Machine Learning 🖥️

Unsupervised Learning

By ⇒ PRINCE 👑❤️

https://github.com/pkvidyarthi/

⟶ In Supervised Learning, models are trained using labeled data under the supervision of training data.

⟶ But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

What is Unsupervised Learning ⟶

- **Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.**
- **This can be helpful for tasks such as detecting patterns in datanor predicting future events.**
- **Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to find the basic structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.**

Usage :

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Unsupervised Learning algorithms:

List of some popular unsupervised learning algorithms:

- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierarchal clustering**
- **Anomaly detection**
- **Neural Networks**
- **Principle Component Analysis**
- **Independent Component Analysis**
- **Apriori algorithm**
- **Singular value decomposition**

Types of Unsupervised Learning →

1. **Clustering**
2. **Association**

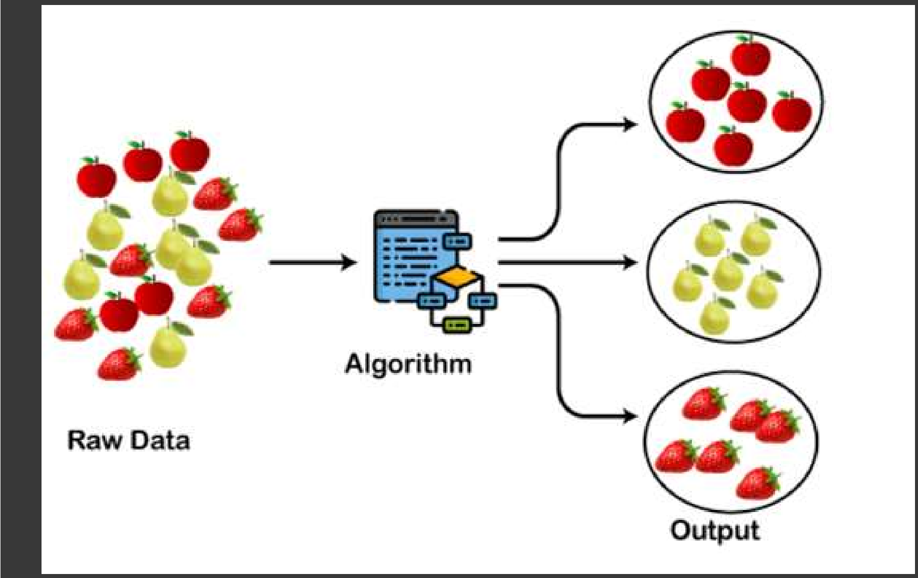
1. Clustering →

- Clustering or cluster analysis is a machine learning technique, which groups the unlabelled dataset.
- A way of grouping the data points into different clusters, consisting of similar data points. The objects with the possible similarities remain in a group that has less or no similarities with another group.
- It does it by finding some similar patterns in the unlabelled dataset such as shape, size, color, behaviour, etc and divides them as per the presence and absence of those similar patterns.
- The clustering technique is commonly used for statistical data analysis.

most common uses of this technique are:

- **Market Segmentation**
- **Statistical data analysis**
- **Social network analysis**
- **Image segmentation**
- **Anomaly detection,** etc.

Example :



2. Association [Association Rule] Learning →

- Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable.
- It is based on different rules to discover the interesting relations between variables in the database.
- Association Rule Learning is employed in Market Basket analysis, Web usage mining, continuous production, etc.

K - Means Clustering Algorithm →

- **" It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties. "**
- ** K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.**
- **It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.**
- **The k-means clustering algorithm mainly performs two tasks:**
  - Determines the best value for K center points or centroids by an iterative process.

  - Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

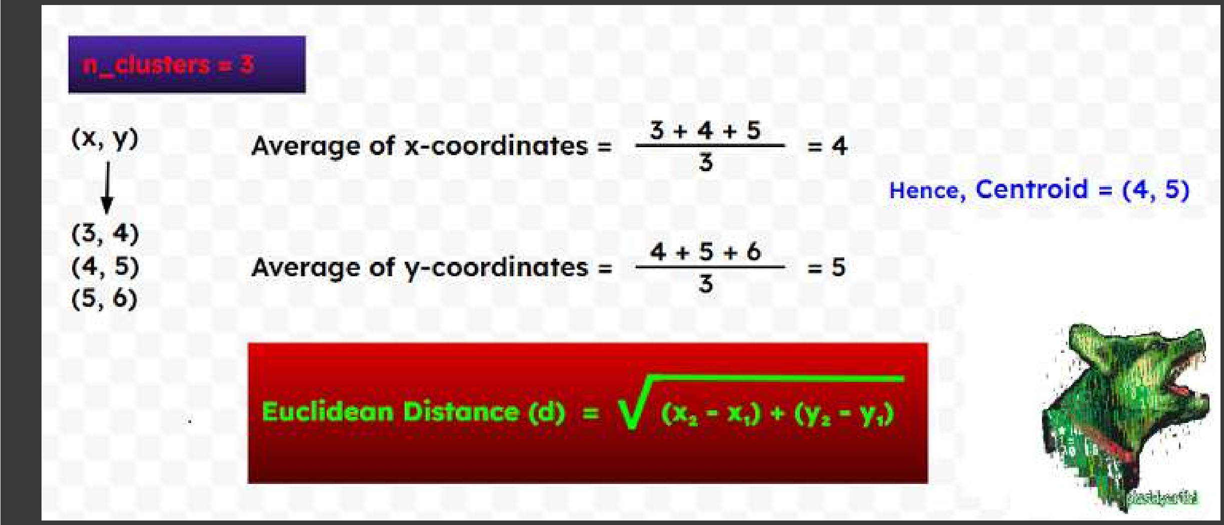
Working of the K-Means Algorithm :

- **Step-1:** Select the number K to decide the number of clusters.
- **Step-2:** Select random K points or centroids. (It can be other from the input dataset).
- **Step-3:** Assign each data point to their closest centroid, which will form the predefined K clusters.
- **Step-4:** Calculate the variance and place a new centroid of each cluster.
- **Step-5:** Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.
- **Step-6:**: If any reassignment occurs, then go to step-4 else go to FINISH.
- **Step-7:** The model is ready.

NOTE ⟹

- Earch centroid of each cluster will check its proximity with all the datapoints by calcutaing **Euclidean Distance**. The ones nearer to the centroid will be converged into that particular cluster.
- **Clustering is an iterative process, it won't stop if we don't define *n_cluster* as a parameterin Python.**
- After the formation of clusters, the next step is to recompute the values of **Centroid.**

Mathematical Example :



n_clusters = 3

(x, y)

(3, 4)
(4, 5)
(5, 6)

Average of x-coordinates = $\frac{3 + 4 + 5}{3}$ = 4

Hence, Centroid = (4, 5)

Average of y-coordinates = $\frac{4 + 5 + 6}{3}$ = 5

Euclidean Distance (d) = $\sqrt{(x_2 - x_1) + (y_2 - y_1)}$

Question :

> centroid1 = (4,3)
>
> entroid2 = (11,6)
>
> centroid3 = (8,10)
>
> Data Point = (7,4)
>
> **Find the nearest centroid to the Datapoint (7,4).**

Solution :

> **dc_1 = √ [(7 - 4)^2 + (4 - 3)^2] = √10**
>
> **dc_2 = √ [(7 - 1)^2 + (4 - 6)^2] = √20**
>
> **dc_3 = √ [(7 - 8)^2 + (4 - 10)^2] = √37**
>
> Hence, **Centroid1 is the nearest centroid to the Datapoint (7,4).**

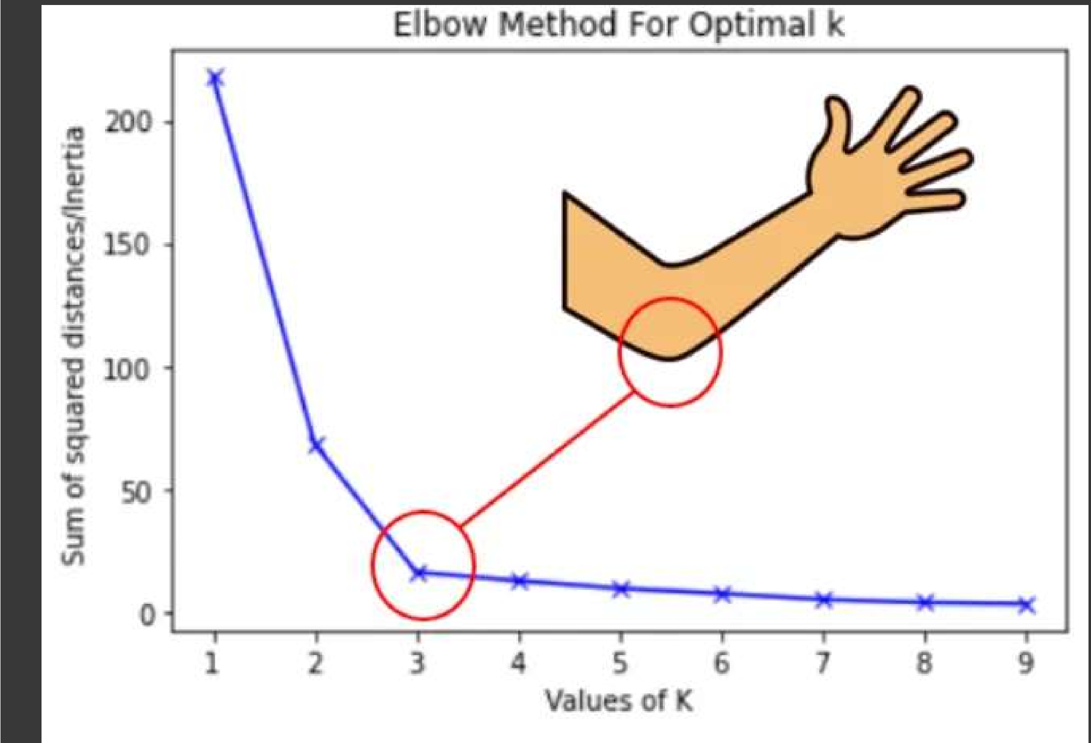
How to choose the optimal value of K ?

