数据分析DAY01

徐铭 15201603213 xuming@tedu.cn

什么是数据分析

数据分析是指用适当的统计分析方法对收集来的大量数据进行分析, 提取有用的信息形成结论,并对数据加以详细研究和概括总结的过程.

使用pyhon做数据分析的常用库

1. numpy 处理基础数值算法
2. scipy 处理科学计算
3. matplotlib 实现数据可视化
4. pandas 提供了序列高级函数

Numpy概述

1. Numerical Python(数值的python),补充了python语言欠缺的数值运算能力.

2. Numpy是其他数据分析和机器学习库的底层库.
3. Numpy完全标准C语言实现, 运行效率充分优化.
4. Numpy开源免费.

Numpy的历史

1. 1995年, Numeric, 用于完成python语言数值运算的扩充.
2. 2001年, Scipy -> Numarray, 用于完成多维数组运算.
3. 2005年, Numeric + Numarray -> Numpy
4. 2006年, Numpy脱离了Scipy成为一个独立项目.

Numpy基础

Numpy的核心: ndarray对象

使用numpy.ndarray对象表示一个数组

```
import numpy as np

ary = np.array([1, 2, 3, 4, 5])
print(ary)
ary = ary * 10
print(ary)
```

内存中的ndarray对象

元数据(metadata)

存储对目标数组的描述信息,如 dim count, dimensions, dtype, data等.

实际数据

完整的数组数据. (Ndarray数组是同质数组, 所有元素类型相同)

将实际数据与元数据分开存放,一方面提高了内存空间的使用效率,另一方面减少了对实际数据的访问频率,提高性能.

ndarray数组对象的创建

np.array(可以被解释为Numpy数组的序列)

np.arange(起始值(0), 终止值, 步长(1))

np.zeros(数组元素的个数, dtype='数组元素类型')

np.ones(数组元素的个数, dtype='数组元素类型')

案例:测试ndarray对象的创建

```
import numpy as np
# 创建二维数组
a = np.array([[1, 2, 3, 4], [5, 6, 7,
8]])
print(a)
# np.arange(起始值, 结束值, 步长)
b = np.arange(1, 10, 1)
print(b)
# np.zeros(数组元素个数, dtype='')
c = np.zeros(10)
print(c, '; c.dtype:', c.dtype)
# np.ones(数组元素个数, dtype='')
d = np.ones(10, dtype='int64')
print(d, '; d.dtype:', d.dtype)
```

ndarray对象属性的基本操作

数组的维度: array.shape

元素的类型: array.dtype

数组元素的个数: array.size

数组的索引(下标): array[0]

案例:测试数组的基本属性

```
import numpy as np

a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
# 测试数组的基本属性
print('a.shape:', a.shape)
# a.shape = (6, )
# print(a, 'a.shape:', a.shape)
print('a.size:', a.size)
print('len(a):', len(a))
# 数组元素的索引
```

```
ary = np.arange(1, 28)
ary.shape = (3, 3, 3)
print(ary, '; ary.shape:', ary.shape)
print('ary[0]:', ary[0])
print('ary[0][0]:', ary[0][0])
print('ary[0][0][0]:', ary[0][0][0])
print('ary[0,0,0]:', ary[0, 0, 0])
# 遍历三维数组
for i in range(ary.shape[0]):
    for j in range(ary.shape[1]):
        for k in range(ary.shape[2]):
            print(ary[i, j, k], end='
')
```

ndarray对象属性操作详解

Numpy内置的基本数据类型

类型名	类型表示符
布尔型	bool_
有符号整型	int8(-128~127) / int16 / int32 / int64
无符号整型	uint8(0~255) / uint16 / uint32 / uint64
浮点型	float16 / float32 / float64
复数型	complex64 / complex128
字符串型	str_ (每个字符串用32位的Unicode编码表示)

numpy自定义复合类型

案例:在ndarray数组中存储3位学生信息

```
import numpy as np

data = [('zs', [10, 15, 2], 3),
        ('ls', [12, 12, 92], 8),
        ('ww', [14, 35, 82], 13)]
```

```
# 第一种设置dtype的方式
a = np.array(data,
                dtype='U2, 3int32,
int32')
print(a, '; zs.age:', a[0]['f2'])

# 第二种设置dtype的方式
b = np.array(data, dtype=[
    ('name', 'str_', 2),
    ('scores', 'int32', 3),
    ('age', 'int32', 1)])
print(b, '; ww.age:', b[2]['age'])

# 第三种设置dtype的方式
c = np.array(data, dtype={
    'names': ['name', 'scores',
'age'],
    'formats': ['U2', '3int32',
'int32']})
print(c, '; ls.name:', c[1]['name'])

# 第四种设置dtype的方式
# 例如scores字段在存储时将会从第16个字节
# 开始输出分数列表数据,3int32将会占用12
```

```
# 字节,那么age字段将会从第28个字节开始
# 向后输出.
# U2占用了8字节, 与scores字段中间将会
# 空出8个字节.虽然浪费了空间,但是这种数据
# 存储对齐的做法在数据访问时将提高效率.
d = np.array(data, dtype={
    'name': ('U2', 0),
    'scores': ('3int32', 16),
    'age': ('int32', 28)})

# 第五种设置dtype的方式
e = np.array([0x1234, 0x5678],
             dtype=('u2',
                    {'lowc': ('u1',
0),
                     'highc': ('u1',
1)}))
print('%x' % e[0])  # 1234
print('%x' % e['lowc'][0])   # 34
print('%x' % e['highc'][0])  # 12

# ndarray对象处理日期类型元素
f = np.array(['2018', '2019-01-01',
              '2019-02-01',
```

```
                    '2019-01-02 01:01:01'])
# 把f数组的元素类型改为日期类型
g = f.astype('M8[D]')
print(g, '; g.dtype:', g.dtype)

h = g.astype('int32')
print(h)
print(h[2] - h[1])
```

类型的简写字符码

类型	字符码
bool_	?
int8 / int16 / int32 / int64	i1 / i2 / i4 / i8
uint8 / uint16 / uint32 / uint64	u1 / u2 / u4 / u8
float16 / float32 / float64	f2 / f4 / f8
complex64 / complex128	c8 / c16
str_	U<字符数>
datetime64	M8[Y]/[M]/[D]/[h]/[m]/[s]

ndarray数组对象的维度操作

视图变维(数据共享): array.reshape() array.ravel()

```
a = np.arange(1, 9)
# 视图变维使用的还是原始数组中的数据,如果修改了原始数组中的数据,那么新数组读到的数据也会发生变化.
b = a.reshape((2, 4))
print(a, b)
a[0] = 999
print(b)
c = b.ravel()
print(c)
```

复制变维(数据独立): flatten()

```
# 测试flatten()方法
d = b.flatten()
d[0] = 110
print(b)
print(d)
```

就地变维: 直接修改数组维度, 不返回新数组. a.shape a.resize()

```
# 就地变维
d.shape = (2, 4)
print(d)
d.resize(2, 2, 2)
print(d)
```

ndarray数组的切片操作

```
# 数组的切片与列表切片参数类似
# 步长+: 默认从前往后切
# 步长-: 默认从后往前切
array[起始位置:终止位置:步长]
```

```
import numpy as np
a = np.arange(1, 10)
print(a)
print(a[:3])
print(a[3:6])
print(a[6:])
print(a[::-1])
print(a[:-4:-1])
print(a[-4:-7:-1])
print(a[-7::-1])
print(a[::])
```

```
print(a[::3])
print(a[1::3])
print(a[2::3])
# a改为2维数组
a.resize(3, 3)
# 切出1 / 2行与0 / 1 / 2列
print(a[1:, :])
# 切出1 / 2行与1 / 2列
print(a[1:, 1:])
```

ndarray数组的掩码操作

```
import numpy as np

a = np.array([1, 2, 3, 4, 5, 6, 7, 8])
f = np.array([True, False, True,
False,
              True, False, True,
False])
print(a[f])
print(a[a > 3])

# 把1~100中3的倍数或7的倍数都打印出来
a = np.arange(1, 100)
```

```
flag_a = a % 3 == 0
flag_b = a % 7 == 0
print(flag_a)
print(flag_b)
flag = np.any([flag_a, flag_b],
axis=0)
print(a[flag])
```

多维数组的组合与拆分

垂直方向的操作: vstack() vsplit()

```
# 垂直方向操作
c = np.vstack((a, b))
print(c)
a, b = np.vsplit(c, 2)  # 拆成2份
print(a, '\n', b)
```

水平方向操作: hstack() hsplit()

```
# 水平方向操作
d = np.hstack((a, b))
print(d)
a, b = np.hsplit(d, 2)
print(a, '\n', b)
```

深度方向操作: dstack() dsplit()

```
# 深度方向操作
e = np.dstack((a, b))
print(e)
a, b = np.dsplit(e, 2)
print(a, '\n', b)
```

多维数组组合与拆分的相关函数:

```
#根据axis所指定的轴向(0,1,2)进行多维数组的组合
#如果待组合的两个数组都是二维数组
#axis=0: 垂直方向
#axis=1: 水平方向
#如果待组合的两个数组都是三维数组
#axis=0: 垂直方向
#axis=1: 水平方向
#axis=2: 深度方向
c = np.concatenate((a, b), axis=0)
#通过给定的 axis轴向 与 拆分的份数 对c数组进行拆分
np.split(c, 2, axis=0)
```

长度不等的两个数组的组合操作

```
np.pad(
    ary,                    #原始数组
    pad_width=(0,1),        #补全方式(头补0个,尾补1个)
    mode='constant',        #设置补全模式为constant
    constant_values=-1      #设置补全的默认值为-1
)
```

```
# 测试不同长度的数组的组合
a = np.arange(1, 7)
b = np.arange(11, 16)
c = np.pad(b, pad_width=(2, 2),
           mode='constant',
           constant_values=-1)
print(a)
print(b)
print(c)
```

简单的一维数组的组合方案

```
a = np.arange(1, 10)
b = np.arange(11, 20)
# 把a与b摞在一起成为两行
np.row_stack((a, b))
# 把a与b并在一起成为两列
np.column_stack((a, b))
```

numpy数组的其他属性

1. shape 维度
2. dtype 元素类型
3. size 数组元素个数
4. ndim 维数
5. itemsize 元素字节数
6. nbytes 数组总字节数
7. real 复数数组的实部
8. imag 复数数组的虚部
9. T 数组对象的转置视图
10. flat 返回数组的扁平迭代器