



# 《python数据分析案例教程》

## 第六章 numpy科学计算

6.1 numpy库简介

6.2 数组对象ndarray

6.3 numpy矩阵

6.4 精选案例

6.4.1 美国总统大选数据统计

6.4.2 约会配对案例



**目录** content





# PART ONE

## 6.1 numpy库简介

Introduction to numpy Library

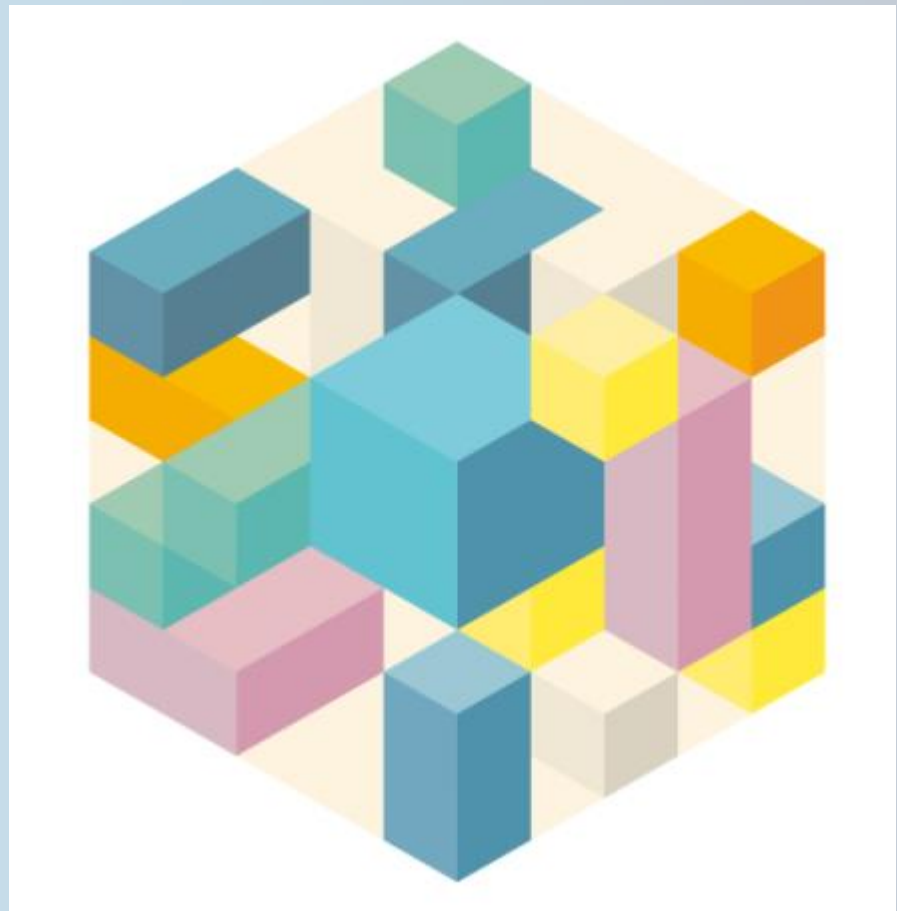
## 6. 1 numpy库简介

Introduction to numpy Library

### ● numpy库

numpy是Python支持科学计算的重要扩展库，也是数据分析领域必备的基础包，提供众多功能，包括：

- 创建n维数组（矩阵）
- 对数组进行快速运算
- 数值积分
- 线性代数运算
- 傅里叶变换
- 随机数产生
- .....



## 6. 1 numpy库简介

Introduction to numpy Library

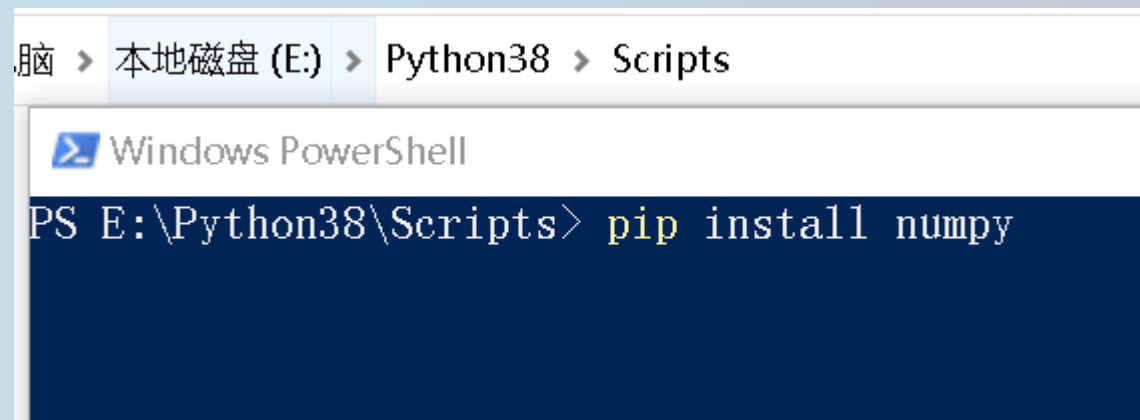
### ● numpy库的安装

numpy是Python的第三方库，在使用之前，需要借助Python工具进行numpy库安装。

在Scripts文件夹下打开命令窗口，运行命令：`pip install numpy`

若在Anaconda集成化环境中，用户可以直接导入numpy而无需额外安装，因为numpy已经默认安装到Anaconda环境。

numpy模块安装完毕，通常使用命令“`import numpy as np`”导入numpy库。这样，在程序的后续部分，使用np代替numpy调用库函数。



The screenshot shows a Windows PowerShell terminal window. The address bar at the top indicates the current directory is '本地磁盘 (E:) > Python38 > Scripts'. The title bar of the window is 'Windows PowerShell'. The command prompt shows the command `PS E:\Python38\Scripts> pip install numpy` being entered.



# PART TWO

## 6.2 数组对象ndarray

Array object ndarray



## 6.2 数组对象ndarray

Array object ndarray

### ● 数组对象ndarray



标准的Python中用list（列表）保存值，可以当作数组使用，但因为列表中的元素可以是任何对象，所以浪费了CPU运算时间和内存。

numpy提供了一个具有**矢量算术运算能力**和**复杂广播能力**的ndarray数组对象。为了保证快速运算且节省空间的优良性能，ndarray对象中许多操作采用代码在本地编译执行的方式。

## 6.2 数组对象ndarray

Array object ndarray

### ● numpy数组 VS Python列表

numpy数组与Python列表的主要区别如下：


1. numpy数组具有固定大小，更改numpy数组的大小将创建一个新数组并删除原来的numpy数组。而Python列表对象包含的元素数目是可以动态增长的。
2. numpy数组中的元素通常具有相同的数据类型，因此在内存中占据的存储空间相同。而Python列表元素可以是不同类型的数据。
3. numpy数组可以实现高效快速的矢量算术运算。与Python列表相比，numpy数组无需使用循环语句，可以完成类似Matlab中的矢量运算，需要编写的代码更少，在处理多维度大规模数据时快速且节省空间。
4. 越来越多基于Python的数学运算和科学计算软件包使用numpy数组参与计算过程。虽然这些工具通常支持Python列表作为参数，但在处理之前会将Python列表转换为numpy数组参与计算，通常输出结果也是numpy数组。



## 6.2 数组对象ndarray

Array object ndarray

### ● 6.2.1 ndarray对象的创建



ndarray对象是numpy模块的基础对象，是用于存放同类型元素的多维数组。可以使用整数索引获取数组中的元素，序号从0开始。在numpy中，ndarray对象的维度（dimensions）称为轴（axis），轴的个数叫做秩（rank）。一维数组的秩为1，二维数组的秩为2，以此类推。二维数组相当于两个一维数组，其中第一维度的每个元素又是一个一维数组。Python中关于ndarray对象的许多计算方法都是基于axis进行。当axis=0，表示沿着第0轴进行操作，即对每一列进行操作；当axis=1，表示沿着第1轴进行操作，即对每一行进行操作。

## 6.2.1 ndarray对象的创建

### 1) 利用array函数创建ndarray对象

创建一个ndarray对象可以使用numpy的array函数，格式如下：

```
np.array(object, dtype = None, copy = True,
order = None, subok = False, ndmin = 0)
```

参数object表示数组或嵌套的数列；可选参数dtype表示数组元素的数据类型；可选参数copy指出对象是否需要复制；参数order描述创建数组的样式，C为行方向，F为列方向，默认值A表示任意方向；参数subok默认返回一个与基类类型一致的数组；参数ndmin指定生成数组的最小维度。

```
In[ ]:import numpy as np
      np.array([1,2,3,4,5]) #将列表转换为数组
out[:]:array([1, 2, 3, 4, 5])
In[ ]:np.array((1,2,3,4,5)) #将元组转换为数组
out[:]:array([1, 2, 3, 4, 5])
In[ ]:np.array(range(5)) #将range对象转换为数组
out[:]:array([0, 1, 2, 3, 4])
In[ ]:l = [[1., 2., 3.], [4., 5., 6.]] # 二维数组
      np.array(l)
out[:]:array([[1., 2., 3.],
              [4., 5., 6.]])
In[ ]:l = [[[1,2],[3,4]],[[5,6],[7,8]]] # 三维数组
      np.array(l)
out[:]:array([[[1, 2],
               [3, 4]],

              [[5, 6],
               [7, 8]]])
```

## 6.2.1 ndarray对象的创建

### ● 2) np.ones()和np.zeros()函数

np.ones((m,n),dtype) 函数  
用于创建一个m行n列的全1数组，  
其中参数dtype指定数组类型。

```
In[ ]:np.zeros((3, 4))
out[]:array([[0., 0., 0., 0.],
             [0., 0., 0., 0.],
             [0., 0., 0., 0.]])
```

np.zeros((m,n),dtype)函数用于创建一个指定m行n列的全0数组。

```
In[ ]:np.ones((2, 3),dtype= np.int)
out[]:array([[1, 1, 1],
             [1, 1, 1]])
```

需要注意的是，np.ones()和np.zeros()函数中第一个参数是元组类型，用来指定数组的大小，如np.zeros((3, 4))表示创建3行4列的全0数组，np. ones((2, 3))中(2,3)表示创建2行3列的全1数组。





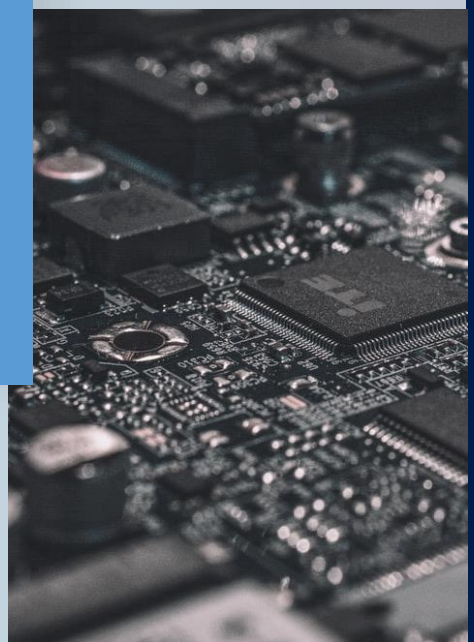
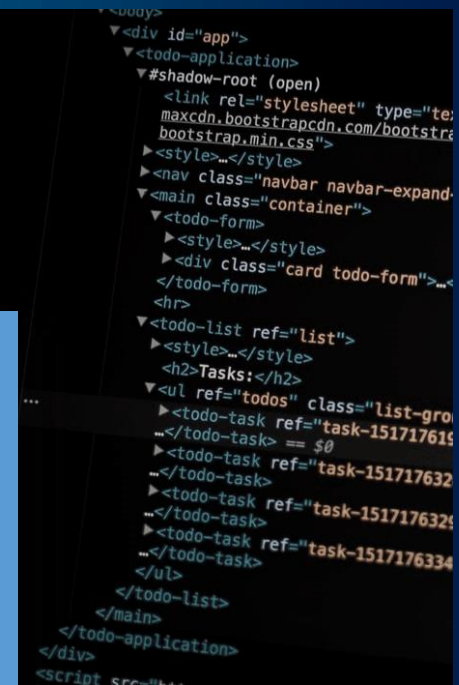
## 6.2.1 ndarray对象的创建

### ● 3) np.random.rand()函数

np.random.rand()  
函数创建一个指定形状  
的随机数组，数组元素  
是服从“0~1”均匀分  
布的随机样本，取值范  
围是[0,1)，不包括1。

```
In[ ]:np.random.rand(3)
out[:array([0.61680187, 0.31953998,
0.75485274])

In[ ]:np.random.rand(3, 4)
out[:array([[0.02201144, 0.90251899,
0.38872952, 0.92436457],
[0.78491163, 0.70545066, 0.70202207,
0.66271789],
[0.70565713, 0.49143328, 0.20526337,
0.96837098]])
```



## 6.2.1 ndarray对象的创建

### ● 4) np.arange()函数

np.arange()函数  
类似Python自带的  
range函数，用于创建  
一个等差序列的  
ndarray数组。

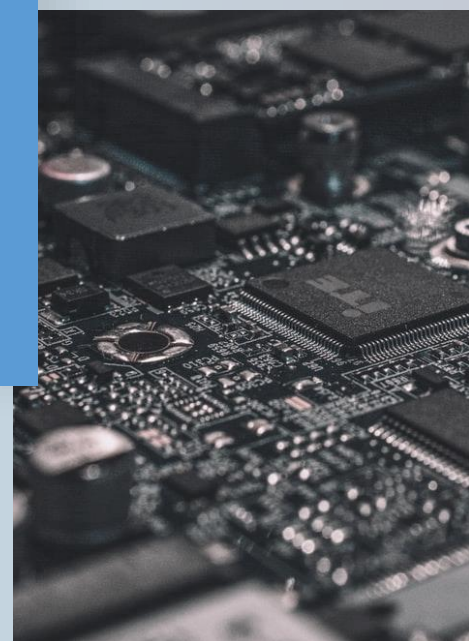
```
In[ ]: np.arange(10)
```

```
out[: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In[ ]: np.arange(10,20,2)
```

```
out[: array([10, 12, 14, 16, 18])
```

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
              <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717619">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-1517176329">...</todo-task>
                <todo-task ref="task-1517176334">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```



## 6.2.1 ndarray对象的创建

### ● 5) np.linspace()函数

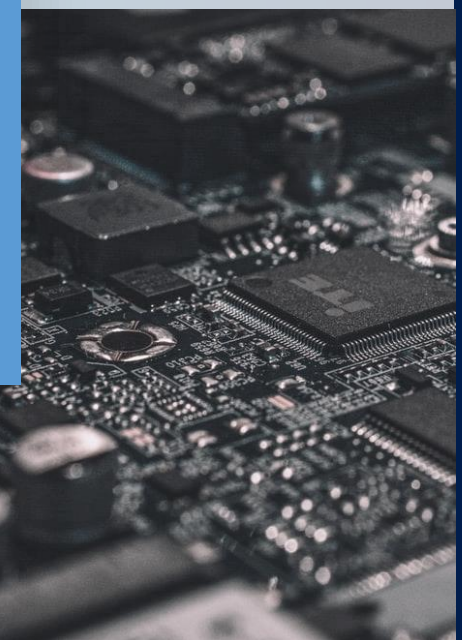
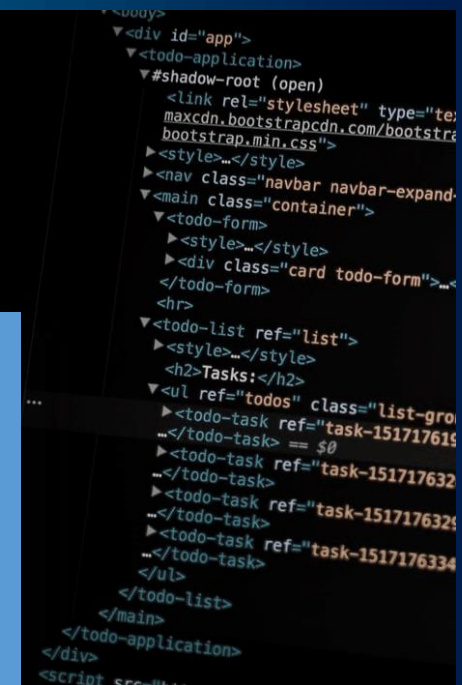
`np.linspace(x,y,n)`  
函数用于创建间隔均匀的数值序列，生成一个以x为起始点，以y为终止点，等分成n个元素的等差数组。

```
In[ ]: np.linspace(1,10,5)      #等差数组，包含5个数
```

```
out[:array([ 1. ,  3.25,  5.5 ,  7.75, 10.
])
```

```
In[ ]: np.linspace(1,10,5,endpoint=False)  #不包含  
终点
```

```
out[:array([1. , 2.8, 4.6, 6.4, 8.2])
```





## 6.2.1 ndarray对象的创建

### ● 6) np.empty()函数

`np.empty((m,n),dtype)` 函数创建一个m行n列的数组，参数dtype指定数组元素的数据类型。此方法生成的数组元素不一定为空，而是随机产生的数据。

```
In[ ]: np.empty((3,4))
      # 返回指定维度的数组，元素值是接近0的随机数
out[:]: array([[6.92145454e-310, 4.66495625e-310,
2.10077583e-312, 6.79038654e-313],
[2.22809558e-312, 2.14321575e-312,
2.35541533e-312, 6.79038654e-313],
[2.22809558e-312, 2.14321575e-312,
2.46151512e-312, 2.41907520e-312]])

In[ ]: np.empty((6,),dtype=list)
      #指定数据类型为list对象，创建空数组
out[:]: array([None, None, None, None, None, None],
dtype=object)
```

`np.empty()`函数默认数据类型为`numpy.float64`。若没有指定数据类型，该方法返回的数据类型为`numpy.float64`，这时生成的数组元素不可能为空。



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.2 ndarray对象常用属性

ndarray对象常用属性ndim返回正整数表示的数组维度个数，即数组的秩；shape属性返回数组维度，返回值为N个正整数组成的元组类型，元组的每个元素对应各维度的大小，元组的长度是秩，即维度个数或者ndim属性；dtype属性返回数组元素的数据类型，每个ndarray对象只有一种dtype类型；size属性返回数组元素总个数，返回值为shape属性中元组元素的乘积。

## 6.2 数组对象ndarray

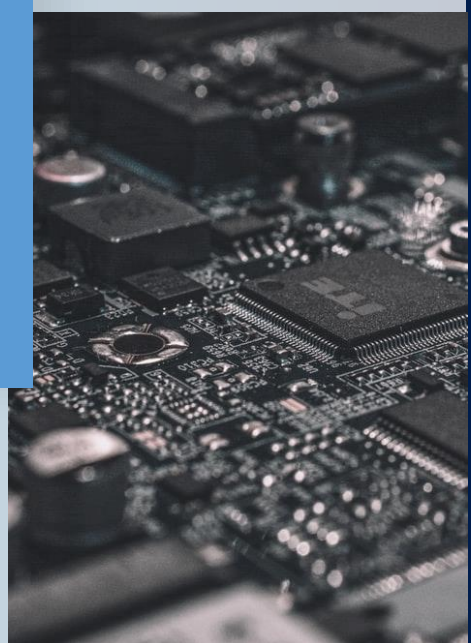
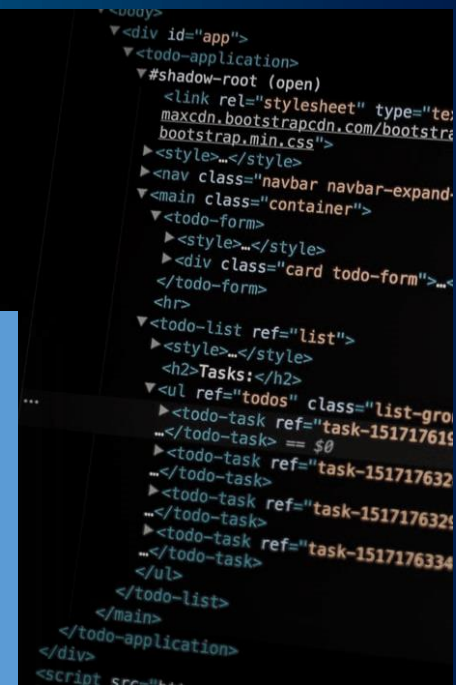
Array object ndarray

### 6.2.2 ndarray对象常用属性

```
In[ ]:1 = [1, 2, 3, 4, 5, 6] # 一维数组
      data = np.array(1)
      print(data)
      print('维度个数', data.ndim)
      print('各维度大小:', data.shape)
      print('数据类型:', data.dtype)
      print('数组元素总个数:', data.size)
```

```
out[:]:[1 2 3 4 5 6]
      维度个数 1
      各维度大小: (6,)
      数据类型: int64
      数组元素总个数: 6
```

通过序列型对象创建 ndarray 数组 data，ndim 属性返回整数 “1” 表示这是一维数组，shape 属性返回数组的维度个数和各维度元素的数量，即只有一维，该维度元素数量为 “6”，dtype 属性表明 data 数组元素的类型为 “int64”，size 属性表明 data 数组各维度元素总数为 “6”。





## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

#### 01 改变数组形状

numpy模块在修改数组形状时，可以使用`reshape()`函数，语法格式如下：

```
np.reshape(arr, newshape, order='C')
```

