

Python库简介

几个相关的库

- Numpy: 高效地处理高维数组; 高效的数学函数
 - Quickstart tutorial:
<https://docs.scipy.org/doc/numpy/user/quickstart.html>
 - A Visual Intro to NumPy and Data Representation:
<http://jalammar.github.io/visual-numpy/>
- Matplotlib: 可视化, 绘制2D或3D图形
 - Pyplot tutorial:
https://matplotlib.org/users/pyplot_tutorial.html
- Pandas: 统计与数学分析
 - 10 Minutes to pandas: http://pandas.pydata.org/pandas-docs/stable/getting_started/10min.html
- Scikit-learn: 机器学习
 - <https://scikit-learn.org/stable/documentation.html>

Numpy

- Numpy (Numerical Python extensions) 是一个第三方的Python包，用于科学计算，前身是1995年就开始开发的一个用于数组运算的库
- 极大地简化了向量和矩阵的操作处理，是一些主力软件包（如scikit-learn、Scipy、pandas和tensorflow）架构的基础部分。

ndarray数据类型

- Numpy提供了一种新的数据结构：**ndarray**（n维数组， n-dimensional array）
- 不同于列表和元组，数组只能存放**相同类型**的对象（如全部整型或全部浮点型）
- 这使得在数组上的一些运算远远快于在列表上的相同运算；另外，数组占用更小的存储
- 数组强有力地扩展了列表的索引机制

创建ndarray

- 首先导入Numpy库

```
>>>import numpy as np
```

- 然后开始创建n维数组

```
>>>np.array([2, 3, 6, 7])  
array([2, 3, 6, 7])
```

```
>>>np.array([2, 3, 6, 7.])  
array([2., 3., 6., 7.])
```

```
>>>np.array([2, 3, 6, 7+1j])  
array([2.+0.j, 3.+0.j, 6.+0.j, 7.+1.j])
```

创建等差数列的数组

- **arange**([start,] stop[, step,], dtype=None)

```
>>>np.arange(5)
array([0, 1, 2, 3, 4])

>>>np.arange(10, 100, 20, dtype=float)
array([10., 30., 50., 70., 90.] )
```

- **linspace**(start, stop, num=50, endpoint=True, retstep=False, dtype=None)

```
>>>np.linspace(0., 2.5, 5)
array([0., 0.625, 1.25, 1.875, 2.5])

>>> from numpy import pi
>>> x = np.linspace( 0, 2*pi, 100 )   # 用于在多个点执行某函数
>>> f = np.sin(x)
```

多维数组表示的矩阵

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

```
>>>a
```

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

```
>>>a.shape      # 行数和列数  
(2, 3)
```

```
>>>a.ndim       # 维数  
2
```

```
>>>a.size       # 元素数  
6
```

改变形状

```
>>>import numpy as np

>>>a = np.arange(0, 20, 1)      # 一维数组

>>>b = a.reshape((4, 5))         # 4行, 5列
>>>c = a.reshape((20, 1))       # 2维
>>>d = a.reshape((-1, 4))       # -1: 自动决定行数

>>>a.shape = (4, 5)             # 改变a的形状
```


形状(N,), (N, 1)和(1, N)不同

- 形状(N,): 数组是一维的
- 形状(N, 1): 数组是二维的, N行一列
- 形状(1, N): 数组是二维的, 一行N列

```
import numpy as np

a = np.array([1, 2, 3, 4, 5])      # 一维数组
b = a.copy()

c1 = np.dot(np.transpose(a), b)    # 转置对一维数组不起作用
print(c1)
c2 = np.dot(a, np.transpose(b))    # 转置也可以写成b.T
print(c2)

ax = np.reshape(a, (5, 1))
bx = np.reshape(b, (1, 5))
c = np.dot(ax, bx)
print(c)
```

用相同元素填充数组

```
>>>np.zeros(3)
array([0., 0., 0.])
```

```
>>>np.zeros((2, 2), complex)
array([[0.+0.j, 0.+0.j],
       [0.+0.j, 0.+0.j]])
```

```
>>>np.ones((2, 3))
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
>>>np.full((2,2), 7)
array([[7, 7],
       [7, 7]])
```

用随机数填充数组

- **rand:** 0到1之间[0, 1)均匀分布的随机数

```
>>>np.random.rand(2, 4)
array([[ 0.94672374,  0.0383632 ,  0.12738539,  0.21592466],
       [ 0.49394559,  0.2216863 ,  0.3053351 ,  0.51381235]])
```

- **randn:** 服从均值为0，方差为1的标准正态（高斯）分布的随机数

```
>>>np.random.randn(2, 4)
array([[ 1.05383548, -1.2142876 , -0.83458293,  0.53291161],
       [ 0.08311765,  0.14007751, -0.06647882,  1.09115942]])
```

- 也有其他标准概率分布的随机数

一维数组索引与切片

- **[start : stop]**的索引形式可用于从数组中抽取片段（从start位置开始直到stop位置但不包括stop）

```
>>>a = np.array([0, 1, 2, 3, 4])
```

```
>>>a[1:3]  
array([1, 2])
```

```
>>>a[:3]  
array([0, 1, 2])
```

```
>>>a[1:]  
array([1, 2, 3, 4])
```

```
>>>a[1:-1]  
array([1, 2, 3])
```

一维数组索引与切片

- 整个数组：a或者a[:]

```
>>>a = np.array([0, 1, 2, 3, 4])  
>>>a[:]  
array([0, 1, 2, 3, 4])
```

- 想取出间隔的元素，可以在第二个冒号之后说明第三个数（步长）：

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

- 步长-1，可用于反转一个数组：

```
>>>a[::-1]  
array([4, 3, 2, 1, 0])
```

二维数组索引

- 多维数组的索引是整数元组:

```
>>>a = np.arange(12); a.shape = (3, 4); a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>>a[1, 2]
```

```
6
```

```
>>>a[1, -1]
```

```
7
```

二维数组切片：单行单列

- 和列表类似

```
>>>a = np.arange(12); a.shape = (3, 4); a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>>a[:, 1]
array([1, 5, 9])
```

```
>>>a[2, :]
array([8, 9, 10, 11])
```

```
>>>a[1][2]
6
```

```
>>>a[2]
array([8, 9, 10, 11])
```

