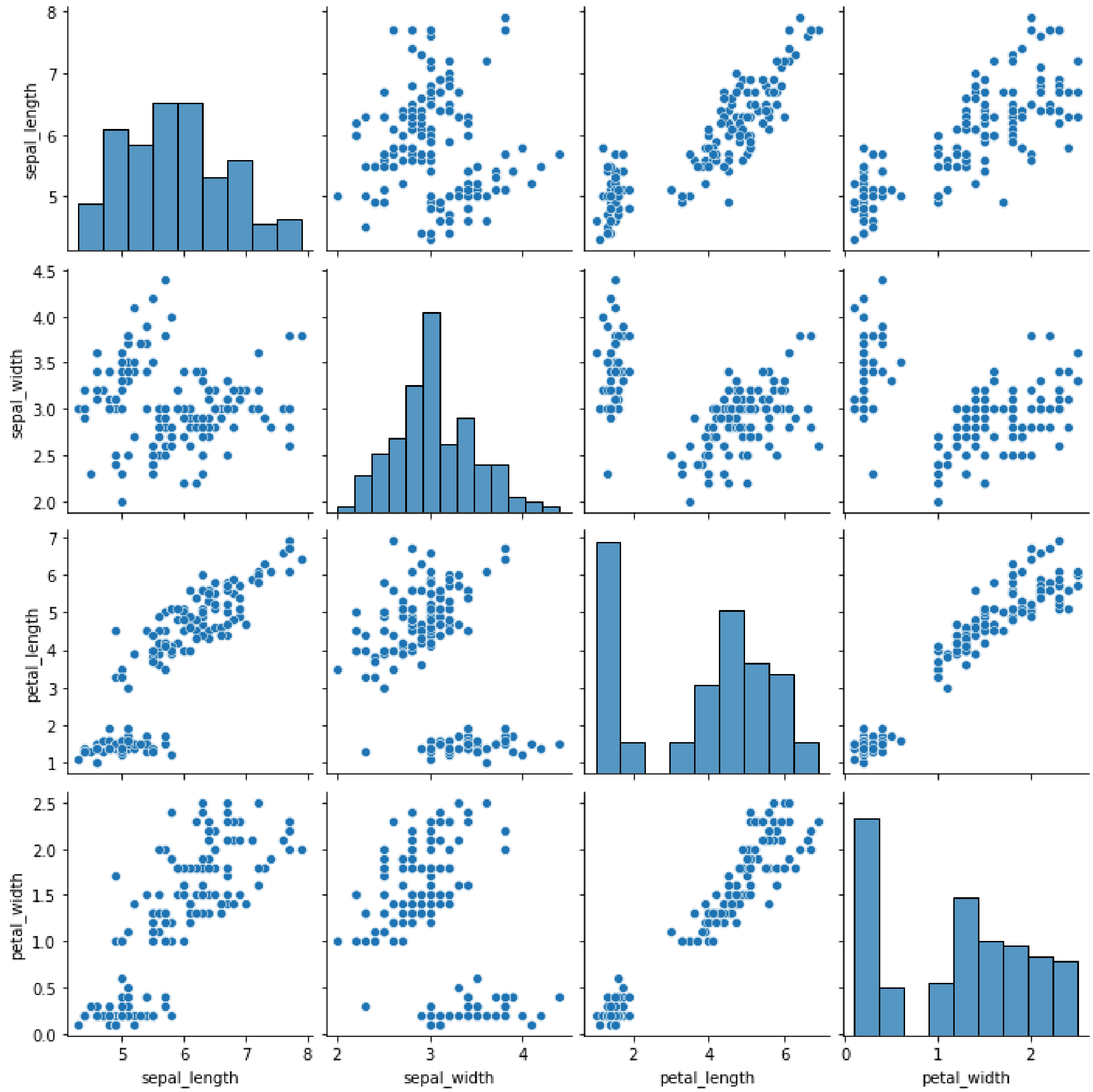
54.1 Making Clusters of Data

- Use K-mean clustering when **your data is linearly separable**

Check the data if it is linearly separable

In [3]:
```
sns.pairplot(data=dataset)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



- In supervised learning, the data is split into training and testing data
- In unsupervised learning, data is not split into training and testing data b/c the data is unlabelled

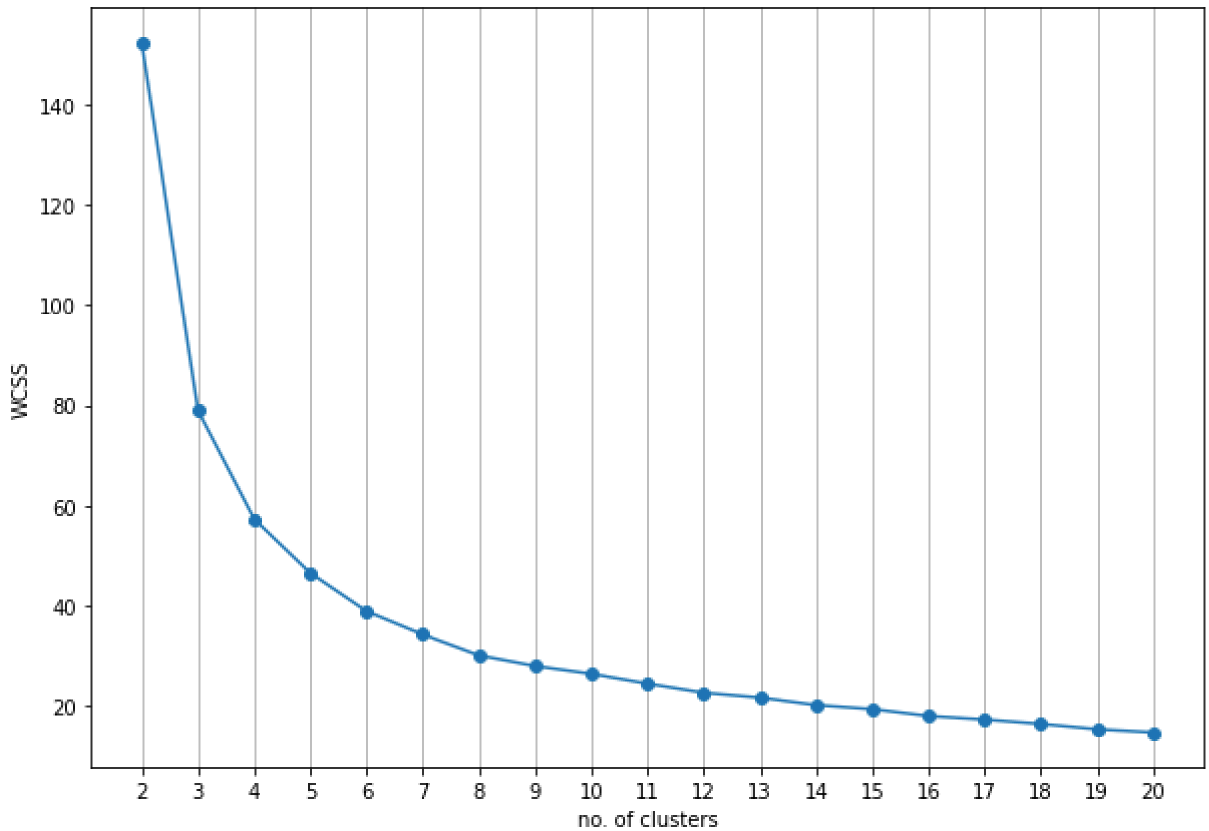
54.1.1 Find Number of clusters

In [7]:
```
from sklearn.cluster import KMeans
```

In [14]:
```
# Use a loop to find best number of clusters from 2 to 20
wcss = []

