

第4讲 科学计算库

■ NumPy

■ Matplotlib

■ Pandas

Fundamentals of Machine Learning_wangbianqin

科学计算库

- **Numpy** : <http://www.numpy.org/>

科学计算的基础库之一，数值计算的基础库，创建多维数组，以及多维数据的各种运算。

- **Matplotlib** : <https://matplotlib.org/>

科学绘图库，其功能为数据的可视化，依赖**Numpy**和**Scipy**两个库

- **Pandas** : <http://pandas.pydata.org/>

数据分析库：建立在**Numpy**之上，用于数据分析处理

NumPy

- ❑ 导入 : `import numpy as np`
- ❑ 多维数组(`ndarray`) : 由同种类型的元素构成, 简称“数组”
- ❑ 数组索引(`index`) : 整数索引, 序号从0开始
- ❑ 轴(`axes`) : 数组的每一个维度
“ In NumPy dimensions are called axes”
- ❑ 秩(`rank`) : 轴的数量, 或者维度的数量, 一个标量

NumPy

◆ 示例：生成数组np.array([list]/(tuple))

```
import numpy as np
```

```
# 创建一维数组
```

```
my_list = [1, 2, 3]
```

```
x = np.array(my_list)
```

```
print('list: ', my_list)
```

```
print('array: ', x, 'type:', type(x))
```

```
list: [1, 2, 3]
```

```
array: [1 2 3] type: <class 'numpy.ndarray'>
```

```
import numpy as np
```

```
# 创建二维数组
```

```
y = np.array([[1, 2, 3], [4, 5, 6]])
```

```
print('y:', y)
```

```
print('type:', type(y))
```

```
y: [[1 2 3]
```

```
 [4 5 6]]
```

```
type: <class 'numpy.ndarray'>
```

NumPy

◆ 示例：ndarray对象属性

```
# 查看ndarray的属性
print('x: ', x)
print('ndim: ', x.ndim)
print('shape: ', x.shape)
print('size: ', x.size)
print('dtype: ', x.dtype)
print('itemsize: ', x.itemsize)
```

```
x: [1 2 3]
ndim: 1
shape: (3,)
size: 3
dtype: int32
itemsize: 4
```

```
# 查看ndarray的属性
print('y: ', y)
print('ndim: ', y.ndim)
print('shape: ', y.shape)
print('size: ', y.size)
print('dtype: ', y.dtype)
print('itemsize: ', y.itemsize)
```

```
y: [[1 2 3]
     [4 5 6]]
ndim: 2
shape: (2, 3)
size: 6
dtype: int32
itemsize: 4
```

NumPy

□ 创建数组时可指定数据类型：dtype

```
# 创建一维数组
x = np.array([1, 2, 3], dtype=np.int32)
print('array: ', x, 'type: ', type(x), 'itemsize: ', x.itemsize)

array: [1 2 3] type: <class 'numpy.ndarray'> itemsize: 4
```

使用列表或元组创建数组时，所创建的数组元素类型由原来的元素类型决定

NumPy

□ 创建序列数组：

np.arange(x, y, i)：创一个由**x**到**y**，以**i**为步长的数字序列数组

- 前闭后开：数字序列中不包括结束数字
- 起始数字省略时，默认从**0**开始
- 步长省略时，默认为**1**

```
m = np.arange(3)
n = np.arange(0, 6, 2)
print('m =', m)
print('n =', n)
```

```
m = [0 1 2]
n = [0 2 4]
```

NumPy

□ 创建等差数列 : `np.linspace(start, stop, num, dtype)`

```
import numpy as np
a = np.linspace(0, 10, num = 3)
print(a)
```

```
[ 0.  5. 10.]
```

□ 创建一个等比数列 : `np.logspace(start, stop, num, base, dtype)`

```
import numpy as np
b = np.logspace(1, 7, 4, base=2)
print(b)
```

```
[ 2.  8. 32. 128.]
```


NumPy

- 创建全1数组：**np.ones()**
- 创建单位矩阵：**np.eye()**
- 创建全0数组：**np.zeros()**
- 创建对角矩阵：**np.diag()**

```
m = np.ones((3, 2))
n = np.zeros((3, 2))
print('m =', m)
print('n =', n)
```

```
m = [[1. 1.]
      [1. 1.]
      [1. 1.]]
n = [[0. 0.]
      [0. 0.]
      [0. 0.]
```

```
m = np.eye(3)
n = np.diag([2, 2, 4])
print('m =', m)
print('n =', n)
```

```
m = [[1. 0. 0.]
      [0. 1. 0.]
      [0. 0. 1.]]
n = [[2 0 0]
      [0 2 0]
      [0 0 4]]
```

NumPy

□ 数组索引：两种方法

一维张量

a[1]

二维张量

b[1][1]

b[1, 1]

三维张量

c[1][1][1]

c[1, 1, 1]

Fundamentals of Machine Learning_wangbianqin

NumPy

◆ 示例：数组的两种索引

```
x = np.array([1, 3, 5, 7, 8, 10])
print('x:', x)
print('x.shape: ', x.shape)
print('x[0]:', x[0])
```

```
x: [ 1  3  5  7  8 10]
x.shape: (6,)
x[0]: 1
```

```
import numpy as np
y = np.array([[1, 2, 3], [4, 5, 6]])
print('y.shape: ', y.shape)
print('y[0][0]= ', y[0][0])
print('y[0,0]= ', y[0,0])
print('y[0][1]= ', y[0][1])
print('y[0,1]= ', y[0,1])
```

```
y.shape: (2, 3)
y[0][0]= 1
y[0,0]= 1
y[0][1]= 2
y[0,1]= 2
```

```
z = np.array([[[1, 2, 3], [4, 5, 6]], [[2, 2, 3], [8, 4, 2]]])
print('z.shape: ', z.shape)
print('z[0][0]= ', z[0][0])
print('z[0,0]= ', z[0, 0])
print('z[0][0][0]= ', z[0][0][0])
print('z[0,0,0]= ', z[0,0,0])
```

```
z.shape: (2, 2, 3)
z[0][0]= [1 2 3]
z[0,0]= [1 2 3]
z[0][0][0]= 1
z[0,0,0]= 1
```

NumPy

□ 数组切片(start: end: step)

- 起始位置：结束位置，前闭后开，切片中不包含结束位置
- 起始位置、结束位置、步长都可以省略
- 步长是负数，表示逆序索引，起始位置的索引号应大于结束位置

NumPy

◆ 示例：一维数组切片

```
import numpy as np
x = np.array([10, 20, 30, 40, 50, 60])
print("x[:]=", x[:])
print("x[:,]= ", x[:,])
print("x[::-1]= ", x[::-1])
print("x[::2]= ", x[::2])
print("x[0::2]= ", x[0::2])
print("x[1::2]= ", x[1::2])
```

```
x[:] = [10 20 30 40 50 60]
x[:,] = [10 20 30 40 50 60]
x[::-1] = [60 50 40 30 20 10]
x[::2] = [10 30 50]
x[0::2] = [10 30 50]
x[1::2] = [20 40 60]
```

数组切片得到的是原始数组的视图，所有修改都会直接反映到源数组

NumPy

◆ 示例：二维数组切片

```
# 二维数组的切片
X = np.array([[10,20,30],[40,50,60]])
print("X=", X)
# 访问二维数组某行和某列:
print("X[0]=", X[0])
print("X[1]=", X[1])
print("X[:,0]=", X[:,0])
print("X[:,2]=", X[:,2])
print("X[:,0:2]=", X[:,0:2])
print("X[:,1:2]=", X[:,1:2])

print("X[:,-1]=", X[:,-1])
print("X[:, :-1]=", X[:, :-1])
print("X[[0,1],:]=", X[[0,1],:])
print("X[:, [0,2]]=", X[:, [0,2]])
print("X[[0,1],[0,1]]=", X[[0,1],[0,1]])
```

第1行
第2行
第1列
第3列
奇数列
偶数列
最后1列
除最后1列之前的所有列
某几行
某几列
某几列某几行上的元素

```
X= [[10 20 30]
     [40 50 60]]
X[0]= [10 20 30]
X[1]= [40 50 60]
X[:,0]= [10 40]
X[:,2]= [30 60]
X[:,0:2]= [[10 30]
           [40 60]]
X[:,1:2]= [[20]
           [50]]
X[:,-1]= [30 60]
X[:, :-1]= [[10 20]
            [40 50]]
X[[0,1],:] = [[10 20 30]
              [40 50 60]]
X[:, [0,2]] = [[10 30]
               [40 60]]
X[[0,1],[0,1]] = [10 50]
```

NumPy

□ 维度变换

- **ndarray.reshape(shape)** : 不改变原数组, 依赖**shape**生成新数组
- **ndarray.resize(shape)** : 直接修改原数组, 为**shape**形状
- **ndarray.flatten()** : 依赖原数组, 生成新的一维数组
- **ndarray.ravel()** : 将原数组展平为一维数组

```
# 创建二维数组
m = np.array([[1, 2, 3], [4, 5, 6]])
print('m =', m)
# 二维数组变形
n = m.reshape(3, 2)
print('n =', n)
q = m.resize(3, 2)
print('q =', q)
print('m =', m)
p = m.flatten()
print('p =', p)
r = m.ravel()
print('r =', r)
```

```
m = [[1 2 3]
      [4 5 6]]
n = [[1 2]
      [3 4]
      [5 6]]
q = None
m = [[1 2]
      [3 4]
      [5 6]]
p = [1 2 3 4 5 6]
r = [1 2 3 4 5 6]
```

NumPy

□ **reshape()用法** : **reshape(c, -1)**、**reshape(-1, c)**

-1: 自动计算行数或列数

d = 数组中所有的元素个数/**c** (**d**必须是整数, 不然报错)

- **reshape(1,-1)**转化成**1**行
- **reshape(2,-1)**转换成**2**行
- **reshape(-1,1)**转换成**1**列
- **reshape(-1,2)**转化成**2**列

NumPy

◆ 示例：创建数组并且改变数组形状

```
m = np.arange(12).reshape(3, 4)
n = np.arange(12).reshape(3, 4)
print('m =', m)
print('n =', n)
```

```
m = [[ 0  1  2  3]
      [ 4  5  6  7]
      [ 8  9 10 11]]
n = [[ 0  1  2  3]
      [ 4  5  6  7]
      [ 8  9 10 11]]
```

NumPy

□ 随机函数模块: **numpy.random**

- 简单随机数：产生简单的随机数据，可以是任何维度
- 排列：将所给对象随机排列
- 分布：产生指定分布的数据，如高斯分布等
- 生成器：种随机数种子，根据同一种子产生的随机数是相同的

NumPy

□ `np.random.rand(d0,d1,...,dn)`

[0,1)区间均匀分布的数组，元素为浮点数

```
# 生成标量
```

```
a1 = np.random.rand()  
print('a1=', a1)
```

```
a1= 0.4564906241221083
```

```
# 生成一维数组
```

```
a2 = np.random.rand(1)  
print('a2=', a2)
```

```
a2= [0.56697356]
```

```
# 生成二维数组
```

```
a3 = np.random.rand(1,2)  
print('a3=', a3)
```

```
a3= [[0.43143001 0.13701779]]
```

□ `np.random.randn(d0,d1,...,dn)`

标准正态分布的数组，元素为浮点数

```
# 生成标量
```

```
b1 = np.random.randn()  
print('b1=', b1)
```

```
b1= -0.6789634821291358
```

```
# 生成一维数组
```

```
b2 = np.random.randn(1)  
print('b2=', b2)
```

```
b2= [0.41955839]
```

```
# 生成二维数组
```

```
b3 = np.random.randn(1,2)  
print('b3=', b3)
```

```
b3= [[-0.31349054 1.26916352]]
```

NumPy

- 生成随机整数数组：
`np.random.randint(low, high, size)` :

```
c1 = np.random.randint(2, size=10)
c2 = np.random.randint(2, 3, (2, 3))
print('c1=', c1)
print('c2=', c2)
```

```
c1= [1 0 0 0 1 0 0 1 0 0]
c2= [[2 2 2]
     [2 2 2]]
```

NumPy

□ np.seed(s) : 随机数种子，s是给定的种子值

设置随机种子之后，产生的随机数组不变

```
np.random.seed(10)
```

创建一个2行3列的标准正态分布数组

```
d1 = np.random.randn(2,3)
```

```
np.random.seed(10)
```

```
d2 = np.random.randn(2,3)
```

```
d3 = np.random.randn(2,3)
```

```
print('d1=', d1)
```

```
print('d2=', d2)
```

```
print('d3=', d3)
```

```
d1= [[ 1.3315865  0.71527897 -1.54540029]
      [-0.00838385  0.62133597 -0.72008556]]
```

```
d2= [[ 1.3315865  0.71527897 -1.54540029]
      [-0.00838385  0.62133597 -0.72008556]]
```

```
d3= [[ 0.26551159  0.10854853  0.00429143]
      [-0.17460021  0.43302619  1.20303737]]
```

NumPy

□ 根据数组的第1维打乱排序：np.random.shuffle()

```
e1 = np.arange(12)
e2 = np.arange(12).reshape(3,4)
print('e1=',e1)
print('e2=',e2)

e1= [ 0  1  2  3  4  5  6  7  8  9 10 11]
e2= [[ 0  1  2  3]
     [ 4  5  6  7]
     [ 8  9 10 11]]

np.random.shuffle(e1)
np.random.shuffle(e2)
print('e1=',e1)
print('e2=',e2)

e1= [ 4  3  8  0  6  5  2 11 10  9  1  7]
e2= [[ 8  9 10 11]
     [ 4  5  6  7]
     [ 0  1  2  3]]
```

np.random.permutation()：同上，但不改变原数组生成新数组

NumPy

- **np.random.choice(a[, size, replace, p])** : 从一维数组**a**中以概率**p**抽取元素，形成**size**形状新数组，**replace**表示是否可以重用元素，默认为**True**

```
a = np.array([1, 2, 3, 4, 5, 6])
np.random.choice(a, (3, 2))

array([[1, 5],
       [4, 1],
       [5, 4]])

np.random.choice(a, (3, 2), replace=False)

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

# p是随机概率，出现几率与数字大小成正比
np.random.choice(a, (3, 2), p=a/np.sum(a))

array([[6, 4],
       [6, 6],
       [5, 2]])
```

NumPy

◆ 示例：产生常用分布

```
# uniform(low,high,size): 产生均匀分布数组, low起始值, high结束值, size形状
u = np.random.uniform(0,10, (2,3))
print(u)
```

```
[[3.7334076  6.74133615 4.41833174]
 [4.34013993 6.17766978 5.13138243]]
```

```
# normal(loc,scale,size): 产生正态分布数组, loc均值, scale标准差, size形状
n = np.random.normal(10,5, (2,3))
print(n)
```

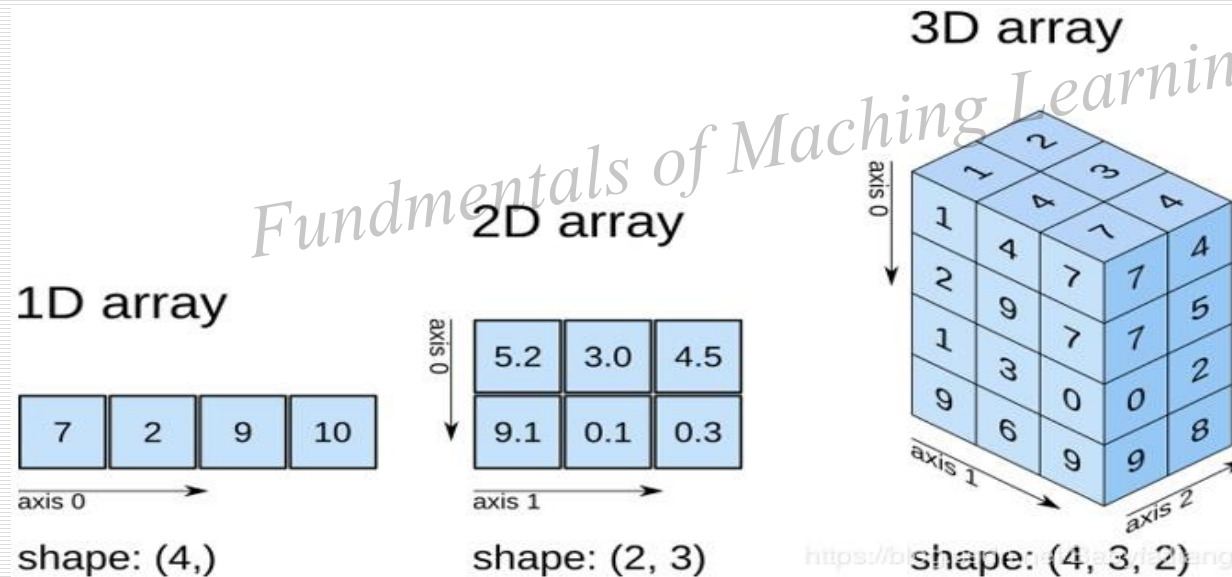
```
[[15.61845627 18.36311107 10.49574608]
 [16.98998189  8.64376006 13.06602092]]
```

```
# poisson(lam,size): 产生泊松分布数组, lam随机事件发生概率, size形状
p = np.random.poisson(0.2, (3,4))
print(p)
```

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


NumPy

□ ndarray对象的axis编号



NumPy

□ 两个数组之间的加、减、乘除运算 要求数组的形状和长度应该一致

```
m = np.ones((3,3))
n = np.arange(9).reshape(3,3)
```

数组间对应元素的运算

```
print('m: \n', m)
print('n: \n', n)
print('m + n = \n', m + n)
print('m - n = \n', m - n)
print('m * n = \n', m * n)
```

m:

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

n:

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

m + n =

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

m - n =

```
[[ 1.  0. -1.]
 [-2. -3. -4.]
 [-5. -6. -7.]]
```

m * n =

```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
```

```
m = np.ones((3,3))
n = np.arange(9).reshape(3,3)
```

数组间对应元素的运算

```
print('m: \n', m)
print('n: \n', n)
print('np.add(m,n)= \n', np.add(m,n))
print('np.subtract(m,n)= \n', np.subtract(m,n))
print('np.multiply(m,n)= \n', np.multiply(m,n))
```

m:

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

n:

