数据科学导论第四周自学要点

1. 学习内容：NumPy基础

2. 参考资料：

- 教材：《利用Python进行数据分析（第二版）》第4章
- 扩展阅读：https://docs.scipy.org/doc/numpy/user/quickstart.html (https://docs.scipy.org/doc/numpy/user/quickstart.html)

3. 学习要点

- NumPy核心数据结构 `ndarray` 的基本概念
  - 注意 `ndarray` 与Python内置数据类型 `list` 的区别
- `ndarray` 的创建
  - 相关函数： `np.array()` 、 `np.ones()` 、 `np.zeros()` 、 `np.empty()` 等，更多参考教材表4-1
  - 相关属性： `ndim` 、 `shape` 、 `dtype` 等
- `ndarray` 的类型与类型转换
  - 数据类型描述：教材表4-2
  - 类型转换函数： `astype()` ，应特别注意转换过程中的数据截断问题（浮点数与整数之间、字符串与数字之间）
- `ndarray` 的代数运算
  - 采用与标量元素类似的方式完成多维数组的元素（element-wise）操作
  - 与传统Python代码的区别：通过向量化（vectorization）的方式避免了 `for` 循环
  - 更多内容参考扩展阅读的"Basic Operations"章节
- 数据基本的索引和切片
  - 与Python列表的重要区别：数据切片是原数据的一个视图view，而非拷贝copy对于切片上的任何操作都会反映在原数据上。在实际的编程实践中，非常容易在这一点上出错。如果希望数据切片是原数据上的一份拷贝，应显式地调用 `copy()` 函数。有关view和copy的区别可以参考扩展阅读的”Copies and Views“章节。下面考虑一个例子：

In [1]:
```
import numpy as np
arr = np.arange(10)
print(arr)
arr_slice = arr[5:8] # 选取一个切片，注意该切片为原数据的视图
arr_slice[0] = 1234
print(arr) # 原数据索引为5的值发生了改变
arr_slice1 = arr[1:3].copy() # 拷贝出一个切片
arr_slice1[0] = 1234
print(arr) # 原数据索引为1的值没有发生了改变
```

```
[0 1 2 3 4 5 6 7 8 9]
[   0    1    2    3    4 1234    6    7    8    9]
[   0    1    2    3    4 1234    6    7    8    9]
```

- 数据切片的方式1：与Python的列表类型类似，可以通过索引进行切片，教材图4-2给出了一个直观的例子
- 数据切片的方式2：布尔型索引，即通过布尔型数组对数据进行切片，详见下例：

In [2]:
```
names = np.array(['Bob','Joe','Bob','Alice'])
data = np.random.randn(4,5)
print(names=='Bob') # names=='Bob'会返回一个布尔型的数组
print(data[names == 'Bob']) # 通过布尔型的数组进行切片
print(names!='Bob') # names!='Bob'会返回一个布尔型的数组
print(data[names != 'Bob']) # 通过布尔型的数组进行切片
print((names=='Bob')|(names=='Alice')) # 可以考虑多个条件
print(data[(names=='Bob')|(names=='Alice')]) # 通过布尔型的数组进行切片

data[data<0] = 0 # 通过布尔型切片选出所有的负数，统一置为0
print(data)
```

```
[ True False  True False]
[[ 1.63691799 -0.53269813  0.01980702  0.13260588 -2.36245474]
 [-0.01958163  0.10687501  0.16590426 -3.24077582 -0.49340068]]
[False  True False  True]
[[-1.53338532 -0.77079493  2.16274676  0.89637885  0.32211041]
 [-0.48750183  2.67493798  1.50098849 -0.06362239  1.15835726]]
[ True False  True  True]
[[ 1.63691799 -0.53269813  0.01980702  0.13260588 -2.36245474]
 [-0.01958163  0.10687501  0.16590426 -3.24077582 -0.49340068]
 [-0.48750183  2.67493798  1.50098849 -0.06362239  1.15835726]]
[[1.63691799 0.         0.01980702 0.13260588 0.        ]
 [0.         0.         2.16274676 0.89637885 0.32211041]
 [0.         0.10687501 0.16590426 0.         0.        ]
 [0.         2.67493798 1.50098849 0.         1.15835726]]
```

- 花式索引（Fancy Indexing）:直接传入整数数组进行切片
- 转置、坐标轴交换与改变 ndarray 的shape
  - 转置操作 T
  - reshape() 函数经常被用到，注意与前面的slice类似，它也返回一个view而非copy
  - 注意 reshape() 函数与 resize() 函数的区别

In [3]:
```
arr = np.random.rand(12)
print('arr', arr)
arr1 = arr.reshape(3,4)
print('arr1',arr1)
arr1[2,3] = 1234
print('arr', arr) # 原数组也会修改
arr.resize(6,2)
print('arr', arr)
```

```
arr [0.22376846 0.67442031 0.366921   0.22542734 0.08602333 0.4000
3216
 0.7887416  0.69232153 0.72743535 0.85500591 0.37853215 0.44184388
]
arr1 [[0.22376846 0.67442031 0.366921   0.22542734]
 [0.08602333 0.40003216 0.7887416  0.69232153]
 [0.72743535 0.85500591 0.37853215 0.44184388]]
arr [2.23768456e-01 6.74420309e-01 3.66921001e-01 2.25427342e-01
 8.60233330e-02 4.00032160e-01 7.88741602e-01 6.92321529e-01
 7.27435350e-01 8.55005906e-01 3.78532149e-01 1.23400000e+03]
arr [[2.23768456e-01 6.74420309e-01]
 [3.66921001e-01 2.25427342e-01]
 [8.60233330e-02 4.00032160e-01]
 [7.88741602e-01 6.92321529e-01]
 [7.27435350e-01 8.55005906e-01]
 [3.78532149e-01 1.23400000e+03]]
```

- 通用函数（ufunc）：对 ndarray 进行元素级操作的函数，调用方式上与标量数据上函数的方式类似，可以简单理解为对数组元素执行批量操作
  - 一元函数与二元函数列表分别参考教材表4-3和表4-4
- 面向数组编程（Array-Oriented Programming）：编程思路的转变
  - 将逻辑操作转换为数组运算：`np.where(cond, x, y)` 函数
  - 基于 ndarray 计算统计量，详细的函数参考教材表4-5
  - 布尔型数组附加的函数 `any()` 和 `all()`
  - ndarray 中元素的排序：`sort()` 与 `np.sort()`。应注意：与数据切片不同，`np.sort()` 返回原数据的一份拷贝，而非视图，详见下例：

In [4]:
```
arr = np.random.randn(10)
print(arr)
arr_sorted = np.sort(arr)
arr_sorted[0]=1234
print(arr_sorted)
print(arr)
```

```
[-0.79310838 -1.04911862 -1.75487273  0.05014495 -0.20053644  0.39
326256
 -2.75371776  0.19398191 -0.69449308 -0.92242645]
[ 1.23400000e+03 -1.75487273e+00 -1.04911862e+00 -9.22426453e-01
 -7.93108382e-01 -6.94493079e-01 -2.00536437e-01  5.01449547e-02
  1.93981907e-01  3.93262562e-01]
[-0.79310838 -1.04911862 -1.75487273  0.05014495 -0.20053644  0.39
326256
 -2.75371776  0.19398191 -0.69449308 -0.92242645]
```

- 集合操作： `unique()` 、 `intersect1d()` 、 `union1d()` ，详细参考教材表4-6
  - `ndarray` 的文件操作，掌握函数 `save()` 、 `savez()` 和 `load()`
  - 矩阵运算函数参考教材表4-7
  - 生成由随机数构成的 `ndarray` ，相关函数在机器学习参数初始化的过程中被广泛应用

4. 课后练习

- **第1题：创建一个8*8的二维ndarray 对象 `A` （即为一个矩阵）。要求： `A` 为上三角矩阵，矩阵每一行的非0元素形成一步长为2的递增等差数列，且对角线上元素从上到下为1, 19, 37, 55, 73, 91, 109, 127。**

In [5]:
```
# 参考答案如下
import numpy as np
a = np.arange(1, 129, 2).reshape(8, 8)
A = np.triu(a)
print(A)
```

```
[[  1   3   5   7   9  11  13  15]
 [  0  19  21  23  25  27  29  31]
 [  0   0  37  39  41  43  45  47]
 [  0   0   0  55  57  59  61  63]
 [  0   0   0   0  73  75  77  79]
 [  0   0   0   0   0  91  93  95]
 [  0   0   0   0   0   0 109 111]
 [  0   0   0   0   0   0   0 127]]
```

- **第2题：针对下面给出的89的 *ndarray 对象 `P` 做如下操作：首先，对 `P` 右上角*88的子矩阵 `Q` 的第1行的第1个元素赋值为3；然后，将该子矩阵 `Q` 的各行根据最后一列的元素从小到大进行排序；最后，将矩阵 `Q` 与矩阵 `P` 左上角8*8的子矩阵相加得到矩阵 `B` ，并打印出来。**

In [6]:
```
P = np.array([[3,6,9,11,5,2,6,99,1],
              [2,4,8,10,1,3,5,7,65],
              [1,5,7,9,2,4,6,8,10],
              [4,5,3,1,7,45,7,2,0],
              [32,5,4,8,1,7,2,0,23],
              [5,3,8,1,4,33,62,4,9],
              [3,6,7,1,9,47,39,1,2],
              [3,5,7,2,5,9,1,1,5]])
P
```

Out[6]:
```
array([[ 3,  6,  9, 11,  5,  2,  6, 99,  1],
       [ 2,  4,  8, 10,  1,  3,  5,  7, 65],
       [ 1,  5,  7,  9,  2,  4,  6,  8, 10],
       [ 4,  5,  3,  1,  7, 45,  7,  2,  0],
       [32,  5,  4,  8,  1,  7,  2,  0, 23],
       [ 5,  3,  8,  1,  4, 33, 62,  4,  9],
       [ 3,  6,  7,  1,  9, 47, 39,  1,  2],
       [ 3,  5,  7,  2,  5,  9,  1,  1,  5]])
```

In [7]:
```
# 参考答案
Q = np.copy(P) # 这里设计一个“坑”：如果不用copy，而是直接切片赋值，会改变P的值，从而在最后加法中出错
Q = Q[:,1:]  # 取P右上角8*8的子矩阵
Q[0, 0] = 3  # 第一行第一个元素赋值为3
Q = Q[np.argsort(Q[:,-1])]  # 将该子矩阵的行根据最后一列进行排序
B = Q + P[:,:-1]  # 与矩阵P左上角8*8的子矩阵相加
print(B)
```

```
[[  8   9  10  18  50   9   8  99]
 [  5  13  19  15   3   9 104   8]
 [  7  12   8  18  49  43   7  10]
 [  9  12   5   6  16  46   8   7]
 [ 35  13   5  12  34  69   6   9]
 [ 10  10  17   3   8  39  70  14]
 [  8  10  15   2  16  49  39  24]
 [  7  13  17   3   8  14   8  66]]
```

- **第3题：求上面得到的矩阵 A 和矩阵 B 的积，得到矩阵 C ，请打印出矩阵 C 。然后，对矩阵C的每一列做正则化，得到矩阵 D 并打印出来。**
  - 正则的概念：假设a是数组中的一个元素，max/min分别是数组元素的最大最小值，则正则化后a = (a - min)/(max - min)） 。

In [8]:
```
# 参考答案如下
C = np.dot(A, B)
print(C)
cmax = C.max(axis=0)
cmin = C.min(axis=0)
delta = cmax - cmin

D = np.array([(line - cmin)/delta for line in C])
print(D)

# 这里的坑是如果D是从C copy过来的，赋值的时候由于类型是整型int，所以小数点会被截断成整数，就像下面这种情况
#D = np.copy(C)
#for i in range(8):
#    D[i, :] = (D[i, :] - cmin)/delta
#print(D)
```

```
[[  755   744   824   407  1138  2470  1862  1759]
 [ 2043  2063  2190  1333  3232  6765  5726  3868]
 [ 3164  2936  2901  1752  5271 10754  5958  5796]
 [ 4009  3420  3549  1502  4770 12635  7795  7346]
 [ 4474  3496  4138  1492  4946 12841  9323  8769]
 [ 2319  3075  4557   744  2976  9436 10757  9776]
 [ 1649  2533  3522   551  2632  6895  5139  9942]
 [  889  1651  2159   381  1016  1778  1016  8382]]
[[0.         0.         0.         0.01896426 0.02867215 0.0625508
5
  0.0868494  0.        ]
 [0.34632966 0.47928779 0.36592553 0.69438366 0.52079906 0.4507818
9
  0.48352325 0.25772944]
 [0.64775477 0.79651163 0.55638896 1.         1.         0.8113531
6
  0.50734011 0.49333985]
 [0.87496639 0.97238372 0.72997589 0.81765135 0.88225617 0.9813793
7
  0.69592444 0.68275694]
 [1.         1.         0.88775784 0.8103574  0.92361927 1.
  0.85278719 0.85665404]
 [0.42054316 0.84702035 1.         0.26477024 0.46063455 0.6922173
  1.         0.97971404]
 [0.2403872  0.65007267 0.7227431  0.12399708 0.37978848 0.4625327
7
  0.4232625  1.        ]
 [0.03603119 0.32957849 0.35762122 0.         0.         0.
  0.         0.80936087]]
```

- **第4题：对于下列ndarray对象 `students` ，每一列的含义分别是：名，姓，年龄，入学日期。请打印出姓 `Smith` 并且年龄大于 `14` 岁的同学的入学日期。**

In [9]:
```
data = np.array([['Bob', 'Smith', '12', '2019-02-01'],
                 ['Joe', 'Lawrence', '13', '2018-03-07'],
                 ['Roy', 'Ratner', '12', '2019-02-05'],
                 ['Rita', 'Smith', '14', '2017-02-16'],
                 ['Alice', 'Holmes', '11', '2020-02-29'],
                 ['Wool', 'Smith', '15', '2016-02-14']])
data
```

Out[9]:
```
array([['Bob', 'Smith', '12', '2019-02-01'],
       ['Joe', 'Lawrence', '13', '2018-03-07'],
       ['Roy', 'Ratner', '12', '2019-02-05'],
       ['Rita', 'Smith', '14', '2017-02-16'],
       ['Alice', 'Holmes', '11', '2020-02-29'],
       ['Wool', 'Smith', '15', '2016-02-14']], dtype='<U10')
```

In [10]:
```
# 参考答案
print(data[(data[:,1]=='Smith') & (data[:,2]>'14')][0,-1])
```

```
2016-02-14
```

- **第5题：利用numpy数组实现Game of Life。用一个15*15的矩阵表示225个细胞：元素取值为 1 表示活细胞，取值为 0 表示死细胞。Game of Life的过程如下：初始矩阵为 κ (如下所示)，表示细胞的初始状态；进而按照以下规则进行迭代，更新细胞状态。请你打印出迭代 100 次之后得到的矩阵**
  - 每个细胞死或活的状态由它周围的八个细胞，称为**邻居**，所决定。
  - “人口过少”：任何活细胞如果活邻居少于2个，则死掉。
  - “正常”：任何活细胞如果活邻居为2个或3个，则继续活。
  - “人口过多”：任何活细胞如果活邻居大于3个，则死掉。
  - “繁殖”：任何死细胞如果活邻居正好是3个，则活过来。

In [11]:
```
K = np.array([[0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1],
              [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
              [0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0],
              [0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
              [1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0],
              [1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0],
              [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
              [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
              [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0],
              [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0],
              [0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1],
              [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1],
              [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1],
              [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0],
              [1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0]])
K
```

Out[11]:
```
array([[0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1],
       [1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0],
       [0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0],
       [0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
       [1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0],
       [1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0],
       [1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0],
       [0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1],
       [0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1],
       [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0],
       [1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0]])
```

In [12]:

```python
# 参考解答如下
# Author: Nicolas Rougier
def iterate(Z):
    # Count neighbours
    N = (Z[0:-2,0:-2] + Z[0:-2,1:-1] + Z[0:-2,2:] +
         Z[1:-1,0:-2]                + Z[1:-1,2:] +
         Z[2:  ,0:-2] + Z[2:  ,1:-1] + Z[2:  ,2:])

    # Apply rules
    birth = (N==3) & (Z[1:-1,1:-1]==0)
    survive = ((N==2) | (N==3)) & (Z[1:-1,1:-1]==1)
    Z[...] = 0
    Z[1:-1,1:-1][birth | survive] = 1
    return Z

# Z = np.random.randint(0,2,(15,15))
for i in range(100):
    K = iterate(K)
print(K)
```

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

In [ ]: