基于 Jupyter Notebook 的 Tensorflow 深度学习教程

环境搭建

学习 Tensorflow，首先要配置好 Tensorflow 的运行环境，这也是很多入门者感到困难的地方。要保证 Tensorflow 能够正常运行，必须确保软件版本，各种依赖库和组件的安装都正确。举例来说，安装 Python 时，有 2.x 和 3.x 不同的版本，有面向 32 位和 64 位不同的操作系统环境，还必须有各种依赖库如 Numpy、Matplotlib 等，往往还要配置环境变量。如果某些版本的模块与当前环境不兼容，那就会出现各种各样的错误。开发者常常碰到的一个场景是：在自己的机器上可以正常运行，换一台机器上就运行不了。环境配置如此麻烦，换一台机器，就要重来一次，费时费力。能不能从根本上解决问题？安装软件的时候，能不能把正确的环境打包复制过来？这就是 Docker 起作用的地方。

关于Docker

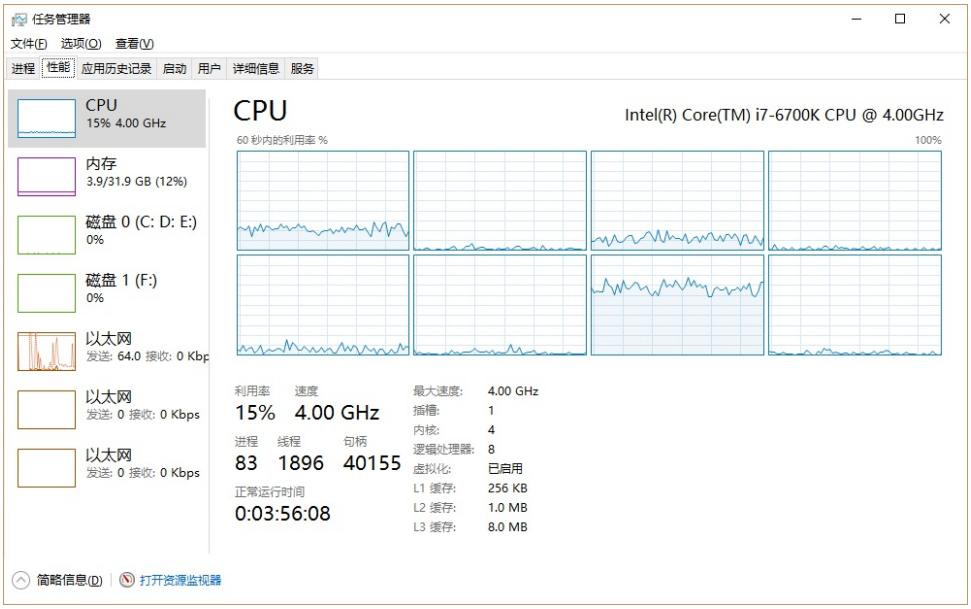
Docker（搬运工）是一个遵从 Apache 协议开源的应用容器引擎，它把要运行的应用以及依赖软件打包到一个轻量级、可移植的自给自足的容器中，然后可以发布到任何流行的 Linux、Windows 和 MacOS 机器上。Docker 的推出具有划时代的意义，它彻底释放了计算虚拟化的威力，极大提高了应用的部署效率，降低了云计算资源的供应成本。使用 Docker，我们可以把配置好的 Tensorflow 资源下载到自己的机器上直接运行，避免了繁琐的安装和配置，极大的提高效率节约时间。Docker 有三个基本要素，1）Docker Containers（容器）：负责应用程序的运行，包括操作系统、用户添加的文件以及元数据；2）Docker Images（镜像）：是一个只读模板，用来运行 Docker 容器；3）DockerFile：文件指令集，用来说明如何自动创建 Docker 镜像。镜像文件是 Docker 应用的基础，在 Docker Hub上存储了大量的公共镜像，包括我们要使用的 Tensorflow 镜像。

安装Docker

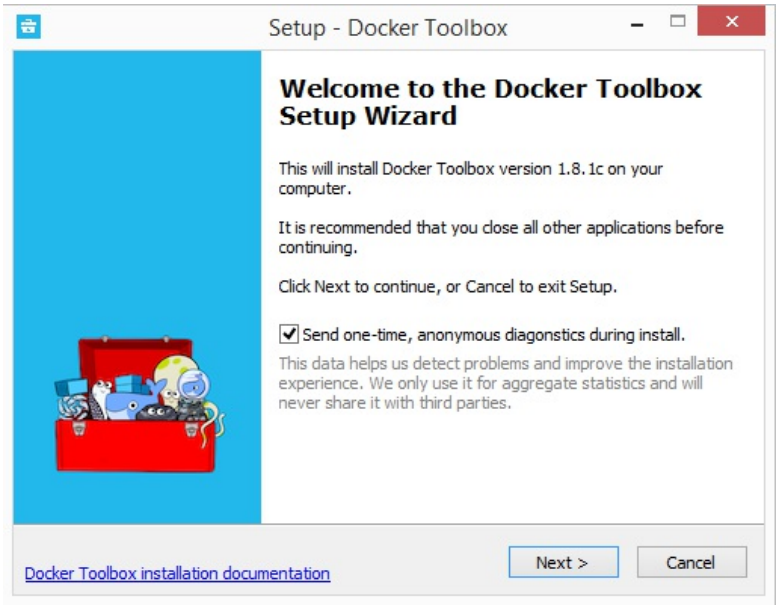
我们以 Windows 系统为例，介绍安装 Docker 相关工具的过程，MacOS 系统的安装可以参照

https://docs.docker.com/toolbox/toolbox_install_mac/ ；Linux 系统（包括Ubuntu、Debian、CentOS和Fedora）可以安装 Dokcer CE（Community Edition），参见 https://docs.docker.com/install/linux/docker-ce/ubuntu/ 。