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

np.add(m,n)=

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
```

np.subtract(m,n)=

```
[[ 1.  0. -1.]
 [-2. -3. -4.]
 [-5. -6. -7.]]
```

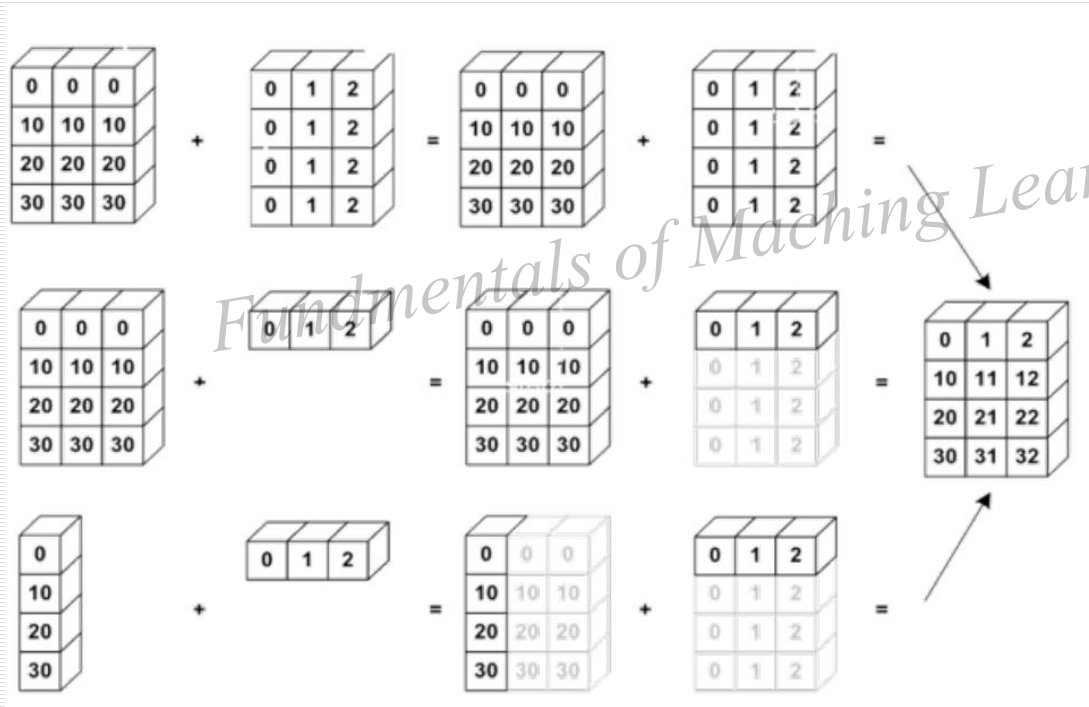
np.multiply(m,n)=

```
[[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
```

函数	
np.add(x1, x2 [, y])	y = x1 + x2
np.subtract(x1, x2 [, y])	y = x1 - x2
np.multiply(x1, x2 [, y])	y = x1 * x2
np.divide(x1, x2 [, y])	y = x1 / x2
np.floor_divide(x1, x2 [, y])	y = x1 // x2, 返回值取整
np.negative(x [, y])	y = -x
np.power(x1, x2 [, y])	y = x1**x2
np.remainder(x1, x2 [, y])	y = x1 % x2

NumPy

□ 广播机制(broadcasting)



NumPy

◆ 示例：一维数组可以和多维数组相加，相加时会将一维数组扩展至多维

```
m = np.array([0,3,2])
n = np.array([[1,0,1],[1,1,1],[2,0,1]])
print('np.add(m,n)= \n', np.add(m,n))

np.add(m,n)=
[[1 3 3]
 [1 4 3]
 [2 3 3]]
```

- 数组之间的减法、乘法、除法运算，和加法运算规则相同
- 两个数组中元素的数据类型不同时，精度低的数据类型，会自动转换为精度更高的数据类型，然后再进行运算

NumPy

□ 矩阵相乘：

按照矩阵相乘的规则运算

```
A = np.array([[0, 1], [1, 2]])
B = np.array([[2, 1], [1, 1]])
C = np.array([0, 2, 1])
D = np.array([1, 0, 3])
print('np.matmul(A, B)= \n', np.matmul(A, B))
print('np.dot(A, B)= \n', np.dot(A, B))
print('np.dot(C, D)= \n', np.dot(C, D))
```

```
np.matmul(A, B) =
[[1 1]
 [4 3]]
np.dot(A, B)=
[[1 1]
 [4 3]]
np.dot(C, D)=
3
```

$$X \cdot Y = \sum_{i=1}^n x_i y_i$$

□ 乘号运算符：

矩阵中对应的元素分别相乘

```
A = np.array([[0, 1], [1, 2]])
B = np.array([[2, 1], [1, 1]])
print('A*B = \n', A*B)
print('np.multiply(A,B)= \n', np.multiply(A,B))
```

```
A*B =
[[0 1]
 [1 2]]
np.multiply(A,B)=
[[0 1]
 [1 2]]
```

NumPy

□ 矩阵转置：np.transpose()

```
A = np.array([[0,1],[1,2]])  
B = np.array([[2,1],[1,1]])  
print('np.transpose(A)= \n', np.transpose(A))  
print('np.transpose(B)= \n', np.transpose(B))
```

```
np.transpose(A)=  
[[0 1]  
 [1 2]]  
np.transpose(B)=  
[[2 1]  
 [1 1]]
```

NumPy

□ 矩阵求逆：np.linalg.inv()

```
A = np.array([[0,1],[1,2]])  
B = np.array([[2,1],[1,1]])  
print('np.linalg.inv(A)= \n', np.linalg.inv(A))  
print('np.linalg.inv(B)= \n', np.linalg.inv(B))
```

```
np.linalg.inv(A)=  
[[-2.  1.]  
 [ 1.  0.]]  
np.linalg.inv(B)=  
[[ 1. -1.]  
 [-1.  2.]]
```

NumPy

□ 数组元素间的运算

- **np.abs(a)** : 取各元素的绝对值 ; **np.sqrt(a)** : 计算各元素的平方根
- **np.square(a)** : 计算各元素的平方
- **np.log(a)** : 计算各元素的自然对数
- **np.log2(a)**、**np.log10(a)** : 分别计算各元素以2为底和以10为底的对数
- **np.ceil(a)** : 各元素向上取整 ; **np.floor(a)** : 各元素向下取整
- **np rint(a)** : 各元素四舍五入
- **np.exp(a)** : 计算各元素的指数值
- **np.sign(a)** : 计算各元素的符号值
- **np.mod(a,b)** : 元素级的模运算

NumPy

□ 统计类函数: 指定轴上的统计值计算

- `np.mean(x [, axis])` : 计算元素的平均值
- `np.sum(x [, axis])` : 计算元素的和
- `np.max(x [, axis])` : 返回元素的最大值
- `np.min(x [, axis])` : 返回元素的最小值
- `np.std(x [, axis])` : 计算元素的标准差
- `np.var(x [, axis])` : 计算元素的方差
- `np.argmax(x [, axis])` : 返回元素最大值的下标索引值
- `np.argmin(x [, axis])` : 返回元素最小值的下标索引值

NumPy

◆ 示例：计算一维数组的统计值

```
# 一维数组
a = np.array([-4, -2, 1, 3, 5])
print('sum:', a.sum())
print('min:', a.min())
print('max:', a.max())
print('mean:', a.mean())
print('var:', a.var())
print('std:', a.std())
print('argmax:', a.argmax())
print('argmin:', a.argmin())
```

```
sum: 3
min: -4
max: 5
mean: 0.6
var: 10.64
std: 3.2619012860600183
argmax: 4
argmin: 0
```

NumPy

◆ 示例：按维度计算统计值：

二维数组

```
m = np.array([[1, 2, 4], [2, 4, 6]])
print('sum:', m.sum(axis=1))
print('min:', m.min(axis=1))
print('max:', m.max(axis=1))
print('mean:', m.mean(axis=1))
print('var:', m.var(axis=1))
print('std:', m.std(axis=1))
print('argmax:', m.argmax(axis=1))
print('argmin:', m.argmin(axis=1))
```

```
sum: [ 7 12]
min: [1 2]
max: [4 6]
mean: [2.33333333 4.          ]
var: [1.55555556 2.66666667]
std: [1.24721913 1.63299316]
argmax: [2 2]
argmin: [0 0]
```

一维数组

```
a = np.array([-4, -2, 1, 3, 5])
print('sum:', a.sum())
print('min:', a.min())
print('max:', a.max())
print('mean:', a.mean())
print('var:', a.var())
print('std:', a.std())
print('argmax:', a.argmax())
print('argmin:', a.argmin())
```

```
sum: 3
min: -4
max: 5
mean: 0.6
var: 10.64
std: 3.2619012860600183
argmax: 4
argmin: 0
```

NumPy

□ 创建矩阵 : np.mat(a)

```
a = np.array([[1, 2, 0], [2, 1, 3]])
m = np.mat(a)
print('type(a):', type(a))
print('type(m):', type(m))
print('m.ndim:', m.ndim)
print('m.shape:', m.shape)
print('m.size:', m.size)
print('m.dtype:', m.dtype)
```

```
type(a): <class 'numpy.ndarray'>
type(m): <class 'numpy.matrix'>
m.ndim: 2
m.shape: (2, 3)
m.size: 6
m.dtype: int32
```

NumPy

□ 矩阵运算：矩阵相乘、矩阵转置、矩阵求逆

```
import numpy as np
```

```
x = np.matrix([[1,2,3], [4,5,6]])
```

```
y = np.matrix([1,2,3,4,5,6])
```

```
# x[1,1]返回行下标和列下标都为1的元素
```

```
# 注意，对于矩阵x来说，x[1,1]和x[1][1]的含义不一样
```

```
print(x, y, x[1,1], sep='\n\n')
```

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

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

