# Frequently Used Python Package

## Requirements

### Install Required Packages

```
pip install -r requirments.txt
```

### Fast Generated the requirements.txt

```
pip freeze > requirements.txt
```

## NumPy

NumPy （Numeric Python）是 Python 中科学计算的基础包，它是一个 Python 库，提供多维数组对象，各种派生对象（如掩码数组和矩阵），以及用于数组快速操作的各种 API，有包括数学、逻辑、形状操作、排序、选择、输入输出、离散傅立叶变换、基本线性代数，基本统计运算和随机模拟等等

NumPy 包的核心是 `ndarray` 对象，它封装了 Python 原生的同数据类型的 `n` 维数组，为了保证其性能优良，其中有许多操作都是代码在本地进行编译后执行的，NumPy 相当于将 Python 变成一种免费的更强大的 MatLab 系统

NumPy 数组 和 原生 Python Array（数组）之间有几个重要的区别

- NumPy 数组在创建时具有固定的大小，与 Python 的原生数组对象（可以动态增长）不同，更改 ndarray 的大小将创建一个新数组并删除原来的数组
- NumPy 数组中的元素都需要具有相同的数据类型，因此在内存中的大小相同，例外情况，Python 的原生数组里包含了 NumPy 的对象的时候，这种情况下就允许不同大小元素的数组
- NumPy 数组有助于对大量数据进行高级数学和其他类型的操作，通常，这些操作的执行效率更高，比使用 Python 原生数组的代码更少
- 越来越多的基于 Python 的科学和数学软件包使用 NumPy 数组; 虽然这些工具通常都支持 Python 的原生数组作为参数，但它们在处理之前会还是会将输入的数组转换为 NumPy 的数组，而且也通常输出为 NumPy 数组，为了高效地使用当今科学/数学基于 Python 的工具（大部分的科学计算工具），知道如何使用 NumPy 数组是必备的

**扩展阅读 【**NumPy 中文文档**】**

## ndarray

### From List to Array

列成 array-like 结构的数值数据可以通过使用 `array()` 函数转换为数组

gocloud

```
In [1]: import numpy as np

In [2]: lst=[[1,2,3],[2,4,6]]

In [3]: type(lst)
Out[3]: list

In [4]: np_lst=np.array(lst)

In [5]: type(np_lst)
Out[5]: numpy.ndarray
```

### Data Types of Arrays

```
In [6]: np_lst=np.array(lst, dtype=np.float)

In [7]: np_lst.dtype
Out[7]: dtype('float64')
```

### Parameters of Array

```
In [8]: np_lst.shape         # 行列数
Out[8]: (2, 3)

In [9]: np_lst.ndim          # 维数
Out[9]: 2

In [10]: np_lst.itemsize     # 每个数据的数据存储大小
Out[10]: 8

In [11]: np_lst.size         # 元素个数
Out[11]: 6
```

## Some kinds of Array

```
In [12]: np.zeros([2, 4])       # 生成 2 行 4 列的全 0 的数组
Out[12]:
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])

In [13]: np.ones([3, 5])        # 生成 3 行 5 列的全 1 的数组
Out[13]:
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

ggcloud

```
In [14]: np.random.rand(2, 4)   # 生成 2 行 4 列数组，每个元素为 0-1 内均匀分布随机数
Out[14]:
array([[0.3492512 , 0.53278383, 0.67421472, 0.37741499],
       [0.13505288, 0.56624554, 0.05743534, 0.47994088]])

In [15]: np.random.randint(1, 10, 3)    # 生成 3 个 1-10 内随机分布整数
Out[15]: array([6, 3, 5])

In [16]: np.random.randn(2, 4)          # 生成 2 行 4 列的标准正态随机数数组
Out[16]:
array([[ 0.7297433 , -1.31910919,  1.3258419 , -0.37062597],
       [ 0.91714998,  2.0291667 ,  0.59648187, -1.54048607]])

In [17]: np.random.choice([10, 20, 30]) # 指定范围内的随机数
Out[17]: 20

In [18]: np.random.beta(1, 10, 20)      # 生成一个包含 20 个元素满足 Beta 分布的数组
Out[18]:
array([0.01745944, 0.19434248, 0.08223912, 0.04432289, 0.2939484 ,
       0.13065389, 0.05528825, 0.20747935, 0.00320723, 0.11942977,
       0.00388593, 0.00574769, 0.07600872, 0.08523846, 0.13702178,
       0.01265392, 0.11381335, 0.01214367, 0.0733919 , 0.0779095 ])
```

# Array Opeartion

## Mathematical Operations of Array

```
In [19]: lst = np.arange(1, 11).reshape([2, 5])

In [20]: lst
Out[20]:
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])

In [21]: np.exp(lst)
Out[21]:
array([[2.71828183e+00, 7.38905610e+00, 2.00855369e+01, 5.45981500e+01,
1.48413159e+02],
       [4.03428793e+02, 1.09663316e+03, 2.98095799e+03, 8.10308393e+03,
2.20264658e+04]])

In [22]: np.exp2(lst)
Out[22]:
array([[   2.,    4.,    8.,   16.,   32.],
       [  64.,  128.,  256.,  512., 1024.]])

In [23]: np.sqrt(lst)
```

ggcloud

```
Out[23]:
array([[1.        , 1.41421356, 1.73205081, 2.        , 2.23606798],
       [2.44948974, 2.64575131, 2.82842712, 3.        , 3.16227766]])

In [24]: np.sin(lst)
Out[24]:
array([[ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427],
       [-0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849, -0.54402111]])

In [25]: np.log(lst)
Out[25]:
array([[0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791],
       [1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509]])
```

## Descriptive Statistics for Array

```
In [26]: lst = np.array([[[1, 2, 3, 4], [4, 5, 6, 7]],
    ...:                 [[7, 8, 9, 10], [10, 11, 12, 13]],
    ...:                 [[14, 15, 16, 17], [18, 19, 20, 21]]])

In [27]: lst
Out[27]:
array([[[ 1,  2,  3,  4],
        [ 4,  5,  6,  7]],

       [[ 7,  8,  9, 10],
        [10, 11, 12, 13]],

       [[14, 15, 16, 17],
        [18, 19, 20, 21]]])

In [28]: lst.sum()          # 所有元素求和
Out[28]: 252

In [29]: lst.sum(axis=0)    # 最外层求和
Out[29]:
array([[22, 25, 28, 31],
       [32, 35, 38, 41]])

In [30]: lst.sum(axis=1)    # 第二层求和
Out[30]:
array([[ 5,  7,  9, 11],
       [17, 19, 21, 23],
       [32, 34, 36, 38]])

In [31]: lst.sum(axis=-1)   # 最里层求和
Out[31]:
array([[10, 22],
       [34, 46],
```

99cloud

```
       [62, 78]])

In [32]: lst.max()
Out[32]: 21

In [33]: lst.min()
Out[33]: 1
```

## Operations between Arrays

```
In [34]: lst1 = np.array([10, 20, 30, 40])

In [35]: lst2 = np.array([4, 3, 2, 1])

In [36]: lst1 + lst2
Out[36]: array([14, 23, 32, 41])

In [37]: lst1 - lst2
Out[37]: array([ 6, 17, 28, 39])

In [38]: lst1 * lst2
Out[38]: array([40, 60, 60, 40])

In [39]: lst1 / lst2
Out[39]: array([ 2.5       ,  6.66666667, 15.        , 40.        ])

In [40]: lst1 ** lst2
Out[40]: array([10000,  8000,   900,    40])

In [41]: np.dot(lst1.reshape([2, 2]), lst2.reshape([2, 2]))
Out[41]:
array([[ 80,  50],
       [200, 130]])

In [42]: np.concatenate((lst1, lst2), axis=0)   # 向量拼接
Out[42]: array([10, 20, 30, 40,  4,  3,  2,  1])

In [43]: np.vstack((lst1, lst2))                # 按照行拼接
Out[43]:
array([[10, 20, 30, 40],
       [ 4,  3,  2,  1]])

In [44]: np.hstack((lst1, lst2))                # 按照列拼接
Out[44]: array([10, 20, 30, 40,  4,  3,  2,  1])

In [45]: np.split(lst1, 2)                      # 向量拆分
Out[45]: [array([10, 20]), array([30, 40])]

In [46]: np.copy(lst1)                          # 向量拷贝
```

99cloud

```
Out[46]: array([10, 20, 30, 40])
```

## Liner Algebra

```
In [47]: np.eye(3)      # 生成单位矩阵
Out[47]:
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])

In [48]: from numpy.linalg import *

In [49]: lst = np.array([[1, 2],
    ...:                 [3, 4]])

In [50]: inv(lst)       # 生成给定矩阵的逆矩阵
Out[50]:
array([[-2. ,  1. ],
       [ 1.5, -0.5]])

In [51]: lst.transpose()    # 生成给定矩阵的转置
Out[51]:
array([[1, 3],
       [2, 4]])

In [52]: det(lst)       # 求矩阵的行列式
Out[52]: -2.0000000000000004

In [53]: eig(lst)       # 求矩阵的特征值和特征向量
Out[53]:
(array([-0.37228132,  5.37228132]),
 array([[-0.82456484, -0.41597356],
        [ 0.56576746, -0.90937671]]))
```

**注意**

关于特征值和特征向量，例子中的两个特征值分别为

$$\lambda_1 = -0.37228132, \quad \lambda_2 = 5.37228132$$

对应的特征向量分别为

$$\xi_1 = \begin{bmatrix} -0.82456484 \\ 0.56576746 \end{bmatrix}, \quad \xi_2 = \begin{bmatrix} -0.41597356 \\ -0.90937671 \end{bmatrix}$$

可以验证如下等式

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} -0.82456484 \\ 0.56576746 \end{bmatrix} = -0.37228132 \times \begin{bmatrix} -0.82456484 \\ 0.56576746 \end{bmatrix}$$

99cloud

### Solve Linear Equations

```
In [54]: y = np.array([[5], [7]])

In [55]: solve(lst, y)
Out[55]:
array([[-3.],
       [ 4.]])
```

**注意** 相当于求解如下线性方程组

$$\begin{cases} x + 2y = 5 \\ 3x + 4y = 7 \end{cases}$$

## NumPy Others

```
In [56]: np.corrcoef([1, 0, 1], [0, 2, 1])
Out[56]:
array([[ 1.       , -0.8660254],
       [-0.8660254,  1.       ]])

In [57]: p = np.poly1d([2, 1, 3])        # 定义一元多项式 2 * x^2 + x + 3

In [58]: p(0.5)      # 多项式在 x = 0.5 时的值
Out[58]: 4.0

In [59]: p.r         # 多项式等于 0 时的根
Out[59]: array([-0.25+1.19895788j, -0.25-1.19895788j])

In [60]: q = np.poly1d([2, 1, 3], True)   # 把数组中的值作为根，反推多项式

In [61]: print(q)
   3     2
1 x - 6 x + 11 x - 6
```

**注意** 把数组中的值作为根，反推多项式，即

$$(x-2)(x-1)(x-3) = x^3 - 6x^2 + 11x - 6$$

# Matplotlib

如果要想象两个变量之间的关系，想要显示值随时间变化，就需要用到可视化工具

简单来说，Matplotlib 提供图形可视化 Python 包，它提供了一种高度交互式界面，便于用户能够做出各种有吸引力的统计图表

我们只需几行代码就可以生成图表、直方图、功率谱、条形图、误差图、散点图等

ggCloud

为了简单绘图，该 `pyplot` 模块提供了类似于 MATLAB 的界面，尤其是与 IPython 结合使用时。 对于高级用户，您可以通过面向对象的界面或 MATLAB 用户熟悉的一组功能来完全控制线型，字体属性，轴属性等

**扩展阅读** 【Matplotlib 中文文档】

**特别提醒**，当使用 Jupyter 时，需要增加如下命令

```
%matplotlib inline
```

具体作用是当你调用 `matplotlib.pyplot` 的绘图函数 `plot()` 等进行绘图的时候，或者生成一个 `figure` 画布的时候，可以直接在你的 Python Console 里面生成图像，但在 Python 的 IDE 如 Spyder 或者 Pycharm 中需要注释掉

## Line

```
In [62]: import matplotlib.pyplot as plt

In [63]: def plt1():
    ...:     x = np.linspace(-np.pi, np.pi, 256, endpoint=True)
    ...:     c, s = np.cos(x), np.sin(x)
    ...:     plt.figure(figsize=(10, 8))
    ...:     # plt.plot(x, c)
    ...:     plt.plot(x, c, color="blue", linewidth=1.5, linestyle="-",
    ...:              label="COS", alpha=0.6)     # 散点图
    ...:     plt.plot(x, s, "r*", label="SIN", alpha=0.6)
    ...:     plt.title("Cos & Sin", size=16)     # 标题
    ...:     ax = plt.gca()  # 轴编辑器
    ...:     ax.spines["right"].set_color("none")
    ...:     ax.spines["top"].set_color("none")
    ...:     ax.spines["left"].set_position(("data", 0))
    ...:     ax.spines["bottom"].set_position(("data", 0))
    ...:     ax.xaxis.set_ticks_position("bottom")
    ...:     ax.yaxis.set_ticks_position("left")
    ...:     plt.xticks([-np.pi, -np.pi / 2, 0, np.pi / 2, np.pi],
    ...:                [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$\pi/2$',
r'$\pi$'])  # 正则表达
    ...:     plt.yticks(np.linspace(-1, 1, 5, endpoint=True))
    ...:
    ...:     for label in ax.get_xticklabels() + ax.get_yticklabels():
    ...:         label.set_fontsize(16)
    ...:     label.set_bbox(dict(facecolor="white", edgecolor="none",
alpha=0.2))
    ...:     plt.legend(loc="upper left")    # 图例位置
    ...:     plt.grid()                      # 网格线
    ...:     # fill
    ...:     plt.fill_between(x, np.abs(x) < 0.5, c, c > 0.5,
color="green", alpha=0.25)
    ...:     t = 1
```

gocloud

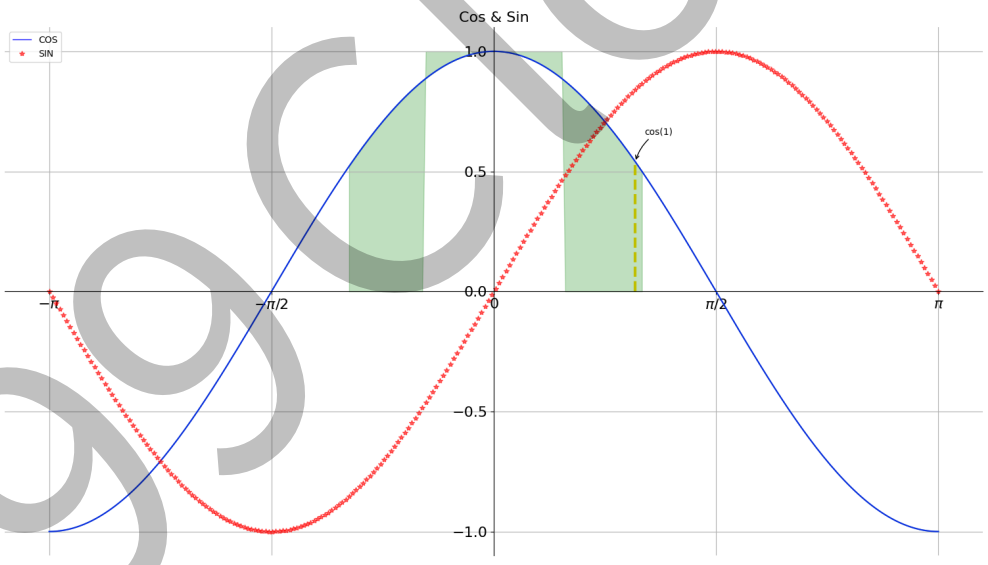
```
    ...:     plt.plot([t, t], [0, np.cos(t)], "y", linewidth=3,
linestyle="--") # 橙色虚线
    ...:     plt.annotate("cos(1)", xy=(t, np.cos(1)), xycoords="data",
    ...:                  xytext=(+10, +30), textcoords="offset points",
    ...:                  arrowprops=dict(arrowstyle="->",
connectionstyle="arc3,rad=.2"))
    ...:     plt.show()      # 显示

In [64]: plt1()
```

在填充画图代码中

```
plt.fill_between(x, np.abs(x) < 0.5, c, c > 0.5, color="green", alpha=0.25)
```

第一个参数 `x` 表示 $x$ 轴，第二个参数 `np.abs(x)` 表示 $x$ 的绝对值，`np.abs(x) < 0.5` 是一个判定变量，`c` 表示 $y$ 轴，`c > 0.5` 是一个判定条件

- 当 `np.abs(x) < 0.5` 为 `True`（即值为 `1`），从 $y$ 轴的 $1$（满足 $c > 0.5$）开始往两边填充（当然 $x$ 轴上是 $-0.5$ 到 $0.5$ 之间的区域），此时填充的也就是图上方的两小块
- 当 `np.abs(x) >= 0.5` 为 `False`（即值为 `0`），从 $y$ 轴的 $0$ 开始向上填充，当然只填充 $c > 0.5$ 的区域，也就是图中那两块大的对称区域

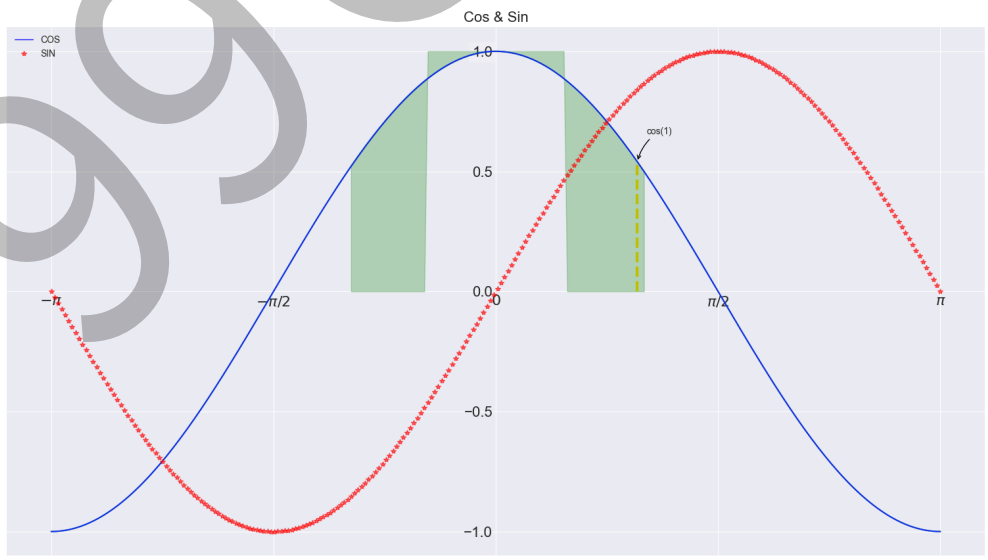



## Style

```
In [65]: plt.style.available       # 查看可用画风
Out[65]:
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
```

99cloud

```
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']

In [66]: plt.style.use('seaborn-dark')     # 应用风格

In [67]: plt1()

In [68]: plt.style.use('default')          # 重回默认风格
```



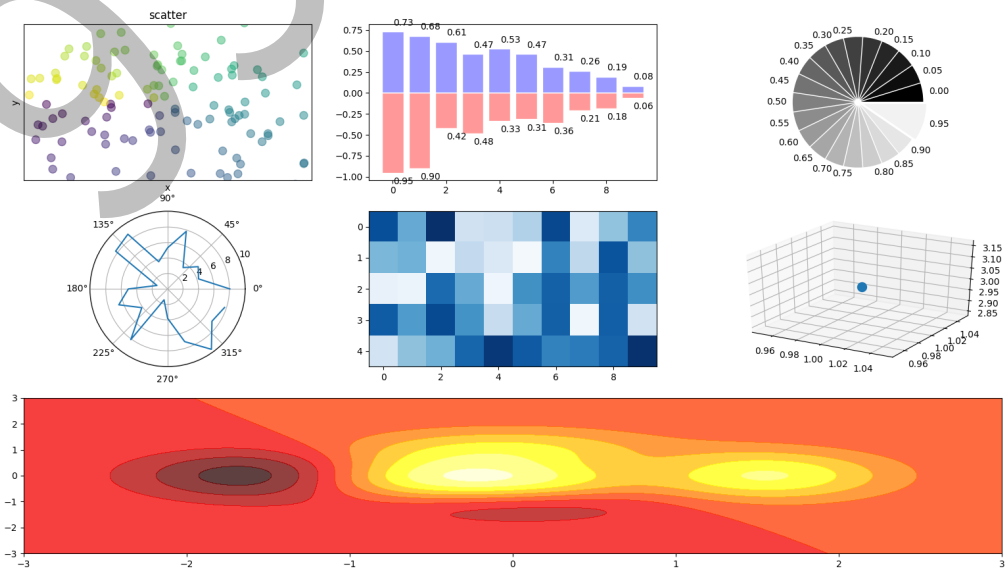

99cloud

## Many types of Figures

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


def plt2():
    fig = plt.figure()
    # scatter
    ax = fig.add_subplot(3, 3, 1)
    n = 128
    X = np.random.normal(0, 1, n)
    Y = np.random.normal(0, 1, n)
    T = np.arctan2(Y, X)
    # plt.axes([0.025, 0.025, 0.95, 0.95])
    plt.scatter(X, Y, s=75, c=T, alpha=.5)
    plt.xlim(-1.5, 1.5), plt.xticks([])
    plt.ylim(-1.5, 1.5), plt.yticks([])
    plt.axis()
    plt.title("scatter")
    plt.xlabel("x")
    plt.ylabel("y")

    # bar
    fig.add_subplot(332)
    n = 10
    X = np.arange(n)
    Y1 = (1 - X / float(n)) * np.random.uniform(0.5, 1, n)
    Y2 = (1 - X / float(n)) * np.random.uniform(0.5, 1, n)
    plt.bar(X, +Y1, facecolor='#9999ff', edgecolor='white')
    plt.bar(X, -Y2, facecolor='#ff9999', edgecolor='white')
    for x, y in zip(X, Y1):
        plt.text(x + 0.4, y + 0.05, '%.2f' % y, ha='center', va='bottom')
    for x, y in zip(X, Y2):
        plt.text(x + 0.4, - y - 0.05, '%.2f' % y, ha='center', va='top')

    # Pie
    fig.add_subplot(333)
    n = 20
    Z = np.ones(n)
    Z[-1] *= 2
    # explode扇形离中心距离
    plt.pie(Z, explode=Z * .05, colors=['%f' % (i / float(n)) for i in
range(n)],
            labels=['%.2f' % (i / float(n)) for i in range(n)])
    plt.gca().set_aspect('equal')  # 圆形
    plt.xticks([]), plt.yticks([])

    # polar
    fig.add_subplot(334, polar=True)
    n = 20
```

99cloud

```
    theta = np.arange(0, 2 * np.pi, 2 * np.pi / n)
    radii = 10 * np.random.rand(n)
    plt.polar(theta, radii)
    # plt.plot(theta, radii)

    # heatmap
    fig.add_subplot(335)
    from matplotlib import cm
    data = np.random.rand(5, 10)
    cmap = cm.Blues
    map = plt.imshow(data, interpolation='nearest', cmap=cmap,
aspect='auto', vmin=0, vmax=1)

    # 3D
    from mpl_toolkits.mplot3d import Axes3D
    ax = fig.add_subplot(336, projection="3d")
    ax.scatter(1, 1, 3, s=100)

    # hot map
    fig.add_subplot(313)

    def f(x, y):
        return (1 - x / 2 + x ** 5 + y ** 3) * np.exp(- x ** 2 - y ** 2)

    n = 256
    x = np.linspace(-3, 3, n * 2)
    y = np.linspace(-3, 3, n)
    X, Y = np.meshgrid(x, y)
    plt.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap=plt.cm.hot)

    plt.show()      # 显示
```




ggcloud

# Scipy

SciPy 函数库在 NumPy 库的基础上增加了众多的数学、科学以及工程计算中常用的库函数，例如线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等等

**扩展阅读【SciPy 官网】**

## Solving Nonlinear Equations

optimize 库中的 `fsolve` 函数可以用来对非线性方程组进行求解，它的基本调用形式如下

```
fsolve(func, x0)
```

`func(x)` 是计算方程组误差的函数，它的参数 $x$ 是一个矢量，表示方程组的各个未知数的一组可能解，`func` 返回将 $x$ 代入方程组之后得到的误差；$x_0$ 为未知数矢量的初始值，如果要对如下方程组进行求解的话

$$\begin{cases} f_1(u_1,u_2,u_3) = 0 \\ f_2(u_1,u_2,u_3) = 0 \\ f_3(u_1,u_2,u_3) = 0 \end{cases}$$

那么 `func` 可以如下定义

```
def func(x):
    u1,u2,u3 = x
    return [f1(u1,u2,u3), f2(u1,u2,u3), f3(u1,u2,u3)]
```

下面是一个实际的例子，求解如下方程组的解

$$\begin{cases} 5 \cdot x_1 + 3 = 0 \\ 4 \cdot {x_0}^2 - 2\sin(x_1 \cdot x_2) = 0 \\ x_1 \cdot x_2 - 1.5 = 0 \end{cases}$$

程序如下

```
from scipy.optimize import fsolve
from math import sin


def f(x):
    x0 = float(x[0])
    x1 = float(x[1])
    x2 = float(x[2])
    return [
        5 * x1 + 3,
        4 * x0 * x0 - 2 * sin(x1 * x2),
        x1 * x2 - 1.5
    ]
```

99cloud

```
14
15
16 result = fsolve(f, [1, 1, 1])
17
18 print('[x0,x1,x2] =', result)
19 print('[f1,f2,f3] =', f(result))
```

输出为

```
1 [x0,x1,x2] = [-0.70622057 -0.6        -2.5       ]
2 [f1,f2,f3] = [0.0, -9.126033262418787e-14, 5.329070518200751e-15]
```

由于 `fsolve` 函数在调用函数f时，传递的参数为数组，因此如果直接使用数组中的元素计算的话，计算速度将会有所降低，因此这里先用 `float` 函数将数组中的元素转换为 Python 中的标准浮点数，然后调用标准 math 库中的函数进行运算

## Function Maximum

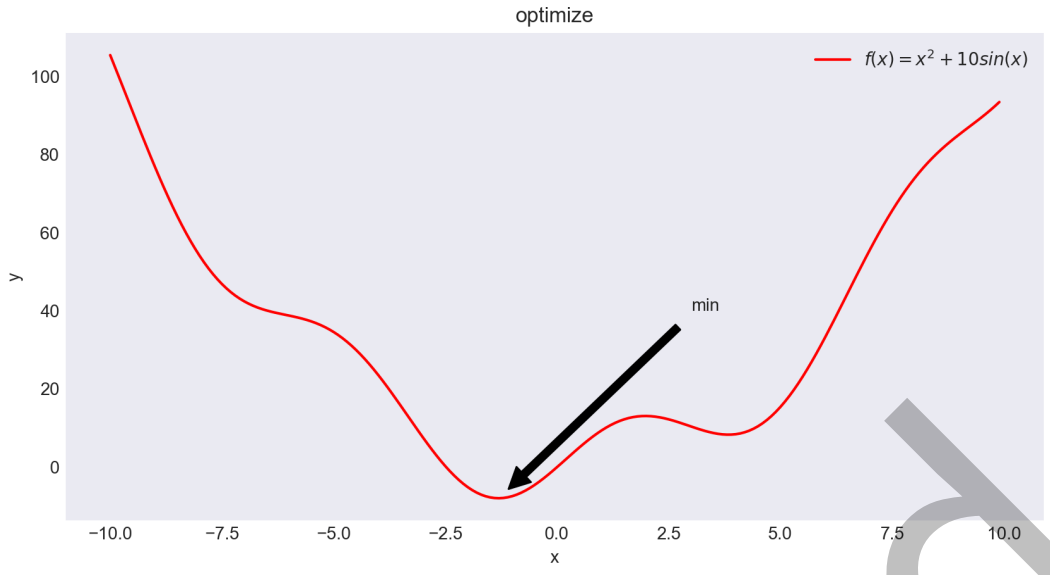
以寻找函数

$$f(x) = x^2 + 10sin(x)$$

的最小值为例，首先绘制目标函数的图形

```
1  from scipy import optimize
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5
6  # 定义目标函数
7  def f(x):
8      return x ** 2 + 10 * np.sin(x)
9
10
11 # 绘制目标函数的图形
12 plt.figure(figsize=(10, 5))
13 x = np.arange(-10, 10, 0.1)
14 plt.xlabel('x')
15 plt.ylabel('y')
16 plt.title('optimize')
17 plt.plot(x, f(x), 'r-', label='$f(x)=x^2+10sin(x)$')
18 # 图像中的最低点函数值
19 a = f(-1.3)
20 plt.annotate('min', xy=(-1.3, a), xytext=(3, 40),
   arrowprops=dict(facecolor='black', shrink=0.05))
21 plt.legend()
22 plt.show()
```

99cloud




显然这是一个非凸优化问题，对于这类函数得最小值问题一般是从给定的初始值开始进行一个梯度下降，在 optimize 中一般使用 `bfgs` 算法

```
optimize.fmin_bfgs(f, 0)
```

结果显示在经过五次迭代之后找到了一个局部最低点 `-7.945823`，显然这并不是函数的全局最小值，只是该函数的一个局部最小值，这也是拟牛顿算法（BFGS）的局限性，如果一个函数有多个局部最小值，拟牛顿算法可能找到这些局部最小值而不是全局最小值，这取决与初始点的选取

```
Optimization terminated successfully.
         Current function value: -7.945823
         Iterations: 5
         Function evaluations: 18
         Gradient evaluations: 6
```

在我们不知道全局最低点，并且使用一些临近点作为初始点，那将需要花费大量的时间来获得全局最优，此时可以采用暴力搜寻算法，它会评估范围网格内的每一个点，对于本例，如下

```
grid = (-10, 10, 0.1)
xmin_global = optimize.brute(f, (grid,))
print(xmin_global)
```

搜寻结果如下

```
[-1.30641113]
```

但是当函数的定义域大到一定程度时，`scipy.optimize.brute()` 变得非常慢，`scipy.optimize.basinhopping()` 提供了一个解决思路

ggcloud

```
x0 = -10
xmin_global_2 = optimize.basinhopping(f, x0, stepsize=5).x
print(xmin_global_2)
```

搜寻结果如下

```
[-1.30644]
```

## Least Square Fitting

假设有一组实验数据 $(x[i], y[i])$，我们知道它们之间的函数关系 $y = f(x)$，通过这些已知信息，需要确定函数中的一些参数项

例如，如果 $f$ 是一个线型函数 $f(x) = k \times x + b$，那么参数 $k$ 和 $b$ 就是我们需要确定的值，如果将这些参数用 $\mathbf{p}$ 表示的话，那么我们就是要找到一组 $*\mathbf{p}$ 值使得如下公式中的 $S$ 函数最小

$$S(\mathbf{p}) = \sum_{i=1}^{m}[y_i - f(x_i, \mathbf{p})]^2$$

这种算法被称之为最小二乘拟合（Least-square fitting）

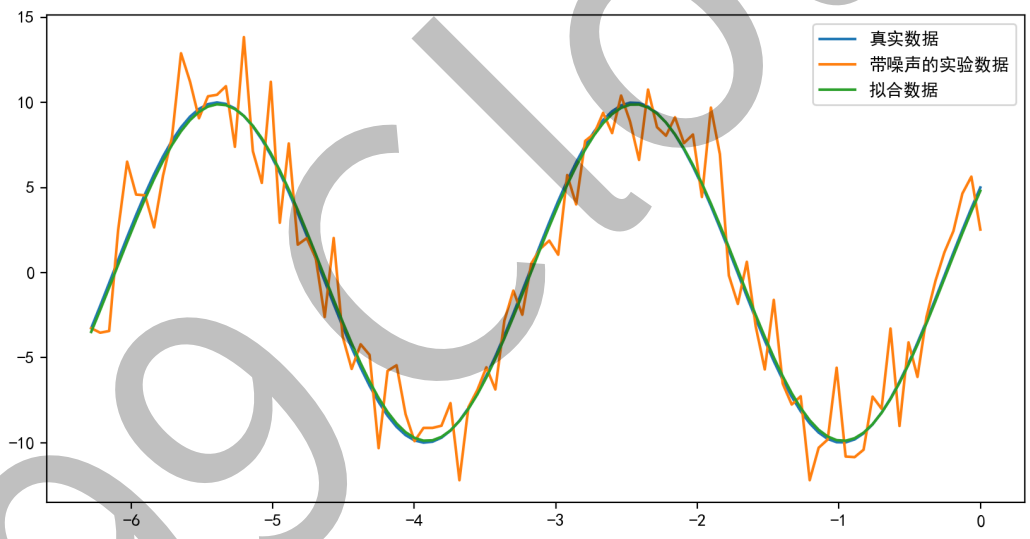
scipy 中的子函数库 optimize 已经提供了实现最小二乘拟合算法的函数 `leastsq`

下面是用 `leastsq` 进行数据拟合的一个例子

```
import numpy as np
from scipy.optimize import leastsq
import pylab as pl
pl.mpl.rcParams['font.sans-serif'] = ['SimHei']
pl.mpl.rcParams['axes.unicode_minus'] = False


def func(x, p):
    """
    数据拟合所用的函数: A*sin(2*pi*k*x + theta)
    """
    A, k, theta = p
    return A*np.sin(2*np.pi*k*x+theta)


def residuals(p, y, x):
    """
    实验数据x, y和拟合函数之间的差，p为拟合需要找到的系数
    """
    return y - func(x, p)


x = np.linspace(0, - 2 * np.pi, 100)
A, k, theta = 10, 0.34, np.pi/6          # 真实数据的函数参数
```

ggcloud

```
y0 = func(x, [A, k, theta])                 # 真实数据
y1 = y0 + 2 * np.random.randn(len(x))       # 加入噪声之后的实验数据

p0 = [7, 0.2, 0]                            # 第一次猜测的函数拟合参数

# 调用 leastsq 进行数据拟合
# residuals 为计算误差的函数
# p0 为拟合参数的初始值
# args 为需要拟合的实验数据
plsq = leastsq(residuals, p0, args=(y1, x))

print(u"真实参数:", [A, k, theta])
print(u"拟合参数:", plsq[0])  # 实验数据拟合后的参数

pl.plot(x, y0, label=u"真实数据")
pl.plot(x, y1, label=u"带噪声的实验数据")
pl.plot(x, func(x, plsq[0]), label=u"拟合数据")
pl.legend()
pl.show()
```




输出结果

```
真实参数: [10, 0.34, 0.5235987755982988]
拟合参数: [10.22216161  0.34359989  0.50580946]
```

这个例子中我们要拟合的函数是一个正弦波函数，它有三个参数 $\mathbf{A}$, $\mathbf{k}$, $\mathbf{theta}$，分别对应振幅、频率、相角，假设我们的实验数据是一组包含噪声的数据 $x, y_1$，其中 $y_1$ 是在真实数据 $y_0$ 的基础上加入噪声得到的

# Pandas

99cloud

Pandas 是一个开源的，BSD 许可的库，为 Python 编程语言提供高性能，易于使用的数据结构和数据分析工具

**扩展阅读 【**Pandas 中文文档**】**

## Series & DataFrame

```
In [1]: import numpy as np

In [2]: import pandas as pd

In [3]: s = pd.Series([i * 2 for i in range(1, 11)])

In [4]: s
Out[4]:
0     2
1     4
2     6
3     8
4    10
5    12
6    14
7    16
8    18
9    20
dtype: int64

In [5]: type(s)
Out[5]: pandas.core.series.Series

In [6]: dates = pd.date_range("20200501", periods=8)

In [7]: df = pd.DataFrame(np.random.randn(8, 5), index=dates,
columns=list("ABCDE"))

In [8]: df
Out[8]:
                   A         B         C         D         E
2020-05-01 -0.910682 -0.780347  0.361256  0.050828  1.065491
2020-05-02 -1.555258 -0.989474  0.913899  1.421703  0.798911
2020-05-03  0.238558 -0.126182 -0.142121 -1.289292 -1.727895
2020-05-04  1.981506  0.723262  1.567475 -1.646015 -0.279556
2020-05-05  0.542094 -0.527112  0.140162  1.093197  0.953332
2020-05-06 -0.428349 -1.180154 -1.219545  0.590974  0.544332
2020-05-07  1.888931 -0.959792  0.747710 -0.862430 -1.560378
2020-05-08 -1.523562 -0.357324 -0.200601  0.160235 -0.229250

In [9]: df = pd.DataFrame({"A": 1,
   ...:                    "B": pd.Timestamp("20200501"),
```

gocloud

```
   ...:                     "C": pd.Series(1, index=list(range(4)),
dtype="float32"),
   ...:                     "D": np.array([3] * 4, dtype="float32"),
   ...:                     "E": pd.Categorical(["police", "student",
"teacher", "doctor"])})

In [10]: df
Out[10]:
   A          B    C    D        E
0  1 2020-05-01  1.0  3.0   police
1  1 2020-05-01  1.0  3.0  student
2  1 2020-05-01  1.0  3.0  teacher
3  1 2020-05-01  1.0  3.0   doctor
```

# Basic & Select & Set

我们先重新设置一个 `df`

```
In [11]: dates = pd.date_range("20200501", periods=8)

In [12]: df = pd.DataFrame(np.random.randn(8, 5), index=dates,
columns=list("ABCDE"))

In [13]: df
Out[13]:
                   A         B         C         D         E
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240
2020-05-05  0.343573  2.268671  0.123661  0.026515  0.819831
2020-05-06  1.149055  0.514129  0.653298  1.014948  0.364408
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486
```

## Basic Operation of DataFrame

```
In [14]: df.head(3)
Out[14]:
                   A         B         C         D         E
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396

In [15]:  df.tail(3)
Out[15]:
                   A         B         C         D         E
2020-05-06  1.149055  0.514129  0.653298  1.014948  0.364408
```

ggcloud

```
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486

In [16]: df.index
Out[16]:
DatetimeIndex(['2020-05-01', '2020-05-02', '2020-05-03', '2020-05-04',
               '2020-05-05', '2020-05-06', '2020-05-07', '2020-05-08'],
              dtype='datetime64[ns]', freq='D')

In [17]: df.values
Out[17]:
array([[-0.63960556, -0.49076281,  0.83405679,  0.19167819,  0.54413525],
       [ 0.74831337,  0.54242504, -0.50424864, -1.00617742,  1.08873712],
       [ 0.56241309, -1.36623581, -0.9994565 ,  1.43409736,  0.08839574],
       [-0.01196919,  1.60814395,  1.0709043 ,  1.89549629,  1.37623995],
       [ 0.34357293,  2.26867082,  0.12366068,  0.02651487,  0.8198315 ],
       [ 1.14905497,  0.51412852,  0.65329835,  1.01494828,  0.36440802],
       [-0.50425931,  0.64804705, -1.17064214,  1.19895757,  0.7565752 ],
       [-0.52141233,  0.1762601 , -0.09084731,  0.0203732 ,  0.04748562]])

In [18]: df.T
Out[18]:
   2020-05-01  2020-05-02  2020-05-03  ...  2020-05-06  2020-05-07  2020-
05-08
A   -0.639606    0.748313    0.562413  ...    1.149055   -0.504259
-0.521412
B   -0.490763    0.542425   -1.366236  ...    0.514129    0.648047
0.176260
C    0.834057   -0.504249   -0.999457  ...    0.653298   -1.170642
-0.090847
D    0.191678   -1.006177    1.434097  ...    1.014948    1.198958
0.020373
E    0.544135    1.088737    0.088396  ...    0.364408    0.756575
0.047486

[5 rows x 8 columns]

In [19]: df.sort_values(by="C", ascending=False)
Out[19]:
                   A         B         C         D         E
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135
2020-05-06  1.149055  0.514129  0.653298  1.014948  0.364408
2020-05-05  0.343573  2.268671  0.123661  0.026515  0.819831
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575

```

ggcloud

```
In [20]: df.sort_index(axis=1, ascending=False)
Out[20]:
                   E         D         C         B         A
2020-05-01  0.544135  0.191678  0.834057 -0.490763 -0.639606
2020-05-02  1.088737 -1.006177 -0.504249  0.542425  0.748313
2020-05-03  0.088396  1.434097 -0.999457 -1.366236  0.562413
2020-05-04  1.376240  1.895496  1.070904  1.608144 -0.011969
2020-05-05  0.819831  0.026515  0.123661  2.268671  0.343573
2020-05-06  0.364408  1.014948  0.653298  0.514129  1.149055
2020-05-07  0.756575  1.198958 -1.170642  0.648047 -0.504259
2020-05-08  0.047486  0.020373 -0.090847  0.176260 -0.521412

In [21]: df.describe()      # 描述统计
Out[21]:
              A         B         C         D         E
count  8.000000  8.000000  8.000000  8.000000  8.000000
mean   0.140763  0.487585 -0.010409  0.596986  0.635726
std    0.664565  1.130620  0.837986  0.949908  0.467467
min   -0.639606 -1.366236 -1.170642 -1.006177  0.047486
25%   -0.508548  0.009504 -0.628051  0.024979  0.295405
50%    0.165802  0.528277  0.016407  0.603313  0.650355
75%    0.608888  0.888071  0.698488  1.257743  0.887058
max    1.149055  2.268671  1.070904  1.895496  1.376240
```

## Selection Operations of DataFrame

```
In [22]: type(df["A"])
Out[22]: pandas.core.series.Series

In [23]: df[:3]
Out[23]:
                   A         B         C         D         E
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396

In [24]: df.head(3)
Out[24]:
                   A         B         C         D         E
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396

In [25]: df["20200501": "20200504"]
Out[25]:
                   A         B         C         D         E
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396
```

ggcloud

```
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240

In [26]: df.loc["20200501": "20200504", ["B", "D"]]     # 行名和列名
Out[26]:
                   B         D
2020-05-01 -0.490763  0.191678
2020-05-02  0.542425 -1.006177
2020-05-03 -1.366236  1.434097
2020-05-04  1.608144  1.895496

In [27]: df.at[dates[0], "C"]
Out[27]: 0.8340567905719413

In [28]: df.iloc[1:3, 2:4]                # 行号和列号
Out[28]:
                   C         D
2020-05-02 -0.504249 -1.006177
2020-05-03 -0.999457  1.434097

In [29]: df.iloc[1, 4]
Out[29]: 1.0887371182725154

In [30]: df.iat[1, 4]
Out[30]: 1.0887371182725154
```

## Data Intercepted by Judgment in DataFrame

```
In [31]: df[df.B > 0][df.A < 0]
....../anaconda3/envs/AIC/bin/ipython:1: UserWarning: Boolean Series key
will be reindexed to match DataFrame index.
  #!....../anaconda3/envs/AIC/bin/python
Out[31]:
                   A         B         C         D         E
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486

In [32]: df[(df.B > 0) & (df.A < 0)]
Out[32]:
                   A         B         C         D         E
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486
```

**布尔型系列键将索引匹配获得对应的索引**

我每一个判断之后，都会返回一个 `True` 和 `False` 的索引列表（矩阵），通过对数据索引位置的布尔判断来筛选条件，如果一条语句出现两个判断条件，会存在语义不明的情况

ogc-loud

```
In [33]: df[df > 0]
Out[33]:
                   A         B         C         D         E
2020-05-01       NaN       NaN  0.834057  0.191678  0.544135
2020-05-02  0.748313  0.542425       NaN       NaN  1.088737
2020-05-03  0.562413       NaN       NaN  1.434097  0.088396
2020-05-04       NaN  1.608144  1.070904  1.895496  1.376240
2020-05-05  0.343573  2.268671  0.123661  0.026515  0.819831
2020-05-06  1.149055  0.514129  0.653298  1.014948  0.364408
2020-05-07       NaN  0.648047       NaN  1.198958  0.756575
2020-05-08       NaN  0.176260       NaN  0.020373  0.047486

In [34]: df[df["E"].isin([1, 2])]
Out[34]:
Empty DataFrame
Columns: [A, B, C, D, E]
Index: []
```

## Setting Value Operation in DataFrame

```
In [35]: sl = pd.Series(list(range(10,18)),index=pd.date_range("20200501",
periods=8))

In [36]: df["F"] = sl

In [37]: df
Out[37]:
                   A         B         C         D         E   F
2020-05-01 -0.639606 -0.490763  0.834057  0.191678  0.544135  10
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737  11
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396  12
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240  13
2020-05-05  0.343573  2.268671  0.123661  0.026515  0.819831  14
2020-05-06  1.149055  0.514129  0.653298  1.014948  0.364408  15
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575  16
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486  17

In [38]: df.at[dates[0], "A"] = 0

In [39]: df
Out[39]:
                   A         B         C         D         E   F
2020-05-01  0.000000 -0.490763  0.834057  0.191678  0.544135  10
2020-05-02  0.748313  0.542425 -0.504249 -1.006177  1.088737  11
2020-05-03  0.562413 -1.366236 -0.999457  1.434097  0.088396  12
2020-05-04 -0.011969  1.608144  1.070904  1.895496  1.376240  13
2020-05-05  0.343573  2.268671  0.123661  0.026515  0.819831  14
2020-05-06  1.149055  0.514129  0.653298  1.014948  0.364408  15
2020-05-07 -0.504259  0.648047 -1.170642  1.198958  0.756575  16
```

gocloud

```
2020-05-08 -0.521412  0.176260 -0.090847  0.020373  0.047486  17

In [40]: df.iat[1, 1] = 1

In [41]: df.loc[:, "D"] = np.array([4] * len(df))

In [42]: df
Out[42]:
                   A         B         C  D         E   F
2020-05-01  0.000000 -0.490763  0.834057  4  0.544135  10
2020-05-02  0.748313  1.000000 -0.504249  4  1.088737  11
2020-05-03  0.562413 -1.366236 -0.999457  4  0.088396  12
2020-05-04 -0.011969  1.608144  1.070904  4  1.376240  13
2020-05-05  0.343573  2.268671  0.123661  4  0.819831  14
2020-05-06  1.149055  0.514129  0.653298  4  0.364408  15
2020-05-07 -0.504259  0.648047 -1.170642  4  0.756575  16
2020-05-08 -0.521412  0.176260 -0.090847  4  0.047486  17

In [43]: df2 = df.copy()

In [44]: df2[df2 > 0] = - df2

In [45]: df2
                   A         B         C  D         E   F
2020-05-01  0.000000 -0.490763 -0.834057 -4 -0.544135 -10
2020-05-02 -0.748313 -1.000000 -0.504249 -4 -1.088737 -11
2020-05-03 -0.562413 -1.366236 -0.999457 -4 -0.088396 -12
2020-05-04 -0.011969 -1.608144 -1.070904 -4 -1.376240 -13
2020-05-05 -0.343573 -2.268671 -0.123661 -4 -0.819831 -14
2020-05-06 -1.149055 -0.514129 -0.653298 -4 -0.364408 -15
2020-05-07 -0.504259 -0.648047 -1.170642 -4 -0.756575 -16
2020-05-08 -0.521412 -0.176260 -0.090847 -4 -0.047486 -17
```

## Missing Data Processing

我们先重新设置一个 `df`

```
In [46]: dates = pd.date_range("20200501", periods=8)

In [47]: df = pd.DataFrame(np.random.randn(8, 5), index=dates,
columns=list("ABCDE"))

In [48]: df
Out[48]:
                   A         B         C         D         E
2020-05-01  1.161780  0.046974  0.317034  0.985277 -0.878156
2020-05-02 -0.511518 -0.462444 -0.090256  1.013958 -0.052817
2020-05-03  0.492906 -0.098113 -1.621421 -0.469094 -0.954550
2020-05-04  0.229874 -0.344795 -0.158310 -0.419449  0.096488
```

ggcloud

```
2020-05-05  1.250265 -1.422900  1.084396  0.902803 -1.138471
2020-05-06 -1.349342 -0.357210 -0.623589  0.331251  0.305456
2020-05-07  0.506861 -1.480997 -0.835471  0.158394  1.484623
2020-05-08 -0.387560 -0.233622  1.192566 -0.510911 -0.855755
```

### Missing Values

```
In [49]: df1 = df.reindex(index=dates[:4],
    ...:                  columns=list("ABCD") + ["G"])

In [50]: df1.loc[dates[0]: dates[1], "G"] = 1

In [51]: df1
Out[51]:
                   A         B         C         D    G
2020-05-01  1.161780  0.046974  0.317034  0.985277  1.0
2020-05-02 -0.511518 -0.462444 -0.090256  1.013958  1.0
2020-05-03  0.492906 -0.098113 -1.621421 -0.469094  NaN
2020-05-04  0.229874 -0.344795 -0.158310 -0.419449  NaN

In [52]: df1.dropna()
Out[52]:
                   A         B         C         D    G
2020-05-01  1.161780  0.046974  0.317034  0.985277  1.0
2020-05-02 -0.511518 -0.462444 -0.090256  1.013958  1.0

In [53]: df1.fillna(value=2)
Out[53]:
                   A         B         C         D    G
2020-05-01  1.161780  0.046974  0.317034  0.985277  1.0
2020-05-02 -0.511518 -0.462444 -0.090256  1.013958  1.0
2020-05-03  0.492906 -0.098113 -1.621421 -0.469094  2.0
2020-05-04  0.229874 -0.344795 -0.158310 -0.419449  2.0
```

## Merge & Reshape

### Pandas Statistic

```
In [54]: df.mean()
Out[54]:
A    0.174158
B   -0.544138
C   -0.091881
D    0.249029
E   -0.249148
dtype: float64

In [55]: df.var()
```

gocloud

```
Out[55]:
A    0.779362
B    0.339436
C    0.911456
D    0.444235
E    0.789096
dtype: float64

In [56]: s = pd.Series([1, 2, 2, np.nan, 5, 7, 9, 10], index=dates)

In [57]: s
Out[57]:
2020-05-01     1.0
2020-05-02     2.0
2020-05-03     2.0
2020-05-04     NaN
2020-05-05     5.0
2020-05-06     7.0
2020-05-07     9.0
2020-05-08    10.0
Freq: D, dtype: float64

In [58]: s.shift(2)         # shift 函数是对数据进行移动的操作
Out[58]:
2020-05-01    NaN
2020-05-02    NaN
2020-05-03    1.0
2020-05-04    2.0
2020-05-05    2.0
2020-05-06    NaN
2020-05-07    5.0
2020-05-08    7.0
Freq: D, dtype: float64

In [59]: s.diff()           # 差分列
Out[59]:
2020-05-01    NaN
2020-05-02    1.0
2020-05-03    0.0
2020-05-04    NaN
2020-05-05    NaN
2020-05-06    2.0
2020-05-07    2.0
2020-05-08    1.0
Freq: D, dtype: float64

In [60]: s.value_counts()   # 频数统计
Out[60]:
2.0     2
```

ogCloud

```
10.0    1
9.0     1
7.0     1
5.0     1
1.0     1
dtype: int64
```

## Pandas Concat

```
In [61]: pieces = [df[:3], df[-3:]]

In [62]: pd.concat(pieces)
Out[62]:
                   A         B         C         D         E
2020-05-01  1.161780  0.046974  0.317034  0.985277 -0.878156
2020-05-02 -0.511518 -0.462444 -0.090256  1.013958 -0.052817
2020-05-03  0.492906 -0.098113 -1.621421 -0.469094 -0.954550
2020-05-06 -1.349342 -0.357210 -0.623589  0.331251  0.305456
2020-05-07  0.506861 -1.480997 -0.835471  0.158394  1.484623
2020-05-08 -0.387560 -0.233622  1.192566 -0.510911 -0.855755

In [63]: left = pd.DataFrame({"key": ["x", "y"], "value": [1, 2]})

In [64]: left
Out[64]:
  key  value
0   x      1
1   y      2

In [65]: right = pd.DataFrame({"key": ["x", "z"], "value": [3, 4]})

In [66]: right
Out[66]:
  key  value
0   x      3
1   z      4

In [67]: pd.merge(left, right, on="key", how="outer")
Out[67]:
  key  value_x  value_y
0   x      1.0      3.0
1   y      2.0      NaN
2   z      NaN      4.0

In [68]: df3 = pd.DataFrame({"A": ["a", "b", "c", "b"], "B":
list(range(4))})

In [69]: df3.groupby("A").sum()    # a: 0; b: 1+3; c: 2
Out[69]:
```

99cloud

```
   B
A
a  0
b  4
c  2
```

## Pandas Reshape

```
In [70]: import datetime

In [71]: df4 = pd.DataFrame({'A': ['one', 'one', 'two', 'three'] * 6,
    ...:                     'B': ['A', 'B', 'C'] * 8,
    ...:                     'C': ['foo', 'foo', 'foo', 'bar', 'bar',
'bar'] * 4,
    ...:                     'D': np.random.randn(24),
    ...:                     'E': np.random.randn(24),
    ...:                     'F': [datetime.datetime(2020, i, 1) for i in
range(1, 13)] +
    ...:                          [datetime.datetime(2020, i, 15) for i in
range(1, 13)]})

In [72]: df4
Out[72]:
        A  B    C         D         E          F
0     one  A  foo  0.158188 -0.194267 2020-01-01
1     one  B  foo  1.997928  0.087429 2020-02-01
2     two  C  foo -2.187545 -0.847917 2020-03-01
3   three  A  bar  0.448641  1.804847 2020-04-01
4     one  B  bar  0.753182  0.349110 2020-05-01
5     one  C  bar -0.124078 -1.079033 2020-06-01
6     two  A  foo -0.260935  0.977308 2020-07-01
7   three  B  foo  1.622864  1.303867 2020-08-01
8     one  C  foo -0.649257 -0.422324 2020-09-01
9     one  A  bar  0.365891 -1.304234 2020-10-01
10    two  B  bar  2.759241  0.412113 2020-11-01
11  three  C  bar  1.247950 -1.616772 2020-12-01
12    one  A  foo  0.630572  0.538397 2020-01-15
13    one  B  foo -0.404018 -0.239144 2020-02-15
14    two  C  foo -0.012304 -0.303686 2020-03-15
15  three  A  bar -0.510348 -0.513858 2020-04-15
16    one  B  bar  1.291382  0.492975 2020-05-15
17    one  C  bar  1.272235  0.131740 2020-06-15
18    two  A  foo -0.101000 -0.700864 2020-07-15
19  three  B  foo -2.651767  0.554233 2020-08-15
20    one  C  foo  0.714918 -0.489591 2020-09-15
21    one  A  bar -0.189745 -0.781274 2020-10-15
22    two  B  bar  0.476801 -0.178456 2020-11-15
23  three  C  bar  1.134483 -0.933998 2020-12-15

```

gocloud

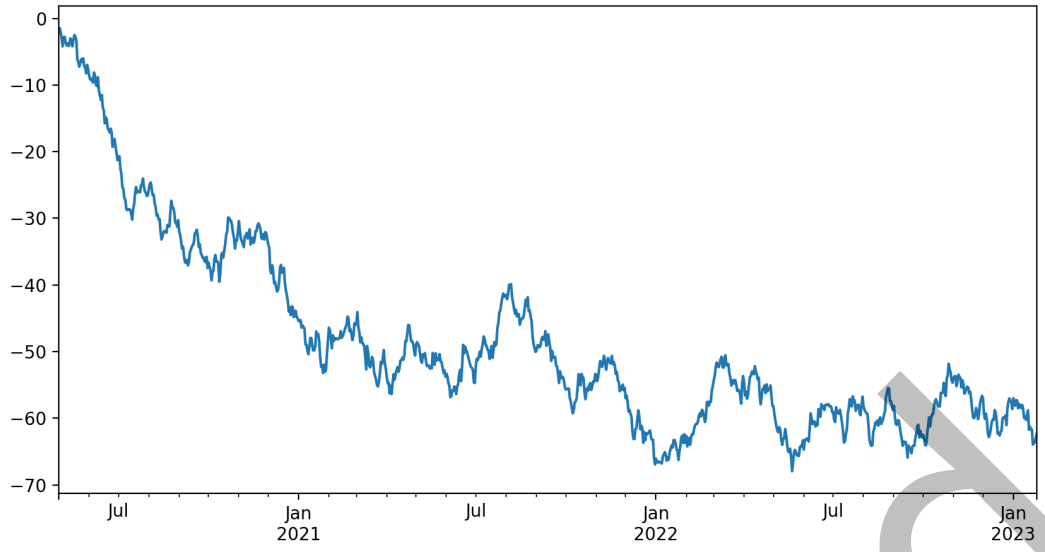
```
In [73]: pd.pivot_table(df4, values="D", index=["A", "B"], columns=["C"])
Out[73]:
C             bar       foo
A     B
one   A  0.088073  0.394380
      B  1.022282  0.796955
      C  0.574079  0.032831
three A -0.030854       NaN
      B       NaN -0.514451
      C  1.191217       NaN
two   A       NaN -0.180968
      B  1.618021       NaN
      C       NaN -1.099925
```

函数 `pivot_table()` 中，默认 `aggfunc='mean'` 计算均值

## Time Series & Graph & File

```
# Time Series
In [74]: from pylab import *

In [75]: t_exam = pd.date_range("20200501", periods=10, freq="S")

In [76]: t_exam
Out[76]:
DatetimeIndex(['2020-05-01 00:00:00', '2020-05-01 00:00:01',
               '2020-05-01 00:00:02', '2020-05-01 00:00:03',
               '2020-05-01 00:00:04', '2020-05-01 00:00:05',
               '2020-05-01 00:00:06', '2020-05-01 00:00:07',
               '2020-05-01 00:00:08', '2020-05-01 00:00:09'],
              dtype='datetime64[ns]', freq='S')

# Graph
In [77]: ts = pd.Series(np.random.randn(1000),
   ...:                 index=pd.date_range("20200501", periods=1000))

In [78]: ts = ts.cumsum()

In [79]: ts.plot()
Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x11b166e90>

In [80]: show()
```

ggcloud




## Files

```
In [81]: df4.to_csv("datas/Test.csv", index=0)

In [82]: df4.to_excel("datas/Test.xlsx")

In [83]: !tree datas
datas
├── Test.csv
└── Test.xlsx

0 directories, 2 files

In [84]: df4_csv = pd.read_csv("datas/Test.csv")

In [85]: df4_csv
Out[85]:
        A  B    C         D         E           F
0     one  A  foo  0.158188 -0.194267  2020-01-01
1     one  B  foo  1.997928  0.087429  2020-02-01
2     two  C  foo -2.187545 -0.847917  2020-03-01
... ...
21    one  A  bar -0.189745 -0.781274  2020-10-15
22    two  B  bar  0.476801 -0.178456  2020-11-15
23  three  C  bar  1.134483 -0.933998  2020-12-15

In [86]: df4_excel = pd.read_excel("datas/Test.xlsx")

In [87]: df4_excel
Out[87]:
    Unnamed: 0      A  B    C         D         E           F
0            0    one  A  foo  0.158188 -0.194267 2020-01-01
1            1    one  B  foo  1.997928  0.087429 2020-02-01
```

ggcloud

```
2            2    two  C  foo -2.187545 -0.847917 2020-03-01
... ...
21          21    one  A  bar -0.189745 -0.781274 2020-10-15
22          22    two  B  bar  0.476801 -0.178456 2020-11-15
23          23  three  C  bar  1.134483 -0.933998 2020-12-15
```

99Cloud