机器学习应用实验手册

小象学院　邹博

2017 年 3 月 16 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、二、三章和第九章(附录)，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。我的微博：@邹博_机器学习。

目录

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

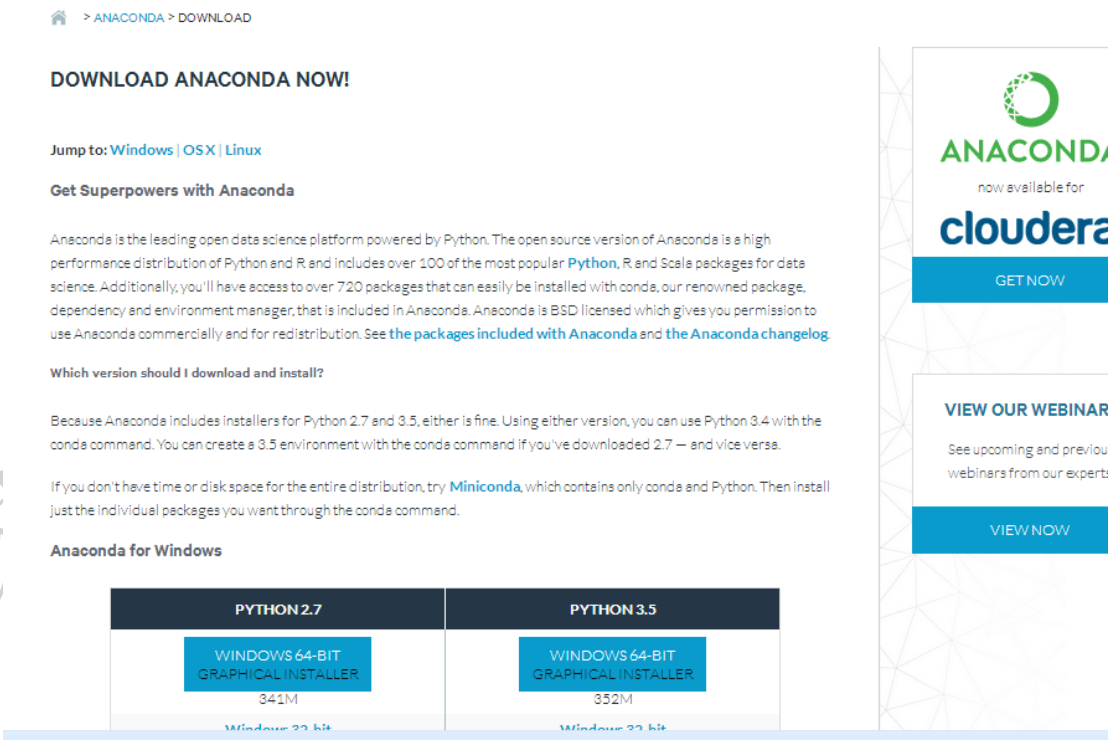
2、配置环境

3、安装 pycharm 编辑环境 IDE

1.1 软件包的下载准备

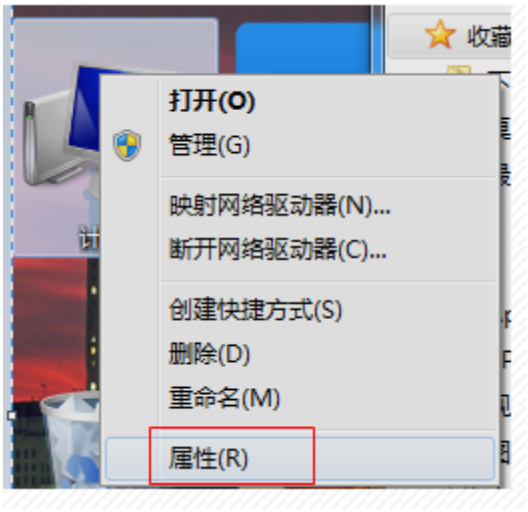
1.1.1 Anaconda 下载

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

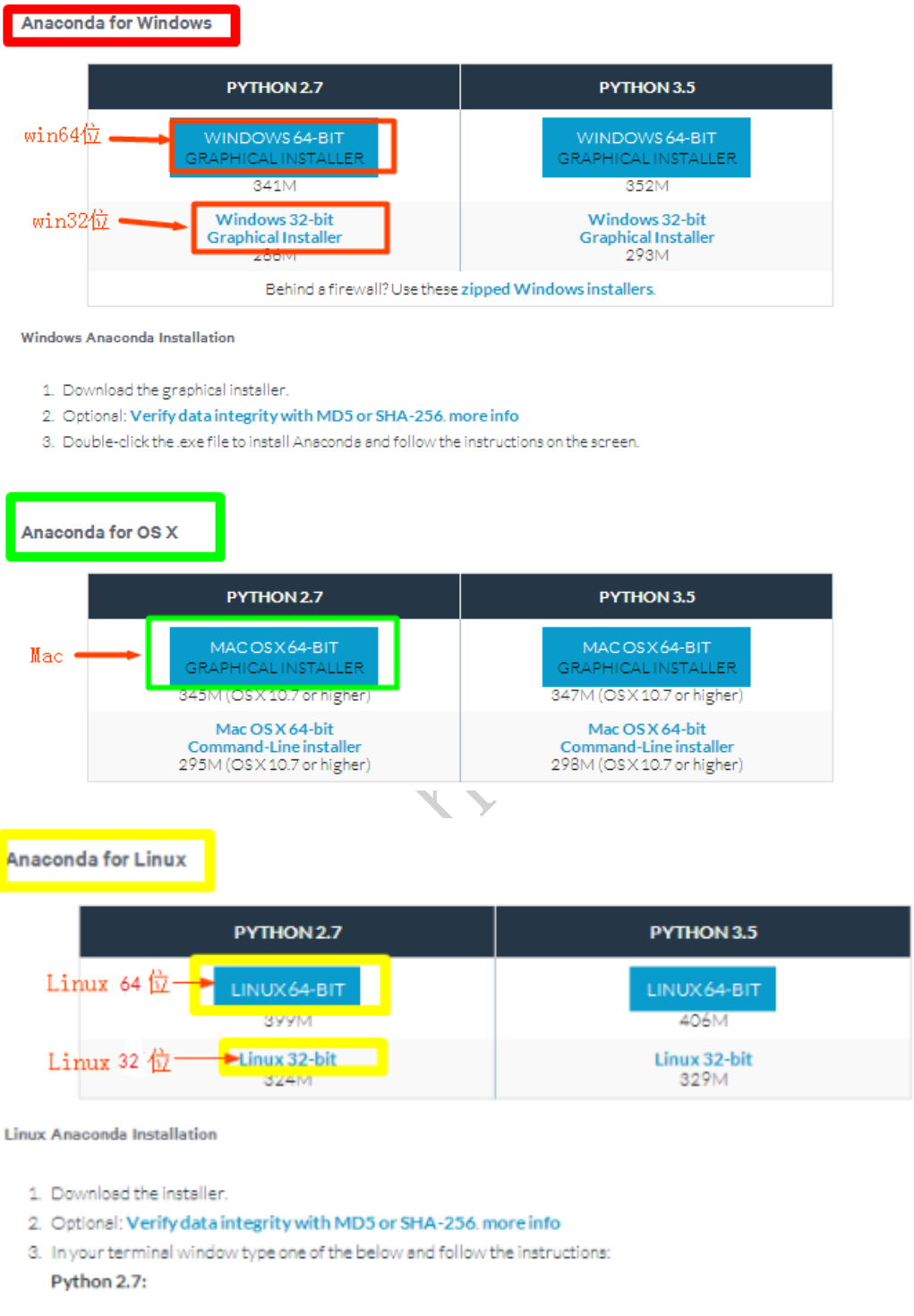
栏中)软件

(2)打开终端，输入命令 uname -a <回车>

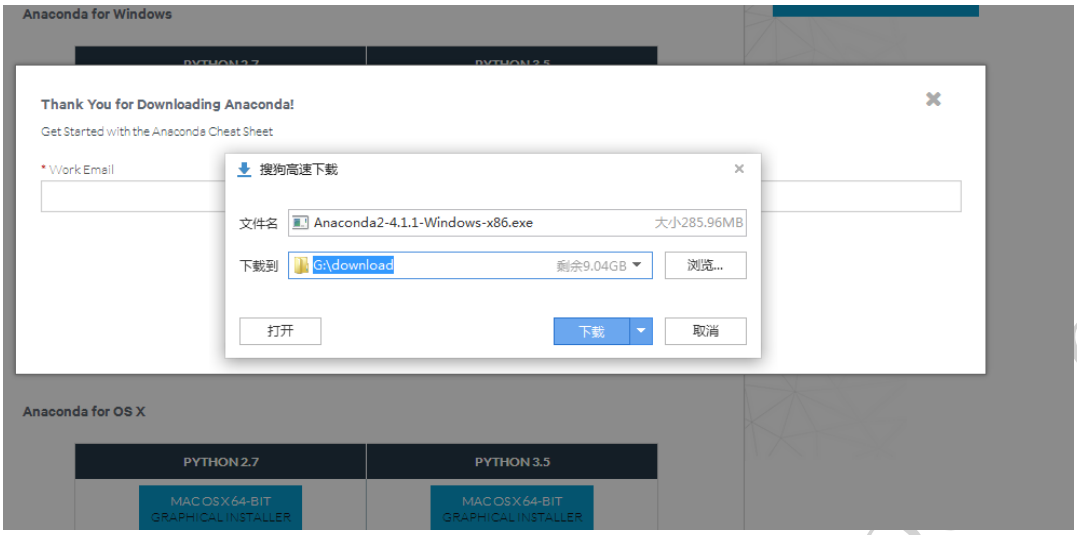
x86_64 表示系统为 64 位

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。

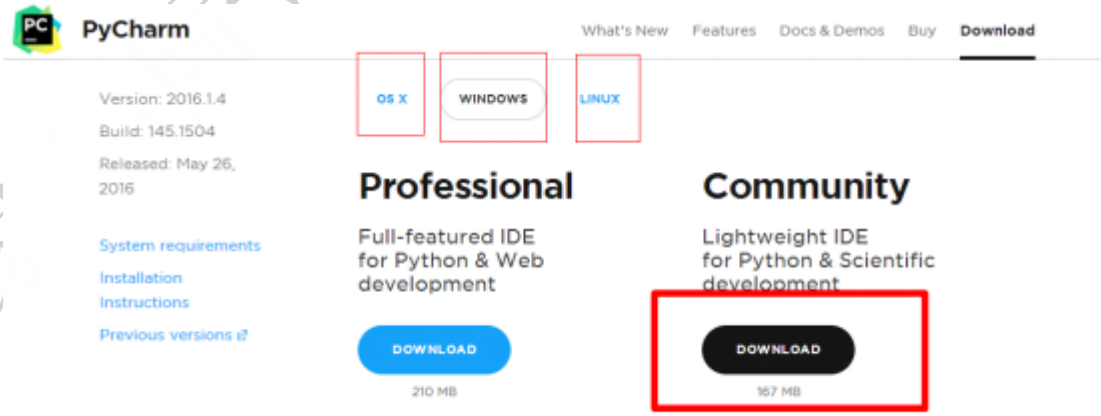


- 选择路径进行下载，下载界面如下。



1.1.2 PyCharm 编辑器下载

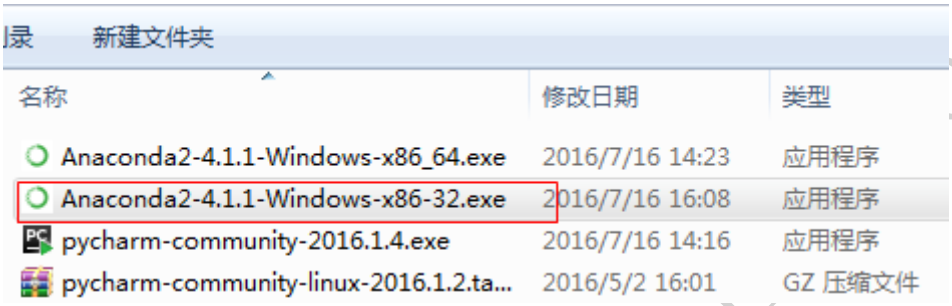
- Anaconda 是自带编辑环境的，可以直接在其中做代码编写。但仍然推荐安装 PyCharm，因为它在程序调试等方面更加出色。我们只需要安装 PyCharm 结束的时候，将 Python 解释器选择 Anaconda 下的 python.exe 即可（该步骤的最后部分介绍如何操作）。
- 在 http://www.jetbrains.com/pycharm/download/#section=windows 上找到与您计算机操作系统相应的版本进行下载。



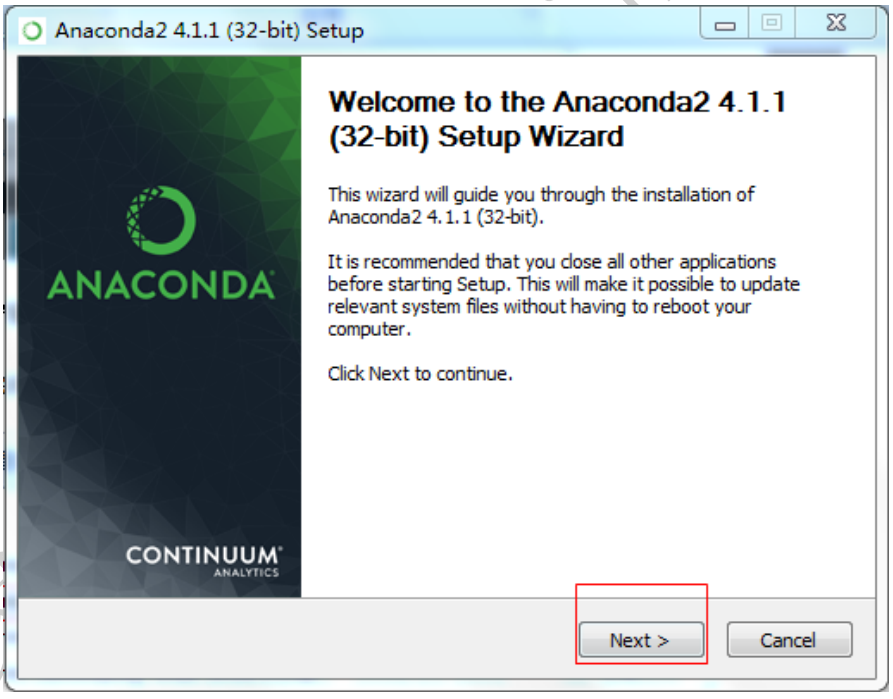
1.2 windows 下安装步骤

1.2.1 安装 python 科学计算 Anaconda

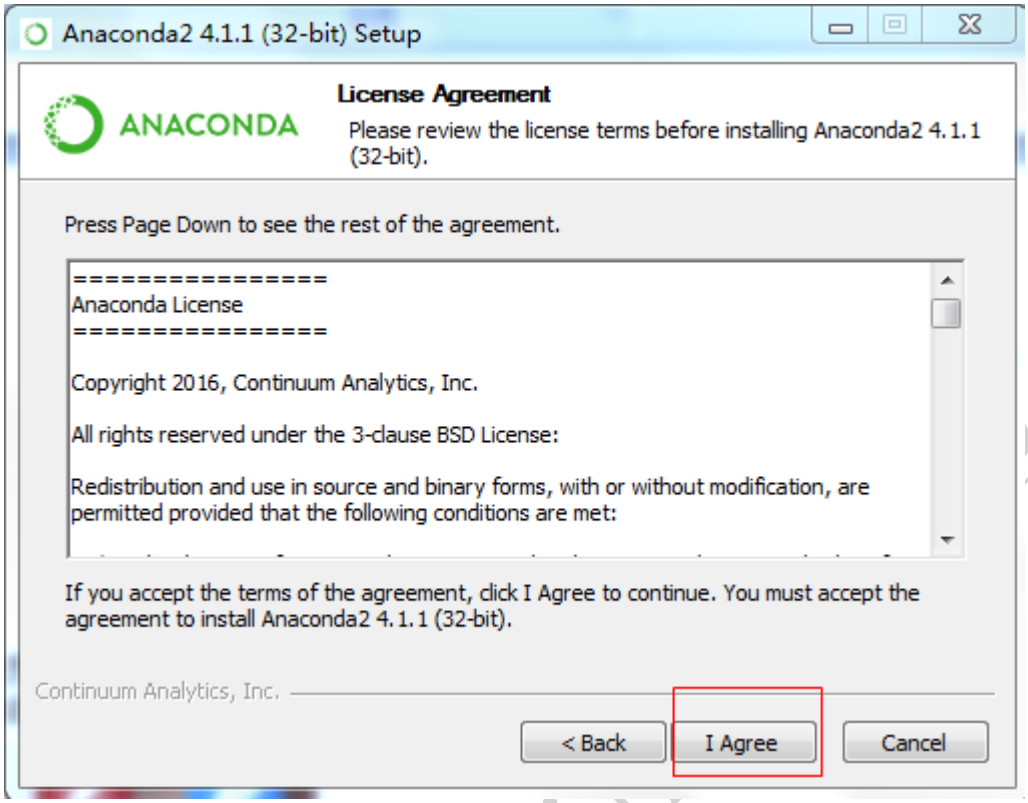
- 下载 Anaconda 完成后，双击与您计算机系统相对应的软件（这里以 win32 位安装过程为例）



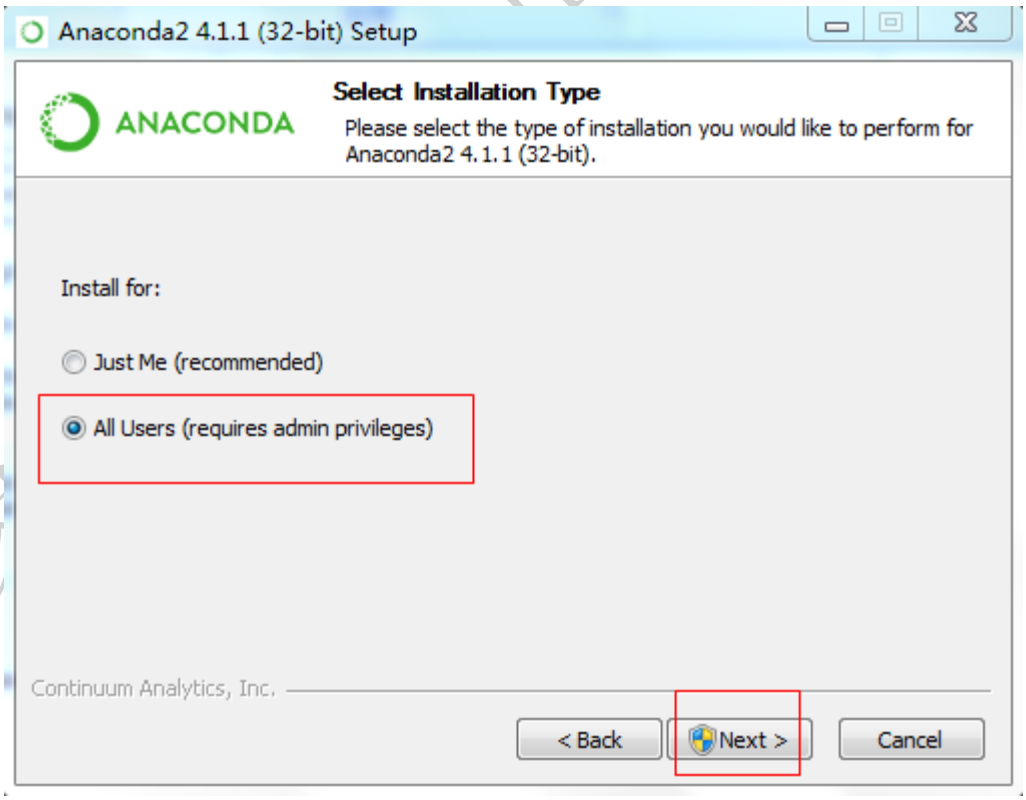
- 打开后对话框界面如下，点击 Next 按钮。



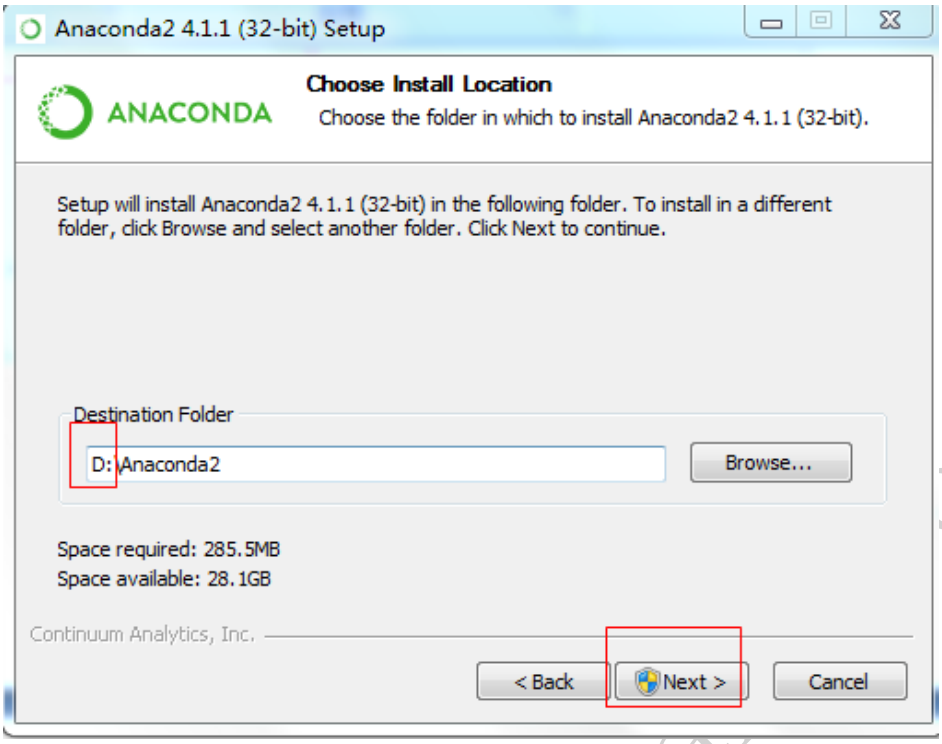
- 点击 I Agree 按钮。



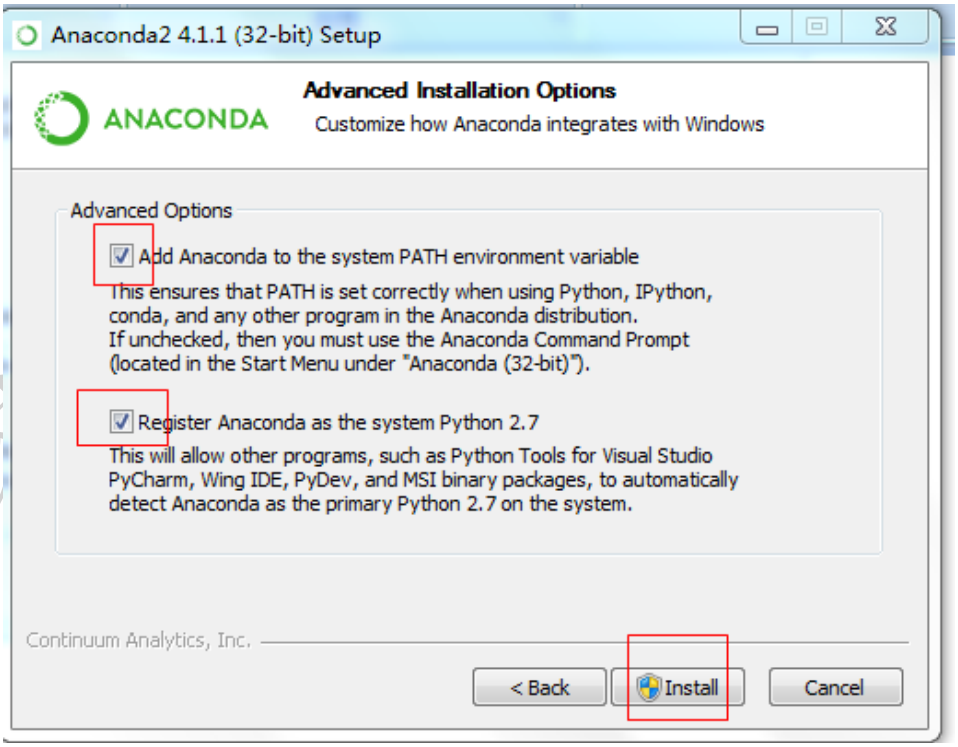
- 单选按钮可以选择 All Users(这个是默认选中)，再点击 Next 按钮。