for i in range(2,21):
    km = KMeans(n_clusters=i, init='k-means++')
    km.fit(dataset)
    wcss.append(km.inertia_) # it assings value of wcss {Elbow graph}
```

In [29]:
```
plt.figure(figsize=(10,7))
plt.plot([i for i in range(2,21)], wcss, marker='o')
plt.xlabel('no. of clusters')
plt.xticks([i for i in range(2,21)])
plt.ylabel('WCSS')
plt.grid(axis='x')
plt.show()
```



Elbow point = 3

It means that will have 3 number of clusters

In [ ]:

In [30]:
```
kmn = KMeans(n_clusters=3)
kmn.fit_predict(dataset)
```

Out[30]:
```
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, 2, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

In [32]:
```
dataset['Predict'] = kmn.fit_predict(dataset)
```

In [33]:
```
dataset
```

Out[33]:

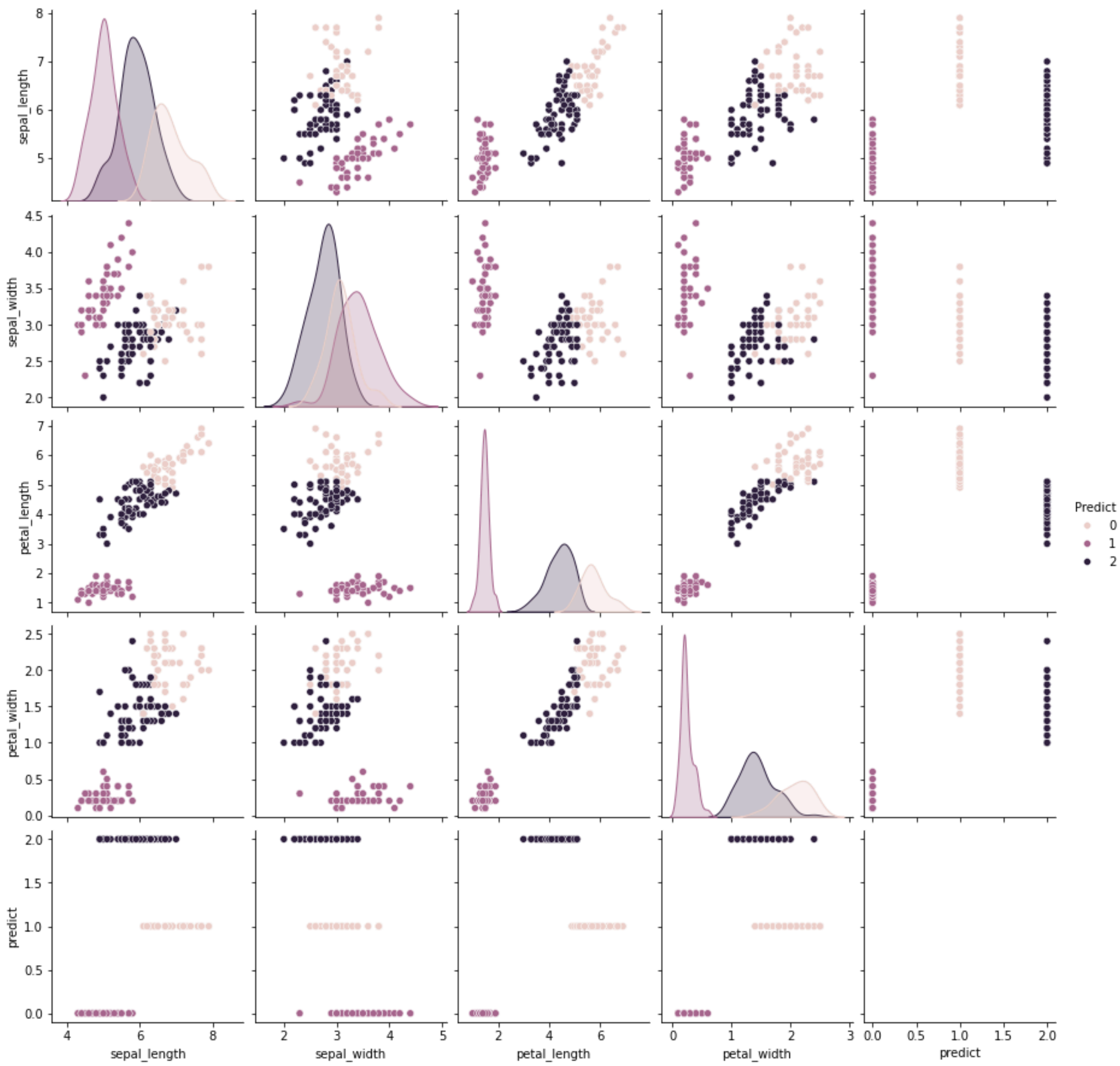
	sepal_length	sepal_width	petal_length	petal_width	predict	Predict
0	5.1	3.5	1.4	0.2	0	1
1	4.9	3.0	1.4	0.2	0	1
2	4.7	3.2	1.3	0.2	0	1
3	4.6	3.1	1.5	0.2	0	1
4	5.0	3.6	1.4	0.2	0	1
...	...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	1	0
146	6.3	2.5	5.0	1.9	2	2
147	6.5	3.0	5.2	2.0	1	0
148	6.2	3.4	5.4	2.3	1	0
149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