```
5
```

```
A = np.mat([[0,1], [1,2]])
```

```
B = np.mat([[2,1], [1,1]])
```

```
print('A*B= \n', A*B)
```

```
print('A.T= \n', A.T)
```

```
print('A.I= \n', A.I)
```

```
print('A*A.I= \n', A*A.I)
```

```
A*B=
```

```
[[1 1]
```

```
 [4 3]]
```

```
A.T=
```

```
[[0 1]
```

```
 [1 2]]
```

```
A.I=
```

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

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

```
A*A.I=
```

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

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

```
A = np.matrix([[0, 1], [1, 2]])
```

```
B = np.matrix([[2, 1], [1, 1]])
```

```
print('A*B= \n', A*B)
```

```
print('A.T= \n', A.T)
```

```
print('A.I= \n', A.I)
```

```
print('A*A.I= \n', A*A.I)
```

```
A*B=
```

```
[[1 1]
```

```
 [4 3]]
```

```
A.T=
```

```
[[0 1]
```

```
 [1 2]]
```

```
A.I=
```

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

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

```
A*A.I=
```

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

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

NumPy

□ 特征值和特征向量

- 如果一个向量 v 是方阵 A 的特征向量，则一定可表示成

$$Av = \lambda v$$

λ 被称为特征向量 v 对应的特征值

- 特征值分解：方阵 A 矩阵分解成

$$A = Q \Sigma Q^{-1}$$

Q 是矩阵 A 的特征向量组成的矩阵，

Σ 是一个对角阵，每一个对角线上的元素是一个特征值

NumPy

◆ 示例：计算矩阵的特征值与特征向量

```
import numpy as np
A = np.array([[1,-3,3],[3,-5,3],[6,-6,4]])

# 特征值与特征向量
e, v = np.linalg.eig(A)
print('e=\n', e)
print('v=\n', v)

print(np.dot(A,v))          # 矩阵与特征向量的乘积
print(e*v)                  # 特征值与特征向量的乘积
print(np.isclose(np.dot(A,v), e*v)) # 验证二者是否相等

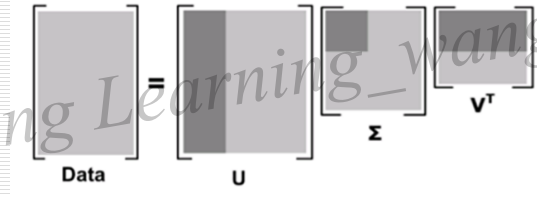
print(np.linalg.det(A-np.eye(3,3)*e))
```

NumPy

□ 奇异值分解(SVD)

对于任意矩阵A，可分解成三个矩阵：

$$A_{m \times n} = U_{m \times n} \Sigma_{m \times m} V_{n \times n}^T$$



计算 Σ 前 r 个奇异值的平方和占所有奇异值的平方和的比例，如果大于90%，选 r 个奇异值重构矩阵（剩余的数据代表的可能是噪声，无用数据）。重构的A'为：

$$A_{m \times n} \approx U_{m \times r} \Sigma_{r \times r} V_{r \times n}^T$$

$$r = \frac{\sum_{i=1}^r \sigma_i^2}{\sum_{i=1}^n \sigma_i^2} > 90\%$$

NumPy

◆ 示例：奇异值分解np.linalg.svd()

```
a = np.matrix([[1,2,3], [4,5,6], [7,8,9]])
u, s, v = np.linalg.svd(a)
print('u=\n', u)
print('s=\n', s)
print('v=\n', v)
```

```
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 1.47280825e-16]
v=
[[-0.47967118 -0.57236779 -0.66506441]
 [-0.77669099 -0.07568647  0.62531805]
 [ 0.40824829 -0.81649658  0.40824829]]
```





```
u*np.diag(s)*v
```

```
matrix([[1., 2., 3.],
        [4., 5., 6.],
        [7., 8., 9.]])
```

NumPy

◆ 示例：利用svd实现对图像的压缩



 a10	2019/3/6 9:07	PNG 图像	380 KB
 a50	2019/3/6 9:07	PNG 图像	548 KB
 a100	2019/3/6 9:07	PNG 图像	642 KB
 org	2019/3/6 9:07	PNG 图像	781 KB

<http://liao.cpython.org/scipy06/>

```
# 利用svd实现对图像的压缩
import scipy.misc
from scipy.linalg import svd
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np

img = scipy.misc.face()[:, :, 0]

print(img.shape, type(img))
img = np.matrix(img)
U, s, Vh = svd(img)
plt.gray()

plt.subplot(221, aspect='equal')
plt.title("original")
plt.imshow(img)
plt.imsave('org.png', img)

A = np.dot(U[:, 0:10], np.dot(np.diag(s[0:10]), Vh[0:10, :]))
plt.subplot(222, aspect='equal')
plt.title(":10")
plt.imshow(A)
plt.imsave('a10.png', A)

A = np.dot(U[:, 0:50], np.dot(np.diag(s[0:50]), Vh[0:50, :]))
plt.subplot(223, aspect='equal')
plt.title(":50")
plt.imshow(A)
plt.imsave('a50.png', A)

A = np.dot(U[:, 0:100], np.dot(np.diag(s[0:100]), Vh[0:100, :]))
plt.subplot(224, aspect='equal')
plt.title(":100")
plt.imshow(A)
plt.imsave('a100.png', A)
```

NumPy

□ 探索下列函数功能和用法

np.sort() 、 np.argsort()、 np.tile()、 np.hstack()、 np.vstack()

np.r_()、 np.c_()、 np.concatenate()、 np.newaxis()、 meshgrid()

□ 区别：np.multiply()、 np.dot()、 (*)

□ 区别：np.transpose方法、T属性

□ Numpy的广播机制

<https://www.jianshu.com/p/41815afbe333>

NumPy

□ Numpy入门与进阶

- **Numpy官网**

<http://www.numpy.org/>

- **Numpy快速入门**

<https://docs.scipy.org/doc/numpy/user/quickstart.html>

- **Numpy_100_exercises**

https://github.com/rougier/numpy-100/blob/master/100_Numpy_exercises.md

数据可视化

◆ 示例：Anscombe's quartet (安斯库姆四重奏)

统计学家弗朗西斯.安斯库姆(Francis Anscombe) 于1973年构造

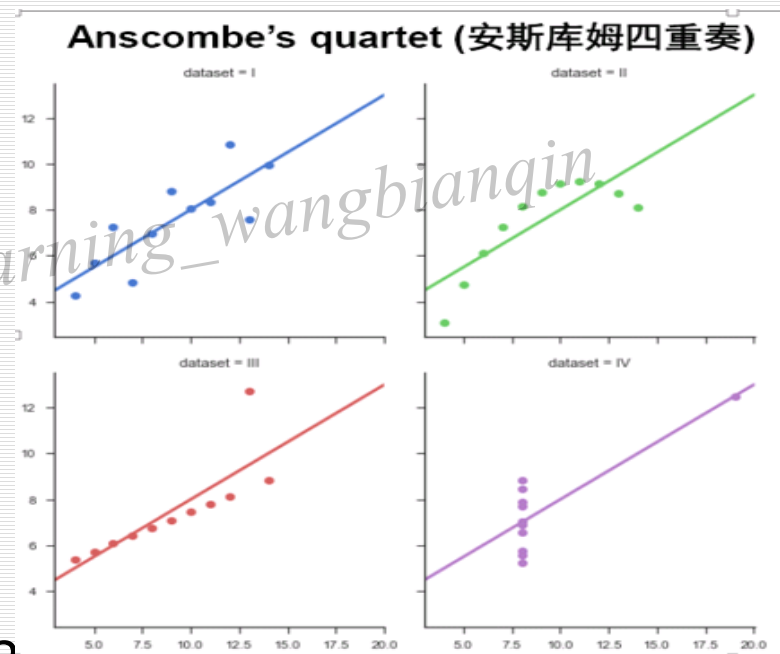
	dataset	x	y
0	I	10.0	8.04
1	I	8.0	6.95
2	I	13.0	7.58
3	I	9.0	8.81
4	I	11.0	8.33
5	I	14.0	9.96
6	I	6.0	7.24
7	I	4.0	4.26
8	I	12.0	10.84
9	I	7.0	4.82
10	I	5.0	5.68
11	II	10.0	9.14
12	II	8.0	8.14
13	II	13.0	8.74
14	II	9.0	8.77
15	II	11.0	9.26
16	II	14.0	8.10
17	II	6.0	6.13
18	II	4.0	3.10
19	II	12.0	9.13
20	II	7.0	7.26
21	II	5.0	4.74
22	III	10.0	7.46
23	III	8.0	6.77
24	III	13.0	12.74
25	III	9.0	7.11
26	III	11.0	7.81
27	III	14.0	8.84
28	III	6.0	6.08
29	III	4.0	5.39
30	III	12.0	8.15
31	III	7.0	6.42
32	III	5.0	5.73
33	IV	8.0	6.58
34	IV	8.0	5.76
35	IV	8.0	7.71
36	IV	8.0	8.84
37	IV	8.0	8.47
38	IV	8.0	7.04
39	IV	8.0	5.25
40	IV	19.0	12.50
41	IV	8.0	5.56
42	IV	8.0	7.91
43	IV	8.0	6.89

数据可视化

◆ 示例：四组数据的统计特性

	x		y	
	mean	var	mean	var
dataset				
I	9.0	11.0	7.500909	4.127269
II	9.0	11.0	7.500909	4.127629
III	9.0	11.0	7.500000	4.122620
IV	9.0	11.0	7.500909	4.123249

- **X**的平均数：9，**x**的方差：11
- **y**的平均数：7.50，**y**的方差：4.122或4.12，
- **X**与**y**的相关系数：0.816（精确到小数点后三位）
- 线性回归线： $y = 3.00 + 0.500x$ （分别精确到小数点后两位和三位）



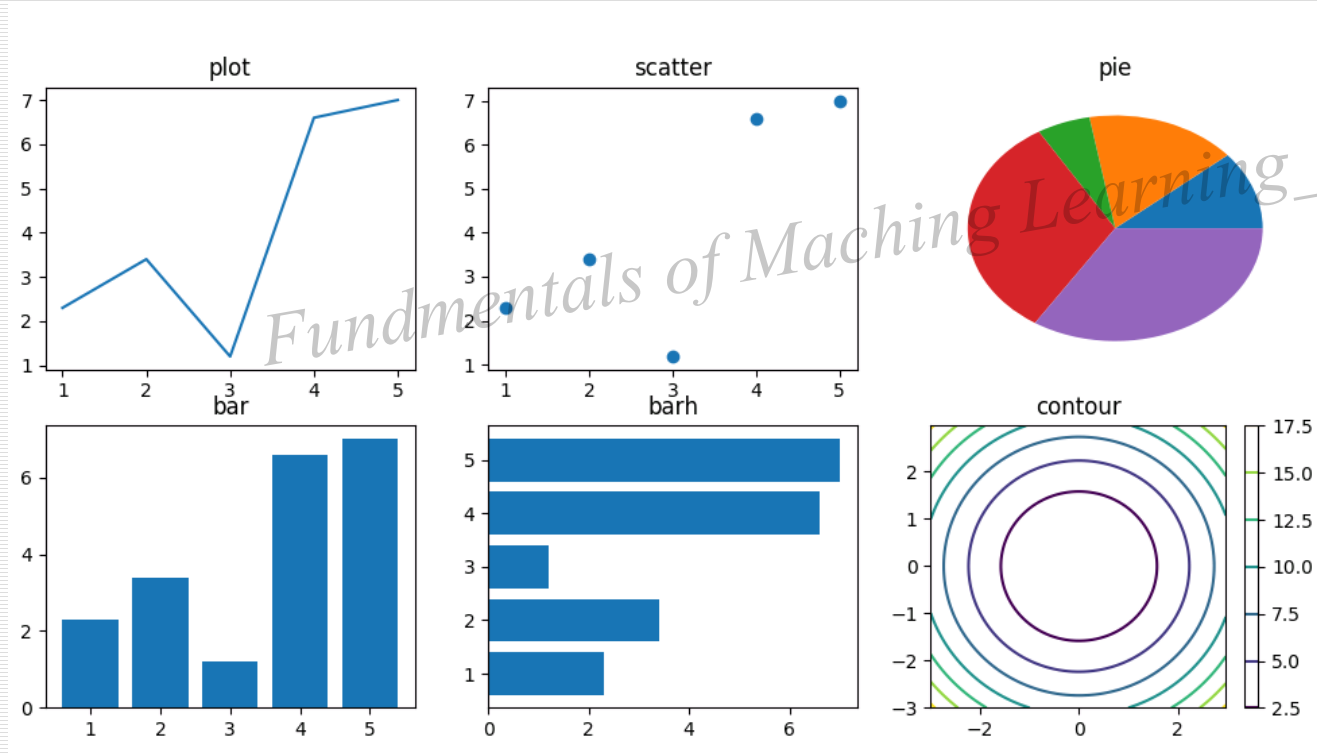
Matplotlib

□ 导入 : `import matplotlib.pyplot as plt`

- **plt**主要用于绘制数据的各种展示图
- **plt**子库提供了许多绘图函数
- 每个函数代表对图像的一个操作
- 函数采用**plt.()**形式调用，其中代表具体函数名称

Matplotlib

◆ 示例：绘制2D图



Matplotlib

□ **plt.figure()**函数创建一个全局绘图区域，并且使它成为当前的绘图对象

- 函数的参数：

num：图形编号或名称，取值为数字/字符串

figsize：绘图对象的宽和高，单位为英寸

dpi：绘图对象的分辨率，缺省值为80

facecolor：背景颜色

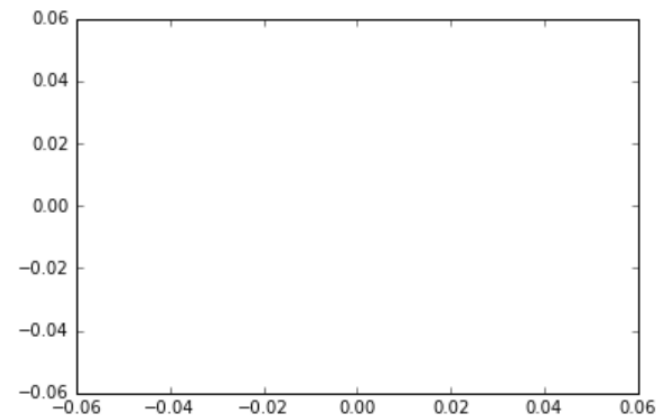
edgecolor：边框颜色

frameon：表示是否显示边框

```
import numpy as np
# 导入matplotlib.pyplot并命名为plt
import matplotlib.pyplot as plt
# 在jupyter中展示图
%matplotlib inline

# 创建一个全局绘图区域(画布)
plt.figure()
# 绘制空白区
plt.plot()
# 显示绘图
plt.show
```

<function matplotlib.pyplot.show>



Matplotlib

□ **plt.subplot**(行数, 列数, 子图序号)函数在全局绘图区中创建一个子绘图区域

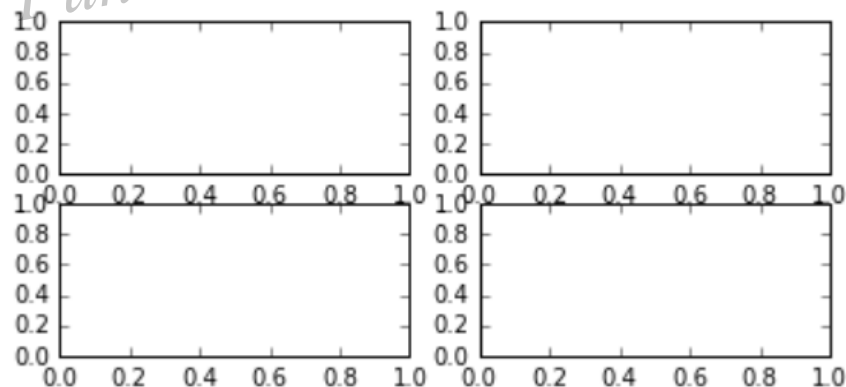
- 当**plt.subplot()**函数中的3个参数都小于**10**时，可以省略参数间的逗号，用一个**3**位数来表示
- 每个**plt.subplot()**函数只创建一个子图

Matplotlib

◆ 示例：将画布划分为 2×2 的子图区域，并绘制4个子图

```
# 在全局绘图区域内创建子绘图区域  
plt.subplot(321)  
plt.subplot(322)  
plt.subplot(323)  
plt.subplot(324)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1962467e828>
```



Matplotlib

□ 设置中文字体

```
plt.rcParams [ " font.sans-serif" ] = "SimHei"
```

- 运行配置参数：指定所绘制图表中的各种默认属性

中文字体	英文描述	中文字体	英文描述
宋体	SimSun	楷体	KaiTi
黑体	SimHei	仿宋	FangSong
微软雅黑	Microsoft YaHei	隶书	LiSu
微软正黑体	Microsoft JhengHei	幼圆	YouYuan

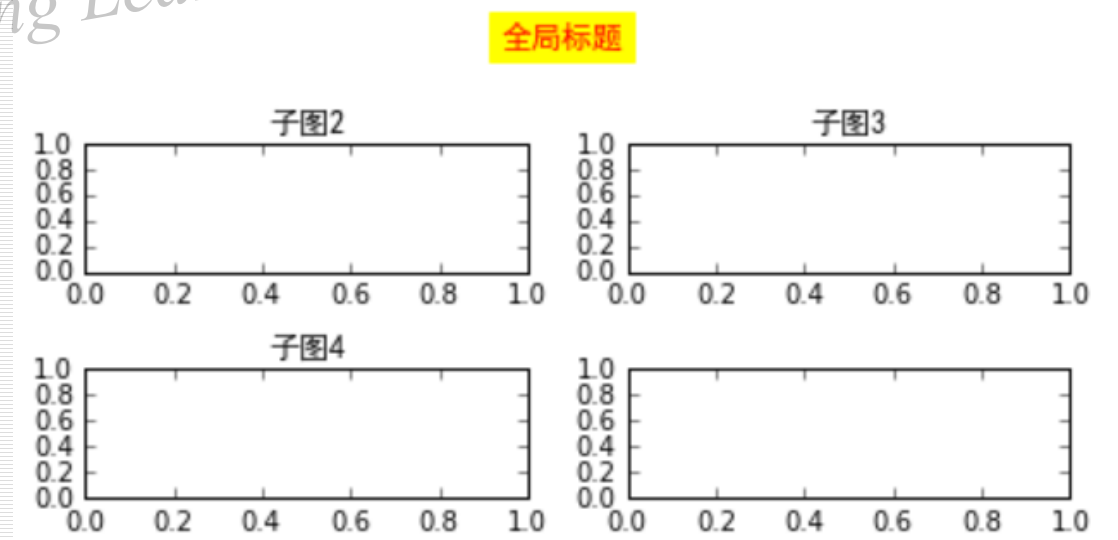
Matplotlib

□ 添加全局标题

`plt.suptitle('标题文字')`函数

□ 添加子标题

`plt.title('标题文字')`函数



Matplotlib

◆ 示例：将画布划分为 2×2 的子图区域，并绘制4个子图

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
plt.rcParams["font.sans-serif"] = "SimHei"
```

```
plt.suptitle('全局标题', fontsize=12, color="red", backgroundcolor="yellow")
```

```
plt.title('子图1')
```

```
plt.subplot(321)
```

```
plt.title('子图2')
```

```
plt.subplot(322)
```

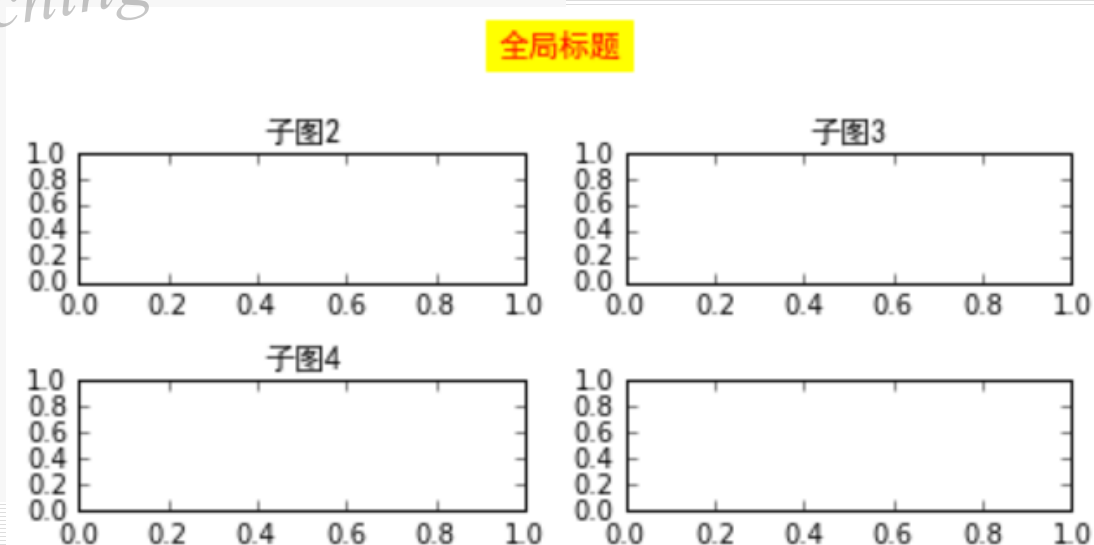
```
plt.title('子图3')
```

```
plt.subplot(323)
```

```
plt.title('子图4')
```

```
plt.subplot(324)
```

```
plt.tight_layout(rect=[0, 0, 1, 0.9])
```



Matplotlib

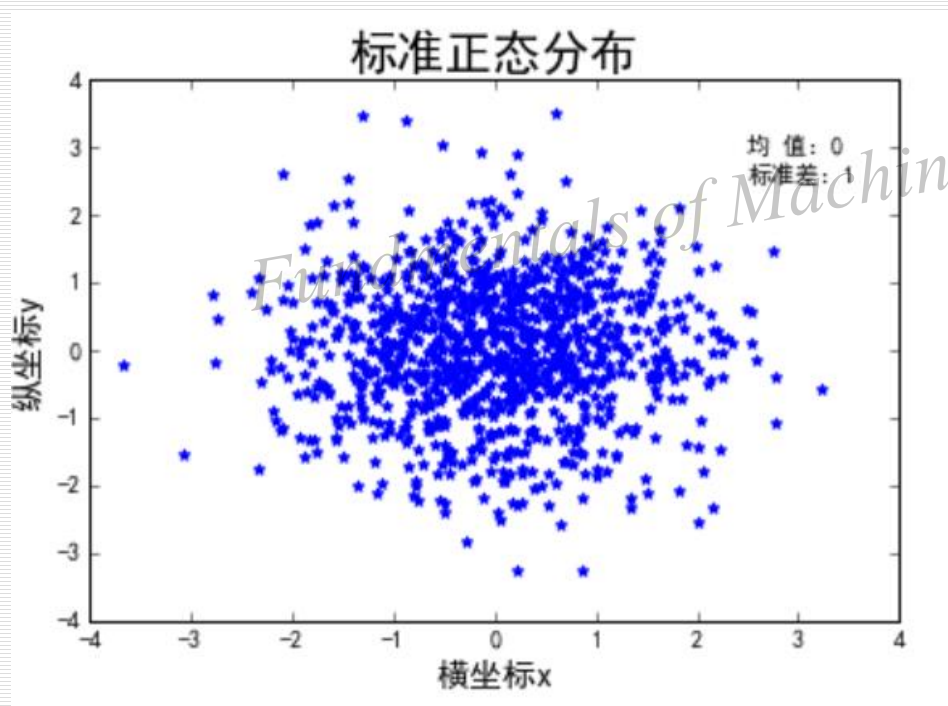
□ 绘制散点图：plt.scatter(x, y, scale, color, marker, label)

参 数	说 明	默认值
x	数据点的x坐标	不可省略
y	数据点的y坐标	不可省略
scale	数据点的大小	36
color	数据点的颜色	
marker	数据点的样式	'o' (圆点)
label	图例文字	

颜 色	缩略字符	颜 色	缩略字符
blue	b	black	k
green	g	white	w
red	r	cyan	c
yellow	y	magenta	m

Matplotlib

◆ 示例：绘制标准正态分布的散点图



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

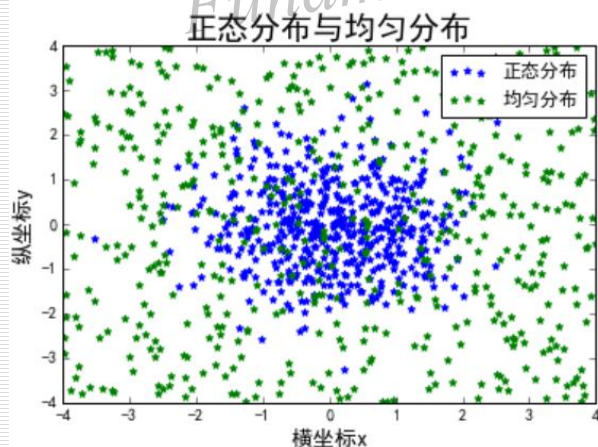
# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False
# 标准正态分布的散点坐标
n = 1024
x = np.random.normal(0,1,n)
y = np.random.normal(0,1,n)
# 绘制散点图
plt.scatter(x, y, color="blue",marker='*')
# 设置标题
plt.title("标准正态分布",fontsize=20)
# 设置文本
plt.text(2.5,2.5,"均值: 0\n标准差: 1")
# 设置坐标轴范围
plt.xlim(-4,4)
plt.ylim(-4,4)
# 设置坐标轴标签
plt.xlabel('横坐标x', fontsize=14)
plt.ylabel('纵坐标y', fontsize=14)
```


Matplotlib

□ 增加图例

- **plt.scatter(x, y, scale, color, marker, label)**
- **plt.legend(loc, fontsize)**

◆ 示例：绘制正态分布、均匀分布的散点图



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 正态分布与均匀分布的散点坐标
n = 512
x1 = np.random.normal(0,1,n)
y1 = np.random.normal(0,1,n)

x2 = np.random.uniform(-4,4,n)
y2 = np.random.uniform(-4,4,n)

# 绘制散点图
plt.scatter(x1, y1, color="blue", marker='*', label="正态分布")
plt.scatter(x2, y2, color="green", marker='*', label="均匀分布")
plt.legend()

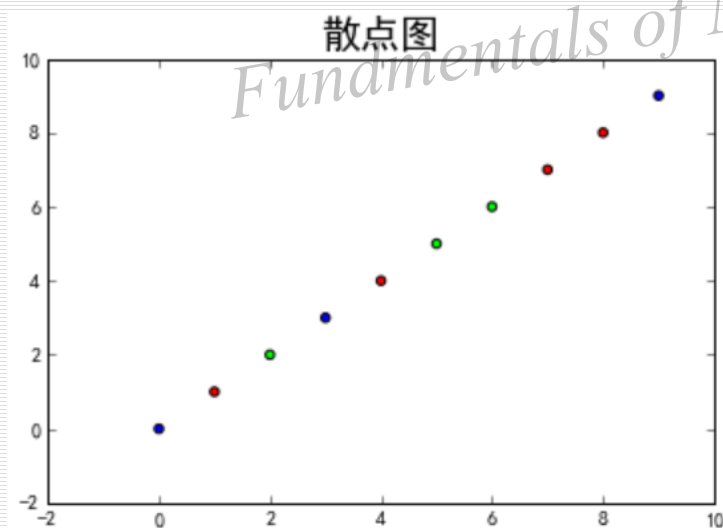
# 设置标题
plt.title("正态分布与均匀分布", fontsize=20)

# 设置坐标轴范围
plt.xlim(-4,4)
plt.ylim(-4,4)
# 设置坐标轴标签
plt.xlabel('横坐标x', fontsize=14)
plt.ylabel('纵坐标y', fontsize=14)
```

Matplotlib

□ plt.scatter(x, y, c, cmap) : 彩色映射

将参数c指定为一个列表或数组，所绘制图形的颜色，可以随这个列表或数组中元素的值而变换，变换所对应的颜色由参数cmap中的颜色所提供



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 生成数据点
x = np.arange(10)
y = np.arange(10)
doct_color = [0,1,2,0,1,2,2,1,1,0]

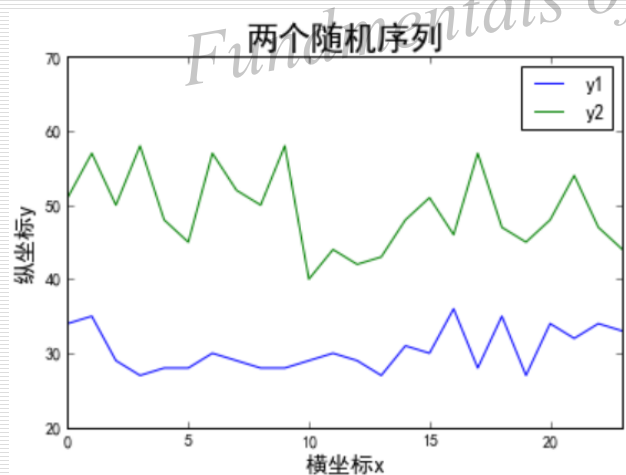
# 绘制散点图
plt.scatter(x, y, c=doct_color, cmap='brg')

# 设置标题
plt.title("散点图", fontsize=20)
```

Matplotlib

□ **plt.plot(x, y, color, marker, label, linewidth, markersize) :**
散点图基础上，将相邻点用线段相连接

◆ 示例：两个随机序列的折线图



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 生成随机数列
n = 24
y1 = np.random.randint(27,37,n)
y2 = np.random.randint(40,60,n)

# 绘制折线图
plt.plot(y1, label='y1')
plt.plot(y2, label='y2')

# 设置标题
plt.title("两个随机序列",fontsize=20)

# 显示图例
plt.legend()

# 设置坐标轴范围
plt.xlim(0,23)
plt.ylim(20,70)

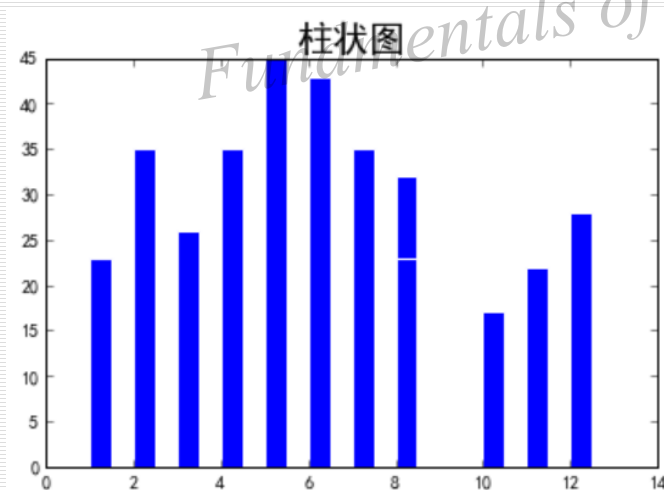
# 设置坐标轴标签
plt.xlabel('横坐标x', fontsize=14)
plt.ylabel('纵坐标y', fontsize=14)
```

Matplotlib

□ **plt.bar(left, height, width, facecolor, edgecolor, label) :**

由一系列高度不等的柱形条纹表示数据分布的情况

◆ 示例: 柱状图的生成



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 生成数据点
x = [1, 2, 3, 4, 5, 6, 7, 8, 8, 10, 11, 12]
y = [23, 35, 26, 35, 45, 43, 35, 32, 23, 17, 22, 28]

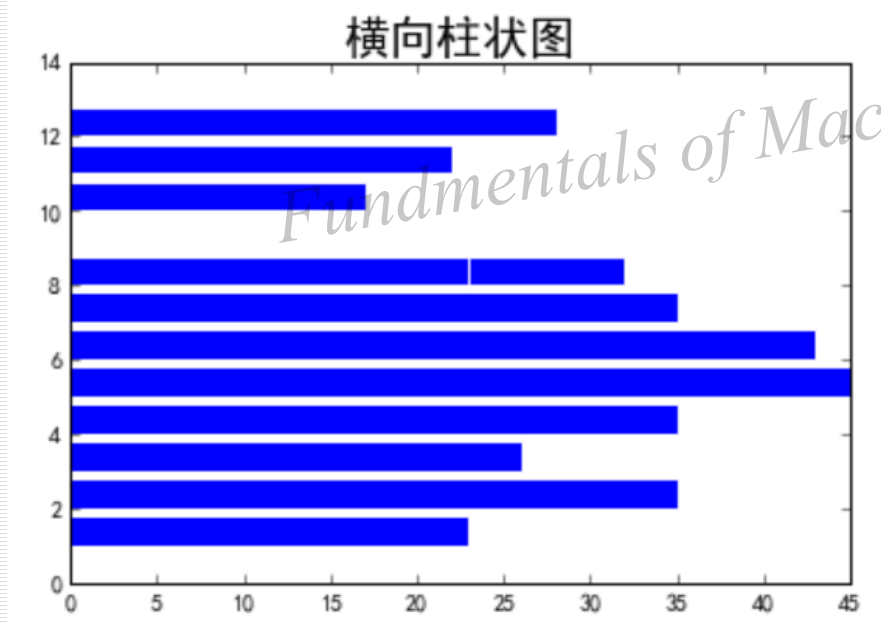
# 绘制柱状图
plt.bar(x, y, width=0.5, facecolor='blue', edgecolor='white')

plt.title("柱状图", fontsize=20)
```

Matplotlib

□ `plt.barh(bottom, width, height, left)`: 绘制横向条形图

◆ 示例: 横向柱状图的生成



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 生成数据点
x = [1, 2, 3, 4, 5, 6, 7, 8, 8, 10, 11, 12]
y = [23, 35, 26, 35, 45, 43, 35, 32, 23, 17, 22, 28]

# 绘制横向柱状图
plt.barh(x, y, facecolor='blue', edgecolor='white')

plt.title("横向柱状图", fontsize=20)
```

Matplotlib

□ plt.pie(data,explode) : 绘制饼图

◆ 示例: 饼图生成



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 生成数据点
x = [1, 2, 3, 4, 5, 6, 7, 8, 8, 10, 11, 12]
y = [23, 35, 26, 35, 45, 43, 35, 32, 23, 17, 22, 28]

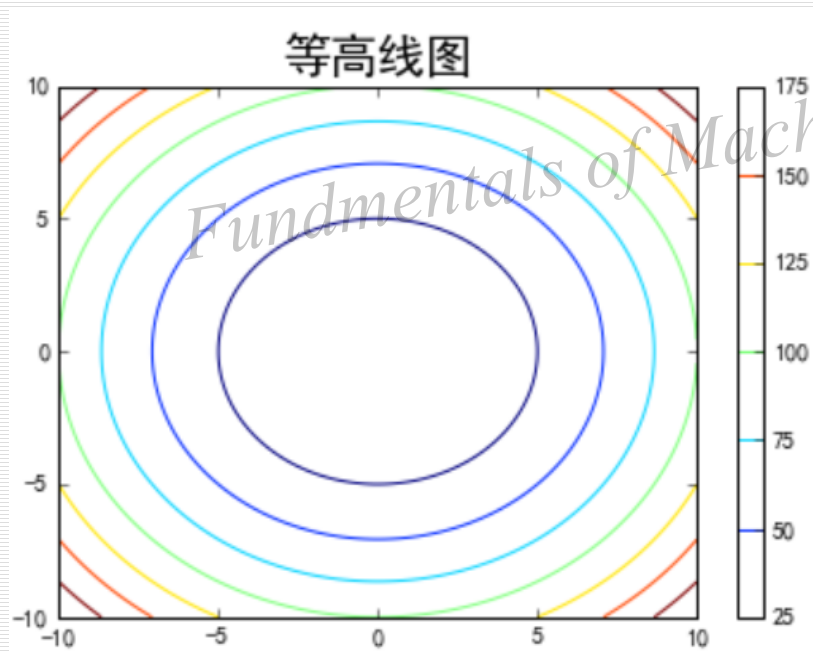
# 绘制饼图
plt.pie(y, labels=x)

plt.title("饼图", fontsize=20)
```

Matplotlib

□ `plt.contour(x,y,z)` : 绘制等高线图

◆ 示例: 生成等高线图



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

# 设置默认字体为中文黑体
plt.rcParams["font.sans-serif"] = "SimHei"
plt.rcParams["axes.unicode_minus"] = False

# 生成数据点
step = 0.01
x = np.arange(-10,10,step)
y = np.arange(-10,10,step)

# 将原始数据变成网格数据形式
X,Y = np.meshgrid(x,y)

# 等高线的高度z值
Z = X**2+Y**2

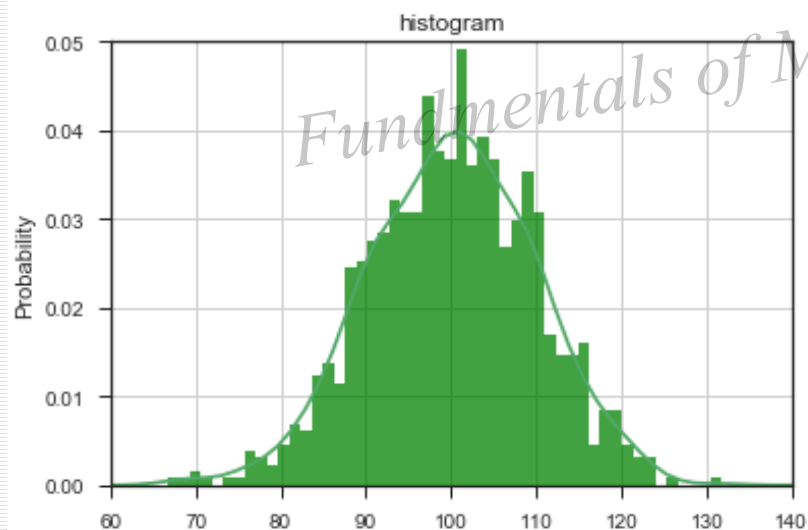
# 填充颜色, f即filled
# plt.contourf(X,Y,Z)

# 画等高线
plt.contour(X,Y,Z)
# 设置颜色条 (显示在图右边)
plt.colorbar()
plt.title("等高线图", fontsize=20)
```

Matplotlib

❑ **plt.hist(x,bins,normed)** : 绘制直方图

◆ 示例: 生成直方图



```
# 绘制直方图
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
%matplotlib inline

# 生成均值为100, 标准差为10的1000个正态分布样本, 随机样本位于[0, 1)中
z = 100 + 10 * np.random.randn(1000)

# 画直方图
n, bins, patches = plt.hist(z, 50, normed=1, facecolor='green', alpha=0.75)

plt.ylabel('Probability')
plt.title('histogram')

# 设置坐标范围
# plt.axis([xmin, xmax, ymin, ymax])
plt.axis([60, 140, 0, 0.05])

plt.grid(True)

# Perform a kernel density estimate on the data:
kernel = stats.gaussian_kde(z)

xs = np.linspace(60, 140)
plt.plot(xs, kernel(xs))

[<matplotlib.lines.Line2D at 0xe13f128>]
```


Matplotlib

□ plt更多绘图函数

- `plt.boxplot(data, notch, position)` : 绘制箱型图
- `plt.vline()` : 绘制垂直线
- `plt.stem(x, y, linefmt, markerfmt, basefmt)` : 绘制曲线每个点到水平轴线垂线
- `plt.polar(theta, r)` : 绘制极坐标图
- `plt.cohere(x, y, NFFT=256, Fs)` : 绘制X-Y相关性函数
- `plt.specgram()` : 绘制功率谱密度图
- `plt.pcolormesh()` : 绘制彩图, 例如, 绘制分类边界
- `plt.matshow()` : 在窗口中显示数组矩阵
- `plt.imshow()` : 在显示对象
- `plt.imsave()` : 保存数组为图像文件
- `plt.imread()` : 从图像文件中读取数组

Matplotlib

□ 可视化学习与进阶

- **Matplotlib**
<https://matplotlib.org/gallery.html>
- **Pandas**
https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
- **Seaborn**
<https://seaborn.pydata.org/examples/index.html>
Kaggle的seaborn例子
<https://www.kaggle.com/kanncaa1/seaborn-tutorial-for-beginners>

Pandas

- ❑ **Pandas库(Panel Data & Data Analysis)**

- ❑ 强大的结构化数据分析、处理工具

- ❑ 高效、方便地操作大型数据集

- ❑ 常与**NumPy** , **Matplotlib**配合使用

- ❑ 导入方式：**import pandas as pd**

Pandas

□ 两种数据类型：Series, DataFrame

- **Series**：类似于一维数组，由一组数据及与之相关的数据标签(即索引)组成。
- **DataFrame**：二维表格型的数据结构，包含有一组有序的列，每列可为不同数据类型(数值、字符串、布尔型等)，有行、列索引，可被看做由**Series**组成的字典

Pandas

□ Series类型创建：pd.Series(data,index=index)

data:

- 列表，index与列表元素个数一致
- 标量值，index表达Series类型的尺寸
- 字典，键值对中的“key”是索引，index从字典中进行选择操作
- ndarray，索引和数据都可以通过ndarray类型创建
- 其他函数，range()函数等

Pandas

□ Series类型索引：自动索引和自定义索引 并存，但不能混用

```
import pandas as pd  
d = pd.Series(range(5))  
d
```

```
0    0  
1    1  
2    2  
3    3  
4    4  
dtype: int32
```

```
import pandas as pd  
d = pd.Series(3, index=['a', 'b', 'c'])  
d
```

```
a    3  
b    3  
c    3  
dtype: int64
```

```
d[['a', 'b', 'c']]
```

```
a    3  
b    3  
c    3  
dtype: int64
```

```
d[[0, 1, 2]]
```

```
a    3  
b    3  
c    3  
dtype: int64
```

```
d[['a', 'b', 2]]
```

```
a    3.0  
b    3.0  
2    NaN  
dtype: float64
```

Pandas

□ Series类型 : index、values

- index 获得索引
- values 获得数据

```
import pandas as pd
```

```
# 使用一个字典生成Series, 字典的键为索引  
s3 = pd.Series({'A':1, 'B':2, 'C':3})
```

```
print('s3:\n', s3)
```

```
print('type(s3)', type(s3))
```

```
print('s3.index', s3.index)
```

```
print('s3.values', s3.values)
```

```
s3:
```

```
A    1
```

```
B    2
```

```
C    3
```

```
dtype: int64
```

```
type(s3) <class 'pandas.core.series.Series'>
```

```
s3.index Index(['A', 'B', 'C'], dtype='object')
```

```
s3.values [1 2 3]
```

Pandas

□ Series类型操作：类似字典类型：

- 通过自定义索引访问
- 保留字in操作
- 使用.get()方法

```
d['a']
```

```
3
```

```
'a' in d
```

```
True
```

```
d.get('a')
```

```
3
```


Pandas

□ DataFrame类型

- 表格型的数据类型，每列值类型可以不同
- 既有行索引、也有列索引
- 常用于表达二维数据，但可以表达多维数据

	A	B	C
1	-0.991668	0.329631	-2.678281
2	-0.212461	0.321946	0.095214
3	-1.193442	-0.756324	1.009201
4	0.242830	0.109956	-0.191901

自动行索引、自动列索引

Pandas

□ DataFrame类型创建：

```
df = pd.DataFrame(data=None, index=None, columns=None,  
dtype=None, copy=False)
```

data:

- 二维ndarray对象
- 由一维ndarray、列表、字典、元组或Series构成的字典
- Series类型
- 其他的DataFrame类型

Pandas

◆ 示例：DataFrame类型创建

```
import pandas as pd
import numpy as np

# 使用numpy函数创建
df1 = pd.DataFrame(np.random.randn(4, 3), index=list('1234'), columns=list('ABC'))
print('df1:\n', df1)
print('type(df1):', type(df1))
print('df1.dtypes:\n', df1.dtypes)
print('df1.index:\n', df1.index)
print('df1.columns:\n', df1.columns)
print('df1.shape[0]:', df1.shape[0])
print('df1.shape[1]:', df1.shape[1])

df1:
      A      B      C
1 -0.991668  0.329631 -2.678281
2 -0.212461  0.321946  0.095214
3 -1.193442 -0.756324  1.009201
4  0.242830  0.109956 -0.191901
type(df1): <class 'pandas.core.frame.DataFrame'>
df1.dtypes:
A      float64
B      float64
C      float64
dtype: object
df1.index:
Index(['1', '2', '3', '4'], dtype='object')
df1.columns:
Index(['A', 'B', 'C'], dtype='object')
df1.shape[0]: 4
df1.shape[1]: 3
```

自动行索引、自动列索引

Pandas

□ DataFrame类型操作

- DataFrame() 创建一个DataFrame对象
- df.values 返回ndarray类型的对象
- df.index 获取行索引
- df.columns 获取列索引
- df.axes 获取行及列索引
- df.T 行与列对调
- df.info() 打印DataFrame对象的信息
- df.head(i) 显示前 i 行数据
- df.tail(i) 显示后 i 行数据
- df.describe() 查看数据按列的统计信息

reindex() : 能够改变或重排Series和DataFrame索引; drop()能够删除Series和DataFrame指定行或列索引

Pandas

□ 示例：DataFrame类型的基本统计分析

```
# 数据的统计摘要  
df1.describe()
```

	A	B	C
count	4.000000	4.000000	4.000000
mean	-0.538685	0.001302	-0.441442
std	0.671091	0.515239	1.576714
min	-1.193442	-0.756324	-2.678281
25%	-1.042111	-0.106614	-0.813496
50%	-0.602064	0.215951	-0.048343
75%	-0.098638	0.323867	0.323711
max	0.242830	0.329631	1.009201

```
# 转置数据  
df1.T
```

	1	2	3	4
A	-0.991668	-0.212461	-1.193442	0.242830
B	0.329631	0.321946	-0.756324	0.109956
C	-2.678281	0.095214	1.009201	-0.191901

```
# 按轴排序
```

```
df1.sort_index(axis=1, ascending=False)
```

	C	B	A
1	-2.678281	0.329631	-0.991668
2	0.095214	0.321946	-0.212461
3	1.009201	-0.756324	-1.193442
4	-0.191901	0.109956	0.242830

```
# 按值排序
```

```
df1.sort_values(by='A')
```

	A	B	C
3	-1.193442	-0.756324	1.009201
1	-0.991668	0.329631	-2.678281
2	-0.212461	0.321946	0.095214
4	0.242830	0.109956	-0.191901

Pandas

□ 输入/输出

- 写入csv文件：**df.to_csv()**
- 读取csv文件：**pd.read_csv()**
- 写入Excel文件：**df.to_excel()**
- 读取Excel文件：**pd.read_excel()**

Pandas

□ Pandas入门与进阶

- **Pandas官网**

<http://pandas.pydata.org/>

- **10分钟入门pandas**

https://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html