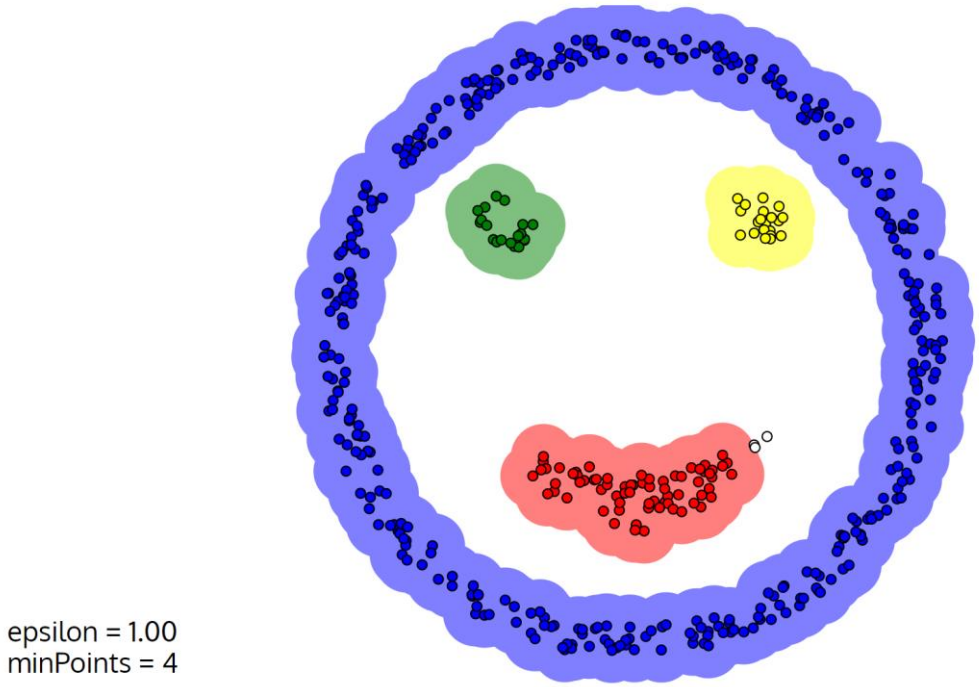
基于密度的噪声应用空间聚类（DBSCAN）

一、简介

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) 是一种基于密度的聚类算法，由 Martin Ester, Hans-Peter Kriegel, Jörg Sander 和 Xiaowei Xu 于 1996 年提出。它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并可在噪声的空间数据库中发现任意形状的聚类。