150 rows × 6 columns

In [39]:

```
sns.pairplot(data=dataset, hue='Predict')
plt.savefig(r"Generated_images/raw-iris-clustering-predict.jpg")
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



54.2 Making raw data with original data

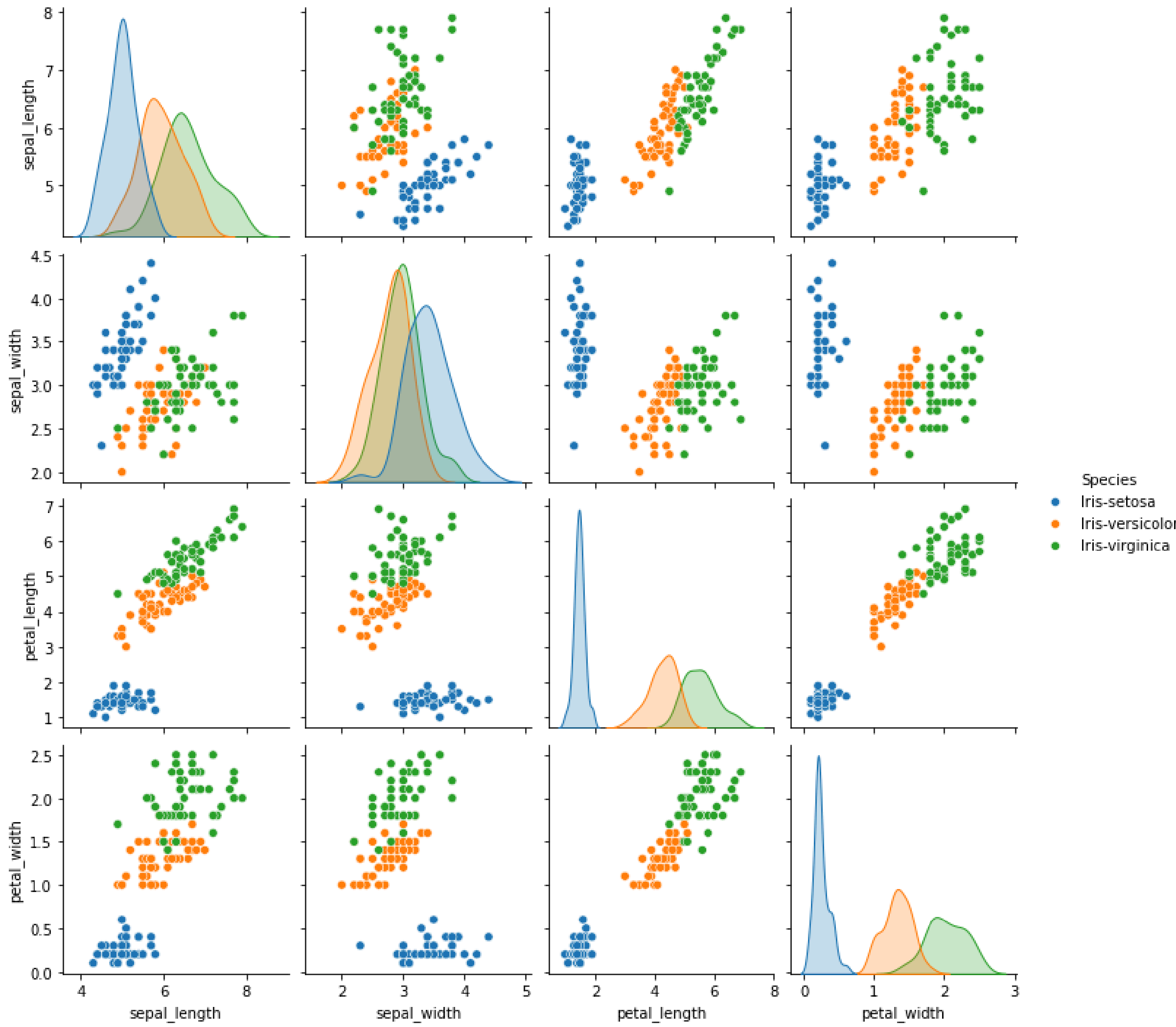
In [35]:
```
org_dataset = pd.read_csv(r'Data/iris.csv')
org_dataset.head(3)
```

Out[35]:

	sepal_length	sepal_width	petal_length	petal_width	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa

In [40]:
```
sns.pairplot(data=org_dataset, hue='Species')
plt.savefig(r"Generated_images/raw-iris-clustering-original-data.jpg")
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```




```
In [ ]:
```

55. Hierarchical Clustering

It is applied for linearly separable data

- It is used to group the unlabelled datasets into a cluster and aslo known as hierarchical cluster analysis or HCA.
- In the algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogam**.

Dendrogram

- It is a tree like structure that is mainly used to store each step as a memory that the HC algorithm performs.
- The dendrogram plot, the Y-axis shows the **Euclidean distances** b/w the data points, and the x-axis shows all the data points of the given dataset.

No description has been provided for this image

Hierarchical clustering technique has two approaches:

1. **Agglomerate:** Agglomerative is a bottom-up approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left. This is popular algorithem and **bottom-up approach**.
2. **Divisive:** Divisive algorithm is the reverse of the agglomerative algorithm as it is **top-down approach**.

Agglomerate Clustering:

No description has been provided for this image

Divisive Clustering:

No description has been provided for this image

Measure for the distance between two clusters

- The closest distance b/w the two clusters is crucial for the hierarchical clustering.
- There are various ways to calculate the distance b/w two clusters, and these ways decided the rule for clustering. These measures are called **Linkage methods:**

- **Single Linkage** - We take minimum distance b/w two clusters
- **Complete Linkage** - We take maximum distance b/w two clusters
- **Average Linkage** - We take average distance b/w two clusters
- **Centroid Linkage** - We take central point and then calculate distance b/w two clusters

No description has been provided for this image

No description has been provided for this image

No description has been provided for this image

No description has been provided for this image

To desing best number of clusters

No description has been provided for this image

In [ ]:

56. Agglomerate Hierarchical (Practical)

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
dataset = pd.read_csv(r'Data/iris_raw.csv')
dataset.head(3)
```

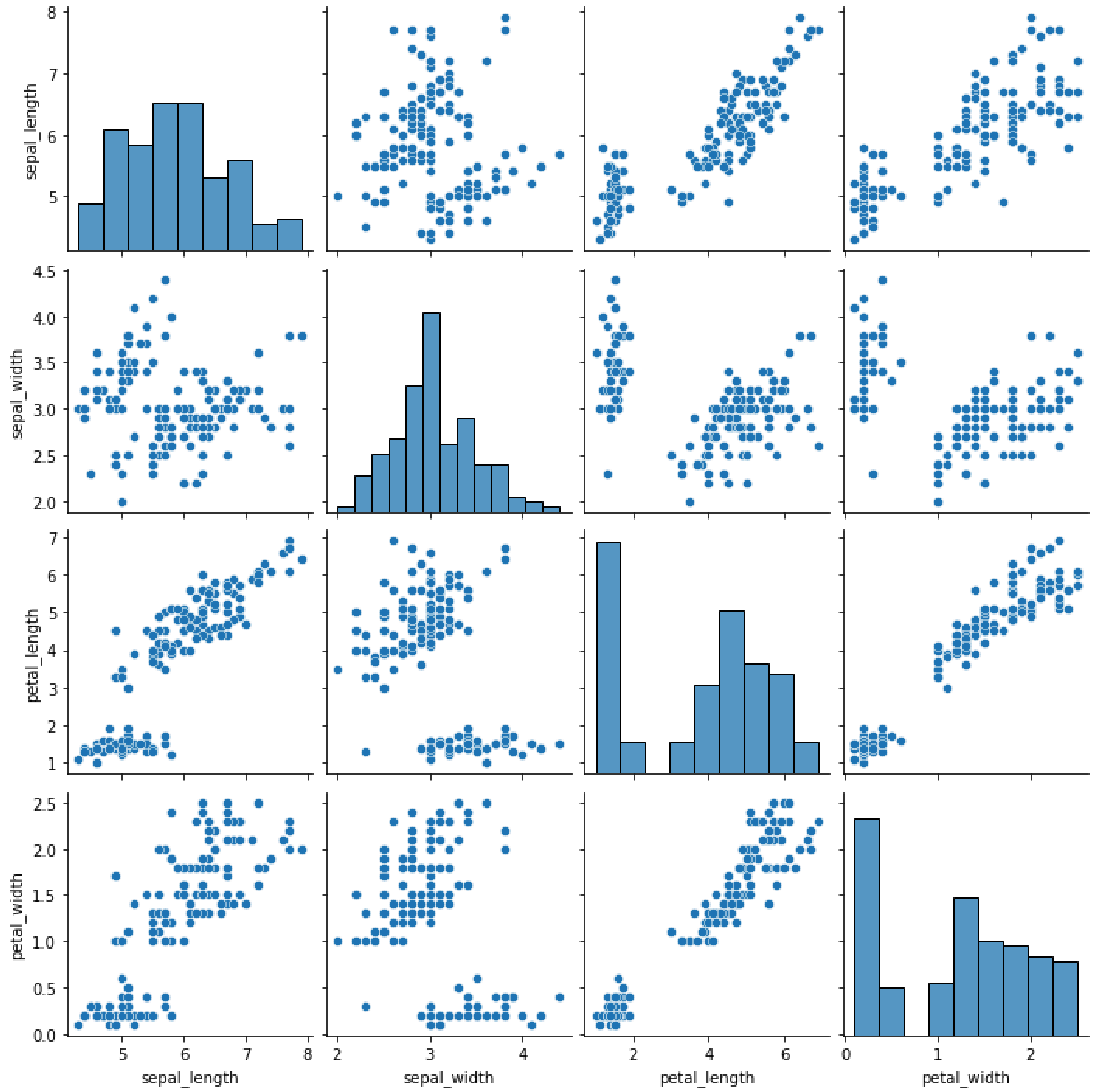
Out[3]:

	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

As agglomerate clustering works on **linearly separable data**, so we will see if our data is linear or not through graph

In [4]:
```
sns.pairplot(data=dataset)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



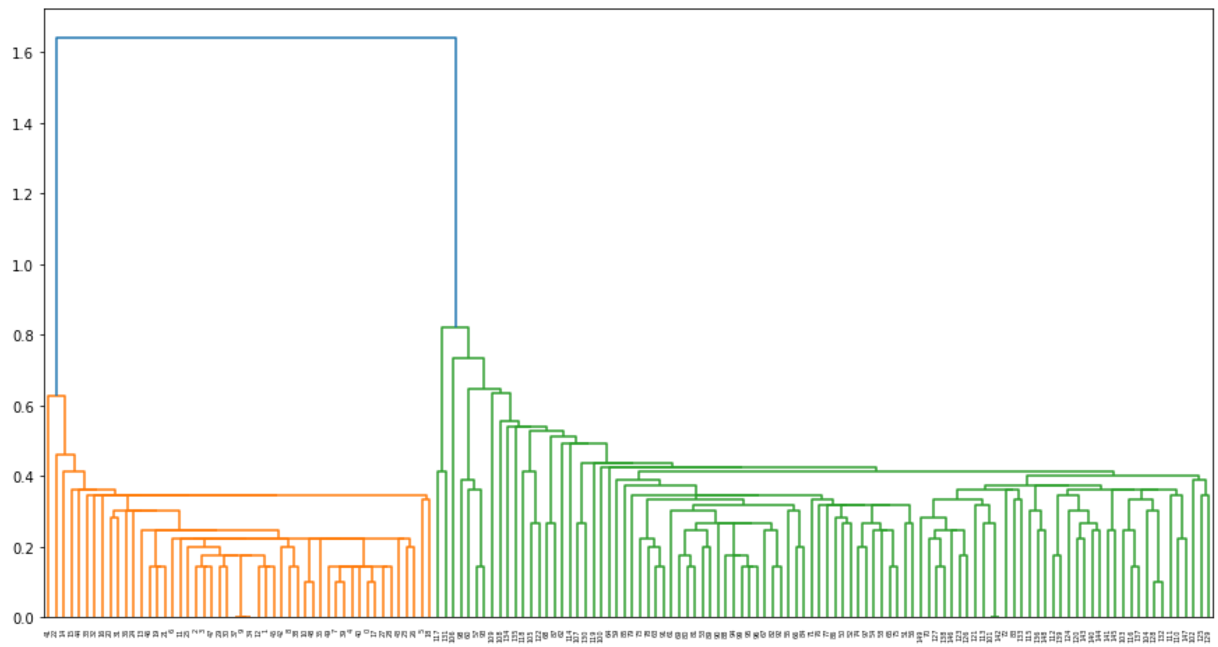
Make Dendrogram

SciPy Library is needed for making dendrogram

In [6]:
```
import scipy.cluster.hierarchy as sc
```

We will need **Linkage** fro making dendrogram

In [10]:
```
'''Z : ndarray
    The linkage matrix encoding the hierarchical clustering to
    render as a dendrogram. See the ``linkage`` function for more
    information on the format of ``Z``.'''
plt.figure(figsize=(15,8))
sc.dendrogram(sc.linkage(dataset, method='single', metric='euclidean'))
plt.savefig(r'Generated_images/dendrogram.jpg')
plt.show()
```



Dendrogram is showing two clusters only in the data

In [11]:
```
from sklearn.cluster import AgglomerativeClustering
```

In [13]:
```
ac = AgglomerativeClustering(n_clusters=2, linkage='single')
ac.fit_predict(dataset)
```

Out[13]:
```
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, 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, 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], dtype=int64)
```

In [14]:
```
dataset['Predict'] = ac.fit_predict(dataset)
```

In [15]:
```
dataset
```

Out[15]:

