METIS

Day 6: Unsupervised Learning

John Navarro
john.navarro@thisismetis.com
https://www.linkedin.com/in/johnnavarro

METIS

```
import pandas as pd
from sklearn.cluster import KMeans,DBSCAN
from sklearn.metrics.cluster import silhouette_score
from sklearn.preprocessing import StandardScaler, Normalizer
import numpy as np
```

METIS

The Dataset

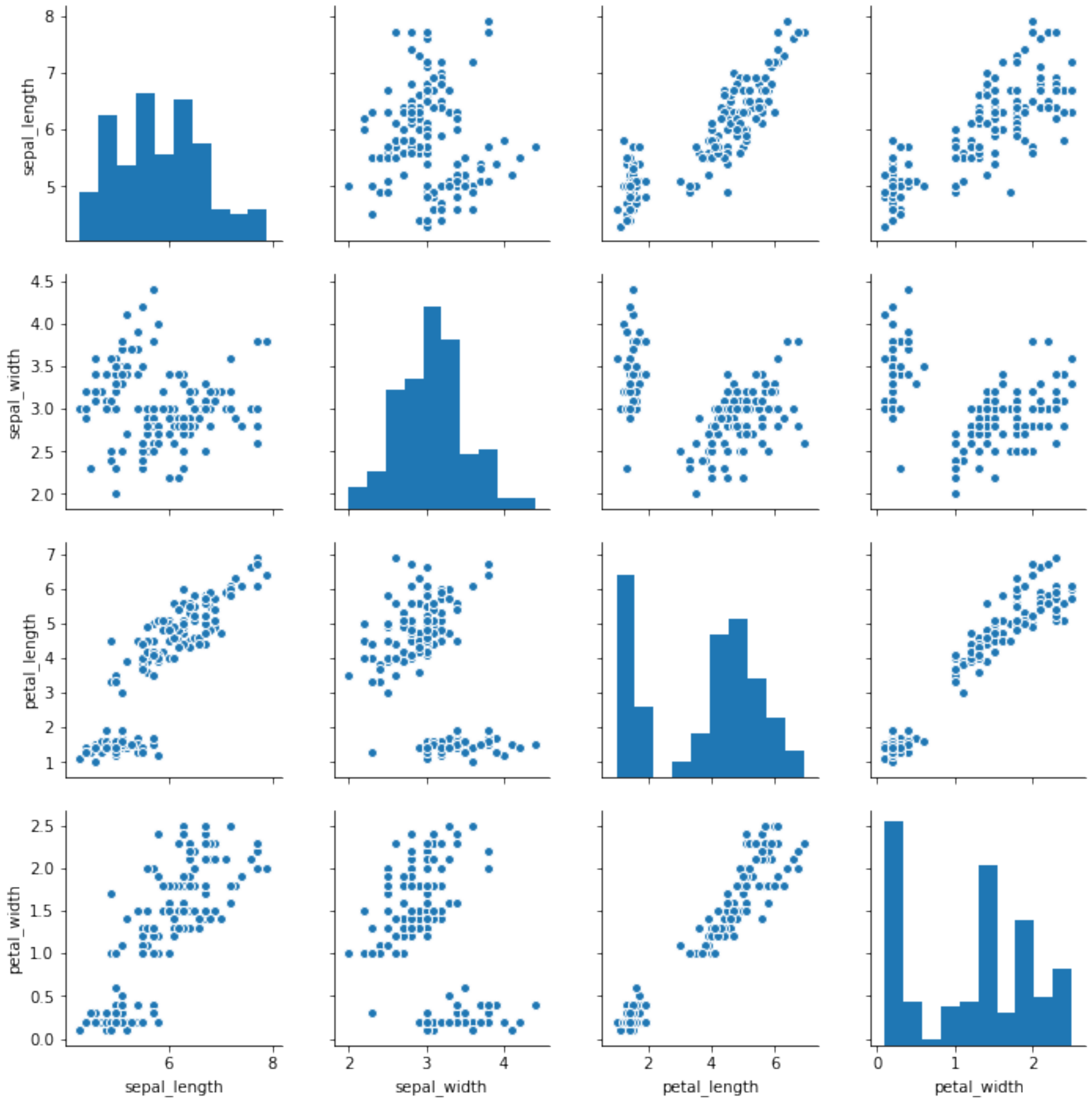
The dataset we will be using today is one of the classic datasets used in machine learning, known as the **iris dataset**. There are 4 features identifying each type of iris:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

METIS

The Dataset

```
sns.pairplot(iris_data)
```



METIS K-Means Clustering

K-means clustering takes a single parameter (k), which is the number of clusters you want the underlying data to fall into, and attempts to find those clusters automatically as follows:

1. Initially generate random cluster centers equal to the number of clusters

2. For each sample (row), label it with the cluster center it is closest to by computing the euclidean distance between it and each cluster center

3. Generate new cluster centers for each cluster based on the labelings for each point.

4. Repeat steps 2-3 until one of the following stopping criteria is met, small fraction of samples change labelings, or cluster centers change position by a very small amount.

METIS

Euclidean Distance

$$d(x, y) = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2}$$

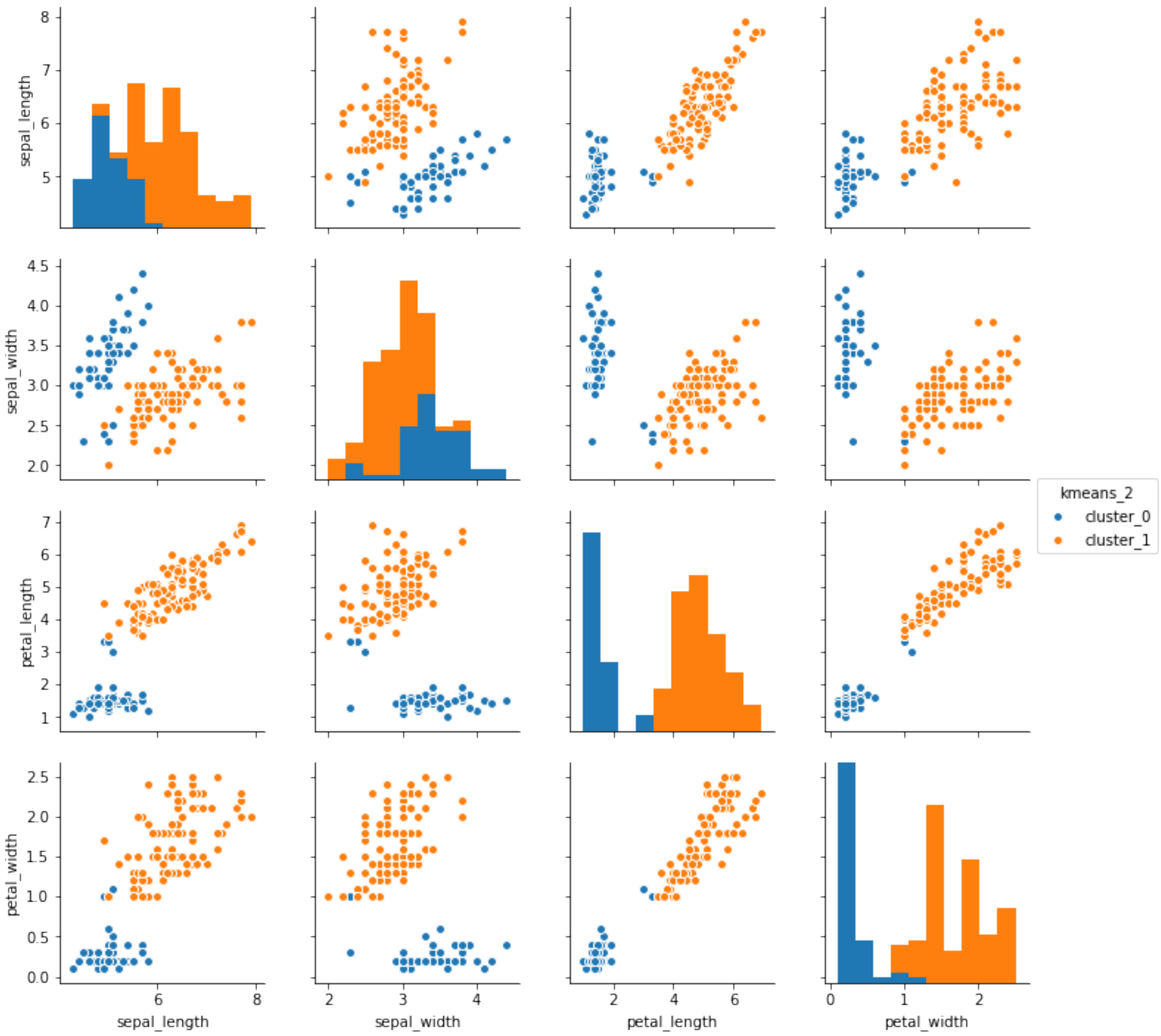
METIS

K-Means Clustering

```
kmeans = KMeans(n_clusters=2,random_state=1234)
kmeans.fit(iris_data_no_names[iris_data_features])
```

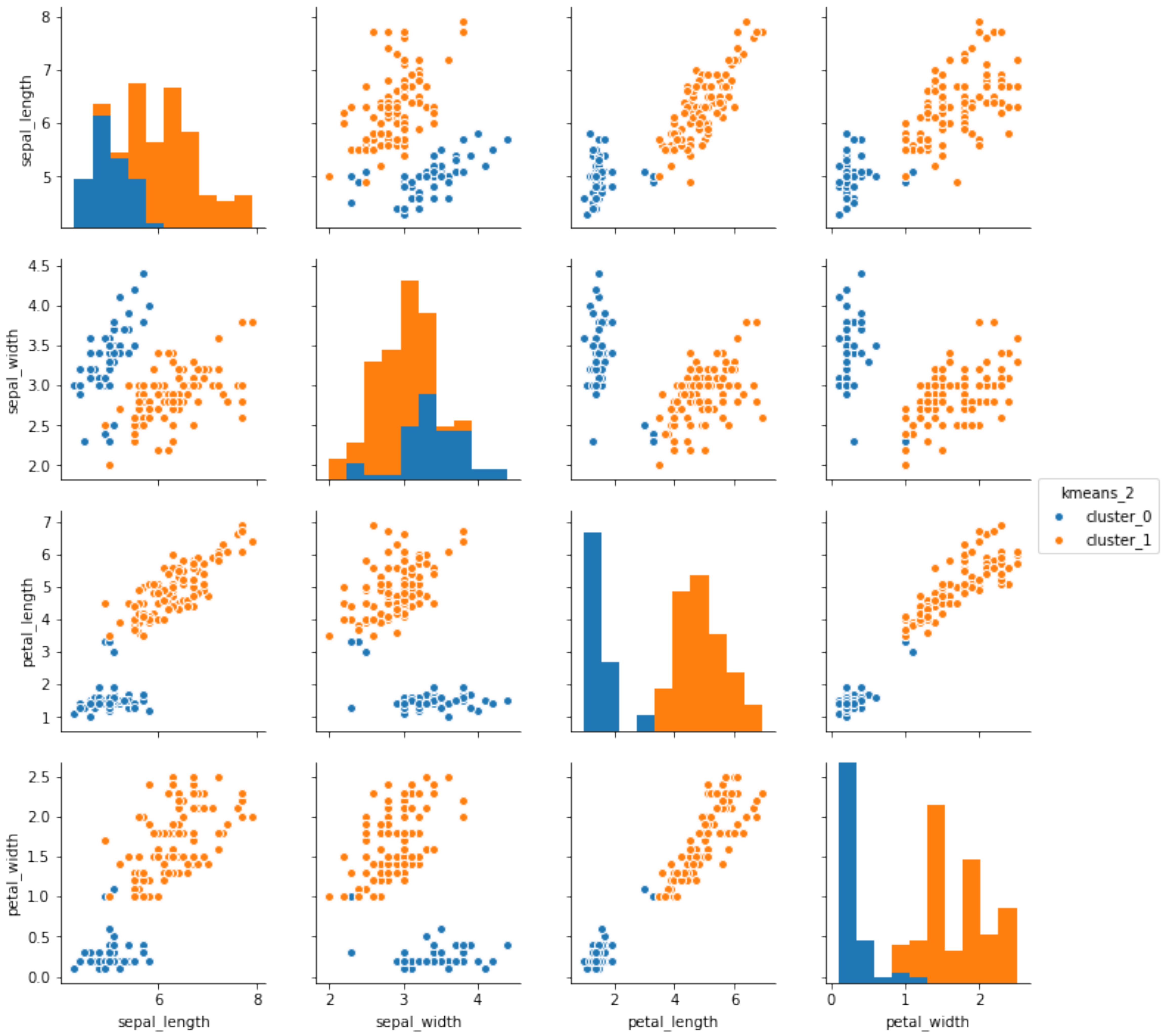
K-Means Clustering

```
sns.pairplot(iris_data_no_names,hue="kmeans_2")
```



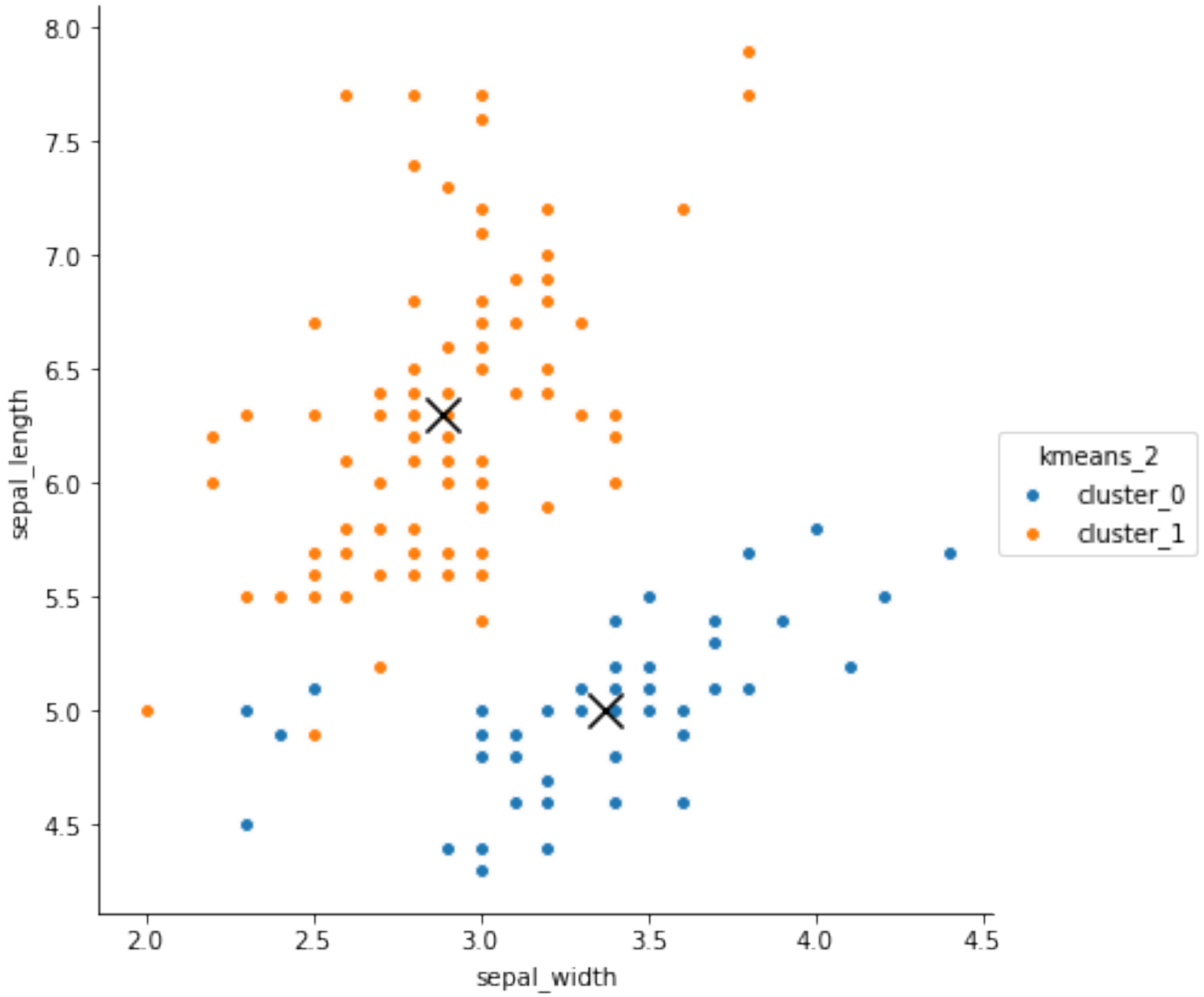
K-Means Clustering

```
sns.pairplot(iris_data_no_names,hue="kmeans_2")
```