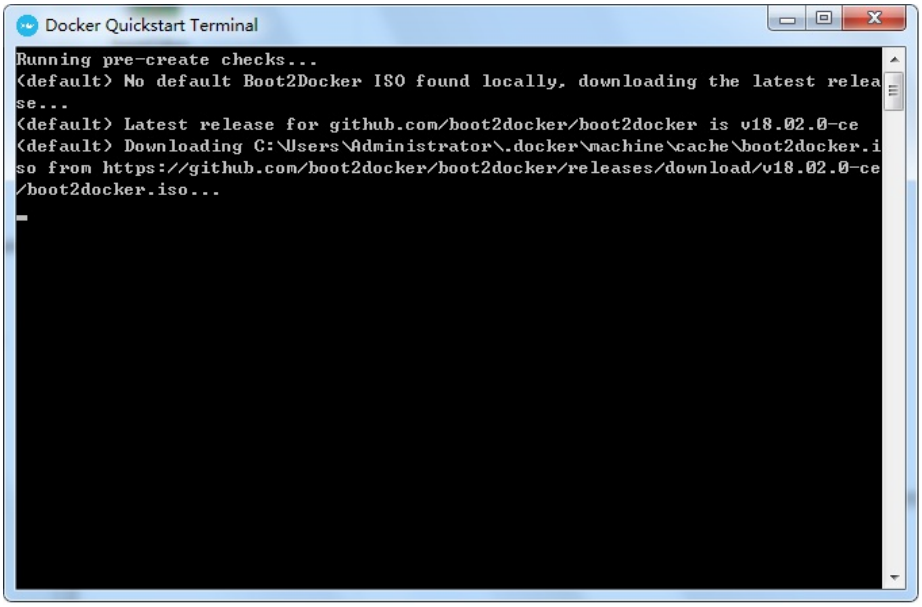
目前支持 Windows 系统的 Docker 工具包括：Docker Toolbox 和 Docker CE。经过测试，我们推荐使用 Docker Toolbox，Docker Toolbox 支持 64 位的 Windows 7、8、10 等系统。首先检查系统是否打开虚拟化支持，用户可以右键点击任务栏，选择“任务管理器”，切换到“性能”页，进行查看。



如果虚拟化功能未启用，需要重新启动计算机进入 BIOS 将虚拟化功能打开。打开 https://docs.docker.com/toolbox/toolbox_install_windows/ 页面，点击相应下载按钮进行下载。因为下载文件尺寸较大，推荐使用下载工具进行下载。下载后运行 DockerToolbox.exe 安装。



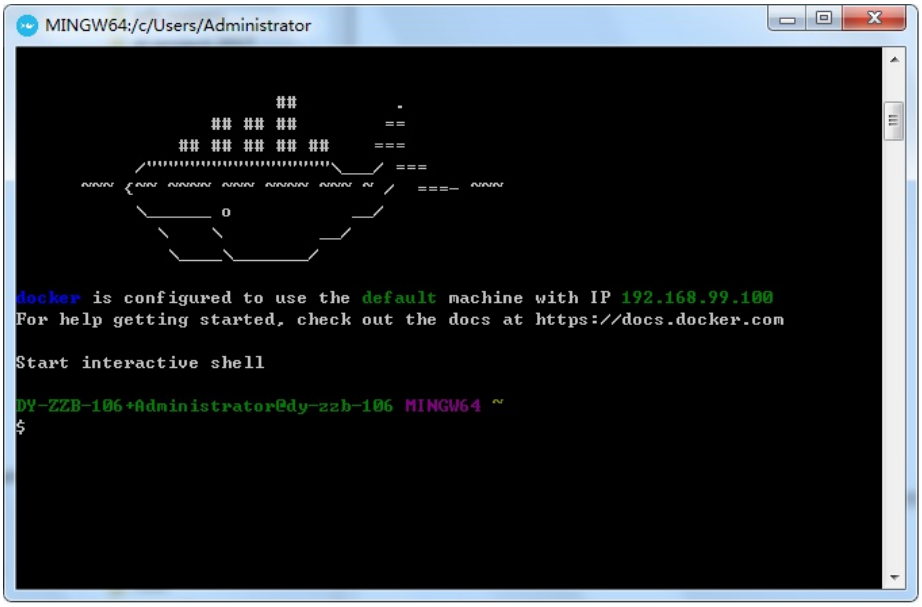
安装过程中接受所有默认选项即可。安装完成后，双击 Docker QuickStart Terminal 启动 Docker 客户端。



在第一次运行 Docker 时，会在线下载最新版本的 Boot2Docker.iso 文件，文件会被下载到 C:\Users\Administrator.docker\machine\cache\ 目标文件夹中，如果当前用户不是 Administrator，就用当前用户名替换路径中的 Administrator，如用户名是 zhangsan，文件夹就是 C:\Users\zhangsan.docker\machine\cache\，在这一步下载耗时较长，有时会超时失败，建议用下载工具下载相应文件，下载后把它复制到目标文件夹中即可。

Docker 命令

运行 Docker 后，初始页面会显示一条载货小鲸鱼的文本图像，这是 Docker 的标志。同时还会显示出为当前机器配置的ip地址，一般是 192.168.99.100 。后面在运行 Tensorflow 镜像时需要使用这个地址。



在 Docker 中运行各种应用都需要使用对应的镜像文件，在 hub.docker.com 网站上可以找到数以万计的镜像文件，我们要使用的 Tensorflow 镜像也可以从该网站下载。我们可以使用 pull 命令从 Docker Hub 网站下载：

```
docker pull tensorflow/tensorflow:latest-py3
```

这条命令将下载 Python 3 版本的最新 Tensorflow（当下是 1.5 版）镜像，tensorflow/tensorflow 是镜像的名字，斜线前的 tensorflow 表示 Docker Hub 用户名，斜线后的 tensorflow 表示资源库名称，冒号后面的 latest-py3，是 Docker 镜像的标签（tag），用以区分镜像的不同的子版本。以 Tensorflow 为例，其镜像支持数十种标签，包括 latest，nightly，latest-gpu，latest-devel-gpu 等等。其中的 latest 是缺省标签，也就是说如果未指定标签的话，就会使用 latest。我们这里没有使用 latest标签，因为 tensorflow/tensorflow:latest 镜像对应 Python 2.7 版，我们要使用的是 Python 3.5 版，对应的是 latest-py3 标签。

pull 命令成功运行后，tensorflow的镜像文件就下载到本地了，我们可以通过 images 命令来查看本地的所有镜像：

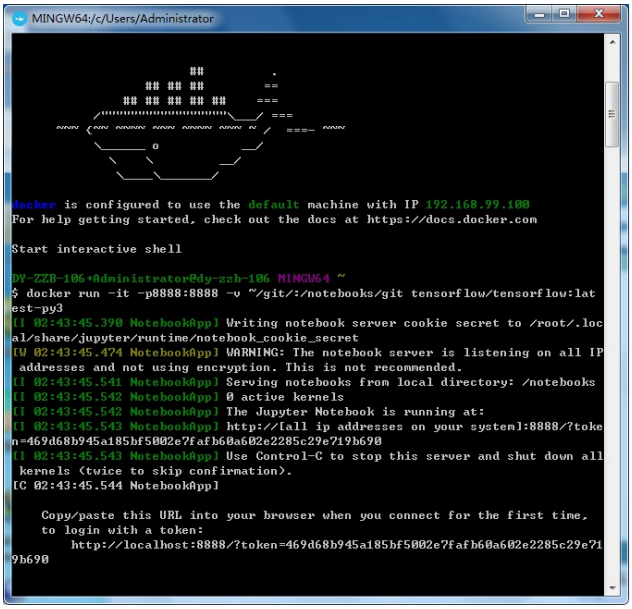
```
docker images
```

在运行Tensorflow镜像之前，我们要准备好要测试的代码资源，我们首先在当前用户目录下新建一个子目录 git，如果用户名是 zhangsan，文件夹就是 C:\Users\zhangsan\git, 将本书配套的代码或者其他练习代码复制到该文件夹中。我们运行镜像时，运行环境和本地环境是隔离的，为了让运行起来的镜像（容器）能够访问本地资源，我们需要进行资源映射。比如 Tensorflow 镜像中封装了 Jupyter 工具，缺省使用 8888 端口提供服务，我们就要使用 -p 命令将本地的 8888 端口映射到容器的 8888 端口。为了让容器能够访问本地 git 目录中的文件，我们使用 -v 命令将本地的 git 目录映射到容器里相应的目录。要运行 Tesnorflow，我们可以使用 run 命令，同时指定端口映射和路径映射：

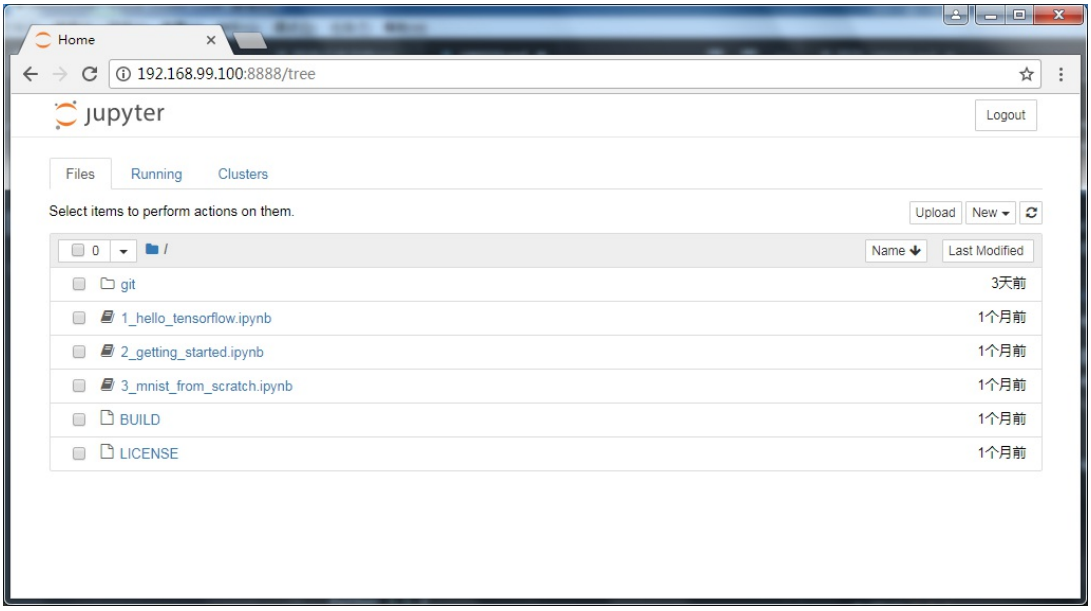
```
docker run -it -p 8888:8888 -v ~/git/:/notebooks/git/
tensorflow/tensorflow:latest-py3
```

在这条 run 命令里， -it 表示要启动一个交互式（interactive）的终端（terminal），-p 表示端口（port）映射，-v 表示路径（volume）映射，映射的格式是：“本地资源:容器资源”。如 ~/git/:/notebooks/git/ 表示把本地当前用户目录（~）下的 git 子目录，映射到容器里根目录（/）下的notebooks/git 子目录。为什么容器目录里要加上notebooks目录呢？如果你打开 tensorflow 容器看一下就明白了，在我们运行的 tensorflow 容器里，事先已经建立了 /notebooks 目录，Jupyter 就是从这个目录启动的，所以 /notebooks 目录就是 Jupyter 的根目录，我们把本地的 git 目录映射到 /notebooks/git，所以打开 Jupyter 页面后就可以看到 git 目录，点击进入就可以测试代码了。

执行完 docker run 命令后，系统会提示我们打开 Jupyter 的服务页面：

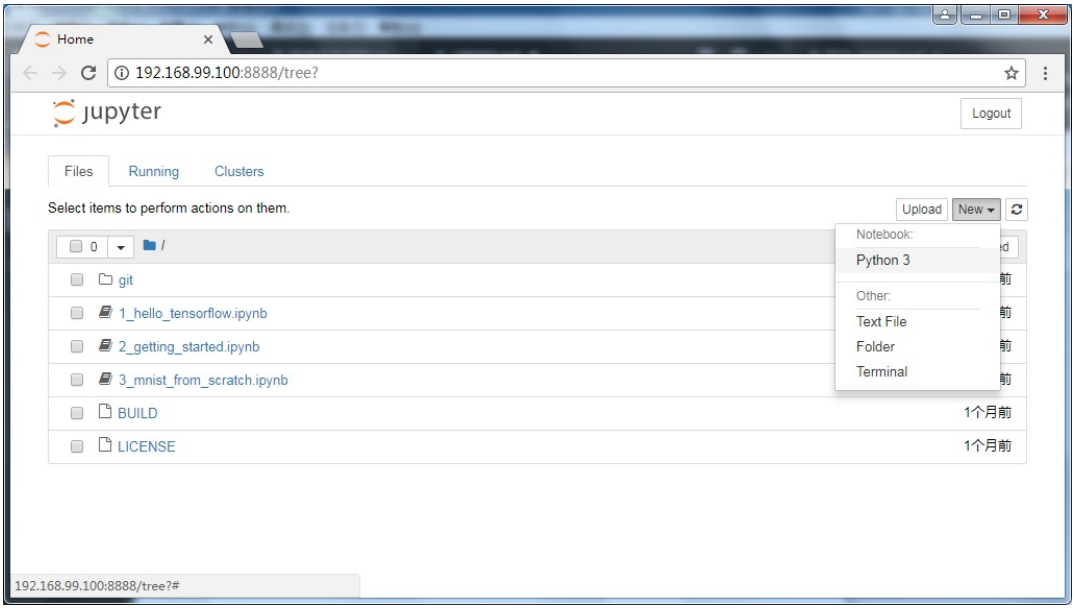


把屏幕最下方的链接地址中的 localhost 换成之前初始页面里面提示的 IP 地址，一般是 192.168.99.100，对应图中的例子，就是：http://192.168.99.100:8888/?token=469d68b945a185bf5002e7fafb60a602e2285c29e719b690 。 用浏览器打开这个链接地址，就会显示出 Jupyter 页面：



另一种打开 Jupyter 的方法是用浏览器打开 http://192.168.99.100:8888/ 这个地址，在显示的 Web 页面里填写相应的 Token，对应图中的例子就是 469d68b945a185bf5002e7fafb60a602e2285c29e719b690 ，也一样会登录进 Jupyter 页面。

到目前为止，我们已经打开 Jupyter 页面，可以进行代码测试了。点击右上角的 New 按钮，选择 Python 3 Notebook，建立一个Python笔记本。



在笔记本的第一个单元格（cell）里输入代码，按 Shift + Enter 组合键执行，也可以点击工具栏里面的 Run 按钮执行。首先检查一下 Python、Tensorflow 和 Numpy 的版本号：

```
! python --version
```

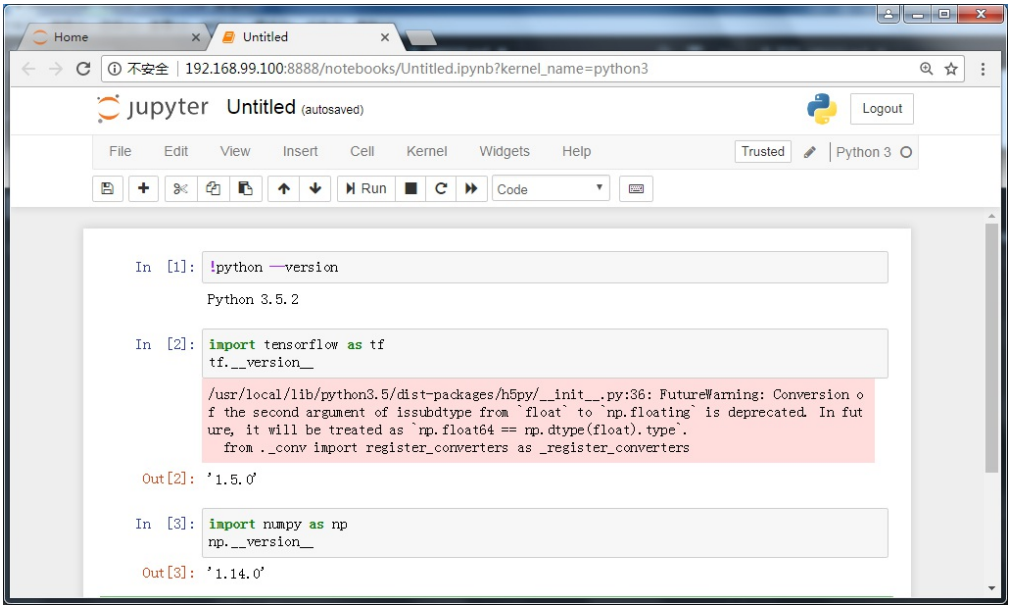
```
Python 3.5.2
```

```
import tensorflow as tf
tf.__version__
```

```
1.5.0
```

```
import numpy as np
np.__version__
```

```
1.14.0
```



Jupyter

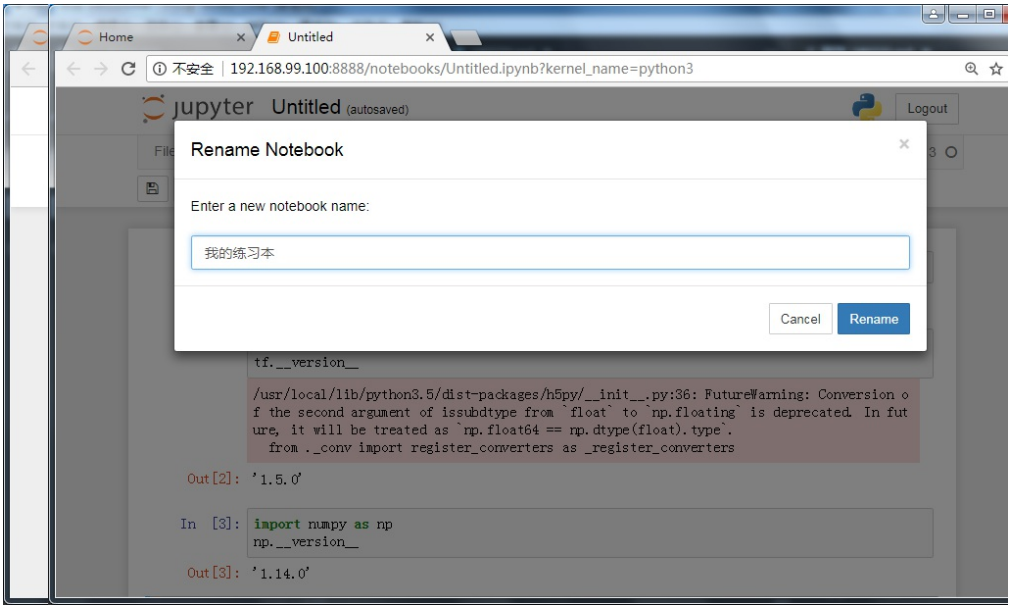
在学习 Tensorflow 之前，有必要熟悉一下我们的工程环境，我们使用 Jupyter notebook 作为我们的练习环境。Jupyter Notebook（之前称为 IPython notebook）是一个交互式笔记本工具，它的核心在于展示与快速迭代，支持运行包括 Python 在内的 40 多种编程语言。在本节中，我们将介绍 Jupyter notebook 的主要特性。在开始使用 notebook 之前，我们先需要安装该库。一般情况下需要 pip install jupyter 就可以了。在我们的 Tensorflow 镜像里面已经集成了 Jupyter 工具，就可以省略安装这一步了。

Jupyter 界面

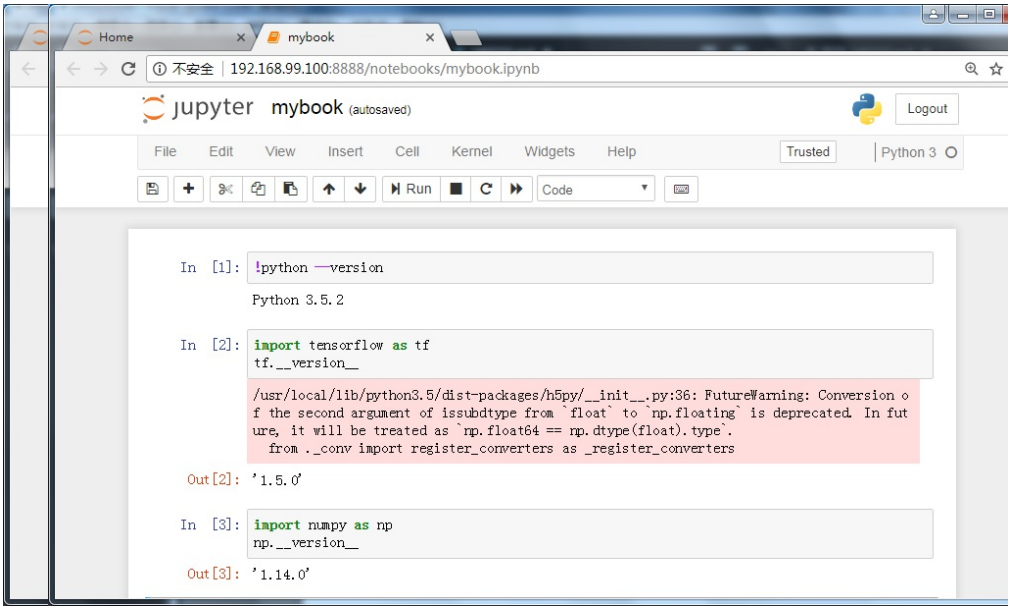
Jupyter notebook 界面由以下部分组成：

- Jupyter 的标志和当前 notebook 的名称（Untitled）；
- 菜单栏，提供了全面的控制选项；
- 工具栏，提供了保存、导出、重载 notebook，以及重启内核等常用选项；
- notebook 编辑区，包含了 notebook 的编辑内容

单击 Untitiled 名称，输入 ‘mybook’，点击 Rename，为自己的笔记本命名：

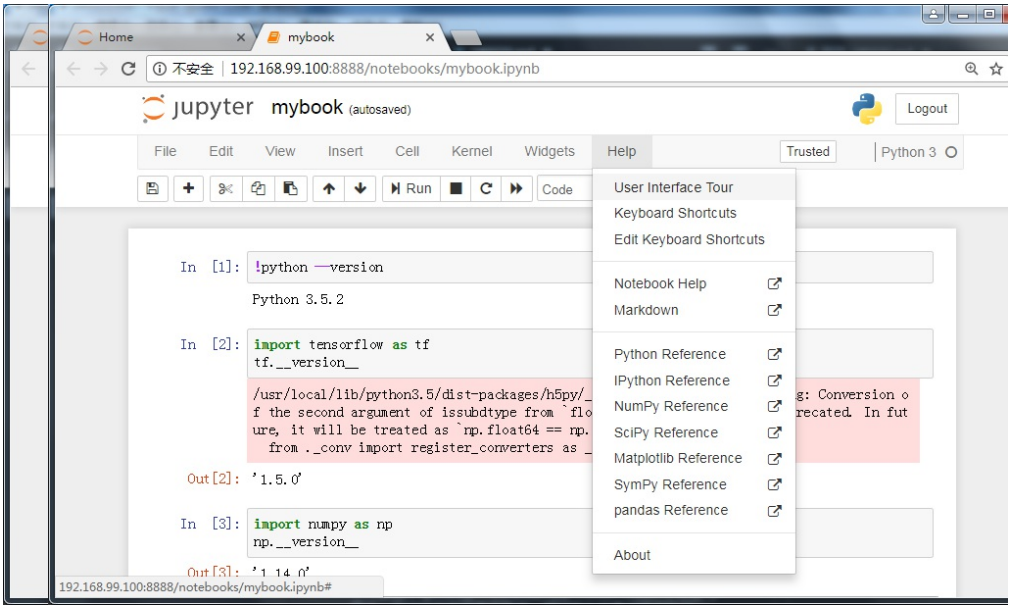


可以看到当前笔记本的名称已经改为'mybook'：



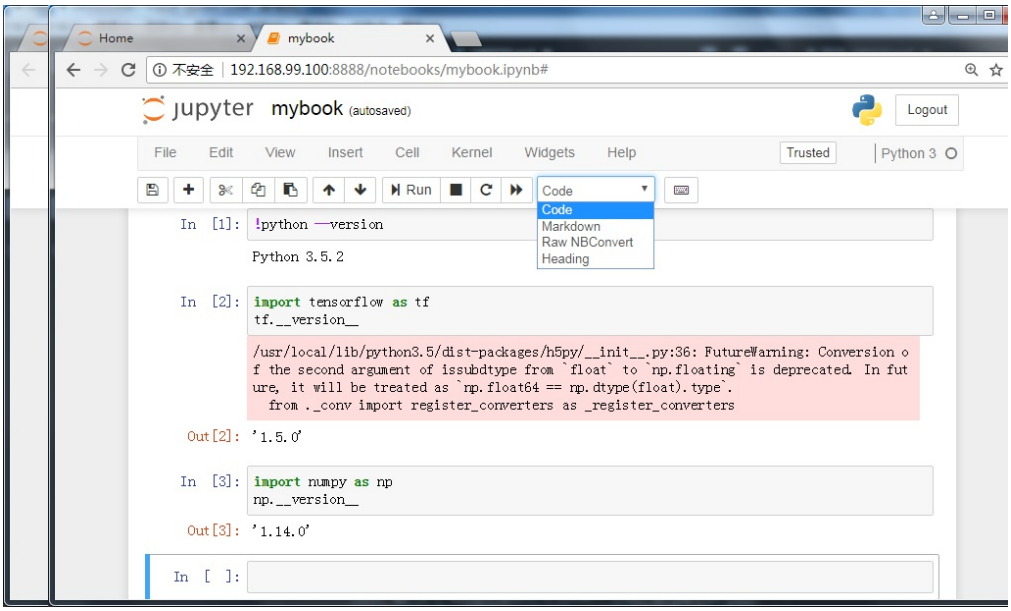
同时，注意到浏览器的地址栏里面，当前文件名称也改变成 mybook.ipynb 。

第一次使用 Jupyter，可以使用菜单栏右侧的帮助菜单，选择 Help -> User Interface Tour（用户界面游览），会显示一系列帮助提示，方便我们了解 Jupyter 界面上各组成部分的功能。

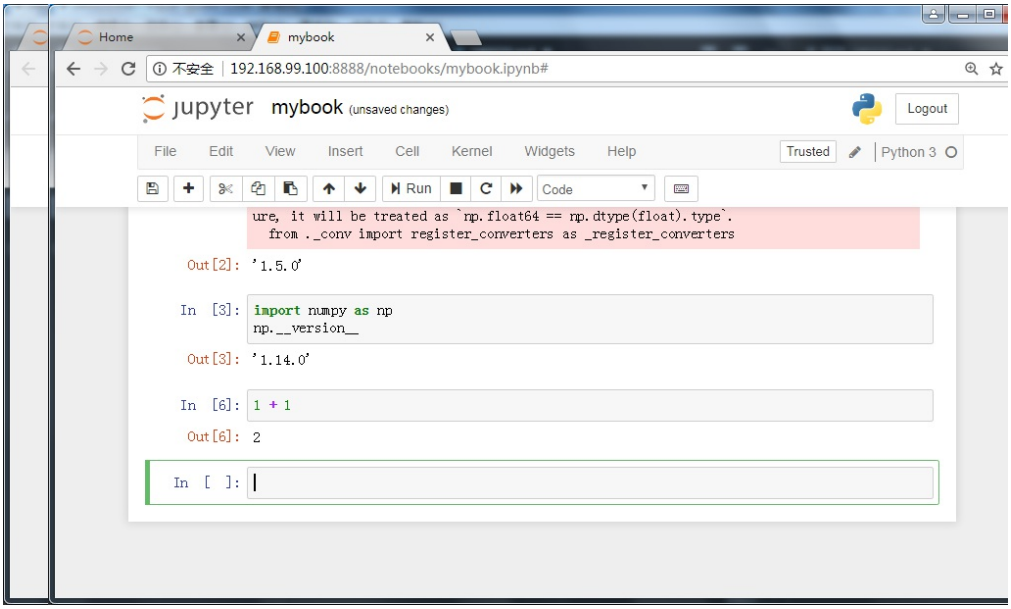


Jupyter 单元格

页面下方的主要区域，由被称为单元格（cell）的部分组成。Notebook 就是由多个单元格构成的，每个单元格又可以有不同的类型。点击工具栏右边的下拉列表框，可以为当前单元格设置不同的类型：



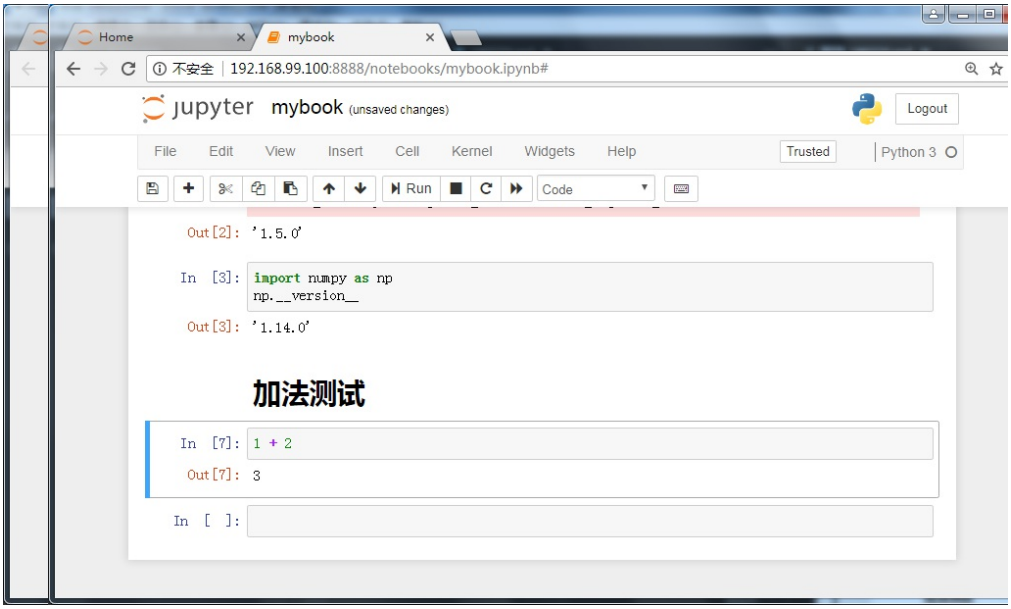
代码单元格（code cell），以 “[ ]” 开头。在这种类型的单元格中，可以输入 Python 代码并执行。例如，输入 1 + 1 并按下 Shift + Enter 之后，单元格中的代码会被计算，光标也会被移动动到下一个新的单元格中：



Jupyter 有一个非常强大的特性，就是可以修改之前的单元格，对其重新计算，这样就可以迭代式的实验，方便的运行多个相关代码。试着把光标移回第一个单元格，并将 1 + 1 修改成 1 + 2，然后按下 Shift + Enter 重新计算该单元格。你会发现结果马上就更新成了 3 。我们也可以重新计算全部单元格，只要选择菜单 Cell -> Run all （执行全部单元格）即可。

Jupyter 的强大功能不止于此，它支持所谓的'文学编程'（Literate programming），除了代码单元格外，我们可以使用标题单元格（Heading）和文本单元格（Markdown）。通过综合运用这些不同类型的单元格，Jupyter 可以一边用文字解释代码，一边运行代码，把整个过程记录成完整的实验报告，这些实验报告可以转换成 html 网页或者 pdf 文档进行输出。

我们在计算单元格的顶部添加一个标题单元格。选中之前的计算单元格，然后点击 Insert -> Insert Cell Above（在上方插入单元格）。在刚刚的计算单元格顶部马上就出现了一个新的单元格。点击工具栏右边的单元格类型下拉列表框，选择 Heading，将其变成一个标题单元格。Jupyter 会提示：将使用 Markdown 语法来定义标题。在单元格中输入 '加法测试'，按下 Shift + Enter :



Tensorflow 的辅助支持库

我们在实验 Tensorflow 代码时，需要处理高维向量，需要进行数据处理，也需要将中间的计算结果用图形呈现出来。下面我们就来了解一下这些辅助支持库的使用。

NumPy

标准 Python 不能直接处理高维向量，需要用循环来处理数组中的每个元素，计算效率非常低。NumPy 弥补了这一不足，NumPy 支持两种高级对象：ndarray（高维向量）和 ufunc（通用函数）。ndarray 是存储单一数据类型的高维数组，ufunc 是一种能对数组的每个元素进行操作的函数，用 C 语言实现，计算速度非常快。

对比一下 Python 和 NumPy 处理方式的不同：

```
import array as ar
a = ar.array('d', [1, 2, 3])
b = ar.array('d', [4, 5, 6])
a + b
```

输出：

```
array('d', [1.0, 2.0, 3.0, 4.0, 5.0, 6.0])
```

```
import numpy as np
a = np.array( [1, 2, 3])
b = np.array( [4, 5, 6])
a + b
```

输出：

```
array([5, 7, 9])
```

可以看到，Python 里面的数组运算并不能对数组的每个元素进行操作，而 NumPy 的数组运算则可以。

Numpy 的主要对象是单一数据类型的高维数组。这是一个所有元素都是同一种类型，并通过正整数元组索引的高维表格。在 Numpy 中维度（dimensions）叫做轴（axes），轴的个数叫做秩（rank）。

[1, 2, 3, 4] 是一个秩为 1 的数组，因为它只有一个轴。轴的长度为 4 。下面的例子中，数组的秩为 2 （它是二维的）。它的第一维度（轴）长度为 2 ，第二维度长度为 3 。

```
a = [[ 1, 2, 3],
     [ 4, 5, 6]]
```

```
print("Rank:", a.ndim)
print("Shape:", a.shape)
print("Axis 0:", a.shape[0])
print("Axis 1:", a.shape[1])
```

输出：

```
Rank: 2
Shape: (2, 3)
Axis 0: 2
Axis 1: 3
```

NumPy 数据

```
import numpy as np

array = np.array([[1,2,3],[2,3,4]])  #列表转化为矩阵
print(array)
```

```
array([[1, 2, 3],
       [2, 3, 4]])
```

```
print('number of dim:',array.ndim)  # 维度
print('shape :',array.shape)    # 行数和列数
print('size:',array.size)   # 元素个数
```

```
number of dim: 2
shape : (2, 3)
size: 6
```

```
a = np.array([2,3,4],dtype=np.int)
print(a.dtype)
```

```
int 64
```

```
a = np.array([2,3,4],dtype=np.float)
print(a.dtype)
```

```
float64
```

```
a = np.zeros((3,4)) # 数据全为0，3行4列
a = np.ones((3,4),dtype = np.int)   # 数据为1，3行4列
a = np.arange(10,20,2) # 10-19 的数据，2步长
a = np.arange(12).reshape((3,4))    # 3行4列，0到11
a = np.linspace(1,10,20)    # 开始端1，结束端10，且分割成20个数据，生成线段
a = np.linspace(1,10,20).reshape((5,4)) # 更改shape
```

```
import numpy as np
a=np.array([10,20,30,40])   # array([10, 20, 30, 40])
b=np.arange(4)              # array([0, 1, 2, 3])
```

NumPy 运算

```
c=a-b  # array([10, 19, 28, 37])
c=a+b   # array([10, 21, 32, 43])
c=a*b   # array([  0,  20,  60, 120])
c=b**2  # array([0, 1, 4, 9])
c=10*np.sin(a)
# array([-5.44021111,  9.12945251, -9.88031624,  7.4511316 ])
print(b<3)
# array([ True,  True,  True, False], dtype=bool)
```

```
a=np.array([[1,1],[0,1]])
b=np.arange(4).reshape((2,2))

print(a)
# array([[1, 1],
#        [0, 1]])

print(b)
# array([[0, 1],
#        [2, 3]])
```

```
c_dot = np.dot(a,b)
# array([[2, 4],
#        [2, 3]])
```

```
import numpy as np
a=np.random.random((2,4))
print(a)
# array([[ 0.94692159,  0.20821798,  0.35339414,  0.2805278 ],
#        [ 0.04836775,  0.04023552,  0.44091941,  0.21665268]])
```

```
np.sum(a)   # 4.4043622002745959
np.min(a)   # 0.23651223533671784
np.max(a)   # 0.90438450240606416
```

```
print("a =",a)
# a = [[ 0.23651224  0.41900661  0.84869417  0.46456022]
# [ 0.60771087  0.9043845   0.36603285  0.55746074]]

print("sum =",np.sum(a,axis=1))
# sum = [ 1.96877324  2.43558896]

print("min =",np.min(a,axis=0))
# min = [ 0.23651224  0.41900661  0.36603285  0.46456022]

print("max =",np.max(a,axis=1))
# max = [ 0.84869417  0.9043845 ]
```

```
import numpy as np
A = np.arange(2,14).reshape((3,4))

# array([[ 2, 3, 4, 5]
#        [ 6, 7, 8, 9]
#        [10,11,12,13]])

print(np.argmin(A))    # 0
print(np.argmax(A))    # 11
```

```
print(np.mean(A))        # 7.5
print(np.average(A))     # 7.5
```

```
print(A.mean())          # 7.5
print(A.median())       # 7.5

print(np.cumsum(A))     # [2 5 9 14 20 27 35 44 54 65 77 90]
print(np.diff(A))

# [[1 1 1]
#  [1 1 1]
#  [1 1 1]]
```

```
import numpy as np
A = np.arange(14,2, -1).reshape((3,4))

# array([[14, 13, 12, 11],
#       [10,  9,  8,  7],
#       [ 6,  5,  4,  3]])

print(np.sort(A))

# array([[11,12,13,14]
#        [ 7, 8, 9,10]
#        [ 3, 4, 5, 6]])
```

```
print(np.transpose(A))
print(A.T)

# array([[14,10, 6]
#        [13, 9, 5]
#        [12, 8, 4]
#        [11, 7, 3]])
# array([[14,10, 6]
#        [13, 9, 5]
#        [12, 8, 4]
#        [11, 7, 3]])
```

```
print(A)
# array([[14,13,12,11]
#        [10, 9, 8, 7]
#        [ 6, 5, 4, 3]])

print(np.clip(A,5,9))
# array([[ 9, 9, 9, 9]
#        [ 9, 9, 8, 7]
#        [ 6, 5, 5, 5]])
```

NumPy 索引

```
import numpy as np
A = np.arange(3,15)

# array([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

print(A[3])    # 6
```

```
A = np.arange(3,15).reshape((3,4))
"""
array([[ 3,  4,  5,  6]
       [ 7,  8,  9, 10]
       [11, 12, 13, 14]])
"""

print(A[2])
# [11 12 13 14]

print(A[1][1])      # 8

print(A[1, 1])      # 8

print(A[1, 1:3])    # [8 9]
```

```
for row in A:
    print(row)
"""
[ 3,  4,  5, 6]
[ 7,  8,  9, 10]
[11, 12, 13, 14]
"""

for column in A.T:
    print(column)
"""
[ 3,  7,  11]
[ 4,  8,  12]
[ 5,  9,  13]
[ 6, 10,  14]
"""
```

```
import numpy as np
A = np.arange(3,15).reshape((3,4))
```

```
print(A.flatten())
# array([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

for item in A.flat:
    print(item)

# 3
# 4
……
# 14
```

NumPy 合并与分割

```
import numpy as np
A = np.array([1,1,1])
B = np.array([2,2,2])

print(np.vstack((A,B)))    # vertical stack
"""
[[1,1,1]
 [2,2,2]]
"""

D = np.hstack((A,B))       # horizontal stack

print(D)
# [1,1,1,2,2,2]
```

```
print(A[np.newaxis,:])
# [[1 1 1]]

print(A[np.newaxis,:].shape)
# (1,3)

print(A[:,np.newaxis])
"""
[[1]
[1]
[1]]
"""

print(A[:,np.newaxis].shape)
# (3,1)
```

```
import numpy as np
A = np.array([1,1,1])[:,np.newaxis]
B = np.array([2,2,2])[:,np.newaxis]

C = np.vstack((A,B))   # vertical stack
D = np.hstack((A,B))   # horizontal stack

print(D)
"""
[[1 2]
[1 2]
[1 2]]
"""

print(A.shape,D.shape)
# (3,1) (3,2)
```

```
C = np.concatenate((A,B,B,A),axis=0)

print(C)
"""
array([[1],
       [1],
       [1],
       [2],
       [2],
       [2],
       [2],
       [2],
       [2],
       [1],
       [1],
       [1]])
"""

D = np.concatenate((A,B,B,A),axis=1)

print(D)
"""
array([[1, 2, 2, 1],
       [1, 2, 2, 1],
       [1, 2, 2, 1]])
"""
```

Pandas

NumPy 是列表式的，而 Pandas 是字典式的。Pandas 是基于 Numpy 构建的，让数据处理操作变得更加简单。

要使用pandas，首先需要了解他主要两个数据结构：Series和DataFrame。

Pandas 对象

```
import pandas as pd
import numpy as np
s = pd.Series([1,3,6,np.nan,44,1])

print(s)
"""
0     1.0
1     3.0
2     6.0
3     NaN
4    44.0
5     1.0
dtype: float64
"""
```

```
dates = pd.date_range('20160101',periods=6)
df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=['a','b','c','d'])

print(df)
"""
                   a         b         c         d
2016-01-01 -0.253065 -2.071051 -0.640515  0.613663
2016-01-02 -1.147178  1.532470  0.989255 -0.499761
2016-01-03  1.221656 -2.390171  1.862914  0.778070
2016-01-04  1.473877 -0.046419  0.610046  0.204672
2016-01-05 -1.584752 -0.700592  1.487264 -1.778293
2016-01-06  0.633675 -1.414157 -0.277066 -0.442545
"""
```

```
print(df['b'])

"""
2016-01-01   -2.071051
2016-01-02    1.532470
2016-01-03   -2.390171
2016-01-04   -0.046419
2016-01-05   -0.700592
2016-01-06   -1.414157
Freq: D, Name: b, dtype: float64
"""
```

```
df1 = pd.DataFrame(np.arange(12).reshape((3,4)))
print(df1)

"""
   0  1   2   3
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
"""
```

```
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'})

print(df2)

"""
     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
"""
```

```
print(df2.dtypes)

"""
df2.dtypes
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
"""
```

```
print(df2.index)

# Int64Index([0, 1, 2, 3], dtype='int64')
```

```
print(df2.columns)

# Index(['A', 'B', 'C', 'D', 'E', 'F'], dtype='object')

print(df2.values)

"""
array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo']],
dtype=object)
"""
```

```
df2.describe()

"""
         A    C    D
count  4.0  4.0  4.0
mean   1.0  1.0  3.0
std    0.0  0.0  0.0
min    1.0  1.0  3.0
25%    1.0  1.0  3.0
50%    1.0  1.0  3.0
75%    1.0  1.0  3.0
max    1.0  1.0  3.0
"""
```

Pandas 选择数据

```
dates = pd.date_range('20130101', periods=6)
df = pd.DataFrame(np.arange(24).reshape((6,4)),index=dates, columns=
['A','B','C','D'])

"""
             A   B   C   D
2013-01-01   0   1   2   3
2013-01-02   4   5   6   7
2013-01-03   8   9  10  11
2013-01-04  12  13  14  15
2013-01-05  16  17  18  19
2013-01-06  20  21  22  23
"""
```

```
print(df['A'])
```

```
print(df.A)

"""
2013-01-01     0
2013-01-02     4
2013-01-03     8
2013-01-04    12
2013-01-05    16
2013-01-06    20
Freq: D, Name: A, dtype: int64
"""
```

```
print(df.loc['20130102'])
"""
A    4
B    5
C    6
D    7
Name: 2013-01-02 00:00:00, dtype: int64
"""

print(df.loc[:,['A','B']])
"""
             A   B
2013-01-01   0   1
2013-01-02   4   5
2013-01-03   8   9
2013-01-04  12  13
2013-01-05  16  17
2013-01-06  20  21
"""

print(df.loc['20130102',['A','B']])
"""
A    4
B    5
Name: 2013-01-02 00:00:00, dtype: int64
"""
```

```
print(df.iloc[3,1])
# 13

print(df.iloc[3:5,1:3])
"""
             B   C
2013-01-04  13  14
2013-01-05  17  18
```

```
"""

print(df.iloc[[1,3,5],1:3])
"""
             B   C
2013-01-02   5   6
2013-01-04  13  14
2013-01-06  21  22

"""
```

```
print(df.ix[:3,['A','C']])
"""
            A   C
2013-01-01  0   2
2013-01-02  4   6
2013-01-03  8  10
"""
```

```
print(df[df.A>8])
"""
             A   B   C   D
2013-01-04  12  13  14  15
2013-01-05  16  17  18  19
2013-01-06  20  21  22  23
"""
```

```
import pandas as pd
import numpy as np
```

```
ds = pd.read_csv('./data/bmi&life.csv')
ds.head(5)
```

```
ds.info()
```

```
ds.shape
```

```
ds.columns
```

```
ds.index
```

Matplotlib