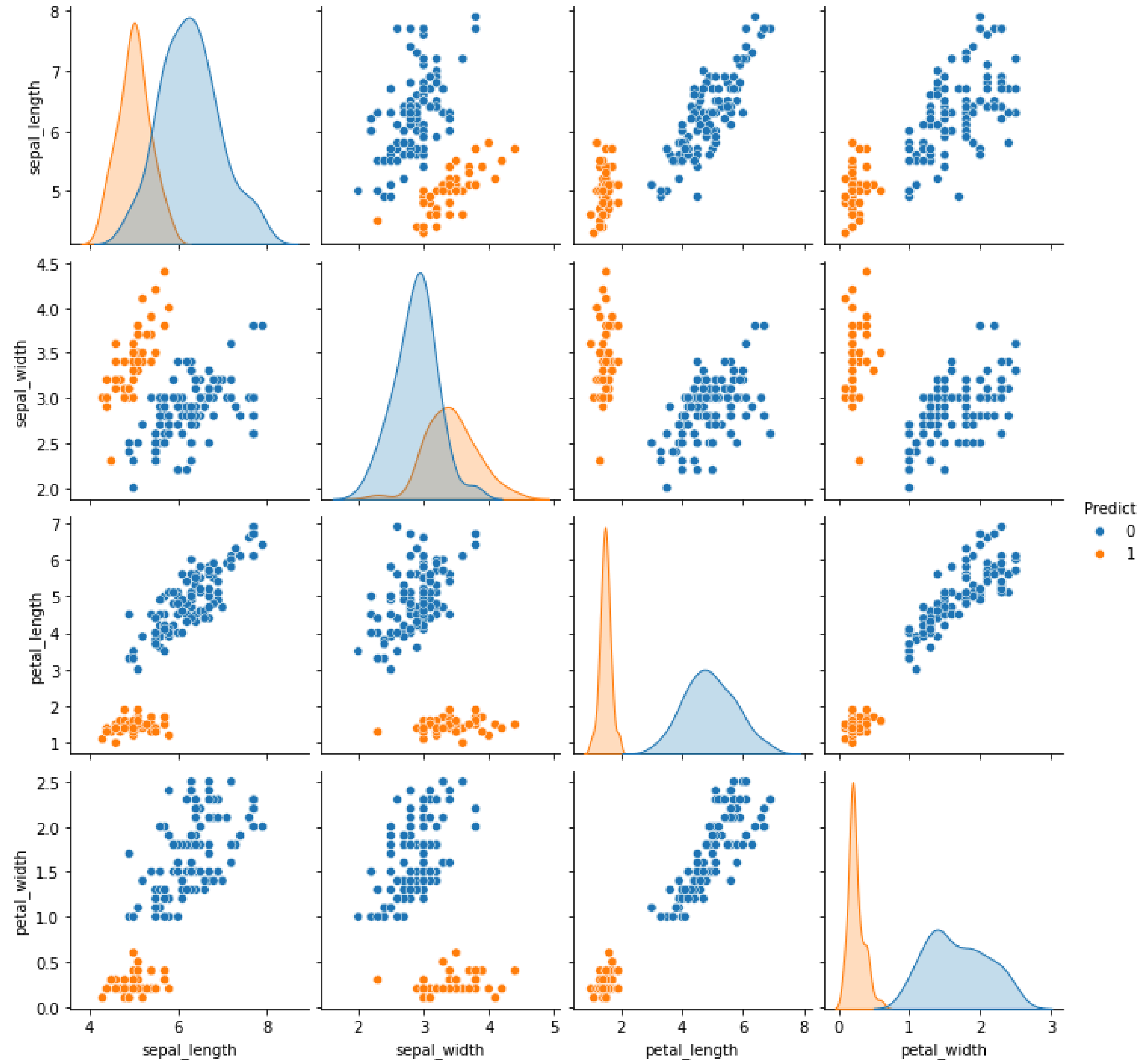
	sepal_length	sepal_width	petal_length	petal_width	Predict
0	5.1	3.5	1.4	0.2	1
1	4.9	3.0	1.4	0.2	1
2	4.7	3.2	1.3	0.2	1
3	4.6	3.1	1.5	0.2	1
4	5.0	3.6	1.4	0.2	1
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	0
146	6.3	2.5	5.0	1.9	0
147	6.5	3.0	5.2	2.0	0
148	6.2	3.4	5.4	2.3	0
149	5.9	3.0	5.1	1.8	0

150 rows × 5 columns

In [16]:

```
sns.pairplot(data=dataset, hue='Predict')
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



Prediction is also showing two clusters only in the data

In [ ]:

57. DBScan Clustering Algorithm

- Density-Based Spatial Clustering of Applications with Noise.
- The clusters found by DBScan can be any shape, which can deal with some special cases that other methods cannot.
- It is used for **non-linear separable data**
- DBScan Clustering also used in **detection of outlier in the data**

No description has been provided for this image

Requirements for DBCLUSTRING

1. Minimum points (at least 4)
2. Espsilon (radius)
3. Core point (#points >= minpoints)
4. Boundary point (#points < minpoints)
5. Noise Point (outlier)

In [ ]:

```

```

58. DBScan Clustering Algorithm (Practical)

In [33]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_moons
import pandas as pd
```

In [34]:
```
x, y = make_moons(n_samples=250, noise=0.05)
```

In [35]:
```
# First column data
x[:,0]
```

```
Out[35]: array([ 1.82154686,  0.33039765,  1.11621412,  1.75338817,  0.79406254,
                 0.09592268,  0.77490969, -0.96252073,  0.78806259,  0.23980237,
                 1.89848596,  0.01310029,  0.07091097,  0.58314369, -0.96699757,
                 1.04928892,  0.57445541,  0.74501548, -1.00650714, -0.82152061,
                 0.86190156,  0.49126856,  1.8694458 ,  1.49802939,  0.6728559 ,
                 0.98720469,  0.45047353, -0.2865276 ,  0.42354262,  0.88007291,
                 0.36500296,  1.03385062,  0.3360013 , -0.01448323,  0.89163708,
                -0.49418874, -0.35425069,  0.50247661,  0.85778933, -0.64054093,
                 0.04832991,  2.01015645,  1.2204363 ,  1.88956628, -0.13252252,
                -0.49523227,  0.33669497,  0.76153465, -0.27186532,  1.28870288,
                 0.93076493,  1.90104103,  0.02543826,  0.44853317,  1.00448433,
                -0.25031522,  1.92847973,  0.42609288,  1.27725596, -0.87659926,
                 0.61413887,  0.22505794, -0.04512895,  0.69131929,  0.259285  ,
                 1.84036998,  0.47866329, -0.2844214 ,  1.31106194, -0.23293884,
                 1.31976039,  1.93209062,  0.01521537,  1.80101873, -0.0391596 ,
                -0.49585375,  1.58238117,  0.01637799,  0.72630669,  0.91538441,
                -0.02865494,  1.45737014,  0.64884606, -0.74483793,  1.06418573,
                 2.01778072, -0.37167832, -0.81173842, -0.91412945,  0.91606385,
                 0.6136725 , -0.43538794, -0.7255368 ,  0.96102712,  1.91454596,
                -0.89253863,  0.12816908,  0.1788169 ,  0.12848506, -0.63151123,
                -0.86587253,  0.71871253,  0.04011572, -0.98749825,  1.50781231,
                 0.87760821, -1.00423248,  1.44746249,  0.78287742, -0.44868368,
                 1.89144999,  0.80058548,  1.88844507, -0.42427662, -1.02605411,
                -0.06589804,  1.92603512, -0.80463147,  1.51462527,  0.09590861,
                 1.67803099, -0.70384095,  0.99436775, -0.10206081,  0.05106181,
                 1.62449215, -0.90402427,  1.42745557,  1.92430944,  0.29091015,
                 0.80306352,  1.4728536 ,  0.6063805 ,  1.07151034,  0.77765877,
                -0.79018459,  0.07756758, -0.6810462 ,  0.23387981, -0.9338624 ,
                 0.4196469 ,  1.0743206 , -1.07958865,  0.66710905, -0.98994215,
                 0.66449098,  0.49932715,  0.82120004,  0.48939705,  1.63825117,
                 1.86809186, -0.91683607,  1.96477994, -0.48955672,  0.56543806,
                 0.22563531, -0.93097574,  0.07804955,  0.68306482,  0.61259675,
                 2.0154066 ,  1.63111271,  0.99459508,  0.613061  ,  0.34693487,
                -0.65125101, -1.08710097,  0.05320543, -0.97583562,  0.14590846,
                 1.28499957, -0.01456049,  0.99183249,  1.95048888,  0.22331433,
                 0.09418366,  0.84157974,  0.15503721, -0.73386657,  1.05716677,
                 0.4758061 ,  1.20194107,  1.76139437,  0.99175489,  0.88443974,
                 1.77304674,  0.72292497,  0.02948542,  0.19793304,  0.77849153,
                 0.23332147, -0.07093789,  0.35773924, -0.80533347,  0.36514192,
                -0.00781175,  0.54951133,  1.74424235,  0.05659029, -0.85842631,
                 1.69412089,  0.08545157,  0.30568356,  1.72318384, -0.04890115,
                -0.07264175, -0.63270247,  0.15343203, -0.69145099,  0.94662445,
                 0.84613359,  0.22515933,  1.64017442,  0.95298684, -0.0749229 ,
                 0.32118047, -0.56206221, -1.06808907,  1.20337727,  0.03416578,
                -1.00746399,  0.94031733,  1.99598102, -0.19158745,  0.28515341,
                 1.30521283, -0.55735516,  1.36320489, -0.83877886,  1.08372752,
                 0.45287718,  0.8876371 ,  1.03408887,  0.23404788,  1.89919484,
                 0.80054814,  2.0020812 ,  1.0442318 ,  1.86594006,  1.72375506,
                -0.3131739 ,  0.13166397,  1.04628395,  1.88899169,  1.98677442,
                 2.03239475,  0.43570842,  1.04879911,  1.04383872, -0.80846152])
```

```
In [36]: df = {"data1":x[:,0], "data2":x[:,1], "output":y}
```

```
In [37]: dataset = pd.DataFrame(df)
```

