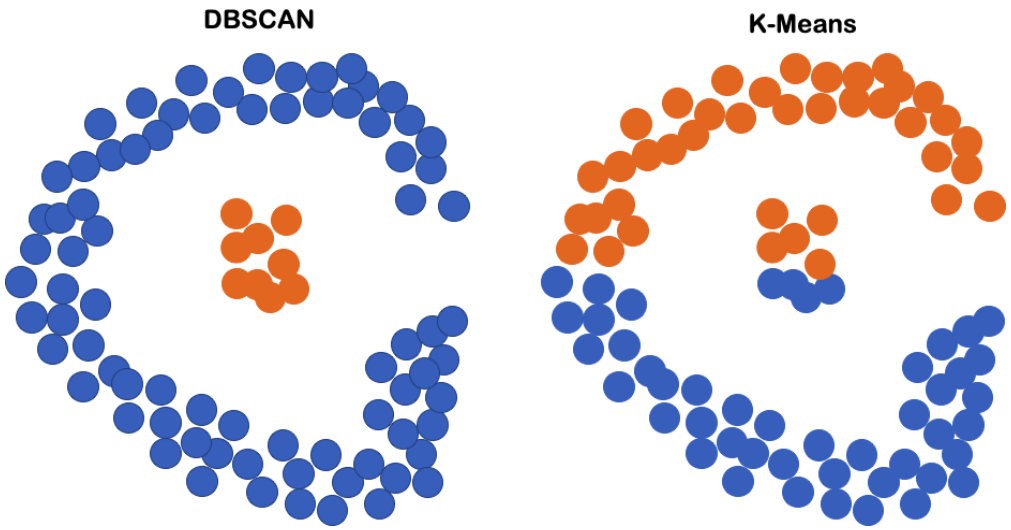
LESSON 8: DBSCAN



This lecture was refered by A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise

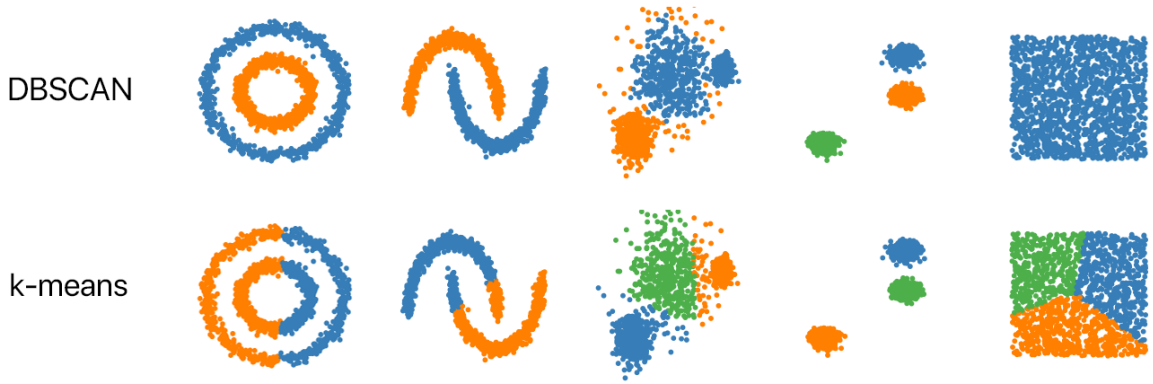
1. DBSCAN introduction

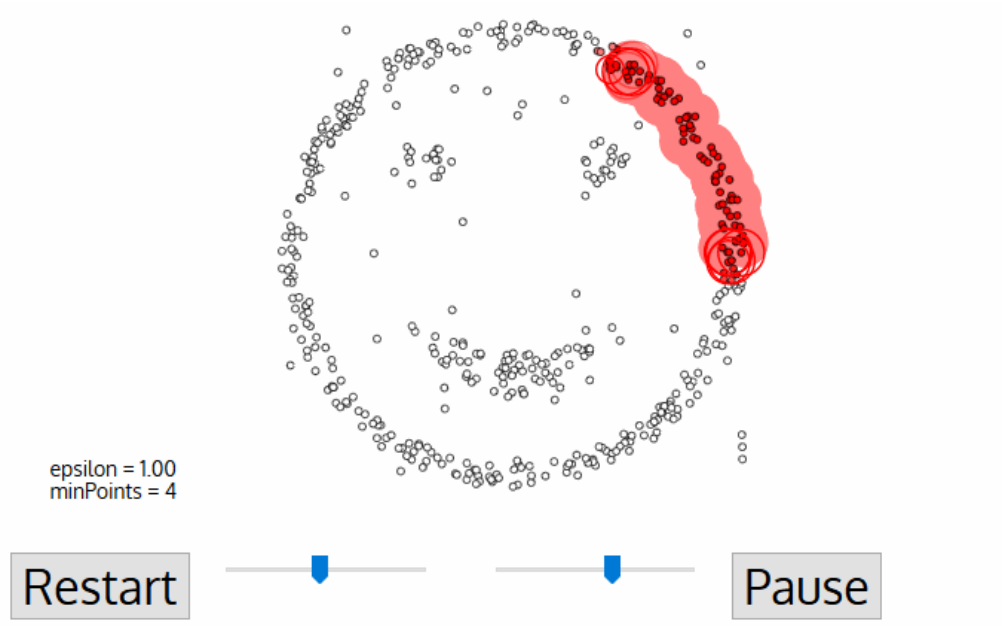
While K-means clustering split the given dataset by choosing center and assign label for each data point, DBSCAN approachs clustering problem in a different way.

Density-**b**ased **s**patial **c**lustering of **a**pplications with **n**oise (or DBSCAN for short), like its name, is based on the density of data points to assign label.

Moreover, DBSCAN accepts noise in its results (noise means some data points will not be assign label).

These properties of DBSCAN solves some remaining problems of K-means clustering.





2. Some definitions in DBSCAN

To prepare for DBSCAN algorithm, we have to understand some definitions

2.1. Eps-neighborhood of a point

Definition 1: Let D be a database of points, the Eps-neighborhood of a point p, denoted by $N_{Eps}(p)$, is defined $N_{Eps}(p) = \{q \in D | dist(p, q) \leq Eps\}$.

2.2. Directly density-reachable

Definition 2: A point p is directly density-reachable from a point q wrt. Eps, MinPts if 1) $p \in N_{Eps}(q)$
2) $|N_{Eps}(q)| > MinPts$ (core point condition)

2.3. Density-reachable

Definition 3: A point p is density-reachable from a point q wrt. Eps and MinPts if there is a chain of points $P_1, \ldots, P_n, P_1 = q, P_n = p$ such that P_{i+1} is directly density-reachable from P_i.

2.4. Density-connected

Definition 4: A point p is density-connected to a point q wrt. Eps and MinPts if there is a point o such that both, p and q are density-reachable from o wrt. Eps and MinPts.

2.5. Cluster

Definition 5: Let D be a database of points. Cluster C wrt. Eps and MinPts is a non-empty subset of D satisfying the following conditions: 1) **Maximality**: $\forall p, q$: if $p \in C$ and q is density-reachable from p wrt. Eps and MinPts, then $q \in C$
2) **Connectivity**: $\forall p, q \in C$: p is density-connected to q wrt. Eps and MinPts.

2.6. Noise

Definition 6: Let $C_1 \ldots . C_k$ be the clusters of the database D wrt. parameters Eps_i and $MinPts_i$, $i = 1, \ldots, k$. Then we define the noise as the set of points in the database D not belonging to any cluster C_i, i.e. noise = $\{p \in D | \forall i : p \notin C_i\}$.

3. The algorithm

With the above definitions, we have DBSCAN algorithm

Input: Dataset, Eps, MinPts

Output: Label of each data point in the dataset.

For each point p in dataset:

Step 1: Get the next cluster id c.

Step 2:

- If p is not assigned cluster, go to ***Step 3***.
- Else, go to the next point p in the dataset.

Step 3: Get all the points around p, we call seeds_list.

- If length of seeds_list < MinPts, **assign label of p to NOISE** and go to the next point p in the dataset.
- Else, **assign all points in seeds_list to cluser id c** and go to ***Step 4***.

Step 4: ***For each point q in seeds_list:***

Step 4.1: Get the points around q.

- If number of points around q < MinPts, remove q from seed_list and go to the next point q in seeds_list.
- Else, go to *Step 4.2*.

Step 4.2: For each point t in the points around q.

- If t doesn't have label, **assign label of t to cluser id c** and add t into seed_list.
- If label of t is NOISE, **assign label of t to cluser id c**.
- Else:
  - If there are remaining points in the points around q, go to the next point in the points around q.
  - Else: end the loop and go to *Step 4.3*

Step 4.3:

- If there are remaining points in seeds_list, go to the next point in seeds_list.
- Else: end the loop and go to ***Step 5***.

Step 5:

- If there are remaining points in the dataset, go to the next point in dataset.
- Else: end the algorithm.

4. Implementation example

4.1. Prepare library and data

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

sns.set()
```

In [2]:
```
iris_df = sns.load_dataset('iris')
iris_df
```

Out[2]:

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

150 rows × 5 columns

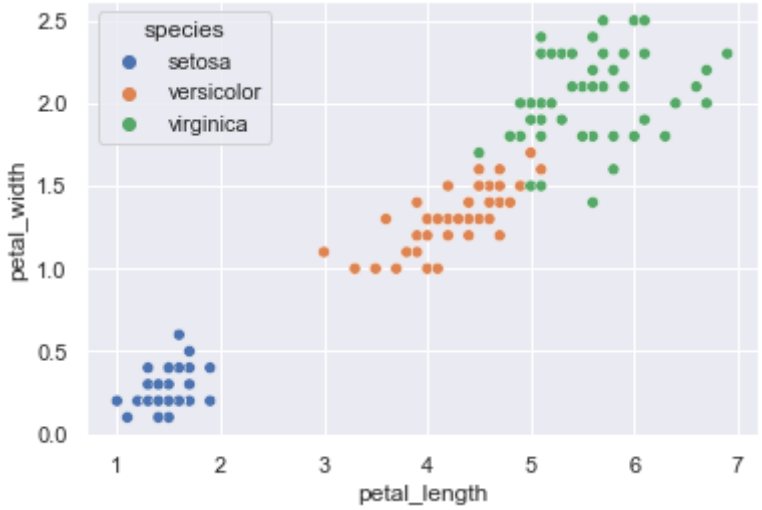
In [3]:
```
df = iris_df.drop(columns=['sepal_length', 'sepal_width'])
df
```

Out[3]:

	petal_length	petal_width	species
0	1.4	0.2	setosa
1	1.4	0.2	setosa
2	1.3	0.2	setosa
3	1.5	0.2	setosa
4	1.4	0.2	setosa
...	...	...	...
145	5.2	2.3	virginica
146	5.0	1.9	virginica
147	5.2	2.0	virginica
148	5.4	2.3	virginica
149	5.1	1.8	virginica

150 rows × 3 columns

In [4]:
```
sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species')
plt.show()
```



In [5]:
```
X = np.array(df.iloc[:, :2])
X.shape
```

Out[5]:
```
(150, 2)
```

In [6]:
```
X
```

Out[6]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.7, 0.4],
       [1.4, 0.3],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
       [1.5, 0.3],
```

```
[1.7, 0.2],
[1.5, 0.4],
[1. , 0.2],
[1.7, 0.5],
[1.9, 0.2],
[1.6, 0.2],
[1.6, 0.4],
[1.5, 0.2],
[1.4, 0.2],
[1.6, 0.2],
[1.6, 0.2],
[1.5, 0.4],
[1.5, 0.1],
[1.4, 0.2],
[1.5, 0.2],
[1.2, 0.2],
[1.3, 0.2],
[1.4, 0.1],
[1.3, 0.2],
[1.5, 0.2],
[1.3, 0.3],
[1.3, 0.3],
[1.3, 0.2],
[1.6, 0.6],
[1.9, 0.4],
[1.4, 0.3],
[1.6, 0.2],
[1.4, 0.2],
[1.5, 0.2],
[1.4, 0.2],
[4.7, 1.4],
[4.5, 1.5],
[4.9, 1.5],
[4. , 1.3],
[4.6, 1.5],
[4.5, 1.3],
[4.7, 1.6],
[3.3, 1. ],
[4.6, 1.3],
[3.9, 1.4],
[3.5, 1. ],
[4.2, 1.5],
[4. , 1. ],
[4.7, 1.4],
[3.6, 1.3],
[4.4, 1.4],
[4.5, 1.5],
```

```
[4.1, 1. ],
[4.5, 1.5],
[3.9, 1.1],
[4.8, 1.8],
[4. , 1.3],
[4.9, 1.5],
[4.7, 1.2],
[4.3, 1.3],
[4.4, 1.4],
[4.8, 1.4],
[5. , 1.7],
[4.5, 1.5],
[3.5, 1. ],
[3.8, 1.1],
[3.7, 1. ],
[3.9, 1.2],
[5.1, 1.6],
[4.5, 1.5],
[4.5, 1.6],
[4.7, 1.5],
[4.4, 1.3],
[4.1, 1.3],
[4. , 1.3],
[4.4, 1.2],
[4.6, 1.4],
[4. , 1.2],
[3.3, 1. ],
[4.2, 1.3],
[4.2, 1.2],
[4.2, 1.3],
[4.3, 1.3],
[3. , 1.1],
[4.1, 1.3],
[6. , 2.5],
[5.1, 1.9],
[5.9, 2.1],
[5.6, 1.8],
[5.8, 2.2],
[6.6, 2.1],
[4.5, 1.7],
[6.3, 1.8],
[5.8, 1.8],
[6.1, 2.5],
[5.1, 2. ],
[5.3, 1.9],
[5.5, 2.1],
[5. , 2. ],
```

```
       [5.1, 2.4],
       [5.3, 2.3],
       [5.5, 1.8],
       [6.7, 2.2],
       [6.9, 2.3],
       [5. , 1.5],
       [5.7, 2.3],
       [4.9, 2. ],
       [6.7, 2. ],
       [4.9, 1.8],
       [5.7, 2.1],
       [6. , 1.8],
       [4.8, 1.8],
       [4.9, 1.8],
       [5.6, 2.1],
       [5.8, 1.6],
       [6.1, 1.9],
       [6.4, 2. ],
       [5.6, 2.2],
       [5.1, 1.5],
       [5.6, 1.4],
       [6.1, 2.3],
       [5.6, 2.4],
       [5.5, 1.8],
       [4.8, 1.8],
       [5.4, 2.1],
       [5.6, 2.4],
       [5.1, 2.3],
       [5.1, 1.9],
       [5.9, 2.3],
       [5.7, 2.5],
       [5.2, 2.3],
       [5. , 1.9],
       [5.2, 2. ],
       [5.4, 2.3],
       [5.1, 1.8]])
```

In [7]:
```
y = df['species']
y
```

Out[7]:
```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
          ...
145    virginica
```

```
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

4.2. Use sklearn

In [8]:
```
from sklearn.cluster import DBSCAN
```

In [9]:
```
sklearn_dbscan = DBSCAN(
    eps=0.22,
    min_samples=3
)
```

In [10]:
```
sklearn_dbscan.fit(X)
```

Out[10]:
```
DBSCAN(eps=0.22, min_samples=3)
```

In [11]:
```
y
```

Out[11]:
```
0         setosa
1         setosa
2         setosa
3         setosa
4         setosa
          ...
145    virginica
146    virginica
147    virginica
148    virginica
149    virginica
Name: species, Length: 150, dtype: object
```

In [12]:
```
sklearn_dbscan.labels_
```

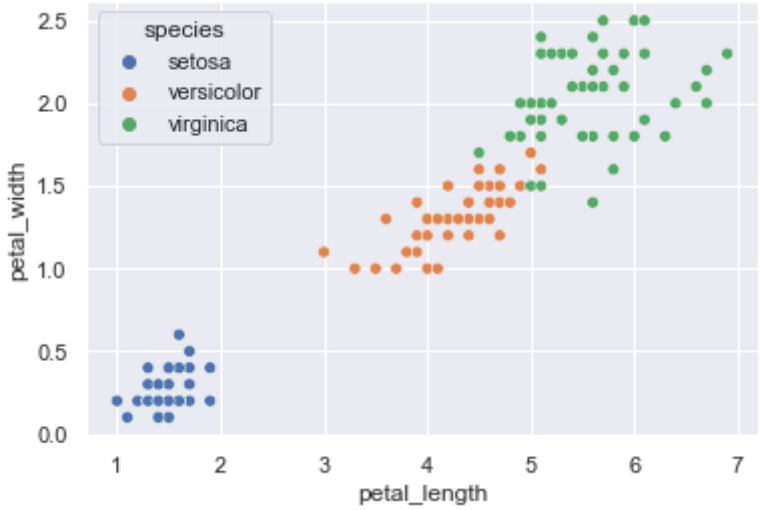
Out[12]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  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,  2,  1,
        2,  3,  2,  4,  1, -1,  3,  2,  1,  1,  2,  1,  2,  2,  3,  4, -1,
```

```
       1,  2,  1,  4,  1,  2,  3,  1,  1,  2,  3,  3, -1,  2,  1, -1,  2,
       2,  3,  1,  2,  2,  2,  1,  2,  2,  2,  1,  1,  2,  1])
```

In [13]:
```
def visualize_clusters(X, cluster_preds):
    colors = ['r', 'g', 'b', 'k', 'c', 'y']
    for x, cluster in zip(X, cluster_preds):
        plt.plot(x[0], x[1], f'{colors[cluster]}o')
    plt.show()
```

In [14]:
```
visualize_clusters(X, sklearn_dbscan.labels_)
```

In [15]:
```
sns.scatterplot(data=df, x='petal_length', y='petal_width', hue='species')
plt.show()
```



5. Homework

5.1. Exercise 1:

Load penguins dataset from seaborn, use `bill_length_mm` and `bill_depth_mm` to build a DBSCAN model to cluster `Adelie` species, `Chinstrap` species and `Gentoo` species.

5.2. Exercise 2:

Similar to Exercise 1, but now using `bill_length_mm` , `bill_depth_mm` , `flipper_length_mm` and `body_mass_g` .

In [ ]: