第13章 数据分析、科学计算、数据可视化

相关标准库和扩展库

- 用于数据分析、科学计算与可视化的扩展模块主要有：numpy、scipy、pandas、SymPy、matplotlib、Traits、TraitsUI、Chaco、TVTK、Mayavi、VPython、OpenCV。

相关标准库和扩展库

- numpy：科学计算包，支持N维数组运算、处理大型矩阵、成熟的广播函数库、矢量运算、线性代数、傅里叶变换、随机数生成，并可与C++/Fortran语言无缝结合。树莓派Python v3默认安装已经包含了numpy。

相关标准库和扩展库

- scipy：scipy依赖于numpy，提供了更多的数学工具，包括矩阵运算、线性方程组求解、积分、优化、插值、信号处理、图像处理、统计等等。

相关标准库和扩展库

- matplotlib模块依赖于numpy模块和tkinter模块，可以绘制多种形式的图形，包括线图、直方图、饼状图、散点图、误差线图等等，图形质量可满足出版要求，是数据可视化的重要工具。

相关标准库和扩展库

- pandas（Python Data Analysis Library）是基于numpy的数据分析模块，提供了大量标准数据模型和高效操作大型数据集所需要的工具，可以说pandas是使得Python能够成为高效且强大的数据分析环境的重要因素之一。

相关标准库和扩展库

✓大量科学扩展库安装包下载：

http://www.lfd.uci.edu/~gohlke/pythonlibs/

✓enthought科学计算解决方案：

https://www.enthought.com/

✓anaconda3下载

https://www.continuum.io/downloads/

13.1 numpy简单应用

- 导入模块

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

13.1 numpy简单应用

- 生成数组

```
>>> np.array([1, 2, 3, 4, 5])          # 把列表转换为数组
array([1, 2, 3, 4, 5])
>>> np.array((1, 2, 3, 4, 5))          # 把元组转换成数组
array([1, 2, 3, 4, 5])
>>> np.array(range(5))                 # 把range对象转换成数组
array([0, 1, 2, 3, 4])
>>> np.array([[1, 2, 3], [4, 5, 6]]) # 二维数组
array([[1, 2, 3],
       [4, 5, 6]])
>>> np.arange(8)                       # 类似于内置函数range()
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> np.arange(1, 10, 2)
array([1, 3, 5, 7, 9])
```

13.1 numpy简单应用

```
>>> np.linspace(0, 10, 11)          # 等差数组，包含11个数
array([  0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,   8.,   9.,  10.])
>>> np.linspace(0, 10, 11, endpoint=False) # 不包含终点
array([ 0.        ,  0.90909091,  1.81818182,  2.72727273,  3.63636364,
        4.54545455,  5.45454545,  6.36363636,  7.27272727,  8.18181818,
        9.09090909])
>>> np.logspace(0, 100, 10)         # 对数数组
array([ 1.00000000e+000,   1.29154967e+011,   1.66810054e+022,
        2.15443469e+033,   2.78255940e+044,   3.59381366e+055,
        4.64158883e+066,   5.99484250e+077,   7.74263683e+088,
        1.00000000e+100])
>>> np.logspace(1,6,5, base=2)      # 对数数组，相当于2 ** np.linspace(1,6,5)
array([  2.        ,   4.75682846,  11.3137085 ,  26.90868529,  64.        ])
>>> np.zeros(3)                     # 全0一维数组
array([ 0.,  0.,  0.])
>>> np.ones(3)                      # 全1一维数组
array([ 1.,  1.,  1.])
```

13.1 numpy简单应用

```
>>> np.zeros((3,3))                    # 全0二维数组，3行3列
[[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
>>> np.zeros((3,1))                    # 全0二维数组，3行1列
array([[ 0.],
       [ 0.],
       [ 0.]])
>>> np.zeros((1,3))                    # 全0二维数组，1行3列
array([[ 0.,  0.,  0.]])
>>> np.ones((1,3))                     # 全1二维数组
array([[ 1.,  1.,  1.]])
>>> np.ones((3,3))                     # 全1二维数组
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

13.1 numpy简单应用

```
>>> np.identity(3)       # 单位矩阵
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> np.identity(2)
array([[ 1.,  0.],
       [ 0.,  1.]])
>>> np.empty((3,3))      # 空数组，只申请空间而不初始化，元素值是不确定的
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

13.1 numpy简单应用

```
>>> np.hamming(20)           # Hamming窗口
array([ 0.08      ,  0.10492407,  0.17699537,  0.28840385,  0.42707668,
        0.5779865 ,  0.7247799 ,  0.85154952,  0.94455793,  0.9937262 ,
        0.9937262 ,  0.94455793,  0.85154952,  0.7247799 ,  0.5779865 ,
        0.42707668,  0.28840385,  0.17699537,  0.10492407,  0.08      ])
>>> np.blackman(10)          # Blackman窗口
array([ -1.38777878e-17,   5.08696327e-02,   2.58000502e-01,
         6.30000000e-01,   9.51129866e-01,   9.51129866e-01,
         6.30000000e-01,   2.58000502e-01,   5.08696327e-02,
        -1.38777878e-17])
>>> np.kaiser(12, 5)         # Kaiser窗口
array([ 0.03671089,  0.16199525,  0.36683806,  0.61609304,  0.84458838,
        0.98167828,  0.98167828,  0.84458838,  0.61609304,  0.36683806,
        0.16199525,  0.03671089])
```

13.1 numpy简单应用

```
>>> np.random.randint(0, 50, 5)        # 随机数组，5个0到50之间的整数
array([13, 47, 31, 26,  9])
>>> np.random.randint(0, 50, (3,5))  # 3行5列，15个介于0和50之间的整数
array([[34,  2, 33, 14, 40],
       [ 9,  5, 10, 27, 11],
       [26, 17, 10, 46, 30]])
>>> np.random.rand(10)                    # 10个小数
array([ 0.98139326,  0.35675498,  0.30580776,  0.30379627,  0.19527425,
        0.59159936,  0.31132305,  0.20219211,  0.20073821,  0.02435331])
>>> np.random.standard_normal(5)      # 从标准正态分布中随机采样
array([ 2.82669067,  0.9773194 , -0.72595951, -0.11343254,  0.74813065])
```

13.1 numpy简单应用

```
>>> x = np.random.standard_normal(size=(3, 4, 2))
>>> x
array([[[ 0.5218421 , -1.10892934],
        [ 2.27295689,  0.9598461 ],
        [-0.92229318,  2.25708573],
        [ 0.0070173 , -0.30608704]],

       [[ 1.05133704, -0.4094823 ],
        [-0.03457527, -2.3034343 ],
        [-0.45156185, -1.26174441],
        [ 0.59367951, -0.78355627]],

       [[ 0.0424474 , -1.75202307],
        [-0.43457619, -0.96445206],
        [ 0.28342028,  1.27303125],
        [-0.15312326,  2.0399687 ]]])
```

13.1 numpy简单应用

```
>>> np.diag([1,2,3])                    # 对角矩阵
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
>>> np.diag([1,2,3,4])                  # 对角矩阵
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

13.1 numpy简单应用

- 测试两个数组是否足够接近

```
>>> x = np.array([1, 2, 3, 4.001, 5])
>>> y = np.array([1, 1.999, 3, 4.01, 5.1])
>>> np.allclose(x, y)
False
>>> np.allclose(x, y, rtol=0.2)          # 设置相对误差参数
True
>>> np.allclose(x, y, atol=0.2)          # 设置绝对误差参数
True
```

13.1 numpy简单应用

- 改变数组元素值

```
>>> x = np.arange(8)
>>> x
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> np.append(x, 8)                    # 返回新数组，增加元素
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> np.append(x, [9,10])
array([0, 1, 2, 3, 4, 5, 6, 7, 9, 10])
>>> x                                  # 不影响原来的数组
array([0, 1, 2, 3, 4, 5, 6, 7])
>>> x[3] = 8                           # 原地修改元素值
>>> x
array([0, 1, 2, 8, 4, 5, 6, 7])
>>> np.insert(x, 1, 8)                 # 返回新数组，插入元素
array([0, 8, 1, 2, 8, 4, 5, 6, 7])
```

13.1 numpy简单应用

```
>>> x.repeat(3)                          # 元素重复，返回新数组
array([0, 0, 0, 1, 1, 1, 2, 2, 2, 8, 8, 8, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7,
7, 7])
>>> x.put(0, 9)                          # 修改指定位置上的元素值
>>> x
array([9, 1, 2, 8, 4, 5, 6, 7])
>>> x = np.array([[1,2,3], [4,5,6], [7,8,9]])
>>> x[0, 2] = 4                          # 修改第0行第2列的元素值
>>> x
array([[1, 2, 4],
       [4, 5, 6],
       [7, 8, 9]])
```

13.1 numpy简单应用

- 数组与数值的运算

```
>>> x = np.array((1, 2, 3, 4, 5))      # 创建数组对象
>>> x
array([1, 2, 3, 4, 5])
>>> x * 2                              # 数组与数值相乘，返回新数组
array([ 2, 4, 6, 8, 10])
>>> x / 2                              # 数组与数值相除
array([ 0.5, 1. , 1.5, 2. , 2.5])
>>> x // 2                             # 数组与数值整除
array([0, 1, 1, 2, 2], dtype=int32)
>>> x ** 3                             # 幂运算
array([1, 8, 27, 64, 125], dtype=int32)
>>> x + 2                              # 数组与数值相加
array([3, 4, 5, 6, 7])
>>> x % 3                              # 余数
array([1, 2, 0, 1, 2], dtype=int32)
```

13.1 numpy简单应用

```
>>> 2 ** x
array([2, 4, 8, 16, 32], dtype=int32)
>>> 2 / x
array([2. ,1. ,0.66666667, 0.5, 0.4])
>>> 63 // x
array([63, 31, 21, 15, 12], dtype=int32)
```

13.1 numpy简单应用

- 数组与数组的运算

```
>>> a = np.array((1, 2, 3))
>>> b = np.array(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
>>> c = a * b                              # 数组与数组相乘
>>> c                                      # a中的每个元素乘以b中的对应列元素
array([[ 1,  4,  9],
       [ 4, 10, 18],
       [ 7, 16, 27]])
>>> c / b                                  # 数组之间的除法运算
array([[ 1.,  2.,  3.],
       [ 1.,  2.,  3.],
       [ 1.,  2.,  3.]])
>>> c / a
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.],
       [ 7.,  8.,  9.]])
```

13.1 numpy简单应用

```
>>> a + a                                  # 数组之间的加法运算
array([2, 4, 6])
>>> a * a                                  # 数组之间的乘法运算
array([1, 4, 9])
>>> a - a                                  # 数组之间的减法运算
array([0, 0, 0])
>>> a / a                                  # 数组之间的除法运算
array([ 1.,  1.,  1.])
```

13.1 numpy简单应用

- 转置

```
>>> b = np.array(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> b.T                                  # 转置
array([[1, 4, 7],
       [2, 5, 8],
       [3, 6, 9]])
>>> a = np.array((1, 2, 3, 4))
>>> a
array([1, 2, 3, 4])
>>> a.T                                  # 一维数组转置以后和原来是一样的
array([1, 2, 3, 4])
```

13.1 numpy简单应用

- 排序

```
>>> x = np.array([3, 1, 2])
>>> np.argsort(x)                    # 返回排序后元素的原下标
array([1, 2, 0], dtype=int64)
>>> x[_]                             # 获取排序后的元素
array([1, 2, 3])
>>> x = np.array([3, 1, 2, 4])
>>> np.argsort(x)
array([1, 2, 0, 3], dtype=int64)
>>> x[_]
array([1, 2, 3, 4])
>>> x.sort()                         # 原地排序
>>> x
array([1, 2, 3, 4])
```

13.1 numpy简单应用

```
>>> x = np.array([[0, 3, 4], [2, 2, 1]])
>>> np.argsort(x, axis=0)              # 二维数组纵向排序，返回原下标
array([[0, 1, 1],
       [1, 0, 0]], dtype=int64)
>>> np.argsort(x, axis=1)              # 二维数组横向排序
array([[0, 1, 2],
       [2, 0, 1]], dtype=int64)
>>> x.sort(axis=1)                     # 原地排序，横向
>>> x                                  # 注意，是每行单独排序
array([[0, 3, 4],
       [1, 2, 2]])
>>> x.sort(axis=0)                     # 原地排序，纵向
>>> x                                  # 每列单独排序
array([[0, 2, 2],
       [1, 3, 4]])
```

13.1 numpy简单应用

- 点积/内积

```
>>> a = np.array((5, 6, 7))
>>> b = np.array((6, 6, 6))
>>> a.dot(b)                                    # 向量内积
108
>>> np.dot(a,b)
108
>>> c = np.array(([1,2,3],[4,5,6],[7,8,9])) # 二维数组
>>> c.dot(a)                                    # 二维数组的每行与一维向量计算内积
array([ 38, 92, 146])
>>> a.dot(c)                    # 一维向量与二维向量的每列计算内积
array([78, 96, 114])
```

13.1 numpy简单应用

- 数组元素访问

```
>>> b = np.array(([1,2,3],[4,5,6],[7,8,9]))
>>> b
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> b[0]                    # 第0行
array([1, 2, 3])
>>> b[0][0]                 # 第0行第0列的元素值
1
>>> b[0,2]                  # 第0行第2列的元素值
3
>>> b[[0,1]]                # 第0行和第1行
array([[1, 2, 3],
       [4, 5, 6]])
>>> b[[0,1], [1,2]]    #第0行第1列的元素和第1行第2列的元素
array([2, 6])
```

13.1 numpy简单应用

```
>>> x = np.arange(0,100,10,dtype=np.floating)
>>> x
array([  0.,  10.,  20.,  30.,  40.,  50.,  60.,  70.,  80.,  90.])
>>> x[[1, 3, 5]]                    # 同时访问多个位置上的元素
array([ 10.,  30.,  50.])
>>> x[[1, 3, 5]] = 3                # 把多个位置上的元素改为相同的值
>>> x
array([  0.,   3.,  20.,   3.,  40.,   3.,  60.,  70.,  80.,  90.])
>>> x[[1, 3, 5]] = [34, 45, 56]  # 把多个位置上的元素改为不同的值
>>> x
array([  0.,  34.,  20.,  45.,  40.,  56.,  60.,  70.,  80.,  90.])
```

13.1 numpy简单应用

- 数组支持函数运算

```
>>> x = np.arange(0, 100, 10, dtype=np.floating)
>>> np.sin(x)                                    # 一维数组中所有元素求正弦值
array([ 0.        , -0.54402111,  0.91294525, -0.98803162,  0.74511316,
       -0.26237485, -0.30481062,  0.77389068, -0.99388865,  0.89399666])
>>> b = np.array(([1, 2, 3], [4, 5, 6], [7, 8, 9]))
>>> np.cos(b)                                    # 二维数组中所有元素求余弦值
array([[ 0.54030231, -0.41614684, -0.9899925 ],
       [-0.65364362,  0.28366219,  0.96017029],
       [ 0.75390225, -0.14550003, -0.91113026]])
>>> np.round(_)                                  # 四舍五入
array([[ 1., -0., -1.],
       [-1.,  0.,  1.],
       [ 1., -0., -1.]])
```

13.1 numpy简单应用

```
>>> x = np.random.rand(10) * 10                    # 包含10个随机数的数组
>>> x
array([ 2.16124573,  2.58272611,  6.18827437,  5.21282916,  4.06596404,
        3.34858432,  5.60654631,  9.49699461,  1.68564166,  2.9930861 ])
>>> np.floor(x)                                    # 所有元素向下取整
array([ 2.,  2.,  6.,  5.,  4.,  3.,  5.,  9.,  1.,  2.])
>>> np.ceil(x)                                     # 所有元素向上取整
array([  3.,   3.,   7.,   6.,   5.,   4.,   6.,  10.,   2.,   3.])
```

13.1 numpy简单应用

```
>>> x = np.linspace(0, 3.14, 10)
>>> x
array([ 0.        ,  0.34888889,  0.69777778,  1.04666667,  1.39555556,
        1.74444444,  2.09333333,  2.44222222,  2.79111111,  3.14      ])
>>> y = np.cos(x)                # 余弦
>>> y
array([ 1.        ,  0.93975313,  0.76627189,  0.50045969,  0.17434523,
       -0.17277674, -0.4990802 , -0.76524761, -0.93920748, -0.99999873])
>>> np.arccos(y)                 # 反余弦
array([ 0.        ,  0.34888889,  0.69777778,  1.04666667,  1.39555556,
        1.74444444,  2.09333333,  2.44222222,  2.79111111,  3.14      ])
```

13.1 numpy简单应用

```
>>> np.absolute(-3)                    # 绝对值或模
3
>>> np.absolute(3+4j)
5.0
>>> np.ceil(np.array([1, 2, 3.1])) # 向上取整
array([ 1.,  2.,  4.])
>>> np.isnan(np.NAN)
True
>>> np.log2(8)                         # 对数
3.0
>>> np.log10([100, 1000, 10000])
array([ 2.,  3.,  4.])
>>> np.sqrt(range(10))                 # 平方根
array([ 0.        ,  1.        ,  1.41421356,  1.73205081,  2.        ,
        2.23606798,  2.44948974,  2.64575131,  2.82842712,  3.        ])
```

13.1 numpy简单应用

- 改变数组大小

```
>>> a = np.arange(1, 11, 1)
>>> a
array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> a.shape = 2, 5                          # 改为2行5列
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
>>> a.shape = 5, -1                         # -1表示自动计算，原地修改
>>> a
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10]])
>>> b = a.reshape(2,5)                      # reshape()方法返回新数组
>>> b
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

13.1 numpy简单应用

- 切片操作

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[::-1]                                    # 反向切片
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
>>> a[::2]                                     # 隔一个取一个元素
array([0, 2, 4, 6, 8])
>>> a[:5]                                      # 前5个元素
array([0, 1, 2, 3, 4])
```

13.1 numpy简单应用

```
>>> c = np.arange(25)          # 创建数组
>>> c.shape = 5,5              # 修改数组大小
>>> c
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
>>> c[0, 2:5]                  # 第0行中下标[2,5)之间的元素值
array([2, 3, 4])
>>> c[1]                       # 第0行所有元素
array([5, 6, 7, 8, 9])
>>> c[2:5, 2:5]                # 行下标和列下标都介于[2,5)之间的元素值
array([[12, 13, 14],
       [17, 18, 19],
       [22, 23, 24]])
```

13.1 numpy简单应用

- 布尔运算

```
>>> x = np.random.rand(10) # 包含10个随机数的数组
>>> x
array([ 0.56707504,  0.07527513,  0.0149213 ,  0.49157657,  0.75404095,
        0.40330683,  0.90158037,  0.36465894,  0.37620859,  0.62250594])
>>> x > 0.5                    # 比较数组中每个元素值是否大于0.5
array([ True, False, False, False,  True, False,  True, False, False,
  True], dtype=bool)
>>> x[x>0.5]                   # 获取数组中大于0.5的元素，可用于检测和过滤异常值
array([ 0.56707504,  0.75404095,  0.90158037,  0.62250594])
>>> x < 0.5
array([False,  True,  True,  True, False,  True, False,  True,  True,
  False], dtype=bool)
>>> np.all(x<1)                # 测试是否全部元素都小于1
True
```

13.1 numpy简单应用

```
>>> np.any([1,2,3,4])              # 是否存在等价于True的元素
True
>>> np.any([0])
False
>>> a = np.array([1, 2, 3])
>>> b = np.array([3, 2, 1])
>>> a > b                          # 两个数组中对应位置上的元素比较
array([False, False,  True], dtype=bool)
>>> a[a>b]
array([3])
>>> a == b
array([False,  True, False], dtype=bool)
>>> a[a==b]
array([2])
```

13.1 numpy简单应用

- 取整运算

```
>>> x = np.random.rand(10)*50        # 10个随机数
>>> x
array([ 43.85639765,  30.47354735,  43.68965984,  38.92963767,
         9.20056878,  21.34765863,   4.61037809,  17.99941701,
        19.70232038,  30.05059154])
>>> np.int64(x)                      # 取整
array([43, 30, 43, 38,  9, 21,  4, 17, 19, 30], dtype=int64)
>>> np.int32(x)
array([43, 30, 43, 38,  9, 21,  4, 17, 19, 30])
>>> np.int16(x)
array([43, 30, 43, 38,  9, 21,  4, 17, 19, 30], dtype=int16)
>>> np.int8(x)
array([43, 30, 43, 38,  9, 21,  4, 17, 19, 30], dtype=int8)
```

13.1 numpy简单应用

- 广播

```
>>> a = np.arange(0,60,10).reshape(-1,1)      # 列向量
>>> b = np.arange(0,6)                        # 行向量
>>> a
array([[ 0],
       [10],
       [20],
       [30],
       [40],
       [50]])
>>> b
array([0, 1, 2, 3, 4, 5])
>>> a[0] + b                                  # 数组与标量的加法
array([0, 1, 2, 3, 4, 5])
>>> a[1] + b
array([10, 11, 12, 13, 14, 15])
```

13.1 numpy简单应用

```
>>> a + b                                          # 广播
array([[ 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]])
>>> a * b
array([[  0,   0,   0,   0,   0,   0],
       [  0,  10,  20,  30,  40,  50],
       [  0,  20,  40,  60,  80, 100],
       [  0,  30,  60,  90,  120, 150],
       [  0,  40,  80,  120, 160, 200],
       [  0,  50,  100, 150,  200, 250]])
```

13.1 numpy简单应用

- 分段函数

```
>>> x = np.random.randint(0, 10, size=(1,10))
>>> x
array([[0, 4, 3, 3, 8, 4, 7, 3, 1, 7]])
>>> np.where(x<5, 0, 1)                    # 小于5的元素值对应0，其他对应1
array([[0, 0, 0, 0, 1, 0, 1, 0, 0, 1]])
>>> np.piecewise(x, [x<4, x>7], [lambda x:x*2, lambda x:x*3])
                                           # 小于4的元素乘以2
                                           # 大于7的元素乘以3
                                           # 其他元素变为0
array([[ 0,  0,  6,  6, 24,  0,  0,  6,  2,  0]])
```

13.1 numpy简单应用

- 计算唯一值以及出现次数

```
>>> x = np.random.randint(0, 10, 7)
>>> x
array([8, 7, 7, 5, 3, 8, 0])
>>> np.bincount(x)    # 元素出现次数，0出现1次，
                      # 1、2没出现，3出现1次，以此类推
array([1, 0, 0, 1, 0, 1, 0, 2, 2], dtype=int64)
>>> np.sum(_)         # 所有元素出现次数之和等于数组长度
7
>>> np.unique(x)      # 返回唯一元素值
array([0, 3, 5, 7, 8])
```

13.1 numpy简单应用

- 矩阵运算

```
>>> a_list = [3, 5, 7]
>>> a_mat = np.matrix(a_list)                    # 创建矩阵
>>> a_mat
matrix([[3, 5, 7]])
>>> a_mat.T                                      # 矩阵转置
matrix([[3],
        [5],
        [7]])
>>> a_mat.shape                                  # 矩阵形状
(1, 3)
>>> a_mat.size                                   # 元素个数
3
```

13.1 numpy简单应用

```
>>> a_mat.mean()                    # 元素平均值
5.0
>>> a_mat.sum()                     # 所有元素之和
15
>>> a_mat.max()                     # 最大值
7
>>> a_mat.max(axis=1)               # 横向最大值
matrix([[7]])
>>> a_mat.max(axis=0)               # 纵向最大值
matrix([[3, 5, 7]])
>>> b_mat = np.matrix((1, 2, 3))    # 创建矩阵
>>> b_mat
matrix([[1, 2, 3]])
>>> a_mat * b_mat.T                 # 矩阵相乘
matrix([[34]])
```

13.1 numpy简单应用

```
>>> c_mat = np.matrix([[1, 5, 3], [2, 9, 6]]) # 创建二维矩阵
>>> c_mat
matrix([[1, 5, 3],
        [2, 9, 6]])
>>> c_mat.argsort(axis=0)                      # 纵向排序后的元素序号
matrix([[0, 0, 0],
        [1, 1, 1]], dtype=int64)
>>> c_mat.argsort(axis=1)                      # 横向排序后的元素序号
matrix([[0, 2, 1],
        [0, 2, 1]], dtype=int64)
>>> d_mat = np.matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
>>> d_mat.diagonal()                           # 矩阵对角线元素
matrix([[1, 5, 9]])
```

13.1 numpy简单应用

