numpy_demo

November 2, 2018

1 Numpy 使用基础

只要我们接触 Python 数据分析，肯定会离不开 Numpy 和 Pandas。Numpy 和 Pandas 的主要区别在于 Numpy 是一个数据计算库，以 ndarray 为核心，主要针对矩阵、计算方面。而 pandas 则着重与表格，更广泛在数据分析、量化以及金融的应用。Numpy 和 Pandas 其实有很多功能也是非常类似的，实际选择看具体的需求，一般偏计算使用 Numpy，有比较复杂的表格合并查询等，可以使用 Pandas。

本 NoteBook 主要以掌握 Numpy 的基础使用为目标，按一下五个部分内容介绍 Numpy 的基本用法：
- ndarray 的特性以及基础切片索引 - 常用计算函数运用 - 常用统计函数运用 - 简单集合运算运用 - 随机函数运用

参考书推荐：《利用 Python 进行数据分析》

ndarray

创建：可以使用以下函数创建 ndarray

函数	解释
array	输入 list，tuple 转 ndarray
asarray	输入转 ndarray (不复制副本)
arange	类似 range
ones、ones_like	全 1 矩阵
zeros、zeros_like	全 0 矩阵
empty、empty_like	新 ndarray，只分配内存
eye	对角方阵

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

```
In [2]: # np.array  list
        np.array([1, 2, 3])
```

```
Out[2]: array([1, 2, 3])
```

```
In [3]: # np.array  tuple
        np.array((4, 5, 6))
```

```
Out[3]: array([4, 5, 6])
```

```
In [4]: # np.array np.asarray 区别
        arr1 = np.array([[1, 2, 3], [4, 5, 6]])
```

```
        arr2 = np.array(arr1)
        arr3 = np.asanyarray(arr1)
        arr1[1] = 0
        print('arr1:\n', arr1)
        print('arr2:\n', arr2)
        print('arr3:\n', arr3)
```

```
arr1:
 [[1 2 3]
 [0 0 0]]
arr2:
 [[1 2 3]
 [4 5 6]]
arr3:
 [[1 2 3]
 [0 0 0]]
```

```
In [5]: # np.arange
        list_range = [x for x in range(1, 10)]
        arr_arrange = np.arange(1, 10)
        print('list:\n', list_range, type(list_range))
        print('arr:\n', arr_arrange, type(arr_arrange))
```

```
list:
 [1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
arr:
 [1 2 3 4 5 6 7 8 9] <class 'numpy.ndarray'>
```

```
In [6]: # np.ones
        arr_ones = np.ones((2, 2))
        print('arr_ones:\n', arr_ones)
```

```
arr_ones:
 [[ 1.  1.]
 [ 1.  1.]]
```

```
In [7]: # np.ones_like
        arr_ones_likes = np.ones_like(arr1)
        print('arr1: \n', arr1)
        print('arr_ones_likes:\n', arr_ones_likes)
```

```
arr1:
 [[1 2 3]
 [0 0 0]]
arr_ones_likes:
 [[1 1 1]
 [1 1 1]]
```

```
In [8]: # np.zeros & np.zeros_like
        arr_zeros = np.zeros((3, 3))
        arr_zeros_like = np.zeros_like(arr1)
        print('arr_zeros:\n', arr_zeros)
        print('arr_zeros_like:\n', arr_zeros_like)
```

```
arr_zeros:
 [[ 0.  0.  0.]
 [ 0.  0.  0.]
 [ 0.  0.  0.]]
arr_zeros_like:
 [[0 0 0]
 [0 0 0]]
```

```
In [9]: # np.empty & np.empty_like
        arr_empty = np.empty((2, 2))
        arr_empty_like = np.empty_like(arr1)
        print('arr_empty:\n', arr_empty)
        print('arr_empty_like:\n', arr_empty_like)
```

```
arr_empty:
 [[ 0.  0.]
 [ 0.  0.]]
arr_empty_like:
 [[0 0 0]
 [0 0 0]]
```

```
In [10]: # np.eye
         arr_eye_3 = np.eye(3)
         arr_eye_5 = np.eye(5)
         print('arr_eye 3:\n', arr_eye_3)
         print('arr_eye 5:\n', arr_eye_5)
```

```
arr_eye 3:
 [[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
arr_eye 5:
 [[ 1.  0.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.]
 [ 0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0.  1.]]
```

索引 & 切片（查找）

```
In [11]: # 一维数组 索引 & 切片
         arr = np.arange(10)
         print('arr:\n', arr)
         print('数组第 4 个元素: ', arr[3])
         print('数组第 6 到第 8 个元素: ', arr[5:8])
         arr[3] = 0
         print('单独修改第三位元素为 0 :\n', arr)
         arr[5:8] = 11
         print('批量修改 6 至 8 位置元素为 11 :\n', arr)
```

```
arr:
 [0 1 2 3 4 5 6 7 8 9]
数组第 4 个元素:  3
数组第 6 到第 8 个元素:  [5 6 7]
单独修改第三位元素为 0 :
 [0 1 2 0 4 5 6 7 8 9]
批量修改 6 至 8 位置元素为 11 :
 [ 0  1  2  0  4 11 11 11  8  9]
```

特别注意 TIPS

将切片赋值给一个新变量，Numpy 并不会自动复制，如果修改，会直接修改源数据!!!

```
In [12]: origin_arr = np.arange(10)
         print('arr: \n', origin_arr)
         arr_slice = origin_arr[5:]
         print('arr_slice: \n', arr_slice)
         arr_slice[-1] = 1024
         print('arr_slice(after slice): \n', arr_slice)
         print('arr(after slice): \n', origin_arr)
```

```
arr:
 [0 1 2 3 4 5 6 7 8 9]
arr_slice:
 [5 6 7 8 9]
arr_slice(after slice):
 [   5    6    7    8 1024]
arr(after slice):
 [   0    1    2    3    4    5    6    7    8 1024]
```

```
In [13]: # 使用 .copy() 可复制新数据，修改不影响原来的数据
         origin_arr = np.arange(10)
         print('arr: \n', origin_arr)
         arr_slice = origin_arr[5:].copy()
         print('arr_slice: \n', arr_slice)
         arr_slice[-1] = 1024
         print('arr_slice(after slice): \n', arr_slice)
         print('arr(after slice): \n', origin_arr)
```

```
arr:
 [0 1 2 3 4 5 6 7 8 9]
arr_slice:
 [5 6 7 8 9]
arr_slice(after slice):
 [   5    6    7    8 1024]
arr(after slice):
 [0 1 2 3 4 5 6 7 8 9]
```

思考!?

为什么要这样设计？默认不会复制新的 ndarray？

```
In [14]: # 二维数组索引 & 切片
         arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         print('arr_2d: \n', arr_2d)
         print('选取第三行第三列: ', arr_2d[2, 2])
         print('选取中间一行: \n', arr_2d[1])
         print('选取最后一列: \n', arr_2d[:, 2])
```

```
arr_2d:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
选取第三行第三列:  9
选取中间一行:
 [4 5 6]
选取最后一列:
 [3 6 9]
```

思考!?

更高纬度 (eg. 3d 4d 5d...) 如何索引和切片？

```
In [15]: # 三维数组 感受一下上面思考的问题
         arr_3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
         print('arr_3d: \n', arr_3d)
         print('axis 0: \n', arr_3d[0])
         print('axis0 = 1, axis1 = 0, axis2 = 2: ',arr_3d[1][0][2])
```

```
arr_3d:
 [[[ 1  2  3]
  [ 4  5  6]]

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

```
axis0 = 1, axis1 = 0, axis2 = 2:  9
```

高级索引

```
In [16]: weekdays = np.array(['Mon', 'Tue', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun'])
         data = np.random.randn(7, 4)
         print('weekdays: \n', weekdays)
         print('data: \n', data)
```

```
weekdays:
 ['Mon' 'Tue' 'Wed' 'Thurs' 'Fri' 'Sat' 'Sun']
data:
 [[ 0.007933    1.03149299  0.13299665 -0.43991748]
 [-0.76542112 -0.81960019  0.60773258 -1.31736239]
 [ 0.94192462  0.40903701 -0.11436537 -0.71822014]
 [ 0.95016136  1.16260355 -1.20433439  0.1934954 ]
 [ 1.05855942  0.177071    0.81754917 -0.55449698]
 [ 0.69813971  0.72033553 -0.38544549 -1.83294227]
 [ 0.14395036  0.00438647  0.08261076 -0.68731269]]
```

```
In [17]: # data 是一组随机正态分布的数据， weekdays 是一组星期一到星期天的数据标签，假定每一天对应
         # 下面以上面模拟数据进行 高级索引 & 切片 演示

