1. 数学基础

①线性代数　②概率统计　③优化理论　④图论

2. 代码部分

①NumPy　②Pandas　③SciPy　④Matplotlib

二、应用部分

1.数据可视化

载入数据

```
import pandas as pd
import os

file_path = os.path.join('input', 'Ch3-HousePrice-train.csv')
df_train = pd.read_csv(file_path)

print("\n载入数据集：")
print(df_train)
print("\n查看数据列名：")
print(df_train.columns)
```

输出

```
载入数据集：
        Id  MSSubClass MSZoning  ...  SaleType  SaleCondition SalePrice
0        1          60       RL  ...        WD         Normal    208500
1        2          20       RL  ...        WD         Normal    181500
2        3          60       RL  ...        WD         Normal    223500
3        4          70       RL  ...        WD        Abnorml    140000
4        5          60       RL  ...        WD         Normal    250000
...    ...         ...      ...  ...       ...            ...       ...
1455  1456          60       RL  ...        WD         Normal    175000
1456  1457          20       RL  ...        WD         Normal    210000
1457  1458          70       RL  ...        WD         Normal    266500
1458  1459          20       RL  ...        WD         Normal    142125
1459  1460          20       RL  ...        WD         Normal    147500

[1460 rows x 81 columns]
```

```
查看数据列名：
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

为什么使用 os.path.join:

正斜杠/是在类 Unix 系统（如 Linux 和 macOS）上使用的路径分隔符。反斜杠\是在

Windows 系统上使用的路径分隔符。对于跨平台的 Python 代码，建议使用 os 模块提供的函数来处理文件路径，这样可以保证在不同的操作系统上都能正常工作。例如，使用 os.path.join()函数可以正确地创建跨平台的文件路径，它会根据当前操作系统自动选择正确的路径分隔符。

描述数据

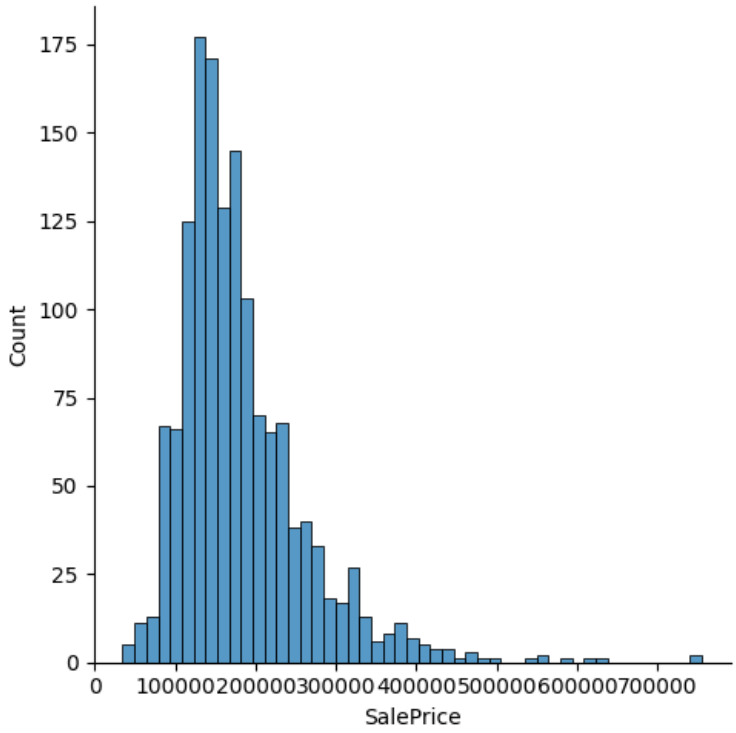
```
print(df_train['SalePrice'].describe())
```

```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

绘制图像

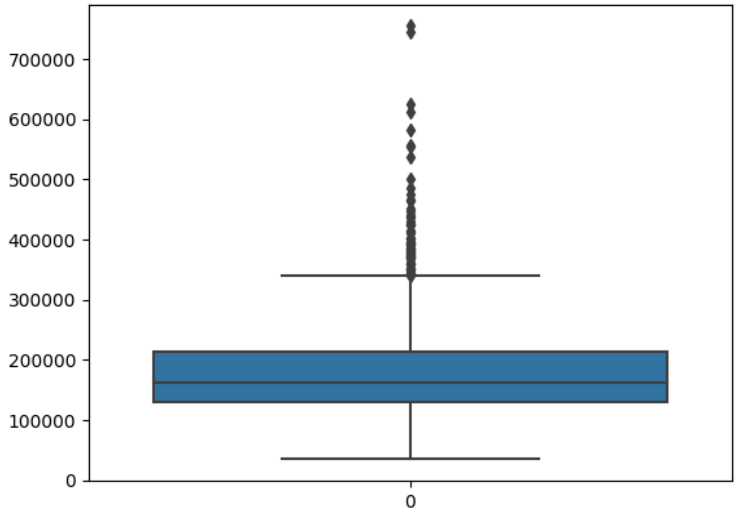
直方图

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.displot(df_train['SalePrice'])  # 直方图
plt.show()
```



箱型图

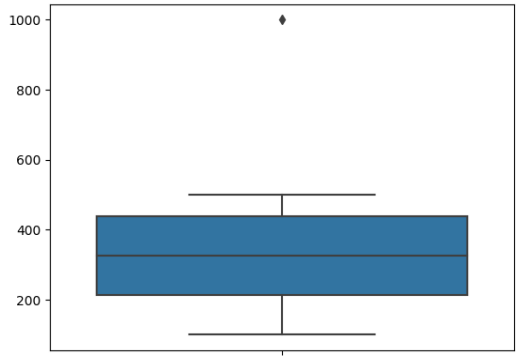
```
sns.boxplot(df_train['SalePrice'])  # 箱型图
plt.show()
```



为计算方便，这里使用一个数目较少的数据集来说明一下箱型图：

```
prices = [100, 150, 200, 250, 300, 350, 400, 450, 500, 1000]
df = pd.DataFrame(prices, columns=['Price'])
summary_stats = df.describe().transpose()
print(summary_stats)
sns.boxplot(y=prices)
plt.show()
```

```
       count   mean         std    min     25%    50%    75%     max
Price   10.0  370.0  256.255081  100.0  212.5  325.0  437.5  1000.0
```



由输出数据可知：总数=10，均值=370，标准差=256.255，Q1=200+(250-200)*25%=212.5，中位数 Q2=(300+350)/2=325.0，Q3=400+(450-400)*0.75%=437.5

计算得：

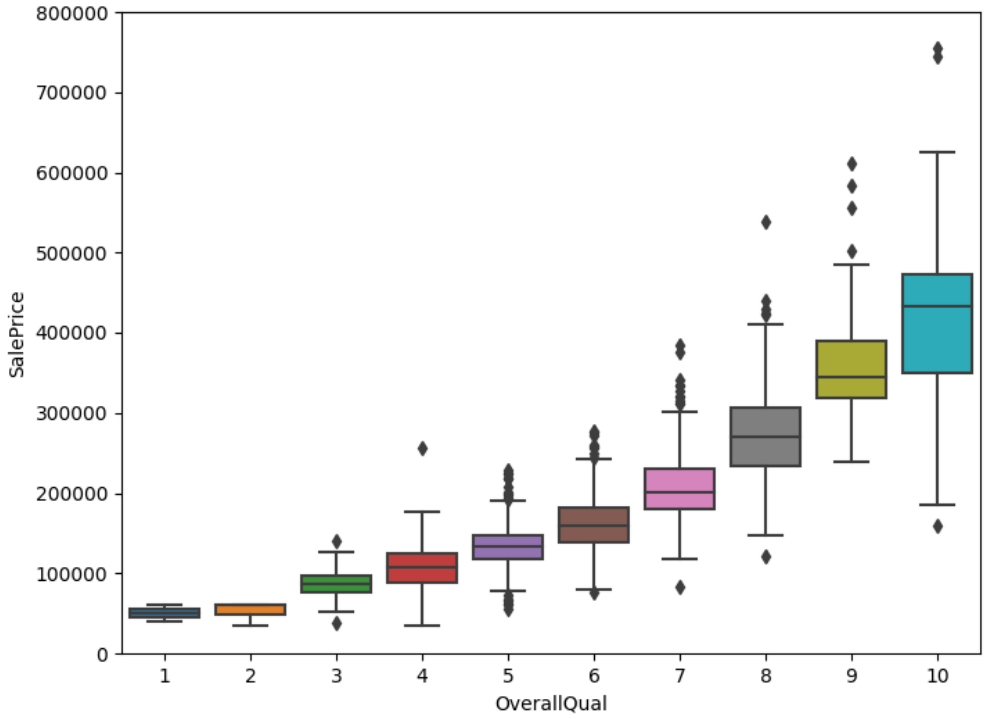
IQR=Q3-Q1=225。

箱型图中对应的五条线依次是：

下限=MIN=100，Q1=212.5，Q2=325.0，Q3=437.5，上限=去掉异常值 1000 后的最大值=500。

一组箱型图