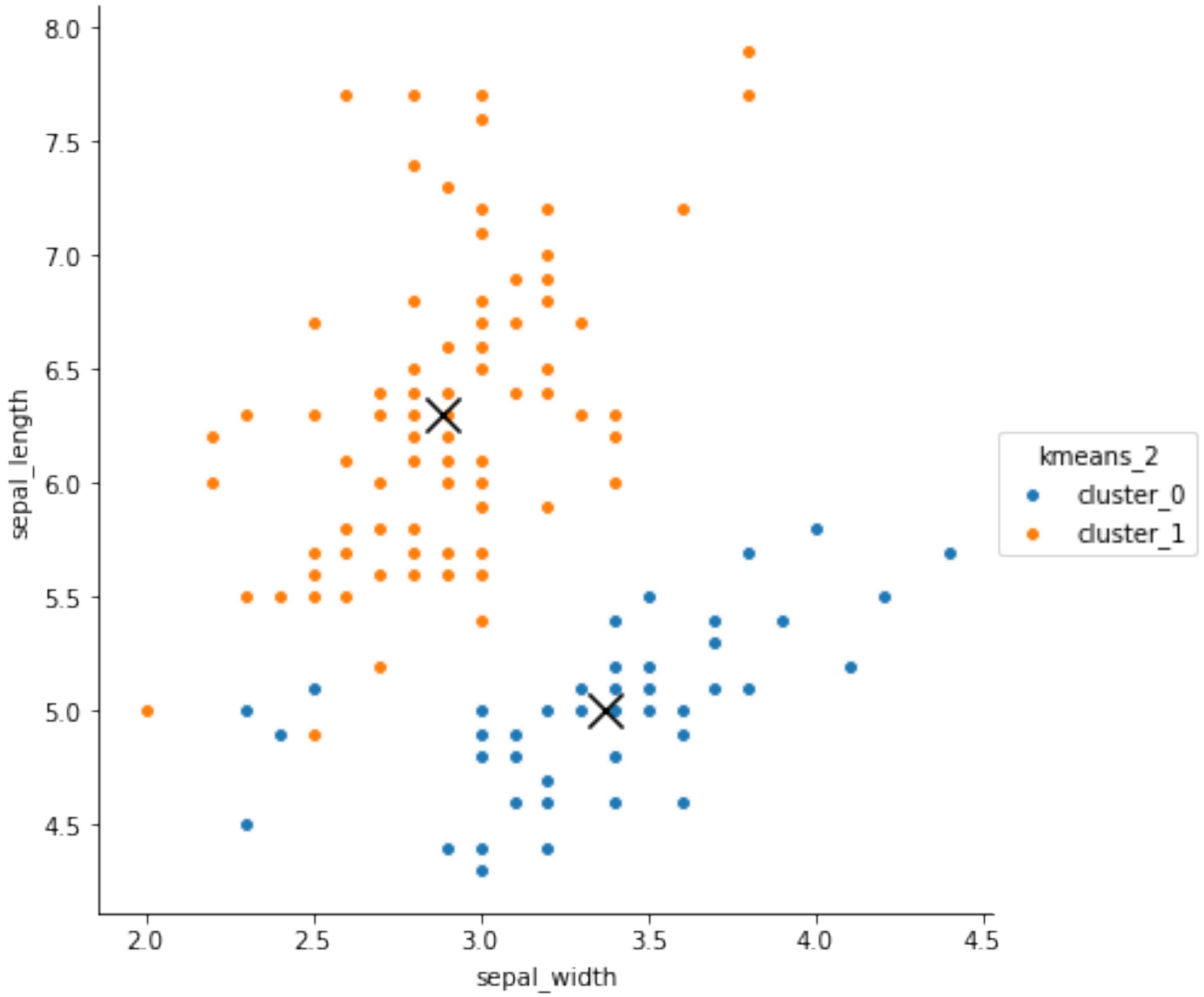
METIS K-Means Clustering

```
sns.pairplot(iris_data_no_names,x_vars="sepal_width",y_vars="sepal_length",hue="k
means_2",size=6)
plt.scatter(iris_2_cluster_centers.sepal_width,
iris_2_cluster_centers.sepal_length, linewidths=3, marker='x', s=200, c='black')
```



METIS K-Means Clustering

```
sns.pairplot(iris_data_no_names,x_vars="sepal_width",y_vars="sepal_length",hue="k
means_2",size=6)
plt.scatter(iris_2_cluster_centers.sepal_width,
iris_2_cluster_centers.sepal_length, linewidths=3, marker='x', s=200, c='black')
```



METIS

K-Means is affected by the scale of every feature.

Feature Scaling

For k-means clustering, features must be scaled to the same ranges of values to contribute "equally" to the euclidean distance calculation.

Each row is transformed per-column by:

- Subtracting from the element in each row the mean for each feature (column) and then taking this value and

- Dividing by that feature's (column's) standard deviation.

METIS

Feature Scaling

```
# center and scale the data
scaler = StandardScaler()

iris_data_scaled =
scaler.fit_transform(iris_data_no_names[iris_data_features])

iris_data_scaled =
pd.DataFrame(iris_data_scaled,columns=iris_data_features)
```

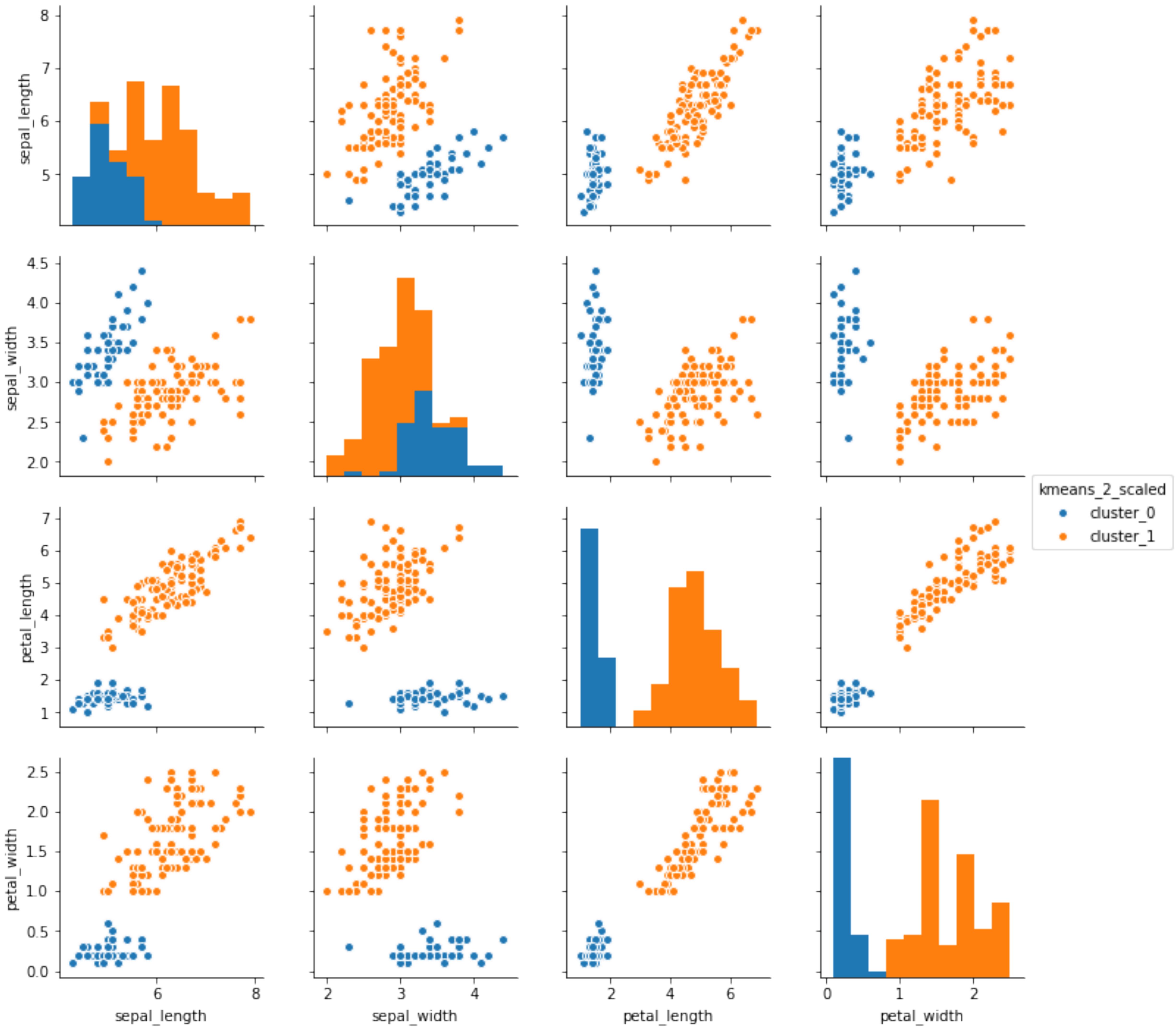
Feature Scaling

```
# K-means on scaled data
km = KMeans(n_clusters=2,random_state=1234)
km.fit(iris_data_scaled)
```