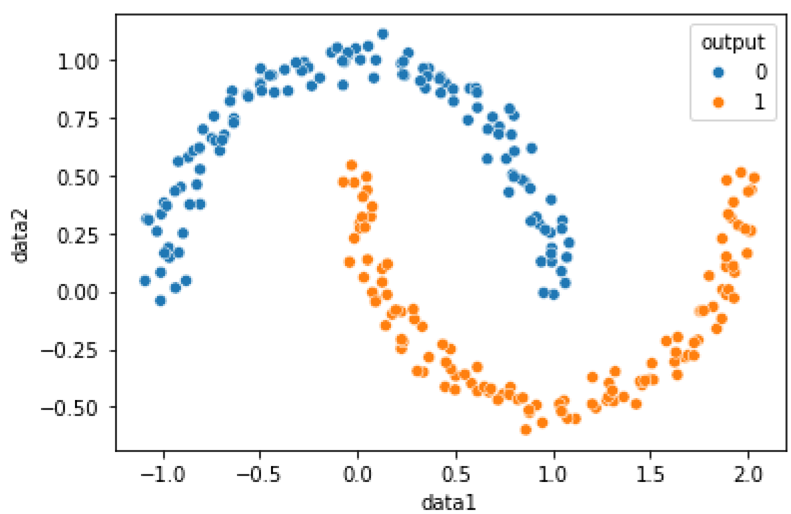
In [38]:
```
dataset
```

Out[38]:

	data1	data2	output
0	1.821547	-0.067198	1
1	0.330398	-0.152611	1
2	1.116214	-0.551311	1
3	1.753388	-0.086491	1
4	0.794063	0.502981	0
...	...	...	...
245	2.032395	0.488380	1
246	0.435708	-0.229602	1
247	1.048799	-0.520698	1
248	1.043839	-0.518346	1
249	-0.808462	0.618646	0

250 rows × 3 columns

In [39]:
```
sns.scatterplot(x='data1', y='data2', data=dataset, hue='output')
plt.show()
```



- The data is **non-linear**, so we will apply DBSCAN Clustering algorithm

In [40]:
```
dataset.head(3)
```

Out[40]:

	data1	data2	output
0	1.821547	-0.067198	1
1	0.330398	-0.152611	1
2	1.116214	-0.551311	1

- We cannot apply DBSCAN on 0 and 1 as this is present in output column, so will remove this column before applying this algo.

In [41]:
```
dataset.drop('output', axis=1, inplace=True)
```

In [42]:
```
dataset
```

Out[42]:

	data1	data2
0	1.821547	-0.067198
1	0.330398	-0.152611
2	1.116214	-0.551311
3	1.753388	-0.086491
4	0.794063	0.502981
...	...	...
245	2.032395	0.488380
246	0.435708	-0.229602
247	1.048799	-0.520698
248	1.043839	-0.518346
249	-0.808462	0.618646

250 rows × 2 columns

In [43]:
```
from sklearn.cluster import DBSCAN
```

In [44]:
```
db = DBSCAN(eps=0.2, min_samples=5)
db.fit_predict(dataset)
```

Out[44]:
```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 1], dtype=int64)
```

In [45]:
```python
dataset['Predict'] = db.fit_predict(dataset)
```

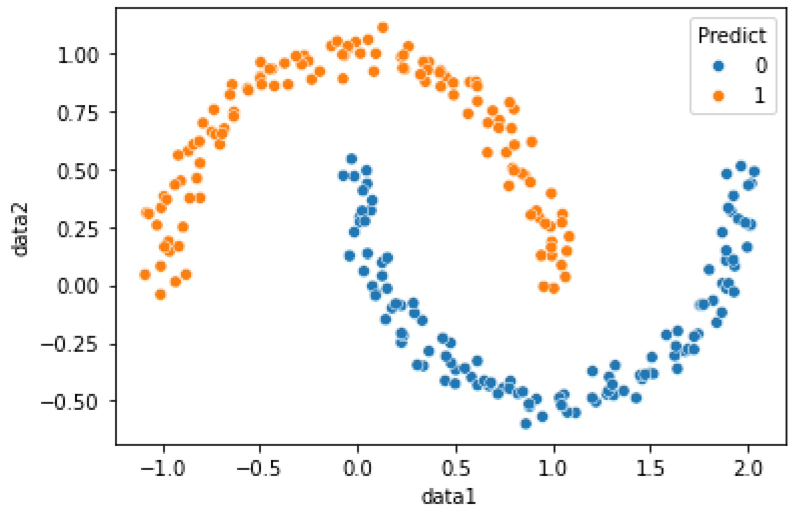
In [46]:
```python
dataset
```

Out[46]:

	data1	data2	Predict
0	1.821547	-0.067198	0
1	0.330398	-0.152611	0
2	1.116214	-0.551311	0
3	1.753388	-0.086491	0
4	0.794063	0.502981	1
...	...	...	...
245	2.032395	0.488380	0
246	0.435708	-0.229602	0
247	1.048799	-0.520698	0
248	1.043839	-0.518346	0
249	-0.808462	0.618646	1