其中参数`arr`表示需要修改形状的数组；参数`newshape`为整数或由整数元素构成的元组，表示修改后的数组形状，要求新数组的形状应当与数组元素数量及形状兼容，否则抛出异常；参数`order`默认取值为“C”，表示按列读取数组元素，若`order='F'`，表示按行读取数据，若`order='A'`，表示按原顺序读取数据。

```
In[ ]: np.reshape(np.array(range(10)), (2,5), order="F")
out[]: array([[0, 4, 8, 3, 7],
              [2, 6, 1, 5, 9]])
```

```
In[ ]: np.reshape(np.array(range(10)), 10, order="C")
out[]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In[ ]: np.reshape(np.array(range(10)), (2,5), order="A")
out[]: array([[0, 1, 2, 3, 4],
              [5, 6, 7, 8, 9]])
```

## 6.2 数组对象ndarray

Array object ndarray

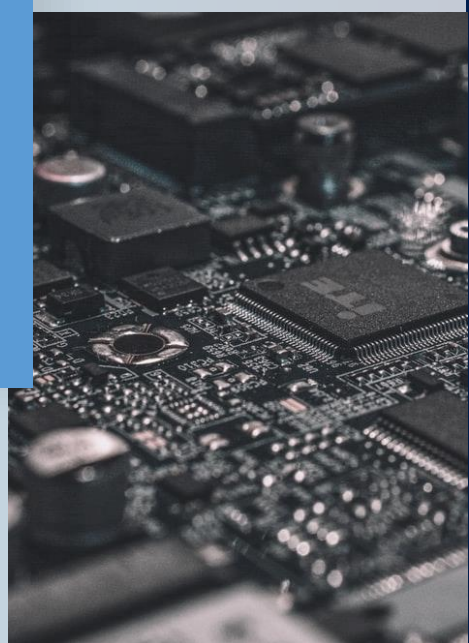
### 6.2.3 ndarray对象常用操作

- 1) 改变数组形状

ndarray数组作为numpy对象，可以调用ndarray.reshape(n, m)函数改变numpy数组的形状。在不改变数据的情况下，返回一个维度为(n,m)的新数组。

```
In[ ]: arr2=np.array(range(10))
        print(arr2)
out[:][0 1 2 3 4 5 6 7 8 9]
In[ ]: arr2 = arr2.reshape((5, 2))
        print(arr2)
out[:][[0 1]
        [2 3]
        [4 5]
        [6 7]
        [8 9]]
```

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717634">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```



## 6.2 数组对象ndarray

Array object ndarray

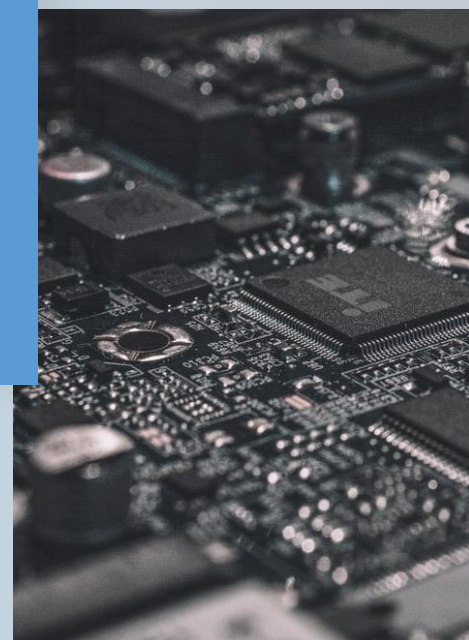
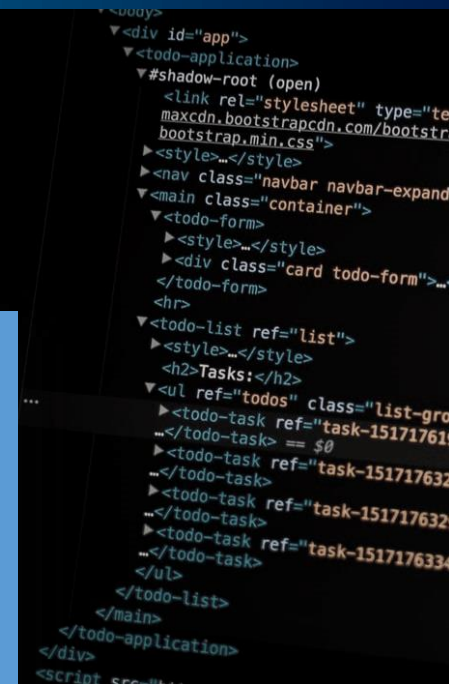
### 6.2.3 ndarray对象常用操作

- 1) 改变数组形状

np.resize()函数  
对数组形状进行原地  
修改，并且根据需要  
补充或丢弃部分元素。

```
In[ ]:np.resize(arr2,(1,15))
out[:array([[0, 1, 2, 3, 4, 5, 6, 7, 8,
9, 0, 1, 2, 3, 4]])

In[ ]:np.resize(arr2,(3,4))
out[:array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 0, 1]])
```





## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

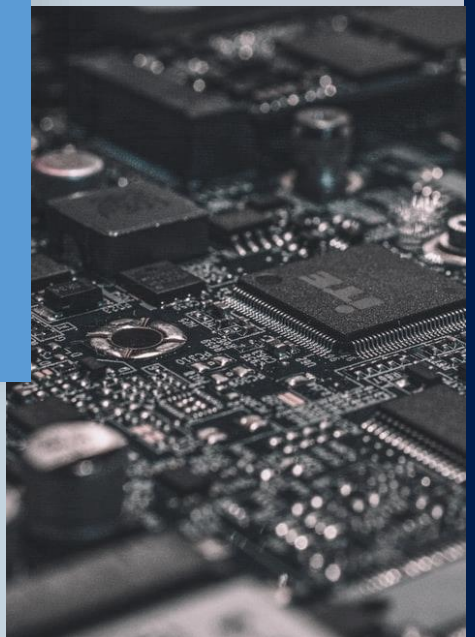
- 1) 改变数组形状

此外，还可以使用数组的shape属性直接原地修改数组大小。

```
In[ ]:arr2.size
out[:]:10

In[ ]:arr2.shape=2,5
out[:]:array([[0, 1, 2, 3, 4],
               [5, 6, 7, 8, 9]])
```

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
              <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717634">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

#### 02 改变数组元素类型

`astype()`方法用于改变数组元素的数据类型。  
在使用`astype`的时候，即使指定的数据类型与原始数组相同，系统也会创建一个新数组。

```
In[ ]: print(arr2.dtype)
out[]: int64

In[ ]: arr3 = arr2.astype(float)
       print(arr3.dtype)

out[]: float64

In[ ]: arr2.astype(dtype = np.float16)
out[]: array([[0., 1.],
              [2., 3.],
              [4., 5.],
              [6., 7.],
              [8., 9.]], dtype=float16)
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

```
In[ ]: arr3 = np.array([[1,2], [3,4],[5,6]])  
      arr3.flatten() #默认按行方向降维  
out[:array([1, 2, 3, 4, 5, 6])  
In[ ]: arr3.flatten('F') #按列方向降维  
out[:array([1, 3, 5, 2, 4, 6])
```

03

### 数组降维

`ndarray.flatten()`方法用于数组降维操作，将二维或者三维数组快速扁平化，返回一个一维数组。默认情况下按照行方向降维。

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

04

#### 数组转置

`np.transpose()`函数用于调换数组的索引值，对于二维数组，相当于求数组的转置对象。

```
In[ ]:data = np.arange(15).reshape(3, 5)
```

```
print(data)
```

```
out[]: [[ 0  1  2  3  4]
        [ 5  6  7  8  9]
        [10 11 12 13 14]]
```

```
In[ ]:print(np.transpose(data))
```

```
out[]: [[ 0  5 10]
        [ 1  6 11]
        [ 2  7 12]
        [ 3  8 13]
        [ 4  9 14]]
```



## 6.2 数组对象ndarray

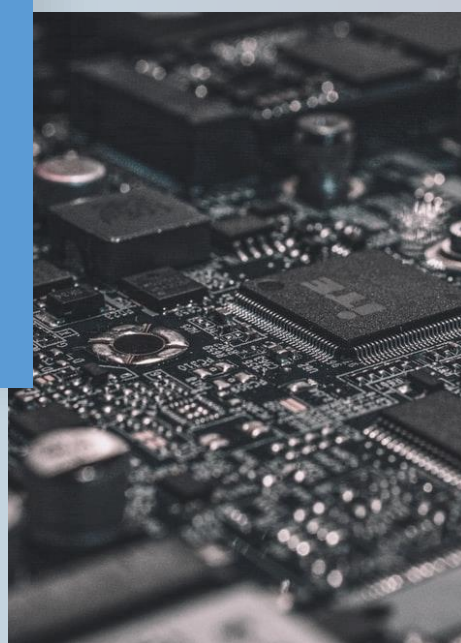
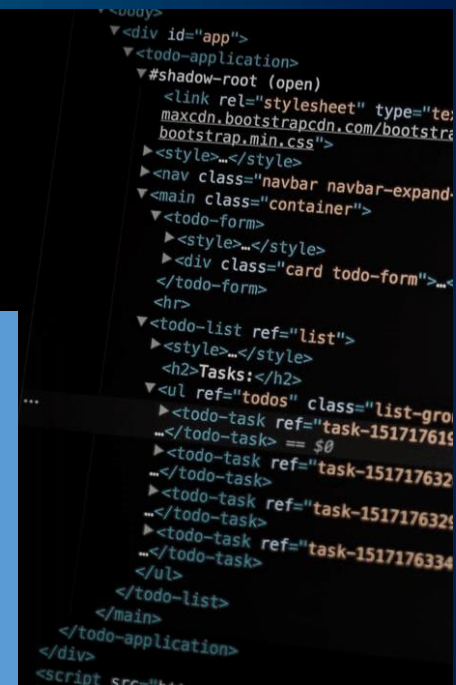
Array object ndarray

### 6.2.3 ndarray对象常用操作

- 4) 数组转置

ndarray.T属性可以实现线性代数中矩阵转置的功能。

```
In[ ]: print(data.T)
out[]: [[ 0  5 10]
         [ 1  6 11]
         [ 2  7 12]
         [ 3  8 13]
         [ 4  9 14]]
```



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

- 4) 数组转置

Numpy的函数

`np.swapaxes(a,x,y)`等价于  
ndarray对象的`a.swapaxes(x,y)`  
方法，用于将n维数组中x、y两个  
维度的数据调换。以维度为(2,3,4)  
的数组a为例，a是三维数组，第0  
维尺寸为2，第1维尺寸为3，第2  
维尺寸为4。使用`a.swapaxes(2,0)`  
方法将数组a第0维与第2维交换，  
返回第0维尺寸为4，第2维尺寸  
为2；第1维尺寸保持不变。

```
In[ ]: print(np.swapaxes(data, 0, 1))
out[]: [[ 0  5 10]
        [ 1  6 11]
        [ 2  7 12]
        [ 3  8 13]
        [ 4  9 14]]

In[ ]: a = np.arange(24).reshape(2,3,4)
        print(a)
out[]: [[[ 0  1  2  3]
        [ 4  5  6  7]
        [ 8  9 10 11]]

        [[12 13 14 15]
        [16 17 18 19]
        [20 21 22 23]]]
```

```
In[ ]: a.swapaxes(2, 0)
out[]: array([[[ 0, 12],
               [ 4, 16],
               [ 8, 20]],

              [[ 1, 13],
               [ 5, 17],
               [ 9, 21]],

              [[ 2, 14],
               [ 6, 18],
               [10, 22]],

              [[ 3, 15],
               [ 7, 19],
               [11, 23]])])
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

05

#### 数组合并

数组合并用于多个数组间的操作，numpy的hstack()和vstack()函数分别用于沿着水平和垂直方向将多个数组合并在一起。

水平方向的数据合并操作将ndarray对象构成的元组作为参数，传递给hstack()函数。

```
In[ ]:arr1=np.array([1,2,3])
      arr2=np.array([4,5,6])
      arr1
out[:array([1, 2, 3])
In[ ]:np.hstack((arr1,arr2))
out[:array([1, 2, 3, 4, 5, 6])
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

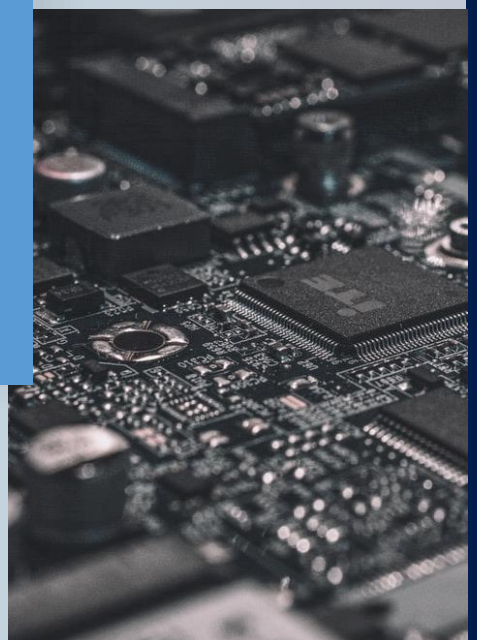
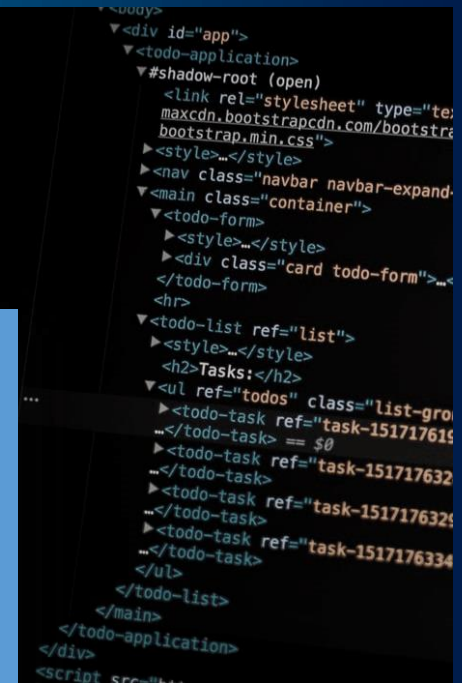
- 5) 数组合并

```
In[ ]: arr3=np.array([[1],[2],[3]])
      arr4=np.array([[4],[5],[6]])
      arr3

out[:]: array([[1],
               [2],
               [3]])

In[ ]: np.vstack((arr3,arr4))
out[:]: array([[1],
               [2],
               [3],
               [4],
               [5],
               [6]])
```

垂直方向的数据合并  
操作将ndarray对象构成的元组作为参数，传递给  
vstack()函数。





## 6.2 数组对象ndarray

Array object ndarray

### 6.2.3 ndarray对象常用操作

- 5) 数组合并

numpy中重要的  
concatenate()函数提供了类似的数据合并功能，其中参数axis指定数据合并的方向或维度。参数axis默认值为0，表示按照垂直方向进行数据合并；参数axis=1表示按照水平方向进行数据合并。

```
In[ ]: np.concatenate((arr1,arr2))
```

```
out[]:array([1, 2, 3, 4, 5, 6])
```

```
In[ ]: np.concatenate((arr3,arr4))
```

```
out[]:array([[1],
```

```
[2],
```

```
[3],
```

```
[4],
```

```
[5],
```

```
[6]])
```

```
In[ ]: np.concatenate((arr3,arr4),axis=1)
```

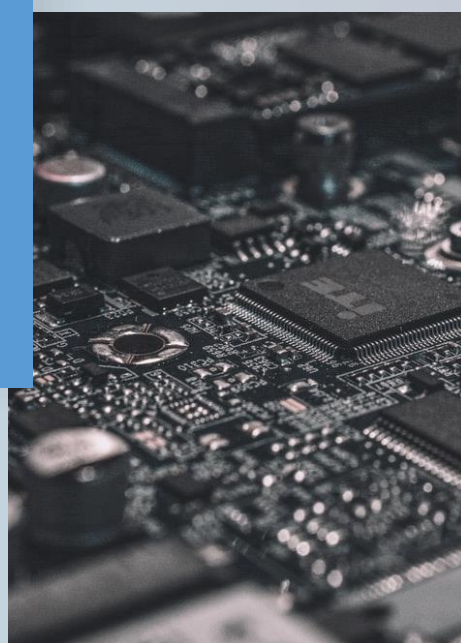
```
out[]:array([[1, 4],
```

```
[2, 5],
```

```
[3, 6]])
```

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
              <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717633">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </div>
      <script src="...">

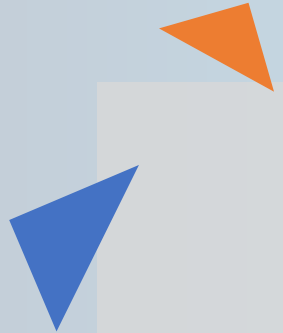
```



## 6.2 数组对象ndarray

Array object ndarray

### ● 6.2.4索引和切片



ndarray对象的索引通常用于获取数组元素，ndarray对象的切片通常获取数组中多个元素组成的数据片段。ndarray对象的基本索引和切片得到的结果都是原始数组的视图，修改视图也会修改原始数组。

## 6.2 数组对象ndarray

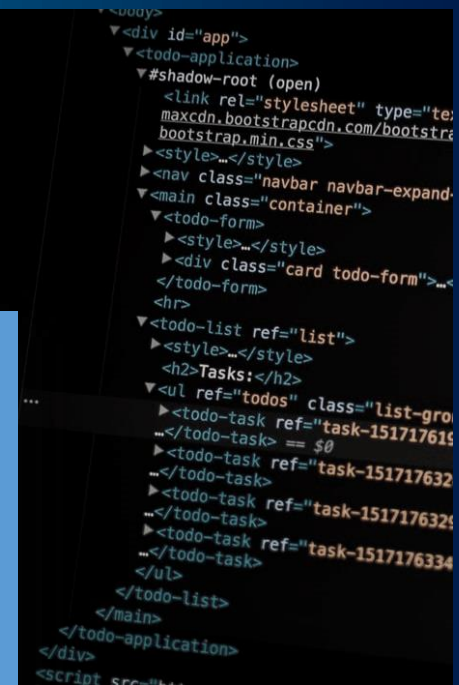
Array object ndarray

### 6.2.4 索引和切片

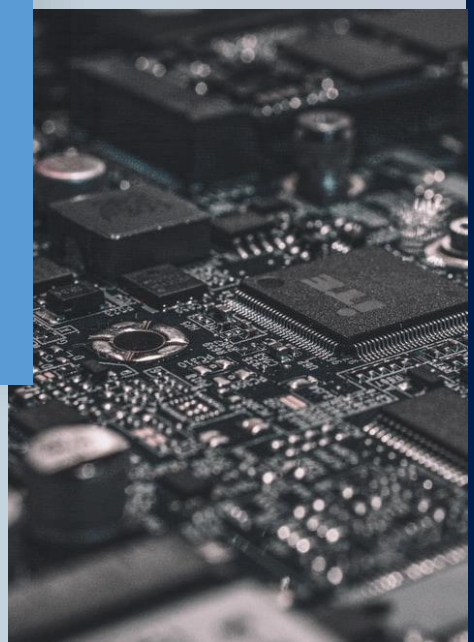
- 1) 一维数组的索引

ndarray对象的一维数组和python列表结构类似，以单个索引值的方式访问ndarray对象的一维数组会返回对应的标量元素值。

```
In[ ]: data = np.array([1, 2, 3, 4, 5, 6])
      print(data)
out[ ]: [1 2 3 4 5 6]
In[ ]: print(data[0]) # 一维数组索引
      print(data[-1])
      print(data[3:]) # 一维数组切片
out[ ]: 1
      6
      [4 5 6]
```



```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717619">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-1517176329">...</todo-task>
                <todo-task ref="task-1517176334">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```



## 6.2.4索引和切片

### • 2) 多维数组的索引

ndarray对象的二维数组可以看作一维数组的嵌套形式，当ndarray对象以一维数组的索引方式访问一个二维数组时，获取的元素就是一个一维数组，然后可以继续按照访问一维数组的方式获取二维数组中的某个标量。

```
In[ ]:arr = np.arange(9).reshape(3, 3) #多维数组
      print(arr) #获取所有元素

out[ ]:[[0 1 2]
        [3 4 5]
        [6 7 8]]

In[ ]: print(arr[2]) #多维数组索引，获取一行元素

out[ ]:[7 8 9]

In[ ]: print(arr[2][0]) #多维数组索引，获取一个元素，

out[ ]:7

In[ ]: print(arr[2, 0]) #多维数组索引，获取第2行第0列的元素

out[ ]:7

In[ ]: print(data[:2, 1:])

