```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import time
```

```
# 西瓜数据集4.0: 密度 含糖率 标签
data = [[0.697, 0.460, 1],
        [0.774, 0.376, 1],
        [0.634, 0.264, 1],
        [0.608, 0.318, 1],
        [0.556, 0.215, 1],
        [0.430, 0.237, 1],
        [0.481, 0.149, 1],
        [0.437, 0.211, 1],
        [0.666, 0.091, 0],
        [0.243, 0.267, 0],
        [0.245, 0.057, 0],
        [0.343, 0.099, 0],
        [0.639, 0.161, 0],
        [0.657, 0.198, 0],
        [0.360, 0.370, 0],
        [0.593, 0.042, 0],
        [0.719, 0.103, 0],
        [0.359, 0.188, 0],
        [0.339, 0.241, 0],
        [0.282, 0.257, 0],
        [0.748, 0.232, 0],
        [0.714, 0.346, 1],
        [0.483, 0.312, 1],
        [0.478, 0.437, 1],
        [0.525, 0.369, 1],
        [0.751, 0.489, 1],
        [0.532, 0.472, 1],
        [0.473, 0.376, 1],
        [0.725, 0.445, 1],
        [0.446, 0.459, 1]]
```

```
# 多维数组中创建DataFrame（二维表）,需要为DataFrame赋值columns和index(默认为数字)
column = ['density', 'sugar_rate', 'label']
dataSet = pd.DataFrame(data, columns=column)
```

```

```

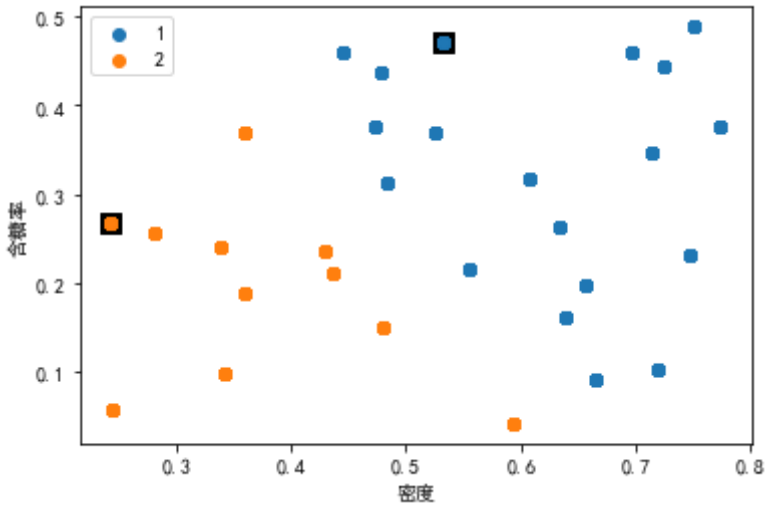
```

```

```

```

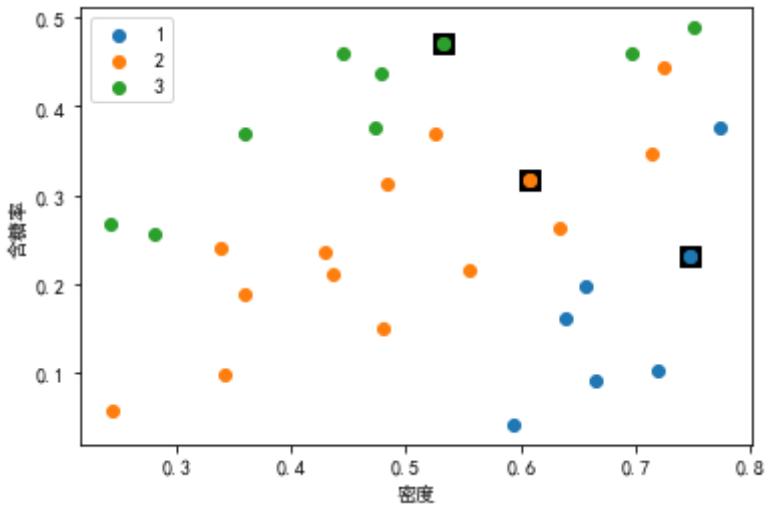
```
# 创建类K_means
class K_means(object):
    # 创建__init__方法，在面向对象编程中，给未来创建的对象所定义的进行初始化属性
    def __init__(self, k, data, loop_times, error):
        self.k = k
        self.data = data
        self.loop_times = loop_times
        self.error = error

    def distance(self, p1, p2):
        # linalg=linear（线性）+algebra（代数），norm则表示范数
        # 求p = 2 时的闵可夫斯基距离，即欧氏距离
        return np.linalg.norm(np.array(p1) - np.array(p2))

    def fitting(self):
        time1 = time.perf_counter()  # 返回性能计数器的值（以分秒为单位），表示程序开始运行到调用这个语句所经历的时间
        mean_vectors = random.sample(self.data, self.k)  # 随机选取k个初始样本
        initial_main_vectors = mean_vectors
        for vec in mean_vectors :
            plt.scatter(vec[0], vec[1], s=100, color = 'black', marker='s')  # 画出初始聚类中心，以黑色正方形（square）表示

        times = 0
        # map(),高阶函数，它接收一个函数 f 和一个 list，并通过把函数 f 依次作用在 list 的每个元素上，得到一个新的 list 并返回
        # lambda:返回可调用的函数对象，通常是在需要一个函数，但又不想命名一个函数时使用，lambda x : [x] 表示输入x，输出为[x]
        clusters = list(map((lambda x:[x]), mean_vectors))
        while times < self.loop_times:
            change_flag = 1  # 标记簇均值向量是否改变
            for sample in self.data:
                dist = []
                for vec in mean_vectors:
                    dist.append(self.distance(vec, sample))  # 计算样本到每个聚类中心的距离
                clusters[dist.index(min(dist))].append(sample)  # 找到离该样本最近的聚类中心，并将它放入该簇

            new_mean_vectors = []
            for c,v in zip(clusters, mean_vectors):  # zip()将两个对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表
                cluster_num = len(c)
                cluster_array = np.array(c)
                new_mean_vector = sum(cluster_array) / cluster_num  # 计算出新的聚类簇均值向量
                mean_vector = np.array(v)
                # np.divide和np.true_divide结果一样（python3.7.2）,np.floor_divide只保留整数结果
                # all(iterable):如果iterable(元组或者列表)的所有元素不为0、False或者iterable为空，all(iterable)返回True，否则返回False
```

```
                if all(np.true_divide((new_mean_vector - mean_vector),
mean_vector) < np.array([self.error, self.error])):
                    new_mean_vectors.append(mean_vector)  # 均值向量未改变
                    change_flag = 0
                else:
                    # dataFrame转List()，括号不能忘
                    new_mean_vectors.append(new_mean_vector.tolist())  # 均值向量
发生改变

            if change_flag == 1:
                mean_vectors = new_mean_vectors
            else:
                break
            times += 1
        time2 = time.perf_counter()
        # str.format()，基本语法是通过 {} 和 : 来代替以前的 %
        print ('本次选取的{}个初始向量为{}'.format(self.k, initial_main_vectors))
        print ('共进行{}轮'.format(times))
        print ('共耗时{:.2f}s'.format(time2 - time1))  # 取2位小数
        for cluster in clusters:
            x = list(map(lambda arr: arr[0], cluster))
            y = list(map(lambda arr: arr[1], cluster))
            plt.scatter(x, y, marker = 'o', label = clusters.index(cluster)+1)

        plt.xlabel('密度')
        plt.ylabel('含糖率')
        plt.rcParams['font.sans-serif'] = ['SimHei']  # 用来正常显示中文标签
        plt.legend(loc='upper left')

        plt.show()
```

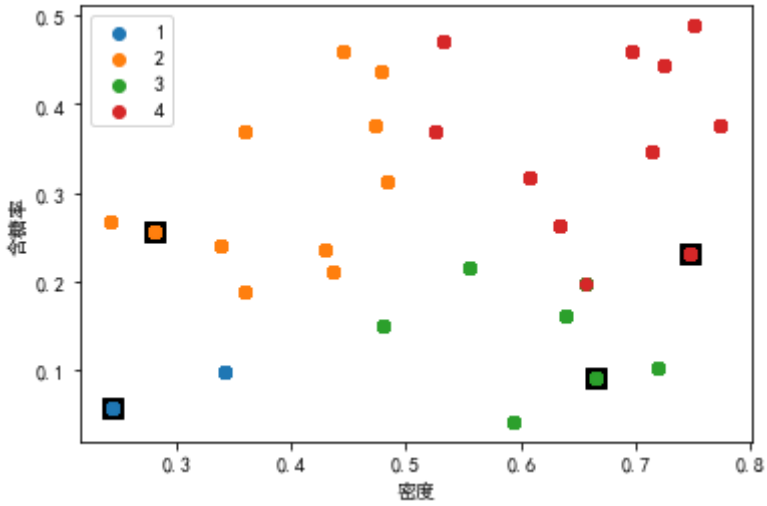
```
本次选取的2个初始向量为[[0.532, 0.472], [0.243, 0.267]]
共进行276轮
共耗时0.90s
```



```
本次选取的3个初始向量为[[0.748, 0.232], [0.608, 0.318], [0.532, 0.472]]
共进行0轮
共耗时0.02s
```