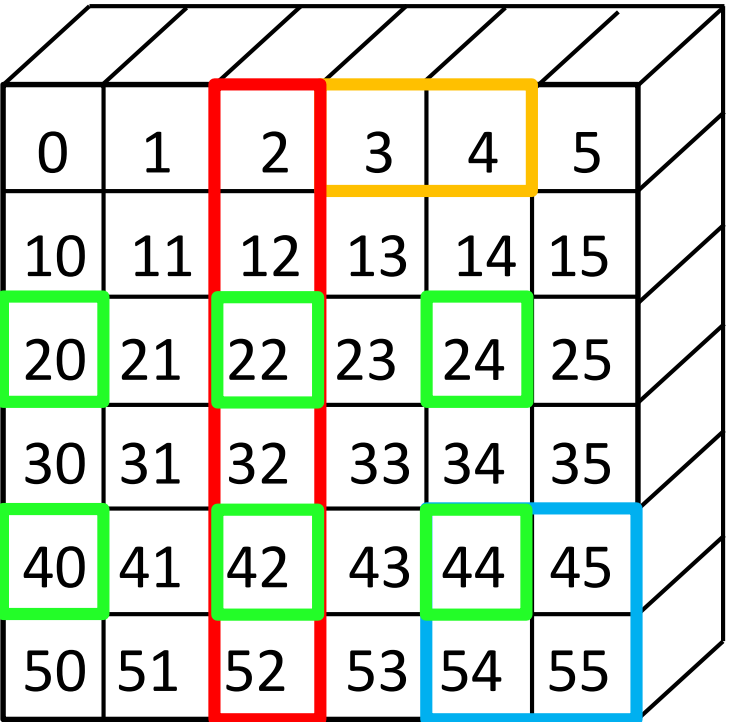
数组索引

```
>>>a[0, 3:5]  
array([3, 4])
```

```
>>>a[4:, 4:]  
array([[44, 45],  
       [54, 55]])
```

```
>>>a[:, 2]  
array([2, 12, 22, 32, 42, 52])
```

```
>>>a[2::2, ::2]  
array([[20, 22, 24]  
       [40, 42, 44]])
```



| | | | | | |
|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 |
| 10 | 11 | 12 | 13 | 14 | 15 |
| 20 | 21 | 22 | 23 | 24 | 25 |
| 30 | 31 | 32 | 33 | 34 | 35 |
| 40 | 41 | 42 | 43 | 44 | 45 |
| 50 | 51 | 52 | 53 | 54 | 55 |

The diagram shows a 6x6 grid of numbers from 0 to 55. The grid is divided into four 3x3 sub-grids. The first sub-grid (top-left) has a red vertical line at column 2 and a yellow horizontal line at row 0. The second sub-grid (top-right) has a green vertical line at column 2 and a green horizontal line at row 0. The third sub-grid (bottom-left) has a green vertical line at column 2 and a green horizontal line at row 0. The fourth sub-grid (bottom-right) has a blue vertical line at column 2 and a blue horizontal line at row 0. The numbers in the grid are: Row 0: 0, 1, 2, 3, 4, 5; Row 1: 10, 11, 12, 13, 14, 15; Row 2: 20, 21, 22, 23, 24, 25; Row 3: 30, 31, 32, 33, 34, 35; Row 4: 40, 41, 42, 43, 44, 45; Row 5: 50, 51, 52, 53, 54, 55.

拷贝与视图

- 标准列表的一个切片是它的一个**拷贝**
- **Numpy**数组的一个切片是数组上的一个**视图**，切片数组和原始数组都引用的是同一块内存区域。因而，当改变视图内容时，原始数组的内容也被同样改变了：

```
>>>a = np.arange(5); a  
array([0,  1,  2,  3, 4])
```

```
>>>b = a[2:]; b  
array([2, 3, 4])
```

```
>>>b[0] = 100; b  
array([100,  3,  4])
```

```
>>>a  
array([0,  1, 100,  3,  4])
```

拷贝与视图

- 为了避免改变原数组，可以拷贝切片：

```
>>>a = np.arange(5); a  
array([0,  1,  2,  3,  4])
```

```
>>>b = a[2:].copy(); b  
array([2,  3,  4])
```

```
>>>b[0] = 100; b  
array([100,  3,  4])
```

```
>>>a  
array([0,  1,  2,  3,  4])
```

数组计算

- 基本的算术运算都作用在数组的元素级别

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

print(x + y)
print(np.add(x, y))

print(x - y)
print(np.subtract(x, y))

print(x * y)
print(np.multiply(x, y))

print(x / y)
print(np.divide(x, y))

print(np.sqrt(x))
```

矩阵乘法

- 矩阵乘法是使用dot函数实现的：

```
>>>A = np.array([[1, 2], [3, 4]])  
>>>np.dot(A, A)  
array([[7, 10],  
       [15, 22]])
```

- dot函数也可用于矩阵和向量的乘法：

```
>>>A  
array([[1, 2],  
       [3, 4]])  
>>>x = np.array([10, 20])  
  
>>>np.dot(A, x)           #等价于A.dot(x)  
array([50, 110])  
  
>>>np.dot(x, A)           #等价于x.dot(A)  
array([70, 100])
```

更高效的数学函数

- Numpy中包含许多常用的数学函数，例如：np.log, np.maximum, np.sin, np.exp, np.abs等等（详见：<https://docs.scipy.org/doc/numpy/reference/routines.math.html>）
- 大多数情况下，Numpy中的函数比math库中类似的函数更高效，尤其是处理大规模数据时

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x))           # Compute sum of all elements;
print(np.sum(x, axis=0))   # Compute sum of each column;
print(np.sum(x, axis=1))   # Compute sum of each row;
```

保存数组到文件

- **savetxt()**函数将一个数组保存到一个文本文件中:

```
>>>a = np.linspace(0, 1, 12); a.shape = (3, 4); a
array([[ 0.          ,  0.09090909,  0.18181818,  0.27272727],
       [ 0.36363636,  0.45454545,  0.54545455,  0.63636364],
       [ 0.72727273,  0.81818182,  0.90909091,  1.          ]])

>>>np.savetxt("myfile.txt", a)
```

- 其他格式的文件也可以（参见文档）
- **save()**函数将一个数组存成一个Numpy的“.npy”格式的二进制文件:

```
>>>np.save("myfile", a)
```

生成一个二进制文件myfile.npy包含数组a，之后可以使用**np.load()**函数读入内存

从文本文件读入数组

- **loadtxt()**函数把一个存成文本文件的数组读入内存
- 缺省地，该函数假设列是用空白符分隔的。可以通过修改可选的参数来改变此假设。**#**开头的行被忽略。
- 示例文本文件data.txt:

| # Year | Min temp. | Max temp. |
|--------|-----------|-----------|
| 1990 | -1.5 | 25.3 |
| 1991 | -3.2 | 21.2 |

```
>>>table = np.loadtxt("data.txt")
>>>table
array([[1.99000000e+03, -1.50000000e+00, 2.53000000e+01],
       [1.99100000e+03, -3.20000000e+00, 2.12000000e+01]])
```

Matplotlib

- Matplotlib是Python中最常用的可视化工具之一，可以非常方便地创建海量类型的2D图表和一些基本的3D图表
- 因为在函数的设计上参考了MATLAB，所以叫做Matplotlib
- 首次发表于2007年，是为了可视化癫痫病人的脑皮层电图相关的信号而研发的，原作者John D. Hunter博士是一名神经生物学家

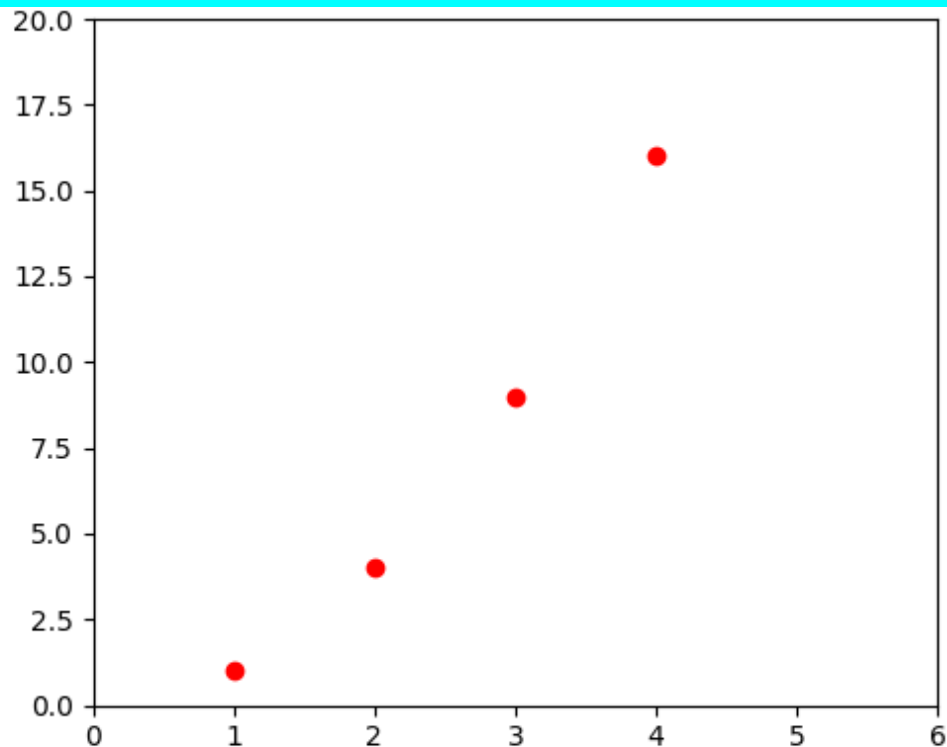
最简单的图表

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1,4,9,16], 'ro')

plt.axis([0, 6, 0, 20])

plt.show()
```

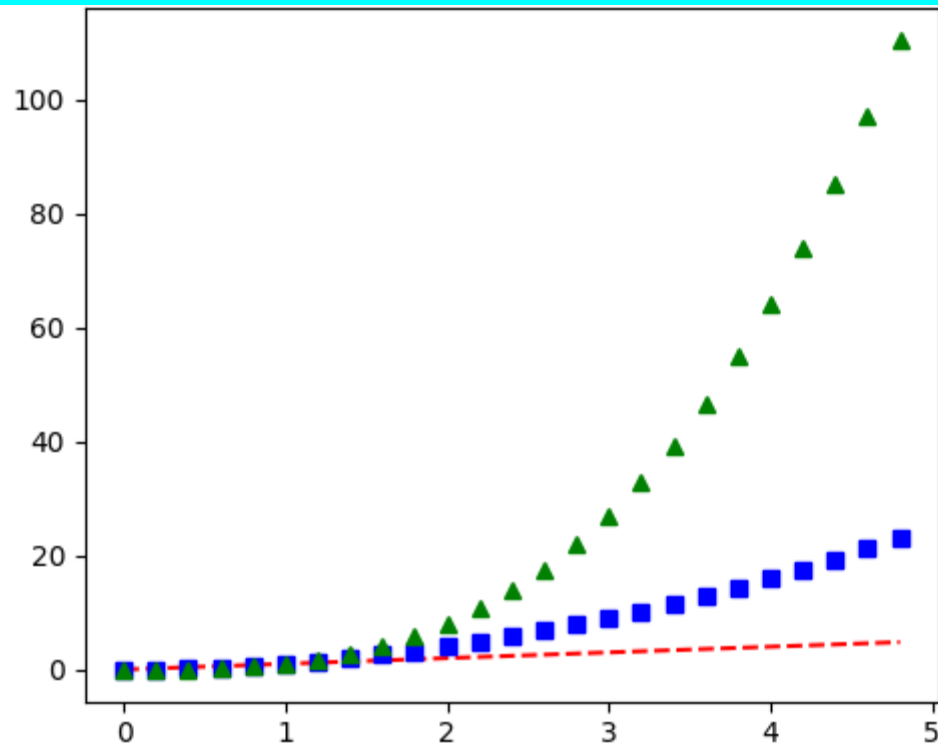


一张图表中多个函数（1）

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

t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')

plt.show()
```



设置线条属性

- 使用键值对参数:

```
plt.plot(x, y, linewidth=2.0)
```

- 使用Line2D类对象的属性设置方法:

```
line, = plt.plot(x, y, '-')  
  
line.set_antialiased(False) # turn off antialiasing
```

- 使用**setp()**命令:

```
lines = plt.plot(x1, y1, x2, y2)  
  
# use keyword args  
plt.setp(lines, color='r', linewidth=2.0)  
# or MATLAB style string value pairs  
plt.setp(lines, 'color', 'r', 'linewidth', 2.0)
```

一张图表中多个函数（2）

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)

plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])

plt.show()
```

多张图表：子图表

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

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

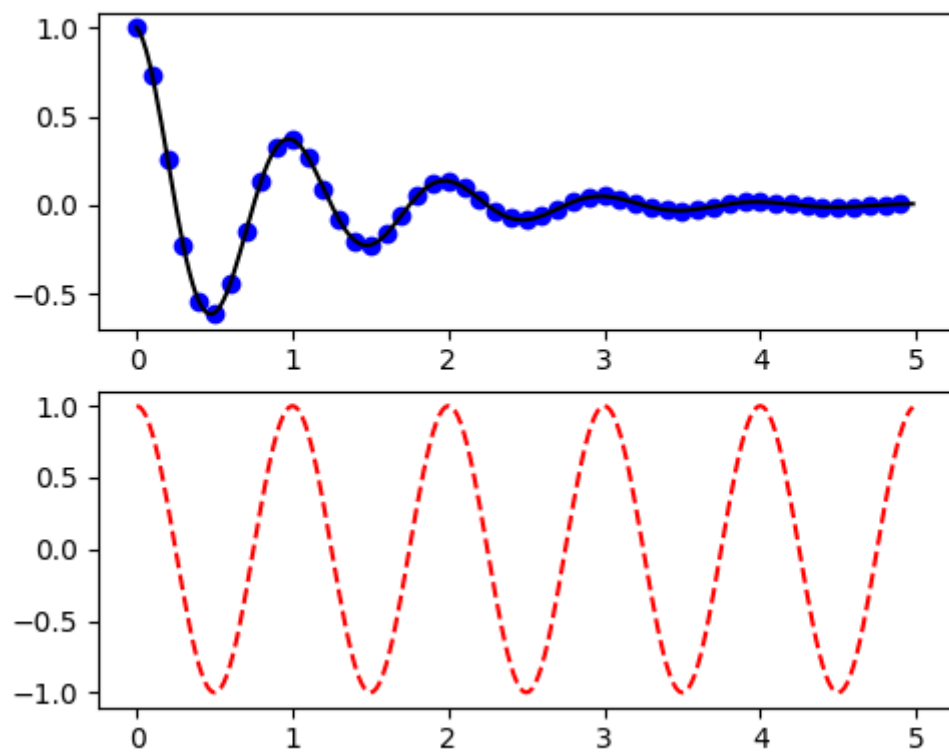
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')

plt.show()
```

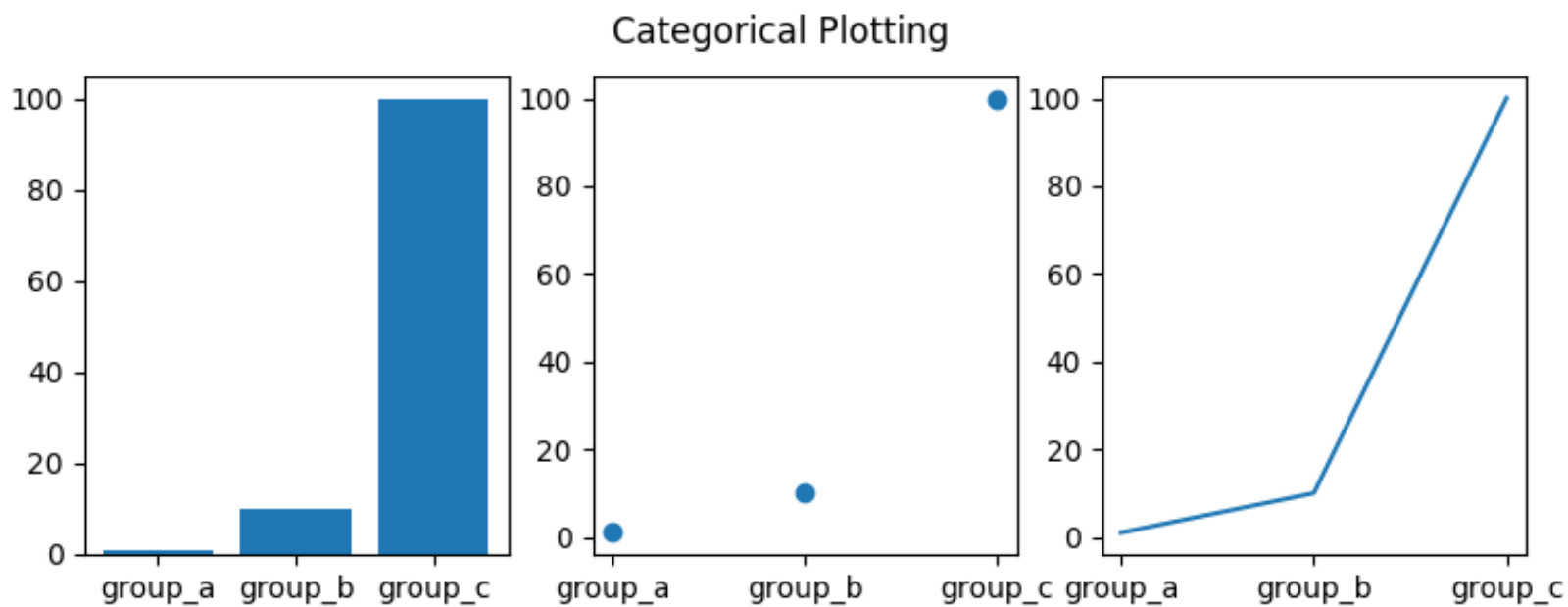
多张图表：子图表



绘制分类变量的图表

```
names = ['group_a', 'group_b', 'group_c']  
values = [1, 10, 100]  
  
plt.figure(1, figsize=(9, 3))  
  
plt.subplot(131)  
plt.bar(names, values)  
  
plt.subplot(132)  
plt.scatter(names, values)  
  
plt.subplot(133)  
plt.plot(names, values)  
  
plt.suptitle('Categorical Plotting')  
  
plt.show()
```

绘制分类变量的图表



添加文本

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

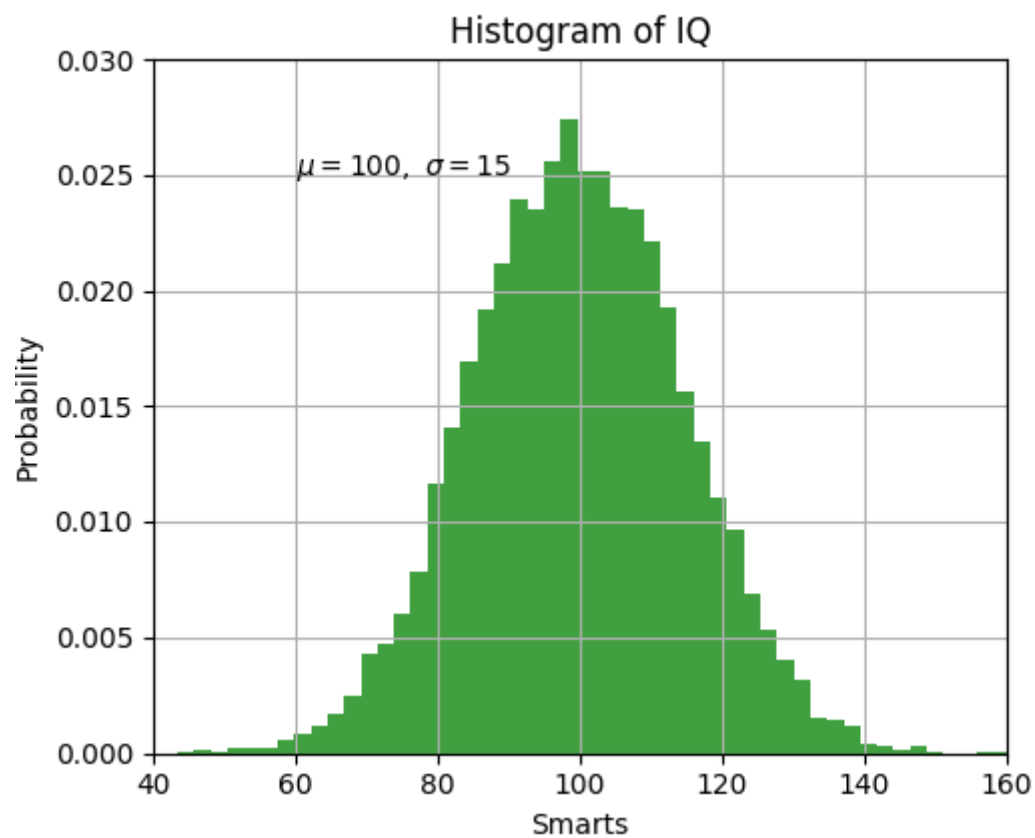
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1, facecolor='g',
alpha=0.75)

plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)

plt.show()
```

添加文本



添加文本注释

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

ax = plt.subplot(111)

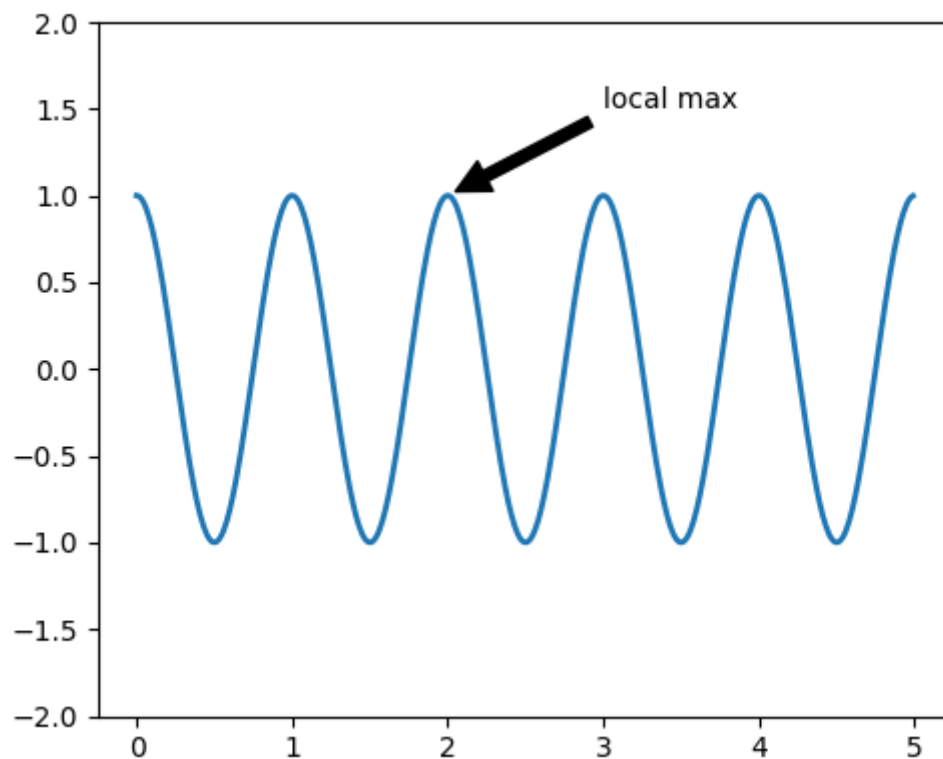
t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)

line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
            arrowprops=dict(facecolor='black', shrink=0.05))
plt.ylim(-2, 2)

plt.show()
```

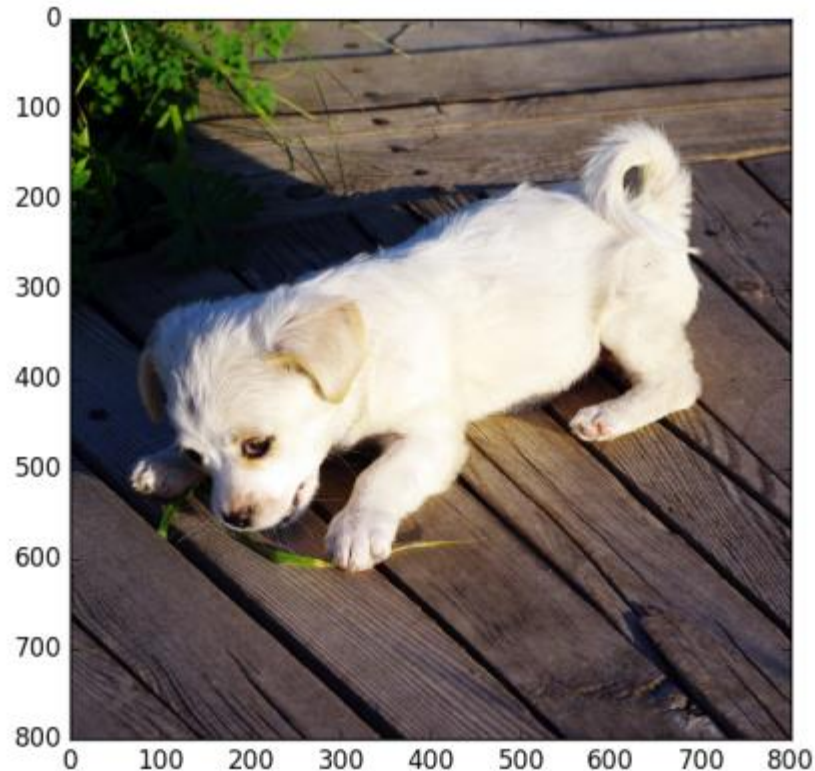
添加文本注释



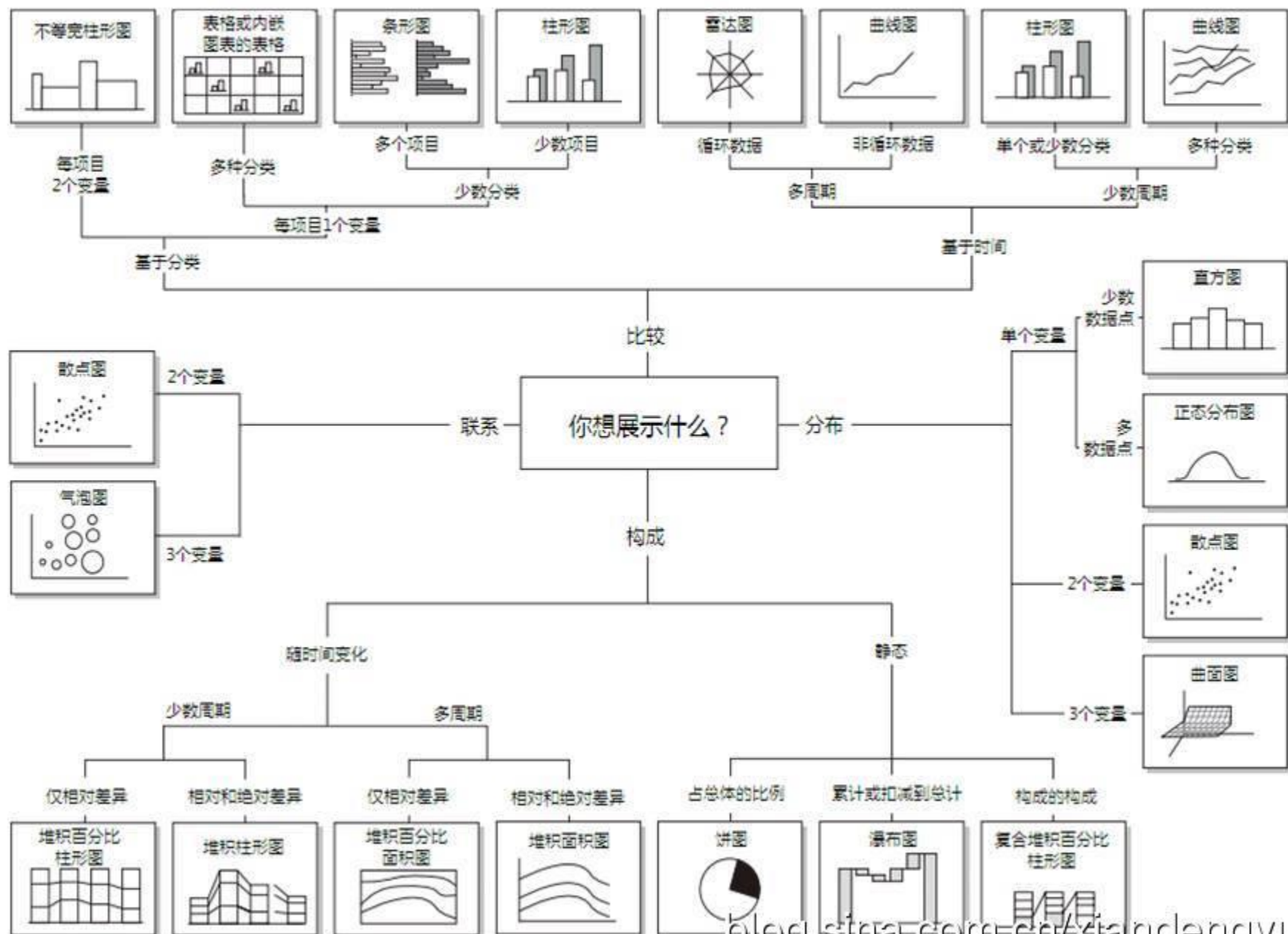
图像显示

```
import matplotlib.pyplot as plt

plt.figure('A Little White Dog')
little_dog_img = plt.imread('little_white_dog.jpg')
plt.imshow(little_dog_img)
plt.show()
```



图表建议—思维指南



Pandas

- Pandas是python的一个数据分析包
- 由AQR Capital Management于2008年4月开发，并于2009年底开源出来
- 导入惯例：

```
>>>from pandas import Series, DataFrame  
>>>import pandas as pd
```

因为Series和DataFrame用的次数非常多，所以将其引入本地命名空间中会更方便

常用数据结构

- Series
 - 一维**标记**数组，由一组**数据**（各种NumPy数据类型）以及一组与之相关的**数据标签**（即索引）组成。
 - 类似于Numpy中的一维数组和Python的列表，不同之处是数组和series中存放的是**相同类型**的元素
- DataFrame
 - 二维表格型数据结构，含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等），
 - 每列都有标签，可看成一个**Series的字典**
- Panel
 - 三维数组，可以理解为DataFrame的容器
 - Panel data源于经济学，也是pan(el)-da(ta)-s的名字来源

创建Series: 传入列表

- 默认整型索引

```
>>>obj = Series([4, 7, -5, 3])
>>>obj
0      4
1      7
2     -5
3      3
dtype: int64

>>>obj.values
array([4, 7, -5, 3], dtype=int64)

>>>obj.index
RangeIndex(start=0, stop=4, step=1)
```

创建Series: 传入列表

- 给定索引

```
>>>obj2 = Series([4,7,-5,3], index=['d','b','a','c'])
>>>obj2
d      4
b      7
a     -5
c      3
dtype: int64

>>> obj2.index
Index(['d', 'b', 'a', 'c'], dtype='object')
```

访问Series中的元素

- 可以使用索引来选取Series中的单个或一组值

```
>>>obj2['a']  
-5  
  
>>>obj2['d']= 6  
  
>>>obj2[['c','a','d']]  
c      3  
a     -5  
d      6  
dtype: int64
```

对Series的操作

- NumPy**数组操作**，如通过一个布尔数组过滤，纯量乘法，或使用数学函数，都会保持索引和值间的关联：

```
>>>obj2[obj2 > 0]
d      4
b      7
c      3
dtype: int64
```

```
>>>obj2*2
d      8
b     14
a    -10
c      6
dtype: int64
```

```
>>>np.exp(obj2)
d      54.598150
b    1096.633158
a      0.006738
c     20.085537
dtype: float64
```

对Series的操作

- 还可将Series看成是一个**定长的有序字典**，因为它是索引值到数据值的一个映射。它可以用在许多原本需要字典参数的函数中：

```
>>>'b' in obj2  
True
```

```
>>>'e' in obj2  
False
```

创建Series: 传入字典

- 如果数据被存放在一个Python字典中，也可以直接通过这个字典来创建Series
- 如果只传入一个字典，则结果Series中的索引就是原字典的键（有序排列）

```
>>>sdata = {'Ohio': 35000, 'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}  
>>>obj3 = Series(sdata)  
>>>obj3  
Ohio          35000  
Oregon        16000  
Texas         71000  
Utah           5000  
dtype: int64
```

创建Series: 传入字典

- 下例中，`sdata`跟`states`索引相匹配的那3个值会被找出来并放到相应的位置上，但由于“California”所对应的`sdata`值找不到，所以其结果就为NaN（Not A Number，非数字）

```
>>>states = ['California', 'Ohio', 'Oregon', 'Texas']
>>>obj4 = Series(sdata, index=states)
>>>obj4
California      NaN
Ohio            35000
Oregon          16000
Texas           71000
dtype: float64
```

检测缺失数据

- pandas的**isnull**和**notnull**函数可用于检测缺失数据
- **Series**也提供了类似的实例方法，如obj4.isnull()

```
>>>pd.isnull(obj4)
California    True
Ohio          False
Oregon        False
Texas         False
dtype: bool
```

```
>>>pd.notnull(obj4)
California    False
Ohio          True
Oregon        True
Texas         True
dtype: bool
```


自动对齐索引

- Series在算术运算中会自动对齐不同索引的数据

```
>>>obj3
Ohio      35000
Oregon    16000
Texas     71000
Utah       5000
dtype: int64
>>>obj4
California    NaN
Ohio          35000
Oregon        16000
Texas         71000
dtype: float64
>>>obj3 + obj4
California    NaN
Ohio          70000
Oregon        32000
Texas        142000
Utah          NaN
dtype: float64
```

Series对象及其索引的name

```
>>>obj4.name = 'population'
```

```
>>>obj4.index.name = 'state'
```

```
>>>obj4
```

```
state
```

```
California      NaN
```

```
Ohio            35000
```

```
Oregon          16000
```

```
Texas           71000
```

```
Name: population, dtype: float64
```

修改索引

```
>>>obj
0      4
1      7
2     -5
3      3

>>>obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']

>>>obj
Bob      4
Steve    7
Jeff    -5
Ryan     3
dtype: int64
```

创建DataFrame (1)

- DataFrame既有行索引也有列索引，它可以被看做由Series组成的字典（共用同一个索引）。
- 最常用的创建方法是直接传入一个由等长列表或NumPy数组构成的字典

```
>>>data={'state':['Ohio','Ohio','Ohio','Nevada','Nevada'],
        'year':[2000, 2001, 2002, 2001, 2002],
        'pop':[1.5, 1.7, 3.6, 2.4, 2.9]}
>>>frame = DataFrame(data)
>>>frame
```

| | pop | state | year |
|---|-----|--------|------|
| 0 | 1.5 | Ohio | 2000 |
| 1 | 1.7 | Ohio | 2001 |
| 2 | 3.6 | Ohio | 2002 |
| 3 | 2.4 | Nevada | 2001 |
| 4 | 2.9 | Nevada | 2002 |

创建DataFrame (1)

- 如果指定了列序列，DataFrame的列就会按指定顺序排列

```
>>> DataFrame(data, columns=['year', 'state', 'pop'])
```

- 跟Series一样，如果传入的列在数据中找不到，就会产生NaN值

```
>>> frame2 = DataFrame(data, columns=['year', 'state', 'pop', 'debt'],  
                        index=['one', 'two', 'three', 'four', 'five'])
```

```
>>> frame2
```

| | year | state | pop | debt |
|-------|------|--------|-----|------|
| one | 2000 | Ohio | 1.5 | NaN |
| two | 2001 | Ohio | 1.7 | NaN |
| three | 2002 | Ohio | 3.6 | NaN |
| four | 2001 | Nevada | 2.4 | NaN |
| five | 2002 | Nevada | 2.9 | NaN |

访问单列

- 通过字典记法或属性，可以将DataFrame的列获取为一个 **Series**:

```
>>>frame2['state']  
one      Ohio  
two      Ohio  
three    Ohio  
four     Nevada  
five     Nevada  
Name: state, dtype: object
```

```
>>>frame2.year  
one      2000  
two      2001  
three    2002  
four     2001  
five     2002  
Name: year, dtype: int64
```

访问单行

- 行也可以使用一些方法通过位置 (**iloc**) 或名字 (**loc**) 来检索

```
>>> frame2.loc['three']  
year      2002  
state     Ohio  
pop        3.6  
debt       NaN  
Name: three, dtype: object
```

```
>>> frame2.iloc[2]  
Out[15]:  
year      2002  
state     Ohio  
pop        3.6  
debt       NaN  
Name: three, dtype: object
```

修改列

```
>>>frame2['debt'] = 16.5
```

```
>>>frame2
```

| | year | state | pop | debt |
|-------|------|--------|-----|------|
| one | 2000 | Ohio | 1.5 | 16.5 |
| two | 2001 | Ohio | 1.7 | 16.5 |
| three | 2002 | Ohio | 3.6 | 16.5 |
| four | 2001 | Nevada | 2.4 | 16.5 |
| five | 2002 | Nevada | 2.9 | 16.5 |

```
>>>frame2['debt'] = np.arange(5)
```

```
>>>frame2
```

| | year | state | pop | debt |
|-------|------|--------|-----|------|
| one | 2000 | Ohio | 1.5 | 0 |
| two | 2001 | Ohio | 1.7 | 1 |
| three | 2002 | Ohio | 3.6 | 2 |
| four | 2001 | Nevada | 2.4 | 3 |
| five | 2002 | Nevada | 2.9 | 4 |

修改列

```
>>>val = Series([-1.2, -1.5, -1.7], index=[ 'two', 'four',  
'five'])  
>>>frame2['debt'] = val  
>>>frame2
```

| | year | state | pop | debt |
|-------|------|--------|-----|------|
| one | 2000 | Ohio | 1.5 | NaN |
| two | 2001 | Ohio | 1.7 | -1.2 |
| three | 2002 | Ohio | 3.6 | NaN |
| four | 2001 | Nevada | 2.4 | -1.5 |
| five | 2002 | Nevada | 2.9 | -1.7 |

增加列和删除列

```
>>>frame2['eastern'] = frame2.state == 'Ohio'
>>>frame2
```

| | year | state | pop | debt | eastern |
|-------|------|--------|-----|------|---------|
| one | 2000 | Ohio | 1.5 | NaN | True |
| two | 2001 | Ohio | 1.7 | -1.2 | True |
| three | 2002 | Ohio | 3.6 | NaN | True |
| four | 2001 | Nevada | 2.4 | -1.5 | False |
| five | 2002 | Nevada | 2.9 | -1.7 | False |

```
>>>del frame2['eastern']
>>>frame2
```

| | year | state | pop | debt |
|-------|------|--------|-----|------|
| one | 2000 | Ohio | 1.5 | NaN |
| two | 2001 | Ohio | 1.7 | -1.2 |
| three | 2002 | Ohio | 3.6 | NaN |
| four | 2001 | Nevada | 2.4 | -1.5 |
| five | 2002 | Nevada | 2.9 | -1.7 |

删除行或列

```
>>> frame2.drop(['pop', 'debt'], axis=1) # 删除pop和debt列
```

| | year | state |
|-------|------|--------|
| one | 2000 | Ohio |
| two | 2001 | Ohio |
| three | 2002 | Ohio |
| four | 2001 | Nevada |
| five | 2002 | Nevada |

```
>>> frame2.drop(columns=['pop', 'debt']) # 删除pop和debt列
```

| | year | state |
|-------|------|--------|
| one | 2000 | Ohio |
| two | 2001 | Ohio |
| three | 2002 | Ohio |
| four | 2001 | Nevada |
| five | 2002 | Nevada |

```
>>> frame2.drop(['one', 'three', 'five'], axis=0) # 删除one, three, five行
```

| | year | state | pop | debt |
|------|------|--------|-----|------|
| two | 2001 | Ohio | 1.7 | -1.2 |
| four | 2001 | Nevada | 2.4 | -1.5 |

```
>>> frame2.drop(['pop', 'debt'], axis=1, inplace=True)
```

创建DataFrame (2)

- 传入嵌套字典（字典的字典），外部键会被解释为列索引，内部键会被解释为行索引：

```
>>>pop = {'Nevada': {2001: 2.4, 2002: 2.9},  
          'Ohio': {2000: 1.5, 2001: 1.7, 2002:3.6}}
```

```
>>>frame3 = DataFrame(pop)
```

```
>>>frame3
```

| | Nevada | Ohio |
|------|--------|------|
| 2000 | NaN | 1.5 |
| 2001 | 2.4 | 1.7 |
| 2002 | 2.9 | 3.6 |

```
>>>frame4 = DataFrame(pop, index=[2001, 2002, 2003])
```

```
>>>frame4
```

| | Nevada | Ohio |
|------|--------|------|
| 2001 | 2.4 | 1.7 |
| 2002 | 2.9 | 3.6 |
| 2003 | NaN | NaN |

缺失数据处理

- 删除任何有缺失数据的行:

```
>>>frame3.dropna(how='any')
      Nevada  Ohio
2001      2.4   1.7
2002      2.9   3.6
```

- 对缺失值进行填充:

```
>>>frame3.fillna(value=5)
      Nevada  Ohio
2000      5.0   1.5
2001      2.4   1.7
2002      2.9   3.6
```

- 判断哪些值是缺失值 (nan) :

```
>>>pd.isna(frame3)
      Nevada  Ohio
2000     True  False
2001    False  False
2002    False  False
```

查看数据

- 查看DataFrame前n行或后n行：

`frame.head(3)`； `frame.tail(3)`

- 查看DataFrame的索引、列以及底层的Numpy数据：

`frame.index`； `frame.columns`； `frame.values`

- 显示数据的快速统计汇总：

`frame.describe()` 对每一列数据进行统计，包括计数、均值、标准差、各个分位数等

- 转置数据：

`frame.T`

- 对轴排序：

`frame.sort_index(axis=1, ascending=False)`，其中axis=1表示对所有的列索引进行排序，下面的数也跟着发生移动。

- 对值排序：

`frame.sort_values(by='Name')` 对name这一列，从小到大进行排序

选择行与列

- 选取多行或多列：

`frame[['state', 'pop']]`，选择'state'和'pop'两列，结果是一个DataFrame

`frame[0:3]`，选择前三行

- `loc`用**标签**选择数据：

`frame.loc['one']`，选择索引为'one'的行

`frame.loc['one', 'pop']`，选择'one'行，'pop'列

`frame.loc[:, ['state', 'pop']]`，选择所有行，'state'和'pop'列

`frame.loc[['one', 'two'], ['state', 'pop']]`，选择'one'和'two'行，'state'和'pop'列

- `iloc`用**位置**选择数据：

`frame.iloc[1:2, 1:2]`

`frame.iloc[[0,2], [1,2]]`

- 使用**条件**来选择：

`frame[frame.year>2001]`，选择year列中大于2001的数据

`frame[frame>2001]`，选择frame中所有大于2001的数据

`frame[frame['year'].isin(['2000','2002'])]`，选择year列的值为'2000','2002'的所有行

相关操作

- 统计数据：
 - `a.mean()`，对a的每一列数据值求平均值；
 - `a.mean(1)`，对a的每一行数据值求平均值
 - `a['x'].value_counts()`，统计列x中各值出现的次数
- 对数据应用函数：
 - `a.apply(lambda x : x.max() - x.min())`，对a的每一列，返回最大值和最小值的差
- 字符串操作：
 - `a['gender1'].str.lower()`，将gender1中所有的英文转化为小写，注意Dataframe没有str属性，只有Series有，所以要选取a中的gender1列。

读取与写入文件

- 写入.csv文件:

```
frame3.to_csv('C:\\Users\\qiuyu\\frame3.csv')
```

- 读取.csv文件:

```
frame4 = pd.read_csv('C:\\Users\\qiuyu\\frame3.csv')
```

```
frame4 = pd.read_csv('C:\\Users\\qiuyu\\frame3.csv', index_col=0)
```