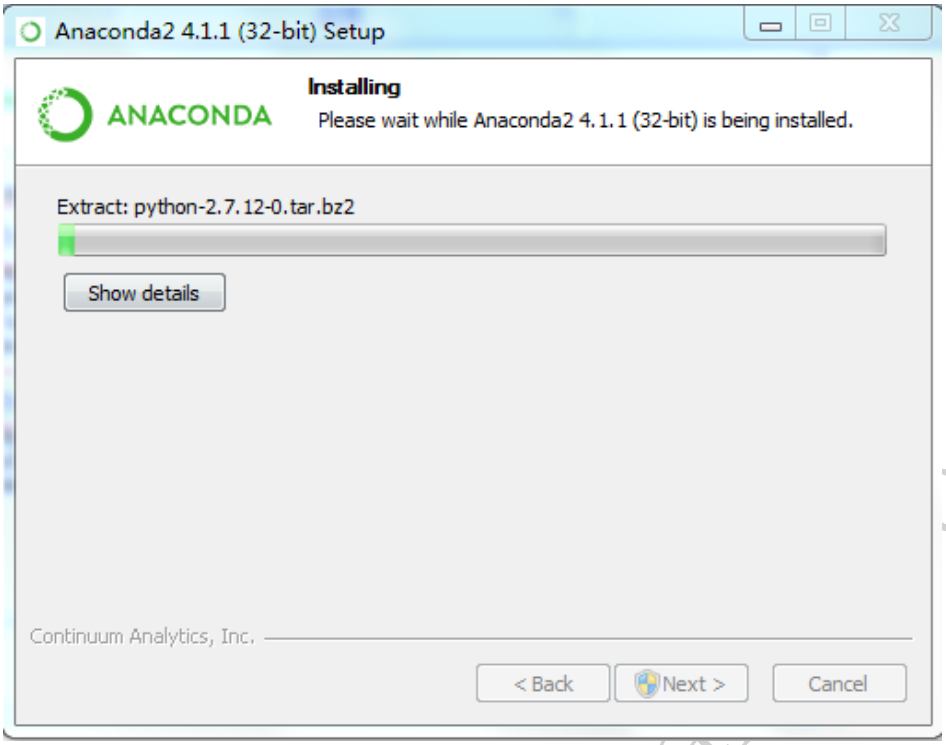
- 更改安装目录到 D 盘（或任意您方便的安装路径），再点击 Next 按钮。



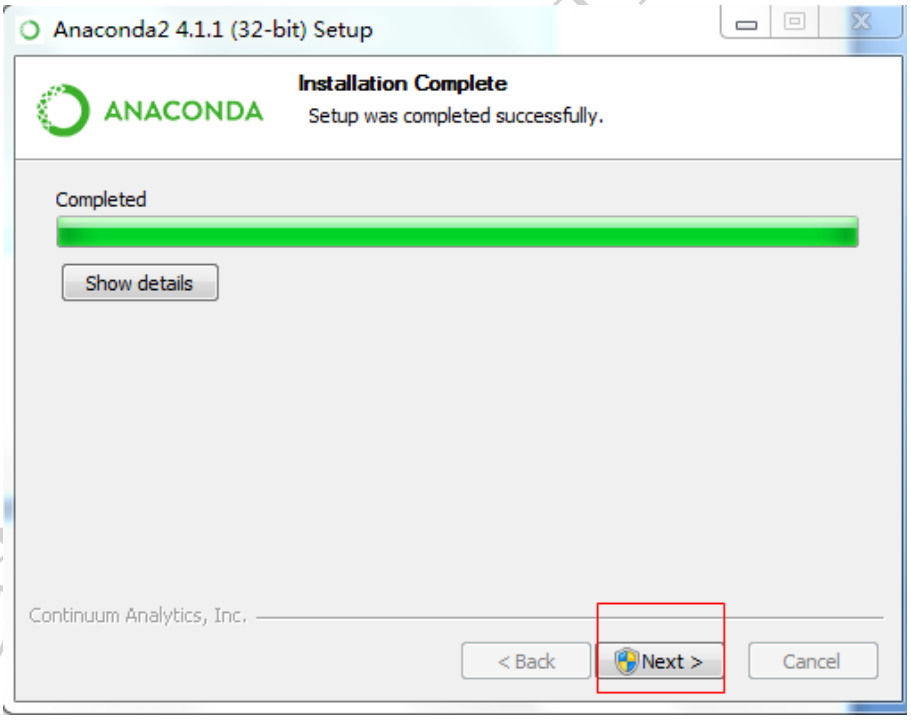
- 确保勾选了复选框“Add Anaconda to the system PATH environment variable”（在系统环境变量 path 中添加 Anaconda 路径）和“Register Anaconda as the system Python 2.7”（将 Anaconda 作为打开 Python 的默认程序），再点击 Install 按钮。



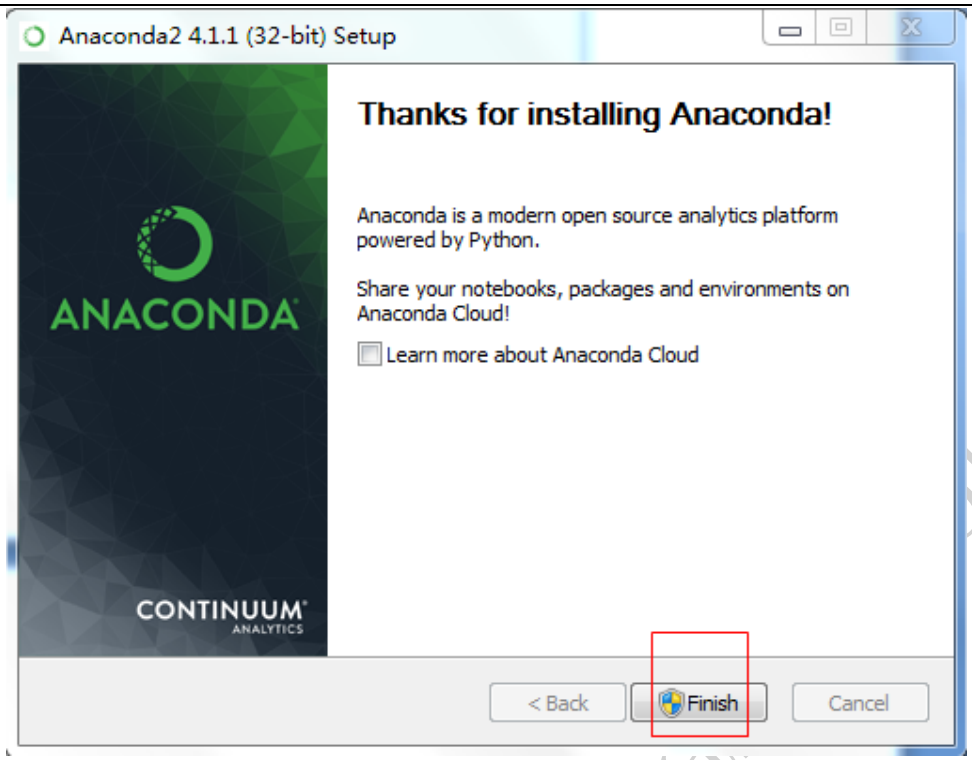
- 安装过程的界面如下。



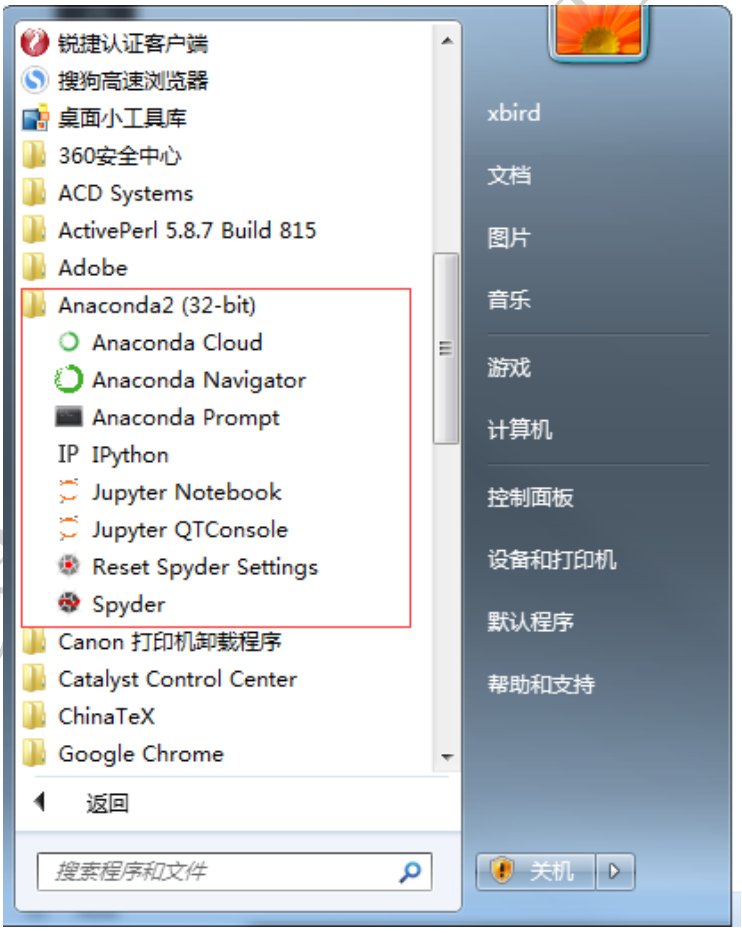
- 安装完成后，点击 Next 按钮。



- 点击 Finish 按钮完成 Anaconda 的安装。

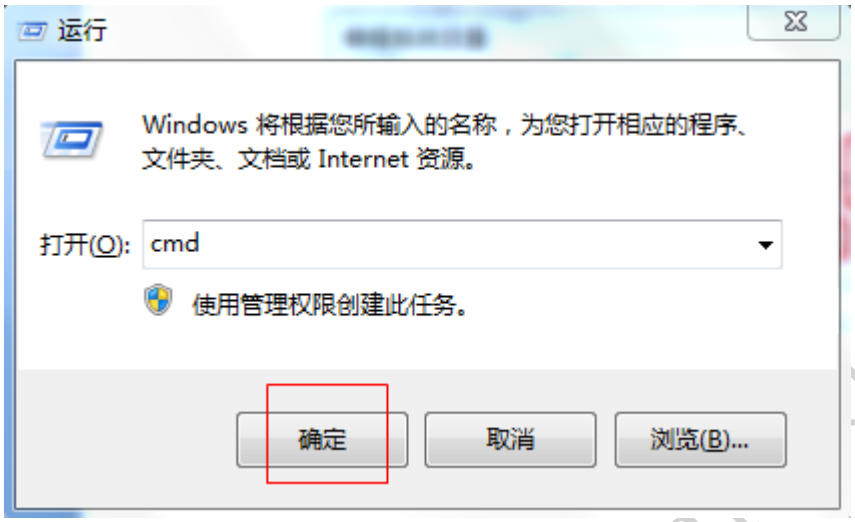


- 在开始菜单中，即可看到安装完成的 Anaconda。

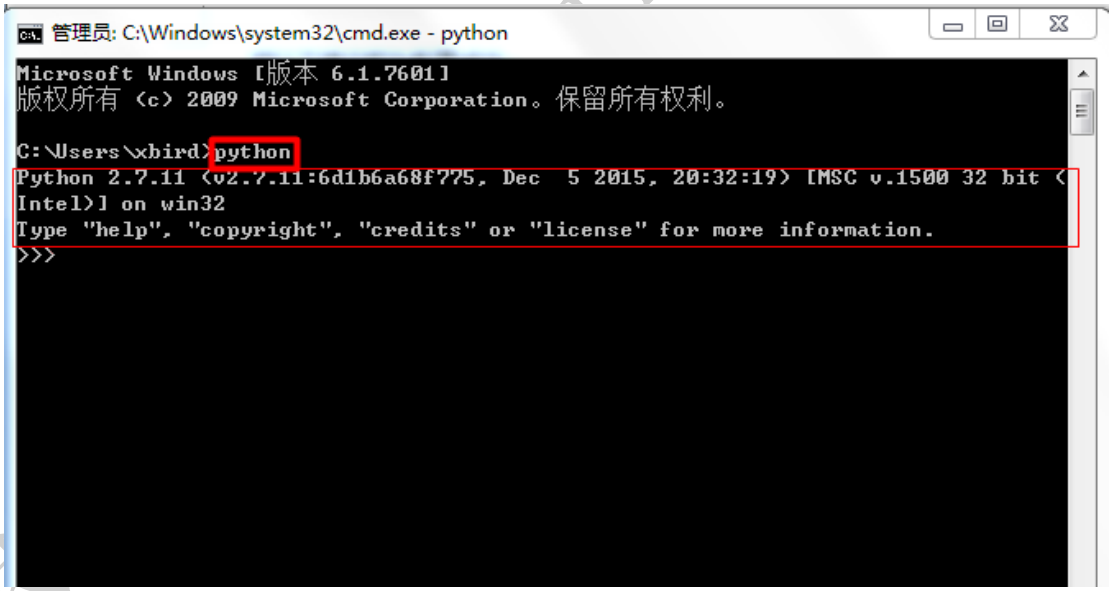


1.2.2 验证环境配置

- 验证环境配置：按 Windows+R 快捷键，并输入 cmd 打开命令行窗口。

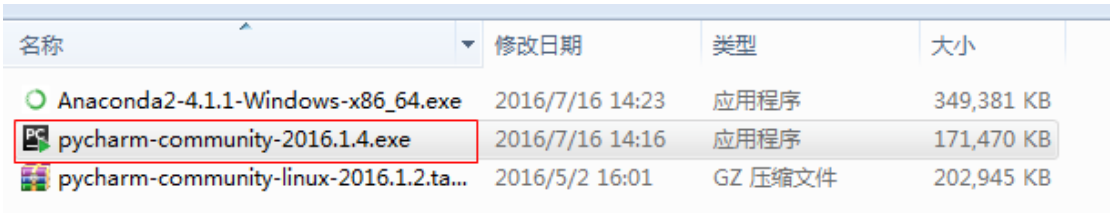


- 输入 python，出现以下提示即为配置成功。

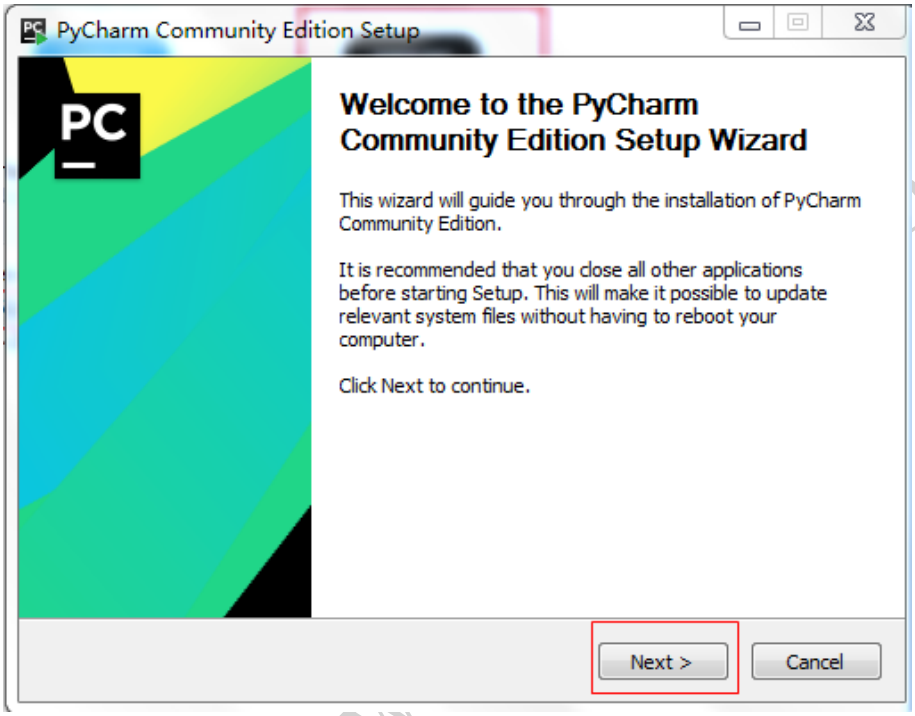


1.2.3 PyCharm 编辑器的安装

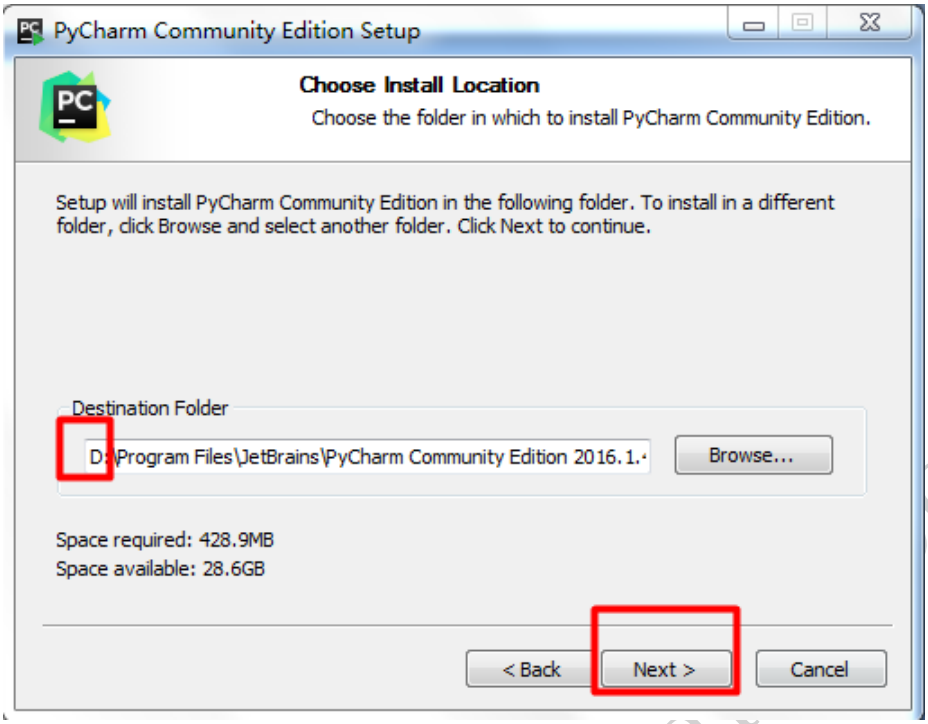
- 双击 pycharm-community-2016.1.4.exe 文件



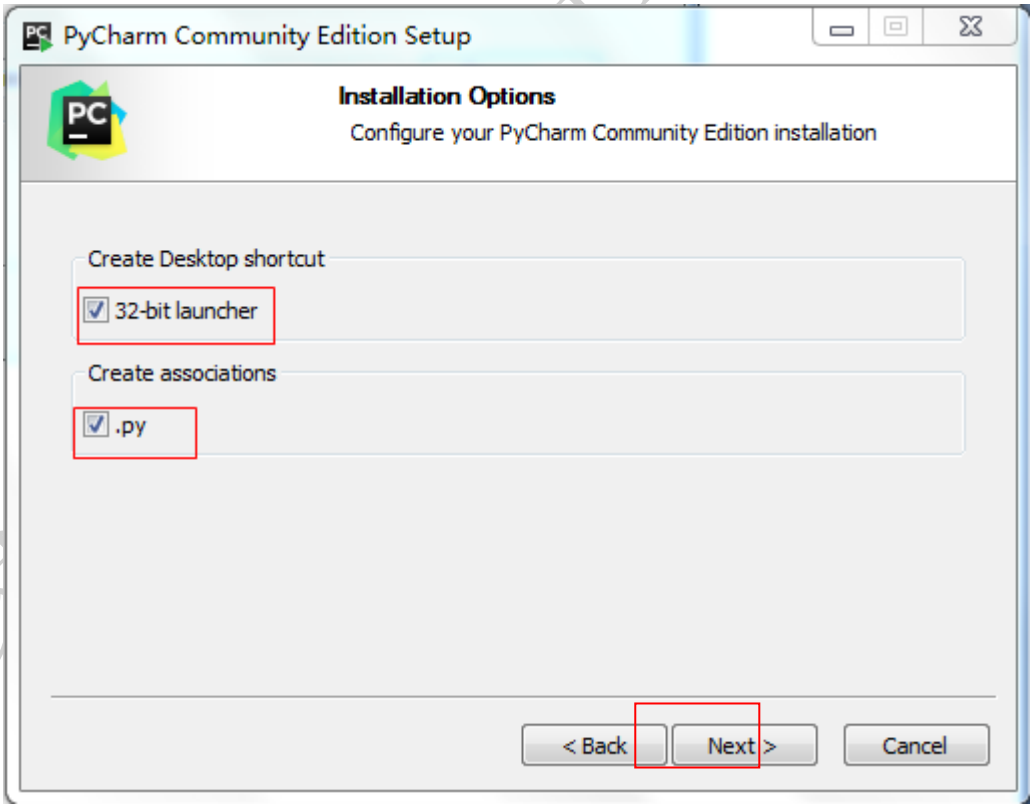
- 点击 Next 按钮。



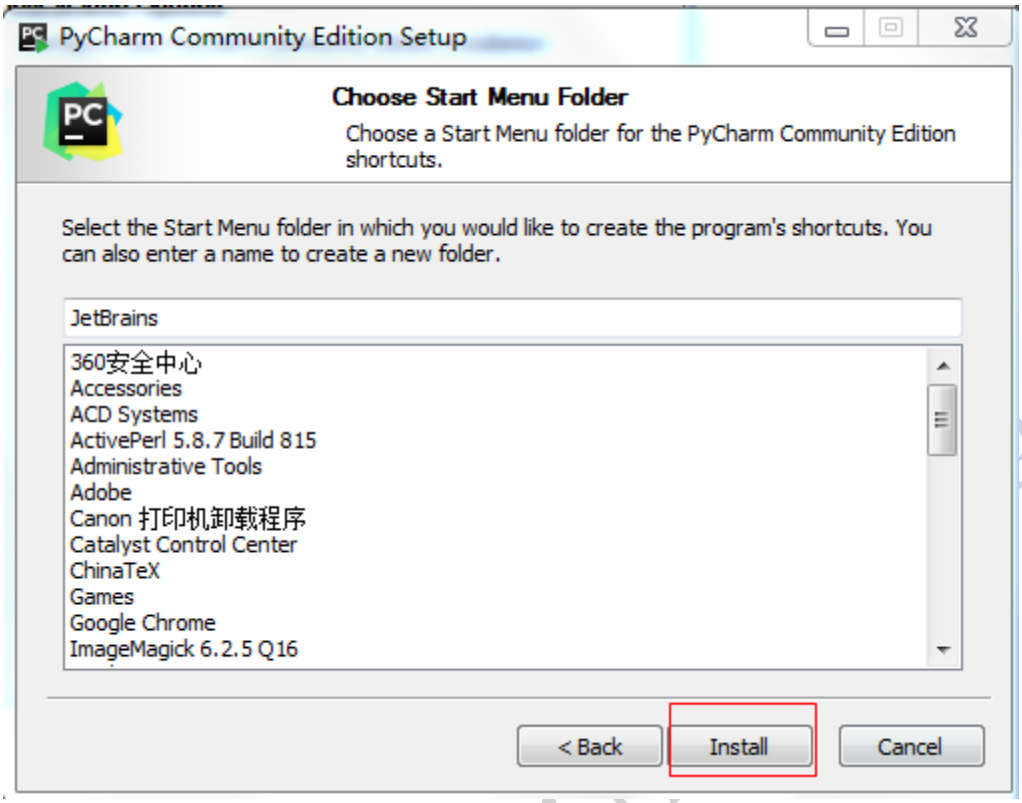
- 更改安装目录到 D 盘（或任意您方便的安装路径），再点击 Next 按钮。



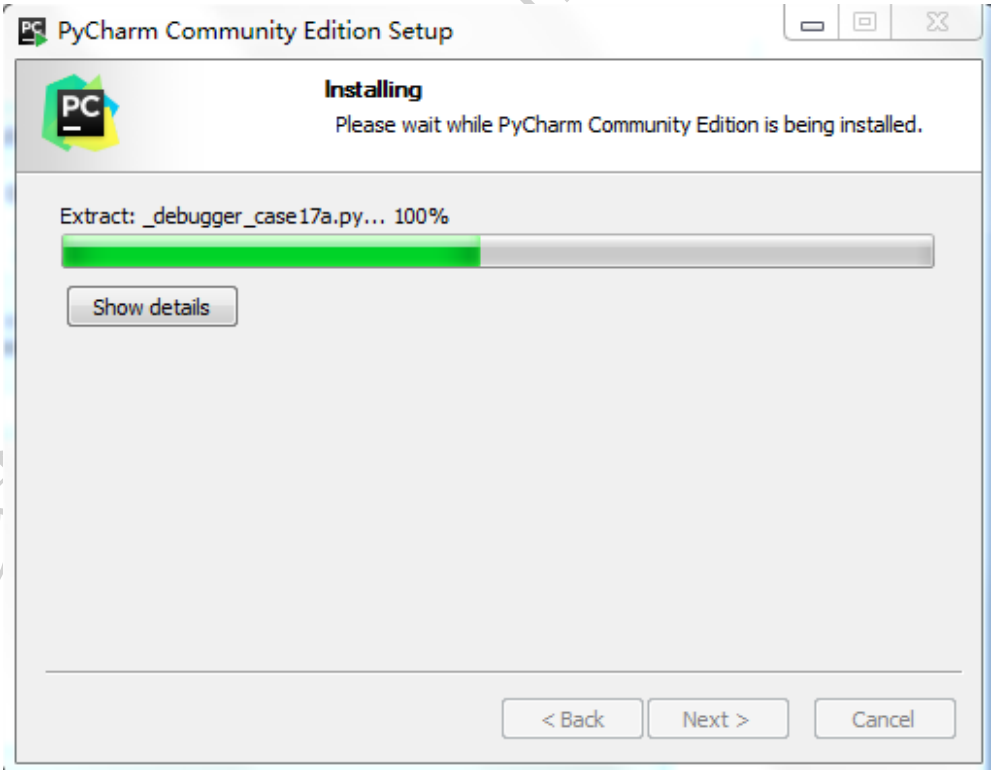
- 确保勾选“桌面创建图标”和“创建与.py 文件联系”复选框，并点击 Next 按钮。



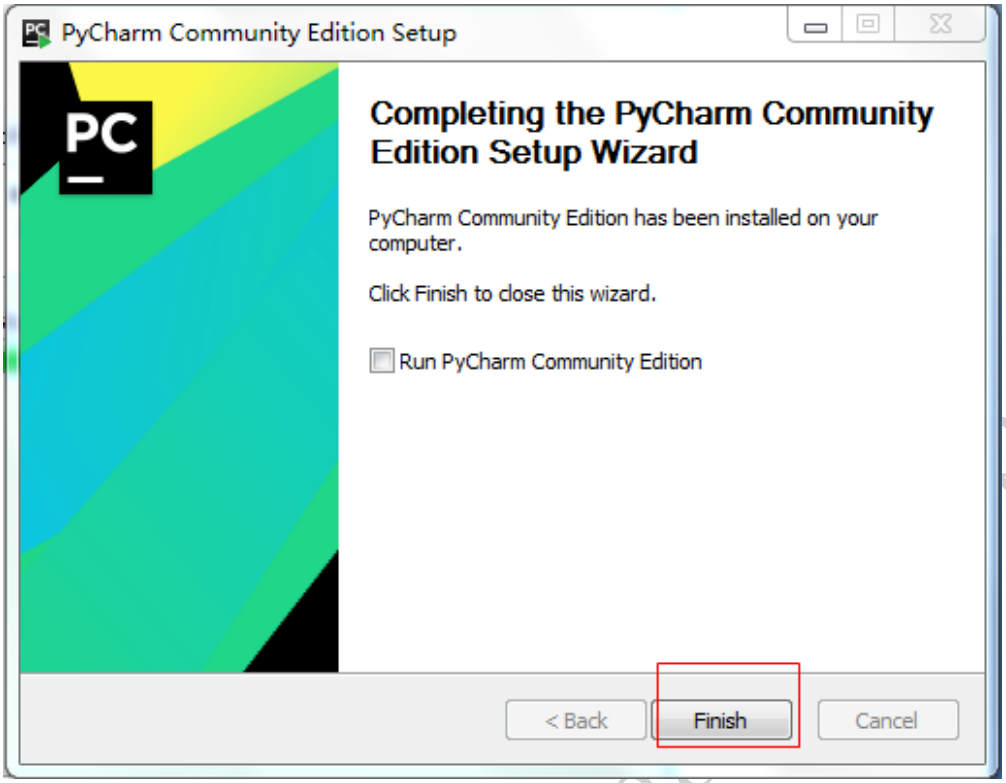
- 点 Install 按钮。



- 安装过程中的界面如下：

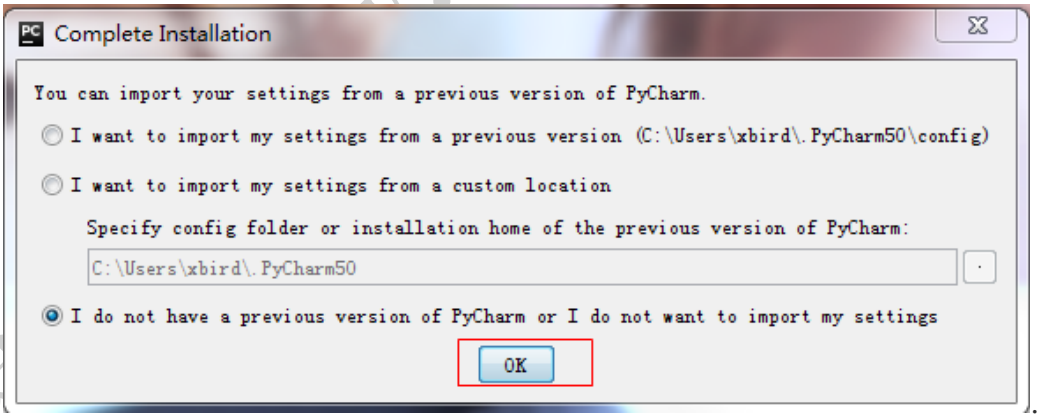


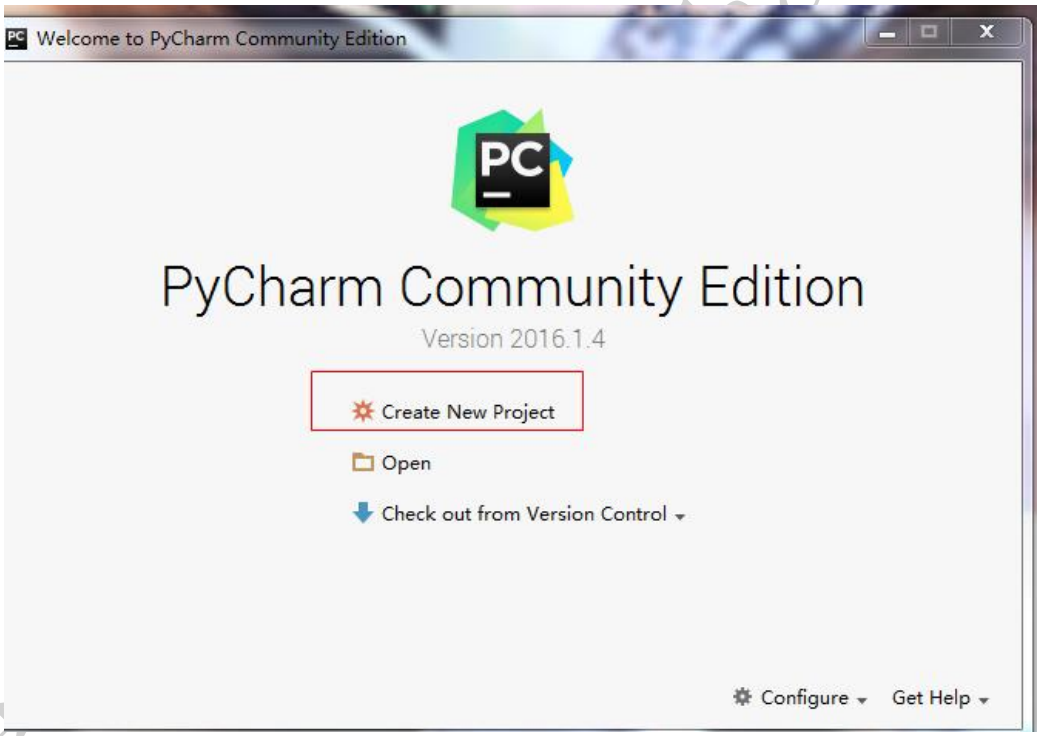
- 点击 Finish 按钮，安装完成。



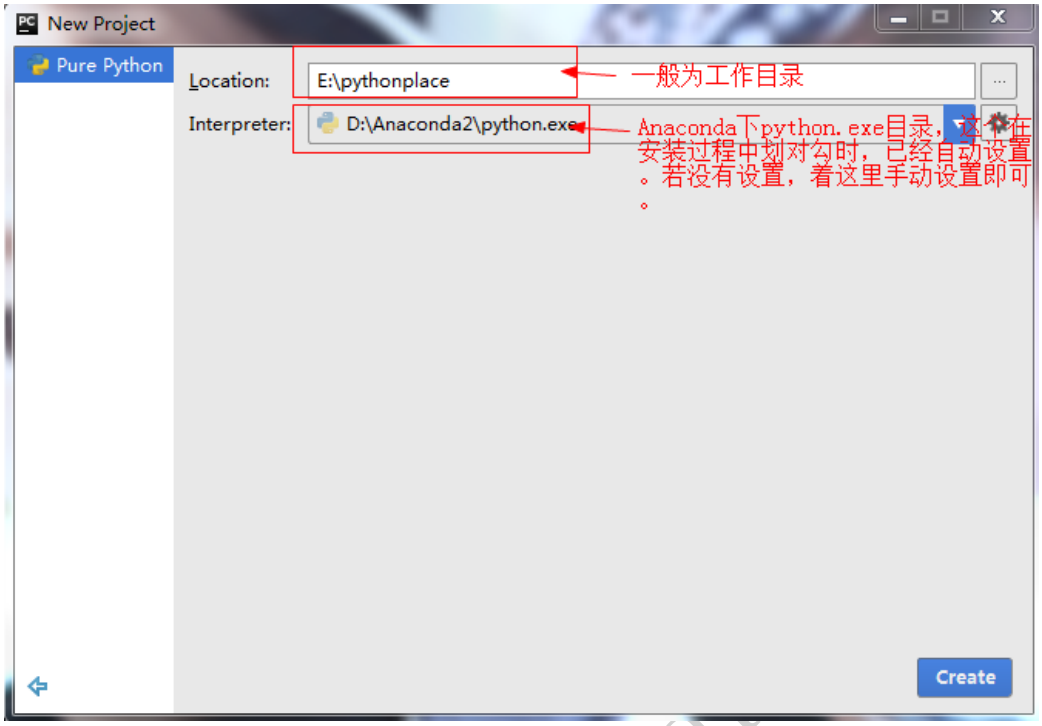
1.2.4 PyCharm 配置

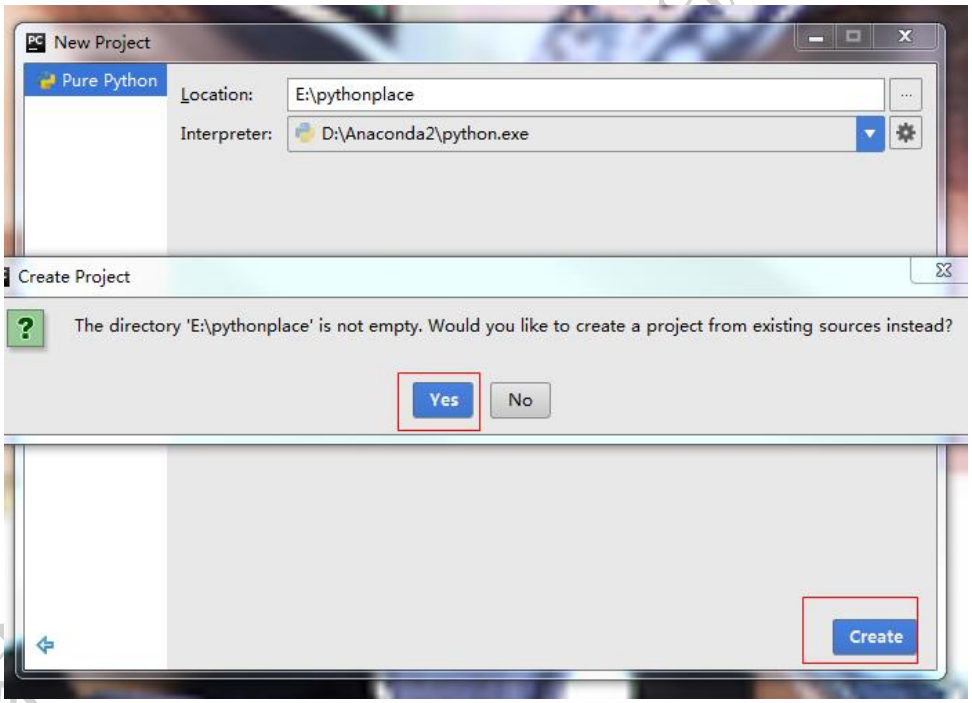
- 初次安装选择按照默认选择最后一个。



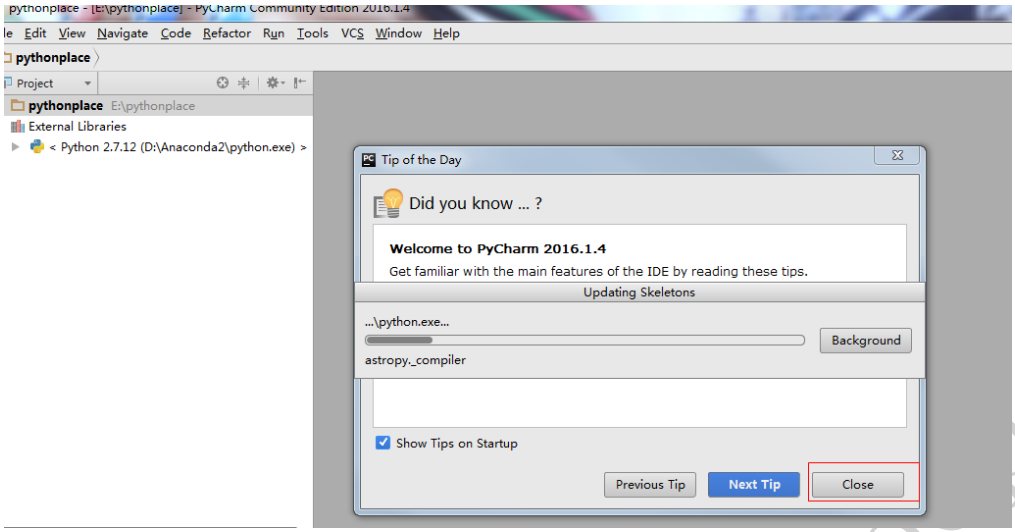


- 设置 python 解释器的位置





- Updating Skeletons 运行完点 close 即可。

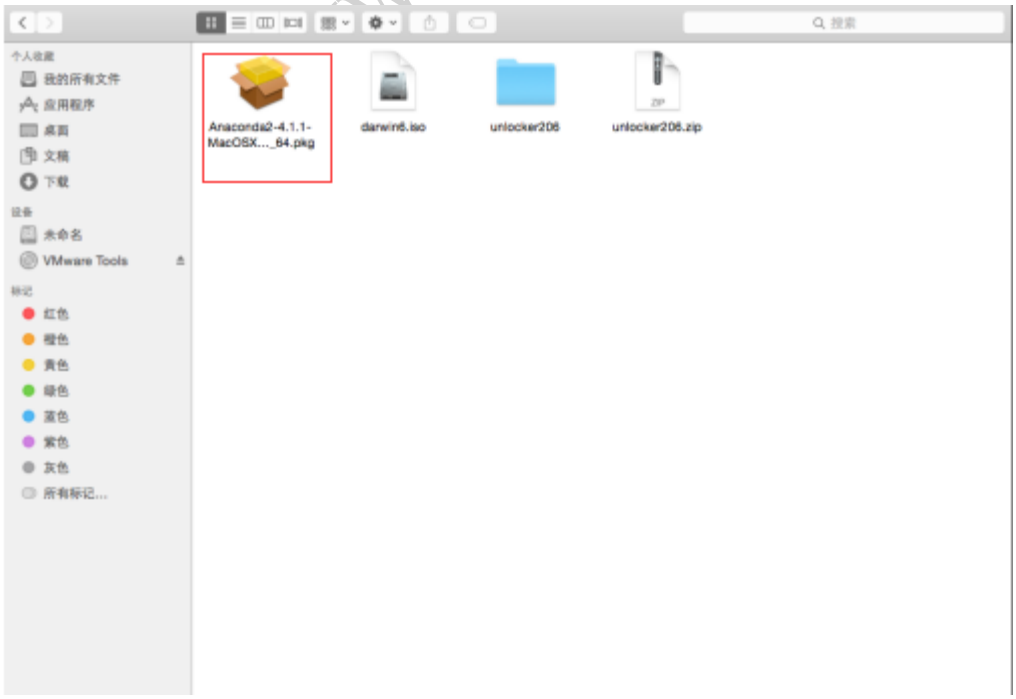


- 然后可以建立 python 工程，点击 run 运行即可。

1.3 Mac OS X 下安装步骤

1.3.1 安装 python 科学计算 Anaconda

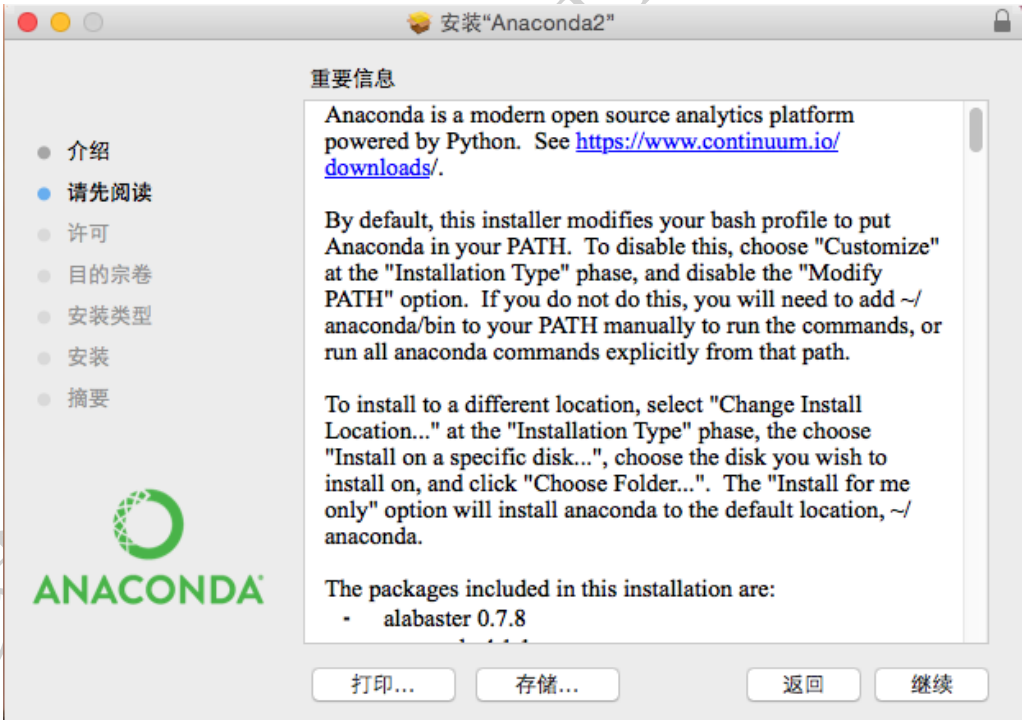
- 下载完成后，双击与您计算机系统相对应的软件

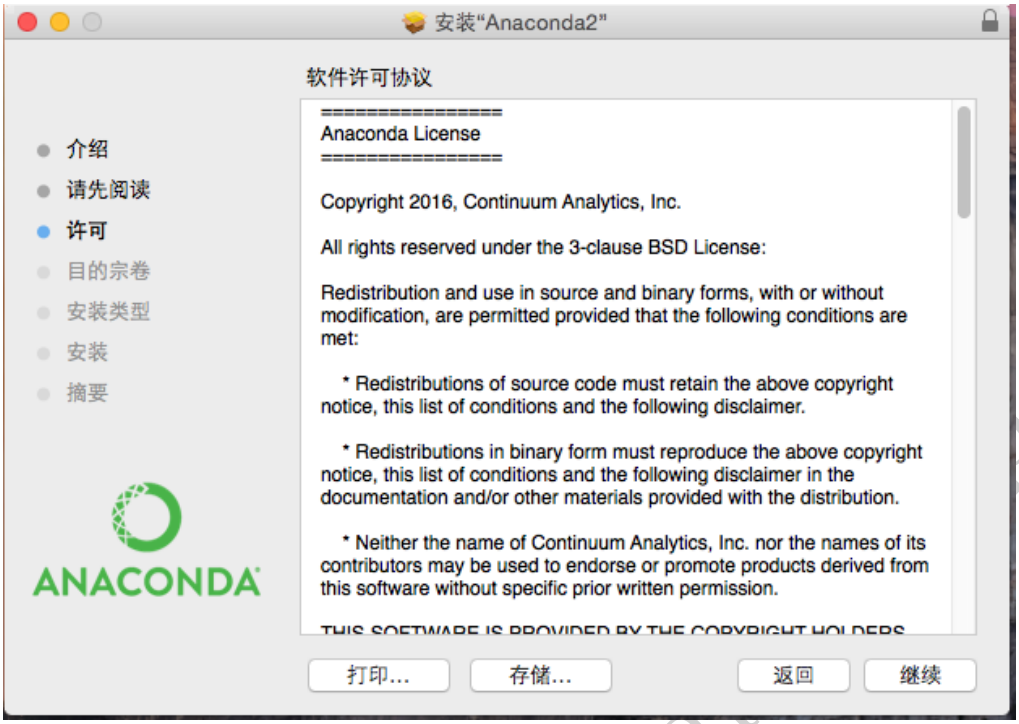


- 打开后对话框界面如下，点击“继续”按钮。


安装“Anaconda2”
欢迎使用“Anaconda2”安装器
介绍
请先阅读
许可
目的宗卷
安装类型
安装
摘要
安装器将引导您完成安装此软件所需要的步骤。
ANACONDA
返回
继续

- 一路点击“继续”

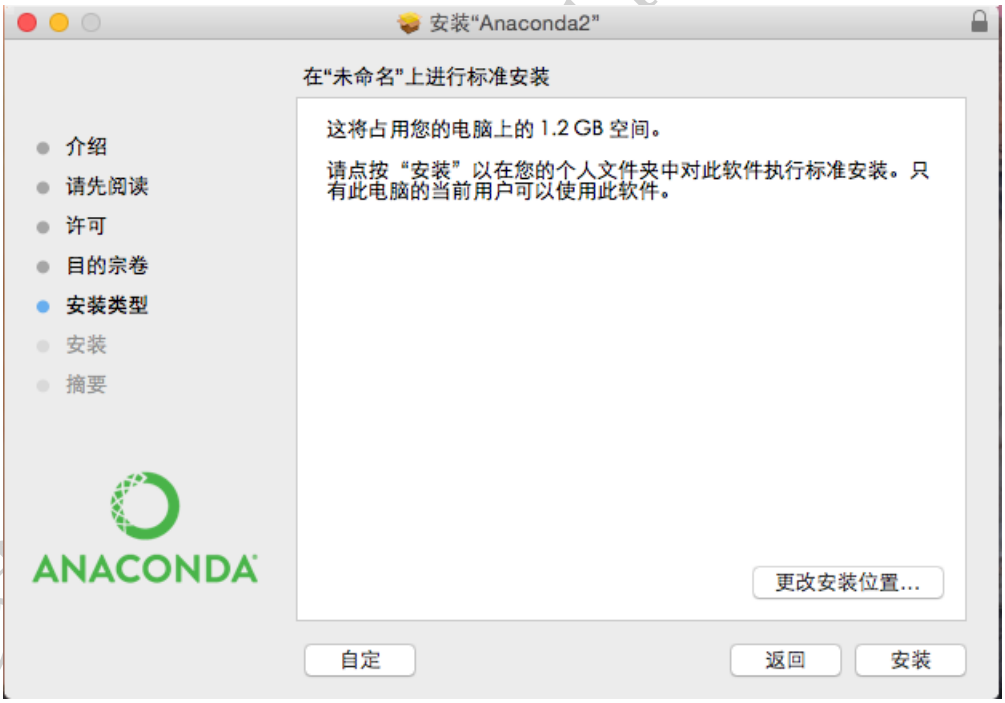

安装“Anaconda2”
重要信息
介绍
请先阅读
许可
目的宗卷
安装类型
安装
摘要
Anaconda is a modern open source analytics platform powered by Python. See https://www.continuum.io/downloads/.
By default, this installer modifies your bash profile to put Anaconda in your PATH. To disable this, choose "Customize" at the "Installation Type" phase, and disable the "Modify PATH" option. If you do not do this, you will need to add ~/anaconda/bin to your PATH manually to run the commands, or run all anaconda commands explicitly from that path.
To install to a different location, select "Change Install Location..." at the "Installation Type" phase, the choose "Install on a specific disk...", choose the disk you wish to install on, and click "Choose Folder...". The "Install for me only" option will install anaconda to the default location, ~/anaconda.
The packages included in this installation are:
- alabaster 0.7.8
ANACONDA
打印...
存储...
返回
继续