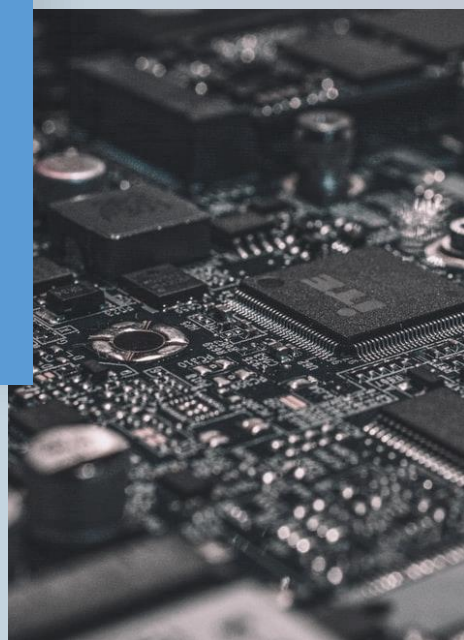
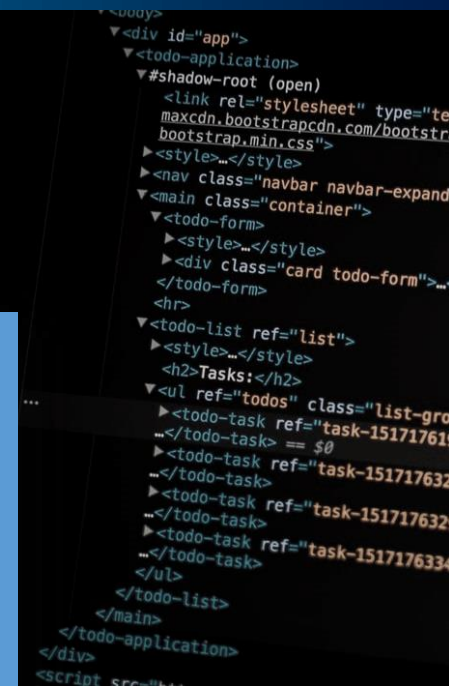
out[ ]: [[1 2]
        [4 5]]

In[ ]: print(data[2, :])
      print(data[2:, :])

out[ ]:[6 7 8]
        [[6 7 8]]

In[ ]: print(data[:, :2])

out[ ]: [[0 1]
        [3 4]
        [6 7]]
```





## 6.2 数组对象ndarray

Array object ndarray

### 6.2.4 索引和切片

#### • 3) 布尔值索引

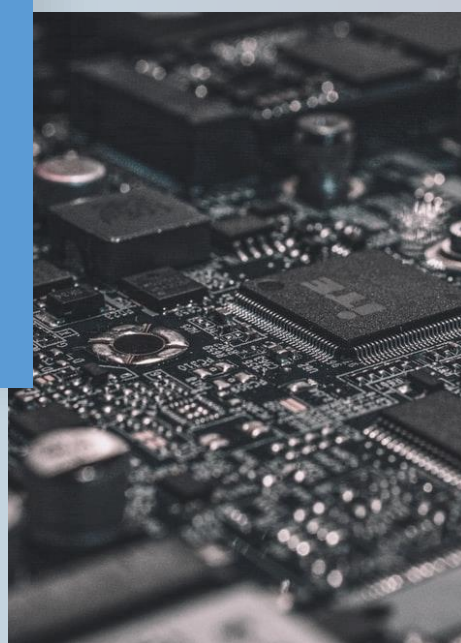
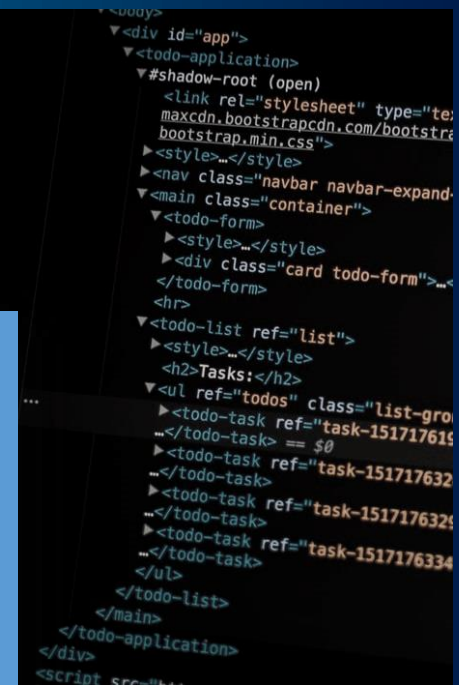
ndarray对象的布尔值索引又称条件索引，是指一个由布尔值组成的数组可以作为另一个数组的索引，返回的数据为新数组中True值对应位置的元素。通过布尔运算（如：比较运算符）来获取符合指定条件的元素组成的数组。

```
In[ ]:is_gt = data > 4 # 条件索引
      print(is_gt)
out[]:[[False False False]
       [False False  True]
       [ True  True  True]]

In[ ]:print(data[is_gt]) # 找出大于4的数据
out[]: [5 6 7 8]

In[ ]:print(data[data > 4]) # 简写
out[]: [5 6 7 8]

In[ ]:print(data[(data > 4) & (data % 2 == 0)])
      # 找出数据中大于4的偶数
out[]: [6 8]
```



# 6.2 数组对象ndarray

Array object ndarray

## 6.2.5 numpy常用函数

### 01 numpy常用统计函数

numpy模块中常用统计函数如表6-1所示。几乎所有numpy统计函数在关于二维数组的运算时都需要注意参数axis的取值。对二维数组而言，如果未设置参数axis，那么针对所有元素进行操作；如果axis=0，表示按照列或垂直方向，即沿着纵轴进行操作；如果axis=1，表示按照行或水平方向，即沿着横轴进行操作。

函数	描述	函数	描述
np.mean()	均值	np.argmax()	最大值的索引
np.sum()	求和	np.argmin()	最小值的索引
np.max()	最大值	np.argsort()	排序后的索引
np.min()	最小值	np.cumsum()	累加
np.std()	标准差	np.cumprod()	累乘
np.median()	中位数	np.average()	加权平均数

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.5 numpy常用函数

- 1) numpy常用统计函数

```
In[ ]: data = np.arange(10).reshape(5, 2)
      print(data)
out[]: [[0 1]
        [2 3]
        [4 5]
        [6 7]
        [8 9]]
```

```
In[ ]: print(np.mean(data)) #返回数组元素的算术平均值
      print(np.mean(data, axis=0))
      # axis=0按列操作, 返回每一列的平均值
      print(np.mean(data, axis=1))
      # axis=1按行操作, 返回每一行的平均值
out[]: 4.5
      [4. 5.]
      [0.5 2.5 4.5 6.5 8.5]
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.5 numpy常用函数

- 1) numpy常用统计函数

```
In[ ]:print(np.sum(data))
      #将数组平铺成为一维向量后求所有元素的和
      print(np.sum(data, axis=0))
      # axis=0表示按列操作, 返回每一列的和
      print(np.sum(data, axis=1))
      # axis=1表示按行操作, 返回每一行的和

out[:]:45
      [20 25]
      [ 1  5  9 13 17]
```

```
In[ ]:print("中位数:",np.median(data))  #返回中位数
      print("最小值的索引:",np.argmin(data))
      #返回最小值的索引
      print("最大值的索引:",np.argmax(data))
      #返回最大值的索引
      print("排序后的索引:",np.argsort(data))
      #排序后的索引

out[:]:中位数: 4.5
      最小值的索引: 0
      最大值的索引: 9
      排序后的索引: [[0 1]
                    [0 1]
                    [0 1]
                    [0 1]
                    [0 1]]
```



## 6.2.5 numpy常用函数

- 1) numpy常用统计函数

`np.average()`函数根据另一个数组指定的权重计算数组元素的加权平均值。所谓加权平均值，是指将各数值乘以相应的权重，然后相加求和得到总体值，再除以总的单位数。该函数可以接受一个轴参数`axis`。如果没有指定轴，则数组将被展开。

```
In[ ]: data=np.swapaxes(data, 0, 1)
        print(data)
        print(np.argsort(data,axis = 0)) #按列从小到大排序，输出对应索引
        print(np.argsort(data,axis = 1)) #按行从小到大排序，输出对应索引
out[]: [[0 2 4 6 8]
        [1 3 5 7 9]]
        [[0 0 0 0 0]
        [1 1 1 1 1]]
        [[0 1 2 3 4]
        [0 1 2 3 4]]
In[ ]: print("加权平均数(无权重参数): ",np.average(data))
        print("加权平均数: ", np.average([1,2,3,4],weights = [4,3,2,1]))
out[]: 加权平均数(无权重参数): 4.5
        加权平均数: 2.0
```

这里`np.average()`函数将数值数组`[1,2,3,4]`和权重数组`[4,3,2,1]`，通过对应元素相乘后相加，并将相加之和除以权重之和，来计算加权平均值。

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.5 numpy常用函数

#### 02 np.all()、np.any()和np.unique()函数

01

np.all()

np.all()函数对所有元素进行与操作，用于判断是否所有元素都满足条件。只有所有元素取值为True，该函数的返回结果才为True。

```
In[ ]: print(np.all(data > 5))  
out[]: False
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.5 numpy常用函数

#### 02 np.all()、np.any()和np.unique()函数

`np.any()`函数对所有元素进行或操作，用于判断是否至少一个元素满足条件。只要任意一个元素取值为`True`，该函数的返回结果即为`True`。

```
In[ ]:print(np.any(data > 5))  
out[]:True
```

02

np.any()



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.5 numpy常用函数

#### 02 np.all()、np.any()和np.unique()函数

03

np.unique()

`np.unique()`函数返回数组中所有不同的值，并按照从小到大排序。该函数用于去除数组中的重复数据，并返回排序后的结果。

```
In[ ]:arr = np.array([[1, 2, 1], [2, 3, 4]])
        print(arr)
out[]: [[1 2 1]
        [2 3 4]]

In[ ]:print(np.unique(arr))
out[]: [1 2 3 4]
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.5 numpy常用函数

#### 02 np.all()、np.any()和np.unique()函数

np.unique()函数的可选参数  
return\_index=True时，该函数返回新数组中每个元素在原数组中第一次出现的索引值，因此元素个数与新数组中元素个数一样。可选参数  
return\_inverse=True时，该函数返回原数组中每个元素在新数组中出现的索引值，因此元素个数与原数组中元素个数一样。

```
In[ ]:A = np.array([1, 2, 5, 3, 4, 3])
      print ("原数组: ", A)
      a, s, p = np.unique(A,
return_index=True,return_inverse=True)
      print ("新数组: ",a)
      print ("return_index", s)
      print ("return_inverse", p)

out[]:原数组:  [1, 2, 5, 3, 4, 3, 2]
      新数组:  [1 2 3 4 5]
      return_index [0 1 3 4 2]
      return_inverse [0 1 4 2 3 2 1]
```





## 6.2 数组对象ndarray

Array object ndarray

### ● 6.2.6 numpy数组运算

- 1) numpy数组的向量化



numpy数组可以使用简单的数组表达式完成多种数据操作任务，而无需使用大量循环语句。这种使用数组表达式替代循环操作的方法，称为向量化。通过向量化操作，可以一次性地在一个复杂对象上操作，或者将函数应用于一个复杂对象，避免了在对象的单个元素上使用复杂循环语句完成操作任务，达到代码更紧凑、执行速度更快的代码实现效果。

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.6 numpy数组运算

- 1) numpy数组的向量化

实际上，numpy中关于ndarray对象的循环操作是通过高效优化的C代码实现，执行速度远快于纯python。通常，向量化的数组操作比纯python的等价实现在速度上至少快1-2个数量级。这为实现高效的数值计算奠定了基础。

从示例中看到，与纯python的循环语句实现相比，使用numpy数据的向量化操作完成同样的功能，运算效率提高了接近10倍。

```
In[ ]:import numpy as np
import time
a = np.arange(100000, dtype=float)
b = np.arange(100000, 0, -1, dtype=float)
In[ ]:begin = time.time() # 未使用向量化
results = []
for i, j in zip(a, b):
    results.append(i * j)
end = time.time()
print('运行时间: ', end - begin)
out[ ]:运行时间: 0.02645111083984375
In[ ]:begin = time.time() # 使用向量化
results = a * b
end = time.time()
print('运行时间: ', end - begin)
out[ ]:运行时间: 0.0027115345001220703
```

## 6.2 数组对象ndarray

Array object ndarray

### 6.2.6 numpy数组运算

- 2) numpy广播机制

numpy广播机制是实现向量化计算的有效方式。当需要处理的数组维度大小不一致时，numpy广播机制提供了不同形状的数组仍然能够计算的实现机制。

作为多维向量的组合，数组（或者称向量）计算大多在相同形状的数组之间进行，要求被处理的数组维度以及每个维度大小是相等的，这时的数组操作应用在元素上，即数组元素一一对应的操作。但是，许多计算可能涉及一个维度与其他所有维度之间的操作，此时被操作的数组大小不同。当两个形状并不相同的数组参与运算时，可以使用扩展数组的方式实现相加、相减、相乘等操作，这种机制称为广播（broadcasting）。numpy采用广播机制来实现不同形状的数组间运算。

## 6.2 数组对象ndarray

Array object ndarray

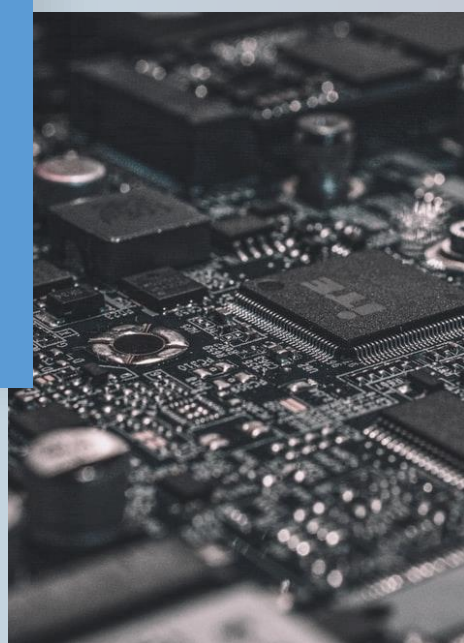
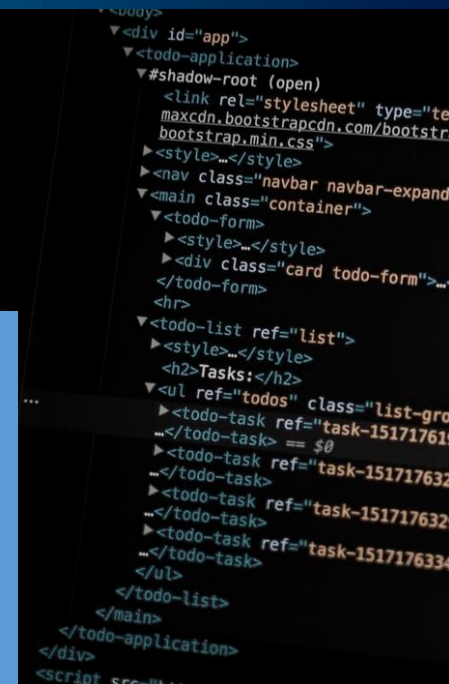
### 6.2.6 numpy数组运算

- 2) numpy广播机制

两个numpy数组的相加、相减以及相乘都是对应元素的操作。如果两个数组x和y形状相同，即满足  $a.shape == b.shape$ ，那么 $x*y$ 的运算结果是数组x与y数组对应元素相乘。这里要求两个数组维数相同，且各维度的长度相同。

```
In[ ]: x = np.array([[2,2,3],[1,2,3]])
      y = np.array([[1,1,3],[2,2,4]])
      print(x*y)  #numpy数组相乘是对应元素的
                  乘积，与线性代数中的矩阵相乘不一样

out[]: [[ 2  2  9]
        [ 2  4 12]]
```



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.6 numpy数组运算

- 2) numpy广播机制

当参与运算的两个数组形状不一致时，如果数组形状符合广播机制的要求，numpy将自动触发广播机制。通俗地说，首先将两个数组的维度大小右对齐，然后比较对应维度上的数值，如果数值相等或其中有一个为1或者为空，那么能够进行广播运算，并且输出维度的大小取数值大的维度值。否则不能进行数组运算。

```
In[ ]:np.arange(6).reshape((2, 3)).shape
out[]: (2, 3)

In[ ]:np.arange(10).reshape(2,5).shape
out[]: (2, 5)

In[ ]:np.arange(6).reshape((3,2))+np.arange(10).reshape(5,2)
out[]:-----ValueError      Traceback (most recent call last)
<ipython-input-16-b265dad701e5> in <module>
----> 1 np.arange(6).reshape((3,2))+np.arange(10).reshape
(5,2)
ValueError: operands could not be broadcast together with
shapes (3,2) (5,2)
```

```
<body>
<div id="app">
  <todo-application>
    <#shadow-root (open)>
      <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
      <style>...</style>
      <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
          </todo-form>
          <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717615">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717634">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </#shadow-root>
    </todo-application>
  </div>
</body>
```



## 6.2 数组对象ndarray

### 6.2.6 numpy数组运算

- 2) numpy广播机制

当参与运算的两个数组形状不一致时，如果数组形状符合广播机制的要求，numpy将自动触发广播机制。通俗地说，首先将两个数组的维度大小右对齐，然后比较对应维度上的数值，如果数值相等或其中有一个为1或者为空，那么能够进行广播运算，并且输出维度的大小取数值大的维度值。否则不能进行数组运算。

```
In[ ]: np.arange(5).shape
```

```
out[]: (5,)
```

```
In[ ]: np.arange(10).reshape(5,2).shape
```

```
out[]: (5, 2)
```

```
In[ ]: np.arange(5)+np.arange(10).reshape(5,2)
```

#一维数组长度不等于二维数组列数

```
out[]: -----ValueError Traceback (most  
recent call last)
```

```
<ipython -input-10-ccb772981d62> in <module>
```

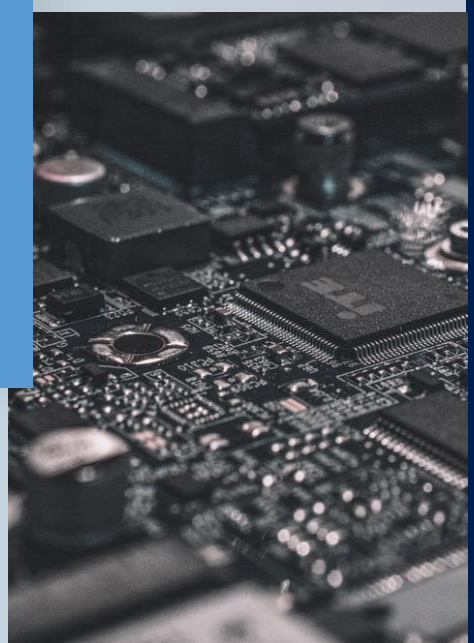
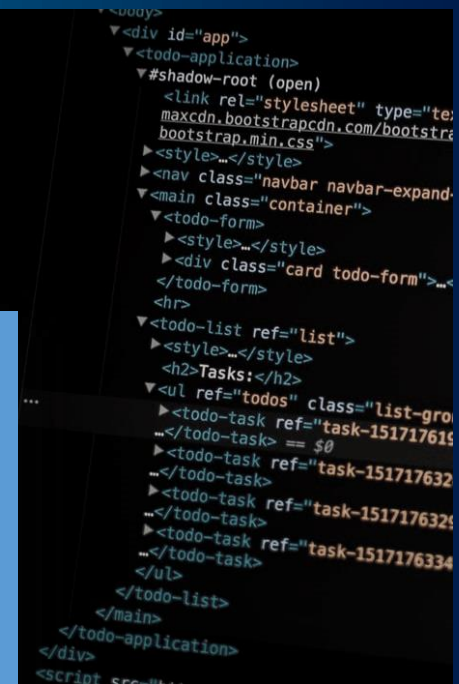
```
----> 1 np.arange(5) +
```

```
np.arange(10).reshape(5,2) #一维数组长度
```

不等于二维数组列

```
ValueError: operands could not be broadcast  
together with shapes
```

```
(5,) (5,2)
```



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.6 numpy数组运算

- 2) numpy广播机制

广播机制遵循的规则：第一，所有输入数组都向其中shape最长的数组看齐，shape中不足的部分通过在小维度数组的前面添加长度为1的轴补齐；第二，输出数组的shape是输入数组shape中各个轴上的最大值；第三，当输入数组的某个轴长度为1时，沿着此轴运算时都用该轴上的第一组值。

```
In[ ]:np.arange(3) + 5 #一维数组与数字的运算，向量与标量的运算
```

```
out[]: array([5, 6, 7])
```

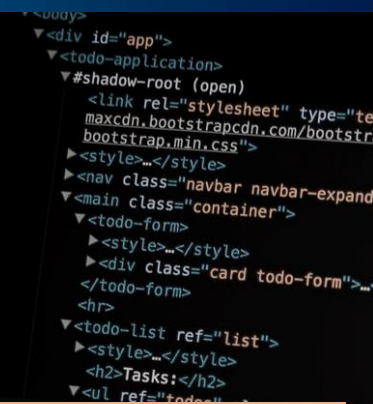
```
In[ ]:np.ones((3, 3)) + np.arange(3)
```

```
out[]:array([[1., 2., 3.],
             [1., 2., 3.],
             [1., 2., 3.]])
```

```
In[ ]:np.arange(3).reshape((3, 1)) + np.arange(3)
```

```
out[]:array([[0, 1, 2],
             [1, 2, 3],
             [2, 3, 4]])
```

广播机制将维度或形状比较小的数组扩展到更大的范围，以便两个参与操作的数组维度相同，各维度的长度相同，达到数组形状可兼容，然后实现两个相同形状数组对应元素之间的操作。



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.6 numpy数组运算

- 3) ufunc通用函数

ufunc ( universal function ) 函数意为“通用函数”，是一种能够对数组中每个元素进行操作的函数。ufunc函数对输入数组进行基于元素级别的运算，输出结果为numpy数组。numpy中许多ufunc内置函数的底层代码是基于C语言实现的，因此对一个数组进行重复运算时，使用ufunc函数的计算速度比使用math标准库函数要快很多。但是，在对单个数值进行运算时，python提供的运算要比numpy执行效率高。

## 6.2 数组对象ndarray

### 6.2.6 numpy数组运算

- 3) ufunc通用函数

这里仅介绍几个常用的ufunc函数：

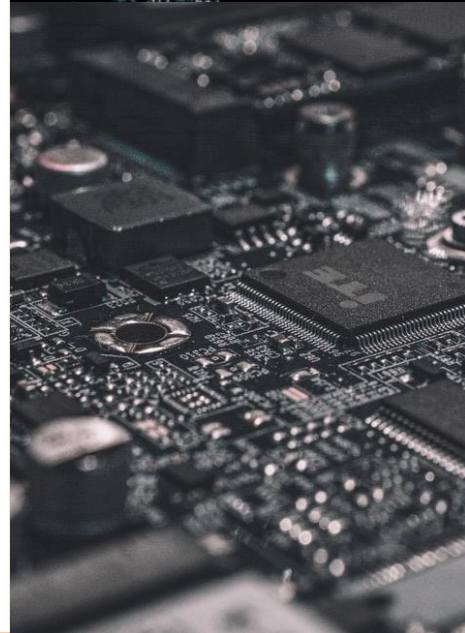
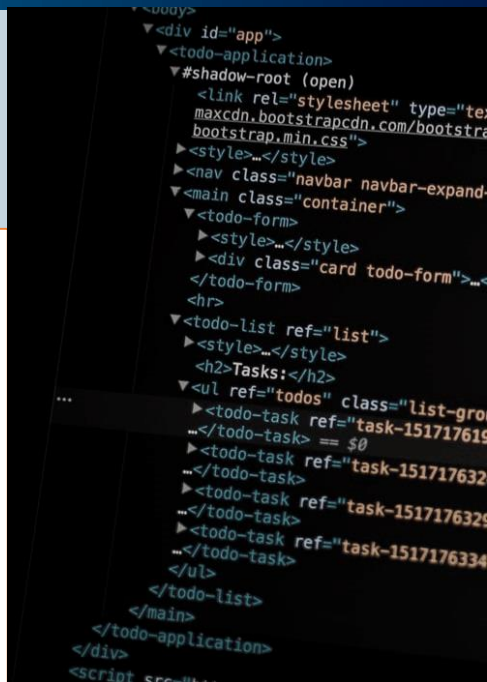
ceil()函数对数组元素向上取整，返回向上最接近该元素的整数；

floor()函数对数组元素向下取整，返回向下最接近该元素的整数；

rint()函数对数组元素进行四舍五入，返回最接近的整数；

isnan()函数用于判断数组元素是否为NaN(Not a Number)。

```
In[ ]: arr = np.random.randn(2,3)
        print(arr)
out[]: [[-1.14243468  0.62343218  1.10789425]
        [ 0.14386543  0.90263521  1.19952443]]
In[ ]: print(np.ceil(arr))
        print(np.floor(arr))
        print(np.rint(arr))
        print(np.isnan(arr))
out[]: [[-1.   1.   2.]
        [ 1.   1.   2.]]
        [[-2.   0.   1.]
        [ 0.   0.   1.]]
        [[-1.   1.   1.]
        [ 0.   1.   1.]]
        [[False False False]
        [False False False]]
```



## 6.2 数组对象ndarray

Array object ndarray

### 6.2.6 numpy数组运算

- 3) ufunc通用函数

multiply()函数用于元素相乘，等同于数组的“\*”操作，需要注意该操作与数学中的矩阵乘法不同；

divide()函数用于元素相除，等同于数组的“/”操作。

```
In[ ]:np.multiply(np.arange(3), 5)
```

```
out[:array([ 0,  5, 10])
```

```
In[ ]:np.arange(3) * 5
```

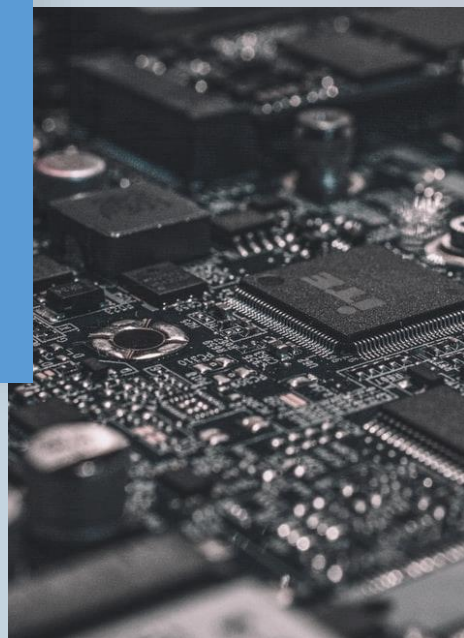
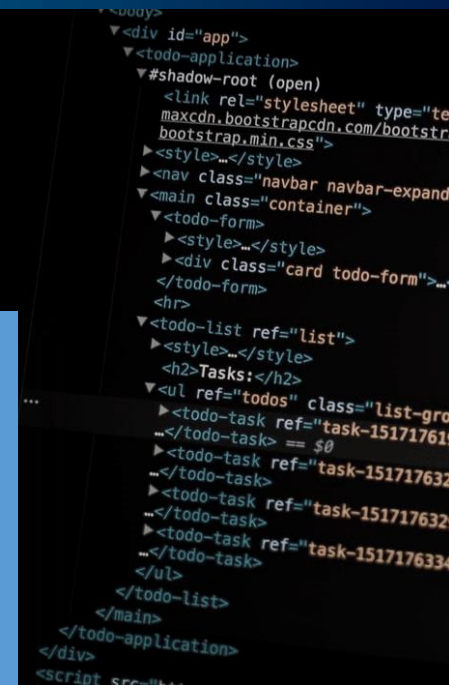
```
out[:array([ 0,  5, 10])
```

```
In[ ]:np.divide(np.arange(3), 5)
```

```
out[:array([0. , 0.2, 0.4])
```

```
In[ ]:np.arange(3) / 5
```

```
out[:array([0. , 0.2, 0.4])
```







# PART THREE

## 6.3 numpy矩阵

Numpy matrix

## 6.3 numpy矩阵

Numpy matrix

### ● numpy矩阵

numpy中包含一个矩阵库  
numpy.matlib，该模块的函数返回  
一个矩阵，而不是ndarray对象。一  
个m\*n的矩阵是一个由m行（row）n  
列（column）元素排列而成的矩形  
阵列。

```
In[ ]:import numpy as np
      np.array(range(6))
out[:array([0, 1, 2, 3, 4, 5])
In[ ]: type(np.array(range(6)))
out[:numpy.ndarray
In[ ]:np.matrix([1,2,3,4,5,6])
out[:matrix([[1, 2, 3, 4, 5, 6]])
In[ ]: type(np.matrix([1,2,3,4,5,6]))
out[:numpy.matrix
```

## 6.3 numpy矩阵

Numpy matrix

### 6.3.1 numpy矩阵简介

numpy函数库中的矩阵matrix和数组array都可以用于处理行列表示的数值型元素。它们在形式上很相似，同时二者存在着一定区别和联系。

- 1)数组array是numpy模块的基础，矩阵matrix可以看作数组的特殊形式；
- 2)矩阵是数学上的概念，而数组是一种数据存储方式；
- 3)矩阵matrix只能包含数字，而数组array可以是任意类型的数据；
- 4)矩阵matrix只能表示二维数据，而数组array可以表示任意维度数据，或者说matrix相当于二维数组。当matrix某维度为1时，如向量(m,1)可称为列向量，而向量(1,n)为行向量；
- 5)矩阵matrix的优势是使用相对简单的运算符号，如矩阵相乘用符号\*，但是数组array相乘使用dot()方法。array的优势是不仅仅表示二维，还能表示三维等更多维度的数据。

实际上，matrix是array的分支，matrix和array在很多时候都是通用的。如果二者可以通用的情况下，官方建议尽量选择array，因为array更灵活，速度更快，很多人也把二维的array翻译成矩阵。在实际应用中，使用numpy数组的情况更常见；但是当应用对矩阵运算有要求时，定义和使用numpy矩阵同样便捷。

## 6.3.2 矩阵生成

扩展库numpy中提供的matrix()函数可以把列表、元组、range对象等python可迭代对象转换为矩阵。

显然，matrix()函数与array()函数生成矩阵所需的数据格式存在差别。matrix()函数处理的数据可以是分号(;)分割的字符串，也可以是逗号(,)分割的列表类型，而array()函数处理的数据大多是逗号(,)分割的列表类型。

```
In[ ]:x=np.matrix([[1,2,3],[4,5,6]])
      y=np.matrix([1,2,3,4,5,6])
      print(x,y,sep='\n\n')
out[]: [[1 2 3]
        [4 5 6]]
        [[1 2 3 4 5 6]]

In[ ]:print(x[1,1]) #返回行下标和列下标都为1的元素
out[]:5
In[ ]:np.matrix(range(10))
out[]:matrix([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
In[ ]:np.matrix('1 3;5 7')
out[]:matrix([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
In[ ]:np.mat(np.eye(2,2,dtype=int)) #2*2对角矩阵
out[]:matrix([[1, 0],
              [0, 1]])

In[ ]:np.mat(np.diag([1,2,3])) #对角线为1,2,3的对角矩阵
out[]:matrix([[1, 0, 0],
              [0, 2, 0],
              [0, 0, 3]])

In[ ]:np.mat(np.random.randint(2,8,size=(2,5))) #元素为2-8的随机整数矩阵
out[]:matrix([[7, 3, 2, 6, 3],
              [3, 4, 6, 4, 6]])
```

### 6.3.3 矩阵特征

numpy扩展库中的max()、min()、sum()、mean()等方法均支持矩阵操作。在大多数矩阵方法中，可以使用参数axis指定计算方向。axis=1表示水平方向的计算；axis=0表示垂直方向的计算；如果不指定axis参数，则对矩阵平铺后的所有元素进行操作。

## 6.3 numpy矩阵

```
In[ ]:print('所有元素之和: ',x.sum(),)
      print('横向元素之和: ',x.sum(axis=1),end='\n===\n')
      print('横向最大值的下标: ',x.argmax(axis=1),end='\n===\n')
      print('对角线元素: ',x.diagonal(),end='\n===\n')
      print('非零元素行、列下标: ',x.nonzero())

out[ ]:所有元素之和:  21
      横向元素之和:  [[ 6]
      [15]]
      ===
      横向最大值的下标:  [[2]
      [2]]
      ===
      对角线元素:  [[1 5]]
      ===
      非零元素行、列下标:  (array([0, 0, 0, 1, 1, 1]), array([0, 1, 2, 0,
1, 2]))
```



### 6.3.3 矩阵特征

numpy扩展库中的max()、min()、sum()、mean()等方法均支持矩阵操作。在大多数矩阵方法中，可以使用参数axis指定计算方向。axis=1表示水平方向的计算；axis=0表示垂直方向的计算；如果不指定axis参数，则对矩阵平铺后的所有元素进行操作。

```
In[ ]: print(x,end='\n===\n')
        print('所有元素平均值:',x.mean(),end='\n===\n')
        print('垂直方向平均值:',x.mean(axis=0),end='\n===\n')
        print('形状',x.mean(axis=0).shape,end='\n===\n')
        print('水平方向平均值:',x.mean(axis=1),end='\n===\n')
        print('形状',x.mean(axis=1).shape)

out[]: [[1 2 3]
        [4 5 6]]
        ===
        所有元素平均值: 3.5
        ===
        垂直方向平均值: [[2.5 3.5 4.5]]
        形状 (1, 3)
        ===
        水平方向平均值: [[2.]
                           [5.]]
        形状 (2, 1)
```

## 6.3 numpy矩阵

Array object ndarray

### 6.3.4 矩阵常用操作

#### 01 矩阵转置

矩阵转置是对矩阵的行和列互换得到新矩阵的操作。原矩阵的第*i*行变为新矩阵的第*i*列，原矩阵的第*j*列成为新矩阵的第*j*行，一个*m*行*n*列的矩阵转置之后得到*n*行*m*列的矩阵。numpy中常用矩阵对象的T属性实现转置操作。

```
In[ ]: print(x.T,y.T,sep='\n\n')
```

```
out[]: [[1 4]
        [2 5]
        [3 6]]
        [1]
        [2]
        [3]
        [4]
        [5]
        [6]]
```

## 6.3 numpy矩阵

Array object ndarray

### 6.3.4 矩阵常用操作

#### 02 矩阵乘法

```
In[ ]:x=np.matrix([[1,2,3],[4,5,6]])
      y=np.matrix([[1,2],[3,4],[5,6]])
      print(x*y)
out[ ]: [[22 28]
        [49 64]]
```

在线性代数中，一个 $m \times p$ 矩阵A和一个 $p \times n$ 矩阵B的乘积为一个 $m \times n$ 矩阵。其中，结果矩阵中每个元素 $C_{ij}$ 的值等于A矩阵中第 $i$ 行和B矩阵中第 $j$ 列的内积。

## 6.3 numpy矩阵

Numpy matrix

### 6.3.4 矩阵常用操作

03

#### 相关系数矩阵

相关系数矩阵是一个对称矩阵，其中对角线上的元素都是1，表示自相关系数；非对角线上的元素表示互相关系数，每个元素的绝对值都小于等于1，反映变量变化趋势的相似程度。对于二维相关系数矩阵而言，如果非对角线元素的值大于0，表示两个变量正相关，二者相互影响的变化方向相同。而且相关系数的绝对值越大，两个变量相互影响的程度越大。

## 6.3 numpy矩阵

Array object ndarray

### 6.3.4 矩阵常用操作

03

#### 相关系数矩阵

numpy提供的  
corrcoef()函数可以计算相  
关系数矩阵。

```
In[ ]:A=np.matrix([1,2,3,4])
      B=np.matrix([4,3,2,1])
      C=np.matrix([1,2,3,40])
      D=np.matrix([4,3,2,10])
      print('负相关, 变化方向相反: ',np.corrcoef(A,B))
      print('负相关, 变化方向接近相反: ',np.corrcoef(A,D))
      print('正相关, 变化方向相近: ',np.corrcoef(A,C))
      print('正相关, 变化方向一致: ',np.corrcoef(A,A))

out[:]:负相关, 变化方向相反:  [[ 1. -1.] [-1.  1.]]
      负相关, 变化方向接近相反:  [[1.  0.61065803] [0.61065803  1.]]
      正相关, 变化方向相近:  [[1.  0.8010362] [0.8010362  1.]]
      正相关, 变化方向一致:  [[1.  1.] [1.  1.] ]
```

## 6.3 numpy矩阵

Array object ndarray

### 6.3.4 矩阵常用操作

04

#### 方差、协方差、标准差

```
In[ ]:print("单变量协方差、方差: ",np.cov(A))
      print("两变量协方差: ",np.cov(A,B))
      print("单变量标准差: ",np.std(A))
      print("单变量行元素标准差: ",np.std(A,axis=1))
out[:]:单变量协方差、方差:  1.6666666666666665
      两变量协方差:  [[ 1.66666667 -1.66666667]
      [-1.66666667  1.66666667]]
      单变量标准差:  1.118033988749895
      单变量行元素标准差:  [[1.11803399]]
```

方差是随机变量或一组数据离散程度的度量，协方差用于衡量两个变量的总体误差。当两个变量相同的情况下，方差与协方差的计算结果相同。如果两个变量的变化趋势一致，即两个变量均大于自身的期望值，那么这两个变量的协方差是正值。如果两个变量的变化趋势相反，即其中一个变量大于自身的期望值，另外一个变量小于自身的期望值，那么这两个变量的协方差是负值。标准差是方差的算术平方根，反映一个数据集的离散程度。平均数相同的两组数据，标准差未必相同。



## 6.3 numpy矩阵

Numpy matrix

### 6.3.4 矩阵常用操作

05

#### 特征值和特征向量

$n \times n$  方阵  $A$  乘以一个向量，就是对这个向量进行了一个变换，将向量从一个坐标系变换到了另一个坐标系。如果矩阵乘以一个向量后仅发生了缩放变化，那么该向量就是矩阵的一个特征向量，特征值就是缩放比例。如果矩阵乘以一个向量后进行了旋转，那么特征向量是对向量旋转之后理想的坐标轴之一，特征值就是原向量在新坐标轴上的投影或者该坐标轴对原向量的贡献。特征值越大，原向量在新坐标轴上的投影越大，新坐标轴对原向量的表达越重要。一个矩阵的所有特征向量组成了矩阵的一组基，也就是新坐标系中的轴。有了特征值和特征向量，向量就可以在新坐标系中表示出来。

## 6.3.4 矩阵常用操作

### 05 特征值和特征向量

numpy中线性代数子模块linalg提供了计算特征值和特征向量的eig()函数，参数可以是Python列表、numpy数组或numpy矩阵。

```
In[ ]:A=np.matrix([[1,-3,3],[3,-5,3],[6,-6,4]])
      e,v=np.linalg.eig(A)
      print("特征值: ",e,sep='\n')
      print("特征向量: ",v,sep="\n")
      print("矩阵与特征向量的乘积: ",np.dot(A,v))
      print("特征值与特征向量的乘积: ",e*v)
      print("验证二者是否相等: ",np.isclose(np.dot(A,v),e*v))

out[:特征值:
      [ 4.+0.00000000e+00j -2.+1.10465796e-15j -2.-1.10465796e-15j]
      特征向量:
      [[-0.40824829+0.j          0.24400118-0.40702229j  0.24400118+0.40702229j]
       [-0.40824829+0.j          -0.41621909-0.40702229j -0.41621909+0.40702229j]
       [-0.81649658+0.j          -0.66022027+0.j          -0.66022027-0.j          ]]
      矩阵与特征向量的乘积:  [[-1.63299316+0.00000000e+00j -0.48800237+8.14044580e-01j
      -0.48800237-8.14044580e-01j]
       [-1.63299316+0.00000000e+00j  0.83243817+8.14044580e-01j
       0.83243817-8.14044580e-01j]
       [-3.26598632+0.00000000e+00j  1.32044054-5.55111512e-16j
       1.32044054+5.55111512e-16j]]
      特征值与特征向量的乘积:  [[0.81649658+4.50974724e-16j  3.12888345-8.14044580e-01j
      3.12888345+8.14044580e-01j]]
      验证二者是否相等:  [[False False False]
       [False False False]
       [False False False]]
```

## 6.3 numpy矩阵

Array object ndarray

### 6.3.4 矩阵常用操作

06

#### 矩阵乘法

```
In[ ]:A=np.matrix([[1,2],[3,4]])  
      B=np.linalg.inv(A)  
      print ("逆矩阵: \n",B)  
      print("AB乘积(对角线元素为1, 其余近似为0): \n",A*B)
```

out[]:逆矩阵:

```
[[ -2.    1. ]
```

```
 [ 1.5  -0.5]]
```

AB乘积(对角线元素为1, 其余近似为0):

```
[[1.0000000e+00  0.0000000e+00]
```

```
 [8.8817842e-16  1.0000000e+00]]
```

$n \times n$ 方阵A和B的乘积为单位矩阵，则称矩阵A为可逆矩阵，矩阵B为矩阵A的逆矩阵。numpy中线性代数子模块linalg提供了计算逆矩阵的inv()函数，参数必须为可逆矩阵，数据类型可以是Python列表、numpy数组或numpy矩阵。

## 6.3 numpy矩阵

Array object ndarray

### 6.3.4 矩阵常用操作

07

#### 范数

线性代数中， $n$ 维向量的长度称为模或2-范数，其模长就是向量与自身内积的平方根。对于 $m \times n$ 矩阵 $A$ ，其2-范数是矩阵 $A$ 的共轭转置矩阵与 $A$ 乘积的最大特征值的平方根。numpy中线性代数子模块linalg提供了计算不同范数的norm()函数，参数必须为可逆矩阵，数据类型可以是Python列表、numpy数组或numpy矩阵，可选参数ord用于指定范数类型，默认为2-范数。

```
In[ ]:A=np.matrix([[1,2],[3,4]])
print("矩阵2-范数: ",np.linalg.norm(A))
print("矩阵最小奇异值: ",np.linalg.norm(A,-2))
print("min(sum(abx),axis=0)=",np.linalg.norm(A,-1))
print("max(sum(abx),axis=0)=",np.linalg.norm(A,1))
print("向量中非0元素个数: ",np.linalg.norm([1,2,0,3,4,0],0))
print("向量2-范数: ",np.linalg.norm([1,2,0,3,4,0],2))

out[:]:矩阵2-范数:  5.477225575051661
矩阵最小奇异值:  0.36596619062625746
min(sum(abx),axis=0)= 4.0
max(sum(abx),axis=0)= 6.0
向量中非0元素个数:  4.0
向量2-范数:  5.477225575051661
```

## 6.3 numpy矩阵

Numpy matrix

### 6.3.4 矩阵常用操作

08

#### 奇异值分解

奇异值分解 ( Singular Value Decomposition , SVD ) 可以把大矩阵分解成几个更小矩阵的乘积，达到数据降维和去噪的效果。这是机器学习算法中主成分分析算法的理论基础。numpy中线性代数子模块linalg提供了计算奇异值分解的svd()函数。

## 6.3 numpy矩阵

Numpy matrix

### 6.3.4 矩阵常用操作

08

#### 奇异值分解

```
In[ ]:A=np.matrix([[1,2,3],[4,5,6],[7,8,9]])
      u,s,v=np.linalg.svd(A)
      print("u=",u)
      print("s=",s)
      print("v=",v)

out[]:u= [[-0.21483724  0.88723069  0.40824829]
          [-0.52058739  0.24964395 -0.81649658]
          [-0.82633754 -0.38794278  0.40824829]]
      s= [1.68481034e+01 1.06836951e+00
          4.41842475e-16]
      v= [[-0.47967118 -0.57236779 -0.66506441]
          [-0.77669099 -0.07568647  0.62531805]
          [-0.40824829  0.81649658 -0.40824829]]
```

svd()函数将矩阵A分解为 $u \cdot \text{np.diag}(s) \cdot v$ 的形式并返回u、s、v，其中数组s中的元素就是矩阵A的奇异值。

```
In[ ]:u*np.diag(s)*v
out[]:matrix([[1., 2., 3.],
              [4., 5., 6.],
              [7., 8., 9.]])
```





# PART **FOUR**

## 6.4 精选案例

Selected cases

## 6.4 精选案例

Selected cases

### 6.4.1 美国总统大选数据统计

- 任务描述



本案例对特朗普和克林顿每个月的民意调查数据进行统计分析，涉及的知识点包括：numpy读取csv文件、处理日期格式的文本数据、numpy的切片与索引、numpy的统计方法以及列表推导式等。

本案例数据分析的主要步骤包括：首先，读取指定列的数据；然后，处理表示日期的文本数据，转换为形如“yyyy-mm”的字符串，提取选票日期；最后，统计每个月的投票数据。

## 6.4.1 美国总统大选数据统计

### 1) 数据集简介

本案例使用的数据集包含了从2015年11月到2016年11月美国总统大选的民意调查选票数据。该数据集由27列不同类型的数据组成，保存为一个CSV文件，如下图所示。此数据集来源于kaggle网站，网址为：  
<https://www.kaggle.com/fivethirtyeight/2016-election-polls>。

F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
state	startdate	enddate	pollstgradesamples	popul	poll	rawpoll_clinton	rawpoll_trump	rawpoll_rawpoll_radjpoll_clinton	adjpoll_trump	adjpoll_johnson	adjpoll				
U.S.	10/25/2016	10/31/2016	GoogleB	24316 lv	6.14	37.69	35.07	6.18				42.6414	40.86509	5.675099	
U.S.	10/27/2016	10/30/2016	ABC NeA+	1128 lv	4.2	45	46	3				43.29659	44.72984	3.401513	
Virginia	10/27/2016	10/30/2016	ABC NeA+	1024 lv	3.88	48	42	6				46.29779	40.72604	6.401513	
Florida	10/20/2016	10/24/2016	SurveyA	1251 lv	3.4	48	45	2				46.35931	45.30585	1.77773	
U.S.	10/20/2016	10/25/2016	Pew ReB+	2120 rv	3.39	46	40	6				45.32744	42.20888	3.61832	
U.S.	10/22/2016	10/25/2016	Fox NeA	1221 lv	3.35	44	41	7				44.6508	42.26663	6.114222	
U.S.	10/26/2016	10/31/2016	IED/TIA-	1018 lv	3.28	44.6	43.7	4.2				46.21834	43.56017	3.15359	
Pennsylvania	10/25/2016	10/30/2016	GravisB-	3217 rv	3.22	47	44	3				46.89049	43.50333	3.466432	
U.S.	10/23/2016	10/27/2016	Ipsos A-	1158 lv	3.14	41.7	36.4	6.3				41.22576	37.24948	6.420006	
U.S.	10/29/2016	10/31/2016	The Times-F	2600 lv	3.04	42	40	5				42.21983	41.6954	4.220173	
U.S.	10/25/2016	10/31/2016	USC Dornsif	3145 lv	2.96	43.28	46.9					44.53217	43.84845		
Florida	10/25/2016	10/27/2016	Siena A	815 lv	2.76	42	46	4				41.81832	47.92262	2.676897	
California	10/14/2016	10/23/2016	PublicA	1024 lv	2.74	54	28	5				55.68839	29.50605	3.17051	
U.S.	10/29/2016	10/30/2016	Morning Cor	1772 lv	2.73	42	39	7				43.31551	40.34972	5.823322	
North Caro	10/25/2016	10/26/2016	MaristA	780 lv	2.73	47	41	8				45.20793	42.01937	6.499082	
Florida	10/25/2016	10/26/2016	MaristA	779 lv	2.72	45	44	5				43.19458	45.07725	3.499082	
U.S.	10/20/2016	10/24/2016	GfK GrB+	1212 lv	2.71	51	37	6				50.18283	39.33826	5.044833	
Florida	10/21/2016	10/24/2016	SelzerA+	805 lv	2.69	43	45	4				42.67789	46.11255	3.054228	
Pennsylvania	10/26/2016	10/28/2016	YouGovB	1091 lv	2.62	48	40	5				47.77047	39.80679	6.359501	
Pennsylvania	10/23/2016	10/25/2016	Siena A	824 lv	2.58	46	39	6				45.74354	41.34735	4.421316	
U.S.	10/14/2016	10/17/2016	SelzerA+	1000 lv	2.54	47	38	8				46.84417	39.99571	6.27284	
Missouri	10/27/2016	10/28/2016	EK Strategi	1698 lv	2.5	39	53					38.51061	50.7572		
Colorado	10/26/2016	10/28/2016	YouGovB	997 lv	2.49	42	39	7				41.75385	38.87231	8.359501	
U.S.	10/22/2016	10/26/2016	YouGovB	1209 lv	2.49	46	41	4				45.63602	41.55637	4.964521	
Arizona	10/26/2016	10/28/2016	YouGovB	994 lv	2.48	42	44	4				41.76	43.84806	5.359501	

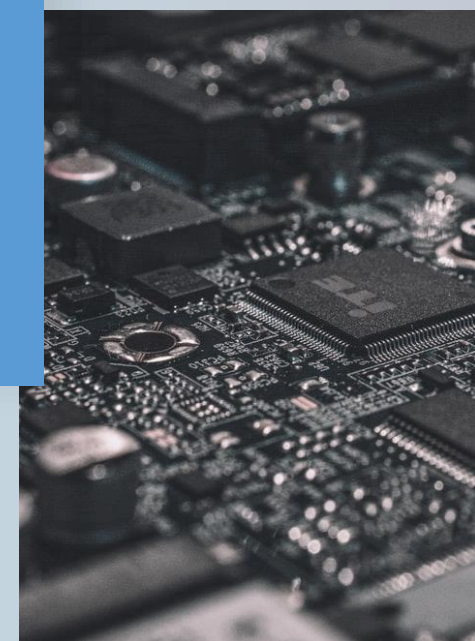
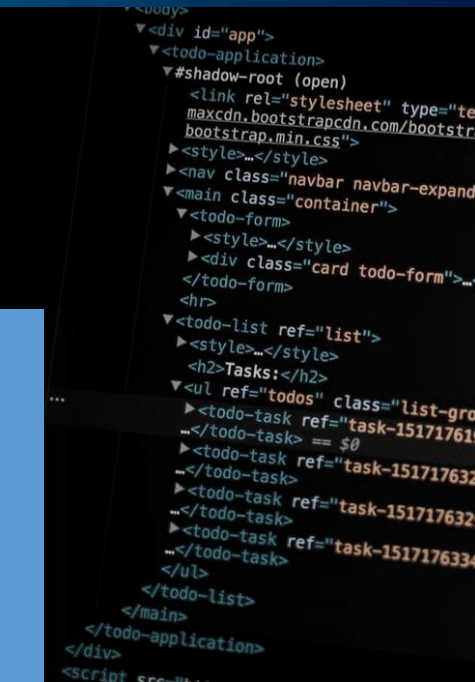
本案例的数据文件中可以用于每个月民意调查统计的数据有3列。其中enddate属性表示统计选票数据的结束日期；rawpoll\_clinton和rawpoll\_trump属性分别表示克林顿和特朗普在这一天获得的选票数。因此，本案例的任务是从enddate属性中提取年份和月份，然后将rawpoll\_clinton和rawpoll\_trump属性的取值以月份为单位进行统计。

## 6.4.1 美国总统大选数据统计

### 2) 导入模块

本案例使用numpy扩展库进行数据统计分析，因此首先导入numpy模块。此外，Python标准库datetime提供了日期和时间处理功能。本案例使用datetime.datetime.strptime ()函数将一个日期和时间的格式化字符串转换为datetime对象，因此需要导入datetime模块。

```
In[ ]:import numpy as np
import datetime
```



### ● 3) 加载数据

[illegible]



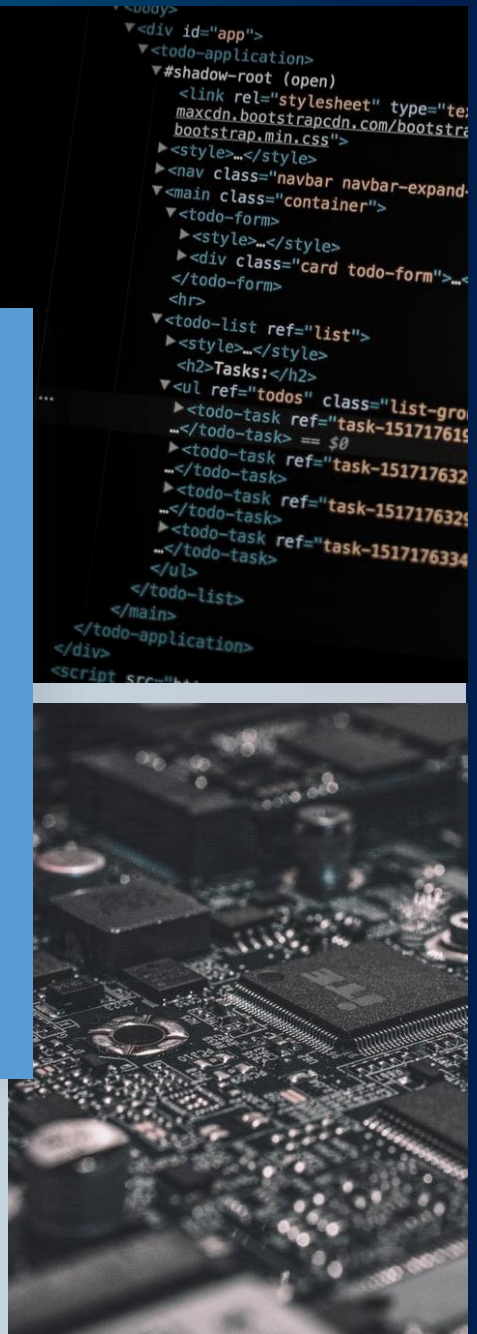
## 6.4.1 美国总统大选数据统计

### 3) 加载数据

自定义函数load\_data用于加载数据文件filename，读取指定列use\_cols的CSV数据。

代码第3~4行：读取CSV文件中第一行数据，也就是字符串类型的“列名”数据。因为CSV文件每一行的末尾以换行符“\n”结束，这里使用readline()方法读取文件第一行的字符串数据，利用切片操作[:-1]避免读取第一行末尾的换行符“\n”。这样，变量col\_names\_str保存了CSV文件的第一行，即由逗号分隔的列名字符串数据。

代码第7行：以逗号为分隔符，使用split方法将列名字符串拆分成列表，列表中每个元素是一个列名字符串。



## 6.4.1 美国总统大选数据统计

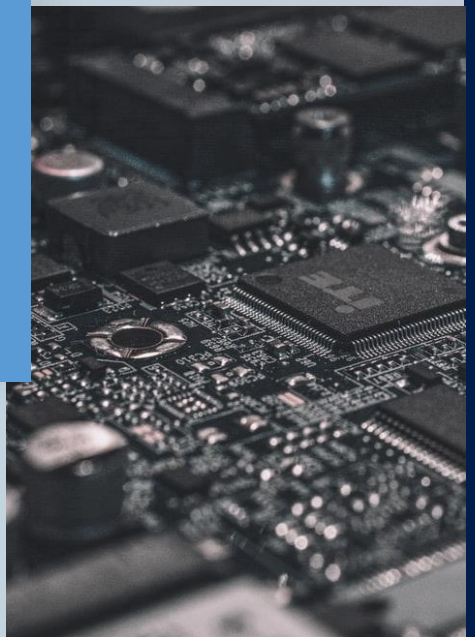
### 3) 加载数据

代码10~11行：使用列表推导式在指定列use\_cols中遍历每一个列名use\_col\_name，对应找到该列名在列名列表col\_name\_lst中的索引col\_name\_lst.index(use\_col\_name)。这样，变量use\_col\_index\_lst中存储了指定列的索引。

代码14~18行：使用numpy读取CSV文件，根据列索引访问指定列的CSV数据。考虑到CSV文件使用逗号作为数据分隔符，loadtxt()函数的参数delimiter指定分隔符为“,”；CSV文件第一行为列名，不是需要读取的列数据，参数skiprows设为1，表示跳过文件第一行，从文件第二行开始读取数据；考虑到numpy数组只存放同类型的数据，参数dtype指定读取的数据为字符串类型，保证数据安全并方便后续数据处理。

代码第20行：load\_data函数的返回值data\_array保存了指定列的字符串数据。

```
<body>
<div id="app">
  <todo-application>
    <#shadow-root (open)>
      <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
      <style>...</style>
      <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
          </todo-form>
          <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717615">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717633">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </div>
    </todo-application>
    <script src="...">
```

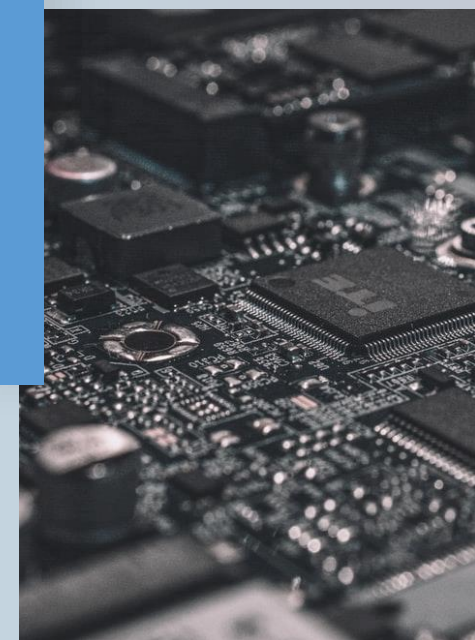


## 6.4.1 美国总统大选数据统计

### 4) 数据预处理

基于统计分析任务目标，本案例从提供的CSV数据文件中指定并读取3列数据：“enddate”、“rawpoll\_clinton”和“rawpoll\_trump”。其中enddate列为字符串类型的数据。观察这些字符串发现，其数据格式并不一致，且基本都是具体到某一天的日期格式字符串。因此，数据预处理阶段需要从enddate日期字符串中提取出月份数据，为后续以月份为单位的选票统计准备数据。

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>_</style>
        <nav class="navbar navbar-expand-sm">
          <main class="container">
            <todo-form>
              <style>_</style>
              <div class="card todo-form">_</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>_</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717619">_</todo-task>
                <todo-task ref="task-151717632">_</todo-task>
                <todo-task ref="task-1517176329">_</todo-task>
                <todo-task ref="task-1517176334">_</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="https://unpkg.com/vue@2.6.10/dist/vue.min.js">_</script>
  </body>
```



## 6.4.1 美国总统大选数据统计

### 4) 数据预处理

```
1  def process_date(data_array):
2      #处理日期格式数据，转换为yyyy-mm字符串
3      enddate_lst = data_array[:, 0].tolist()
4      # 将日期字符串格式统一，即'yy/dd/mm'
5      enddate_lst = [enddate.replace('-', '/') for enddate in enddate_lst]
6      # 将日期字符串转换成日期
7      date_lst= [datetime.datetime.strptime(enddate, '%m/%d/%Y') for enddate
8  in enddate_lst]
9      # 构造年份-月份列表
10     month_lst = ['{}-{:02d}'.format(date_obj.year, date_obj.month) for
11 date_obj in date_lst]
12
13     month_array = np.array(month_lst)
14     data_array[:, 0] = month_array
15     return data_array
```



## 6.4.1 美国总统大选数据统计

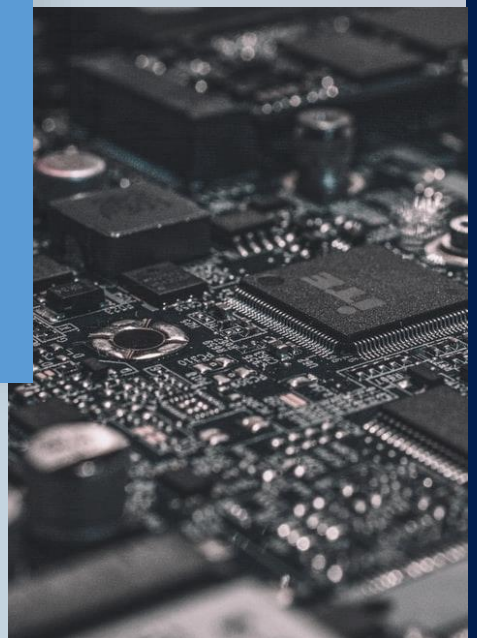
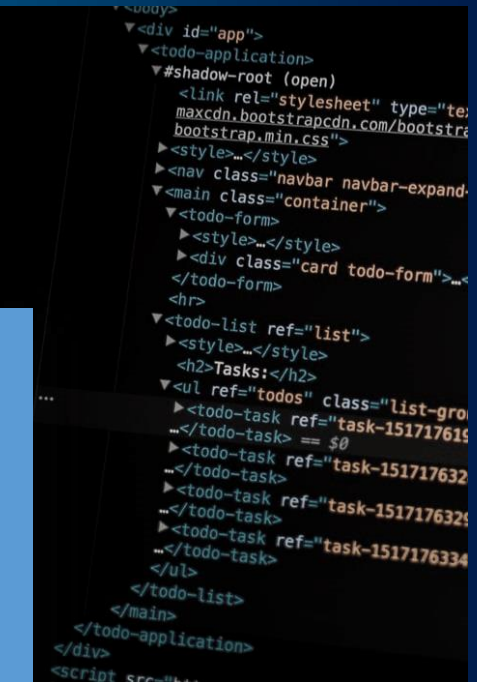
### 4) 数据预处理

自定义函数process\_date中的实参data\_array接收“enddate、rawpoll\_clinton和rawpoll\_trump”三列字符串数据。

代码第3行：将实参data\_array中第一列数据enddate转换为列表类型。

代码第5行：将日期字符串中形为“yy-dd-mm”的数据转换为“yy/dd/mm”形式，实现所有enddate数据格式的统一。

代码8~9行：使用datetime.datetime.strptime()函数将日期字符串转换为形如“month/day/year”的datetime对象，使用列表推导式将enddate\_lst列表元素生成的datetime对象作为元素构成新列表date\_lst。




## 6.4.1 美国总统大选数据统计

### 4) 数据预处理

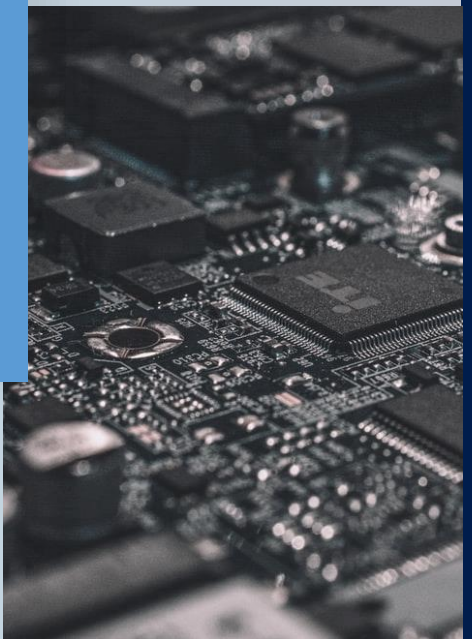
代码11~12行：遍历列表date\_lst，从每一个日期数据中提取年、月，生成形为“year-month”字符串的列表元素，进而构造一个“年份-月份”列表month\_lst。

代码14行：将列表month\_lst转换为numpy数组month\_array。

代码15~16行：将实参数组data\_array中第一列，即索引为0的enddate数据替换为“年份-月份”形式的字符串。至此，函数返回值data\_array保存了从enddate日期字符串中提取出的年份和月份，以及CSV文件的rawpoll\_clinton和rawpoll\_trump数据。



```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717633">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```





## 6.4.1 美国总统大选数据统计

### 5) 数据统计分析

经过数据预处理，本案例已经得到了每条记录中选票所在的年份和月份。在数据统计分析阶段，可以基于月份对每个月的选票数据进行统计。

```
1 def get_month_stats(data_array):      # 统计每月的投票数据
2     months = np.unique(data_array[:, 0])
3     for month in months: # 根据月份过滤数据
4         filtered_data = data_array[data_array[:, 0] == month]
5         # 获取投票数据，字符串数组转换为数值型数组
6         try:
7             filtered_poll_data = filtered_data[:, 1:].astype(float)
8         except ValueError:
9             # 遇到不能转换为数值的字符串，跳过循环
10            continue
11        result = np.sum(filtered_poll_data, axis=0)
12        print('{} , Clinton票数: {}, Trump票数: {}'.format(month,
13 result[0], result[1]))      # 列方向求和
```

```
<body>
  <div id="app">
    <todo-application>
      <shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style></style>
        <nav class="navbar navbar-expand-sm">
          <main class="container">
            <todo-form>
              <style></style>
              <div class="card todo-form">
                </div>
            </main>
          </nav>
        </shadow-root>
      </todo-application>
    </div>
  </body>
```




## 6.4.1 美国总统大选数据统计

### 5) 数据统计分析

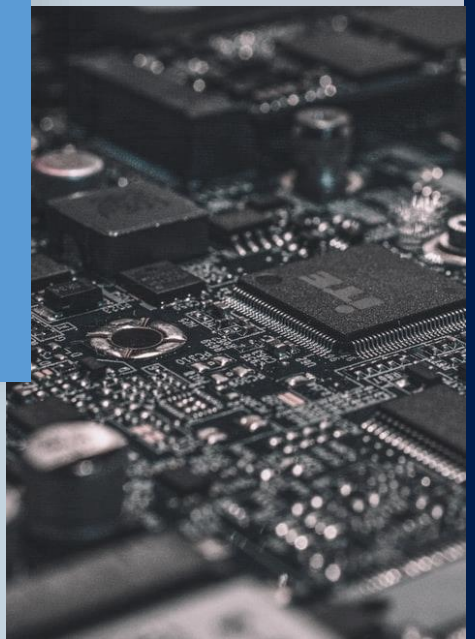
代码第6~10行：进行异常处理。考虑到原始数据文件中可能存在数据缺失等问题，将筛选出的filtered\_data数组中所有数据去除第一列“年份-月份”数据之后，转换成浮点数据类型。其实就是将CSV文件中rawpoll\_clinton和rawpoll\_trump两列数据转换为浮点数；若存在数据缺失或者遇到不能转换为数值的字符串，就跳过本次循环并忽略那些有问题的数据。这也是常用的数据清理操作之一。至此，numpy数组filtered\_poll\_data中保存了符合指定条件的、质量合格的选票数据。

代码11行：对符合要求的选票数据求和，参数axis=0表示按列操作，分别统计出克林顿和特朗普每个月的民意调查选票数据。

代码12~14行：打印输出统计结果。



```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717633">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
  </body>
```



## 6.4.1 美国总统大选数据统计

### ● 6) 主函数

主函数中，首先找到数据文件及其路径，指定需要读取的3列数据：

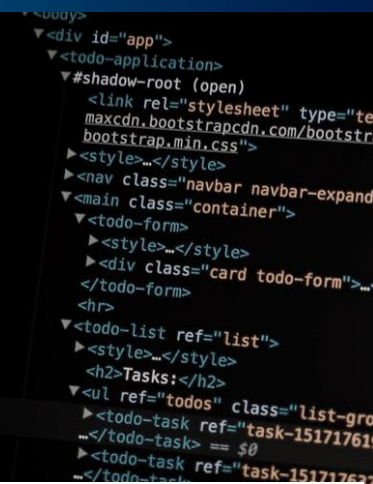
“enddate”、“rawpoll\_clinton”和“rawpoll\_trump”。

代码第4行调用load\_data函数，使用numpy读取CSV数据文件；

代码第6行调用process\_date函数处理日期格式的文本数据，转换为“yyyy-mm”格式的字符串；

代码第9行调用get\_month\_stats函数统计每月的投票数据并输出统计结果。

```
1 filename='/home/aistudio/data/data76670/presidential_polls.csv'
2 # 读取指定列的数据
3 use_cols = ['enddate', 'rawpoll_clinton', 'rawpoll_trump']
4 data_array = load_data(filename, use_cols)
5 # 处理日期格式数据，转换为yyyy-mm字符串
6 proc_data_array = process_date(data_array)
7
8 # 统计每月的投票数据
9 get_month_stats(proc_data_array)
```





## 6.4.1 美国总统大选数据统计

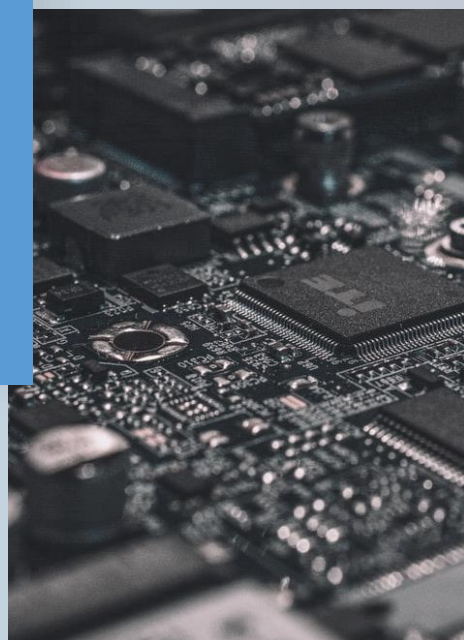
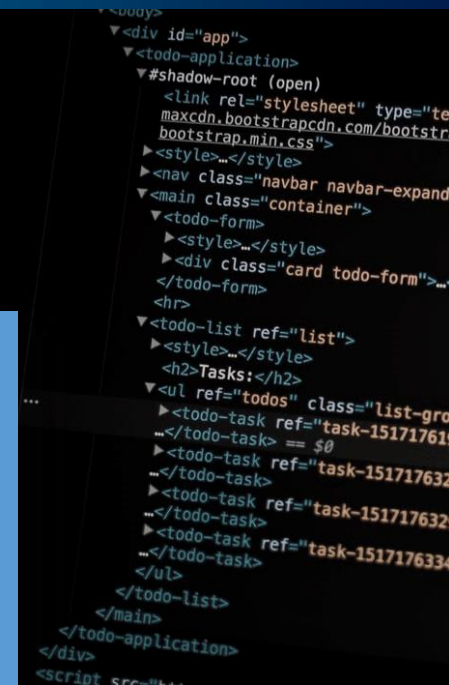
### 6) 主函数

- 运行结果：

```
out[]:
2015-11, Clinton 票数: 1920.0, Trump 票数: 1948.2
2015-12, Clinton 票数: 4816.799999999999, Trump 票数: 4164.299999999999
2016-01, Clinton 票数: 6861.600076850002, Trump 票数: 6267.0
2016-02, Clinton 票数: 8271.600253600001, Trump 票数: 7528.200000000002
2016-03, Clinton 票数: 11656.202546, Trump 票数: 9626.699999999999
2016-04, Clinton 票数: 11911.803926800001, Trump 票数: 9396.300000000005
2016-07, Clinton 票数: 22007.013854599994, Trump 票数: 21426.99
2016-08, Clinton 票数: 63619.39624199996, Trump 票数: 59529.000000000015
```

统计结果显示，“2016-05”和“2016-06”没有统计数据，说明这两个月份存在数据缺失等问题，在数据预处理模块中忽略了这些有问题的数据，因此没有在统计结果中呈现。

本案例采用numpy数组对CSV数据进行统计分析，利用np.sum函数简洁方便地实现了功能要求，避免了繁琐而低效的循环操作。然而，本案例在统计分析之前，需要完成繁琐的数据预处理，这也是扩展库numpy用于数据分析的缺陷。实际上，Python扩展库numpy更擅长科学计算应用，后面章节的扩展库pandas将为用户提供简洁而方便的专业化数据分析功能。





# PART **FOUR**

## 6.4 精选案例

Selected cases




## 6.4 精选案例

Selected cases

### ● 6.4.2 约会配对案例

- 任务描述



任务描述 本案例采用K-近邻算法（KNN）筛选可能的约会对象。案例主要使用扩展库numpy提供的数组操作完成数据分类预测。主要步骤包括：

第一，设计KNN分类器：实现KNN分类算法；

第二，读取数据：从文本文件中读取数据并转换成numpy数组之后进行数据解析；

第三，数据预处理：将数值数据归一化处理；

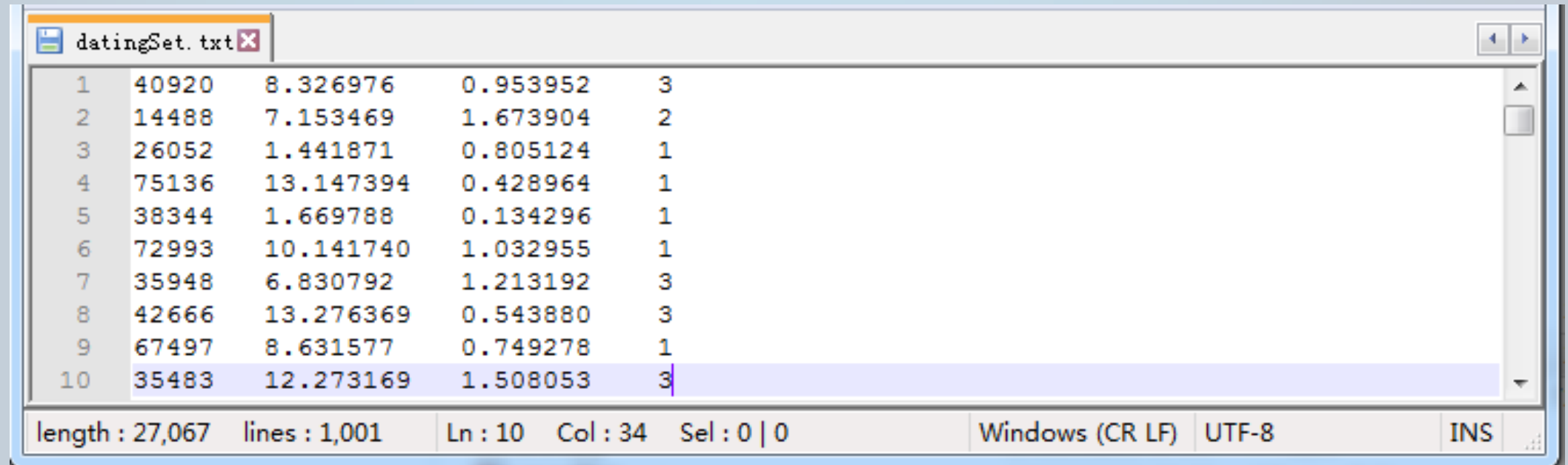
第四，测试分类器性能：提取部分数据作为测试样本，使用错误率检测分类器性能；

第五，分类应用：使用KNN分类器完成约会对象筛选任务。

## 6.4.2 约会配对案例

### 1) 数据集简介

本案例使用公开的约会配对数据集。每个样本数据占据一行，由三个样本特征和一个类标签组成，所有数据保存在文本文件datingSet.txt中，如下图所示：



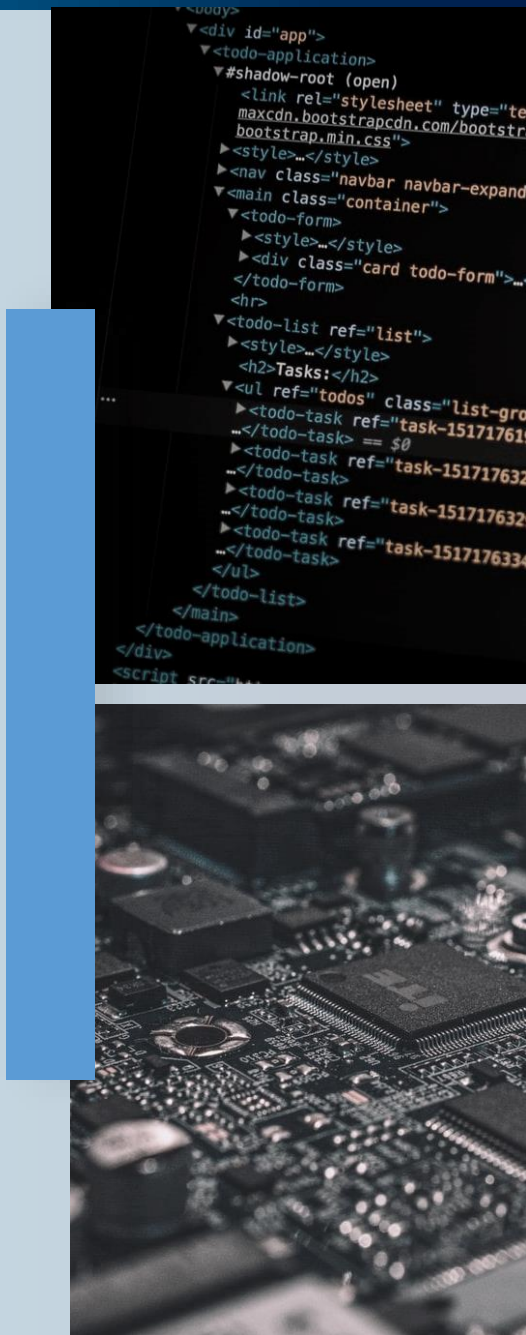
1	40920	8.326976	0.953952	3
2	14488	7.153469	1.673904	2
3	26052	1.441871	0.805124	1
4	75136	13.147394	0.428964	1
5	38344	1.669788	0.134296	1
6	72993	10.141740	1.032955	1
7	35948	6.830792	1.213192	3
8	42666	13.276369	0.543880	3
9	67497	8.631577	0.749278	1
10	35483	12.273169	1.508053	3

## 6.4.2 约会配对案例

### 1) 数据集简介

该数据文件包含1001行约会配对数据，但并不包含列名。其中前三列为样本特征，依次表示每个潜在约会对象每年飞行的里程数、玩游戏和视频所占的时间比以及每周消费冰淇淋的公升数；文件第四列表示对这个潜在约会对象的喜欢程度，分“1”、“2”和“3”三个类别，分别代表“不喜欢的人”、“有点喜欢的人”和“比较喜欢的人”。

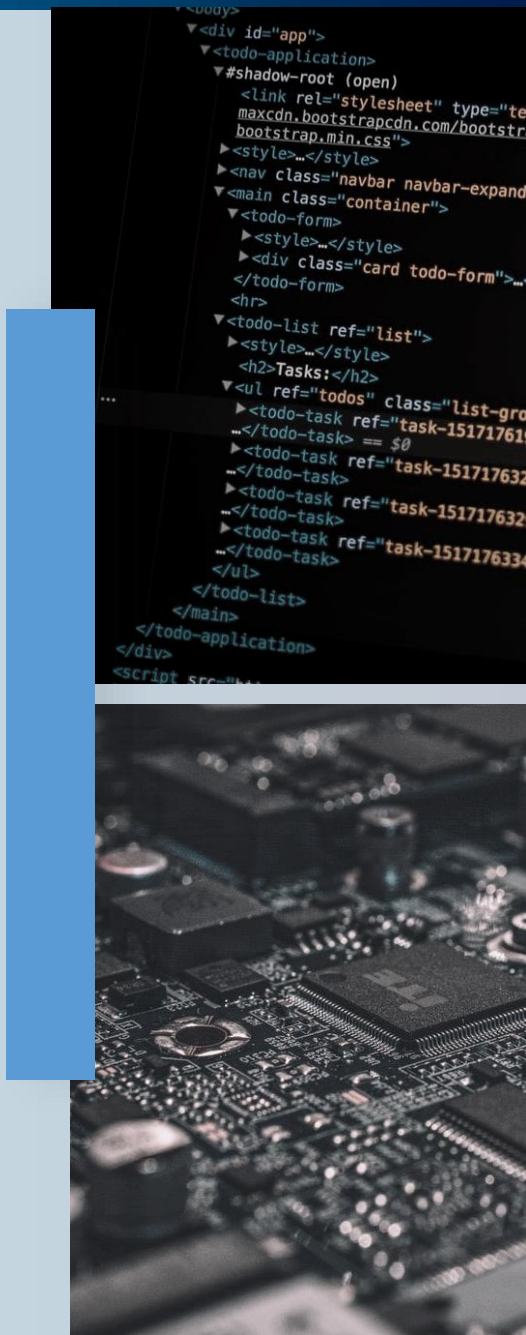
本案例将构建一个可用的约会对象筛选系统。用户输入潜在约会对象的信息，即上述三个特征数据，系统利用KNN算法预测出用户对该约会对象的喜欢程度。



## 6.4.2 约会配对案例

### 2) 导入模块

本案例将运用大量的numpy库函数和方法进行操作，因此使用 “from numpy import \*” 命令导入numpy模块，避免频繁地书写库名；使用命令 “import operator” 导入operator 模块，以便使用与Python内置运算符对应的高效率函数，例如：operator.itemgetter()函数用于获取对象指定维度上的数据；使用命令 “from os import listdir” 导入os库，以便使用 os.listdir() 函数返回指定文件夹包含的文件或文件夹名称的列表。

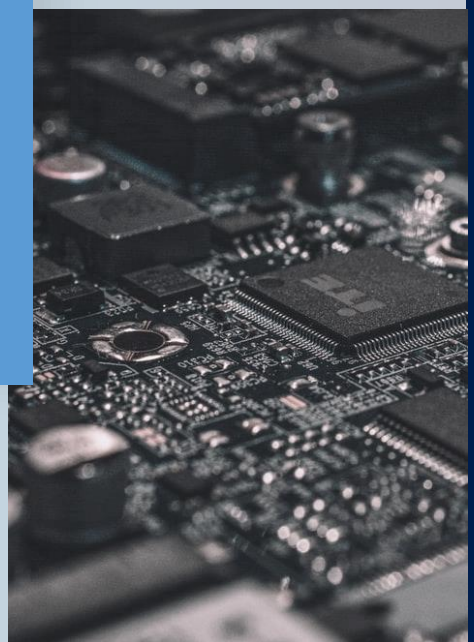


## 6.4.2 约会配对案例

### 3) 构建KNN分类器

KNN算法通常应用于含有标签的样本集合，其基本原理是：存在一个样本集合，其中每个样本都有标签；输入没有标签的新样本之后，将新样本的每个特征与样本集数据对应的特征进行比较；然后提取样本集中与新样本特征最相似的K个数据的分类标签作为新样本的标签。也就是说，给定含有m个实例的数据集 $T=\{(x_1,y_1),(x_2,y_2),(x_3,y_3),\dots,(x_m,y_m)\}$ ，分别计算新样本到m个实例的距离，然后找出与新样本特征最接近的k个实例，这k个实例中哪个类别的实例最多，那么新样本就拥有哪个实例对应的分类标签。

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>_</style>
        <nav class="navbar navbar-expand">
          <main class="container">
            <todo-form>
              <style>_</style>
              <div class="card todo-form">_</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>_</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-15171761">_</todo-task>
                <todo-task ref="task-151717632">_</todo-task>
                <todo-task ref="task-151717632">_</todo-task>
                <todo-task ref="task-1517176334">_</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      <script src="...">_</script>
    </div>
  </body>
```





## 6.4.2 约会配对案例

### 3) 构建KNN分类器

自定义函数classify0()接收四个输入参数：参数inX接收用于分类的numpy数组，参数dataSet表示输入的训练样本集，参数label为样本的类别标签，参数k表示选择最近邻居的数目，其中标签向量label的元素数目和numpy数组dataSet的行数相同。

```
1 def classify0(inX, dataSet, labels, k):
2     dataSetSize = dataSet.shape[0]
3     diffMat = tile(inX, (dataSetSize,1)) - dataSet
4     sqDiffMat = diffMat**2
5     sqDistances = sqDiffMat.sum(axis=1)
6     distances = sqDistances**0.5
7     sortedDistIndicies = distances.argsort()
8     #argsort方法返回将数组值从小到大排序之后的索引值
9     classCount={} #字典
10    for i in range(k):
11        voteIlabel = labels[sortedDistIndicies[i]]
12        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
13    sortedClassCount = sorted(classCount.items(),
14    key=operator.itemgetter(1), reverse=True)
15    return sortedClassCount[0][0]
```

```
<body>
  <div id="app">
    <todo-application>
      <shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style></style>
        <nav class="navbar navbar-expand-lg">
          <main class="container">
            <todo-form>
              <div>
                <input type="text">
                <button type="button" value="Add">
              </div>
            </main>
          </nav>
        </shadow-root>
      </todo-application>
    </div>
  </body>
```



## 6.4.2 约会配对案例

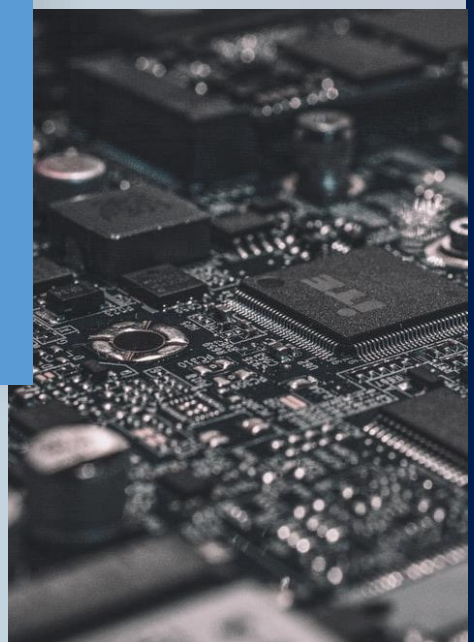
### 3) 构建KNN分类器

代码第2行使用shape[0]属性获取矩阵第一维度的长度，这里返回dataSet矩阵的行数。

代码第3行元组(dataSetSize,1) 中第一个元素表示复制之后的行数，第二个元素表示对inx的重复次数。这里使用title()函数将numpy数组inX重复dataSetSize行1列后构成一个新数组，与样本数据集进行减操作，实现数据集中A(x0,y0)和B(x1,y1) 两个向量点的减操作；再经过代码3~6行对numpy数组diffMat求平方、累加、开根号后，得到向量点A(x0,y0)和B(x1,y1)之间的距离d。

$$d = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \quad (\text{公式6-1})$$

```
<body>
  <div id="app">
    <todo-application>
      <shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
          </todo-form>
          <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717615">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717634">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </shadow-root>
    </todo-application>
  </div>
  <script src="...">
```



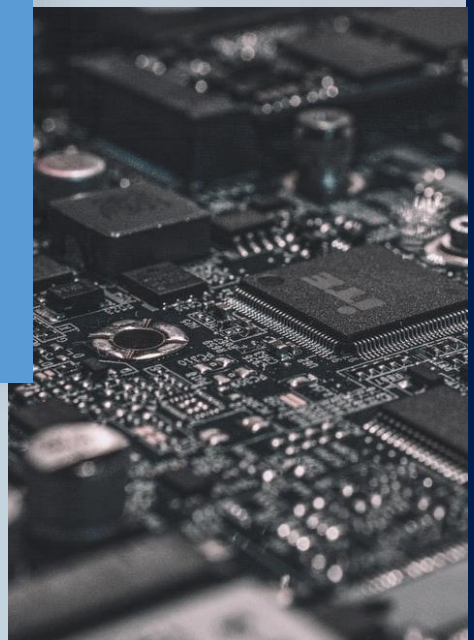
## 6.4.2 约会配对案例

### 3) 构建KNN分类器

计算了输入向量点与样本集中所有点之间的距离，代码第7行对距离数据从小到大排序，利用argsort()方法返回数组元素值顺序排列后的索引值。

接着，代码10~14行确定样本集中与输入向量点距离最小的前k个元素对应的类别标签，根据前k个最接近输入数据的类别标签预测当前输入数据的类别标签。代码第11行依次获取前k个距离最小的索引对应的类别标签取值；代码第12行利用字典classCount统计这k个标签各自出现的频率，其中字典元素的“键”为“类别标签”，“值”为“标签出现频率”。

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
            <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717619">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-151717634">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </todo-application>
    </div>
    <script src="...">
```

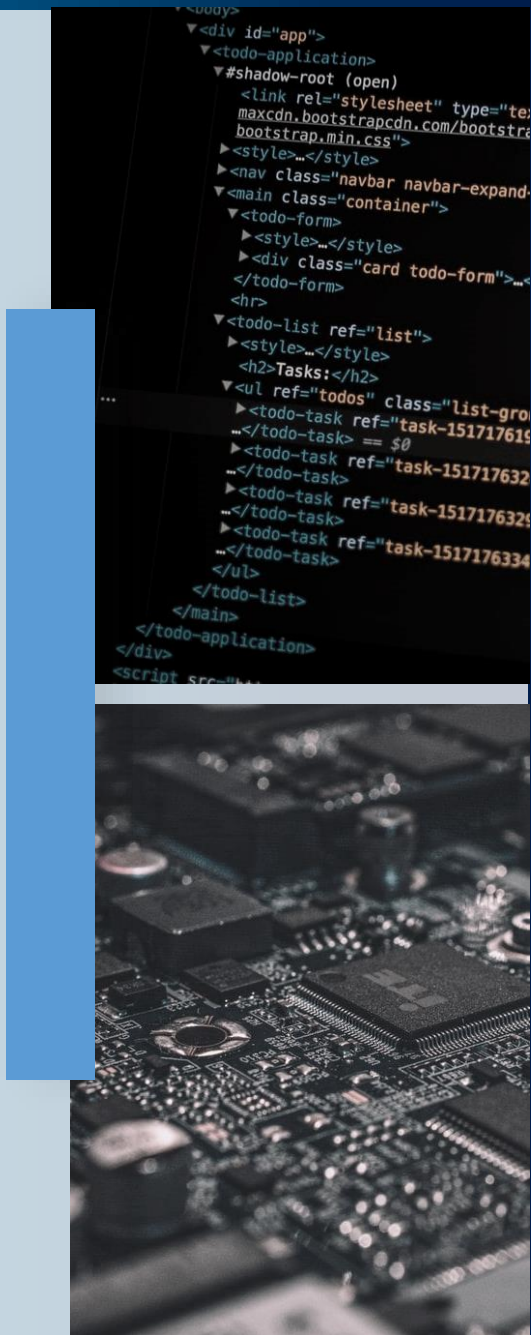


## 6.4.2 约会配对案例

### 3) 构建KNN分类器

代码第13~14行使用sorted()函数对统计结果进行排序，参数reverse=True表示逆序排列，出现频率高的类别排在首位，即索引值为0的位置；参数classCount.items()返回“键值对”元组数据组成的列表；函数operator.itemgetter(1)表示元组元素中索引为1的数据，也就是字典元素的“值”对应的“标签出现频率”；参数key=operator.itemgetter(1)表示按照类别标签出现频率进行排序。因此，将前k个最相似向量点的类标签按照出现频率的逆序排列后，排在最前面的就是出现频率最高的类标签。

最后，函数返回值sortedClassCount[0][0]将此类别标签作为KNN算法对输入向量点的预测结果返回。



```

<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand">
          <main class="container">
            <todo-form>
              <style>...</style>
              <div>

```

#### ● 4) 读取数据

从文本文件读入的约会对象特征数据，需要转换为分类器可以读取的格式，才能输入到KNN分类器。自定义函数file2matrix用于处理输入数据的格式问题。该函数的输入为文件名字符串，输出为样本特征矩阵和类标签向量。

```

1 def file2matrix(filename):
2     fr = open(filename)
3     arrayOfLines=fr.readlines()
4     numberOfLines=len(arrayOfLines) # 得到文件行数
5     returnMat = zeros((numberOfLines,3)) # 创建二维numpy数组
6     classLabelVector = [] # 创建存放标签的列表
7     index = 0
8     for line in arrayOfLines:
9         line = line.strip() # 移除字符串头尾的空格，生成新字符串
10        listFromLine = line.split('\t')
11        returnMat[index,:] = listFromLine[0:3]
12        #向全零元素的ndarray中添加数据
13        classLabelVector.append(int(listFromLine[-1]))
14        index += 1
15    return returnMat,classLabelVector

```



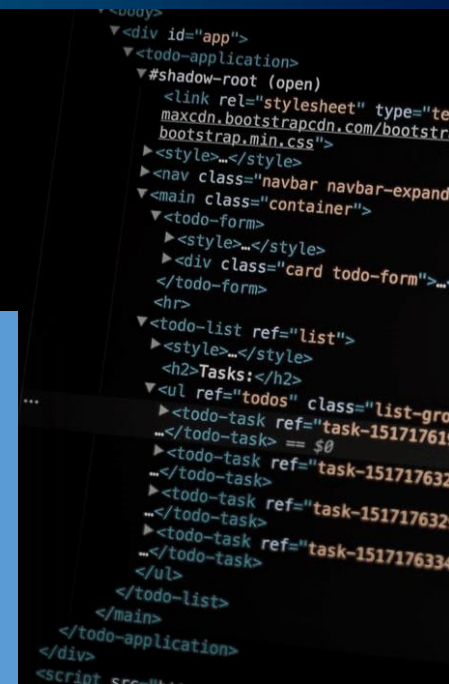
## 6.4.2 约会配对案例

### 4) 读取数据

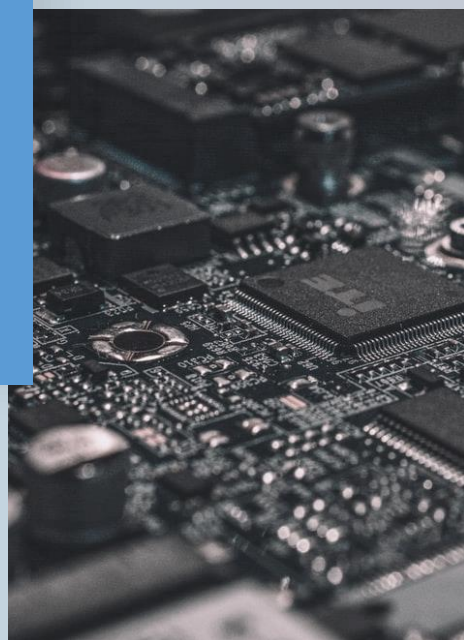
代码第2~4行打开文本文件，读取文件中所有行，返回以文件中每行数据为字符串元素的列表。使用len()函数得到列表长度，即文件的行数numberOfLines。

代码第5行创建一个行数为numberOfLines，列数为3的全零矩阵，用于存放处理后准备输出的样本特征。

代码第8~15行依次读取文本文件中的每一行数据，使用strip()方法移除字符串首尾的空格，依据tab字符“\t”将每行数据拆分成数据元素组成的列表listFromLine；将列表中的前三列元素，即约会对象的特征保存至特征矩阵returnMat；将列表中的最后一列元素，即类别标签追加至类别向量中。如此处理文本文件中的每一行数据，将最后的特征矩阵和类别向量作为函数的返回值。



```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
            <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <li><todo-task ref="task-151717619">...</li>
              <li><todo-task ref="task-151717632">...</li>
              <li><todo-task ref="task-151717632">...</li>
              <li><todo-task ref="task-151717634">...</li>
            </ul>
          </todo-list>
        </main>
      </todo-application>
    </div>
  </body>
```

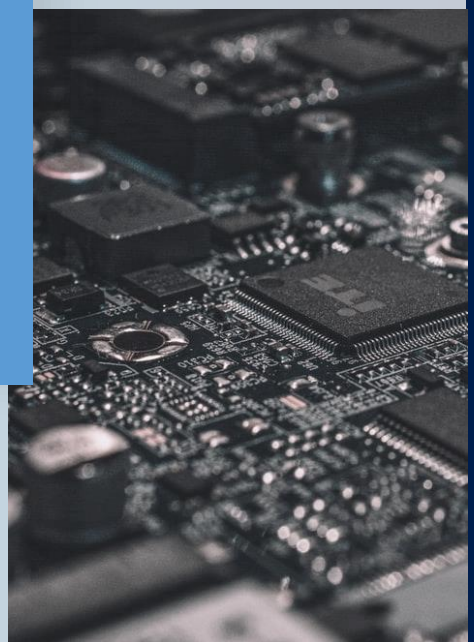


## 6.4.2 约会配对案例

### 4) 读取数据

需要注意的是，列表变量listFromLine中存储的是取自文本文件的字符串数据。代码13行将列表变量listFromLine中最后一列，即类别标签字符串，通过int函数转换成整型数据后保存至列表classLabelVector。对于列表变量listFromLine前三列的特征数据，本案例没有使用纯Python语句处理变量值类型转换问题，而是使用numpy函数库自动将字符串类型的列表元素转换成浮点型数据并进行后续操作。这也是扩展库numpy的优势之一。

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-sm">
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717619">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-1517176329">...</todo-task>
                <todo-task ref="task-1517176334">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
  </body>
```





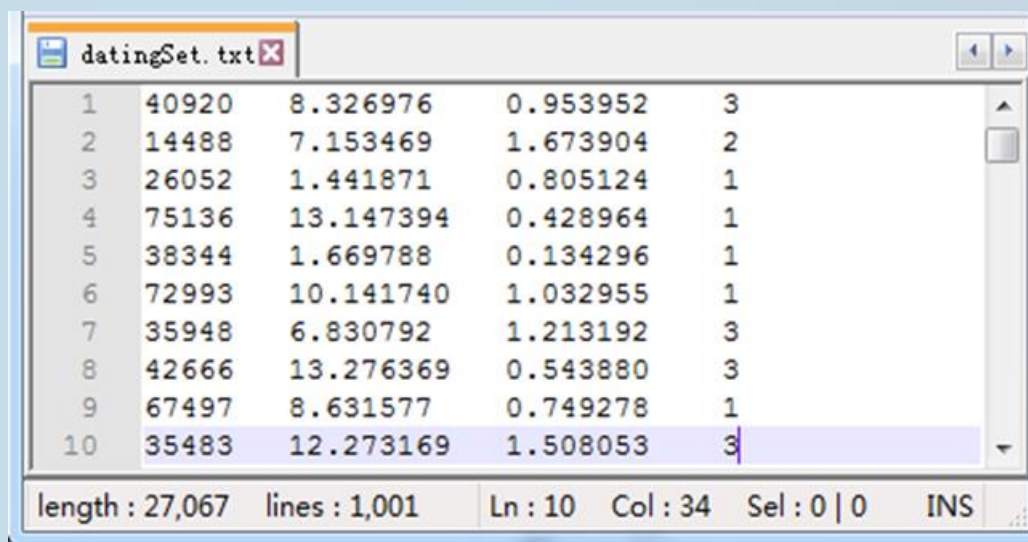
## 6.4.2 约会配对案例

### 5) 数据预处理

如右图所示，从矩阵returnMat可以提取出所有样本三个方面的特征数据，依次为：每年完成的飞行里程数，每周消费的冰激淋公升数和玩游戏看视频消耗的时间百分比。

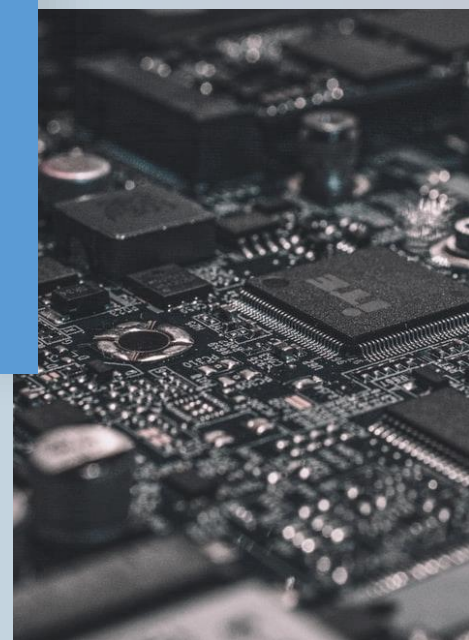
以样本3和样本4为例，根据公式6-1可以计算出样本之间的距离。

$$d_{3,4} = \sqrt{(75136 - 26052)^2 + (13.147394 - 1.441871)^2 + (0.428964 - 0.805124)^2}$$



	1	2	3	4	5
1	40920	8.326976	0.953952	3	
2	14488	7.153469	1.673904	2	
3	26052	1.441871	0.805124	1	
4	75136	13.147394	0.428964	1	
5	38344	1.669788	0.134296	1	
6	72993	10.141740	1.032955	1	
7	35948	6.830792	1.213192	3	
8	42666	13.276369	0.543880	3	
9	67497	8.631577	0.749278	1	
10	35483	12.273169	1.508053	3	

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style></style>
        <nav class="navbar navbar-expand">
          <main class="container">
            <todo-form>
              <style></style>
              <div class="card todo-form">
                <hr>
              </div>
            </todo-form>
            <todo-list ref="list">
              <style></style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">
                  <todo-task> == $0
                </todo-task>
                <todo-task ref="task-151717632">
                  <todo-task>
                </todo-task>
                <todo-task ref="task-151717632">
                  <todo-task>
                </todo-task>
                <todo-task ref="task-151717633">
                  <todo-task>
                </todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
  </body>
```



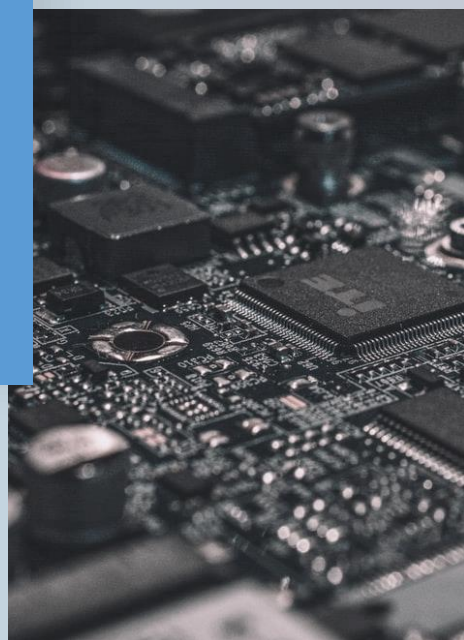
## 6.4.2 约会配对案例

### 5) 数据预处理

$$d_{3,4} = \sqrt[2]{(75136 - 26052)^2 + (13.147394 - 1.4441871)^2 + (0.428964 - 0.805124)^2}$$

从上述计算过程可见，不同样本之间相同特征的数值差对计算结果的影响呈现显著不均衡性。其中数值差距较大的特征，如“每年完成的飞行里程数”对计算结果起着决定性作用，而其他两个特征对计算结果几乎不产生任何影响。究其原因，在于“飞行里程数”数据与“消费冰淇淋公升数”、“玩乐消耗时间百分比”数据存在多个数量级的差距，致使“飞行里程数”数据主导了计算结果的变化趋势。因此，原本同等重要的三个特征对分类决策结果的影响呈现出不均衡性。

```
<div id="app">
  <todo-application>
    <#shadow-root (open)>
      <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
      <style>...</style>
      <nav class="navbar navbar-expand-...>
      <main class="container">
        <todo-form>
          <style>...</style>
          <div class="card todo-form">...</div>
        </todo-form>
        <hr>
        <todo-list ref="list">
          <style>...</style>
          <h2>Tasks:</h2>
          <ul ref="todos" class="list-group">
            <todo-task ref="task-151717615">...</todo-task>
            <todo-task ref="task-151717632">...</todo-task>
            <todo-task ref="task-151717632">...</todo-task>
            <todo-task ref="task-151717633">...</todo-task>
          </ul>
        </todo-list>
      </main>
    </todo-application>
  </div>
<script src="...">
```



## 6.4.2 约会配对案例

### 5) 数据预处理

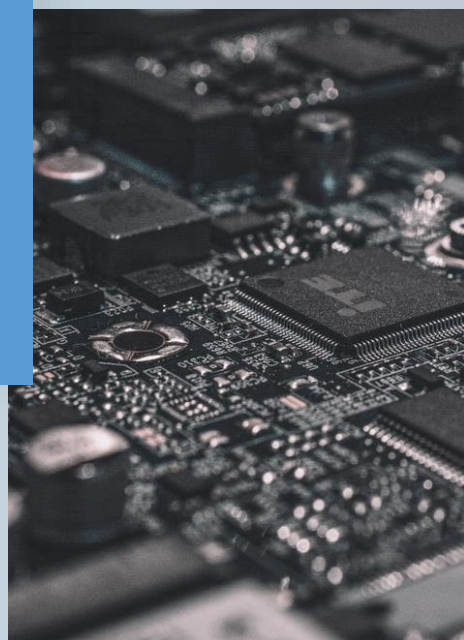
通常，为了消除不同量纲和量纲单位对数据分析结果的影响，需要进行数据归一化（数据标准化）处理，使各数据处于同一数量级，以解决数据之间的可比性，实现数据的综合对比评价。本案例为了处理取值范围不同的特征值对计算结果的影响，采用min-max标准化方法将取值范围转换至[0,1]范围内。

min-max标准化，也称为离差标准化，是对原始数据的线性变换，使结果值映射到[0,1]范围内。转换函数如下：

$$x^* = \frac{x - \min}{\max - \min} \quad (\text{公式6-2})$$

其中max为样本数据的最大值，min为样本数据的最小值。此方法可能的缺陷是当新数据加入时，也许导致max和min取值的变化，但是本案例不存在此顾虑。因此这里编写自定义函数autoNorm()，自动地将特征数据的取值转换到[0,1]区间。

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717619">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-1517176329">...</todo-task>
                <todo-task ref="task-1517176334">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```






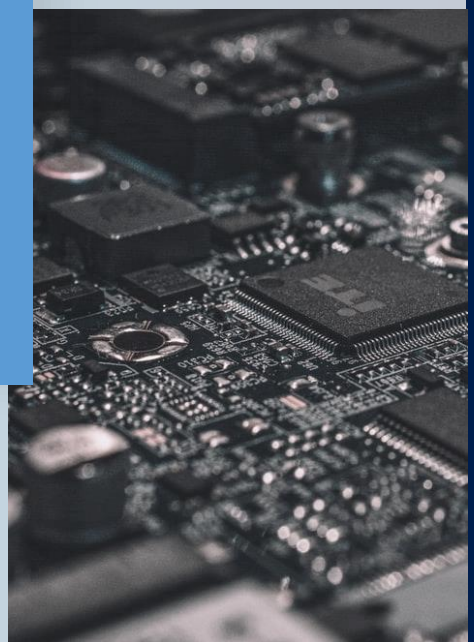
## 6.4.2 约会配对案例

### 5) 数据预处理

```
1 def autoNorm(dataSet):
2     minVals = dataSet.min(0) #axis=0 按列统计，跨行统计，所有列的最小值
3     maxVals = dataSet.max(0)
4     ranges = maxVals - minVals
5     normDataSet = zeros(shape(dataSet))
6     m = dataSet.shape[0]
7     normDataSet = dataSet - tile(minVals, (m,1)) # tile函数将minVals视为一
8 个整体，复制(m,1)遍，生成与输入矩阵相同维度的数组
9     normDataSet = normDataSet/tile(ranges, (m,1)) #元素除操作
10    return normDataSet, ranges, minVals
```



```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
            <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717619">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-1517176329">...</todo-task>
              <todo-task ref="task-1517176334">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </todo-application>
    </div>
    <script src="...">
```



## 6.4.2 约会配对案例

### 5) 数据预处理

函数autoNorm()中，输入变量接收样本特征矩阵dataSet，变量minVals保存列最小值，变量maxVals保存列最大值。

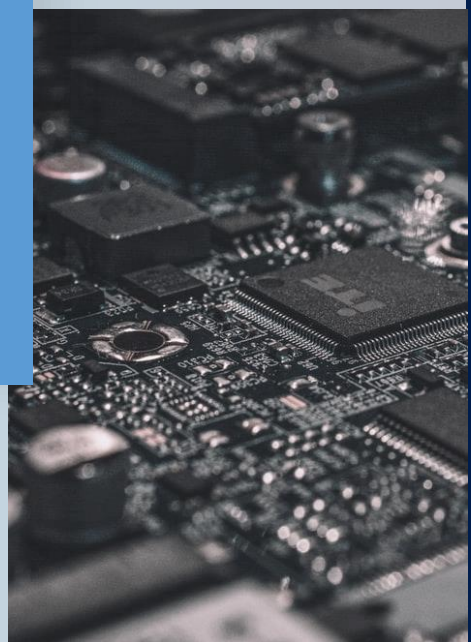
代码2~3行函数min()和max()中的参数取值为0表示从列中选取最值；

代码第4行生成函数计算的取值范围，用于获取公式6-2的分母部分，并创建返回矩阵；

代码第5行创建元素全零的输出矩阵normDataSet，用于存放公式6-2的计算结果；

代码第6行获取特征矩阵的行数。

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
            <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717619">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-1517176329">...</todo-task>
              <todo-task ref="task-1517176334">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </todo-application>
    </div>
    <script src="...">
```



## 6.4.2 约会配对案例

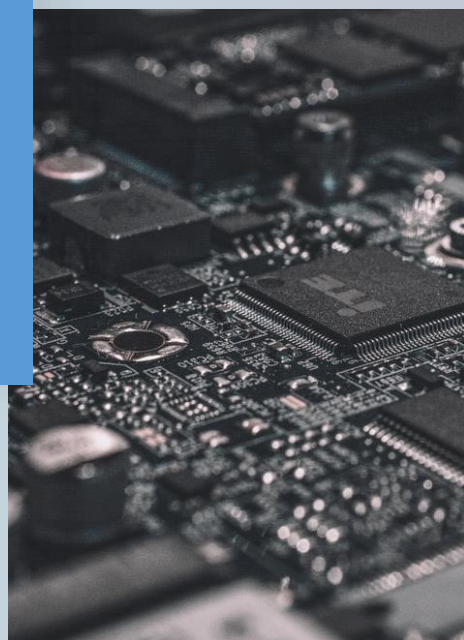
### 5) 数据预处理

准备工作完成之后，代码7~9行依据公式6-2完成数据归一化计算：用特征矩阵的当前值减去最小值，再除以取值范围。

需要注意的是，特征矩阵有10013个数值，而变量minVals和range均为13向量。为解决维度不一致数据间的操作问题，本案例使用numpy库的tile()函数分别将变量minVals和range复制(m,1)遍，生成与输入矩阵dataSet维度相同的新矩阵，然后进行元素级别的特征值相除。

最后，函数返回归一化之后的特征矩阵normDataSet、取值范围range和特征矩阵列最小值minVals。

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>_</style>
        <nav class="navbar navbar-expand">
          <main class="container">
            <todo-form>
              <style>_</style>
              <div class="card todo-form">_</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>_</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-15171761">_</todo-task>
                <todo-task ref="task-151717632">_</todo-task>
                <todo-task ref="task-151717632">_</todo-task>
                <todo-task ref="task-151717633">_</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="https://unpkg.com/vue@2.6.10/dist/vue.min.js">
  </script>
</body>
```





## 6)测试分类器性能

### 自定义函数

datingClassTest()用于测试分类器的性能。如果测试结果符合要求，就可以使用这个分类器筛选约会对象；否则，需要进一步优化分类器，以满足任务需求。

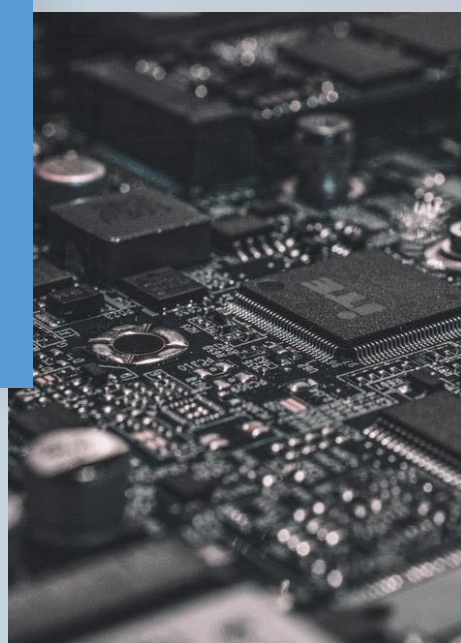
```
1 def datingClassTest():
2     TestRatio = 0.10      # 留出10%做测试数据
3     datingDataMat, datingLabels =
4     file2matrix('/home/aistudio/data/data76810/datingSet.txt')  #加载约会配对数据集
5     normMat, ranges, minVals = autoNorm(datingDataMat)
6     m = normMat.shape[0]
7     numTestVecs = int(m*TestRatio)
8     errorCount = 0.0
9     for i in range(numTestVecs):
10         classifierResult =
11         classify0(normMat[i,:], normMat[numTestVecs:m,:], datingLabels[numTestVecs:m], 3)
12         print ("the classifier came back with: %d, the real answer is: %d" %
13         (classifierResult, datingLabels[i]))
14         if (classifierResult != datingLabels[i]):
15             errorCount += 1.0
16     print ("the total error rate is: %f" % (errorCount/float(numTestVecs)))
17     print ("the total error number is:", errorCount)
```

## 6.4.2 约会配对案例

### 6) 测试分类器性能

函数`datingClassTest()`首先使用自定义函数`file2matrix`加载完整数据集，使用函数`autoNorm`进行特征数据归一化，获取数据集行数。代码9~15行依次提取测试集中的每一行数据，使用分类器函数`classify0`完成测试数据的类别预测。具体过程是：函数`classify0`读取当前第`i`个测试数据，指定将数据集的前90%作为训练数据，将数据集的后10%作为测试数据，执行 $k=3$ 的KNN算法；依据公式6-1计算得到与当前测试数据最为相似的前三个训练数据；查看这三个训练数据的类别标签，取三个类别标签中出现频率最高的标签值作为当前测试数据的类别预测值并保存于变量`classifierResult`。然后，查看当前测试数据的真实标签值`datingLabels[i]`，如果与类别预测值`classifierResult`相同，表示分类成功；否则，代表分类错误，变量`errorCount`取值加1，旨在统计错误分类的测试样本数目。最后，代码16-17行分别输出此次分类错误的测试样本总数和分类错误率，后者为分类错误总数除以测试样本总个数。

```
<body>
  <div id="app">
    <todo-application>
      <#shadow-root (open)>
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717615">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-151717633">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
    <script src="...">
```

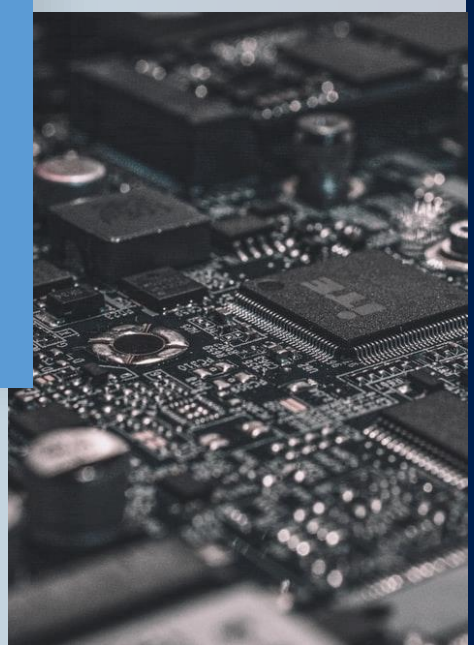
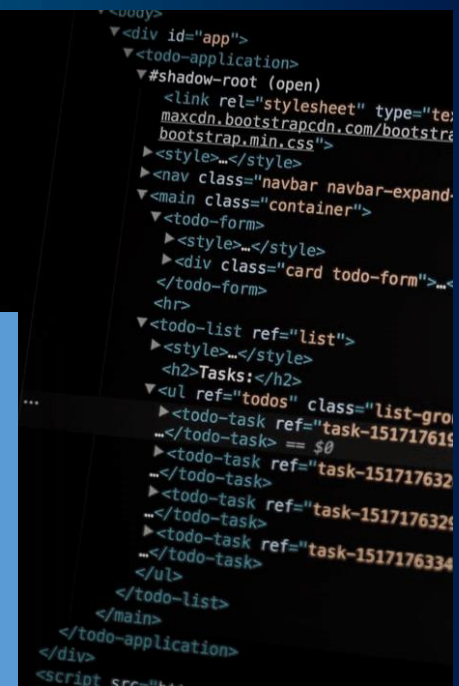


## 6.4.2 约会配对案例

### 6) 测试分类器性能

运行函数`datingClassTest()`后，部分输出结果如下：

```
out[]:... ..  
the classifier came back with: 1, the real answer is: 1  
the classifier came back with: 2, the real answer is: 2  
the classifier came back with: 1, the real answer is: 1  
the classifier came back with: 3, the real answer is: 3  
the classifier came back with: 3, the real answer is: 3  
the classifier came back with: 2, the real answer is: 2  
the classifier came back with: 1, the real answer is: 1  
the classifier came back with: 3, the real answer is: 1  
the total error rate is: 0.050000  
the total error number is: 5.0
```

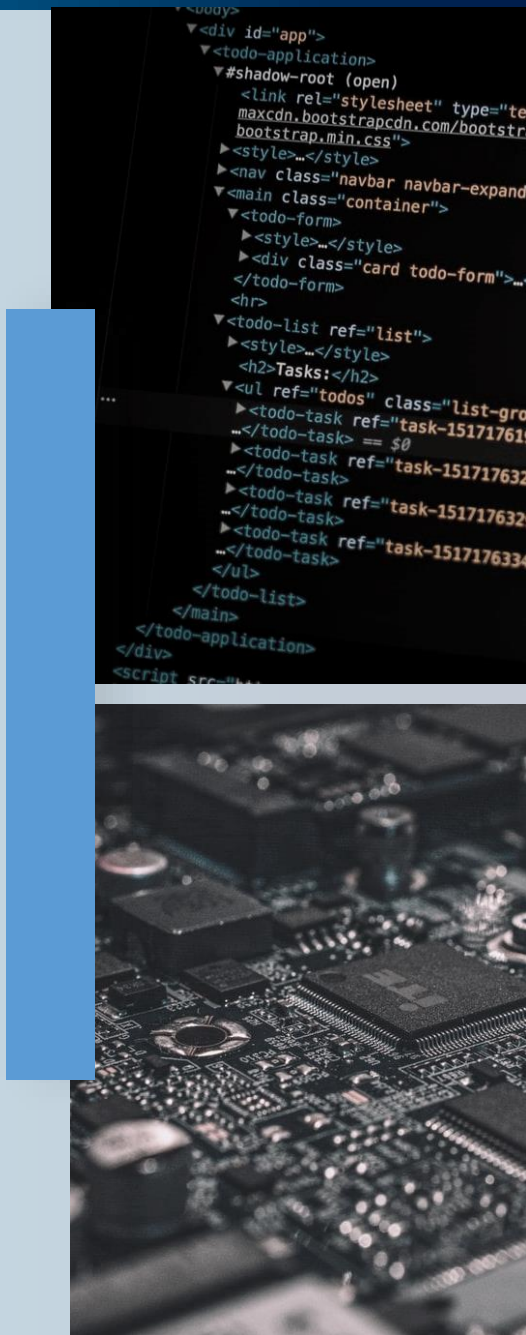


## 6.4.2 约会配对案例

### 6) 测试分类器性能

分类器处理约会配对数据集的错误率为5%，这是一个不错的结果。用户也可以尝试修改变量TestRatio和k的取值，检测分类错误率是否随着变量取值的变化而增加。参数取值不同，分类器的输出结果可能出现较大波动。需要注意的是，参数k一般为奇数，为的是便于依据少数服从多数的原则根据k个训练数据的类别确定测试数据的类别标签。

经过测试，分类器正确率满足本案例约会对象筛选要求。下面可以输入潜在约会对象的特征数据，由分类器帮助判定这一约会对象的可交往程度。

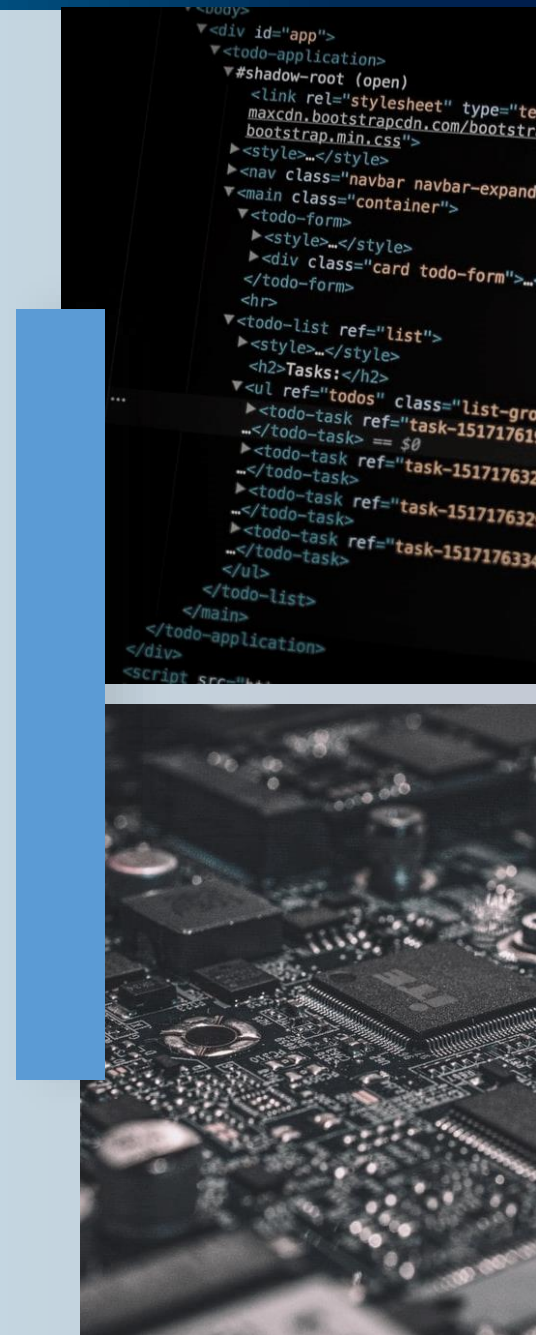




## 6.4.2 约会配对案例

### 7) 分类器应用：约会对象筛选

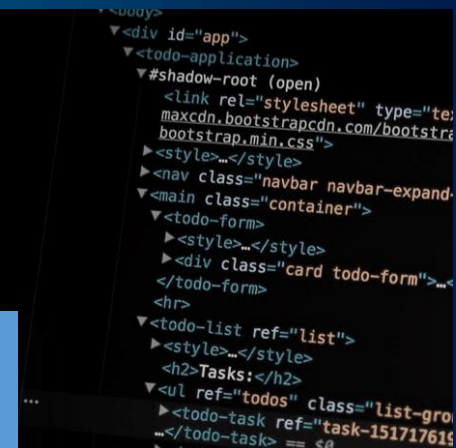
自定义函数classifyPerson使用经过训练和测试的KNN分类器，为用户筛选约会对象。具体筛选过程是：用户根据提示信息输入待筛选对象的特征数据，程序调用KNN分类器预测用户对该约会对象的喜欢程度。



## 6.4.2 约会配对案例

### 7) 分类器应用：约会对象筛选

```
1 def classifyPerson():
2     resultList=[ '一点也不喜欢', '有点喜欢', '比较喜欢' ]
3     percentTats=float(input("玩游戏、看视频消耗时间百分比: "))
4     ffMiles=float(input("每年飞行的里程数: "))
5     iceCream=float(input("每周消费冰淇淋的公升数: "))
6
7     datingDataMat,datingLabels=file2matrix('/home/aistudio/data/data76810/datingSet.txt')
8     norMat,ranges,minVals=autoNorm(datingDataMat)
9     inArr=array([ffMiles,percentTats,iceCream])
10    classifierResult=classify0((inArr-minVals)/ranges,norMat,datingLabels,3)
11    print("\n你对约会对象可能的喜欢程度:",resultList[classifierResult-1])
```





## 6.4.2 约会配对案例

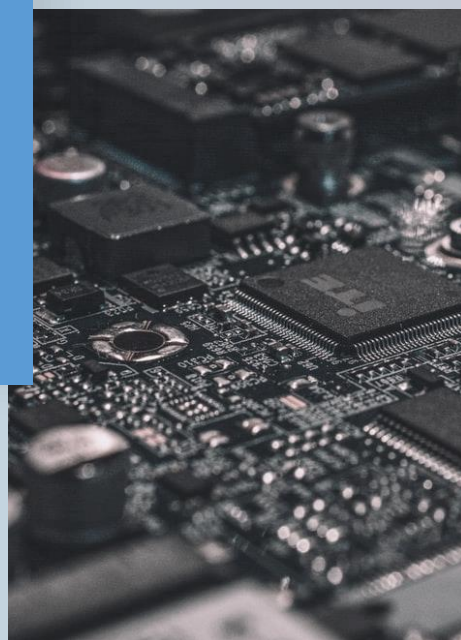
### 7) 分类器应用：约会对象筛选

代码2~5行使用input()函数接收字符串类型的特征数据，通过float()函数将输入数据转换为浮点型。然后依次调用file2matrix函数对文本文件进行数据解析，调用autoNorm函数进行数据归一化；

代码第8行将输入的三个特征数据转换为ndarray数组；至此，获取了代码第9行classify0函数所需的全部形式参数，于是调用KNN分类器完成约会对象类别标签分类预测任务。

最后，代码第10行返回分类预测结果。

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
        <main class="container">
          <todo-form>
            <style>...</style>
            <div class="card todo-form">...</div>
            <hr>
          <todo-list ref="list">
            <style>...</style>
            <h2>Tasks:</h2>
            <ul ref="todos" class="list-group">
              <todo-task ref="task-151717619">...</todo-task>
              <todo-task ref="task-151717632">...</todo-task>
              <todo-task ref="task-1517176329">...</todo-task>
              <todo-task ref="task-1517176334">...</todo-task>
            </ul>
          </todo-list>
        </main>
      </todo-application>
    </div>
    <script src="...">
```



## 6.4.2 约会配对案例

### 7) 分类器应用：约会对象筛选

这里，用户依次输入特征数据“30”、“300”和“20”，此函数运行结果如下：

out [] :玩游戏、看视频消耗时间百分比：30

每年飞行的里程数：300

每周消费冰淇淋的公升数：20

你对约会对象可能的喜欢程度： 比较喜欢

```
<body>
  <div id="app">
    <todo-application>
      #shadow-root (open)
        <link rel="stylesheet" type="text/css" href="maxcdn.bootstrapcdn.com/bootstrap.min.css">
        <style>...</style>
        <nav class="navbar navbar-expand-...>
          <main class="container">
            <todo-form>
              <style>...</style>
              <div class="card todo-form">...</div>
            </todo-form>
            <hr>
            <todo-list ref="list">
              <style>...</style>
              <h2>Tasks:</h2>
              <ul ref="todos" class="list-group">
                <todo-task ref="task-151717619">...</todo-task>
                <todo-task ref="task-151717632">...</todo-task>
                <todo-task ref="task-1517176329">...</todo-task>
                <todo-task ref="task-1517176334">...</todo-task>
              </ul>
            </todo-list>
          </main>
        </todo-application>
      </div>
      <script src="...">
```

