

基于Python的数据分析

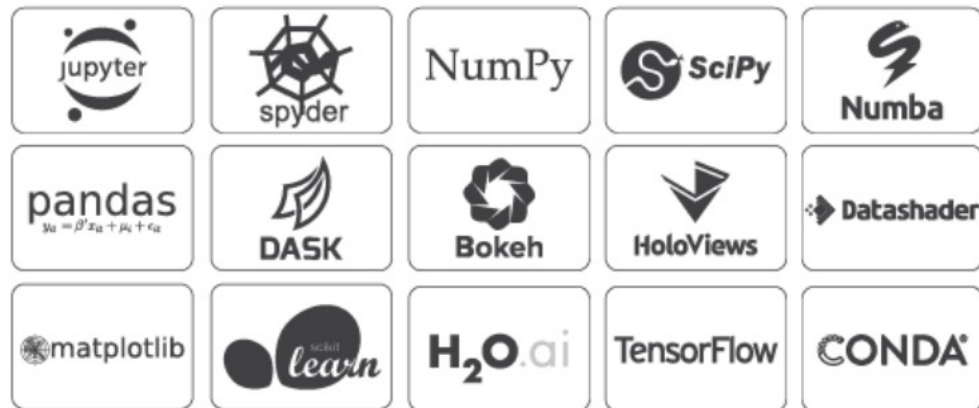
- Python简介、安装和编程环境
- 高性能计算NumPy
- 数据导入和预处理pandas
- 机器学习库scikit-learn
- 画图和可视化matplotlib

Python简介、安装和编程环境

- 于90年代早期在荷兰由吉多·范罗苏姆（Guido van Rossum）发明；
- 是一种开源的脚本语言，但比普通脚本具有更多功能；
- Python不需要编译
 - C需要编译，通过编译得到二进制机器码后才能被执行
 - Python是解释器，逐行解释后立即执行
- Python语言具有简洁性、易读性以及可扩展性；
- 目前流行用Python做科学计算；
- 经典的科学计算扩展库：NumPy、SciPy和matplotlib，分别为Python提供了快速进行数组处理、数值运算以及绘图功能。

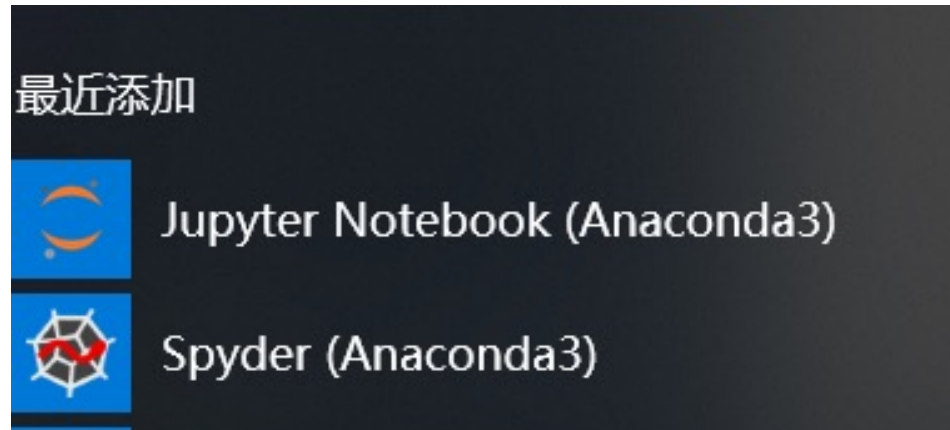
Python 安装：（方法一）Anaconda（建议采用）

- 用于科学计算的开源发行版，包括 python 环境和 IDE (spyder) ；
- 集成了上百个科学计算的第三方模块，需要时可直接导入 ；
- 解决多版本并存
- 官方下载地址：<https://www.anaconda.com/download/>

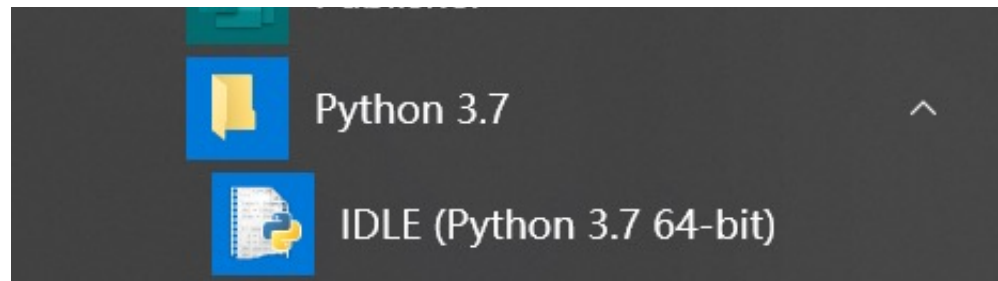


运行和编辑

- 安装完Anaconda3后就已经装了jupyter nootbook和Spyder两个编辑器， 可以从开始菜单里面找到：



- Python 也自带的编辑器IDLE




本课程采用 jupyter notebook

- 启动 jupyter notebook

```
Jupyter Notebook (Anaconda3)
[I 16:37:28.798 NotebookApp] JupyterLab extension loaded from D:\Anaconda3\lib\site-packages\jupyterlab
[I 16:37:28.798 NotebookApp] JupyterLab application directory is D:\Anaconda3\share\jupyter\lab
[I 16:37:28.801 NotebookApp] Serving notebooks from local directory: C:\Users\梅建萍
[I 16:37:28.802 NotebookApp] The Jupyter Notebook is running at:
[I 16:37:28.802 NotebookApp] http://localhost:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
[I 16:37:28.802 NotebookApp] or http://127.0.0.1:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
[I 16:37:28.803 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 16:37:28.846 NotebookApp]

To access the notebook, open this file in a browser:
file:///C:/Users/%E6%A2%85%E5%BB%BA%E8%90%8D/AppData/Roaming/jupyter/runtime/nbserver-38172-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
or http://127.0.0.1:8888/?token=516ea08db27994ebb1aa35ebfa25b6bbd605316d4737cbcf
```

有时需要手动复制URL到浏览器：

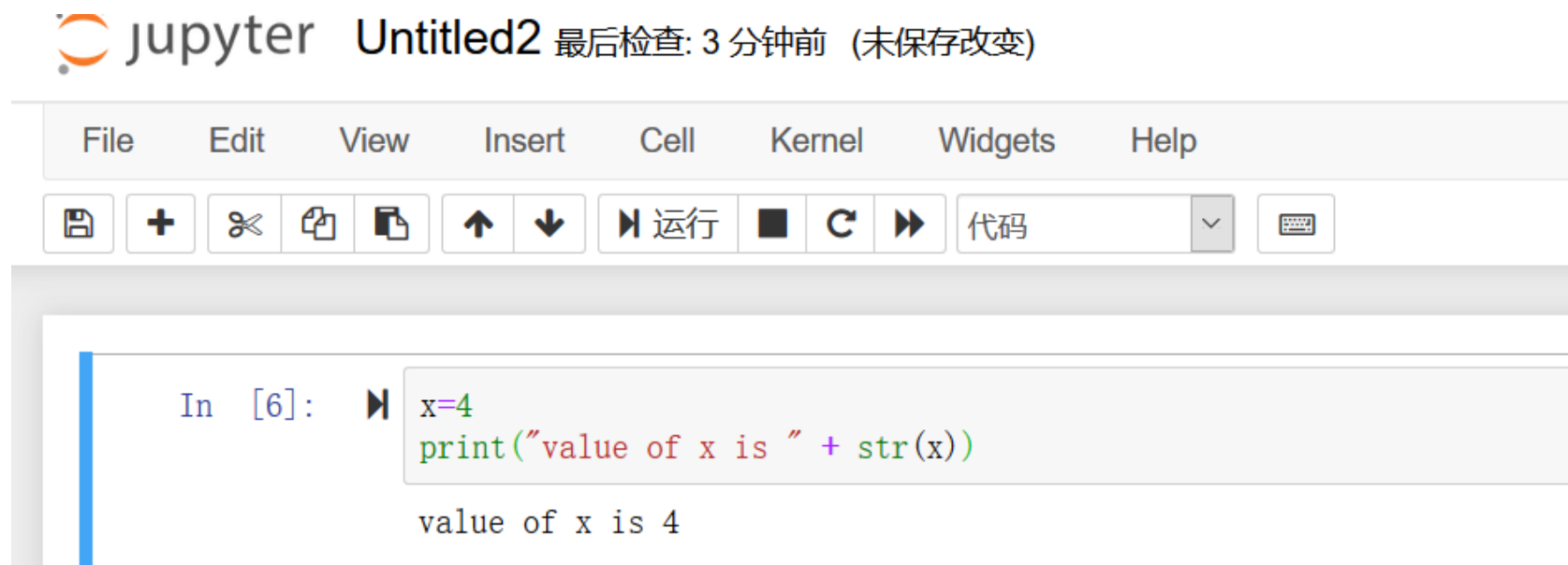
 Quit Logout

Files Running Clusters

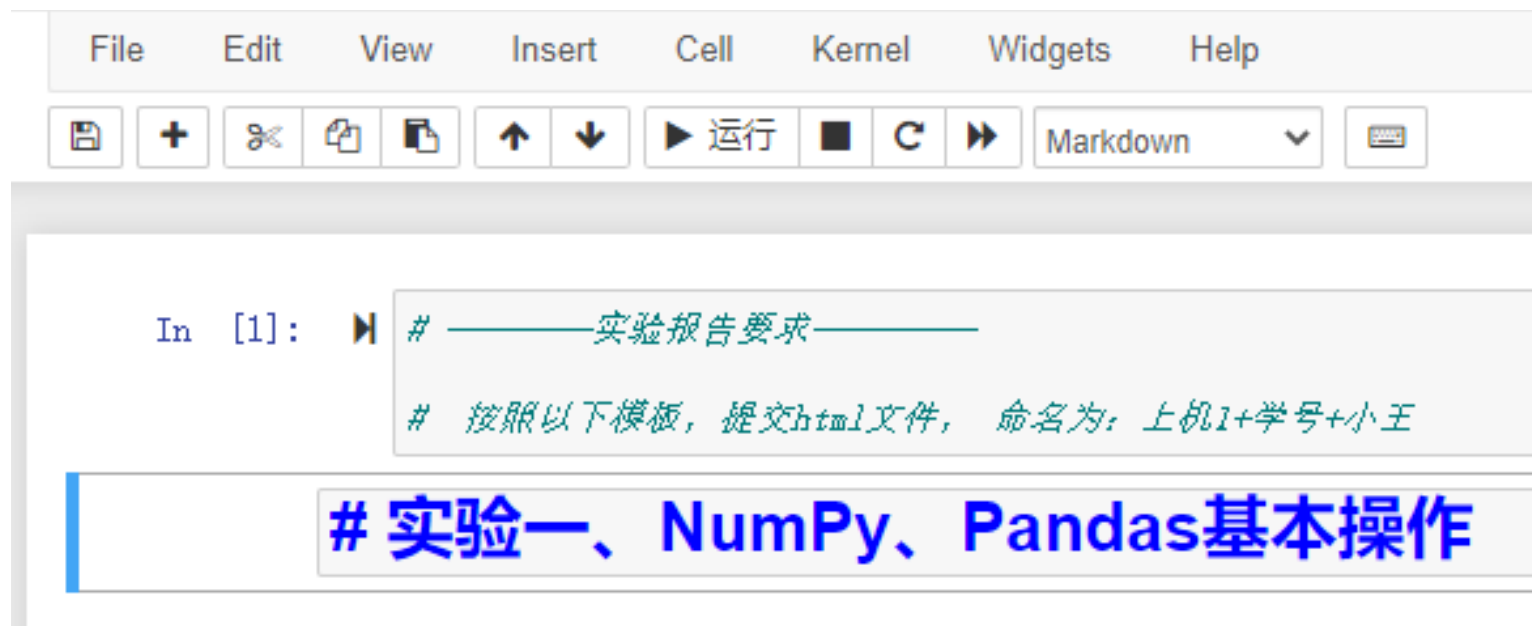
Select items to perform actions on them. Upload New ▾ ↻

<input type="checkbox"/> 0 ▾	📁 /	Name ▾	Last Modified	File size
<input type="checkbox"/>	📄	Untitled.ipynb	4 个月前	8.13 kB
<input type="checkbox"/>	📄	Untitled1.ipynb	1 个月前	22.9 kB
<input type="checkbox"/>	📄	easy_install-3.7.exe	4 个月前	103 kB
<input type="checkbox"/>	📄	iptest.exe	4 个月前	103 kB

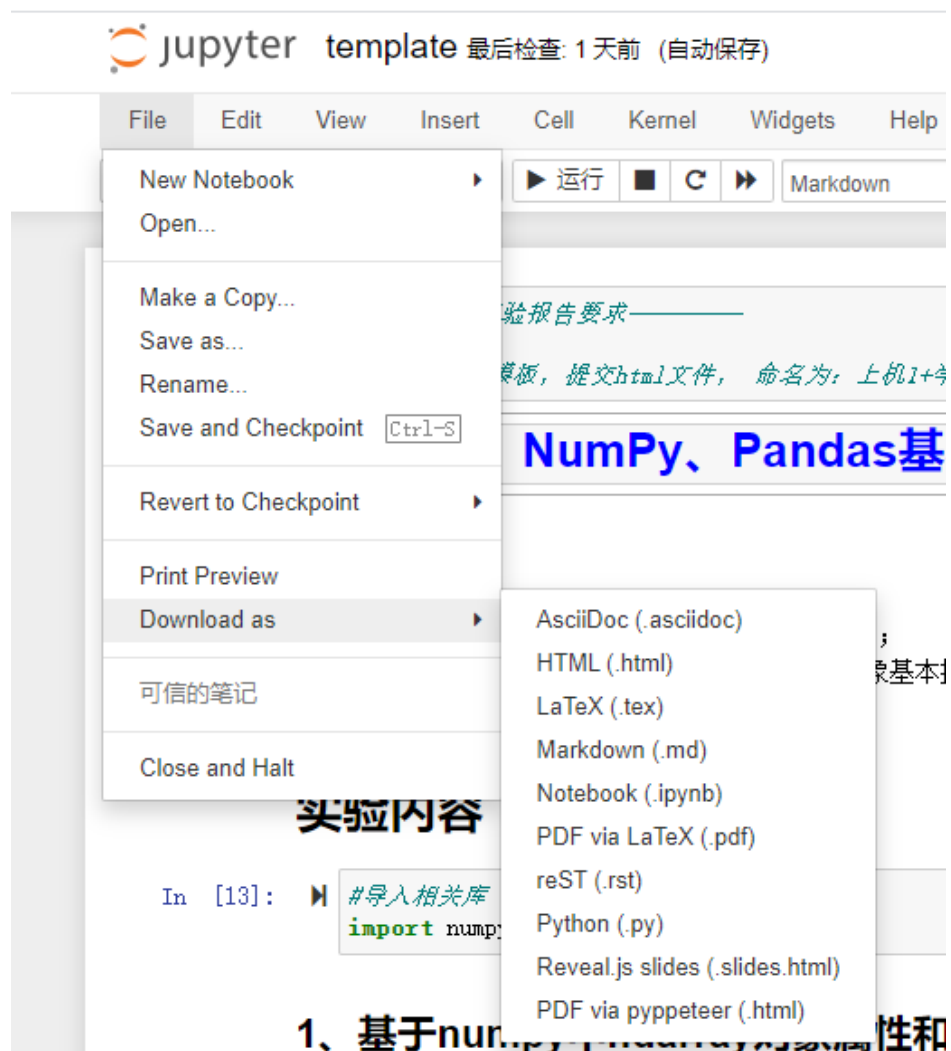
进入主页后点击右上角new 选择python3创建.ipynb文件



选择Markdown进行文档编辑



导出不同类型文件

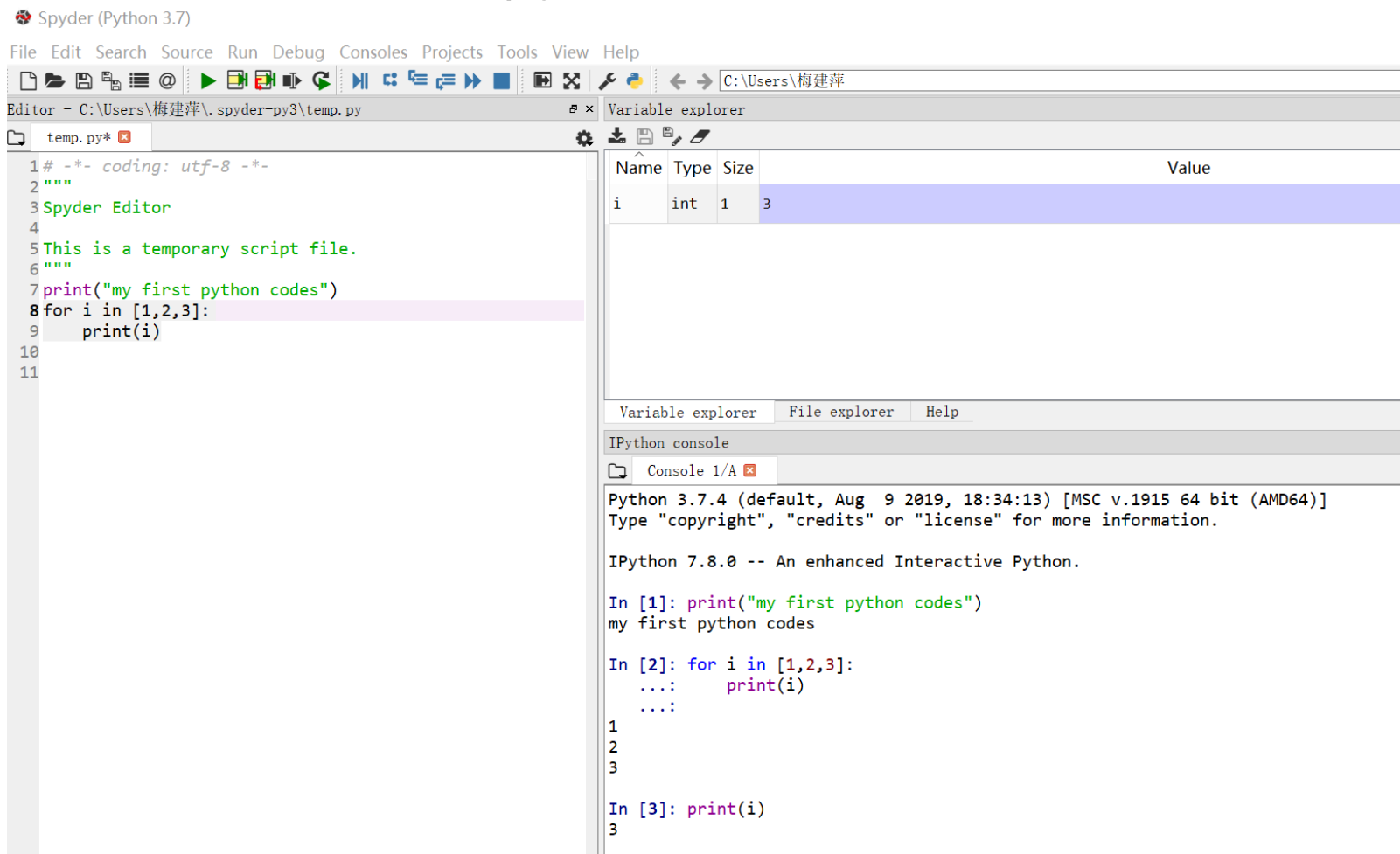


可以选择导出不同类型文件，包括：

- .html （实验报告）
- .ipynb （jupyter 代码）
- .py （python 代码）

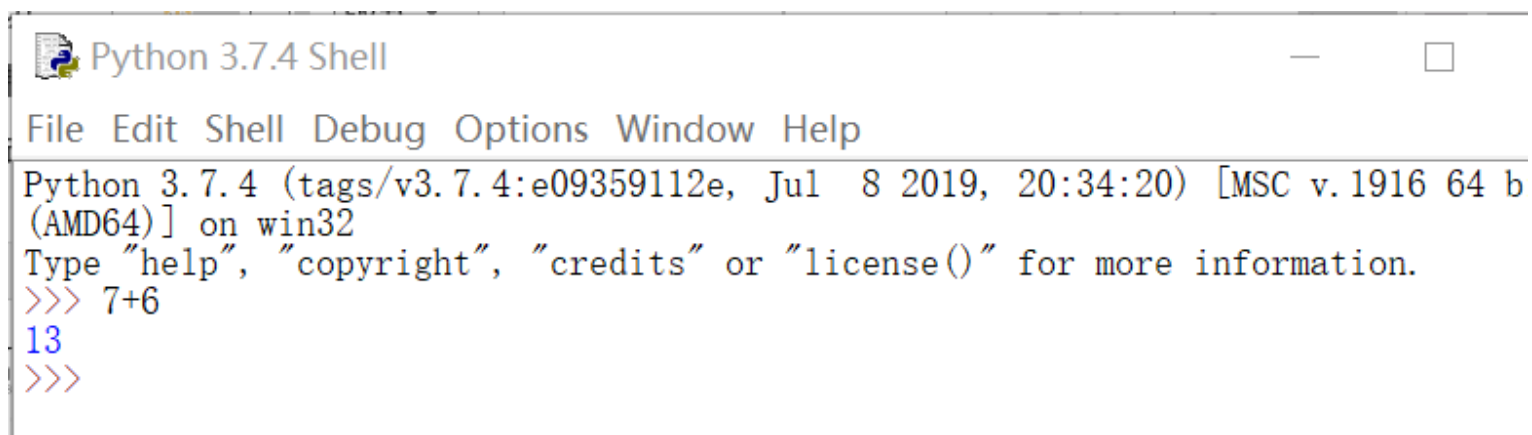
Spyder

- 与Matlab类似的python编辑器



如果你习惯
Eclipse, 也可以用
PyCharm。

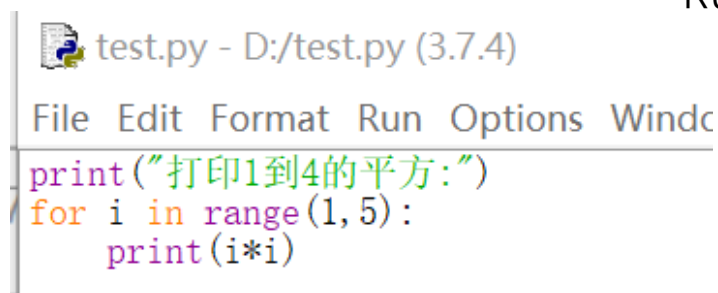
IDLE (python自带编辑器)



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 b
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 7+6
13
>>>
```

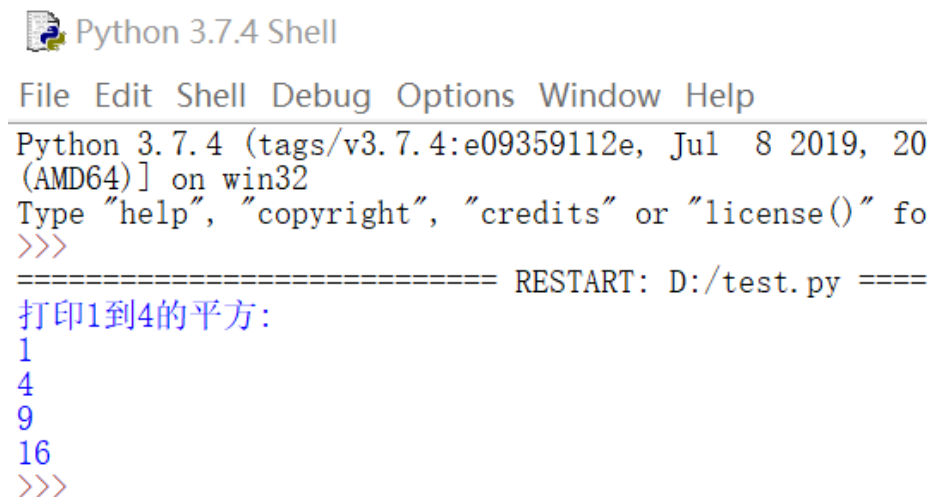
File->New File

编写程序



```
test.py - D:/test.py (3.7.4)
File Edit Format Run Options Window
print("打印1到4的平方:")
for i in range(1,5):
    print(i*i)
```

Run->Run Module(F5)



```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
>>>
==== RESTART: D:/test.py ====
打印1到4的平方:
1
4
9
16
>>>
```

Python 安装: (方法二) 通过pip自定义安装

1. 安装 python 3.9.2 官方[下载](#)

2. 安装完python后用pip安装其他packages

在python 安装目录下, 命令行输入:

```
python -m pip install --upgrade pip
```

```
python -m pip install --user ipython jupyter numpy scipy matplotlib pandas
```

ipython、jupyter为基于网页的交互式应用

NumPy, SciPy, matplotlib, pandas最常用的 Python科学计算和数据分析工具包/库, 用户说明和例子

<https://www.scipy.org/docs.html>

Python 基本语法（自学）

- 变量赋值、算术运算、关系运算、字符串
- 条件和循环
- 函数定义和调用
- Python标准数据类型和函数：列表、元组、字典、range()函数

自学资料：课件Python基本语法，Python官方网站文档

基于Python的数据分析

- Python简介、安装和编程环境
- 高性能计算NumPy
- 数据导入和预处理pandas
- 机器学习库scikit-learn
- 画图和可视化matplotlib

高性能计算NumPy

- 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](#)

例子：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

```
import numpy as np
```

```
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

ndarray对象基本属性

dtype 数据类型

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）

行数：shape[0]

```
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')
```

Python 对象包含属性 (*attributes*)
和方法 (*methods*)。

向量化运算效率更高

- 例1：求两个 p 维向量 \mathbf{x}, \mathbf{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 $d = ((x-y)**2).sum()**0.5$

- 例2：矩阵 $\mathbf{X}_{3 \times 3}$ 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
```

sum() 是Python中的向量求和函数，np.sqrt()是NumPy下面的求开方函数。而.sum()是ndarray对象的方法，因此必须是ndarray对象。

对ndarray的基本操作

- 创建ndarray对象
- 索引、切片、遍历

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.]])

In [46]:  x=np.random.random((2,3))
x

Out[46]: array([[0.73880115, 0.9282281 , 0.12093993],
[0.03371026, 0.95523213, 0.79642864]])

可以在创建时通过dtype指定数据类型

```
a=np.zeros((2,3),dtype='int32')
```

arange、linspace

arange(起始值, 终止值, 步长)

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

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

arange的参数可以是实数

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ndarray
2. range的参数只能是整数, arange可以是浮点数

array 索引和切片

- 和 python 里面的列表相似, 但是更加灵活

注意：
第一行下标是0, 最后一行不包括

```
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行第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

```
In [90]: ▶ #最后一行  
gArray[-1]
```

```
Out[90]: array([94, 67, 88])
```

1. 最后两列怎么表示？

```
In [91]: ▶ #最后两行  
gArray[-2:]
```

```
Out[91]: array([[82, 89, 61],  
                [94, 67, 88]])
```

2. gArray[: -1]表示什么？

```
In [92]: ▶ #最后一列  
gArray[:, -1]
```

```
Out[92]: array([60, 61, 84, 97, 98, 96, 76, 80, 61, 88])
```

array索引和切片

```
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]])
```

In [50]:  #向量a中指定的行

```
a=[1, 3, 5]
gArray[a]
```

Out[50]: array([[95, 60, 61],
 [76, 76, 97],
 [70, 69, 96]])

#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]])
```


Boolean索引进行子集提取/过滤

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 的学生

```
In [265]: gArray[gArray[:, 2]<70, :]
```

```
Out[265]:
```

```
array([[95, 60, 61],
       [82, 89, 61]])
```

结果是什么？

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]])
```

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 gArray)
```

```
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])
```

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]])
```

ndarray的排序方法.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

ndarray的求和.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
```

ndarray求最小.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])
```

ndarray的判断是否存在.any()

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

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

Table 4-4. Binary universal functions

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

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

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]])
```

矩阵

- 一个 $n \times d$ 的矩阵 A 表示为:

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

- 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 \mathbf{a} 为一个行向量，则 \mathbf{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 $\mathbf{a} = (a_1 \ a_2 \ \cdots \ a_n)$, $\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

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

```
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)          #或sum(a*b)
Out[307]: 30
```

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 \mathbf{x}, \mathbf{y} , 它们的内积为 $\mathbf{x}^T \mathbf{y}$ 。

矩阵相乘

NumPy矩阵相乘 .dot()

```
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：

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

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]])
```

79	95	60
95	60	61
99	60	61
...		
67	73	80
82	89	61
94	67	88

 \times

0.3
0.3
0.4

 $=$

76.2
70.9
83.4
...
74.0
75.7
83.5

```
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])
```

自动对列表
进行转换后
相乘。

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]: 四舍五入

```
array([[ 80., 100., 61.],
       [ 96., 63., 62.],
       [100., 71., 86.],
       ...,
       [ 68., 77., 82.],
       [ 83., 94., 62.],
       [ 95., 71., 90.]])
```

基于Python的数据分析

- Python简介、安装和编程环境
- 高性能计算NumPy
- 数据导入和预处理pandas
- 机器学习库scikit-learn
- 画图和可视化matplotlib

回顾：NumPy的ndarray对象

- 创建ndarray对象矩阵
- 基本属性：dtype、shape、ndim等
- 基本方法：.sum()、.min()/.max()、.sort()、.mean () 等
- 索引、切片：某几行、某几列、某几行的某几列、最后几行/几列等
- 按条件删选/过滤：大于某个值的行/列、.any()、.all()
- 矩阵/向量元素级别运算、相乘.dot()、转置.T
- 常用参数设置，如设置axis为0或1选择按列/按行等

pandas的data frame对象

NumPy 的ndarray对象是包含**相同数据类型**的矩阵；

原始数据每一列类型很可能不同

	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

某个记录，或样本

某个特征或属性

基于pandas从外部文件中导入数据

```
#读入 csv 文件, 路径用 "/" , 文件名中的.csv不能省略, 默认第一行为对每列的说明  
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")  
#如果文件中的第一行就是数据, 而不是每列的说明, 则用header=None选项, 不然第一  
行数据会认为是属性名。  
df.columns=["", "", ...] 手动加上属性名
```

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')
```

```
df.to_csv("tzzs_data.csv") 将df保存为csv文件
```

Data Frames 属性

df.attribute	描述	
shape	返回一个元组，表示数据每个维度大小（几行几列）	和ndarray的对应属性一样
ndim	维度（如果是表格则等于2）	
size	所有元素的数目	
dtypes	每一列 的数据类型	类似ndarray的dtype属性
columns	每列对应的名称	

Data Frame 数据类型

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

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

```
In [5]: #查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（一行代表一个记录）？
- ✓ 一共有多少个元素？
- ✓ 每列的名字？

Data Frames 方法

与属性不同, python中对象的方法有括号

ndarray 转成dataframe
pd.DataFrame(a) #a为ndarray类型

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
info()	完整摘要, 包括所有列及其数据类型、每列中非空值的数量, 以及索引范围
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

- ✓ 尝试列出前面的 10 个记录？最后3个记录？
- ✓ 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

快速查看数据集基本信息

- info()

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 6462 entries, 0 to 6461
```

```
Data columns (total 27 columns):
```

#	Column	Non-Null Count	Dtype
0	Cust_No	6462 non-null	int64
1	Target	6462 non-null	int64
2	Birth_Place	6462 non-null	int64
3	Gender	6462 non-null	int64
4	Age	6462 non-null	int64
5	Marriage_State	4465 non-null	float64
6	Work_Years	6462 non-null	int64
7	Unit_Kind	2445 non-null	object
8	Title	2767 non-null	float64
9	Year_Income	6462 non-null	float64
10	Couple_Year_Income	2008 non-null	float64
11	L12_Month_Pay_Amount	6462 non-null	float64
12	Ast_Curr_Bal	6462 non-null	float64
13	Std_Cred_Limit	6462 non-null	int64
14	Loan_Curr_Bal	6462 non-null	float64
15	ZX_Max_Account_Number	6462 non-null	int64
16	ZX_Max_Link_Banks	6462 non-null	int64
17	ZX_Max_Overdue_Account	6462 non-null	int64
18	ZX_Link_Max_Overdue_Amount	6462 non-null	int64
19	ZX_Total_Overdu_Months	6462 non-null	int64
20	ZX_Max_Overdue_Duration	6462 non-null	int64
21	ZX_Max_Credits	6462 non-null	int64
22	ZX_Max_Credit_Banks	6462 non-null	int64
23	ZX_Max_Overdue_Credits	6462 non-null	int64

查看不同类型属性的统计信息

.describe()给出数值列基本统计信息

数值列是指数据类型为float, int类型的列，不一定是连续属性。如果类别属性被表示成int类型，则其取值也被当做数值型。

```
df['Year_Income'].describe()
```

```
count    6.462000e+03
mean     6.445621e+05
std      2.728305e+07
min      0.000000e+00
25%      6.000000e+04
50%      1.000000e+05
75%      2.000000e+05
max      1.600000e+09
Name: Year_Income, dtype: float64
```

.value_counts()给出每个取值以及
对应该取值的样本数

```
df['Unit_Kind'].value_counts()
```

```
1000    2096
1175     208
9999     70
2000     42
1172     18
1120      8
ae        1
Be        1
Da        1
Name: Unit_Kind, dtype: int64
```

Data Frame: 过滤 (filtering)

用Boolean索引进行数据子集提取，也叫过滤，与之前NumPy中ndarray类似。

```
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)

和ndarray类似，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

如果想选择多列且返回Data Frame对象，用列名，且*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, 返回Series对象:  
df['salary']  
#选择salary, 返回DataFrame对象:  
df[['salary']]
```

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 (与ndarray的下标类似)

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

```
df.iloc[:, 0]  # 第1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
```


列分割与拼接

若数据集df的 'Target' 列代表类别，把df分成数据和标签：

```
X=df.drop('Target', axis=1)  
y=df['Target']
```

把数据和标签拼接起来

```
data_label=pd.concat([X,y], axis=1)
```

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()
```

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

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

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

ascending=True表示升序，False表示降序

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()
```

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

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

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。

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 `pd.DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False,)`

处理data frame中缺失值的方法，用 `pd.DataFrame.dropna`?查看详细例子

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

基于Python的数据分析

- Python简介、安装和编程环境
- 高性能计算NumPy
- 数据导入和预处理pandas
- 机器学习库scikit-learn
- 画图和可视化matplotlib

scikit-learn中机器学习算法：监督、无监督

1. Supervised learning

- 1.1. Linear Models
- 1.2. Linear and Quadratic Discriminant Analysis
- 1.3. Kernel ridge regression
- 1.4. Support Vector Machines
- 1.5. Stochastic Gradient Descent
- 1.6. Nearest Neighbors
- 1.7. Gaussian Processes
- 1.8. Cross decomposition
- 1.9. Naive Bayes
- 1.10. Decision Trees
- 1.11. Ensemble methods
- 1.12. Multiclass and multioutput algorithms
- 1.13. Feature selection

2. Unsupervised learning

- 2.1. Gaussian mixture models
- 2.2. Manifold learning
- 2.3. Clustering
- 2.4. Biclustering
- 2.5. Decomposing signals in components (matrix factorization problems)
- 2.6. Covariance estimation
- 2.7. Novelty and Outlier Detection
- 2.8. Density Estimation
- 2.9. Neural network models (unsupervised)

算法中的`.fit()` 和 `.predict()`方法

建模/模型训练：`.fit()`

基于训练集 (`X_train`, `y_train`) 建立分类模型,
其中`X_train`为数据矩阵, `y_train`为每个训练样本的标签

预测：`.predict(X_test)`

用以上模型预测给定样本的标签

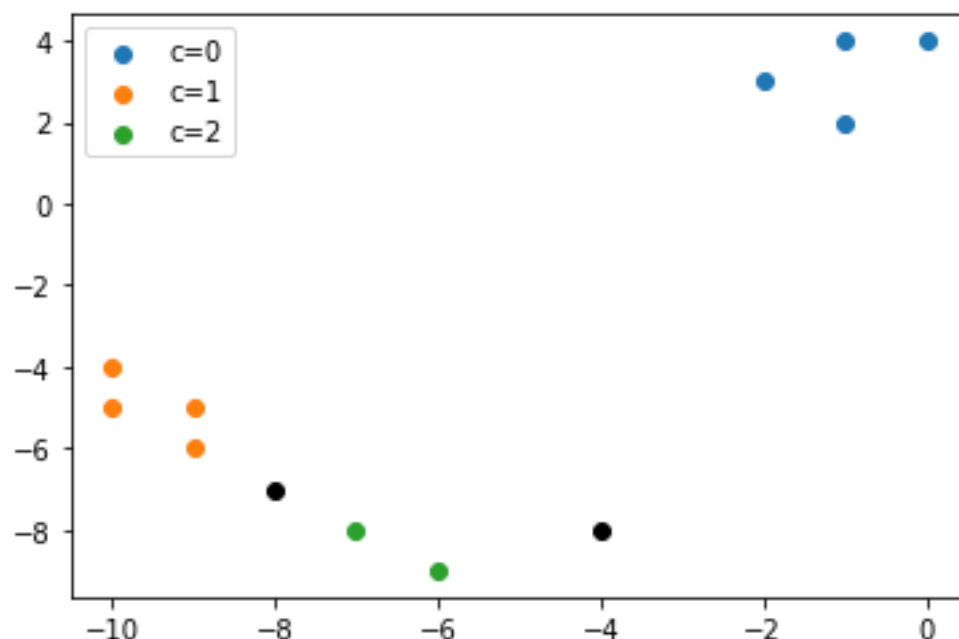
[sklearn.tree](#).DecisionTreeClassifier

[具体用法](#)


```
X=np.array([[ -2.,  3.],
             [  0.,  4.],
             [-10., -5.],
             [ -6., -9.],
             [ -9., -6.],
             [-10., -4.],
             [ -1.,  4.],
             [ -1.,  2.],
             [ -9., -5.],
             [ -7., -8.],
             [ -8., -7.],
             [ -4., -8.]])
```

```
X_train=X[:-2]
y_train=y[:-2]
X_test=X[-2:]
y_test=y[-2:]
```

```
y=np.array([0, 0, 1, 2, 1, 1, 0, 0, 1, 2, 2, 2])
```



In [109]: `from sklearn import tree`

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train, y_train)
y_p=clf.predict(X_test)
y_p
```

Out[109]: `array([1, 2])`

scikit-learn模型选择和评估

3. Model selection and evaluation

- ▶ [3.1. Cross-validation: evaluating estimator performance](#)
- ▶ 3.2. Tuning the hyper-parameters of an estimator
- ▶ 3.3. Metrics and scoring: quantifying the quality of predictions
- ▶ 3.4. Validation curves: plotting scores to evaluate models

```
In [114]: ▶ from sklearn.metrics import accuracy_score  
accuracy_score(y_test, y_p)
```

```
Out[114]: 0.5
```

从sklearn中导入数据集

7. Dataset loading utilities

- 7.1. Toy datasets
- 7.2. Real world datasets
- 7.3. Generated datasets
- 7.4. Loading other datasets



通过[sklearn.datasets](#)获得数据集

`load_boston(*[, return_X_y])`

Load and return the boston house-prices dataset (regression).

`load_iris(*[, return_X_y, as_frame])`

Load and return the iris dataset (classification).

`load_diabetes(*[, return_X_y, as_frame])`

Load and return the diabetes dataset (regression).

`load_digits(*[, n_class, return_X_y, as_frame])`

Load and return the digits dataset (classification).

`load_linnerud(*[, return_X_y, as_frame])`

Load and return the physical exercise linnerud dataset.

`load_wine(*[, return_X_y, as_frame])`

Load and return the wine dataset (classification).

`load_breast_cancer(*[, return_X_y, as_frame])`

Load and return the breast cancer wisconsin dataset (classification).

从sklearn内置数据集中导入

导入iris数据集

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

```
# 装载内置数据集
from sklearn.datasets import load_iris
#把参数return_X_y设为True时，只装载数据和标签
X, y = load_iris(return_X_y=True)
```

```
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]])
```

```
In [177]: irisdata.feature_names
```

```
Out[177]:
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
iris.target_names
```

```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

把数据和标签合并成一个矩阵用DataFrame类型存储

```
import pandas as pd

X=pd.DataFrame(iris.data,columns=iris.feature_names)
y=pd.DataFrame(iris.target,columns=['label'])
data_label=pd.concat([X, y], axis=1)
data_label.head()
```

pd.cancat()方法把两个表拼接起来。axis=1时横向拼接。

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	label
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

基于Python的数据分析

- Python简介、安装和编程环境
- 高性能计算NumPy
- 数据导入和预处理pandas
- 机器学习库scikit-learn
- 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
```

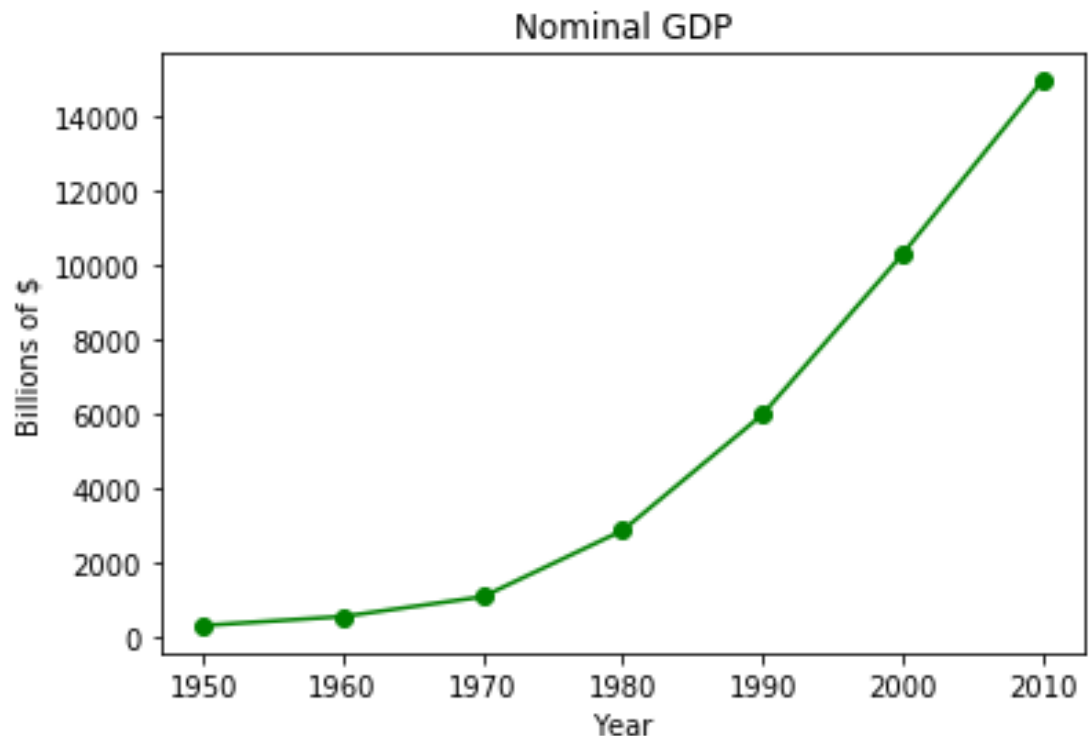
```
plt.ylabel("Billions of $")
```

```
# 在x轴添加说明
```

```
plt.xlabel("Year")
```

```
plt.show()
```

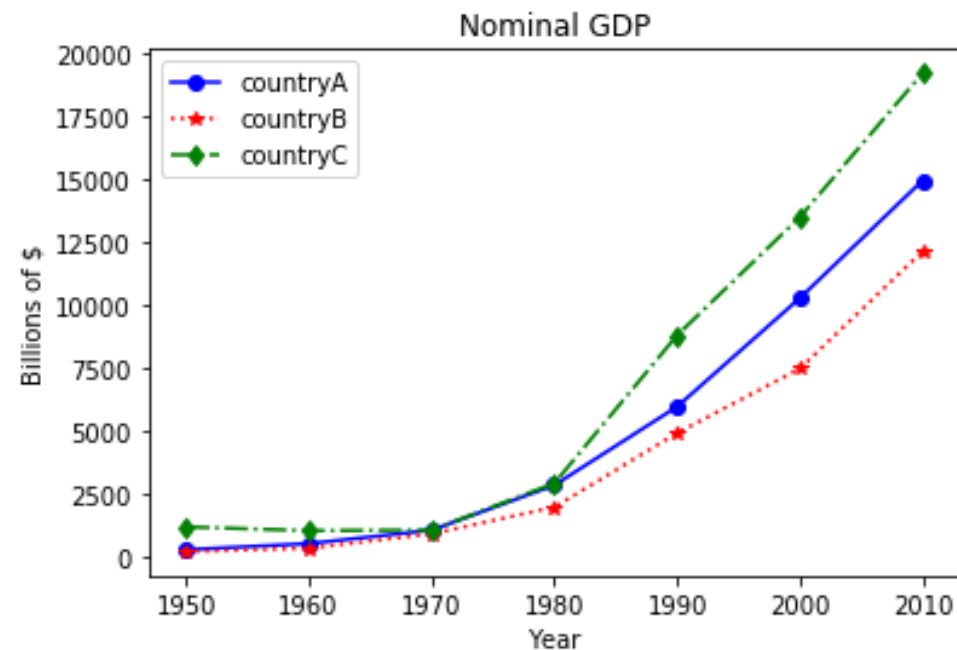
输入 plt.plot? 查看更多用法



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=[0,1,2,3,4]
```

```
# 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)
```

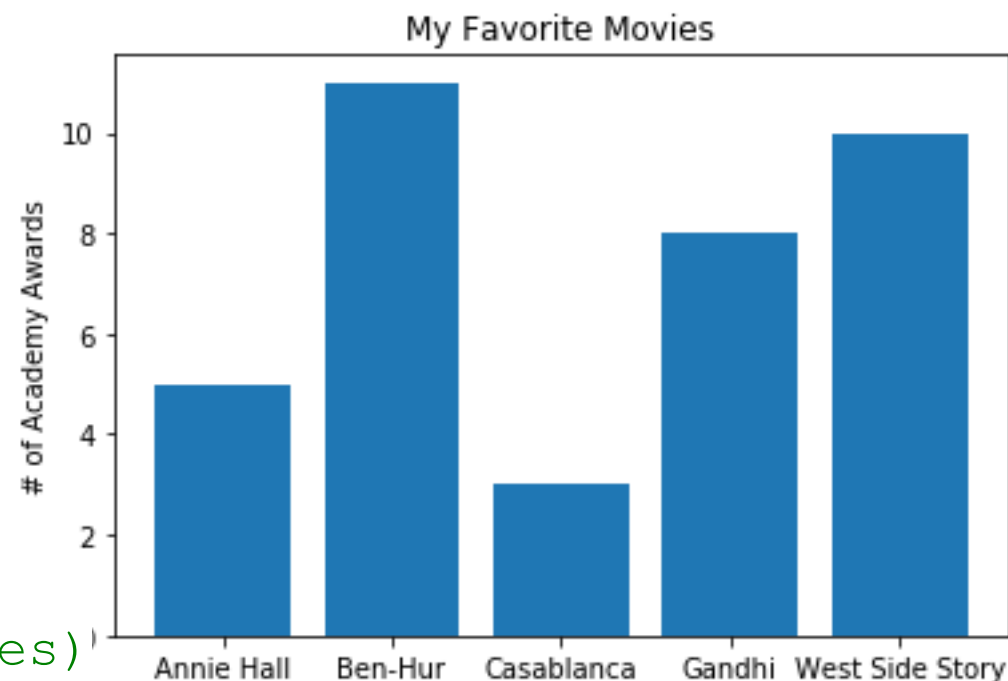
```
# 或者用以下代码代替上面两行
```

```
#plt.bar(xs, num_oscars, tick_label=movies)
```

```
plt.ylabel("# of Academy Awards")
```

```
plt.title("My Favorite Movies")
```

```
plt.show()
```

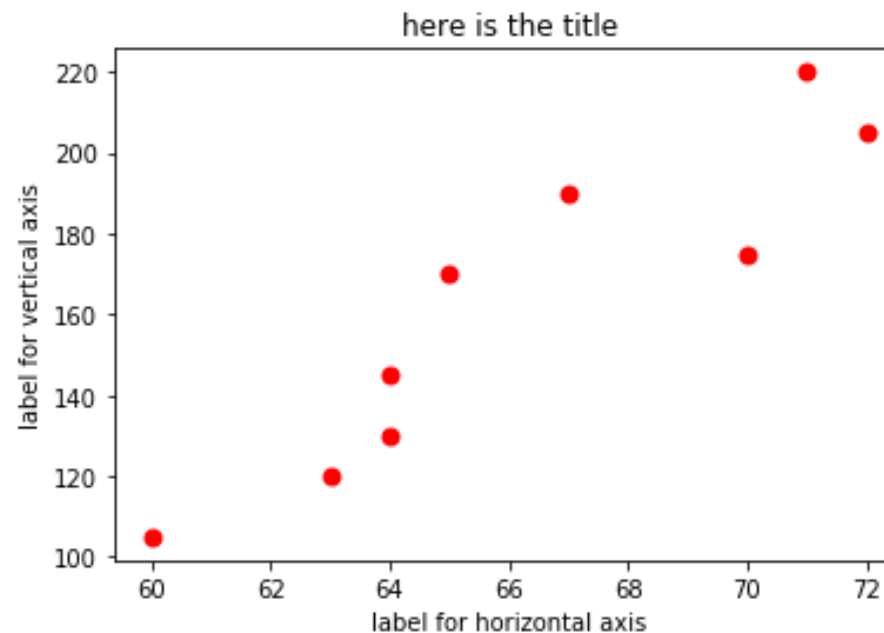


散点图：对2维数据分布进行可视化

```
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()
```

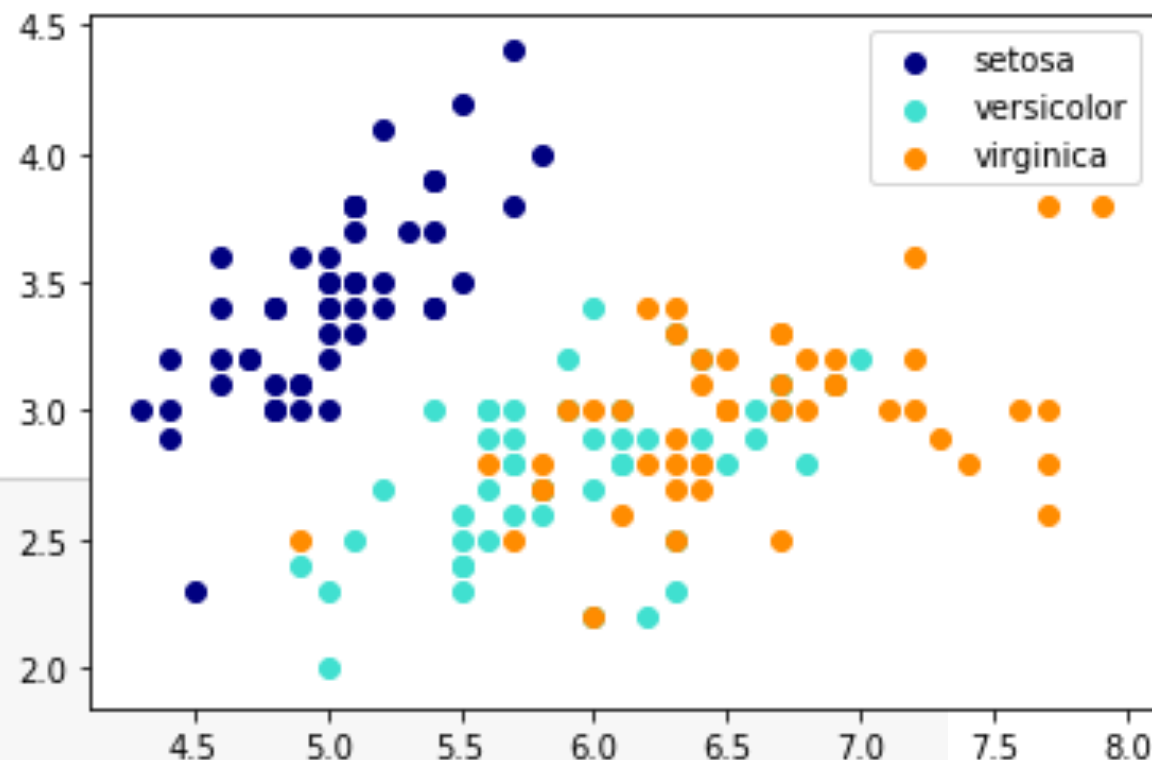
lw表示线段宽度或者marker的大小



散点图：对iris数据进行可视化

#画散点图

```
import matplotlib.pyplot as plt
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()
```



拓展：matplotlib的进阶版seaborn

- 可以画出很多美图！

matplotlib包含在anaconda里面，但是seaborn需要额外安装。

seaborn: statistical data visualization