```
data = pd.concat([df_train['SalePrice'], df_train['OverallQual']], axis=1)
f, ax = plt.subplots(figsize=(8, 6))  # 创建画像 figure 与一组子图 subplots
fig = sns.boxplot(x='OverallQual', y='SalePrice', data=data)
fig.axis(ymin=0, ymax=800000)
plt.show()
```



语法：

① **new_dataframe = pd.concat([dataframe1, dataframe2], axis=0/1)**。dataframe1 和 dataframe2：要合并的数据集，可以是 pandas DataFrame 或 Series 的列表。axis：指定合并的方向。axis=0 表示按行合并(将两个 DataFrame 垂直合并，行索引相同的列会对齐)，axis=1 表示按列合并(将两个 DataFrame 水平合并，列索引相同的列会对齐)。

```
import pandas as pd
df1 = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df2 = pd.DataFrame({'B': [7, 8, 9], 'C': [10, 11, 12]})
merged_rows = pd.concat([df1, df2], axis=0)
merged_columns = pd.concat([df1, df2], axis=1)
print(df1)
print(df2)
print(merged_rows)
print(merged_columns)
```

如下图，分别为 df1,df2,按行处理,按列处理的结果。注意没有对应的列的值被填充为缺失值 NaN。

	A	B
0	1	4
1	2	5
2	3	6

	B	C
0	7	10
1	8	11
2	9	12

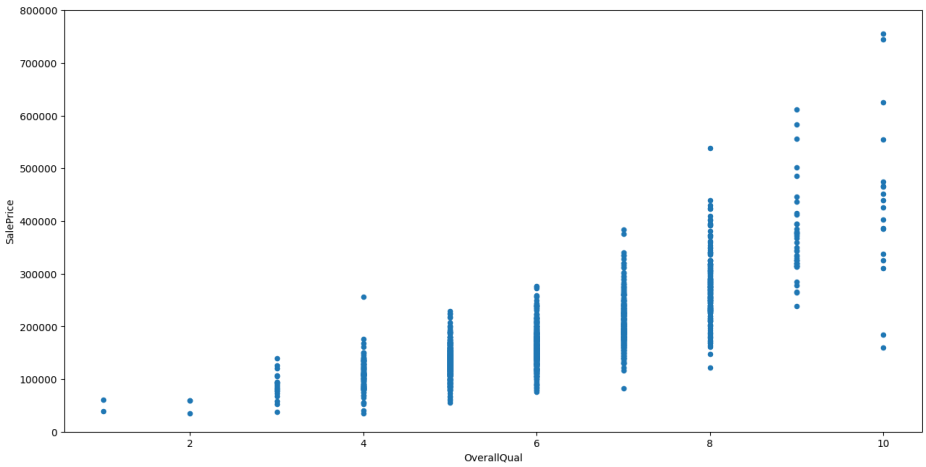
	A	B	C
0	1.0	4	NaN
1	2.0	5	NaN
2	3.0	6	NaN
0	NaN	7	10.0
1	NaN	8	11.0
2	NaN	9	12.0

	A	B	B	C
0	1	4	7	10
1	2	5	8	11
2	3	6	9	12

② (8, 6)用于设置画布（figure）的大小，宽度为 8 英寸，高度为 6 英寸。

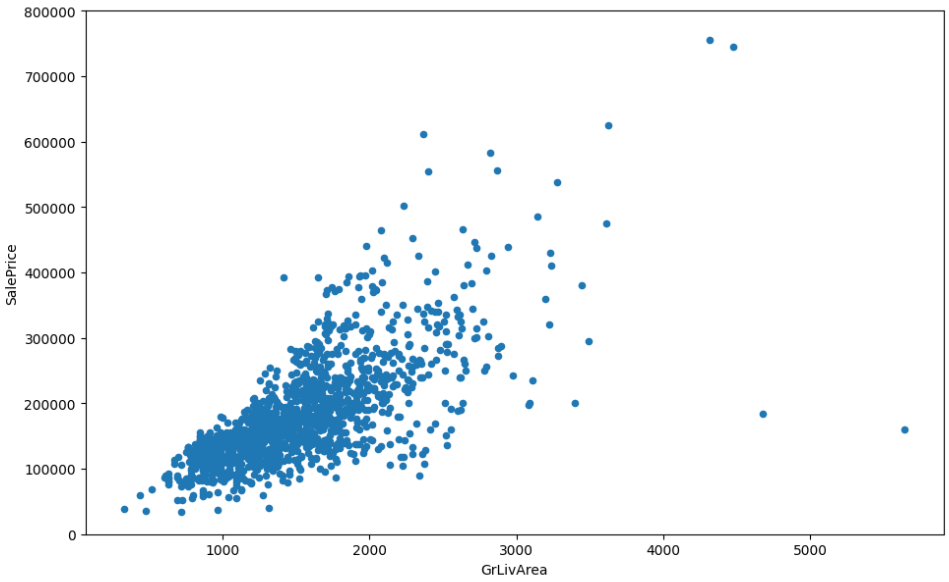
散点图

```
data = pd.concat([df_train['SalePrice'], df_train['OverallQual']], axis=1)
data.plot.scatter(x='OverallQual', y='SalePrice', ylim=(0, 800000))  # 散点图
plt.show()
```



该散点图显示的不够好看，因为参数 x 的值过少。我们可以把参数 OverallQual(总体质量)换成 GrLivArea(平方英尺)，此时生成的散点图就与平时常见的类似了。

```
data = pd.concat([df_train['SalePrice'], df_train['GrLivArea']], axis=1)
data.plot.scatter(x='GrLivArea', y='SalePrice', ylim=(0, 800000))
plt.show()
```



2.数据清洗

缺失值处理

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

df = pd.DataFrame(np.random.randn(6, 4), columns=list('abcd'))  # 随机生成数据
df.iloc[4, 3] = np.nan  # 选择第 5 行、第 4 列
df.loc[3] = np.nan  # 选择第 4 行
print(df)

total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum() / df.isnull().count().sort_values(ascending=False))
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
print(missing_data)  # 查看缺失值的数量

df_cleaned = df.dropna(how='all')  # 删除全为缺失值的行
print(df_cleaned)

imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit(df)
df_cleaned_1 = imp.transform(df_cleaned)
print(df_cleaned_1)
```

输出依次为 df,missing_data,df_cleaned,df_cleaned_1

```
          a         b         c         d
0  1.254260  0.232693  1.631425 -0.053841
1 -0.462248  0.360201  0.386029  0.312850
2  1.500807  0.796929  1.392375 -1.914533
3       NaN       NaN       NaN       NaN
4  0.642735 -1.030269 -0.397600       NaN
5  0.962220  0.147848 -0.625509  0.710948
```

	Total	Percent
d	2	0.333333
a	1	0.166667
b	1	0.166667
c	1	0.166667

```
          a         b         c         d
0  1.254260  0.232693  1.631425 -0.053841
1 -0.462248  0.360201  0.386029  0.312850
2  1.500807  0.796929  1.392375 -1.914533
4  0.642735 -1.030269 -0.397600       NaN
5  0.962220  0.147848 -0.625509  0.710948
```

```
[[ 1.25425977  0.23269252  1.63142545 -0.05384106]
 [-0.46224823  0.36020057  0.38602899  0.31284953]
 [ 1.50080682  0.79692885  1.39237459 -1.91453265]
 [ 0.64273546 -1.03026891 -0.39760032 -0.23614407]
 [ 0.96221962  0.14784837 -0.62550899  0.71094789]]
```

注：

(1)df.iloc 和 df.loc 是 Pandas 中用于访问和选择 DataFrame 数据的两种方法，区别在于索引方式不同：①df.iloc 使用**整数**位置索引，通过指定行号和列号来选择数据。例如，df.iloc[0, 0]代表选择第一行第一列的元素。②df.loc 使用**标签**索引，通过指定行标签和列标签来选择数据。比如：df.loc[0,'column_name']代表选择行标签为 0、列标签为'column_name'的元素。

(2)df.isnull().sum()返回一个 Series 对象，其中每列的标签作为索引，对应的值是该列中缺失值的数量。sort_values(ascending=False)用于对 Series 进行降序排列。参数 ascending=False 表示按递减顺序(即：从大到小)排列。

(3)df.dropna(how='all')中的 how 参数表示删除的条件。how='all'表示只删除全为缺失值的行，如果一行中所有元素都是缺失值，那么该行才将被删除。默认值 how='any'表示删除含有任意缺失值的行，只要有一个元素是缺失值，该行就会被删除。

(4)imp = SimpleImputer(missing_values=np.nan, strategy='mean')用于处理缺失值。SimpleImputer 是 scikit-learn 库中的一个类，用于使用指定的策略填充缺失值。missing_values=np.nan 表示缺失值的标识，默认为 np.nan。strategy='mean'表示使用均值作为策略来填充缺失值。其他可选的策略包括 'median'（中位数）和 'most_frequent'（众数）。

(5)imp.fit(df)是用于拟合填充器对象 imp 的方法。通过调用 fit 方法，填充器会根据指定的策略和数据集 df 来学习缺失值的填充规则。在这个过程中，df 并没有被修改，而是用于计算填充所需的统计信息。fit 方法没有返回值，但是填充器对象 imp 本身被修改以

存储学习到的规则。

(6)df_cleaned_1 = imp.transform(df_cleaned)是用于将填充器应用于数据集的语法，transform 方法会根据学习到的规则填充数据集中的缺失值。在这里，参数 df_cleaned 是传递给 transform 方法的数据集，而不是原始的 df，因为 df_cleaned 中不包含全为缺失值的行。transform 方法返回填充后的数据集，并存储在 df_cleaned_1 中。

举一个简单的例子来说明 SimpleImputer:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer

data = {'A': [7, 1, 5],
        'B': [7, 2, 7],
        'C': [1, 2, 3],
        'D': [6, 4, 8]}
df = pd.DataFrame(data)  # 使用字典创建数据表
df.iloc[1, 2] = np.nan  # 选择行、列
print("df(with NaN):")
print(df)

total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum() / df.isnull().count().sort_values(ascending=False))
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
print("missing_data:")
print(missing_data)  # 查看缺失值的数量

imp = SimpleImputer(missing_values=np.nan, strategy='mean')
imp.fit(df)
df_fill = imp.transform(df)
print("df_fill:")
print(df_fill)
```

输出

```
df(with NaN):
   A  B    C  D
0  7  7  1.0  6
1  1  2  NaN  4
2  5  7  3.0  8
```

```
missing_data:
   Total   Percent
C      1  0.333333
A      0  0.000000
B      0  0.000000
D      0  0.000000
```

```
df_fill:
[[7. 7. 1. 6.]
 [1. 2. 2. 4.]
 [5. 7. 3. 8.]]
```

这是使用的是均值填补，也就是取该列其他值的平均数：(1+3)/2=2。

使用中位数，众数，常量(不指定，默认为 0)，常量(指定为 1)

```
imp = SimpleImputer(missing_values=np.nan, strategy="median")
imp = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
imp = SimpleImputer(missing_values=np.nan, strategy="constant")
imp = SimpleImputer(missing_values=np.nan, strategy="constant", fill_value=1)
```

得到 2 1 0 1

噪声平滑

首先创建带有噪声的函数：

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0, 2 * np.pi, 500)  # 创建一个包含 500 个点的时间数组 t，范围是从 0 到 2π
y = 100 * np.sin(t)  # y = 100sint
```

```
noise = np.random.normal(0, 15, 500)  # 创建一个服从正态分布的，共 500 个值的噪声数组，均值为 0，标准差为 15
y = y + noise  # 添加噪声
plt.figure()  # 创建一个新的图形窗口
plt.plot(t, y)  # 绘制时间 t 与带有噪声的信号 y 之间的关系曲线
plt.xlabel('t')
plt.ylabel('y = sin(t) + noise')
plt.show()  # 显示图形
```

移动平均 Moving Average：

```
window_size = 10  # 窗口大小
y_smoothed = np.convolve(y, np.ones(window_size) / window_size, mode='same')  # mode='same'表示保持：与原始信号相同的长度
plt.figure()
plt.plot(t, y_smoothed)
plt.xlabel('t')
plt.ylabel('Smoothed y (Moving Average)')
plt.show()
```

中值滤波 Median Filtering：

```
from scipy.signal import medfilt
window_size = 11  # 窗口大小
y_smoothed = medfilt(y, kernel_size=window_size)  # 中值滤波器的窗口大小 kernel_size 要求是奇数
plt.figure()
plt.plot(t, y_smoothed)
plt.xlabel('t')
plt.ylabel('Smoothed y (Median Filtering)')
plt.show()
```

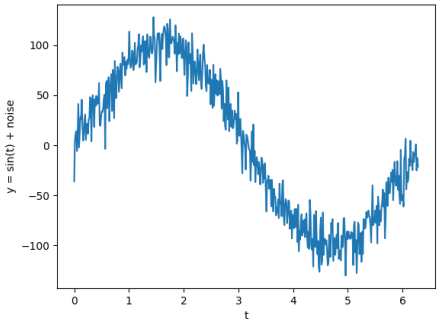
加权平均 Weighted Averaging：

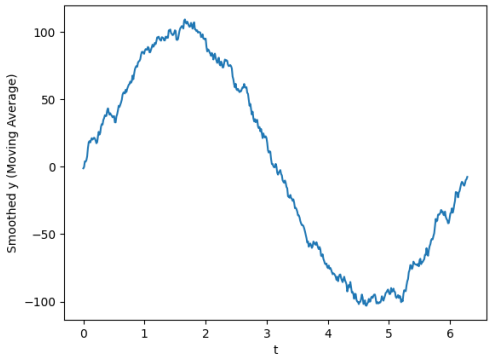
```
weights = np.exp(-np.arange(window_size) / window_size)  # 离中心位置越远的权重越小
weights /= np.sum(weights)  # 进行除法运算后的 weight 总和为 1
y_smoothed = np.convolve(y, weights, mode='same')
plt.figure()
plt.plot(t, y_smoothed)
plt.xlabel('t')
plt.ylabel('Smoothed y (Weighted Averaging)')
plt.show()
```

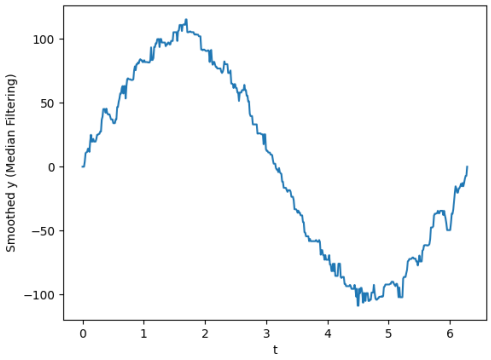
指数平滑 Exponential Smoothing：

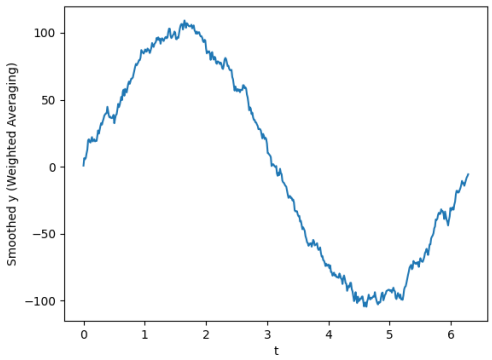
```
alpha = 0.2  # 数据波动大，平滑系数 α 取大一些。数据波动小就取小一些
y_smoothed = [y[0]]
for i in range(1, len(y)):
    y_smoothed.append(alpha * y[i] + (1 - alpha) * y_smoothed[i - 1])
plt.figure()
plt.plot(t, y_smoothed)
plt.xlabel('t')
plt.ylabel('Smoothed y (Exponential Smoothing)')
plt.show()
```

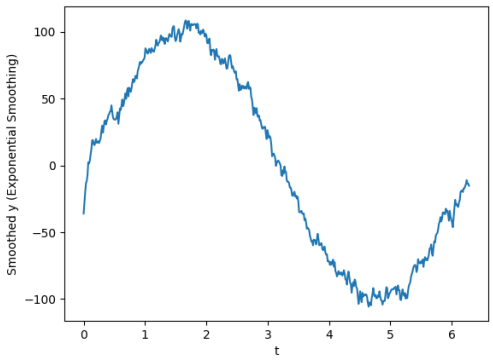
输出如下图：











然后来详细说说移动平均中的卷积（因为其他三个理解起来比较简单）：

移动平均：

convolve 计算的是 y 与[np.ones(window_size) / window_size]的卷积。

你应该知道 y 是 500 个数的数组，而 np.ones(window_size) / window_size =[0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]是 10 个数的数组。

那**什么是卷积**？

先说一个奇妙的比喻：如果你收到一个原味鸡肉卷，但又想吃酸甜口的鸡肉卷，那就打开鸡肉卷，把色拉酱均匀的淋下去，重新卷回来，你就可以美味享用了。

你可以把卷积想象成：把毛巾沿着某条线卷起来。卷起来后某点 A 处的函数值正好为这条直线上函数值的积分（这条直线即为 A 点在卷的时候沿卷的方向形成的线）。

而所谓两个函数的卷积，本质上就是先将一个函数翻转，然后进行滑动叠加。

因为卷积是为了表示一个函数对另一个函数所有微量的响应的总叠加效果。如果不翻转过来，叠加出来结果的**时序**就是反的，所以就人为规定把其中一个信号翻转过来。

Therefore, the operation is in some sense the “revolving” of one of the input functions with the other one(从某种意义上说，操作是将一个输入函数与另一个输入函数进行某种形式的"旋转"). Particularly in mathematics, physics, and related areas, the verb commonly used to designate such a revolving is “to convolve.” This verb comes from the Latin words con and volve˘re, which mean “**together**” and “**roll up**,” respectively; thus, convolve means “**roll up together**.” Accordingly, the action of convolving is called convolution.

卷积的数学定义：

称 $(f*g)(n)$ 为 f,g 的卷积（翻转积分/褶积）。

其连续的定义为：$(f*g)(n)=\int_{-\infty}^{\infty} f(\tau)g(n-\tau)\mathrm{d}\tau$

其离散的定义为：$(f*g)(n)=\sum_{\tau=-\infty}^{\infty} f(\tau)g(n-\tau)$

一些例子：

离散型的例子：骰子之和为 4 的概率

记 $f(n),g(n)$ 为投到 n 的概率， $P=f(1)g(3)+f(2)g(2)+f(3)g(1)=\sum_{m=1}^{3}f(4-m)g(m)=(f*g)(4)$

连续型的例子：馒头的腐败

记馒头生产速度为 $f(n)$，腐败函数为 $g(n)$，则 24 小时后馒头总共腐败了 $\int_0^{24}f(t)g(24-t)\mathrm{d}t$，

因为在 t 时刻产出的馒头只回腐败 $24-t$ 小时。（这就是为什么卷积要翻转一个函数，因为是物理意义给出的）

图像处理：

记 $g=\begin{bmatrix}b_{-1,-1}&b_{-1,0}&b_{-1,1}\\b_{0,-1}&b_{0,0}&b_{0,1}\\b_{1,-1}&b_{1,0}&b_{1,1}\end{bmatrix}$，新矩阵为 $c=\begin{bmatrix}c_{0,0}&\cdots&c_{0,n}\\\vdots&\ddots&\vdots\\c_{m,0}&\cdots&c_{m,n}\end{bmatrix}$，取 $f=\begin{bmatrix}a_{0,0}&a_{0,1}&a_{0,2}\\a_{1,0}&a_{1,1}&a_{1,2}\\a_{2,0}&a_{2,1}&a_{2,2}\end{bmatrix}$，则有：

$$c_{1,1}=f*g=\ a_{0,0}b_{1,1}+a_{0,1}b_{1,0}+a_{0,2}b_{1,-1}\ +\ a_{1,0}b_{0,1}+a_{1,1}b_{0,0}+a_{1,2}b_{0,-1}\ +\ a_{2,0}b_{-1,1}+a_{2,1}b_{-1,0}+a_{2,2}b_{-1,1}\ ,$$

即：$(f*g)(1,1)=\sum_{k=0}^{2}\sum_{h=0}^{2}f(h,k)g(1-h,1-k)$

二维离散形式的卷积公式：$(f*g)(u,v)=\sum_i\sum_j f(i,j)g(u-i,v-j)=\sum_i\sum_j a_{i,j}b_{u-i,v-j}$

对于一般的计算方法：

①在原始图像上取出点 (u,v) 处的矩阵（大小与 g 同型）：$f=\begin{bmatrix}a_{u-1,v-1}&a_{u-1,v}&a_{u-1,v+1}\\a_{u,v-1}&a_{u,v}&a_{u,v+1}\\a_{u+1,v-1}&a_{u+1,v}&a_{u+1,v+1}\end{bmatrix}$

②将图像处理矩阵 $g=\begin{bmatrix}b_{-1,-1}&b_{-1,0}&b_{-1,1}\\b_{0,-1}&b_{0,0}&b_{0,1}\\b_{1,-1}&b_{1,0}&b_{1,1}\end{bmatrix}$ 沿 x 轴翻转后沿 y 轴翻转得 $g'=\begin{bmatrix}b_{1,1}&b_{1,0}&b_{1,-1}\\b_{0,1}&b_{0,0}&b_{0,-1}\\b_{-1,1}&b_{-1,0}&b_{-1,-1}\end{bmatrix}$

③卷积即为 f,g' 的内积：$f*g(u,v)=\ a_{u-1,v-1}b_{1,1}+a_{u-1,v}b_{1,0}+a_{u-1,v+1}b_{1,-1}\ +$
$a_{u,v-1}b_{0,1}+a_{u,v}b_{0,0}+a_{u,v+1}b_{0,-1}\ +\ a_{u+1,v-1}b_{-1,1}+a_{u+1,v}b_{-1,0}+a_{u+1,v+1}b_{-1,-1}$

事实上，数学中的卷积和卷积神经网络 CNN 中的卷积严格意义上是两种不同的运算。

数学中卷积，主要是为了诸如信号处理、求两个随机变量和的分布等而定义的运算，所以需要“翻转”是根据问题的需要而确定的。

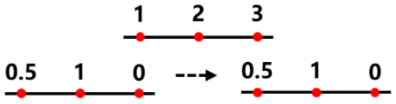
卷积神经网络中“卷积”，是为了提取图像的特征，其实只借鉴了“**加权求和**”的特点。

数学中的“卷积核”都是已知的或者给定的，卷积神经网络中“卷积核”本来就是 trainable 的参数，不是给定的，根据数据训练学习的，那么翻不翻转还有什么关系呢？因为无论翻转与否对应的都是未知参数！

具体图案可以参考：https://mlnotebook.github.io/post/CNN1/

计算举例：np.convolve([1, 2, 3], [0, 1, 0.5])

计算方法 1：将其中一个数组反向，将两个数组上下排放，然后上下对应的项相乘相加，如右图。



计算过程：

①0.5,1,0 在最左边，只有 0 与 1 对齐了，1*0=0　②0.5,1,0 往右平移一格，1*1+2*0=1

③右平移，1*0.5+2*1+3*0=2.5　④右平移，2*0.5+3*1=4　⑤右平移，3*0.5=1.5

答案：[0.　1.　2.5 4.　1.5]

计算方法 2：视作 x^2+2x+3 与 $0 \cdot x^2+x+0.5$ 相乘，得到 $0 \cdot x^4+x^3+2.5x^2+4x+1.5$

比如：

```
y = [0, 1, 3, 2, 3, 2, 3, 1, 4, 3, 4, 3]
print(np.convolve(y, [0.2, 0.2, 0.2, 0.2, 0.2]))
print(np.convolve(y, [0.2, 0.2, 0.2, 0.2, 0.2], mode='same'))
```

输出

```
[0.  0.2 0.8 1.2 1.8 2.2 2.6 2.2 2.6 2.6 3.  3.  2.8 2.  1.4 0.6]
[0.8 1.2 1.8 2.2 2.6 2.2 2.6 2.6 3.  3.  2.8 2. ]
```

将第二行与 y 相比

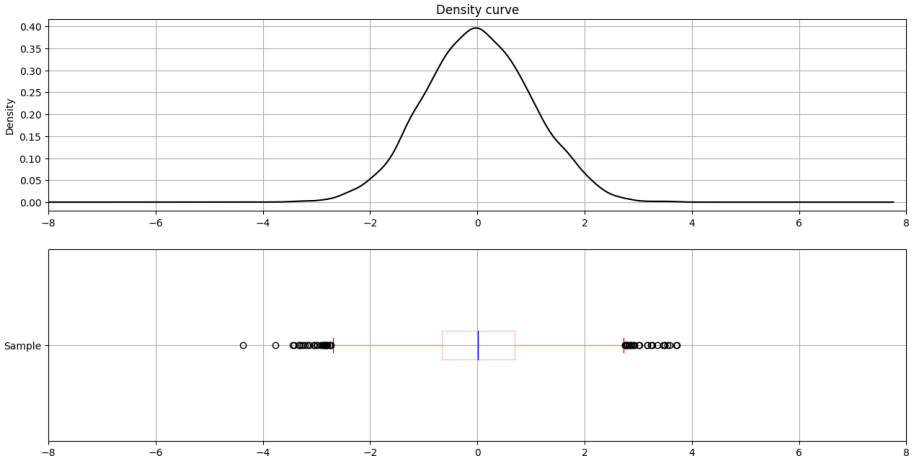
```
[0   1   3   2   3   2   3   1   4   3   4   3]
```

明显缓和了很多。

异常值的识别

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats

data = pd.Series(np.random.randn(10000))
u = data.mean()
std = data.std()
stats.kstest(data, 'norm', (u, std))
print("均值:%.3f,标准差:%.3f" % (u, std))

fig = plt.figure(figsize=(10, 6))
ax1 = fig.add_subplot(2, 1, 1)
data.plot(kind='kde', grid=True, style='-k', title='Density curve')
ax1.set_xlim(-8, 8)

error_std = data[np.abs(data - u) > 3 * std]
data_c_std = data[np.abs(data - u) <= 3 * std]
print("使用 3std 原则得到的异常值个数:%d" % (len(error_std)))

ax2 = fig.add_subplot(2, 1, 2)
color = dict(boxes='Pink', whiskers='Orange', medians='Blue', caps='Red')
data.plot.box(vert=False, grid=True, color=color, ax=ax2, label='Sample')
ax2.set_xlim(-8, 8)
plt.show()

q1 = data.quantile(0.25)
q3 = data.quantile(0.75)
IQR = q3 - q1
MIN = q1 - 1.5 * IQR
MAX = q3 + 1.5 * IQR
print("分位差:%.3f,下限:%.3f,上限:%.3f" % (IQR, MIN, MAX))

error_iqr = data[(data < MIN) | (data > MAX)]
data_c = data[(data <= MIN) & (data <= MAX)]
print("使用 IQR 原则得到的异常值个数:%d" % (len(error_iqr)))
```

图片输出



文字输出