250 rows × 3 columns

In [47]:
```python
sns.scatterplot(x='data1', y='data2', data=dataset, hue='Predict')
plt.show()
```



So predicted data resembles with actual output as shown in the graphs of predict and original data

In [ ]:

59. Silhouette Score

- It validates that cluster predicted from the data are right or wrong number of clusters
- Silhouette refers to a method of interpretation and validation of consistency within clusters of data.
- Silhouette Coefficient or Silhouette Score is a metric used to calculate the goodness of a clustering technique
- Its values ranges from -1 to 1

No description has been provided for this image

Silhouette Score Formula

The Silhouette score is calculated using the following formula:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where:

- (s(i)) is the silhouette score for a data point (i).
- (a(i)) is the mean distance between (i) and all other data points in the same cluster.
- (b(i)) is the mean distance between (i) and all data points in the nearest neighboring cluster.

The Silhouette score ranges from -1 to 1, where:

- A score of 1 indicates that the data point is well clustered.
- A score of 0 indicates that the data point lies on the boundary between clusters.
- A score of -1 indicates that the data point is poorly clustered.

In [ ]:

60. Silhouette Score (Practical)

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
dataset = pd.read_csv(r'Data/iris_raw.csv')
dataset.head(3)
```

Out[2]:

	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

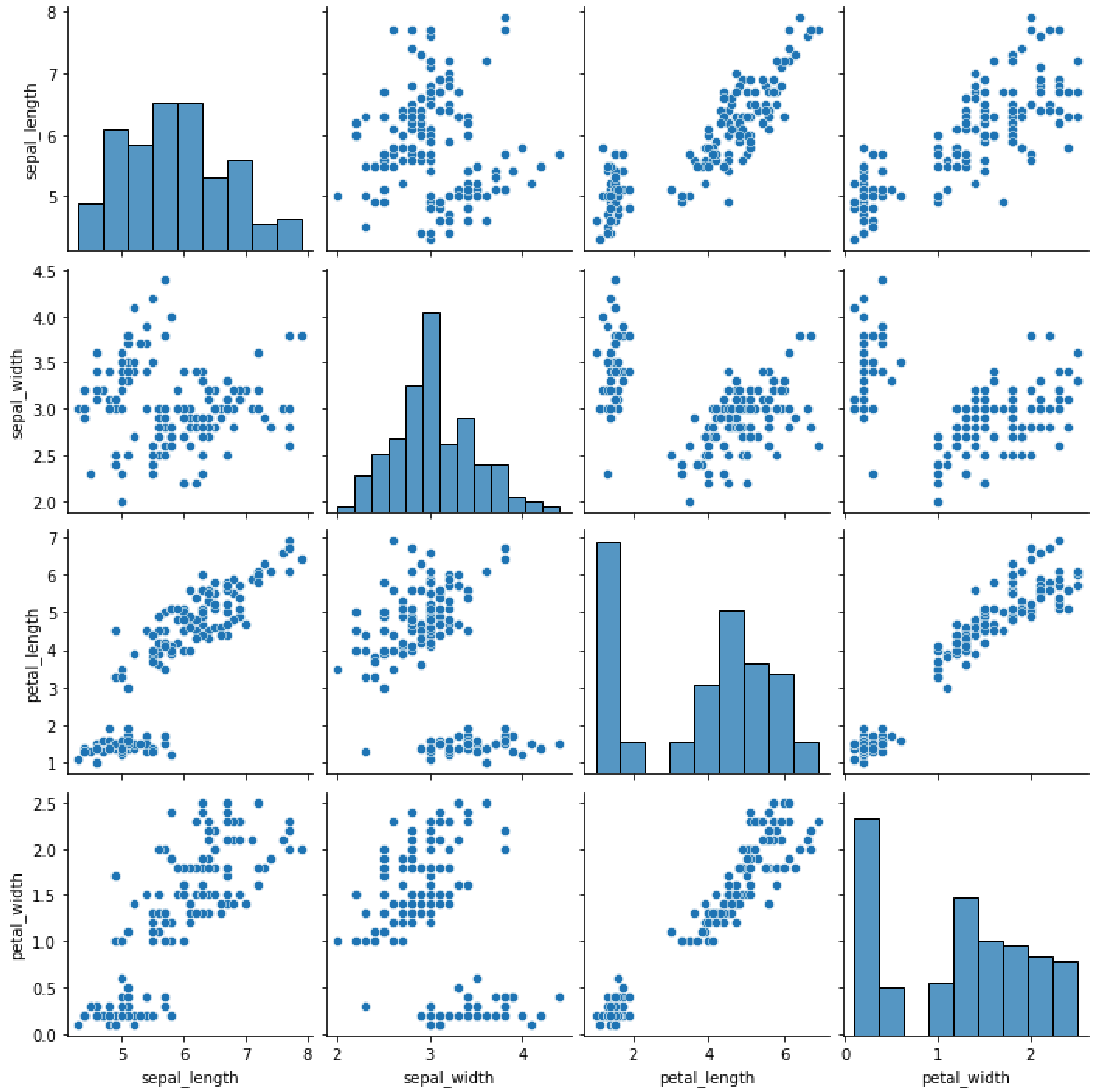
54.1 Making Clusters of Data

- Use K-mean clustering when **your data is linearly separable**

Check the data if it is linearly separable

In [3]:
```
sns.pairplot(data=dataset)
plt.show()
```

```
C:\Users\rashi\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\axis
grid.py:123: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```



- In supervised learning, the data is split into training and testing data
- In unsupervised learning, data is not split into training and testing data b/c the data is unlabelled

