Python 数据分析与数据挖掘（Python for Data Analysis&Data Mining）

Chap 3 Python数据分析和挖掘工具库

内容：

- Python数据分析与数据挖掘相关库

实践：

- 库的导入和添加
- NumPy
- Pandas
- SciPy
- Matplotlib
- StatsModels
- Scikit（即Sklearn或Scikit-learn）
- Keras
- Gensim

1. 库的导入和添加

库的导入模式有如下几种：

- import math # 采用math.sin()方式引用
- import math as m # m.sin()引用
- from math import exp as e # 如果不需要导入math库中的其他函数，只导入math库中的exp函数，e(1) 计算指数
- from math import * # 直接的导入，去掉math， exp(1)，容易引起命名冲突

建议用法：

对于导入库起个常用的别名

- `import numpy as np`
- `import pandas as pd`
- `import matplotlib.pyplot as plt`
- `import math as m`

在Python软件开发过程中，：

- 不建议直接引入类似NumPy这种大型库的全部内容（即，不建议 `from numpy import *`），如果大量引入第三方库，容易引起命名冲突。

当看到np.arange时，就应该想到它引用的是NumPy中的arange函数。

In [1]:
```
# 必要准备工作：导入库，配置环境等
# 导入python特征（为Python2）
# 2版本中3/2=1， 3版本中3/2=1.5， 3//2=1
from __future__ import division

br = '\n'
```

In [2]:
```
# 导入库并为库起个别名
import numpy as np
import pandas as pd
np.set_printoptions(precision=4, suppress=True)
```

In [3]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
```

Python数据分析和数据挖掘中常使用到的相关扩展库

扩展库	简介
Numpy	提供数组支持，以及相应的高效处理函数
Scipy	提供矩阵支持，以及矩阵相关的数值计算模块
Matplotlib	强大的数据可视化工具、作图库
Pandas	强大、灵活的数据分析和探索工具
StatsModels	统计建模和计量经济学，包括描述统计、统计模型估计和推断
Scikit-learn	支持回归、分类、聚类等的强大的机器学习库
NLTK	自然语言处理相关的语料和技术支持库
Gensim	文本主题模型LDA、文本相似度计算、Google的Word2Vec等的库

此外，Keras（深度学习库，建立神经网络模型以及深度学习模型），视频处理的OpenSC等等。

练习： 进行库的安装和更新

检查是否安装StatsModels库

注意：下面命令，如果库没有安装，则会进行下载安装，后台需要花时间

```
!pip install statsmodels
```

Python 中的list列表对象与Numpy的ndarray类型

Python没有提供数组功能，虽然列表可以完成基本的数组功能，但数据量较大时，列表的速度就会慢得难以接受。

NumPy提供了真正的数组功能，以及对数组进行快速处理的函数。而且NumPy还是很多高级的扩展库的依赖库，其他SciPy、Matplotlib、Pandas等库都依赖于它。NumPy内置函数处理数据的速度都是C语言级别的，因此，尽量使用内置函数。

1. NumPy的ndarray: 一种多维数组对象(a multidimensional array object)

NumPy，是 Numerical Python 的简称，是Python中高性能科学计算和数据分析的基础包。是数据分析中几乎所有高级工具的构建基础。

NumPy最重要的一个特点就是其N维数组对象（即ndarray），该对象是一个快速而灵活的大数据集容器。可以利用这种数组对整块数据执行一些数学运算，其语法跟标量元素之间的运算一样。 ndarray是一个通用的同构数据多维容器，也就是说，其中的所有元素必须是相同类型的。每个数组都有一个shape（一个表示各维度大小的元组）和一个dtype（一个用于说明数组数据类型的对象）

对于数值型数据，NumPy数组在存储和处理数据时要比内置的Python数据结构高效得多。其部分功能如下：

- ndarray，一个具有矢量算术运算和复杂广播能力的快速高效且节省空间的多维数组
- 用于对数组执行快速元素级计算的标准数学函数（无需编写循环）
- 用于读写硬盘上基于数组的数据的工具以及用于操作内存映射文件的工具
- 线性代数运算、傅立叶变换，以及随机数生成等功能
- 用于将C、C++、Fortran代码集成到Python的工具（由C和Fortran编写的库可以直接操作NumPy数组中的数据，无需进行数据复制工作）
- 作为在算法之间传递数据的容器

NumPy本身并没有提供多么高级的数据分析功能，理解NumPy数组以及面向数组的计算将有助于更加高效的使用诸如pandas之类的工具。

对于大部分数据分析应用而言，我们最关注的功能主要集中在：

- 用于数据整理和清理、子集构造和过滤、转换等快速的矢量化数组运算
- 常用的数组算法，如排序、唯一化、集合运算等
- 高效的描述统计和数据聚合/摘要运算
- 用于异构数据集的合并/连接运算的数据对齐和关系型数据运算
- 将条件逻辑表述为数组表达式（而不是带有`if-elif-else`分支的循环）
- 数组的分组运算（聚合、转换、函数应用等）

主要介绍NumPy数组的基本用法。精通面向数组的编程和思维方式是成为Python数据分析达人的关键步骤。

注意： 按照标准的NumPy约定，我们总是使用`import numpy as np`， 尽量不要使用 `from numpy import *`

In [4]:
```
# 导入numpy库并给个别名
import numpy as np
```

1) 创建ndarrays （ Creating ndarrays ）

创建数组最简单的办法一是使用array函数。它接受一切序列型的对象（包括其他数组），然后产生一个新的含有传入数据的NumPy数组。

- Python列表数据被 np.array 转换为一个NumPy数组
- 嵌套序列（有一组等长列表组成的列表）将会被转换为一个多维数组

除法显示说明，np.array 会尝试给新建的这个数组推断出一个较为合适的数据类型，数据类型保存在一个特殊的dtype对象中。

除了np.array之外， 还有一些函数可以新创建数组。比如：

- zeros和ones分别可以创建指定长度或形状的全0或全1数组。
- empty可以创建一个没有任何具体值的数组;**注意：empty并不是返回全0数组。通常返回的都是一些未初始化的垃圾值。**
- arange 是Python内置函数range的数组版。 **注意：** 要用这些方法创建多维数组，只需传入一个表示形状的元组即可。

In [5]:
```
#Python标准的列表
a = [6, 7, 8, 0, 1]
print type(a), a
```

```
<type 'list'> [6, 7, 8, 0, 1]
```

In [6]:
```
# NumPy的array函数将列表转换为NumPy数组
arr1 = np.array(a)
print type(arr1)
arr1
```

```
<type 'numpy.ndarray'>
```

Out[6]:
```
array([6, 7, 8, 0, 1])
```

In [7]:
```
# 嵌套序列（等长列表组成的列表）将会被转换为一个多维数组
# 创建多维数组
b = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(b)
print type(arr2)
arr2
```

```
<type 'numpy.ndarray'>
```

Out[7]:
```
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

In [8]:
```
# 多维数组的维度
arr2.ndim
```

Out[8]:
```
2
```

In [9]:
```
# 多维数组的形状shape
arr2.shape
```

Out[9]:
```
(2L, 4L)
```

In [10]:
```
# 数组类型
print arr1.dtype
print arr2.dtype
```

```
int32
int32
```

In [11]:
```
# 创建全0或全1的一维数组，需要输入shape
np.zeros(5)
```

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

In [12]:
```
# 创建全0或1的多维数组
np.ones((3, 4))
```

Out[12]:
```
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

In [13]:
```
# empty创建一个没有任何具体值的数组;并不是返回全0数组。通常返回的都是一些未初始化的垃圾值。
np.empty((3,2))
```

Out[13]:
```
array([[ -1.1380e-054,   8.2865e-314],
       [  2.5219e+249,   1.4854e-313],
       [  1.3944e-258,   2.3378e-313]])
```

In [14]:
```
# arange是Python内置函数range的数组版
a = range(5)
print type(a), a
b = np.array(a)
print type(b), b
```

```
<type 'list'> [0, 1, 2, 3, 4]
<type 'numpy.ndarray'> [0 1 2 3 4]
```

In [15]:
```
c = np.zeros(5)
print type(c), c
d= np.ones(5)
print type(d), d
```

```
<type 'numpy.ndarray'> [ 0.  0.  0.  0.  0.]
<type 'numpy.ndarray'> [ 1.  1.  1.  1.  1.]
```

In [16]:
```
# 将多个ndarray数组进行合并
d = np.array((a,b,c))
print type(d)
d
```

```
<type 'numpy.ndarray'>
```

Out[16]:
```
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  1.,  2.,  3.,  4.],
       [ 0.,  0.,  0.,  0.,  0.]])
```

In [17]:
```
np.eye(3)
```

Out[17]:
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

下表列出了数组创建函数。

由于NumPy关注的是数值计算，如果没有特别指定，数据类型基本都是float64（浮点数）

函数	说明
array	将输入数据（列表，元组，数组或其他序列类型）转换为ndarray。要么推断出dtype，那么显示指定dtype。默认直接复制数据
asarray	将输入转换为ndarray，如果输入本身就是ndarray就不进行复制
arange	类似于内置的range，但返回的是一个ndarray而不是列表
ones、ones_like	根据指定的形状和dtype创建一个全1数据。ones_like以另一个数组为参数，并根据其形状和dtype创建一个全1数组
zeros、zeros_like	类似于ones和ones_like，只不过产生的是全0数组而已
empty、empty_line	创建新数组，只分配内存空间但不填充任何值
eye、identity	创建一个正方的 $N * N$ 单位矩阵（对角线为1， 其余为0）

2) ndarray的数据类型（Data Types for ndarrays）

dtype（数据类型）是一个特殊的对象，它含有ndarray将一块内存解释为特定数据类型所需的信息。

dtype是NumPy如此强大和灵活的原因之一。多数情况下，它们直接映射到响应的机器表示，使得“读写磁盘上的二进制数据流”和“集成低级语言代码（如C、Fortran）”等工作变得更加简单。

数值型dtype的命名方式相同：一个类型名（如float或int），后面跟一个用于表示各元素位长的数字。标准的双精度浮点值（即Python中的float对象）需要占用8字节（即64位）。因此，该类型在NumPy中就记作float64。

可以通过ndarray的astype方法显示的转换dtype。

注意： 调用astype如论如何都会创建出一个新的数组（原始数据的一份拷贝），即使新的dtype跟老的dtype相同也是如此。

注意：浮点数（比如float64和float32）只能表示近似的分数值。在复杂计算中，由于可能会积累一些浮点错误，因此比较操作只能在一定小数位以内有效。

下面的表列出了NumPy所支持的全部数据类型。

类型	类型代码	说明
int8、uint8	i1, u1	有符号和无符号的8位（1个字节）整型
int16、uint16	i2, u2	有符号和无符号的16位（2个字节）整型
int32、uint32	i4, u4	有符号和无符号的32位（4个字节）整型
int64、uint64	i8, u8	有符号和无符号的64位（8个字节）整型
float16	f2	半精度浮点数
float32	f4或f	标准的单精度浮点数。与C的float兼容
float64	f8或d	标准的双精度浮点数。与C的double和Python的float对象兼容
float128	f16或g	扩展精度浮点数
complex64、complex128 、complex256	c8、c16、c32	分别用两个32位、64位或128位浮点数表示的复数
bool	?	存储True和False值的布尔类型
object	O	Python对象类型
string_	S	固定长度的字符串类型（每个字符1个字节）。例如，要创建一个长度为10的字符串，应使用S10
unicode_	U	固定长度的unicode类型（字节数由平台决定）。跟字符串的定义方式一样（如U10）

In [18]:
```
# 创建数组时，指定数组类型
arr1 = np.array([1, 2, 3], dtype=np.float64)
arr2 = np.array([1, 2, 3], dtype=np.int32)
print arr1.dtype, arr2.dtype
```

```
float64 int32
```

In [19]:
```
# 可以通过ndarray的astype方法显示的转换dtype，整数转成浮点数
arr = np.array([1, 2, 3, 4, 5])
print arr.dtype

float_arr = arr.astype(np.float64)
print float_arr.dtype,float_arr
```

```
int32
float64 [ 1.  2.  3.  4.  5.]
```

In [20]:
```
# 浮点数转成整数，则小数部分直接被截断
arr = np.array([3.7, -1.2, -2.6, 0.5, 12.9, 10.1])
arr
arr.astype(np.int32)
```

Out[20]:
```
array([ 3, -1, -2,  0, 12, 10])
```

In [21]:
```
# 字符串类型如果都是数字，也可以转换为数值形式
numeric_strings = np.array(['1.25', '-9.6', '42'], dtype=np.string_)
numeric_strings.astype(float)
```

Out[21]:
```
array([  1.25,  -9.6 ,  42.  ])
```

In [22]:
```
# 把一个数组的dtype转换为另外一个数组的数据类型 A.astype(B.dtype),将B的数据类型赋予A
int_array = np.arange(10)
calibers = np.array([.22, .270, .357, .380, .44, .50], dtype=np.float64)
int_array.astype(calibers.dtype)
```

Out[22]:
```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.])
```

In [23]:
```
# 可以使用简洁的类型代码来表示dtype
empty_uint32 = np.empty(8, dtype='u4')
empty_uint32
```

Out[23]:
```
array([1, 0, 2, 0, 3, 0, 4, 0], dtype=uint32)
```

3 数组和标量之间的运算

（ Operations between arrays and scalars ）

数组很重要，可以使我们不用编写循环即可对数据执行批量运算。这通常叫做矢量化（vectorization）。

- 大小相等的数组之间的任何算术运算都将运算应用到元素级，加减乘除和方乘
- 数组与标量的算术运算也会将那个标量值传播到各个元素。
- 不同大小的数组之间的运算叫做广播（broadcasting）

In [24]:
```
# 等大小数组的算术运算
arr = np.array([[1., 2., 3.], [4., 5., 6.]])
print 'arr: ', arr
print 'sum: ', arr + arr
print 'multiplication: ', arr * arr
print 'difference: ', arr - arr
print 'inverse: ', 1 / arr
print 'squared root:', arr ** 0.5
```

```
arr:  [[ 1.  2.  3.]
 [ 4.  5.  6.]]
sum:  [[  2.   4.   6.]
 [  8.  10.  12.]]
multiplication:  [[  1.   4.   9.]
 [ 16.  25.  36.]]
difference:  [[ 0.  0.  0.]
 [ 0.  0.  0.]]
inverse:  [[ 1.      0.5     0.3333]
 [ 0.25    0.2     0.1667]]
squared root: [[ 1.      1.4142  1.7321]
 [ 2.      2.2361  2.4495]]
```

4）基本的索引和切片（Basic indexing and slicing）

NumPy数组的索引是一个内容丰富的主题，因为选取数据子集或单个元素的方式有很多。

> 一维数组很简单，从表面看，和Python列表的功能差不多。将一个标量值赋值给一个切片，该值会自动传播（即广播）到整个选区。但是跟列表最重要的区别是， 数组切片是原始数组的视图。这意味着数据不会被复制，视图上的任何修改都会直接反映到源数组上。这是因为NumPy的设计目的是处理大数据，可以想象一下，如果坚持将数据复制的话会产生何等的性能和内存问题。 **注意：** 如果要想得到的是ndarray切片的一份副本而不是视图，就需要显示地进行复制操作。`arr[5:8].copy()`
>
> 对于高维数组，各索引位置上的元素不再是标量，而是数组。可以传入一个以逗号隔开的索引列表来选取单个元素，例如`arr[0][2] = arr[0,2]`这两个方式等价，都是取数组中第一个维度的第三个位置的元素。在多维数组中，如果省略了后面的索引，则返回对象是一个维度低一些的ndarray（它含有高一级维度上的所有数据）。

In [25]:
```
arr = np.arange(10)
print arr
print arr[5]
print arr[5:8]

# 将一个标量值赋值给一个切片，该值会自动传播（即广播）到整个选区。
arr[5:8] = 12
print arr
```

```
[0 1 2 3 4 5 6 7 8 9]
5
[5 6 7]
[ 0  1  2  3  4 12 12 12  8  9]
```

In [26]:
```
# 数组切片操作直接修改源数组
arr_slice = arr[5:8]
arr_slice[1] = 12345
print arr

arr_slice[:] = 64
print arr
```

```
[    0     1     2     3     4    12 12345    12     8     9]
[ 0  1  2  3  4 64 64 64  8  9]
```

In [27]:
```
# 对于高维数组，各索引位置上的元素不再是标量，而是一个数组
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr2d[2]
```

Out[27]:
```
array([7, 8, 9])
```

In [28]:
```
# 对于高维数组，对索引递归可以对元素进行访问。下面两个方式是等价访问数组中的一个元素
print arr2d[0][2]
print arr2d[0, 2]
```

```
3
3
```

In [29]:
```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
arr3d
```

Out[29]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

In [30]:
```
arr3d[0]
```

Out[30]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [31]:
```
old_values = arr3d[0].copy()
# 标量值和数组都可以被赋值给ndarray
arr3d[0] = 42
print arr3d
print

arr3d[0] = old_values
print arr3d
```

```
[[[42 42 42]
  [42 42 42]]

 [[ 7  8  9]
  [10 11 12]]]

[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
```

In [32]:
```
#在多维数组中，如果省略了后面的索引，则返回对象是一个维度低一些的ndarray
#（它含有高一级维度上的所有数据）。
arr3d[1, 0]
```

Out[32]: array([7, 8, 9])

2) 数学和统计方法（ Mathematical and statistical methods ）

可以通过数组上的一组数学函数对整个数组或某个轴向的数据进行统计计算。

sum、mean以及标准差std等聚合计算即可以当做数组的实例方法调用(arr.mean())，也可以当做顶级NumPy函数使用(np.mean(arr))。

- arr.mean()与np.mean(arr)相等，都是返回arr数组的均值mean。
- arr.sum()与np.sum(arr)相等，都是返回arr数组的加和sum。
- arr.std()与np.std(arr)相等，都是返回arr数组的标准差std。

mean和sum这类函数可以接受一个axis参数（用于计算该轴向上的统计值），最终返回结果是一个*n*-1维的数组（即少一维的数组）

下面的表列出全部的基本数组统计方法。

下表列出了基本数组统计方法

方法	说明
sum	对数组中全部或某轴向的元素求和。零长度的数组的sum为0
mean	算术平均数。零长度的数组的mean为NaN
std、var	分别为标准差和方差，自由度可调（默认为n）
min、max	最大值和最小值
argmin、argmax	分别为最大和最小元素的索引
cumsum	所有元素的累积和
cumprod	所有元素的累积乘积

In [33]:
```
# 生成正态分布的二维数组5*4
arr = np.random.randn(5, 4) # normally-distributed data
print arr,br
print arr.mean(),br
print np.mean(arr),br
print arr.sum(),np.sum(arr),br
print arr.std(), np.std(arr), br
```

```
[[-0.7889  2.0995 -1.5724  1.6151]
 [-1.9568  0.1783  0.2354 -0.5527]
 [-0.223   1.0075  0.5577  0.5363]
 [-0.5889  1.8966 -0.409  -0.3255]
 [ 1.0387  0.3123  1.3479 -0.3411]]

0.203342154888

0.203342154888

4.06684309776 4.06684309776

1.0605829394 1.0605829394
```

In [34]:
```
# mean可以接受一个axis参数（用于计算该轴向上的统计值），返回结果是n-1维的数组
# 在axis=0的轴上求均值，即合并第一个维度上的值，原先是5*4的二维，返回是4的一维
print arr[0],br  # 第一行数据
print arr[:,0],br  # 第一列数据

# 合并第一个维度的加和
print arr.sum(0),br

# 求第一个维度的均值
print arr.mean(axis=0),br

# 在第二个维度上求加和
print arr.sum(1),br
```

```
[-0.7889  2.0995 -1.5724  1.6151]

[-0.7889 -1.9568 -0.223  -0.5889  1.0387]

[-2.519   5.4942  0.1596  0.932 ]

[-0.5038  1.0988  0.0319  0.1864]

[ 1.3533 -2.0959  1.8785  0.5733  2.3577]
```

In [35]:
```
# 其他如cumsum和cumprod之类的方法则不聚合，而是产生一个由中间结果组成的数组。
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print arr,br

# 在第一个维度上的元素累积和，cumsum--所有元素的累积和
print arr.cumsum(0),br

# 在第二个维度上的元素累积乘积，cumprod--所有元素的累积乘积
print arr.cumprod(1),br
```

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

[[ 0  1  2]
 [ 3  5  7]
 [ 9 12 15]]

[[  0   0   0]
 [  3  12  60]
 [  6  42 336]]
```

2. SciPy

两个NumPy数组相乘时，只是对应元素相乘，并不是矩阵乘法。SciPy提供了真正的矩阵，以及大量基于矩阵运算的对象和函数。

In [36]:
```
# NumPy只是对应元素相乘
a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print a
print  a * a
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
```

In [37]:
```
# 导入Scipy库
import scipy
from scipy import linalg
```

In [38]:
```
# 真正的矩阵乘法
scipy.mat(a) * scipy.mat(a)
```

Out[38]:
```
matrix([[ 15,  18,  21],
        [ 42,  54,  66],
        [ 69,  90, 111]])
```

In [39]:
```
# 将NumPy的ndarray类型转换为矩阵
ma=scipy.mat(a)
print type(ma)
print ma
```

```
<class 'numpy.matrixlib.defmatrix.matrix'>
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [40]:
```
# 1.逆矩阵的求解
ma = scipy.mat(a) * scipy.mat(a)
mb = linalg.inv(ma)
print mb
```

```
[[ -1.5600e+14   3.1200e+14  -1.5600e+14]
 [  3.1200e+14  -6.2399e+14   3.1200e+14]
 [ -1.5600e+14   3.1200e+14  -1.5600e+14]]
```

In [41]:
```
ma * mb
```

Out[41]:
```
matrix([[ 1., -1.,  0.],
        [ 2.,  0.,  2.],
        [ 4., -4.,  4.]])
```

In [42]:
```
# 2.求行列式的值
linalg.det(ma)
```

Out[42]:
```
-9.592326932761341e-14
```

In [43]:
```
# 3.求ma的模
linalg.norm(ma)
```

Out[43]:
```
187.63794925334267
```

In [44]:
```
# 4.求特征值及特征向量
r,v = linalg.eig(ma)
print "Root: ", r
print "Vector: ", v
```

```
Root:  [ 178.1816+0.j    1.8184+0.j    0.0000+0.j]
Vector:  [[ 0.1648  0.7997  0.4082]
 [ 0.5058  0.1042 -0.8165]
 [ 0.8468 -0.5913  0.4082]]
```

In [45]:
```
# 5.LU矩阵分解
x,y,z=linalg.lu(ma)
print x
print 'L矩阵： ', y
print 'U矩阵： ', z
```

```
[[ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]]
L矩阵：  [[ 1.      0.      0.    ]
 [ 0.2174  1.      0.    ]
 [ 0.6087  0.5     1.    ]]
U矩阵：  [[ 69.      90.     111.    ]
 [  0.      -1.5652  -3.1304]
 [  0.       0.       0.    ]]
```

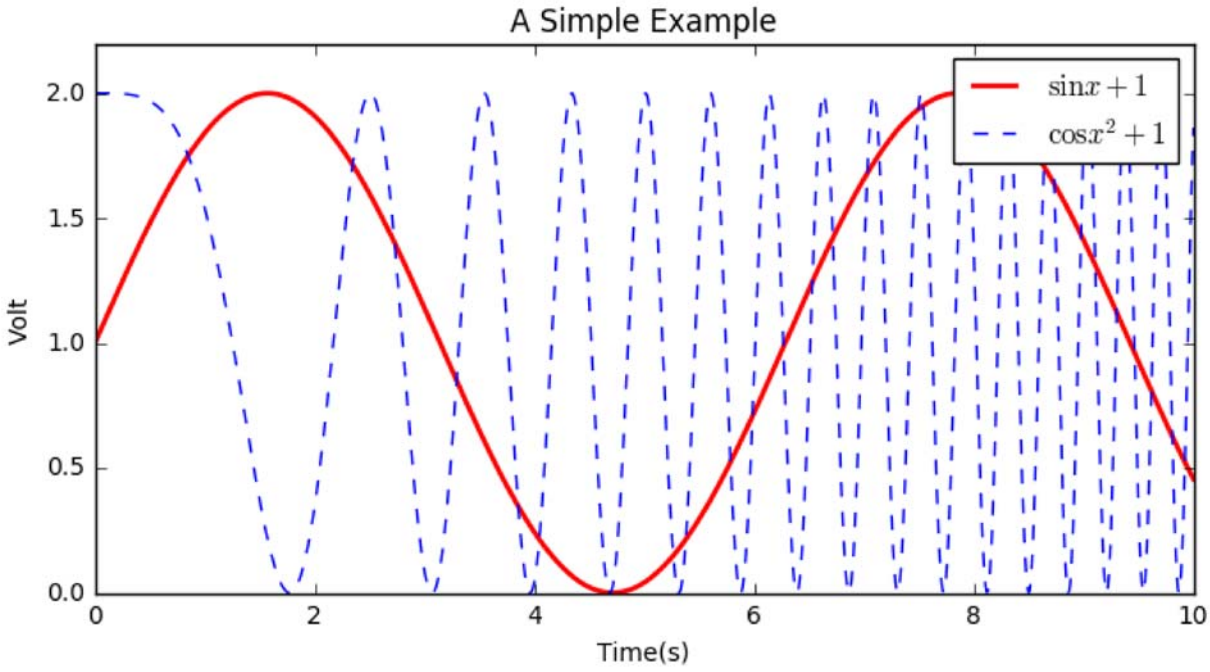
3. Matplotlib

Python中最著名的绘图库。主要用于二维绘图，用于数据的可视化

In [46]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
```

In [47]:
```
x = np.linspace(0, 10, 1000) # 自变量
y = np.sin(x) + 1  # 因变量
z = np.cos(x ** 2) + 1  # 因变量
```

In [48]:
```
plt.figure(figsize=(8,4))  # 设置图像大小
plt.plot(x,y,label='$\sin x+1$', color='red',linewidth=2) # 作图，设置标签，线条颜色大小
plt.plot(x,z,'b--', label='$\cos x^2+1$') # 作图，设置标签、线条类型
plt.xlabel('Time(s)')  # x轴名称
plt.ylabel('Volt')  # y轴名称
plt.title(u'A Simple Example')  # 标题
plt.ylim(0, 2.2)  # 显示的y轴范围
plt.legend()  # 显示图例
plt.show()  # 显示作图结果
```



4. Pandas

pandas是后续内容的首选库，Python中最强大的数据分析和探索工具。因为它含有使数据分析工作变得更快更简单的高级数据结构和操作工具。pandas是基于NumPy构建的，让以NumPy为中心的应用变得更加简单。

2008年时，pandas还无法提供数据分析的工具。在以后的几年中，pandas逐渐成长为一个非常大的库，能解决越来越多的数据分析问题，但是也逐渐背离了简洁性和易用性。

pandas名字源于panel data（面板数据，是计量经济学中关于多维结构化数据集的一个术语）以及Python data analysis（Python数据分析），很适合金融数据分析应用的工具

- 兼具NumPy高性能的数组计算功能以及电子表格和关系型数据库（如SQL）灵活的数据处理能力
- 提供了复杂精细的索引功能，以便更为便捷地完成重塑、切片和切块、聚合以及选取数据子集等操作
- 对于金融行业的用户，pandas提供了大量适用于金融数据的高性能时间序列功能和工具
- 用的最多的pandas对象是DataFrame，是一个面向列（column-oriented）的二维表结构，含有行标和列标

pandas能够满足的需求：

- 具备按轴自动或显式数据对齐功能的数据结构。这可以防止许多由于数据未对齐以及来自不同数据源（索引方式不同）的数据而导致的常见错误
- 集成时间序列功能
- 既能处理时间序列数据也能处理非时间序列数据的数据结构
- 数学运算和约简（比如对某个轴求和）可以根据不同的元数据（轴编号）执行
- 灵活处理缺失数据
- 合并及其他出现在常见数据库（例如基于SQL的）中的关系型运算

Pandas基本的数据结构是Series（序列，一维数组）和 DataFrame（二维数据表格，每列是一个Series）。 我们使用下面的pandas引入约定：

```
from pandas import Series, DataFrame
import pandas as pd
```

因为Series和DataFrame用的次数非常多，因此将其引入本地命名空间中会更方便

由于后续我们会频繁读取和写入Excel，而默认的Pandas还不能读写Excel文件，需要安装xlrd（读）和xlwt（写）库才能支持excel读写。安装方法：

`pip install xlrd, xlwt` 或者conda 安装

In [49]:
```
from pandas import Series, DataFrame
import pandas as pd
```

In [50]:
```
# 创建一个Series序列
s = Series([1,2,3], index=['a', 'b', 'c'])
s
```

Out[50]:
```
a    1
b    2
c    3
dtype: int64
```

In [51]:
```
# 创建一个表DataFrame
d = DataFrame([[1,2,3],[4,5,6]], columns=['a', 'b', 'c'])
d
```

Out[51]:

	a	b	c
0	1	2	3
1	4	5	6

In [52]:
```
# 使用已有的序列来创建表，series是表的一个列
d2 = DataFrame(s)
d2
```

Out[52]:

	0
a	1
b	2
c	3

In [53]:
```
d.describe()  # 数据的基本统计量
```

Out[53]:

	a	b	c
count	2.00000	2.00000	2.00000
mean	2.50000	3.50000	4.50000
std	2.12132	2.12132	2.12132
min	1.00000	2.00000	3.00000
25%	1.75000	2.75000	3.75000
50%	2.50000	3.50000	4.50000
75%	3.25000	4.25000	5.25000
max	4.00000	5.00000	6.00000

In [54]:
```
# 读取文件，注意文件的存储路径不能有中文，否则出错
df = pd.read_csv('data/WeatherResult.csv', header=None, encoding='GBK')#读取csv并创建DF
df.head() # 预览前5行数据
```

Out[54]:

	0	1	2	3	4
0	北京	18日（今天）	多云转小雪	2	-4
1	北京	19日（明天）	阴转晴	1	-8
2	北京	20日（后天）	晴转多云	0	-8
3	北京	21日（周六）	晴	1	-8
4	北京	22日（周日）	晴	-1	-9

In [55]:
```
df.tail()  # 预览最后5行数据
```

Out[55]:

	0	1	2	3	4
72	五大连池	23日（后天）	晴	-17	-31
73	五大连池	24日（周二）	晴	-14	-29
74	五大连池	25日（周三）	晴转多云	-11	-19
75	五大连池	26日（周四）	小雪	-8	-21
76	五大连池	27日（周五）	多云	-11	-22

In [56]:
```
df.describe() # 天气数据的基本统计量
```

Out[56]:

	3	4
count	77.000000	77.000000
mean	6.597403	-2.220779
std	9.143965	10.238759
min	-19.000000	-33.000000
25%	2.000000	-6.000000
50%	8.000000	-2.000000
75%	12.000000	3.000000
max	21.000000	16.000000

5. StatsModels

Pandas着重在数据的读取、处理和探索，而StatsModels则更注重数据的统计建模分析，支持与Pdandas进行数据交互。

安装statsmodels 库

In [57]:
```
# 检查是否安装StatsModels库 慎重：没有安装的，需要在后台花费时间下载安装，较慢！
#!pip install statsmodels
```

In [58]:
```
# 使用StatsModels来进行ADF平稳性检验
# ADF平稳性检验用来检验金融、经济时间序列数据的平稳性
from statsmodels.tsa.stattools import adfuller as ADF  # 导入ADF检验
import numpy as np
ADF(np.random.rand(100))  # 返回的结果有ADF值、p值等
```

Out[58]:
```
(-9.2323285613489343,
 1.6654493137660062e-15,
 0L,
 99L,
 {'1%': -3.4981980821890981,
  '10%': -2.5825959973472097,
  '5%': -2.8912082118604681},
 39.292724137897778)
```

In [59]:
```
#ADF?  # 查阅帮助
```

6. Scikit-Learn

Scikit-Learn是Python下强大的机器学习工具包。 Scikit-Learn官网 (http://scikit-learn.org/stable/index.html)

包含数据预处理、分类、回归、聚类、预测和模型分析等。Scikit-Learn依赖于NumPy、SciPy和Matplotlib，因此，需要提前安装好这些库。

所有的模型提供的接口有：

- model.fit(): 训练模型，对于监督模型来说就是fit(X,y)，对于非监督模型就是fit(X)

监督模型提供的接口有：

- model.predict(X_new): 预测新样本
- model.predict_proba(X_new): 预测概率，仅对某些模型有用（比如LR）
- model.score(): 得分越高，fit越好

非监督模型提供的接口有：

- model.transform(): 从数据中学到的新的“基空间”
- model.fit_transform(): 从数据中学到新的基并将数据按照这组“基”进行转换

In [60]:
```
# 导入库
from sklearn.linear_model import LinearRegression  # 导入线性回归模型
model = LinearRegression() # 建立线性回归模型
print (model)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

In [61]:
```
# 导入sklearn本身提供的实例数据
from sklearn import datasets

iris = datasets.load_iris()  # 加载数据集
print (iris.data.shape)  # 查看数据集大小 150个样本，4个特征
```

```
(150L, 4L)
```

In [62]:
```
b = datasets.load_boston()
print (b.data.shape)
```

```
(506L, 13L)
```

In [63]:
```
iris.data[:5]  # 前5条数据
```

Out[63]:
```
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])
```

In [64]:
```
iris.target[:5]  # 前5条数据的类标（class label）
```

Out[64]:
```
array([0, 0, 0, 0, 0])
```

In [65]:
```
from sklearn import svm  # 导入SVM模型

clf = svm.LinearSVC()  # 建立线性SVM分类器
clf.fit(iris.data, iris.target)  # 用于数据训练模型
clf.coef_  # 查看训练好的模型的参数,这里是支持向量
```

Out[65]:
```
array([[ 0.1842,  0.4512, -0.8079, -0.4507],
       [ 0.0488, -0.8893,  0.4054, -0.9373],
       [-0.8506, -0.9867,  1.3809,  1.8654]])
```

In [66]:
```
# 训练好模型后，输入新的数据样本1进行类标预测
clf.predict([[5.0, 3.6, 1.3, 0.25]])
```

Out[66]:
```
array([0])
```

In [67]:
```
# 训练好模型后，输入新的数据样本2进行类标预测
clf.predict([[5.0, 1.6, 2.3, 1.95]])
```

Out[67]:
```
array([1])
```

In [68]:
```
# 训练好模型后，输入新的数据样本3进行类标预测
clf.predict([[5.0, 1.6, 4.3, 1.95]])
```

Out[68]:
```
array([2])
```

7. Keras

Scikit-Learn足够强大，但是没有包含一个强大的模型--人工神经网络。近年来火热的“深度学习”算法，本质上就是一种神经网络。

Keras库可以用来搭建神经网络，它本身并非简单的神经网络库，而是一个基于Theano的强大的深度学习库，还可以搭建各种深度学习模型，如自编码器、循环神经网络、递归神经网络、卷积神经网络等。Keras建立在Theano上，速度也是很快。

Theano本身也是Python的一个库，由深度学习专家Yoshua Bengio带领的实验室开放，用来定义、优化和解决多维数组数据对应数学表达式的模拟估计问题，具有高效地实现符号分解、高度优化的速度和稳定性等特点，重要的是它还实现了GPU加速，使得密集型数据的处理速度是CPU的数十倍。

Keras大大简化了搭建各种神经网络模型的步骤。

```
conda install theano
conda install keras
```

下面的例子是简单搭建一个LSTM,详见代码L03/code/LSTM.py, 输出结果在LSTM-Outputs.txt 输出结果每行为每个测试样本在10个类别中的预测概率分布。

注意： 目前keras中使用到TensorFlow，而TensorFlow目前无法在Windows系统上安装，只支持Mac、Linux系统，因此下面的代码需要在服务器端运行。

8. Gensim

Gensim处理语言方面的任务，例如，文本相似度、LDA语言模型、Word2Vec（Google公司在2013年开源的著名的词向量构造工具）等。

Gensim对Word2Vec代码进行了优化，在Linux环境下运行更快。

NLTK中包含了许多用于进行语言处理和分析的数据和工具，安装如下： `conda install nltk` (注意：下载安装时间比较久) 运行Python后，在提示符后面输入： nltk.download()

In [ ]:
```
import gensim, logging  # 安装logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
# logging是用来输出训练日志
```

In [70]:
```
# 得到分词后的句子，每个句子以词列表的形式输入
sentences = [ ['first', 'sentence'],['second', 'sentence']]
```

In [ ]:
```
# 用上面的句子训练词向量模型
model = gensim.models.Word2Vec(sentences, size=10, min_count=1) # size是词向量维度
```

In [72]:
```
# 输出单词sentence的词向量
print model['sentence']
```

```
[ 0.0109 -0.0151 -0.0321 -0.0099  0.0032  0.0437 -0.037  -0.0334 -0.0454
  0.01  ]
```

In [ ]:
```

```