1. **Whatever the problem statement (in Hackathon), whatever the client says.**

2. **Elbow Method**
3. **Silhoutte Method**

Elbow Method ⟶

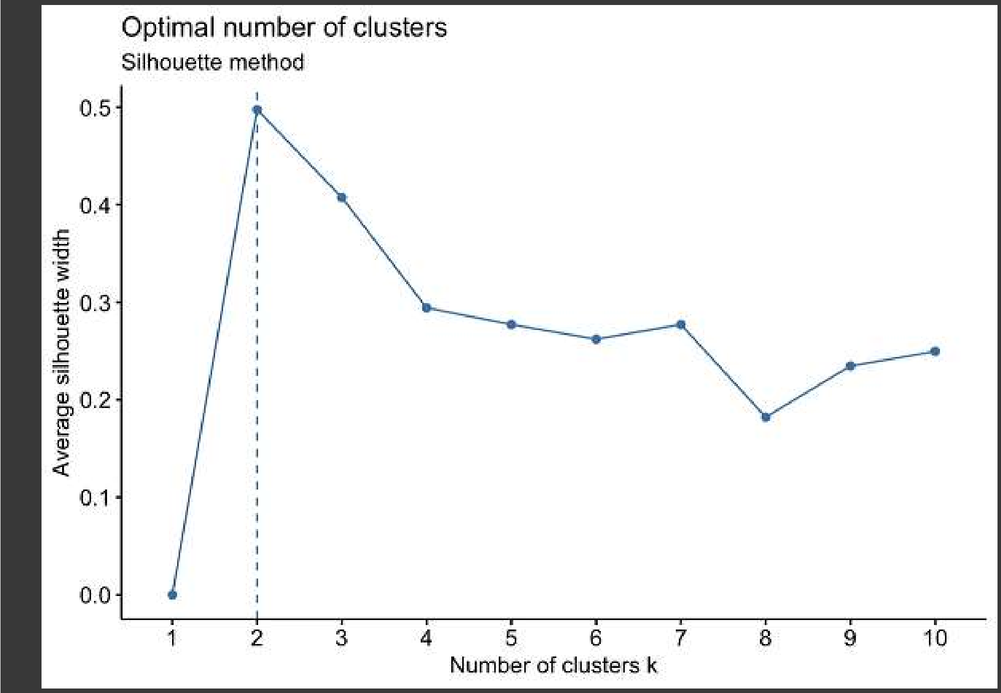
- The elbow method is a graphical representation of finding the optimal 'K' in a K-means clustering. It works by finding WCSS (Within-Cluster Sum of Square) i.e. the sum of the square distance between points in a cluster and the cluster centroid.
- The elbow graph shows WCSS values(on the y-axis) corresponding to the different values of K(on the x-axis). When we see an elbow shape in the graph, we pick the K-value where the elbow gets created. We can call this point the Elbow point. Beyond the Elbow point, increasing the value of 'K' does not lead to a significant reduction in WCSS.
- **So in the majority of the real-world datasets, it is not very clear to identify the right 'K' using the Elbow Method.**



Line plot between K and inertia

Silhouette Method ⟶

- The Silhouette score is a very useful method to find the number of K when the Elbow method doesn't show the Elbow point.
- The value of the Silhouette score ranges from -1 to 1. Following is the interpretation of the Silhouette score.
    - **1 :** Points are perfectly assigned in a cluster and clusters are easily distinguishable.
    - **0 :** Points are perfectly assigned in a cluster and clusters are easily distinguishable.
    - **-1 :** Points are wrongly assigned in a cluster.

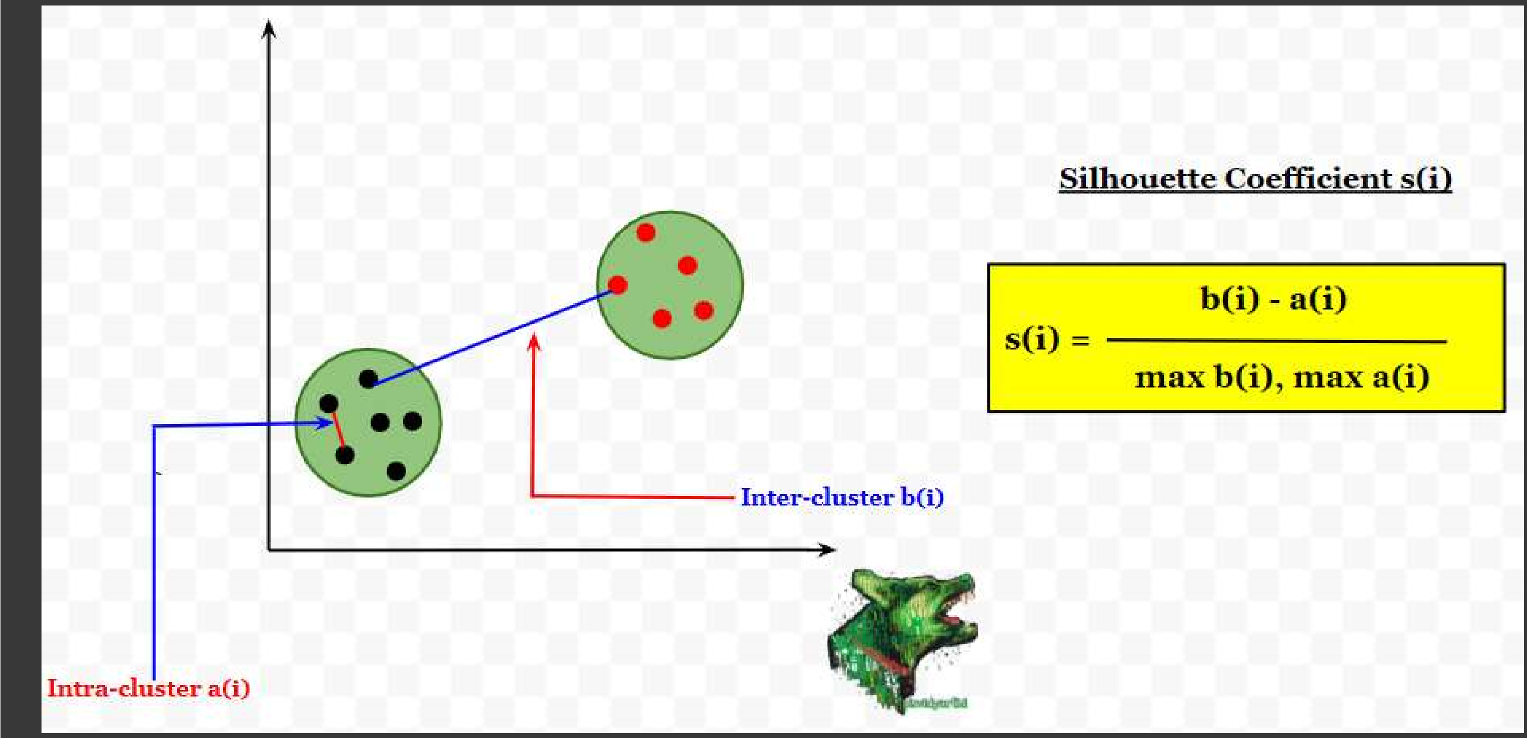


Intercluster and Intracluster :

- In the context of data clustering, **Intercluster distance** refers to the distance between different clusters, while **Intracluster distance** refers to the distance between points within the same cluster.

- **a(i)** ⟶ Average distance of **n** number of points of one cluster and the other cluster.
- **b(i)** ⟶ Average distance of **n** number of data points within one cluster.

s(i) = b(i) - a(i) / max b(i), a(i)



Scalling of Data in K-Means clustering :

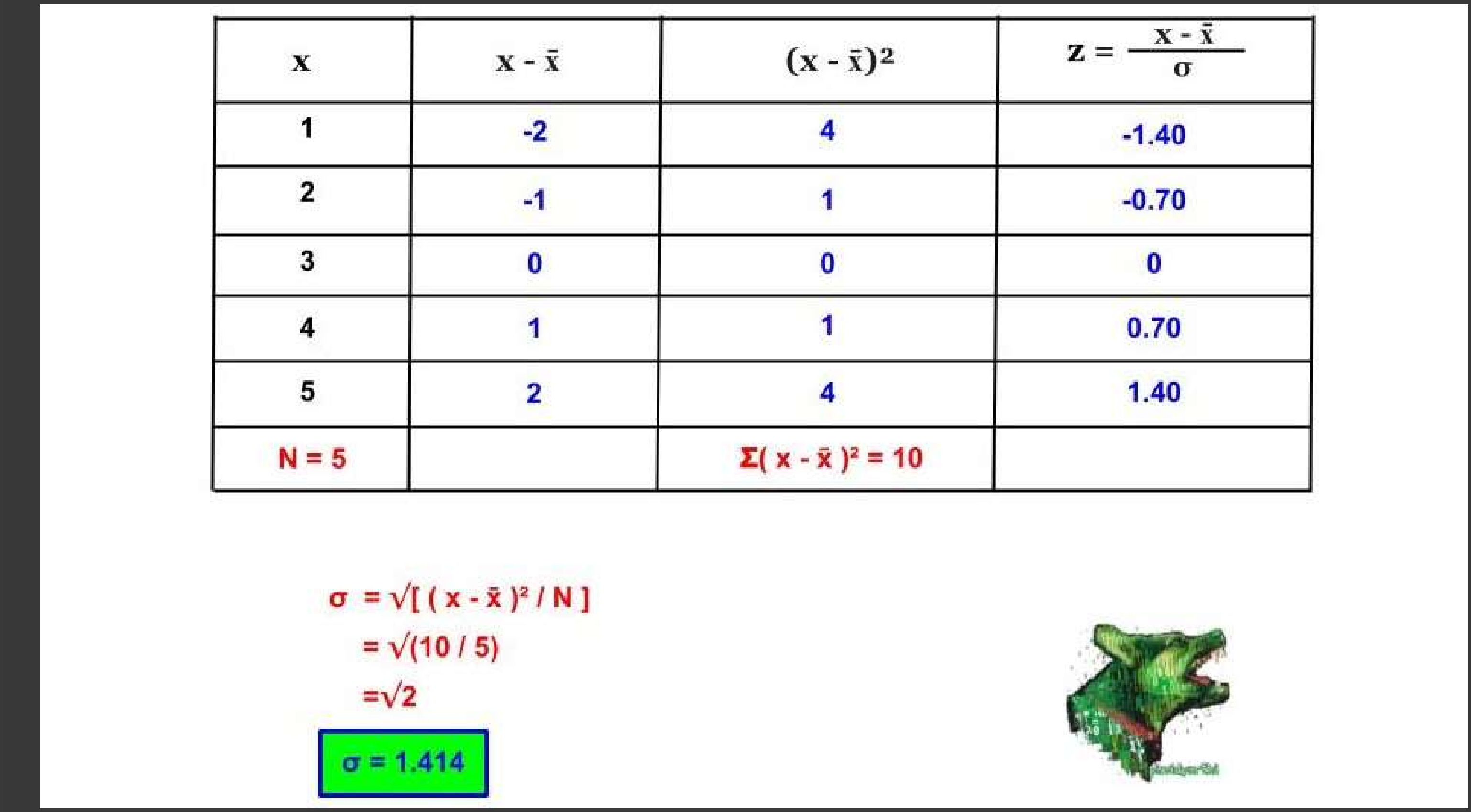
⟹ Scalling K-means to massive data is relatively easy due to its simple iterative nature. Given a set of cluster centers, each point can independently decide which center is closest to it and, given an assignment of points to clustres, computing the optimum center can be done by simply averaging the points.

Normalization :

- Mean = Median ⟶ Normalization
- df ['MaxElement'] / Max (df ['MaxElement'])
- Ranges from 0 to +1.

Standard Scaler :

- from sklearn.preprocessing import StandardScaler ⟶ Calculates Z-Score
- Formula :
  - $Z = (x - \bar{x}) / \sigma$
  - where,
    - $\bar{x}$ → Average (mean) of x
    - σ → Standard Deviation → $\sqrt{[(x - \bar{x})^2 / n]}$



x	$x - \bar{x}$	$(x - \bar{x})^2$	$z = \frac{x - \bar{x}}{\sigma}$
1	-2	4	-1.40
2	-1	1	-0.70
3	0	0	0
4	1	1	0.70
5	2	4	1.40
N = 5		$\Sigma(x - \bar{x})^2 = 10$	

$$\sigma = \sqrt{[(x - \bar{x})^2 / N]}$$
$$= \sqrt{(10 / 5)}$$
$$= \sqrt{2}$$
$$\boxed{\sigma = 1.414}$$

MinMaxScaler :

- from sklearn.preprocessing import MinMaxScaler
- $X({}_{minmaxscaler}) = (X - X_{min}) / (X_{max} - X_{min})$
- Ranges from 0 to 1
- This also normalizes the data.

Inertia [K-means : Inertia] ⟶

Inertia measures how well a dataset was clustered by K-Means. It is calculated by measuring the distance between each data point and its centroid, squaring this distance, and summing these squares across one cluster. A good model is one with low inertia AND a low number of clusters (K).

Python Implementation for K-Means Clustering :

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

Mounting Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount
```

```
df = pd.read_csv('/content/drive/MyDrive/Dataset/Iris.csv')
```

```
df.head()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
df.tail()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

```
df.shape
```

```
(150, 6)
```

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Id	150.0	75.500000	43.445368	1.0	38.25	75.50	112.75	150.0
SepalLengthCm	150.0	5.843333	0.828066	4.3	5.10	5.80	6.40	7.9
SepalWidthCm	150.0	3.054000	0.433594	2.0	2.80	3.00	3.30	4.4
PetalLengthCm	150.0	3.758667	1.764420	1.0	1.60	4.35	5.10	6.9
PetalWidthCm	150.0	1.198667	0.763161	0.1	0.30	1.30	1.80	2.5

Saving Profile Report in HTML format

```
from pandas_profiling import ProfileReport
profile = ProfileReport(df, title = 'Iris Dataset ProfieReport', html = {'style':{'full_width':True}})
profile.to_file(output_file = 'Iris.html')
```

```
Summarize dataset: 100%                                    40/40 [00:04<00:00, 7.61it/s, Completed]
```

Preprocessing ⟶ StandardScaler

- **Calculates Z-Score**
- **Z-Score as an output of all the independent variables**

```
#import StandardScaler
from sklearn.preprocessing import StandardScaler
std = StandardScaler()
df_std = df.drop(['Id', 'Species'], axis = 1)
#Fit Transform
df_std = std.fit_transform(df_std)
df_std
# Array Output => 600 values (150 x 4)
```

```
            [-2.94841818e-01, -5.87763531e-01,  6.49027235e-01,
              1.05353673e+00,  1.18298605e+00],
            [ 2.24968346e+00, -5.87763531e-01,  1.67260991e+00,
              1.05353673e+00, -2.41425724e-02],
            [ 5.53333275e-01, -8.19166497e-01,  6.49027235e-01,
              7.90590793e-01,  1.18298605e+00],
            [ 1.03800476e+00,  5.69251294e-01,  1.10395287e+00,
              1.18500970e+00, -2.41425724e-02],
            [ 1.64384411e+00,  3.37848329e-01,  1.27454998e+00,
              7.90590793e-01, -2.41425724e-02],
            [ 4.32165405e-01, -5.87763531e-01,  5.92161531e-01,
              7.90590793e-01,  1.18298605e+00],
            [ 3.10997534e-01, -1.24957601e-01,  6.49027235e-01,
              7.90590793e-01, -2.41425724e-02],
            [ 6.74501145e-01, -5.87763531e-01,  1.04708716e+00,
              1.18500970e+00, -2.41425724e-02],
            [ 1.64384411e+00, -1.24957601e-01,  1.16081857e+00,
              5.27644853e-01, -2.41425724e-02],
            [ 1.88617985e+00, -5.87763531e-01,  1.33141568e+00,
              9.22063763e-01, -2.41425724e-02],
            [ 2.49201920e+00,  1.72626612e+00,  1.50201279e+00,
              1.05353673e+00, -2.41425724e-02],
            [ 6.74501145e-01, -5.87763531e-01,  1.04708716e+00,
              1.31648267e+00, -2.41425724e-02],
            [ 5.53333275e-01, -5.87763531e-01,  7.62758643e-01,
              3.96171883e-01,  1.18298605e+00],
            [ 3.10997534e-01, -1.05056946e+00,  1.04708716e+00,
              2.64698913e-01,  1.18298605e+00],
            [ 2.24968346e+00, -1.24957601e-01,  1.33141568e+00,
              1.44795564e+00, -2.41425724e-02],
            [ 5.53333275e-01,  8.00654259e-01,  1.04708716e+00,
              1.57942861e+00, -2.41425724e-02],
            [ 6.74501145e-01,  1.06445364e-01,  9.90221459e-01,
              7.90590793e-01, -2.41425724e-02],
            [ 1.89829664e-01, -1.24957601e-01,  5.92161531e-01,
              7.90590793e-01,  1.18298605e+00],
            [ 1.28034050e+00,  1.06445364e-01,  9.33355755e-01,
              1.18500970e+00, -2.41425724e-02],
            [ 1.03800476e+00,  1.06445364e-01,  1.04708716e+00,
              1.57942861e+00, -2.41425724e-02],
            [ 1.28034050e+00,  1.06445364e-01,  7.62758643e-01,
              1.44795564e+00, -2.41425724e-02],
            [-5.25060772e-02, -8.19166497e-01,  7.62758643e-01,
              9.22063763e-01,  1.18298605e+00],
            [ 1.15917263e+00,  3.37848329e-01,  1.21768427e+00,
              1.44795564e+00, -2.41425724e-02],
            [ 1.03800476e+00,  5.69251294e-01,  1.10395287e+00,
              1.71090158e+00, -2.41425724e-02],
            [ 1.03800476e+00, -1.24957601e-01,  8.19624347e-01,
              1.44795564e+00, -2.41425724e-02],
            [ 5.53333275e-01, -1.28197243e+00,  7.05892939e-01,
              9.22063763e-01,  1.18298605e+00],
            [ 7.95669016e-01, -1.24957601e-01,  8.19624347e-01,
              1.05353673e+00, -2.41425724e-02],
            [ 4.32165405e-01,  8.00654259e-01,  9.33355755e-01,
              1.44795564e+00, -2.41425724e-02],
            [ 6.86617933e-02, -1.24957601e-01,  7.62758643e-01,
              7.90590793e-01,  1.18298605e+00]])
```