```
>>> np.cov([1,1,1,1,1])                    # 协方差
array(0.0)
>>> x = [-2.1, -1,  4.3]
>>> y = [3,  1.1,  0.12]
>>> X = np.vstack((x,y))
>>> print(np.cov(X))                       # 协方差
[[ 11.71        -4.286     ]
 [ -4.286        2.14413333]]
>>> print(np.cov(x, y))
[[ 11.71        -4.286     ]
 [ -4.286        2.14413333]]
>>> print(np.cov(x))
11.709999999999999
```

13.1 numpy简单应用

```
>>> np.linalg.eig([[1,1],[2,2]])        # 特征值与特征向量
(array([ 0.,  3.]), array([[-0.70710678, -0.4472136 ],
       [ 0.70710678, -0.89442719]]))
>>> x = np.matrix([[1,2], [3,4]])
>>> y = np.linalg.inv(x)                # 逆矩阵
>>> x * y
matrix([[  1.00000000e+00,   1.11022302e-16],
        [  0.00000000e+00,   1.00000000e+00]])
>>> y * x
matrix([[  1.00000000e+00,   4.44089210e-16],
        [  0.00000000e+00,   1.00000000e+00]])
>>> np.corrcoef(x[0], x[1])             # Pearson积矩相关系数
array([[ 1.,  1.],
       [ 1.,  1.]])
```

13.1 numpy简单应用

- 矩阵QR分解

```
>>> a = np.matrix([[1,2,3], [4,5,6]])
>>> np.linalg.qr(a)
(matrix([[-0.24253563, -0.9701425 ],
        [-0.9701425 ,  0.24253563]]), matrix([[-4.12310563, -5.33578375, -
6.54846188],
        [ 0.        , -0.72760688, -1.45521375]]))
>>> q, r = np.linalg.qr(a)
>>> np.dot(q,r)
matrix([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.]])
```

13.1 numpy简单应用

- 矩阵不同维度上的计算

```
>>> x = np.matrix(np.arange(0,10).reshape(2,5))  # 二维矩阵
>>> x
matrix([[0, 1, 2, 3, 4],
        [5, 6, 7, 8, 9]])
>>> x.sum()                                      # 所有元素之和
45
>>> x.sum(axis=0)                                # 纵向求和
matrix([[ 5,  7,  9, 11, 13]])
>>> x.sum(axis=1)                                # 横向求和
matrix([[10],
        [35]])
>>> x.mean()                                     # 平均值
4.5
>>> x.mean(axis=1)
matrix([[ 2.],
        [ 7.]])
>>> x.mean(axis=0)
matrix([[ 2.5,  3.5,  4.5,  5.5,  6.5]])
```

13.1 numpy简单应用

```
>>> x.max()                                    # 所有元素最大值
9
>>> x.max(axis=0)                              # 纵向最大值
matrix([[5, 6, 7, 8, 9]])
>>> x.max(axis=1)                              # 横向最大值
matrix([[4],
        [9]])
>>> weight = [0.3, 0.7]                        # 权重
>>> np.average(x, axis=0, weights=weight)
matrix([[ 3.5,  4.5,  5.5,  6.5,  7.5]])
```

13.1 numpy简单应用

```
>>> x = np.matrix(np.random.randint(0, 10, size=(3,3)))
>>> x
matrix([[3, 7, 4],
        [5, 1, 8],
        [2, 7, 0]])
>>> x.std()                                  # 标准差
2.6851213274654606
>>> x.std(axis=1)                            # 横向标准差
matrix([[ 1.69967317],
        [ 2.86744176],
        [ 2.94392029]])
>>> x.std(axis=0)                            # 纵向标准差
matrix([[ 1.24721913,  2.82842712,  3.26598632]])
>>> x.var(axis=0)                            # 纵向方差
matrix([[  1.55555556,   8.        ,  10.66666667]])
```

13.1 numpy简单应用

- 读写文件

```
>>> x = np.random.rand(4, 10)
>>> np.save('data.npy', x)
>>> y = np.load('data.npy')
>>> y
array([[ 0.07925715,  0.22961054,  0.88920655,  0.00662773,  0.04686686,
         0.00751701,  0.20792476,  0.18253408,  0.57074963,  0.89410328],
       [ 0.04090589,  0.09324791,  0.15263598,  0.98564644,  0.74931515,
         0.79126167,  0.19940871,  0.74923295,  0.43874089,  0.51553475],
       [ 0.5749905 ,  0.68089027,  0.19490823,  0.2631205 ,  0.53732501,
         0.58207636,  0.89361896,  0.43969519,  0.11009907,  0.96794452],
       [ 0.29274478,  0.67495611,  0.13427721,  0.57206913,  0.78126455,
         0.34121099,  0.74407954,  0.34712801,  0.55393827,  0.78458682]])
```

13.1 numpy简单应用

```
>>> a_mat = np.matrix([3, 5, 7])
>>> a_mat.tostring()
b'\x03\x00\x00\x00\x05\x00\x00\x00\x07\x00\x00\x00'
>>> a_mat.dumps()
b'\x80\x02cnumpy.core.multiarray\n_reconstruct\nq\x00cnumpy.matrixlib.def
matrix\nmatrix\nq\x01K\x00\x85q\x02c_codecs\nencode\nq\x03X\x01\x00\x00\x
00bq\x04X\x06\x00\x00\x00latin1q\x05\x86q\x06Rq\x07\x87q\x08Rq\t(K\x01K\x
01K\x03\x86q\ncnumpy\ndtype\nq\x0bX\x02\x00\x00\x00i4q\x0cK\x00K\x01\x87q
\rRq\x0e(K\x03X\x01\x00\x00\x00<q\x0fNNNJ\xff\xff\xff\xffJ\xff\xff\xff\xf
fK\x00tq\x10b\x89h\x03X\x0c\x00\x00\x00\x03\x00\x00\x00\x05\x00\x00\x00\x
07\x00\x00\x00q\x11h\x05\x86q\x12Rq\x13tq\x14b.'
>>> np.loads(_)
matrix([[3, 5, 7]])
>>> a_mat.dump('x.dat')
>>> np.load('x.dat')
matrix([[3, 5, 7]])
```

13.1 numpy简单应用

- 常用常量

```
>>> np.Inf                          # 正无穷大
inf
>>> np.NAN                          # 非数字
nan
>>> np.NaN
nan
>>> np.Infinity
inf
>>> np.MAXDIMS
32
>>> np.NINF                         # 负无穷大
-inf
>>> np.NZERO                        # 负0
-0.0
```

13.2 scipy简单应用

- scipy在numpy的基础上增加了大量用于数学计算、科学计算以及工程计算的模块，包括线性代数、常微分方程数值求解、信号处理、图像处理、稀疏矩阵等等。

13.2 scipy简单应用

- scipy主要模块有：

模块	说明
constants	常数
special	特殊函数
optimize	数值优化算法，如最小二乘拟合（leastsq）、函数最小值（fmin系列）、非线性方程组求解（fsolve）等等
interpolate	插值（interp1d、interp2d等等）
integrate	数值积分
signal	信号处理
ndimage	图像处理，包括滤波器模块filters、傅里叶变换模块fourier、图像插值模块interpolation、图像测量模块measurements、形态学图像处理模块morphology等等
stats	统计
misc	提供了读取图像文件的方法和一些测试图像
io	提供了读取Matlab和Fortran文件的方法

13.2.1 常数与特殊函数

- scipy的constants模块包含了大量用于科学计算的常数

```
>>> from scipy import constants as C
>>> C.pi                          # 圆周率
3.141592653589793
>>> C.golden                      # 黄金比例
1.618033988749895
>>> C.c                           # 真空中的光速
299792458.0
>>> C.h                           # 普朗克常数
6.62606896e-34
>>> C.mile                        # 一英里等于多少米
1609.3439999999998
>>> C.inch                        # 一英寸等于多少米
0.0254
>>> C.degree                      # 一度等于多少弧度
0.017453292519943295
>>> C.minute                      # 一分钟等于多少秒
60.0
>>> C.g                           # 标准重力加速度
9.80665
```

13.2.1 常数与特殊函数

- scipy的special模块包含了大量函数库，包括基本数学函数、特殊函数以及numpy中的所有函数。

```
>>> from scipy import special as S
>>> S.cbrt(8)                    # 立方根
2.0
>>> S.exp10(3)                   # 10**3
1000.0
>>> S.sindg(90)                  # 正弦函数，参数为角度
1.0
>>> S.round(3.1)                 # 四舍五入函数
3.0
>>> S.round(3.5)
4.0
>>> S.round(3.499)
3.0
```

13.2.2 scipy简单应用

```
>>> S.comb(5,3)                    # 从5个中任选3个的组合数
10.0
>>> S.perm(5,3)                    # 排列数
60.0
>>> S.gamma(4)                     # gamma函数
6.0
>>> S.beta(10, 200)                # beta函数
2.8396077777781333e-18
>>> S.sinc(0)                      # sinc函数
1.0
>>> S.sinc(1)
3.8981718325193755e-17
```

13.2.2 scipy简单应用

- signal模块包含大量滤波函数、B样条插值算法等等。

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> h = np.array([4, 5, 6])
>>> import scipy.signal
>>> scipy.signal.convolve(x, h)                    # 一维卷积运算
array([ 4, 13, 28, 27, 18])
```

13.2.2 scipy简单应用

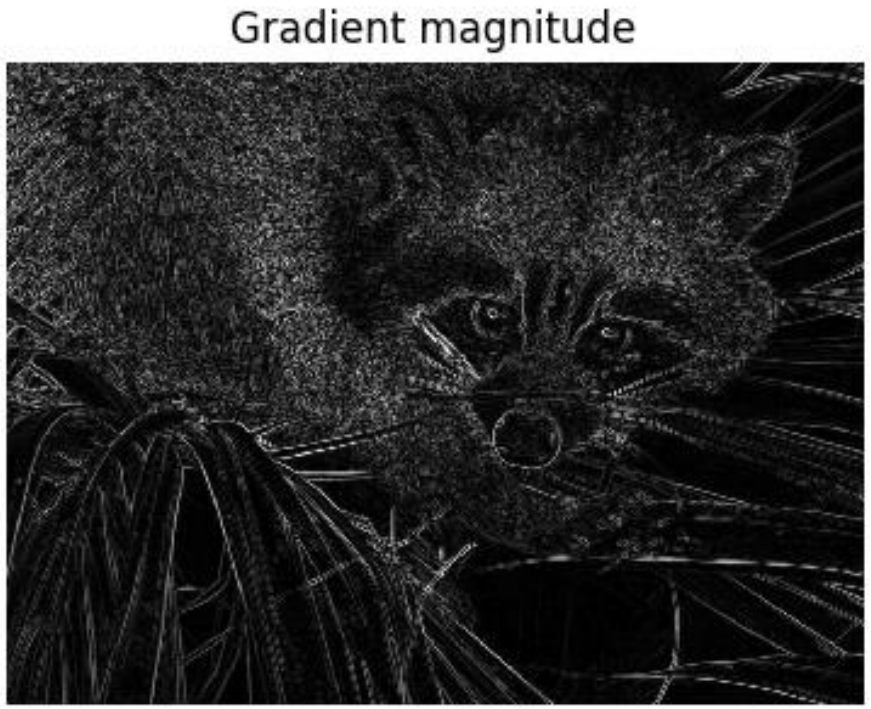
- 二维图像卷积运算。

```
import numpy as np
from scipy import signal
from scipy import misc
import matplotlib.pyplot as plt

face = misc.face(gray=True)
scharr = np.array([[ -3-3j, 0-10j,  +3 -3j],
                   [-10+0j, 0+ 0j, +10 +0j],
                   [ -3+3j, 0+10j,  +3 +3j]]) # Gx + j*Gy
grad = signal.convolve2d(face, scharr, boundary='symm', mode='same')
```

13.2.2 scipy简单应用

```
fig, (ax_orig, ax_mag) = plt.subplots(1, 2, figsize=(10, 6))
ax_orig.imshow(face, cmap='gray')
ax_orig.set_title('Original')
ax_orig.set_axis_off()

ax_mag.imshow(np.absolute(grad), cmap='gray')
ax_mag.set_title('Gradient magnitude')
ax_mag.set_axis_off()

fig.show()
```

13.2.2 scipy简单应用



13.2.2 scipy简单应用

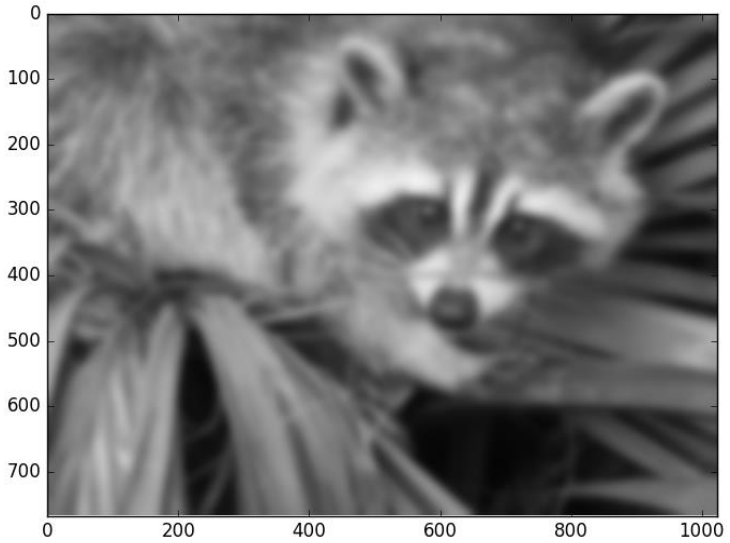
- 模块ndimage提供了大量用于N维图像处理的方法

✓图像滤波

```
>>> from scipy import misc
>>> from scipy import ndimage
>>> import matplotlib.pyplot as plt
>>> face = misc.face()          # face是测试图像之一
>>> plt.figure()                # 创建图形
>>> plt.imshow(face)            # 绘制测试图像
>>> plt.show()                  # 原始图像
```

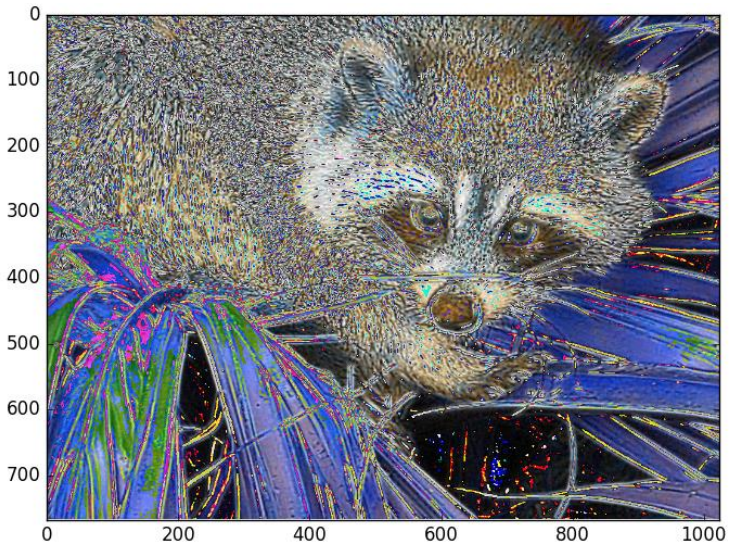
13.2.2 scipy简单应用

```
>>> blurred_face = ndimage.gaussian_filter(face, sigma=7)  # 高斯滤波
>>> plt.imshow(blurred_face)
>>> plt.show()                                             # 显示结果图像
```



13.2.2 scipy简单应用

```
>>> blurred_face1 = ndimage.gaussian_filter(face, sigma=1)        # 边缘锐化
>>> blurred_face3 = ndimage.gaussian_filter(face, sigma=3)
>>> sharp_face = blurred_face3 + 6*(blurred_face3-blurred_face1)
>>> plt.imshow(sharp_face)
>>> plt.show()
```



13.2.2 scipy简单应用

```
>>> median_face = ndimage.median_filter(face, 7)        # 中值滤波
>>> plt.imshow(median_face)
>>> plt.show()
```

13.2.2 scipy简单应用

✓图像测量

```
>>> ndimage.measurements.maximum(face)              # 最大值
255
>>> ndimage.measurements.maximum_position(face)     # 最大值位置
(242, 560, 2)
>>> ndimage.measurements.extrema(face)              # 最小值、最大值及其位置
(0, 255, (0, 883, 2), (242, 560, 2))
>>> ndimage.measurements.mean(face)                 # 平均值
110.16274388631184
>>> ndimage.measurements.median(face)               # 中值
109.0
```

13.2.2 scipy简单应用

```
>>> ndimage.measurements.center_of_mass(face)          # 重心
(356.67522474359157, 469.24020108368114, 0.9739971895510848 3)
>>> ndimage.measurements.sum(face)
259906521
>>> ndimage.measurements.variance(face)                # 方差
3307.17544034096
>>> ndimage.measurements.standard_deviation(face)      # 标准差
57.508046744268405
>>> ndimage.measurements.histogram(face, 0, 255, 256)  # 直方图
```

13.2.2 scipy简单应用（补充）

- 使用scipy进行多项式计算与符号计算

```
>>> from scipy import poly1d

>>> p1 = poly1d([1,2,3,4])
# 输出结果中，第一行的数字为第二行对应位置项中x的指数
>>> print(p1)
   3     2
1 x + 2 x + 3 x + 4

# 等价于p2=(x-1)(x-2)(x-3)(x-4)
>>> p2 = poly1d([1,2,3,4], True)
>>> print(p2)
   4      3       2
1 x - 10 x + 35 x - 50 x + 24
```

13.2.2 scipy简单应用（补充）

```
# 使用z作为变量
>>> p3 = poly1d([1,2,3,4], variable='z')
>>> print(p3)
   3     2
1 z + 2 z + 3 z + 4

# 把多项式中的变量替换为指定的值
>>> p1(0)
4
>>> p1(1)
10
```

13.2.2 scipy简单应用（补充）

```
# 计算多项式对应方程的根
>>> p1.r
array([-1.65062919+0.j         , -0.17468540+1.54686889j,
       -0.17468540-1.54686889j])
>>> p1(p1.r[0])
(-8.881784197001252e-16+0j)
```

13.2.2 scipy简单应用（补充）

```
# 查看和修改多项式的系数
>>> p1.c
array([1, 2, 3, 4])
>>> print(p3)
   3     2
1 z + 2 z + 3 z + 4
>>> p3.c[0] = 5
>>> print(p3)
   3     2
5 z + 2 z + 3 z + 4

# 查看多项式最高阶
>>> p1.order
3
```

13.2.2 scipy简单应用（补充）

```
# 查看指定指数对应的项的系数
# 例如，在p1多项式中，指数为3的项的系数为1
>>> p1[3]
1
>>> p1[0]
4
```

13.2.2 scipy简单应用（补充）

```
# 加、减、乘、除、幂运算
>>> print(p1)
   3     2
1 x + 2 x + 3 x + 4
>>> print(-p1)
    3     2
-1 x - 2 x - 3 x - 4
>>> print(p2)
   4      3      2
1 x - 10 x + 35 x - 50 x + 24
>>> print(p1 + 3)
   3     2
1 x + 2 x + 3 x + 7
>>> print(p1+p2)
   4     3      2
1 x - 9 x + 37 x - 47 x + 28
```

13.2.2 scipy简单应用（补充）

```
>>> print(p1-5)
   3     2
1 x + 2 x + 3 x - 1
>>> print(p2 - p1)
   4      3      2
1 x - 11 x + 33 x - 53 x + 20
>>> print(p1 * 3)
   3     2
3 x + 6 x + 9 x + 12
>>> print(p1*p2)
   7     6      5     4      3      2
1 x - 8 x + 18 x - 6 x - 11 x + 38 x - 128 x + 96
>>> print(p1*p2/p2)
(poly1d([ 1.,  2.,  3.,  4.]), poly1d([ 0.]))
>>> print(p2/p1)
(poly1d([  1., -12.]), poly1d([ 56., -18.,  72.]))
```

13.2.2 scipy简单应用（补充）

```
# 多项式的幂运算
>>> print(p1 ** 2)
   6      5       4       3       2
1 x + 4 x + 10 x + 20 x + 25 x + 24 x + 16
>>> print(p1 * p1)
   6      5       4       3       2
1 x + 4 x + 10 x + 20 x + 25 x + 24 x + 16
```

13.2.2 scipy简单应用（补充）

```
# 一阶导数
>>> print(p1.deriv())
   2
3 x + 4 x + 3
# 二阶导数
>>> print(p1.deriv(2))

6 x + 4
```

13.2.2 scipy简单应用（补充）

```
# 多项式的不定积分
# 一重不定积分，设常数项为0
>>> print(p1.integ(m=1, k=0))
      4           3       2
0.25 x + 0.6667 x + 1.5 x + 4 x
# 二重不定积分，设常数项为3
>>> print(p1.integ(m=2, k=3))
      5           4       3     2
0.05 x + 0.1667 x + 0.5 x + 2 x + 3 x + 3
```

13.3 数据分析模块pandas

- pandas主要提供了3种数据结构：1）Series，带标签的一维数组；2）DataFrame，带标签且大小可变的二维表格结构；3）Panel，带标签且大小可变的三维数组。

13.3 数据分析模块pandas

（1）生成一维数组

```
>>> import numpy as np
>>> import pandas as pd
>>> x = pd.Series([1, 3, 5, np.nan])
>>> x
0    1.0
1    3.0
2    5.0
3    NaN
dtype: float64
```

13.3 数据分析模块pandas

```
>>> pd.date_range(start='20130101', end='20131231', freq='H')
DatetimeIndex(['2013-01-01 00:00:00', '2013-01-01 01:00:00',
               '2013-01-01 02:00:00', '2013-01-01 03:00:00',
               '2013-01-01 04:00:00', '2013-01-01 05:00:00',
               '2013-01-01 06:00:00', '2013-01-01 07:00:00',
               '2013-01-01 08:00:00', '2013-01-01 09:00:00',
               ...
               '2013-12-30 15:00:00', '2013-12-30 16:00:00',
               '2013-12-30 17:00:00', '2013-12-30 18:00:00',
               '2013-12-30 19:00:00', '2013-12-30 20:00:00',
               '2013-12-30 21:00:00', '2013-12-30 22:00:00',
               '2013-12-30 23:00:00', '2013-12-31 00:00:00'],
              dtype='datetime64[ns]', length=8737, freq='H')
```

13.3 数据分析模块pandas

```
>>> dates = pd.date_range(start='20130101', end='20131231', freq='D')
                                                          # 间隔为天
>>> dates
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06', '2013-01-07', '2013-01-08',
               '2013-01-09', '2013-01-10',
               ...
               '2013-12-22', '2013-12-23', '2013-12-24', '2013-12-25',
               '2013-12-26', '2013-12-27', '2013-12-28', '2013-12-29',
               '2013-12-30', '2013-12-31'],
              dtype='datetime64[ns]', length=365, freq='D')
```

13.3 数据分析模块pandas

```
>>> dates = pd.date_range(start='20130101', end='20131231', freq='M')          # 间隔为月
>>> dates
DatetimeIndex(['2013-01-31', '2013-02-28', '2013-03-31', '2013-04-30',
               '2013-05-31', '2013-06-30', '2013-07-31', '2013-08-31',
               '2013-09-30', '2013-10-31', '2013-11-30', '2013-12-31'],
              dtype='datetime64[ns]', freq='M')
```

13.3 数据分析模块pandas

```
>>> pd.period_range('20170601', '20170630', freq='W')
PeriodIndex(['2017-05-29/2017-06-04', '2017-06-05/2017-06-11',
             '2017-06-12/2017-06-18', '2017-06-19/2017-06-25',
             '2017-06-26/2017-07-02'],
            dtype='period[W-SUN]', freq='W-SUN')
>>> pd.period_range('20170601', '20170630', freq='D')
PeriodIndex(['2017-06-01', '2017-06-02', '2017-06-03', '2017-06-04',
             '2017-06-05', '2017-06-06', '2017-06-07', '2017-06-08',
             '2017-06-09', '2017-06-10', '2017-06-11', '2017-06-12',
             '2017-06-13', '2017-06-14', '2017-06-15', '2017-06-16',
             '2017-06-17', '2017-06-18', '2017-06-19', '2017-06-20',
             '2017-06-21', '2017-06-22', '2017-06-23', '2017-06-24',
             '2017-06-25', '2017-06-26', '2017-06-27', '2017-06-28',
             '2017-06-29', '2017-06-30'],
            dtype='period[D]', freq='D')
```

13.3 数据分析模块pandas

```
>>> pd.period_range('20170601', '20170630', freq='H')
PeriodIndex(['2017-06-01 00:00', '2017-06-01 01:00', '2017-06-01 02:00',
             '2017-06-01 03:00', '2017-06-01 04:00', '2017-06-01 05:00',
             '2017-06-01 06:00', '2017-06-01 07:00', '2017-06-01 08:00',
             '2017-06-01 09:00',
             ...
             '2017-06-29 15:00', '2017-06-29 16:00', '2017-06-29 17:00',
             '2017-06-29 18:00', '2017-06-29 19:00', '2017-06-29 20:00',
             '2017-06-29 21:00', '2017-06-29 22:00', '2017-06-29 23:00',
             '2017-06-30 00:00'],
            dtype='period[H]', length=697, freq='H')
```

13.3 数据分析模块pandas

（2）生成DataFrame

```
>>> pd.DataFrame(np.random.randn(12,4), index=dates, columns=list('ABCD'))
                   A         B         C         D
2013-01-31  1.628310 -0.281223  0.247675 -1.604243
2013-02-28  0.071069  1.310116 -0.945838 -0.613267
2013-03-31  0.956887 -1.691863  0.170843 -0.387298
2013-04-30  0.869391 -1.939210  2.220454  1.654112
2013-05-31 -0.802416  0.558953  1.086787 -0.870317
2013-06-30  0.463761  2.451659  0.165985  0.913551
2013-07-31  1.755720  1.246089 -0.237590 -0.892358
2013-08-31  0.191604 -1.481263 -0.142491 -2.672721
2013-09-30 -0.146444  0.493261 -1.719681  0.676592
2013-10-31  1.153289  0.179862 -1.879004 -0.616305
2013-11-30 -0.500726  1.057525  0.140623 -0.113951
2013-12-31  0.229572 -0.778378 -0.682233  0.009218
```

13.3 数据分析模块pandas

```
>>> pd.DataFrame([np.random.randint(1, 100, 4) for i in range(12)],
                 index=dates, columns=list('ABCD'))   # 4列随机数
             A   B   C   D
2013-01-31  17  72  26  13
2013-02-28  61  42  88   3
2013-03-31  14  61  97  95
2013-04-30  73  87  55   1
2013-05-31  58  80  20   2
2013-06-30  41   6  40  70
2013-07-31  51  48  81  77
2013-08-31  56  54  76  61
2013-09-30  32  27  82  76
2013-10-31  21  78  91  15
2013-11-30  75  77  17  50
2013-12-31  54  12  75  53
```

13.3 数据分析模块pandas

```
>>> pd.DataFrame({'A':np.random.randint(1, 100, 4),
                  'B':pd.date_range(start='20130101', periods=4, freq='D'),
                  'C':pd.Series([1, 2, 3, 4],index=list(range(4)),dtype='float32'),
                  'D':np.array([3] * 4,dtype='int32'),
                  'E':pd.Categorical(["test","train","test","train"]),
                  'F':'foo'})
    A           B    C  D      E    F
0  65 2013-01-01  1.0  3   test  foo
1  18 2013-01-02  2.0  3  train  foo
2  24 2013-01-03  3.0  3   test  foo
3  32 2013-01-04  4.0  3  train  foo
```

13.3 数据分析模块pandas

```
>>> df = pd.DataFrame({'A':np.random.randint(1, 100, 4),
                       'B':pd.date_range(start='20130101', periods=4, freq='D'),
                       'C':pd.Series([1, 2, 3, 4],\
                              index=['zhang', 'li', 'zhou', 'wang'],dtype='float32'),
                       'D':np.array([3] * 4,dtype='int32'),
                       'E':pd.Categorical(["test","train","test","train"]),
                       'F':'foo'})
>>> df
        A          B    C  D      E    F
zhang  20 2013-01-01  1.0  3   test  foo
li     26 2013-01-02  2.0  3  train  foo
zhou   63 2013-01-03  3.0  3   test  foo
wang   69 2013-01-04  4.0  3  train  foo
```

13.3 数据分析模块pandas

（3）二维数据查看

```
>>> df.head()          # 默认显示前5行
        A           B    C  D      E    F
zhang  20 2013-01-01  1.0  3   test  foo
li     26 2013-01-02  2.0  3  train  foo
zhou   63 2013-01-03  3.0  3   test  foo
wang   69 2013-01-04  4.0  3  train  foo
>>> df.head(3)         # 查看前3行
        A           B    C  D      E    F
zhang  20 2013-01-01  1.0  3   test  foo
li     26 2013-01-02  2.0  3  train  foo
zhou   63 2013-01-03  3.0  3   test  foo
>>> df.tail(2)         # 查看最后2行
       A           B    C  D      E    F
zhou  63 2013-01-03  3.0  3   test  foo
wang  69 2013-01-04  4.0  3  train  foo
```

13.3 数据分析模块pandas

（4）查看二维数据的索引、列名和数据

```
>>> df.index
Index(['zhang', 'li', 'zhou', 'wang'], dtype='object')
>>> df.columns
Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')
>>> df.values
array([[20, Timestamp('2013-01-01 00:00:00'), 1.0, 3, 'test', 'foo'],
       [26, Timestamp('2013-01-02 00:00:00'), 2.0, 3, 'train', 'foo'],
       [63, Timestamp('2013-01-03 00:00:00'), 3.0, 3, 'test', 'foo'],
       [69, Timestamp('2013-01-04 00:00:00'), 4.0, 3, 'train', 'foo']],
dtype=object)
```

13.3 数据分析模块pandas

（5）查看数据的统计信息

```
>>> df.describe()   # 平均值、标准差、最小值、最大值等信息
                A          C    D
count    4.000000  4.000000  4.0
mean    44.500000  2.500000  3.0
std     25.066578  1.290994  0.0
min     20.000000  1.000000  3.0
25%     24.500000  1.750000  3.0
50%     44.500000  2.500000  3.0
75%     64.500000  3.250000  3.0
max     69.000000  4.000000  3.0
```

13.3 数据分析模块pandas

（6）二维数据转置

```
>>> df.T
                    zhang                   li                    zhou  \
A                      20                   26                      63
B   2013-01-01 00:00:00  2013-01-02 00:00:00  2013-01-03 00:00:00
C                       1                    2                       3
D                       3                    3                       3
E                    test                train                    test
F                     foo                  foo                     foo

                    wang
A                     69
B   2013-01-04 00:00:00
C                      4
D                      3
E                  train
F                    foo
```

13.3 数据分析模块pandas

（7）排序

```
>>> df.sort_index(axis=0, ascending=False)      # 对轴进行排序
        A          B    C  D      E    F
zhou   63 2013-01-03  3.0  3   test  foo
zhang  20 2013-01-01  1.0  3   test  foo
wang   69 2013-01-04  4.0  3  train  foo
li     26 2013-01-02  2.0  3  train  foo
>>> df.sort_index(axis=0, ascending=True)
        A          B    C  D      E    F
li     26 2013-01-02  2.0  3  train  foo
wang   69 2013-01-04  4.0  3  train  foo
zhang  20 2013-01-01  1.0  3   test  foo
zhou   63 2013-01-03  3.0  3   test  foo
```

13.3 数据分析模块pandas

```
>>> df.sort_index(axis=1, ascending=False)
          F      E  D    C           B   A
zhang   foo   test  3  1.0 2013-01-01  20
li      foo  train  3  2.0 2013-01-02  26
zhou    foo   test  3  3.0 2013-01-03  63
wang    foo  train  3  4.0 2013-01-04  69
>>> df.sort_values(by='A')  # 对数据进行排序
                            # 也可以使用by=['A','B']按多列进行排序
         A           B    C  D      E    F
zhang   20 2013-01-01  1.0  3   test  foo
li      26 2013-01-02  2.0  3  train  foo
zhou    63 2013-01-03  3.0  3   test  foo
wang    69 2013-01-04  4.0  3  train  foo
```

13.3 数据分析模块pandas

（8）数据选择

```
>>> df['A']                                    # 选择列
zhang    20
li       26
zhou     63
wang     69
Name: A, dtype: int32
>>> 69 in df['A']
False
>>> 69 in df['A'].values
True
```

13.3 数据分析模块pandas

```
>>> df.iloc[0,1]                          # 查询第0行第1列位置的数据值
Timestamp('2013-01-01 00:00:00')
>>> df.iloc[2,2]                          # 查询第2行第2列位置的数据值
3.0
>>> df[df.A>50]                           # 按给定条件进行查询
       A          B    C  D       E    F
zhou  63 2013-01-03  3.0  3    test  foo
wang  69 2013-01-04  4.0  3   train  foo
>>> df[df['E']=='test']                   # 按给定条件进行查询
        A          B    C  D     E    F
zhang  20 2013-01-01  1.0  3  test  foo
zhou   63 2013-01-03  3.0  3  test  foo
>>> df[df['A'].isin([20,69])]
        A          B    C  D      E    F
zhang  20 2013-01-01  1.0  3   test  foo
wang   69 2013-01-04  4.0  3  train  foo
```

13.3 数据分析模块pandas

```
>>> df.nlargest(3, ['C'])          # 返回指定列最大的前3行
       A          B    C  D      E    F
wang  69 2013-01-04  4.0  3  train  foo
zhou  63 2013-01-03  3.0  3   test  foo
li    26 2013-01-02  2.0  3  train  foo
>>> df.nlargest(3, ['A'])
       A          B    C  D      E    F
wang  69 2013-01-04  4.0  3  train  foo
zhou  63 2013-01-03  3.0  3   test  foo
li    26 2013-01-02  2.0  3  train  foo
```

13.3 数据分析模块pandas

（9）数据修改

```
>>> df.iat[0, 2] = 3                          # 修改指定行、列位置的数据值
>>> df.loc[:, 'D'] = np.random.randint(50, 60, 4)
                                              # 修改某列的值
>>> df['C'] = -df['C']                        # 对指定列数据取反
>>> df                                        # 查看修改结果
        A          B    C   D      E    F
zhang  20 2013-01-01 -3.0  53   test  foo
li     26 2013-01-02 -2.0  59  train  foo
zhou   63 2013-01-03 -3.0  59   test  foo
wang   69 2013-01-04 -4.0  50  train  foo
```

13.3 数据分析模块pandas

```
>>> dff = df[:]                          # 切片
>>> dff
        A          B    C   D      E    F
zhang  20 2013-01-01 -3.0  53   test  foo
li     26 2013-01-02 -2.0  59  train  foo
zhou   63 2013-01-03 -3.0  59   test  foo
wang   69 2013-01-04 -4.0  50  train  foo
>>> dff['C'] = dff['C'] ** 2             # 替换列数据
>>> dff
        A          B     C   D      E    F
zhang  20 2013-01-01   9.0  53   test  foo
li     26 2013-01-02   4.0  59  train  foo
zhou   63 2013-01-03   9.0  59   test  foo
wang   69 2013-01-04  16.0  50  train  foo
```

13.3 数据分析模块pandas

```
>>> dff = df[:]
>>> dff
        A          B    C   D      E    F
zhang  20 2013-01-01 -3.0  53   test  foo
li     26 2013-01-02 -2.0  59  train  foo
zhou   63 2013-01-03 -3.0  59   test  foo
wang   69 2013-01-04 -4.0  50  train  foo
>>> dff.loc[dff['C']==-3.0, 'D'] = 100  # 修改特定行的指定列
>>> dff
        A          B    C    D      E    F
zhang  20 2013-01-01 -3.0  100   test  foo
li     26 2013-01-02 -2.0   59  train  foo
zhou   63 2013-01-03 -3.0  100   test  foo
wang   69 2013-01-04 -4.0   50  train  foo
```

13.3 数据分析模块pandas

```
>>> data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,
                         'k2':[1, 1, 2, 3, 3, 4, 4]})
>>> data.replace(1, 5)      # 把所有1替换为5
    k1  k2
0  one   5
1  one   5
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data.replace([1,2],[5,6])     # 1->5, 2->6
    k1  k2
0  one   5
1  one   5
2  one   6
3  two   3
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data.replace({1:5, 'one':'ONE'})  # 使用字典指定替换关系
    k1  k2
0  ONE   5
1  ONE   5
2  ONE   2
3  two   3
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,
                         'k2':[1, 1, 2, 3, 3, 4, 4]})
>>> data
    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data.drop(5, axis=0)        # 删除指定行
    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
6  two   4
```

13.3 数据分析模块pandas

```
>>> data.drop(3, inplace=True)        # 原地删除
>>> data
    k1  k2
0  one   1
1  one   1
2  one   2
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data.drop('k1', axis=1)   # 删除指定列
   k2
0   1
1   1
2   2
4   3
5   4
6   4
```

13.3 数据分析模块pandas

（10）缺失值处理

```
>>> df
         A           B     C   D       E    F
zhang   20 2013-01-01   9.0  53    test  foo
li      26 2013-01-02   4.0  59   train  foo
zhou    63 2013-01-03   9.0  59    test  foo
wang    69 2013-01-04  16.0  50   train  foo
>>> df1 = df.reindex(columns=list(df.columns) + ['G'])
>>> df1
         A           B     C   D       E    F   G
zhang   20 2013-01-01   9.0  53    test  foo NaN
li      26 2013-01-02   4.0  59   train  foo NaN
zhou    63 2013-01-03   9.0  59    test  foo NaN
wang    69 2013-01-04  16.0  50   train  foo NaN
```

13.3 数据分析模块pandas

```
>>> df1.iat[0, 6] = 3          # 修改指定位置元素值，该列其他元素为缺失值NaN
>>> df1
        A          B     C   D      E    F    G
zhang  20 2013-01-01   9.0  53   test  foo  3.0
li     26 2013-01-02   4.0  59  train  foo  NaN
zhou   63 2013-01-03   9.0  59   test  foo  NaN
wang   69 2013-01-04  16.0  50  train  foo  NaN
```

13.3 数据分析模块pandas

```
>>> pd.isnull(df1)      # 测试缺失值，返回值为True/False阵列
          A      B      C      D      E      F      G
zhang  False  False  False  False  False  False  False
li     False  False  False  False  False  False   True
zhou   False  False  False  False  False  False   True
wang   False  False  False  False  False  False   True
```

13.3 数据分析模块pandas

```
>>> df1.dropna()                                # 返回不包含缺失值的行
        A           B    C   D      E    F    G
zhang  20 2013-01-01  9.0  53   test  foo  3.0
>>> df1['G'].fillna(5, inplace=True)    # 使用指定值填充缺失值
>>> df1
        A           B     C   D      E    F    G
zhang  20 2013-01-01   9.0  53   test  foo  3.0
li     26 2013-01-02   4.0  59  train  foo  5.0
zhou   63 2013-01-03   9.0  59   test  foo  5.0
wang   69 2013-01-04  16.0  50  train  foo  5.0
```

13.3 数据分析模块pandas

（11）重复值处理

```
>>> data = pd.DataFrame({'k1':['one'] * 3 + ['two'] * 4,
                         'k2':[1, 1, 2, 3, 3, 4, 4]})
>>> data
    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data.duplicated()      # 检查重复行
0    False
1     True
2    False
3    False
4     True
5    False
6     True
dtype: bool
```

13.3 数据分析模块pandas

```
>>> data.drop_duplicates()          # 返回新数组，删除重复行
    k1  k2
0  one   1
2  one   2
3  two   3
5  two   4
>>> data.drop_duplicates(['k1'])  # 删除k1列的重复数据
    k1  k2
0  one   1
3  two   3
>>> data.drop_duplicates(['k1'], keep='last')
    k1  k2
2  one   2
6  two   4
```

13.3 数据分析模块pandas

（12）异常值处理

```
>>> import numpy as np
>>> import pandas as pd
>>> data = pd.DataFrame(np.random.randn(500, 4))
>>> data.describe()                    # 查看数据的统计信息
                0           1           2           3
count  500.000000  500.000000  500.000000  500.000000
mean    -0.077138    0.052644   -0.045360    0.024275
std      0.983532    1.027400    1.009228    1.000710
min     -2.810694   -2.974330   -2.640951   -2.762731
25%     -0.746102   -0.695053   -0.808262   -0.620448
50%     -0.096517   -0.008122   -0.113366   -0.074785
75%      0.590671    0.793665    0.634192    0.711785
max      2.763723    3.762775    3.986027    3.539378
```

13.3 数据分析模块pandas

```
>>> col2 = data[2]                    # 第2列
>>> col2[col2>3.5]                    # 该列中大于3.5的数值
12    3.986027
Name: 2, dtype: float64
>>> col2[col2>3.0]
12    3.986027
Name: 2, dtype: float64
>>> col2[col2>2.5]
11     2.528325
12     3.986027
41     2.775205
157    2.707940
365    2.558892
483    2.990861
Name: 2, dtype: float64
```

13.3 数据分析模块pandas

```
>>> data[(data>3).any(1)]          # 任意一列中有大于3的数值的行
            0         1         2         3
4    1.008617  3.104177  0.522157  0.148458
12  -0.099386  0.218586  3.986027  0.997698
58  -1.553998  3.489834  0.438321 -0.276171
121 -2.101393  3.762775  1.124320 -0.210449
312 -0.945021  3.408861  1.143247 -0.005104
410 -0.279519  1.232496 -0.190450  3.539378
```

13.3 数据分析模块pandas

```
>>> data[np.abs(data)>2.5] = np.sign(data) * 2.5
                                # 把所有数据都限定到[-2.5, 2.5]之间
>>> data.describe()
                0           1           2           3
count  500.000000  500.000000  500.000000  500.000000
mean    -0.076439    0.046131   -0.049867    0.021888
std      0.978170    0.998113    0.992184    0.990873
min     -2.500000   -2.500000   -2.500000   -2.500000
25%     -0.746102   -0.695053   -0.808262   -0.620448
50%     -0.096517   -0.008122   -0.113366   -0.074785
75%      0.590671    0.793665    0.634192    0.711785
max      2.500000    2.500000    2.500000    2.500000
```

13.3 数据分析模块pandas

（13）映射

```
>>> data['k1'] = data['k1'].map(str.upper)  # 使用函数进行映射
>>> data
    k1  k2
0  ONE   1
1  ONE   1
2  ONE   2
3  TWO   3
4  TWO   3
5  TWO   4
6  TWO   4
```

13.3 数据分析模块pandas

```
>>> data['k1'] = data['k1'].map({'ONE':'one', 'TWO':'two'})
                                                    # 使用字典表示映射关系
>>> data
    k1  k2
0  one   1
1  one   1
2  one   2
3  two   3
4  two   3
5  two   4
6  two   4
```

13.3 数据分析模块pandas

```
>>> data['k2'] = data['k2'].map(lambda x:x+5)  # lambda表达式
>>> data
    k1  k2
0  one   6
1  one   6
2  one   7
3  two   8
4  two   8
5  two   9
6  two   9
```

13.3 数据分析模块pandas

```
>>> data.index = data.index.map(lambda x:x+5)  # 修改索引
>>> data
     k1  k2
5   one   6
6   one   6
7   one   7
8   two   8
9   two   8
10  two   9
11  two   9
```

13.3 数据分析模块pandas

```
>>> data.columns = data.columns.map(str.upper) # 修改列名
>>> data
     K1  K2
5   one   6
6   one   6
7   one   7
8   two   8
9   two   8
10  two   9
11  two   9
```

13.3 数据分析模块pandas

（14）数据离散化

```
>>> from random import randrange
>>> data = [randrange(100) for _ in range(10)]
>>> category = [0,25,50,100]
>>> pd.cut(data, category)
[(50, 100], (0, 25], (50, 100], (0, 25], (50, 100], (50, 100], (50, 100],
(0, 25], (0, 25], (50, 100]]
Categories (3, interval[int64]): [(0, 25] < (25, 50] < (50, 100]]
>>> pd.cut(data, category, right=False)  # 左闭右开区间
[[50, 100), [0, 25), [50, 100), [25, 50), [50, 100), [50, 100), [50, 100),
[0, 25), [0, 25), [50, 100)]
Categories (3, interval[int64]): [[0, 25) < [25, 50) < [50, 100)]
```

13.3 数据分析模块pandas

```
>>> labels = ['low', 'middle', 'high']
>>> pd.cut(data, category, right=False, labels=labels)
                                                # 指定标签
[high, low, high, middle, high, high, high, low, low, high]
Categories (3, object): [high < low < middle]
>>> data
[74, 19, 59, 25, 53, 60, 54, 22, 24, 55]
>>> pd.cut(data,4)                              # 四分位
[(60.25, 74.0], (18.945, 32.75], (46.5, 60.25], (18.945, 32.75], (46.5,
60.25], (46.5, 60.25], (46.5, 60.25], (18.945, 32.75], (18.945, 32.75], (46.5,
60.25]]
Categories (4, interval[float64]): [(18.945, 32.75] < (32.75, 46.5] < (46.5,
60.25] < (60.25, 74.0]]
```

13.3 数据分析模块pandas

（15）频次统计与移位

```
>>> df1.shift(1)                              # 数据下移一行，负数表示上移
            A          B    C     D       E    F    G
zhang     NaN        NaT  NaN   NaN     NaN  NaN  NaN
li       20.0 2013-01-01  9.0  53.0    test  foo  3.0
zhou     26.0 2013-01-02  4.0  59.0   train  foo  5.0
wang     63.0 2013-01-03  9.0  59.0    test  foo  5.0
>>> df1['D'].value_counts()                   # 直方图统计
59    2
50    1
53    1
Name: D, dtype: int64
```

13.3 数据分析模块pandas

（16）拆分与合并/连接

```
>>> df2 = pd.DataFrame(np.random.randn(10, 4))
>>> df2
          0         1         2         3
0  2.064867 -0.888018  0.586441 -0.660901
1 -0.465664 -0.496101  0.249952  0.627771
2  1.974986  1.304449 -0.168889 -0.334622
3  0.715677  2.017427  1.750627 -0.787901
4 -0.370020 -0.878282  0.499584  0.269102
5  0.184308  0.653620  0.117899 -1.186588
6 -0.364170  1.652270  0.234833  0.362925
7 -0.329063  0.356276  1.158202 -1.063800
8 -0.778828 -0.156918 -0.760394 -0.040323
9 -0.391045 -0.374825 -1.016456  0.767481
```

13.3 数据分析模块pandas

```
>>> p1 = df2[:3]                          # 数据行拆分
>>> p1
          0         1         2         3
0  2.064867 -0.888018  0.586441 -0.660901
1 -0.465664 -0.496101  0.249952  0.627771
2  1.974986  1.304449 -0.168889 -0.334622
>>> p2 = df2[3:7]
>>> p3 = df2[7:]
>>> df3 = pd.concat([p1, p2, p3])  # 数据行合并
```

13.3 数据分析模块pandas

```
>>> df2 == df3       # 测试两个二维数据是否相等，返回True/False阵列
      0     1     2     3
0  True  True  True  True
1  True  True  True  True
2  True  True  True  True
3  True  True  True  True
4  True  True  True  True
5  True  True  True  True
6  True  True  True  True
7  True  True  True  True
8  True  True  True  True
9  True  True  True  True
```

13.3 数据分析模块pandas

```
>>> df1 = pd.DataFrame({'a':range(5), 'b':range(50,55), c':range(60,65)})
>>> df3 = pd.DataFrame({'a':range(3,8), 'd':range(30,35)})
>>> df1
   a   b   c
0  0  50  60
1  1  51  61
2  2  52  62
3  3  53  63
4  4  54  64
>>> df3
   a   d
0  3  30
1  4  31
2  5  32
3  6  33
4  7  34
```

13.3 数据分析模块pandas

```
>>> pd.merge(df1, df3)                          # 内连接
   a   b   c   d
0  3  53  63  30
1  4  54  64  31
>>> pd.merge(df1, df3, how='right')             # 右连接
   a     b     c   d
0  3  53.0  63.0  30
1  4  54.0  64.0  31
2  5   NaN   NaN  32
3  6   NaN   NaN  33
4  7   NaN   NaN  34
>>> pd.merge(df1, df3, how='left')              # 左连接
   a   b   c     d
0  0  50  60   NaN
1  1  51  61   NaN
2  2  52  62   NaN
3  3  53  63  30.0
4  4  54  64  31.0
```

13.3 数据分析模块pandas

```
>>> pd.merge(df1, df3, how='outer')    # 外连接
   a     b     c     d
0  0  50.0  60.0   NaN
1  1  51.0  61.0   NaN
2  2  52.0  62.0   NaN
3  3  53.0  63.0  30.0
4  4  54.0  64.0  31.0
5  5   NaN   NaN  32.0
6  6   NaN   NaN  33.0
7  7   NaN   NaN  34.0
```

13.3 数据分析模块pandas

（17）分组计算

```
>>> df4 = pd.DataFrame({'A':np.random.randint(1,5,8),
                        'B':np.random.randint(10,15,8),
                        'C':np.random.randint(20,30,8),
                        'D':np.random.randint(80,100,8)})
>>> df4
   A   B   C   D
0  1  13  26  81
1  3  14  29  88
2  1  13  28  88
3  2  10  21  90
4  4  14  28  83
5  4  11  24  81
6  2  11  26  99
7  3  13  25  91
```

13.3 数据分析模块pandas

```
>>> df4.groupby('A').sum()          # 数据分组计算
    B   C    D
A
1  26  54  169
2  21  47  189
3  27  54  179
4  25  52  164
```

13.3 数据分析模块pandas

```
>>> df4.groupby(['A','B']).mean()
          C     D
A  B
1 13   27.0  84.5
2 10   21.0  90.0
  11   26.0  99.0
3 13   25.0  91.0
  14   29.0  88.0
4 11   24.0  81.0
  14   28.0  83.0
```

13.3 数据分析模块pandas

（18）透视转换

```
>>> df = pd.DataFrame({'a':[1,2,3,4],
                       'b':[2,3,4,5],
                       'c':[3,4,5,6],
                       'd':[3,3,3,3]})
>>> df
   a  b  c  d
0  1  2  3  3
1  2  3  4  3
2  3  4  5  3
3  4  5  6  3
```

13.3 数据分析模块pandas

```
>>> df.pivot(index='a', columns='b', values='c')
b    2    3    4    5
a
1  3.0  NaN  NaN  NaN
2  NaN  4.0  NaN  NaN
3  NaN  NaN  5.0  NaN
4  NaN  NaN  NaN  6.0
>>> df.pivot(index='a', columns='b', values='d')
b    2    3    4    5
a
1  3.0  NaN  NaN  NaN
2  NaN  3.0  NaN  NaN
3  NaN  NaN  3.0  NaN
4  NaN  NaN  NaN  3.0
```

13.3 数据分析模块pandas

（19）数据差分

```
>>> df = pd.DataFrame({'a':np.random.randint(1, 100, 10),
                       'b':np.random.randint(1, 100, 10)},
                       index=map(str, range(10)))
>>> df
    a   b
0  21  54
1  53  28
2  18  87
3  56  40
4  62  34
5  74  10
6   7  78
7  58  79
8  66  80
9  30  21
```

13.3 数据分析模块pandas

```
>>> df.diff()                    # 纵向一阶差分
      a     b
0   NaN   NaN
1  32.0 -26.0
2 -35.0  59.0
3  38.0 -47.0
4   6.0  -6.0
5  12.0 -24.0
6 -67.0  68.0
7  51.0   1.0
8   8.0   1.0
9 -36.0 -59.0
```

13.3 数据分析模块pandas

```
>>> df.diff(axis=1)          # 横向一阶差分
    a     b
0 NaN  33.0
1 NaN -25.0
2 NaN  69.0
3 NaN -16.0
4 NaN -28.0
5 NaN -64.0
6 NaN  71.0
7 NaN  21.0
8 NaN  14.0
9 NaN  -9.0
```

13.3 数据分析模块pandas

```
>>> df.diff(periods=2)       # 纵向二阶差分
      a     b
0   NaN   NaN
1   NaN   NaN
2  -3.0  33.0
3   3.0  12.0
4  44.0 -53.0
5  18.0 -30.0
6 -55.0  44.0
7 -16.0  69.0
8  59.0   2.0
9 -28.0 -58.0
```

13.3 数据分析模块pandas

（20）计算相关系数

```
>>> df = pd.DataFrame({'A':np.random.randint(1, 100, 10),
                       'B':np.random.randint(1, 100, 10),
                       'C':np.random.randint(1, 100, 10)})
>>> df
    A   B   C
0   5  91   3
1  90  15  66
2  93  27   3
3  70  44  66
4  27  14  10
5  35  46  20
6  33  14  69
7  12  41  15
8  28  62  47
9  15  92  77
```

13.3 数据分析模块pandas

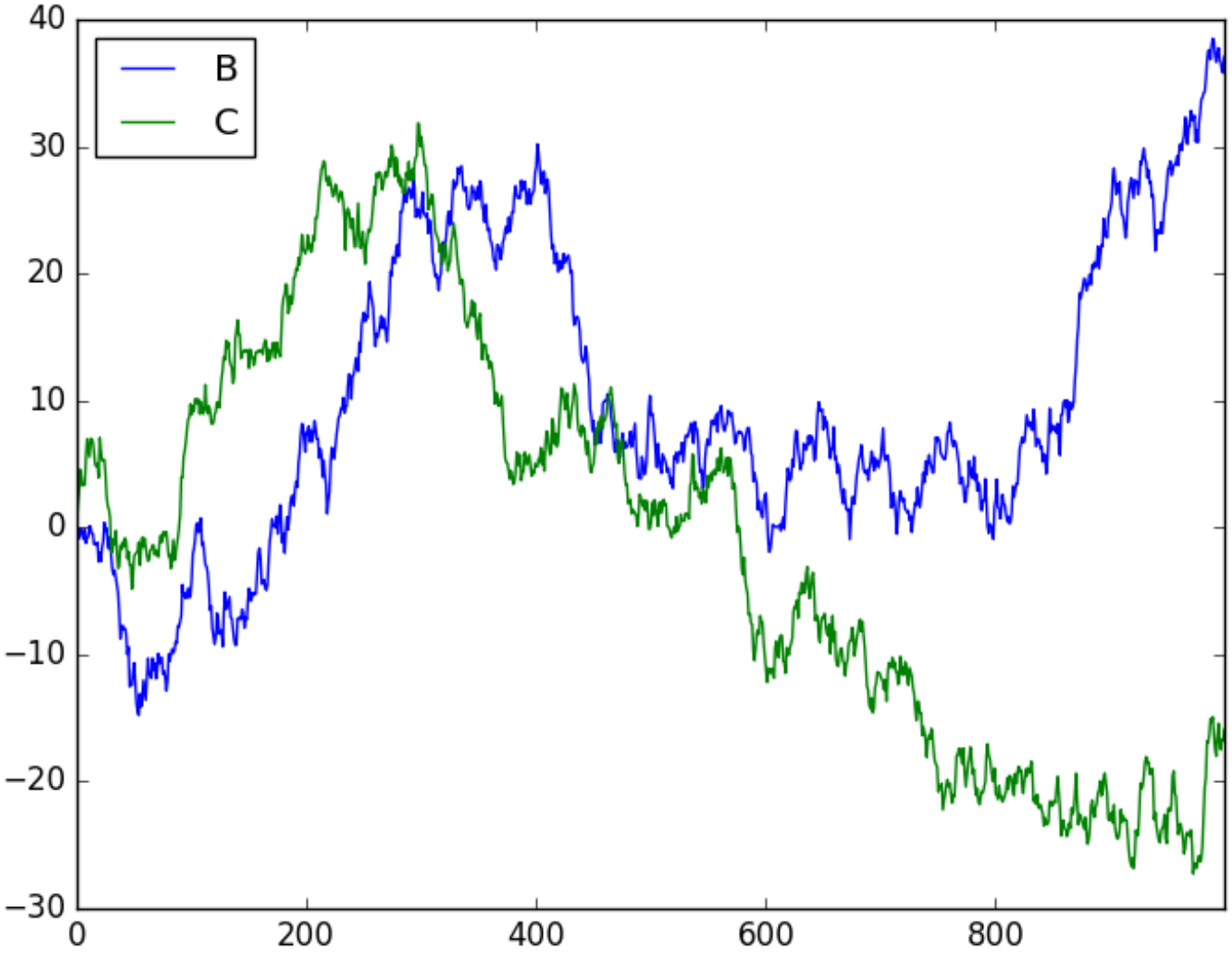
```
>>> df.corr()                                   # pearson相关系数
          A         B         C
A  1.000000 -0.560009  0.162105
B -0.560009  1.000000  0.014687
C  0.162105  0.014687  1.000000
>>> df.corr('kendall')                          # Kendall Tau相关系数
          A         B         C
A  1.000000 -0.314627  0.113666
B -0.314627  1.000000  0.045980
C  0.113666  0.045980  1.000000
>>> df.corr('spearman')                         # spearman秩相关
          A         B         C
A  1.000000 -0.419455  0.128051
B -0.419455  1.000000  0.067279
C  0.128051  0.067279  1.000000
```

13.3 数据分析模块pandas

（21）结合matplotlib绘图

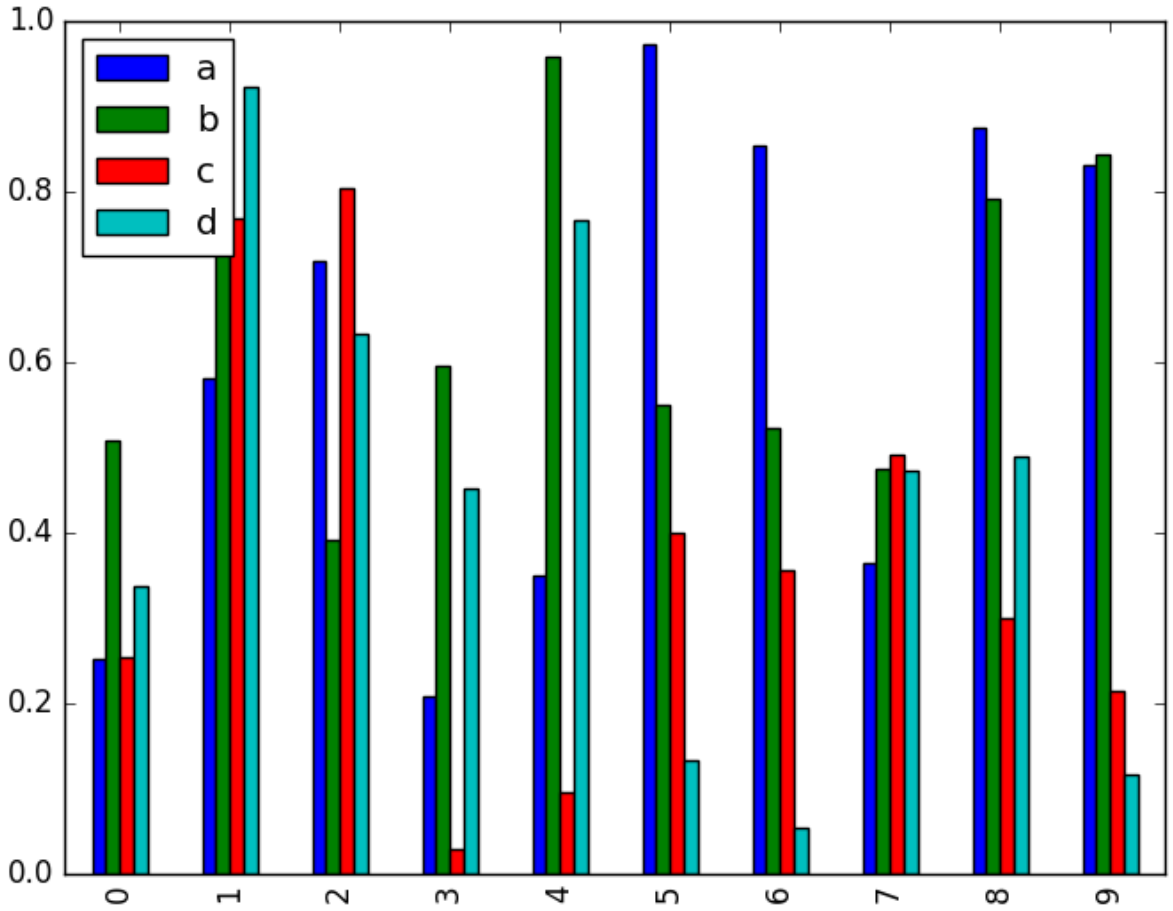
```
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> df = pd.DataFrame(np.random.randn(1000, 2), columns=['B', 'C']).cumsum()
>>> df['A'] = pd.Series(list(range(len(df))))
>>> plt.figure()
>>> df.plot(x='A')
>>> plt.show()
```

13.3 数据分析模块pandas



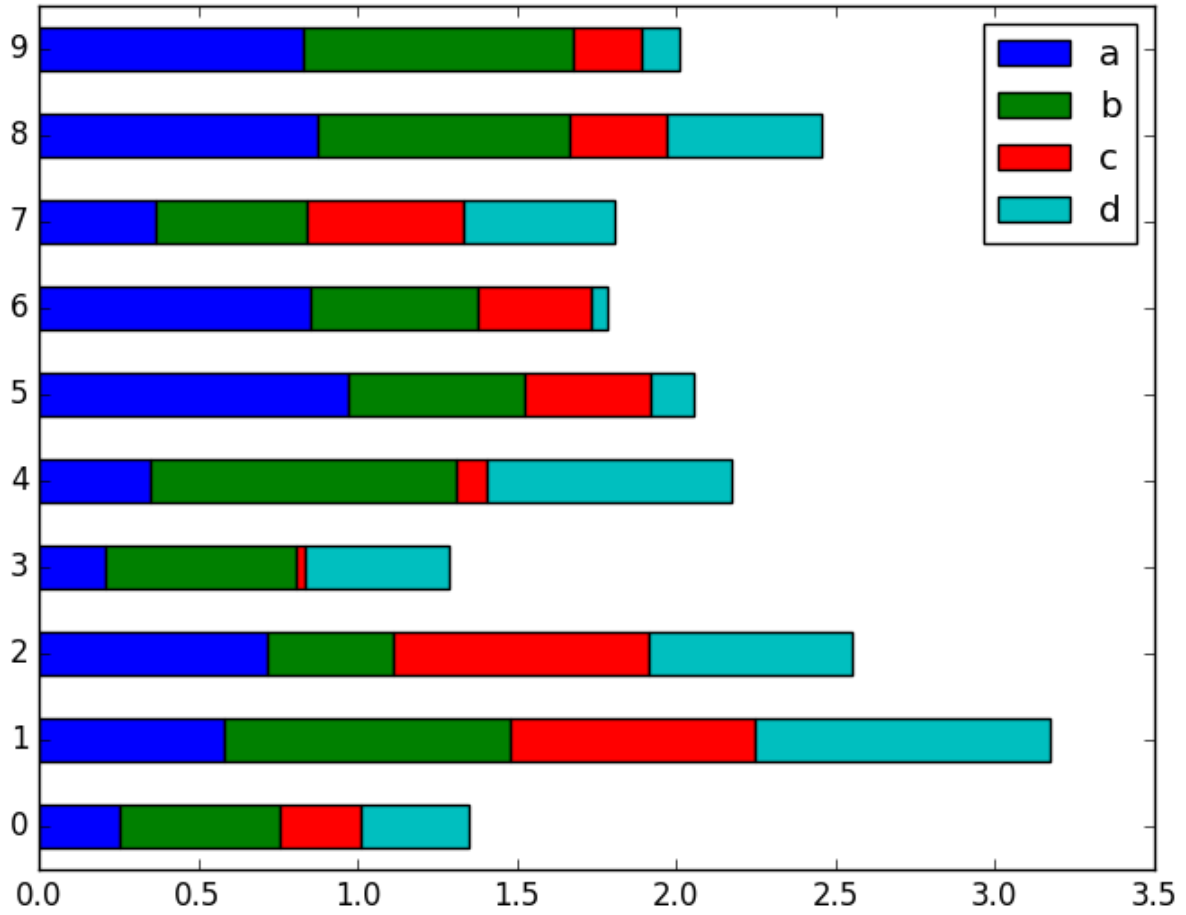
13.3 数据分析模块pandas

```
>>> df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
>>> df.plot(kind='bar')
>>> plt.show()
```



13.3 数据分析模块pandas

```
>>> df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
>>> df.plot(kind='barh', stacked=True)
>>> plt.show()
```



13.3 数据分析模块pandas

（23）文件读写

```
>>> df.to_excel('d:\\test.xlsx', sheet_name='dfg')  # 将数据保存为Excel文件
>>> df = pd.read_excel('d:\\test.xlsx', 'dfg', index_col=None, na_values=['NA'])
>>> df.to_csv('d:\\test.csv')                       # 将数据保存为csv文件
>>> df = pd.read_csv('d:\\test.csv')                # 读取csv文件中的数据
```

13.3 数据分析模块pandas

- **问题解决：**假设有个Excel 2007文件“电影导演演员.xlsx”，其中有三列分别为电影名称、导演和演员列表（同一个电影可能会有多个演员，每个演员姓名之间使用逗号分隔），要求统计每个演员的参演电影数量，并统计最受欢迎的前3个演员。

13.3 数据分析模块pandas

	A	B	C
1	电影名称	导演	演员
2	电影1	导演1	演员1，演员2，演员3，演员4
3	电影2	导演2	演员3，演员2，演员4，演员5
4	电影3	导演3	演员1，演员5，演员3，演员6
5	电影4	导演1	演员1，演员4，演员3，演员7
6	电影5	导演2	演员1，演员2，演员3，演员8
7	电影6	导演3	演员5，演员7，演员3，演员9
8	电影7	导演4	演员1，演员4，演员6，演员7
9	电影8	导演1	演员1，演员4，演员3，演员8
10	电影9	导演2	演员5，演员4，演员3，演员9
11	电影10	导演3	演员1，演员4，演员5，演员10
12	电影11	导演1	演员1，演员4，演员3，演员11
13	电影12	导演2	演员7，演员4，演员9，演员12
14	电影13	导演3	演员1，演员7，演员3，演员13
15	电影14	导演4	演员10，演员4，演员9，演员14
16	电影15	导演5	演员1，演员8，演员11，演员15
17	电影16	导演6	演员14，演员4，演员13，演员16
18	电影17	导演7	演员3，演员4，演员9
19	电影18	导演8	演员3，演员4，演员10

13.3 数据分析模块pandas

```
>>> import pandas as pd
>>> df = pd.read_excel('电影导演演员.xlsx')
>>> df
    电影名称    导演                          演员
0     电影1   导演1       演员1，演员2，演员3，演员4
1     电影2   导演2       演员3，演员2，演员4，演员5
2     电影3   导演3       演员1，演员5，演员3，演员6
3     电影4   导演1       演员1，演员4，演员3，演员7
4     电影5   导演2       演员1，演员2，演员3，演员8
5     电影6   导演3       演员5，演员7，演员3，演员9
6     电影7   导演4       演员1，演员4，演员6，演员7
7     电影8   导演1       演员1，演员4，演员3，演员8
8     电影9   导演2       演员5，演员4，演员3，演员9
9    电影10   导演3     演员1，演员4，演员5，演员10
10   电影11   导演1     演员1，演员4，演员3，演员11
11   电影12   导演2     演员7，演员4，演员9，演员12
12   电影13   导演3     演员1，演员7，演员3，演员13
13   电影14   导演4    演员10，演员4，演员9，演员14
14   电影15   导演5    演员1，演员8，演员11，演员15
15   电影16   导演6   演员14，演员4，演员13，演员16
16   电影17   导演7          演员3，演员4，演员9
17   电影18   导演8         演员3，演员4，演员10
```

13.3 数据分析模块pandas

```
>>> pairs = []
>>> for i in range(len(df)):
    actors = df.at[i, '演员'].split(', ')
    for actor in actors:
        pair = (actor, df.at[i, '电影名称'])
        pairs.append(pair)
>>> pairs = sorted(pairs, key=lambda item:int(item[0][2:]))
```

13.3 数据分析模块pandas

```
>>> pairs
[('演员1', '电影1'), ('演员1', '电影3'), ('演员1', '电影4'), ('演员1', '电影5'), ('演员1', '电影7'), ('演员1', '电影8'), ('演员1', '电影10'), ('演员1', '电影11'), ('演员1', '电影13'), ('演员1', '电影15'), ('演员2', '电影1'), ('演员2', '电影2'), ('演员2', '电影5'), ('演员3', '电影1'), ('演员3', '电影2'), ('演员3', '电影3'), ('演员3', '电影4'), ('演员3', '电影5'), ('演员3', '电影6'), ('演员3', '电影8'), ('演员3', '电影9'), ('演员3', '电影11'), ('演员3', '电影13'), ('演员3', '电影17'), ('演员3', '电影18'), ('演员4', '电影1'), ('演员4', '电影2'), ('演员4', '电影4'), ('演员4', '电影7'), ('演员4', '电影8'), ('演员4', '电影9'), ('演员4', '电影10'), ('演员4', '电影11'), ('演员4', '电影12'), ('演员4', '电影14'), ('演员4', '电影16'), ('演员4', '电影17'), ('演员4', '电影18'), ('演员5', '电影2'), ('演员5', '电影3'), ('演员5', '电影6'), ('演员5', '电影9'), ('演员5', '电影10'), ('演员6', '电影3'), ('演员6', '电影7'), ('演员7', '电影4'), ('演员7', '电影6'), ('演员7', '电影7'), ('演员7', '电影12'), ('演员7', '电影13'), ('演员8', '电影5'), ('演员8', '电影8'), ('演员8', '电影15'), ('演员9', '电影6'), ('演员9', '电影9'), ('演员9', '电影12'), ('演员9', '电影14'), ('演员9', '电影17'), ('演员10', '电影10'), ('演员10', '电影14'), ('演员10', '电影18'), ('演员11', '电影11'), ('演员11', '电影15'), ('演员12', '电影12'), ('演员13', '电影13'), ('演员13', '电影16'), ('演员14', '电影14'), ('演员14', '电影16'), ('演员15', '电影15'), ('演员16', '电影16')]
```

13.3 数据分析模块pandas

```
>>> index = [item[0] for item in pairs]
>>> data = [item[1] for item in pairs]
>>> df1 = pd.DataFrame({'演员':index, '电影名称':data})
>>> result = df1.groupby('演员', as_index=False).count()
```

13.3 数据分析模块pandas

```
>>> result
       演员   电影名称
0     演员1     10
1    演员10      3
2    演员11      2
3    演员12      1
4    演员13      2
5    演员14      2
6    演员15      1
7    演员16      1
8     演员2      3
9     演员3     12
10    演员4     13
11    演员5      5
12    演员6      2
13    演员7      5
14    演员8      3
15    演员9      5
```

13.3 数据分析模块pandas

```
>>> result.columns = ['演员', '参演电影数量']
>>> result
      演员  参演电影数量
0    演员1       10
1   演员10        3
2   演员11        2
3   演员12        1
4   演员13        2
5   演员14        2
6   演员15        1
7   演员16        1
8    演员2        3
9    演员3       12
10   演员4       13
11   演员5        5
12   演员6        2
13   演员7        5
14   演员8        3
15   演员9        5
```

13.3 数据分析模块pandas

```
>>> result.sort_values('参演电影数量')
      演员  参演电影数量
3   演员12       1
6   演员15       1
7   演员16       1
2   演员11       2
4   演员13       2
5   演员14       2
12   演员6       2
1   演员10       3
8    演员2       3
14   演员8       3
11   演员5       5
13   演员7       5
15   演员9       5
0    演员1      10
9    演员3      12
10   演员4      13
```

13.3 数据分析模块pandas

```
>>> result.nlargest(3, '参演电影数量') # 参演电影数量最多的3个演员
     演员  参演电影数量
10  演员4       13
9   演员3       12
0   演员1       10
```

13.3 数据分析模块pandas

- **问题解决：**运行下面的程序，在当前文件夹中生成饭店营业额模拟数据文件data.csv。

13.3 数据分析模块pandas

```
import csv
import random
import datetime

fn = 'data.csv'

with open(fn, 'w') as fp:
    wr = csv.writer(fp)                          # 创建csv文件写入对象
    wr.writerow(['日期', '销量'])                  # 写入表头
    startDate = datetime.date(2017, 1, 1)        # 起始日期

    # 生成365个模拟数据，可以根据需要进行调整
    for i in range(365):
        # 生成一个模拟数据，写入csv文件
        amount = 300 + i*5 + random.randrange(100)
        wr.writerow([str(startDate), amount])
        # 下一天
        startDate = startDate + datetime.timedelta(days=1)
```

13.3 数据分析模块pandas

- 然后完成下面的任务：

1）使用pandas读取文件data.csv中的数据，创建DataFrame对象，并删除其中所有缺失值；

2）使用matplotlib生成折线图，反应该饭店每天的营业额情况，并把图形保存为本地文件first.jpg；

3）按月份进行统计，使用matplotlib绘制柱状图显示每个月份的营业额，并把图形保存为本地文件second.jpg；

4）按月份进行统计，找出相邻两个月最大涨幅，并把涨幅最大的月份写入文件maxMonth.txt；

5）按季度统计该饭店2018年的营业额数据，使用matplotlib生成饼状图显示2018年4个季度的营业额分布情况，并把图形保存为本地文件third.jpg。

13.3 数据分析模块pandas

```
import pandas as pd
import matplotlib.pyplot as plt

# 读取数据，丢弃缺失值
df = pd.read_csv('data.csv', encoding='cp936')
df = df.dropna()

# 生成营业额折线图
plt.figure()
df.plot(x=df['日期'])
plt.savefig('first.jpg')
```

13.3 数据分析模块pandas

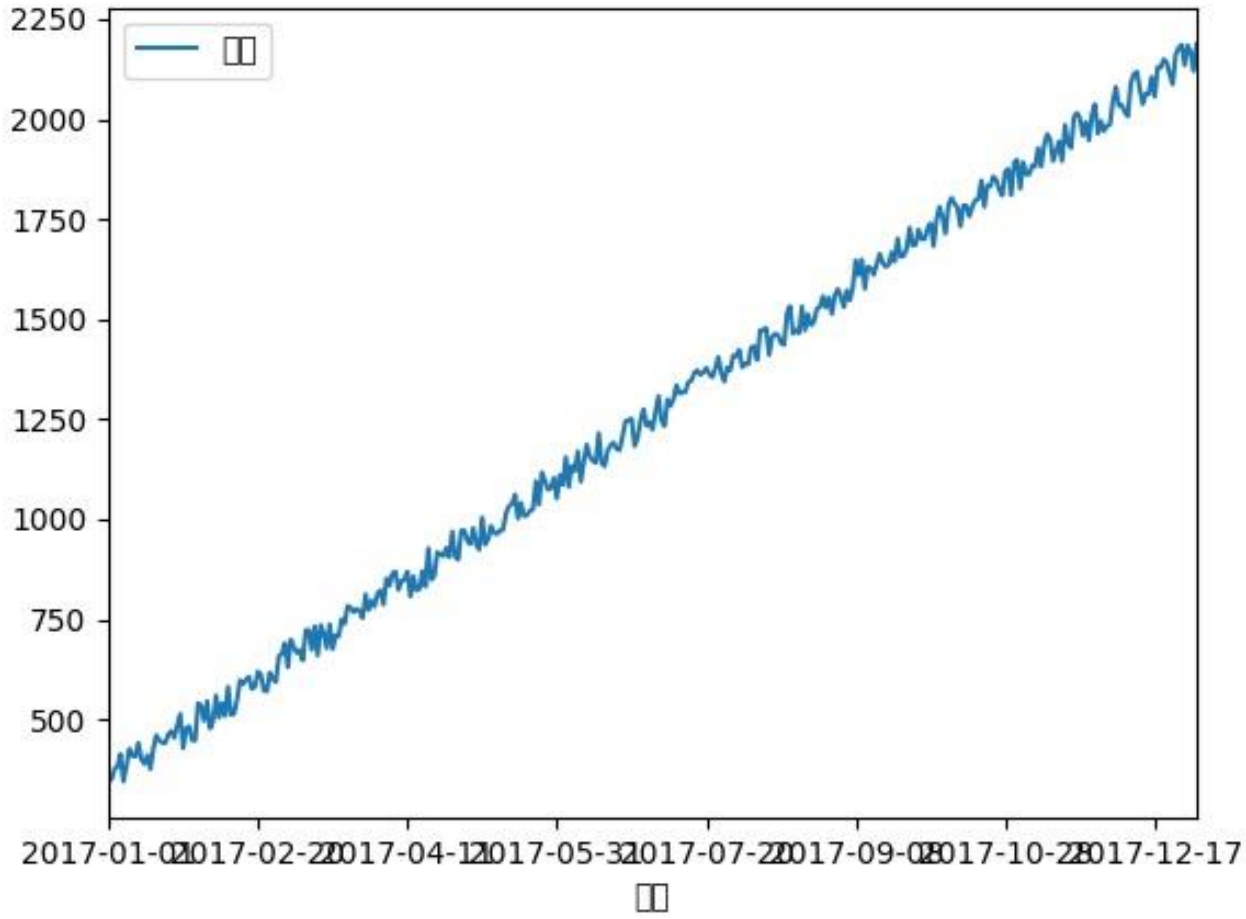
```
# 按月统计，生成柱状图
plt.figure()
df1 = df[:]
df1['month'] = df1['日期'].map(lambda x: x[:x.rindex('-')])
df1 = df1.groupby(by='month', as_index=False).sum()
df1.plot(x=df1['month'], kind='bar')
plt.savefig('second.jpg')

# 查找涨幅最大的月份，写入文件
plt.figure()
df2 = df1.drop('month', axis=1).diff()
m = df2['销量'].nlargest(1).keys()[0]
with open('maxMonth.txt', 'w') as fp:
    fp.write(df1.loc[m, 'month'])
```

13.3 数据分析模块pandas

```
# 按季度统计，生成饼状图
plt.figure()
one = df1[:3]['销量'].sum()
two = df1[3:6]['销量'].sum()
three = df1[6:9]['销量'].sum()
four = df1[9:12]['销量'].sum()
plt.pie([one, two, three, four],labels=['one', 'two', 'three', 'four'])
plt.savefig('third.jpg')
```

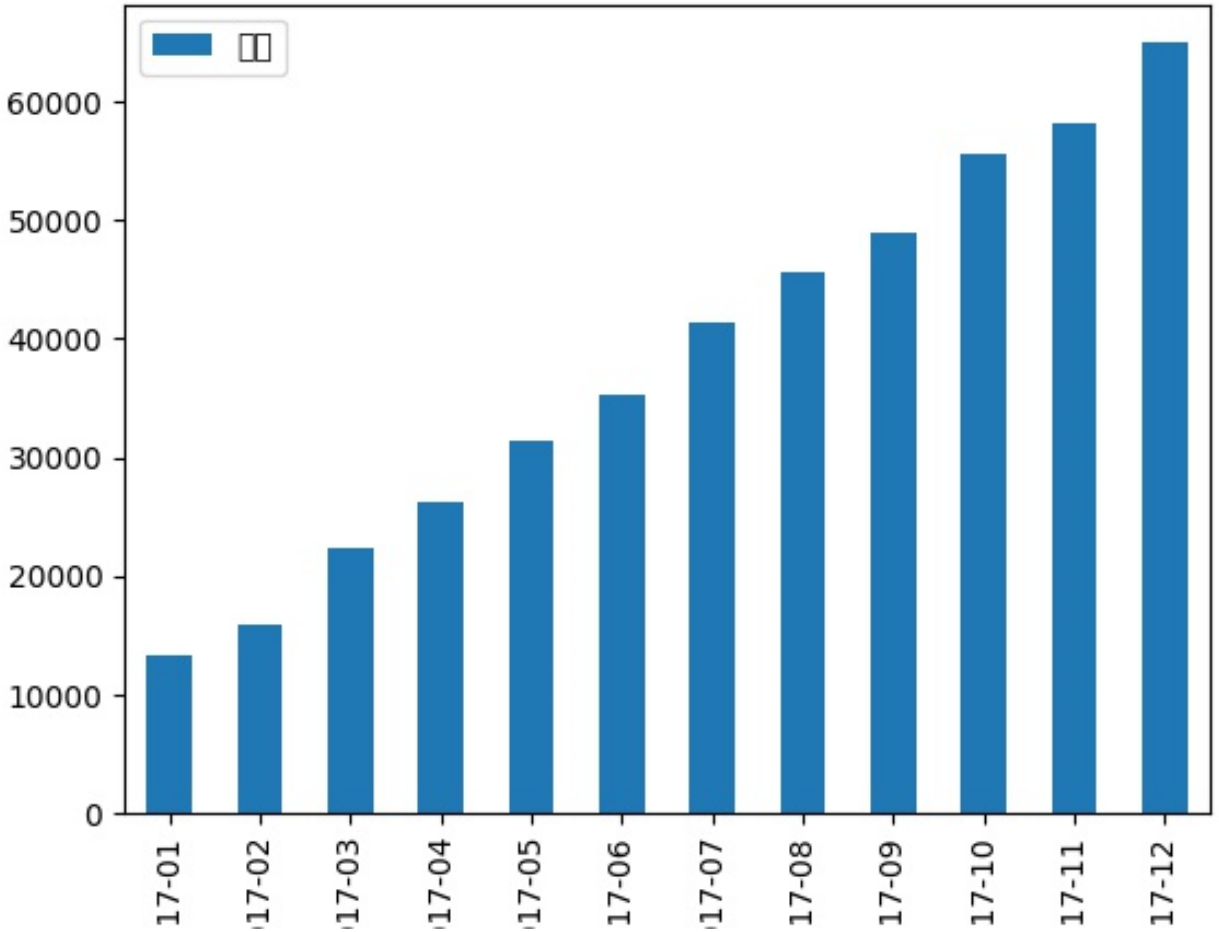
13.3 数据分析模块pandas

- first.jpg



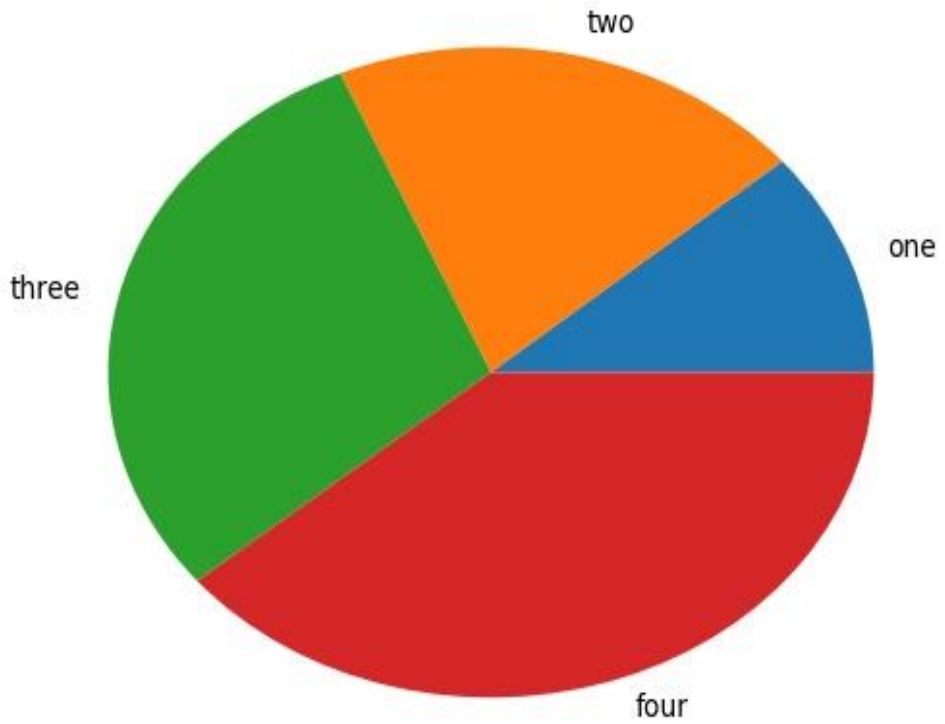
13.3 数据分析模块pandas

- second.jpg



13.3 数据分析模块pandas

- third.jpg



13.3 数据分析模块pandas

- **问题解决：**在分析时序数据的有些场合下，可能每个月只能拿到一个数据，然而实际处理时，需要把这个数据扩展到该月的每天，且每天的数据相同。

13.3 数据分析模块pandas

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


df = pd.DataFrame({'日期':pd.date_range(start='20170101',
                                        end='20171231',
                                        freq='M'),
                   '数量':np.random.randint(100, 1000, 12)})

# 每个月的天数
daysEveryMonth = [None, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
newDf = []
```

13.3 数据分析模块pandas

```
# 把每个月扩展到该月每一天
for i in range(len(df)):
    # 获取年份和月份，适当修改2月天数
    year, month = str(df.iloc[i]['日期'])[:7].split('-')
    y = int(year)
    m = int(month)
    if calendar.isleap(y):
        daysEveryMonth[2] = 29
    else:
        daysEveryMonth[2] = 28
    # 该月数量
    data = df.iloc[i]['数量']
    # 生成每个月的DataFrame，每天数量都相同
    tempDf = pd.DataFrame({'日期':pd.date_range(start=year+month+'01',
                                                periods=daysEveryMonth[m],
                                                freq='D'),
                           '数量':data})
    newDf.append(tempDf)

# 合并多个DataFrame
resultDf = pd.concat(newDf, ignore_index=True)
print(resultDf)
```

13.4 统计分析模块statistics

（1）计算平均数函数mean()

```
>>> import statistics
>>> statistics.mean([1, 2, 3, 4, 5, 6, 7, 8, 9])
                                        # 使用整数列表做参数
5.0
>>> statistics.mean(range(1,10))        # 使用range对象做参数
5.0
```

13.4 统计分析模块statistics

```
>>> import fractions
>>> x = [(3, 7), (1, 21), (5, 3), (1, 3)]
>>> y = [fractions.Fraction(*item) for item in x]
>>> y
[Fraction(3, 7), Fraction(1, 21), Fraction(5, 3), Fraction(1, 3)]
>>> statistics.mean(y)          # 使用包含分数的列表做参数
Fraction(13, 21)
>>> import decimal
>>> x = ('0.5', '0.75', '0.625', '0.375')
>>> y = map(decimal.Decimal, x)
>>> statistics.mean(y)          # 使用包含高精度实数的map对象做参数
Decimal('0.5625')
```

13.4 统计分析模块statistics

（2）中位数函数median()、median_low()、median_high()、median_grouped()

```
>>> statistics.median([1, 3, 5, 7])        # 偶数个样本时取中间两个数的平均数
4.0
>>> statistics.median_low([1, 3, 5, 7])    # 偶数个样本时取中间两个数的较小者
3
>>> statistics.median_high([1, 3, 5, 7])   # 偶数个样本时取中间两个数的较大者
5
>>> statistics.median(range(1,10))
5
```

13.4 统计分析模块statistics

```
>>> statistics.median_low([5, 3, 7]), statistics.median_high([5, 3, 7])
(5, 5)
>>> statistics.median_grouped([5, 3, 7])
5.0
>>> statistics.median_grouped([52, 52, 53, 54])
52.5
>>> statistics.median_grouped([1, 3, 3, 5, 7])
3.25
>>> statistics.median_grouped([1, 2, 2, 3, 4, 4, 4, 4, 4, 5])
3.7
>>> statistics.median_grouped([1, 2, 2, 3, 4, 4, 4, 4, 4, 5], interval=2)
3.4
```

13.4 统计分析模块statistics

（3）返回最常见数据或出现次数最多的数据（most common data）的函数mode()

```
>>> statistics.mode([1, 3, 5, 7]) # 无法确定出现次数最多的唯一元素
statistics.StatisticsError: no unique mode; found 4 equally common values
>>> statistics.mode([1, 3, 5, 7, 3])
3
>>> statistics.mode(["red", "blue", "blue", "red", "green", "red", "red"])
'red'
```

13.4 统计分析模块statistics

（4）pstdev()，返回总体标准差（population standard deviation ， the square root of the population variance)。

```
>>> statistics.pstdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
0.986893273527251
>>> statistics.pstdev(range(20))
5.766281297335398
```

13.4 统计分析模块statistics

（5）pvariance()，返回总体方差（population variance）或二次矩（second moment）。

```
>>> statistics.pvariance([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
0.9739583333333334
>>> x = [1, 2, 3, 4, 5, 10, 9, 8, 7, 6]
>>> mu = statistics.mean(x)
>>> mu
5.5
>>> statistics.pvariance([1, 2, 3, 4, 5, 10, 9, 8, 7, 6], mu)
8.25
>>> statistics.pvariance(range(20))
33.25
>>> statistics.pvariance((random.randint(1,10000) for i in range(30)))
10903549.933333334
```

13.4 统计分析模块statistics

（6）variance()、stdev()，计算样本方差（sample variance）和样本标准差（sample standard deviation，the square root of the sample variance，也叫均方差）。

```
>>> statistics.variance(range(20))
35.0
>>> statistics.stdev(range(20))
5.916079783099616
>>> _ * _
35.0
>>> lst = [3, 3, 3, 3, 3, 3]
>>> statistics.variance(lst), statistics.stdev(lst)
(0, 0.0)
```

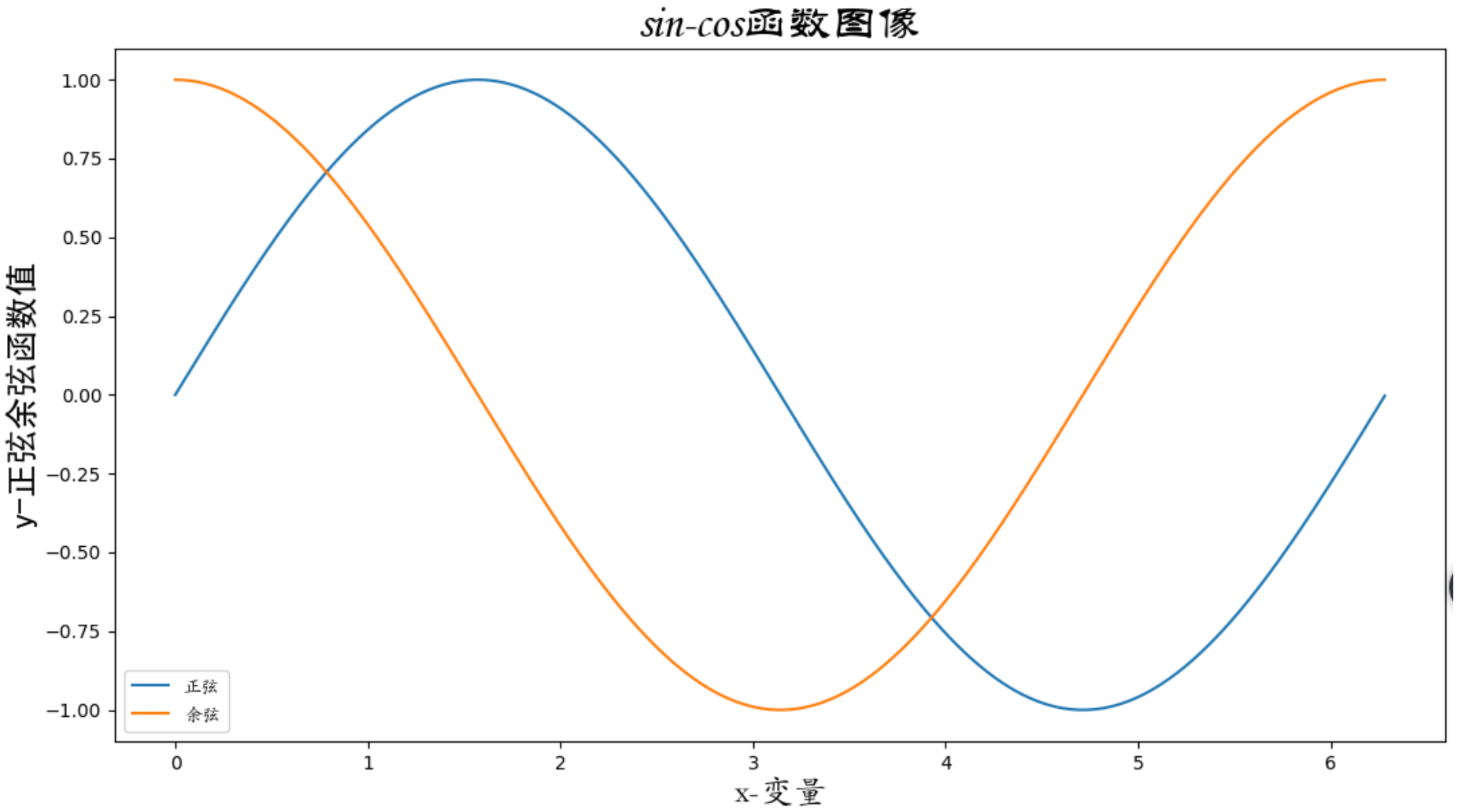
13.5 matplotlib简单应用

- matplotlib模块依赖于numpy模块和tkinter模块，可以绘制多种形式的图形，包括线图、直方图、饼状图、散点图、误差线图等等。

13.5.1 绘制带有中文标签和图例的图

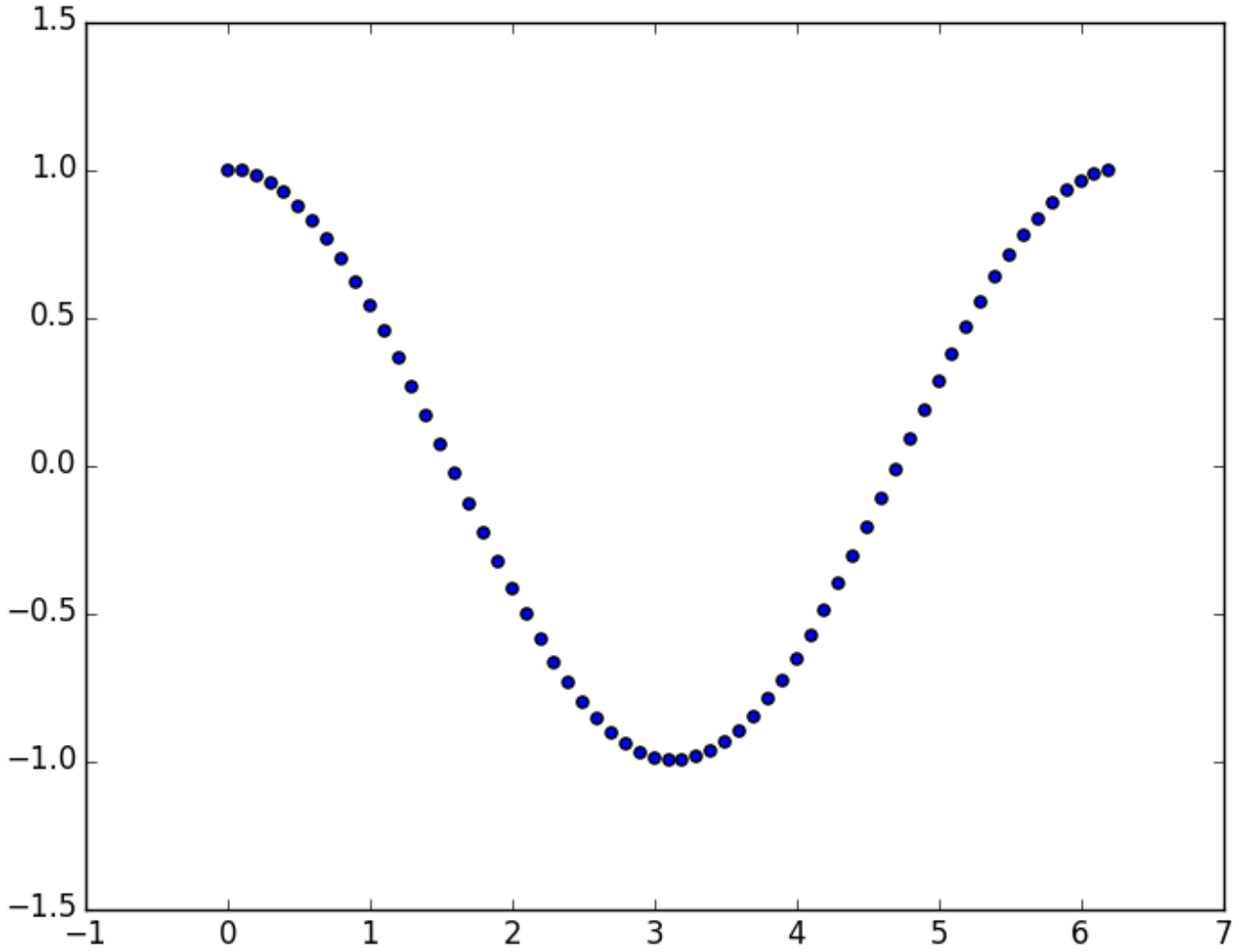
```
import numpy as np
import pylab as pl
import matplotlib.font_manager as fm

myfont = fm.FontProperties(fname=r'C:\Windows\Fonts\STKAITI.ttf') #设置字体
t = np.arange(0.0, 2.0*np.pi, 0.01)                                # 自变量取值范围
s = np.sin(t)                                                      # 计算正弦函数值
z = np.cos(t)                                                      # 计算余弦函数值
pl.plot(t, s, label='正弦')
pl.plot(t, z, label='余弦')
pl.xlabel('x-变量', fontproperties='STKAITI', fontsize=18) # 设置x标签
pl.ylabel('y-正弦余弦函数值', fontproperties='simhei', fontsize=18)
pl.title('sin-cos函数图像', fontproperties='STLITI', fontsize=24)
pl.legend(prop=myfont)
# 设置图例
pl.show()
```

13.5.1 绘制带有中文标签和图例的图



13.5.2 绘制散点图

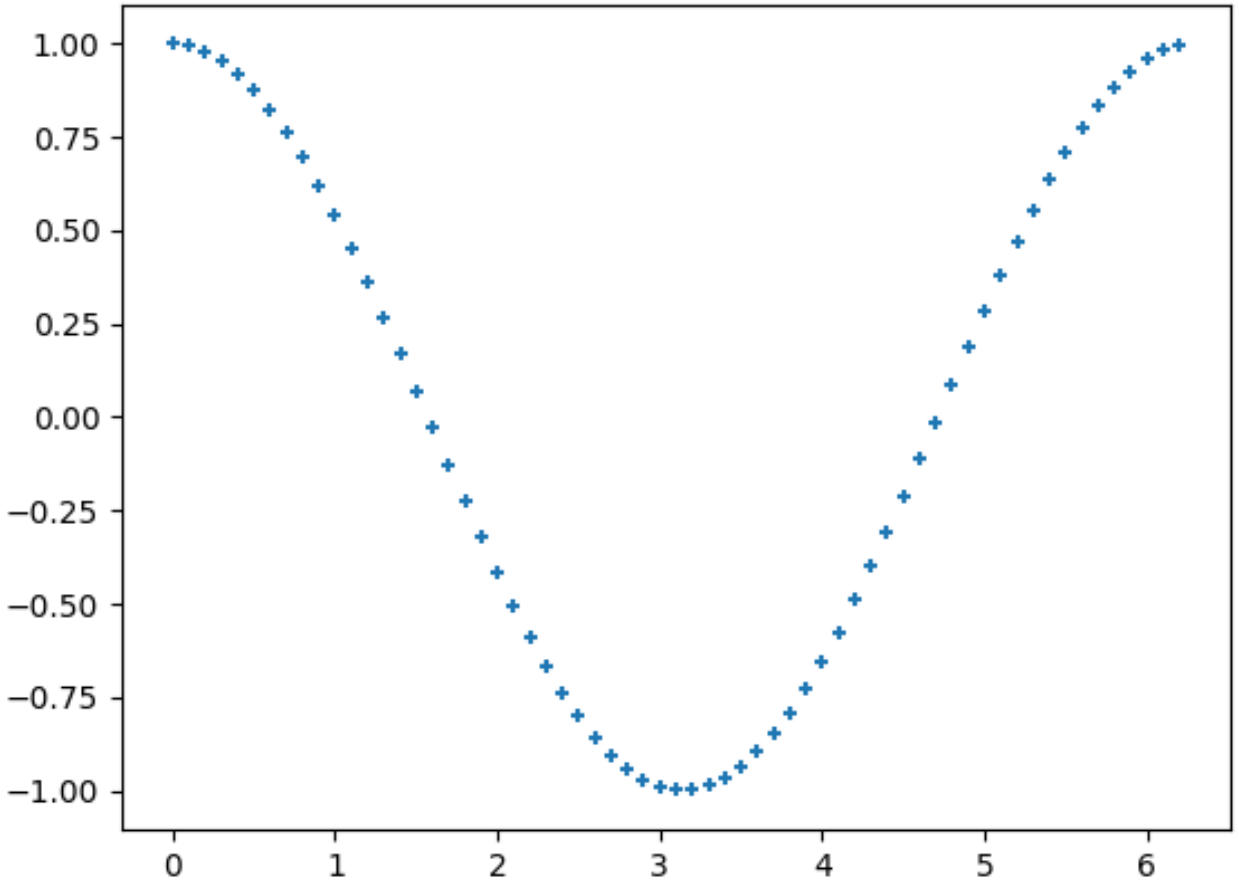
```
>>> a = np.arange(0, 2.0*np.pi, 0.1)
>>> b = np.cos(a)
>>> pl.scatter(a,b)
>>> pl.show()
```



13.5.2 绘制散点图

- 修改散点符号与大小

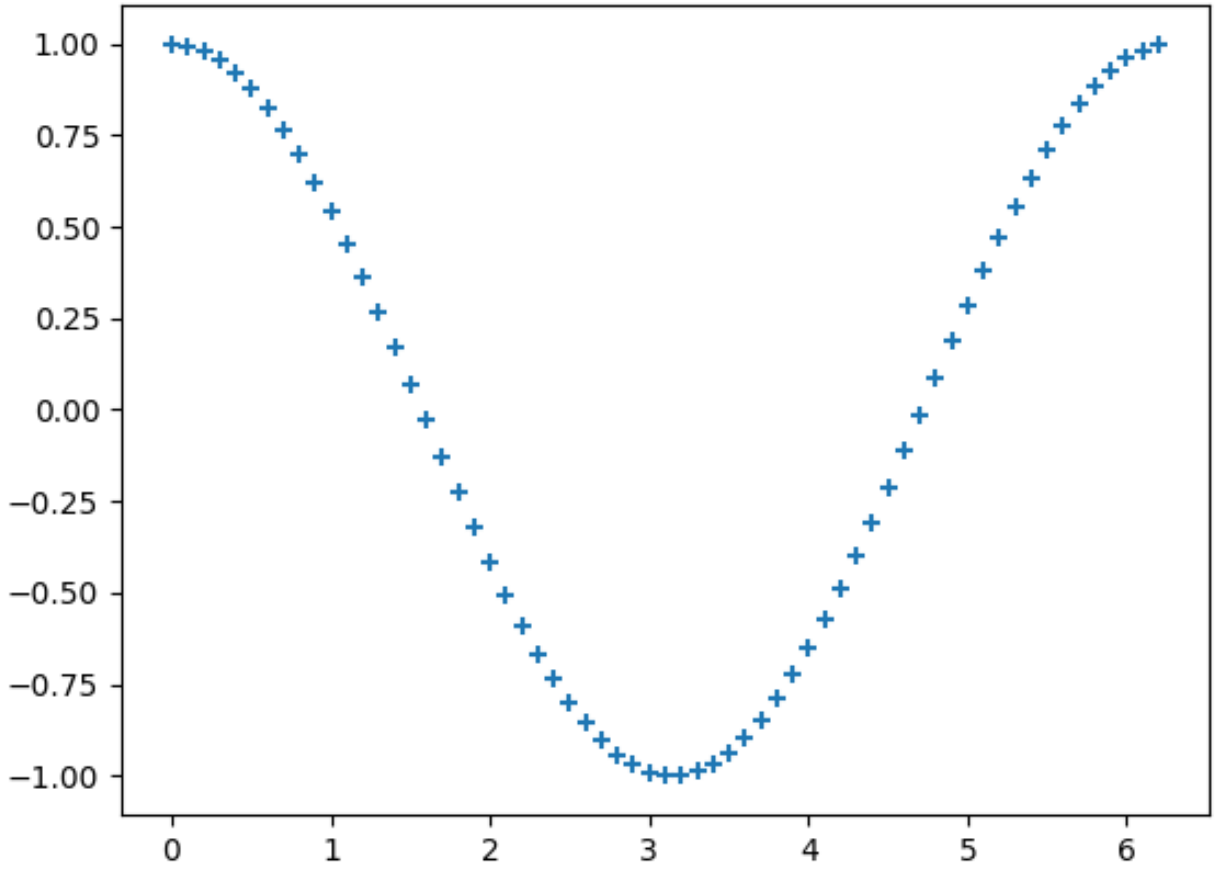
```
>>> pl.scatter(a,b,s=20,marker='+')
>>> pl.show()
```



13.5.2 绘制散点图

- 修改线宽

```
>>> pl.scatter(a,b,s=20,linewidths=5,marker='+')
>>> pl.show()
```

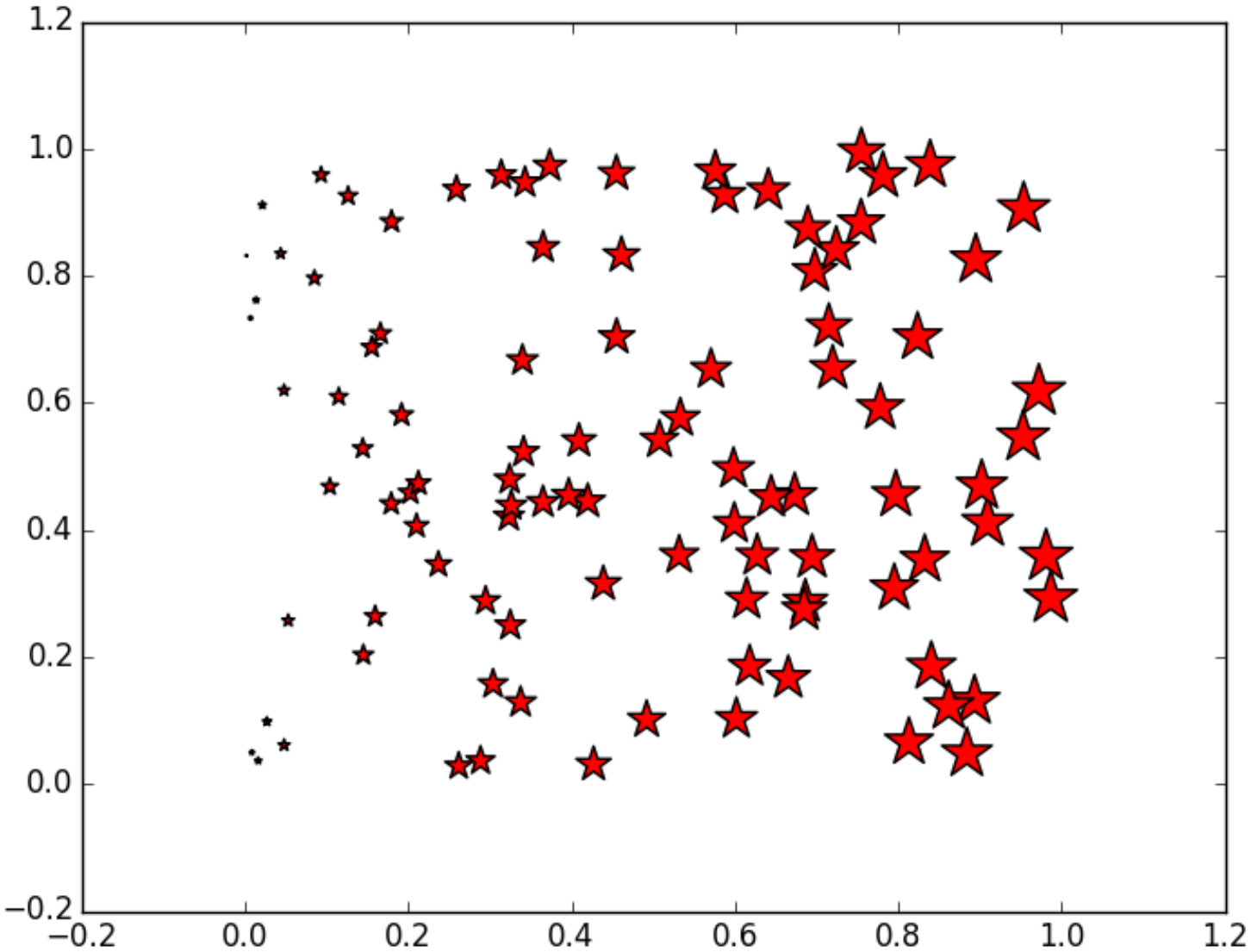


13.5.2 绘制散点图

- 修改颜色

```
>>> import matplotlib.pylab as pl
>>> import numpy as np
>>> x = np.random.random(100)
>>> y = np.random.random(100)
>>> pl.scatter(x,y,s=x*500,c=u'r',marker=u'*')
# s指大小，c指颜色，marker指符号形状
>>> pl.show()
```

13.5.2 绘制散点图



13.5.3 绘制饼状图

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


#The slices will be ordered and plotted counter-clockwise.
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
colors = ['yellowgreen', 'gold', '#FF0000', 'lightcoral']
explode = (0, 0.1, 0, 0.1)                    # 使饼状图中第2片和第4片裂开


fig = plt.figure()
ax = fig.gca()
```

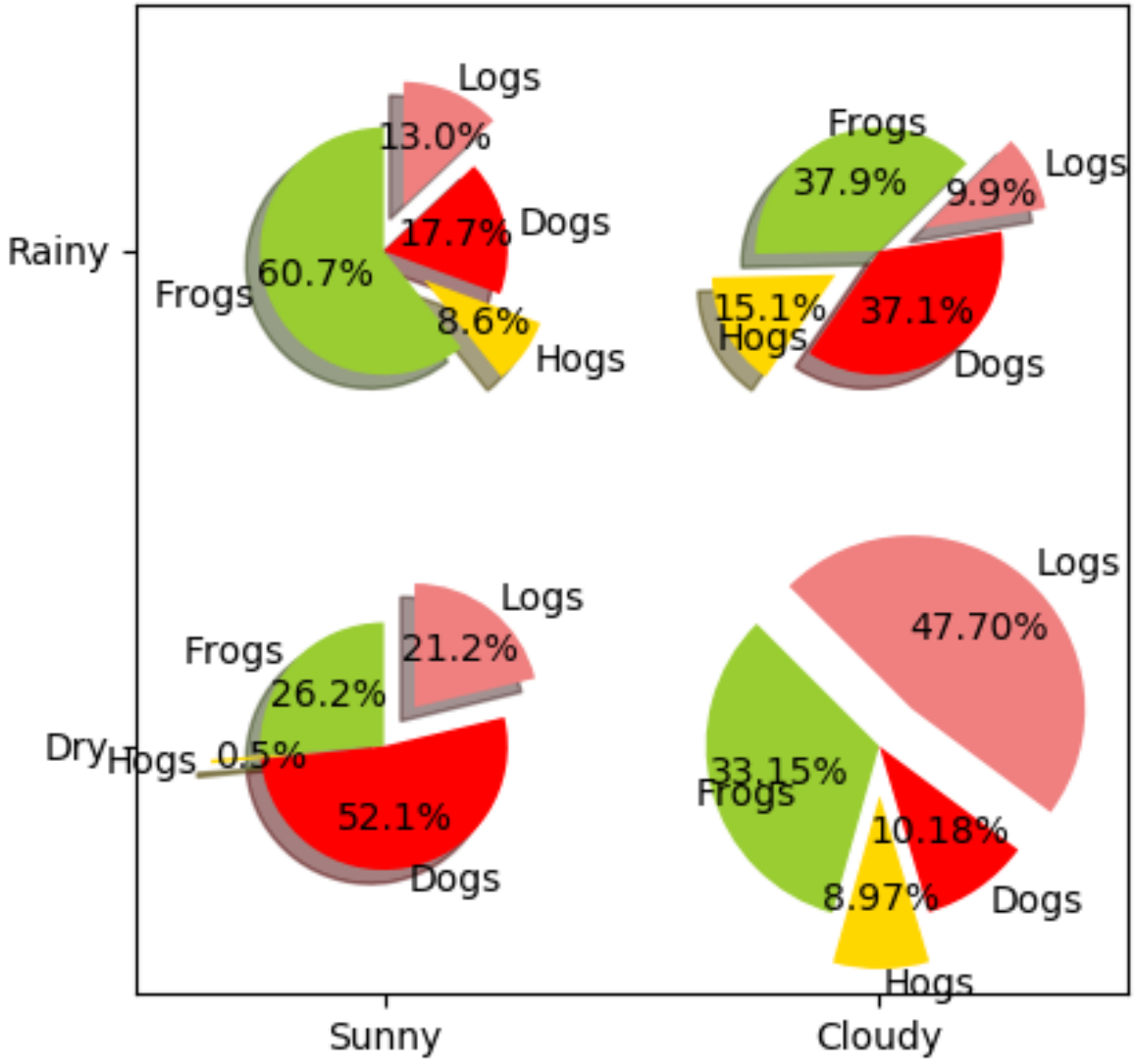
13.5.3 绘制饼状图

```
ax.pie(np.random.random(4), explode=explode, labels=labels, colors=colors,
       autopct='%1.1f%%', shadow=True, startangle=90,
       radius=0.25, center=(0, 0), frame=True)   # autopct设置饼内百分比的格式
ax.pie(np.random.random(4), explode=explode, labels=labels, colors=colors,
       autopct='%1.1f%%', shadow=True, startangle=45,
       radius=0.25, center=(1, 1), frame=True)
ax.pie(np.random.random(4), explode=explode, labels=labels, colors=colors,
       autopct='%1.1f%%', shadow=True, startangle=90,
       radius=0.25, center=(0, 1), frame=True)
ax.pie(np.random.random(4), explode=explode, labels=labels, colors=colors,
       autopct='%1.2f%%', shadow=False, startangle=135,
       radius=0.35, center=(1, 0), frame=True)
```

13.5.3 绘制饼状图

```
ax.set_xticks([0, 1])                          # 设置坐标轴刻度
ax.set_yticks([0, 1])

ax.set_xticklabels(["Sunny", "Cloudy"])  # 设置坐标轴刻度上的标签
ax.set_yticklabels(["Dry", "Rainy"])

ax.set_xlim((-0.5, 1.5))                       # 设置坐标轴跨度
ax.set_ylim((-0.5, 1.5))

ax.set_aspect('equal')                         # 设置纵横比相等

plt.show()
```

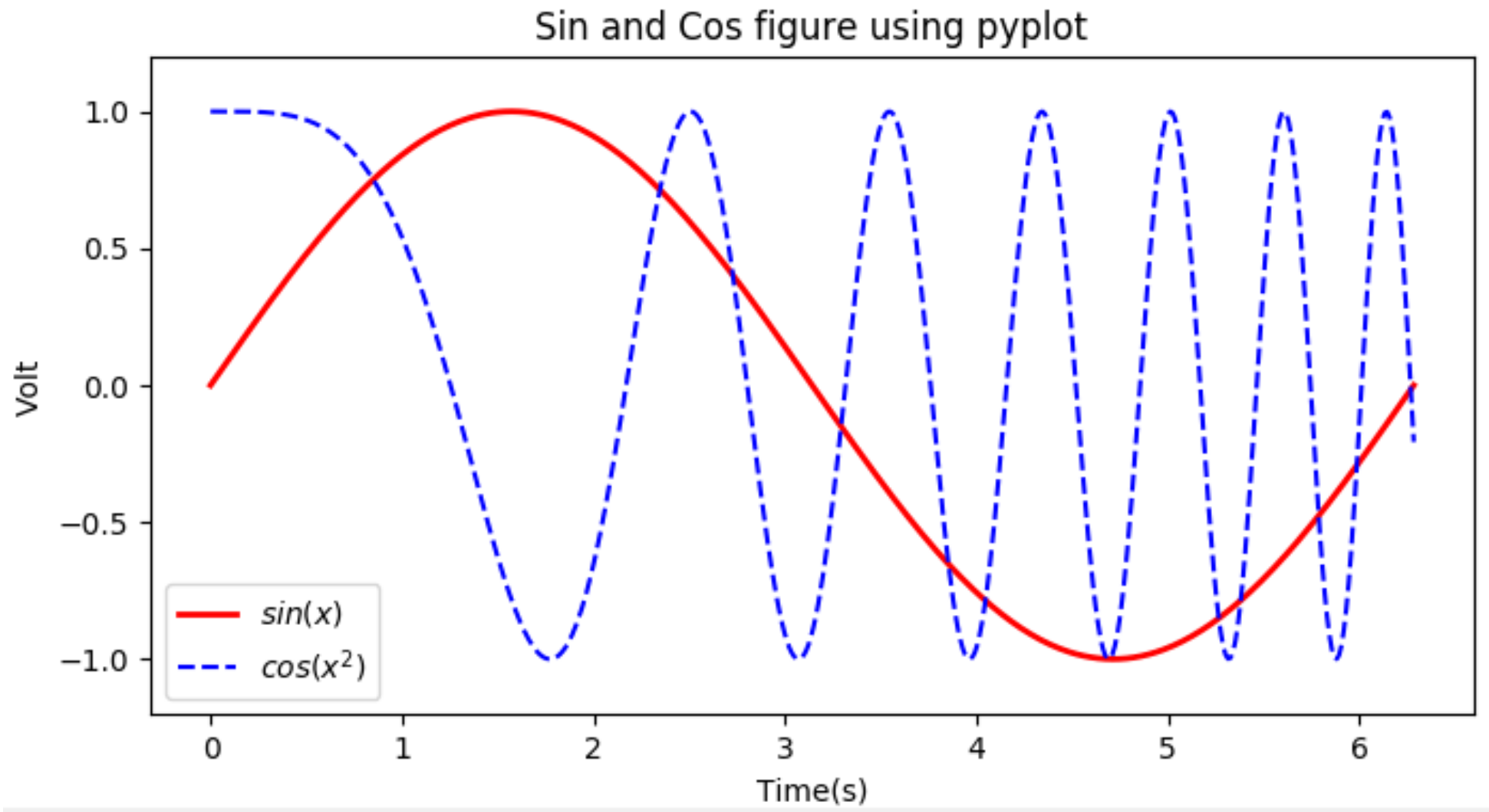
13.5.3 绘制饼状图


Rainy
Dry
Sunny
Cloudy
Logs
13.0%
Frogs
60.7%
Dogs
17.7%
Hogs
8.6%
Frogs
37.9%
Logs
9.9%
Hogs
15.1%
Dogs
37.1%
Logs
21.2%
Frogs
26.2%
Hogs
0.5%
Dogs
52.1%
Logs
47.70%
Frogs
33.15%
Dogs
10.18%
Hogs
8.97%

13.5.4 使用pyplot绘制，多个图形在一起显示

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 500)
y = np.sin(x)
z = np.cos(x*x)
plt.figure(figsize=(8,4))
# 标签前后加$将使用内嵌的LaTex引擎将其显示为公式
plt.plot(x,y,label='$sin(x)$',color='red',linewidth=2)    # 红色，2个像素宽
plt.plot(x,z,'b--',label='$cos(x^2)$')                    # 蓝色，虚线
plt.xlabel('Time(s)')
plt.ylabel('Volt')
plt.title('Sin and Cos figure using pyplot')
plt.ylim(-1.2,1.2)
plt.legend()                                              # 显示图例
plt.show()                                                # 显示绘图窗口
```

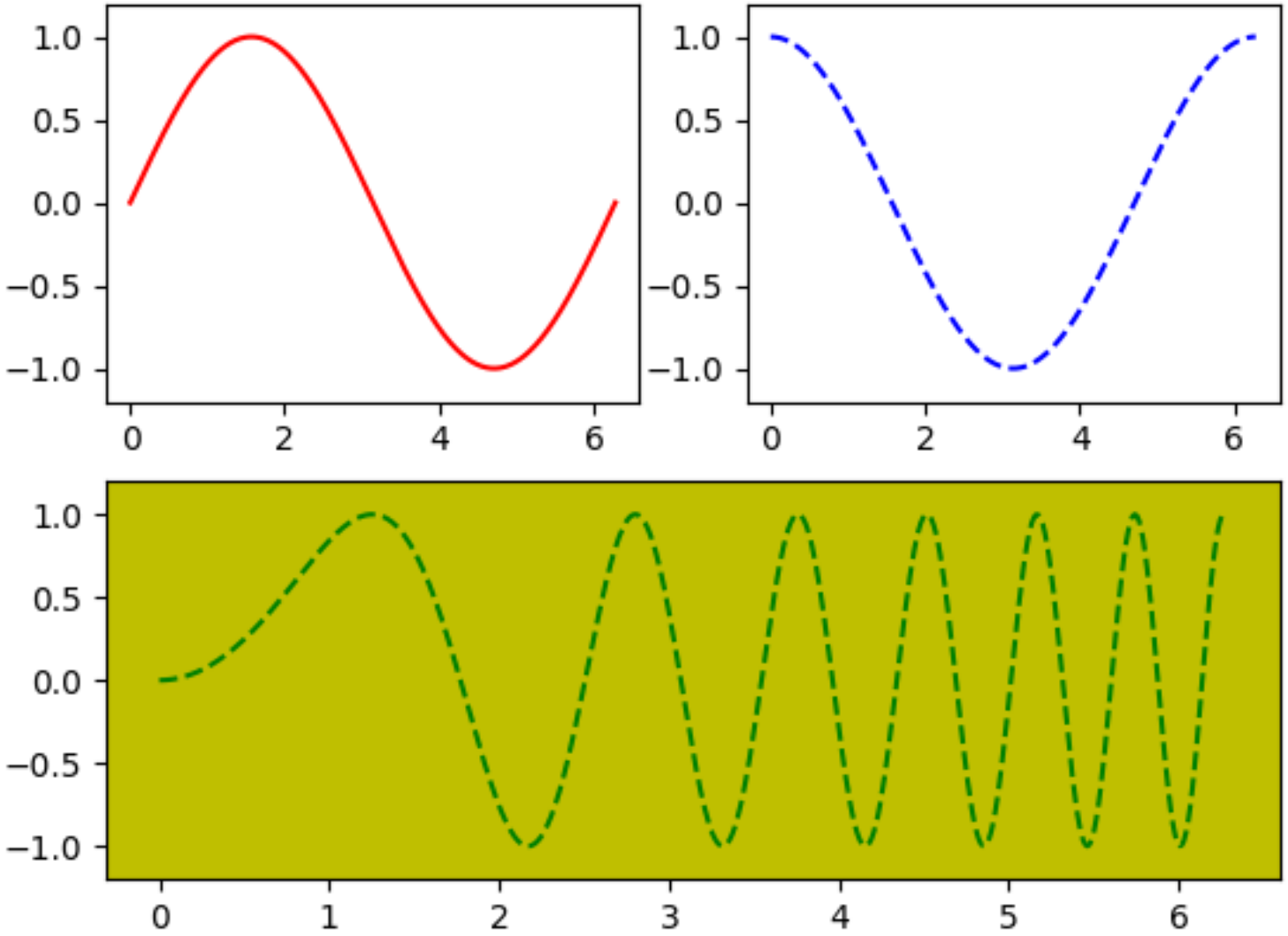
13.5.4 使用pyplot绘制，多个图形在一起显示



13.5.5 使用pyplot绘制，多个图形单独显示

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

x= np.linspace(0, 2*np.pi, 500)              # 创建自变量数组
y1 = np.sin(x)                               # 创建函数值数组
y2 = np.cos(x)
y3 = np.sin(x*x)
plt.figure(1)                                # 创建图形
ax1 = plt.subplot(2,2,1)                     # 第一行第一列图形
ax2 = plt.subplot(2,2,2)                     # 第一行第二列图形
ax3 = plt.subplot(212, facecolor='y')        # 第二行
plt.sca(ax1)                                 # 选择ax1
plt.plot(x,y1,color='red')                   # 绘制红色曲线
plt.ylim(-1.2,1.2)                           # 限制y坐标轴范围
plt.sca(ax2)                                 # 选择ax2
plt.plot(x,y2,'b--')                         # 绘制蓝色曲线
plt.ylim(-1.2,1.2)
plt.sca(ax3)                                 # 选择ax3
plt.plot(x,y3,'g--')
plt.ylim(-1.2,1.2)
plt.show()
```

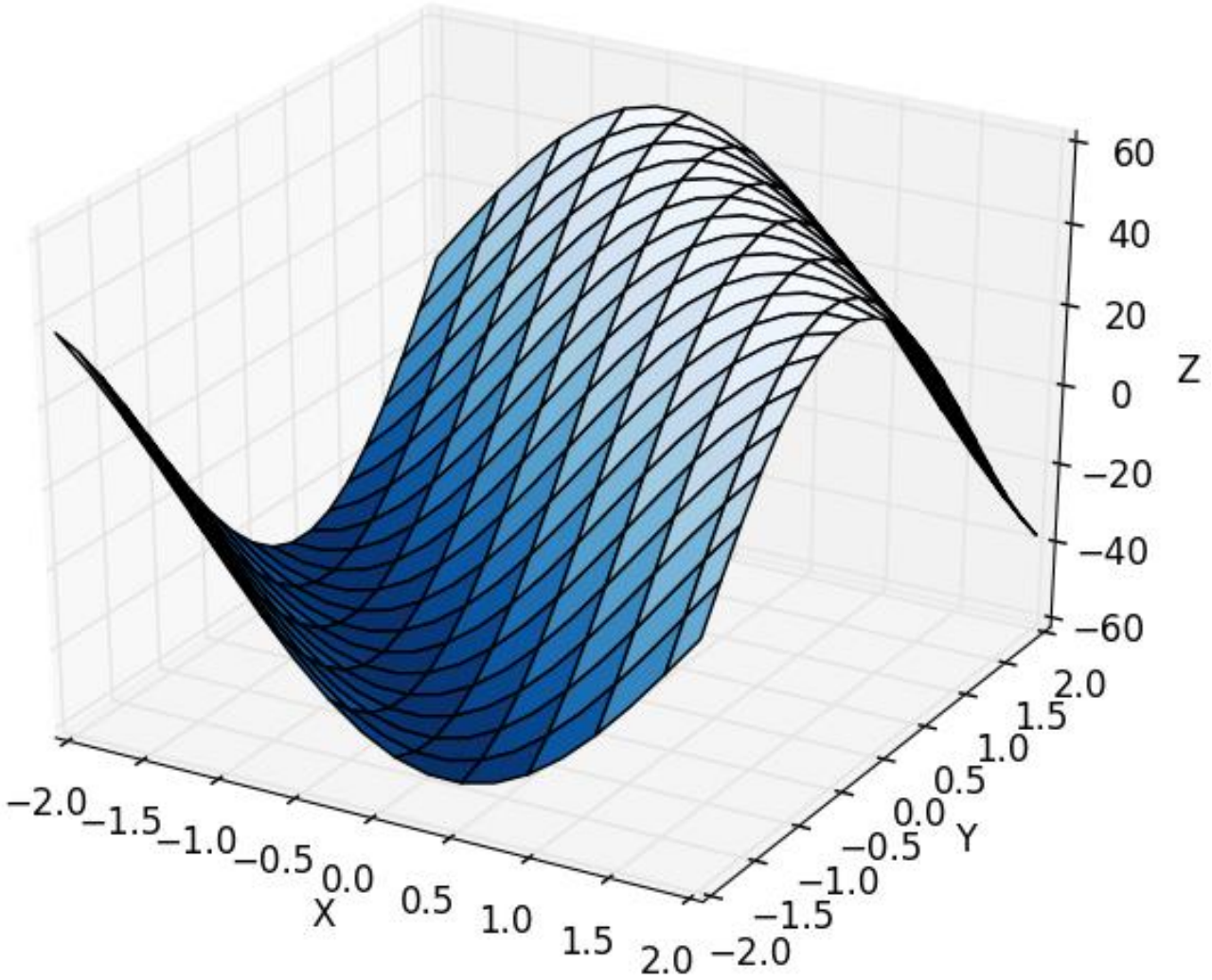
13.5.5 使用pyplot绘制，多个图形单独显示



13.5.6 绘制三维图形

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d

x,y = np.mgrid[-2:2:20j, -2:2:20j]          # 步长使用虚数
                                            # 虚部表示点的个数
                                            # 并且包含end
z = 50 * np.sin(x+y)                        # 测试数据
ax = plt.subplot(111, projection='3d')      # 三维图形
ax.plot_surface(x,y,z,rstride=2, cstride=1, cmap=plt.cm.Blues_r)
ax.set_xlabel('X')                          # 设置坐标轴标签
ax.set_ylabel('Y')
ax.set_zlabel('Z')
plt.show()
```

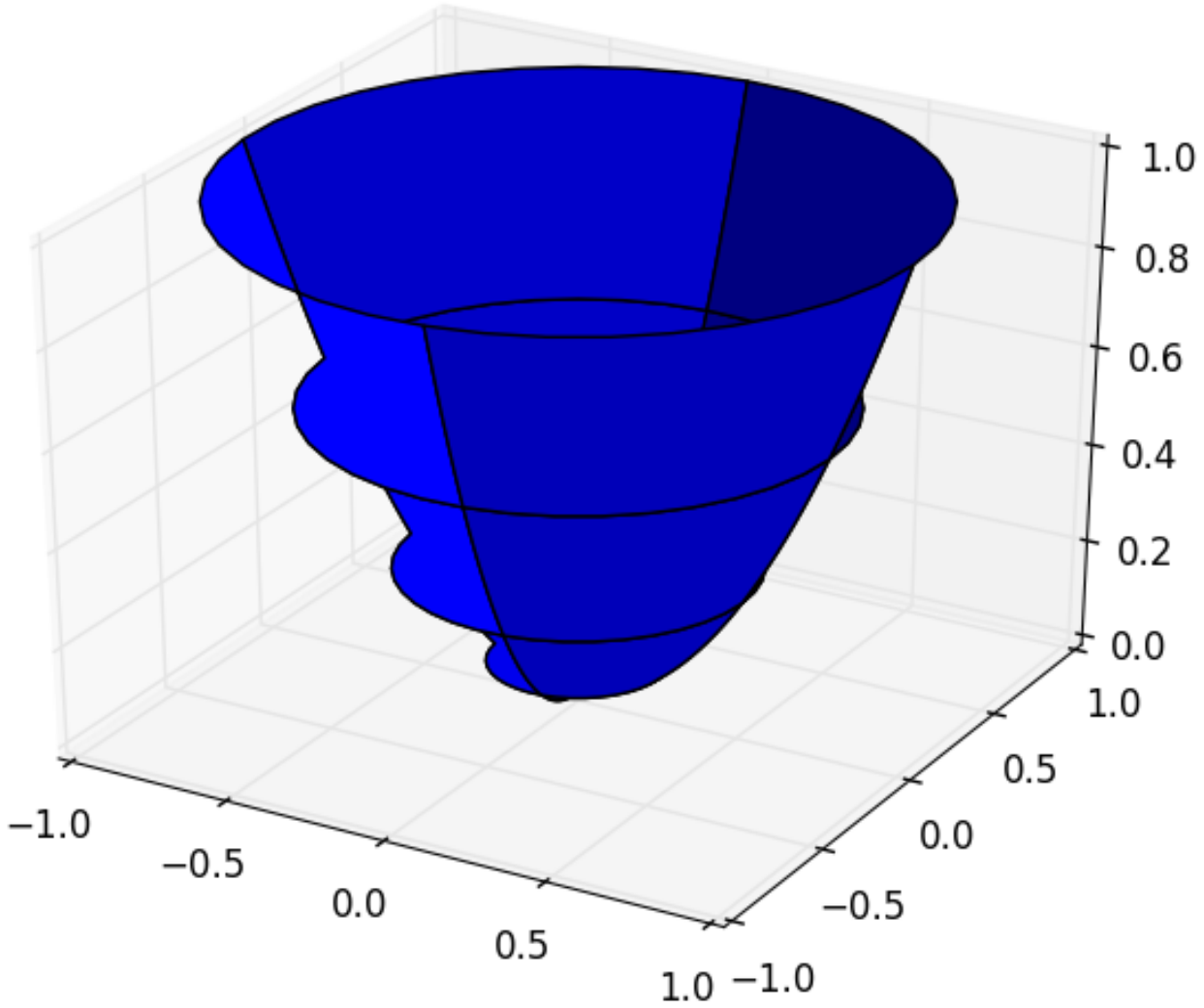
13.5.6 绘制三维图形


60
40
20
0
−20
−40
−60
Z
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
X
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
Y

13.5.6 绘制三维图形

```
import pylab as pl
import numpy as np
import mpl_toolkits.mplot3d
rho, theta = np.mgrid[0:1:40j, 0:2*np.pi:40j]
z = rho**2
x = rho*np.cos(theta)
y = rho*np.sin(theta)
ax = pl.subplot(111, projection='3d')
ax.plot_surface(x,y,z)
pl.show()
```

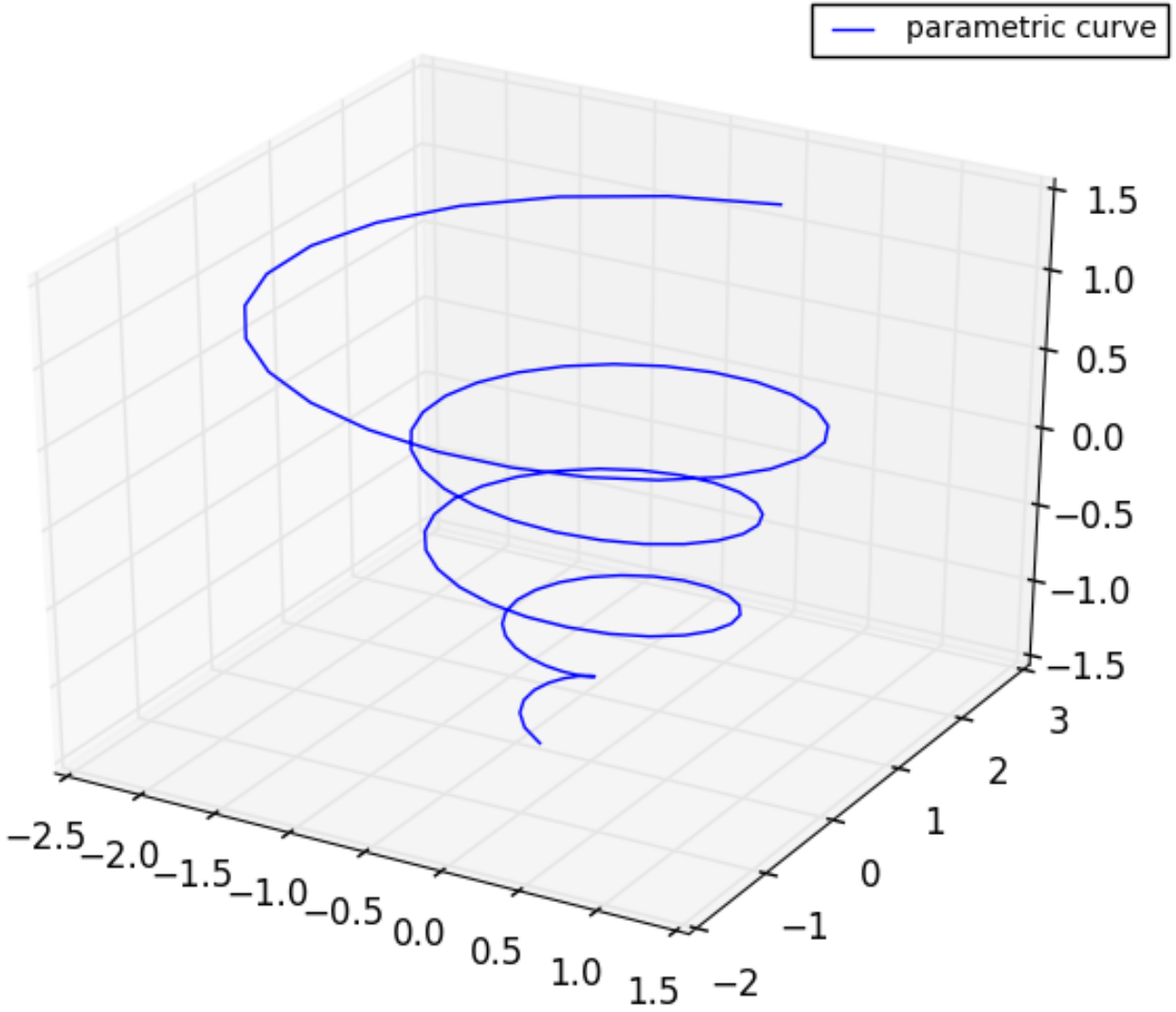
13.5.6 绘制三维图形



13.5.7 绘制三维曲线

```
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10          # 图例字号
fig = plt.figure()
ax = fig.gca(projection='3d')                 # 三维图形
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-4, 4, 100)*0.3               # 测试数据
r = z**3 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='parametric curve')
ax.legend()
plt.show()
```

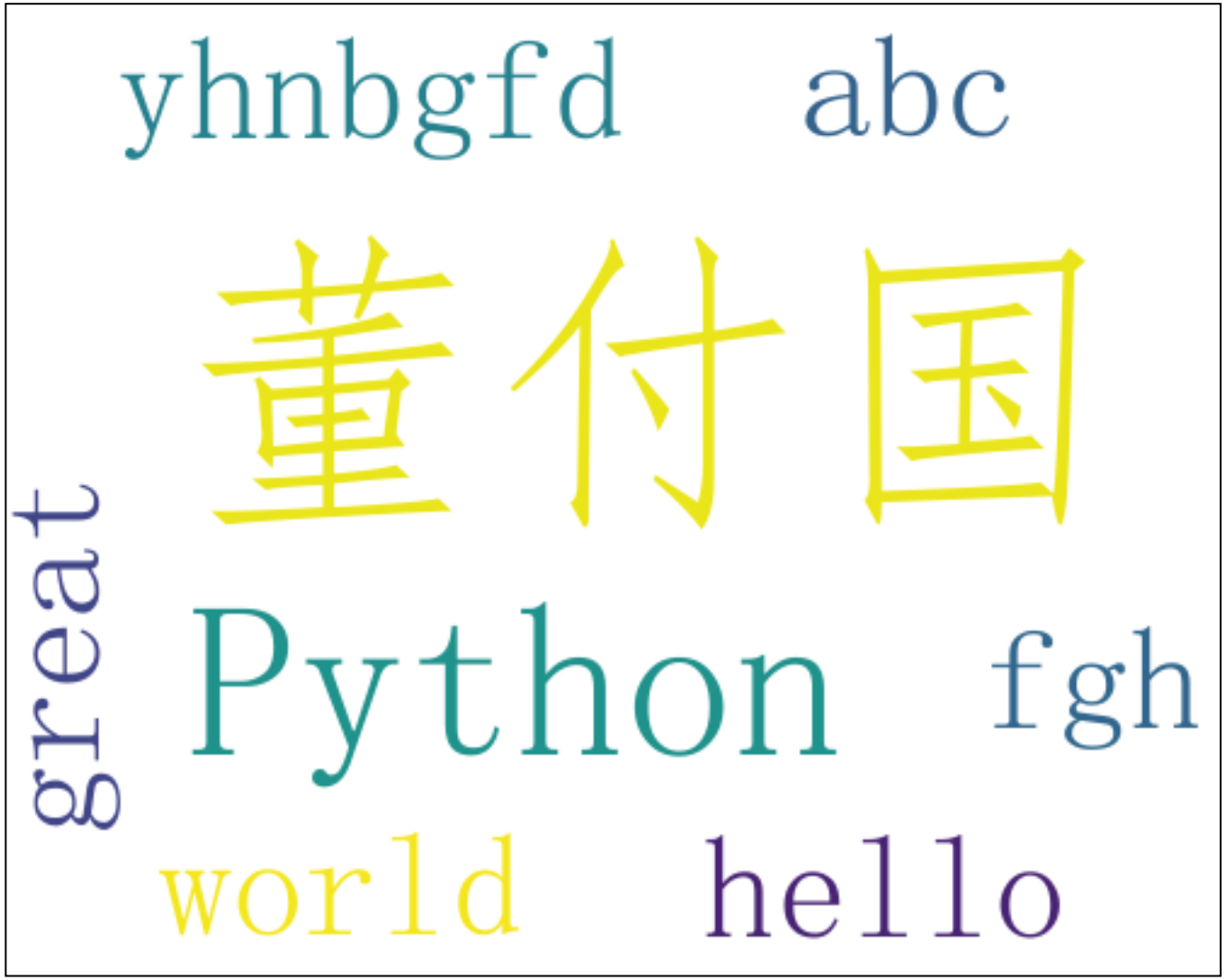
13.5.7 绘制三维曲线


parametric curve
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
3
2
1
0
−1
−2
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

13.6 生成词云

```
import random
import string
import wordcloud

def show(s):
    # 创建wordcloud对象
    wc = wordcloud.WordCloud(
        r'C:\windows\fonts\simfang.ttf', width=500, height=400,
        background_color='white', font_step=3,
        random_state=False, prefer_horizontal=0.9)
    # 创建并显示词云
    t = wc.generate(s)
    t.to_image().save('t.png')

# 如果空间足够，就全部显示
# 如果词太多，就按频率显示，频率越高的词越大
show('''hello world 董付国 董付国 董付国 董付国
 abc fgh yhnbgfd 董付国 董付国 董付国 董付国 Python great Python Python''')
```

13.6 生成词云



补充1：使用线性回归拟合平面最佳直线及预测

- 代码采用sklearn扩展库实现，使用线性回归算法解决下面的问题：根据平面上已知3个点的坐标，拟合最佳直线斜率k和截距b，然后根据拟合的结果对给出的x坐标进行预测，得到y坐标。

补充1：使用线性回归拟合平面最佳直线及预测

```
from sklearn import linear_model


def linearRegressionPredict(x, y):
    lr = linear_model.LinearRegression()
    # 拟合
    lr.fit(x, y)
    return lr


# 平面上三个点的x轴坐标
x = [[1], [5], [7]]
# 平面上三个点的y轴坐标
y = [[3], [100], [120]]
```

补充1：使用线性回归拟合平面最佳直线及预测

```
# 根据已知3个点拟合最佳直线的系数和截距
lr = linearRegressionPredict(x, y)
# 查看最佳拟合系数
print('k:', lr.coef_)
# 截距
print('b:', lr.intercept_)

# 测试代码，预测
xs = [[[3]], [[5]], [[7]], [[10]]]
for item in xs:
    print(item, ':', lr.predict(item))
```

补充1：使用线性回归拟合平面最佳直线及预测

运行结果：

```
k: [[ 20.17857143]]
b: [-13.10714286]
[[3]] : [[ 47.42857143]]
[[5]] : [[ 87.78571429]]
[[7]] : [[ 128.14285714]]
[[10]] : [[ 188.67857143]]
```

补充2：Python+sklearn使用线性回归算法预测儿童身高

- **问题描述：**一个人的身高除了随年龄变大而增长之外，在一定程度上还受到遗传和饮食以及其他因素的影响，代码中假定受年龄、性别、父母身高、祖父母身高和外祖父母身高共同影响，并假定大致符合线性关系。

补充2：Python+sklearn使用线性回归算法预测儿童身高

```
import copy
import numpy as np
from sklearn import linear_model

def linearRegressionPredict(x, y):
    lr = linear_model.LinearRegression()
    # 拟合
    lr.fit(x, y)
    return lr
```

补充2：Python+sklearn使用线性回归算法预测儿童身高

```
# 儿童年龄,性别（0女1男）,父亲身高,母亲身高,祖父身高,祖母身高,外祖父身高,外祖母身高
x = np.array([[1, 0, 180, 165, 175, 165, 170, 165],\
              [3, 0, 180, 165, 175, 165, 173, 165],\
              [4, 0, 180, 165, 175, 165, 170, 165],\
              [6, 0, 180, 165, 175, 165, 170, 165],\
              [8, 1, 180, 165, 175, 167, 170, 165],\
              [10, 0, 180, 166, 175, 165, 170, 165],\
              [11, 0, 180, 165, 175, 165, 170, 165],\
              [12, 0, 180, 165, 175, 165, 170, 165],\
              [13, 1, 180, 165, 175, 165, 170, 165],\
              [14, 0, 180, 165, 175, 165, 170, 165],\
              [17, 0, 170, 165, 175, 165, 170, 165]])
```

补充2：Python+sklearn使用线性回归算法预测儿童身高

```
# 儿童身高，单位：cm
y = np.array([60, 90, 100, 110,\
              130, 140, 150, 164,\
              160, 163, 168])
# 根据已知数据拟合最佳直线的系数和截距
lr = linearRegressionPredict(x, y)
# 查看最佳拟合系数
print('k:', lr.coef_)
# 截距
print('b:', lr.intercept_)
```

补充2：Python+sklearn使用线性回归算法预测儿童身高

```
# 预测
xs = np.array([[10, 0, 180, 165, 175, 165, 170, 165],\
               [17, 1, 173, 153, 175, 161, 170, 161],\
               [34, 0, 170, 165, 170, 165, 170, 165]])
for item in xs:
    # 深复制，假设超过18岁以后就不再长高了
    item1 = copy.deepcopy(item)
    if item1[0] > 18:
        item1[0] = 18
    print(item, ':', lr.predict(item1.reshape(1,-1)))
```

补充2：Python+sklearn使用线性回归算法预测儿童身高

运行结果：

```
k: [  8.03076923e+00  -4.65384615e+00   2.87769231e+00  -5.61538462e-01
   7.10542736e-15   5.07692308e+00   1.88461538e+00   0.00000000e+00]
b: -1523.15384615
[ 10   0 180 165 175 165 170 165] : [ 140.56153846]
[ 17   1 173 153 175 161 170 161] : [ 158.41]
[ 34   0 170 165 170 165 170 165] : [ 176.03076923]
```

补充3：KNN分类算法实现根据身高和体重对体型分类

- KNN算法是k-Nearest Neighbor Classification的简称，也就是k近邻分类算法。基本思路是在特征空间中查找k个最相似或者距离最近的样本，然后根据k个最相似的样本对未知样本进行分类。基本步骤为：
  - 计算已知样本空间中所有点与未知样本的距离；
  - 对所有距离按升序排列；
  - 确定并选取与未知样本距离最小的k个样本或点；
  - 统计选取的k个点所属类别的出现频率；
  - 把出现频率最高的类别作为预测结果，即未知样本所属类别。
- 下面的代码模拟了上面的算法思路和步骤，以身高+体重对肥胖程度进行分类为例，采用欧几里得距离。

补充3：KNN分类算法实现根据身高和体重对体型分类

```
from collections import Counter
import numpy as np


# 已知样本数据
# 每行数据分别为性别,身高,体重
knownData = ((1, 180, 85), (1, 180, 86), (1, 180, 90), (1, 180, 100),
             (1, 185, 120), (1, 175, 80), (1, 175, 60), (1, 170, 60),
             (1, 175, 90), (1, 175, 100), (1, 185, 90), (1, 185, 80))
knownTarget = ('稍胖', '稍胖', '稍胖', '过胖',
               '太胖', '正常', '偏瘦', '正常',
               '过胖', '太胖', '正常', '偏瘦')
```

补充3：KNN分类算法实现根据身高和体重对体型分类

```
def KNNPredict(current, knownData=knownData,  knownTarget=knownTarget, k=3):
    # current为未知样本，格式为(性别,身高,体重)
    data = dict(zip(knownData, knownTarget))
    # 如果未知样本与某个已知样本精确匹配，直接返回结果
    if current in data.keys():
        return data[current]
    # 按性别过滤，只考虑current性别一样的样本数据
    g = lambda item:item[0][0]==current[0]
    samples = list(filter(g, data.items()))
    g = lambda item:((item[0][1]-current[1])**2+\
                     (item[0][2]-current[2])**2)**0.5
    distances = sorted(samples, key=g)
    # 选取距离最小的前k个
    distances = (item[1] for item in distances[:k])
    # 计算选取的k个样本所属类别的出现频率
    # 选择频率最高的类别作为结果
    return Counter(distances).most_common(1)[0][0]
```

补充3：KNN分类算法实现根据身高和体重对体型分类

```
unKnownData = [(1, 180, 70), (1, 160, 90), (1, 170, 85)]
for current in unKnownData:
    print(current, ':', KNNPredict(current))
```

运行结果为：

```
(1, 180, 70) : 偏瘦
(1, 160, 90) : 过胖
(1, 170, 85) : 正常
```

补充3：KNN分类算法实现根据身高和体重对体型分类

❖ 下面的代码使用扩展库sklearn中的k近邻分类算法处理了同样的问题：

```
# 使用sklearn库的k近邻分类模型
from sklearn.neighbors import KNeighborsClassifier

# 创建并训练模型
clf = KNeighborsClassifier(n_neighbors=3, weights='distance')
clf.fit(knownData, knownTarget)

# 分类
for current in unKnownData:
    print(current, end=' : ')
    current = np.array(current).reshape(1,-1)
    print(clf.predict(current)[0])
```

运行结果为：

```
(1, 180, 70) : 偏瘦
(1, 160, 90) : 过胖
(1, 170, 85) : 正常
```

补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

- 时序图、自相关图和偏相关图是判断时间序列数据是否平稳的重要依据。

```
from random import randrange
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

```
def generateData(startDate, endDate):
    df = pd.DataFrame([300+i*30+randrange(50) for i in range(31)],\
                      columns=['营业额'],\
                      index=pd.date_range(startDate, endDate, freq='D'))
    return df

# 生成测试数据，模拟某商店营业额
data = generateData('20170601',  '20170701')
print(data)
```

补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

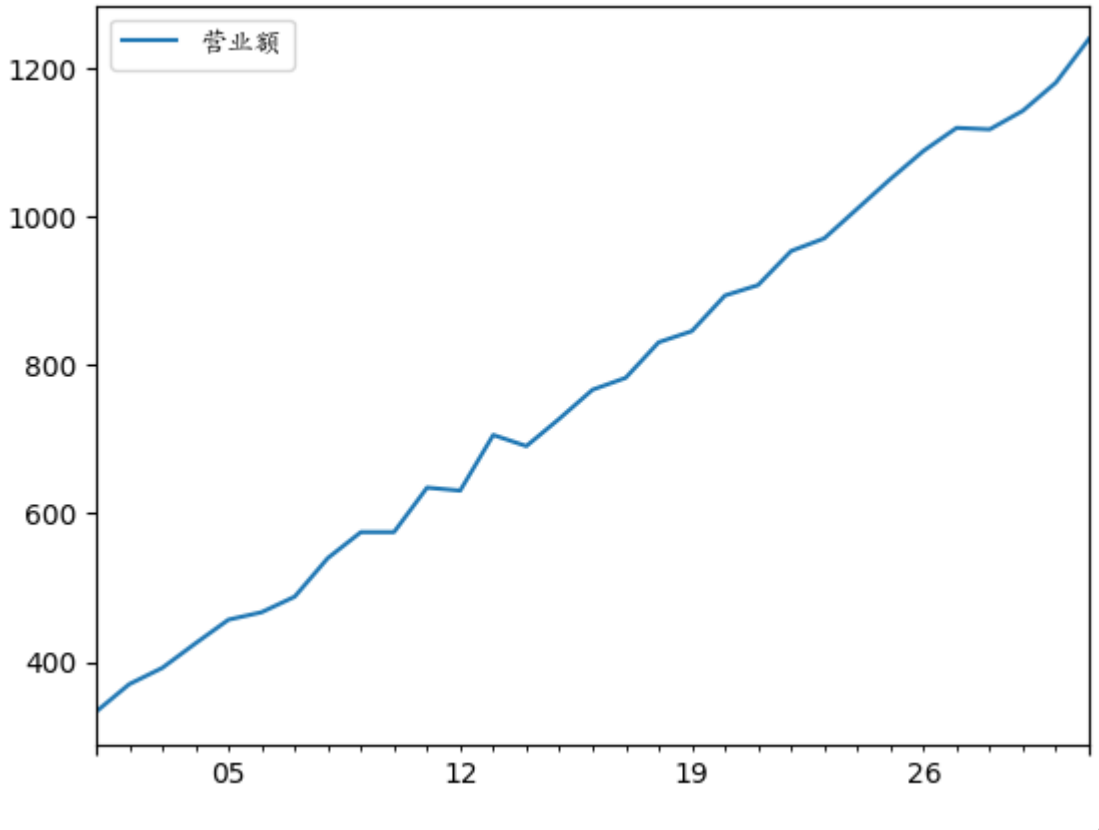
```
# 绘制时序图
myfont = fm.FontProperties(fname=r'C:\Windows\Fonts\STKAITI.ttf')
data.plot()
plt.legend(prop=myfont)
plt.show()
# 绘制自相关图
plot_acf(data).show()
# 绘制偏自相关图
plot_pacf(data).show()
```

补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

	营业额
2017-06-01	333
2017-06-02	370
2017-06-03	392
2017-06-04	425
2017-06-05	457
2017-06-06	467
2017-06-07	488
2017-06-08	540
2017-06-09	575
2017-06-10	575
2017-06-11	635
2017-06-12	631
2017-06-13	706
2017-06-14	691
2017-06-15	728
2017-06-16	767
2017-06-17	783
2017-06-18	831
2017-06-19	846
2017-06-20	894
2017-06-21	908
2017-06-22	954
2017-06-23	971
2017-06-24	1011
2017-06-25	1051
2017-06-26	1089
2017-06-27	1120
2017-06-28	1118
2017-06-29	1143
2017-06-30	1181
2017-07-01	1240

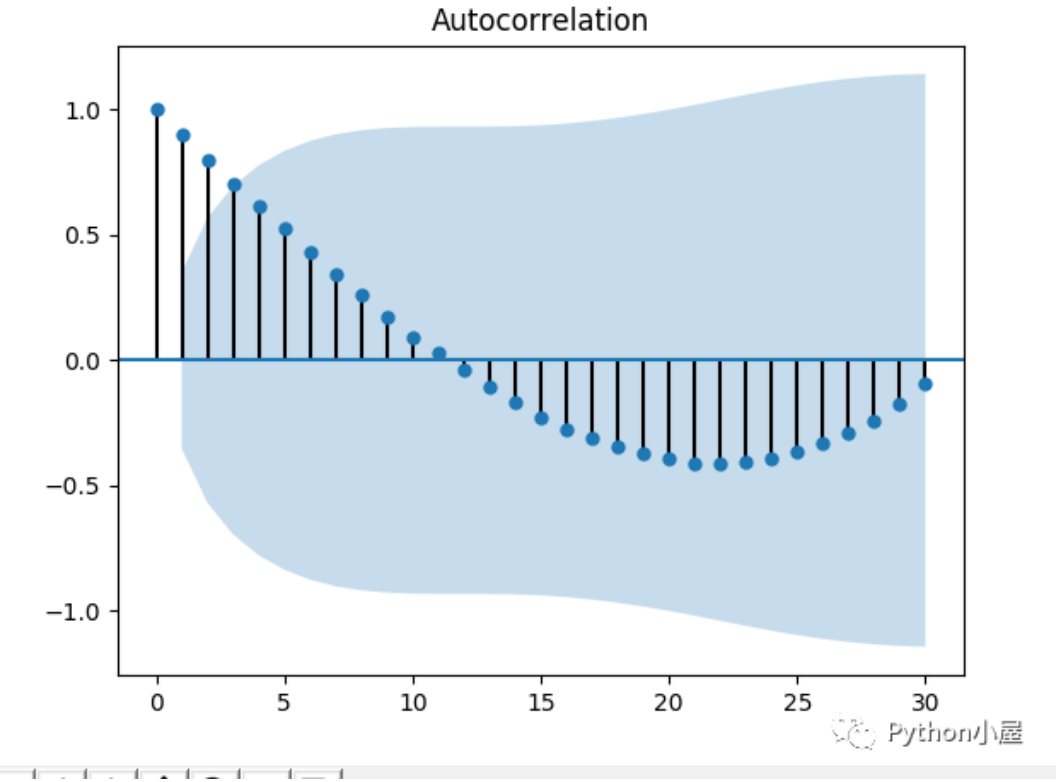
补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

- 时序图：有明显的增长趋势，原始数据属于不平稳序列。



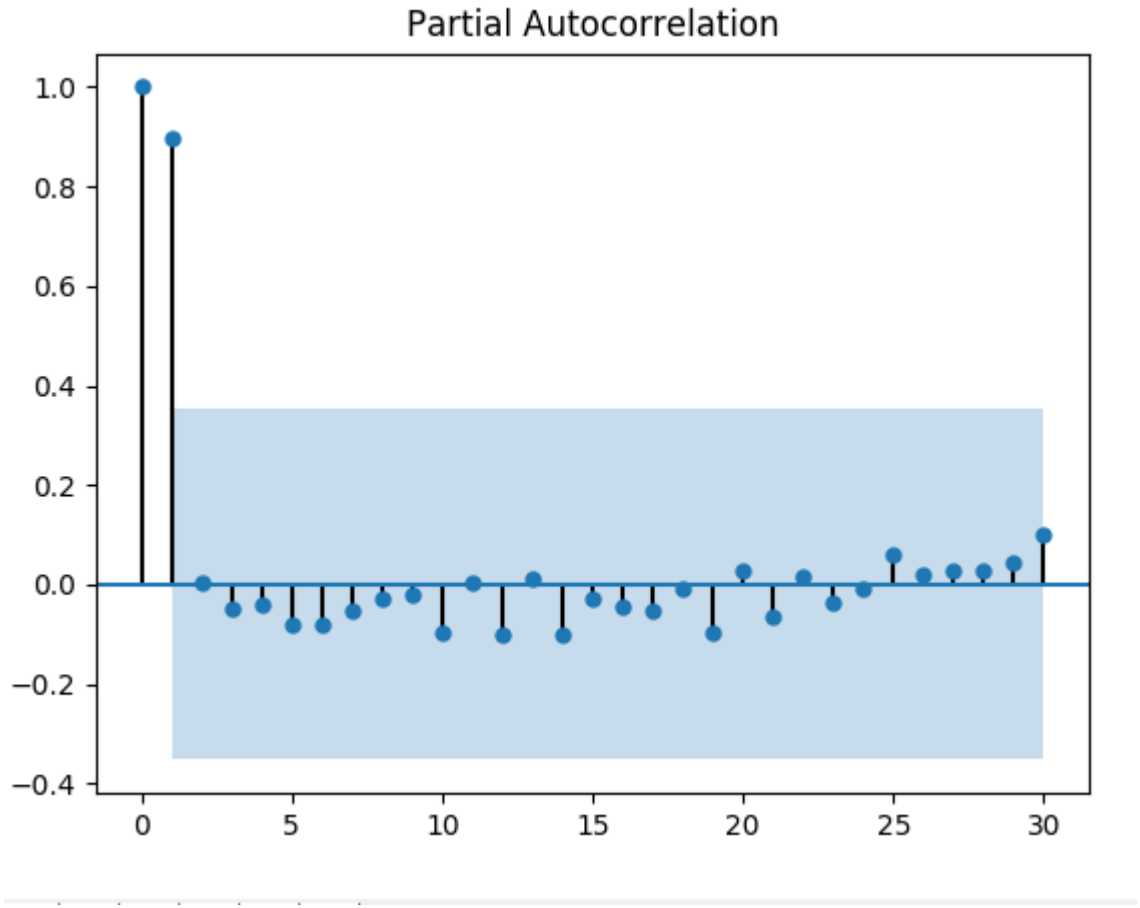
补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

- 自相关图：呈现三角对称形式，不存在截尾或拖尾，属于单调序列的典型表现形式，原始数据属于不平稳序列。



补充4：绘制时间序列数据的时序图、自相关图和偏自相关图

- 偏自相关图：不存在截尾或拖尾，属于不平稳序列。



补充5：使用系统聚类算法对随机元素进行分类

- 系统聚类算法又称层次聚类或系谱聚类，首先把样本看作各自一类，定义类间距离，选择距离最小的一对元素合并成一个新的类，重复计算各类之间的距离并重复上面的步骤，直到将所有原始元素分成指定数量的类。
- 该算法的计算复杂度比较高，不适合大数据聚类问题。

补充5：使用系统聚类算法对数据进行分类

- Python扩展库sklearn.cluster.AgglomerativeClustering提供了系统聚类算法的实现。

```
from random import randrange
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
```

补充5：使用系统聚类算法对数据进行分类

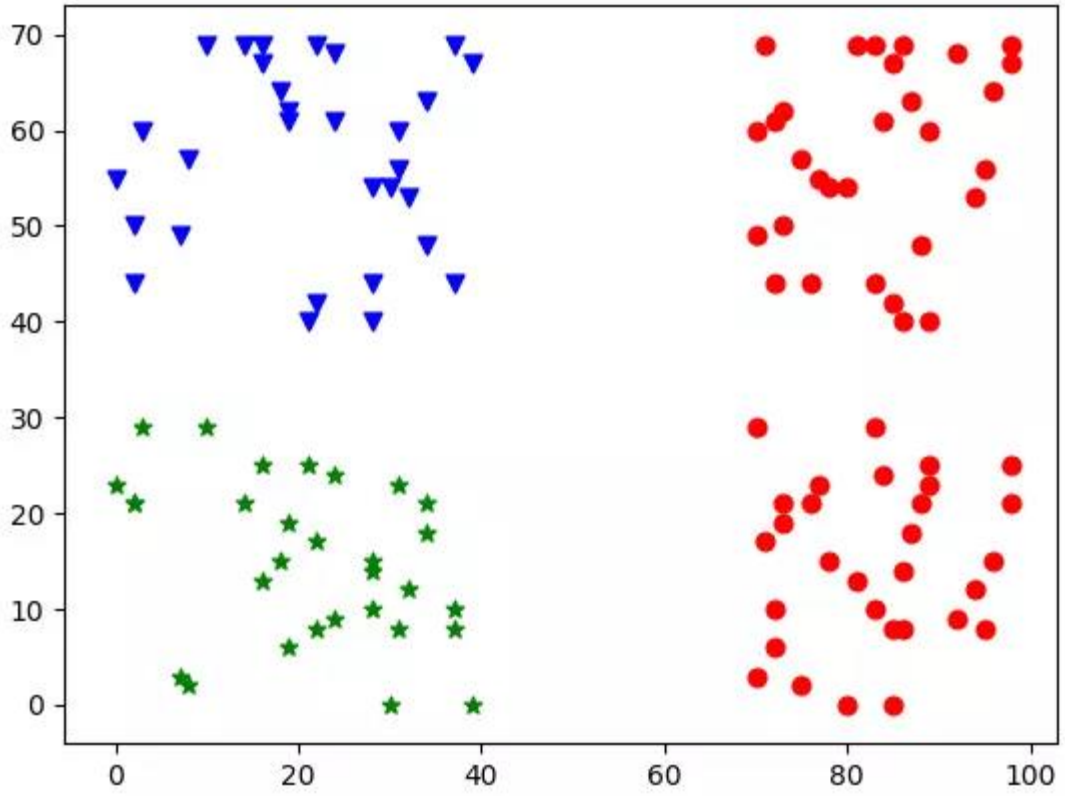
```
def generateData():
    '''生成测试数据'''
    def get(start, end):
        return [randrange(start, end) for _ in range(30)]

    x1 = get(0, 40)
    x2 = get(70, 100)
    y1 = get(0, 30)
    y2 = get(40, 70)

    data = list(zip(x1, y1)) + list(zip(x1, y2))+\
           list(zip(x2, y1)) + list(zip(x2, y2))
    return np.array(data)
```

补充5：使用系统聚类算法对数据进行分类

```
def AgglomerativeTest(n_clusters):
    '''聚类，指定类的数量，并绘制图形'''
    assert 1 <= n_clusters <= 4
    predictResult = AgglomerativeClustering(n_clusters=n_clusters,
                                            affinity='euclidean',
                                            linkage='ward').fit_predict(data)
    colors = 'rgby'
    markers = 'o*v+'
    for i in range(n_clusters):
        subData = data[predictResult==i]
        plt.scatter(subData[:,0], subData[:,1], c=colors[i], marker=markers[i], s=40)
    plt.show()

# 生成随机数据
data = generateData()
# 聚类为3个不同的类
AgglomerativeTest(3)
# 聚类为4个不同的类
AgglomerativeTest(4)
```

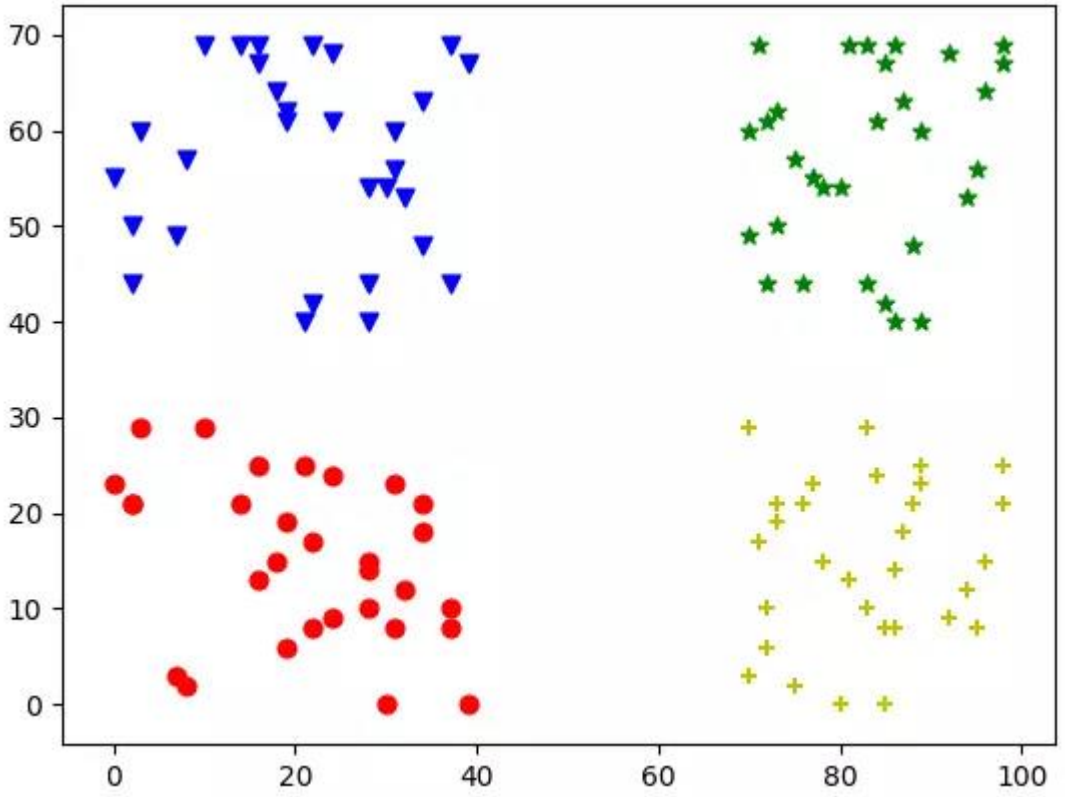
补充5：使用系统聚类算法对数据进行分类

- 聚类个数为3时的效果



补充5：使用系统聚类算法对数据进行分类

- 聚类个数为4时的效果



补充6：使用k-means聚类算法进行分类

- K-means算法的基本思想是：以空间中k个点为中心进行聚类，对最靠近他们的对象归类。通过迭代的方法，逐次更新各聚类中心的值，直至得到最好的聚类结果。
- 最终的k个聚类具有以下特点：各聚类本身尽可能的紧凑，而各聚类之间尽可能的分开。
- 该算法的最大优势在于简洁和快速，算法的关键在于预期分类数量的确定以及初始中心和距离公式的选择。

补充6：使用k-means聚类算法进行分类

- 假设要把样本集分为c个类别，算法描述如下：

（1）适当选择c个类的初始中心；

（2）在第k次迭代中，对任意一个样本，求其到c个中心的距离，将该样本归到距离最短的中心所在的类；

（3）利用均值等方法更新该类的中心值；

（4）对于所有的c个聚类中心，如果利用（2）（3）的迭代法更新后，值保持不变，则迭代结束，否则继续迭代。

补充6：使用k-means聚类算法进行分类

```
from numpy import array
from random import randrange
from sklearn.cluster import KMeans

# 获取模拟数据
X = array([[1,1,1,1,1,1,1],
           [2,3,2,2,2,2,2],
           [3,2,3,3,3,3,3],
           [1,2,1,2,2,1,2],
           [2,1,3,3,3,2,1],
           [6,2,30,3,33,2,71]])
```

补充6：使用k-means聚类算法进行分类

```
# 训练
kmeansPredicter = KMeans(n_clusters=3).fit(X)
# 原始数据分类
category = kmeansPredicter.predict(X)
print('分类情况：', category)
print('='*30)

def predict(element):
    result = kmeansPredicter.predict(element)
    print('预测结果：', result)
    print('相似元素：\n', X[category==result])

# 测试
predict([[1,2,3,3,1,3,1]])
print('='*30)
predict([[5,2,23,2,21,5,51]])
```