         # Bool 数组
         print('weekdays Bool Array: \n', weekdays == 'Wed')
         # 组合 Bool 条件 (Wed & Sat)
         wed_and_sat_bool_arr = (weekdays == 'Wed') | (weekdays == 'Sat')
         print('weekdays cond-& Array : \n', wed_and_sat_bool_arr)
         # 选取 Fri 对应的数据
         print('Fir - data: \n', data[weekdays=='Fri'])
         # 组合 Bool 条件 对应数据
         print('Wed & Sat - data : \n', data[wed_and_sat_bool_arr])
```

```
weekdays Bool Array:
 [False False  True False False False False]
weekdays cond-& Array :
 [False False  True False False  True False]
Fir - data:
 [[ 1.05855942  0.177071    0.81754917 -0.55449698]]
Wed & Sat - data :
 [[ 0.94192462  0.40903701 -0.11436537 -0.71822014]
 [ 0.69813971  0.72033553 -0.38544549 -1.83294227]]
```

```
In [18]: # 数值型逻辑
         print('data < 0: \n', data[data < 0])
         # 数值型逻辑 赋值 (eg. 数据清洗)
```

```
        data_clean = data.copy()
        data_clean[data_clean < 0] = 0
        print('data < 0 (=0): \n', data_clean)
        # 结合 weekdays 组合逻辑索引
        data[(weekdays == 'Wed') | (weekdays == 'Sat')] = 10
        print('Wed & Sat - data (=10) : \n', data)
```

```
data < 0:
 [-0.43991748 -0.76542112 -0.81960019 -1.31736239 -0.11436537 -0.71822014
 -1.20433439 -0.55449698 -0.38544549 -1.83294227 -0.68731269]
data < 0 (=0):
 [[ 0.007933    1.03149299  0.13299665  0.        ]
 [ 0.          0.          0.60773258  0.        ]
 [ 0.94192462  0.40903701  0.          0.        ]
 [ 0.95016136  1.16260355  0.          0.1934954 ]
 [ 1.05855942  0.177071    0.81754917  0.        ]
 [ 0.69813971  0.72033553  0.          0.        ]
 [ 0.14395036  0.00438647  0.08261076  0.        ]]
Wed & Sat - data (=10) :
 [[  7.93299692e-03   1.03149299e+00   1.32996653e-01  -4.39917484e-01]
 [ -7.65421116e-01  -8.19600186e-01   6.07732584e-01  -1.31736239e+00]
 [  1.00000000e+01   1.00000000e+01   1.00000000e+01   1.00000000e+01]
 [  9.50161361e-01   1.16260355e+00  -1.20433439e+00   1.93495395e-01]
 [  1.05855942e+00   1.77071000e-01   8.17549172e-01  -5.54496978e-01]
 [  1.00000000e+01   1.00000000e+01   1.00000000e+01   1.00000000e+01]
 [  1.43950357e-01   4.38646677e-03   8.26107621e-02  -6.87312686e-01]]
```

常用数学计算函数

以下函数对一个数组作用 (unary ufunc)

函数	说明
abs fasb	绝对值 fabs 更快一点
sqrt	平方根
square	平方
exp	指数 e^x
log log10 log2 log1p	对数分别对应底 (e、10、2、1+x)
sign	计算正负号
ceil	大于等于该值得最小整数
floor	小于等于该值得最大整数
rint	四舍五入整数
modf	小数、整数分为两个部分返回
isnan	判断是否空 (NaN) 返回 Bool 数组
isfinite isInf	判断无穷元素
cos cosh sin sinh tan tanh	三角函数
arccos arccosh arcsin arcsinh arctan arctanh	反三角函数
logical_not	计算真值

```
In [19]: arr = np.arange(10) - 10
         print('arr: \n', arr)

arr:
 [-10  -9  -8  -7  -6  -5  -4  -3  -2  -1]


In [20]: # np.abs()
         print('arr abs: \n', np.abs(arr))
         # np.sqrt()
         print('arr sqrt: \n', np.sqrt(arr+10))
         # np.square()
         print('arr square: \n', np.square(arr))
         # np.exp()
         print('arr exp: \n', np.exp(arr+10))
         # np.log()
         print('arr log: \n', np.log(np.exp(arr+10)))
         # np.sign()
         print('arr sign: \n', np.sign(arr+5))
         # np.ceil
         print('arr ceil: \n', np.ceil(arr+0.3))
         # np.floor
         print('arr floor: \n', np.floor(arr+0.3))
         # np.rint
         print('arr rint: \n', np.rint(arr+0.3))
         # np.modf
         print('arr modf: \n', np.modf(arr+0.3))

arr abs:
 [10  9  8  7  6  5  4  3  2  1]
arr sqrt:
 [ 0.          1.          1.41421356  1.73205081  2.          2.23606798
  2.44948974  2.64575131  2.82842712  3.        ]
arr square:
 [100  81  64  49  36  25  16   9   4   1]
arr exp:
 [  1.00000000e+00   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]
arr log:
 [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
arr sign:
 [-1 -1 -1 -1 -1  0  1  1  1  1]
arr ceil:
 [-9. -8. -7. -6. -5. -4. -3. -2. -1. -0.]
arr floor:
 [-10.  -9.  -8.  -7.  -6.  -5.  -4.  -3.  -2.  -1.]
arr rint:
```

```
 [-10.  -9.  -8.  -7.  -6.  -5.  -4.  -3.  -2.  -1.]
arr modf:
 (array([-0.7, -0.7, -0.7, -0.7, -0.7, -0.7, -0.7, -0.7, -0.7, -0.7]), array([-9., -8., -7., -6
```

```
In [21]: arr_float = arr.astype(np.float)
         arr_float[[0, 2, 4]] = np.NaN
         arr_float[-1] = np.Inf
         print('arr with Nan: \n', arr_float)

arr with Nan:
 [ nan  -9.  nan  -7.  nan  -5.  -4.  -3.  -2.  inf]
```

```
In [22]: # np.isnan()
         print('arr nan: \n', np.isnan(arr_float))
         # np.isinf()
         print('arr inf: \n', np.isinf(arr_float))

arr nan:
 [ True False  True False  True False False False False False]
arr inf:
 [False False False False False False False False False  True]
```

以下函数对两个数组作用 (binary ufunc)

函数	说明
add	元素相加
subtract	元素相减
multply	元素相乘
divide floor_divide	除法
power	A^B
maximum fmax	最大值 fmax 忽略 NaN
minimum fmin	最小值 fmin 忽略 NaN
mod	求余数
copysign	将第二个数组的符号复制给第一个数组的值
greater greater_equal less less_equal equal not_equal	> >= < <= == !=
logical_and logical_or logical_xor	& | ^

```
In [23]: # 创建两个随机数组
         x = np.random.randn(10)
         y = np.random.randn(10)
         print('x: \n', x)
         print('y: \n', y)

x:
 [ 0.09836809 -1.12265088  0.33041685  0.26390636  0.39234656  2.3312102
```

```
 -0.64102884 -0.01100052 -0.81015187  0.2194367 ]
y:
 [-0.29661387  1.09419825  0.3371075   1.09272952  0.40148001 -2.09319417
 -0.51035631  0.12446398 -0.04842607 -2.95193171]
```

```
In [24]: # np.add()
         print('add: \n', np.add(x, y))
         # np.subtract()
         print('subtract: \n', np.subtract(x, y))
         # np.multiply()
         print('multiply: \n', np.multiply(x, y))
         # np.divide
         print('divide: \n', np.divide(x, y))
         # np.power Ps: 避免无法计算, x 去绝对值
         print('power: \n', np.power(np.abs(x), y))
         # np.maximum
         print('maximum: \n', np.maximum(x, y))
         # np.minimum
         print('minimum: \n', np.minimum(x, y))
         # np.mod
         print('mod: \n', np.mod(x, y))
         # np.copysign
         print('copysign: \n', np.copysign(x, y))
         # np.greater
         print('>: \n', np.greater(x, y))
         # np.less
         print('<: \n', np.less(x, y))
         # np.logical_and
         print('&: \n', np.logical_and(np.greater(x, y), np.less(x, y)))
```

```
add:
 [-0.19824578 -0.02845262  0.66752435  1.35663588  0.79382657  0.23801603
 -1.15138516  0.11346346 -0.85857794 -2.73249502]
subtract:
 [ 0.39498196 -2.21684913 -0.00669065 -0.82882316 -0.00913345  4.42440437
 -0.13067253 -0.13546449 -0.7617258   3.17136841]
multiply:
 [ -2.91773390e-02  -1.22840263e+00   1.11385997e-01   2.88378273e-01
   1.57519299e-01  -4.87967561e+00   3.27153116e-01  -1.36916807e-03
   3.92324716e-02  -6.47762148e-01]
divide:
 [ -0.33163684  -1.02600317   0.98015278   0.24151115   0.97725054
  -1.11370948   1.25604176  -0.08838314  16.7296635   -0.07433664]
power:
 [  1.98945172   1.1349525    0.68844977   0.23323901   0.68685738
   0.17005158   1.25476169   0.57046169   1.01024746  87.98529319]
maximum:
```

```
 [ 0.09836809  1.09419825  0.3371075   1.09272952  0.40148001  2.3312102
 -0.51035631  0.12446398 -0.04842607  0.2194367 ]
minimum:
 [-0.29661387 -1.12265088  0.33041685  0.26390636  0.39234656 -2.09319417
 -0.64102884 -0.01100052 -0.81015187 -2.95193171]
mod:
 [-0.19824578  1.06574563  0.33041685  0.26390636  0.39234656 -1.85517815
 -0.13067253  0.11346346 -0.03533474 -2.73249502]
copysign:
 [-0.09836809  1.12265088  0.33041685  0.26390636  0.39234656 -2.3312102
 -0.64102884  0.01100052 -0.81015187 -0.2194367 ]
>:
 [ True False False False False  True False False False  True]
<:
 [False  True  True  True  True False  True  True  True False]
&:
 [False False False False False False False False False False]
```

统计

函数	说明
sum	求和
mean	算术平均数
std var	标准差方差
min max	最大值最小值
argmin argmax	最大值最小值索引
cumsum	累计和
cumprod	累计积

```
In [25]: data_arr = np.random.randn(20)
         print('data_arr: \n', data_arr)
```

```
data_arr:
 [ 0.05971164 -0.36858479  2.20454677 -0.31501527 -0.61352051 -1.55790321
  1.21193682  0.72389227 -0.01436951 -2.03380081  0.89729443 -0.17276275
  0.40233089  0.42856698  0.17459529 -0.6379456   0.88294305 -0.17196833
 -1.42163028  1.35401547]
```

```
In [26]: # sum
         print('sum: ', data_arr.sum())
         print('mean: ', data_arr.mean())
         print('std: ', data_arr.std())
         print('var: ', data_arr.var())
         print('max: ', data_arr.max())
         print('min: ', data_arr.min())
         print('argmin: ', data_arr.argmin())
```

```
        print('argmax: ', data_arr.argmax())
        print('cumsum: \n', data_arr.cumsum())
        print('cumprod: \n', data_arr.cumprod())
```

```
sum:  1.03233254968
mean:  0.0516166274842
std:  1.0064731223
var:  1.01298814591
max:  2.2045467652
min:  -2.03380080752
argmin:  9
argmax:  2
cumsum:
 [ 0.05971164 -0.30887316  1.89567361  1.58065834  0.96713783 -0.59076538
  0.62117144  1.34506371  1.3306942  -0.7031066   0.19418783  0.02142508
  0.42375597  0.85232294  1.02691824  0.38897264  1.27191569  1.09994736
 -0.32168292  1.03233255]
cumprod:
 [  5.97116360e-02  -2.20088009e-02  -4.85194309e-02   1.52843617e-02
  -9.37726937e-03   1.46088780e-02   1.77050373e-02   1.28165396e-02
  -1.84167368e-04   3.74559741e-04   3.36090370e-04  -5.80638963e-05
  -2.33608991e-05  -1.00117099e-05  -1.74799739e-06   1.11512724e-06
   9.84593852e-07  -1.69318959e-07   2.40708958e-07   3.25923653e-07]
```

```
In [27]: # Tips: 找出大于 0 的个数
        print('>0 count: ', (data_arr > 0).sum())
        # 排序
        data_arr.sort()
        print('sort: \n', data_arr)
```

```
>0 count:  10
sort:
 [-2.03380081 -1.55790321 -1.42163028 -0.6379456  -0.61352051 -0.36858479
 -0.31501527 -0.17276275 -0.17196833 -0.01436951  0.05971164  0.17459529
  0.40233089  0.42856698  0.72389227  0.88294305  0.89729443  1.21193682
  1.35401547  2.20454677]
```

集合

函数	说明
unique(x)	返回唯一有序结果
intersect1d(x, y)	返回公共元素
union1d(x, y)	并集
in1d(x, y)	x 是否包含于 y
setdiff1d(x, y)	差集在 x 不在 y
setxor1d(x, y)	对称差存在一个数组但不同时存在两个数组

```
In [28]: # 创建两个数组
         nums_1 = np.array([1, 0, 0, 3, 2, 5])
         nums_2 = np.array([4, 9, 0, 3, 2, 6])
         print('nums 1 : \n', nums_1)
         print('nums 2 : \n', nums_2)

nums 1 :
 [1 0 0 3 2 5]
nums 2 :
 [4 9 0 3 2 6]
```

```
In [29]: # unique
         print('unique: \n', np.unique(nums_1))
         # intersect1d
         print('intersect1d: \n', np.intersect1d(nums_1, nums_2))
         # union1d
         print('union1d: \n', np.union1d(nums_1, nums_2))
         # in1d
         print('in1d: \n', np.in1d(nums_1, nums_2))
         # setdiff1d
         print('setdiff1d: \n', np.setdiff1d(nums_1, nums_2))
         # setxor1d
         print('setxor1d: \n', np.setxor1d(nums_1, nums_2))

unique:
 [0 1 2 3 5]
intersect1d:
 [0 2 3]
union1d:
 [0 1 2 3 4 5 6 9]
in1d:
 [False  True  True  True  True False]
setdiff1d:
 [1 5]
setxor1d:
 [1 4 5 6 9]
```

随机函数

函数	说明
seed	设置随机数生成种子
permutation	返回序列随机排列
shuffle	对序列随机排列
rand	均匀分布样本生成
randint	给定上下范围随机整数
randn	生成正态分布 ($\mu = 0, \sigma = 1$)

函数	说明
binomial	生成二项分布
normal	生成高斯分布样本
beta	生成 Beta 分布样本
chisquare	生成卡方分布样本
gamma	生成 Gamma 分布样本
uniform	生成 [0, 1) 均匀分布

```
In [30]: # seed 设置 seed 后，每次随机一样
         i = 0
         while(i < 5):
             np.random.seed(3)
             print(np.random.random())
             i+=1
```

```
0.5507979025745755
0.5507979025745755
0.5507979025745755
0.5507979025745755
0.5507979025745755
```

```
In [46]: # permutation
         some_values = np.random.permutation(10)
         print('permutation: \n', some_values)
         # shuffle
         np.random.shuffle(some_values)
         print('shuffle: \n', some_values)
         # rand
         print('rand: \n', np.random.rand(10))
         # randint
         print('randint: \n', np.random.randint(-10, 10, 10))
         # randn
         print('randn: \n', np.random.randn(10))
         # binomial (模拟抛硬币)
         print('binonmial: \n', np.random.binomial(1, 0.5, size=10))
         # normal mean =5 , std = 2
         print('normal: \n', np.random.normal(5, 2, size=10))
         # beta
         print('beta: \n', np.random.beta(1, 5, 10))
         # chisquare
         print('chisquare: \n', np.random.chisquare(2, 4))
         # gamma
         print('gamma: \n', np.random.gamma(2, 2, 10))
         # uniform
         print('uniform: \n', np.random.uniform(size=10))
```

```
permutation:
 [0 1 6 8 2 9 4 7 5 3]
```

```
shuffle:
 [5 1 7 4 8 6 2 9 3 0]
rand:
 [ 0.76417583  0.603321    0.35805149  0.45451153  0.67056627  0.37636471
  0.06472975  0.70592838  0.71051248  0.28455966]
randint:
 [-2 -3 -6 -7 -7 -2 -5 -8 -9  0]
randn:
 [ 0.53506478 -1.10629888  2.56855154 -0.69984367 -0.74851361 -1.09191704
  0.01313595 -0.27315237  0.46102057 -0.31553938]
binonmial:
 [0 1 0 1 1 0 1 1 0 1]
normal:
 [  4.30118074   7.85123151  10.70513844   4.83277858   5.28996051
   3.43478947   5.223257     2.41342652   3.56571969   6.10955394]
beta:
 [ 0.00173355  0.17882975  0.08370215  0.25266827  0.02606328  0.14744944
  0.19149676  0.368885    0.29722945  0.25401569]
chisquare:
 [ 0.13948117  4.91411402  0.10756093  3.33832593]
gamma:
 [  0.83560505   1.3464553    5.68831766   4.40994569   5.51855504
  11.97568781   0.94295759   4.41429685   9.2843684    1.02813234]
uniform:
 [ 0.28734346  0.71863768  0.23602865  0.59537643  0.31699907  0.04258288
  0.02150672  0.30158334  0.46327813  0.44023079]
```