基本概念：

ε-邻域：对于样本集中的 xi，它的 ε-邻域为样本集中与它距离小于 ε 的样本所构成的集合。

在 DBSCAN 算法中，数据点被分为以下三类：

1.核心点：若样本 xi 的 ε-邻域内至少包含了 MinPts 个点，则为核心点。

2.边界点：若样本 xi 的 ε-邻域内包含的点的数量小于 MinPts，但它在其他核心点的 ε-邻域内，则为边界点。

3.噪声点：既非核心点也非边界点则为噪声点。

密度直达：若 q 处于 p 的 ε-邻域内，且 p 为核心点，则称 q 由 p 密度直达。

密度可达：若有一个点的序列 q0、q1、….、qk，若对任意的 qi->qi-1 是密度直达的，则称从 q0 到 qk 密度可达。

密度相连：若从某核心点 p 出发，点 q 和点 k 都是密度可达的，则称点 q 和点 k 密度相连。

算法需要定义参数：

1、阈值 minPts

2、半径 ε

二、用代码实现 DBSCAN 算法

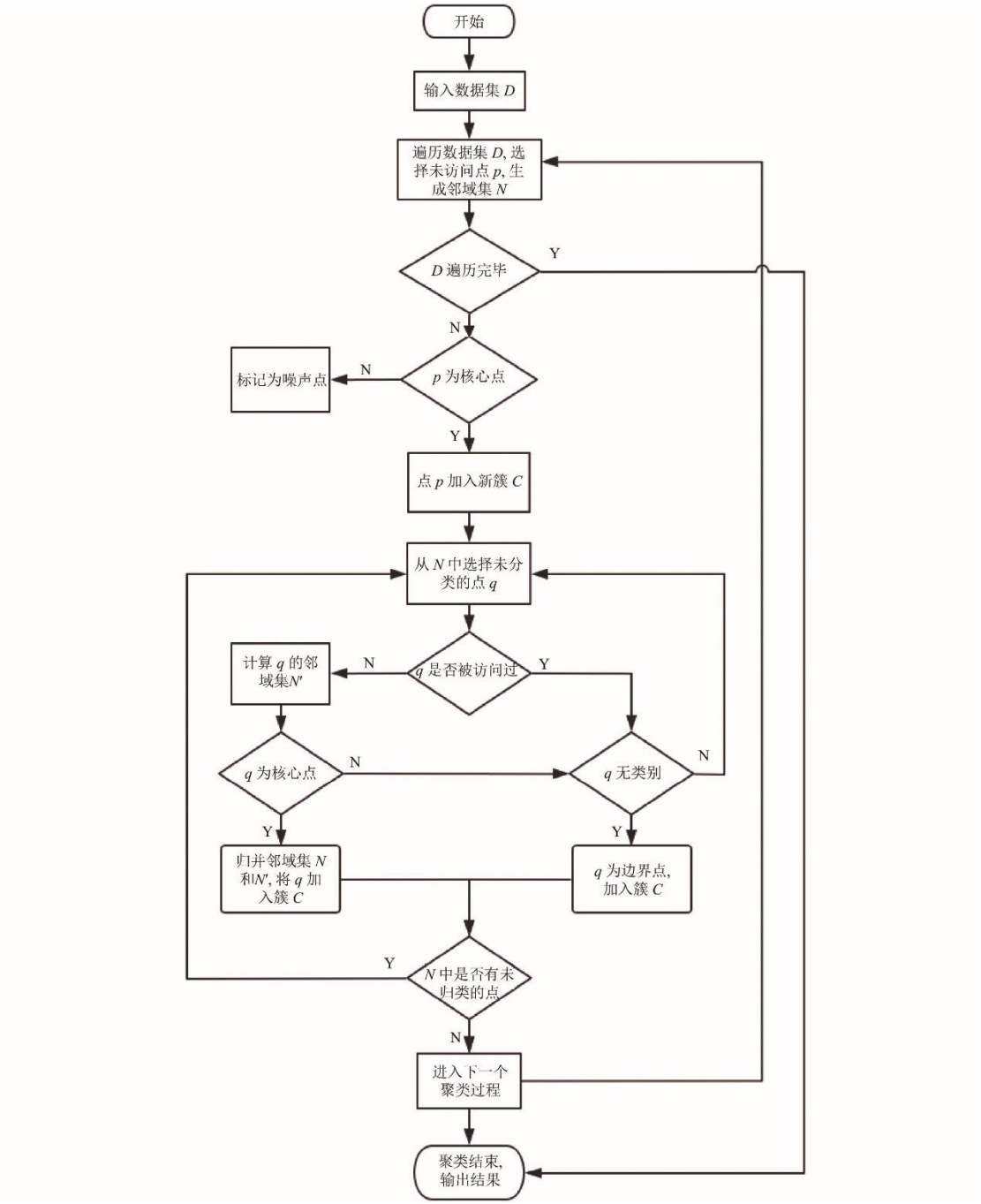
DBSCAN 算法的基本思路：

1、选择核心点：如果一个点的 ε-邻域内点数超过 minPts，将其标记为核心点。

2、构建邻域链：对每个核心点，将它的 ε-邻域内所有点（包括其他核心点）连接起来，形成一个聚类。

3、边界点的归属：将边界点分配给与之相连的核心点的聚类。

4、标记噪声：最后，未被归入任何聚类的点被标记为噪声。



伪代码：

```
对于数据集D：
首先将所有数据都标记为unvisited；

DO
任取一个未标记数据点p：
将p标记为visited；
    if p是核心点：
        将其添加到新的簇C中；
        将p邻域中的每个点添加到N中；
        for p' in N：
            if p'是 unvisited：
                将p'标记为visited；
                if p'是核心点：
                    将p'邻域内的点添加到N中；
                if p'未被分配到簇中，将其添加到簇C中；
    else p为噪声，将其添加到-1簇中；
Until 没有unvisited的对象
```

