实验二：K-Means聚类算法

本实验旨在帮助同学们理解K-Means聚类算法的基本概念，掌握K-Means算法的手动实现，学习使用Scikit-learn进行聚类和应用。

1. K-Means算法

基于中心点的算法，旨在将数据点划分为 K 个不同的簇（clusters），其中 K 是预先指定的簇的数量。算法的目标是最小化每个簇内数据点与簇中心（centroid）的距离平方和。

1.1算法流程

（1） **初始化**：随机选择 K 个数据点作为初始簇中心。

$$C = \{c_1, c_2, \ldots, c_K\}$$

（2） **分配**：对于每个数据点，计算它与每个簇中心的距离，并将其分配给最近的簇。

$$L_2(x_i, x_j) = \sqrt{\sum (x_i - x_j)^2}$$

（3） **更新**：计算每个簇的新的簇中心以及计算并更新误差平方和（SSE）。

$$c_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

$$SSE = \sum_{i=1}^{m} \min_{c_j} L_2(x_i - c_j)^2$$

（4） **迭代**：重复步骤 （2） 和 （3），直到满足停止条件（例如簇中心不变或移动小于某个阈值或达到预定的迭代次数）。

1.2算法实现框架

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

def L2(vecXi, vecXj):
    '''
    计算欧氏距离
    para vecXi: 点坐标，向量
    para vecXj: 点坐标，向量
    retrurn: 两点之间的欧氏距离
    '''



def kMeans(S, k, distMeas=L2):
    '''
    K均值聚类
    para S: 样本集，多维数组
```

```
    para k：簇个数
    para distMeas：距离度量函数，默认为欧氏距离计算函数
    return sampleTag：一维数组，存储样本对应的簇标签
    return clusterCents：一维数组，各簇中心
    retrun SSE:误差平方和
    '''




if __name__=='__main__':
    #创建数据样本
    samples = np.array([
        [ 8.76474369, 14.97536963],
        [ 4.54577845, 7.39433243],
        [ 5.66184177, 10.45327224],
        [ 6.02005553, 18.60759073],
        [12.56729723, 5.50656992],
        [ 4.18694228, 14.02615036],
        [ 5.72670608, 8.37561397],
        [ 4.09989928, 14.44273323],
        [ 2.25717893, 1.97789559],
        [ 4.66913545, 0.77178038] ,
        [ 8.1219476, 0.79762128] ,
        [ 0.07972278, -1.9386662 ],
        [ 8.37004706, 10.77781799] ,
        [ 6.6809732, 15.53118858] ,
        [ 5.99194694, 16.57732864] ,
        [ 5.64199016, 15.54671014],
        [-2.92514764, 11.0884457 ],
        [ 4.99694961, 1.98673206] ,
        [ 3.8665841, -1.75282591] ,
        [ 2.62642744, 22.08897582] ,
        [ 5.65622583, 14.77736975] ,
        [-0.33882279, 5.56931142] ,
        [10.93574482, 11.24487206] ,
        [ 4.65023576, 12.78869503] ,
        [ 8.49848513, 9.78769711] ,
        [ 7.53046709, 8.50232567] ,
        [ 6.17118371, 21.74394049] ,
        [-0.93339496, 1.59414249] ,
        [-6.37700491, 3.46389409] ,
        [ 7.13598091, 14.17794597]
        ])
    k = 3
    clusterCents, sampleTag, SSE = kMeans(samples, k)
```

2. K-means++算法

K-means++算法是基于传统K-means的改进版，改进的地方在于，它的聚类中心不是随机产生的，而是通过一种有效的方式给选出来了：

- **随机选择第一个聚类中心**：从数据集中随机选择一个样本作为第一个簇中心。
- **按概率分布选择后续簇中心**：对于数据集中的每个样本点，计算它与已有簇中心中最近的距离(D(x_i))。然后，计算每个样本被选为下一个聚类中心的概率。最后按照轮盘法选择下一个聚类中心。

$$P(x_i) = \frac{D(x_i)^2}{\sum_j D(x_j)^2}$$

- **重复直到选择了 K 个质心**：重复上述过程，直到选择了 K 个簇中心。

3. Scikit-learn

Scikit-learn（通常简称为 sklearn）是一个开源的机器学习库，包括了几乎所有主流的机器学习算法，如分类、回归、聚类和降维等。它是建立在NumPy、SciPy和matplotlib这些Python科学计算库之上的，拥有高效的多维数组操作和数学函数处理能力。

3.1安装和导入

- **pip安装**:

```
In [ ]: pip install scikit-learn
```

- **conda安装**:

```
In [ ]: conda install scikit-learn
```

安装完成后，可以在Python代码中通过以下方式导入sklearn：

```
In [1]: import sklearn
```

3.2应用sklearn聚类

sklearn的cluster包中提供了两种实现k-means算法的方法，分别是KMeans类和k_means函数。

使用KMeans类聚类：

```
In [ ]: from sklearn.cluster import KMeans
        model = KMeans(n_clusters=3, init='random', n_init=10, random_state=0)# 构建聚类
        model.fit(samples) # 训练模型
```

```
In [32]: model.cluster_centers_ # 获取簇中心
```

```
Out[32]: array([[ 1.34171482,  2.35583309],
                [ 7.97954595,  9.00531267],
                [ 5.63555039, 16.27366653]])
```

```
In [24]: model.labels_ # 获取样本的簇标签
```

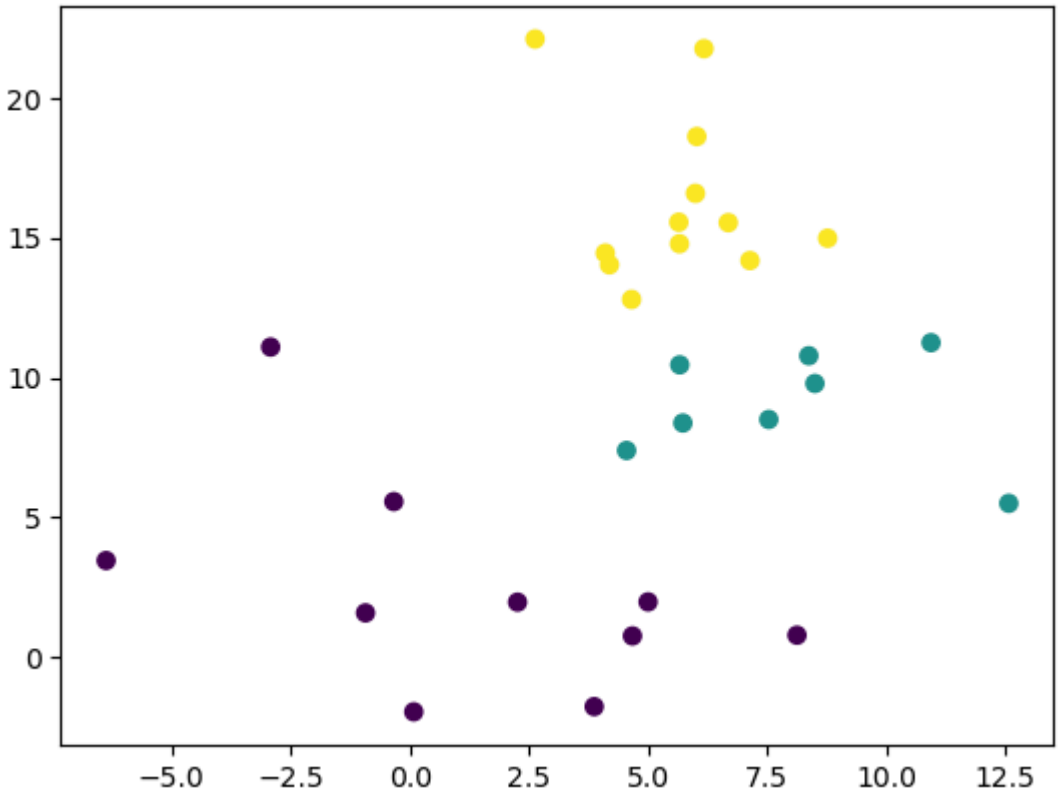
```
Out[24]: array([2, 1, 1, 2, 1, 2, 1, 2, 0, 0, 0, 0, 1, 2, 2, 2, 0, 0, 0, 2, 2, 0,
                1, 2, 1, 1, 2, 0, 0, 2], dtype=int32)
```

使用k_means函数聚类：

```
In [25]: from sklearn.cluster import k_means
         centroid, label, inertia = k_means(samples, n_clusters=3, init='random', n_init=
```

可视化聚类结果：

In [28]:
```
import matplotlib.pyplot as plt
plt.scatter(samples[:, 0], samples[:, 1], c=label)
plt.show()
```



KMeans类提供了更丰富的接口和功能，包括方法如`fit()`，`predict()`，和`transform()`；而k_means函数提供了更快速、更直接的方法执行聚类，直接返回聚类结果，如簇中心、数据点标签和点到最近簇心的平方距离和。