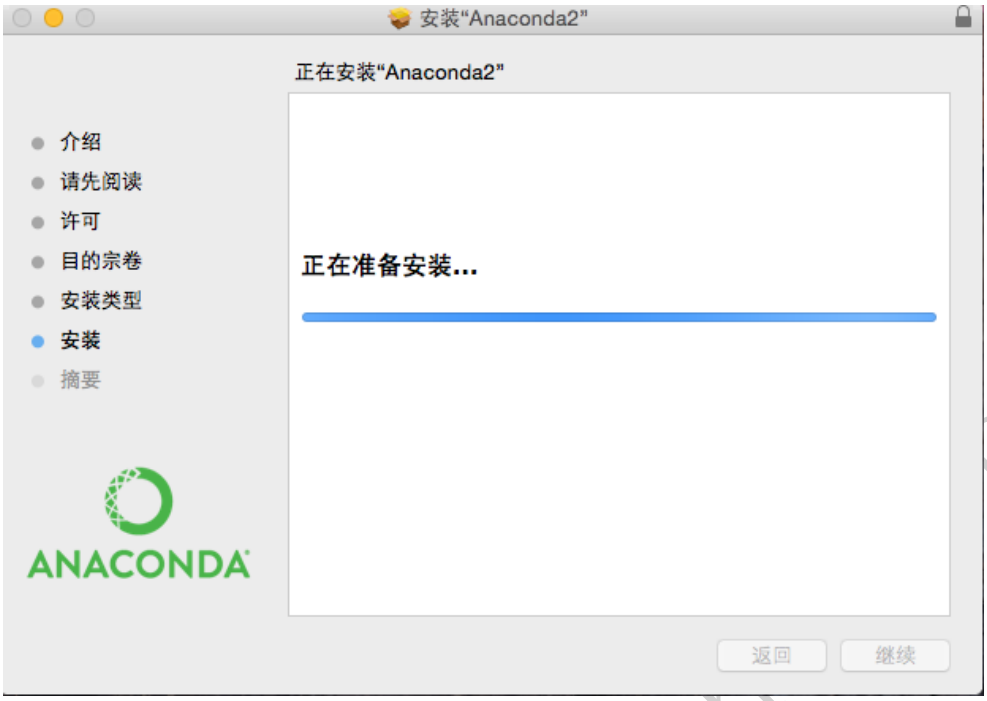
- 点击“同意”。

- 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”。

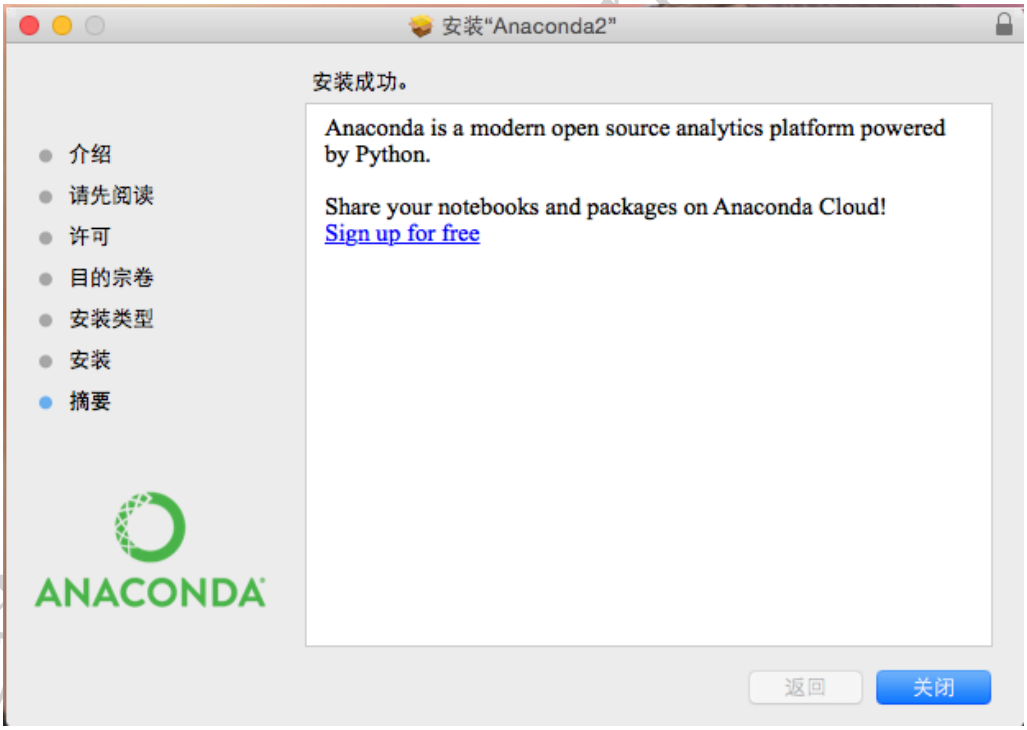
- 点击“安装”。



- 安装进行中界面

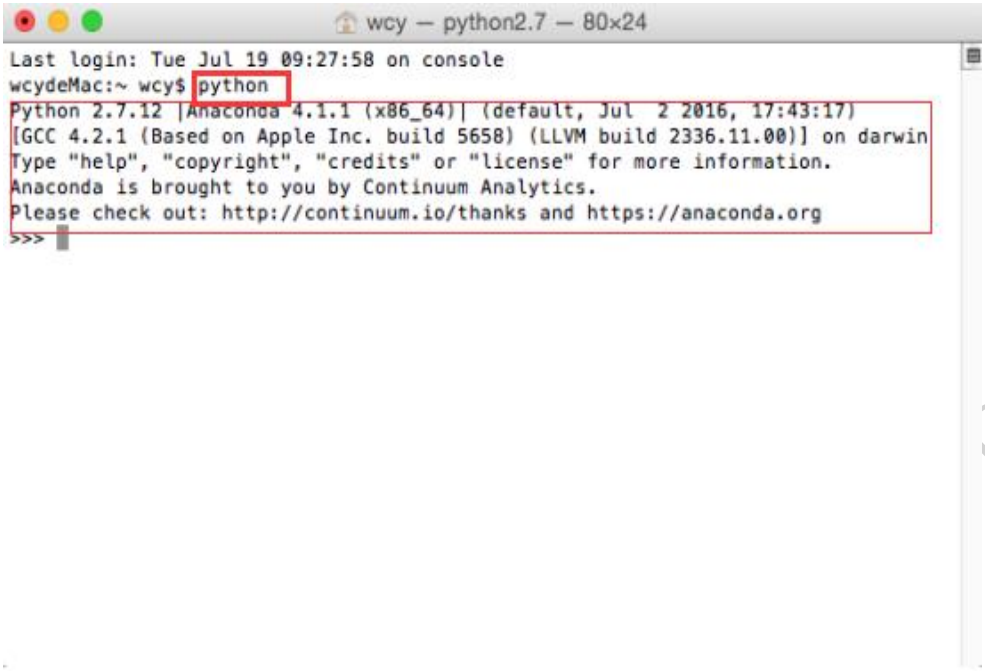


- 安装成功，点击“关闭”按钮。



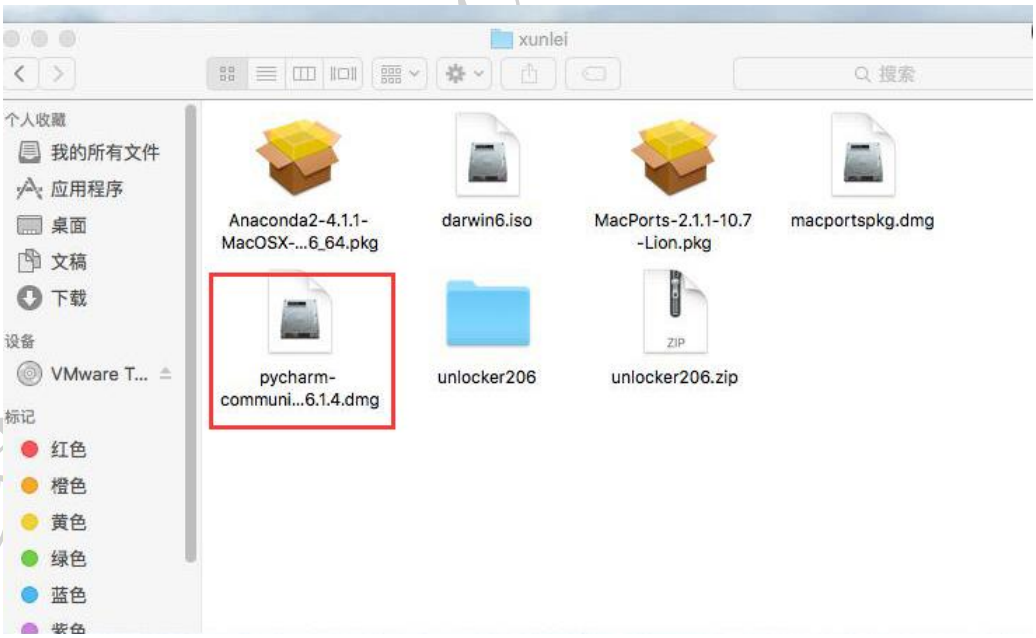
1.3.2 验证环境配置

- 打开 Mac 命令行终端，输入 python，出现以下提示即为配置成功。

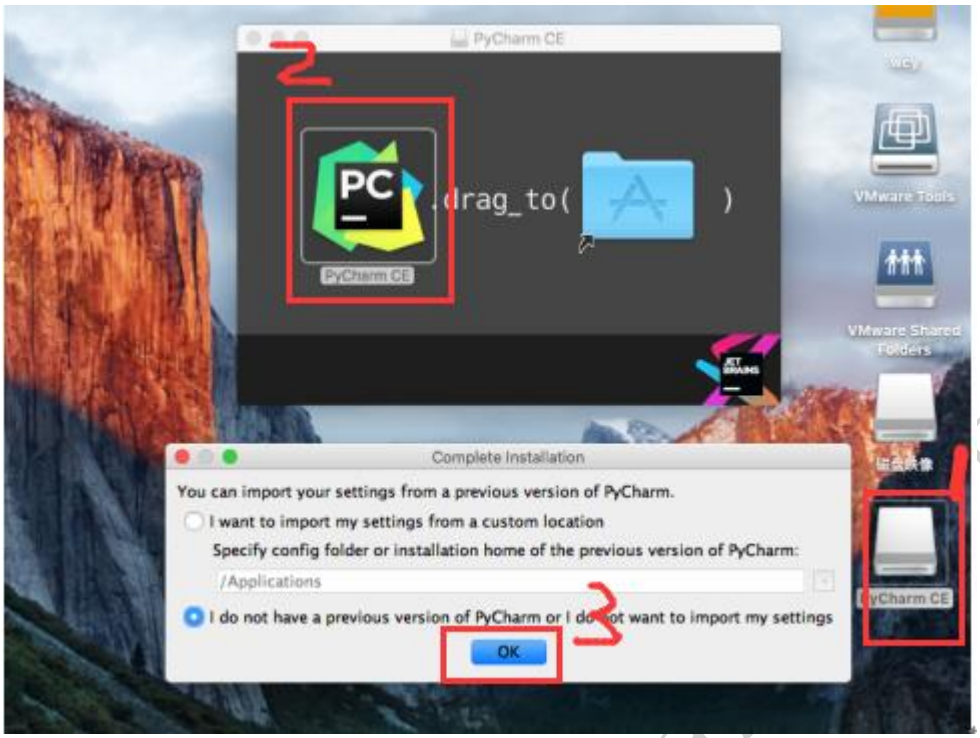


1.3.3 PyCharm 编辑器的安装及配置

- 双击相应 PyCharm 的软件包。

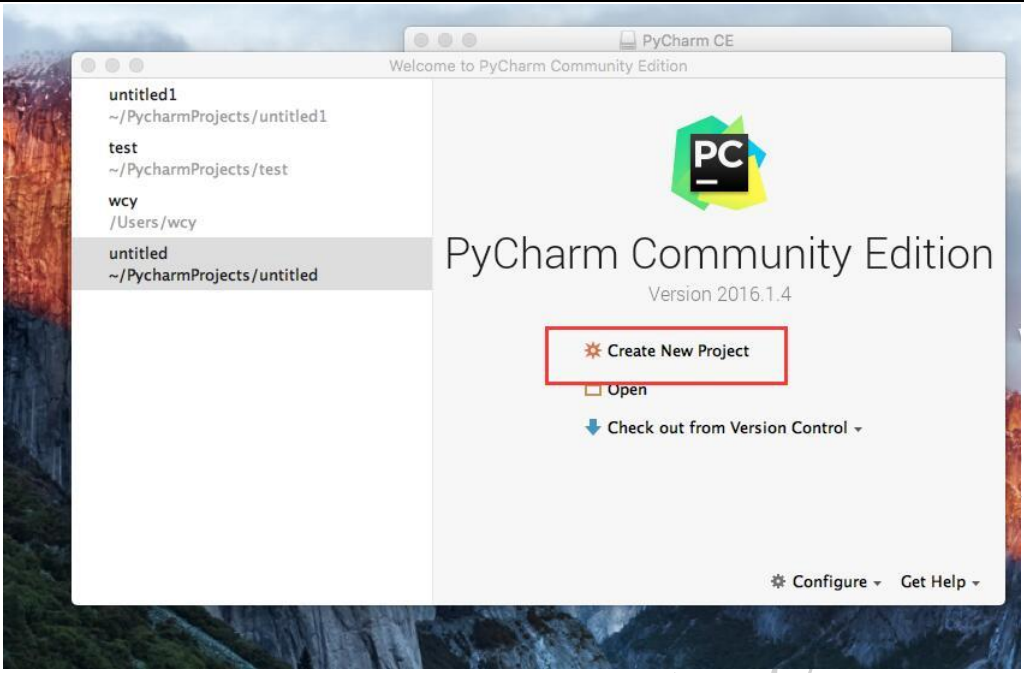


- 安装过程一路默认即可，最后点击“OK”。

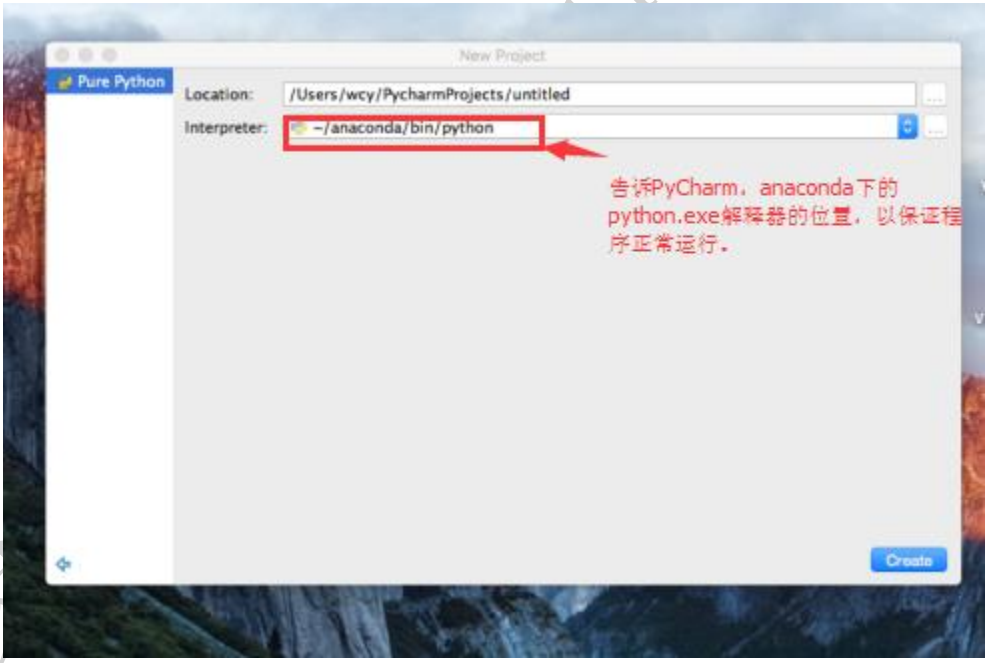


- 出现启动界面

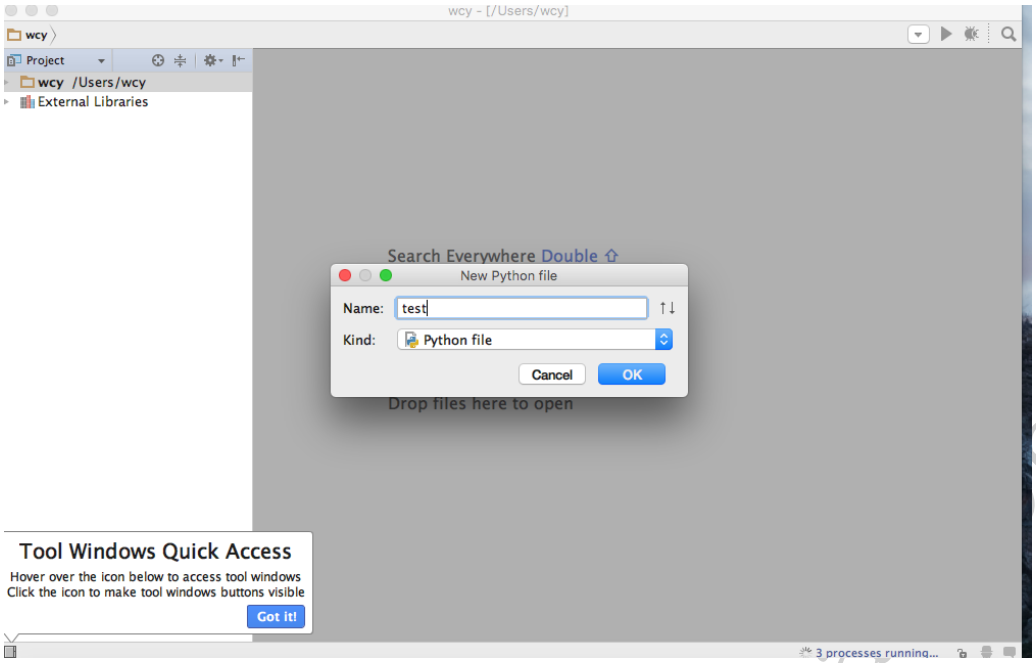
- 打开后出现下面的界面，选择 Create New Project ，即可创建一个工程。



- 选择 anaconda 作为 python 解释器

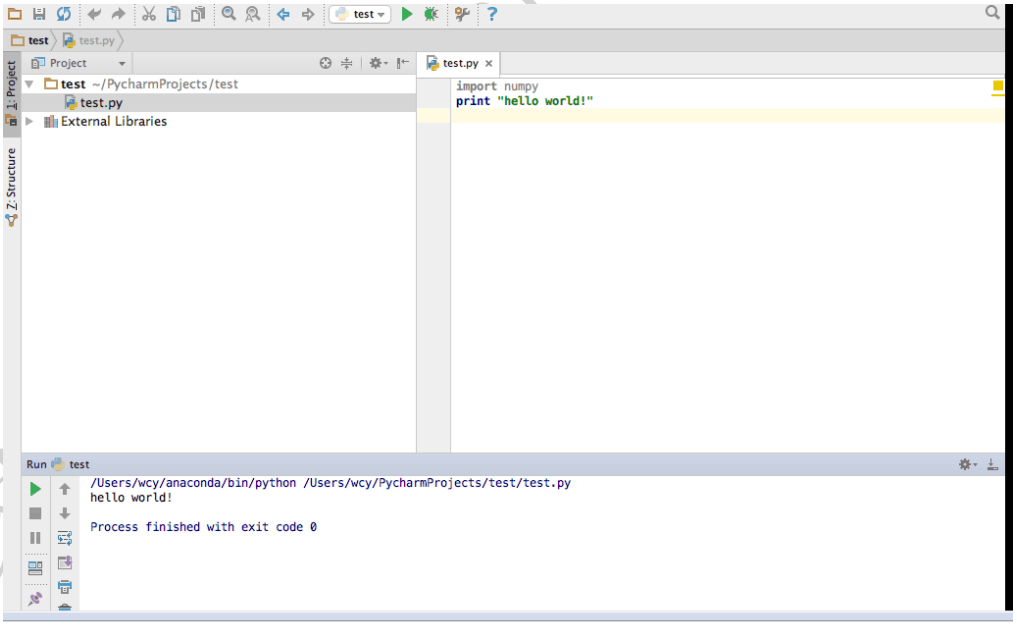


- 在上述工程下新建一个 python 文件，名字设为 test 。

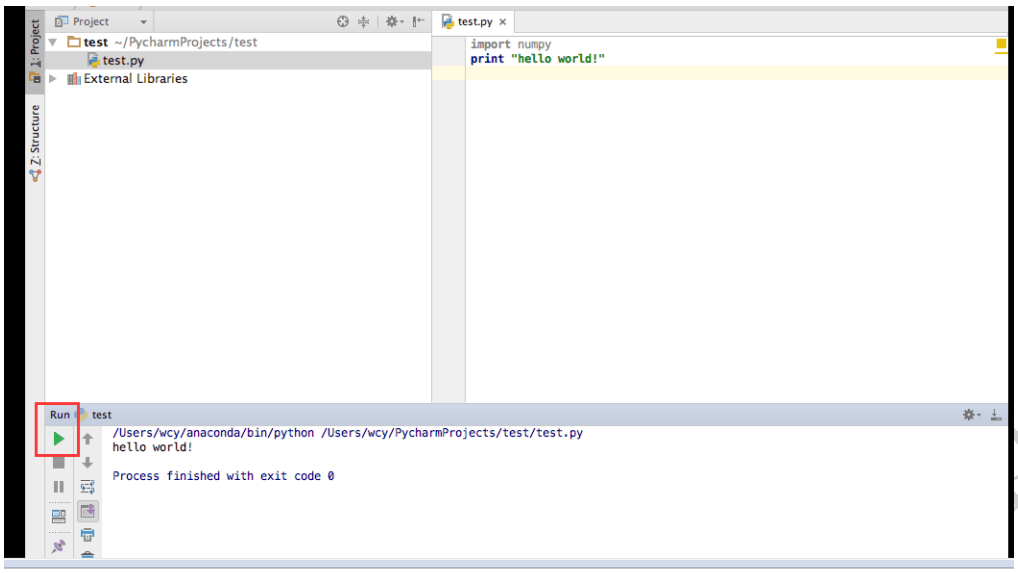


- 打开 test.py，

  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!