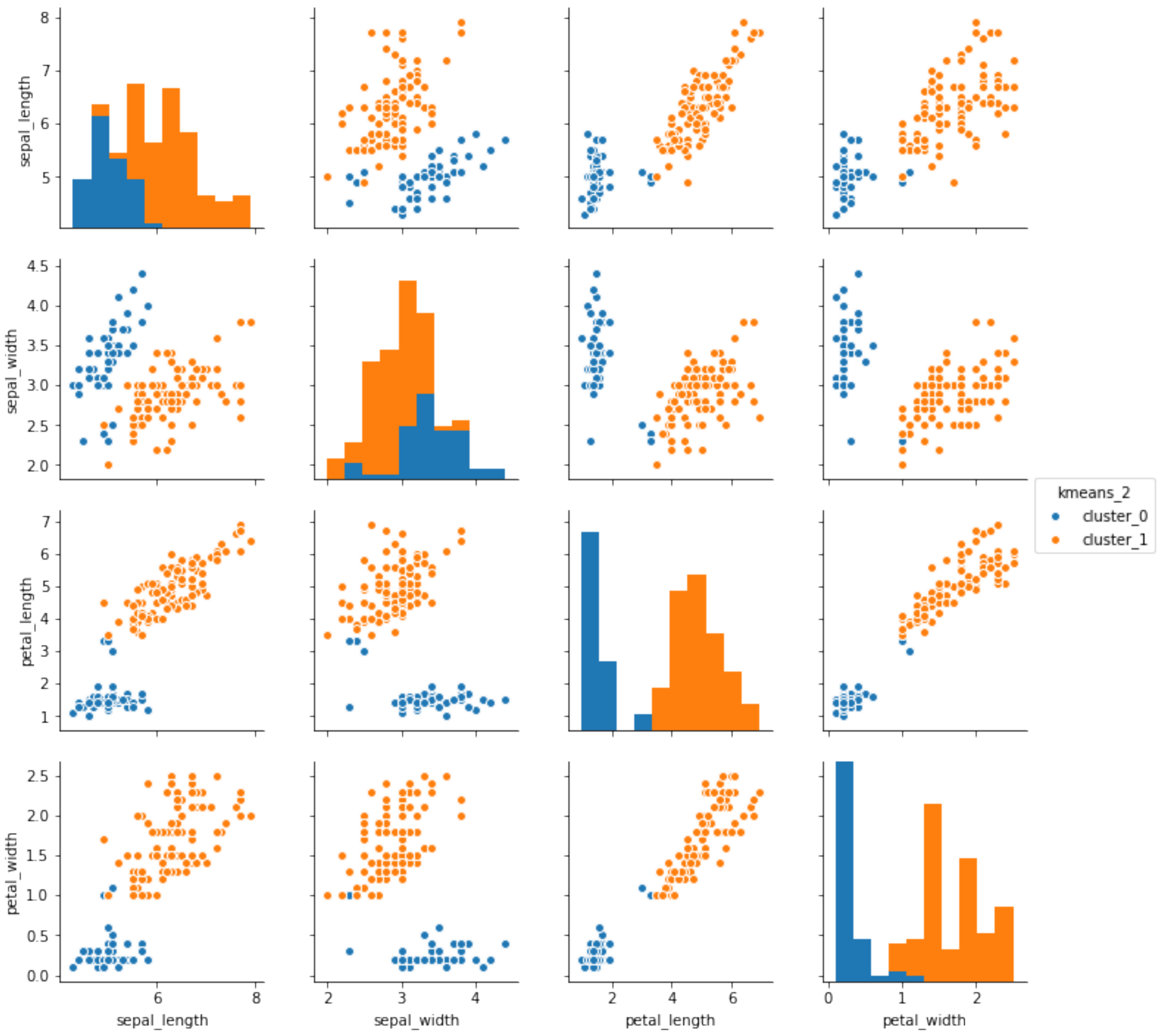
METIS

Feature Scaling

Scaled Features

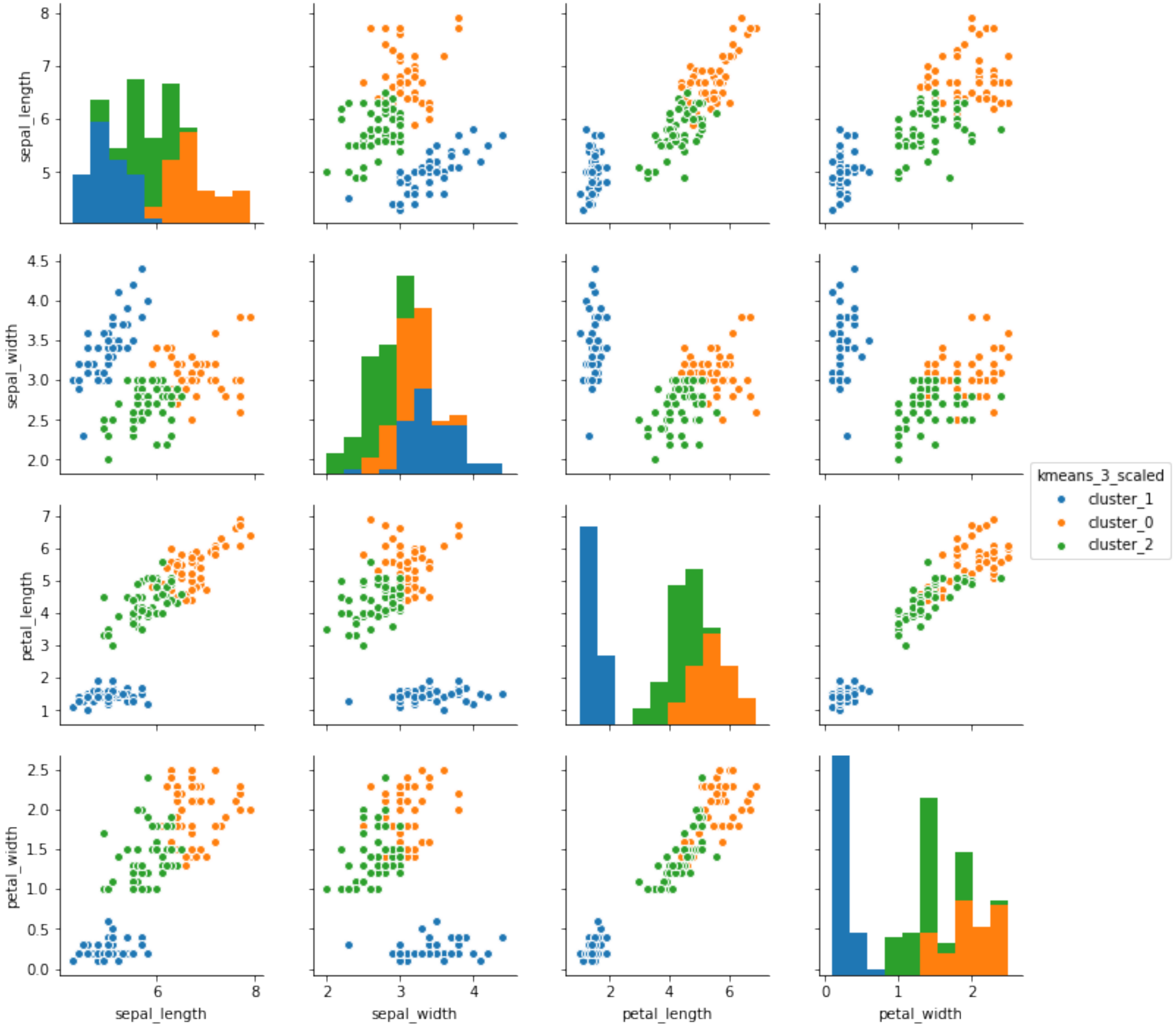


Unscaled Features



More Clusters

```
km3 = KMeans(n_clusters=3,random_state=1234)
km3.fit(iris_data_scaled)
```



METIS

Exercise

- Generate k-means clustering for 4, 5, and 6 clusters.
- How many samples are there per cluster for each clustering type?
- How do you decide which number of clusters is best?

METIS

Evaluating your Model

- Generate k-means clustering for 4, 5, and 6 clusters.
- How many samples are there per cluster for each clustering type?
- How do you decide which number of clusters is best?

METIS

Evaluating your Model

The Silhouette Coefficient is a common metric for evaluating clustering "performance" in situations when the "true" cluster assignments are not known.

b = mean distance to next nearest cluster

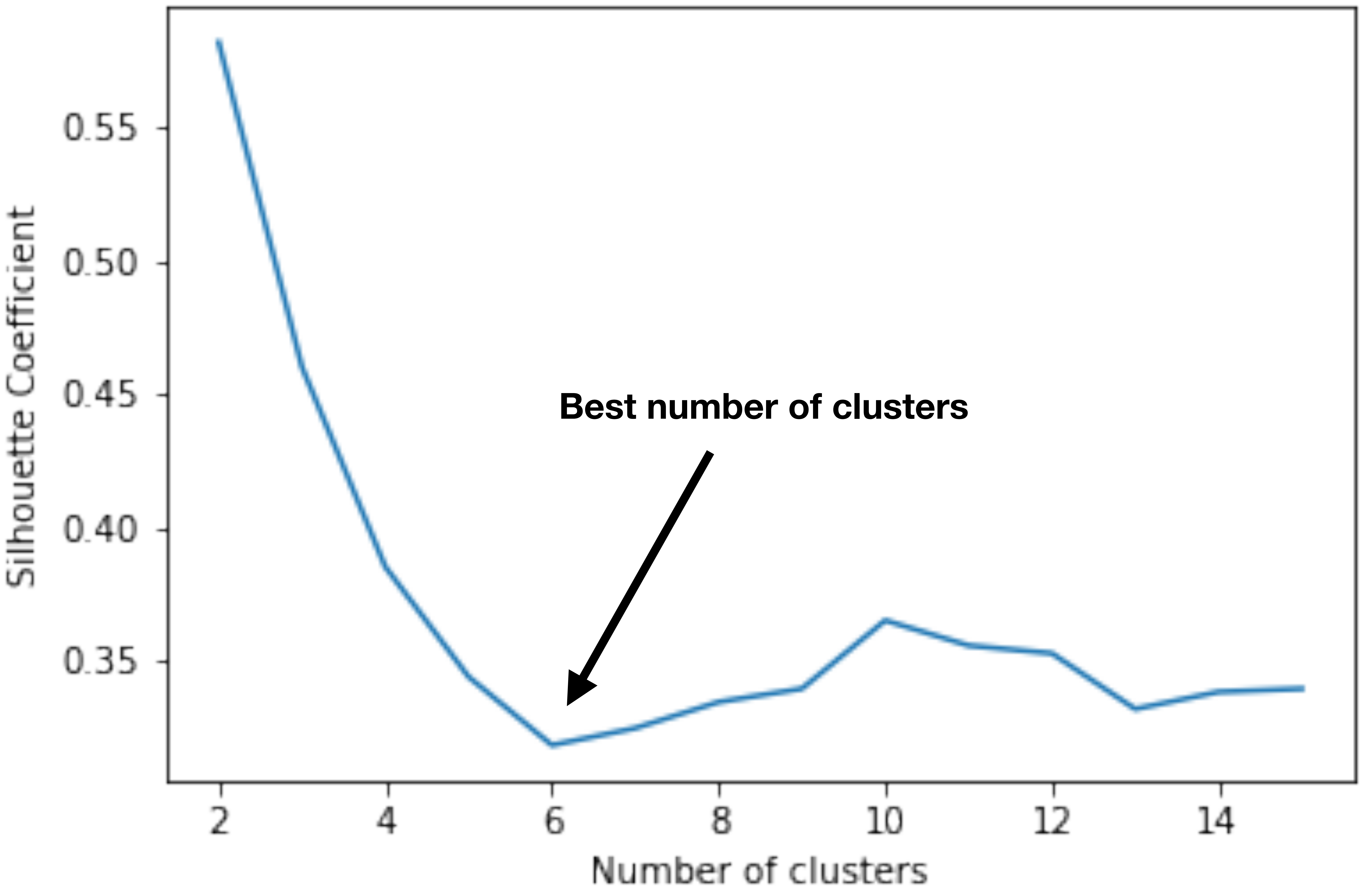
a = mean distance to other points in cluster

```
silhouette_coeff = (b - a) / max(a,b)
```

Evaluating your Model

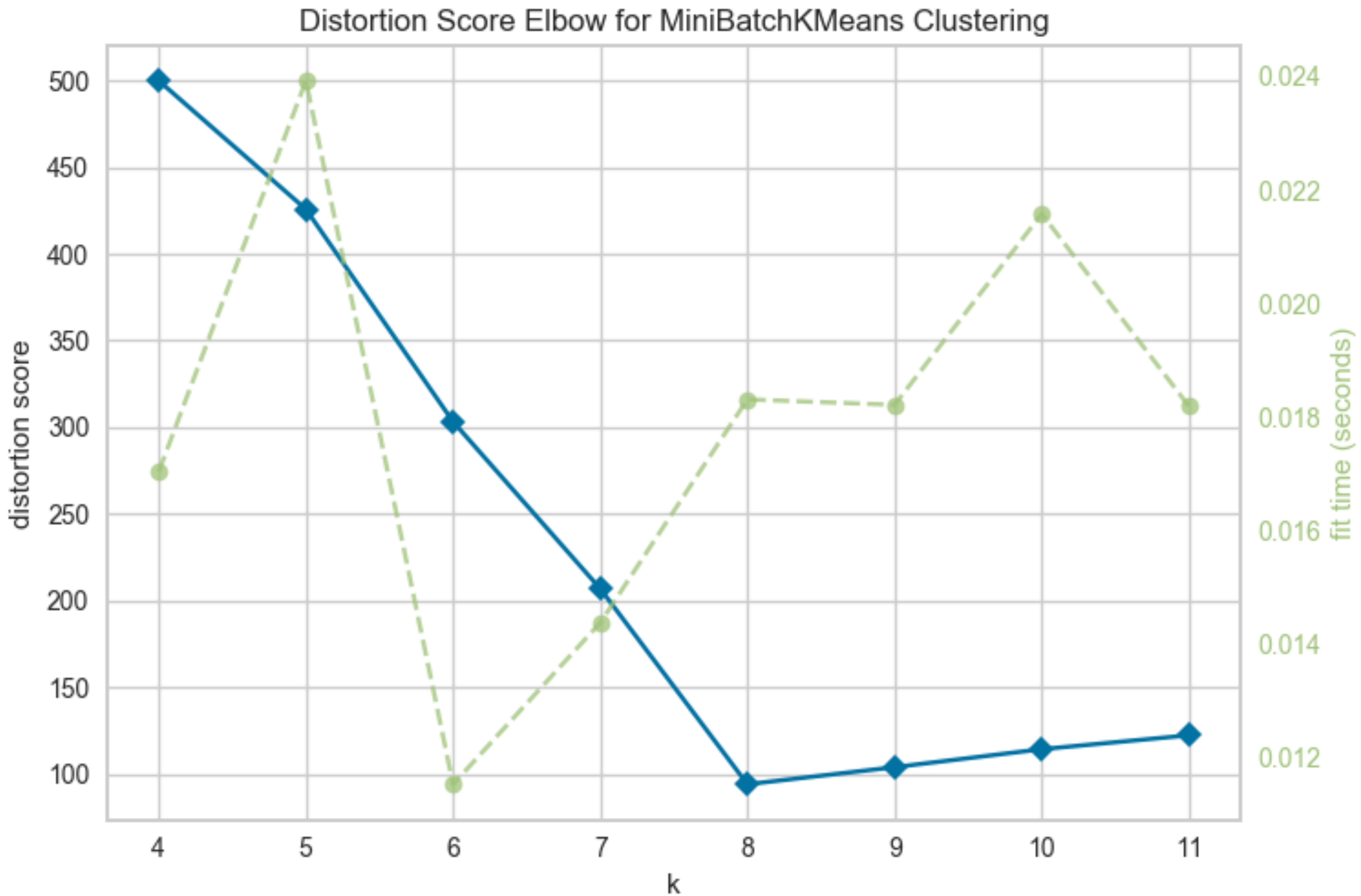
```
k_range = range(2,16)
scores = []
for k in k_range:
    km_ss = KMeans(n_clusters=k, random_state=1)
    km_ss.fit(iris_data_scaled)
    scores.append(silhouette_score(iris_data_scaled,
km_ss.labels_))
```

Evaluating your Model



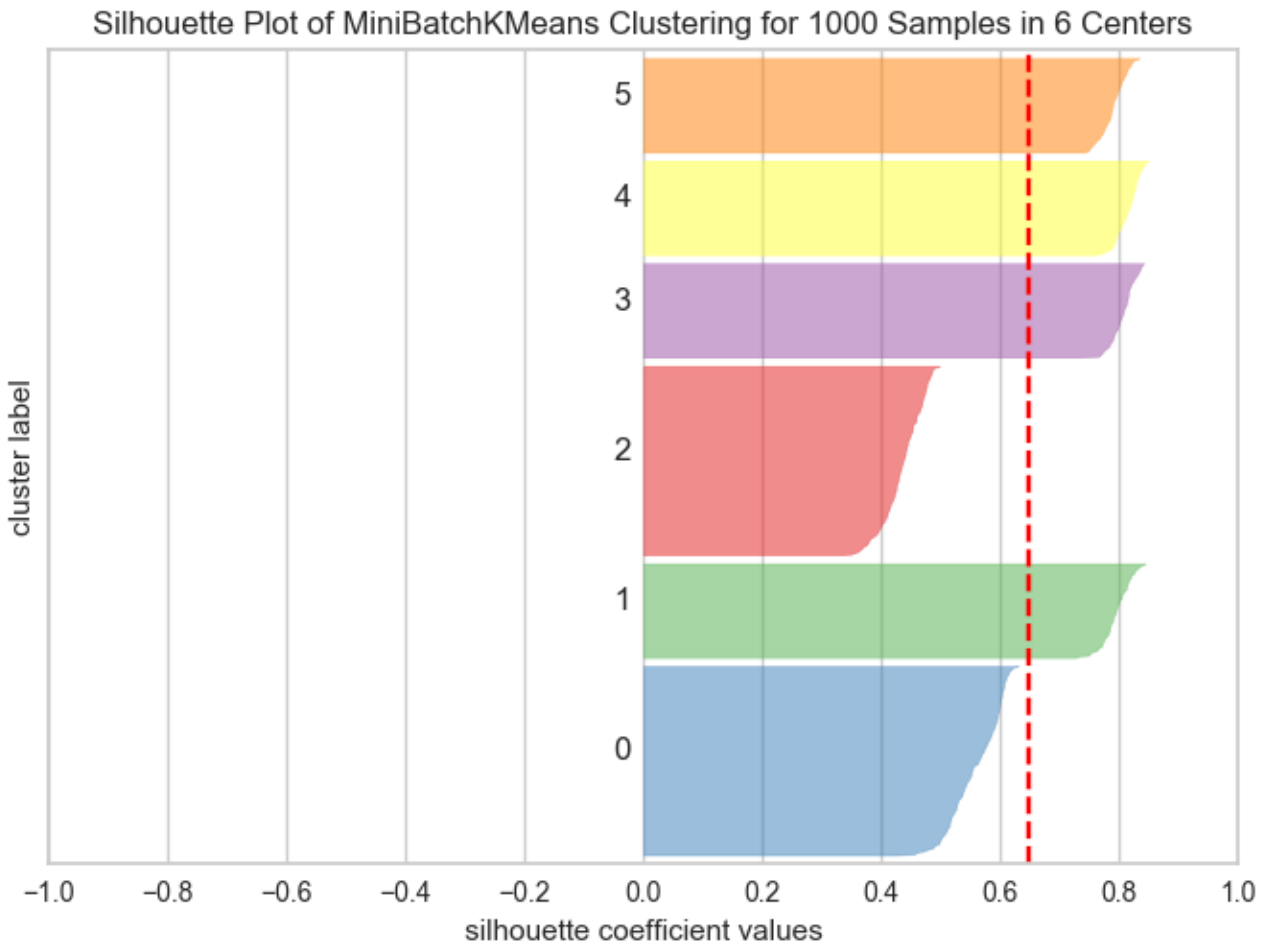
Evaluating your Model

```
from yellowbrick.cluster import KElbowVisualizer
visualizer = KElbowVisualizer(KMeans(), k=(4,12))

visualizer.fit(X)
visualizer.poof()
```



Evaluating your Model

```
from yellowbrick.cluster import SilhouetteVisualizer
model = MiniBatchKMeans(6)
visualizer = SilhouetteVisualizer(model)

visualizer.fit(X)
visualizer.poof()
```



METIS

Exercise

You have the following seeds dataset. Each row in the dataset is an individual seed. The individual columns are as follows:

- seed area
- seed perimeter
- compactness
- length of kernel
- width of kernel
- asymmetry coefficient
- length of kernel groove

In the data the labs have been removed so that you can explore the data yourself.

Please do the following:

- Perform clustering using a variety of cluster sizes
- Calculate the silhouette score for each cluster size and determine an optimal cluster number
- Visualize the clustering and compute statistics on those clusters. What distinguishes each cluster you've created?

METIS

DBSCAN

DBSCAN stands for **D**ensity-**B**ased **S**patial **C**lustering of **A**pplications with **N**oise.

Whereas K-means does not care about the density of data, DBSCAN does, under the assumption that regions of high density in your data should be treated as clusters.

DBSCAN

DBSCAN does not allow you to specify how many clusters you want.
Instead, you specify 2 parameters:

- **ϵ (epsilon)**: This is the maximum distance between two points to allow them to be neighbors

- **min_samples**: The number of neighbors a given point is allowed to have to be able to be part of a cluster

Any points that don't satisfy the criteria of being close enough to other points are labeled outliers and all fall into a single "cluster" (their cluster label by default is -1).

METIS

DBSCAN

DBSCAN works as follows:

1. Choose an arbitrary starting point in your dataset that has not been seen.

2. Retrieve this point's ϵ-neighborhood (all points that are within a distance ϵ from it), and if it contains at least ***min_samples**, a cluster is started.

3. Otherwise, the point is labeled as an outlier (-1). Note: This point might later be found in a sufficiently sized ε-environment of a different point and hence be made part of a cluster.

4. If a point is found to be a dense part of a cluster, its ϵ-neighborhood is also part of that cluster. All points that are found within the ϵ-neighborhood are added, as is their own ϵ-neighborhood when they are also dense.

5. Continue until the density-connected cluster is completely found.

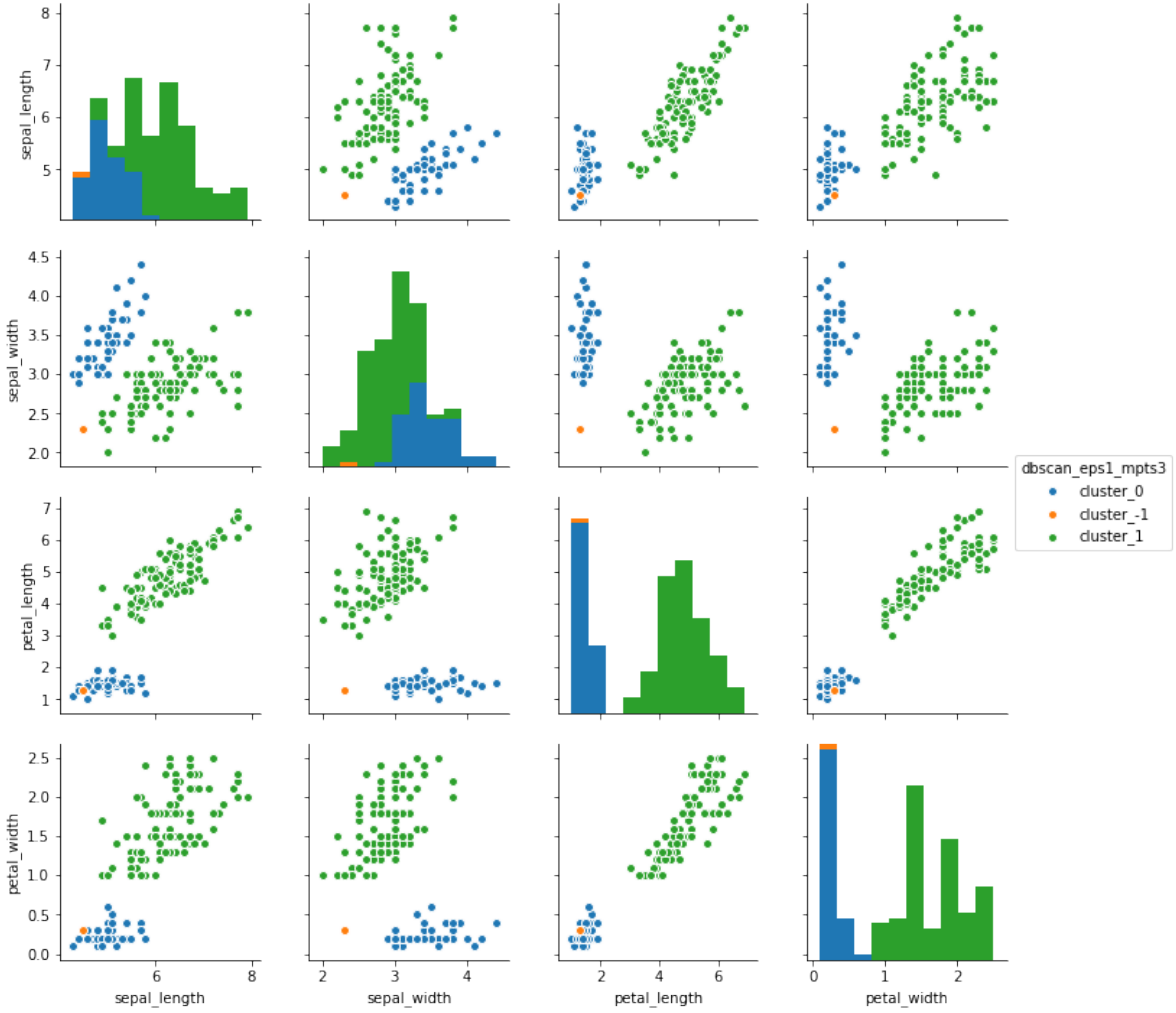
6. Find a new unvisited point to process, rinse and repeat.

DBSCAN

```
db = DBSCAN(eps=1, min_samples=3)
db.fit(iris_data_scaled)
```

DBSCAN

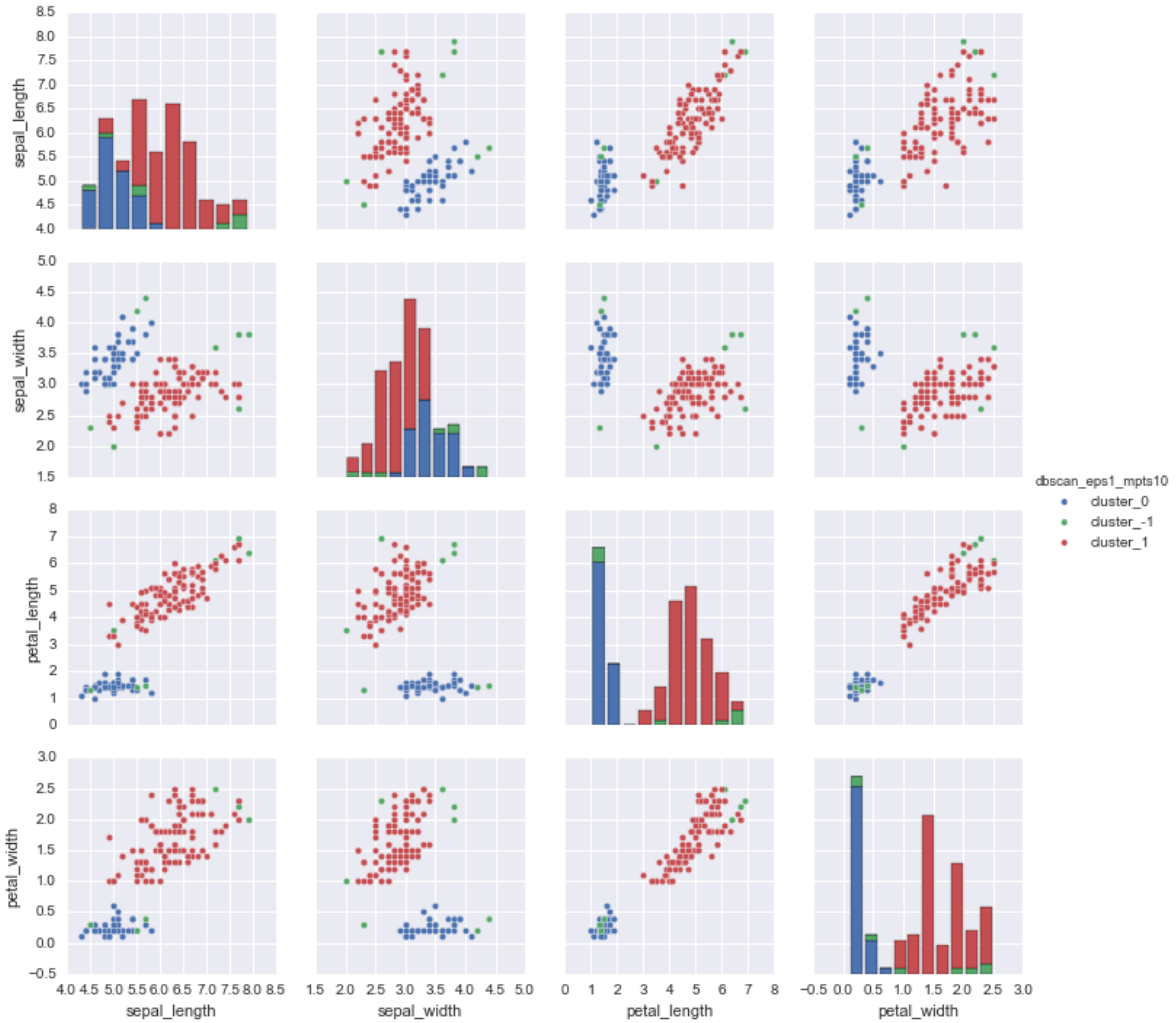
```
iris_data_no_names['dbscan_eps1_mpts3'] = [ "cluster_" + str(label) for label in db.labels_ ]
sns.pairplot(iris_data_no_names,hue="dbscan_eps1_mpts3")
```



METIS

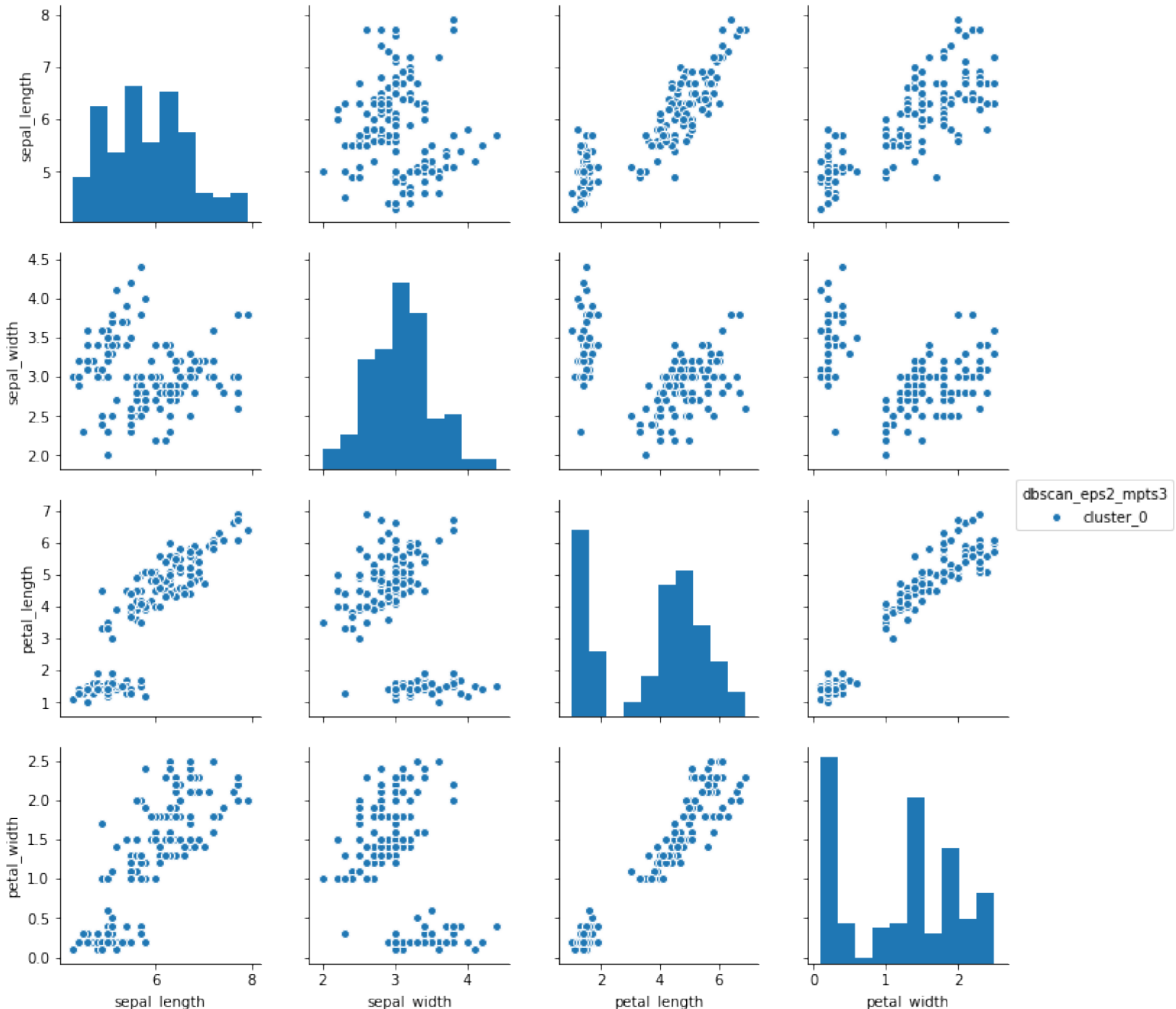
DBSCAN

```
db2 = DBSCAN(eps=1, min_samples=10)
db2.fit(iris_data_scaled)
```



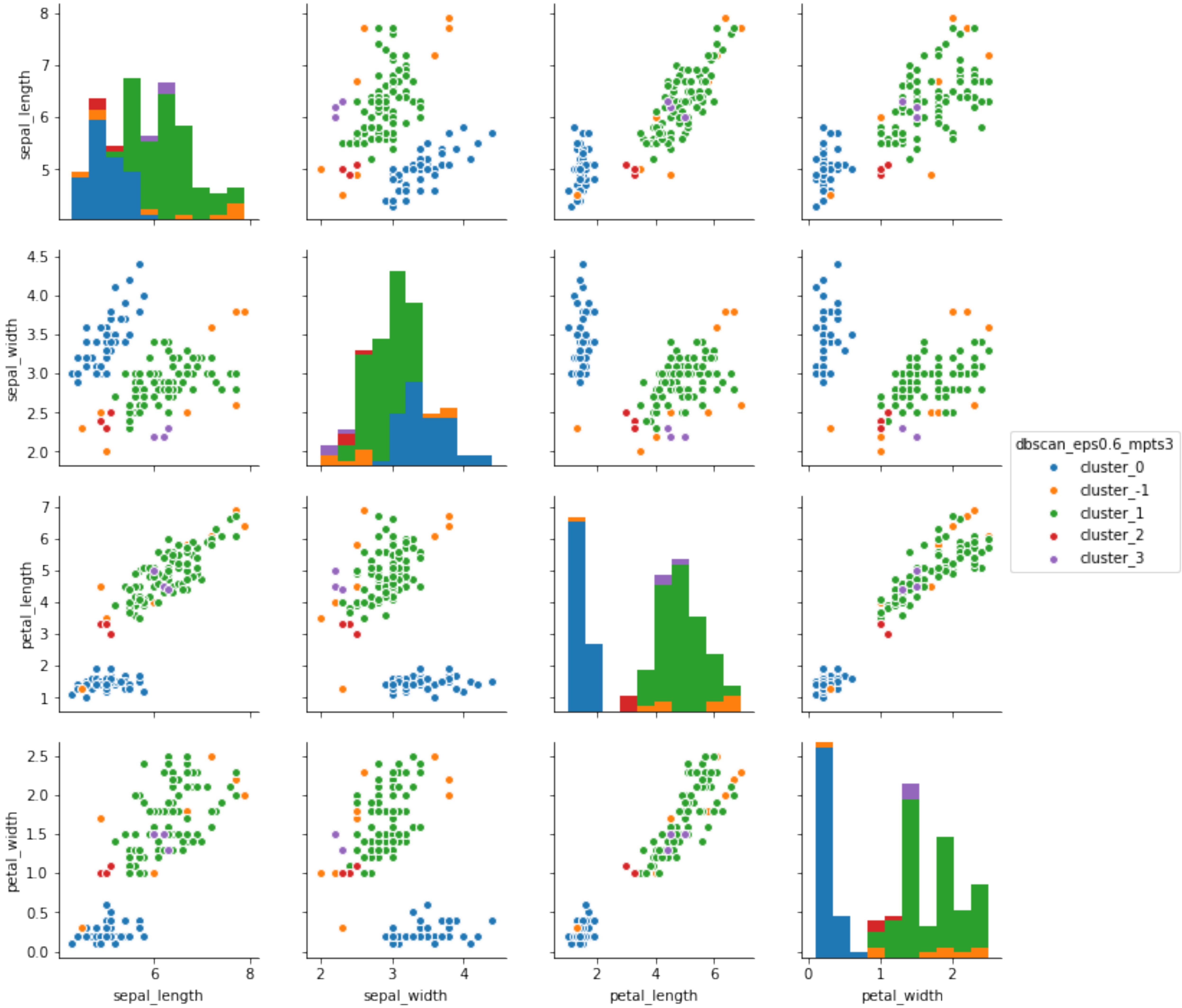
DBSCAN

```
db2 = DBSCAN(eps=2, min_samples=3)
db2.fit(iris_data_scaled)
```



DBSCAN

```
db2 = DBSCAN(eps=0.6, min_samples=3)
db2.fit(iris_data_scaled)
```



METIS

Exercise

Using the seeds dataset we looked at above, please do the following:

- Perform clustering using a variety of ϵ and **min_samples** values

- Calculate the silhouette score for each group of parameters and determine an optimal configuration

- Visualize the clustering and compute statistics on those clusters. What distinguishes each cluster you've created?