代码：

```python
import numpy as np
from sklearn.metrics.pairwise import euclidean_distances
from collections import deque
import pandas as pd

def dbscan(X, eps, minPts):
    n = X.shape[0]
    labels = np.full(n, -1, dtype=int)
    cluster_id = 0
    visited = np.zeros(n, dtype=bool)

    # 获取邻域内的点
    def region_query(p):
        distances = euclidean_distances(X[p].reshape(1, -1), X).flatten()
        return np.where(distances <= eps)[0]

    # 扩展聚类
    def expand_cluster(p, neighbors):
        labels[p] = cluster_id
        queue = deque(neighbors)
        while queue:
            point = queue.popleft()
            if not visited[point]:
                visited[point] = True
                point_neighbors = region_query(point)
                if len(point_neighbors) >= minPts:
                    queue.extend(point_neighbors)
                if labels[point] == -1:
                    labels[point] = cluster_id

    for p in range(n):
        if not visited[p]:
            visited[p] = True
            p_neighbors = region_query(p)
            if len(p_neighbors) >= minPts:
                expand_cluster(p, p_neighbors)
                cluster_id += 1

    return labels
```

执行步骤：

1、初始化：

labels 数组，用于存储每个点的聚类标签，初始值为-1 表示未分类。

visited 数组，用于标记点是否被访问过，以避免重复处理。

2、邻域查询函数 (region_query)：

对于每个点，计算与其他所有点的欧式距离。

return 距离小于或等于 eps 的所有点的索引。

3、扩展聚类函数 (expand_cluster)：

从一个核心点开始，通过递归地探索其邻域中的所有点来扩展聚类。

使用队列来管理待探索的点，以实现广度优先搜索。

对于队列中的每个点，如果它是未访问的，则检查它是否能成为新的核心点，并继续扩展。

4、迭代处理：

遍历所有点，对于每个未访问的点，执行邻域查询。

如果邻域内的点数满足 minPts，则从该点开始扩展聚类。

每成功扩展一个聚类，聚类 ID 递增。

时间复杂度优化：

采用了 sklearn 库中的 euclidean_distances 函数来计算点与所有其他点之间的距离，这是一种向量化操作，可以比逐个计算距离更快地完成。然后进行了区域查询的优化，即通过先计算所有点的距离，然后筛选出在 ε 范围内的点，减少了重复的距离计算。虽然这种方法在最坏情况下的时间复杂度仍然为 O(n^2)，但在实际应用中，由于距离矩阵的预计算和内存中的高效访问，性能得到了实质提升。如需要后续提升，可使用空间索引（如 k-d 树、R 树）优化 ε 距离内的点的访问以降低复杂性。

空间复杂度优化：

该算法主要空间开销来自于存储每个点的标签 labels 和访问状态 visited，这两者都是 O(n)。额外的空间开销来自于存储距离计算结果和队列。通过使用简单的布尔数组来标记访问过的点，并利用双端队列（deque）来处理聚类扩展。队列只有在极端情况下可能需要存储接近 n 个元素，这样做既高效又节省空间。

三、参数调优

数据集选择：

为了测试 DBSCAN 算法，我使用了 UCI 机器学习库中的 Iris 数据集。

该数据集包含了 150 个样本和 4 个特征，每个样本描述了鸢尾花的花萼和花瓣的长度和宽度。Iris 数据集的规模适中，且特征维度较低，这不仅有助于展示 DBSCAN 算法处理多维数据的能力，而且便于进行计算和可视化。此外，这个数据集包含三个不同种类的鸢尾花，其在特征空间中呈现出明显的聚类结构，这使得它成为测试聚类算法特别是 DBSCAN 性能的理想选择。由于数据集已经包含了真实的类标签，它还允许我们使用监督学习的评价指标来评估聚类的效果，例如调整兰德指数。但是，Iris 数据集中并不包含噪声点或异常值，不足以展示 DBSCAN 在处理噪声方面的能力，所以可以人为添加一些随机分布的数据点。

参数影响：

DBSCAN 算法的效果在很大程度上取决于 ε 和 minPts 这两个参数的选择。参数的不同取值可能会导致聚类结果的显著变化。

较小的 ε 值将导致大多数数据点被视为噪声，因为不够多的邻近点满足 minPts 条件，造成大量的单点聚类或者完全没有聚类，从而导致较低的 ARI 值；较大的 ε 值将本应分开的多个聚类合并为一个聚类，因为不同聚类之间的边界点可以互相到达，导致过度聚类，同样降低 ARI 值。

较小的 minPts 值使得较少的邻近点就可以形成核心点，导致更多的点被包含在聚类中，增加了噪声点被错误归类的风险；而较大的 minPts 值需要更多的邻近点才能形成核心点，这可以帮助算法识别出真正的密集区域，减少噪声点的干扰，但同时可能会导致一些边缘但有效的聚类点被视为噪声。

参数选择：

对于选择邻域半径 ε，除了遍历枚举，有一个常见方法是使用 k-距离图。

k-距离：给定数据集 P={p(i)；i=0，1，2，.....n}，计算点 P(i)到集合 D 的子集 S 中所有点之间的距离，距离按照从小到大的顺序排序，d(k)就被称为 k-距离。简单来说，对于数据集中的每一个点，计算它与最近的 k 个点之间的距离，并绘制这些距离的图。通常，这个图会在合适的 ε 值处出现一个拐点，在后面会具体说明。

对于选择 minPts 则较多凭经验。minPts 定义了一个点的邻域中需要有多少个点才能将其视为核心点。minPts 的选择与数据的维度、密度和噪声水平密切相关。一般来说，更高的维度和噪声水平需要更大的 minPts 值。

实验设定：

我将在 Iris 数据集上，尝试不同的ε和 MinPts 参数组合，以此来比较不同的组合对于结果的影响，尝试找到最好的参数组合。

由于轮廓系数不考虑噪声点，而 DBSCAN 可能识别出大量的噪声点，这可能会扭曲聚类效果的评价。所以采用调整兰德指数（Adjusted Rand Index, ARI）衡量聚类结果。而这就需要已经包含真实的类标签的数据集，所以 Iris 数据集被纳入考量。同时，之前提到过我会在数据集中人为添加一些随机分布的数据点，这些点在特征空间中与其他数据点明显不同。

```python
# 加载Iris数据集
iris = datasets.load_iris()
X = iris.data
y = iris.target

# 数据标准化
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 添加噪声点
np.random.seed(42)
noise = np.random.uniform(-3, 3, (20, 4))  # 在四个特征上均匀分布的噪声
X_noisy = np.vstack([X_scaled, noise])

# 测试不同的eps和minPts
eps_values = np.arange(0.1, 2.0, 0.1)
minPts_values = range(2, 6)
best_ari = -1
best_eps = None
best_minPts = None

for eps in eps_values:
    for minPts in minPts_values:
        labels = dbscan(X_scaled, eps, minPts)
        # 只计算有效聚类的ARI（忽略全部为噪声的情况）
        if len(set(labels)) > 1:
            ari = adjusted_rand_score(y, labels)
            print(f'EPS: {round(eps, 2)}, minPts: {minPts}, Adjusted Rand Index: {ari}')

            if ari > best_ari:
                best_ari = ari
                best_eps = eps
                best_minPts = minPts

# 打印最佳结果
print(f'Best EPS: {round(best_eps, 2)}, Best minPts: {best_minPts}, Best ARI: {best_ari}')
```

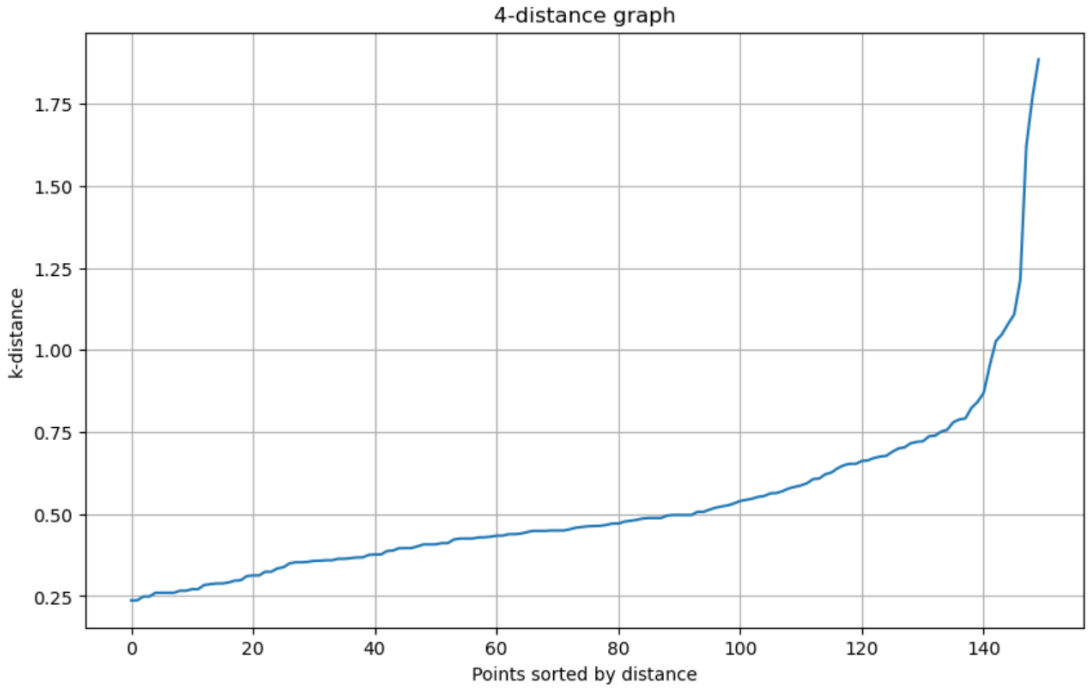
运行得到最佳参数组合：

```
Best EPS: 1.4, Best minPts: 2, Best ARI: 0.5681159420289855
```

然后我尝试绘制 **k-距离图**来确定最佳参数组合。

DBSCAN 的核心思想是基于密度的聚类。在 k-距离图中，如果某个点的 k-距离突然增大，这通常意味着从该点到其第 k 近邻的距离比其他点要大得多。这样的变化暗示了数据点之间密度的显著变化，即从密集区到相对稀疏区的过渡。因此，这个拐点附近的距离可以作为 eps 的一个理想选择，用来分隔密集的聚类与稀疏区或噪声。此外，当选择一个较小的 eps 值时，DBSCAN 可能只能识别出非常密集的聚类，而忽视较少点的区域，这些区域实际上也可以构成有效的聚类。相反，如果 eps 过大，则可能将本应分开的不同聚类合并成一个。k-距离图中的拐点提供了一个平衡点，能有效区分不同聚类，同时识别出噪声。

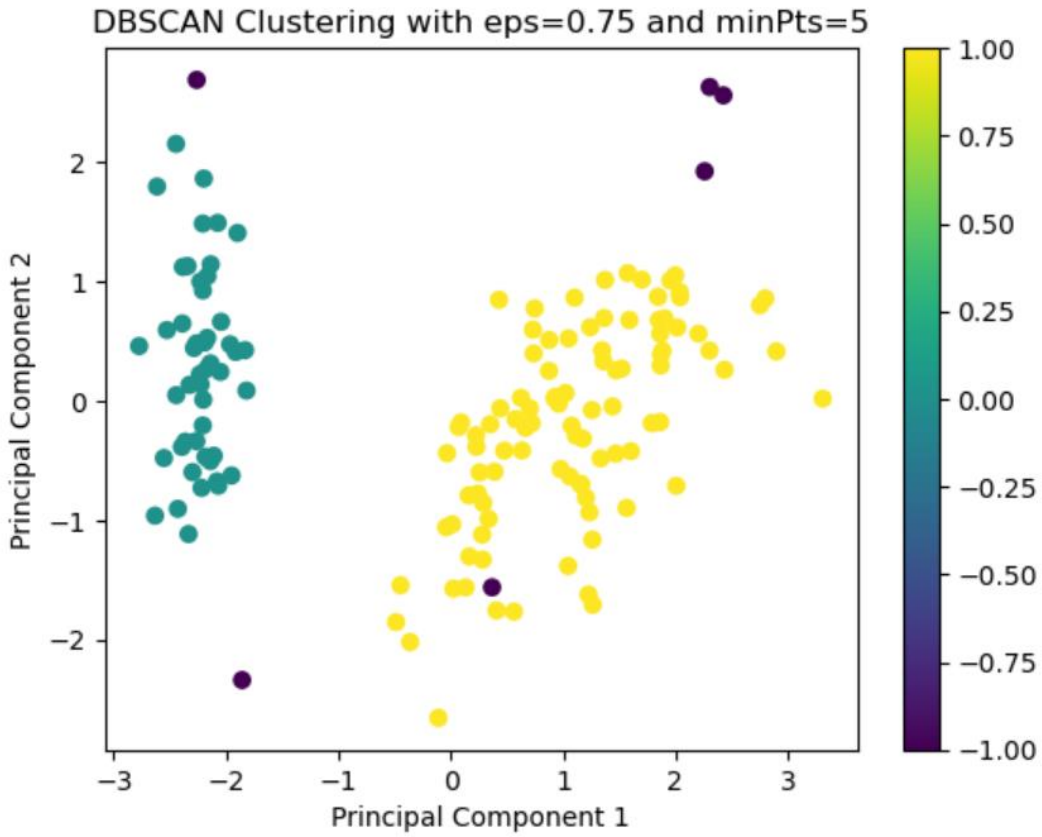
```python
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import euclidean_distances
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# 计算k-距离
def k_distances(X, k):
    dist_matrix = euclidean_distances(X)
    sorted_dist_matrix = np.sort(dist_matrix, axis=1)
    k_dist = sorted_dist_matrix[:, k]
    return k_dist

# 计算k-距离并绘制k-距离图
k = 4  # 通常选取minPts-1，假设我们正在考虑minPts为5
k_dist = k_distances(X_scaled, k)
k_dist_sorted = np.sort(k_dist)
plt.figure(figsize=(10, 6))
plt.plot(k_dist_sorted)
plt.ylabel('k-distance')
plt.xlabel('Points sorted by distance')
plt.title('4-distance graph')
plt.grid(True)
plt.show()
```



