实验 数据可视化与聚类

实验目的

1. 实现kMeans聚类算法，并将其应用于把客户群体划分（可供营销团队参考并相应地制定营销策略）
2. 体验使用pandas和seaborn类库对数据进行统计和可视化

实验任务数据

mall_Customers.csv - 超市购物中心200名客户的一些基本数据，包括如下5个属性：

- CustomerID：客户ID
- Gender：年龄
- Age：年龄
- Annual Income ：年收入（单位：千元）
- Spending Score (1-100)：消费指数

【注】.csv文件是使用逗号和回车区分数据列和行的文本数据集，能够被文本编辑器、Excel和各类数据处理类库访问的一种文件格式。

1. 数据统计及可视化

1.1 使用Pandas数据读取及属性显示

使用pandas.read_csv()读取商场消费者数据，并展示数据集的相关信息。

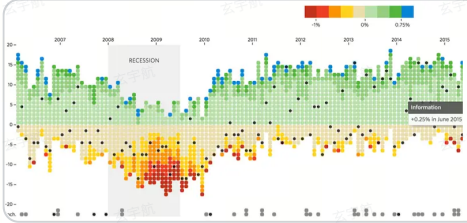
```
import pandas as pd

df = pd.read_csv('mall_Customers.csv') #返回对象为pandas.DataFrame类型的，后面几个都是DataFrame的方法
df.head()     # 用来显示数据集的前5个样本数据
df.shape      # 显示数据集的尺寸
df.describe()     #显示各个特征的统计信息，包括样本数、均值、标准差、最小值、1/4分位数、1/2分位数、3/4分位数和最大值。
df.dtypes     # 展示各个字段的类型信息
df.isnull().sum()   # 该数据集没有取值为null的属性值，可用此函数找空值。
```

1.2 使用seaborn和matplotlib数据可视化

https://seaborn.pydata.org/

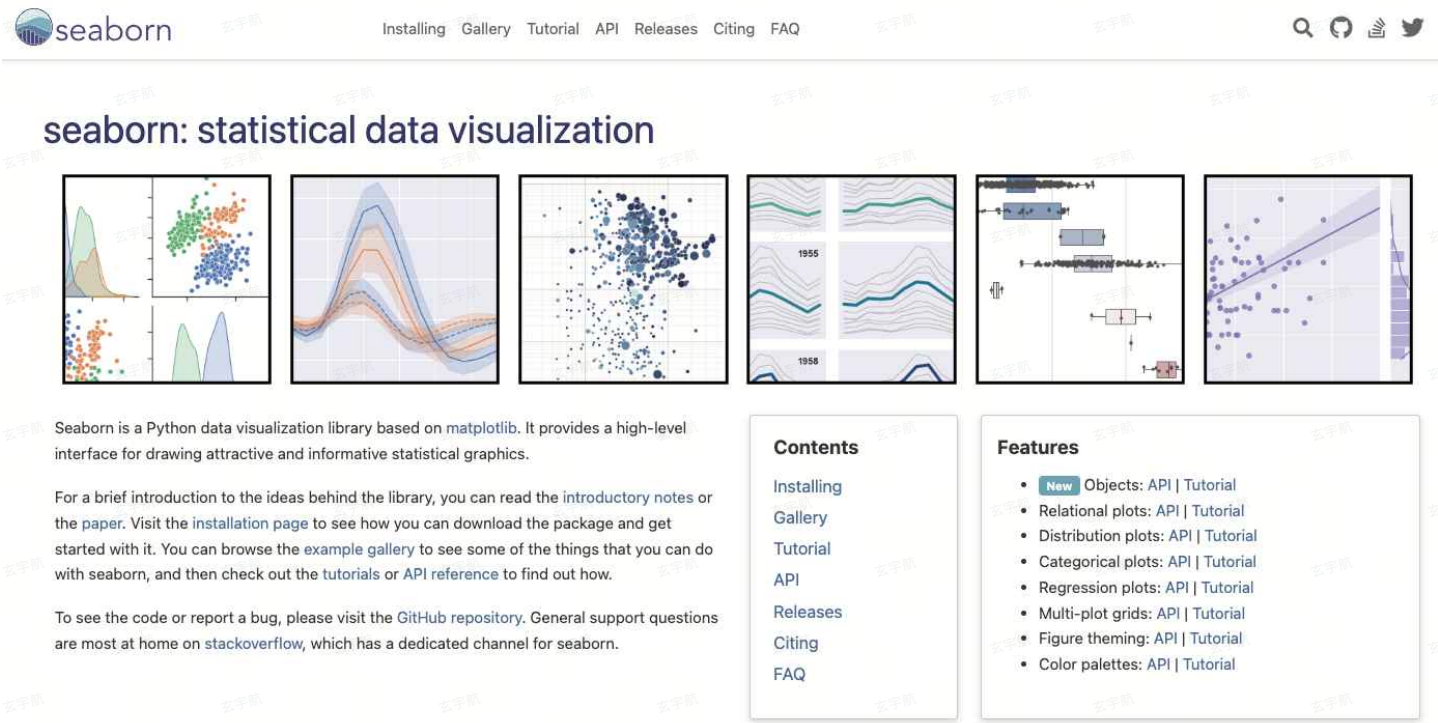
seaborn: statistical data visualization — seaborn 0.13.2 documentation



知乎 https://www.zhihu.com/tardis/bd/art/81553421?source_id=1001

数据可视化，Seaborn画图原来这么好看

数据可视化，Seaborn画图原来这么好看 1927 赞同 38 评论 8119 收藏 matplotlib是python最常见的绘图包，强大之处不言而喻。然而在数据科学领域，可视化库-···



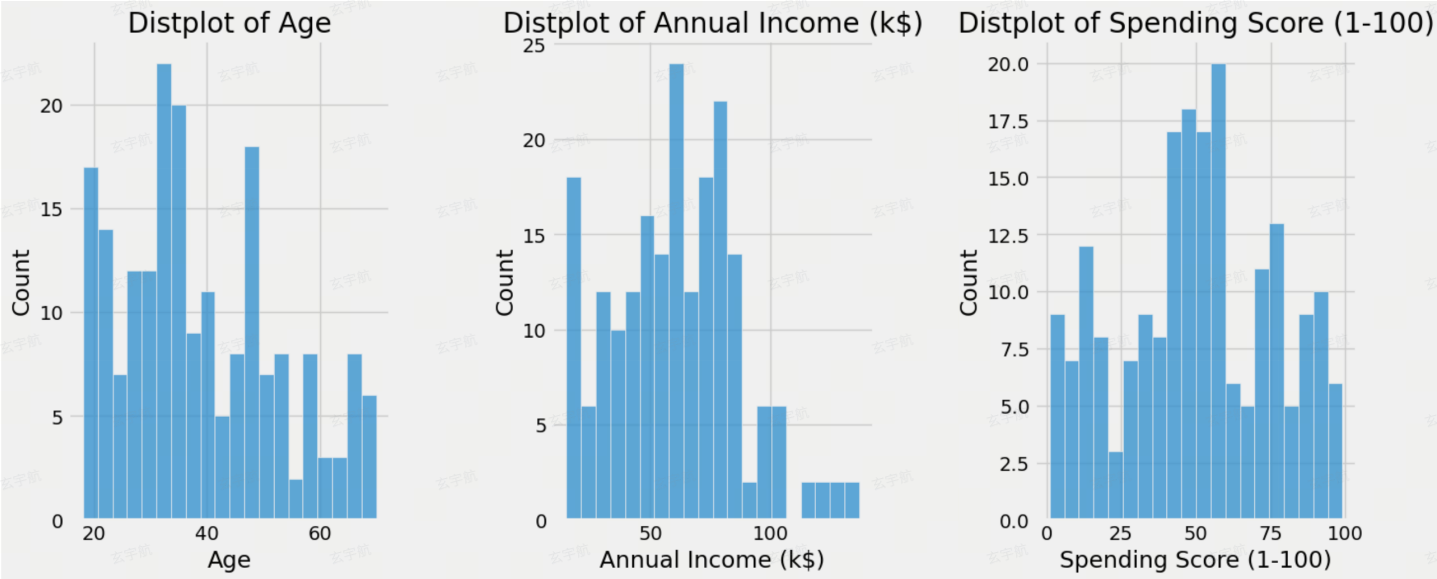
1.2.1 统计直方图1

绘制数据的统计直方图，使用下方代码对其中三个主要属性的统计直方图，包含年龄、收入、消费指数。此处涉及matplotlib的FiveThirtyEight模式、多子图绘制，以及seaborn的分布直方图显示函数histplot()

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('fivethirtyeight')
plt.figure(1 , figsize = (15 , 6))
n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1
    plt.subplot(1 , 3 , n)
    plt.subplots_adjust(hspace =0.5 , wspace = 0.5)
```

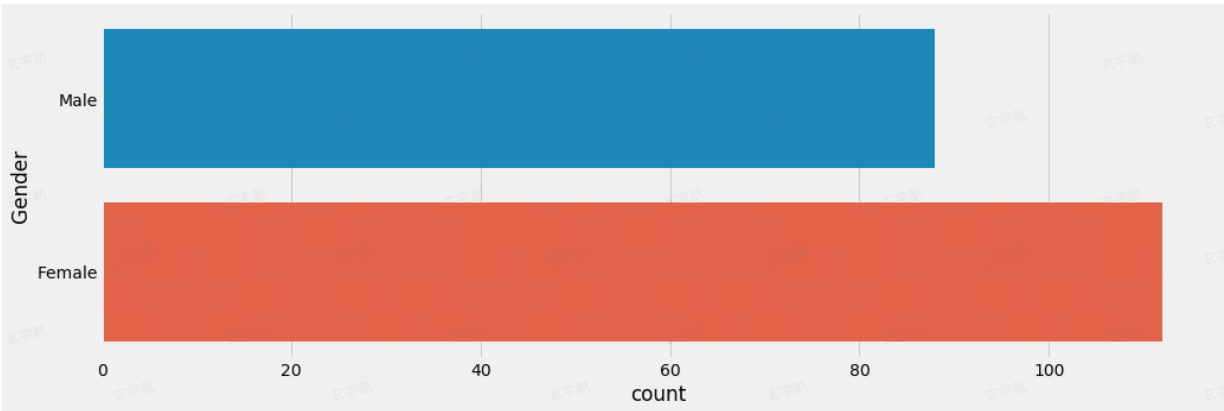
```
    sns.histplot(df[x] , bins = 20)
    plt.title('Distplot of {}'.format(x))
plt.show()
```



1.2.2 统计直方图2

使用seaborn的计数直方图绘制男、女样本数量分布。

```
plt.figure(1 , figsize = (15 , 5))
sns.countplot(y = 'Gender' , data = df)
plt.show()
```



1.2.3 展示属性间的相关性

通过多子图绘制、散点图和回归模型（seaborn的regplot()）展现主要属性（年龄、年收入、消费指数）间的线性相关程度。

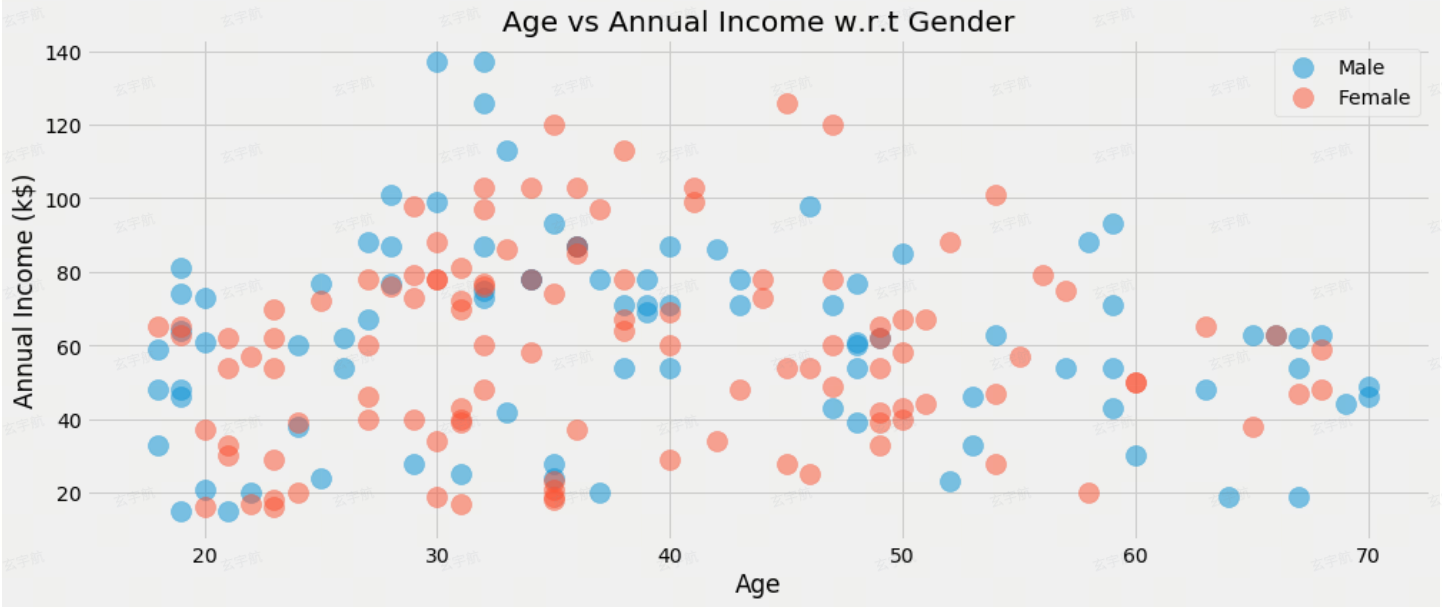
```
plt.figure(1 , figsize = (15 , 7))
n = 0
```

```
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    for y in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
        n += 1
        plt.subplot(3 , 3 , n)
        plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
        sns.regplot(x = x , y = y , data = df)
        plt.ylabel(y.split()[0]+' '+y.split()[1] if len(y.split()) > 1 else y )
plt.show()
```

1.2.4 指定属性显示分类数据

在上述选定两属性进行样本点显示基础上，可以使用不同风格显示不同类别的数据。比如在“年龄”和“年收入”坐标系下按性别显示相关用户。

```
plt.figure(1 , figsize = (15 , 6))
for gender in ['Male' , 'Female']:
    plt.scatter(x = 'Age' , y = 'Annual Income (k$)' ,
                data = df[df['Gender'] == gender] ,
                s = 200 , alpha = 0.5 , label = gender)
plt.xlabel('Age')
plt.ylabel('Annual Income (k$)')
plt.title('Age vs Annual Income w.r.t Gender')
plt.legend()
plt.show()
```

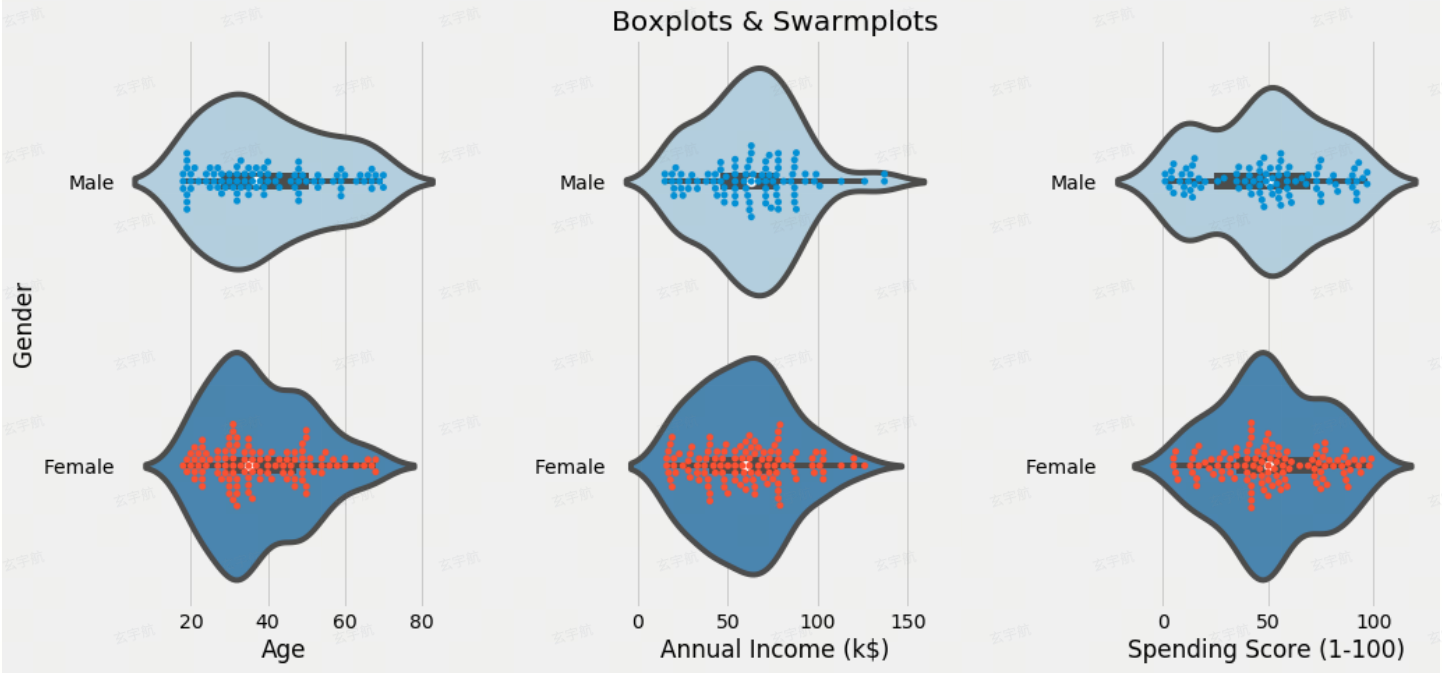


1.2.5 使用小提琴风格的数据分布可视化

数据分布也可以指定部分数据（如gender = male or female）下其他属性的取值分布。涉及到seaborn的violinplot()和swarmplot()

```
plt.figure(1 , figsize = (15 , 7))
n = 0

for cols in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1
    plt.subplot(1 , 3 , n)
    plt.subplots_adjust(hspace = 0.5 , wspace = 0.5)
    sns.violinplot(x = cols , y = 'Gender' , data = df, palette='Blues')
    sns.swarmplot(x = cols , y = 'Gender' , data = df)
    plt.ylabel('Gender' if n == 1 else '')
    plt.title('Boxplots & Swarmplots' if n == 2 else '')
plt.show()
```



2. k均值聚类（k-means clustering）

k-means聚类算法是应用最为广泛最容易被理解的聚类算法，它通过数据相似程度将m个样本分成k个簇，期望达到的目标是：每个样本和同簇样本更为接近，而非同簇样本相距更远。具体的算法在下图中给出。

Algorithm 1: k-means（k 均值）聚类

Input: 无标注数据集 D，聚簇个数 k，最大迭代次数 T

Output: k 个簇：$C_1, C_2, \cdots, C_k$，$C_1 \cup C_2 \cup \cdots \cup C_k = D$

随机初始化 k 个簇中心 $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \cdots, \boldsymbol{\mu}_k$
for $t = 1, \cdots, T$ **do**
  for $i = 1, \cdots, m$ **do**
    $c^{(i)} = \arg\min_{j \in \{1,2,\cdots,k\}} dist(\boldsymbol{x}^{(i)}, \boldsymbol{\mu}_j)$
  end
  for $j = 1, \cdots, k$ **do**
    $\boldsymbol{\mu}_j = \frac{1}{|C_i|} \sum_{\boldsymbol{x} \in C_i} \boldsymbol{x}$
  end
  if k 个簇的均值 $\boldsymbol{\mu}_j$ 均不发生变化 **then**
    break
  end
end

2.1 实现k-means聚类算法

请根据上面的算法和脚手架代码完成该算法，主要用于计算每个样本到k个均值间的距离。

```python
import numpy as np

def kmeans(X, K, max_iters=100):
    # 随机初始化质心
    centroids = X[np.random.choice(range(X.shape[0]), K, replace=False)]

    for t in range(max_iters):
        # 一个临时的距离矩阵distances，用于存储每个数据点与每个质心之间的距离,规模为X.shape[0]*K。
        #distances第i行第j列表示第i个数据点到第j个质心的距离，计算两个点x1,x2之间的距离距离可以使用np.linalg.norm（x1-x2）
        '''

        code here

        '''

        # 分配每个数据点到最近的质心
        labels = np.argmin(distances, axis=1)

        # 更新质心位置
        new_means = np.array([X[labels == k].mean(axis=0) for k in range(K)])

        # 如果质心位置不再改变，停止迭代
        if np.all(means == new_means):
            break

        means = new_means

    # 计算各样本到所属簇中心的距离和
    # inertia = np.sum('''distances''')
    return labels, means, inertia

# 示例用法
X = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])
K = 2
labels, means = kmeans(X, K)
print("聚类结果：", labels)
print("质心位置：", means)
```

2.2 使用k-means对客户数据进行聚类

从mall_consumers数据集中选取“年龄”和“消费指数”两列，分别使用刚实现的k-means()和sklearn.cluster.KMeans()对上述子集进行聚类，并针对不同K取值，展现出聚类评价指标的变化。

其中，inertia（惯性）是指数据点与其最近聚类中心之间的距离的加权总和。它值越小越好，表示样本在类间的分布越集中。这也可看作k-means算法的优化目标。即：

$$L(kMeans, D) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in C_i} dist(\boldsymbol{x}, \boldsymbol{\mu}_i)$$

2.2.1 使用刚实现的k-means算法进行聚类

```
'''Age and spending Score'''
X1 = df[['Age' , 'Spending Score (1-100)']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    labels, means, inertia = kmeans(X, K, 300)
    inertia.append(algorithm.inertia_)
```

2.2.2 使用sklearn.cluster.KMeans算法进行聚类

关于函数参数的说明：

init--质心的初始化方式， ‘random’

n_init--用不同的质心初始化值运行算法的次数，最终解是在inertia意义下选出的最优结果

tol--迭代停止

random_state--用于初始化质心的生成器（generator）

algorithm--“auto”，“full” or “elkan”三种选择。“full”就是我们传统的K-Means算法，“elkan”是我们原理篇讲的elkan K-Means算法。默认的”auto”则会根据数据值是否是稀疏的，来决定如何选择”full”和“elkan”

```
from sklearn.cluster import KMeans
'''Age and spending Score'''
X1 = df[['Age' , 'Spending Score (1-100)']].iloc[: , :].values
inertia = []
for n in range(1 , 11):
    algorithm = (KMeans(n_clusters = n ,init='k-means++', n_init = 10
,max_iter=300,
                        tol=0.0001,  random_state= 111  , algorithm='elkan') )
    algorithm.fit(X1)
    inertia.append(algorithm.inertia_)
```

2.2.3 可视化k-means算法的优化目标与参数k之间的关系

观察10次聚类的inertias，并以如下折线图进行统计。其中，inertias值越小越好，越小表示样本在类间的分布越集中。

```
plt.figure(1 , figsize = (15 ,6))
plt.plot(np.arange(1 , 11) , inertia , 'o')
plt.plot(np.arange(1 , 11) , inertia , '-' , alpha = 0.5)
plt.xlabel('Number of Clusters') , plt.ylabel('Inertia')
plt.show()
```

2.2.4 可视化聚类结果

该图可以进一步反映上一节中训练得到的线性回归模型无法很好地拟合训练数据，具体体现为训练误差（error_train）和验证误差（error_val）都比较高。尤其是作为基本条件的训练误差很高（high bias），而此种情况反映的是模型过于简单导致线性模型无法拟合非线性数据集。需要增强模型的描述能力，即将线性回归扩展为可以拟合非线性数据的多项式回归。

```
algorithm = (KMeans(n_clusters = 4 ,init='k-means++', n_init = 10
,max_iter=300,
                    tol=0.0001,  random_state= 111  , algorithm='elkan') )
algorithm.fit(X1)
labels1 = algorithm.labels_
centroids1 = algorithm.cluster_centers_
```

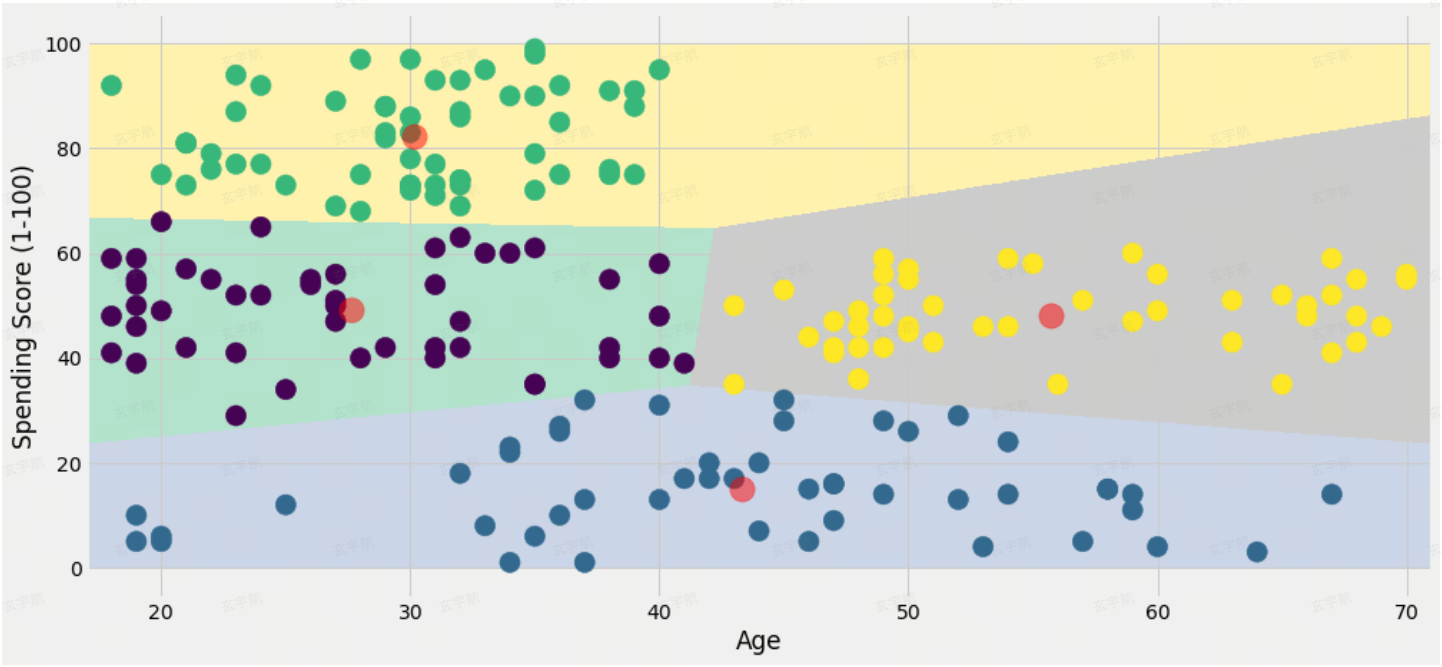
把4个聚类中心的聚类结果在下图进行展示。横坐标是年龄，纵坐标是消费指数，4个红点为4个聚类中心，4块不同颜色区域就是4个不同的用户群体。

```
h = 0.02
x_min, x_max = X1[:, 0].min() - 1, X1[:, 0].max() + 1
y_min, y_max = X1[:, 1].min() - 1, X1[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])

plt.figure(1 , figsize = (15 , 7) )
plt.clf()
Z = Z.reshape(xx.shape)
plt.imshow(Z , interpolation='nearest',
           extent=(xx.min(), xx.max(), yy.min(), yy.max()),
           cmap = plt.cm.Pastel2, aspect = 'auto', origin='lower')

plt.scatter( x = 'Age' ,y = 'Spending Score (1-100)' , data = df , c = labels1
,
            s = 200 )
plt.scatter(x = centroids1[: , 0] , y =  centroids1[: , 1] , s = 300 , c =
'red' , alpha = 0.5)
plt.ylabel('Spending Score (1-100)') , plt.xlabel('Age')
plt.show()
```

运行结果显示如下：



2.2.5 聚类及可视化2

请仿照上一节的代码，完成一下工作

1. 从数据集中选择“年收入”和“消费指数”两列对用户进行聚类（使用sklearn.cluster.KMeans()），在簇中心个数k的选择上依然使用1到10；
2. 观察10次聚类的代价函数值inertias，并使用折线图进行展示；

3. 设定聚簇个数k=5，再次使用sklearn.cluster.KMeans()进行聚类，并仿照上一节对数据结果进行可视化。