```
均值:0.017,标准差:1.004
使用 3std 原则得到的异常值个数:27
分位差:1.360,下限:-2.703,上限:2.736
使用 IQR 原则得到的异常值个数:58
```

语法：

①stats.kstest(data, 'norm', (u, std))

data：输入的数据数组或序列，用于进行假设检验。

'norm'： 进行假设检验的分布函数，这里是正态分布。'uniform' 表示均匀分布、'expon' 表示指数分布等。

(u, std)：指定分布函数(这里是正态分布)的参数值，其中 u 是均值，std 是标准差。

该函数使用 Kolmogorov-Smirnov 检验来比较输入数据与指定的概率分布函数之间的拟合程度。输出如 KstestResult(statistic=0.006038808990869726, pvalue=0.8568670428232958, statistic_location=0.9815395126873121, statistic_sign=-1)所示。

②data.plot(kind='kde', grid=True, style='-k', title='Density curve')

kind='kde'：指定绘制的图形类型为核密度估计图（Kernel Density Estimate），该图形可以用来估计数据的概率密度函数。

grid=True：在图形中显示网格线。

style='-k'：设置曲线的样式为实线'-'且颜色为黑色'k'。

title='Density curve'：设置图形的标题。

图形绘制时，使用核密度估计方法对数据进行平滑处理，从而得到数据的概率密度曲线图 Density curve。

③data[np.abs(data - u) > 3 * std]是根据条件索引，选取满足条件的数据值构成一个新的数组 error_std。

3.数据集成

元组重复

```
import numpy as np
import pandas as pd
from scipy import stats

df = pd.read_csv('input/Ch3-HousePrice-train.csv')
YearBuilt = df.YearBuilt
YearRemodAdd = df.YearRemodAdd
print(stats.chisquare(YearBuilt, f_exp=YearBuilt))  # just 为了输出

# 将房屋面积和售价连接为 df_Area_Price
GrLivArea = df.GrLivArea
SalePrice = df.SalePrice
Area_Price = np.array([GrLivArea, SalePrice])
df_Area_Price = pd.DataFrame(Area_Price.T, columns=['GrLivArea', 'SalePrice'])

covariance = df_Area_Price.GrLivArea.cov(df_Area_Price.SalePrice)
print(covariance)
correlation = df_Area_Price.GrLivArea.corr(df_Area_Price.SalePrice)
print(correlation)
```

输出

```
Power_divergenceResult(statistic=0.0, pvalue=1.0)
29581866.743236598
0.7086244776126521
```

语法：scipy.stats.chisquare(f_obs, f_exp=None, ddof=0, axis=0)

参数：f_obs 观测频率，f_exp 预期频率(默认值 the categories are equally likely)

返回值：statistic：卡方统计量，度量观测值与期望值之间的**偏离程度**。较大的统计量说明存在显著的差异，反之则意味着观测值与期望值非常接近，没有显著的差异。

pvalue：卡方检验的 p 值，用于确定观测值与期望值之间的**差异是否显著**。如果 pvalue 非常小（通常小于显著性水平，例如 0.05）表示差异是显著的，我们可以拒绝零假设。反之则不能拒绝零假设，表示没有足够的证据表明观测值与期望值之间存在显著差异。

书中使用 print(stats.chisquare(YearBuilt, f_exp=YearRemodAdd))会报错 ValueError: For each axis slice, the sum of the observed frequencies must agree with the sum of the expected frequencies to a relative tolerance of 1e-08, but the percent differences are:0.006898070951487656(对于每个 axis ，f_obs 的总和与 f_exp 的总和必须一致(容忍量为 1e-08))，因为可以通过检验协方差和相关系数=369.67545607330976，0.5928549763436504，由此说明这两个其实偏离程度很大。

协方差 cov 测量两个变量之间的总体线性关系。正协方差表示正相关，负协方差表示负相关，协方差接近零表示变量之间没有明显的线性关系。协方差的值的大小并不容易解释，因为它受到变量单位的影响。

相关系数 corr 是协方差的标准化版本，衡量两个变量之间的线性关系的**强度和方向**。接近 1 或-1，表示存在强烈的线性关系，接近 0 表示两个变量之间几乎没有线性关系。

4.数据归约

PCA 与属性子集选择

```
import numpy as np
import pandas as pd
from sklearn.feature_selection import RFE
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA

data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep=r"\s+", skiprows=22, header=None)  # 跳过了文件的前 22 行，没有使用文件的第一行作为列名

# data 包含了特征数据，而 target 包含了目标数据。
# 因为原数据集是从 23 行开始，每两行对应一行的数据。
# 所以 data 选择 raw_df 中的奇数行（raw_df.values[::2, :]）和偶数行的前两列（raw_df.values[1::2, :2]）
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
# target 由 raw_df 中的偶数行的第三列（raw_df.values[1::2, 2]）组成
target = raw_df.values[1::2, 2]
print(data.shape)

pca = PCA()
pca.fit(data)
print(pca.explained_variance_ratio_)  # 输出方差百分比

pca = PCA(3)
pca.fit(data)
low_d = pca.transform(data)  # 降维
print(low_d.shape)

names = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT", "MEDV"]
IR = LinearRegression()  # 创建了一个线性回归模型的实例
# 创建了一个递归特征消除（RFE）对象
rfe = RFE(IR, n_features_to_select=1)  # 使用指定的模型(线性回归模型 IR)来选择 n_features_to_select 个最重要的特征。
rfe.fit(data, target)  # 使用 RFE 对象拟合数据集和目标变量。RFE 将使用线性回归模型来评估每个特征的重要性，并选择最重要的特征。
# 将 RFE 对象的特征排名(ranking)与特征名称进行组合，然后对其进行排序。
# map(lambda x: round(x, 4), rfe.ranking_) 用于将排名四舍五入到小数点后四位。最后，将排名和特征名称组合在一起并打印出来。
print(sorted(zip(map(lambda x: round(x, 4), rfe.ranking_), names)))
```

输出

```
(506, 13)
[8.05823175e-01 1.63051968e-01 2.13486092e-02 6.95699061e-03
 1.29995193e-03 7.27220158e-04 4.19044539e-04 2.48538539e-04
 8.53912023e-05 3.08071548e-05 6.65623182e-06 1.56778461e-06
 7.96814208e-08]
(506, 3)
[(1, 'NOX'), (2, 'RM'), (3, 'CHAS'), (4, 'PTRATIO'), (5, 'DIS'), (6, 'LSTAT'), (7, 'RAD'), (8, 'CRIM'), (9, 'INDUS'), (10, 'ZN'), (11, 'TAX'), (12, 'B'), (13, 'AGE')]
```

从方差百分比可以看出前三维的贡献率达到 99%以上(8.05823175e-01+1.63051968e-01+2.13486092e-02)。可以用来判断应该选取几个主成分。

在递归特征消除（RFE）中，目标是识别哪些特征对模型预测的贡献最大。从全集开始，每次删除当前最坏的属性，剩下的特征将被用于重新训练模型。以此类推，每次迭代都会删

除一个最不重要的特征，直到所有特征都被考虑。因此，RFE 的排名指的是特征被删除的顺序。排名 1 的特征是最后一个被考虑删除的，也就是说它是最重要的。

数据集描述：

1- CRIM	犯罪率	2- ZN	超过 25,000 sq.ft.的住宅用地比例
3- INDUS	非零售商业用地占比	4- CHAS	是否临 Charles 河
5- NOX	氮氧化物浓度	6- RM	房屋房间数
7- AGE	房屋年龄	8- DIS	和就业中心的距离
9- RAD	是否容易上高速路	10- TAX	税率
11- PTRATIO	学生人数比老师人数	12- B	城镇黑人比例计算的统计值
13- LSTAT	低收入人群比例	14- MEDV	房价中位数

5.数据变换

数据规范化

先说一个 bug 的解决方法：

```
original_data1 = [-2, -1, 0, 1, 2, 3, 4]
original_data1 = np.array(original_data1)
min_val = original_data1.min
max_val = original_data1.max
min_max_normalized = (original_data1 - min_val) / (max_val - min_val)
```

报错 TypeError: unsupported operand type(s) for -: 'int' and 'builtin_function_or_method'，并且输出 min_val 是<built-in method min of numpy.ndarray object at 0x000001F2666F51D0>

这是因为省略了 min 的()会导致 Python 将其**解释为属性而不是函数调用**。

前置知识：函数加括号是指对此**函数**的调用，函数不加括号是指调用函数本身（的**内存地址**），也可以理解成对函数重命名。

通过以下代码很容易知道区别：

```
def greet(name):
    return f"Hello, {name}!"

# 函数加括号是函数调用，返回函数执行结果
message = greet("A")  # 调用函数 greet，将返回值赋给变量 message
print(message)  # 输出 "Hello, A!"

# 函数不加括号，将函数本身赋值给一个变量
greeting_function = greet
# 此时 greeting_function 包含了函数 greet 的引用

# 使用函数引用来调用函数
message = greeting_function("B")  # 调用 greeting_function，其实是调用 greet
print(message)  # 输出 "Hello, B!"
```

同理：

```
a = min_val()
print(a)  # 输出 -2
```

因为 min_val 其实就是 original_data1.min 了，original_data1.min()理所当然的就是-2。

各种规范化方法的实现：

```
import numpy as np


class DataNorm:
    def __init__(self, input_array):
        self.arr = np.array(input_array)
        self.x_max = self.arr.max(initial=None)  # 最大值
        self.x_min = self.arr.min(initial=None)  # 最小值
        self.x_mean = self.arr.mean()  # 平均值
        self.x_std = self.arr.std()  # 标准差
        self.decimals_setting = 3  # 小数位数

    def Min_MaxNorm(self):
        arr = np.around(((self.arr - self.x_min) / (self.x_max - self.x_min)),
decimals=self.decimals_setting)
        print("Min_Max 标准化:{}".format(arr))

    def Z_ScoreNorm(self):
        arr = np.around((self.arr - self.x_mean) / self.x_std,
decimals=self.decimals_setting)
        print("Z_Score 标准化:{}".format(arr))

    def Decimal_ScalingNorm(self):
        power = 1
```

```
        maxValue = self.x_max
        while maxValue / 10 >= 1.0:
            power += 1
            maxValue /= 10
        arr = np.around((self.arr / pow(10, power)), decimals=self.decimals_setting)
        print("小数定标标准化:{}".format(arr))

    def MeanNorm(self):
        first_arr = np.around((self.arr - self.x_mean) / (self.x_max - self.x_min),
decimals=self.decimals_setting)
        second_arr = np.around((self.arr - self.x_mean) / self.x_max,
decimals=self.decimals_setting)
        print("均值归一法：\n 公式一(分母 max):{}\n 公式二(分母 max-
min):{}".format(first_arr, second_arr))

    def Vector(self):
        if self.arr.sum() != 0:
            arr = np.around((self.arr / self.arr.sum()),
decimals=self.decimals_setting)
        else:
            arr = "未满足使用 sum()的要求(数据的和不为 0)"
        print("向量归一法:{}".format(arr))

    def exponential(self):
        if all(x > 1 for x in self.arr):
            first_arr = np.around(np.log10(self.arr) / np.log10(self.x_max),
decimals=self.decimals_setting)
        else:
            first_arr = "未满足使用 log10 的要求(数据均大于 1)"
        second_arr = np.around(np.exp(self.arr) / sum(np.exp(self.arr)),
decimals=self.decimals_setting)
        third_arr = np.around(1 / (1 + np.exp(-1 * self.arr)),
decimals=self.decimals_setting)
        print("lg 函数:{}\nSoftmax 函数:{}\nSigmoid 函数:{}\n".format(first_arr,
second_arr, third_arr))


if __name__ == "__main__":
    original_data1 = [-2, -1, 0, 1, 2, 3, 4]
    original_data2 = [-3, -2, -1, 0, 1, 2, 3]
    # 创建类的实例，并传递数组作为参数
    data1 = DataNorm(original_data1)
    data2 = DataNorm(original_data2)

    def FuckingFunction(data):
        data.Min_MaxNorm()
        data.Z_ScoreNorm()
        data.Decimal_ScalingNorm()
        data.MeanNorm()
        data.Vector()
        data.exponential()

    FuckingFunction(data1)
    FuckingFunction(data2)
```

输出：

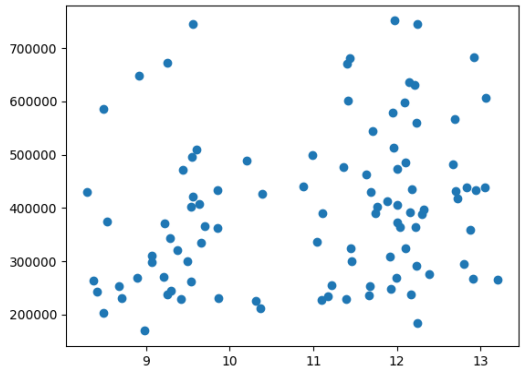
```
Min_Max标准化:[0.    0.167 0.333 0.5   0.667 0.833 1.   ]
Z_Score标准化:[-1.5 -1.  -0.5  0.   0.5  1.   1.5]
小数定标标准化:[-0.2 -0.1  0.   0.1  0.2  0.3  0.4]
均值归一法:
公式一(分母max):[-0.5   -0.333 -0.167  0.     0.167  0.333  0.5  ]
公式二(分母max-min):[-0.75 -0.5  -0.25  0.    0.25  0.5   0.75]
向量归一法:[-0.286 -0.143  0.     0.143  0.286  0.429  0.571]
lg函数:未满足使用log10的要求(数据均大于1)
Softmax函数:[0.002 0.004 0.012 0.032 0.086 0.233 0.633]
Sigmoid函数:[0.119 0.269 0.5   0.731 0.881 0.953 0.982]
```

```
Min_Max标准化:[0.    0.167 0.333 0.5   0.667 0.833 1.   ]
Z_Score标准化:[-1.5 -1.  -0.5  0.   0.5  1.   1.5]
小数定标标准化:[-0.3 -0.2 -0.1  0.   0.1  0.2  0.3]
均值归一法:
公式一(分母max):[-0.5   -0.333 -0.167  0.     0.167  0.333  0.5  ]
公式二(分母max-min):[-1.    -0.667 -0.333  0.     0.333  0.667  1.   ]
向量归一法:未满足使用sum()的要求(数据的和不为0)
lg函数:未满足使用log10的要求(数据均大于1)
Softmax函数:[0.002 0.004 0.012 0.032 0.086 0.233 0.633]
Sigmoid函数:[0.047 0.119 0.269 0.5   0.731 0.881 0.953]
```

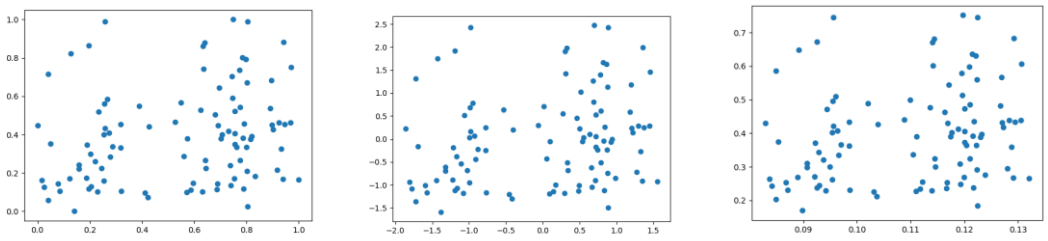
实际应用：

```
import numpy as np
import pandas as pd
import sklearn.preprocessing as skp
import matplotlib.pyplot as plt

df = pd.read_csv('input/Ch3-StockTrading.csv')
print(df.describe())

# 选择最后 100 行的 close 和 volume 两列，.values 将 DataFrame 转换为 NumPy 数组
data_original = df.tail(100)[['close', 'volume']].values
plt.scatter(data_original[:, 0], data_original[:, 1])
plt.show()
```

	Unnamed: 0	open	close	high	low	volume	code
count	640.000000	640.000000	640.000000	640.000000	640.000000	6.400000e+02	640.0
mean	319.500000	11.990019	11.991128	12.236808	11.756233	2.269220e+05	300274.0
std	184.896367	3.682276	3.688834	3.791381	3.574300	1.714809e+05	0.0
min	0.000000	5.220000	5.330000	5.470000	5.170000	3.066200e+04	300274.0
25%	159.750000	9.568500	9.582500	9.717500	9.402250	9.588225e+04	300274.0
50%	319.500000	11.275000	11.285000	11.439500	11.137000	1.879755e+05	300274.0
75%	479.250000	15.060500	14.976500	15.373500	14.726000	3.026530e+05	300274.0
max	639.000000	21.870000	21.563000	22.495000	20.749000	1.472658e+06	300274.0




```
data_scale = skp.MinMaxScaler().fit_transform(data_original)
plt.scatter(data_scale[:, 0], data_scale[:, 1])
plt.show()

data_zscore = skp.scale(data_original)
plt.scatter(data_zscore[:, 0], data_zscore[:, 1])
plt.show()

scaling_factor_close = 10**np.ceil(np.log10(np.max(np.abs(data_original[:, 0]))))
scaling_factor_volume = 10**np.ceil(np.log10(np.max(np.abs(data_original[:, 1]))))
data_decimal = data_original
for index in range(len(data_decimal)):
    data_decimal[index, 0] = data_decimal[index, 0]/scaling_factor_close
    data_decimal[index, 1] = data_decimal[index, 1]/scaling_factor_volume
plt.scatter(data_decimal[:, 0], data_decimal[:, 1])
plt.show()
```



One-hot 编码

```
from sklearn import preprocessing

enc = preprocessing.OneHotEncoder()  # 创建对象
enc.fit([[0, 0, 3], [1, 1, 0], [0, 2, 1], [1, 0, 2]])  # 拟合
array = enc.transform([[0, 1, 3]]).toarray()  # 转化
print(array)
```

输出

```
[[1. 0. 0. 1. 0. 0. 0. 0. 1.]]
```

fit 的具体过程：

对于训练对象 $\begin{pmatrix}0&0&3\\1&1&0\\0&2&1\\1&0&2\end{pmatrix}$：第一列是二元特征(0,1)，编码为 $\begin{matrix}0\to(1,0)\\1\to(0,1)\end{matrix}$。第二列是三元特征(0,1,2)，编码为 $\begin{matrix}0\to(1,0,0)\\1\to(0,1,0)\\2\to(0,0,1)\end{matrix}$。第三列是四元特征(0,1,2,3)，编码为 $\begin{matrix}0\to(1,0,0,0)\\1\to(0,1,0,0)\\2\to(0,0,1,0)\\3\to(0,0,0,1)\end{matrix}$。

因此对于[0,1,3]，第一列是 0，对应(1,0)，第二列的 1 对应(0,1,0)，第三列的 3 对应(0,0,0,1)，所以输出的是 1,0,0,1,0,0,0,0,1。

TF-IDF

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer

tag_list = ['hutao klee keqing',
            'klee klee klee',
            'hutao klee klee',
            'klee hutao hutao']
vectorizer = CountVectorizer()  # 将文本中的词语转换为词频矩阵
X = vectorizer.fit_transform(tag_list)  # 计算每个词语出现的次数
print(X)  # X 形如(a,b) c，表示再第 a 个文档中编号为 b 的单词出现了 c 次
print(vectorizer.vocabulary_)  # word_dict。顺序对应出现顺序，编码依据字母顺序

transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X)  # 将词频矩阵 X 统计成 TF-IDF 值
print(tfidf.toarray())
```

输出(左：X、vectorizer.vocabulary_，右：tfidf.toarray())

```
  (0, 0)    1
  (0, 2)    1
  (0, 1)    1
  (1, 2)    3
  (2, 0)    1
  (2, 2)    2
  (3, 0)    2
  (3, 2)    1
{'hutao': 0, 'klee': 2, 'keqing': 1}
```

```
[[0.49248889 0.77157901 0.40264194]
 [0.         0.         1.        ]
 [0.52173612 0.         0.85310692]
 [0.92564688 0.         0.37838849]]
```

离散化

```
import pandas as pd
from sklearn.cluster import KMeans

df_train = pd.read_csv('input/Ch3-E-Commerce.csv', encoding='ISO-8859-1')
total = df_train.isnull().sum().sort_values(ascending=False)  # 计算了每列中的缺失值总数，并按降序排列
```

```
percent = (df_train.isnull().sum() /
df_train.isnull().count()).sort_values(ascending=False)  # 缺失值百分比 降序排列
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])  # 合并 total 与 percent
print(missing_data)
price = df_train['UnitPrice']  # 通过直方图可以发现其分布极其不均衡(横跨范围很大[-11062.06, 38970.0]，但非常集中[10%:0.63, 90%:7.95])

""" 等宽离散化 """
K = 5  # 将价格划分为 K 个区间
df_train['price_discretized_1'] = pd.cut(price, K, labels=range(K))  # pd.cut 将 price 划分为 K 个区间，labels 即组名称
print(df_train.groupby('price_discretized_1')['price_discretized_1'].count())  # 照 price_discretized_1 列的值进行分组，并计算每个组的数量，最后打印结果

""" 等频离散化 """
w = [1.0*i/K for i in range(K+1)]  # [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
# 为什么是加 2？首先，分割点应该比区间 K 多 1，其次 describe 自带一个 50%项(下文会删除这个 50%项)。
# 如果只+1 会发现 w 少了"100%"这一项，需要 pd.concat([w, pd.Series([price.max()], index=["100%"])])重新补回来
w = price.describe(percentiles=w)[4:4+K+2]  # 前四项是 count,mean,std,min
w = w.drop('50%')  # 删除 index="50%"的 w
df_train['price_discretized_2'] = pd.cut(price, w, labels=range(K))  # 原数据集 541909 项，等频后 541907 项
print(df_train.groupby('price_discretized_2')['price_discretized_2'].count())

""" K 均值离散化 """
price_re = price.values.reshape((price.index.size, 1))  # price.index.size = 541909。将 p 一维数组 rice 重塑为二维[541909,1]
k_model = KMeans(n_clusters=K, n_init=10)
df_train['price_discretized3'] = k_model.fit_predict(price_re)
print(df_train.groupby('price_discretized3')['price_discretized3'].count())
```

输出(省略查看缺失值的输出)

```
price_discretized_1
0          2
1     541897
2          9
3          0
4          1
Name: price_discretized_1, dtype: int64
price_discretized_2
0     113124
1     119971
2      93042
3     123061
4      92709
Name: price_discretized_2, dtype: int64
price_discretized3
0     541861
1          1
2          9
3         36
4          2
Name: price_discretized3, dtype: int64
```

6.Sklearn

数据集的使用

```
from sklearn import datasets
print(dir(datasets))  # 这个列表包括了 datasets 模块中的各种函数、类、子模块等。
```

输出

```
['__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__getattr__', '__loader__', '__name__', '__package__', '__path__', '__spec__', '_arff_parser', '_base', '_california_housing', '_covtype', '_kddcup99', '_lfw', '_olivetti_faces', '_openml', '_rcv1', '_samples_generator', '_species_distributions', '_svmlight_format_fast', '_svmlight_format_io', '_twenty_newsgroups', 'clear_data_home', 'dump_svmlight_file', 'fetch_20newsgroups', 'fetch_20newsgroups_vectorized', 'fetch_california_housing', 'fetch_covtype', 'fetch_kddcup99', 'fetch_lfw_pairs', 'fetch_lfw_people', 'fetch_olivetti_faces', 'fetch_openml', 'fetch_rcv1', 'fetch_species_distributions', 'get_data_home', 'load_breast_cancer', 'load_diabetes', 'load_digits', 'load_files', 'load_iris', 'load_linnerud', 'load_sample_image', 'load_sample_images', 'load_svmlight_file', 'load_svmlight_files', 'load_wine', 'make_biclusters', 'make_blobs', 'make_checkerboard', 'make_circles', 'make_classification', 'make_friedman1', 'make_friedman2', 'make_friedman3', 'make_gaussian_quantiles', 'make_hastie_10_2', 'make_low_rank_matrix', 'make_moons', 'make_multilabel_classification', 'make_regression', 'make_s_curve', 'make_sparse_coded_signal', 'make_sparse_spd_matrix', 'make_sparse_uncorrelated', 'make_spd_matrix', 'make_swiss_roll', 'textwrap']
```

①数据集：Iris

```
Iris = datasets.load_iris()  # 导入鸢尾花数据集
data = Iris.data  # 获得样本特征向量
target = Iris.target  # 获得样本 label
# print(data, target)  # 查看 data(4 个属性)与 target(3 个属性值)
print(type(data), type(target))  # 均为<class 'numpy.ndarray'>
print(Iris.feature_names)  # 4 个属性特征
print(Iris.target_names)  # 3 种鸢尾花的名称
print(Iris.data.shape, Iris.target.shape)  # 规格
```

导入： Iris = datasets.load_iris()

描述： 含了三种不同种类的鸢尾花(Iris)的测量数据。

目的： 用于模式识别、分类问题、聚类分析和数据可视化等领域。

规格： data(150, 4)　target(150,)

属性解释：

data

属性	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
解释	花萼长度	花萼宽度	花瓣长度	花瓣宽度
第一行的值	5.1	3.5	1.4	0.2

target

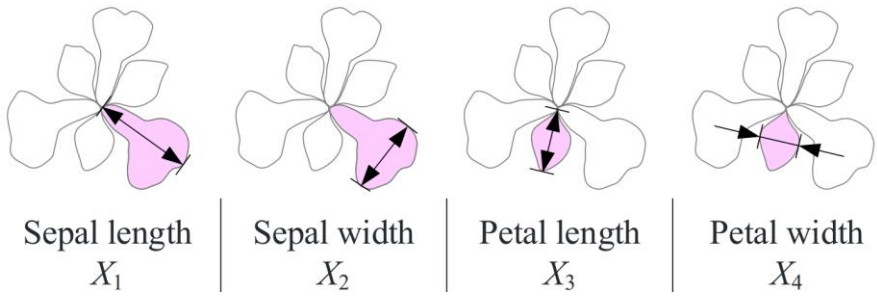
setosa	versicolor	virginica
山鸢尾	变色鸢尾	维吉尼亚鸢尾
0	1	2

注：

山鸢尾 Iris setosa：一种较小的鸢尾花，通常花瓣较短，直立生长，花朵白色。

变色鸢尾 Iris versicolor：具有较长的花瓣，通常为淡蓝色或紫色，花朵的颜色会变化。

维吉尼亚鸢尾 Iris virginica：花瓣较长，通常为深紫色。



②数据集 Boston

```
import numpy as np
import pandas as pd
```

```
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep=r"\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
print(data.shape, target.shape)  # 规格
np.set_printoptions(threshold=np.inf)  # 设置打印选项，将数组的所有元素都显示出来
np.set_printoptions(suppress=True)  # 设置打印选项，将数组的元素以普通小数表示法显示
print(data)
print(target)
```

描述：提供了与波士顿地区不同社区相关的一些特征，以及这些社区的房价中位数。

目的：房价预测。评估各种回归算法，如线性回归、决策树回归、随机森林回归等，以了解不同特征如何影响房价，以及如何建立准确的房价预测模型。

规格：data(506, 13)　target(506,)

属性解释：

属性	解释	第一行的值
CRIM	犯罪率	0.00632
ZN	>25000ft^2 的规划住宅用地面积	18.00
INDUS	非零售商业用地占比	2.310
CHAS	是否临 Charles 河(1 临 0 不临)	0
NOX	一氧化氮浓度	0.5380
RM	住宅平均房间数目	6.5750
AGE	1940 年以前建成用房的居住率	65.20
DIS	和波士顿五大就业中心的距离	4.0900
RAD	是否容易上高速路	1
TAX	全额财产税的税率	296.0
PTRATIO	城镇小学教师的比例	15.30
B	黑人所占人口比例	396.9
LSTAT	低收入人群比例	4.98
MEDV	居住房屋的房价中位数(千美元)	24.00

③数据集 Digits

```
import matplotlib.pyplot as plt
Digits = datasets.load_digits()  # 导入手写数字数据集
data = Digits.data
target = Digits.target
images = Digits.images
print(data.shape)  # (1797, 64)
print(target.shape)  # (1797,)
print(images.shape)  # (1797, 8, 8)
print(Digits.feature_names)  # 64 个属性特征 pixel_0_0~pixel_7_7
print(Digits.target_names)  # 数字 0~9
plt.matshow(Digits.images[0])  # 绘制第一个 image
plt.show()
print(Digits.target[0])  # 查看第一个 image 对应是数字是几
```

导入：Digits = datasets.load_digits()

描述：包含一系列手写数字的图像，每个图像都代表一个从 0 到 9 的数字。

目的：数字识别和分类任务(支持向量机、神经网络、决策树等)。

规格：data (1797, 64)　target(1797,)　image(1797, 8, 8)

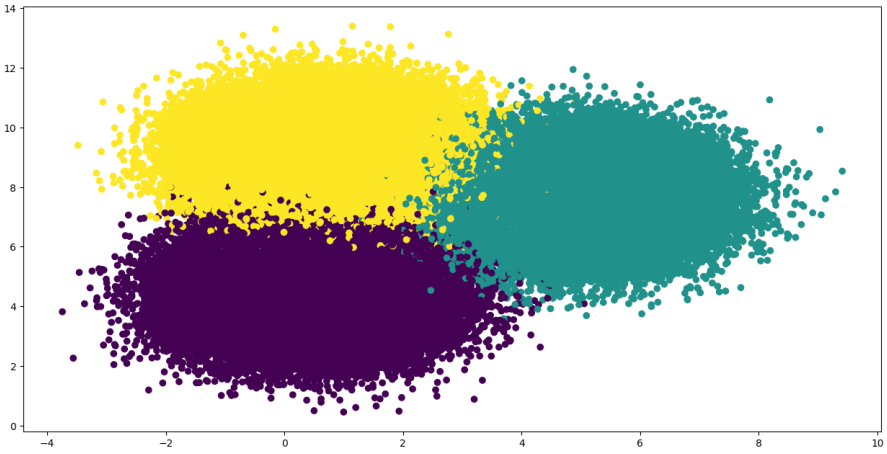
数据特点：总共 1797 张 8x8 像素的图像。每个图像代表一个手写数字，10 个类别分别对应一个数字。每个像素的值表示图像中的灰度强度。

注：

data 与 image 的区别：data 被展平为一维数组，image 以 8*8 二维矩阵保存。data 可以被看作是将 images 中的每个二维图像展平为一个一维数组的结果(扁平化结果)。

④自拟数据集

```
from sklearn.datasets import make_regression
from sklearn.datasets import make_classification
from sklearn.datasets import make_blobs

X, y = make_regression(n_samples=300, n_features=10, n_targets=1)  # 创建回归数据集
X, y = make_classification(n_samples=300, n_features=10, n_classes=2)  # 创建分类数据集
X, y = make_blobs(n_samples=221027, n_features=2, centers=3)  # 创建分类数据集
plt.scatter(X[:, 0], X[:, 1], c=y)  # 展示聚类的散点图
plt.show()
```



make_regression:

n_samples:样本数量。n_features:特征数量。n_targets:生成的目标(输出)的数量。

make_classification:

n_samples:样本数量。n_features:特征数量。n_classes:类别的数量。

make_blobs:

n_samples:样本数量。n_features:特征数量。centers:聚类的中心数量(簇的数量)。

数据集的拆分

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, target, test_size=0.4, random_state=0)
print(X_train.shape, X_test.shape)  # (90, 4) (60, 4)  [y_test,y_train 是(90,) (60,)]
```

train_test_split:

data：特征向量。target：样本标签。

test_size：测试集大小，浮点数表示测试样本所占百分比，整数表示多少个测试样本。

random_state：随机种子，非 0 代表随机数编号，每次运行代码时，分割数据的结果都将是相同的。0 或不填每次产生的划分都是不一样的。

返回四个数组：X_train：训练集的特征向量。X_test：测试集的特征向量。y_train：训练集的标签。y_test：测试集的标签。

预处理模块、常用模型的导入语句

```
from sklearn.preprocessing import StandardScaler  # 数据标准化
from sklearn.preprocessing import MinMaxScaler  # 数据归一化
from sklearn.preprocessing import LabelEncoder  # 标签编码
from sklearn.preprocessing import OneHotEncoder  # 独热编码
from sklearn.impute import SimpleImputer  # 缺失值处理
from sklearn.preprocessing import PolynomialFeatures  # 多项式特征生成
from sklearn.feature_selection import SelectKBest  # 特征选择
from sklearn.feature_selection import VarianceThreshold  # 方差阈值
from sklearn.decomposition import PCA  # 主成分分析
from sklearn.decomposition import NMF  # 非负矩阵分解

from sklearn.linear_model import LinearRegression  # 线性回归
from sklearn.linear_model import LogisticRegression  # 逻辑回归
from sklearn.svm import SVC  # 支持向量机分类
from sklearn.svm import SVR  # 支持向量机回归
from sklearn import naive_bayes  # 朴素贝叶斯算法 NB
from sklearn import tree  # 决策树算法
from sklearn import neighbors  # K 临近算法
from sklearn import neural_network  # 神经网络
from sklearn.ensemble import RandomForestClassifier  # 随机森林分类
from sklearn.ensemble import RandomForestRegressor  # 随机森林回归
from sklearn.neighbors import KNeighborsClassifier  # K 近邻分类
from sklearn.neighbors import KNeighborsRegressor  # K 近邻回归
from sklearn.tree import DecisionTreeClassifier  # 决策树分类
from sklearn.tree import DecisionTreeRegressor  # 决策树回归
from sklearn.cluster import KMeans  # K 均值聚类
from sklearn.naive_bayes import GaussianNB  # 朴素贝叶斯分类
from sklearn.neural_network import MLPClassifier  # 多层感知器分类
from sklearn.neural_network import MLPRegressor  # 多层感知器回归
from sklearn.ensemble import GradientBoostingClassifier  # 梯度提升分类
from sklearn.ensemble import GradientBoostingRegressor  # 梯度提升回归
```

模型预测与评估

依据 Iris 数据集的导入与模型的导入，有：

```
model = SVC()  # 创建一个 SVC 模型的实例
model.fit(X_train, y_train)  # 拟合模型--使用训练数据进行模型训练
y_pre = model.predict(X_test)  # 预测模型--使用训练好的模型对测试数据进行预测
print(model.get_params())  # 获得模型的参数
print(model.score(X_test, y_test))  # 模型在测试数据的准确度
print(y_pre)  # 模型对测试数据的预测结果
print(y_test)  # 测试数据的实际结果
```

输出(预测与实际不相同的 4 个数据已经加粗)

```
{'C': 1.0, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma':
'scale', 'kernel': 'rbf', 'max_iter': -1, 'probability': False, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}
0.9333333333333333
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 1 0 0 1 1 0 2 1 0 2 2 1 0
 2 1 1 2 0 2 0 0 1 2 2 1 2 1 2 1 1 2 2 1 2 1 2]
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
 1 1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2]
```

Sklearn 的模型接口都是类似的。

fit()用于模型的训练，传入两个参数(训练集数据和训练集标签)。

predict()用于模型的预测，传入测试集数据，输出预测的结果。

get_pramas()用于获得模型参数。

score()用于对训练好的模型打分。

模型保存与加载

两种方法二选一：

```
import pickle
with open('model.pickle', 'wb') as f:  # 模型保存
    pickle.dump(model, f)
with open('model.pickle', 'rb') as f:  # 模型加载
    model = pickle.load(f)
model.predict(X_test)
```

```
import joblib
joblib.dump(model, 'model2.pickle')  # 模型保存
model = joblib.load('model2.pickle')  # 模型加载
```