Fit Transform

K-Means Cluster Algorithm

Import K-Means :

from sklearn.cluster import KMeans

```
#import KMeans
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters= 3)
# after applying KMeans fit df_std
x = kmeans.fit(df_std)
x
```

```
KMeans(n_clusters=3)
```

.labels_ ⟶ Gives us the output of segmentation

```
#.labels_ ⟶ Gives us the output of segmentation
cluster = x.labels_
cluster
```

```
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, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 1,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 1, 1, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 1, 2, 1, 1, 1, 1, 1,
       1, 2, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 2], dtype=int32)
```

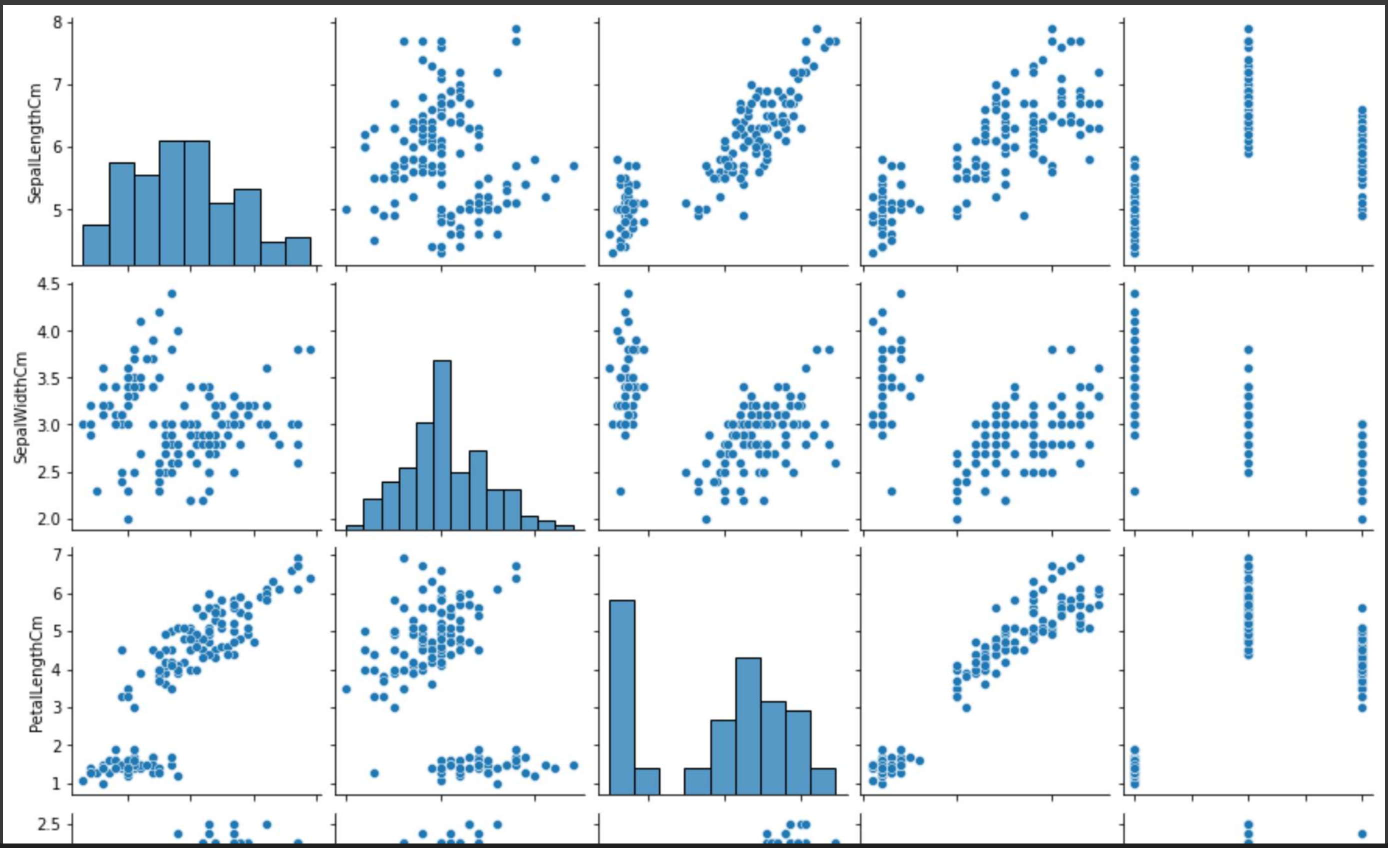
```
df['Cluster'] = cluster
```

```
df.head()
```

