Implementation of DBSCAN Algorithm on a Simple Dataset

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 simple dataset.

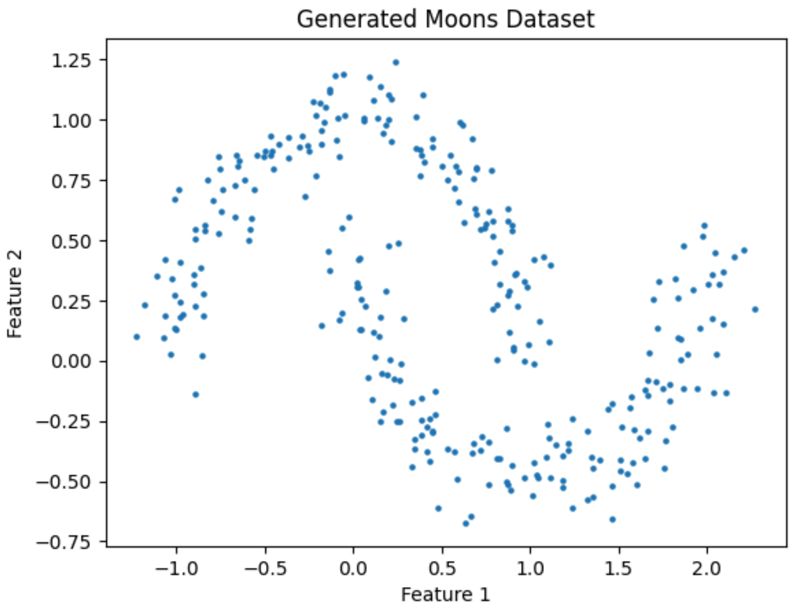
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
```

2. Defining the dataset

Here, we use a synthetic dataset created with the `make_moons` function, which generates two interleaving half-circles. We also visualise the dataset with a scatterplot, making it easier to observe clustering patterns.

```
X, _ = make_moons(n_samples=300, noise=0.1, random_state=42)

plt.scatter(X[:, 0], X[:, 1], s=5)
plt.title('Generated Moons Dataset')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



3. Implementing the DBSCAN algorithm

Now, we define a function to apply DBSCAN to the dataset. The DBSCAN algorithm identifies clusters based on the density of points in a region.

Parameters:

1. `eps` : The maximum distance between two samples for them to be considered as in the same neighbourhood.
2. `min_samples` : The number of samples (or total weight) in a neighbourhood for a point to be considered a core point.

```
def apply_dbscan(X, eps, min_samples):
  dbscan = DBSCAN(eps=eps, min_samples=min_samples)
  labels = dbscan.fit_predict(X)
  return labels
```

4. Experimenting with different parameters

Next, we evaluate the performance of the DBSCAN algorithm by testing various combinations of the parameters `eps` and `min_samples` . The goal is to determine which combination yields the best clustering results based on the silhouette score. This score quantifies how well-separated the clusters are, with higher values indicating better-defined clusters.

```
eps_values = [0.1, 0.2, 0.3, 0.4]
min_samples_values = [2, 5, 10]

results = []

for eps in eps_values:
    for min_samples in min_samples_values:
        labels = apply_dbscan(X, eps, min_samples)

        if len(set(labels)) > 1:    # Excluding noise-only labels
            score = silhouette_score(X, labels)
            results.append((eps, min_samples, score))

results_df = pd.DataFrame(results, columns=['eps', 'min_samples', 'Silhouette Score'])

# Displaying the results
print(results_df)
```

```
   eps  min_samples  Silhouette Score
0  0.1            2          0.073279
1  0.1            5          0.063499
2  0.1           10         -0.141363
3  0.2            2          0.324138
4  0.2            5          0.324138
5  0.2           10          0.271250
```

The optimal combination of parameters is identified by locating the one with the highest silhouette score. This indicates the best configuration for the DBSCAN algorithm based on the evaluations performed.

```
best_params = results_df.loc[results_df['Silhouette Score'].idxmax()]

print('Optimal combination:')
print(f'eps: {best_params["eps"]}, min_samples: {best_params["min_samples"]}')
print(f'Silhouette score: {best_params["Silhouette Score"]:.2f}')
```

```
Optimal combination:
eps: 0.2, min_samples: 2.0
Silhouette score: 0.32
```

5. Visualising the results

Finally, after finding the optimal parameters, we visualise the clustering results using those parameters.

```
optimal_eps = best_params['eps']
optimal_min_samples = int(best_params['min_samples'])

optimal_labels = apply_dbscan(X, optimal_eps, optimal_min_samples)

plt.figure(figsize=(6,6))
unique_labels = set(optimal_labels)
colors = plt.cm.get_cmap('Spectral', len(unique_labels))

for k in unique_labels:
    class_member_mask = (optimal_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=30)

plt.title(f'DBSCAN with Optimal Parameters (eps: {optimal_eps}, min_samples: {optimal_min_samples})')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

```
<ipython-input-9-1978ed9abe5c>:8: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed in
  colors = plt.cm.get_cmap('Spectral', len(unique_labels))
```