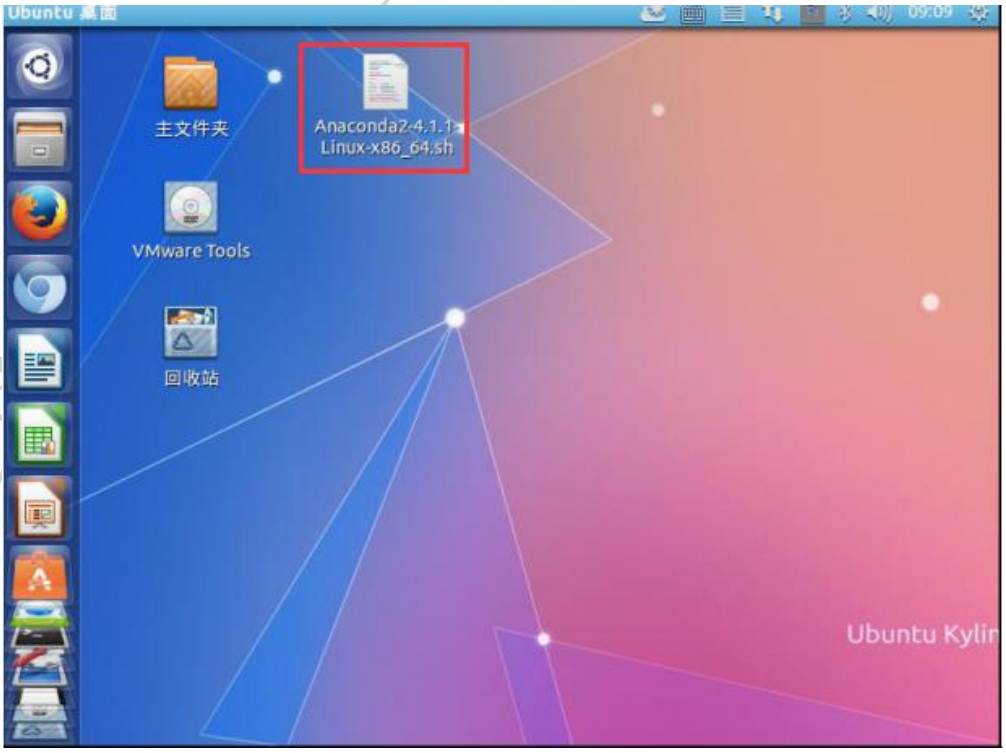


至此安装成功！

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：

```
robert@robert-VirtualBox:~$ cd ~/Downloads
robert@robert-VirtualBox:~/Downloads$ ls
Anaconda2-4.0.0-Linux-x86.sh
robert@robert-VirtualBox:~/Downloads$ bash Anaconda2-4.0.0-Linux-x86.sh

Welcome to Anaconda2 4.0.0 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

- 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。

```
Do you approve the license terms? [yes|no]
>>>
Please answer 'yes' or 'no':
>>>
```

- 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 。

```
Anaconda2 will now be installed into this location:
/home/robert/anaconda2

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/robert/anaconda2] >>>
```

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

```
creating default environment...
installation finished.
Do you wish the installer to prepend the Anaconda2 install location
to PATH in your /home/robert/.bashrc ? [yes|no]
[no] >>>
```

- 安装进行中

```
installing: jedi-0.9.0-py27_1 ...
installing: jinja2-2.8-py27_1 ...
installing: jpeg-8d-1 ...
installing: jsonschema-2.5.1-py27_0 ...
installing: jupyter-1.0.0-py27_3 ...
installing: jupyter_client-4.3.0-py27_0 ...
installing: jupyter_console-4.1.1-py27_0 ...
installing: jupyter_core-4.1.0-py27_0 ...
installing: libdynd-0.7.2-0 ...
installing: libffi-3.2.1-0 ...
installing: libgfortran-3.0.0-1 ...
installing: libpng-1.6.22-0 ...
installing: libsodium-1.0.10-0 ...
installing: libtiff-4.0.6-2 ...
installing: libxml2-2.9.2-0 ...
installing: libxslt-1.1.28-0 ...
installing: llvmlite-0.11.0-py27_0 ...
installing: locket-0.2.0-py27_1 ...
installing: lxml-3.6.0-py27_0 ...
installing: markupsafe-0.23-py27_2 ...
installing: matplotlib-1.5.1-np111py27_0 ...
installing: mistune-0.7.2-py27_0 ...
installing: mkl-11.3.3-0 ...
```

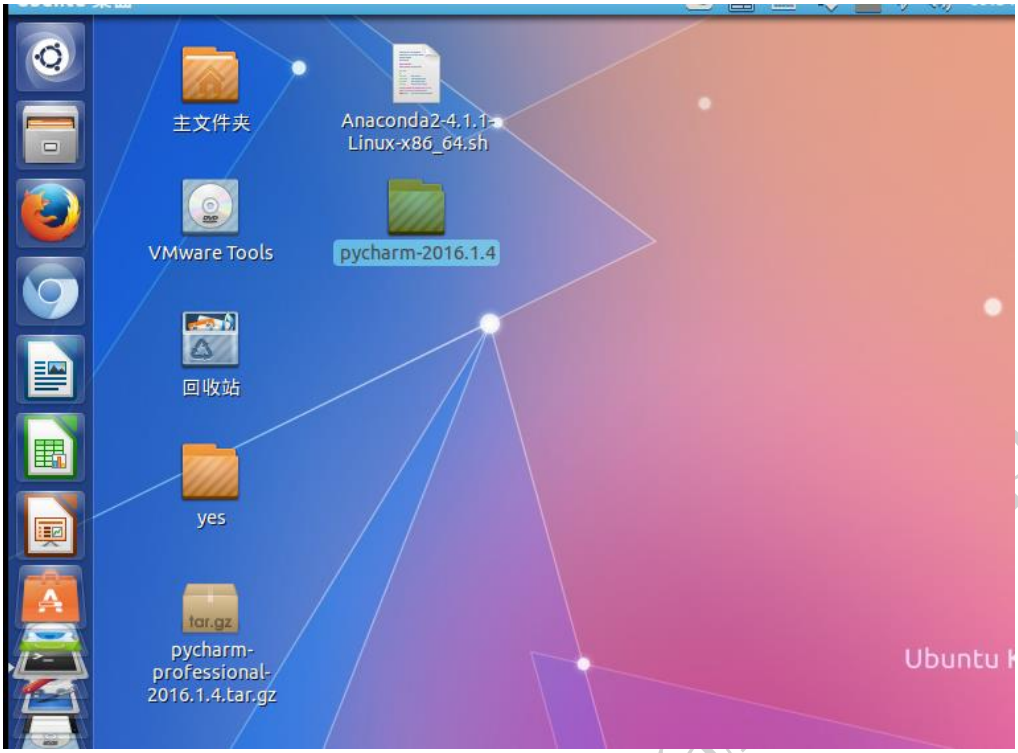
1.4.2 验证环境配置

- 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

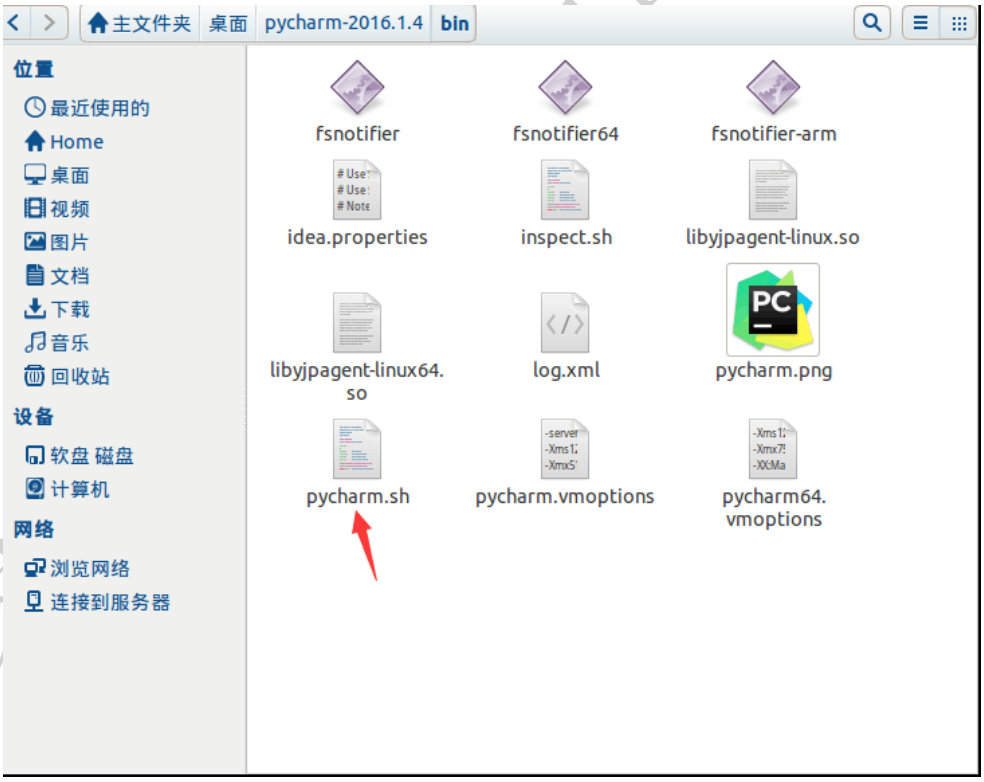
```
wcy@wcy:~/桌面$ python
Python 2.7.12 |Anaconda 4.1.1 (64-bit)| (default, Jul  2 2016, 17:42:40)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>>
```

1.4.3 PyCharm 编辑器的安装及配置

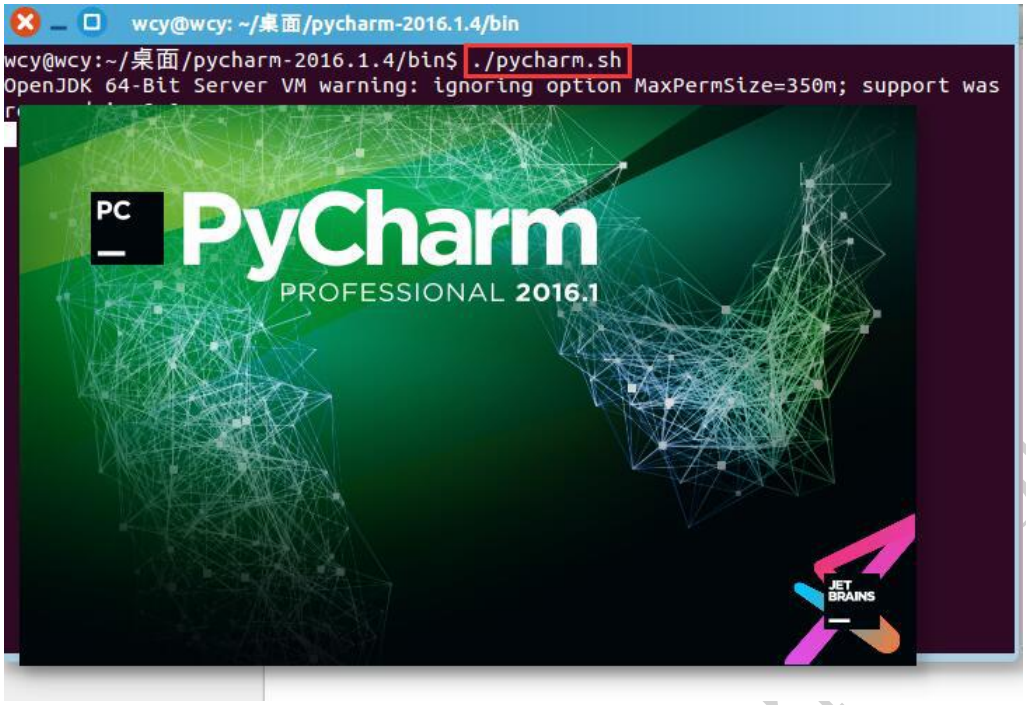
- 下载相应 PyCharm 的软件包。



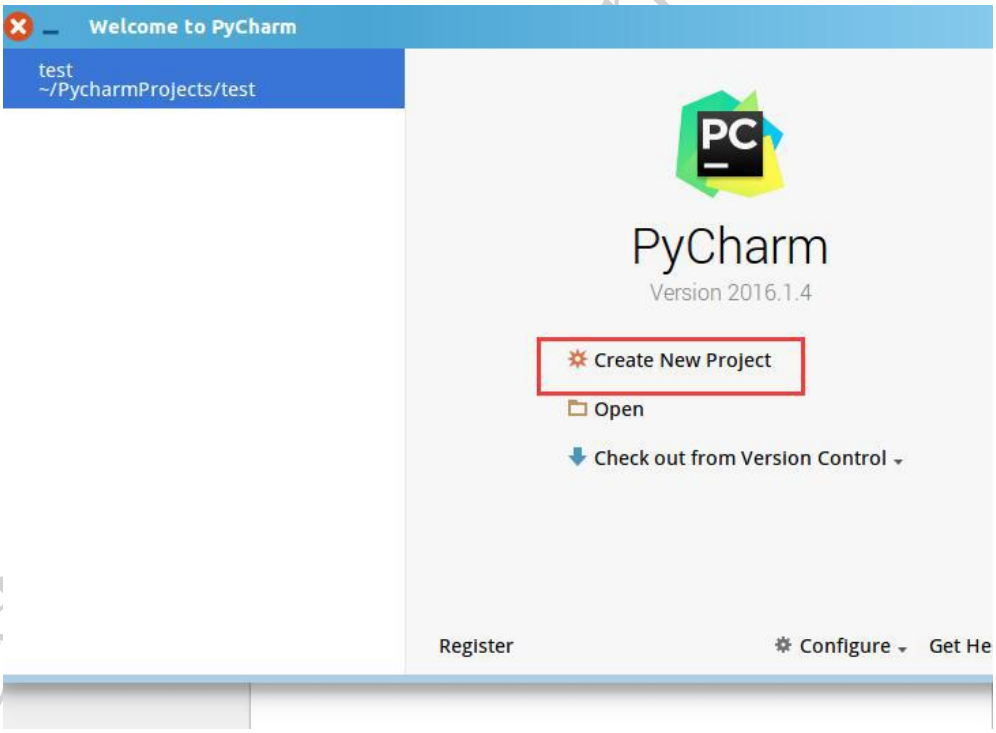
- 进入 bin 目录，如下图：



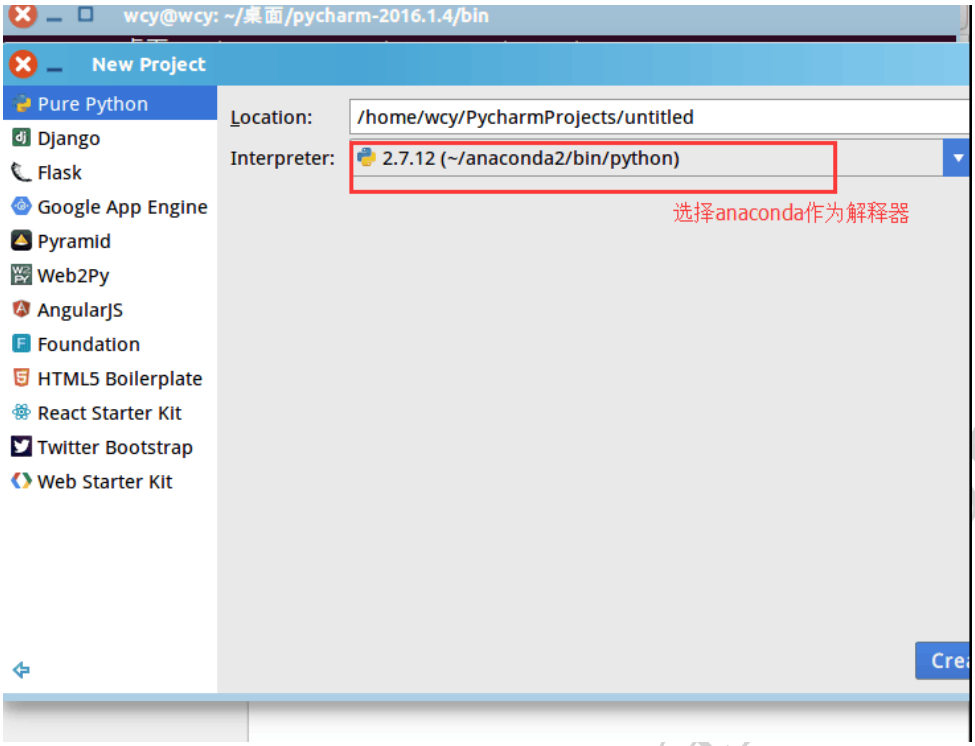
在当前目录下，右击打开终端，输入 ./pycharm.sh 如图：



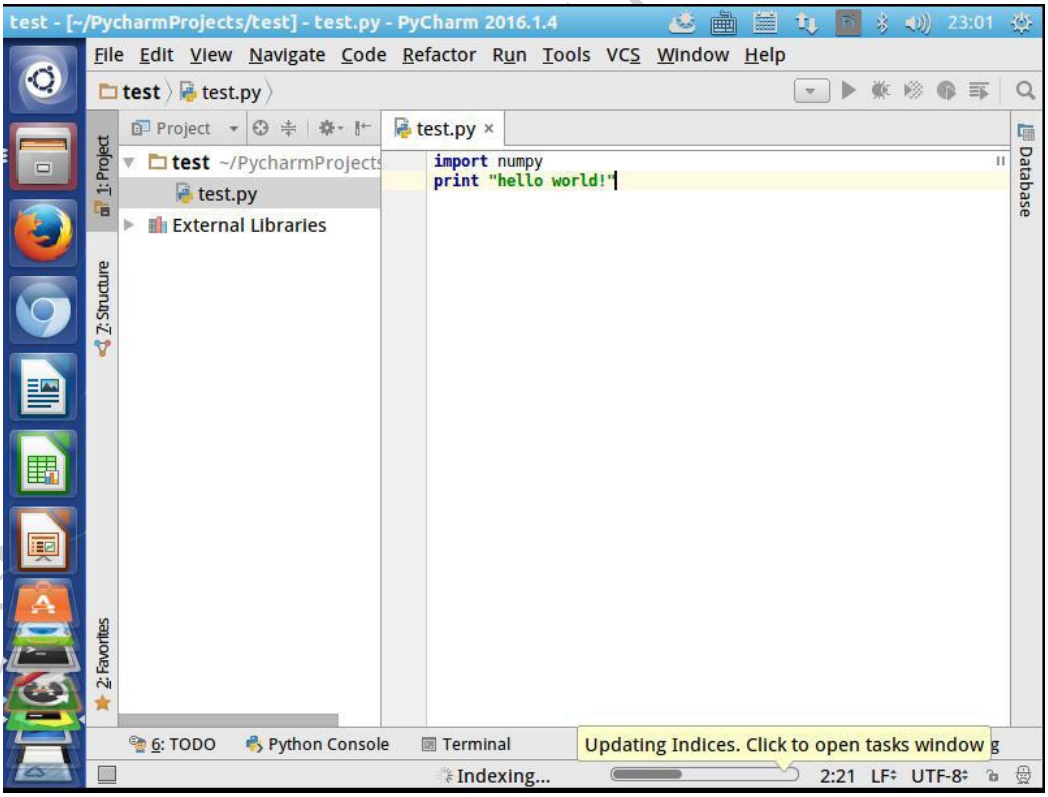
- 创建工程。



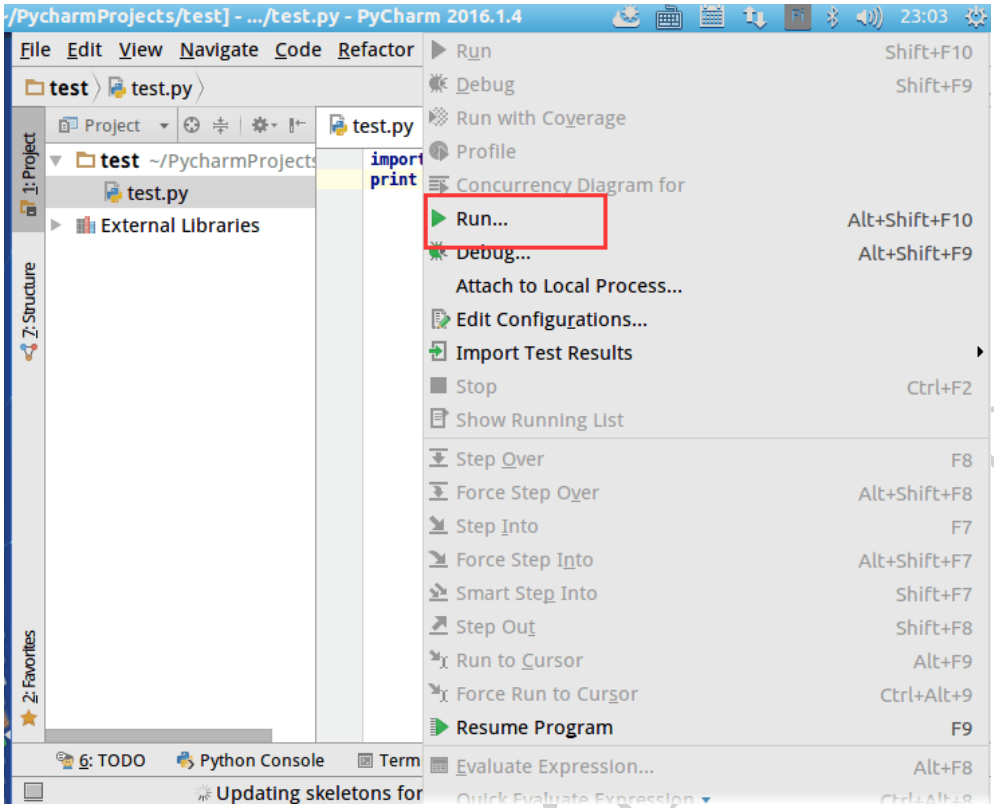
- Create New Project，创建一个测试工程，解释器默认选择了 Python2.7。



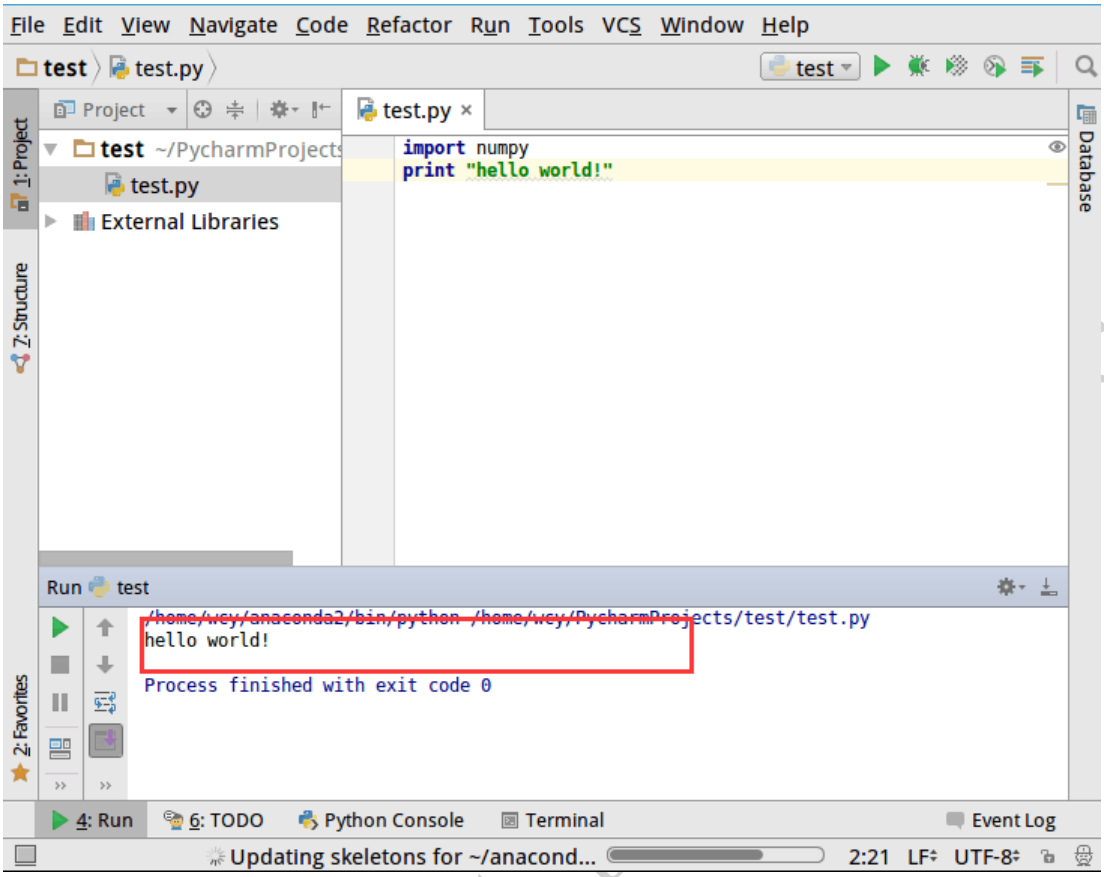
- 新建一个 python 文件，键入如下代码：



- 点击 run



- 输出结果：



至此成功。

第二章 回归

2.1 线性回归

2.1.1 实验数据:

（一）原始数据：

1、数据描述：

数据来自出版书籍《An Introduction to Statistical Learning with Applications in R》(Springer, 2013)，作者 Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani。共 200 条数据，每条数据 4 个属性。该数据可以从这个链接直接下载得到：

http://www-bcf.usc.edu/~gareth/ISL/Advertising.csv

2、**数据集信息**：Advertising.csv

数据共 4 列 200 行，每一行为一个特定的商品，前 3 列为输入特征，最后一列为输出特征。

输入特征：

(1)TV：该商品用于电视上的广告费用（以千元为单位，下同）

(2)Radio：在广播媒体上投资的广告费用

(3)Newspaper：用于报纸媒体的广告费用

输出特征：

Sales：该商品的销量

3、数据样例：

	TV	Radio	Newspaper	Sales
1	230.1	37.8	69.2	22.1
2	44.5	39.3	45.1	10.4
3	17.2	45.9	69.3	9.3
4	151.5	41.3	58.5	18.5
5	180.8	10.8	58.4	12.9
6	8.7	48.9	75	7.2
7	57.5	32.8	23.5	11.8
8	120.2	19.6	11.6	13.2
9	8.6	2.1	1	4.8
10	199.8	2.6	21.2	10.6
11	66.1	5.8	24.2	8.6
12	214.7	24	4	17.4
13	23.8	35.1	65.9	9.2
14	97.5	7.6	7.2	9.7
15	204.1	32.9	46	19
16	195.4	47.7	52.9	22.4
17	67.8	36.6	114	12.5
18	281.4	39.6	55.8	24.4
19	69.2	20.5	18.3	11.3
20	147.3	23.9	19.1	14.6
21	218.4	27.7	53.4	18
22	237.4	5.1	23.5	12.5
23	13.2	15.9	49.6	5.6
24	228.3	16.9	26.2	15.5
25	62.3	12.6	18.3	9.7
26	262.9	3.5	19.5	12
27	142.9	29.3	12.6	15
28	240.1	16.7	22.9	15.9
29	248.8	27.1	22.9	18.9
30	70.6	16	40.8	10.5
31	292.9	28.3	43.2	21.4
32	112.9	17.4	38.6	11.9
33	97.2	1.5	30	9.6
34	265.6	20	0.3	17.4
35	95.7	1.4	7.4	9.5
36	290.7	4.1	8.5	12.8
37	266.9	43.8	5	25.4
38	74.7	49.4	45.7	14.7
39	43.1	26.7	35.1	10.1
40	228	37.7	32	21.5

2.1.2 实验过程:

实验步骤: 在数据集所在目录下运行 python，复制如下代码到命令窗口：

1、收集、准备数据

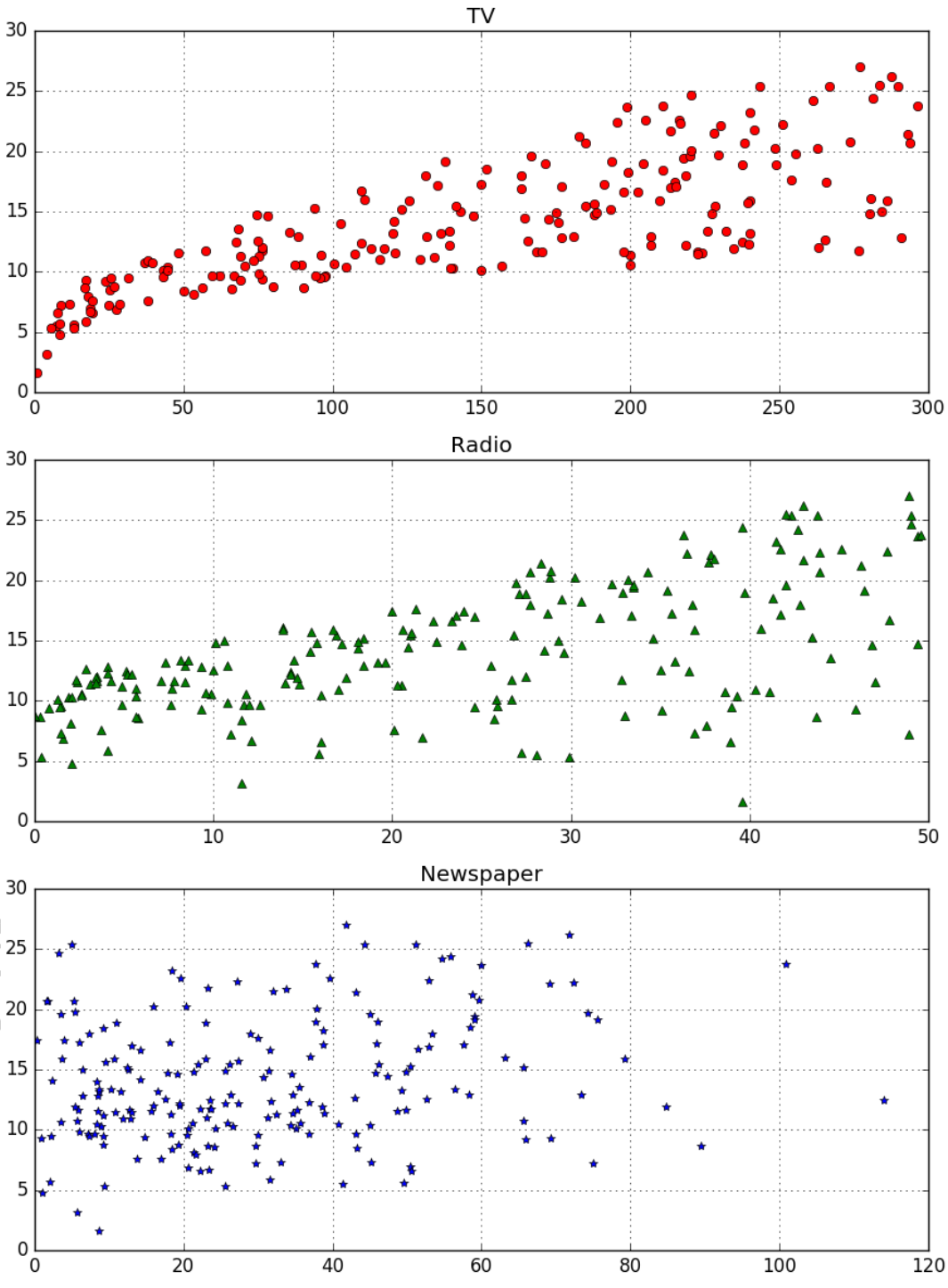
显示前五行数据。

```
import pandas as pd

data = pd.read_csv('Advertising.csv')

data.head()
```

```
>>> import pandas as pd
>>> data = pd.read_csv('Advertising.csv')
>>> data.head()
   Unnamed: 0     TV  Radio  Newspaper  Sales
0           1  230.1   37.8       69.2   22.1
1           2   44.5   39.3       45.1   10.4
2           3   17.2   45.9       69.3    9.3
3           4  151.5   41.3       58.5   18.5
4           5  180.8   10.8       58.4   12.9
>>>
```

```
if __name__ == "__main__":
path = '4.Advertising.csv'
# pandas 读入
data = pd.read_csv(path)    # TV、Radio、Newspaper、Sales
x = data[['TV', 'Radio', 'Newspaper']]
# x = data[['TV', 'Radio']]
y = data['Sales']

# 绘制1
# plt.plot(data['TV'], y, 'ro', label='TV')
# plt.plot(data['Radio'], y, 'g^', label='Radio')
# plt.plot(data['Newspaper'], y, 'b*', label='Newspaer')
# plt.legend(loc='lower right')
# plt.grid()
# plt.show()

# 绘制2
plt.figure(figsize=(9,12))
plt.subplot(311)
plt.plot(data['TV'], y, 'ro')
plt.title('TV')
plt.grid()
plt.subplot(312)
plt.plot(data['Radio'], y, 'g^')
plt.title('Radio')
```


```
plt.grid()
plt.subplot(313)
plt.plot(data['Newspaper'], y, 'b*')
plt.title('Newspaper')
plt.grid()
plt.tight_layout()
plt.show()
```



3、使用 pandas 来构建 X(特征向量)和 y(标签列)

```
#create a python list of feature names

feature_cols = ['TV', 'Radio', 'Newspaper']

# use the list to select a subset of the original DataFrame

X = data[feature_cols]

# equivalent command to do this in one line

# X = data[['TV', 'Radio', 'Newspaper']]

# print the first 5 rows

print X.head()

# check the type and shape of X

print type(X)

print X.shape

# select a Series from the DataFrame

y = data['Sales']

# equivalent command that works if there are no spaces in the column name

# y = data.Sales

# print the first 5 values

print y.head()
```

4、构建训练集与测试集

```
from sklearn.cross_validation import train_test_split

X_train,X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```


默认分割为 75%的训练集，25%的测试集

```
print X_train.shape

print y_train.shape

print X_test.shape

print y_test.shape
```

5、sklearn 的线性回归

```
from sklearn.linear_model import LinearRegression

linreg = LinearRegression()

model=linreg.fit(X_train, y_train)

print model

print linreg.intercept_

print linreg.coef_
```

```
# pair the feature names with the coefficients

zip(feature_cols, linreg.coef_)
```

由此得到各项系数，y=2.8769+0.0465∗TV+0.1791∗Radio+0.00345∗Newspaper。

6、预测

```
y_pred = linreg.predict(X_test)

print y_pred

print type(y_pred)
```


```
>>> y_pred = linreg.predict(X_test)
>>> print y_pred
[ 21.70910292  16.41055243   7.60955058  17.80769552  18.6146359
  23.83573998  16.32488681  13.43225536   9.17173403  17.333853
  14.44479482   9.83511973  17.18797614  16.73086831  15.05529391
  15.61434433  12.42541574  17.17716376  11.08827566  18.00537501
   9.28438889  12.98458458   8.79950614  10.42382499  11.3846456
  14.98082512   9.78853268  19.39643187  18.18099936  17.12807566
  21.54670213  14.69809481  16.24641438  12.32114579  19.92422501
  15.32498602  13.88726522  10.03162255  20.93105915   7.44936831
   3.64695761   7.22020178   5.9962782   18.43381853   8.39408045
  14.08371047  15.02195699  20.35836418  20.57036347  19.60636679]
>>> print type(y_pred)
<type 'numpy.ndarray'>
```

7、回归问题的评价测度

对于分类问题，评价测度(evaluation metrics)是准确率，但这种方法不适用于回归问题。我们使用针对连续数值的评价测度。

这里介绍 3 种常用的针对线性回归的测度。

1)平均绝对误差(Mean Absolute Error, MAE)

(2)均方误差(Mean Squared Error, MSE)

(3)均方根误差(Root Mean Squared Error, RMSE)

这里给出的代码示例使用 RMES。

```
print type(y_pred),type(y_test)

print len(y_pred),len(y_test)

print y_pred.shape,y_test.shape

from sklearn import metrics

import numpy as np

sum_mean=0

for i in range(len(y_pred)):

    sum_mean+=(y_pred[i]-y_test.values[i])**2
```

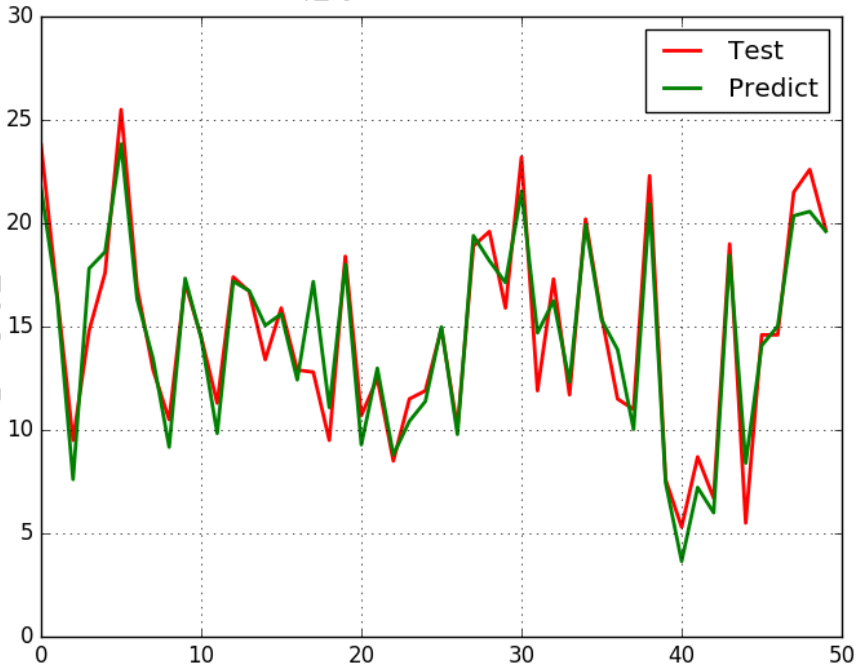

```
print "RMSE by hand:", np.sqrt(sum_mean/len(y_pred))
```

运行计算得到 RMES=1.4046。

8、作图：

```
import matplotlib.pyplot as plt

plt.figure()

plt.plot(range(len(y_pred)),y_pred,'b',label="predict")

plt.plot(range(len(y_pred)),y_test,'r',label="test")

plt.legend(loc="upper right") #显示图中的标签

plt.xlabel("the number of sales")

plt.ylabel('value of sales')

plt.show()
```

显示结果如下：



2.1.3 结果分析：

根据结果 **y=2.8769+0.0465*TV+0.1791*Radio+0.00345*Newspaper**，发现 Newspaer 的系数很小，进一步观察“收益-Newspaper” 散点图，发现 Newspaer 的线性关系并不明显，因此，尝试将这个特征移除，看看线性回归预测的结果的 RMSE 如何。

只需将实验步骤 2 中的 x 部分修改为：

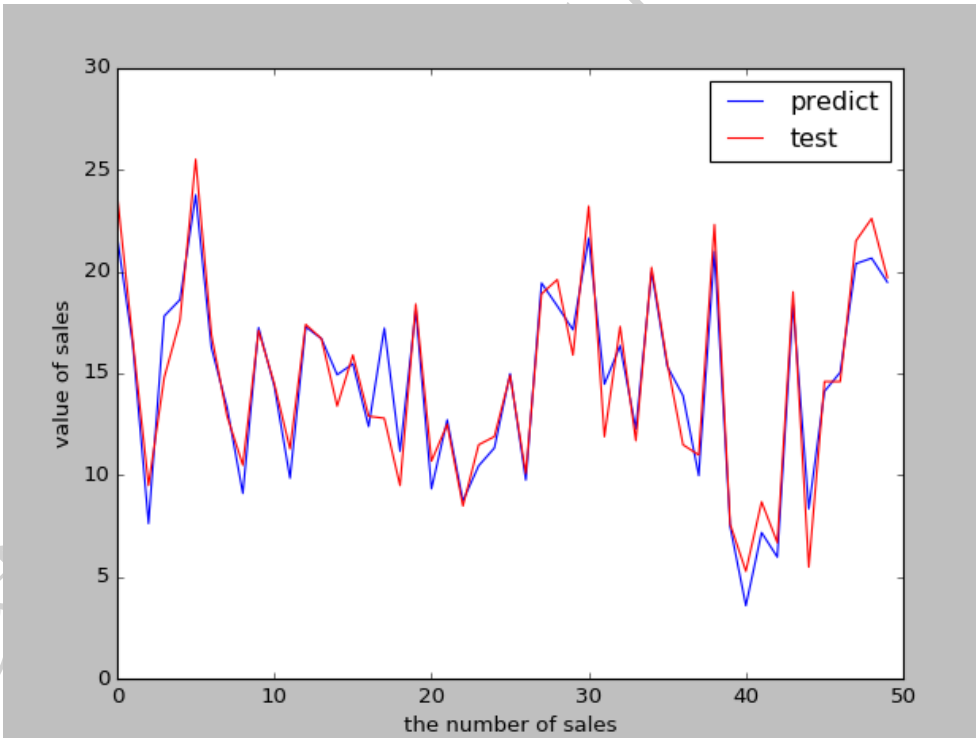
```
X = data[['TV', 'Radio']]
```

```
>>> zip(feature_cols, linreg.coef_)
[('TV', 0.046602340710768547), ('Radio', 0.18117959203112888)]
```

```
RMSE by hand: 1.38790346994
```

最后得到结果为 **1.387**

移除 Newspaper 特征后，再次将新的预测值和实际值显示在图像中。



我们在将 Newspaper 这个特征移除之后，得到 RMSE 变小了，说明 Newspaper 特征可能不适合作为预测销量的特征，于是，我们得到了新的模型。我们还可以通过不同的特征组合得到新的模型，看看最终的误差是如何的。

2.1.4 注意事项：

本模型虽然简单，但它涵盖了机器学习的相当部分的内容。如使用 75%的训练集和 25%的测试集，这往往是探索机器学习模型的第一步。分析结果的权值和特征的数据分布，我们使用了最为简单的方法：直接删除；但这样做，仍然得到了更好的预测结果。

在机器学习中有“奥卡姆剃刀”的原理，即 :如果能够用简单模型解决问题，则不使用更为复杂的模型。因为复杂模型往往增加了不确定性，造成过多的人力和物力成本，且容易过拟合。

2.2 Logistic 回归

2.2.1 实验数据:

鸢尾花数据集或许是最有名的模式识别测试数据。早在1936年，模式识别的先驱Fisher就在论文"The use of multiple measurements in taxonomic problems"中使用了它（直至今日该论文仍然被频繁引用）。该数据集包括 3 个鸢尾花类别，每个类别有 50 个样本。其中一个类别是与另外两类线性可分的，而另外两类不能线性可分。

由于 Fisher 的最原始数据集存在两个错误(35 号和 38 号样本)，因此，我们实验中使用的是修正过的数据。

该数据可以从这个链接直接下载得到：http://archive.ics.uci.edu/ml/datasets/Iris

```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
5.4,3.7,1.5,0.2,Iris-setosa
4.8,3.4,1.6,0.2,Iris-setosa
4.8,3.0,1.4,0.1,Iris-setosa
4.3,3.0,1.1,0.1,Iris-setosa
5.8,4.0,1.2,0.2,Iris-setosa
5.7,4.4,1.5,0.4,Iris-setosa
5.4,3.9,1.3,0.4,Iris-setosa
5.1,3.5,1.4,0.3,Iris-setosa
5.7,3.8,1.7,0.3,Iris-setosa
5.1,3.8,1.5,0.3,Iris-setosa
5.4,3.4,1.7,0.2,Iris-setosa
5.1,3.7,1.5,0.4,Iris-setosa
4.6,3.6,1.0,0.2,Iris-setosa
```

2.2.2 实验过程:

(一)数据描述:

该数据集共包括 150 行 ,每行为 1 个样本。每个样本有 5 个字段 ,分别是 :花萼长度 (单位 cm)、花萼宽度(单位：cm)、花瓣长度(单位：cm)、花瓣宽度(单位：cm)、类别（共 3 类，分别是：Iris Setosa、Iris Versicolour、Iris Virginica ）

下表中总结了数据集的相关信息：

数据集特征：	多变量	**记录数：**	150
属性特征：	实数	**属性数目：**	4

数据集特征：	多变量	**记录数：**	150
相关应用：	分类	**缺失值：**	无
数据网址：	http://archive.ics.uci.edu/ml/datasets/Iris		

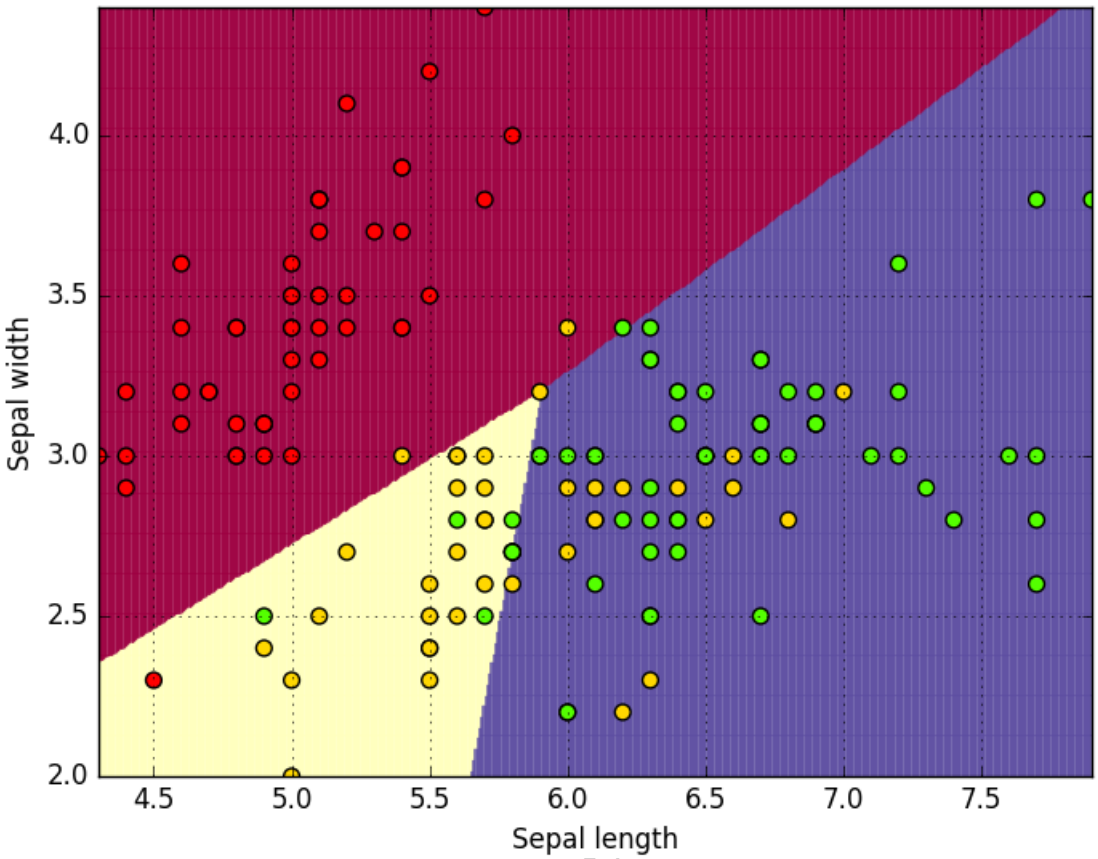
(二)实验代码:

```
#!/usr/bin/python
# -*- coding:utf-8 -*-

import numpy as np
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt


def iris_type(s):
    it = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
    return it[s]


if __name__ == "__main__":
    path = '4.iris.data'  # 数据文件路径

    # 路径，浮点型数据，逗号分隔，第4列使用函数iris_type单独处理
    data = np.loadtxt(path, dtype=float, delimiter=',', converters={4:
iris_type})

    # 将数据的0到3列组成x，第4列得到y
    x, y = np.split(data, (4,), axis=1)

    # 为了可视化，仅使用前两列特征
    x = x[:, :2]

    # print x
    # print y

    logreg = LogisticRegression()  # Logistic回归模型
    logreg.fit(x, y.ravel())       # 根据数据[x,y]，计算回归参数
```

```
    # 画图
    N, M = 500, 500     # 横纵各采样多少个值
    x1_min, x1_max = x[:, 0].min(), x[:, 0].max()   # 第0列的范围
    x2_min, x2_max = x[:, 1].min(), x[:, 1].max()   # 第1列的范围
    t1 = np.linspace(x1_min, x1_max, N)
    t2 = np.linspace(x2_min, x2_max, M)
    x1, x2 = np.meshgrid(t1, t2)                    # 生成网格采样点
    x_test = np.stack((x1.flat, x2.flat), axis=1)   # 测试点

    y_hat = logreg.predict(x_test)                  # 预测值
    y_hat = y_hat.reshape(x1.shape)                 # 使之与输入形状相同
    plt.pcolormesh(x1, x2, y_hat, cmap=plt.cm.prism)   # 预测值的显示
    plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', cmap=plt.cm.prism)
# 样本的显示
    plt.xlabel('Sepal length')
    plt.ylabel('Sepal width')
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.grid()
    plt.show()

    # 训练集上的预测结果
    y_hat = logreg.predict(x)
    y = y.reshape(-1)
    print y_hat.shape
    print y.shape
    result = y_hat == y
    print y_hat
    print y
    print result
    c = np.count_nonzero(result)
    print c
    print 'Accuracy: %.2f%%' % (100 * float(c) / float(len(result)))
```

2.2.3 结果分析：

（一）实验结果



（二）结果分析:

1、仅仅使用两个特征：花萼长度和宽度，在 150 个样本中，有 115 个分类正确，正确率为 76.67%——分类效果基本堪用，但并不十分令人满意。

2、请尝试使用更多的特征（如 4 个全部使用），再次运行该程序，比较正确率。注：试验后会发现，在 150 个样本中，将有 144 个分类正确，正确率为 96%——分类效果良好。

第三章 决策树和随机森林

3.1 目标任务

1、学习决策树和随机森林的原理、特性。

2、学习编写构造决策树的 python 代码。

3、学习使用 sklearn 训练决策树和随机森林，并使用工具进行决策树可视化。

3.2 实验数据

数据集：Iris（鸢尾花）数据集

该数据集在 Logistic 回归中已经有所介绍 ,请参见 2.2.1 节下表中列出了数据集的相关信息。

3.3 决策树（Decision Tree）特性和使用

3.3.1 决策树的特性

决策树（Decision Tree）是一种简单但是广泛使用的分类器。通过训练数据构建决策树，可以高效地对未知的数据进行分类。决策数有两大优点：

1）决策树模型可以读性好，具有描述性，有助于人工分析；

2）效率高，决策树只需要一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。

- 决策树优点：计算复杂度不高，输出结果易于理解，对中间值的缺失不敏感，可以处理不相关的特征数据。缺点：可能产生过度匹配问题（过拟合）。

- 整体思路：大原则是“将无序的数据变得更加有序”。从当前可供学习的数据集中，选择一个特性，根据这个特性划分出来的数据分类，可以获得最高的信息增益（在划分数据集前后信息发生的变化）。信息增益是熵的减少，或者是数据无序度的减少。在此划分之后，对划分出的各个分类再次进行算法，直到所有分类中均为同一类元素，或所有特性均已使用。

