**实验 聚类**

**一、K-Means聚类**

**1、实验目的**

学习K-Means算法基本原理，实现Iris数据聚类。

**2、实验内容**

应用K-Means算法对iris数据集进行聚类。

**3、实验代码**

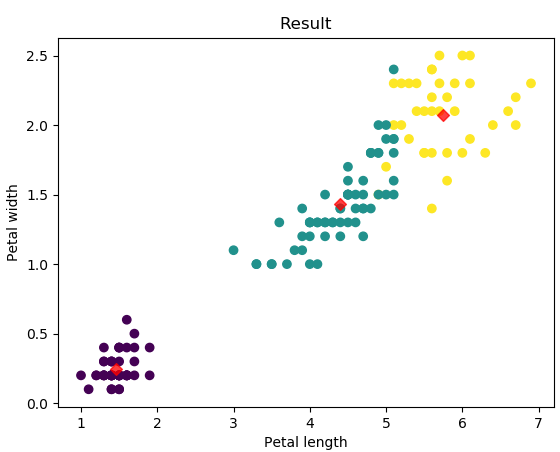
import matplotlib.pyplot as plt  
from sklearn import datasets  
from sklearn.cluster import KMeans  
import pandas as pd  
import numpy as np  
  
def print\_data(want\_print, print\_iris):  
 """  
 展示iris的数据  
 :return: None  
 """  
 print("iris{0}为:\n{1}".format(want\_print, print\_iris))  
 print("=" \* 85)  
  
def show\_data(length, width, title):  
 """  
 画图  
 :param length: 长度  
 :param width: 宽度  
 :param title: 标题  
 :return: None  
 """  
 # 建立一个画布  
 plt.figure(figsize=(14, 7))  
  
 plt.scatter(length, width, c=colormap[y.Targets], s=40)  
 plt.title(title)  
 plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # 导入iris数据  
 iris = datasets.load\_iris()  
 # 展示iris真实数据  
 print\_data(want\_print="数据", print\_iris=iris.data)  
 # 展示iris特征名字  
 print\_data(want\_print="特征名字", print\_iris=iris.feature\_names)  
 # 展示目标值  
 print\_data(want\_print="目标值", print\_iris=iris.target)  
 # 展示目标值的名字  
 print\_data(want\_print="目标值的名字", print\_iris=iris.target\_names)

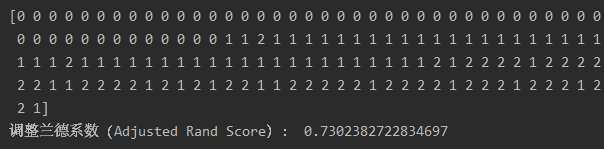
# 创建色板图  
colormap = np.array(['red', 'lime', 'black'])  
  
 # 为了便于使用，将iris数据转换为pandas库数据结构，并设立列的名字  
 # 将iris数据转为pandas数据结构  
 x = pd.DataFrame(iris.data)  
 # 将iris数据的名字设为‘Sepal\_Length’，‘Sepal\_Width’，‘Sepal\_Width’，‘Petal\_Width’  
 x.columns = ['Sepal\_Length', 'Sepal\_Width', 'Petal\_Length', 'Petal\_Width']  
 # 将iris目标值也转为pandas数据结构  
 y = pd.DataFrame(iris.target)  
 # 将iris目标值得名字设为‘Targets’  
 y.columns = ['Targets']  
  
 # 开始画Sepal长度和宽度的关系  
 show\_data(length=x.Sepal\_Length, width=x.Sepal\_Width, title='Sepal')  
 # 开始画Petal长度和宽度的关系  
 show\_data(length=x.Petal\_Length, width=x.Petal\_Width, title='Petal')  
 # 调用Kmeans进行聚类  
  
 # 画出聚类结果，三种颜色表示  
  
 # 画出聚类中心  
  
 # 计算并输出聚类效果指标

**4、实验任务**

完成代码，调用sklearn的K-Means模块实现要求的功能。调整各参数，书写实验心得。

预期结果如下：





**二、DBSCAN聚类**

**1、实验目的**

学习DBSCAN算法基本原理，实现模拟数据的聚类。

**2、实验内容**

使用DBSCAN算法对模拟数据进行聚类，并调整参数观察不同效果。

参数含义：

（1）**eps**：DBSCAN算法参数，即我们的ϵ-邻域的距离阈值，和样本距离超过ϵ的样本点不在ϵ-邻域内。默认值是0.5，一般需要在多组值里面选择一个合适的阈值。eps过大，则更多的点会落在核心对象的ϵ-邻域，此时我们的类别数可能会减少， 本来不应该是一类的样本也会被划为一类。反之则类别数可能会增大，本来是一类的样本却被划分开。

（2）**min\_samples**：DBSCAN算法参数，即样本点要成为核心对象所需要的*ϵ*ϵ-邻域的样本数阈值。默认值是5， 一般需要在多组值里面选择一个合适的阈值。通常和eps一起调参。在eps一定的情况下，min\_samples过大，则核心对象会过少，此时簇内部分本来是一类的样本可能会被标为噪音点，类别数也会变多。反之min\_samples过小的话，则会产生大量的核心对象，可能会导致类别数过少。

（3）**metric**：最近邻距离度量参数。默认为欧氏距离。

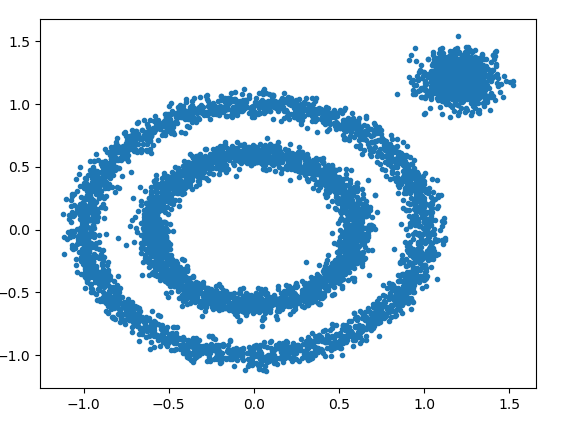
（4）**algorithm**：最近邻搜索算法参数。默认为“auto”，在“brute”、“kd\_tree”、“ball\_tree”中选择最优。

**3、实验代码**

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import datasets  
from sklearn.cluster import DBSCAN  
  
# 为了体现DBSCAN算法在非凸数据上的聚类优点，生成三簇数据，其中两组非凸。对数据进行可视化。  
X1, y1 = datasets.make\_circles(n\_samples=5000, factor=.6, noise=.05)  
X2, y2 = datasets.make\_blobs(n\_samples=1000, n\_features=2, centers=[[1.2, 1.2]], cluster\_std=[[.1]], random\_state=9)  
X = np.concatenate((X1, X2))  
plt.scatter(X[:, 0], X[:, 1], marker='.')  
plt.show()  
  
# 调用sklearn库的DBSCAN模块对模拟数据进行聚类。  
# 修改DBSCAN的参数，优化效果

# 填写代码

模拟数据如图：



**4、实验任务**

完成代码，调整参数，对数据进行聚类，对比不同参数下的聚类效果并分析。

预期结果如下：

