第二章 Python简单介绍

NumPy简介

- NumPy代表 Numerical Python
- 用于高性能计算和数据表示的基本库
- 主要功能包含:
  - 用于创建表示多维数据的ndarray 对象
  - 实现用于标准数学函数的对array数据的整体操作而不需要循环
  - array数据的读/写
  - 线性代数（向量、矩阵操作和运算）

更多内容和例子: 用户手册和快速入门

ndarray vs list of lists(列表的列表)

- 一个班级10个学生的三个成绩(2 个平时，1个期末)
- examGrades = [[79, 95, 60],
  [95, 60, 61],
  [99, 67, 84],
  [76, 76, 97],
  [91, 84, 98],
  [70, 69, 96],
  [88, 65, 76],
  [67, 73, 80],
  [82, 89, 61],
  [94, 67, 88]]

标准实现：
列表的列表

```
In [45]: examGrades[0][2]   #第0个学生的期末成绩
Out[45]: 60

In [46]: examGrades[2]      #第2个学生的所有成绩
Out[46]: [99, 67, 84]
```

- 所有学生的第一个平时成绩?
- 前三个学生的两个平时成绩?

examGrades	list	10	[[79, 95, 60], [95, 60, 61], [99, 67, 84], [76, 76, 97], [91, 84, 98], ...

用ndarray

- gArray = np.array(examGrades)

```
In [3]: gArray
Out[3]:
array([[79, 95, 60],
[95, 60, 61],
[99, 67, 84],
...,
[67, 73, 80],
[82, 89, 61],
[94, 67, 88]])
```

```
In [5]: gArray[0,2]
Out[5]: 60

In [7]: gArray[2,:]
Out[7]: array([99, 67, 84])

#所有学生的第一个平时成绩
In [8]: gArray[:, 0]
Out[8]: array([79, 95, 99, 76, 91, 70,
88, 67, 82, 94])

#前三个学生的两个平时成绩
In [9]: gArray[:3, :2]
Out[9]:
array([[79, 95],
[95, 60],
[99, 67]])
```

ndarray的特点

- ndarray 用于存储类型相同的数据
  所有元素类型相同
- 每个array必须有一个shape和一个dtype
- 支持切片，索引，以及向量化运算
- 避免循环，更加快速有效

```
In [15]: type(gArray)
Out[15]: numpy.ndarray

In [16]: gArray.ndim
Out[16]: 2

In [17]: gArray.shape
Out[17]: (10, 3)

In [18]: gArray.dtype
Out[18]: dtype('int32')
```

向量化运算效率更高

- 例1：求两个p维向量$\boldsymbol{x}, \boldsymbol{y}$的欧式距离 $d = \sqrt{\sum_{j=1}^{p}(x_j - y_j)^2}$

标准python实现

```
d=0
for i in range(len(x)):
    d=d+(x[i]-y[i])**2
d=d**(0.5)
```

Numpy实现

```
d=np.sqrt(sum((x-y)**2))
```

或者 d= np.sqrt(((a-b)**2).sum())

- 例2：矩阵$\boldsymbol{X}_{3\times3}$的每个元素乘以10

```
for i in range(3):
    for j in range(3):
        x[i,j]=x[i,j]*10
```

Numpy实现

```
x=x*10
```

创建 arrays

np.array

np.zeros

np.ones

np.eye

np.arange

np.random

```
In [65]: np.array([[0,1,2],[2,3,4]])
Out[65]:
array([[0, 1, 2],
[2, 3, 4]])

In [66]: np.zeros((2,3))
Out[66]:
array([[ 0., 0., 0.],
[ 0., 0., 0.]])

In [67]: np.ones((2,3))
Out[67]:
array([[ 1., 1., 1.],
[ 1., 1., 1.]])
```

```
In [69]: np.eye(3)
Out[69]:
array([[ 1., 0., 0.],
[ 0., 1., 0.],
[ 0., 0., 1.]])
```

arange、linspace

arange(起始值，结束值，步长)

In [**70**]: np.arange(0, 10, 2)

Out[**70**]: array([0, 2, 4, 6, 8]) #结束值不包含

arange参数可以是实数

In [63]: np.arange(0.2, 5.3, 0.5)

Out[63]: array([0.2, 0.7, 1.2, 1.7, 2.2, 2.7, 3.2, 3.7, 4.2, 4.7, 5.2])

linspace(起始点，终止值，插值个数)

In [**71**]: np.linspace(0, 10, 5)　#默认包含终止点

Out[**71**]: array([0. , 2.5, 5. , 7.5, 10.])

range vs. arange

range为python内置函数，arange是numpy函数，两者功能类似。主要差别：

1. range返回list, arange返回adarray
2. range的参数只能是整数，arange可以是浮点数

随机数np.random

- 随机小数

  np.random.random(n) 返回n个[0,1)之间随机小数

  x=np.random.random((2,3)) 返回2行3列随机值矩阵

- 随机整数

  randint(low, high=None, size=None, dtype='l') 返回从`low` (包括) 到 `high` (不包括)的随机整数

```
In [295]: np.random.randint(0, 10, (3,3))
Out[295]:
 Array([[8, 7, 6],
[0, 8, 9],
[9, 0, 4]])
```

array 运算

- arrays和常数、相同大小array元素之间的运算

```
In [87]: arr = np.array([[0,1,2],[3,4,5]])
```

等价于np.arange(6).reshape(2,3)

```
In [88]: arr * 2
Out[88]:
array([[ 0, 2, 4],
[ 6, 8, 10]])

In [90]: arr ** 2
Out[90]:
array([[ 0, 1, 4],
[ 9, 16, 25]])
```

```
In [94]: arr * arr
Out[94]:
array([[ 0, 1, 4],
[ 9, 16, 25]])

In [95]: arr / (arr+1)
Out[95]:
array([[ 0. , 0.5 , 0.66666667],
[ 0.75 , 0.8 , 0.83333333]])
```

array 索引和切片-1

- 和 python 里面的列表相似, 但是更加灵活

```
In [152]: gArray
Out[152]:
array([[79, 95, 60],
       [95, 60, 61],
       [99, 67, 84],
       ...,
       [67, 73, 80],
       [82, 89, 61],
       [94, 67, 88]])
```

```
#第0行
In [153]: gArray[0]
Out[153]: array([79, 95, 60])
#第1, 2行
In [154]: gArray[1:3]
Out[154]:
array([[95, 60, 61],
       [99, 67, 84]])
#第0行2列
In [156]: gArray[0,2]
Out[156]: 60
```

```
#第2列
In [157]: gArray[:, 2]
Out[157]: array([60, 61, 84, 97, 98, 96, 76, 80, 61, 88])
```

array索引和切片-2

```
In [152]: gArray
Out[152]:
array([[79, 95, 60],
       [95, 60, 61],
       [99, 67, 84],
       ...,
       [67, 73, 80],
       [82, 89, 61],
       [94, 67, 88]])
```

```
#前两行的第0，2列
In [160]: gArray[:2, [0, 2]]
Out[160]:
array([[79, 60],
       [95, 61]])
```

```
#第0，2行的所有列
In [175]: gArray[[0, 2], :]
Out[175]:
array([[79, 95, 60],
       [99, 67, 84]])
```

```
#第0行的第0列，第2行的第2列
两个列分别表示行号和列号
In [178]: gArray[[0, 2], [0, 2]]
Out[178]: array([79, 84])
```

```
#对第0，2行取第0，2列
In [200]: gArray[[0,2]][:,[0,2]]
Out[200]:
array([[79, 60],
       [99, 84]])
```

array赋值

```
In [202]: gArray[0,:]=100

In [203]: gArray
Out[203]:
array([[100, 100, 100],
       [ 95,  60,  61],
       [ 99,  67,  84],
       ...,
       [ 67,  73,  80],
       [ 82,  89,  61],
       [ 94,  67,  88]])
```

与列表一样，用=只能建立array索引

```
In [245]: y=gArray;
          print(y is x)
Out[245]: True   #改变y就是改变gArray
```

用 .copy() 对一个array进行复制.

```
In [254]: arr2 = gArray.copy()

In [255]: arr2 is gArray
Out[255]: False

In [258]: arr2[1,:]=100

In [260]: gArray[1,:]
Out[260]: array([95, 60, 61])
```

基于Boolean做选择

```
# 选择女生的成绩
In [262]: female = [ True, False,
True,  True, False,  True, False,
False, False, False]

In [263]: gArray[female, :]
Out[263]:
array([[100, 100, 100],
       [ 99,  67,  84],
       [ 76,  76,  97],
       [ 70,  69,  96]])
```

```
# 选择期末考试成绩 <= 70
In [265]: gArray[gArray[:, 2]<70,:]
Out[265]:
array([[95, 60, 61],
       [82, 89, 61]])

# 把所有<70的分数变成70
In [267]: gArray[gArray < 70] = 70

In [268]: gArray
Out[268]:
array([[100, 100, 100],
       [ 95,  70,  70],
       [ 99,  70,  84],
       ...,
       [ 70,  73,  80],
       [ 82,  89,  70],
       [ 94,  70,  88]])
```

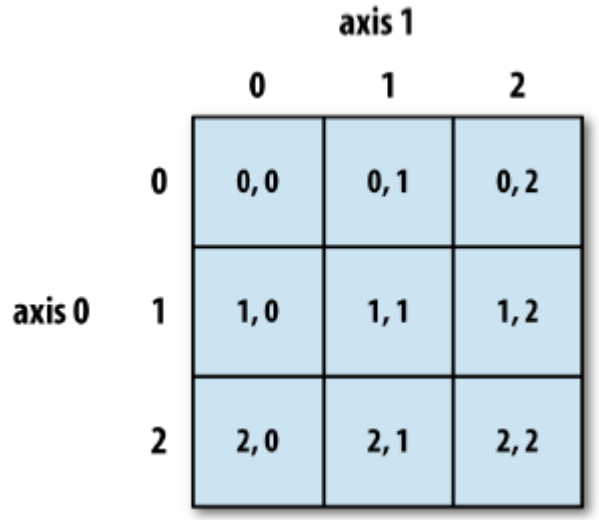
改变形状reshape 和转置 transpose

默认按行

```
In [280]: np.arange(6).reshape((2,3))
Out[280]:
array([[0, 1, 2],
       [3, 4, 5]])
```

按列

```
In [281]: np.arange(6).reshape((2,3), order='F')
Out[281]:
array([[0, 2, 4],
       [1, 3, 5]])
```



转置

```
In [290]:
np.arange(6).reshape(2,3).T
Out[290]:
array([[0, 3],
       [1, 4],
       [2, 5]])
```

排序numpy.sort()

```
a=np.array([ [5, 8, 0, 4],
             [7, 6, 7, 1],
             [1, 5, 1, 1]])
```

默认按行从小到大排序，即axis=1

每行从小到大排序

```
In [407]: a.sort()

In [408]: a
Out[408]:
array([[0, 4, 5, 8],
       [1, 6, 7, 7],
       [1, 1, 1, 5]])
```

每列从小到大排序

```
In [410]: a.sort(axis=0)

In [411]: a
Out[411]:
array([[1, 5, 0, 1],
       [5, 6, 1, 1],
       [7, 8, 7, 4]])
```

b=np.sort(a, axis=0) #返回排序好的结果，a本身不变

求和numpy.sum()

```
In [397]: a.sum(axis=0)   #每列之和
Out[397]: array([13, 19, 8,  6])
In [398]: a.sum(axis=1)#每行之和
Out[398]: array([17, 21, 8])
In [396]: a.sum()   #所有元素之和
Out[396]: 46
```

求最小numpy.min()

```
In [90]: a.min(axis=0) #每列最小
Out[90]: array([1, 5, 0, 1])

In [91]: a.max(axis=1) #每行最大
Out[91]: array([8, 7, 5])
```

判断是否存在numpy.any()

```
In [405]: (a > 5).any(axis=0)   #每列是否有>5的元素
Out[405]: array([ True, True, True, False])
```

判断是否所有都满足numpy.all()

```
In [408]: (a > 0).all(axis=1)   #每行是否全部>0
Out[408]: array([False, True, True])
```

Table 4-5. Basic array statistical methods

Method	Description
`sum`	Sum of all the elements in the array or along an axis. Zero-length arrays have sum 0.
`mean`	Arithmetic mean. Zero-length arrays have NaN mean.
`std, var`	Standard deviation and variance, respectively, with optional degrees of freedom adjustment (default denominator n).
`min, max`	Minimum and maximum.
`argmin, argmax`	Indices of minimum and maximum elements, respectively.
`cumsum`	Cumulative sum of elements starting from 0
`cumprod`	Cumulative product of elements starting from 1

元素级别函数

Table 4-3. Unary ufuncs

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating point, or complex values. Use `fabs` as a faster alternative for non-complex-valued data
`sqrt`	Compute the square root of each element. Equivalent to `arr ** 0.5`
`square`	Compute the square of each element. Equivalent to `arr ** 2`
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
`ceil`	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
`floor`	Compute the floor of each element, i.e. the largest integer less than or equal to each element
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as separate array
`isnan`	Return boolean array indicating whether each value is NaN (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-NaN) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise. Equivalent to `-arr`.

Table 4-4. Binary universal functions

Function	Description
`add`	Add corresponding elements in arrays
`subtract`	Subtract elements in second array from first array
`multiply`	Multiply array elements
`divide, floor_divide`	Divide or floor divide (truncating the remainder)
`power`	Raise elements in first array to powers indicated in second array
`maximum, fmax`	Element-wise maximum. `fmax` ignores NaN
`minimum, fmin`	Element-wise minimum. `fmin` ignores NaN
`mod`	Element-wise modulus (remainder of division)
`copysign`	Copy sign of values in second argument to values in first argument
`greater, greater_equal, less, less_equal, equal, not_equal`	Perform element-wise comparison, yielding boolean array. Equivalent to infix operators `>, >=, <, <=, ==, !=`
`logical_and, logical_or, logical_xor`	Compute element-wise truth value of logical operation. Equivalent to infix operators `&, |, ^`

更具体用法：https://docs.scipy.org/doc/numpy/reference/

矩阵

- 一个$n \times d$ 的矩阵A表示为：

$$\mathrm{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d} \\ a_{21} & a_{22} & \cdots & a_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd} \end{pmatrix}$$

- 其中a_{ij}表示为矩阵A中第i行第j列的元素。
- 第i行的所有元素： $a_{i1}, a_{i2}, \cdots, a_{id}$.
- 第j列的所有元素： $a_{1j}, a_{2j}, \cdots, a_{nj}$.

向量

- 若一个矩阵的行或列只有一维，则可以称其为向量。
  - 例：一个行向量可表示为 $(1,\ 10,\ 11,\ 12,\ 7)$
  - 例：一个列向量可表示为 $\begin{pmatrix} 7 \\ -2 \\ 5 \\ 11 \end{pmatrix}$
- 若$\boldsymbol{a}$为一个行向量，则$\boldsymbol{a^T}$表示为该向量的转置，为一个列向量。

对角矩阵

- 一个对角矩阵是一个方阵（为$n \times n$矩阵），且非主对角线上的元素均为零（在主对角线上的元素可以为零或非零）。

- 一个$n \times n$的对角矩阵用D_n表示。
- 例：$D_n = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 7 \end{pmatrix}$, $D_n = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 5 \end{pmatrix}$
- 单位矩阵：主对角线的元素全部为1时的对角矩阵

一个$n \times n$阶的点位矩阵用I_n表示, 如 $I_n = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

向量的点积

- 点积(dot product)又称标积(scalar product)，是指对两个向量计算得到一个实数标量的运算。

- 将一个行向量的第i个元素于与另一个行向量的第i个元素相乘，并且将各元素相乘的结果求和，得到点积。

- 如有$\boldsymbol{a} = (a_1\ a_2\ \cdots a_n), \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

- 则$\boldsymbol{a} \cdot \boldsymbol{b} = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$

```
In [303]: a = b = np.arange(5)
In [305]: a
Out[305]: array([0, 1, 2, 3, 4])
In [306]: b
Out[306]: array([0, 1, 2, 3, 4])
In [307]: a.dot(b)
Out[307]: 30
```
#或sum(a*b)

点积也是向量的内积，表示为一个
行向量与一个列向量的乘，假设两个列向量$\boldsymbol{x}, \boldsymbol{y}$，它们的内积为$\boldsymbol{x^T y}$。

矩阵相乘

Numpy矩阵相乘

C=A.dot(B)

```
In [204]: a.shape
Out[204]: (5, 2)

In [205]: b.shape
Out[205]: (2, 3)

In [206]: c=a.dot(b)

In [207]: c.shape
Out[207]: (5, 3)
```

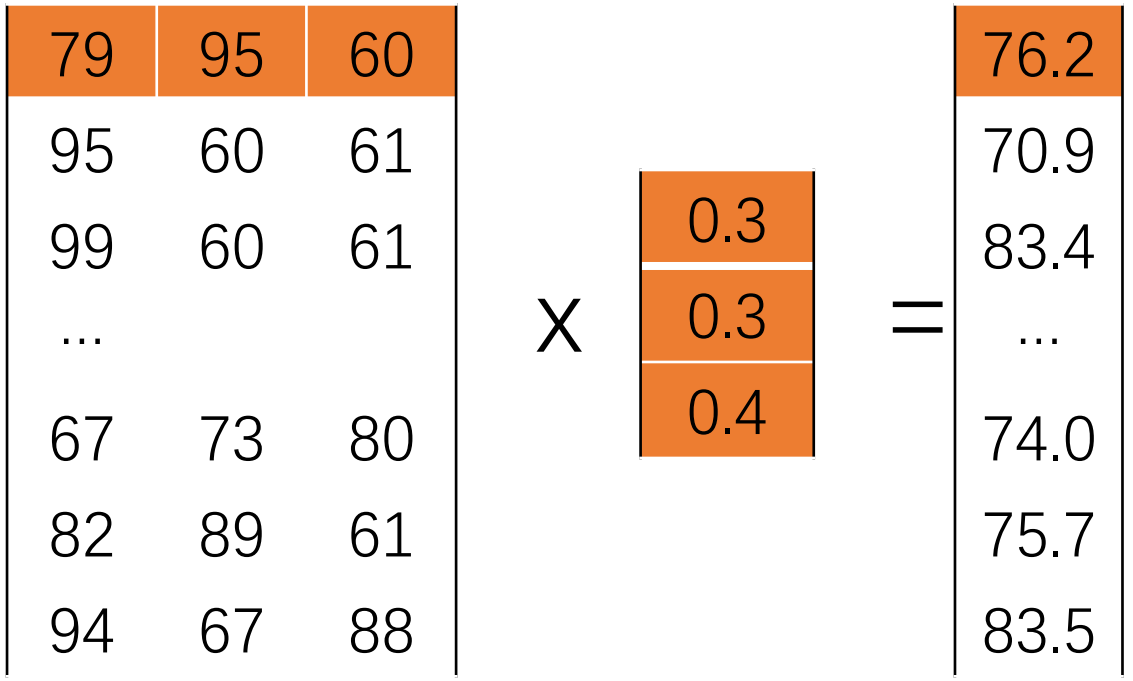
- 假设矩阵A是一个$m \times p$矩阵， B是一个$p \times n$矩阵（A 的列数等于B的行数）。则$C = AB$为一个$m \times n$的矩阵（其行等于矩阵A的行，其列等于矩阵B的列）。
- 元素c_{ij}为矩阵A的第i行与矩阵B的第j列的点积。例：

$$A = \begin{pmatrix} a_{12} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \\ b_{41} & b_{42} \end{pmatrix}$$

在矩阵C中$(2,1)$位置的元素的值应为矩阵A的第2行与矩阵B的第一列的点积：

$$a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} + a_{24}b_{41}$$

```
In [152]: gArray
Out[152]:
array([[79, 95, 60],
       [95, 60, 61],
       [99, 67, 84],
       ...,
       [67, 73, 80],
       [82, 89, 61],
       [94, 67, 88]])
```




```
In [321]: gArray.dot([0.3, 0.3, 0.4])
Out[321]: array([ 76.2,  70.9,  83.4,
84.4,  91.7,  80.1,  76.3,  74. ,
75.7,  83.5])
```

结果代表什么?

```
In [152]: gArray
Out[152]:
array([[79, 95, 60],
       [95, 60, 61],
       [99, 67, 84],
       ...,
       [67, 73, 80],
       [82, 89, 61],
       [94, 67, 88]])
```

结果代表什么?

```
In [329]: scaling = [1.1, 1.05, 1.03]
```

向量->对角矩阵

```
In [330]: np.diag(scaling)
Out[330]:
Array([[ 1.1 ,  0.  ,  0.  ],
       [ 0.  ,  1.05,  0.  ],
       [ 0.  ,  0.  ,  1.03]])

In [331]: gArray.dot(np.diag(scaling))
Out[331]:
array([[  86.9 ,   99.75,   61.8 ],
       [ 104.5 ,   63.  ,   62.83],
       [ 108.9 ,   70.35,   86.52],
       ...,
       [  73.7 ,   76.65,   82.4 ],
       [  90.2 ,   93.45,   62.83],
       [ 103.4 ,   70.35,   90.64]])
```

```
In [152]: gArray
Out[152]:
array([[79, 95, 60],
       [95, 60, 61],
       [99, 67, 84],
       ...,
       [67, 73, 80],
       [82, 89, 61],
       [94, 67, 88]])
```

结果代表什么?

求最大值（axis=0/1代表按列/行）

```
In [338]: maxInExam = gArray.max(axis=0)

In [339]:
gArray.dot(np.diag(100/maxInExam)).round()
Out[339]:
array([[  80.,  100.,   61.],
       [  96.,   63.,   62.],
       [ 100.,   71.,   86.],
       ...,
       [  68.,   77.,   82.],
       [  83.,   94.,   62.],
       [  95.,   71.,   90.]])
```

四舍五入

第二章 Python简单介绍

从外部文件中导入

```
#读入 csv 文件，路径用“/”，文件名中的.csv不能省略
import pandas as pd
df = pd.read_csv("D:\教学\数据挖掘\数据挖掘\数据挖掘
\python\datasets\Salaries.csv")
```

用pandas读如其他类型文件:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None)
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

data frame对象

In [3]:
```
#列出最前面的5个记录
df.head()
```

Out[3]:

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

- numpy 的ndarray对象是包含**相同数据类型**的矩阵；
- pandas中的data frame对象的；每一列类型可以不同，方便用来存储原始数据。

动手练习

- ✓尝试列出前面的 10, 20, 50 个记录;
- ✓猜猜怎么列出最后的若干个记录？比如最后3个？

Data Frames 属性

Python 对象有属性（*attributes*）和方法（*methods*）.

df.attribute	描述
dtypes	每一列的数据类型
columns	每列对应的名称
axes	每行序号和每列名称
ndim	维度（如果是表格则等于2）
size	所有元素的数目
shape	返回一个元组，表示数据每个维度大小（几行几列）

Data Frame 数据类型

```
In [4]: #查看某一列的数据类型
        df['salary'].dtypes
```

```
Out[4]: dtype('int64')
```

```
In [5]: #产看所有列的数据类型
        df.dtypes
```

```
Out[4]: rank          object
        discipline    object
        phd           int64
        service       int64
        sex           object
        salary        int64
        dtype: object
```

动手练习

- ✓一共有多少个记录?
- ✓一共有多少个元素?
- ✓每列的名字?
- ✓每列的数据类型?

Data Frames 方法

与属性不同, python中对象的方法有括号

df.method()	描述
head([n]), tail([n])	列出最前面/后面 n 行， n不给时默认5行
describe()	得到数值列的基本统计信息， 平均/最大/最小值等
max(), min()	数值列的最大/最小值
mean(), median()	数值列的平均/中位数
std()	标准差
sample([n])	返回一个随机抽样集合
dropna()	丢弃所有包含缺失值的记录/行
to_numpy()	转成numpy 的array

动手练习

- ✓查看数据集中数值列的统计信息?
- ✓所有数值列的方差?
- ✓前面 50个记录的平均值? ***提示:*** 先用 head() 方法提取前面的 50 的记录，然后计算平均值

从Data Frame中选择一列

方法1: 用列的名字:

df['salary']

方法2: 用列的名字作为属性:

df.salary

计算平均salary:

df['salary'].mean() 或 df.salary.mean()

注意: 因为 pandas data frames有个属性叫rank, 所以如果有一列的名字也是rank, 那么要用方法1来提取这一列.

Data Frames 的*groupby*方法

用 “groupby” 方法可以:

- 按照某些标准把数据集划分到不同的组;
- 对每个组进行操作，应用函数

In [ ]:
```
#用rank分组
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#计算每个组中数值列的平均值
df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

In [ ]:
```
#对不同级别计算平均salary:
df.groupby('rank')[['salary']].mean()
```

	salary
rank	
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

注意: 上面'salary'外面是双重[ ]时返回的是Date Frame对象，如果是单重[ ]则返回Series对象。

Data Frame: 过滤（filtering）

用Boolean索引进行数据子集提取，也叫过滤。

In [ ]:
```
#提取salary高于 $120K的行:
df_sub = df[ df['salary'] > 120000 ]
```

Boolean 运算都可以用于提取子集:

> ; >=;

<; <=;

==; !=;

In [ ]:
```
#选择对应女性的记录:
df_f = df[ df['sex'] == 'Female' ]
```

Data Frames: 切片（Slicing）

有好几种方法可以提取Data Frame中的子集:

- 一列或几列
- 一行或几行
- 某些行的某些列

行和列可以通过下标/位置或标签来指定

如果想选择多列并且返回Data Frame对象，那就要用双重[ ]:

```
#选择rank和salary:
df[['rank','salary']]
```

如果选择一列, 可以用单对方括号，但得到的是 Series对象不是 Data Frame:

```
#选择salary:
df['salary']
```

```
#通过位置/下标选择行。注意：第一行下标是0, 最后一行不包括:
df[10:20]
```

Data Frames: loc和 iloc

用方法loc基于标签进行选择

```
df_sub.loc[10:20,['rank','sex','salary']]
```

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

用方法iloc基于位置进行选择

```
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: iloc方法 (总结)

```
df.iloc[0]   # 第一行
df.iloc[i]   #第(i+1)行
df.iloc[-1] # 最后一行
```

```
df.iloc[:, 0]   # 第一列
df.iloc[:, -1] # 最后一列
```

```
df.iloc[0:7]          #前面7行
df.iloc[:, 0:2]       #前面2列
df.iloc[1:3, 0:2]     #第2，3行的前面两列
df.iloc[[0,5], [1,3]]   #第1和 6行的第2、4列
```

Data Frames: 排序

默认从小到大排序，返回一个新的 data frame。

```
# 返回按照service从小到大排好的data frame
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

```
# 用多个标准排序
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

先按照service排序，然后对service的值一样的情况下按salary排序

缺失值

NaN代表缺失值

In [ ]:
```
# 读入包含缺失值的文件
flights = pd.read_csv("D:\flights.csv")
```

In [ ]:
```
# 选择至少包含一个缺失值的行
flights[flights.isnull().any(axis=1)].head()
```

Out[ ]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

缺失值

处理data frame中缺失值的方法，用 pd.DataFrame.dropna?查看例子

df.method()	描述
dropna()	丢弃包含缺失值的行
dropna(how='all')	丢弃每个元素都是 NaN的行
dropna(axis=1, how='all')	丢弃所有元素缺失的列
dropna(thresh = 5)	丢弃包含少于 5个值的行
fillna(0)	用0代替缺失值
isnull()	如果缺失返回 True
notnull()	如果不缺失返回 True

对缺失值的默认处理：

- 相加的时候，缺失值当作0处理；如果所有值缺失，和为NaN。
- 在GroupBy 方法中缺失值不被包括。
- 很多统计方法有 skipna 选项来控制是否不包含缺失值 . 该选项默认True，即跳过缺失值。

从sklearn内置数据集中导入

关键字	描述
DESCR	对该数据集的文字介绍
data	数据矩阵，类型为ndarray实数
feature_names	特征（属性）名，字符类型
filename	文件名
target	类标签，表示为整型数值
target_names	类标签描述，字符类型

通过sk-learn.datasets获得数据集 website

更多关于scikit-learn的用法: website

导入iris数据集

```
In [173]: from sklearn.datasets import load_iris

In [174]: irisdata = load_iris()

In [175]: irisdata.data[:3]
Out[175]:
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2]])
```

把数据矩阵转成DataFrame类型:
x=pd.DataFrame(irisdata.data)

```
In [177]: irisdata.feature_names
Out[177]:
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

第二章 Python简单介绍

2.1 python介绍和基本语法

2.1.1 python简介、安装、编辑器

2.1.2 基本语法

2.2 数据分析常用库的介绍

2.2.1 高性能计算numpy

2.2.2 数据导入和预处理pandas

2.2.3 画图和可视化matplotlib

画图和可视化matplotlib

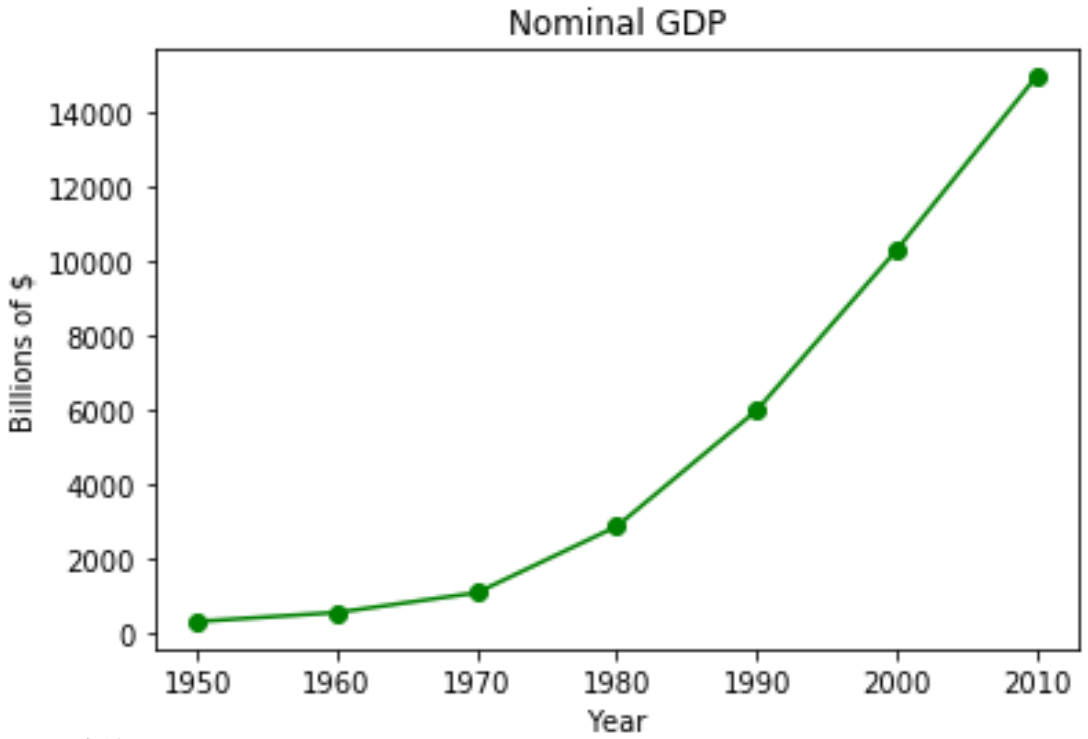
基于matplotlib的画图功能对数据或结果进行可视化;

常用图形包括:

- 折线图 Line graph
- 柱状图 Bar Chart
- 散点图 Scatter

折线图： 用于看趋势

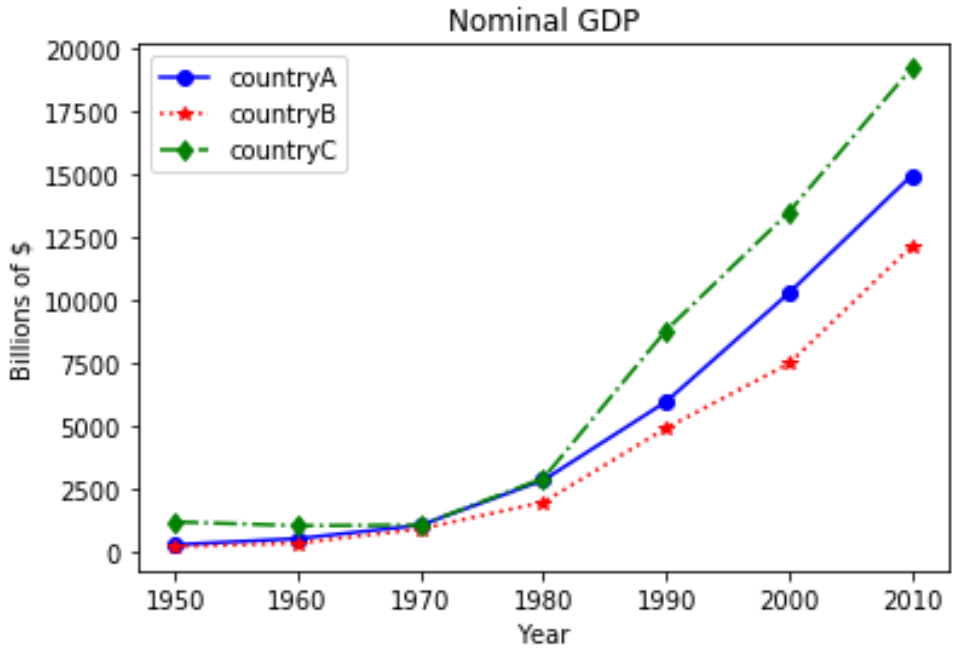
```
import matplotlib.pyplot as plt

years = list(range(1950, 2011, 10))
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
# 创建折线图，x轴位年份，y轴为gdp
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')

# 加入标题
plt.title("Nominal GDP")

# 在y轴添加说明
plt.ylabel("Billions of $")

# 在x轴添加说明
plt.xlabel("Year")
plt.show()
```



输入 plt.plot? 查看更多用法

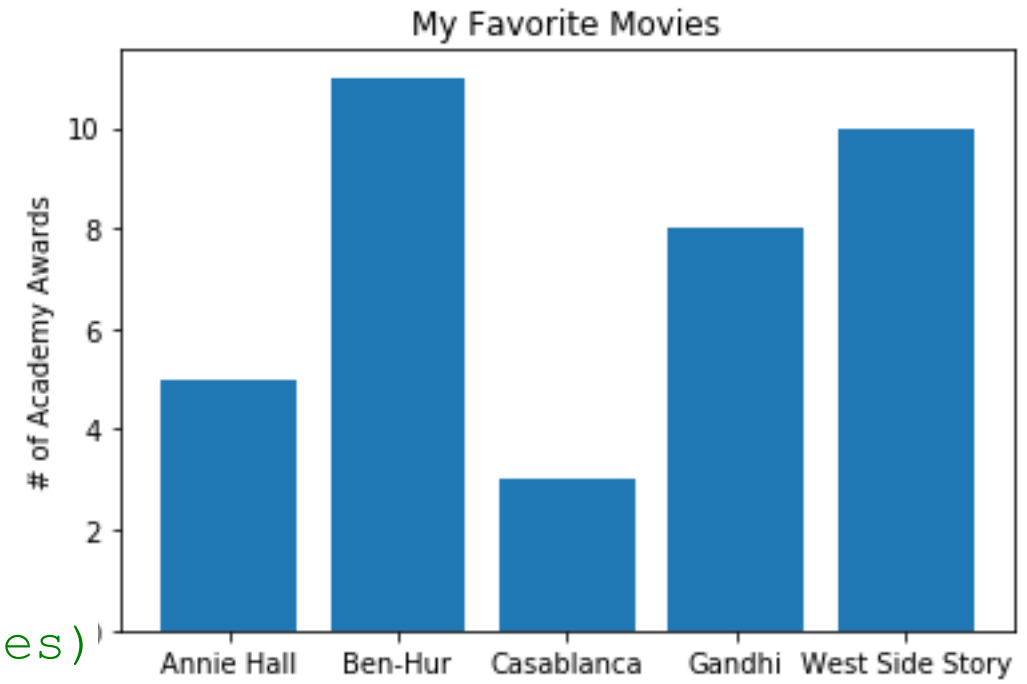
折线图： 多条线在一张图

```
import matplotlib.pyplot as plt
years = list(range(1950, 2011, 10))
gdp1 = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]
gdp2 = [226.0, 362.0, 928.0, 1992.0, 4931.0, 7488.0, 12147.0]
gdp3 = [1206.0, 1057.0, 1081.0, 2940.0, 8813.0, 13502.0, 19218.0]

# 用不同颜色、标记、和线型来代表三个不同的gdp.
# 例如 'bo-'代表蓝色， 标记为o，线段类型为实线
plt.plot(years, gdp1, 'bo-',
         years, gdp2, 'r*:',
         years, gdp3, 'gd-.')
plt.title("Nominal GDP") # 添加标题
plt.ylabel("Billions of $") #在y轴添加说明
plt.xlabel("Year") #在x轴添加说明
# 添加图例
plt.legend(['countryA', 'countryB', 'countryC'])
plt.show()
```



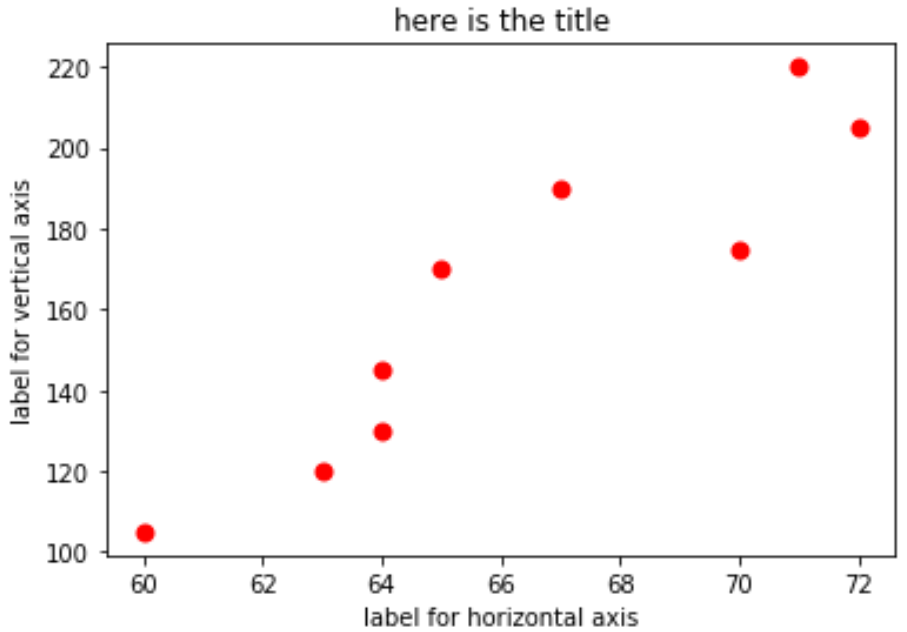
柱状图：用于对比不同分组的结果

```
movies = ["Annie Hall", "Ben-Hur", "Casablanca", "Gandhi", "West Side Story"]
num_oscars = [5, 11, 3, 8, 10]

xs = range(len(movies)) # xs is range(5)

# 以xs为横坐标，num_oscars为纵坐标画出柱状图
plt.bar(xs, num_oscars)
# 把x轴用电影名标出
plt.xticks(xs, movies)
# 或者用以下代码代替上面两行
#plt.bar(xs, num_oscars, tick_label=movies)

plt.ylabel("# of Academy Awards")
plt.title("My Favorite Movies")
plt.show()
```



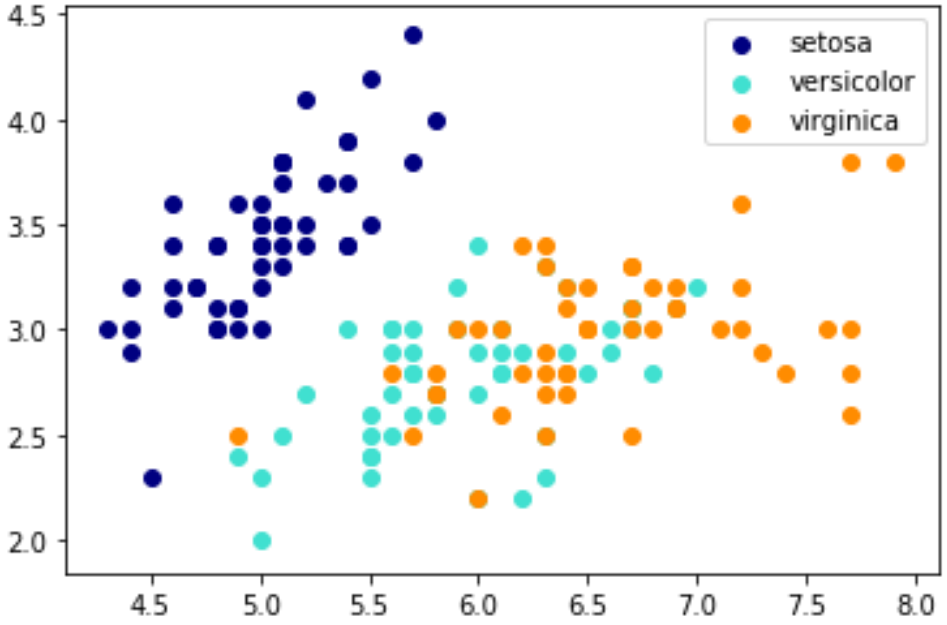
散点图： 对2维数据分布进行可视化

```
friends = [ 70, 65, 72, 63, 71, 64, 60, 64, 67]
minutes = [175, 170, 205, 120, 220, 130, 105, 145, 190]

plt.scatter(friends, minutes, lw=2,color="red")
plt.title("here is the title")
plt.xlabel("label for horizontal axis")
plt.ylabel("label for vertical axis")
plt.show()
```



散点图： 对iris数据进行可视化

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
X = iris.data
y = iris.target
colors = ['navy', 'turquoise', 'darkorange']
```




```
for color, i, target_name in zip(colors, [0, 1, 2], iris.target_names):
    plt.scatter(X[y == i, 0], X[y == i, 1],color=color, label=target_name)
plt.legend()
plt.show()
```