机器学习升级版

实验手册

小象学院 邹博

2016年8月22日

说明：

1、该实验手册尚未完全稳定完善，内容和用词都存在诸多问题。，若发现错误，烦请您不吝指出。

2、当前文档只包含了第一章和第九章，其他章节将随着课程的进行逐步分享。

3、若希望参与本手册的编写工作的，欢迎联系我。

我的邮箱：[visio@163.com](mailto:visio@163.com)。

# 目录

[目录 2](#_Toc457837858)

[第一章 软件包的下载、安装及配置 7](#_Toc457837859)

[1.1软件包的下载准备 7](#_Toc457837860)

[1.1.1 Anaconda 下载 7](#_Toc457837861)

[1.1.2 PyCharm编辑器下载 11](#_Toc457837862)

[1.2 windows下安装步骤 12](#_Toc457837863)

[1.2.1 安装python科学计算Anaconda 12](#_Toc457837864)

[1.2.2 验证环境配置 17](#_Toc457837865)

[1.2.3 PyCharm编辑器的安装 17](#_Toc457837866)

[1.2.4 PyCharm配置 21](#_Toc457837867)

[1.3 Mac OS X下安装步骤 24](#_Toc457837868)

[1.3.1 安装python科学计算Anaconda 24](#_Toc457837869)

[1.3.2 验证环境配置 28](#_Toc457837870)

[1.3.3 PyCharm编辑器的安装及配置 29](#_Toc457837871)

[1.4 Linux下安装步骤（以ubuntu系统为例） 33](#_Toc457837872)

[1.4.1 安装python科学计算Anaconda 33](#_Toc457837873)

[1.4.2 验证环境配置 35](#_Toc457837874)

[1.4.3 PyCharm编辑器的安装及配置 35](#_Toc457837875)

[第二章 回归 41](#_Toc457837876)

[2.1 线性回归 41](#_Toc457837877)

[2.1.1 实验数据 41](#_Toc457837878)

[2.1.2 实验过程: 42](#_Toc457837879)

[2.1.3结果分析： 51](#_Toc457837880)

[2.1.4注意事项： 52](#_Toc457837881)

[2.2 Logistic回归 53](#_Toc457837882)

[2.2.1 实验数据: 53](#_Toc457837883)

[2.2.2 实验过程: 63](#_Toc457837884)

[2.2.3 结果分析： 65](#_Toc457837885)

[2.2.4 注意事项: 66](#_Toc457837886)

[第三章 决策树和随机森林 68](#_Toc457837887)

[3.1实验数据 68](#_Toc457837888)

[3.2特性和思想 69](#_Toc457837889)

[3.3实验过程 70](#_Toc457837890)

[3.3.1决策树实现 70](#_Toc457837891)

[3.3.2 Matplotlib绘制树形图 71](#_Toc457837892)

[3.3.3使用决策树分类 73](#_Toc457837893)

[3.3.4使用随机生成的数据集构造可视化决策树 74](#_Toc457837894)

[3.3.5通过真实汽车数据构造实际可视化决策树 74](#_Toc457837895)

[3.3.6检验生成的决策树性能 75](#_Toc457837896)

[3.4实验结果 75](#_Toc457837897)

[3.4.1初步结果 75](#_Toc457837898)

[3.4.2改进结果 77](#_Toc457837899)

[3.5实验总结 78](#_Toc457837900)

[第四章 支持向量机SVM 79](#_Toc457837901)

[4.1 实验数据 79](#_Toc457837902)

[4.2 LIBSVM简介 84](#_Toc457837903)

[4.3 LIBSVM在当前实验数据的调用过程 84](#_Toc457837904)

[4.4 实验效果分析 88](#_Toc457837905)

[第五章 聚类 89](#_Toc457837906)

[5.1实验概要 89](#_Toc457837907)

[5.2实验输入描述 89](#_Toc457837908)

[5.3 实验步骤 90](#_Toc457837909)

[5.4 评价标准 91](#_Toc457837910)

[5.4.1 Adjusted Rand Index（ARI） 91](#_Toc457837911)

[5.4.2 Homogeneity（同质性） 92](#_Toc457837912)

[5.4.3 Completeness（完整性） 92](#_Toc457837913)

[5.5 实验结果及分析 92](#_Toc457837914)

[第六章 EM算法 93](#_Toc457837915)

[6.1实验概要 93](#_Toc457837916)

[6.2实验输入描述 93](#_Toc457837917)

[6.3 实验步骤 94](#_Toc457837918)

[6.4 实验结果及分析 95](#_Toc457837919)

[第七章 LDA 97](#_Toc457837920)

[7.1目标任务 97](#_Toc457837921)

[7.2 实验环境 97](#_Toc457837922)

[7.3 实验数据 97](#_Toc457837923)

[数据集一：网易新闻 98](#_Toc457837924)

[数据集二：搜狗新闻 100](#_Toc457837925)

[7.4 实验设计 102](#_Toc457837926)

[7.5 实验过程 103](#_Toc457837927)

[7.5.1 数据的输入输出： 103](#_Toc457837928)

[7.5.2 实验步骤 104](#_Toc457837929)

[实验准备：配置环境 104](#_Toc457837930)

[实验一：测试数据集的主题分布 106](#_Toc457837931)

[实验二：测试LDA参数的改变对主题分布的影响 109](#_Toc457837932)

[7.6 附：Java版LDA的使用和结果 111](#_Toc457837933)

[7.6.1 原始数据： 111](#_Toc457837934)

[7.6.2 分词处理： 112](#_Toc457837935)

[7.6.3 LDA主题分类 113](#_Toc457837936)

[第八章 隐马尔科夫模型HMM 116](#_Toc457837937)

[8.1 目标任务： 116](#_Toc457837938)

[8.2实验数据: 116](#_Toc457837939)

[8.3 实验过程： 116](#_Toc457837940)

[8.3.1 实验准备：相关库的安装 116](#_Toc457837941)

[8.3.2 数据的输入输出: 128](#_Toc457837942)

[8.3.3 实验步骤: 129](#_Toc457837943)

[8.4 结果分析 135](#_Toc457837944)

[第9章 Python数值计算与机器学习常见库 137](#_Toc457837945)

[9.1 Numpy 137](#_Toc457837946)

[9.1.1总体说明: 137](#_Toc457837947)

[9.1.2 代表性函数使用介绍 137](#_Toc457837948)

[9.2 Scipy库 143](#_Toc457837949)

[9.2.1 总体说明 143](#_Toc457837950)

[9.2.2 代表性函数使用介绍 143](#_Toc457837951)

[9.3 Pandas库 159](#_Toc457837952)

[9.3.1 pandas库总体说明 159](#_Toc457837953)

[9.3.2 代表性函数的使用介绍: 161](#_Toc457837954)

[9.4 机器学习包sk-learn 181](#_Toc457837955)

[9.4.1 总体说明 181](#_Toc457837956)

[9.4.2 代表性函数使用介绍 181](#_Toc457837957)

[9.4.3 机器学习算法的使用 182](#_Toc457837958)

[9.4.4 如何优化算法参数 187](#_Toc457837959)

# 第一章 软件包的下载、安装及配置

**目标任务：**

1、安装python解释器

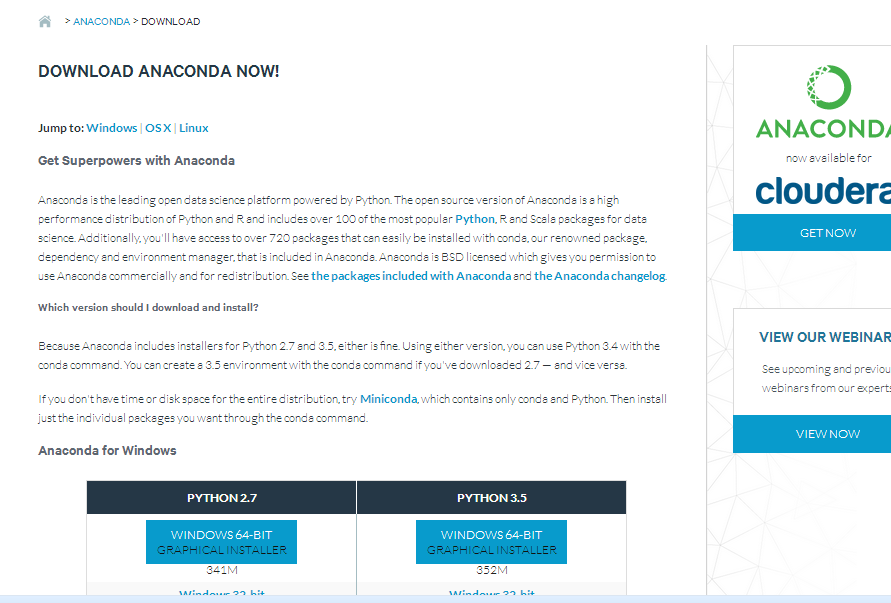
2、配置环境

3、安装pycharm编辑环境IDE

## 1.1软件包的下载准备

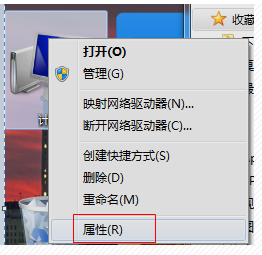
### 1.1.1 Anaconda 下载

Anaconda作为出色的编辑环境，除了提供Python解释器，还集成了python科学计算的各种包。下载Anaconda的地址为：<http://continuum.io/downloads>。页面如下：



根据自己的操作系统选择相应版本下载，Windows/Mac/Linux分别下载，请注意您的操作系统是64位还是32位。

* Tip：若您在使用Windows系统且不清楚操作系统是64位或者32位，可右击桌面上“计算机”点击“属性”。



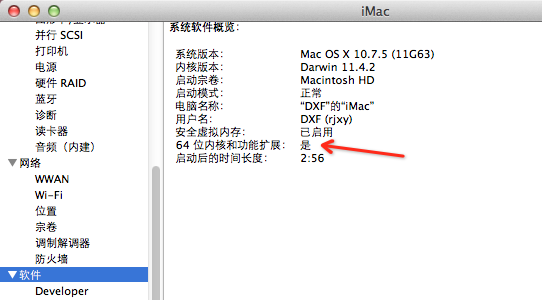
* 打开窗口如下所示，即可查看到计算机系统类型。



* Mac OS X分别下载注意自己的电脑是64位还是32位。（注意：Mac OS X一般为64位，官网上的软件为64位软件我们选择第一个MAC OS X的第一个软件为图形化界面）

Tip：查看计算机系统参数，有两个简单方法：

(1)在工具栏左上角点击（苹果Logo）标志，关于本机🡪更多信息🡪系统报告🡪 (左侧栏中)软件

[](http://d.hiphotos.baidu.com/zhidao/pic/item/314e251f95cad1c85c697e3d783e6709c83d51de.jpg)

(2)打开终端，输入命令 uname -a <回车>

x86\_64 表示系统为64位

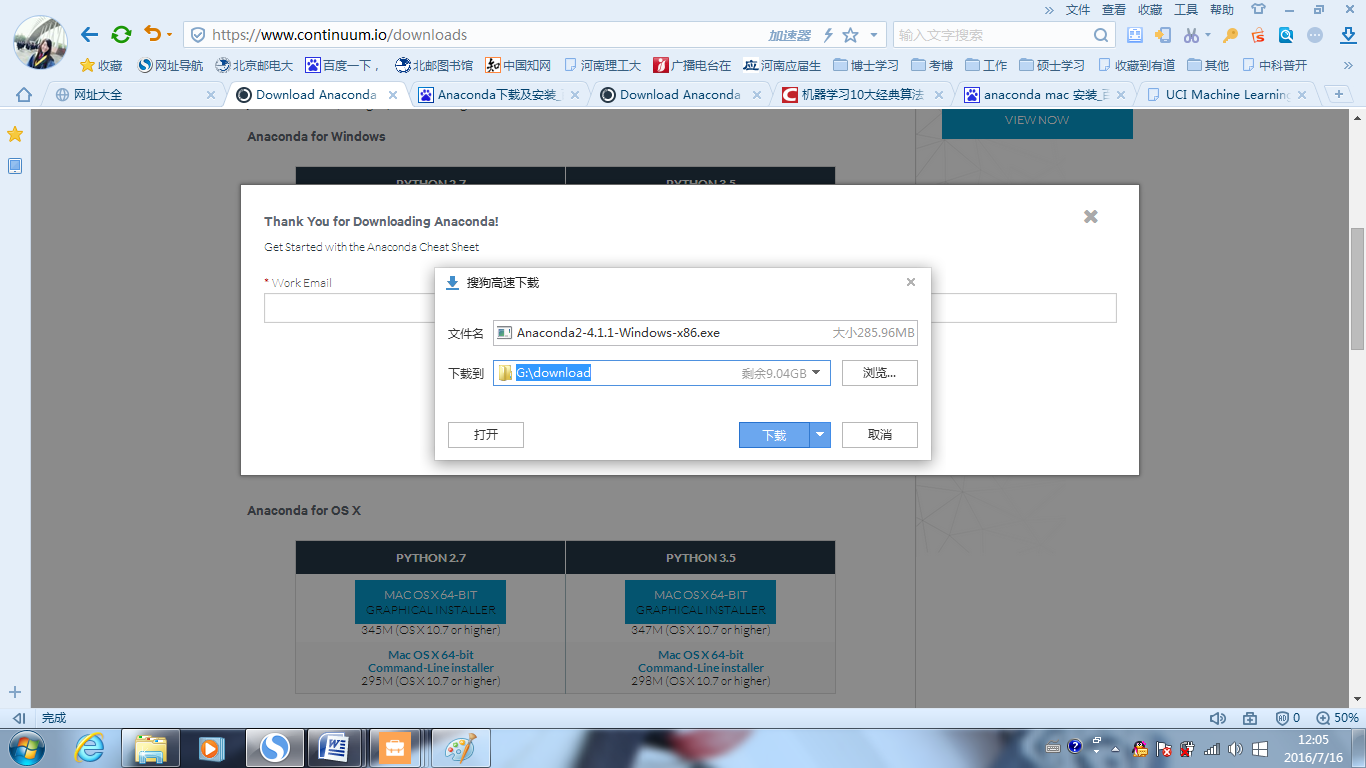
i686 表示系统32位的

* 在<http://continuum.io/downloads>上找到自己的系统相应的版本进行下载。



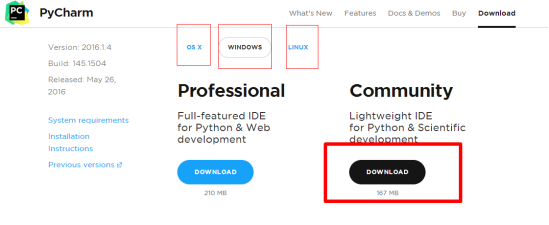


* 选择路径进行下载，下载界面如下。



### 1.1.2 PyCharm编辑器下载

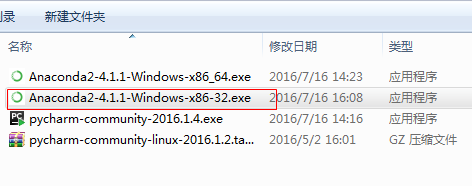
* Anaconda是自带编辑环境的，可以直接在其中做代码编写。但仍然推荐安装PyCharm，因为它在程序调试等方面更加出色。我们只需要安装PyCharm结束的时候，将Python解释器选择Anaconda下的python.exe即可（该步骤的最后部分介绍如何操作）。
* 在[http://www.jetbrains.com/pycharm/download/#section=windows](http://www.jetbrains.com/pycharm/download/" \l "section=windows)上找到与您计算机操作系统相应的版本进行下载。



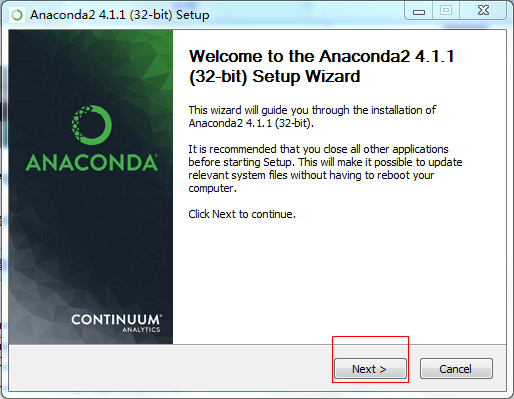
## 1.2 windows下安装步骤

### 1.2.1 安装python科学计算Anaconda

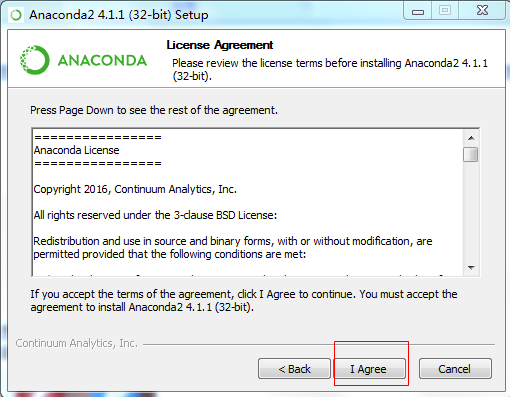
* 下载Anaconda完成后，双击与您计算机系统相对应的软件（这里以win32位安装过程为例）



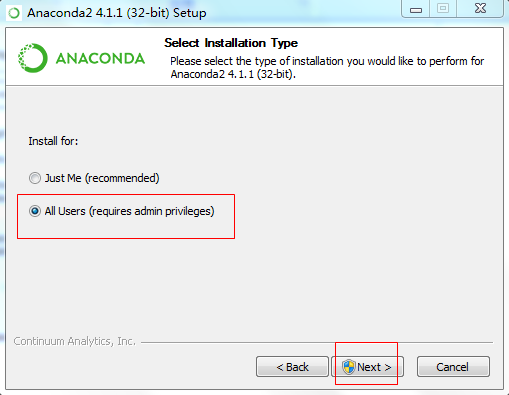
* 打开后对话框界面如下，点击Next按钮。



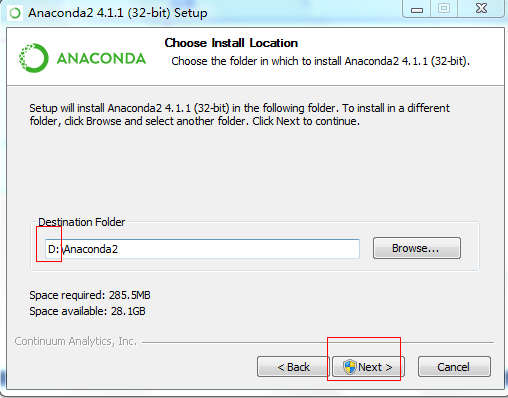
* 点击I Agree 按钮。



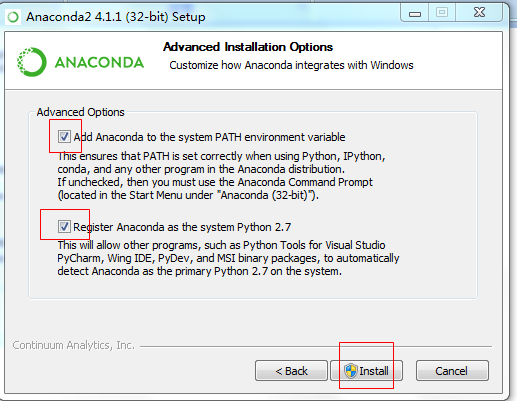
* 单选按钮可以选择All Users(这个是默认选中)，再点击 Next 按钮。



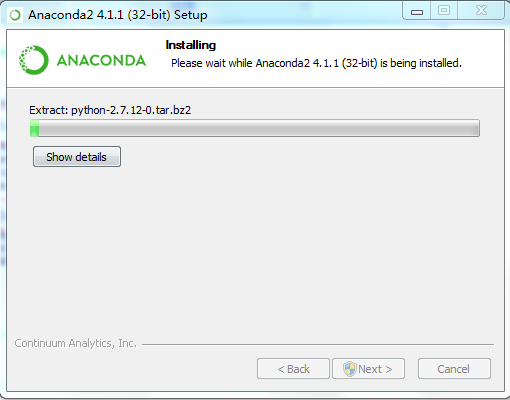
* 更改安装目录到D盘（或任意您方便的安装路径），再点击Next 按钮。



* 确保勾选了复选框“Add Anaconda to the system PATH environment variable”（在系统环境变量path中添加Anaconda路径）和“Register Anaconda as the system Python 2.7”（将Anaconda作为打开Python的默认程序），再点击 Install 按钮。



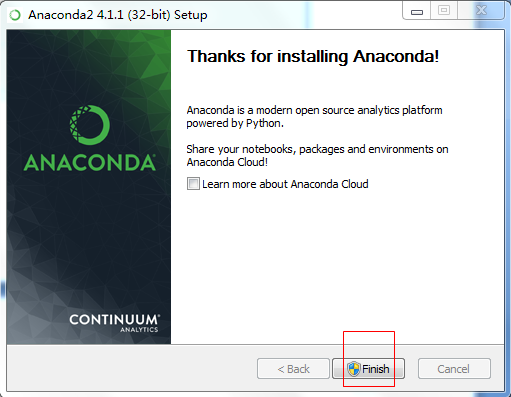
* 安装过程的界面如下。



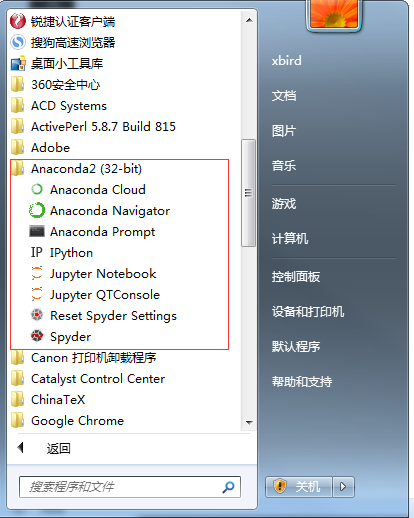
* 安装完成后，点击Next按钮。



* 点击Finish 按钮完成Anaconda的安装。

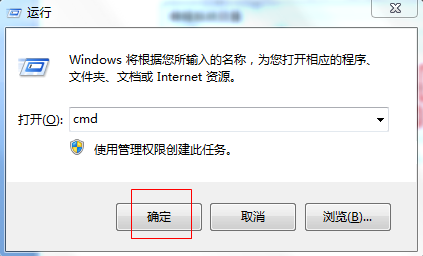


* 在开始菜单中，即可看到安装完成的Anaconda。

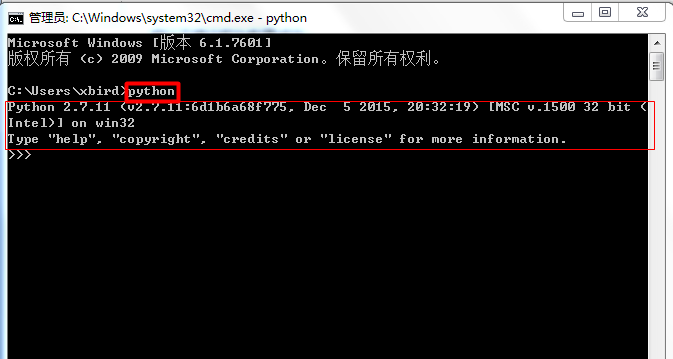


### 1.2.2 验证环境配置

* 验证环境配置：按Windows+R快捷键，并输入cmd打开命令行窗口。

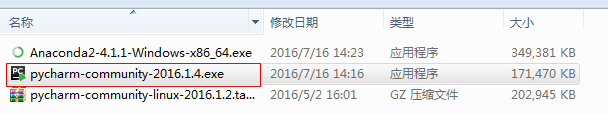


* 输入python，出现以下提示即为配置成功。

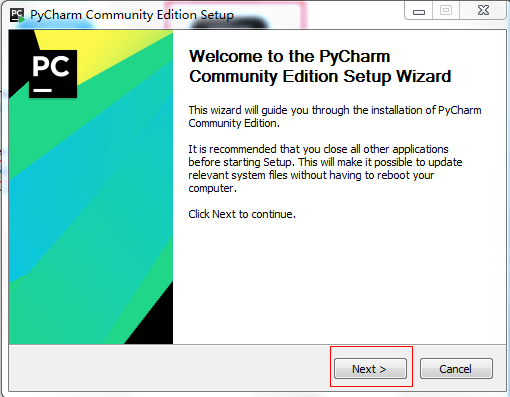


### 1.2.3 PyCharm编辑器的安装

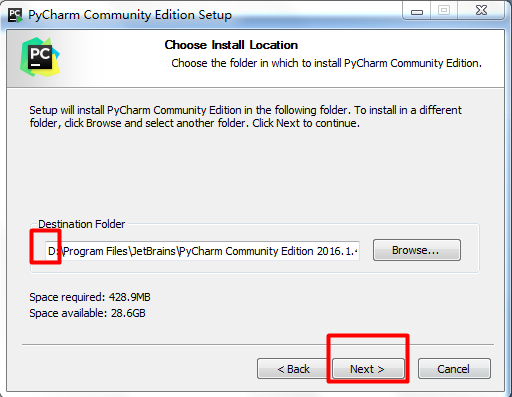
* 双击pycharm-community-2016.1.4.exe文件



* 点击 Next按钮。



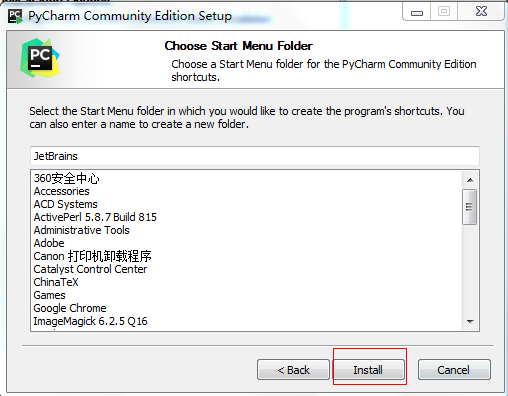
* 更改安装目录到D盘（或任意您方便的安装路径），再点击Next 按钮。



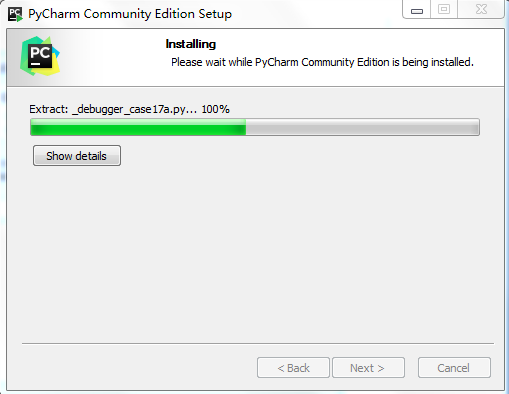
* 确保勾选“桌面创建图标”和“创建与.py文件联系”复选框，并点击Next 按钮。



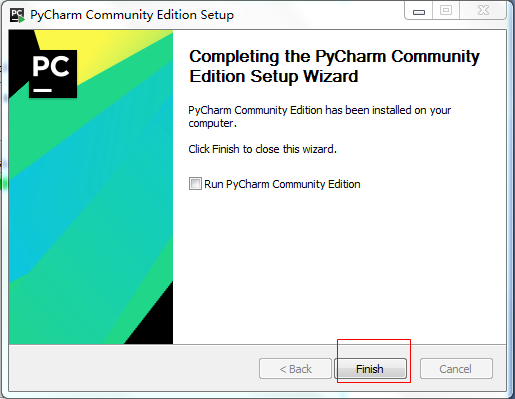
* 点Install按钮。



* 安装过程中的界面如下：

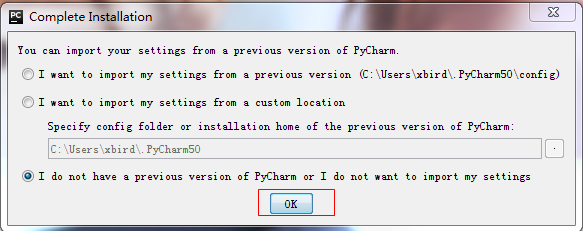


* 点击Finish按钮，安装完成。

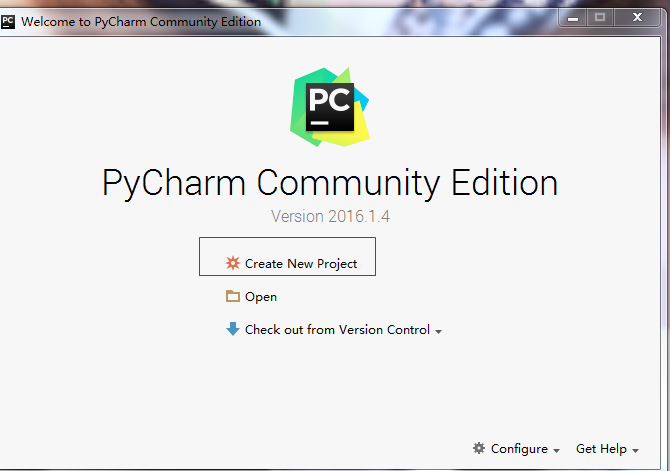


### 1.2.4 PyCharm配置

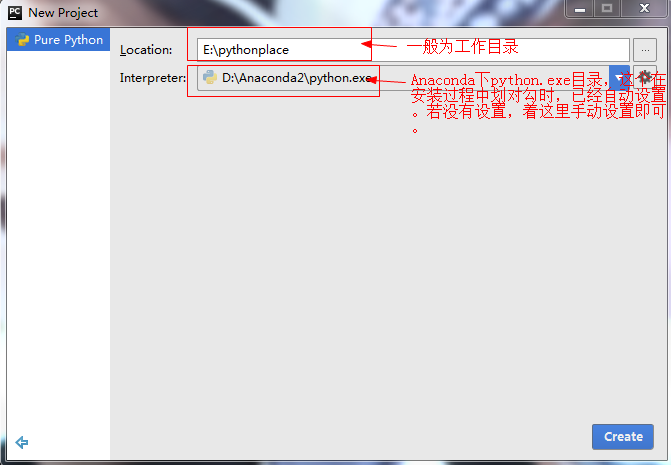
* 初次安装选择按照默认选择最后一个。

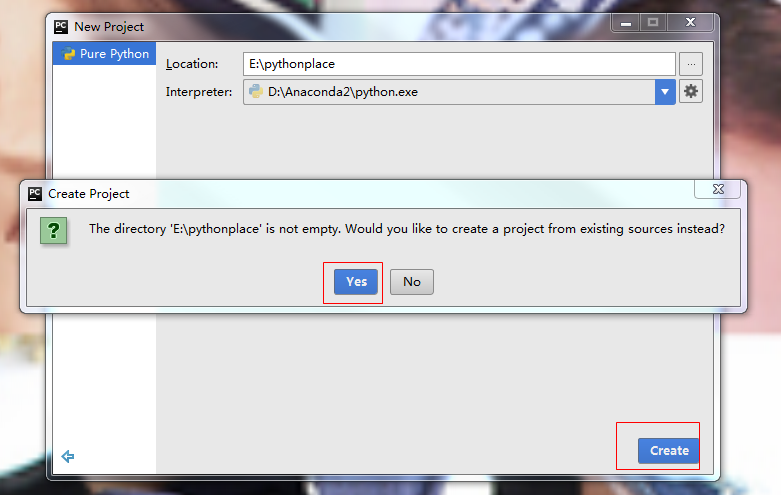
.



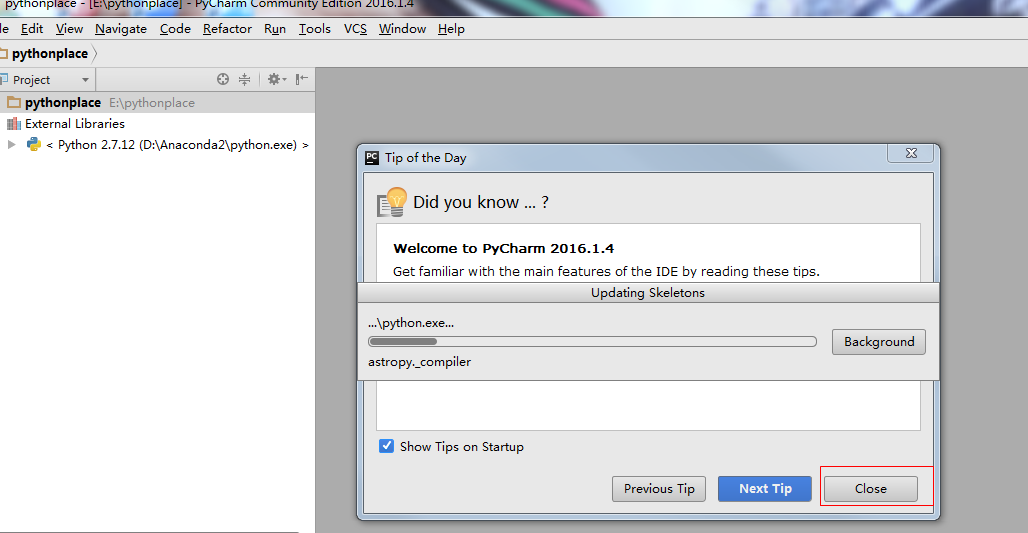


* 设置python解释器的位置





* Updating Skeletons运行完点close即可。

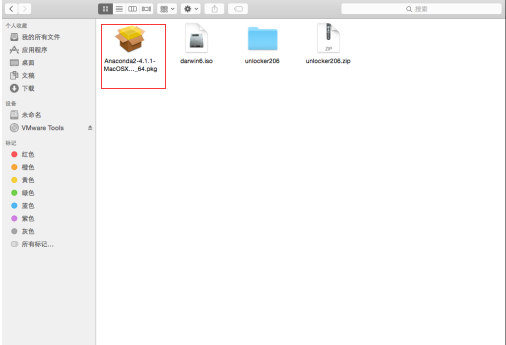


* 然后可以建立python工程，点击run运行即可。

## 1.3 Mac OS X下安装步骤

### 1.3.1 安装python科学计算Anaconda

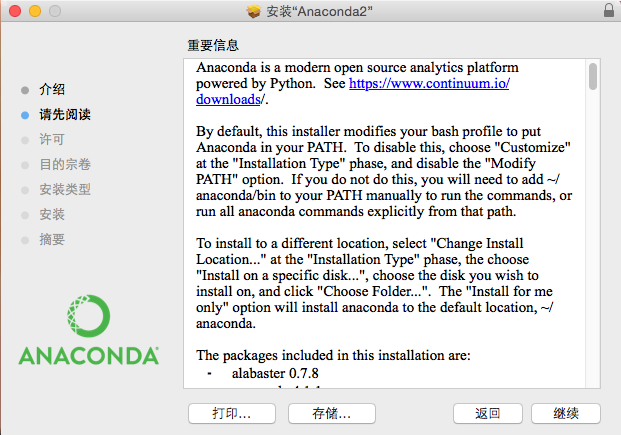
* 下载完成后，双击与您计算机系统相对应的软件

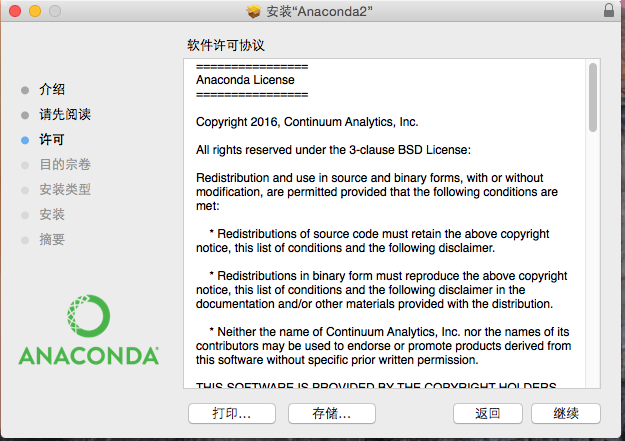


* 打开后对话框界面如下，点击“继续”按钮。



* 一路点击“继续”





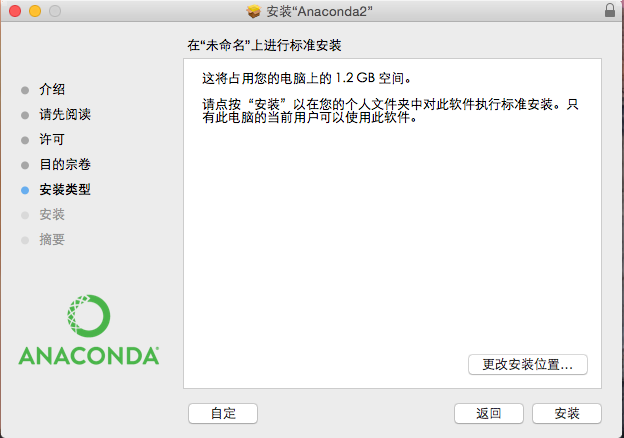
* 点击“同意”。



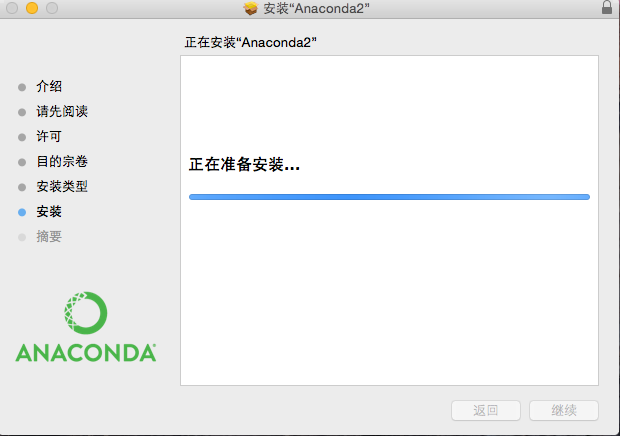
* 选择仅为我安装（也可选择为这台电脑上的所有用户安装），点击“继续”。



* 点击“安装”。



* 安装进行中界面

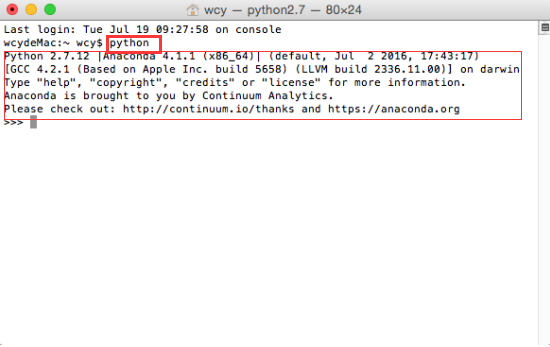


* 安装成功，点击“关闭”按钮。



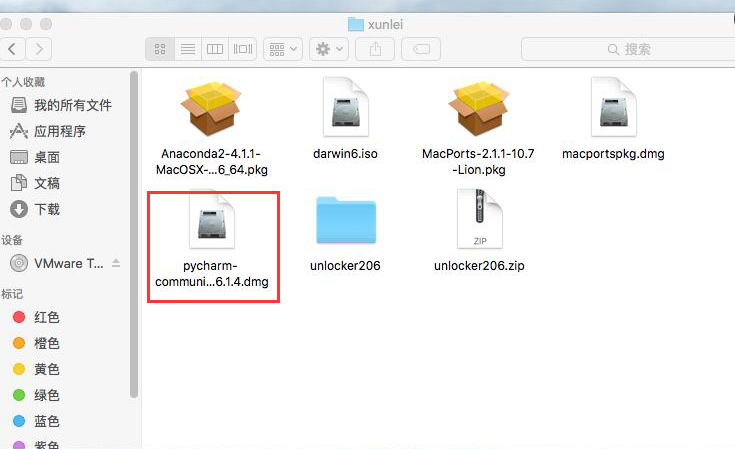
### 1.3.2 验证环境配置

* 打开Mac命令行终端，输入python，出现以下提示即为配置成功。



### 1.3.3 PyCharm编辑器的安装及配置

* 双击相应PyCharm的软件包。



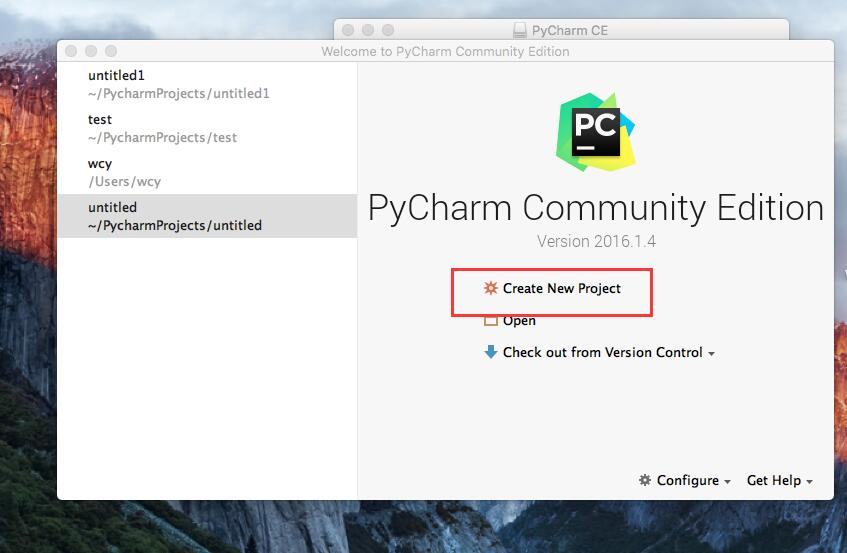
* 安装过程一路默认即可，最后点击“OK”。



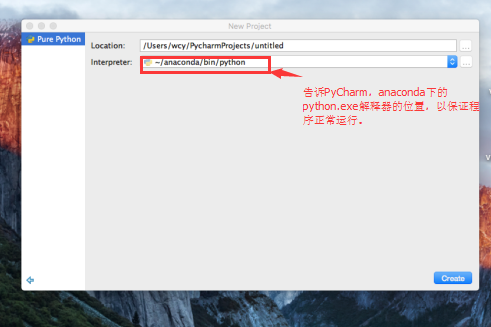
* 出现启动界面



* 打开后出现下面的界面，选择Create New Project ，即可创建一个工程。



* 选择anaconda作为python解释器

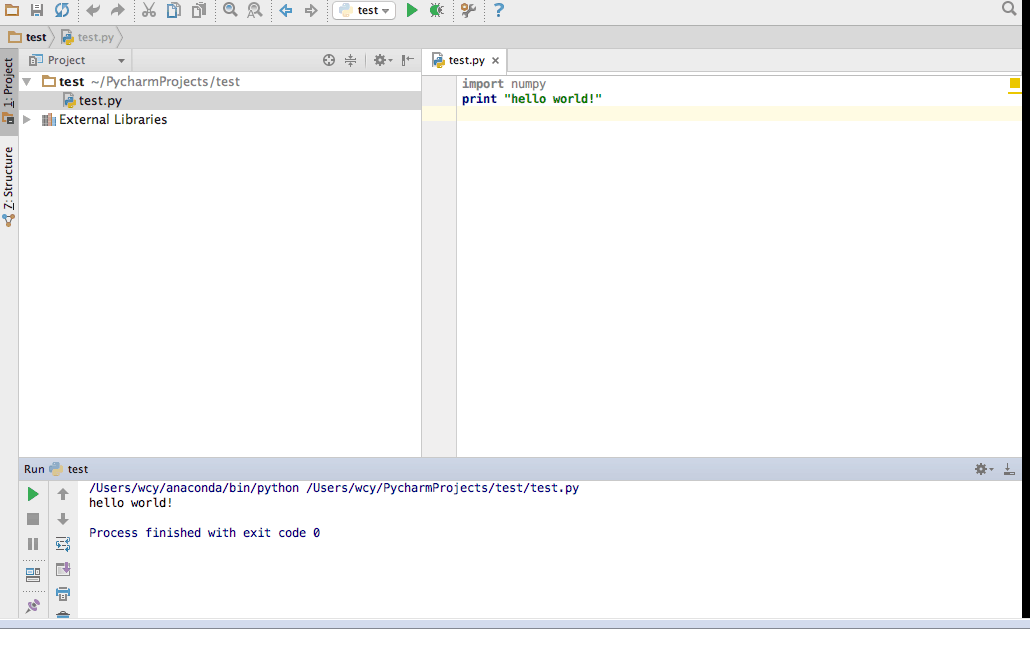


* 在上述工程下新建一个python文件，名字设为test 。



* 打开test.py，

如下图输入以下代码。



* 点击 run 运行后得下面的运行结果，打印出 hello word!

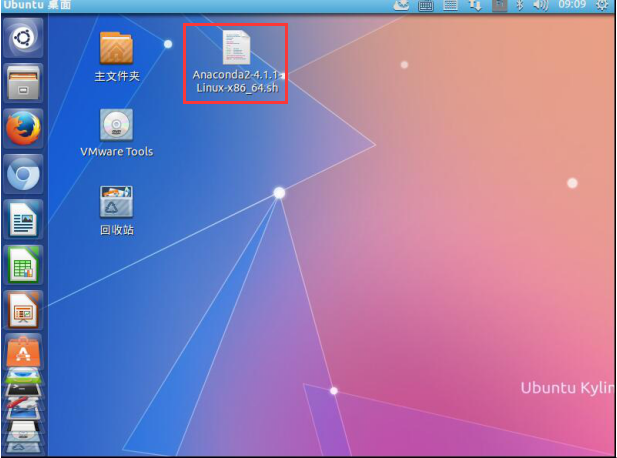


至此安装成功！

## 1.4 Linux下安装步骤（以ubuntu系统为例）

### 1.4.1 安装python科学计算Anaconda

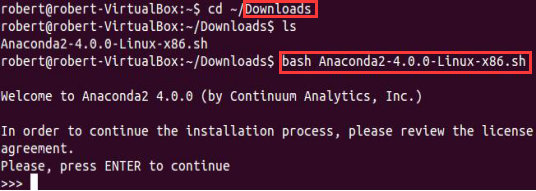
* 下载完成后，双击与您计算机系统相对应的软件。



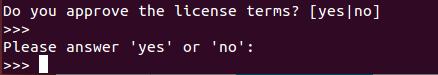
* 打开Terminal，执行以下命令：

bash Anaconda2-4.0.0-Linux-x86.sh

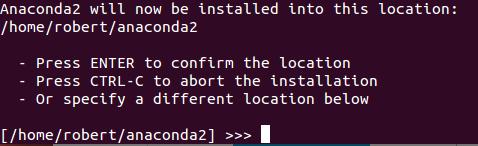
如下图所示：



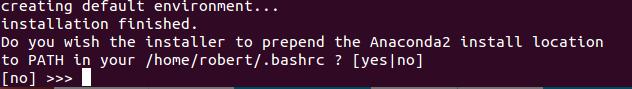
* 在上图上输入Enter，进行浏览license agreement的过程。浏览完毕后，会出现下图，输入yes同意license agreement。



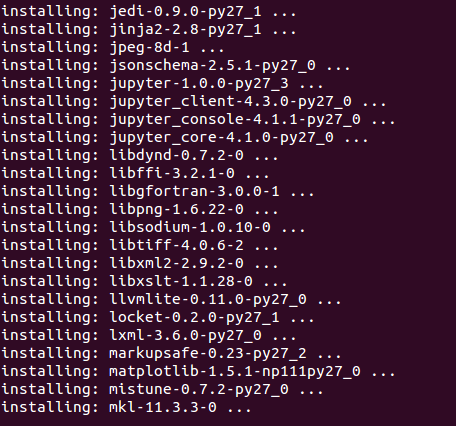
* 会提示要确定安装的位置，这里采用默认位置即可（也可按需选择按照提示操作即可），输入Enter 。



* 会提示你是否将anaconada的路径添加到Path中，这里我们输入yes 。

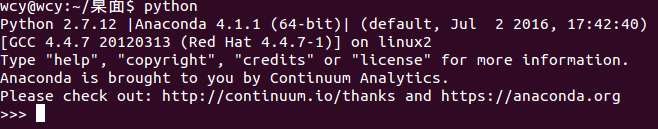


* 安装进行中



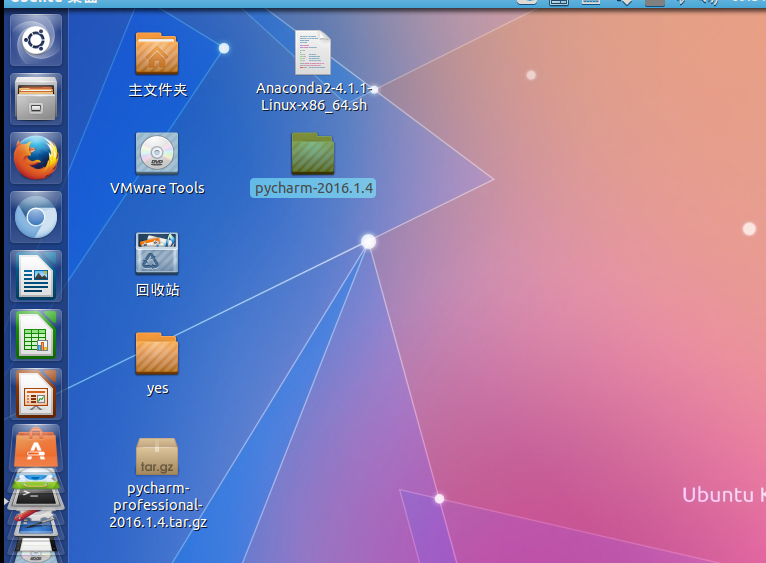
### 1.4.2 验证环境配置

* 安装成功后。重新启动Terminal，我们会发现Python版本（Linux自带python）已经是anaconda的版本了

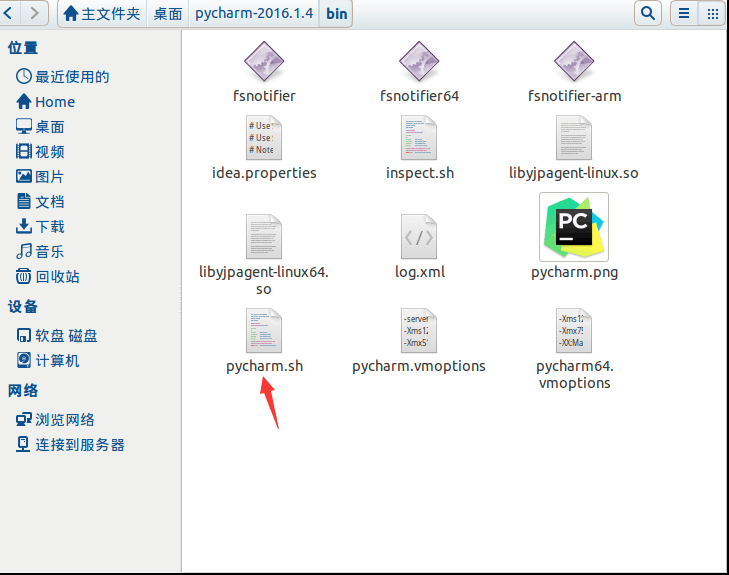


### 1.4.3 PyCharm编辑器的安装及配置

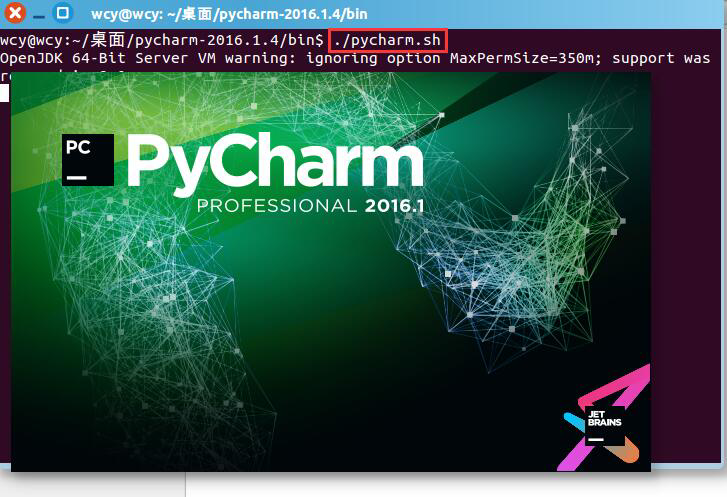
* 下载相应PyCharm的软件包。



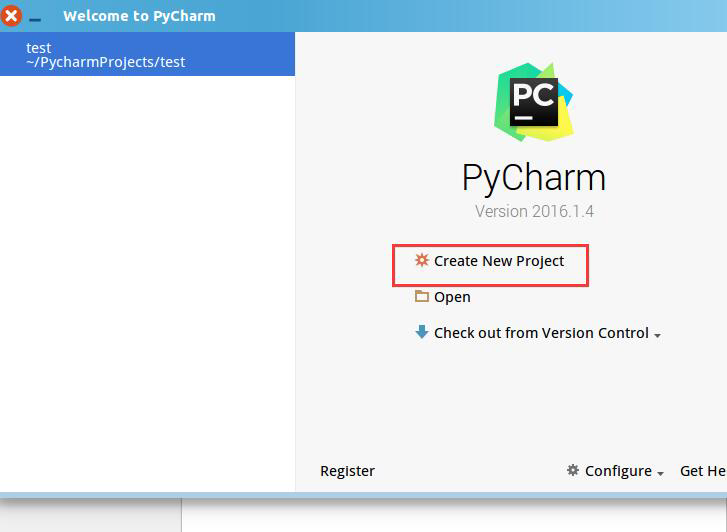
* 进入bin目录，如下图：



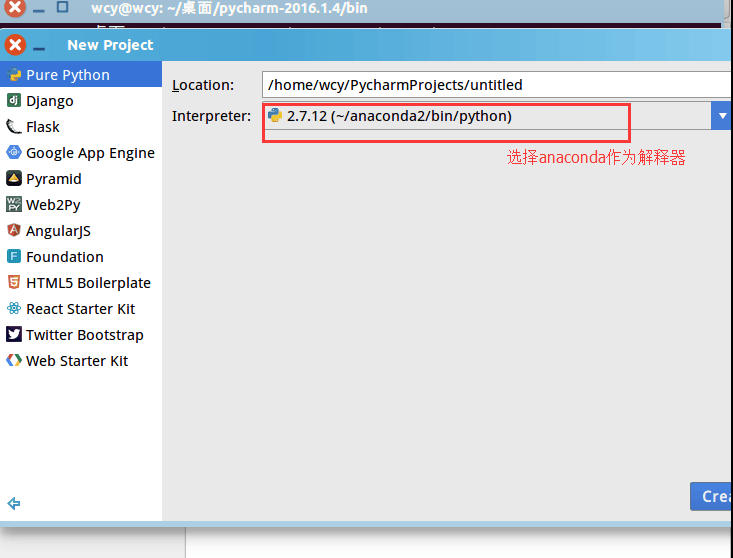
在当前目录下，右击打开终端，输入 ./pycharm.sh如图：



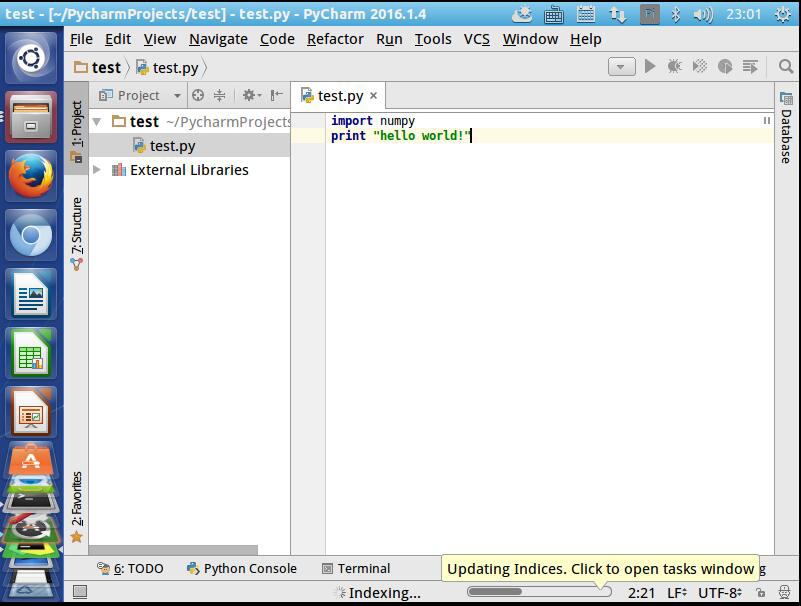
* 创建工程。



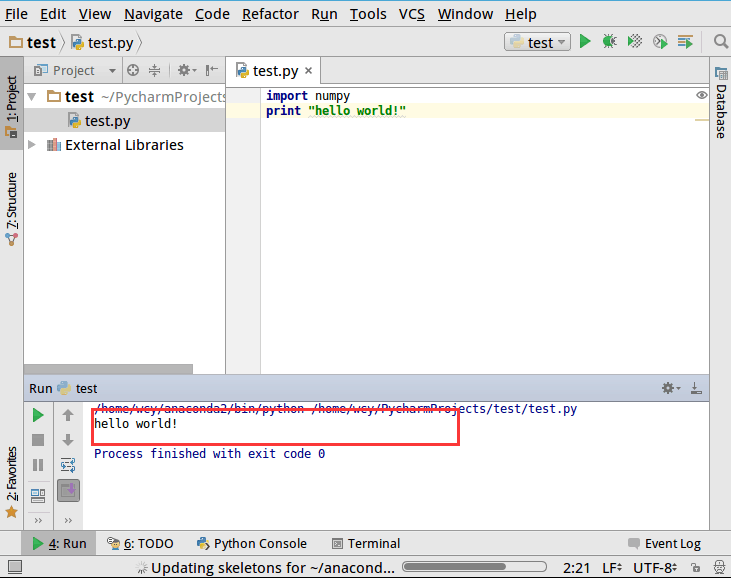
* Create New Project，创建一个测试工程，解释器默认选择了Python2.7。



* 新建一个python文件，键入如下代码：



* 点击run  
  
* 输出结果：



至此成功。

# 第9章 Python数值计算与机器学习常见库

## 9.1 Numpy

### 9.1.1总体说明:

NumPy（Numeric Python）是Python的开源数值计算扩展，它可用来存储和处理大型矩阵，比Python自身的嵌套列表（nested list structure)结构要高效的多。NumPy常常被评价为将Python变成了免费且更强大的MatLab。

NumPy包括了强大的N维数组，比较成熟的函数库，用于整合C/C++和Fortran代码的工具包，以及实用的线性代数、傅里叶变换和随机数生成函数。NumPy和稀疏矩阵运算包SciPy配合使用更加方便。它提供了许多方便的数值编程工具，如：矩阵数据类型、矢量处理，以及精密的运算库。

### 9.1.2 代表性函数使用介绍

**1.numpy数组与python列表效率对比**

import numpy as np

# 创建大小为 10^7 的数组

arr = np.arange(1e7)

larr = arr.tolist()

def list\_times(alist, scalar):

for i, val in enumerate(alist):

alist[i] = val \* scalar

return alist

# 利用 IPython 的魔术方法

timeit 计算运行时间 timeit arr \* 1.1

>>> 1 loops, best of 3: 76.9 ms per loop

timeit list\_times(larr, 1.1)

>>> 1 loops, best of 3: 2.03 s per loop

**2. 创建数组并设置数据类型**

（1）从列表转换

alist = [1, 2, 3]

arr = np.array(alist)

（2）np.arange()

arr = np.arange(100)

arr = np.arange(10,100)

（3）np.zeros()

arr = np.zeros(5)

np.zeros((5,5))

cube = np.zeros((5,5,5)).astype(int) + 1

cube = np.ones((5, 5, 5)).astype(np.float16)

arr = np.zeros(2, dtype=int)

arr = np.zeros(2, dtype=np.float32)

（4）reshape()

arr1d = np.arange(1000)

arr3d = arr1d.reshape((10,10,10))

arr3d = np.reshape(arr1s, (10, 10, 10))

（5）revel()

作用与reshape相反

（6）shape 显示数据对象的形状

arr1d.shape

注意：对数据形状结构的改变只是改变了数据的显示形式，即只是改变了数据的引用， 对一个数据的改变另一个也会被改变。

**3. 记录数组**

（1）创建记录数组并赋值

recarr = np.zeros((2,), dtype=('i4,f4,a10'))

# 创建大小为2的记录数组，类型为4字节整数、4字节浮点数和10字节字符

recarr[:] = [(1,2.,'Hello'),(2,3.,"World")]

（2）使用zip()

recarr = np.zeros((2,), dtype=('i4,f4,a10'))

col1 = np.arange(2) + 1

col2 = np.arange(2, dtype=np.float32)

col3 = ['Hello', 'World']

recarr[:] = zip(col1, col2, col3)

（3）为每列数据命名

recarr.dtype.names = ('Integers' , 'Floats', 'Strings')

（4）使用列名访问数据

recarr('Integers')

**4. 索引和切片**

（1）numpy提供了类似于matlab的索引和切片

alist=[[1,2],[3,4]]

alist[0][1] #python 方式

arr = np.array(alist)

arr[0,1] # 单个元素

arr[:,1] # 第 1 列

arr[1,:] # 第 1 行

（2）np.where()

根据条件获取索引号

index = np.where(arr>2)

new\_arr = arr[index]

new\_arr = np.delete(arr, index)

也可以这样操作：

index = arr > 2 # 得到一个逻辑数组

new\_arr = arr[index]

注意：第二种方法速度更快，而且可以用“∼ index”很容易的得到与index相反的 逻辑数组。

**5. NumPy数组的布尔操作**

NumPy数组元素可以通过逻辑表达式方便的操作

例：

# 创建如 Plot A 所示的数组

img1 = np.zeros((20, 20)) + 3

img1[4:-4, 4:-4] = 6

img1[7:-7, 7:-7] = 9

# 获取数值大于2且小于6的元素索引

index1 = img1 > 2

index2 = img1 < 6

compound\_index = index1 & index2

# 上式与下式结果相同

compound\_index = (img1 > 3) & (img1 < 7)

img2 = np.copy(img1)

img2[compound\_index] = 0

# 得到 Plot B.

# 更复杂的数组逻辑操作

index3 = img1 == 9

index4 = (index1 & index2) | index3 img3 = np.copy(img1)

img3[index4] = 0 # 得到 Plot C.



例：

import numpy.random as rand

a = rand.randn(100)

index = a > 0.2

b = a[index]

b = b \*\* 2 – 2

a[index] = b

**6. 读写操作**

（1）Python读写文本文件

f = open('somefile.txt', 'r') # 以只读方式打开文件， 'r' 表示读

alist = f.readlines() # 将文件内容读入列表，每一行为一个列表元素

file f.close() # 关闭文件

f = open('newtextfile.txt', 'w') # 以可写方式打开文件， 'w' 表示写

f.writelines(newdata) # 写入数据

f.close() # 关闭文件

注意：读写完毕之后要将文件关闭

（2）Numpy文件文件读写

Python读写文件文件虽然方便且效率很好，但是不太适合处理极大的文件。当文件内容有结构，且为数字时用NumPy处理，存numpy.ndarray会更合适。

例：

import numpy as np

arr = np.loadtxt('somefile.txt')

np.savetxt('somenewfile.txt')

如果文件各列数据类型不一样，则需要指明数据类型，NumPy 用来保存数据的类 型为recarray，可以用处理ndarray同样的方法来对元素进行操作。recarray数据类型 不能直接保存为文本文件，如果需要的话可以使用matplotlib.mlab实现。

例：

文件example.txt内容如下

XR21 32.789 1

XR22 33.091 2

读入数据

table = np.loadtxt('example.txt',

dtype='names': ('ID', 'Result', 'Type'),

'formats': ('S4', 'f4', 'i2'))

提示：如果文本数据为ASCII格式的，使用Asciitable包读写会更加高效。

（3）二进制文件

文本文件处理简单方便，但是读写速度和文件大小都不能和二进制文件相比，因此 大数据处理适合使用二进制文件。

例：

import numpy as np

data = np.empty((1000, 1000)) # 创建一个较大的数组

np.save('test.npy', data) # 保存数据

np.savez('test.npz', data) # 压缩保存数据

newdata = np.load('test.npy') # 读入数据

注意：NumPy使用numpy.save和numpy.load来读写二进制文件，但这种二进制文件 只能在NumPy下读写，scipy.io可以处理更通用的二进制文件

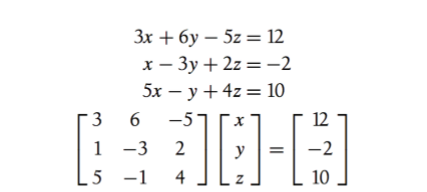
**7. 数学运算**

（1）线性代数

NumPy 数组间的运算只是相对应元素间的远算，不能用运算符进行矩阵运算，可 以使用 numpy.dot 和 numpy.transpose 分别来进行矩阵乘法运算和矩阵转置。其优点 在于常规操作时避免了对数据遍历。

NumPy的matrix类型则可以直接用运算符号进行运算。

例：使用matrix解方程组



import numpy as np

A = np.matrix([[3, 6, -5], [1, -3, 2], [5, -1, 4]]) # 定义矩阵

B = np.matrix([[12], [-2], [10]])

X = A \*\* (-1) \* B # 求方程组

print(X)

例：使用数组解方程组

import numpy as np

a = np.array([[3, 6, -5], [1, -3, 2], [5, -1, 4]])

b = np.array([12, -2, 10])

x = np.linalg.inv(a).dot(b)

print(x)

注意：数组的运算速度更快，而且为了在使用中保持数据类型一致，建议使用数组

## 9.2 Scipy库

### 9.2.1 总体说明

SciPy是一款方便、易于使用、专为科学和工程设计的Python工具包。它包括统计、优化、涉及线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解器等众多数学包。

### 9.2.2 代表性函数使用介绍

**1. 最优化**

（1）数据建模和拟合

SciPy 函数 curve\_fit 使用基于卡方的方法进行线性回归分析。下面，首先使用 f(x)=ax+b生成带有噪声的数据，然后使用用curv\_fit来拟合。

例：线性回归

import numpy as np from scipy.optimize

import curve\_fit

#创建函数 f(x)=ax+b

def func(x, a, b):

return a \* x + b

# 创建干静数据

x = np.linspace(0, 10, 100)

y = func(x, 1, 2)

# 添加噪声

yn = y + 0.9 \* np.random.normal(size=len(x))

# 拟合噪声数据

popt, pcov = curve\_fit(func, x, yn)

# 输出最优参数

print(popt)



例：高斯分布拟合

# 创建函数

def func(x, a, b, c):

return a\*np.exp(-(x-b)\*\*2/(2\*c\*\*2))

# 生成干静数据

x = np.linspace(0, 10, 100)

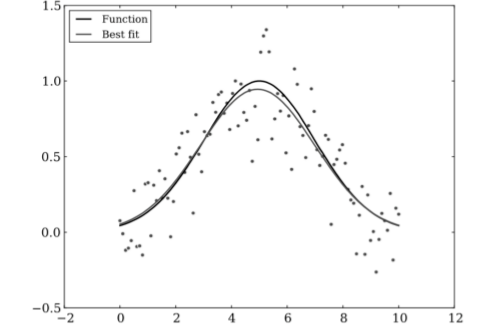
y = func(x, 1, 5, 2)

# 添加噪声

yn = y + 0.2 \* np.random.normal(size=len(x))

# 拟合

popt, pcov = curve\_fit(func, x, yn)



（2）函数求解

SciPy的optimize模块中有大量的函数求解工具，fsolve是其中最常用的。

例：线性函数求解

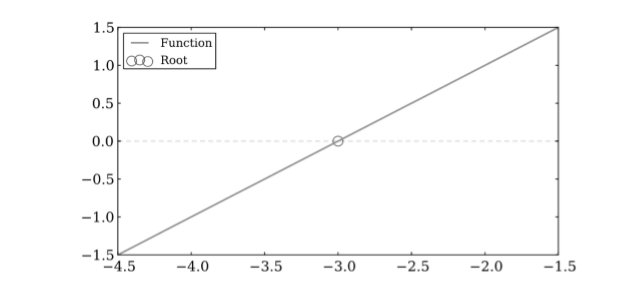
from scipy.optimize import fsolve

import numpy as np

line = lambda x: x + 3

solution = fsolve(line, -2)

print solution



例：求函数交叉点

from scipy.optimize import fsolve

import numpy as np

# 用于求解的解函

def findIntersection(func1, func2, x0):

return fsolve(lambda x : func1(x) - func2(x), x0)

# 两个函数

funky = lambda x : np.cos(x / 5) \* np.sin(x / 2)

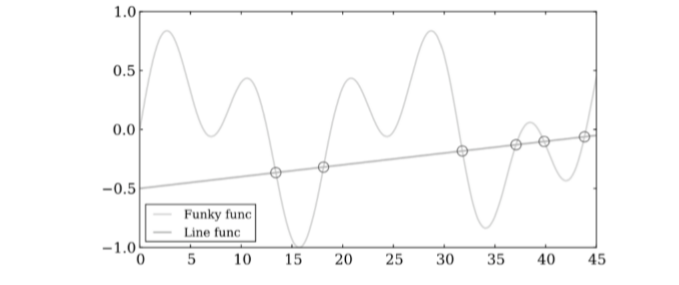
line = lambda x : 0.01 \* x - 0.5

x = np.linspace(0,45,10000)

result = findIntersection(funky, line, [15, 20, 30, 35, 40, 45])

# 输出结果

print(result, line(result))



**2. 插值**

（1）interp1d

例：正弦函数插值

import numpy as np from scipy.interpolate

import interp1d

# 创建待插值的数据

x = np.linspace(0, 10 \* np.pi, 20) y = np.cos(x)

# 分别用 linear 和 quadratic 插值

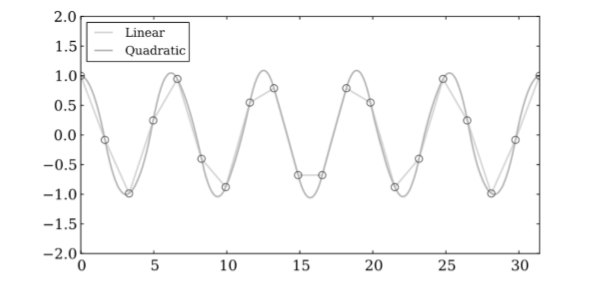
fl=interp1d(x,y,kind='linear')

fq = interp1d(x, y, kind='quadratic')

xint = np.linspace(x.min(), x.max(), 1000)

yintl = fl(xint)

yintq = fq(xint)



（2）UnivariateSpline

例：噪声数据插值

import numpy as np from scipy.interpolate

import UnivariateSpline

# 创建含噪声的待插值数据

sample = 30

x=np.linspace(1,10\*np.pi,sample)

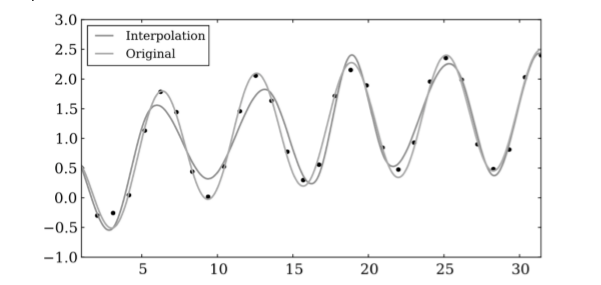
y=np.cos(x)+np.log10(x)+np.random.randn(sample)/10

#插值,参数 s 为 smoothing factor

f = UnivariateSpline(x, y, s=1)

xint = np.linspace(x.min(), x.max(), 1000)

yint = f(xint)



（3）griddata

例：利用插值重构图片

import numpy as np from scipy.interpolate

import griddata

#定义一个函数

ripple = lambda x, y: np.sqrt(x\*\*2 + y\*\*2)+np.sin(x\*\*2 + y\*\*2)

#生成grid数据,复数定义了生成grid数据的step，若无该复数则step为5

grid\_x,grid\_y=np.mgrid[0:5:1000j,0:5:1000j]

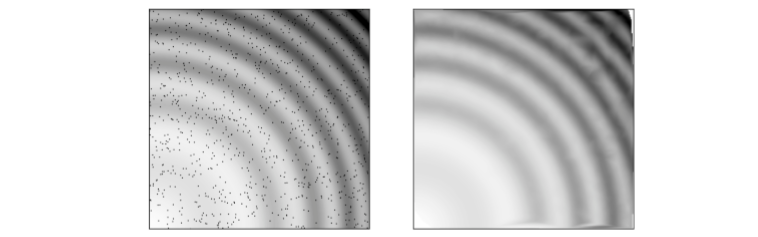
# 生成待插值的样本数据

xy = np.random.rand(1000, 2)

sample = ripple(xy[:,0] \* 5 , xy[:,1] \* 5)

# 用 cubic 方法插值

grid\_z0 = griddata(xy \* 5, sample, (grid\_x, grid\_y), method='cubic')



上图中，左侧为原始数据，其中的黑点是待插值的样本，右图为插值后的数据。要 想提高质量，生成更大的样本数据即可。

（4）SmoothBivariateSpline

例：利用插值重构图片

import numpy as np from scipy.interpolate

import SmoothBivariateSpline as SBS

# 定义函数

ripple = lambda x, y: np.sqrt(x\*\*2 + y\*\*2)+np.sin(x\*\*2 + y\*\*2)

# 生成待插值样本

xy= np.random.rand(1000, 2)

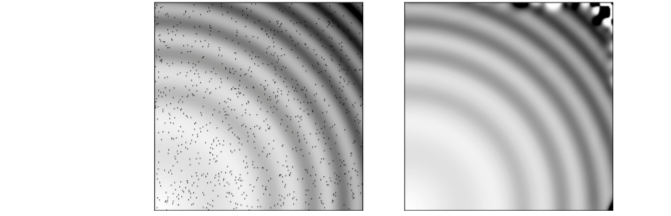
x, y = xy[:,0], xy[:,1]

sample = ripple(xy[:,0] \* 5 , xy[:,1] \* 5)

# 插值

fit = SBS(x \* 5, y \* 5, sample, s=0.01, kx=4, ky=4)

interp = fit(np.linspace(0, 5, 1000), np.linspace(0, 5, 1000))

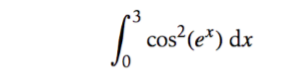


注意：SmoothBivariateSpline有时候表现比 Spline更好一些，但是它对样本数据更敏 感一些，相对而言Spline更加健壮。

**3. 积分**

SciPy中的积分是近似的数值积分，SymPy是一个符号积分的工具包。

(1)解析积分 例：



import numpy as np

from scipy.integrate import quad

# 定义被积函数

func = lambda x: np.cos(np.exp(x)) \*\* 2

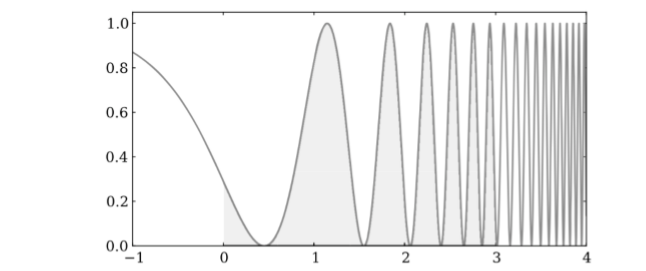
# 积分

solution = quad(func, 0, 3)

print solution

# 输出结果中 第一个数字为积分值，第二个为误差

# (1.296467785724373, 1.397797186265988e-09)



（2）数值积分

例：

import numpy as np

from scipy.integrate import quad, trapz

# Setting up fake data

x = np.sort(np.random.randn(150) \* 4 + 4).clip(0,5)

func = lambda x: np.sin(x) \* np.cos(x \*\* 2) + 1

y = func(x)

# Integrating function with upper and lower # limits of 0 and 5, respectively

fsolution = quad(func, 0, 5)

dsolution = trapz(y, x=x)

print('fsolution = ' + str(fsolution[0]))

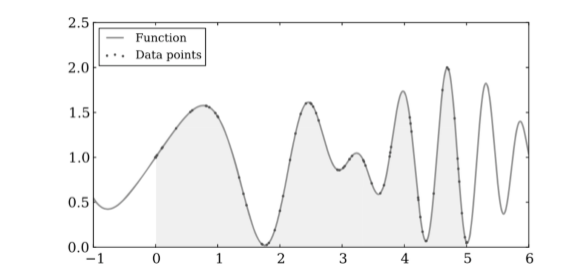
print('dsolution = ' + str(dsolution))

print('The difference is ' + str(np.abs(fsolution[0] - dsolution)))

#fsolution=5.10034506754

# dsolution = 5.04201628314

# The difference is 0.0583287843989.



**4. 统计**

SciPy中有包括mean, std, median, argmax, 及argmin等在内的基本统计函数， 而且numpy.arrays类型中内置了大部分统计函数，以便于使用。

例：

import numpy as np

# 创建大小为 1000 的随机数组

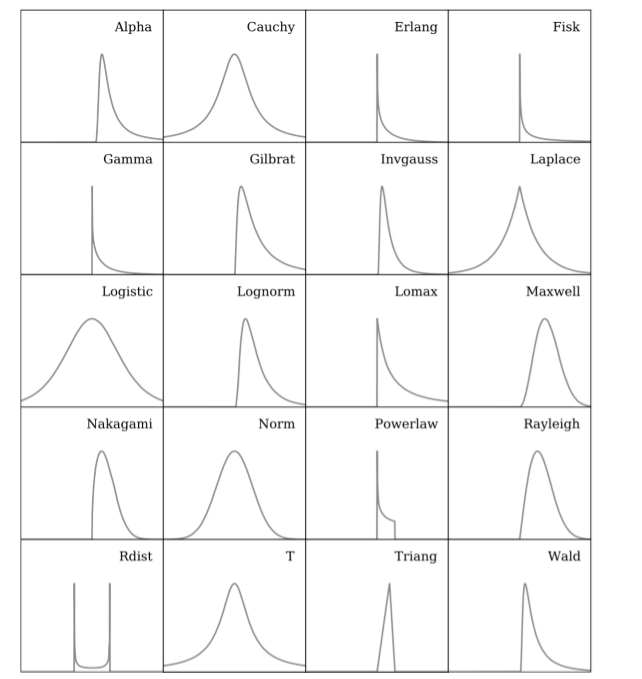
elements x = np.random.randn(1000)

mean = x.mean() # 均值

std = x.std() # 标准差

var = x.var() # 方差

SciPy中还包括了各种分布、函数等工具。连续和离散分布 SciPy的scipy.stats包中包含了大概80种连续分布和10种离散分布。下图是其中 的 20 种连续分布的概率密度函数。这些分布函数其实都依赖于 numpy.random 函数。



有几种方法来使用 scipy.stats 中的分布时：概率密度函数（PDFs）、累积分布函 数（CDFs）、随机变量样本（RVSs）、百分比点函数（PPFs）等。下面基于标准正态 分布函数，来演示如何使用这些分布。



例：

import numpy as np

import scipy.stats import norm

# 创建样本区间

x = np.linspace(-5,5,1000)

# 设置正态分布参数，loc 为均值，scale为标准差

dist = norm(loc=0, scale=1)

# 得到正态分布的 PDF 和 CDF

pdf = dist.pdf(x)

cdf = dist.cdf(x)

# 根据分布生成 500 个随机数

sample = dist.rvs(500)

可以基于SciPy.stats中的任何连续分布生成随机数，有需要请查看文档。除此外， 如泊松分布、二项分布、几何分布等离散分布的使用也很简单。下式为几何分布的概率 分布函数（PMF）：



例：

import numpy as np

from scipy.stats import geom

# 设置几何分布的参数

p = 0.5

dist = geom(p)

# 设置样本区间

x = np.linspace(0, 5, 1000)

# 得到几何分布的 PMF 和 CDF

pmf = dist.pmf(x)

cdf = dist.cdf(x)

# 生成 500 个随机数 sample = dist.rvs(500)

(2)函数

SciPy 中有超过 60 种统计函数。stats 包中包括了诸如 kstest 和 normaltest 等的 样

本测试函数，用来检测样本是否服从某种分布。提示：在使用这些工具前，要对数据 有较好的理解，否则可能会误读它们的结果。

例：样本分布检验

import numpy as np

from scipy import stats

# 生成 包括 100 个服从正态分布的随机数样本

sample = np.random.randn(100)

#用 normaltest 检验原假设

out=stats.normaltest(sample)

print('normaltest output')

print('Z-score = ' + str(out[0]))

print('P-value = ' + str(out[1]))

# kstest是检验拟合度的Kolmogorov-Smirnov检验，这里针对正态分布进行检验 ，

# D是KS统计量的值,越接近0越好

out = stats.kstest(sample, 'norm')

print('\nkstest output for the Normal distribution')

print('D = ' + str(out[0]))

print('P-value = ' + str(out[1]))

# 类似地可以针对其他分布进行检验,例如 Wald分布

out = stats.kstest(sample, 'wald')

print('\nkstest output for the Wald distribution')

print('D = ' + str(out[0]))

print('P-value = ' + str(out[1]))

SciPy的stats模块中还提供了一些描述函数，如几何平均（gmean）、偏度（ skew）、 样本频数（itemfreq）等。

例：

import numpy as np

from scipy import stats

# 生成包括 100 个服从正态分布的随机数样本

sample = np.random.randn(100)

# 调和平均数,样本值须大于 0

out = stats.hmean(sample[sample > 0])

print('Harmonic mean = ' + str(out))

# 计算 -1 到 1 之间样本的均值

out = stats.tmean(sample, limits=(-1, 1))

print('\nTrimmed mean = ' + str(out))

# 计算样本偏度

out = stats.skew(sample)

print('\nSkewness = ' + str(out))

# 函数 describe 可以一次给出样本的多种描述统计结果

out = stats.describe(sample)

print('\nSize = ' + str(out[0]))

print('Min = ' + str(out[1][0]))

print('Max = ' + str(out[1][1]))

print('Mean = ' + str(out[2]))

print('Variance = ' + str(out[3]))

print('Skewness = ' + str(out[4]))

print('Kurtosis = ' + str(out[5]))

SciPy的stats模块中还有很多统计工具，可以满足绝大多数需要。还可以用RPy， 通过它能够在Python中调用R语言进行统计分析。此外，Pandas是python的一个强 大的工具包，它可以在大数据上进行快速的统计分析。

**5. 空间和聚类分析**

SciPy包括scipy.spatial类和 scipy.cluster类分别用于空间和聚类分析。前者用于 分析数据点之间的距离，后者包括两个子类矢量量化（vq）和层次聚类（hierarchy）。

(1)矢量量化（Vector Quantization）

矢量量化是信号处理、数据压缩和聚类等领域通用的术语。这里仅关注其在聚类中 的应用。

例：k均值聚类

import numpy as np

from scipy.cluster import vq

# 生成数据

c1 = np.random.randn(100, 2) + 5

c2 = np.random.randn(30, 2) - 5

c3 = np.random.randn(50, 2)

# 将所有数据放入一个 180 x 2 的数组

data = np.vstack([c1, c2, c3])

# 利用 k 均值方法计算聚类的质心和方差

centroids, variance = vq.kmeans(data, 3)

# 变量identified中存放关于数据聚类的信息

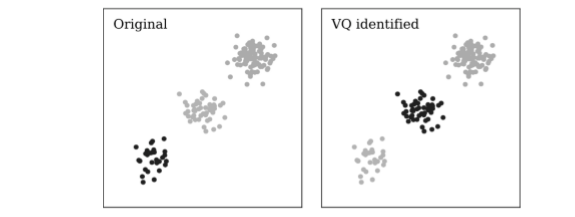
identified, distance = vq.vq(data, centroids)

# 获得各类别的数据

vqc1 = data[identified == 0]

vqc2 = data[identified == 1]

vqc3 = data[identified == 2]



(2)层次聚类

层次聚类是一种重要的聚类方法，但其输出结果比较复杂，不能像k均值那样给出 清晰的聚类结果。下面是一个层次聚类的例子，输入一个距离矩阵，输出为一个树状图。 例：

# coding:utf-8

import numpy as np

import matplotlib.pyplot as mpl

from scipy.spatial.distance import pdist, squareform

import scipy.cluster.hierarchy as hy

# 用于生成聚类数据的函数

def clusters(number=20, cnumber=5, csize=10):

# 聚类服从高斯分布

rnum = np.random.rand(cnumber, 2)

rn = rnum[:, 0] \* number

rn = rn.astype(int)

rn[np.where(rn < 5)] = 5

rn[np.where(rn > number / 2.)] = round(number / 2., 0)

ra = rnum[:, 1] \* 2.9

ra[np.where(ra < 1.5)] = 1.5

cls = np.random.randn(number, 3) \* csize

# Random multipliers for central point of cluster

rxyz = np.random.randn(cnumber - 1, 3)

for i in xrange(cnumber - 1):

tmp = np.random.randn(rn[i + 1], 3)

x = tmp[:, 0] + (rxyz[i, 0] \* csize)

y = tmp[:, 1] + (rxyz[i, 1] \* csize)

z = tmp[:, 2] + (rxyz[i, 2] \* csize)

tmp = np.column\_stack([x, y, z])

cls = np.vstack([cls, tmp])

return cls

# 创建待聚类数据及矩离矩阵

cls = clusters()

D = pdist(cls[:, 0:2])

D = squareform(D)

#绘制左侧树状图

fig = mpl.figure(figsize=(8, 8))

ax1 = fig.add\_axes([0.09, 0.1, 0.2, 0.6])

Y1 = hy.linkage(D, method='complete')

cutoff = 0.3 \* np.max(Y1[:, 2])

Z1=hy.dendrogram(Y1,orientation='left',color\_threshold=cutoff) ax1.xaxis.set\_visible(False) ax1.yaxis.set\_visible(False)

# 绘制顶部树状图

ax2 = fig.add\_axes([0.3, 0.71, 0.6, 0.2])

Y2 = hy.linkage(D, method='average')

cutoff = 0.3 \* np.max(Y2[:, 2])

Z2=hy.dendrogram(Y2,color\_threshold=cutoff)

ax2.xaxis.set\_visible(False) ax2.yaxis.set\_visible(False)

# 显示距离矩阵

ax3 = fig.add\_axes([0.3, 0.1, 0.6, 0.6])

idx1 = Z1['leaves']

idx2 = Z2['leaves']

D = D[idx1, :]

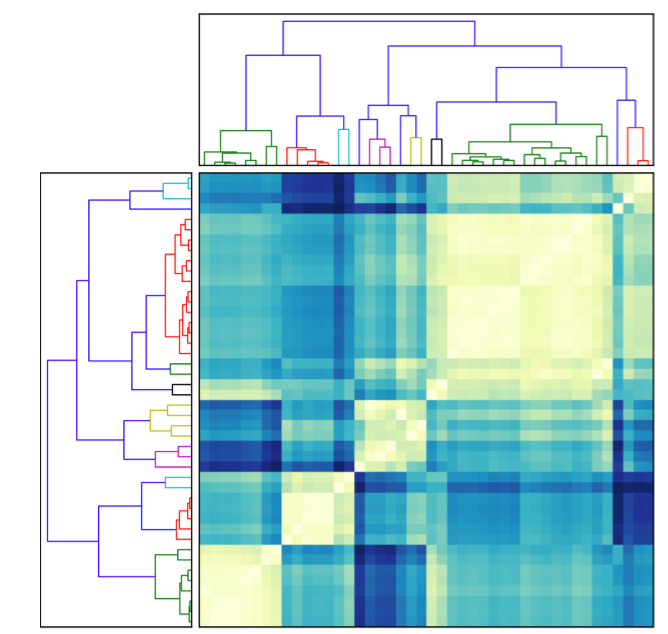
D = D[:, idx2]

ax3.matshow(D,aspect='auto',origin='lower',cmap=mpl.cm.YlGnBu) ax3.xaxis.set\_visible(False) ax3.yaxis.set\_visible(False)

# 保存图片 ，显示图片

fig.savefig('cluster\_hy\_f01.pdf', bbox = 'tight')

mpl.show()



在上图虽然计算了数据点之间的距离，但是还是难以将各类区分开。函数 fcluster 可以根据阈值来区分各类，其输出结果依赖于linkage函数所采用的方法，如complete 或 single 等，它的第二个参数即是阈值。dendrogram 函数中默认的阈值是 0.7 \* np.max(Y[:, 2])，这里还使用0.3。

例：

# 导入的包同上例一致,函数cluster同上例

# 获得不同类别数据点的坐标

def group(data, index):

number = np.unique(index)

groups = []

for i in number:

groups.append(data[index == i])

return groups

# 创建数据

cls = clusters()

# 计算 kinkage矩阵

Y = hy.linkage(cls[:,0:2], method='complete')

# 从层次数据结构中, 用 fcluster 函数将层次结构的数据转为

flat clusters

cutoff = 0.3 \* np.max(Y[:, 2])

index = hy.fcluster(Y, cutoff, 'distance')

# 使用 grooup 函数将数据划分类别

groups = group(cls, index)

# 绘制数据点

fig = mpl.figure(figsize=(6, 6))

ax = fig.add\_subplot(111)

colors = ['r', 'c', 'b', 'g', 'orange', 'k', 'y', 'gray']

for i, g in enumerate(groups):

i = np.mod(i, len(colors))

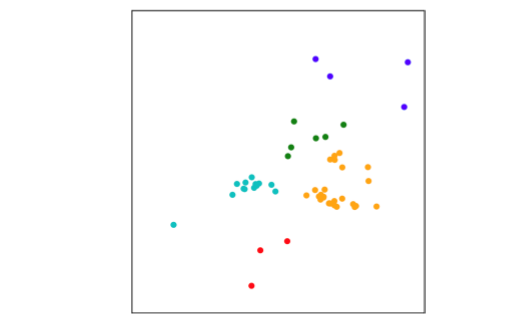
ax.scatter(g[:,0], g[:,1], c=colors[i], edgecolor='none', s=50)

ax.xaxis.set\_visible(False)

ax.yaxis.set\_visible(False)

fig.savefig('cluster\_hy\_f02.pdf', bbox = 'tight')

mpl.show()



**6. 稀疏矩阵**

NumPy 处理 106 级别的数据没什么大问题，当数据量达到 107 级别时速度开始变慢，内存受到限制（具体情况取决于实际内存大小）。当处理超大规模数据集，比如1010 级别，且数据中包含大量的0时，可采用稀疏矩阵可显著的提高速度和效率。

提示：使用data.nbytes可查看数据所占空间大小

例：矩阵与稀疏矩阵运算对比

# coding:utf-8

import numpy as np

from scipy.sparse.linalg import eigsh

from scipy.linalg import eigh

import scipy.sparse

import time

N = 3000

# 创建随机稀疏矩阵

m = scipy.sparse.rand(N, N)

# 创建包含相同数据的数组

a = m.toarray()

print('The numpy array data size: ' + str(a.nbytes) + ' bytes')

print('The sparse matrix data size: ' + str(m.data.nbytes) + ' bytes')

# 数组求特征值

t0 = time.time()

res1 = eigh(a)

dt = str(np.round(time.time() - t0, 3)) + ' seconds'

print('Non-sparse operation takes ' + dt)

# 稀疏长阵求特征值

t0 = time.time()

res2 = eigsh(m)

dt = str(np.round(time.time() - t0, 3)) + ' seconds'

print('Sparse operation takes ' + dt)

非几何的稀疏矩阵可用于优化、经济建模、数学和统计,和网络/图等的运算。利用 scipy.io模块可读写Matrix Market、Harwell-Boeing或MatLab格式的稀疏矩阵数据文件。

## 9.3 Pandas库

### 9.3.1 pandas库总体说明

Pandas基于NumPy、SciPy补充了大量数据操作功能，能实现统计、分组、排序、透视表，可以代替Excel的绝大部分功能。

Pandas主要有2种重要数据类型：Series、DataFrame（一维序列、二维表）。数据类型的转换需要用到pd.Series/DataFrame.

**1）Series**

可以是一个样本的所有观测值或一组样本的某一属性的观测值。

如利用NumPy生成一个正态分布的随机数列，共含4个值。

Series1 = pd.Series(np.random.randn(4))

结果就自动添加了行索引index。

0 -1.344609

1 0.177173

2 0.554958

3 -0.576237

过滤Series的方法是：print Series1 < 0或print Series1[Series1 < 0]。前者给出Boolean类型的输出，后者给出具体的数值，仅仅输出Series中小于0的数值。

可以使用Key-Value的方式存储数据：

Series2 = pd.Series(Series1.values, index = ["row\_" + unicode(i) for i in range(4)])

同样，Python的基础数据结构字典也可以转化为Series。

Series3 = pd.Series({"China": "Beijing", "England": "GB", "Japan": "Tokyo"})

输出结果依旧是一个序列，但是因为字典本身是无序的，所有有可能会打乱原字典的顺序。如果需要顺便不变，可以使用下面的方法明确指定这种秩序：

Series4\_IndexList = ["China", "Japan", "England"]

Series4 = pd.Series(Series3, index = Series4\_IndexList)

某些时候，Index列表没有相应的对应值，这样会默认填补为空值，可以使用isnull(0, notnull()来返回Boolean结果。

Series5\_IndexList = ["A", "B", "C", "C"]

Series5 = pd.Series(Series1.values, index = Series5\_IndexList)

index允许重复，但是这样容易导致错误。

**2）DataFrame**

DataFrame可以视作Series的有序集合， 可以从数据库、NumPy二维数组、JSON中定义数据框。

NumPy二维数组：

DF1 = pd.DataFrame(np.asarray([("Japan", "Tokyo", 4000), ("S.Korea", "Seoul", 1000), ("China", "Beijing", 9000)]), columns = ["nation", "capital", "GDP"])

JSON：

DF2 = pd.DataFrame({"nation": ["Japan", "S.Korea", "China"], "capital": ["Tokyo", "Seoul", "Beijing"], "GDP": [4000, 1000, 9000]})

但是字典的key是无序的,所以我们又要用到刚才Series中的类似方法加以解决：

DF3 = pd.DataFrame(DF2, columns = ["nation", "capital", "GDP"])

对应地，还可以人为指定行标秩序。

DF4 = pd.DataFrame(DF2, columns = ["nation", "capital", "GDP"], index = [2, 0, 1])

在DataFrame中切片：

取列：推荐使用DF4["GDP"]，最好别用DF4.GDP，容易与一些关键字（保留字）冲突

取行：DF4[0: 1]或者DF4.ix[0]

区别在于前者取了第一行，后者取了index（行标）为0的第一行。

此外，如果要在数据框动态增加列，不能用.的方式，而要用[]

DF4["region"] = "East Asian"

### 9.3.2 代表性函数的使用介绍:

**In [1]: import** **pandas** **as** **pd**

**In [2]: import** **numpy** **as** **np**

**In [3]: import** **matplotlib.pyplot** **as** **plt**

一、创建对象

1、可以通过传递一个list对象来创建一个Series：

**In [4]:** s = pd.Series([1,3,5,np.nan,6,8])

**In [5]:** s

Out[5]:

0 1.0

1 3.0

2 5.0

3 NaN

4 6.0

5 8.0

dtype: float64

2、通过传递一个numpy array，时间索引以及列标签来创建一个DataFrame：

**In [6]:** dates = pd.date\_range('20130101', periods=6)

**In [7]:** dates

Out[7]:

DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',

'2013-01-05', '2013-01-06'],

dtype='datetime64[ns]', freq='D')

**In [8]:** df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

**In [9]:** df

Out[9]:

A B C D

2013-01-01 0.469112 -0.282863 -1.509059 -1.135632

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804

2013-01-04 0.721555 -0.706771 -1.039575 0.271860

2013-01-05 -0.424972 0.567020 0.276232 -1.087401

2013-01-06 -0.673690 0.113648 -1.478427 0.524988

3、通过传递一个能够被转换成类似序列结构的字典对象来创建一个DataFrame：

**In [10]:** df2 = pd.DataFrame({ 'A' : 1.,

**....:** 'B' : pd.Timestamp('20130102'),

**....:** 'C' : pd.Series(1,index=list(range(4)),dtype='float32'),

**....:** 'D' : np.array([3] \* 4,dtype='int32'),

**....:** 'E' : pd.Categorical(["test","train","test","train"]),

**....:** 'F' : 'foo' })

**....:**

**In [11]:** df2

Out[11]:

A B C D E F

0 1.0 2013-01-02 1.0 3 test foo

1 1.0 2013-01-02 1.0 3 train foo

2 1.0 2013-01-02 1.0 3 test foo

3 1.0 2013-01-02 1.0 3 train foo

4、查看不同列的数据类型：

**In [12]:** df2.dtypes

Out[12]:

A float64

B datetime64[ns]

C float32

D int32

E category

F object

dtype: object

二、查看数据

1、 查看frame中头部和尾部的行：

**In [14]:** df.head()

Out[14]:

A B C D

2013-01-01 0.469112 -0.282863 -1.509059 -1.135632

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804

2013-01-04 0.721555 -0.706771 -1.039575 0.271860

2013-01-05 -0.424972 0.567020 0.276232 -1.087401

**In [15]:** df.tail(3)

Out[15]:

A B C D

2013-01-04 0.721555 -0.706771 -1.039575 0.271860

2013-01-05 -0.424972 0.567020 0.276232 -1.087401

2013-01-06 -0.673690 0.113648 -1.478427 0.524988

2、 显示索引、列和底层的numpy数据：

**In [16]:** df.index

Out[16]:

DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',

'2013-01-05', '2013-01-06'],

dtype='datetime64[ns]', freq='D')

**In [17]:** df.columns

Out[17]: Index([u'A', u'B', u'C', u'D'], dtype='object')

**In [18]:** df.values

Out[18]:

array([[ 0.4691, -0.2829, -1.5091, -1.1356],

[ 1.2121, -0.1732, 0.1192, -1.0442],

[-0.8618, -2.1046, -0.4949, 1.0718],

[ 0.7216, -0.7068, -1.0396, 0.2719],

[-0.425 , 0.567 , 0.2762, -1.0874],

[-0.6737, 0.1136, -1.4784, 0.525 ]])

3、 describe()函数对于数据的快速统计汇总：

**In [19]:** df.describe()

Out[19]:

A B C D

count 6.000000 6.000000 6.000000 6.000000

mean 0.073711 -0.431125 -0.687758 -0.233103

std 0.843157 0.922818 0.779887 0.973118

min -0.861849 -2.104569 -1.509059 -1.135632

25% -0.611510 -0.600794 -1.368714 -1.076610

50% 0.022070 -0.228039 -0.767252 -0.386188

75% 0.658444 0.041933 -0.034326 0.461706

max 1.212112 0.567020 0.276232 1.071804

4、 对数据的转置：

**In [20]:** df.T

Out[20]:

2013-01-01 2013-01-02 2013-01-03 2013-01-04 2013-01-05 2013-01-06

A 0.469112 1.212112 -0.861849 0.721555 -0.424972 -0.673690

B -0.282863 -0.173215 -2.104569 -0.706771 0.567020 0.113648

C -1.509059 0.119209 -0.494929 -1.039575 0.276232 -1.478427

D -1.135632 -1.044236 1.071804 0.271860 -1.087401 0.524988

5、 按轴进行排序

**In [21]:** df.sort\_index(axis=1, ascending=False)

Out[21]:

D C B A

2013-01-01 -1.135632 -1.509059 -0.282863 0.469112

2013-01-02 -1.044236 0.119209 -0.173215 1.212112

2013-01-03 1.071804 -0.494929 -2.104569 -0.861849

2013-01-04 0.271860 -1.039575 -0.706771 0.721555

2013-01-05 -1.087401 0.276232 0.567020 -0.424972

2013-01-06 0.524988 -1.478427 0.113648 -0.673690

6、 按值进行排序

**In [22]:** df.sort\_values(by='B')

Out[22]:

A B C D

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804

2013-01-04 0.721555 -0.706771 -1.039575 0.271860

2013-01-01 0.469112 -0.282863 -1.509059 -1.135632

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-06 -0.673690 0.113648 -1.478427 0.524988

2013-01-05 -0.424972 0.567020 0.276232 -1.087401

三、选择

**虽然标准的Python/Numpy的选择和设置表达式都能够直接派上用场，但是作为工程使用的代码，推荐使用经过优化的pandas数据访问方式： .at, .iat, .loc, .iloc 和 .ix**。

获取

1、 选择一个单独的列，这将会返回一个Series，等同于df.A：

**In [23]:** df['A']

Out[23]:

2013-01-01 0.469112

2013-01-02 1.212112

2013-01-03 -0.861849

2013-01-04 0.721555

2013-01-05 -0.424972

2013-01-06 -0.673690

Freq: D, Name: A, dtype: float64

2、 通过[]进行选择，这将会对行进行切片

**In [24]:** df[0:3]

Out[24]:

A B C D

2013-01-01 0.469112 -0.282863 -1.509059 -1.135632

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804

**In [25]:** df['20130102':'20130104']

Out[25]:

A B C D

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804

2013-01-04 0.721555 -0.706771 -1.039575 0.271860

通过标签选择

1、 使用标签来获取一个交叉的区域

**In [26]:** df.loc[dates[0]]

Out[26]:

A 0.469112

B -0.282863

C -1.509059

D -1.135632

Name: 2013-01-01 00:00:00, dtype: float64

2、 通过标签来在多个轴上进行选择

**In [27]:** df.loc[:,['A','B']]

Out[27]:

A B

2013-01-01 0.469112 -0.282863

2013-01-02 1.212112 -0.173215

2013-01-03 -0.861849 -2.104569

2013-01-04 0.721555 -0.706771

2013-01-05 -0.424972 0.567020

2013-01-06 -0.673690 0.113648

3、 标签切片

**In [28]:** df.loc['20130102':'20130104',['A','B']]

Out[28]:

A B

2013-01-02 1.212112 -0.173215

2013-01-03 -0.861849 -2.104569

2013-01-04 0.721555 -0.706771

4、 对于返回的对象进行维度缩减

**In [29]:** df.loc['20130102',['A','B']]

Out[29]:

A 1.212112

B -0.173215

Name: 2013-01-02 00:00:00, dtype: float64

5、 获取一个标量

**In [30]:** df.loc[dates[0],'A']

Out[30]: 0.46911229990718628

6、 快速访问一个标量（与上一个方法等价）

**In [31]:** df.at[dates[0],'A']

Out[31]: 0.46911229990718628

通过位置选择

1、 通过传递数值进行位置选择（选择的是行）

**In [32]:** df.iloc[3]

Out[32]:

A 0.721555

B -0.706771

C -1.039575

D 0.271860

Name: 2013-01-04 00:00:00, dtype: float64

2、 通过数值进行切片，与numpy/python中的情况类似

**In [33]:** df.iloc[3:5,0:2]

Out[33]:

A B

2013-01-04 0.721555 -0.706771

2013-01-05 -0.424972 0.567020

3、 通过指定一个位置的列表，与numpy/python中的情况类似

**In [34]:** df.iloc[[1,2,4],[0,2]]

Out[34]:

A C

2013-01-02 1.212112 0.119209

2013-01-03 -0.861849 -0.494929

2013-01-05 -0.424972 0.276232

4、对行进行切片

**In [35]:** df.iloc[1:3,:]

Out[35]:

A B C D

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804

5、 对列进行切片

**In [36]:** df.iloc[:,1:3]

Out[36]:

B C

2013-01-01 -0.282863 -1.509059

2013-01-02 -0.173215 0.119209

2013-01-03 -2.104569 -0.494929

2013-01-04 -0.706771 -1.039575

2013-01-05 0.567020 0.276232

2013-01-06 0.113648 -1.478427

6、 获取特定的值

**In [37]:** df.iloc[1,1]

Out[37]: -0.17321464905330858

**In [38]:** df.iat[1,1]

Out[38]: -0.17321464905330858

布尔索引

1、 使用一个单独列的值来选择数据：

**In [39]:** df[df.A > 0]

Out[39]:

A B C D

2013-01-01 0.469112 -0.282863 -1.509059 -1.135632

2013-01-02 1.212112 -0.173215 0.119209 -1.044236

2013-01-04 0.721555 -0.706771 -1.039575 0.271860

2、选择数据：

**In [40]:** df[df > 0]

Out[40]:

A B C D

2013-01-01 0.469112 NaN NaN NaN

2013-01-02 1.212112 NaN 0.119209 NaN

2013-01-03 NaN NaN NaN 1.071804

2013-01-04 0.721555 NaN NaN 0.271860

2013-01-05 NaN 0.567020 0.276232 NaN

2013-01-06 NaN 0.113648 NaN 0.524988

3、 使用isin()方法来过滤：

**In [41]:** df2 = df.copy()

**In [42]:** df2['E'] = ['one', 'one','two','three','four','three']

**In [43]:** df2

Out[43]:

A B C D E

2013-01-01 0.469112 -0.282863 -1.509059 -1.135632 one

2013-01-02 1.212112 -0.173215 0.119209 -1.044236 one

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804 two

2013-01-04 0.721555 -0.706771 -1.039575 0.271860 three

2013-01-05 -0.424972 0.567020 0.276232 -1.087401 four

2013-01-06 -0.673690 0.113648 -1.478427 0.524988 three

**In [44]:** df2[df2['E'].isin(['two','four'])]

Out[44]:

A B C D E

2013-01-03 -0.861849 -2.104569 -0.494929 1.071804 two

2013-01-05 -0.424972 0.567020 0.276232 -1.087401 four

设置

1、 设置一个新的列：

**In [45]:** s1 = pd.Series([1,2,3,4,5,6], index=pd.date\_range('20130102', periods=6))

**In [46]:** s1

Out[46]:

2013-01-02 1

2013-01-03 2

2013-01-04 3

2013-01-05 4

2013-01-06 5

2013-01-07 6

Freq: D, dtype: int64

**In [47]:** df['F'] = s1

2、 通过标签设置新的值：

**In [48]:** df.at[dates[0],'A'] = 0

3、 通过位置设置新的值：

**In [49]:** df.iat[0,1] = 0

4、 通过一个numpy数组设置一组新值：

**In [50]:** df.loc[:,'D'] = np.array([5] \* len(df))

上述操作结果如下：

**In [51]:** df

Out[51]:

A B C D F

2013-01-01 0.000000 0.000000 -1.509059 5 NaN

2013-01-02 1.212112 -0.173215 0.119209 5 1.0

2013-01-03 -0.861849 -2.104569 -0.494929 5 2.0

2013-01-04 0.721555 -0.706771 -1.039575 5 3.0

2013-01-05 -0.424972 0.567020 0.276232 5 4.0

2013-01-06 -0.673690 0.113648 -1.478427 5 5.0

5、 通过where操作来设置新的值：

**In [52]:** df2 = df.copy()

**In [53]:** df2[df2 > 0] = -df2

**In [54]:** df2

Out[54]:

A B C D F

2013-01-01 0.000000 0.000000 -1.509059 -5 NaN

2013-01-02 -1.212112 -0.173215 -0.119209 -5 -1.0

2013-01-03 -0.861849 -2.104569 -0.494929 -5 -2.0

2013-01-04 -0.721555 -0.706771 -1.039575 -5 -3.0

2013-01-05 -0.424972 -0.567020 -0.276232 -5 -4.0

2013-01-06 -0.673690 -0.113648 -1.478427 -5 -5.0

四、缺失值处理

在pandas中，使用np.nan来代替缺失值，这些值将默认不会包含在计算中

1、reindex()方法可以对指定轴上的索引进行改变/增加/删除操作，这将返回原始数据的一个拷贝：

**In [55]:** df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ['E'])

**In [56]:** df1.loc[dates[0]:dates[1],'E'] = 1

**In [57]:** df1

Out[57]:

A B C D F E

2013-01-01 0.000000 0.000000 -1.509059 5 NaN 1.0

2013-01-02 1.212112 -0.173215 0.119209 5 1.0 1.0

2013-01-03 -0.861849 -2.104569 -0.494929 5 2.0 NaN

2013-01-04 0.721555 -0.706771 -1.039575 5 3.0 NaN

2、 去掉包含缺失值的行：

**In [58]:** df1.dropna(how='any')

Out[58]:

A B C D F E

2013-01-02 1.212112 -0.173215 0.119209 5 1.0 1.0

3、 对缺失值进行填充：

**In [59]:** df1.fillna(value=5)

Out[59]:

A B C D F E

2013-01-01 0.000000 0.000000 -1.509059 5 5.0 1.0

2013-01-02 1.212112 -0.173215 0.119209 5 1.0 1.0

2013-01-03 -0.861849 -2.104569 -0.494929 5 2.0 5.0

2013-01-04 0.721555 -0.706771 -1.039575 5 3.0 5.0

4、 对数据进行布尔填充：

**In [60]:** pd.isnull(df1)

Out[60]:

A B C D F E

2013-01-01 False False False False True False

2013-01-02 False False False False False False

2013-01-03 False False False False False True

2013-01-04 False False False False False True

操作

统计（相关操作通常情况下不包括缺失值）

1、 执行描述性统计：

**In [61]:** df.mean()

Out[61]:

A -0.004474

B -0.383981

C -0.687758

D 5.000000

F 3.000000

dtype: float64

2、 在其他轴上进行相同的操作：

**In [62]:** df.mean(1)

Out[62]:

2013-01-01 0.872735

2013-01-02 1.431621

2013-01-03 0.707731

2013-01-04 1.395042

2013-01-05 1.883656

2013-01-06 1.592306

Freq: D, dtype: float64

3、 对于拥有不同维度，需要对齐的对象进行操作。Pandas会自动的沿着指定的维度进行广播：

**In [63]:** s = pd.Series([1,3,5,np.nan,6,8], index=dates).shift(2)

**In [64]:** s

Out[64]:

2013-01-01 NaN

2013-01-02 NaN

2013-01-03 1.0

2013-01-04 3.0

2013-01-05 5.0

2013-01-06 NaN

Freq: D, dtype: float64

**In [65]:** df.sub(s, axis='index')

Out[65]:

A B C D F

2013-01-01 NaN NaN NaN NaN NaN

2013-01-02 NaN NaN NaN NaN NaN

2013-01-03 -1.861849 -3.104569 -1.494929 4.0 1.0

2013-01-04 -2.278445 -3.706771 -4.039575 2.0 0.0

2013-01-05 -5.424972 -4.432980 -4.723768 0.0 -1.0

2013-01-06 NaN NaN NaN NaN NaN

Apply

1、 对数据应用函数：

**In [66]:** df.apply(np.cumsum)

Out[66]:

A B C D F

2013-01-01 0.000000 0.000000 -1.509059 5 NaN

2013-01-02 1.212112 -0.173215 -1.389850 10 1.0

2013-01-03 0.350263 -2.277784 -1.884779 15 3.0

2013-01-04 1.071818 -2.984555 -2.924354 20 6.0

2013-01-05 0.646846 -2.417535 -2.648122 25 10.0

2013-01-06 -0.026844 -2.303886 -4.126549 30 15.0

**In [67]:** df.apply(**lambda** x: x.max() - x.min())

Out[67]:

A 2.073961

B 2.671590

C 1.785291

D 0.000000

F 4.000000

dtype: float64

直方图

**In [68]:** s = pd.Series(np.random.randint(0, 7, size=10))

**In [69]:** s

Out[69]:

0 4

1 2

2 1

3 2

4 6

5 4

6 4

7 6

8 4

9 4

dtype: int64

**In [70]:** s.value\_counts()

Out[70]:

4 5

6 2

2 2

1 1

dtype: int64

字符串方法

Series对象在其str属性中配备了一组字符串处理方法，可以很容易的应用到数组中的每个元素，如下段代码所示。变成小写。

**In [71]:** s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])

**In [72]:** s.str.lower()

Out[72]:

0 a

1 b

2 c

3 aaba

4 baca

5 NaN

6 caba

7 dog

8 cat

dtype: object

六、合并

Pandas提供了大量的方法能够轻松的对Series，DataFrame和Panel对象进行各种符合各种逻辑关系的合并操作。

用[**concat()**](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.concat.html#pandas.concat)把pandas类合并到一起:

**In [73]:** df = pd.DataFrame(np.random.randn(10, 4))

**In [74]:** df

Out[74]:

0 1 2 3

0 -0.548702 1.467327 -1.015962 -0.483075

1 1.637550 -1.217659 -0.291519 -1.745505

2 -0.263952 0.991460 -0.919069 0.266046

3 -0.709661 1.669052 1.037882 -1.705775

4 -0.919854 -0.042379 1.247642 -0.009920

5 0.290213 0.495767 0.362949 1.548106

6 -1.131345 -0.089329 0.337863 -0.945867

7 -0.932132 1.956030 0.017587 -0.016692

8 -0.575247 0.254161 -1.143704 0.215897

9 1.193555 -0.077118 -0.408530 -0.862495

*# break it into pieces*

**In [75]:** pieces = [df[:3], df[3:7], df[7:]]

**In [76]:** pd.concat(pieces)

Out[76]:

0 1 2 3

0 -0.548702 1.467327 -1.015962 -0.483075

1 1.637550 -1.217659 -0.291519 -1.745505

2 -0.263952 0.991460 -0.919069 0.266046

3 -0.709661 1.669052 1.037882 -1.705775

4 -0.919854 -0.042379 1.247642 -0.009920

5 0.290213 0.495767 0.362949 1.548106

6 -1.131345 -0.089329 0.337863 -0.945867

7 -0.932132 1.956030 0.017587 -0.016692

8 -0.575247 0.254161 -1.143704 0.215897

9 1.193555 -0.077118 -0.408530 -0.862495

Join

Join 类似于SQL类型的合并

**In [77]:** left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})

**In [78]:** right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})

**In [79]:** left

Out[79]:

key lval

0 foo 1

1 foo 2

**In [80]:** right

Out[80]:

key rval

0 foo 4

1 foo 5

**In [81]:** pd.merge(left, right, on='key')

Out[81]:

key lval rval

0 foo 1 4

1 foo 1 5

2 foo 2 4

3 foo 2 5

Append

Append 将一行连接到一个DataFrame上

**In [82]:** df = pd.DataFrame(np.random.randn(8, 4), columns=['A','B','C','D'])

**In [83]:** df

Out[83]:

A B C D

0 1.346061 1.511763 1.627081 -0.990582

1 -0.441652 1.211526 0.268520 0.024580

2 -1.577585 0.396823 -0.105381 -0.532532

3 1.453749 1.208843 -0.080952 -0.264610

4 -0.727965 -0.589346 0.339969 -0.693205

5 -0.339355 0.593616 0.884345 1.591431

6 0.141809 0.220390 0.435589 0.192451

7 -0.096701 0.803351 1.715071 -0.708758

**In [84]:** s = df.iloc[3]

**In [85]:** df.append(s, ignore\_index=True)

Out[85]:

A B C D

0 1.346061 1.511763 1.627081 -0.990582

1 -0.441652 1.211526 0.268520 0.024580

2 -1.577585 0.396823 -0.105381 -0.532532

3 1.453749 1.208843 -0.080952 -0.264610

4 -0.727965 -0.589346 0.339969 -0.693205

5 -0.339355 0.593616 0.884345 1.591431

6 0.141809 0.220390 0.435589 0.192451

7 -0.096701 0.803351 1.715071 -0.708758

8 1.453749 1.208843 -0.080952 -0.264610

七、分组

对于”group by”操作，我们通常是指以下一个或多个操作步骤：

（Splitting）按照一些规则将数据分为不同的组；

（Applying）对于每组数据分别执行一个函数；

（Combining）将结果组合到一个数据结构中；

**In [86]:** df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',

**....:** 'foo', 'bar', 'foo', 'foo'],

**....:** 'B' : ['one', 'one', 'two', 'three',

**....:** 'two', 'two', 'one', 'three'],

**....:** 'C' : np.random.randn(8),

**....:** 'D' : np.random.randn(8)})

**....:**

**In [87]:** df

Out[87]:

A B C D

0 foo one -1.202872 -0.055224

1 bar one -1.814470 2.395985

2 foo two 1.018601 1.552825

3 bar three -0.595447 0.166599

4 foo two 1.395433 0.047609

5 bar two -0.392670 -0.136473

6 foo one 0.007207 -0.561757

7 foo three 1.928123 -1.623033

1、 分组并对每个分组执行sum函数：

**In [88]:** df.groupby('A').sum()

Out[88]:

C D

A

bar -2.802588 2.42611

foo 3.146492 -0.63958

2、 通过多个列进行分组形成一个层次索引，然后执行函数：

**In [89]:** df.groupby(['A','B']).sum()

Out[89]:

C D

A B

bar one -1.814470 2.395985

three -0.595447 0.166599

two -0.392670 -0.136473

foo one -1.195665 -0.616981

three 1.928123 -1.623033

two 2.414034 1.600434

Reshaping

Stack

**In [90]:** tuples = list(zip(\*[['bar', 'bar', 'baz', 'baz',

**....:** 'foo', 'foo', 'qux', 'qux'],

**....:** ['one', 'two', 'one', 'two',

**....:** 'one', 'two', 'one', 'two']]))

**....:**

**In [91]:** index = pd.MultiIndex.from\_tuples(tuples, names=['first', 'second'])

**In [92]:** df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=['A', 'B'])

**In [93]:** df2 = df[:4]

**In [94]:** df2

Out[94]:

A B

first second

bar one 0.029399 -0.542108

two 0.282696 -0.087302

baz one -1.575170 1.771208

two 0.816482 1.100230

.

**In [95]:** stacked = df2.stack()

**In [96]:** stacked

Out[96]:

first second

bar one A 0.029399

B -0.542108

two A 0.282696

B -0.087302

baz one A -1.575170

B 1.771208

two A 0.816482

B 1.100230

dtype: float64

**In [97]:** stacked.unstack()

Out[97]:

A B

first second

bar one 0.029399 -0.542108

two 0.282696 -0.087302

baz one -1.575170 1.771208

two 0.816482 1.100230

**In [98]:** stacked.unstack(1)

Out[98]:

second one two

first

bar A 0.029399 0.282696

B -0.542108 -0.087302

baz A -1.575170 0.816482

B 1.771208 1.100230

**In [99]:** stacked.unstack(0)

Out[99]:

first bar baz

second

one A 0.029399 -1.575170

B -0.542108 1.771208

two A 0.282696 0.816482

B -0.087302 1.100230

数据透视表

**In [100]:** df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] \* 3,

**.....:** 'B' : ['A', 'B', 'C'] \* 4,

**.....:** 'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] \* 2,

**.....:** 'D' : np.random.randn(12),

**.....:** 'E' : np.random.randn(12)})

**.....:**

**In [101]:** df

Out[101]:

A B C D E

0 one A foo 1.418757 -0.179666

1 one B foo -1.879024 1.291836

2 two C foo 0.536826 -0.009614

3 three A bar 1.006160 0.392149

4 one B bar -0.029716 0.264599

5 one C bar -1.146178 -0.057409

6 two A foo 0.100900 -1.425638

7 three B foo -1.035018 1.024098

8 one C foo 0.314665 -0.106062

9 one A bar -0.773723 1.824375

10 two B bar -1.170653 0.595974

11 three C bar 0.648740 1.167115

可以从这个数据中轻松的生成数据透视表：

**In [102]:** pd.pivot\_table(df, values='D', index=['A', 'B'], columns=['C'])

Out[102]:

C bar foo

A B

one A -0.773723 1.418757

B -0.029716 -1.879024

C -1.146178 0.314665

three A 1.006160 NaN

B NaN -1.035018

C 0.648740 NaN

two A NaN 0.100900

B -1.170653 NaN

C NaN 0.536826

九、导入和保存数据

CSV

*1、 写入csv文件：*

**In [136]:** df.to\_csv('foo.csv')

*2、 从csv文件中读取：*

**In [137]:** pd.read\_csv('foo.csv')

Out[137]:

Unnamed: 0 A B C D

0 2000-01-01 0.266457 -0.399641 -0.219582 1.186860

1 2000-01-02 -1.170732 -0.345873 1.653061 -0.282953

2 2000-01-03 -1.734933 0.530468 2.060811 -0.515536

3 2000-01-04 -1.555121 1.452620 0.239859 -1.156896

4 2000-01-05 0.578117 0.511371 0.103552 -2.428202

5 2000-01-06 0.478344 0.449933 -0.741620 -1.962409

6 2000-01-07 1.235339 -0.091757 -1.543861 -1.084753

.. ... ... ... ... ...

993 2002-09-20 -10.628548 -9.153563 -7.883146 28.313940

994 2002-09-21 -10.390377 -8.727491 -6.399645 30.914107

995 2002-09-22 -8.985362 -8.485624 -4.669462 31.367740

996 2002-09-23 -9.558560 -8.781216 -4.499815 30.518439

997 2002-09-24 -9.902058 -9.340490 -4.386639 30.105593

998 2002-09-25 -10.216020 -9.480682 -3.933802 29.758560

999 2002-09-26 -11.856774 -10.671012 -3.216025 29.369368

[1000 rows x 5 columns]

HDF5

1、 写入HDF5存储：

**In [138]:** df.to\_hdf('foo.h5','df')

2、 从HDF5存储中读取：

**In [139]:** pd.read\_hdf('foo.h5','df')

Out[139]:

A B C D

2000-01-01 0.266457 -0.399641 -0.219582 1.186860

2000-01-02 -1.170732 -0.345873 1.653061 -0.282953

2000-01-03 -1.734933 0.530468 2.060811 -0.515536

2000-01-04 -1.555121 1.452620 0.239859 -1.156896

2000-01-05 0.578117 0.511371 0.103552 -2.428202

2000-01-06 0.478344 0.449933 -0.741620 -1.962409

2000-01-07 1.235339 -0.091757 -1.543861 -1.084753

... ... ... ... ...

2002-09-20 -10.628548 -9.153563 -7.883146 28.313940

2002-09-21 -10.390377 -8.727491 -6.399645 30.914107

2002-09-22 -8.985362 -8.485624 -4.669462 31.367740

2002-09-23 -9.558560 -8.781216 -4.499815 30.518439

2002-09-24 -9.902058 -9.340490 -4.386639 30.105593

2002-09-25 -10.216020 -9.480682 -3.933802 29.758560

2002-09-26 -11.856774 -10.671012 -3.216025 29.369368

[1000 rows x 4 columns]

Excel

1、 写入excel文件：

**In [140]:** df.to\_excel('foo.xlsx', sheet\_name='Sheet1')

2、 从excel文件中读取：

**In [141]:** pd.read\_excel('foo.xlsx', 'Sheet1', index\_col=None, na\_values=['NA'])

Out[141]:

A B C D

2000-01-01 0.266457 -0.399641 -0.219582 1.186860

2000-01-02 -1.170732 -0.345873 1.653061 -0.282953

2000-01-03 -1.734933 0.530468 2.060811 -0.515536

2000-01-04 -1.555121 1.452620 0.239859 -1.156896

2000-01-05 0.578117 0.511371 0.103552 -2.428202

2000-01-06 0.478344 0.449933 -0.741620 -1.962409

2000-01-07 1.235339 -0.091757 -1.543861 -1.084753

... ... ... ... ...

2002-09-20 -10.628548 -9.153563 -7.883146 28.313940

2002-09-21 -10.390377 -8.727491 -6.399645 30.914107

2002-09-22 -8.985362 -8.485624 -4.669462 31.367740

2002-09-23 -9.558560 -8.781216 -4.499815 30.518439

2002-09-24 -9.902058 -9.340490 -4.386639 30.105593

2002-09-25 -10.216020 -9.480682 -3.933802 29.758560

2002-09-26 -11.856774 -10.671012 -3.216025 29.369368

[1000 rows x 4 columns]

## 9.4 机器学习包sk-learn

### 9.4.1 总体说明

Scikit-Learn是基于Python的开源机器学习模块，最早由David Cournapeau 在2007 年发起的，目前也是由社区自愿者进行维护。官方网站是http://scikit-learn.org/stable/，在上面可以找到相关的Scikit-Learn的资源、模块下载、文档、例程等。

Scikit-Learn的安装需要numpy，scipy，matplotlib等模块，Windows系统可以在http://www.lfd.uci.edu/~gohlke/pythonlibs直接下载编译好的安装包以及依赖包，也可以到网址下载http://sourceforge.jp/projects/sfnet\_scikit-learn/。

scikit-learn的基本功能主要被分为六个部分：分类，回归，聚类，数据降维，模型选择，数据预处理。对于具体的机器学习问题，通常可以分为三个步骤，数据准备与预处理，模型选择与训练，模型验证与参数调优。

### 9.4.2 代表性函数使用介绍

**1.加载数据(Data Loading)**

我们假设输入是一个特征矩阵或者csv文件。首先，数据应该被载入内存中。scikit-learn的实现使用了NumPy中的arrays，所以，我们要使用NumPy来载入csv文件。以下是从UCI机器学习数据仓库中下载的数据。

import numpy as np

import urllib

# url with dataset

url = "http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"

# download the file

raw\_data = urllib.urlopen(url)

# load the CSV file as a numpy matrix

dataset = np.loadtxt(raw\_data, delimiter=",")

# separate the data from the target attributes

X = dataset[:,0:7]

y = dataset[:,8]

我们要使用该数据集作为例子，将特征矩阵作为X，目标变量作为y。

**2.数据归一化(Data Normalization)**

大多数机器学习算法中的梯度方法对于数据的缩放和尺度都是很敏感的，在开始跑算法之前，我们应该进行归一化或者标准化的过程，这使得特征数据缩放到0-1范围中。scikit-learn提供了归一化的方法：

from sklearn import preprocessing

# normalize the data attributes

normalized\_X = preprocessing.normalize(X)

# standardize the data attributes

standardized\_X = preprocessing.scale(X)

**3.特征选择(Feature Selection)**

在解决一个实际问题的过程中，选择合适的特征或者构建特征的能力特别重要。这成为特征选择或者特征工程。  
特征选择时一个很需要创造力的过程，更多的依赖于直觉和专业知识，并且有很多现成的算法来进行特征的选择。  
 下面的树算法(Tree algorithms)计算特征的信息量：

from sklearn import metrics

from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()

model.fit(X, y)

# display the relative importance of each attribute

print(model.feature\_importances\_)

### 9.4.3 机器学习算法的使用

scikit-learn实现了机器学习的大部分基础算法，让我们快速了解一下。

**1.逻辑回归**

大多数问题都可以归结为二元分类问题。这个算法的优点是可以给出数据所在类别的概率。

from sklearn import metrics

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression()

model.fit(X, y)

print(model)

# make predictions

expected = y

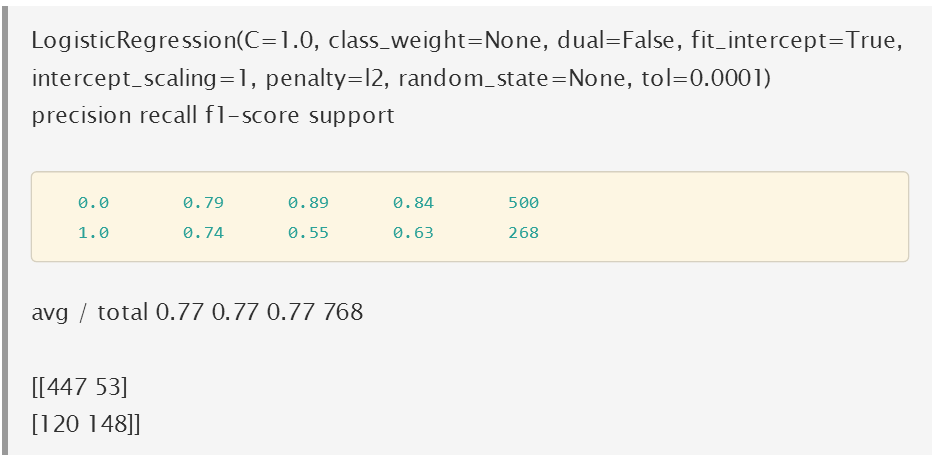
predicted = model.predict(X)

# summarize the fit of the model

print(metrics.classification\_report(expected, predicted))

print(metrics.confusion\_matrix(expected, predicted))

**结果：**



**2.朴素贝叶斯**

这也是著名的机器学习算法，该方法的任务是还原训练样本数据的分布密度，其在多类别分类中有很好的效果。

from sklearn import metrics

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(X, y)

print(model)

# make predictions

expected = y

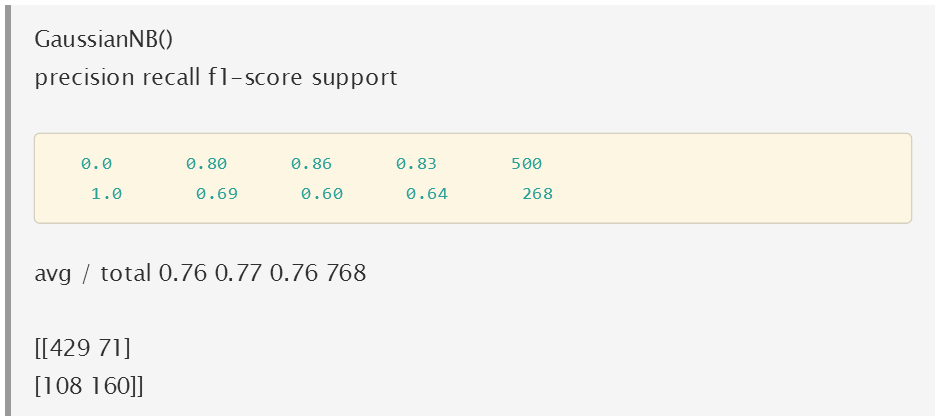
predicted = model.predict(X)

# summarize the fit of the model

print(metrics.classification\_report(expected, predicted))

print(metrics.confusion\_matrix(expected, predicted))

**结果：**



**3.k近邻**

k近邻算法常常被用作是分类算法一部分，比如可以用它来评估特征，在特征选择上我们可以用到它。

from sklearn import metrics

from sklearn.neighbors import KNeighborsClassifier

# fit a k-nearest neighbor model to the data

model = KNeighborsClassifier()

model.fit(X, y)

print(model)

# make predictions

expected = y

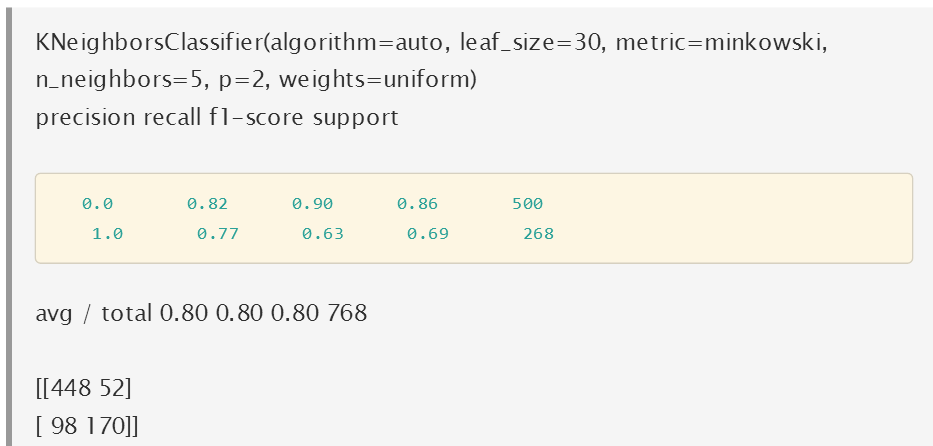
predicted = model.predict(X)

# summarize the fit of the model

print(metrics.classification\_report(expected, predicted))

print(metrics.confusion\_matrix(expected, predicted))

**结果：**



**4.决策树**

分类与回归树(Classification and Regression Trees ,CART)算法常用于特征含有类别信息的分类或者回归问题，这种方法非常适用于多分类情况。

from sklearn import metrics

from sklearn.tree import DecisionTreeClassifier

# fit a CART model to the data

model = DecisionTreeClassifier()

model.fit(X, y)

print(model)

# make predictions

expected = y

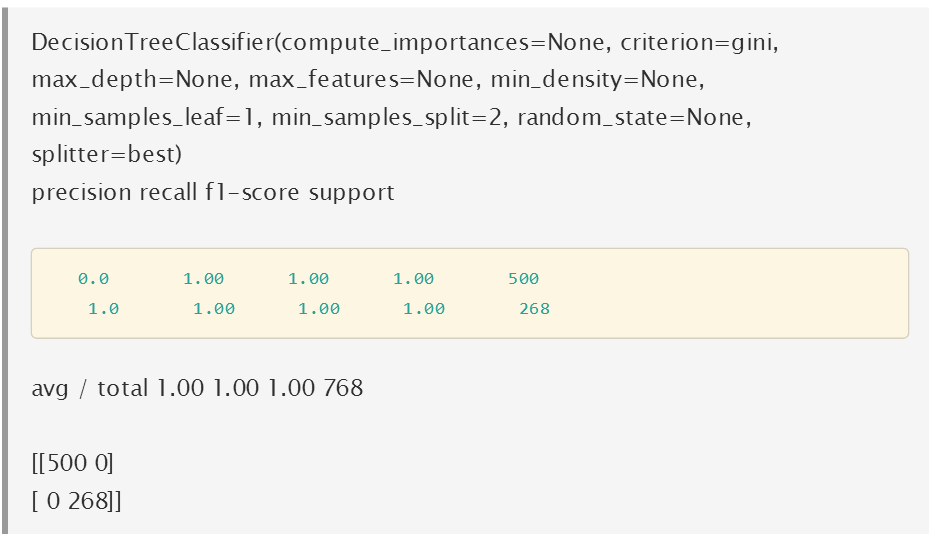
predicted = model.predict(X)

# summarize the fit of the model

print(metrics.classification\_report(expected, predicted))

print(metrics.confusion\_matrix(expected, predicted))

**结果：**



**5.支持向量机**

SVM是非常流行的机器学习算法，主要用于分类问题，如同逻辑回归问题，它可以使用一对多的方法进行多类别的分类。

from sklearn import metrics

from sklearn.svm import SVC

# fit a SVM model to the data

model = SVC()

model.fit(X, y)

print(model)

# make predictions

expected = y

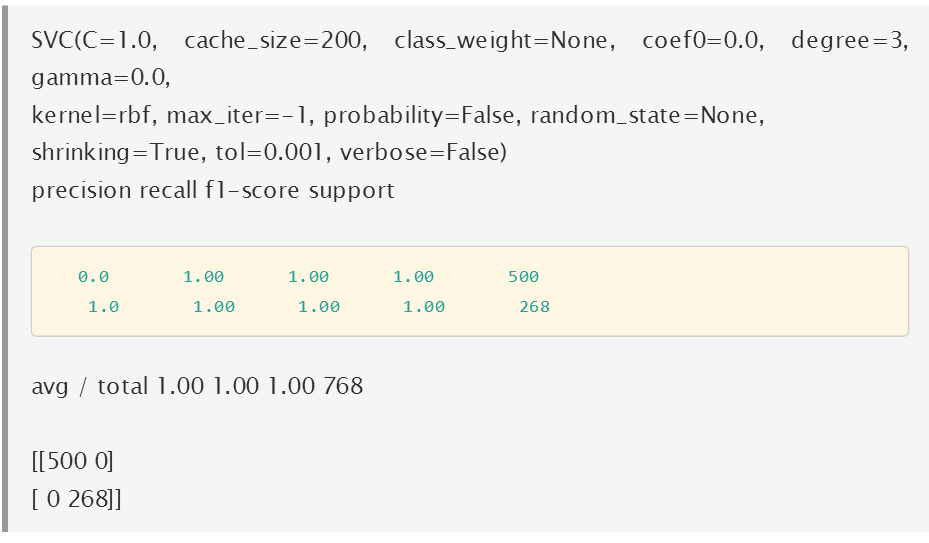
predicted = model.predict(X)

# summarize the fit of the model

print(metrics.classification\_report(expected, predicted))

print(metrics.confusion\_matrix(expected, predicted))

**结果：**



除了分类和回归算法外，scikit-learn提供了更加复杂的算法，比如聚类算法，还实现了算法组合的技术，如Bagging和Boosting算法。

### 9.4.4 如何优化算法参数

一项更加困难的任务是构建一个有效的方法用于选择正确的参数，我们需要用搜索的方法来确定参数。scikit-learn提供了实现这一目标的函数。  
 下面的例子是一个进行正则参数选择的程序：

import numpy as np

from sklearn.linear\_model import Ridge

from sklearn.grid\_search import GridSearchCV

# prepare a range of alpha values to test

alphas = np.array([1,0.1,0.01,0.001,0.0001,0])

# create and fit a ridge regression model, testing each alpha

model = Ridge()

grid = GridSearchCV(estimator=model, param\_grid=dict(alpha=alphas))

grid.fit(X, y)

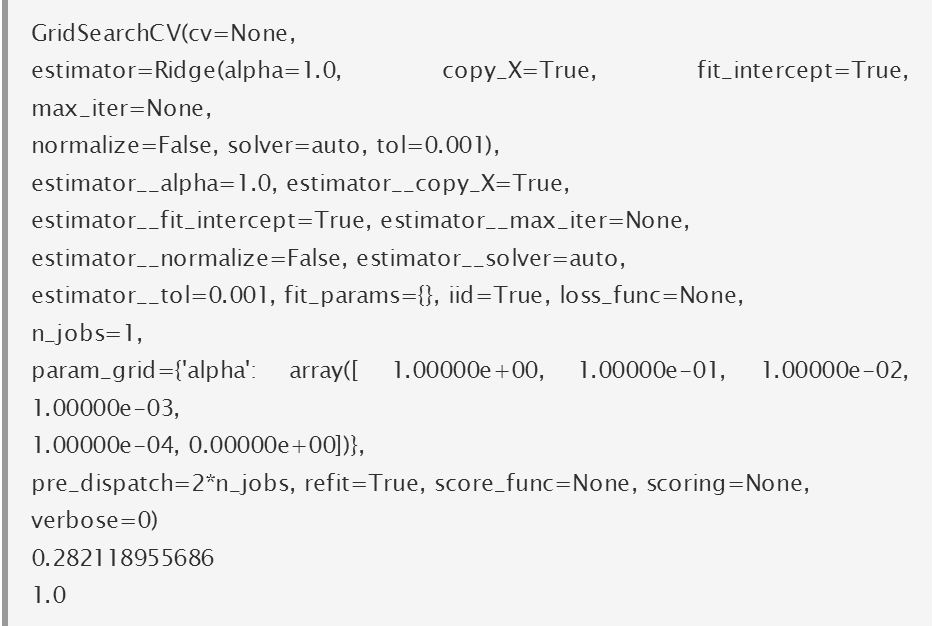
print(grid)

# summarize the results of the grid search

print(grid.best\_score\_)

print(grid.best\_estimator\_.alpha)

**结果：**



有时随机从给定区间中选择参数是很有效的方法，然后根据这些参数来评估算法的效果进而选择最佳的那个。

import numpy as np

from scipy.stats import uniform as sp\_rand

from sklearn.linear\_model import Ridge

from sklearn.grid\_search import RandomizedSearchCV

# prepare a uniform distribution to sample for the alpha parameter

param\_grid = {'alpha': sp\_rand()}

# create and fit a ridge regression model, testing random alpha values

model = Ridge()

rsearch = RandomizedSearchCV(estimator=model, param\_distributions=param\_grid, n\_iter=100)

rsearch.fit(X, y)

print(rsearch)

# summarize the results of the random parameter search

print(rsearch.best\_score\_)

print(rsearch.best\_estimator\_.alpha)

**结果：**