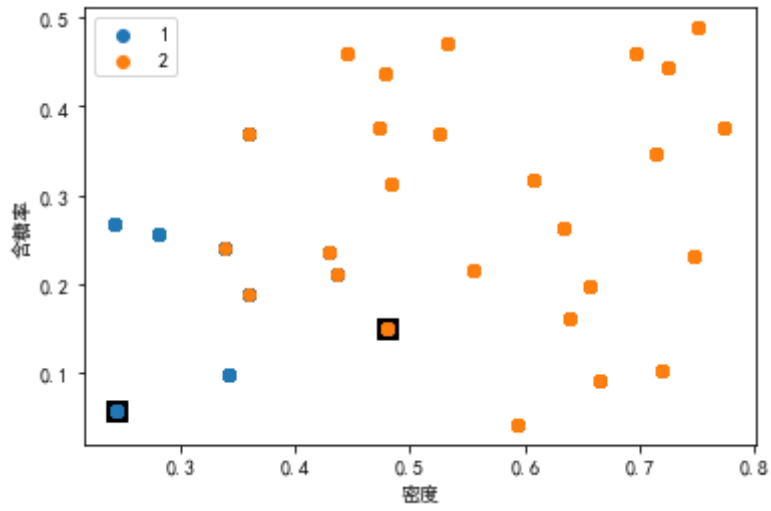



```
本次选取的4个初始向量为[[0.245, 0.057], [0.282, 0.257], [0.666, 0.091], [0.748,
0.232]]
共进行508轮
共耗时3.02s
```

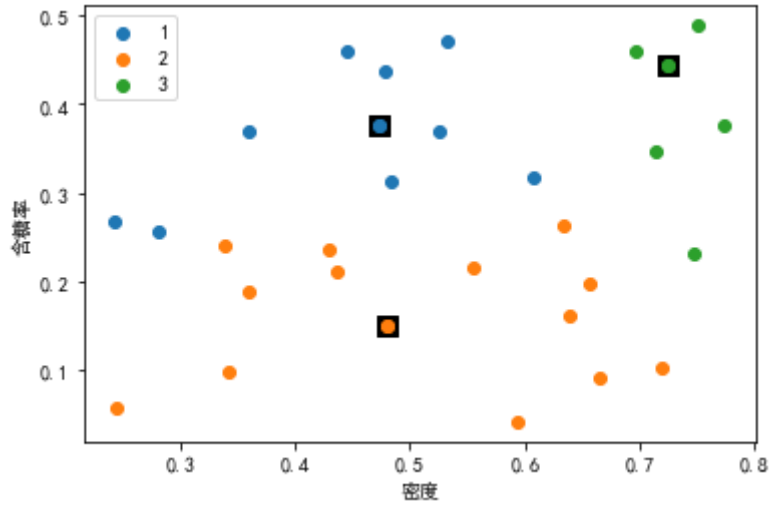



```python
for i in [2, 3, 4]:
    # 调用K_means，执行方法fitting()
    k_means = K_means(i, dataSet[['density', 'sugar_rate']].values.tolist(),
1000, 0.0000001)
    k_means.fitting()
```

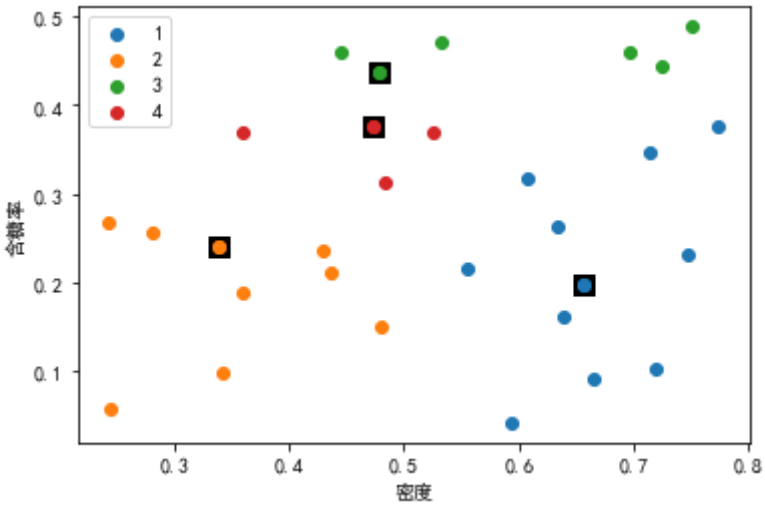
本次选取的2个初始向量为[[0.245, 0.057], [0.481, 0.149]]
共进行1000轮
共耗时11.07s



本次选取的3个初始向量为[[0.473, 0.376], [0.481, 0.149], [0.725, 0.445]]
共进行0轮
共耗时0.09s



本次选取的4个初始向量为[[0.657, 0.198], [0.339, 0.241], [0.478, 0.437], [0.473, 0.376]]
共进行0轮
共耗时0.01s


1
2
3
4
0.5
0.4
0.3
0.2
0.1
含糖率
0.3
0.4
0.5
0.6
0.7
0.8
密度