The Iris Dataset: DBSCAN Algorithm

1. Importing necessary libraries

First, we need to import the libraries required for our analysis: `pandas` for data manipulation, `numpy` for numerical operations, `matplotlib` for data visualisation, and `sklearn` for implementing DBSCAN (Density-Based Spatial Clustering of Applications with Noise) on a sample dataset.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
```

2. Loading the dataset

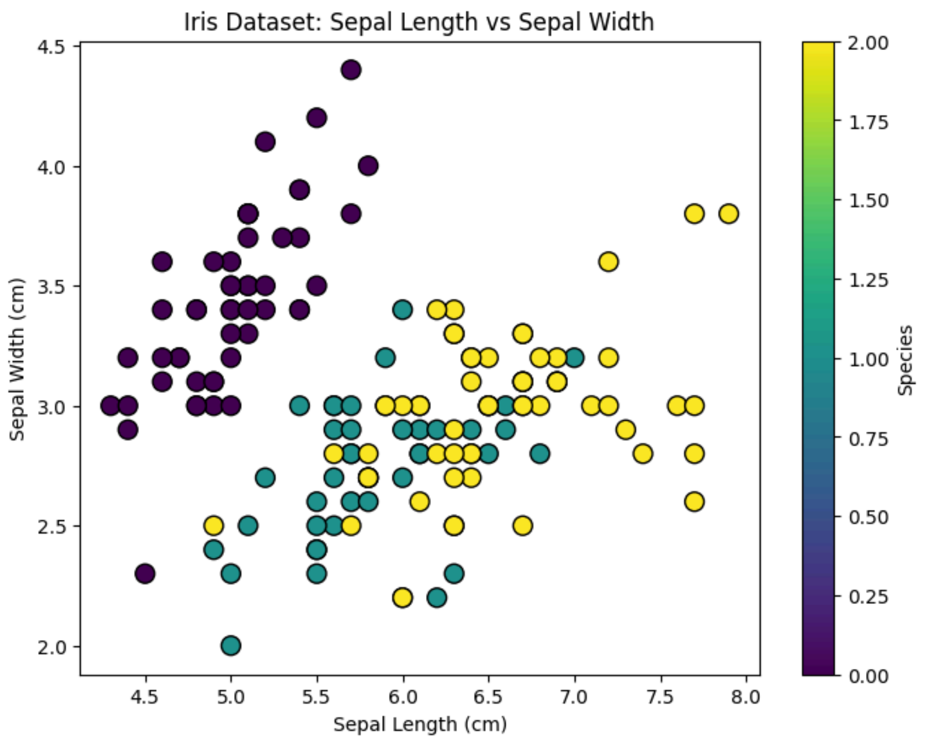
Here, we load the Iris dataset from Scikit-learn. The dataset consists of 150 samples of iris flowers, each described by four features: sepal length, sepal width, petal length, and petal width. The target variable `y` contains the species labels, but we will use `X` (the features) for clustering.

```
iris = load_iris()
X = iris.data
y = iris.target
```

3. Visualising the original dataset

We visualise the original dataset by plotting the first two features: sepal length and sepal width. The points are coloured based on the species labels, allowing us to see the distribution of different species in the feature space.

```
plt.figure(figsize=(8,6))
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis', edgecolor='k', s=100)
plt.title('Iris Dataset: Sepal Length vs Sepal Width')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.colorbar(label='Species')
plt.show()
```



4. Applying the DBSCAN algorithm

Now, we apply the DBSCAN clustering algorithm to the dataset. We define `eps` (the maximum distance to consider two points as neighbours) as 0.5 and `min_samples` (the minimum number of points required to form a dense region) as 5.

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)
```

5. Evaluating clustering performance

Next, we evaluate the clustering performance using the silhouette score, which measures how similar an object is to its own cluster compared to other clusters.

```
if len(set(labels)) > 1:    # Excluding noise-only labels
  score = silhouette_score(X, labels)
  print(f'Silhouette score: {score:.2f}')
else:
  print('No clusters found')
```

```
Silhouette score: 0.49
```

6. Visualising the clustering results

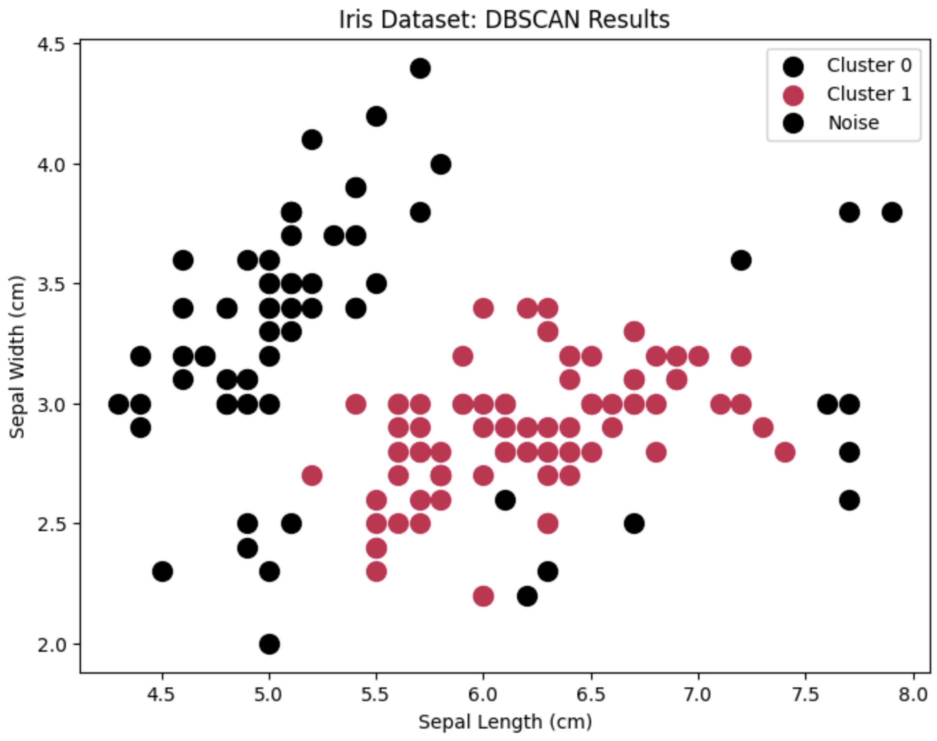
Finally, we visualise the clustering results obtained from DBSCAN with a scatterplot. Each cluster is represented in a different colour, and noise points (if any) are labeled distinctly.

```
plt.figure(figsize=(8,6))
unique_labels = set(labels)
colors = plt.cm.get_cmap('inferno', len(unique_labels))

for k in unique_labels:
    class_member_mask = (labels == k)
    plt.scatter(X[class_member_mask, 0], X[class_member_mask, 1],
                color=colors(k), label=f'Cluster {k}' if k != -1 else 'Noise', s=100)

plt.title('Iris Dataset: DBSCAN Results')
plt.xlabel('Sepal Length (cm)')
plt.ylabel('Sepal Width (cm)')
plt.legend()
plt.show()
```

```
<ipython-input-14-e572e8abaed2>:3: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in
  colors = plt.cm.get_cmap('inferno', len(unique_labels))
```


Iris Dataset: DBSCAN Results
● Cluster 0
● Cluster 1
● Noise
Sepal Width (cm)
Sepal Length (cm)