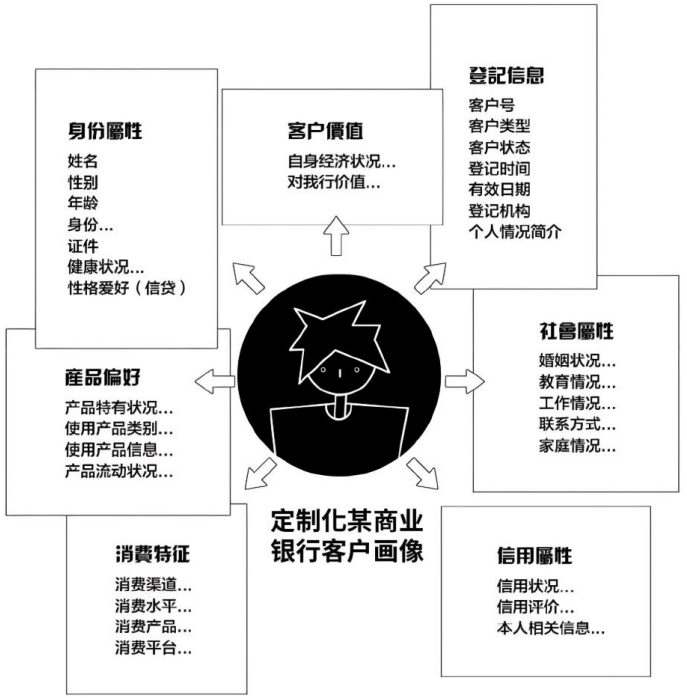
Python与金融数据挖掘(14)

文欣秀

wenxinxiu@ecust.edu.cn

机器学习分类

分类：对事物所属类别判断，类型数量已知。如**判断银行客户类型**、判断垃圾邮件、手写数字识别等。



K折交叉验证

KFold(n_splits=10，shuffle=False，random_state=None)

n_splits : 整数，表示交叉验证的折数（即将数据集分为几份）。

shuffle : 布尔值, 表示是否要将数据打乱顺序后再进行划分，若为True时，每次划分的结果都不一样。

random_state : 默认为None。 当shuffle为True时, random_state的值影响标签的顺序。

贝叶斯算法

贝叶斯分类算法：是统计学的一种分类方法，它是一类利用概率统计知识进行分类的算法。朴素贝叶斯（Naive Bayes，**NB**）是基于贝叶斯定理与特征条件独立假设的分类方法，可用于**垃圾邮件分类**、**客户信用评估**等问题。

$$P(A \mid B) = \frac{P(B \mid A) * P(A)}{P(B)} = \frac{P(A \cap B)}{P(B)}$$

机器学习分类

常用分类算法

- K近邻算法(K-Nearest Neighbor，**KNN**)
- 贝叶斯算法(Naive Bayes，**NB**)
- 支持向量机(Support Vector Machine，**SVM**)
- 决策树(Decision tree，**DT**)
- 逻辑回归(Logistic Regression，**LR**)

案例分析

	A	B	C	D	E	F
1	收入	年龄	性别	历史授信额度	历史违约次数	是否违约
2	503999	46	1	0	1	1
3	452766	36	0	13583	0	1
4	100000	33	1	0	1	1
5	100000	25	0	0	1	1
6	258000	35	1	0	0	1
7	933333	31	0	28000	3	1
8	665000	40	1	5000	1	1
9	291332	38	0	0	0	1
10	259000	45	1	0	1	1
11	3076666	39	1	71000	2	1
12	695000	40	1	5000	0	1
13	600000	35	1	18000	3	1
14	440000	36	1	0	2	1

案例分析（一）

```
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
```

案例分析（二）

```
# 导入数据
dataset = pd. read_excel('客户信息及违约表现.xlsx')
array = dataset.values # 分离数据集
X = array[:, 0:5]
Y = array[:, 5]
seed = 7
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
```

案例分析（三）

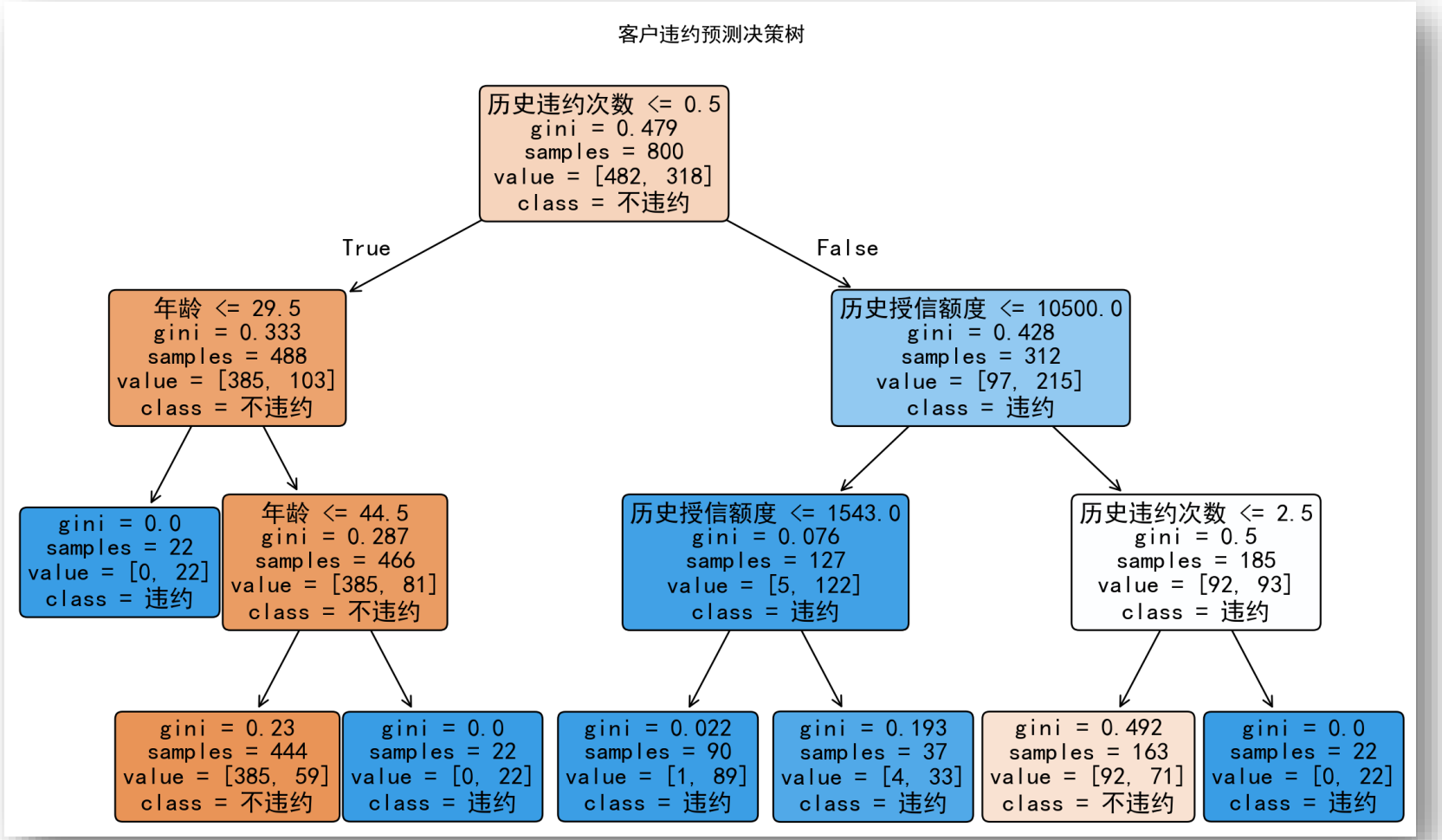
```
# 算法审查
models = {}
models['LR'] = LogisticRegression(max_iter=10000)
models['DT'] = DecisionTreeClassifier()
models['KNN'] = KNeighborsClassifier()
models['NB'] = GaussianNB()
models['SVM'] = SVC()
```

案例分析（四）

```
# 评估算法
for key in models:
    #cross_val_score:得到K折验证中每一折的得分
    cv_results = cross_val_score(models[key], X, Y, cv=kfold)
    print(cv_results)
    print('%s: %f (%f)' %(key, cv_results.mean(), cv_results.std()))
```

```
LR: 0.751000 (0.042767)
DT: 0.781000 (0.027731)
KNN: 0.596000 (0.070880)
NB: 0.595000 (0.065154)
SVM: 0.601000 (0.068037)
```

决策树基本算法


客户违约预测决策树
历史违约次数 <= 0.5
gini = 0.479
samples = 800
value = [482, 318]
class = 不违约
True
False
年龄 <= 29.5
gini = 0.333
samples = 488
value = [385, 103]
class = 不违约
历史授信额度 <= 10500.0
gini = 0.428
samples = 312
value = [97, 215]
class = 违约
gini = 0.0
samples = 22
value = [0, 22]
class = 违约
年龄 <= 44.5
gini = 0.287
samples = 466
value = [385, 81]
class = 不违约
历史授信额度 <= 1543.0
gini = 0.076
samples = 127
value = [5, 122]
class = 违约
历史违约次数 <= 2.5
gini = 0.5
samples = 185
value = [92, 93]
class = 违约
gini = 0.23
samples = 444
value = [385, 59]
class = 不违约
gini = 0.0
samples = 22
value = [0, 22]
class = 违约
gini = 0.022
samples = 90
value = [1, 89]
class = 违约
gini = 0.193
samples = 37
value = [4, 33]
class = 违约
gini = 0.492
samples = 163
value = [92, 71]
class = 不违约
gini = 0.0
samples = 22
value = [0, 22]
class = 违约

决策树基本算法

- 按照自顶向下递归划分的方法构造决策树
- 起初所有的训练样本均在根结点
- 所有属性都是分类的 (若连续，则先进行离散化处理)
- 样本基于被选择的属性被递归划分
- 测试属性选择基于启发式规则或者统计度量（如信息增益)

CART算法：基尼系数

基尼系数： 基尼系数代表了模型的不纯度，基尼系数越小，则不纯度越低，特征越好。

$$Gini(p) = \sum_{i=1}^{k} p_i(1-p_i) = 1 - \sum_{i=1}^{k} p_i^2$$

案例分析

此时 k=1，数据纯度高，则： $Gini(D) = 1 - \sum_{i=1}^{1} \left(\frac{|C_i|}{|D|}\right)^2 = 1 - (1)^2 = 0$

此时 k=3，数据纯度不太高，则： $Gini(D) = 1 - \sum_{i=1}^{3} \left(\frac{|C_i|}{|D|}\right)^2 = 1 - \left(\left(\frac{3}{6}\right)^2 + \left(\frac{2}{6}\right)^2 + \left(\frac{1}{6}\right)^2\right) = 0.61$

此时 k=5，数据纯度极低，则： $Gini(D) = 1 - \sum_{i=1}^{5} \left(\frac{|C_i|}{|D|}\right)^2 = 1 - \left(\left(\frac{2}{6}\right)^2 + \left(\frac{1}{6}\right)^2 + \left(\frac{1}{6}\right)^2 + \left(\frac{1}{6}\right)^2 + \left(\frac{1}{6}\right)^2\right) = 0.78$

划分为训练集和测试集

train_test_split()函数： 将数据集按需求划分为训练集和测试集

X_train,X_test, y_train, y_test =

train_test_split(X, y, test_size, random_state)

X、y: 待划分的样本特征集、样本结果

X_train、y_train: 划分出的训练数据集数据、标签

test_size： 测试集占原始样本比例或测试集的数目

random_state： 随机数种子

案例分析（五）

```
#选择DT算法预测
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
X_train, X_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2, random_state=42)
```

案例分析（六）

```
clf = DecisionTreeClassifier() # 创建决策树分类器
clf. fit(X_train, y_train)
y_pred = clf. predict(X_test)
accuracy = accuracy_score(y_test, y_pred) # 评估模型准确率
print(f"模型准确率: {accuracy}")
print("分类报告:")
print(classification_report(y_test, y_pred))
```

```
模型准确率: 0.715
分类报告:
              precision    recall  f1-score   support

           0       0.77      0.74      0.76       119
           1       0.64      0.68      0.66        81

    accuracy                           0.71       200
   macro avg       0.71      0.71      0.71       200
weighted avg       0.72      0.71      0.72       200
```

评价指标

分类常用的评价指标：混淆矩阵 (Confusion Matrix) 、精确率 (Precision)、召回率 (Recall)、F1分数 (F1 Score)和准确率 (**Accuracy**)等。

混淆矩阵

混淆矩阵（误差矩阵）： 是表示精度评价的一种标准格式，用n行n列的矩阵形式来表示。

真实结果	预测结果	
	正例	反例
正例	(True Positive , TP) (预测为正，真实为正)	(False Negative, FN) (预测为负，真实为正)
反例	FP(False Positive , FP) (预测为正，真实为负)	(True Negative , TN) (预测为负，真实为负)

分类评价指标

精确率（Precision）又叫查准率：它是针对预测结果而言的，它的含义是在**所有被预测为正的样本中实际为正的样本比例**，其计算公式为：

$$P=\frac{TP}{TP+FP}$$

分类评价指标

召回率（Recall）又叫查全率：它是针对原样本而言的，它的含义是在**实际为正的样本中被预测为正样本的**比例，其公式如下：

$$R=\frac{TP}{TP+FN}$$

分类评价指标

F1分数（F1 Score）：是一个**综合精确率和召回率**的评价指标，当模型的精确率和召回率冲突时，可以采用该指标来衡量模型的优劣，其计算公式为：

$$F1 = \frac{2 \cdot P \cdot R}{P + R}$$

分类评价指标

准确率（Accuracy）：是分类问题中最为常用的评价指标，准确率的定义是**预测正确的样本数占总样本数的比例**，其计算公式为：

$$ACC=\frac{TP+TN}{TP+TN+FP+FN}$$

案例分析

y_true	spam	spam	spam	spam	spam	ham	ham	ham	ham	ham
y_pred	spam	spam	spam	spam	ham	spam	ham	ham	spam	ham

其中：y_true代表样本的真实值，**y_pred**代表样本的模型预测值

假设：以垃圾邮件为正，非垃圾邮件为负

结果：TP=4，FN=1，FP=2，TN=3。

案例分析

通过上述公式，可以计算出相关评价指标：

$$P=\frac{4}{4+2}\approx 0.67$$

$$R=\frac{4}{4+1}=0.8$$

$$F1=\frac{2\cdot P\cdot R}{P+R}\approx 0.73$$

$$ACC=\frac{4+3}{4+3+1+2}=0.7$$

标签二值化

二值化：指将数值特征向量转换为布尔类型向量。通过人为设定阈值，将大于阈值的数值映射为1，而小于或等于阈值的数值映射为0。

fit_transform()函数：用于把转换器实例应用到数据上，并返回转换后的数据。

标签二值化

标签二值化：可以把**非数字化**的数据标签转化为**数字化形式**的数据标签，例如可把“Yes”和“No”等文本标签转化为“1”和“0”的数字形式。

scikit-learn库中preprocessing.**LabelBinarizer类**实现了标签二值化处理的功能，常用于文本类型的数据标签的处理。

标签二值化案例

构建数据集，进行标签二值化处理并打印显示。

```
from sklearn import preprocessing
label = ['Yes', 'No', 'Yes', 'No', 'No'] # 设置数据集
lb = preprocessing. LabelBinarizer()    # 转换器实例化
label_bin = lb. fit_transform(label)  # 标签数据二值化
print(label_bin)
```

```
[[1]
 [0]
 [1]
 [0]
 [0]]
>>>
```

案例分析

```
from sklearn import metrics    #评估指标
from sklearn.preprocessing import LabelBinarizer   #标签二值化
lb = LabelBinarizer()
y_true = ['spam', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'ham', 'ham', 'ham']
y_pred = ['spam', 'spam', 'spam', 'spam', 'ham', 'spam', 'ham', 'ham', 'spam', 'ham']
# （1）计算混淆矩阵
print('Confusion Matrix：')
print(metrics. confusion_matrix(y_true, y_pred, labels=['spam', 'ham']))
```

```
Confusion Matrix:
[[4 1]
 [2 3]]
```

案例分析

```
# （2）将标签二值化，计算精确率、召回率、F1分数和准确率
y_true_binarized = lb. fit_transform(y_true)
y_pred_binarized = lb. fit_transform(y_pred)
print('精确率：  %s' % metrics. precision_score(y_true_binarized, y_pred_binarized))
print('召回率：  %s' % metrics. recall_score(y_true_binarized, y_pred_binarized))
print('F1分数：%s' % metrics. f1_score(y_true_binarized, y_pred_binarized))
print('准确率：  %s' % metrics. accuracy_score(y_true_binarized, y_pred_binarized))
```

```
精确率：  0.6666666666666666
召回率：  0.8
F1分数：0.7272727272727273
准确率：  0.7
```

案例分析

```
# （3）classification_report()函数实现对精确率、召回率、F1分数和准确率的计算
print('Classification Report： ')
print(metrics.classification_report(y_true, y_pred))
```

```
Confusion Matrix:
[[4 1]
 [2 3]]
精确率： 0.6666666666666666
召回率： 0.8
F1分数: 0.7272727272727272
准确率： 0.7
Classification Report:
              precision    recall  f1-score   support

         ham       0.75      0.60      0.67         5
        spam       0.67      0.80      0.73         5

    accuracy                           0.70        10
   macro avg       0.71      0.70      0.70        10
weighted avg       0.71      0.70      0.70        10
```

scikit-learn

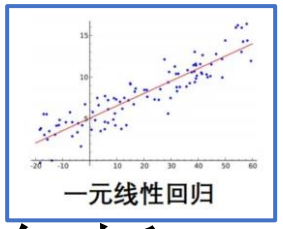
监督学习： 指数据中包括了想要预测的属性。

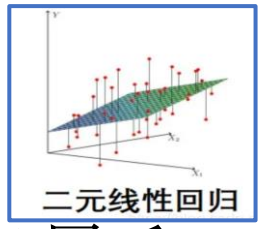
分类： 对事物所属类别判断，类型数量已知。如判别鸟的种类、判断垃圾邮件等。

回归： 预测目标是连续变量，如根据父母身高预测孩子身高，**根据企业财务指标预测收益**等。

线性回归

线性回归（Linear Regression）： 利用数理统计中回归分析来确定**两种**或**两种以上变量**间相互依赖的定量关系的一种统计分析方法，运用十分广泛。

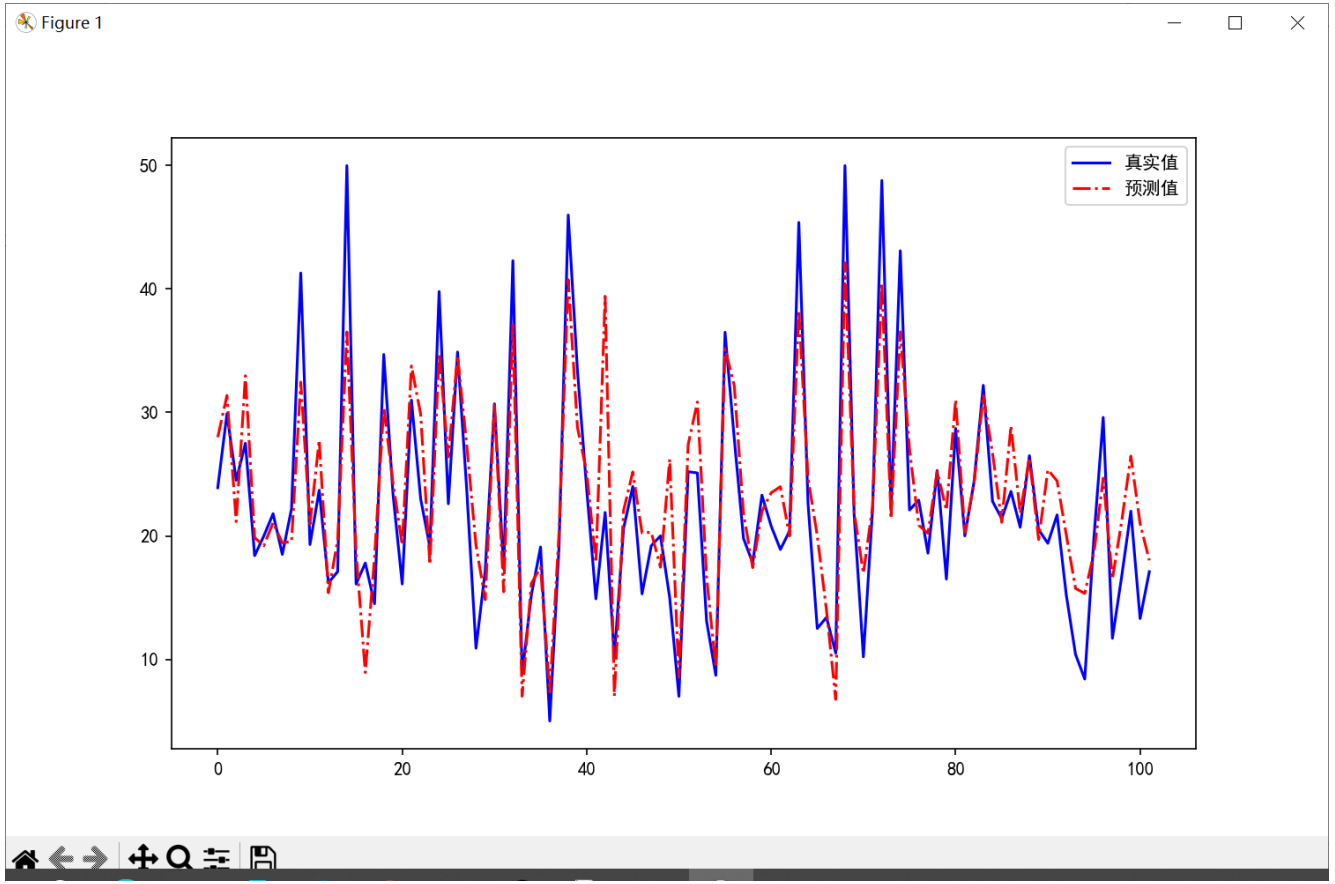

一元线性回归


二元线性回归

一元线性回归分析： 分析中只包括一个自变量和一个因变量。

多元线性回归分析： 分析中包括两个或两个以上自变量，且因变量和自变量之间是线性关系。

波士顿房价问题



波士顿房价

sklearn提供的波士顿房价数据集统计20世纪70年代中期**波士顿郊区房价**。该数据集包含**506**条记录，**13**个特征指标，第**14列**通常为目标列房价。试图能找到特征指标与房价的关系。

波士顿房价

本例首先将506组数据的数据集划分为训练集和测试集，其中**404组数据是训练样本**，剩下的**102组数据作为验证样本**。然后构建回归模型并训练模型，查看模型的**13个特征的系数以及截距**，获取模型的预测结果，最后绘制折线图对比预测值和真实。

数据的归一化

归一化：把每列数据都映射到**0-1**范围之内处理。scikit-learn库提供**preprocessing. MinMaxScaler**类实现了将数据缩放到一个指定的最大值和最小值（通常是1-0）之间的功能。

波士顿房价数据归一化

对boston数据集的前5行5列数据进行归一化并打印显示。

```
import pandas as pd
from sklearn. preprocessing import MinMaxScaler
boston=pd. read_csv("boston.csv", encoding="gb2312")
X = boston. iloc[:5, :5]
minmax_scaler = MinMaxScaler()                    # 转换器实例化
boston_minmax = minmax_scaler. fit_transform(X)   # 数据归一化
print(boston_minmax)
```

波士顿房价数据归一化结果

	A	B	C	D	E
1	犯罪率	区	产业	查尔斯河	氧化氮
2	0.00632	18	2.31	0	0.538
3	0.02731	0	7.07	0	0.469
4	0.02729	0	7.07	0	0.469
5	0.03237	0	2.18	0	0.458
6	0.06905	0	2.18	0	0.458

```
[[0.          1.          0.02658487  0.          1.        ]
 [0.33460864  0.          1.          0.          0.1375    ]
 [0.33428981  0.          1.          0.          0.1375    ]
 [0.4152718   0.          0.          0.          0.        ]
 [1.          0.          0.          0.          0.        ]]
>>>
```

数据的标准化

数据标准化： scikit-learn库提供了对数据进行标准化处理的函数，包括Z-score标准化、稀疏数据标准化和带离群值的标准化。

Z-Score标准化： scikit-learn库中preprocessing. StandardScaler类实现了Z-Score标准化。

Z-Score公式： **z = (x-平均值) / 标准差**

波士顿房价数据标准化

对boston数据集的前5行5列数据进行标准化并打印显示。

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
boston=pd. read_csv("boston.csv", encoding="gb2312")
X = boston.iloc[:5, :5]
standerd_scaler = StandardScaler()                    # 转换器实例化
boston_standerd = standerd_scaler. fit_transform(X)   # 数据标准化
print(boston_standerd)
```

波士顿房价数据标准化结果

	A	B	C	D	E
1	犯罪率	区	产业	查尔斯河	氧化氮
2	0.00632	18	2.31	0	0.538
3	0.02731	0	7.07	0	0.469
4	0.02729	0	7.07	0	0.469
5	0.03237	0	2.18	0	0.458
6	0.06905	0	2.18	0	0.458

```
[[-1.2834352    2.         -0.77983987  0.          1.97329359]
 [-0.25317266  -0.5         1.22450018  0.         -0.31122416]
 [-0.25415433  -0.5         1.22450018  0.         -0.31122416]
 [-0.00481018  -0.5        -0.83458025  0.         -0.67542264]
 [ 1.79557237  -0.5        -0.83458025  0.         -0.67542264]]
>>>
```

数据的正则化

数据正则化：scikit-learn库提供了对数据进行正则化处理的函数，其中preprocessing. Normalizer类实现了将单个样本缩放到单位范数的功能。

正则化应用：在数据集之间各个指标有共同重要比率的关系时，正则化处理有比较好的效果。

波士顿房价数据正则化

对boston数据集的前5行5列数据进行正则化并打印显示。

```
import pandas as pd
from sklearn.preprocessing import Normalizer
boston=pd. read_csv("boston.csv", encoding="gb2312")
X = boston. iloc[:5, :5]
normalizer_scaler = Normalizer()                    # 转换器实例化
boston_normalizer = normalizer_scaler. fit_transform(X) # 数据正则化
print(boston_normalizer)
```

波士顿房价数据正则化结果

	A	B	C	D	E
1	犯罪率	区	产业	查尔斯河	氧化氮
2	0.00632	18	2.31	0	0.538
3	0.02731	0	7.07	0	0.469
4	0.02729	0	7.07	0	0.469
5	0.03237	0	2.18	0	0.458
6	0.06905	0	2.18	0	0.458

```
[[3.48102083e-04 9.91429984e-01 1.27233515e-01 0.00000000e+00
  2.96327406e-02]
 [3.85430066e-03 0.00000000e+00 9.97799549e-01 0.00000000e+00
  6.61906632e-02]
 [3.85147808e-03 0.00000000e+00 9.97799560e-01 0.00000000e+00
  6.61906639e-02]
 [1.45298555e-02 0.00000000e+00 9.78532126e-01 0.00000000e+00
  2.05581520e-01]
 [3.09827227e-02 0.00000000e+00 9.78165612e-01 0.00000000e+00
  2.05504518e-01]]
>>>
```

Python实现线性回归步骤

- 导入对应库
- 加载数据集并划分数据集
- 在训练集上训练线性回归模型
- 使用测试集实现预测
- 绘图输出，结果可视化对比

波士顿房价回归模型

```
# （1）导入库
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt
import pandas as pd
```

波士顿房价回归模型

```
# （2）加载数据集
boston=pd. read_csv("boston.csv",encoding="gb2312")
X=boston. iloc[:,0:13]. values
y=boston. iloc[:,13]. values
# 分割数据为训练集和测试集
x_train,x_test,y_train,y_test=train_test_split(X,y,
                                test_size=0.2,random_state=22)
print('x_train前3行数据为：', x_train[0:3])
print('y_train前3行数据为：',y_train[0:3])
```

波士顿房价回归模型

```
# （3）创建线性回归模型对象
lr=LinearRegression()
#使用训练集训练模型
lr.fit(x_train,y_train)
#显示模型
print(lr)
print("13个系数:",lr.coef_)
print("模型截距:",lr.intercept_ )
# （4）使用测试集获取预测结果
print("预测结果:",lr.predict(x_test[:5]))
```

```
LinearRegression()
```

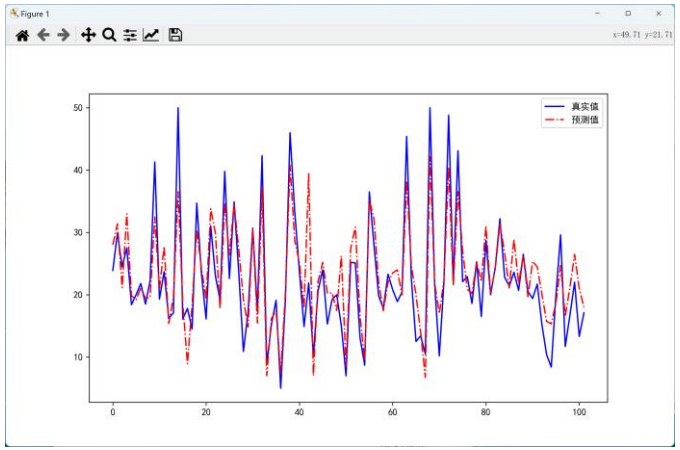
```
13个系数: [-1.01199845e-01  4.67962110e-02 -2.06902678e-02  3.58072311e+00
 -1.71288922e+01  3.92207267e+00 -5.67997339e-03 -1.54862273e+00
  2.97156958e-01 -1.00709587e-02 -7.78761318e-01  9.87125185e-03
 -5.25319199e-01]
```

```
模型截距: 32.428252866991 19
```

```
预测结果: [27.99617259 31.37458822 21.16274236 32.97684211 19.85350998]
```

波士顿房价回归模型

```
# （5）绘图对比预测值和真实值
plt.rcParams['font.sans-serif']='SimHei'
fig=plt.figure(figsize=(10,6))
y_pred=lr.predict(x_test)
plt.plot(range(y_test.shape[0]),y_test,color="blue",linestyle="-")
plt.plot(range(y_test.shape[0]),y_pred,color="red",linestyle="-.")
plt.legend(['真实值','预测值'])
plt.show()
```



评价指标

分类常用的评价指标：混淆矩阵 (Confusion Matrix) 、精确率 (Precision)、召回率 (Recall)、F1分数 (F1 Score)和准确率 (Accuracy)等。

回归主要评价指标：平均绝对误差(MAE，Mean absolute error)、均方误差(MSE，Mean squared error)、均方根误差(RMSE，Root Mean squared error)、R^2等。

案例分析

y_true	1	2	3
y_pred	1	3	5

y_true代表样本的真实值，**y_pred**代表该样本的模型预测值。

回归评价指标

MAE（平均绝对误差）：对于回归模型性能评估最直观的思路是**利用模型的预测值与真实值的差值来衡量**，误差越小，回归模型的拟合程度就越好，其计算公式为：

$$\mathrm{MAE}=\frac{1}{n}\sum_{i=1}^{n}\left|y_i-\hat{y}_i\right|$$

n 为样本的个数，y_i 为第 i 个样本的真实值，$\hat{y}_i$ 为第 i 个样本的模型预测值。

案例分析

通过上述公式以y_true和y_pred为例计算出相关**MAE**：

y_true	1	2	3
y_pred	1	3	5

$$\mathrm{MAE}=\frac{|1-1|+|2-3|+|3-5|}{3}=1$$

回归评价指标

MSE（均方误差）：它是一种常用的回归损失函数，计算方法是**求误差的平方和**，由这两个指标的原理可知MSE比MAE对异常值更敏感，其计算公式为：

$$\mathrm{MSE}=\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$$

RMSE（均方根误差）：对均方误差进行开平方运算。

案例分析

通过上述公式以y_true和y_pred为例计算出**MSE**：

y_true	1	2	3
y_pred	1	3	5

$$\mathrm{MSE}=\frac{(1-1)^2+(2-3)^2+(3-5)^2}{3}=\frac{5}{3}\approx 1.67$$

回归评价指标

决定系数R^2： 由MAE和MSE的公式可知，随着样本数量的增加，这两个指标也会随之增大，而且针对不同量纲的数据集，其计算结果也有差异，所以很难直接用这些评价指标来衡量模型的优劣，可以使用**决定系数R^2来评价回归模型的预测能力**。

回归评价指标

R^2计算公式为·

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

其中，$\bar{y}$ 表示 y 的均值。R^2 取值范围一般是 0～1，越接近 1，回归的拟合程度就越好。但当回归模型的拟合效果差于取平均值时的效果时，也可能为负数。

案例分析

通过上述公式以y_true和y_pred为例计算**R^2**：

y_true	1	2	3
y_pred	1	3	5

$$\overline{y}=\frac{1+2+3}{3}=2$$

$$R^2=1-\frac{(1-1)^2+(2-3)^2+(3-5)^2}{(1-2)^2+(2-2)^2+(3-2)^2}=1-\frac{5}{2}=-1.5$$

案例分析

```
from sklearn import metrics
y_true = [1, 2, 3]
y_pred = [1, 3, 5]
# （1）计算MAE
print('MAE：')
print('y_pred MAE：  %s' % metrics.mean_absolute_error(y_true, y_pred))
```

```
MAE：
y_pred MAE：  1.0
```

案例分析

```
# （2）计算MSE
print('MSE：')
print('y_pred MSE：  %s' % metrics.mean_squared_error(y_true, y_pred))

# （3）计算决定系数
print('R2：')
print('y_pred R2：  %s' % metrics.r2_score(y_true, y_pred))
```

```
MSE：
y_pred MSE：  1.6666666666666667
```

```
R2：
y_pred R2：  -1.5
```

机器学习分类

scikit-learn

无监督学习：指数据集中的数据是无类别标签，算法只根据数据集本身的数据特性来分析，即训练数据包括了输入向量X的集合，但没有相应的目标变量。

常见问题：数据降维(Dimension Reduction)、聚类问题(Clustering)

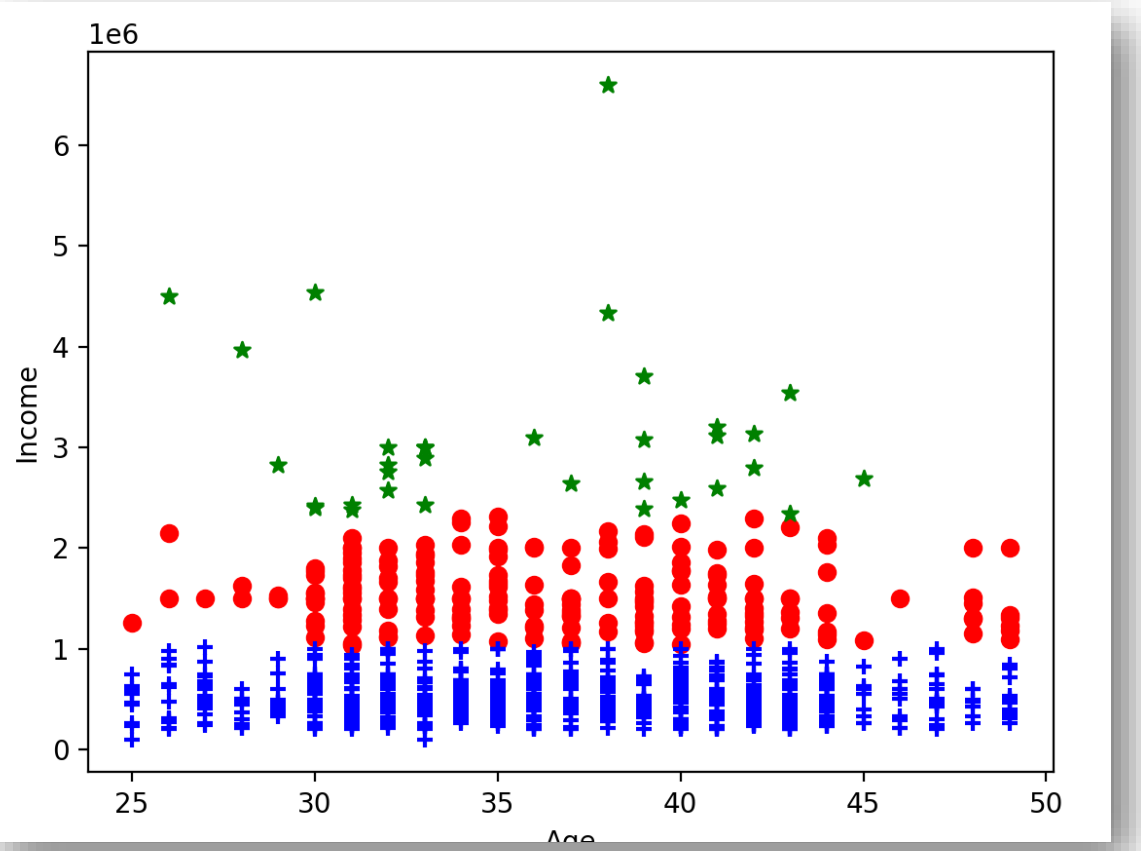
聚 类

聚类(Clustering Approach)：是按一定的距离或相似性系数将数据分成一系列相互区分的组，常用的经典聚类方法有K-means, DBSCAN，Mean Shift等。

聚类算法的应用场景：市场分析、商业经营、图像处理、决策支持、模式识别。

客户类型聚类分析

	A	B
1	收入	年龄
2	503999	46
3	452766	36
4	100000	33
5	100000	25
6	258000	35
7	933333	31
8	665000	40
9	291332	38
10	259000	45



K-Means聚类算法

- K-Means算法属于聚类分析中划分方法里较为**经典**的一种，由于该算法的效率高，所以在对大规模数据进行聚类时被广泛应用。
- K-Means算法通过**将样本划分k个簇类**来实现数据聚类，该算法需要指定划分类的个数。

K-Means聚类算法

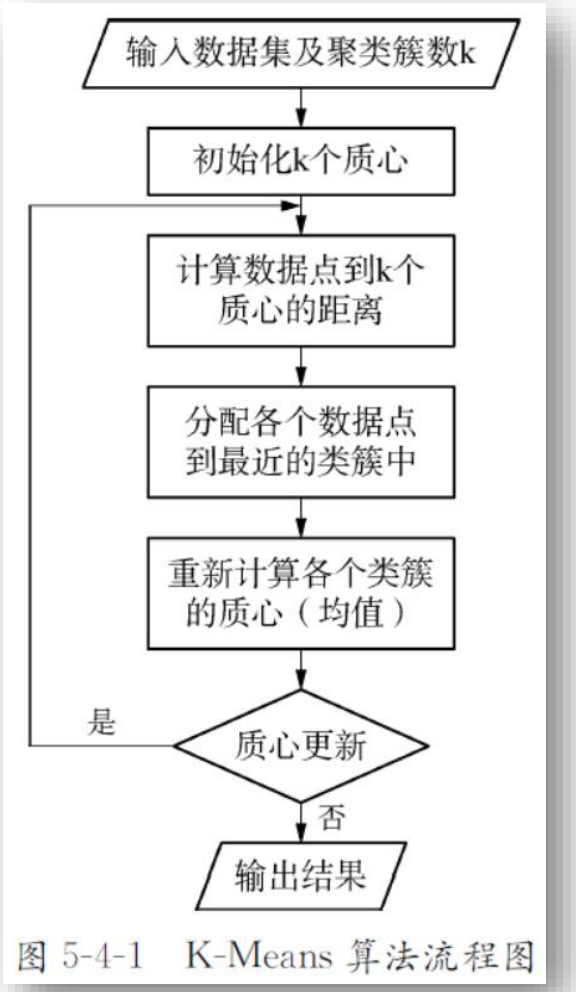


图 5-4-1　K-Means 算法流程图

K- Means算法示例

例： 对表中二维数据，使用k- means算法将其划分为2个簇，假设初始簇中心选为P7(4,5),P10(5,5)。

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	3	3	7	4	3	8	4	4	7	5
y	4	6	3	7	8	5	5	1	4	5

K- Means算法示例

- 根据题目,假设划分的两个簇分别为C1和C2,中心分别为(4,5)和(5,5),下面计算10个样本到这2个簇中心的距离,并将10个样本指派到与其最近的簇;

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	3	3	7	4	3	8	4	4	7	5
y	4	6	3	7	8	5	5	1	4	5

- 第一轮迭代结果如下:

属于簇C1的样本有:{P7,P1,P2,P4,P5,P8};

属于簇C2的样本有:{P10,P3,P6,P9};

- 重新计算新的簇中心,有:C1的中心为(3.5,5.167),C2的中心为(6.75,4.25);

K- Means算法示例

- 继续计算10个样本到新的簇的中心的距离，重新分配到新的簇中，第二轮迭代结果如下：
  属于簇C1的样本有:{P1,P2,P4,P5,P7,P10}；
  属于簇C2的样本有:{P3,P6,P8,P9}；

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
X	3	3	7	4	3	8	4	4	7	5
y	4	6	3	7	8	5	5	1	4	5

- 重新计算新的簇的中心，有:C1的中心为(3.67，5.83)，C2的中心为(6.5,3.25)；
- 继续计算10个样本到新的簇的中心的距离，重新分配到新的簇中，发现簇中心不再发生变化，算法终止。

K- Means聚类算法

Scikit-learn的Cluster类提供聚类分析的方法：

- 模型初始化

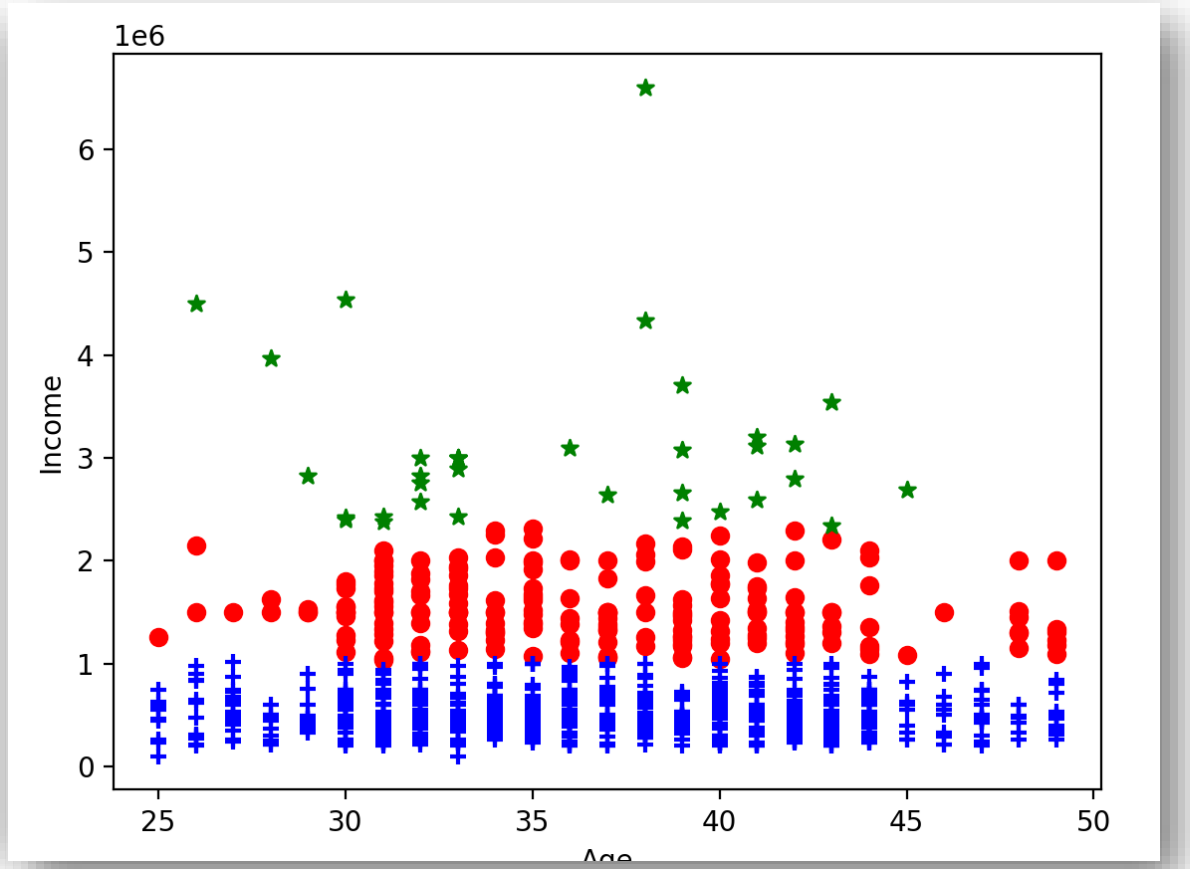
  kmeans=**Kmeans**(n_clusters)　#参数为簇的个数

- 模型学习

  kmeans. **fit**(X)　　#参数为样本二维数组

客户类型聚类分析

	A	B
1	收入	年龄
2	503999	46
3	452766	36
4	100000	33
5	100000	25
6	258000	35
7	933333	31
8	665000	40
9	291332	38
10	259000	45



客户类型聚类（1）

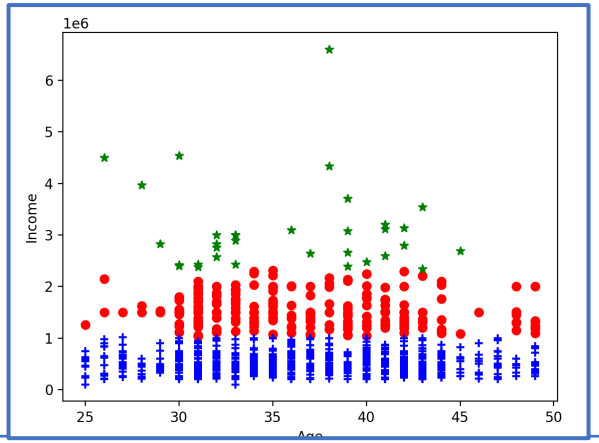
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
data = pd.read_csv("client.csv",encoding="utf-8")
```

客户类型聚类（2）

```
X=data.iloc[:,0:2].values
estimator = KMeans(n_clusters=3)#模型初始化
estimator.fit(X) #模型学习
label_pred = estimator.labels_ #获取聚类标签
x0 = X[label_pred == 0]
x1 = X[label_pred == 1]
x2 = X[label_pred == 2]
```

客户类型聚类（3）

```
plt.scatter(x0[:,1], x0[:,0], c = "red", marker='o')

plt.scatter(x1[:,1], x1[:,0], c = "green", marker='*')

plt.scatter(x2[:,1], x2[:,0], c = "blue", marker='+')

plt.xlabel('Age')

plt.ylabel('Income')

plt.show()
```



谢　谢