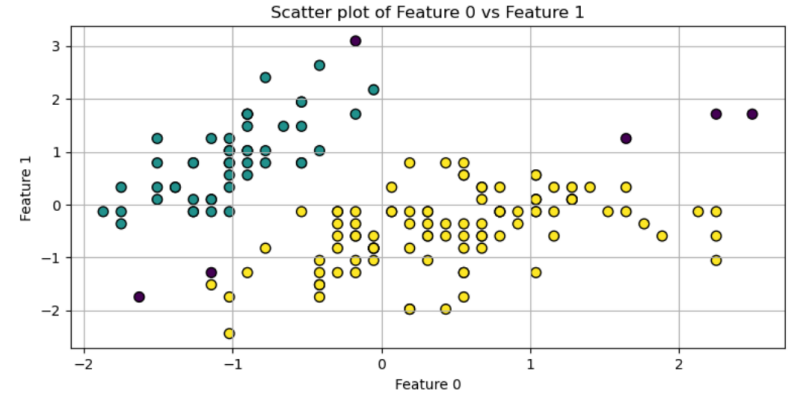
可观察到当纵坐标大于 0.75 时，曲线陡然上升，可取 eps=0.75。

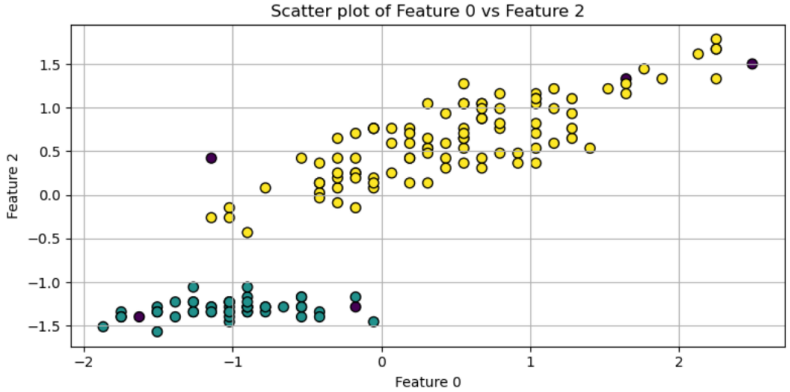
聚类结果展示：

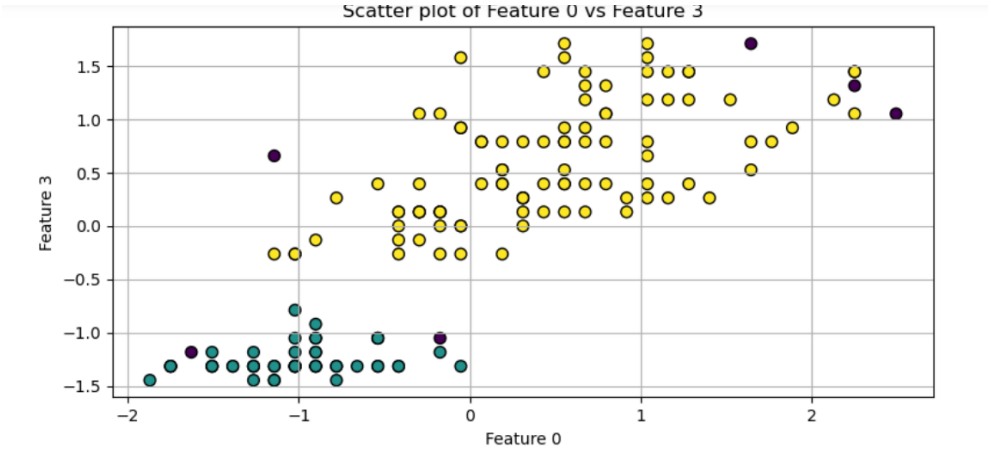
1、通过 PCA 降维，绘制二维可视化图。

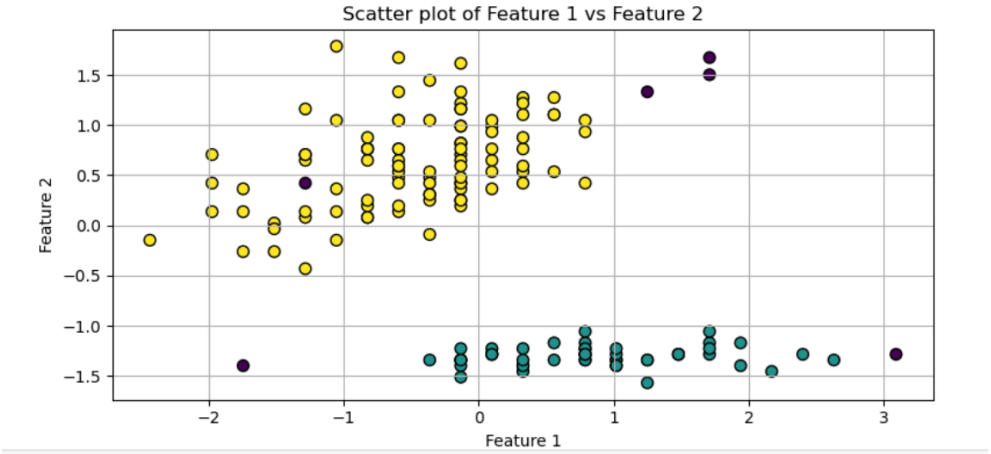


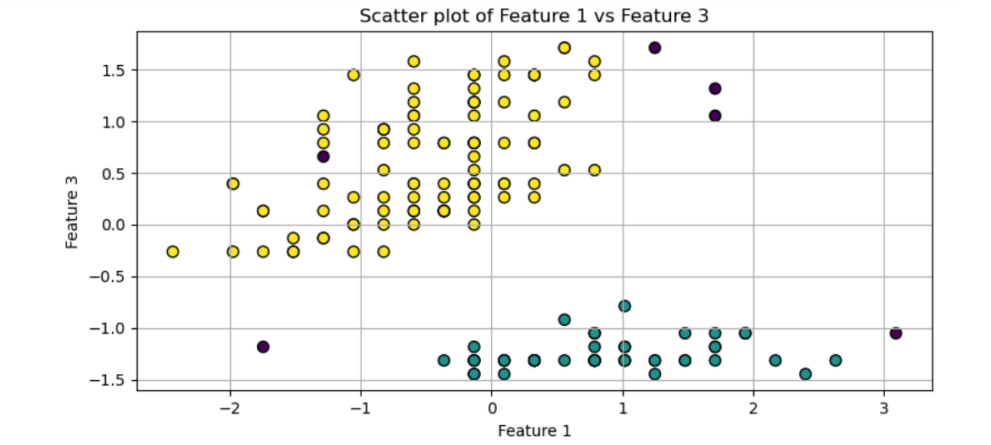
2、通过绘制特征对的散点图矩阵，这样可以观察不同特征组合下的聚类表现。

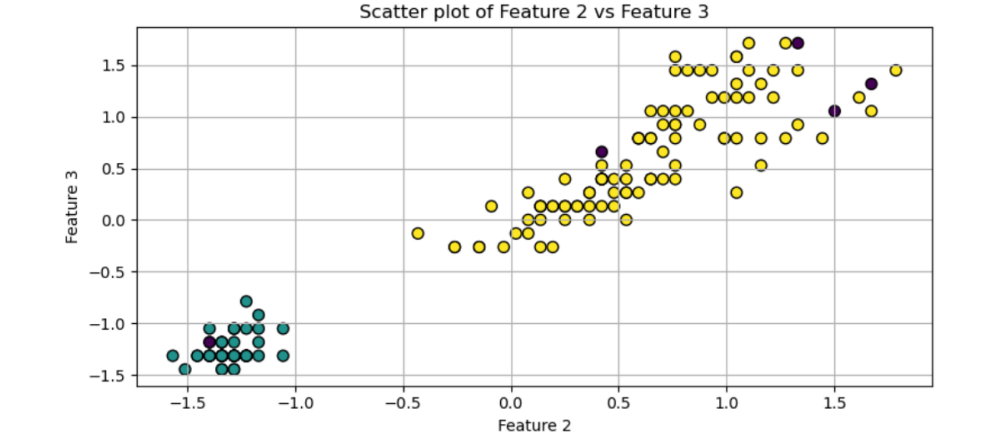





Scatter plot of Feature 0 vs Feature 3
Feature 3
Feature 0


Scatter plot of Feature 1 vs Feature 2
Feature 2
Feature 1


Scatter plot of Feature 1 vs Feature 3
Feature 3
Feature 1


Scatter plot of Feature 2 vs Feature 3
Feature 3
Feature 2

四、DBSCAN 算法的优缺点

优势：

（1）不需要指定簇个数

（2）可以发现任意形状的簇

（3）擅长找到离群点

（4）对于数据库中样本的顺序不敏感

（5）只需要两个参数就足够

劣势：

（1）不能很好反映高维数据（可以做降维）

（2）参数难以选择（参数对结果的影响非常大）

（3）如果样本集的密度不均匀、聚类间距差相差很大时，聚类质量较差