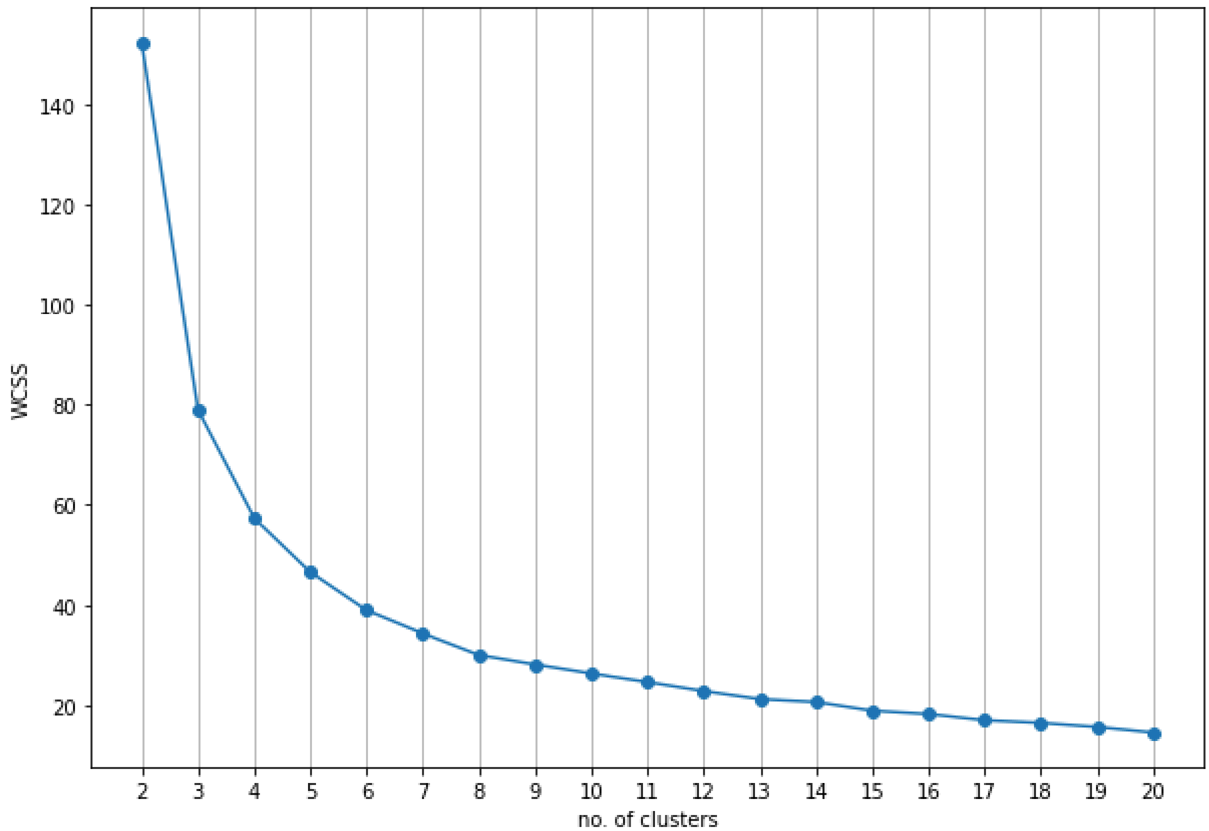
54.1.1 Find Number of clusters

In [4]:
```
from sklearn.cluster import KMeans
```

In [5]:
```
# Use a loop to find best number of clusters from 2 to 20
wcss = []

for i in range(2,21):
    km = KMeans(n_clusters=i, init='k-means++')
    km.fit(dataset)
    wcss.append(km.inertia_) # it assings value of wcss {Elbow graph}
```

In [6]:
```
plt.figure(figsize=(10,7))
plt.plot([i for i in range(2,21)], wcss, marker='o')
plt.xlabel('no. of clusters')
plt.xticks([i for i in range(2,21)])
plt.ylabel('WCSS')
plt.grid(axis='x')
plt.show()
```



Elbow point = 3

It means that will have 3 number of clusters

In [7]:
```
kmn = KMeans(n_clusters=3)
kmn.fit_predict(dataset)
```

Out[7]:
```
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, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

In [8]:
```
dataset['Predict'] = kmn.fit_predict(dataset)
```

In [9]:
```
dataset
```

Out[9]:

	sepal_length	sepal_width	petal_length	petal_width	Predict
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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	1
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	1

150 rows × 5 columns

54.2 Apply Silhouette Score to validate above results

In [17]:
```
from sklearn.metrics import silhouette_score
```

In [18]:
```
kmn.labels_
```

Out[18]:
```
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, 2, 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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 1, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 1, 1, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 1])
```

In [19]:
```
silhouette_score(dataset, labels=kmn.labels_)
```

Out[19]:
```
0.6126634972047179
```

We are not sure about the results. So will apply loop to determine which silhouette_score is best and determine what is acutal number of clusters.

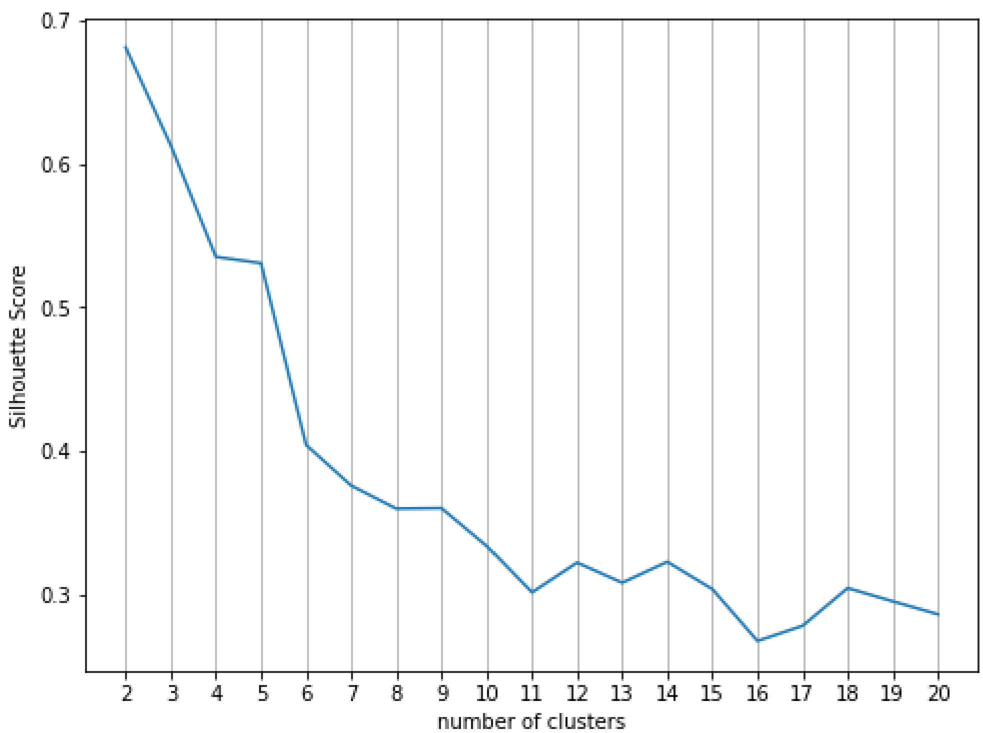
In [29]:
```
ss = []
n_clusters= [j for j in range(2,21)]

for i in range(2,21):
    kmn1 = KMeans(n_clusters=i)
```

```
    kmn1.fit(dataset)
    ss.append(silhouette_score(dataset, labels=kmn1.labels_))
```

- We are going to make graph b/w ss vs #clusters

In [39]:
```
plt.figure(figsize=(8,6))
plt.plot(n_clusters, ss)
plt.xlabel('number of clusters')
plt.ylabel('Silhouette Score')
plt.xticks(n_clusters)
plt.grid(axis='x')
plt.show()
```



Best Silhouette Score = 2

In [ ]: