1 作业清单（4/20）

1.1 问题一环境安装

【1】安装Python 3.X 和 Orange3 软件，是否完成？

回答：已完成

[131]:
```
import Orange
data = Orange.data.Table("iris")


print("Attributes",",".join(x.name for x in data.domain.attributes))
```

```
Attributes sepal length,sepal width,petal length,petal width
```

1.2 问题二教师身份判断实验

【2】完成课堂实验（给定教师数据，判断身份的实验），是否完成？

回答：已完成

[3]:
```
# 给定数据集
test_data = {'Tom': {'RANK': 'A1', 'YEARS': 2, 'TENURED': 'N'},
             'Merlisa': {'RANK': 'A2', 'YEARS': 7, 'TENURED': 'N'},
             'George': {'RANK': 'P', 'YEARS': 5, 'TENURED': 'Y'},
             'Joseph': {'RANK': 'A1', 'YEARS': 7, 'TENURED': 'Y'}}
```

[4]:
```
# 给定新的数据
name = "TQ"
rank = "A1"
year = 10
result = "N"


# 交互式输入数据
# name = input("What is your name?")
# rank = input("What is your rank?")
# year = int(input("What is your year?"))
# result = 'N'
```

[12]:
```
# m 用来记录是否生成了该职位的规则，c 用来记录规则
c = {}
```

```
m = {'P': False, 'A1': False, 'A2': False}

# 找到某一职称下的终身职位的工作年限最小值，作为该职称的分类规则
# 若该职称无终身职位数据，则记录为未发现规则，并将工作年限最大值作为作为可能的分类规则
for key, value in test_data.items():
    if value['TENURED'] == 'Y':
        if len(c) == 0:
            c[value['RANK']] = value['YEARS']
        else:
            if value['RANK'] not in c.keys():
                c[value['RANK']] = value['YEARS']
            else:
                if c[value['RANK']] > value['YEARS']:
                    c[value['RANK']] = value['YEARS']
        m[value['RANK']] = True
    else:
        if len(c) == 0:
            c[value['RANK']] = value['YEARS']
        else:
            if value['RANK'] not in c.keys():
                c[value['RANK']] = value['YEARS']
            else:
                if c[value['RANK']] < value['YEARS']:
                    c[value['RANK']] = value['YEARS']
```

[13]:
```
m
```

[13]:
```
{'P': True, 'A1': True, 'A2': False}
```

[14]:
```
c
```

[14]:
```
{'A1': 2, 'A2': 7, 'P': 5}
```

[17]:
```
# 若测试数据岗位有规则，按照规则分类
# 否则若工作年限大于给定数据最大值，暂时判定为终身职位
if m[rank]:
    if year >= c[rank]:
```

```
        result = 'Y'
else:
    if year > c[rank]:
        result = 'Y'

if result == 'Y':
    print("You are TENURED")
else:
    print("You aren't TENURED")
```

```
You are TENURED
```

1.3 问题三复习 Numpy

【3】复习Numpy的主要功能（按照课堂PPT完成相关实验）

[21]:
```
import numpy as np
# ndarray
a = np.array([1, 2, 3, 4])
b = np.array((5, 6, 7, 8))
c = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
```

[22]:
```
a,b,c
```

[22]:
```
(array([1, 2, 3, 4]), array([5, 6, 7, 8]), array([[1, 2, 3, 4],
        [5, 6, 7, 8]]))
```

[25]:
```
# arange 的用法，生成取值在 [start,stop) 范围内步长为 step 的列表
a = np.arange(0, 1, 0.1)
a
```

[25]:
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

[27]:
```
# linspace 的用法，生成取值在 [start,stop) 范围内等分的 num 个数
b = np.linspace(0, 1, 10)
b
```

[27]:
```
array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
       0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])
```

[29]:
```
# 广播
c = np.arange(3)
d = np.arange(3)+5
c,d
```

[29]: (array([0, 1, 2]), array([5, 6, 7]))

1.4 问题四教材习题

【4】完成参考教材第1章 1.9的第1题和第4题（抄题）

1.什么是数据挖掘？在你的回答中，强调以下问题：

数据挖掘是从大量数据中“挖掘”有趣的信息或模式以能够基于该决策做出决策的过程或方法。

(a) 它是又一种广告宣传吗？

回答：数据挖掘是另一个炒作，但实际上数据挖掘的需求是由于海量数据的广泛可用性以及将此类数据转换为有用的信息以供我们做出决定或进行分析的需要。因此，数据挖掘是信息技术发展的结果。

(b) 它是一种从数据库、统计学、机器学习和模式识别发展而来的技术的简单转换或应用吗？

回答：不，数据挖掘远不止于此。数据挖掘不仅仅是对从数据库，统计数据和机器学习中开发的技术进行的简单转换。它将多种学科的技术（例如数据库技术，统计，机器学习，高性能计算，模式识别，神经网络，数据可视化，信息检索，图像和信号处理以及空间数据分析）进行集成，而不是简单地进行转换。

(c) 我们提出了一种观点，说数据挖掘是数据库技术进化的结果。你认为数据挖掘也是机器学习研究进化的结果吗？你能基于该学科的发展历史提出这一观点吗？针对统计学和模式识别领域，做相同的事。

回答：数据库技术始于数据收集和数据库创建机制的发展，从而导致了数据管理的有效机制的发展，包括数据存储，检索，查询和事务处理。大量提供查询和事务处理的数据库系统最终自然地导致了对数据分析和理解的需求。因此，数据挖掘正是出于这种需要而开始发展的。而数据挖掘概念的提出则早于机器学习概念的提出。

(d) 当把数据挖掘看做知识发现过程时，描述数据挖掘所涉及的步骤。

回答： - 数据清理（消除噪声或不一致数据） - 数据集成（多种数据源可以组合在一起） - 数据选择（从数据库中检索与分析任务相关的数据） - 数据变换（数据变换或统一成适合挖掘的形式） - 挖掘方法（使用各种方法提取数据模式） - 模式评估（使用某种度量，识别真正有价值的模式） - 知识表示（使用可视化和知识表示技术，向用户提供挖掘的知识）

答案参考

给出一个例子，其中数据挖掘对于工商企业的成功是至关重要的。该工商企业需要什么数据挖掘功能（例如考虑可以挖掘何种类型的模式）？这种模式能够通过简单的查询处理或统计分析得到吗？

回答：对在线购物平台（如淘宝、亚马逊等）上，用户对商品的打分、评价和其他人对该评论的支持度等数据的挖掘（如关联规则、排名等），可以帮助商铺和平台发现用户的关注点以及商品的市场信誉，从而做出合适的营销方案调整。这种模式不能通过简单的查询处理或统计分析得到，可能需要对评论进行情感分析，建立合适的关系模型等。

2 作业清单（4/22）

2.1 问题一环境安装

【1】继续安装Python 3.X 和 Orange3 软件，是否完成？

回答：已完成

2.2 问题二常用概率分布

【2】完成常用的概率分布代码

[333]:
```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

2.2.1 两点分布(Two point distribution)

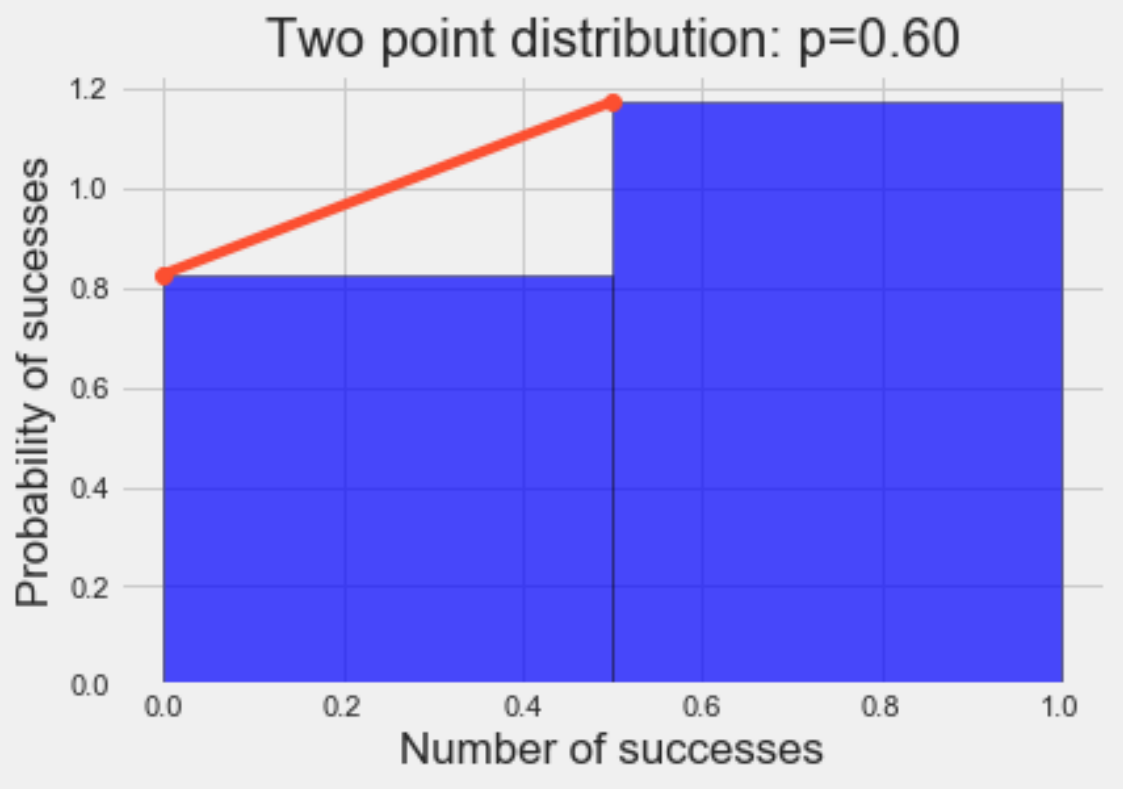
$$P(X=k)=\begin{cases} p & x=1 \\ 1-q & x=0 \end{cases} \tag{1}$$

$$E(x)=p \tag{2}$$

$$D(x)=pq \tag{3}$$

[105]:
```
p=0.6
x = np.random.binomial(1, p, size=10000)
pillar = 2
```

```
a = plt.hist(x, pillar, density=1, facecolor="blue", edgecolor="black", alpha=0.
↪7)
plt.plot(a[1][0:pillar], a[0], 'o-')
plt.title('Two point distribution: p=%0.2f' % p)
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.show()
```



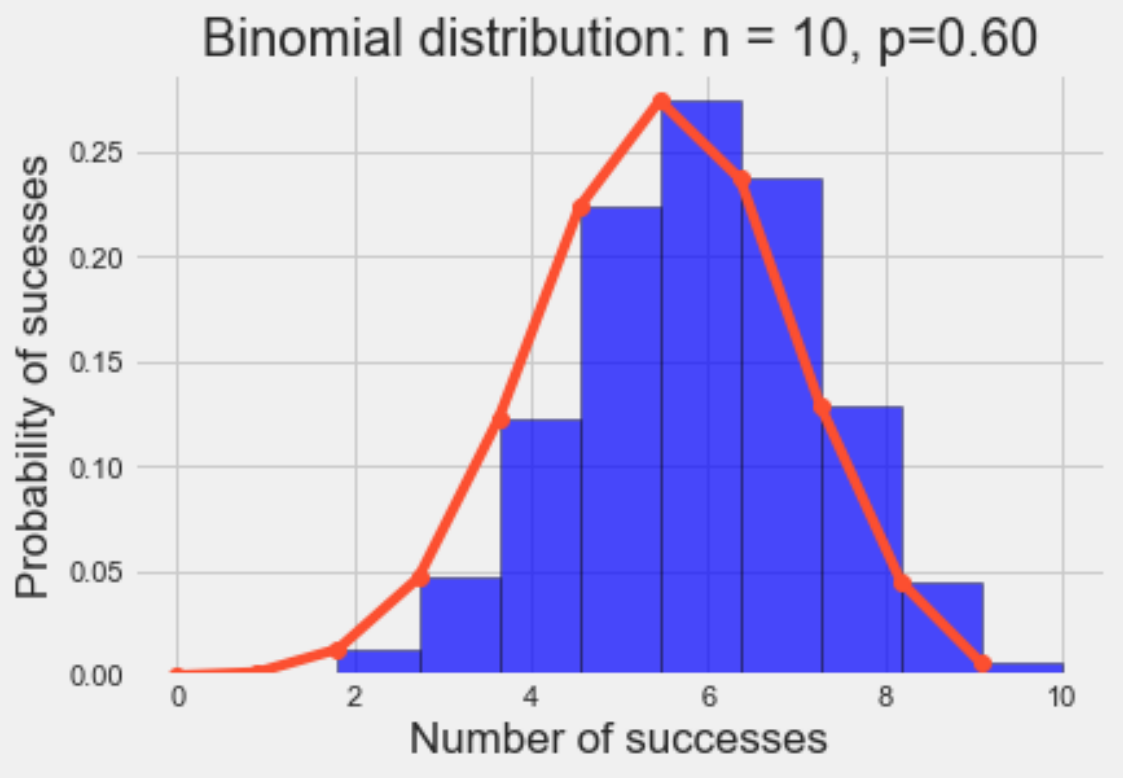
2.2.2 二项分布(Binomial distribution)

$$P(X = k) = C_n^k p^k (1-p)^{n-k} \tag{4}$$

$$E(x) = n(n-1)p^2 + np \tag{5}$$

$$D(x) = npq \tag{6}$$

[106]:
```
n=10
p=0.6
x = np.random.binomial(n, p, size=10000)
pillar = n+1
a = plt.hist(x, pillar, density=1, facecolor="blue", edgecolor="black", alpha=0.
↪7)
plt.plot(a[1][0:pillar], a[0], 'o-')
plt.title('Binomial distribution: n = %i, p=%0.2f' % (n, p))
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.show()
```



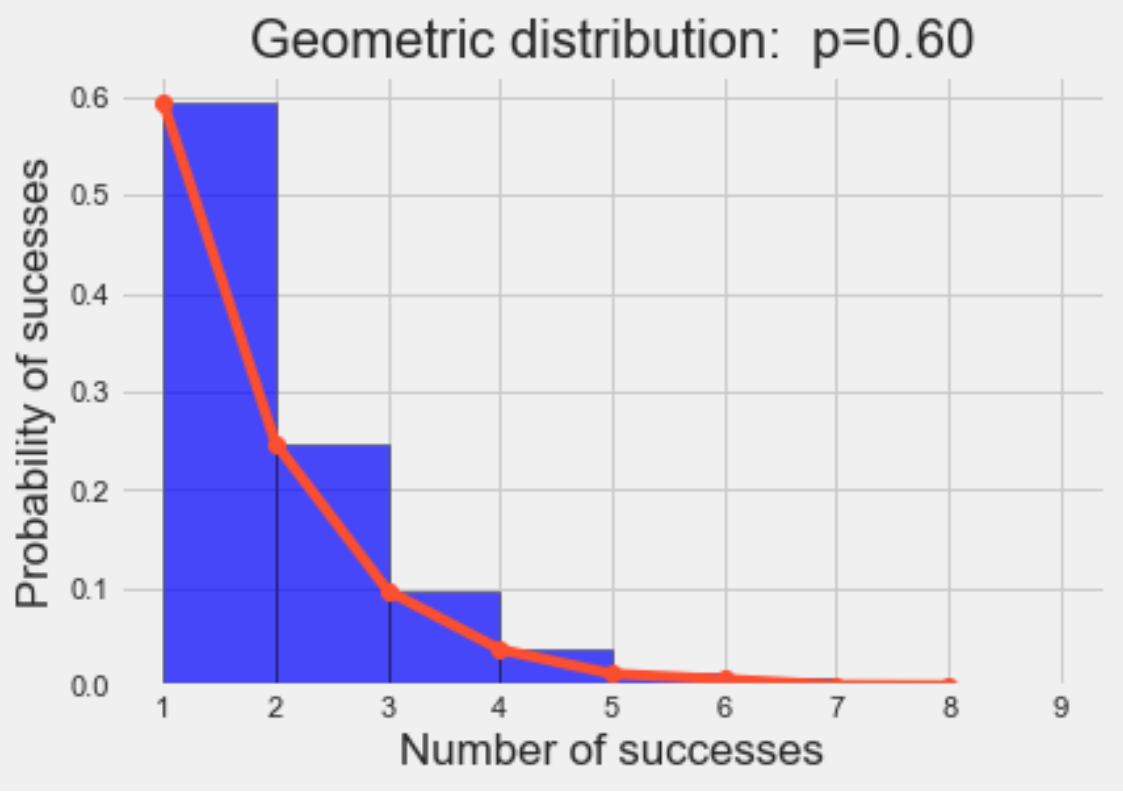
2.2.3 几何分布(Geometric distribution)

$$P(X=k)=(1-p)^{k-1}p, k=1,2,\cdots \tag{7}$$

$$E(x) = \frac{1}{p} \tag{8}$$

$$D(x) = \frac{1-p}{p^2} \tag{9}$$

[107]:
```
p=0.6
x = np.random.geometric(p,size=10000)
pillar = 8
a = plt.hist(x, pillar, density=1, facecolor="blue", edgecolor="black", alpha=0.
↪7)
plt.plot(a[1][0:pillar], a[0], 'o-')
plt.title('Geometric distribution:  p=%0.2f' % p)
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.show()
```



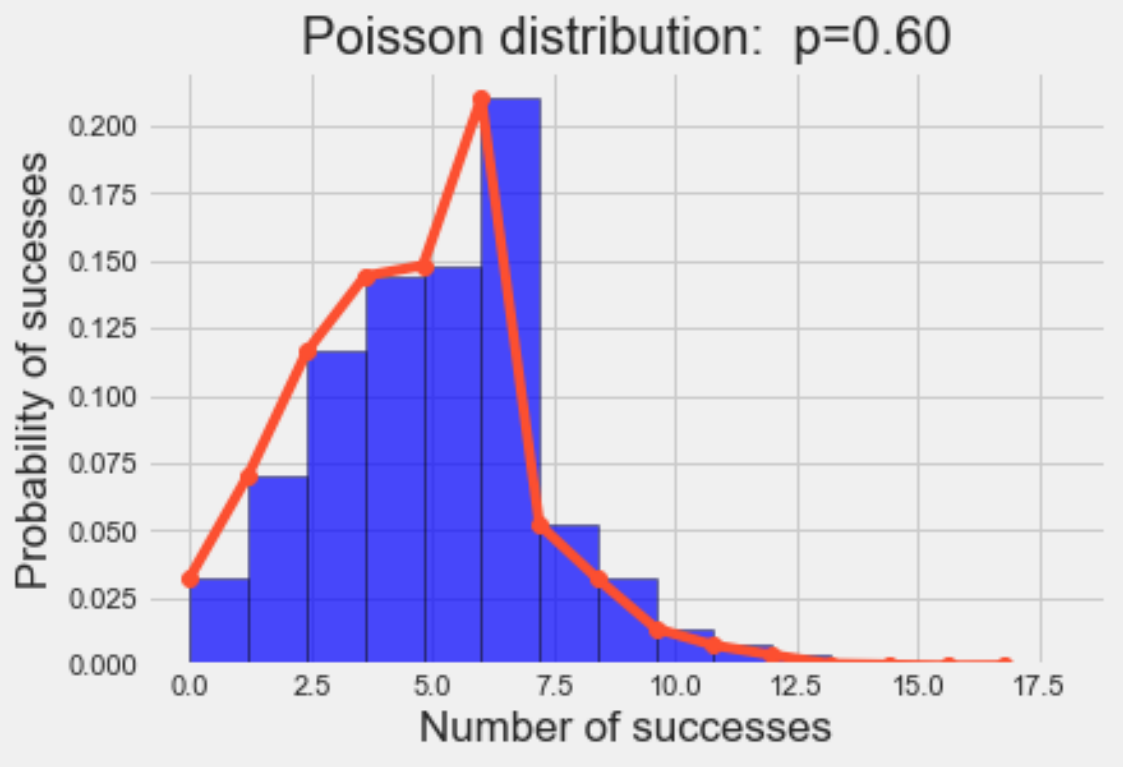
2.2.4 泊松分布(Possion distribution)

$$P(X=k)=\frac{\lambda^k}{k!}e^{-\lambda}, k=0,1,\cdots \tag{10}$$

$$E(x)=\lambda \tag{11}$$

$$D(x)=\lambda \tag{12}$$

[108]:
```
x = np.random.poisson(lam=5, size=10000)
pillar = 15
a = plt.hist(x, pillar, density=1, facecolor="blue", edgecolor="black", alpha=0.
↪7)
plt.plot(a[1][0:pillar], a[0], 'o-')
plt.title('Poisson distribution:  p=%0.2f' % p)
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.show()
```



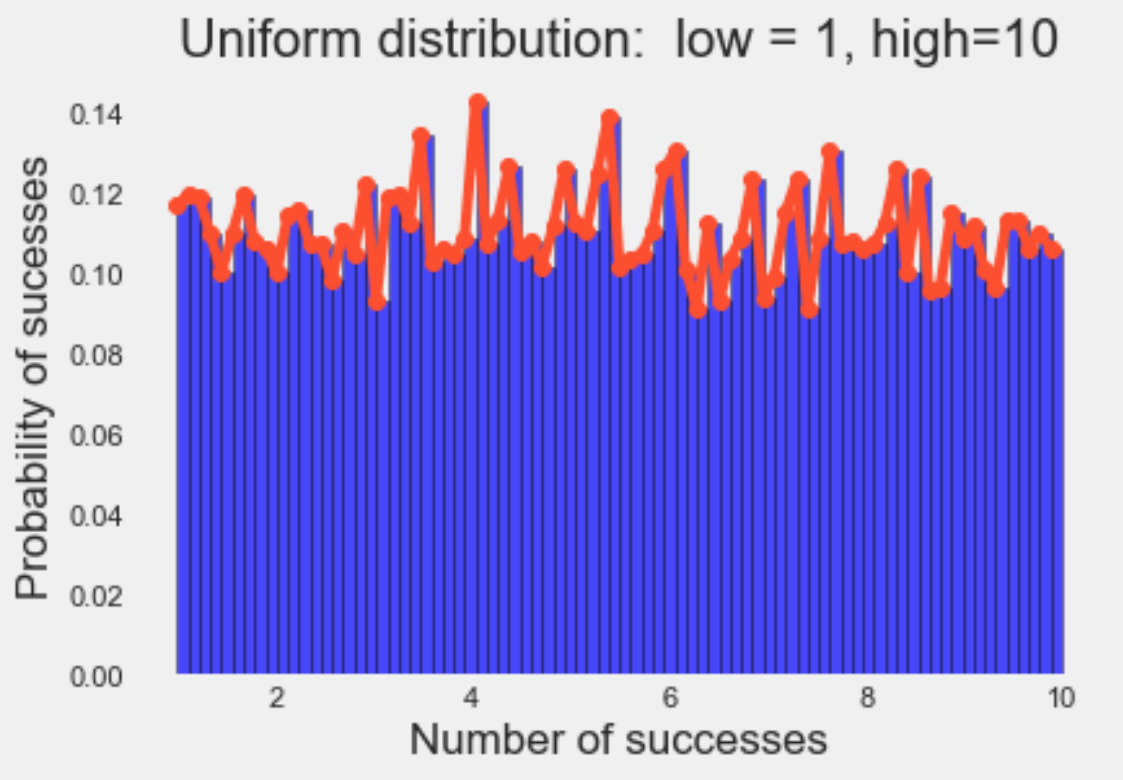
2.2.5 均匀分布(Uniform distribution)

$$f(x) = \begin{cases} \frac{1}{b-a}, a < x < b \\ 0 \end{cases} \tag{13}$$

$$E(x) = \frac{a+b}{2} \tag{14}$$

$$D(x) = \frac{(b-a)^2}{12} \tag{15}$$

[109]:
```
low=1
high=10
x = np.random.uniform(low,high,size=10000)
pillar = 80
a = plt.hist(x, pillar, density=1, facecolor="blue", edgecolor="black", alpha=0.
↪7)
plt.plot(a[1][0:pillar], a[0], 'o-')
plt.title('Uniform distribution:  low = %i, high=%i' % (low,high))
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.grid()
plt.show()
```



2.2.6 指数分布(Exponential distribution)

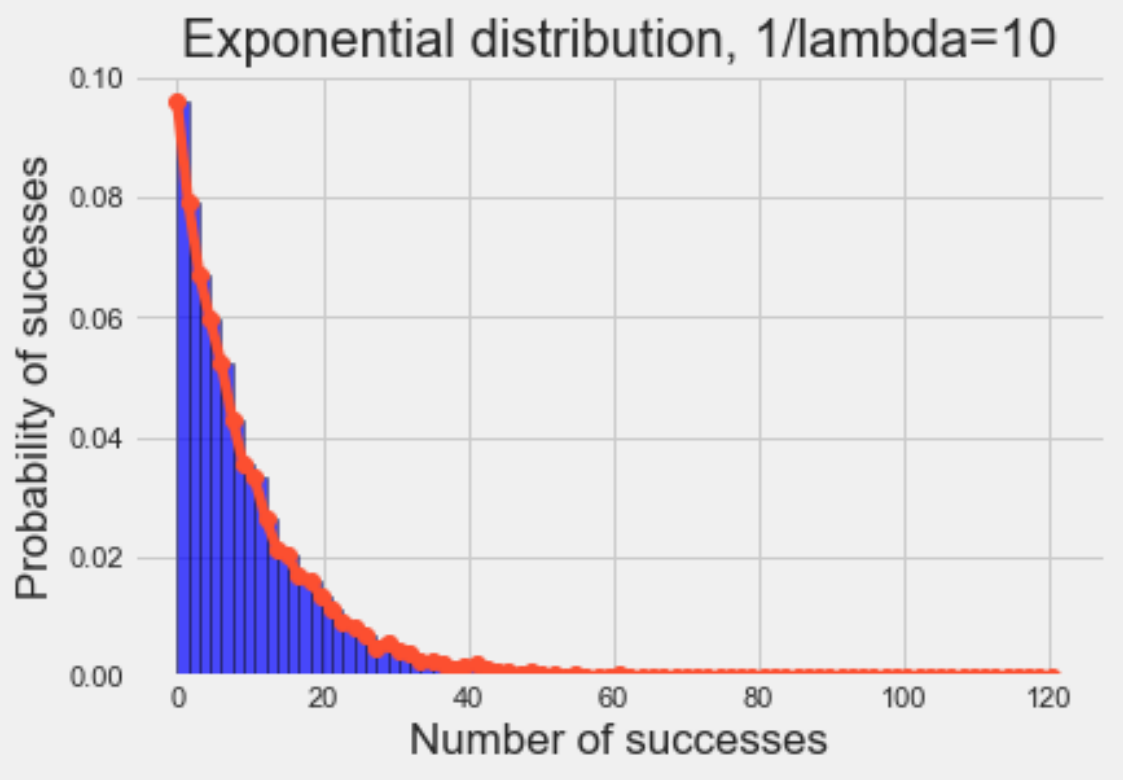
$$f(x) = \lambda e^{-\lambda x} \tag{16}$$

$$E(x) = \frac{1}{\lambda} \tag{17}$$

$$D(x) = \frac{1}{\lambda^2} \tag{18}$$

[110]:
```
tau = 10
pillar=80
x = np.random.exponential(tau, size=10000)
a = plt.hist(x, pillar, density=1, facecolor="blue", edgecolor="black", alpha=0.
↪7)
plt.plot(a[1][0:pillar], a[0], 'o-')
plt.title('Exponential distribution, 1/lambda=10')
```

```
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.show()
```



2.2.7 正态分布(Normal distribution)

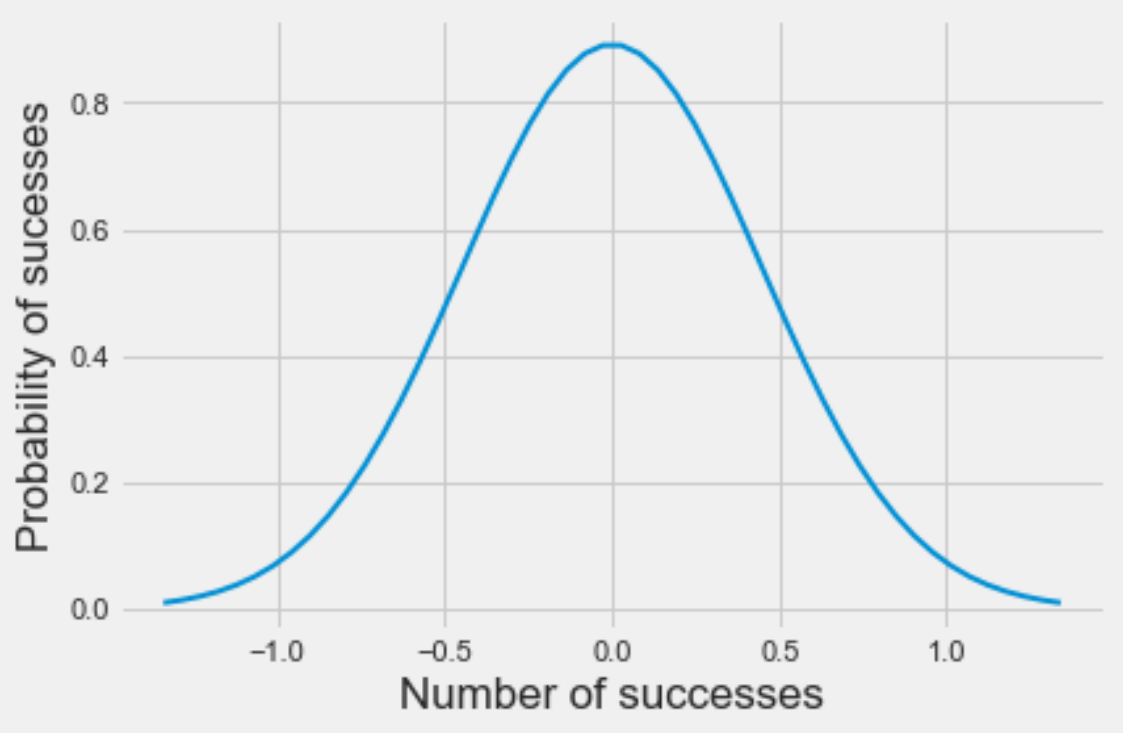
$$f(x,\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{19}$$

$$E(x) = \mu \tag{20}$$

$$D(x) = \sigma^2 \tag{21}$$

[111]:
```
import math
u = 0
sig = math.sqrt(0.2)
```

```
x = np.linspace(u - 3 * sig, u + 3 * sig, 50)
y_sig=np.exp(-(x-u)**2/(2*sig**2))/(math.sqrt(2*math.pi)*sig)
# print(x)
# print("="*20)
# print(y_sig)
plt.plot(x,y_sig,linewidth=2)
plt.xlabel('Number of successes')
plt.ylabel('Probability of sucesses')
plt.show()
```



2.3 问题三最大似然估计(MLE)

【3】完成最大似然估计MLE的Python代码

数学小白用python做极大似然估计MLE

[100]:
```
from scipy.stats import norm
norm.pdf(3, 3, 1)
```

[100]: 0.3989422804014327

[112]:
```
norm.pdf(3, 7, 2)
```

[112]: 0.02699548325659403

[113]:
```
norm.pdf(2, 2, 1) * norm.pdf(7, 2, 1)
```

[113]: 5.931152735254122e-07

[114]:
```
p1 = norm.pdf(2, 2, 1) * norm.pdf(7, 2, 1)
p2 = norm.pdf(2, 3, 2) * norm.pdf(7, 3, 2)
print('p1:', p1)
print('p2:', p2)
```

```
p1: 5.931152735254122e-07
p2: 0.0047520868169464775
```

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{22}$$

[115]:
```
# 求导数的python模块sympy
# log(p)
from sympy import symbols, pi, exp, log
from sympy.stats import Probability, Normal

# 样本
X = [1,2,3,4,5, 3,4,2,5,6,]
x = symbols('x')
# 总体 mu 和 sigma
m, s = symbols('m s')
# pdf
pdf = 1/ (s * (2*pi)**0.5)*exp( -(x-m)**2 / (2*s**2) )
logpdf = log(pdf)
logpdf
```

[115]: $$\log\left(\frac{0.707106781186547 e^{-\frac{(-m+x)^2}{2s^2}}}{\pi^{0.5} s}\right)$$

[116]:
```
# 上面输出的内容就是logpdf公式
# 求已知样本后的累加log联合概率，也叫似然函数
logP = 0
for xi in X:
    logP += logpdf.subs({x: xi})

logP
```

[116]:
$$\log\left(\frac{0.707106781186547e^{-\frac{(1-m)^2}{2s^2}}}{\pi^{0.5}s}\right) + 2\log\left(\frac{0.707106781186547e^{-\frac{(2-m)^2}{2s^2}}}{\pi^{0.5}s}\right) + 2\log\left(\frac{0.707106781186547e^{-\frac{(3-m)^2}{2s^2}}}{\pi^{0.5}s}\right) + 2\log\left(\frac{0.707106781186547e^{-\frac{(4-m)^2}{2s^2}}}{\pi^{0.5}s}\right) + 2\log\left(\frac{0.707106781186547e^{-\frac{(5-m)^2}{2s^2}}}{\pi^{0.5}s}\right) + \log\left(\frac{0.707106781186547e^{-\frac{(6-m)^2}{2s^2}}}{\pi^{0.5}s}\right)$$

[117]:
```
from sympy import diff

logp_diff_m = diff(logP, m)
logp_diff_s = diff(logP, s)

logp_diff_m
```

[117]:
$$-\frac{0.5\,(2m-12)}{s^2} - \frac{1.0\,(2m-10)}{s^2} - \frac{1.0\,(2m-8)}{s^2} - \frac{1.0\,(2m-6)}{s^2} - \frac{1.0\,(2m-4)}{s^2} - \frac{0.5\,(2m-2)}{s^2}$$

[118]:
```
logp_diff_s
```

[118]:
$$1.4142135623731\pi^{0.5}s\left(-\frac{0.707106781186547e^{-\frac{(1-m)^2}{2s^2}}}{\pi^{0.5}s^2} + \frac{0.707106781186547\,(1-m)^2\,e^{-\frac{(1-m)^2}{2s^2}}}{\pi^{0.5}s^4}\right)e^{\frac{(1-m)^2}{2s^2}} + 2.82842712474619\pi^{0.5}s\left(-\frac{0.707106781186547e^{-\frac{(2-m)^2}{2s^2}}}{\pi^{0.5}s^2} + \frac{0.707106781186547\,(2-m)^2\,e^{-\frac{(2-m)^2}{2s^2}}}{\pi^{0.5}s^4}\right)e^{\frac{(2-m)^2}{2s^2}} + 2.82842712474619\pi^{0.5}s\left(-\frac{0.707106781186547e^{-\frac{(3-m)^2}{2s^2}}}{\pi^{0.5}s^2} + \frac{0.707106781186547\,(3-m)^2\,e^{-\frac{(3-m)^2}{2s^2}}}{\pi^{0.5}s^4}\right)e^{\frac{(3-m)^2}{2s^2}} + 2.82842712474619\pi^{0.5}s\left(-\frac{0.707106781186547e^{-\frac{(4-m)^2}{2s^2}}}{\pi^{0.5}s^2} + \frac{0.707106781186547\,(4-m)^2\,e^{-\frac{(4-m)^2}{2s^2}}}{\pi^{0.5}s^4}\right)e^{\frac{(4-m)^2}{2s^2}} + 2.82842712474619\pi^{0.5}s\left(-\frac{0.707106781186547e^{-\frac{(5-m)^2}{2s^2}}}{\pi^{0.5}s^2} + \frac{0.707106781186547\,(5-m)^2\,e^{-\frac{(5-m)^2}{2s^2}}}{\pi^{0.5}s^4}\right)e^{\frac{(5-m)^2}{2s^2}} +$$

$$1.4142135623731\pi^{0.5}s\left(-\frac{0.707106781186547e^{-\frac{(6-m)^2}{2s^2}}}{\pi^{0.5}s^2}+\frac{0.707106781186547\left(6-m\right)^2e^{-\frac{(6-m)^2}{2s^2}}}{\pi^{0.5}s^4}\right)e^{\frac{(6-m)^2}{2s^2}}$$

[119]:
```
# 联立方程组,然后解方程就可以得到参数的值
from sympy import simplify

logp_diff_m = simplify(logp_diff_m) # 化简
logp_diff_m
```

[119]: $\frac{35.0-10.0m}{s^2}$

[120]:
```
logp_diff_s= simplify(logp_diff_s)
logp_diff_s
```

[120]: $\frac{10.0m^2-70.0m-10.0s^2+145.0}{s^3}$

[121]:
```
from sympy import solve

funcs = [logp_diff_s, logp_diff_m]
solve(funcs, [m, s]) # s>0, 所以第二组解是对的
```

[121]:
```
[(3.50000000000000, -1.50000000000000), (3.50000000000000, 1.50000000000000)]
```

3 作业清单（4/27）

3.1 问题一熟悉读写 CSV 文件

【1】熟悉CSV文件的打开、读取和写入数据。

[123]:
```
import pandas as pd
from io import StringIO
```

3.1.1 字符串读入 CSV

[124]:
```
csv_data='square_feet,price\n150,6450\n200,7450\n250,8450\n300,9450\n350,11450\n600,18450\n'
df=pd.read_csv(StringIO(csv_data))
df
```

[124]:

	square_feet	price
0	150	6450

```
1          200   7450
2          250   8450
3          300   9450
4          350  11450
5          600  18450
```

3.1.2 从文件读入 CSV

[127]:
```
data=pd.read_csv('test1.csv')
data.head()
```

[127]:
```
   序号  活动推广费  销售额
0   1     19   60
1   2     45  113
2   3     35   94
3   4     31   90
4   5     25   60
```

3.1.3 写入CSV

[128]:
```
df.to_csv('myDataFrame.csv')
```

3.2 问题二 Orange 的使用

【2】利用Orange3和Python orange方法完成下图

[228]:
```
# DataFrame 和 Orange Table 对象的转换
from Orange.data.pandas_compat import table_from_frame,table_to_frame
out_data = table_from_frame(df)
out_data.domain
```

[228]:
```
[square_feet, price]
```

[229]:
```
# 生成训练模型的数据
sf=np.array(df.square_feet)
sf=np.reshape(sf, (-1, 1))
p=np.array(df.price)
domain= Domain.from_numpy(sf,p)
domain
```

[229]:
```
[Feature | Target]
```

[230]:
```
data = Orange.data.Table(domain, sf, p)
data
```

[230]:
```
[[150 | 6450],
 [200 | 7450],
 [250 | 8450],
 [300 | 9450],
 [350 | 11450],
 ...
]
```

[206]:
```
# 最小二乘法的线性回归模型
from Orange.regression.linear import LinearRegressionLearner
mean_ = LinearRegressionLearner()
```

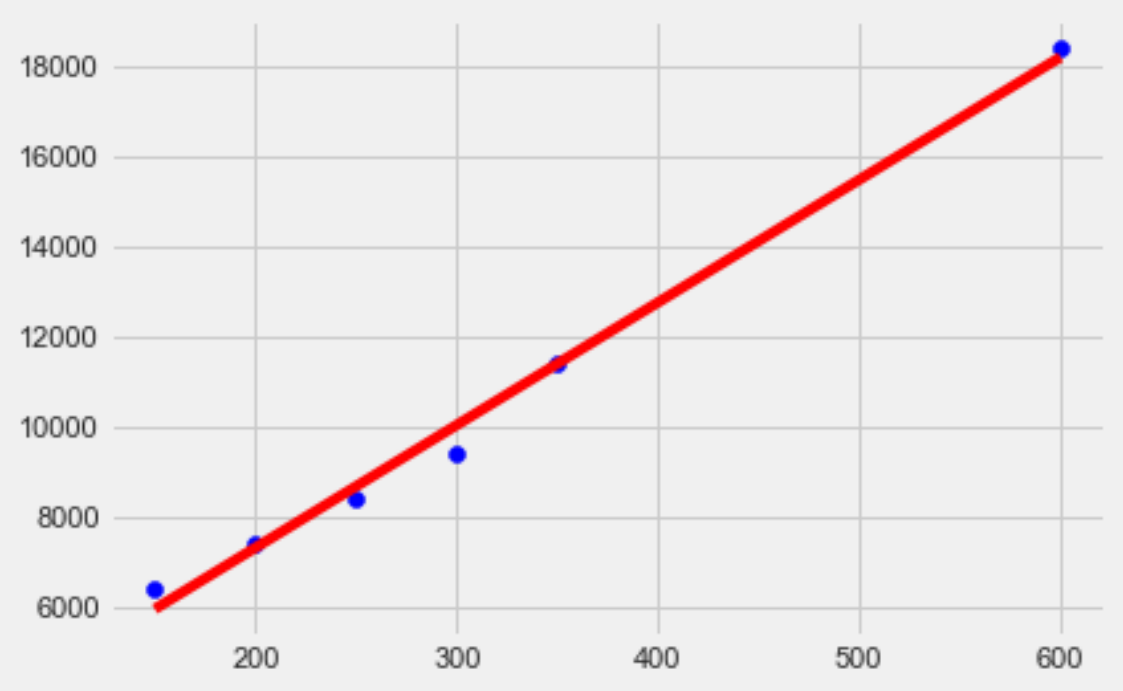
[215]:
```
model = mean_(data)
model
```

[215]:
```
LinearModel(skl_model=LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=None, normalize=False))  # params={}
```

[225]:
```
# 预测数据
model(data[:])
```

[225]:
```
array([ 5974.59016393,  7335.24590164,  8695.90163934, 10056.55737705,
       11417.21311475, 18220.49180328])
```

[227]:
```
plt.scatter(df['square_feet'],df['price'],color='blue')
plt.plot(df['square_feet'],model(data[:]),color='red',linewidth=4)
plt.show()
```



问题三最小二乘法和极大似然法

【3】思考最大似然估计MLE和最小二乘之间的关系？

两种常用于从随机样本估计总体参数的方法是最大似然估计方法（默认）和最小二乘估计方法。

最大似然估计方法（MLE）似然函数指明了观测的样本作为可能参数值函数的几率有多大。因此，通过最大化似然函数，可以确定最可能产生观测数据的参数。从统计学观点来看，一般建议对大样本使用 MLE，因为此方法是通用的，适用于大多数模型和不同类型的数据，而且会产生最精确的估计值。

最小二乘估计方法（LSE）最小二乘估计值是通过将回归线拟合到数据集中的点来计算的，这些数据集具有最小的平方差和（最小二乘误）。在可靠性分析中，回归线和数据将标绘在概率图上。

直觉上，我们可以通过理解两种方法的目的来解释这两种方法之间的联系。对于最小二乘参数估计，我们想要找到最小化数据点和回归线之间距离平方之和的直线。在最大似然估计中，我们想要最大化数据同时出现的总概率。当待求分布被假设为高斯分布时，最大概率会在数据点接近平均值时找到。由于高斯分布是对称的，这等价于最小化数据点与平均值之间的距离。

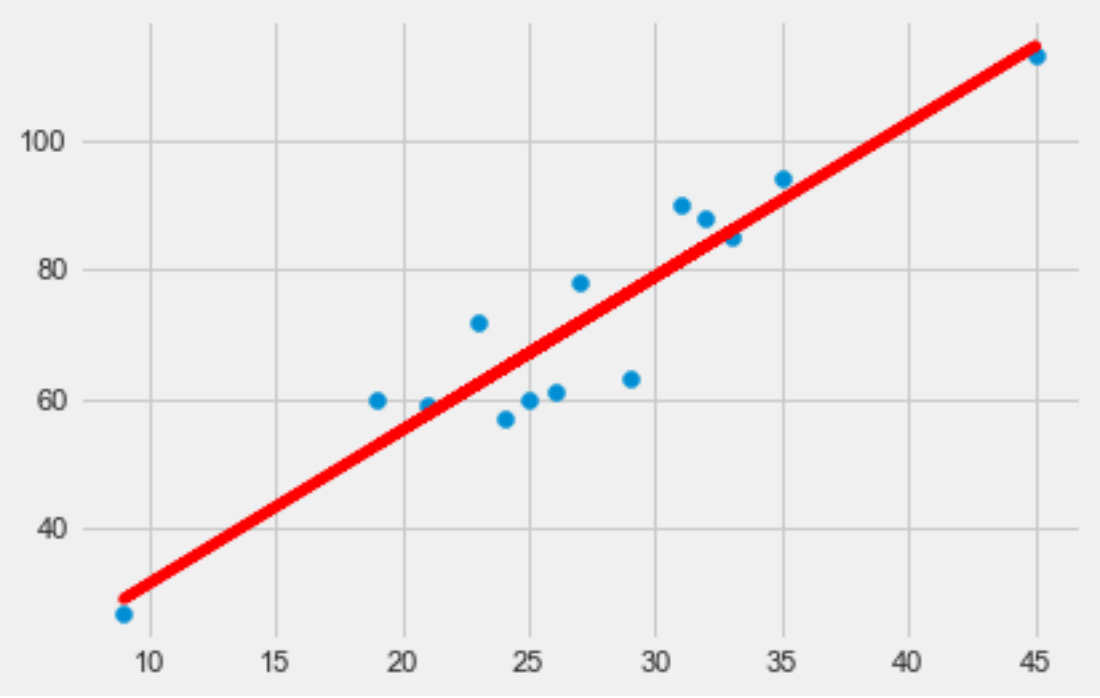
参考资料1 参考资料2

3.3 问题四绘制拟合曲线

【4】根据DM Lab3数据散点图，画出一元回归线。

[239]:
```
import numpy as np
from pandas import read_csv
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
```

[241]:
```
data=read_csv('test1.csv')
lrModel=LinearRegression()
x=data.活动推广费
y=data.销售额
lrModel.fit(x.values.reshape(-1,1),y)
plt.scatter(data.活动推广费,data.销售额)
plt.plot(data.活动推广费,lrModel.predict(data.活动推广费.values.
 ↪reshape(-1,1)),color='red',linewidth=4)
plt.show()
```



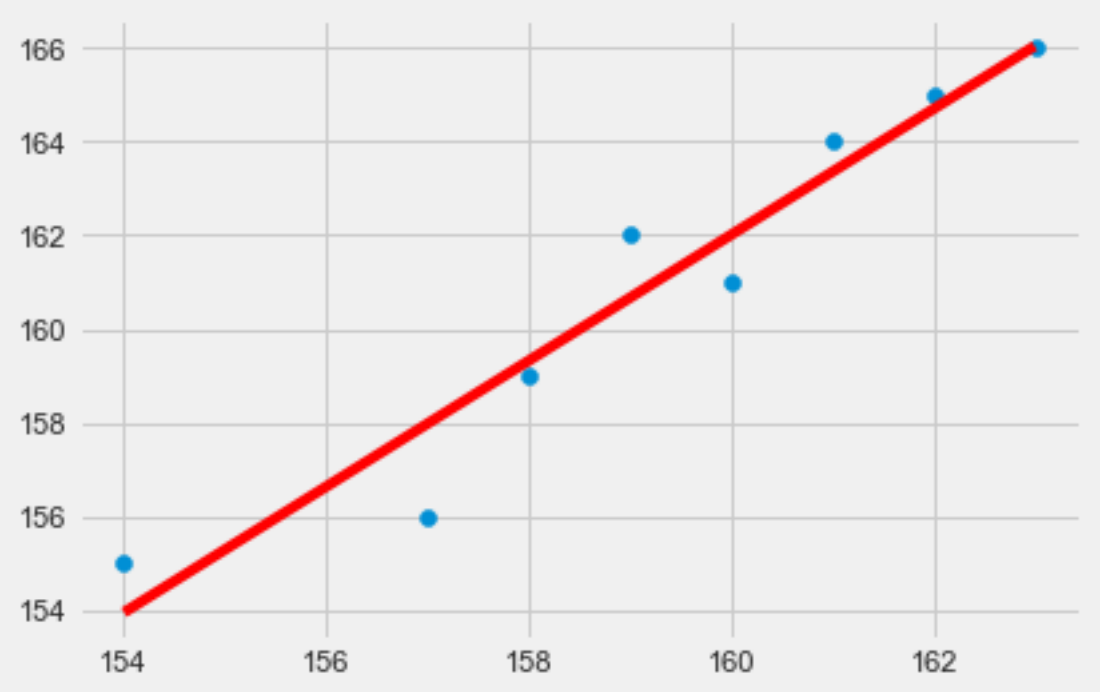
3.4 问题五一元线性回归

【5】根据DM Lab3实验过程，母亲身高167cm，预测孩子身高可能是多少？

回答：孩子的身高可能是171.42cm。

[242]:
```
import matplotlib.pyplot as plt
from sklearn import linear_model
import pandas as pd
```

[243]:
```
x=[154,157,158,159,160,161,162,163]
y=[155,156,159,162,161,164,165,166]
regr=linear_model.LinearRegression()
x=pd.DataFrame(x)
regr.fit(x.values.reshape(-1,1),y)
plt.scatter(x,y)
plt.plot(x,regr.predict(x.values.reshape(-1,1)),color='red',linewidth=4)
plt.grid(True)
plt.show()
```



[244]:
```
regr.predict([[167]]) # 修改预测值
# print(regr.predict([[167]]))
alpha=regr.intercept_
beta=regr.coef_
new_r=alpha+beta*np.array([167])
new_r
```

[244]:
```
array([171.42016807])
```

4 作业清单（4/29、5/4）

4.1 问题一房价实验

【1】根据下列数据集（数据表存为csv格式）建立线性回归模型。

（1）预测面积为1000平方英尺的房子价格。

（2）建立多元回顾模型。至少增加2项房子价格的特征，例如：地段、新旧等因素。

（3）将（1）和（2）整理成实验报告。5月6日上课检查实验报告情况。

4.1.1 一元回归模型

实验报告

1 实验过程：

- 建立数据集：通过 pandas 读入 csv 文件，存到 DataFrame 对象中，去除掉数据中不需要的属性（如行号），通过 sklearn 中的 train_test_split 函数，将原始数据中的 80% 作为训练数据，其余的作为测试数据。
- 如何建立模型：通过绘制散点图和计算相关系数矩阵，发现变量之间存在近似的线性相关关系，通过 sklearn 中的 LinearRegression 对数据进行拟合，计算模型的参数。
- 预测：通过模型的参数计算预测值或通过 LinearRegression 的 predict 函数得到预测值。

2 程序源代码

[329]:
```
# 一元线性回归模型
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.linear_model import LinearRegression
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

[330]:
```
data = pd.read_csv("week10_homework.csv")
data.head()
```

[330]:

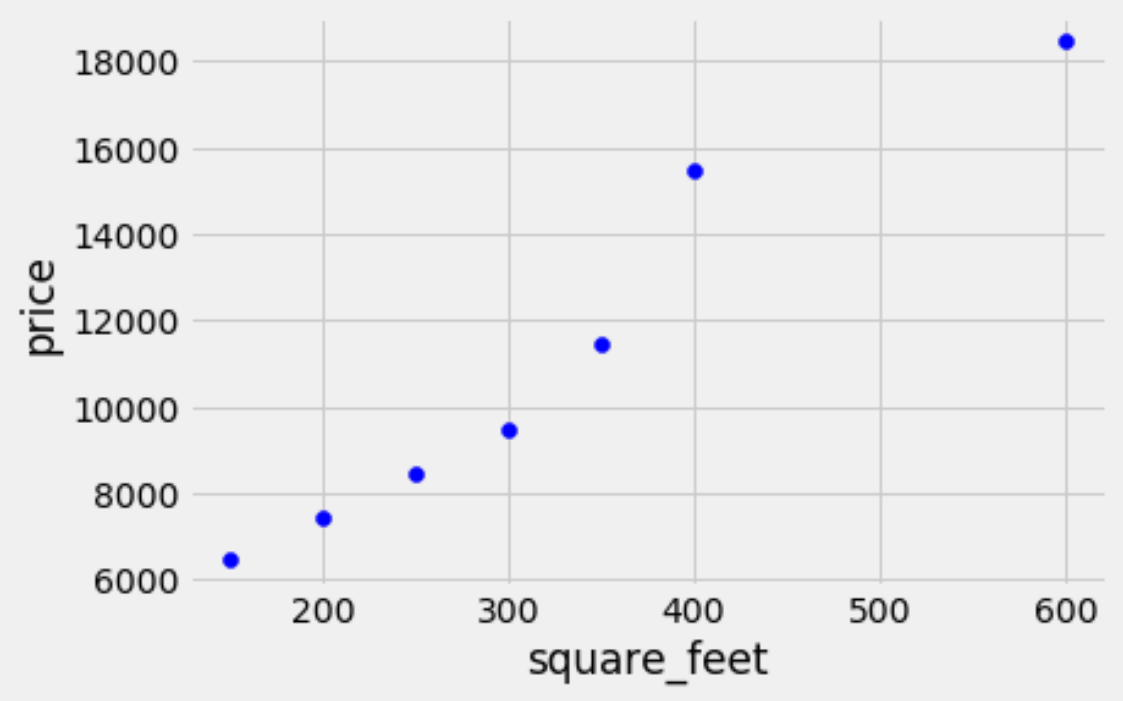
	No	square_feet	price
0	1	150	6450
1	2	200	7450
2	3	250	8450
3	4	300	9450
4	5	350	11450

[331]:
```
# 去除不需要的数据
new_data = data.iloc[:,1:]
new_data.head()
```

[331]:

	square_feet	price
0	150	6450
1	200	7450
2	250	8450
3	300	9450
4	350	11450

[334]:
```
# 绘图观察变量之间的关系
plt.scatter(new_data.square_feet,new_data.price,color = 'b',label = "Exam Data")
plt.xlabel("square_feet")
plt.ylabel("price")
plt.show()
```



[335]:
```
# 计算相关系数
rDf = new_data.corr()
rDf
```

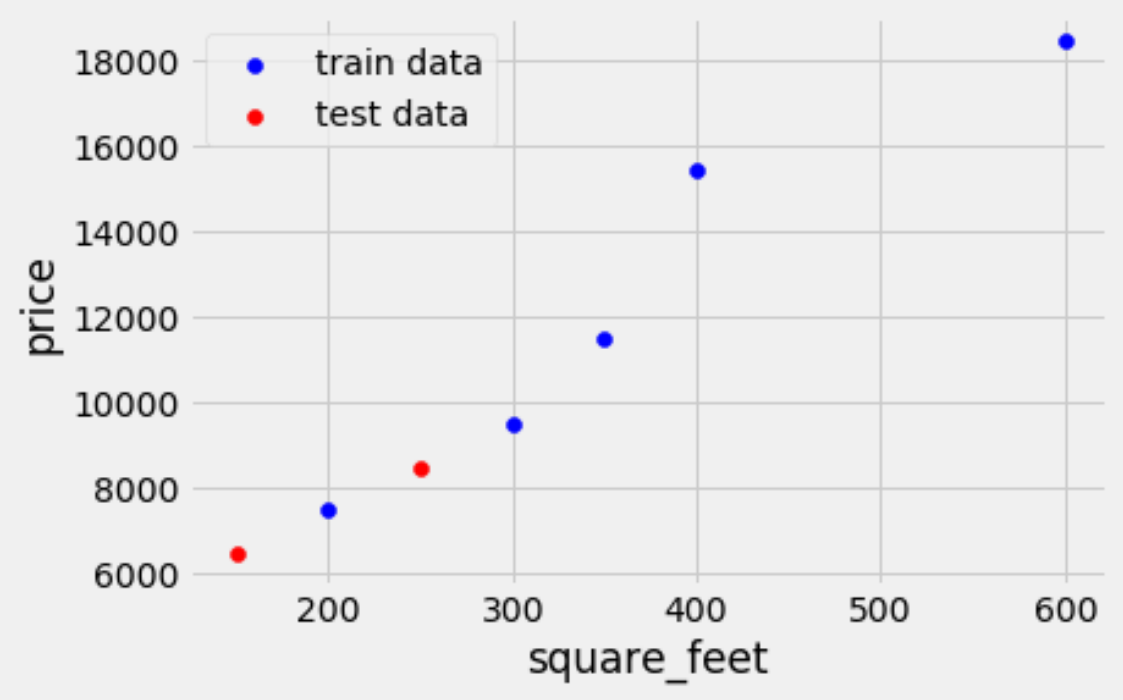
[335]:

	square_feet	price
square_feet	1.000000	0.971941
price	0.971941	1.000000

[336]:
```
exam_X = data.square_feet
exam_Y = data.price
X_train,X_test,Y_train,Y_test = train_test_split(exam_X,exam_Y,train_size=.8)
#X_train为训练数据标签,X_test为测试数据标签,exam_X为样本特征,exam_y为样本标签,
train_size 训练数据占比
print("原始数据特征:",exam_X.shape,      ",训练数据特征:",X_train.shape,      ␣
 ↪",测试数据特征:",X_test.shape)
print("原始数据标签:",exam_Y.shape,      ",训练数据标签:",Y_train.shape,      ␣
 ↪",测试数据标签:",Y_test.shape)
```

```
原始数据特征: (7,) ,训练数据特征: (5,) ,测试数据特征: (2,)
原始数据标签: (7,) ,训练数据标签: (5,) ,测试数据标签: (2,)
```

[337]:
```
plt.scatter(X_train, Y_train, color="blue", label="train data")
plt.scatter(X_test, Y_test, color="red", label="test data")
plt.legend(loc=2)
plt.xlabel("square_feet")
plt.ylabel("price")
plt.show()
```

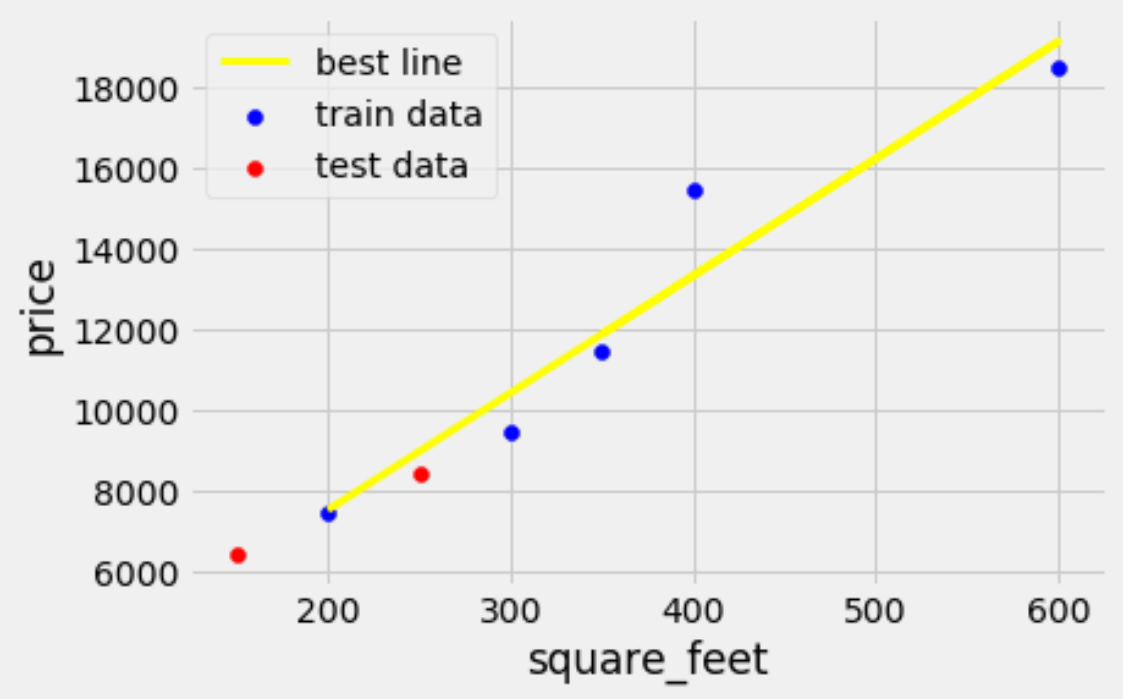


[338]:
```
model = LinearRegression()
X_train = X_train.values.reshape(-1,1)
X_test = X_test.values.reshape(-1,1)
model.fit(X_train,Y_train)
a  = model.intercept_
b = model.coef_
print("最佳拟合线:截距",a,",回归系数: ",b)
```

最佳拟合线:截距 1728.40909090909 ,回归系数: [28.97727273]

[339]:
```
y_train_pred = model.predict(X_train)
plt.plot(X_train, y_train_pred, color='yellow', linewidth=3, label="best line")
```

```
plt.scatter(X_train, Y_train, color="blue", label="train data")
plt.scatter(X_test, Y_test, color='red', label="test data")
plt.legend(loc=2)
plt.xlabel("square_feet")
plt.ylabel("price")
plt.show()
```



[300]:
```
model.predict([[1000]])
print(model.predict([[1000]]))
alpha=model.intercept_
beta=model.coef_
new_r=alpha+beta*np.array([1000])
new_r
```

```
[35450.]
```

[300]:
```
array([35450.])
```

3 程序运行结果及分析

根据程序运行结果，可以得知面积为1000平方英尺的房子价格约为29171.24，模型的准确率

为77.38%。

[301]:
```
model.score(X_test, Y_test)
```

[301]:
```
0.5906441466287042
```

4 知识点总结

- 训练集与测试集的划分
- 一元线性回归模型的实现
- 相关库

4.1.2 多元回归模型

实验报告

1 实验过程:

- 建立数据集:
  - 数据来源
  - pandas 读入 csv 文件
  - 数据预处理
    - 通过均值填充**缺失值**
    - 对分类数据进行**独热编码**
- 如何建立模型：通过绘制散点图和计算相关系数矩阵，发现变量之间存在近似的线性相关关系，通过 sklearn 中的 LinearRegression 对数据进行拟合，计算模型的参数。
- 预测：通过模型的参数计算预测值或通过 LinearRegression 的 predict 函数得到预测值。

[340]:
```
train_df = pd.read_csv('train.csv', index_col=0)
test_df = pd.read_csv('test.csv', index_col=0)
```

[303]:
```
# 训练集
train_df.head()
```

[303]:

Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape
1	60	RL	65.0	8450	Pave	NaN	Reg
2	20	RL	80.0	9600	Pave	NaN	Reg
3	60	RL	68.0	11250	Pave	NaN	IR1
4	70	RL	60.0	9550	Pave	NaN	IR1
5	60	RL	84.0	14260	Pave	NaN	IR1

```
   LandContour Utilities LotConfig  ... PoolArea PoolQC Fence MiscFeature  \
Id                                  ...
1          Lvl    AllPub    Inside  ...        0    NaN   NaN         NaN
2          Lvl    AllPub       FR2  ...        0    NaN   NaN         NaN
3          Lvl    AllPub    Inside  ...        0    NaN   NaN         NaN
4          Lvl    AllPub    Corner  ...        0    NaN   NaN         NaN
5          Lvl    AllPub       FR2  ...        0    NaN   NaN         NaN

   MiscVal MoSold  YrSold  SaleType  SaleCondition  SalePrice
Id
1        0      2    2008        WD         Normal     208500
2        0      5    2007        WD         Normal     181500
3        0      9    2008        WD         Normal     223500
4        0      2    2006        WD        Abnorml     140000
5        0     12    2008        WD         Normal     250000

[5 rows x 80 columns]
```

[304]:
```
# 测试集
test_df.head()
```

[304]:
```
      MSSubClass MSZoning  LotFrontage  LotArea Street Alley LotShape  \
Id
1461          20       RH         80.0    11622   Pave   NaN      Reg
1462          20       RL         81.0    14267   Pave   NaN      IR1
1463          60       RL         74.0    13830   Pave   NaN      IR1
1464          60       RL         78.0     9978   Pave   NaN      IR1
1465         120       RL         43.0     5005   Pave   NaN      IR1

     LandContour Utilities LotConfig  ... ScreenPorch PoolArea PoolQC  Fence  \
Id                                    ...
1461         Lvl    AllPub    Inside  ...         120        0    NaN  MnPrv
1462         Lvl    AllPub    Corner  ...           0        0    NaN    NaN
1463         Lvl    AllPub    Inside  ...           0        0    NaN  MnPrv
1464         Lvl    AllPub    Inside  ...           0        0    NaN    NaN
1465         HLS    AllPub    Inside  ...         144        0    NaN    NaN
```

Id	MiscFeature	MiscVal	MoSold	YrSold	SaleType	SaleCondition
1461	NaN	0	6	2010	WD	Normal
1462	Gar2	12500	6	2010	WD	Normal
1463	NaN	0	3	2010	WD	Normal
1464	NaN	0	6	2010	WD	Normal
1465	NaN	0	1	2010	WD	Normal

[5 rows x 79 columns]

结合数据描述以及以上输出结果可以大致得出哪些数据需要人为处理一下，为了方便统一处理，先将训练数据和测试数据合并，等所有的需要的预处理进行完之后，再把他们分隔开。需注意的是SalePrice作为训练目标，只会出现在训练集中，不会在测试集中。所以，先把SalePrice这一列给拿出来。

[341]:
```
#将训练目标单独拿出
#y_train则是SalePrice那一列
y_train = np.log1p(train_df.pop('SalePrice'))
#把剩下的部分合并起来
all_df = pd.concat((train_df, test_df), axis=0)
all_df.shape
```

[341]: (2919, 79)

下面进行变量转化，把不方便处理或不一致的数据给统一了，首先，注意到 MSSubClass 的值应该是一个分类型的，但是 Pandas 会将这类数字符号记成数字，因而需要把它变回成string。

[342]:
```
all_df['MSSubClass'].dtypes
all_df['MSSubClass'] = all_df['MSSubClass'].astype(str)
#变成str以后，做个统计
all_df['MSSubClass'].value_counts()
```

[342]:
```
20     1079
60      575
50      287
120     182
30      139
70      128
```

```
160     128
80      118
90      109
190      61
85       48
75       23
45       18
180      17
40        6
150       1
Name: MSSubClass, dtype: int64
```

[343]:
```
#MSSubClass被分成了12个column，每一个代表一个类。是就是1，不是就是0。
pd.get_dummies(all_df['MSSubClass'], prefix='MSSubClass').head()
```

[343]:

Id	MSSubClass_120	MSSubClass_150	MSSubClass_160	MSSubClass_180	MSSubClass_190	MSSubClass_20	MSSubClass_30	MSSubClass_40	MSSubClass_45	MSSubClass_50	MSSubClass_60	MSSubClass_70	MSSubClass_75
1	0	0	0	0	0	0	0	0	0	0	1	0	0
2	0	0	0	0	0	1	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	0	0	0	1	0
5	0	0	0	0	0	0	0	0	0	0	1	0	0

```
5               0               0               1               0               0

    MSSubClass_80  MSSubClass_85  MSSubClass_90
Id
1               0              0              0
2               0              0              0
3               0              0              0
4               0              0              0
5               0              0              0
```

[344]:
```
#同理，把所有的类数据，都给One-Hot了
all_dummy_df = pd.get_dummies(all_df)
all_dummy_df.head()
```

[344]:
```
    LotFrontage  LotArea  OverallQual  OverallCond  YearBuilt  YearRemodAdd  \
Id
1          65.0     8450            7            5       2003          2003
2          80.0     9600            6            8       1976          1976
3          68.0    11250            7            5       2001          2002
4          60.0     9550            7            5       1915          1970
5          84.0    14260            8            5       2000          2000

    MasVnrArea  BsmtFinSF1  BsmtFinSF2  BsmtUnfSF  ...  SaleType_ConLw  \
Id                                                 ...
1        196.0       706.0         0.0      150.0  ...               0
2          0.0       978.0         0.0      284.0  ...               0
3        162.0       486.0         0.0      434.0  ...               0
4          0.0       216.0         0.0      540.0  ...               0
5        350.0       655.0         0.0      490.0  ...               0

    SaleType_New  SaleType_Oth  SaleType_WD  SaleCondition_Abnorml  \
Id
1              0             0            1                      0
2              0             0            1                      0
3              0             0            1                      0
4              0             0            1                      1
5              0             0            1                      0
```

```
    SaleCondition_AdjLand  SaleCondition_Alloca  SaleCondition_Family  \
Id
1                       0                     0                     0
2                       0                     0                     0
3                       0                     0                     0
4                       0                     0                     0
5                       0                     0                     0

    SaleCondition_Normal  SaleCondition_Partial
Id
1                      1                     0
2                      1                     0
3                      1                     0
4                      0                     0
5                      1                     0

[5 rows x 303 columns]
```

[345]:
```
all_dummy_df.isnull().sum().sort_values(ascending=False).head(10)
```

[345]:
```
LotFrontage     486
GarageYrBlt     159
MasVnrArea       23
BsmtHalfBath      2
BsmtFullBath      2
BsmtFinSF2        1
GarageCars        1
TotalBsmtSF       1
BsmtUnfSF         1
GarageArea        1
dtype: int64
```

[346]:
```
#计算平均值
mean_cols = all_dummy_df.mean()
mean_cols.head(10)
```

[346]:
```
LotFrontage        69.305795
LotArea         10168.114080
OverallQual         6.089072
OverallCond         5.564577
YearBuilt        1971.312778
YearRemodAdd     1984.264474
MasVnrArea        102.201312
BsmtFinSF1        441.423235
BsmtFinSF2         49.582248
BsmtUnfSF         560.772104
dtype: float64
```

[347]:
```
#用平均值填补缺失值
all_dummy_df = all_dummy_df.fillna(mean_cols)
#查看填补后是否还有缺失值
all_dummy_df.isnull().sum()#.sum()
```

[347]:
```
LotFrontage             0
LotArea                 0
OverallQual             0
OverallCond             0
YearBuilt               0
                       ..
SaleCondition_AdjLand   0
SaleCondition_Alloca    0
SaleCondition_Family    0
SaleCondition_Normal    0
SaleCondition_Partial   0
Length: 303, dtype: int64
```

[348]:
```
#查看哪些数据是数值型的
numeric_cols = all_df.columns[all_df.dtypes != 'object']
numeric_cols
```

[348]:
```
Index(['LotFrontage', 'LotArea', 'OverallQual', 'OverallCond', 'YearBuilt',
       'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinSF2', 'BsmtUnfSF',
       'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF', 'GrLivArea',
       'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath', 'BedroomAbvGr',
```

```
       'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces', 'GarageYrBlt',
       'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal',
       'MoSold', 'YrSold'],
      dtype='object')
```

[349]:
```
all_df.describe()
```

[349]:

	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	\
count	2433.000000	2919.000000	2919.000000	2919.000000	2919.000000	
mean	69.305795	10168.114080	6.089072	5.564577	1971.312778	
std	23.344905	7886.996359	1.409947	1.113131	30.291442	
min	21.000000	1300.000000	1.000000	1.000000	1872.000000	
25%	59.000000	7478.000000	5.000000	5.000000	1953.500000	
50%	68.000000	9453.000000	6.000000	5.000000	1973.000000	
75%	80.000000	11570.000000	7.000000	6.000000	2001.000000	
max	313.000000	215245.000000	10.000000	9.000000	2010.000000	

	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtUnfSF	...	\
count	2919.000000	2896.000000	2918.000000	2918.000000	2918.000000	...	
mean	1984.264474	102.201312	441.423235	49.582248	560.772104	...	
std	20.894344	179.334253	455.610826	169.205611	439.543659	...	
min	1950.000000	0.000000	0.000000	0.000000	0.000000	...	
25%	1965.000000	0.000000	0.000000	0.000000	220.000000	...	
50%	1993.000000	0.000000	368.500000	0.000000	467.000000	...	
75%	2004.000000	164.000000	733.000000	0.000000	805.500000	...	
max	2010.000000	1600.000000	5644.000000	1526.000000	2336.000000	...	

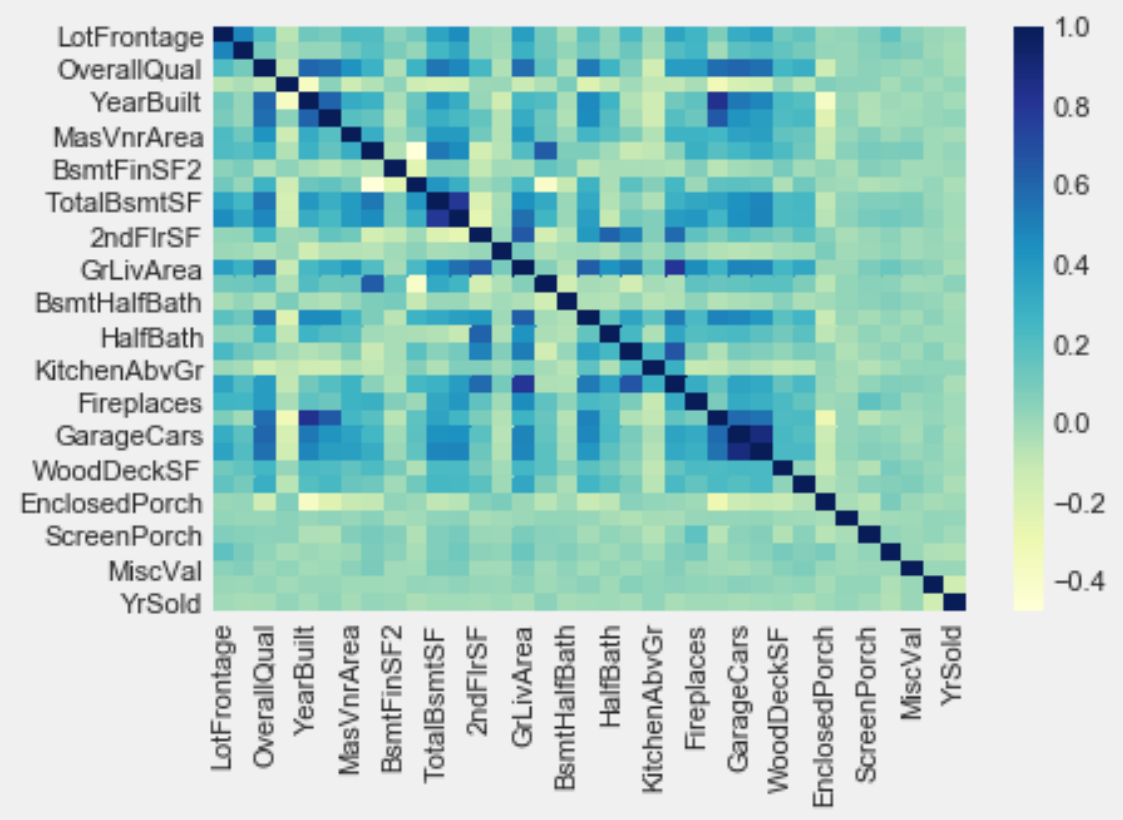
	GarageArea	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	\
count	2918.000000	2919.000000	2919.000000	2919.000000	2919.000000	
mean	472.874572	93.709832	47.486811	23.098321	2.602261	
std	215.394815	126.526589	67.575493	64.244246	25.188169	
min	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	320.000000	0.000000	0.000000	0.000000	0.000000	
50%	480.000000	0.000000	26.000000	0.000000	0.000000	
75%	576.000000	168.000000	70.000000	0.000000	0.000000	
max	1488.000000	1424.000000	742.000000	1012.000000	508.000000	

	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold
count	2919.000000	2919.000000	2919.000000	2919.000000	2919.000000
mean	16.062350	2.251799	50.825968	6.213087	2007.792737
std	56.184365	35.663946	567.402211	2.714762	1.314964
min	0.000000	0.000000	0.000000	1.000000	2006.000000
25%	0.000000	0.000000	0.000000	4.000000	2007.000000
50%	0.000000	0.000000	0.000000	6.000000	2008.000000
75%	0.000000	0.000000	0.000000	8.000000	2009.000000
max	576.000000	800.000000	17000.000000	12.000000	2010.000000

[8 rows x 35 columns]

[350]:
```
import seaborn as sns
import matplotlib.style as style
# 可视化观察变量之间的关系
sns.heatmap(all_df.corr(),cmap='YlGnBu')
```

[350]: <matplotlib.axes._subplots.AxesSubplot at 0x1bd8ced3208>



[351]:
```
dummy_train_df = all_dummy_df.loc[train_df.index]
dummy_test_df = all_dummy_df.loc[test_df.index]
#查看训练集和测试集的维度
dummy_train_df.shape, dummy_test_df.shape
```

[351]: ((1460, 303), (1459, 303))

[352]:
```
X_train,X_test,Y_train,Y_test = train_test_split(dummy_train_df.
↪values,y_train,train_size=.8)
print("原始数据特征:",dummy_train_df.shape,      ",训练数据特征:",X_train.shape,␣
↪      ",测试数据特征:",X_test.shape)
print("原始数据标签:",y_train.shape,      ",训练数据标签:",Y_train.shape,      ␣
↪",测试数据标签:",Y_test.shape)
```

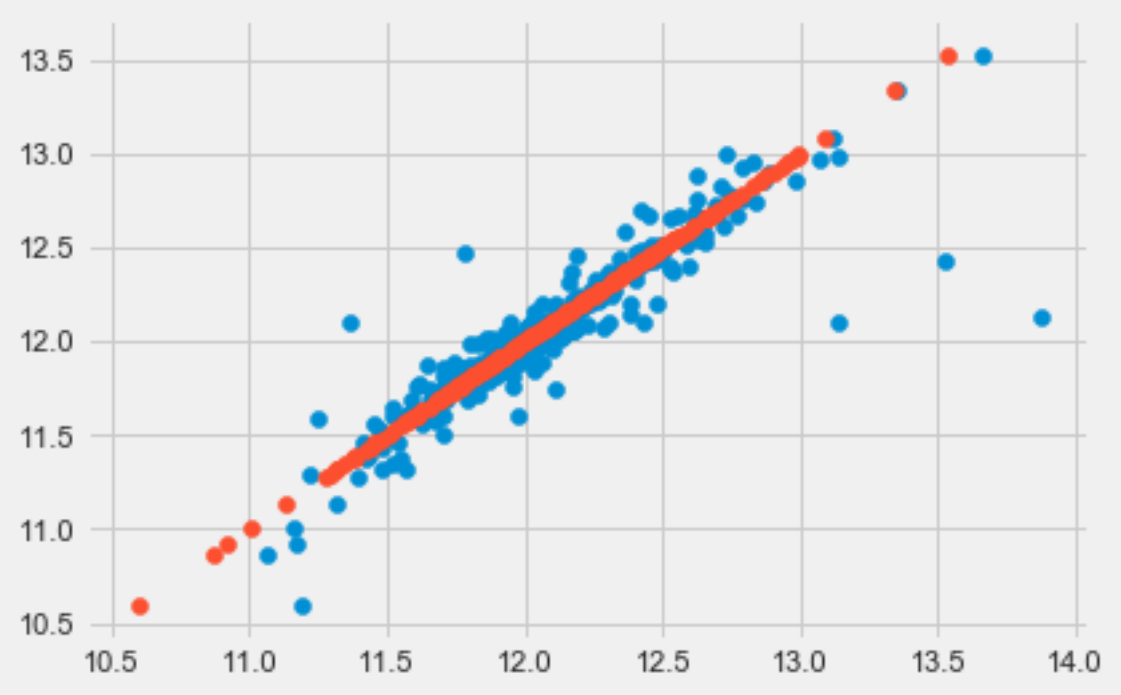
原始数据特征: (1460, 303) ,训练数据特征: (1168, 303) ,测试数据特征: (292, 303)
原始数据标签: (1460,) ,训练数据标签: (1168,) ,测试数据标签: (292,)

[317]:
```
X_Pred = dummy_test_df.values
```

[321]:
```
model.fit(X_train,Y_train)
Y_pre = model.predict(X_test)
```

3 程序运行结果及分析

[322]:
```
# 将预测值与目标值放入一张图进行比较。
plt.scatter(y_pre,Y_test,marker='o')
plt.scatter(Y_test,Y_test)
plt.show()
```



[323]:
```
# 得分
model.score(X_test, Y_test)
```

[323]: 0.80276550006996

4 知识点总结

- 数据预处理
- 多元回归模型的实现

4.2 问题二过拟合和欠拟合

【2】结合下图(a)和(b)解释什么是过拟合和欠拟合？通常用什么方法解决这两个问题？

回答：

欠拟合常常在模型学习能力较弱，而数据复杂度较高的情况出现，此时模型由于学习能力不足，无法学习到数据集中的“一般规律”，因而导致泛化能力弱。

- 增加新特征，可以考虑加入进特征组合、高次特征，来增大假设空间
- 添加多项式特征，这个在机器学习算法里面用的很普遍，例如将线性模型通过添加二次项或者三次项使模型泛化能力更强

- 减少正则化参数，正则化的目的是用来防止过拟合的，但是模型出现了欠拟合，则需要减少正则化参数
- 使用非线性模型，比如核SVM 、决策树、深度学习等模型
- 调整模型的容量(capacity)，通俗地，模型的容量是指其拟合各种函数的能力
- 容量低的模型可能很难拟合训练集；使用集成学习方法，如Bagging ,将多个弱学习器Bagging

过拟合常常在模型学习能力过强的情况中出现，此时的模型学习能力太强，以至于将训练集单个样本自身的特点都能捕捉到，并将其认为是“一般规律”，同样这种情况也会导致模型泛化能力下降。

- 数据层面：
  - 数据扩增，即增加训练数据样本，这是解决过拟合最有效的方法，只要给足够多的数据，让模型「看见」尽可能多的「例外情况」，它就会不断修正自己，从而得到更好的结果。
    * 从数据源头获取更多数据
    * 根据当前数据集估计数据分布参数，使用该分布产生更多数据：这个一般不用，因为估计分布参数的过程也会代入抽样误差
    * 数据增强（Data Augmentation）：通过一定规则扩充数据。如在物体分类问题里，物体在图像中的位置、姿态、尺度，整体图片明暗度等都不会影响分类结果。我们就可以通过图像平移、翻转、缩放、切割等手段将数据库成倍扩充
  - 特征工程，筛选组合得到更高质量的特征。
- 模型层面：
  - 正则化（Regularization）（L1和L2）以及树模型的剪枝策略，XGBoost中的正则项惩罚

  模型训练的过程中，需要降低 loss 以达到提高 accuracy 的目的。此时，使用正则化之类的方法直接将权值的大小加入到 loss 里，在训练的时候限制权值变大。训练过程需要降低整体的 loss，这时候，一方面能降低实际输出与样本之间的误差，也能降低权值大小正则化方法包括 L0 正则、 L1正则和 L2 正则，而正则一般是在目标函数之后加上范数。L2 范数是指向量各元素的平方和然后求平方根。可以使得 W 的每个元素都很小，都接近于0，但不会让它等于0，而是接近于0。 L2正则项起到使得参数 W 变小加剧的效果，关于它为什么能防止过拟合简答的理解为：更小的参数值 W意味着模型的复杂度更低，对训练数据的拟合刚刚好，不会过分拟合训练数据，从而使得不会过拟合，以提高模型的泛化能力。
  - 选择较为简单的模型
  - 集成学习，Bagging策略组合模型降低模型方差。

- 更多方法：
  - Dropout：在训练时，每次随机（如50%概率）忽略隐层的某些节点；这样，我们相当于随机从 2n(n个神经元的网络) 个模型中采样选择模型
  - Early stopping

Early stopping便是一种迭代次数截断的方法来防止过拟合的方法，即在模型对训练数据集迭代收敛之前停止迭代来防止过拟合。具体做法是，在每一个Epoch结束时计算validation data的accuracy，当accuracy不再提高时，就停止训练。当然我们并不会在accuracy一降低的时候就停止训练，因为可能经过这个Epoch后，accuracy降低了，但是随后的Epoch又让accuracy又上去了，所以不能根据一两次的连续降低就判断不再提高。一般的做法是，在训练的过程中，记录到目前为止最好的validation accuracy，当连续10次Epoch（或者更多次）没达到最佳accuracy时，则可以认为accuracy不再提高了。此时便可以停止迭代了（Early Stopping）。这种策略也称为“No-improvement-in-n”，n即Epoch的次数，可以根据实际情况取，如10、20、30.....

4.3 问题三鲍鱼年龄预测

【3】预测鲍鱼的年龄。网上下载“鲍鱼数据集”（见微信群，鲍鱼数据集.csv），建立线性回归模型，指出简单线性回归模型进行预测的问题，思考如何解决？

数据集字段说明

[325]:
```
import plotly
import plotly.graph_objs as go
```

[353]:
```
train_df = pd.read_csv('train_by.csv', index_col=0)
test_df = pd.read_csv('test_by.csv', index_col=0)
```

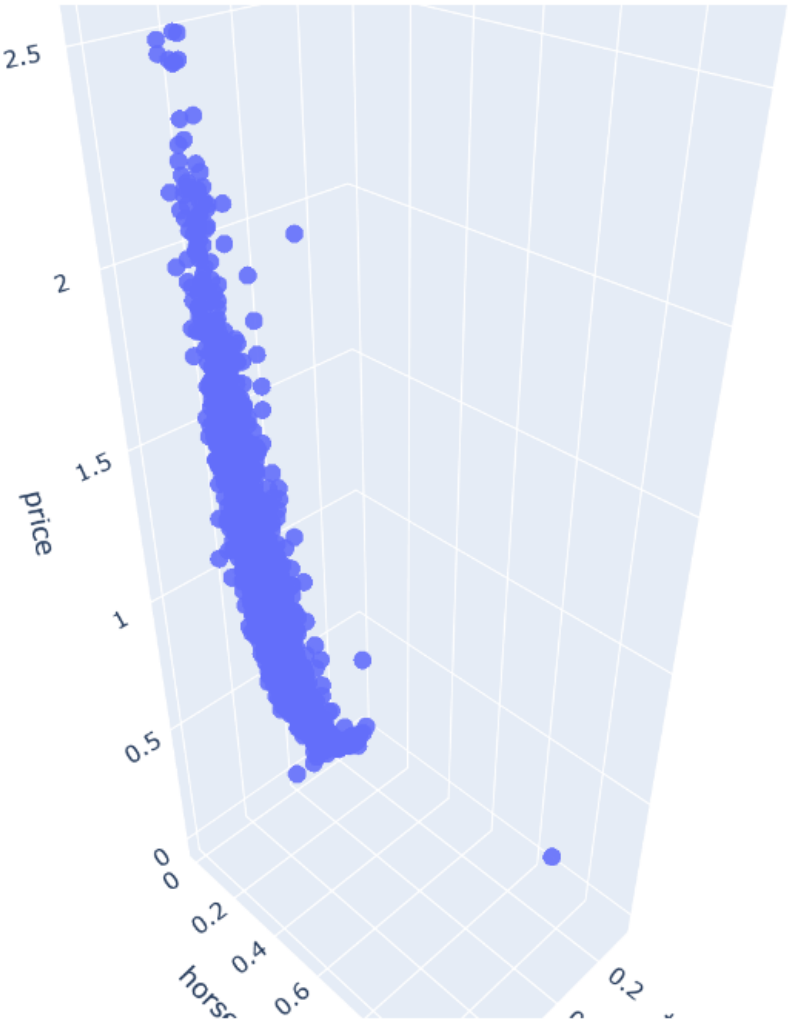
[354]:
```
fig1 = go.Scatter3d(x=train_df['Length'],
                    y=train_df['Height'],
                    z=train_df['Whole weight'],
                    marker=dict(opacity=0.9,
                                reversescale=True,
                                colorscale='Blues',
                                size=5),
                    line=dict (width=0.02),
                    mode='markers')

mylayout = go.Layout(scene=dict(xaxis=dict( title="curb-weight"),
```

```
                                yaxis=dict( title="horsepower"),
                                zaxis=dict(title="price")),)

plotly.offline.plot({"data": [fig1],
                     "layout": mylayout},
                     auto_open=True,
                     filename=("3DPlot.html"))
```

[354]: '3DPlot.html'



[355]:
```
train_df.head()
```

[355]:

Id	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
1	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
2	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
3	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
4	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
5	I	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	8

[356]:
```
test_df.head()
```

[356]:

Id	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight
3343	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055
3344	F	0.550	0.440	0.150	0.8945	0.3145	0.1510	0.320
3345	F	0.470	0.355	0.100	0.4755	0.1675	0.0805	0.185
3346	M	0.450	0.320	0.100	0.3810	0.1705	0.0750	0.115
3347	F	0.615	0.480	0.165	1.1615	0.5130	0.3010	0.305

[357]:
```
#将训练目标单独拿出
y_train = np.log1p(train_df.pop("Rings"))
all_df = pd.concat((train_df, test_df), axis=0)
```

```
all_df.shape
```

[357]: (4080, 8)

[358]:
```
#Sex被分成了3个column，每一个代表一个类。是就是1，不是就是0。
pd.get_dummies(all_df['Sex'], prefix='Sex').head()
```

[358]:

Id	Sex_F	Sex_I	Sex_M
1	0	0	1
2	0	0	1
3	1	0	0
4	0	0	1
5	0	1	0

[359]:
```
all_dummy_df = pd.get_dummies(all_df)
all_dummy_df.head()
```

[359]:

Id	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Sex_F	Sex_I	Sex_M
1	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	0	0	1
2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	0	0	1
3	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	1	0	0
4	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	0	0	1
5	0.425	0.300	0.095	0.3515	0.1410	0.0775	0.120	0	1	0

[360]:
```
all_dummy_df.isnull().sum().sort_values(ascending=False).head(10)
```

[360]:
```
Sex_M             0
Sex_I             0
Sex_F             0
```

```
Shell weight      0
Viscera weight    0
Shucked weight    0
Whole weight      0
Height            0
Diameter          0
Length            0
dtype: int64
```

[361]:
```
all_dummy_df.describe()
```

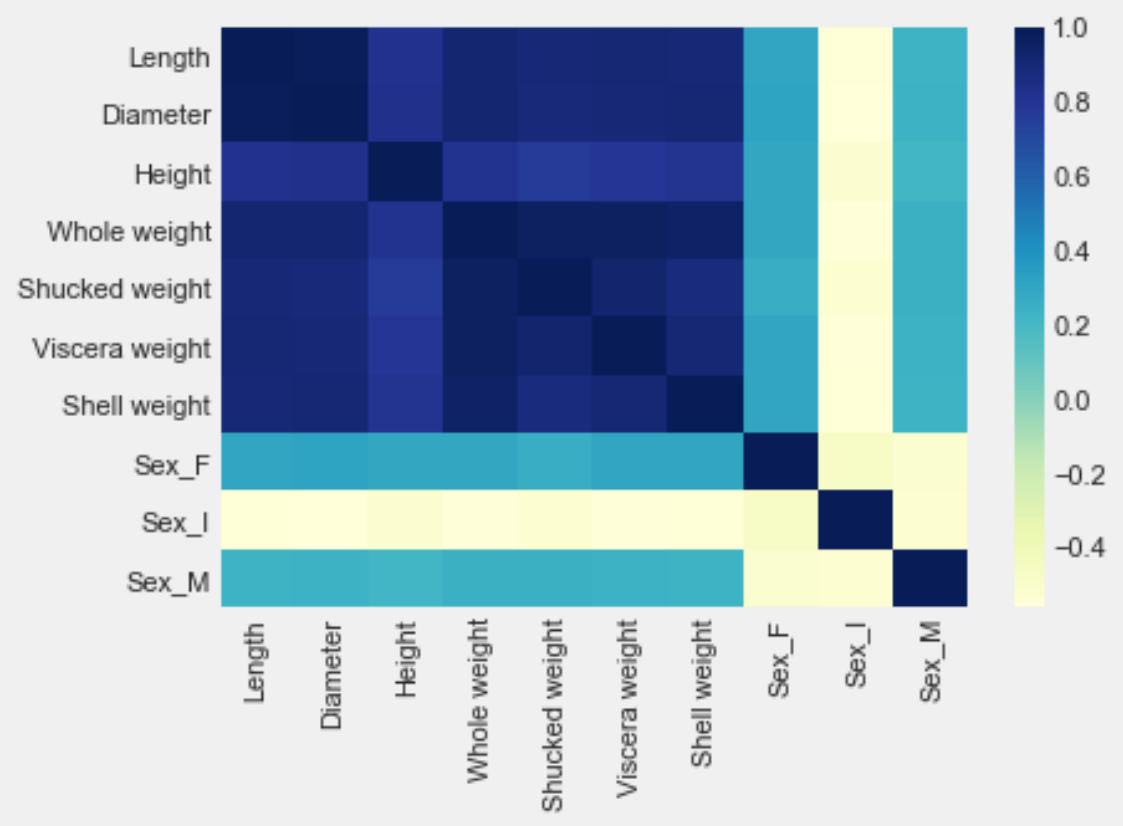
[361]:

	Length	Diameter	Height	Whole weight	Shucked weight	\
count	4080.000000	4080.000000	4080.000000	4080.000000	4080.000000	
mean	0.523960	0.407908	0.139507	0.828630	0.359151	
std	0.119964	0.099141	0.041883	0.490842	0.222106	
min	0.075000	0.055000	0.000000	0.002000	0.001000	
25%	0.450000	0.350000	0.115000	0.441375	0.186000	
50%	0.545000	0.425000	0.140000	0.799750	0.336000	
75%	0.615000	0.480000	0.165000	1.151500	0.500625	
max	0.815000	0.650000	1.130000	2.825500	1.488000	

	Viscera weight	Shell weight	Sex_F	Sex_I	Sex_M
count	4080.000000	4080.000000	4080.000000	4080.000000	4080.000000
mean	0.180484	0.238873	0.313480	0.321569	0.364951
std	0.109632	0.139466	0.463965	0.467136	0.481475
min	0.000500	0.001500	0.000000	0.000000	0.000000
25%	0.093000	0.130000	0.000000	0.000000	0.000000
50%	0.170500	0.232750	0.000000	0.000000	0.000000
75%	0.252500	0.328625	1.000000	1.000000	1.000000
max	0.760000	1.005000	1.000000	1.000000	1.000000

[362]:
```
import seaborn as sns
import matplotlib.style as style
sns.heatmap(all_dummy_df.corr(),cmap='YlGnBu')
```

[362]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd8cee0308>
```



[363]:
```
dummy_train_df = all_dummy_df.loc[train_df.index]
dummy_test_df = all_dummy_df.loc[test_df.index]
```

[364]:
```
X_train = dummy_train_df.values
# X_test = dummy_test_df.values
X_train,X_test,Y_train,Y_test = train_test_split(dummy_train_df.
↪values,y_train,train_size=.8)
print("原始数据特征:",dummy_train_df.shape,      ",训练数据特征:",X_train.shape,␣
↪      ",测试数据特征:",X_test.shape)
print("原始数据标签:",y_train.shape,      ",训练数据标签:",Y_train.shape,      ␣
↪",测试数据标签:",Y_test.shape)
```

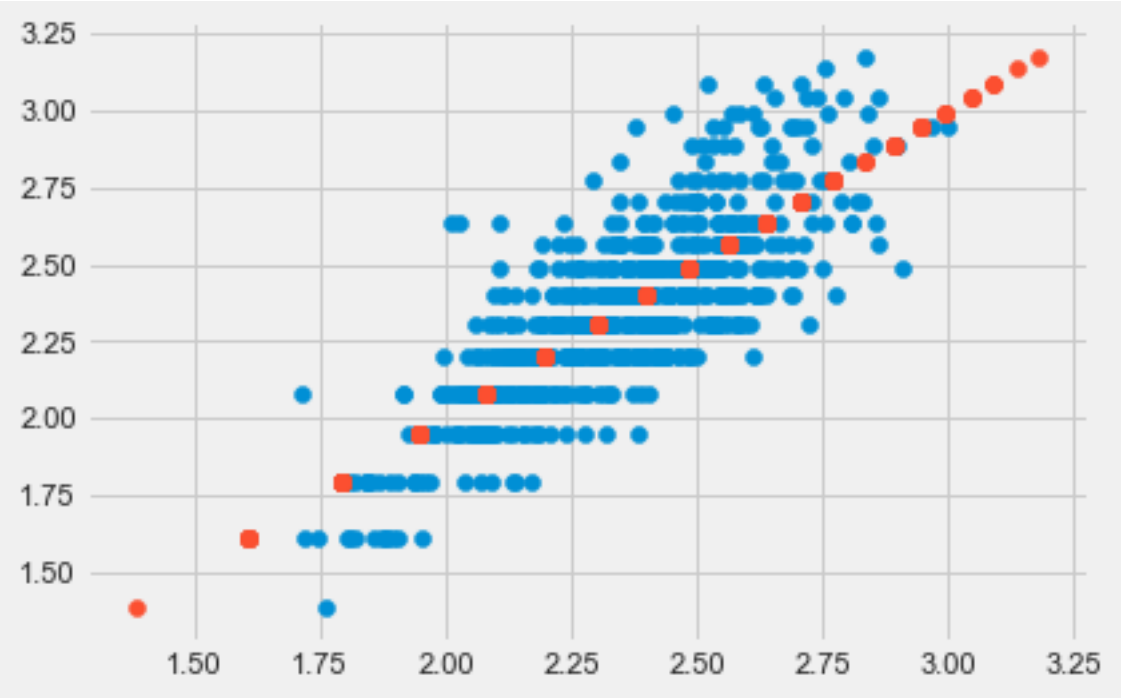
```
原始数据特征: (3342, 10) ,训练数据特征: (2673, 10) ,测试数据特征: (669, 10)
原始数据标签: (3342,) ,训练数据标签: (2673,) ,测试数据标签: (669,)
```

[368]:
```
model.fit(X_train,Y_train)
Y_pre = model.predict(X_test)
```

[369]:
```
a  = model.intercept_
b = model.coef_
print("最佳拟合线:截距",a,",回归系数: ",b)
```

最佳拟合线:截距 624148452518.2838 ,回归系数： [3.75423803e-01 1.39775905e+00
8.90195444e-01 5.46505689e-01
 -1.50963697e+00 -7.08881891e-01 5.58664600e-01 -6.24148453e+11
 -6.24148453e+11 -6.24148453e+11]

[371]:
```
plt.scatter(Y_pre,Y_test,marker='o')
plt.scatter(Y_test,Y_test)
plt.show()
```



- 问题：欠拟合
- 解决办法：局部加权回归（更复杂的模型）

5 作业清单（5/6）

5.1 问题一数据清洗

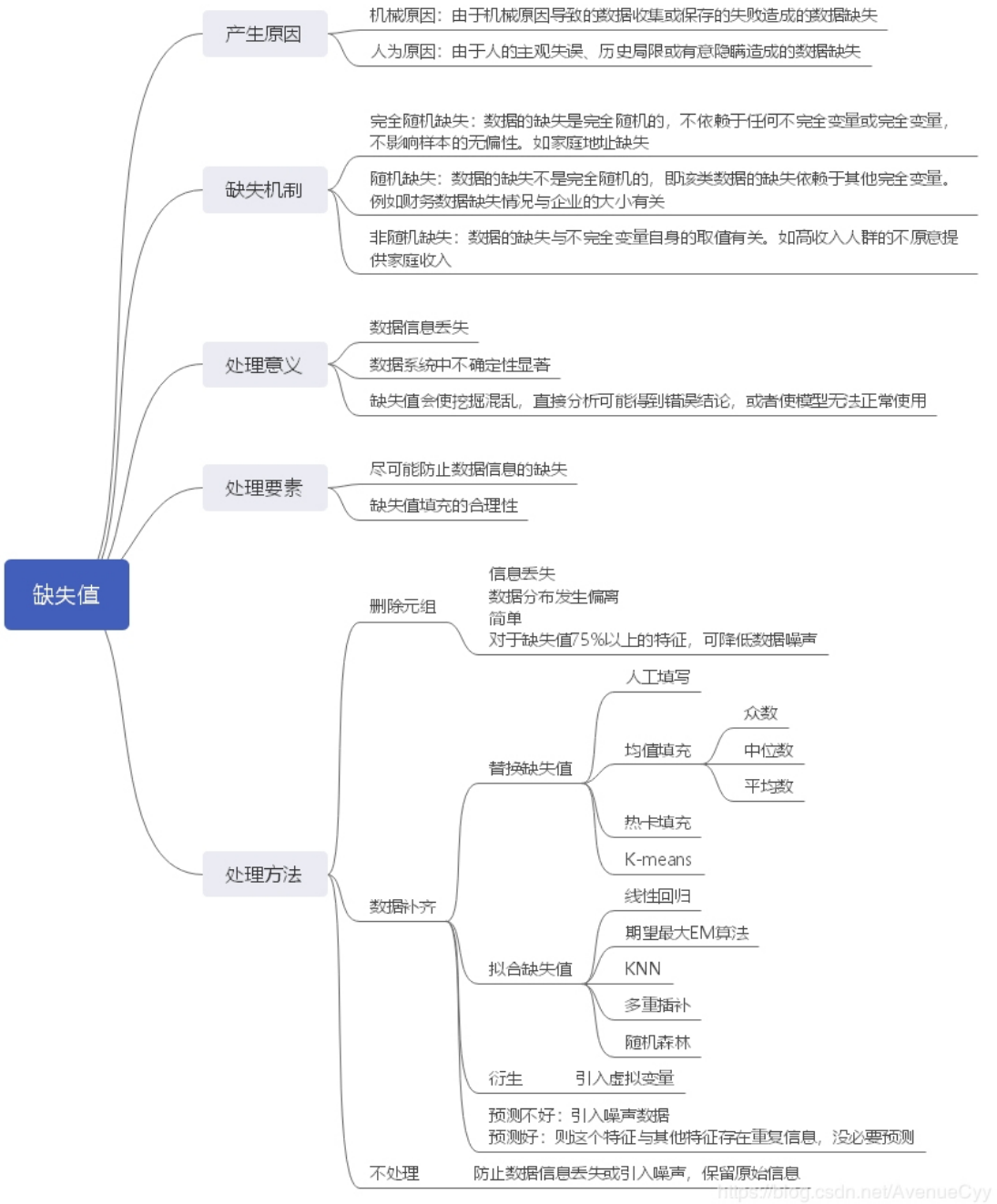
【1】数据清洗是数据挖掘模型建立过程中很重要的一步吗一般，清洗的方法包括什么？

数据清洗是数据挖掘模型建立过程中很重要的一步。数据清理方法

通过填写缺失的值，光滑噪声数据，识别或删除离群点，并解决不一致性来“清理”数据。如果用户认为数据是脏的，则他们可能不会相信这些数据上的挖掘结果。此外，脏数据可能使挖掘过程陷入混乱，导致不可靠的输出。

5.1.1 缺失值

- **忽略元组：**当缺少类标号时通常这样做（假定挖掘任务涉及分类）。除非元组有多个属性缺少值，否则该方法不是很有效。当每个属性缺失值的百分比变化很大时，它的性能特别差。采用忽略元组，你不能使用该元组的剩余属性值。这些数据可能对手头的任务是有用的。
- **人工填写缺失值：**一般来说，该方法很费时，并且当数据集很大、确实很多值时，该方法可能行不通。
- **使用一个全局常量填充缺失值：**将缺失的属性值用同一个常量（如“Unknown”或 $-\infty$）替换。如果缺失值都用“Unknown”替换，则挖掘程序可能误以为他们形成了一个有趣的概念，因为它们都具有相同的值---“Unknown”。因此，尽管该方法简单，但并不十分可靠。
- **使用属性的中心度量（如均值或中位数）填充缺失值：**均值等中心趋势度量，指示数据分布的“中间”值。对于正常的（对称的）数据分布而言，可以使用均值，而倾斜数据分布应该使用中位数。例如，收入的数据是对称的，并且平均收入为 56,000 美元，则使用该值替换收入属性中的缺失值。
- **使用与给定元组属同一类的所有样本的属性均值或中位数：**例如，将顾客按信用等级分类，则用具有相同信用风险的顾客的平均收入替换收入属性中的缺失值。如果给定类的数据分布是倾斜的，则中位数是更好的选择。
- **使用最可能的值填充缺失值：**可以用回归、使用贝叶斯形式化方法的基于推理的工具或决策树归纳确定。例如，利用数据集中其他顾客的属性，可以构造一颗决策树，来预测收入的缺失值。



5.1.2 噪声数据

噪声是被测量的变量的随机误差或方差。

数据光滑技术： - **分箱：**通过考察数据的‘‘近邻’’（即周围的值）来光滑有序数据值。这些有序的值被分布到一些‘‘桶’’或箱中。由于分箱方法考察临近的值，因此它进行局部光滑。 - **回归：**可以用一个函数拟合数据来光滑数据。 - **离群点分析：**可以通过如聚类来检测离群点。聚类将类似的值组织成群或‘‘簇’’。

5.2 问题二删除和填补

【2】对下图的数据采用删除和填补两种方法进行清洗。

[ ]:
```
import pandas as pd
```

[377]:
```
data=pd.read_csv("homework_3.csv")
data.head()
```

[377]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	NaN	0.2	setosa
2	4.7	3.2	NaN	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.1	3.8	NaN	0.2	setosa

[378]:
```
data.describe()
```

[378]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width
count	14.000000	14.000000	9.000000	15.000000
mean	5.500000	3.171429	3.177778	0.973333
std	0.775589	0.267261	2.090322	0.889194
min	4.600000	2.700000	1.400000	0.200000
25%	4.925000	3.000000	1.400000	0.200000
50%	5.200000	3.200000	1.500000	0.200000
75%	6.200000	3.275000	5.100000	1.800000
max	7.000000	3.800000	6.000000	2.500000

[379]:
```
data.isnull().sum().sort_values(ascending=False)
```

[379]:
```
Pepal.Length    6
Sepal.Width     1
Sepal.Length    1
Species         0
Pepal.Width     0
dtype: int64
```

5.2.1 删除

[380]:
```
# 删除 nan 所在行
data1=data.dropna()
data1
```

[380]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
5	4.6	3.2	1.4	0.2	setosa
7	5.0	3.3	1.4	0.2	setosa
8	7.0	3.2	4.7	1.4	versicolor
10	6.3	3.3	6.0	2.5	virginica
11	5.8	2.7	5.1	1.9	virginica
13	6.3	2.9	5.6	1.8	virginica

[381]:
```
# 删除 nan 所在列
data2=data.dropna(axis=1)
data2
```

[381]:

	Pepal.Width	Species
0	0.2	setosa
1	0.2	setosa
2	0.2	setosa
3	0.2	setosa
4	0.2	setosa
5	0.2	setosa
6	0.2	setosa
7	0.2	setosa
8	1.4	versicolor
9	1.5	versicolor
10	2.5	virginica
11	1.9	virginica
12	2.1	virginica
13	1.8	virginica
14	1.8	virginica

5.2.2 填补

[382]:
```
# 用0填充缺省值
data3=data.fillna(0)
data3
```

[382]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	0.0	0.2	setosa
2	4.7	3.2	0.0	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.1	3.8	0.0	0.2	setosa
5	4.6	3.2	1.4	0.2	setosa
6	5.3	0.0	1.5	0.2	setosa
7	5.0	3.3	1.4	0.2	setosa
8	7.0	3.2	4.7	1.4	versicolor
9	6.4	3.2	0.0	1.5	versicolor
10	6.3	3.3	6.0	2.5	virginica
11	5.8	2.7	5.1	1.9	virginica
12	0.0	3.0	0.0	2.1	virginica
13	6.3	2.9	5.6	1.8	virginica
14	5.9	3.0	0.0	1.8	virginica

[383]:
```
# 用上一个数据填充缺省值
data4=data.fillna(method="pad")
data4
```

[383]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.4	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.1	3.8	1.5	0.2	setosa
5	4.6	3.2	1.4	0.2	setosa
6	5.3	3.2	1.5	0.2	setosa
7	5.0	3.3	1.4	0.2	setosa
8	7.0	3.2	4.7	1.4	versicolor
9	6.4	3.2	4.7	1.5	versicolor

10	6.3	3.3	6.0	2.5	virginica
11	5.8	2.7	5.1	1.9	virginica
12	5.8	3.0	5.1	2.1	virginica
13	6.3	2.9	5.6	1.8	virginica
14	5.9	3.0	5.6	1.8	virginica

[384]:
```
# 用平均值填充缺省值
data5=data.fillna(data.mean())
data5
```

[384]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.500000	1.400000	0.2	setosa
1	4.9	3.000000	3.177778	0.2	setosa
2	4.7	3.200000	3.177778	0.2	setosa
3	4.6	3.100000	1.500000	0.2	setosa
4	5.1	3.800000	3.177778	0.2	setosa
5	4.6	3.200000	1.400000	0.2	setosa
6	5.3	3.171429	1.500000	0.2	setosa
7	5.0	3.300000	1.400000	0.2	setosa
8	7.0	3.200000	4.700000	1.4	versicolor
9	6.4	3.200000	3.177778	1.5	versicolor
10	6.3	3.300000	6.000000	2.5	virginica
11	5.8	2.700000	5.100000	1.9	virginica
12	5.5	3.000000	3.177778	2.1	virginica
13	6.3	2.900000	5.600000	1.8	virginica
14	5.9	3.000000	3.177778	1.8	virginica

5.3 问题三回归方法填补

【3】对【2】题数据采用回归方法填补。

5.4 背景描述：

数据清洗过程中经常会遇到异常值和缺失值等问题，有时候，会把异常值看作缺失值来处理。一般的缺失值处理方法包括：删除、统计值充填（均值、中位数等）、回归方程预测充填等。使用直接删除这种方法简单易行，但缺点是，在记录数据较少的情况下，会造成样本量的进一步减少，可能会改变响应变量的原有分布，造成分析结果不准确。因此，将异常值视为缺失值来处理的益处在于可以利用现有变量的信息进行建模挖掘，对异常值（缺失值）进行填补。

5.5 应用场景：

回归方程充填法是选择若干能预测缺失值的自变量，通过建立回归方程估算缺失值。

该方法能尽可能地利用原数据集中的信息，但也存在一些不足之处： - 虽然这是一个无偏估计，但会忽视随机误差，低估标准差和其他未知性质的测量值； - 使用前，必须假设存在缺失值所在的变量与其他变量是存在线性关系的,但现实它们不一定存在这样的线性关系，这可以借助统计工具来辨析，但往往更需要建模人员的实践经验和业务知识来进行分析和判断。

5.6 方法步骤：

- 确定充填缺失值的变量（特征列）
- 拆分原始数据集：根据需要充填缺失值的变量，把原始数据集拆分为2个子集
  - 不含有缺失值：dataset_train
  - 只含有缺失值：dataset_pred
- 辨析并检验相关变量的相关性：经验分析判定与充填缺失值的变量相关的属性列有哪些，应用统计分析工具，在dataset_train数据集上查看验证所选择的属性列之间的相关性。
- 建模并预测：使用dataset_train数据集建立线性回归模型，并应用建好的模型对dataset_pred数据集中的缺失变量进行预测估计
- 合并还原数据集：将两个子集合并还原为一个数据集，为后续建模准备好数据。

[386]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15 entries, 0 to 14
Data columns (total 5 columns):
Sepal.Length    14 non-null float64
Sepal.Width     14 non-null float64
Pepal.Length    9 non-null float64
Pepal.Width     15 non-null float64
Species         15 non-null object
dtypes: float64(4), object(1)
memory usage: 728.0+ bytes
```

[387]:
```
data.isnull().sum().sort_values(ascending=False)
```

[387]:
```
Pepal.Length    6
Sepal.Width     1
```

```
Sepal.Length    1
Species         0
Pepal.Width     0
dtype: int64
```

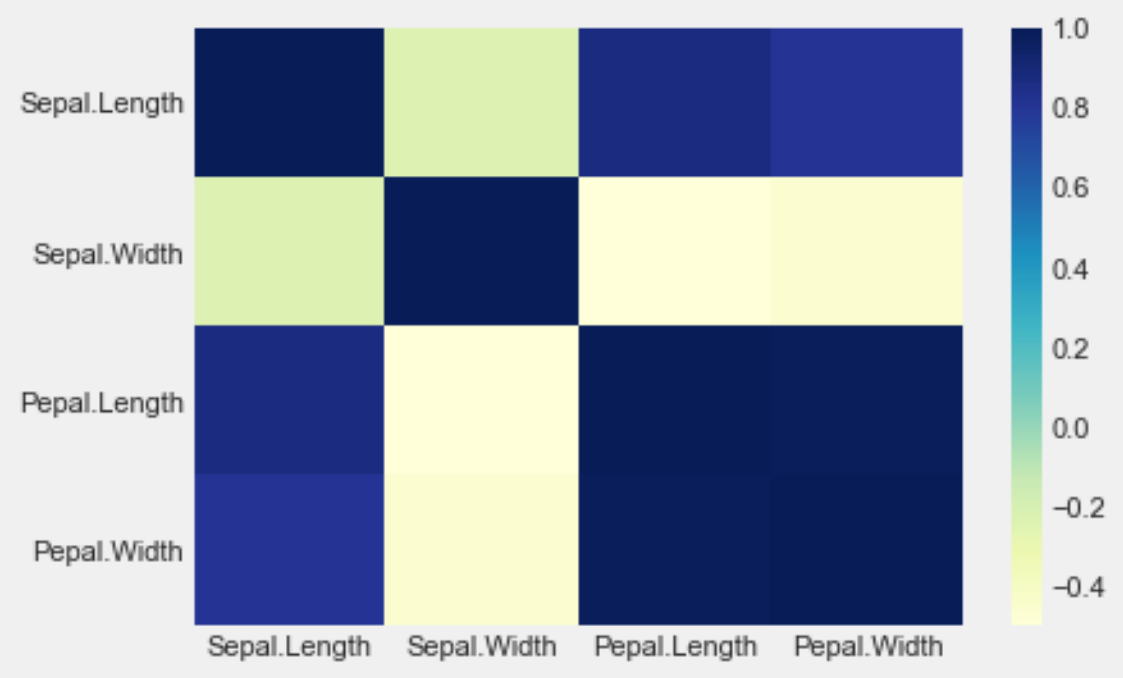
[388]:
```
# 观察变量之间的相关性
dataset_train=data.dropna()
dataset_train
```

[388]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
5	4.6	3.2	1.4	0.2	setosa
7	5.0	3.3	1.4	0.2	setosa
8	7.0	3.2	4.7	1.4	versicolor
10	6.3	3.3	6.0	2.5	virginica
11	5.8	2.7	5.1	1.9	virginica
13	6.3	2.9	5.6	1.8	virginica

[389]:
```
import seaborn as sns
import matplotlib.style as style
# style.use("fivethirtyeight")
sns.heatmap(dataset_train.corr(),cmap='YlGnBu')
```

[389]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1bd8e41e088>
```



通过对数据的观察，发现 Pepal.Length 这一属性缺失数据最多，且与其他属性有一定的相关性，故先通过回归法填充该属性。

[390]:
```
from sklearn.linear_model import LinearRegression
import numpy as np
line_reg = LinearRegression()
tmp=dataset_train
y_train=np.log1p(tmp.pop("Pepal.Length"))
x_train=tmp
x_train
```

[390]:

	Sepal.Length	Sepal.Width	Pepal.Width	Species
0	5.1	3.5	0.2	setosa
3	4.6	3.1	0.2	setosa
5	4.6	3.2	0.2	setosa
7	5.0	3.3	0.2	setosa
8	7.0	3.2	1.4	versicolor
10	6.3	3.3	2.5	virginica
11	5.8	2.7	1.9	virginica
13	6.3	2.9	1.8	virginica

[391]:
```
# 独热编码
pd.get_dummies(x_train['Species'], prefix='Species').head()
```

[391]:

	Species_setosa	Species_versicolor	Species_virginica
0	1	0	0
3	1	0	0
5	1	0	0
7	1	0	0
8	0	1	0

[392]:
```
x_train=pd.get_dummies(x_train)
x_train
```

[392]:

	Sepal.Length	Sepal.Width	Pepal.Width	Species_setosa	\
0	5.1	3.5	0.2	1	
3	4.6	3.1	0.2	1	
5	4.6	3.2	0.2	1	
7	5.0	3.3	0.2	1	
8	7.0	3.2	1.4	0	
10	6.3	3.3	2.5	0	
11	5.8	2.7	1.9	0	
13	6.3	2.9	1.8	0	

	Species_versicolor	Species_virginica
0	0	0
3	0	0
5	0	0
7	0	0
8	1	0
10	0	1
11	0	1
13	0	1

[393]:
```
line_reg.fit(x_train,y_train)
```

[393]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

[395]:
```
line_reg.intercept_
```

[395]:
```
0.9093043320607699
```

[396]:
```
line_reg.coef_
```

[396]:
```
array([ 0.15633679, -0.1995122 ,  0.23313542, -0.17117971,  0.0488538 ,
        0.12232591])
```

[401]:
```
# 预测
data_pred=data[np.isnan(data["Pepal.Length"])]
data_pred=data_pred.dropna(subset=["Sepal.Width","Sepal.Length"])
data_pred
```

[401]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
1	4.9	3.0	NaN	0.2	setosa
2	4.7	3.2	NaN	0.2	setosa
4	5.1	3.8	NaN	0.2	setosa
9	6.4	3.2	NaN	1.5	versicolor
14	5.9	3.0	NaN	1.8	virginica

[398]:
```
X_pred=pd.get_dummies(data_pred)
np.log1p(X_pred.pop("Pepal.Length"))
y_pred=line_reg.predict(X_pred)
y_pred
```

[398]:
```
array([0.95226535, 0.88109555, 0.82392295, 1.66997764, 1.77512442])
```

[399]:
```
data_pred["Pepal.Length"]=y_pred
data_pred
```

[399]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
1	4.9	3.0	0.952265	0.2	setosa
2	4.7	3.2	0.881096	0.2	setosa
4	5.1	3.8	0.823923	0.2	setosa
9	6.4	3.2	1.669978	1.5	versicolor
14	5.9	3.0	1.775124	1.8	virginica

[400]:
```
# 合并数据
dataset_train=data.dropna()
dataset_train
data_new=dataset_train.append(data_pred).sort_index()
```

```
data_new
```

[400]:

	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
0	5.1	3.5	1.400000	0.2	setosa
1	4.9	3.0	0.952265	0.2	setosa
2	4.7	3.2	0.881096	0.2	setosa
3	4.6	3.1	1.500000	0.2	setosa
4	5.1	3.8	0.823923	0.2	setosa
5	4.6	3.2	1.400000	0.2	setosa
7	5.0	3.3	1.400000	0.2	setosa
8	7.0	3.2	4.700000	1.4	versicolor
9	6.4	3.2	1.669978	1.5	versicolor
10	6.3	3.3	6.000000	2.5	virginica
11	5.8	2.7	5.100000	1.9	virginica
13	6.3	2.9	5.600000	1.8	virginica
14	5.9	3.0	1.775124	1.8	virginica

6 作业清单（5/11）

6.1 问题一 Pandas 基础

【1】 Pandas Series是什么？ Pandas中的DataFrame是什么？如何将numpy数据转成DataFrame格式的数据？如何将Series数据转成DataFrame格式的数据？如何将DataFrame转换为NumPy数组？如何对DataFrame进行排序？什么是数据聚合？（注：每一小问，举例说明）

6.1.1 Series 对象

Pandas 的 Series 对象是一个带索引数据构成的一维数组。可以用一个**数组**创建 Series 对象，如下所示：

[402]:
```
import pandas as pd
data = pd.Series([0.25, 0.5, 0.75, 1.0])
data
```

[402]:
```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

通过上面的例子发现 Series 对象将一组数据和一组索引绑定在一起，可以通过 values 属性和 index 属性获取数据。index 属性返回的结果是一个类型为pd.Index 的类数组对象，values 属性返回的结果与NumPy 数组类似。

[403]:
```
data.index
```

[403]:
```
RangeIndex(start=0, stop=4, step=1)
```

[404]:
```
data.values
```

[404]:
```
array([0.25, 0.5 , 0.75, 1.  ])
```

和 NumPy 数组一样，数据可以通过 Python 的中括号索引标签获取：

[405]:
```
data[1]
```

[405]:
```
0.5
```

[406]:
```
data[1:3]
```

[406]:
```
1    0.50
2    0.75
dtype: float64
```

NumPy 数组通过隐式定义的整数索引获取数值，而 Pandas 的 Series 对象用一种显式定义的索引与数值关联。显式索引的定义让 Series 对象拥有了更强的能力。例如，索引不再仅仅是整数，还可以是任意想要的类型。如果需要，完全可以用字符串定义索引：

[407]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0],index=['a', 'b', 'c', 'd'])
data
```

[407]:
```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

[409]:
```
data['b']
```

[409]:
```
0.5
```

[410]:
```
# 也可以使用不连续或不按顺序的索引：
data = pd.Series([0.25, 0.5, 0.75, 1.0],index=[2, 5, 3, 7])
```

```
data
```

[410]:
```
2    0.25
5    0.50
3    0.75
7    1.00
dtype: float64
```

还可以直接用 Python 的**字典**创建一个 Series 对象，用字典创建 Series 对象时，其索引默认按照顺序排列。

[411]:
```
population_dict = {'California': 38332521,
'Texas': 26448193,
'New York': 19651127,
'Florida': 19552860,
'Illinois': 12882135}
# Series 对象
population = pd.Series(population_dict)
population
```

[411]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

[412]:
```
population['California']
```

[412]:
```
38332521
```

[413]:
```
population['California':'Illinois']
```

[413]:
```
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
dtype: int64
```

我们已经见过几种创建 Pandas 的 Series 对象的方法，都是像这样的形式：

$$pd.Series(data, index = index)$$

其中，index 是一个可选参数，data 参数支持多种数据类型。

例如，data 可以是列表或 NumPy 数组，这时 index 默认值为整数序列：

[416]:
```
pd.Series([2, 4, 6])
```

[416]:
```
0    2
1    4
2    6
dtype: int64
```

data 也可以是一个标量，创建 Series 对象时会重复填充到每个索引上：

[417]:
```
pd.Series(5, index=[100, 200, 300])
```

[417]:
```
100    5
200    5
300    5
dtype: int64
```

data 还可以是一个字典，index 默认是排序的字典键：

[418]:
```
pd.Series({2:'a', 1:'b', 3:'c'})
```

[418]:
```
2    a
1    b
3    c
dtype: object
```

每一种形式都可以通过显式指定索引筛选需要的结果：

[419]:
```
pd.Series({2:'a', 1:'b', 3:'c'}, index=[3, 2])
```

[419]:
```
3    c
2    a
dtype: object
```

6.1.2 DataFrame 对象

如果将 Series 类比为带灵活索引的一维数组，那么 DataFrame 就可以看作是一种既有灵活的行索引，又有灵活列名的二维数组。就像你可以把二维数组看成是有序排列的一维数组一样，你也可以把 DataFrame 看成是有序排列的若干 Series 对象。这里的“排列”指的是它们拥有共同的索引。

[420]:
```
area_dict = {'California': 423967, 'Texas': 695662, 'New York':␣
↪141297,'Florida': 170312, 'Illinois': 149995}
area = pd.Series(area_dict)
area
```

[420]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
dtype: int64
```

[421]:
```
states = pd.DataFrame({'population': population,'area': area})
states
```

[421]:
```
            population    area
California    38332521  423967
Texas         26448193  695662
New York      19651127  141297
Florida       19552860  170312
Illinois      12882135  149995
```

和 Series 对象一样，DataFrame 也有一个 index 属性可以获取索引标签：

[422]:
```
states.index
```

[422]:
```
Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'],
dtype='object')
```

另外，DataFrame 还有一个 columns 属性，是存放列标签的 Index 对象：

[423]:
```
states.columns
```

[423]:
```
Index(['population', 'area'], dtype='object')
```

因此DataFrame 可以看作一种通用的NumPy 二维数组，它的行与列都可以通过索引获取。

[424]:
```
states['area']
```

[424]:
```
California    423967
Texas         695662
New York      141297
Florida       170312
Illinois      149995
Name: area, dtype: int64
```

[425]:
```
print(type(states['area']))
```

```
<class 'pandas.core.series.Series'>
```

DataFrame 对象可以通过许多方式创建，这里举几个常用的例子。

- 通过单个 Series 对象创建。DataFrame 是一组 Series 对象的集合，可以用单个 Series创建一个单列的 DataFrame:

[426]:
```
pd.DataFrame(population, columns=['population'])
```

[426]:
```
            population
California    38332521
Texas         26448193
New York      19651127
Florida       19552860
Illinois      12882135
```

- 通过字典列表创建。任何元素是字典的列表都可以变成 DataFrame。用一个简单的列表综合来创建一些数据:

[427]:
```
data = [{'a': i, 'b': 2 * i} for i in range(3)]
pd.DataFrame(data)
```

[427]:

	a	b
0	0	0
1	1	2
2	2	4

即使字典中有些键不存在，Pandas 也会用缺失值 NaN 来表示:

[428]:
```
pd.DataFrame([{'a': 1, 'b': 2}, {'b': 3, 'c': 4}])
```

[428]:

	a	b	c
0	1.0	2	NaN
1	NaN	3	4.0

- 通过 Series 对象字典创建。DataFrame 可以用一个由 Series 对象构成的字典创建：

[429]:
```
pd.DataFrame({'population': population,'area': area})
```

[429]:

	population	area
California	38332521	423967
Texas	26448193	695662
New York	19651127	141297
Florida	19552860	170312
Illinois	12882135	149995

- 通过 NumPy 二维数组创建。假如有一个二维数组，就可以创建一个可以指定行列索引值的 DataFrame。如果不指定行列索引值，那么行列默认都是整数索引值：

[430]:
```
import numpy as np
pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],index=['a', 'b', 'c'])
```

[430]:

	foo	bar
a	0.065071	0.381762
b	0.209573	0.339590
c	0.499499	0.665693

- 通过 NumPy 结构化数组创建。由于 Pandas 的 DataFrame与结构化数组十分相似，因此可以通过结构化数组创建 DataFrame：

[431]:
```
A = np.zeros(3, dtype=[('A', 'i8'), ('B', 'f8')])
A
```

[431]:
```
array([(0, 0.), (0, 0.), (0, 0.)], dtype=[('A', '<i8'), ('B', '<f8')])
```

6.1.3 DataFrame 转换

将Pandas中的DataFrame转换成Numpy中数组三种方法：

[433]:
```
pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],index=['a', 'b', 'c']).
↪values
```

[433]:
```
array([[0.60178555, 0.95176841],
       [0.46533468, 0.54044558],
       [0.1926436 , 0.26596257]])
```

[434]:
```
pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],index=['a', 'b', 'c']).
↪as_matrix()
```

```
c:\programs\python-3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning:

Method .as_matrix will be removed in a future version. Use .values instead.
```

[434]:
```
array([[0.93181914, 0.2684139 ],
       [0.66502241, 0.37798961],
       [0.37269048, 0.1263394 ]])
```

[435]:
```
np.array(pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],index=['a',␣
↪'b', 'c']))
```

[435]:
```
array([[0.44492152, 0.22325347],
       [0.82401279, 0.49230146],
       [0.08015841, 0.75963855]])
```

6.1.4 对DataFrame 排序

[436]:
```
pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],index=['b', 'a', 'c']).
↪sort_index()
```

[436]:

	foo	bar
a	0.594649	0.038893
b	0.110420	0.060037
c	0.064105	0.216973

[437]:
```
pd.DataFrame(np.random.rand(3, 2),columns=['foo', 'bar'],index=['a', 'b', 'c']).
↪sort_values(by="foo")
```

[437]:

	foo	bar
c	0.060045	0.647899
a	0.515206	0.025026
b	0.967830	0.802893

6.1.5 数据聚合

面对大量的数据时，第一个步骤通常都是计算相关数据的概括统计值。最常用的概括统计值可能是均值和标准差，这两个值能让你分别概括出数据集中的“经典”值，但是其他一些形式的聚合也是非常有用的（如求和、乘积、中位数、最小值和最大值、分位数，等等）。

下图提供了一个NumPy 中可用的聚合函数的清单。

函数名称	NaN安全版本	描述
np.sum	np.nansum	计算元素的和
np.prod	np.nanprod	计算元素的积
np.mean	np.nanmean	计算元素的平均值
np.std	np.nanstd	计算元素的标准差
np.var	np.nanvar	计算元素的方差
np.min	np.nanmin	找出最小值
np.max	np.nanmax	找出最大值
np.argmin	np.nanargmin	找出最小值的索引
np.argmax	np.nanargmax	找出最大值的索引
np.median	np.nanmedian	计算元素的中位数
np.percentile	np.nanpercentile	计算基于元素排序的统计值
np.any	N/A	验证任何一个元素是否为真
np.all	N/A	验证所有元素是否为真

大多数的聚合都有对 NaN 值的安全处理策略（NaN-safe），即计算时忽略所有的缺失值。

[439]:
```
big_array = np.random.rand(1000000)
big_array
```

[439]:
```
array([0.50814981, 0.16175075, 0.29048705, ..., 0.46435639, 0.01625442,
       0.59372462])
```

[440]:
```
min(big_array), max(big_array)
```

[440]:
```
(1.2420378219246686e-06, 0.9999996092814999)
```

在对较大的数据进行分析时，一项基本的工作就是有效的数据累计（summarization）：计算累计（aggregation）指标，如sum()、mean()、median()、min() 和max()，其中每一个指标都呈现了

大数据集的特征。

[441]:
```
# seaborn 行星数据中包含了截至2014 年已被发现的一千多颗外行星的资料。
import seaborn as sns
planets = sns.load_dataset('planets')
planets.shape
```

[441]:
```
(1035, 6)
```

[442]:
```
planets.head()
```

[442]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

与一维 NumPy 数组相同，Pandas 的 Series 的累计函数也会返回一个统计值：

[443]:
```
rng = np.random.RandomState(42)
ser = pd.Series(rng.rand(5))
ser
```

[443]:
```
0    0.374540
1    0.950714
2    0.731994
3    0.598658
4    0.156019
dtype: float64
```

[444]:
```
ser.sum()
```

[444]:
```
2.811925491708157
```

[445]:
```
ser.mean()
```

[445]:
```
0.5623850983416314
```

DataFrame 的累计函数默认对每列进行统计：

[446]:
```
df = pd.DataFrame({'A': rng.rand(5),'B': rng.rand(5)})
df
```

[446]:

	A	B
0	0.155995	0.020584
1	0.058084	0.969910
2	0.866176	0.832443
3	0.601115	0.212339
4	0.708073	0.181825

[447]:
```
df.mean()
```

[447]:
```
A    0.477888
B    0.443420
dtype: float64
```

设置axis 参数，可以对每一行进行统计

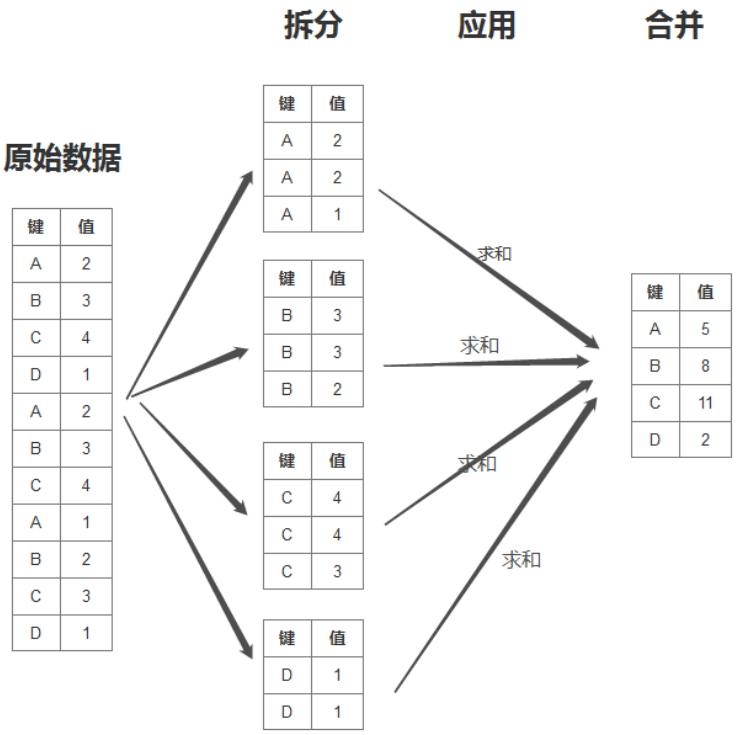
[448]:
```
df.mean(axis='columns')
```

[448]:
```
0    0.088290
1    0.513997
2    0.849309
3    0.406727
4    0.444949
dtype: float64
```

Pandas 内置的一些累计方法如图所示。

指标	描述
count()	计数项
first()、last()	第一项与最后一项
mean()、median()	均值与中位数
min()、max()	最小值与最大值
std()、var()	标准差与方差
mad()	均值绝对偏差（mean absolute deviation）
prod()	所有项乘积
sum()	所有项求和

简单的累计方法可以让我们对数据集有一个笼统的认识，但是我们经常还需要对某些标签或索引的局部进行累计分析，这时就需要用到 groupby 了。虽然“分组”（group by）这个名字是借用 SQL 数据库语言的命令，但其理念引用发明 R 语言 frame 的 Hadley Wickham 的观点可能更合适：分割（split）、应用（apply）和组合（combine）。



- 分割步骤将 DataFrame 按照指定的键分割成若干组。

- 应用步骤对每个组应用函数，通常是累计、转换或过滤函数。
- 组合步骤将每一组的结果合并成一个输出数组。

[451]:
```
data=pd.read_csv("hw4_data1.csv",encoding="gbk")
data.head()
```

[451]:

	CLASS_ID	STD_ID	SUBJECT	SCORE	LAST_SCORE
0	A1231	1	语文	97	94
1	A1231	1	数学	120	124
2	A1231	1	英语	107	109
3	A1231	1	生物	86	87
4	A1231	1	化学	92	88

假如对于上面的数据我们想要分科目查看平均分

[452]:
```
group_sub=data["SCORE"].groupby(data["SUBJECT"])
group_sub
```

[452]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000001BD911A8B88>
```

这里的结果只是一个groupby对象，也就是我们指示图中的拆分，想要得到运算结果需要传入想要的函数

[453]:
```
group_sub.mean()
```

[453]:
```
SUBJECT
化学     85.00
数学     98.50
物理     82.50
生物     82.50
英语    102.75
语文    100.00
Name: SCORE, dtype: float64
```

[454]:
```
group_sub.sum()
```

[454]:
```
SUBJECT
化学    340
数学    394
物理    330
生物    330
```

```
英语    411
语文    400
Name: SCORE, dtype: int64
```

如果既要分科目也要分班级的话

[455]:
```
data["SCORE"].groupby([data["CLASS_ID"],data["SUBJECT"]]).mean()
```

[455]:
```
CLASS_ID  SUBJECT
A1231     化学          80.0
          数学         109.0
          物理          77.0
          生物          82.5
          英语         114.0
          语文         101.5
A1232     化学          90.0
          数学          88.0
          物理          88.0
          生物          82.5
          英语          91.5
          语文          98.5
Name: SCORE, dtype: float64
```

GroupBy 对象的 aggregate()、filter()、transform() 和 apply() 方法，在数据组合之前实现了大量高效的操作。

[456]:
```
rng = np.random.RandomState(0)
df = pd.DataFrame({'key': ['A', 'B', 'C', 'A', 'B', 'C'],'data1':␣
 ↪range(6),'data2': rng.randint(0, 10, 6)},
                  columns = ['key', 'data1', 'data2'])
df
```

[456]:

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

累计。我们目前比较熟悉的 GroupBy 累计方法只有 sum() 和 median() 之类的简单函数，但是 aggregate() 其实可以支持更复杂的操作，比如字符串、函数或者函数列表，并且能一次性计算所有累计值。

[457]:
```
df.groupby('key').aggregate(['min', np.median, max])
```

[457]:

	data1			data2		
	min	median	max	min	median	max
key						
A	0	1.5	3	3	4.0	5
B	1	2.5	4	0	3.5	7
C	2	3.5	5	3	6.0	9

另一种用法就是通过Python 字典指定不同列需要累计的函数：

[458]:
```
df.groupby('key').aggregate({'data1': 'min','data2': 'max'})
```

[458]:

	data1	data2
key		
A	0	5
B	1	7
C	2	9

过滤。过滤操作可以让你按照分组的属性丢弃若干数据。例如，我们可能只需要保留标准差超过某个阈值的组：

[459]:
```
def filter_func(x):
    return x['data2'].std() > 4
df
```

[459]:

	key	data1	data2
0	A	0	5
1	B	1	0
2	C	2	3
3	A	3	3
4	B	4	7
5	C	5	9

[460]:
```
df.groupby('key').std()
```

[460]:
```
        data1     data2
key
A    2.12132  1.414214
B    2.12132  4.949747
C    2.12132  4.242641
```

[461]:
```
df.groupby('key').filter(filter_func)
```

[461]:
```
  key  data1  data2
1   B      1      0
2   C      2      3
4   B      4      7
5   C      5      9
```

filter() 函数会返回一个布尔值，表示每个组是否通过过滤。由于 A 组 'data2' 列的标准差不大于4，所以被丢弃了。

转换。累计操作返回的是对组内全量数据缩减过的结果，而转换操作会返回一个新的全量数据。数据经过转换之后，其形状与原来的输入数据是一样的。常见的例子就是将每一组的样本数据减去各组的均值，实现数据标准化：

[462]:
```
df.groupby('key').transform(lambda x: x - x.mean())
```

[462]:
```
   data1  data2
0   -1.5    1.0
1   -1.5   -3.5
2   -1.5   -3.0
3    1.5   -1.0
4    1.5    3.5
5    1.5    3.0
```

apply() 方法。apply() 方法让你可以在每个组上应用任意方法。这个函数输入一个 DataFrame，返回一个 Pandas 对象（DataFrame 或Series）或一个标量（scalar，单个数值）。组合操作会适应返回结果类型。

[463]:
```
# 将第一列数据以第二列的和为基数进行标准化
def norm_by_data2(x):
    # x是一个分组数据的DataFrame
    x['data1'] /= x['data2'].sum()
    return x
```

```
df.groupby('key').apply(norm_by_data2)
```

[463]:

	key	data1	data2
0	A	0.000000	5
1	B	0.142857	0
2	C	0.166667	3
3	A	0.375000	3
4	B	0.571429	7
5	C	0.416667	9

GroupBy 里的 apply() 方法非常灵活，唯一需要注意的地方是它总是输入分组数据的 DataFrame，返回 Pandas 对象或标量。具体如何选择需要视情况而定。

6.2 问题二 KNN Model

【2】利用iris.csv数据集，建立KNN模型，预测Sepal.Length/Sepal.Width/Petal.Length/Petal.Width分别为(6.3，3.1，4.8，1.4)时，属于鸢尾花的哪个类别？编写KNN源代码。

[464]:
```
data=pd.read_csv("HW4_DATA2.csv",index_col="index")
data
```

[464]:

index	Sepal.Length	Sepal.Width	Pepal.Length	Pepal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
49	5.3	3.7	1.5	0.2	setosa
50	5.0	3.3	1.4	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
53	6.9	3.1	4.9	1.5	versicolor
59	6.6	2.9	4.6	1.3	versicolor
118	7.7	3.8	6.7	2.2	virginica
119	7.7	2.6	6.9	2.3	virginica

[465]:
```
labels=data.pop("Species")
labels=np.array(labels)
```

```
labels
```

[465]:
```
array(['setosa', 'setosa', 'setosa', 'setosa', 'setosa', 'setosa',
       'setosa', 'versicolor', 'versicolor', 'versicolor', 'versicolor',
       'virginica', 'virginica'], dtype=object)
```

[466]:
```
#print(type(data.loc[1,:]))
data=np.array(data)
data
```

[466]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [4.6, 3.2, 1.4, 0.2],
       [5.3, 3.7, 1.5, 0.2],
       [5. , 3.3, 1.4, 0.2],
       [7. , 3.2, 4.7, 1.4],
       [6.4, 3.2, 4.5, 1.5],
       [6.9, 3.1, 4.9, 1.5],
       [6.6, 2.9, 4.6, 1.3],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3]])
```

[467]:
```
from numpy import *
new_t=np.array([6.3,3.1,4.8,1.4])

numSamples = data.shape[0]
diff=tile(new_t,(numSamples,1))-data #重复数组
squreDiff = diff**2
squreDist = sum(squreDiff, axis=1)
distance = squreDist ** 0.5
distance
```

[467]:
```
array([3.82099463, 3.86910842, 4.03236903, 3.90128184, 3.98748041,
       3.7       , 3.83796821, 0.71414284, 0.34641016, 0.6164414 ,
       0.42426407, 2.58843582, 2.72580263])
```

[468]:
```
sortedDistIndices = argsort(distance)
sortedDistIndices
```

[468]:
```
array([ 8, 10,  9,  7, 11, 12,  5,  0,  6,  1,  3,  4,  2], dtype=int64)
```

[469]:
```
classCount = {}
K = 4
for i in range(K):
    voteLabel = labels[sortedDistIndices[i]]
    print(voteLabel)
    classCount[voteLabel] = classCount.get(voteLabel, 0) + 1
print(classCount)

maxCount = 0
for k, v in classCount.items():
    if v > maxCount:
        maxCount = v
        maxIndex = k

print("Your input is:", new_t, "and classified to class: ", maxIndex)
```

```
versicolor
versicolor
versicolor
versicolor
{'versicolor': 4}
Your input is: [6.3 3.1 4.8 1.4] and classified to class:  versicolor
```

6.3 问题三常用距离

【3】计算X = [1,2,3]和Y = [0,1,2]的曼哈顿距离(Manhattan Distance)，切比雪夫距离，闵可夫斯基距离，标准化欧氏距离，马氏距离。给出计算公式，并根据公式计算。利用Python实现上述距离。

6.3.1 曼哈顿距离((Manhattan Distance))

$$d(x,y) = \sqrt{\sum_{k=1}^{n} |x_k - y_k|}$$

[470]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d import proj3d
import math
def Manhattan_Dist(X,Y):
    return math.sqrt(sum([abs(x-y) for (x,y) in zip(X,Y)]))
```

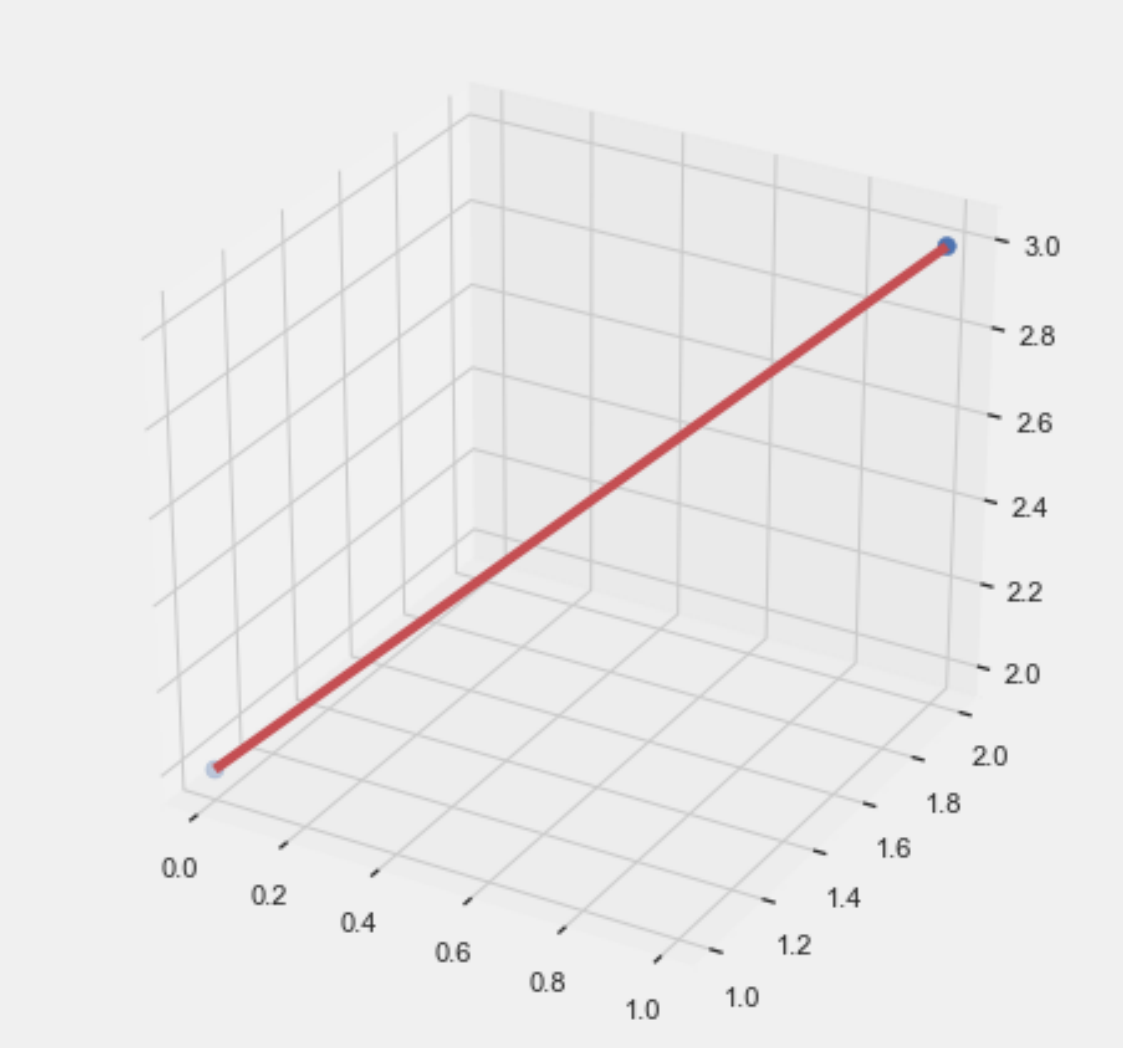
[471]:
```
X=[1,2,3]
Y=[0,1,2]
```

[472]:
```
Manhattan_Dist(X,Y)
```

[472]:
```
1.7320508075688772
```

[473]:
```
fig=plt.figure(figsize=(7,7))
ax=fig.add_subplot(111,projection='3d')
ax.scatter((X[0],Y[0]),(X[1],Y[1]),(X[2],Y[2]),color='b',s=50)
ax.plot((X[0],Y[0]),(X[1],Y[1]),(X[2],Y[2]),color='r')
```

[473]:
```
[<mpl_toolkits.mplot3d.art3d.Line3D at 0x1bd93b82f48>]
```



6.3.2 切比雪夫距离(Chebyshev Distance)

$$d(x,y) = \max_{k=1}^{n} |x_k - y_k|$$

[474]:
```
def Chebyshev_Dist(X,Y):
    return max([abs(x-y) for (x,y) in zip(X,Y)])
```

[475]:
```
Chebyshev_Dist(X,Y)
```

[475]: 1

6.3.3 闵可夫斯基距离(Minkowski Distance)

$$d(x,y) = \sqrt[p]{\sum_{k=1}^{n} |x_k - y_k|^p}$$

$p = 1$æ¨

$p = 2$æ′ffl

$p \to \infty$æ■+

[476]:
```
def Minkowski_Dist(X, Y, p):
    return (sum([abs(x-y)**p for (x,y) in zip(X,Y)]))**(1/p)
```

[477]:
```
Minkowski_Dist(X, Y, 5)
```

[477]:
```
1.2457309396155174
```

6.3.4 标准化欧式距离(Standardized Euclidean Distance)

$$d(x,y) = \sqrt{\sum_{k=1}^{n} \left(\frac{x_k - y_k}{s_k}\right)^2}$$

[478]:
```
def Std_Euclidean_Dist(X,Y):
    D=[]
    for index in range(0,len(X)):
        D.append([X[index],Y[index]])
    return math.sqrt(sum([((x-y)/np.var(np.array(d)))**2 for (x,y,d) in␣
 ↪zip(X,Y,D)]))
```

[479]:
```
Std_Euclidean_Dist(X,Y)
```

[479]:
```
6.928203230275509
```

6.3.5 马氏距离(Mahalanobis Distance)

$$d(x,y) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}$$

[480]:
```
def Mahalanobis_Dist(X,Y):
    V=np.vstack([X,Y])
    VT=V.T
```

```
    S=np.cov(V)    #协方差矩阵
    try:
        SI = np.linalg.inv(S)
        return math.sqrt((X-Y).T*SI*(X-Y))
    except:
        mark=False
        for index in range(0,len(S)):
            if S[index][index]==1:
                mark=True
        if mark:
            return math.sqrt(sum([(x-y)**2 for (x,y) in zip(X,Y)]))
        else:
            return Std_Euclidean_Dist(X,Y)
```

[481]:
```
Mahalanobis_Dist(X,Y)
```

[481]: 1.7320508075688772

[482]:
```
M=[3,4,5]
N=[10,8,7]
Mahalanobis_Dist(M,N)
```

[482]: 8.306623862918075

参考资料1 参考资料2 参考资料2

7 作业清单（5/13）

7.1 问题1 KMeans(I)

选择4名同学A、B、C、D，两次小测成绩，利用Kmeans算法分为“优秀”和“及格”两类。@注意：不能直接调用sklearn第三方库的KMeans函数，根据课堂讲授的分类过程，编写代码。撰写实验报告。

实验报告

1 实验过程：

- 建立数据集：

学生姓名	小测1	小测2
A	1	1

学生姓名	小测1	小测2
B	2	1
C	4	3
D	5	4

- 建立模型：由题可知 K 应取2，通过选择学生 A 和学生 B 作为初始聚类中心，通过计算每个点与聚类中心的距离，将每个点分到距离最近的簇中，更新聚类中心，不断迭代，直到簇中心不再发生改变，则完成聚类。

2 程序源代码

[129]:
```
import numpy as np
import math
```

[130]:
```
# 定义欧式距离
def Euclidean_Dist(X,Y):
    return math.sqrt(sum([(x-y)**2 for (x,y) in zip(X,Y)]))
```

[145]:
```
# 选择初始聚类中心
center=np.array([[1,1],[2,1]])
data=np.array([[1,1],[2,1],[4,3],[5,4]])
```

[146]:
```
# 记录到中心的距离和标签
labels=np.zeros((2,4))
```

[147]:
```
# 递归更新簇的中心点
def KMeans(c):
    for i in range(0,4):
        x=Euclidean_Dist(c[0],data[i])
        y=Euclidean_Dist(c[1],data[i])
        if x>y:
            labels[0,i]=0
            labels[1,i]=1
        else:
            labels[0,i]=1
            labels[1,i]=0
    num=np.array([0,0])
    sum=np.array([[0,0],[0,0]])
    for i in range(0,4):
```

```
        if labels[0,i]==1:
            num[0]=num[0]+1
            sum[0][0]=sum[0][0]+data[i][0]
            sum[0][1]=sum[0][1]+data[i][1]
        else:
            num[1]=num[1]+1
            sum[1][0]=sum[1][0]+data[i][0]
            sum[1][1]=sum[1][1]+data[i][1]
    new_c=np.zeros((2,2))
    for i in range(0,2):
        new_c[i]=sum[i]/num[i]
    diff=new_c-c
    if diff[0,0]==0 and diff[0,1]==0 and diff[1,0]==0 and diff[1,1]==0:
        center=new_c
        return center,labels
    else:
        center=new_c
        print(center)
        return KMeans(center)
```

[149]:
```
center,labels=KMeans(center)
```

```
[[1.         1.        ]
 [3.66666667 2.66666667]]
[[1.5 1. ]
 [4.5 3.5]]
```

3 程序运行结果及分析

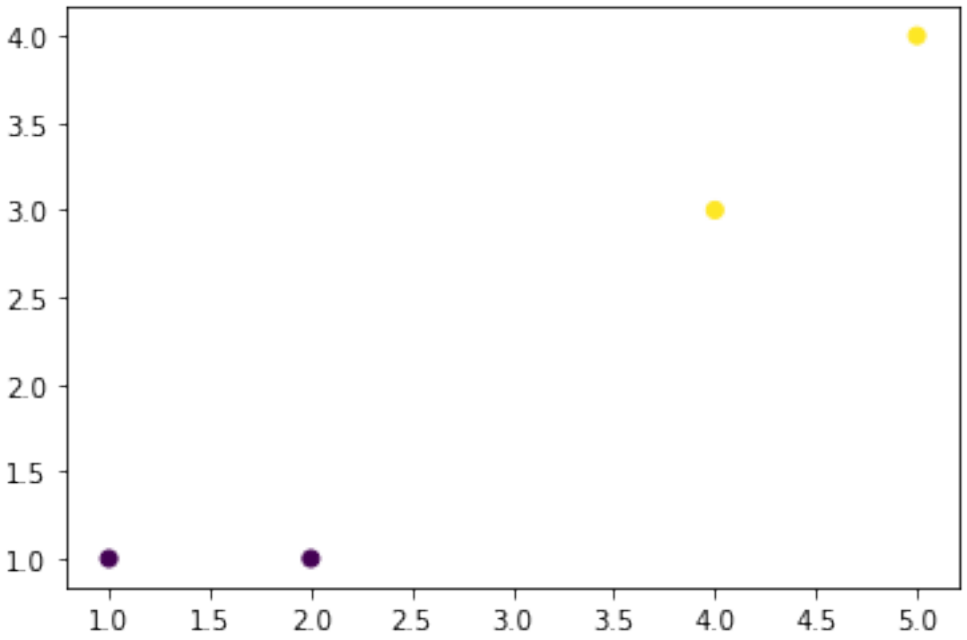
[150]:
```
center
```

[150]:
```
array([[1.5, 1. ],
       [4.5, 3.5]])
```

[151]:
```
labels
```

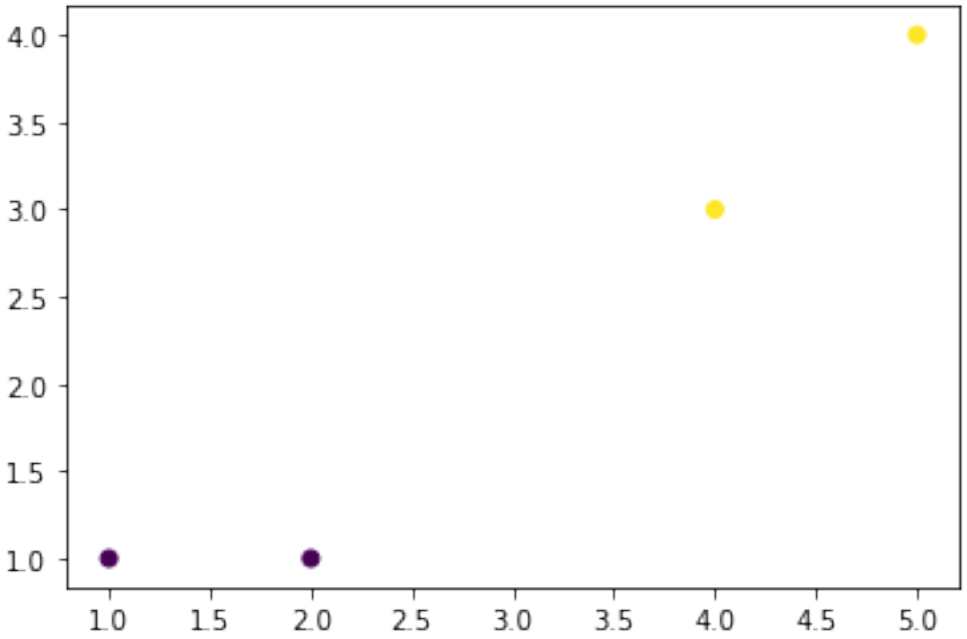
[151]:
```
array([[1., 1., 0., 0.],
       [0., 0., 1., 1.]])
```

[152]:
```
plt.scatter(data[:, 0], data[:, 1], marker='o',c=1-labels[0])
plt.show()
```



通过调用 sklearn 的函数比对，可知聚类结果一致

[115]:
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
KMeansCluster = KMeans(n_clusters=2)
y2 = KMeansCluster.fit_predict(data)
plt.scatter(data[:, 0], data[:, 1], marker='o',c=y2)
plt.show()
```



不像监督学习的分类问题和回归问题，我们的无监督聚类没有样本输出，也就没有比较直接的聚类评估方法。但是我们可以从簇内的稠密程度和簇间的离散程度来评估聚类的效果。常见的方法有轮廓系数Silhouette Coefficient和Calinski-Harabasz Index(CH)。这里使用CH法来评估。

$$CH(k) = \frac{trB(k)/(k-1)}{trW(k)/(n-k)}$$

其中，n表示聚类的数目 ,k表示当前的类，$trB(k)$表示类间离差矩阵的迹，$trW(k)$ 表示类内离差矩阵的迹。可以得出 CH 越大代表着类自身越紧密，类与类之间越分散，即更优的聚类结果。

[153]:
```
labels=1-labels[0]
```

[154]:
```
from sklearn import metrics
# 求解CH值
score = metrics.calinski_harabasz_score(data, labels)
score
```

[154]: 20.333333333333332

4 知识点总结

- KMeans 中的 K 指的是聚类之后簇的个数
- 欧氏距离

7.2 问题二 KMeans(II)

根据下列成绩单，将5名同学成绩归为A类、B类、C类，利用Kmeans算法实现。@注意：不能直接调用sklearn第三方库的KMeans函数，根据课堂讲授的分类过程，编写代码。撰写实验报告。

实验报告

1 实验过程：

- 建立数据集：

学生姓名	小测1	小测2	小测3	期末成绩	项目答辩	成绩
张三	12	15	13	28	24	?
李四	7	11	10	19	21	?
王五	12	14	11	27	23	?
赵六	6	7	4	13	20	?
刘七	13	14	13	27	25	?

建立模型：由题可知 K 应取3，

- 选择学生张三、李四和王五的成绩作为初始聚类中心
- 选择学生张三、李四和赵六的成绩作为初始聚类中心通过计算每个点与聚类中心的距离，将每个点分到距离最近的簇中，更新聚类中心，不断迭代，直到簇中心不再发生改变，则完成聚类。

[306]:
```
# 选择第一种初始聚类中心
center=np.array([[12,15,13,28,24],[7,11,10,19,21],[12,14,11,27,23]])
data=np.
 ↪array([[12,15,13,28,24],[7,11,10,19,21],[12,14,11,27,23],[6,7,4,13,20],[13,14,13,27,25]])
```

[307]:
```
# 记录到中心的距离和标签
labels=np.zeros((3,5))
```

[308]:
```
# 递归更新簇的中心点
def KMeans(c):
    for i in range(0,5):
        x=Euclidean_Dist(c[0],data[i])
        y=Euclidean_Dist(c[1],data[i])
        z=Euclidean_Dist(c[2],data[i])
        if x<y and x<z:
            labels[0,i]=1
```

```
            labels[1,i]=0
            labels[2,i]=0
        elif y<x and y<z:
            labels[0,i]=0
            labels[1,i]=1
            labels[2,i]=0
        elif z<x and z<y:
            labels[0,i]=0
            labels[1,i]=0
            labels[2,i]=1
    num=np.array([0,0,0])
    sum=np.zeros((3,5))
    for i in range(0,5):
        if labels[0,i]==1:
            num[0]=num[0]+1
            for j in range(0,5):
                sum[0][j]=sum[0][j]+data[i][j]
        elif labels[1,i]==1:
            num[1]=num[1]+1
            for j in range(0,5):
                sum[1][j]=sum[1][j]+data[i][j]
        else:
            num[2]=num[2]+1
            for j in range(0,5):
                sum[2][j]=sum[2][j]+data[i][j]
    new_c=np.zeros((3,5))
    for i in range(0,3):
        new_c[i]=sum[i]/num[i]
    diff=new_c-c
    center=new_c
    if diff[0,0]==0 and diff[0,1]==0 and diff[1,0]==0 and diff[1,1]==0 and ␣
↪diff[2,0]==0 and diff[2,1]==0 :
            return center,labels
    else:
        return KMeans(center)
```

[309]:
```
center,labels=KMeans(center)
```

[310]:
```
center
```

[310]:
```
array([[12.5, 14.5, 13. , 27.5, 24.5],
       [ 6.5,  9. ,  7. , 16. , 20.5],
       [12. , 14. , 11. , 27. , 23. ]])
```

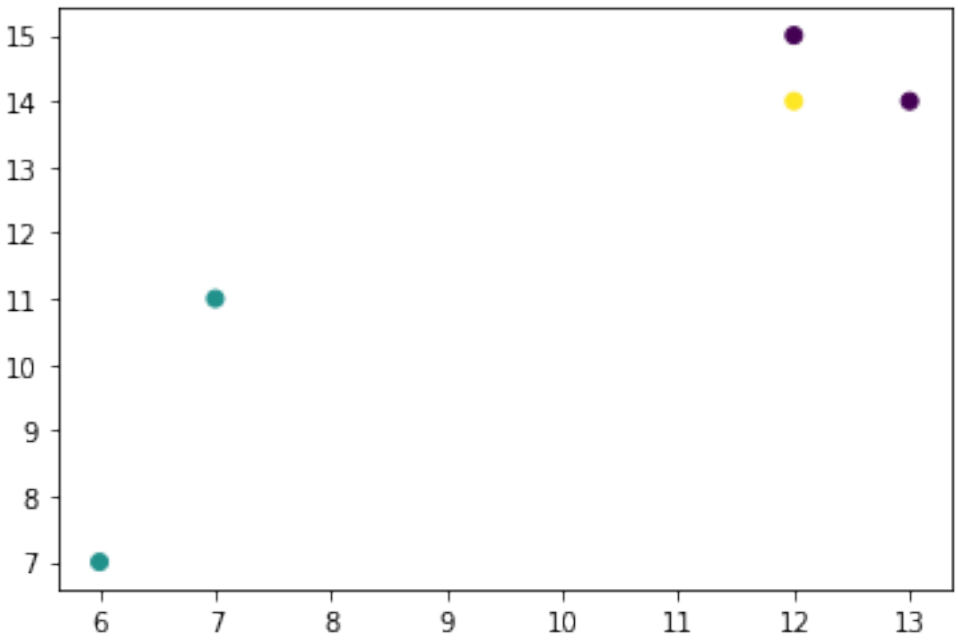
[311]:
```
labels
```

[311]:
```
array([[1., 0., 0., 0., 1.],
       [0., 1., 0., 1., 0.],
       [0., 0., 1., 0., 0.]])
```

[312]:
```
label=[]
for i in range(0,5):
    label.append(labels[0,i]+labels[1,i]*2+labels[2,i]*3)
label
```

[312]:
```
[1.0, 2.0, 3.0, 2.0, 1.0]
```

[313]:
```
plt.scatter(data[:, 0], data[:, 1], marker='o',c=label)
plt.show()
```



[315]:
```
from sklearn import metrics
# 求解CH值
score = metrics.calinski_harabasz_score(data, label)
score
```

[315]:
```
6.012765957446809
```

[316]:
```
# 选择第二种初始聚类中心
center=np.array([[12,15,13,28,24],[7,11,10,19,21],[6,7,4,13,20]])
data=np.
 ↪array([[12,15,13,28,24],[7,11,10,19,21],[12,14,11,27,23],[6,7,4,13,20],[13,14,13,27,25]])
```

[317]:
```
center,labels=KMeans(center)
```

[318]:
```
center
```

[318]:
```
array([[12.33333333, 14.33333333, 12.33333333, 27.33333333, 24.        ],
       [ 7.        , 11.        , 10.        , 19.        , 21.        ],
       [ 6.        ,  7.        ,  4.        , 13.        , 20.        ]])
```

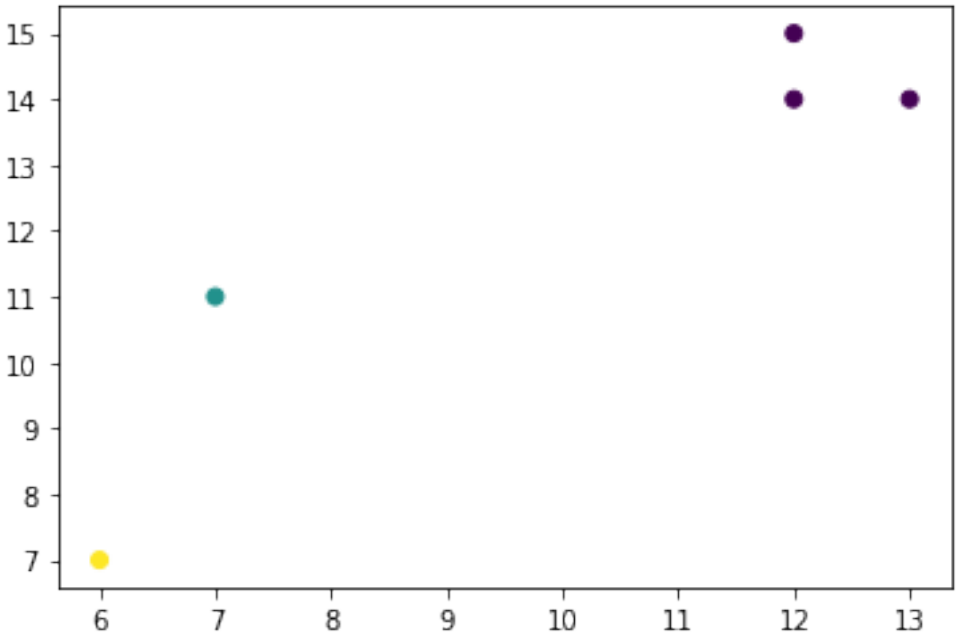
[319]:
```
labels
```

[319]:
```
array([[1., 0., 1., 0., 1.],
       [0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0.]])
```

[320]:
```
label=[]
for i in range(0,5):
    label.append(labels[0,i]+labels[1,i]*2+labels[2,i]*3)
label
```

[320]:
```
[1.0, 2.0, 1.0, 3.0, 1.0]
```

[321]:
```
plt.scatter(data[:, 0], data[:, 1], marker='o',c=label)
plt.show()
```



[322]:
```
from sklearn import metrics
# 求解CH值
score = metrics.calinski_harabasz_score(data, label)
score
```

[322]: 48.440000000000005

3 程序运行结果及分析

根据上面的实验可以发现，选取不同的初始聚类中心，聚类的结果可能不同，而显然通过散点图和 CH 值均可以看出第二种初始中心点的聚类结果较好

4 知识点总结

KMeans 初始聚类中心的选择将会影响聚类结果

7.3 问题三 KNN 和 KMeans

利用Sklearn的标准KNN和KMeans方法，数据集为“wine.csv”（见微信群），通过KNN算法，对葡萄酒的测试集进行标注，然后对比预测标签值和已知标签值，得到KNN算法的预测准确率。通过Kmeans算法，对无标签的“wine.csv”进行分类，自己设定K值和初始中心点值。

[390]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

[404]:
```
wine = pd.read_csv("wine.csv")
wine.head()
```

[404]:

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	\
0	14.23	1.71	2.43	15.6	127	2.80	
1	13.20	1.78	2.14	11.2	100	2.65	
2	13.16	2.36	2.67	18.6	101	2.80	
3	14.37	1.95	2.50	16.8	113	3.85	
4	13.24	2.59	2.87	21.0	118	2.80	

	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	\
0	3.06	0.28	2.29	5.64	1.04	
1	2.76	0.26	1.28	4.38	1.05	
2	3.24	0.30	2.81	5.68	1.03	
3	3.49	0.24	2.18	7.80	0.86	
4	2.69	0.39	1.82	4.32	1.04	

	OD280/OD315 of diluted wines	Proline	Class
0	3.92	1065	one
1	3.40	1050	one
2	3.17	1185	one
3	3.45	1480	one
4	2.93	735	one

[405]:
```
exam_Y=wine.pop("Class")
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test = train_test_split(wine,exam_Y,train_size=.7)
```

[406]:
```
transfer = StandardScaler()
X_train = transfer.fit_transform(X_train)
X_test = transfer.fit_transform(X_test)
```

[410]:
```
estimator = KNeighborsClassifier()
estimator.fit(X_train,Y_train)
```

[410]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[411]:
```
predict=estimator.predict(X_test)
```

[412]:
```
estimator.score(X_test,Y_test)
```

[412]:
```
0.9444444444444444
```

[413]:
```
KMeansCluster = KMeans(n_clusters=3)
y = KMeansCluster.fit(wine)
```

[415]:
```
label_pred = KMeansCluster.labels_
label_pred
```

[415]:
```
array([1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 0, 2, 0, 2, 2, 0,
       2, 2, 0, 0, 0, 2, 2, 1, 0, 2, 2, 2, 0, 2, 2, 0, 0, 2, 2, 2, 2, 2,
       0, 0, 2, 2, 2, 2, 2, 0, 0, 2, 0, 2, 0, 2, 2, 2, 0, 2, 2, 2, 2, 0,
       2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2,
       2, 0, 0, 0, 0, 2, 2, 2, 0, 0, 2, 2, 0, 0, 2, 0, 0, 2, 2, 2, 2, 0,
       0, 0, 2, 0, 0, 0, 2, 0, 2, 0, 0, 2, 0, 0, 0, 0, 2, 2, 0, 0, 0, 0,
       0, 2])
```

[416]:
```
from sklearn import metrics
score = metrics.calinski_harabasz_score(wine, label_pred)
score
```

[416]:
```
561.815657860671
```

7.4 问题四 KMeans(III)

【4】利用KMeans算法对“iris.csv”数据集的无标签数据分为3类，用三维图形可视化分类结果。

[417]:
```
from sklearn .datasets import load_iris
```

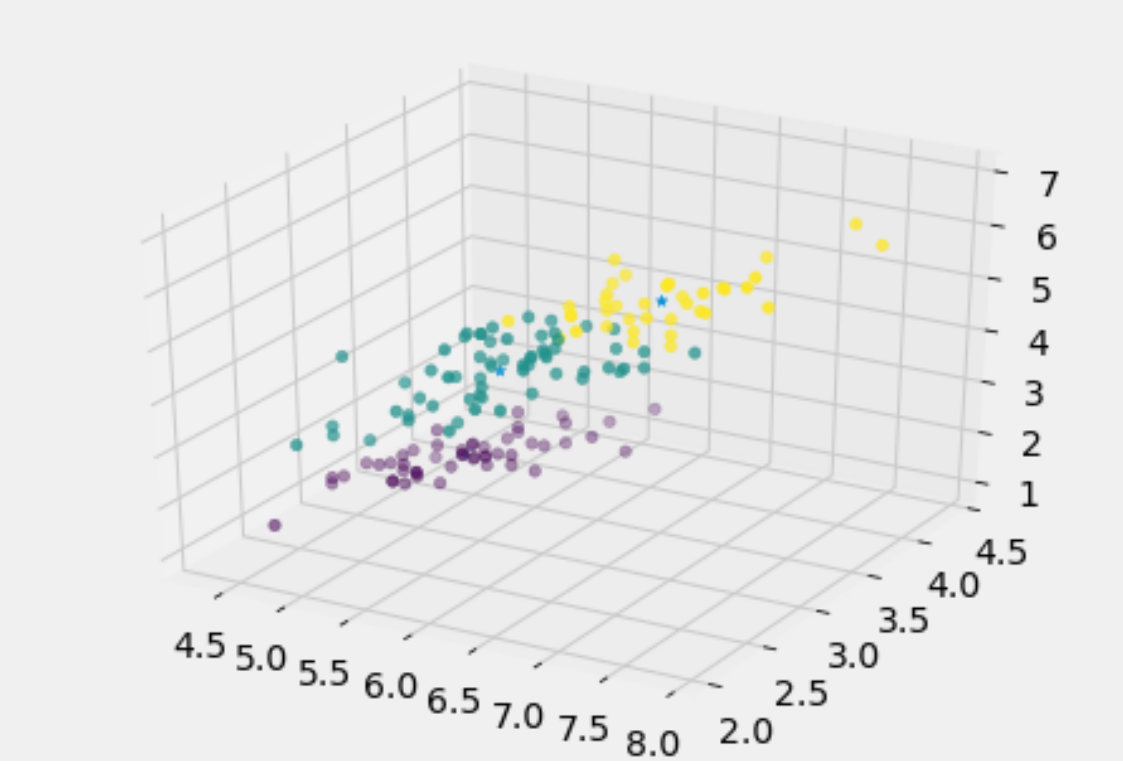
[434]:
```
iris = load_iris()
X=iris.data[:]
```

[451]:
```
KMeansCluster = KMeans(n_clusters=3)
y = KMeansCluster.fit_predict(X)
label_pred = KMeansCluster.labels_
centroids = KMeansCluster.cluster_centers_
centroids
```

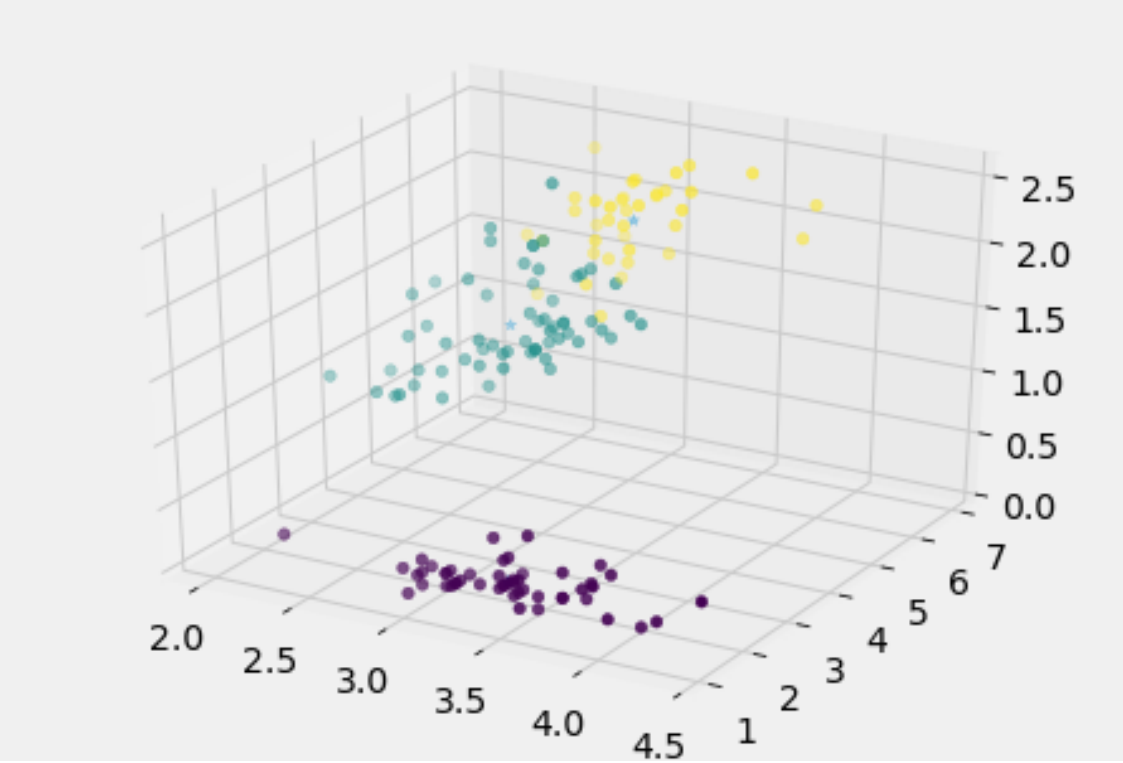
[451]:
```
array([[5.006     , 3.428     , 1.462     , 0.246     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

[452]:
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
```

[455]:
```
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=y)
ax.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], marker='*')
plt.show()
```



[456]:
```
fig = plt.figure()
ax = Axes3D(fig)
ax.scatter(X[:, 1], X[:, 2], X[:, 3], c=y)
ax.scatter(centroids[:, 1], centroids[:, 2], centroids[:, 3], marker='*')
plt.show()
```



7.5 问题五 KMeans(IV)

【5】利用KMeans算法对“iris.csv”数据集的无标签数据分为3类，任取2个特征值，显示分类结果，用二维图形可视化分类结果。

[1]:
```
from sklearn .datasets import load_iris
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

[2]:
```
iris = load_iris()
X=iris.data[:]
KMeansCluster = KMeans(n_clusters=3)
y = KMeansCluster.fit_predict(X)
label_pred = KMeansCluster.labels_
centroids = KMeansCluster.cluster_centers_
centroids
```

[2]:
```
array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
       [5.006     , 3.428     , 1.462     , 0.246     ],
       [6.85      , 3.07368421, 5.74210526, 2.07105263]])
```

[30]:
```
import numpy as np
import pandas as pd
new_X=np.concatenate([X,KMeansCluster.labels_.reshape(-1,1)],axis=1)
new_iris=pd.
 ↪DataFrame(new_X,columns=['sepal_length','sepal_width','petal_length','petal_width','species
new_iris.head()
```

[30]:

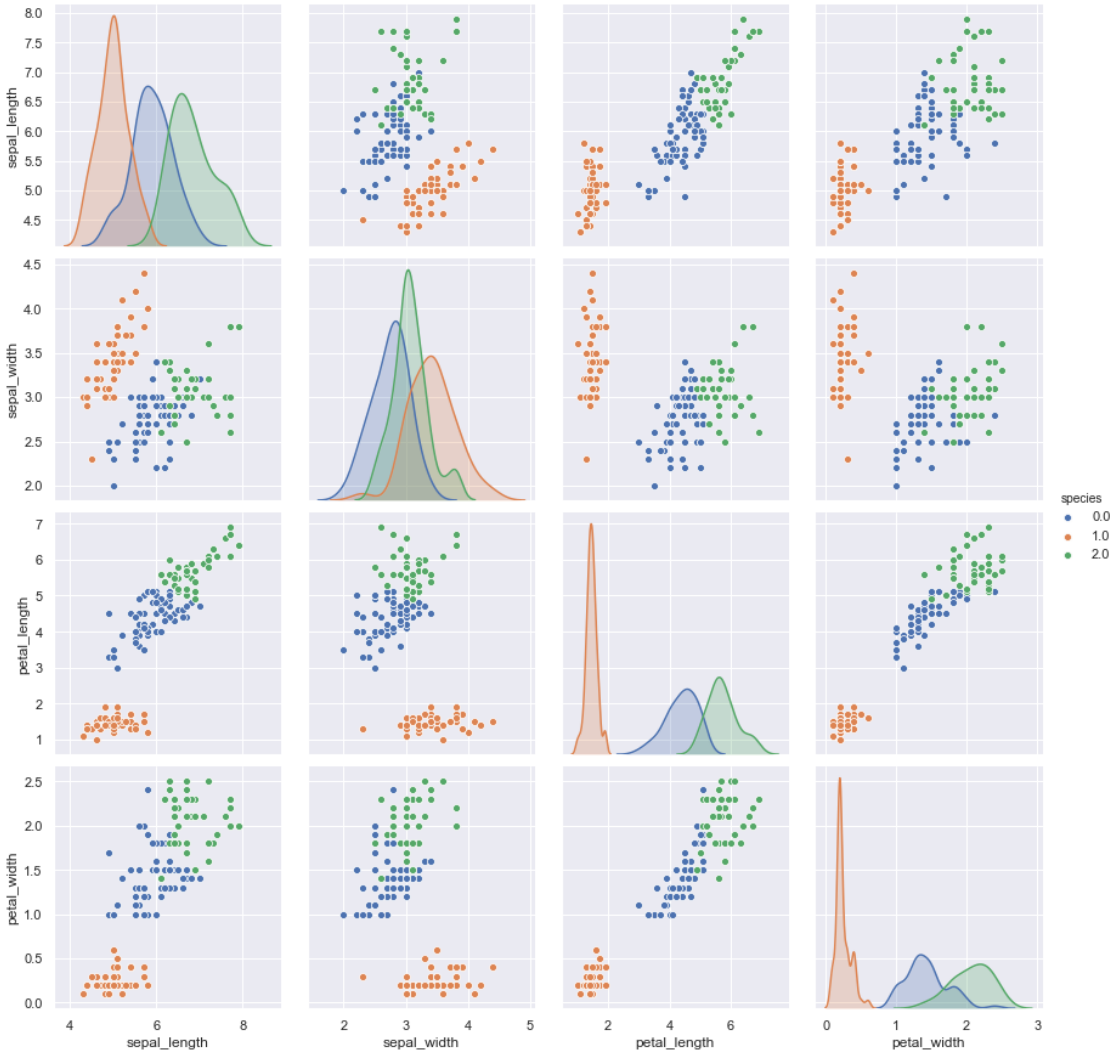
	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	1.0
1	4.9	3.0	1.4	0.2	1.0
2	4.7	3.2	1.3	0.2	1.0
3	4.6	3.1	1.5	0.2	1.0
4	5.0	3.6	1.4	0.2	1.0

[57]:
```
sns.pairplot(new_iris, hue="species", height=3,diag_kind="hist")
```

[57]:
```
<seaborn.axisgrid.PairGrid at 0x1d68ff5dc08>
```

[58]:
```
sns.pairplot(new_iris, hue="species", height=3,diag_kind="kde")
```

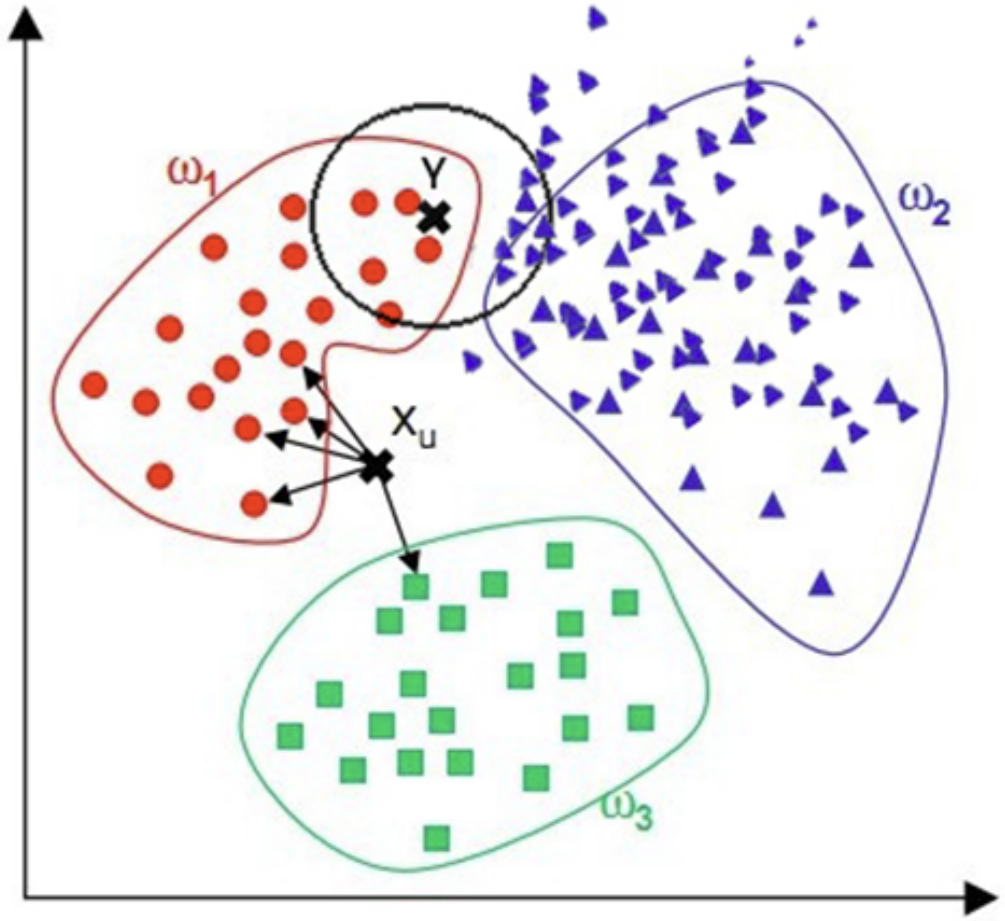
[58]: <seaborn.axisgrid.PairGrid at 0x1d69116ef48>



7.6 问题六 KNN 和 KMeans 的缺陷

【6】你认为KMeans算法和KNN算法的缺陷是什么？针对这些缺点，通过查阅资料，了解到有什么改进的方法？

KNN算法在分类时有个主要的不足是，当样本不平衡时，如一个类的样本容量很大，而其他类样本容量很小时，有可能导致当输入一个新样本时，该样本的K个邻居中大容量类的样本占多数，如下图所示。



该算法只计算最近的邻居样本，某一类的样本数量很大，那么或者这类样本并不接近目标样本，或者这类样本很靠近目标样本。无论怎样，数量并不能影响运行结果。可以采用权值的方法(和该样本距离小的邻居权值大)来改进。

该方法的另一个不足之处是计算量较大，因为对每一个待分类的文本都要计算它到全体已知样本的距离，才能求得它的K个最近邻点。

可理解性差，无法给出像决策树那样的规则。

K-means算法的缺点首先是在 K-means 算法中 K 是事先给定的，这个 K 值的选定是非常难以估计的。很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适；其次，在 K-means 算法中，首先需要根据初始聚类中心来确定一个初始划分，然后对初始划分进行优化。这个初始聚类中心的选择对聚类结果有较大的影响，一旦初始值选择的不好，可能无法得到有效的聚类结果(如在问题二中选择不同的同学作为初始聚类中心，会有不同的聚类结果)；最后，该算法需要不断地进行样本分类调整，不断地计算调整后的新的聚类中心，因此当数据量非常大时，算法的时间开销是非常大的。

- K-means算法对于不同的初始值，可能会导致不同结果。解决方法：
  - 多设置一些不同的初值，对比最后的运算结果，一直到结果趋于稳定结束
  - 很多时候，事先并不知道给定的数据集应该分成多少个类别才最合适。通过类的自动合并和分裂，得到较为合理的类型数目 K，例如 ISODATA 算法。
- K-means算法的其他改进算法如下：

鉴于K-means算法和人工蜂群算法各自特性，提出一种基于改进人工蜂群的K-means聚类算法IABC-Kmeans。该算法首先对人工蜂群算法进行改进：利用提出的最大最小距离积法初始化蜂群，保证初始点的选择能够尽可能代表数据集的分布特征；在迭代过程中使用新的适应度函数和位置更新公式完成寻优进化。然后将改进后的人工蜂群算法应用到K-means算法中完成聚类。 论文及github地址