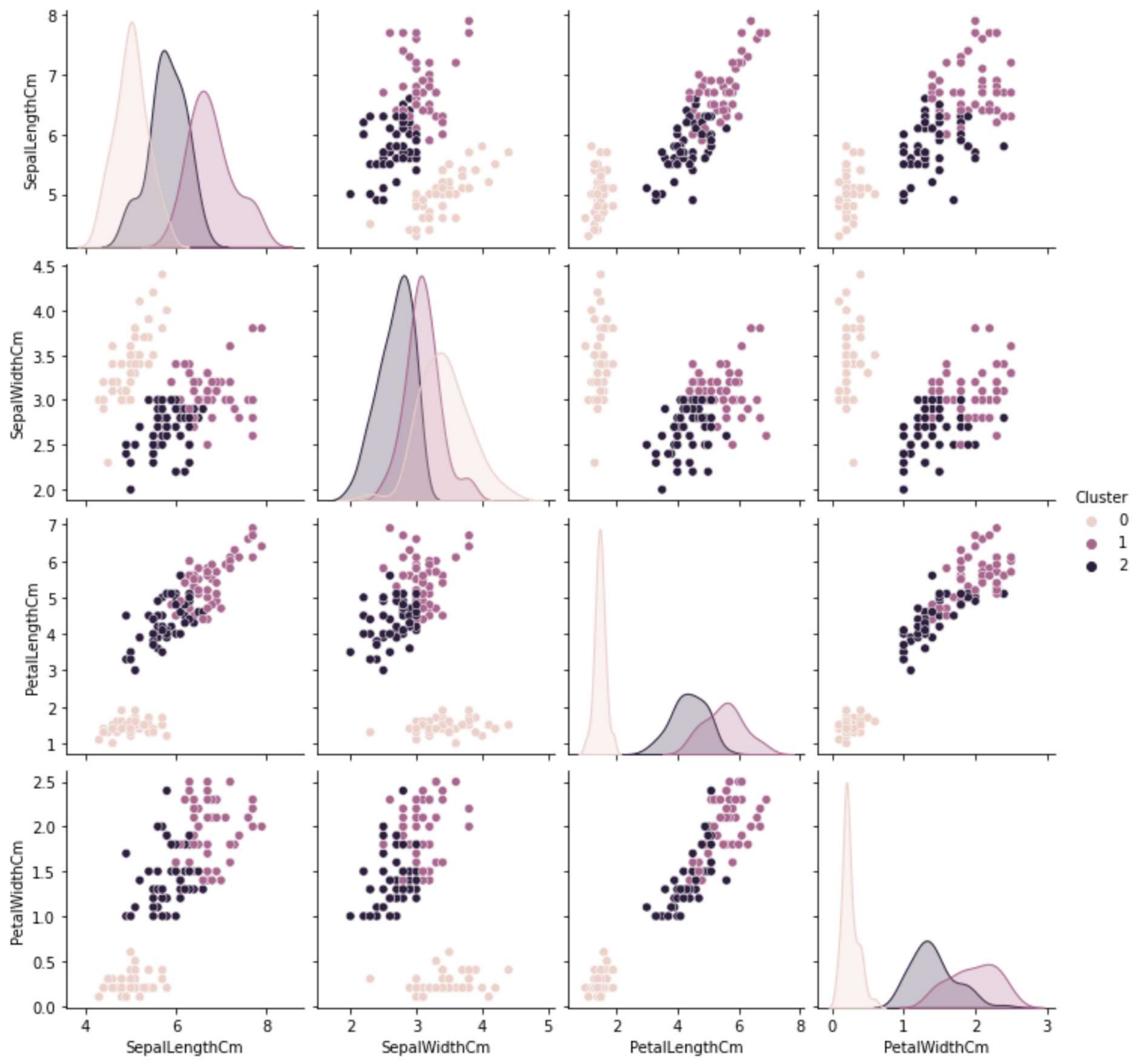
	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	Cluster
0	1	5.1	3.5	1.4	0.2	Iris-setosa	0
1	2	4.9	3.0	1.4	0.2	Iris-setosa	0
2	3	4.7	3.2	1.3	0.2	Iris-setosa	0
3	4	4.6	3.1	1.5	0.2	Iris-setosa	0
4	5	5.0	3.6	1.4	0.2	Iris-setosa	0

```
df_cluster = df.drop(['Id', 'Species'], axis = 1)
```

```
sns.pairplot(df_cluster)
plt.show()
```



```
sns.pairplot(df_cluster, hue = 'Cluster')
plt.show()
```

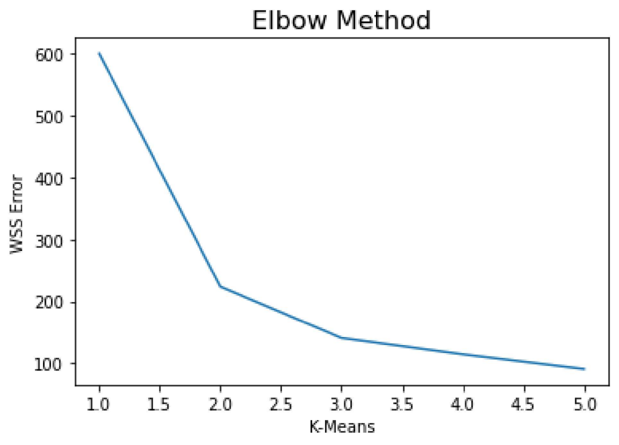


Elbow Method ⟶

```
# Elbow Method
wss = []
cluster_list = list(range(1,6))
for i in cluster_list:
  kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter=300, n_init = 10, random_state= 1)
  kmeans.fit(df_std)
  wss.append(kmeans.inertia_)
```

Plotting Elnow

```
# Plot the Elbow
plt.plot(cluster_list, wss)
plt.title('Elbow Method', fontsize = 16)
plt.xlabel('K-Means')
plt.ylabel('WSS Error')
plt.show()
```



```
# Two Cluster formed
```

```
kmeans1 = KMeans(n_clusters = 2)
kmeans1.fit(df_std)
```

```
KMeans(n_clusters=2)
```

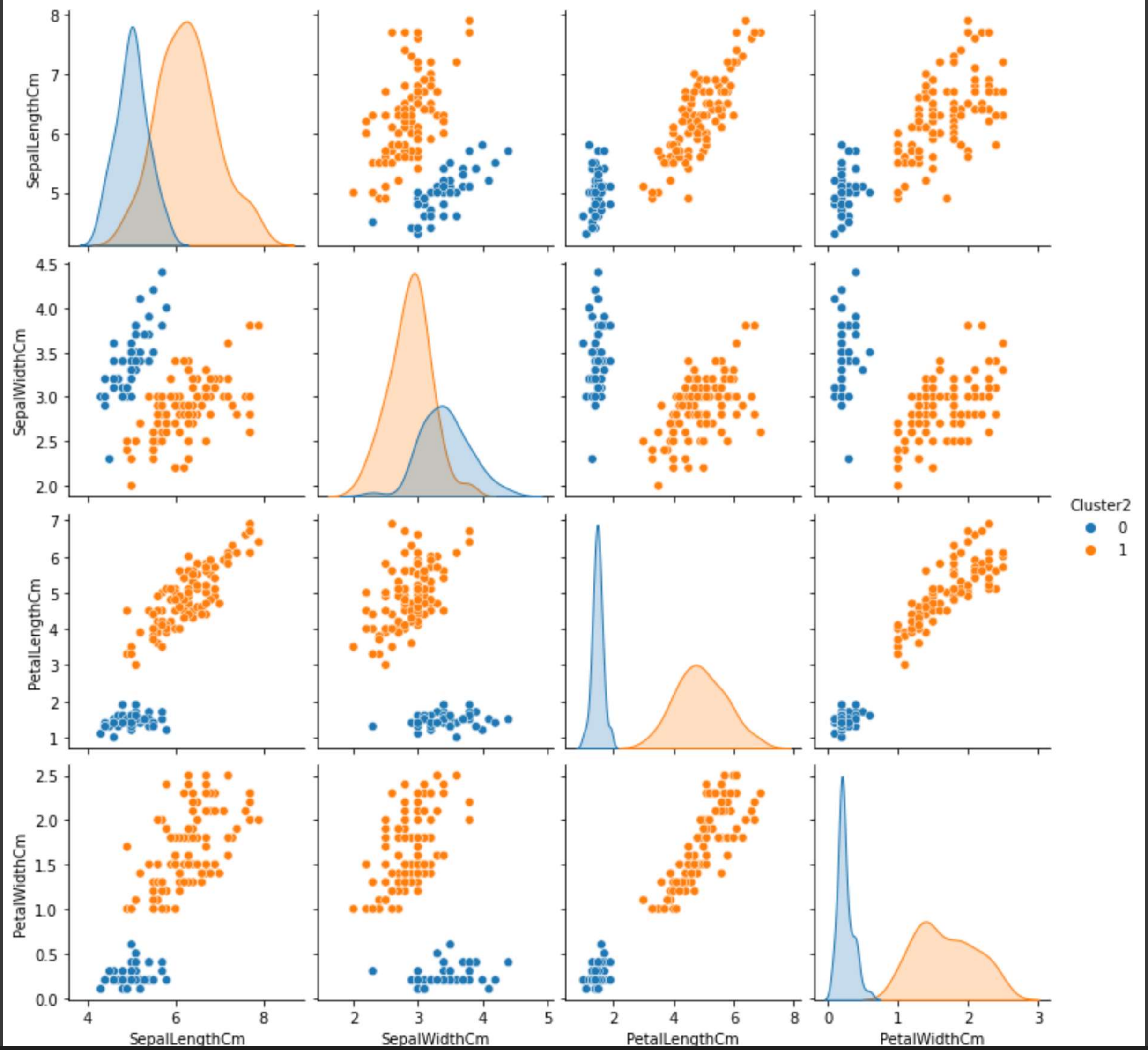
```
cluster2 = kmeans1.labels_
cluster2
```

```
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, 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], dtype=int32)
```

```
df_cluster2 = df.drop(['Id','Species','Cluster'], axis = 1)
df_cluster2['Cluster2'] = cluster2
df_cluster2.head()
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Cluster2
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

```
sns.pairplot(df_cluster2, hue = 'Cluster2')
plt.show()
```



✓ 0s completed at 11:43 PM