3.3.2 sk-learn 中决策树的使用

sklearn 中提供了决策树的相关方法，即 DecisionTreeClassifier 分类器，它能够对数据进行多分类，具体定义及部分参数详细含义如下表所示，详细可查看项目主页（http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html）

class **sklearn.tree.DecisionTreeClassifier**(criterion='gini', splitter='best', max_depth=None, min_samples_split=2,min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,class_weight=None, presort=False)		
参数说明	criterion:string	衡量分类的质量。支持的标准有"gini"（默认）代表的是 Gini impurity 与"entropy"代表的是 information gain
	splitter:string	一种用来在节点中选择分类的策略。支持的策略有"best"（默认），选择最好的分类;"random"选择最好的随机分类。
	max_depth:int or None	树的最大深度。如果是"None"（默认），则节点会一直扩展直到所有的叶子都是纯的或者所有的叶子节点都包含少于 min_samples_split 个样本点。忽视 max_leaf_nodes 是不是为 None。
	persort:bool	是否预分类数据以加速训练时最好分类的查找。在有大数据集的决策树中，如果设为 true 可能会减慢训练的过程。当使用一个小数据集或者一个深度受限的决策树中，可以减速训练的过程。默认 False

和其他分类器一样，DecisionTreeClassifier 有两个向量输入：X，稀疏或密集，大小为[n_sample,n_fearure],存放训练样本； Y，值为整型，大小为[n_sample],存放训练样本的分类标签。但由于 DecisionTreeClassifier 不支持输入文本属性和文本标签，因此需要将原始数据集中的文本标签转化为数字标签，及 X、Y 应为数字矩阵。接着将 X、Y 传给 fit()函数进行训练，得到的模型即可对样本进行预测。

决策树的简单使用

```
from sklearn import tree
X = [[0, 0], [1, 1]]
Y = [0, 1]
clf = tree.DecisionTreeClassifier()      # 初始化
clf = clf.fit(X, Y)                      # 根据XY数据训练决策树
clf.predict([[2., 2.]])                  # 根据训练出的模型预测样本
```

随机森林（Random Forest）

随机森林（Random Forest），指的是利用多棵树对样本进行训练并预测的一种分类器。它通过对数据集中的子样本进行训练，从而得到多棵决策树，以提高预测的准确性并控制在单棵决策树中极易出现的过拟合情况。

sklearn 中提供了随机森林的相关方法，即 RandomForestClassifier 分类器，它能够对数据进行多分类，具体定义及部分参数详细含义如下表所示，其中很多参数都与决策树中的参数相似。

class **sklearn.ensemble.RandomForestClassifier**(n_estimators=10, criterion='gini', max_depth=None,min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',max_leaf_nodes=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,warm_start=False, class_weight=None)		
参数说明	n_estimators: integer	随机森林中的决策树的棵树，默认为 10
	bootstrap:boolean	在训练决策树的过程中是否使用 bootstrap 抽样方法，默认为 True
	max_depth:int or None	树的最大深度。如果是"None"（默认）,则

		节点会一直扩展直到所有的叶子都是纯的或者所有的叶子节点都包含少于 min_samples_split 个样本点。忽视 max_leaf_nodes 是不是为 None。
	criterion:string	衡量分类的质量。支持的标准有"gini"（默认）代表的是 Gini impurity 与"entropy"代表的是 information gain

同样 RandomForestClassifier 也有两个向量输入：X，稀疏或密集，大小为[n_sample,n_fearure],存放训练样本； Y，值为整型，大小为[n_sample],存放训练样本的分类标签。将 X、Y 转化为数字矩阵后传给 fit()函数进行训练，得到的模型即可对样本进行预测。

3.4 实验过程

3.4.1 实验准备

1) 根据实验手册第一章安装 Python 解释器，确保 Numpy/Matplotlib/sklearn 等库正确安装。

2) 安装 graphviz 工具，便于查看决策树结构(读取 dot 脚本写成的文本文件，做图形化显示)。Graphviz 是一个开源的图形可视化软件，用于表达有向图、无向图的连接关系，它在计算机网络、生物信息，软件工程，数据库和网页设计，机器学习等诸多领域都被技术人员广泛使用。该工具的官网下载地址是：

http://www.graphviz.org/Download.php。

3.4.2 代码实现

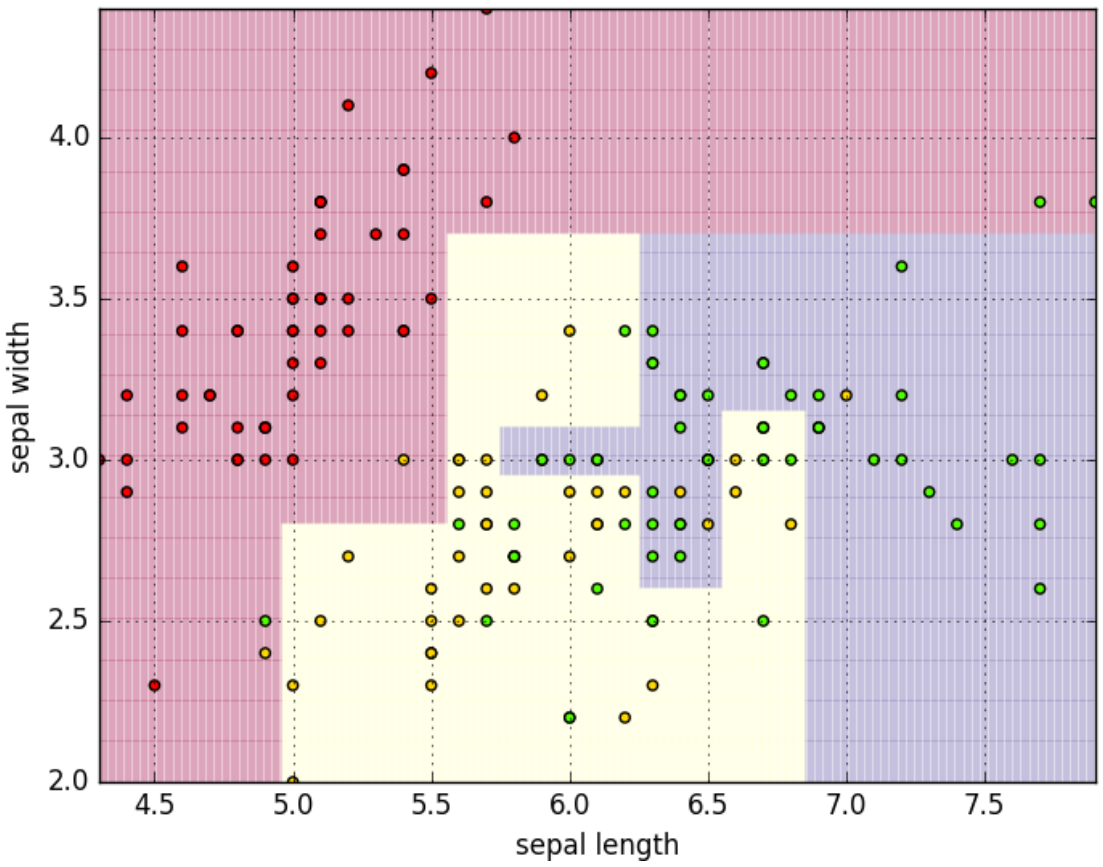
（一）使用 sklearn 的决策树做分类的相关代码：

```
 def iris_type(s):
it = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
return it[s]
```

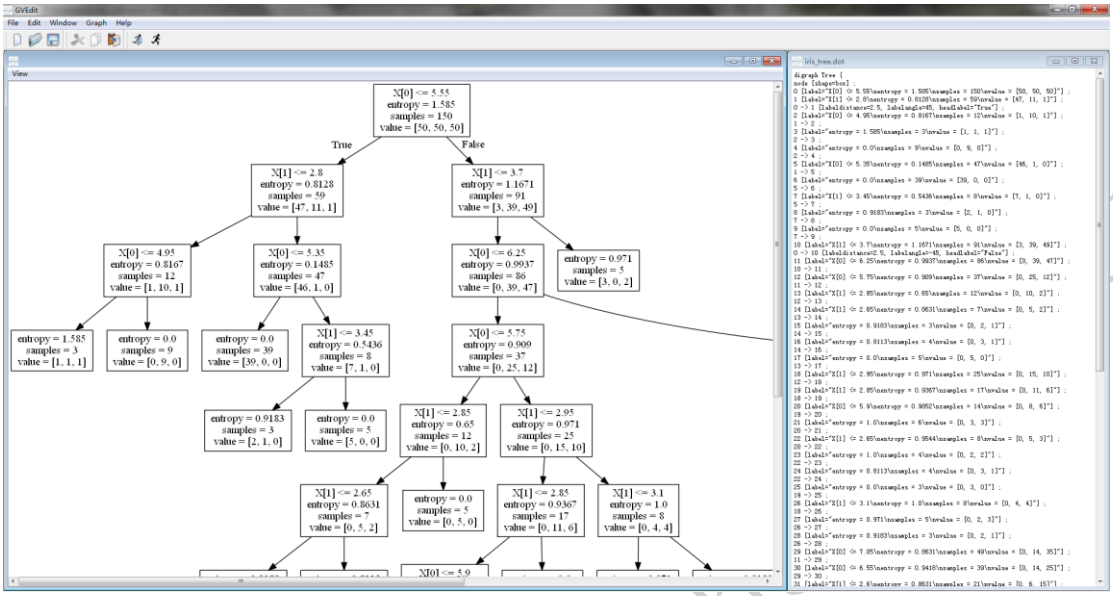
```
# 花萼长度、花萼宽度，花瓣长度，花瓣宽度
iris_feature = 'sepal length', 'sepal width', 'petal length', 'petal
width'

if __name__ == "__main__":
    path = u'4.iris.data'  # 数据文件路径

    # 路径，浮点型数据，逗号分隔，第4列使用函数iris_type单独处理
    data = np.loadtxt(path, dtype=float, delimiter=',', converters={4:
iris_type})

    # 将数据的0到3列组成x，第4列得到y
    x, y = np.split(data, (4,), axis=1)
    # 为了可视化，仅使用前两列特征
    x = x[:, :2]

    # 决策树参数估计
    # min_samples_split = 10：如果该结点包含的样本数目大于10，则(有可能)
对其分支
    # min_samples_leaf = 10：若将某结点分支后，得到的每个子结点样本数目都
大于10，则完成分支；否则，不进行分支
    clf = DecisionTreeClassifier(criterion='entropy',
min_samples_leaf=3)
    dt_clf = clf.fit(x, y)

    # 保存
    f = open("iris_tree.dot", 'w')
    tree.export_graphviz(dt_clf, out_file=f)

    # 画图
    N, M = 500, 500  # 横纵各采样多少个值
    x1_min, x1_max = x[:, 0].min(), x[:, 0].max()  # 第0列的范围
    x2_min, x2_max = x[:, 1].min(), x[:, 1].max()  # 第1列的范围
    t1 = np.linspace(x1_min, x1_max, N)
    t2 = np.linspace(x2_min, x2_max, M)
    x1, x2 = np.meshgrid(t1, t2)  # 生成网格采样点
    x_test = np.stack((x1.flat, x2.flat), axis=1)  # 测试点


    y_hat = dt_clf.predict(x_test)  # 预测值
    y_hat = y_hat.reshape(x1.shape)  # 使之与输入的形状相同
    plt.pcolormesh(x1, x2, y_hat, cmap=plt.cm.Spectral, alpha=0.1)  #
预测值的显示Paired/Spectral/coolwarm/summer/spring/OrRd/Oranges
    plt.scatter(x[:, 0], x[:, 1], c=y, edgecolors='k', cmap=plt.cm.prism)
```


```
# 样本的显示
    plt.xlabel(iris_feature[0])
    plt.ylabel(iris_feature[1])
    plt.xlim(x1_min, x1_max)
    plt.ylim(x2_min, x2_max)
    plt.grid()
    plt.show()

    # 训练集上的预测结果
    y_hat = dt_clf.predict(x)
    y = y.reshape(-1)       # 此转置仅仅为了print时能够集中显示
    print y_hat.shape       # 不妨显示下y_hat的形状
    print y.shape
    result = (y_hat == y)   # True则预测正确，False则预测错误
    print y_hat
    print y
    print result
    c = np.count_nonzero(result)    # 统计预测正确的个数
    print c
    print 'Accuracy: %.2f%%' % (100 * float(c) / float(len(result)))
```

实验结果为：



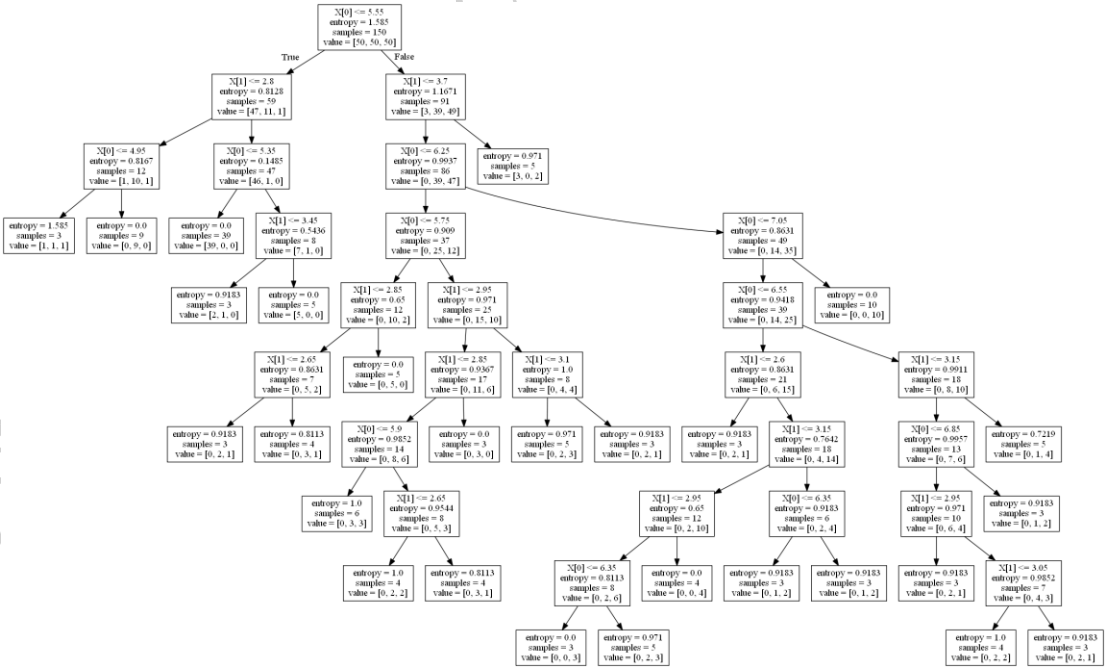
使用 Graphviz 可以讲得到的决策树 dot 文件做可视化显示：



也可以在命令行下使用 dot 保存为图片格式，如：

dot -Tpng -o iris.png iris_tree.dot

得到的图片为：

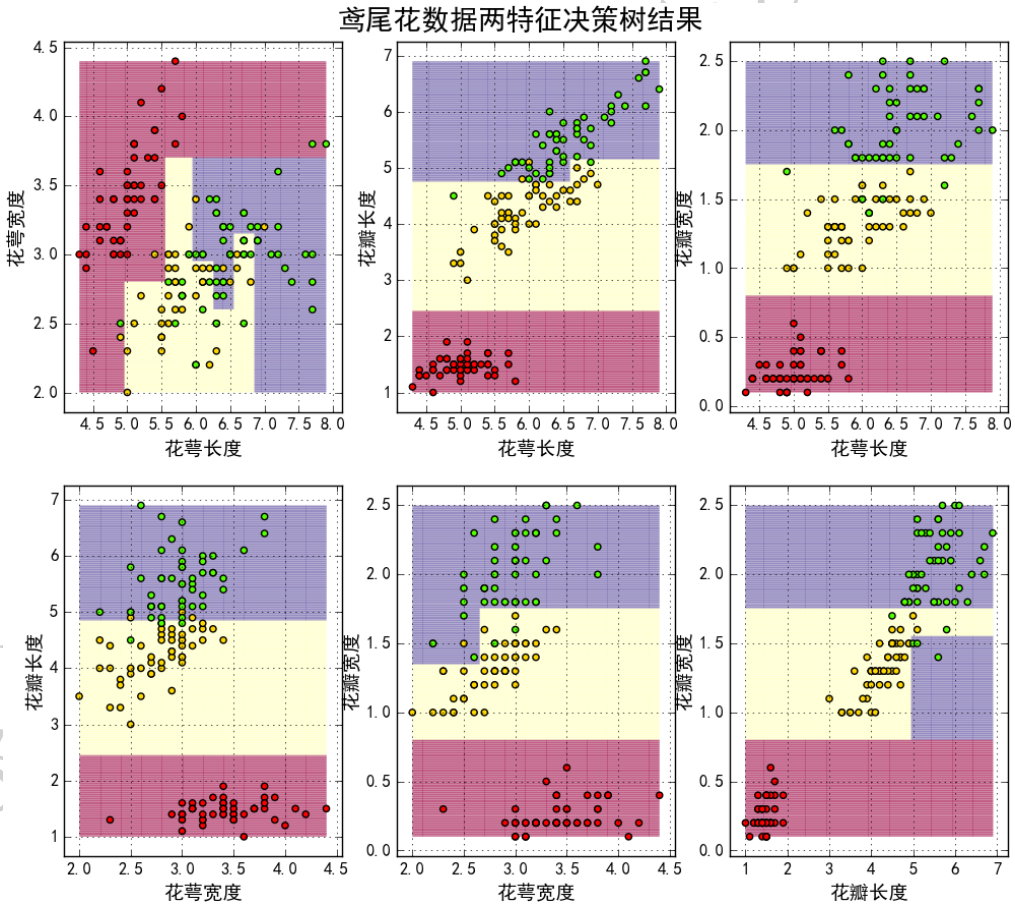


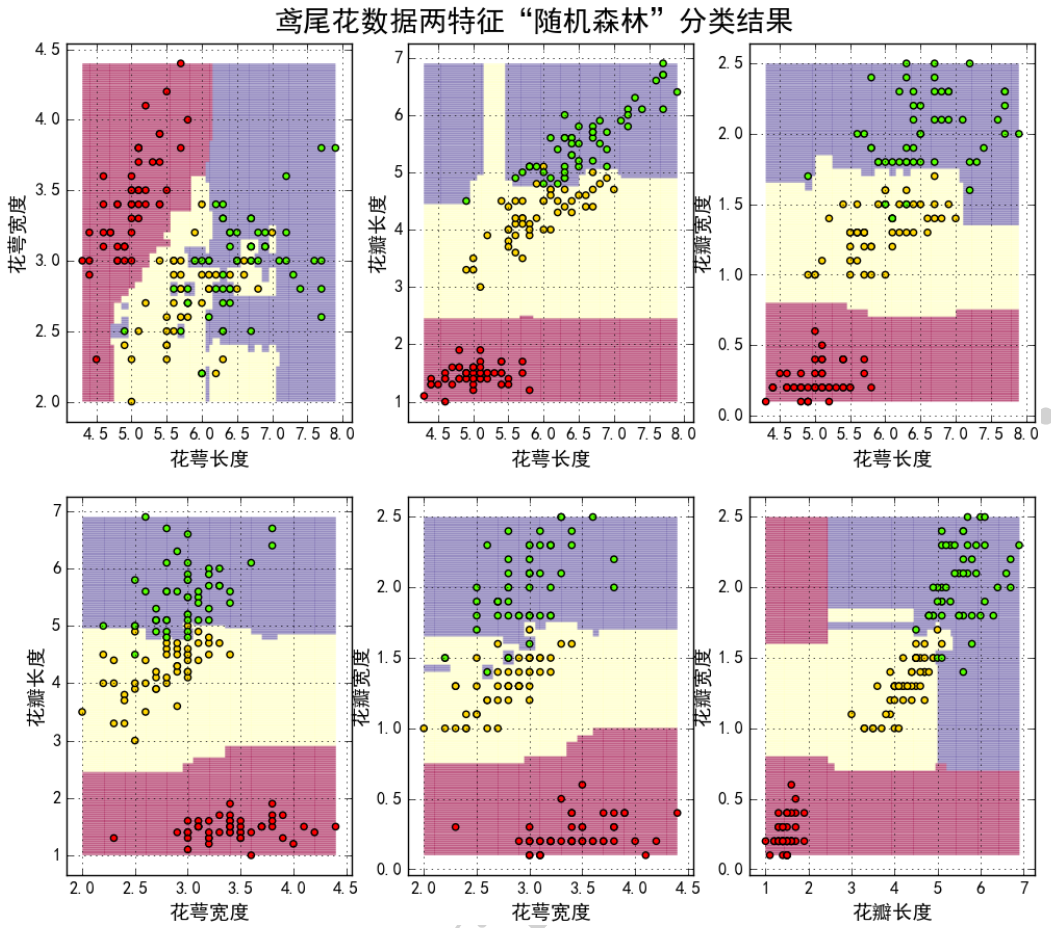
结果分析:

1. 仅仅使用两个特征：花萼长度和宽度，在 150 个样本中，有 123 个分类正确，正确率为 82%——该分类效果基本堪用，但并不十分令人满意；与 2.2 节中的 Logistic 回归相比，正确率略有上升。

2. 可以使用随机森林的方式，进一步提高在训练集上的正确率——当然，要注意给定 max_deep、min_samples_spli、min_samples_leaf 等参数，防止过拟合。

3．请使用不同的特征、随机森林的补全相关代码，这里给出实验结果如下，方便做效果参考：

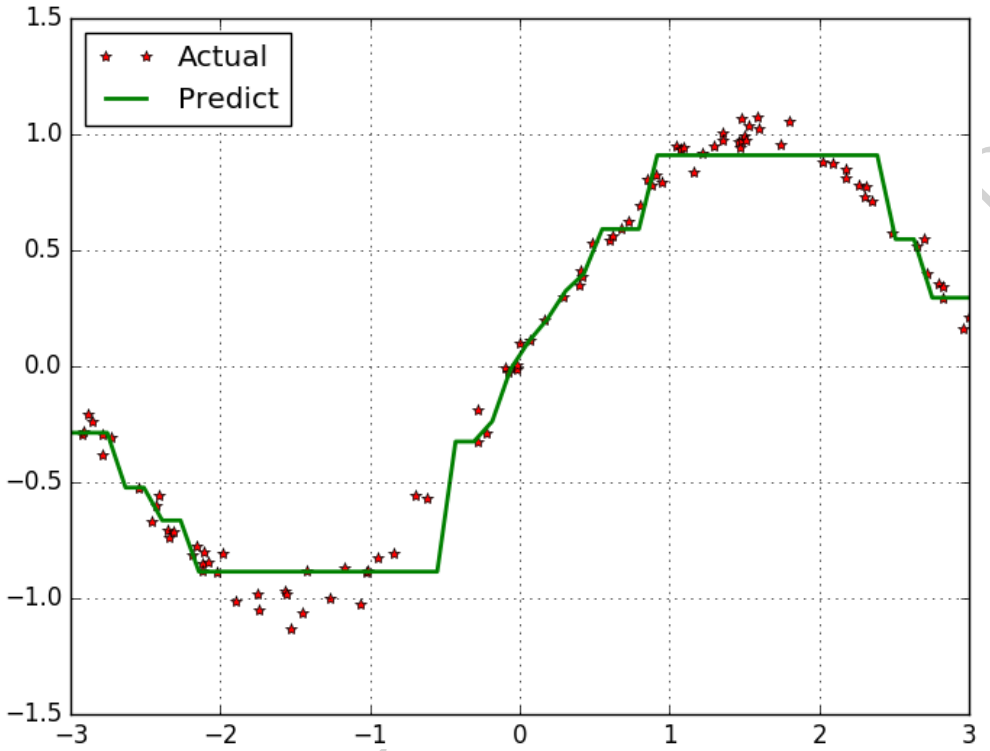


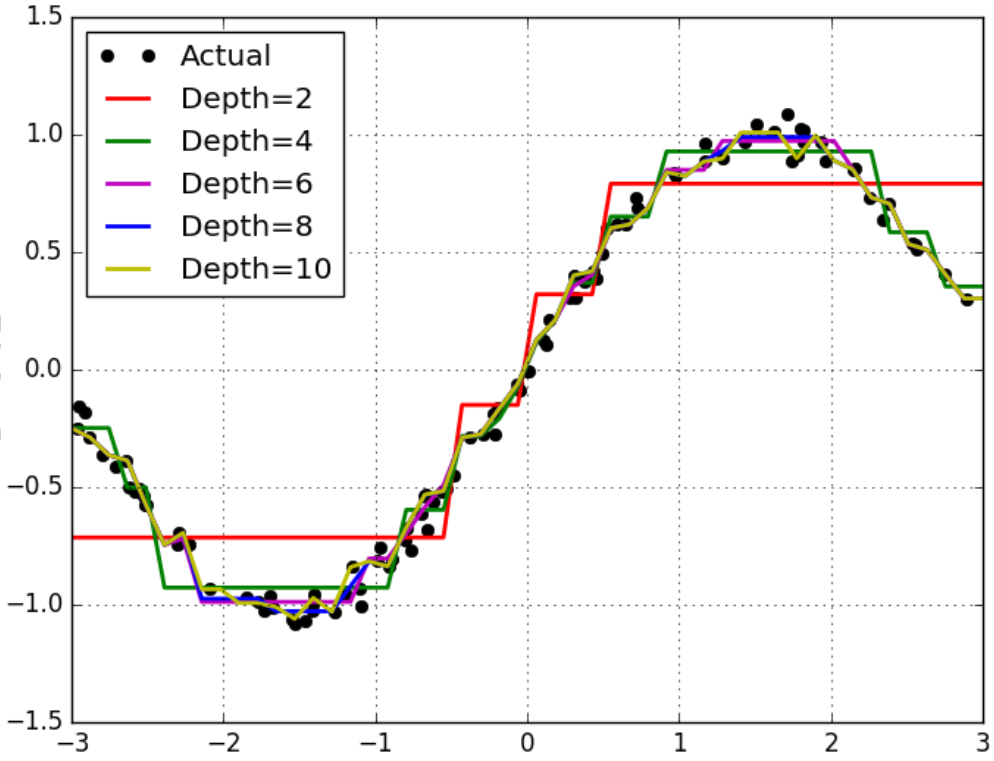


（二）使用 sklearn 的决策树做回归的相关代码：

```
N = 100
x = np.random.rand(N) * 6 - 3      # [-3,3)
x.sort()
y = np.sin(x) + np.random.randn(N) * 0.05
x = x.reshape(-1, 1)  # 转置后，得到N 个样本，每个样本都是1 维的

# 比较决策树的深度影响
depth = [2, 4, 6, 8, 10]
clr = 'rgmby'
reg = [ DecisionTreeRegressor(criterion='mse', max_depth=depth[0]),
    DecisionTreeRegressor(criterion='mse', max_depth=depth[1]),
    DecisionTreeRegressor(criterion='mse', max_depth=depth[2]),
    DecisionTreeRegressor(criterion='mse', max_depth=depth[3]),
    DecisionTreeRegressor(criterion='mse', max_depth=depth[4])]

plt.plot(x, y, 'ko', linewidth=2, label='Actual')
x_test = np.linspace(-3, 3, 50).reshape(-1, 1)
for i, r in enumerate(reg):
    dt = r.fit(x, y)
```

```
    y_hat = dt.predict(x_test)
    plt.plot(x_test, y_hat, '-', color=clr[i], linewidth=2,
label='Depth=%d' % depth[i])
plt.legend(loc='upper left')
plt.grid()
plt.show()
```



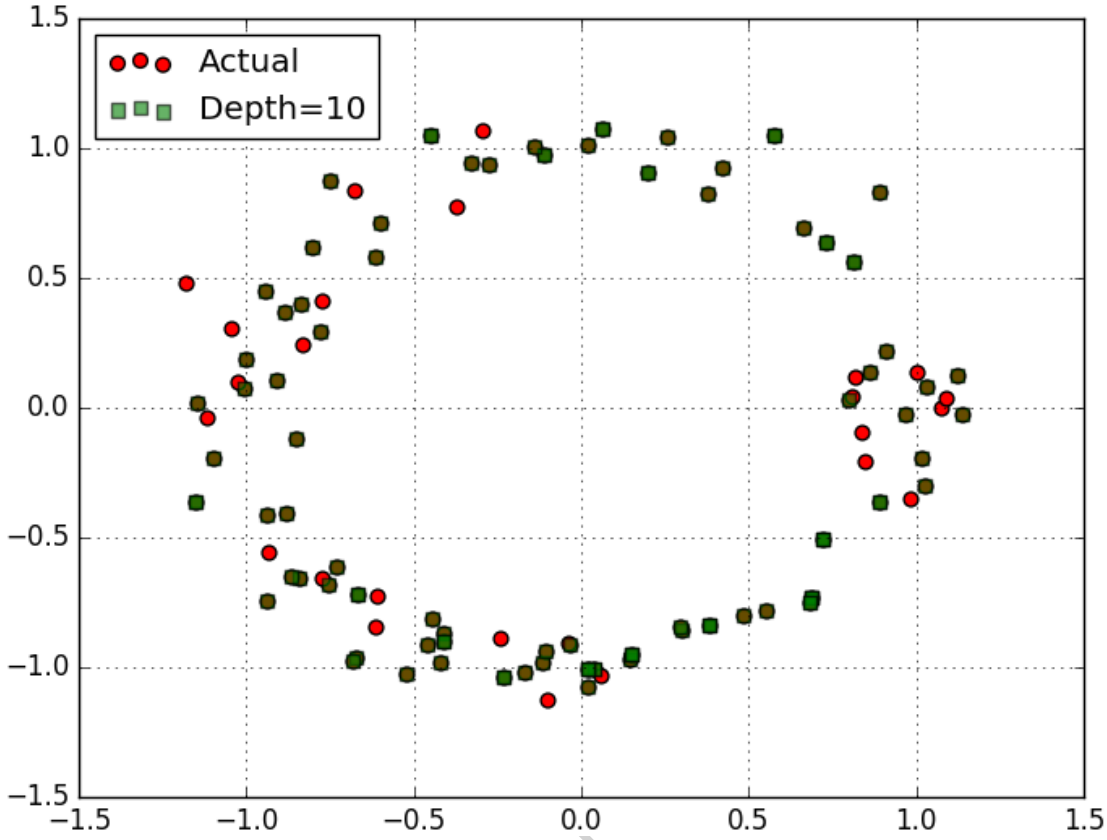


（三）决策树多输出预测

代码：

```
N = 100
x = np.random.rand(N) * 8 - 4     # [-4,4)
x.sort()
y1 = np.sin(x) + np.random.randn(N) * 0.1
y2 = np.cos(x) + np.random.randn(N) * 0.1
y = np.vstack((y1, y2)).T
x = x.reshape(-1, 1)  # 转置后，得到N 个样本，每个样本都是 1 维的

deep = 10
reg = DecisionTreeRegressor(criterion='mse', max_depth=deep)
dt = reg.fit(x, y)

x_test = np.linspace(-4, 4, num=100).reshape(-1, 1)
y_hat = dt.predict(x_test)
plt.scatter(y[:, 0], y[:, 1], c='r', s=40, label='Actual')
plt.scatter(y_hat[:, 0], y_hat[:, 1], c='g', marker='s', s=40,
label='Depth=%d' % deep, alpha=0.6)
plt.legend(loc='upper left')
plt.grid()
plt.show()
```

效果：



第九章 Python 数值计算与机器学习库

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]

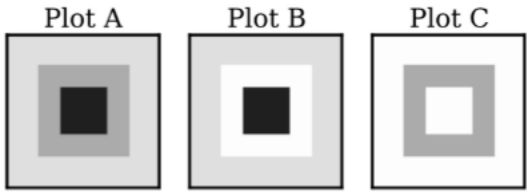
注意 :第二种方法速度更快 ,而且可以用“~ index”很容易的得到与 index 相反的 逻辑数组。

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

$$\begin{aligned} 3x + 6y - 5z &= 12 \\ x - 3y + 2z &= -2 \\ 5x - y + 4z &= 10 \end{aligned}$$

$$\begin{bmatrix} 3 & 6 & -5 \\ 1 & -3 & 2 \\ 5 & -1 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 12 \\ -2 \\ 10 \end{bmatrix}$$

```
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

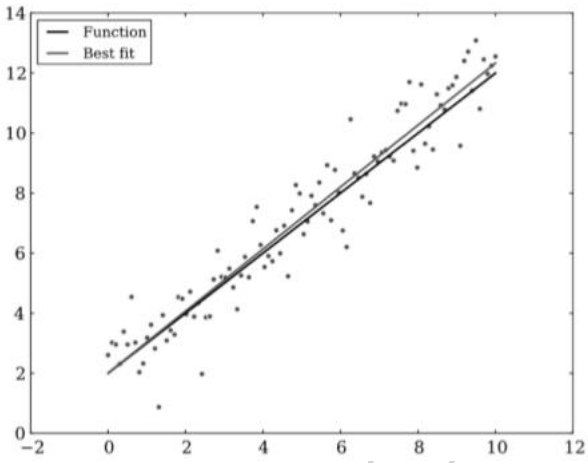
（1）数据建模和拟合

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

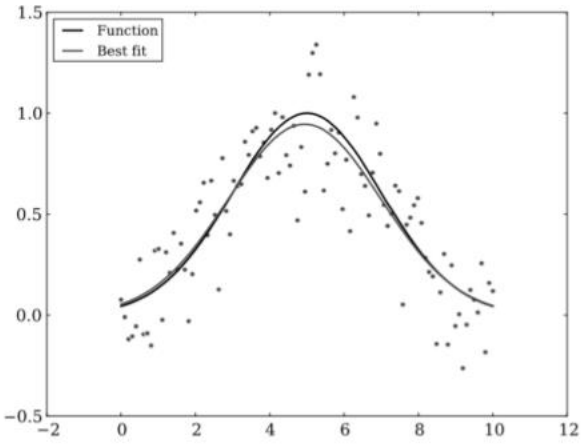
```
import numpy as np from scipy.optimize
import curve_fit

#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)
```

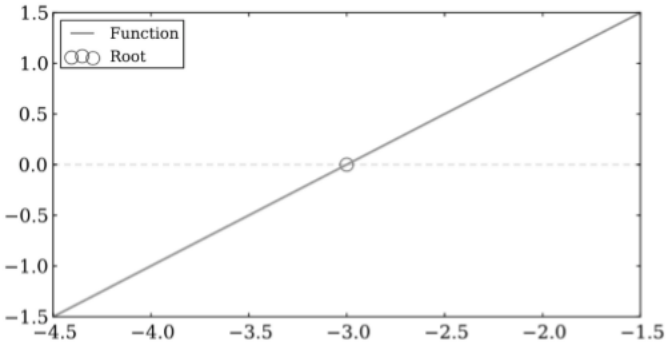


（2）函数求解

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

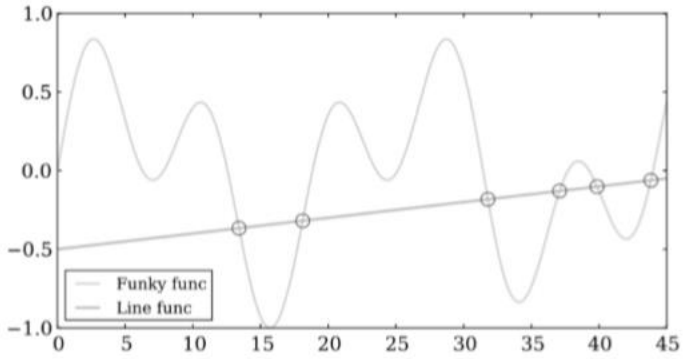
```
from scipy.optimize import fsolve
import numpy as np
line = lambda x: x + 3
solution = fsolve(line, -2)
print solution
```



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))
```

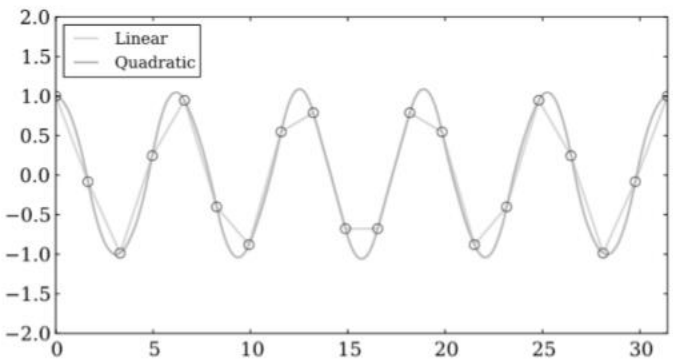


2. 插值

（1）interp1d

例：正弦函数插值

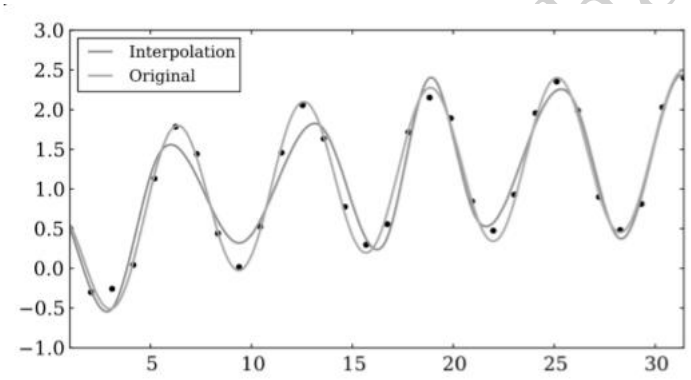
```
import numpy as np from scipy.interpolate
import interp1d

# 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)
```

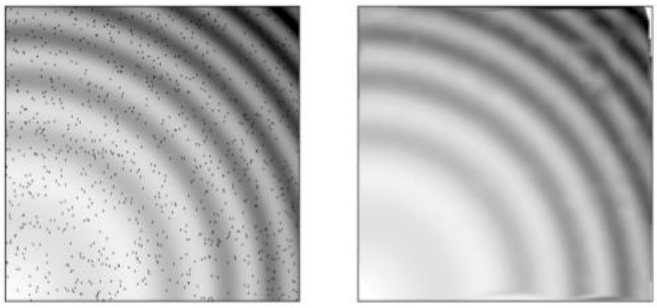


（3）griddata

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')
```

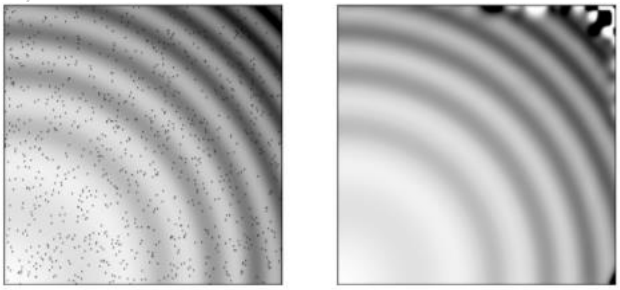


上图中，左侧为原始数据，其中的黑点是待插值的样本，右图为插值后的数据。要 想提高质量，生成更大的样本数据即可。

（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。

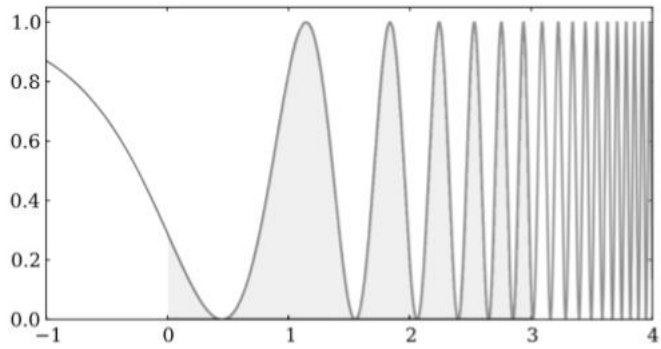
3. 积分

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：

$$\int_0^3 \cos^2(e^x)\,dx$$

```
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)
```

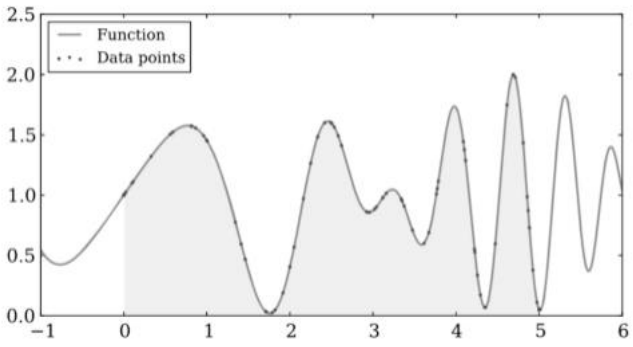


（2）数值积分

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.
```



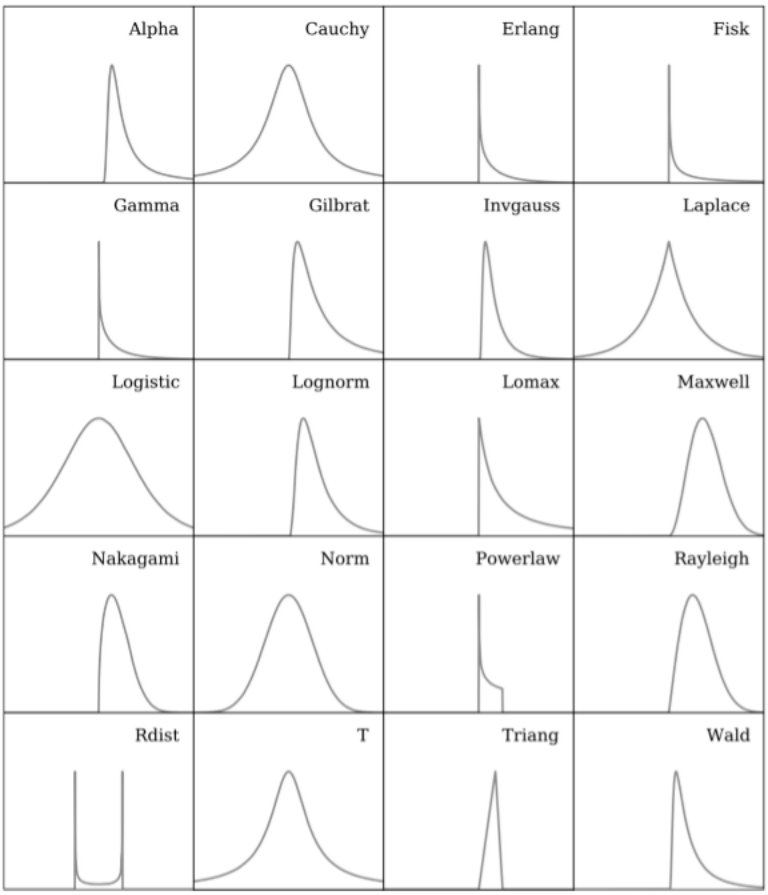
4. 统计

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。

$$\mathrm{PDF} = e^{(-x^2/2)/\sqrt{2\pi}}$$

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）:

$$\mathrm{PMF} = (1-p)^{(k-1)}\, p$$

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）。

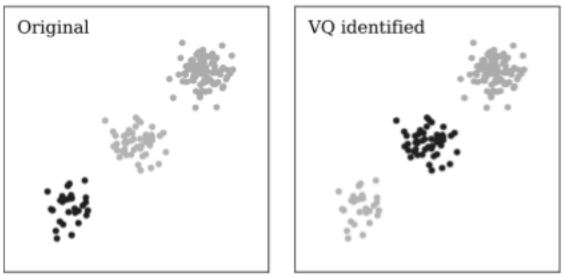
(1)矢量量化（Vector Quantization）

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

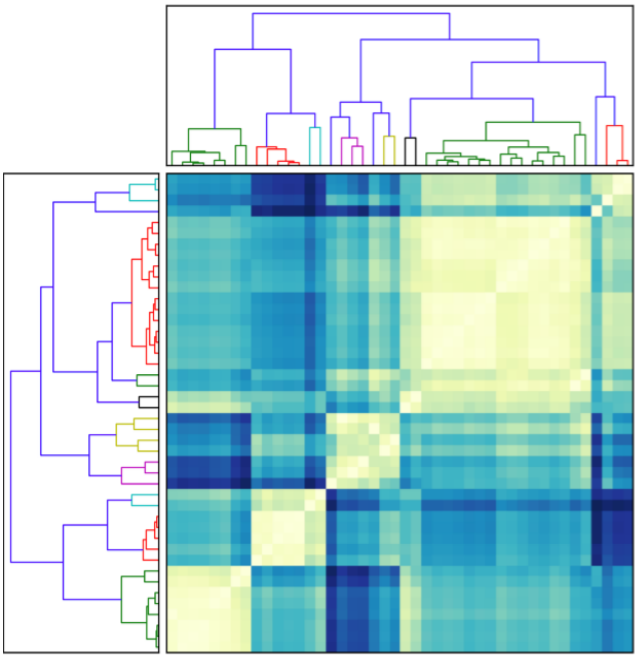
```
    ax2 = fig.add_axes([0.3, 0.71, 0.6, 0.2])
    Y2 = hy.linkage(D, method='average')
    cutoff = 0.3 * np.max(Y2[:, 2])
    Z2=hy.dendrogram(Y2,color_threshold=cutoff)
    ax2.xaxis.set_visible(False) ax2.yaxis.set_visible(False)
```

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()
```

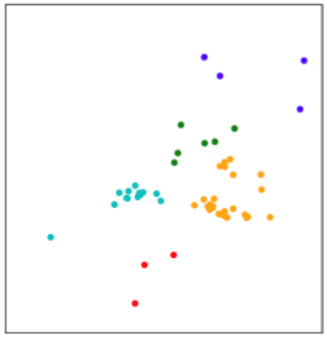



在上图虽然计算了数据点之间的距离，但是还是难以将各类区分开。函数 fcluster 可以根据阈值来区分各类，其输出结果依赖于 linkage 函数所采用的方法，如 complete 或 single 等，它的第二个参数即是阈值。dendrogram 函数中默认的阈值是 0.7 * np.max(Y[:, 2])，这里还使用 0.3。

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 10^6 级别的数据没什么大问题，当数据量达到 10^7 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 10^{10} 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()** 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：

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, penalty=l2, random_state=None, tol=0.0001)
precision recall f1-score support

  0.0       0.79      0.89      0.84       500
  1.0       0.74      0.55      0.63       268

avg / total 0.77 0.77 0.77 768

[[447 53]
[120 148]]
```

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：

GaussianNB()
precision recall f1-score support

```
  0.0       0.80      0.86      0.83       500
  1.0       0.69      0.60      0.64       268
```

avg / total 0.76 0.77 0.76 768

[[429 71]
[108 160]]

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：

KNeighborsClassifier(algorithm=auto, leaf_size=30, metric=minkowski, n_neighbors=5, p=2, weights=uniform)
precision recall f1-score support

```
  0.0       0.82      0.90      0.86       500
  1.0       0.77      0.63      0.69       268
```

avg / total 0.80 0.80 0.80 768

[[448 52]
[98 170]]

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：

```
DecisionTreeClassifier(compute_importances=None, criterion=gini,
max_depth=None, max_features=None, min_density=None,
min_samples_leaf=1, min_samples_split=2, random_state=None,
splitter=best)
precision recall f1-score support

  0.0       1.00      1.00      1.00       500
  1.0       1.00      1.00      1.00       268

avg / total 1.00 1.00 1.00 768

[[500 0]
[ 0 268]]
```

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：

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0, degree=3,
gamma=0.0,
kernel=rbf, max_iter=-1, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False)
precision recall f1-score support

  0.0       1.00      1.00      1.00       500
  1.0       1.00      1.00      1.00       268

avg / total 1.00 1.00 1.00 768

[[500 0]
[ 0 268]]
```

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：

```
GridSearchCV(cv=None,
estimator=Ridge(alpha=1.0,          copy_X=True,          fit_intercept=True,
max_iter=None,
normalize=False, solver=auto, tol=0.001),
estimator__alpha=1.0, estimator__copy_X=True,
estimator__fit_intercept=True, estimator__max_iter=None,
estimator__normalize=False, estimator__solver=auto,
estimator__tol=0.001, fit_params={}, iid=True, loss_func=None,
n_jobs=1,
param_grid={'alpha':  array([  1.00000e+00,  1.00000e-01,  1.00000e-02,
1.00000e-03,
1.00000e-04, 0.00000e+00])},
pre_dispatch=2*n_jobs, refit=True, score_func=None, scoring=None,
verbose=0)
0.282118955686
1.0
```

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：

```
RandomizedSearchCV(cv=None,
estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
max_iter=None,
normalize=False, solver=auto, tol=0.001),
estimator__alpha=1.0, estimator__copy_X=True,
estimator__fit_intercept=True, estimator__max_iter=None,
estimator__normalize=False, estimator__solver=auto,
estimator__tol=0.001, fit_params={}, iid=True, n_iter=100,
n_jobs=1,
param_distributions={'alpha': <scipy.stats.distributions.rv_frozen object at
0x04B86DD0>},
pre_dispatch=2*n_jobs, random_state=None, refit=True,
scoring=None, verbose=0)
0.282118643885
0.988443794636
```