[Anaconda](https://anaconda.org/" \t "_blank) 是一个包含数据科学常用包的 Python 发行版本。它基于 conda ——一个包和环境管理器——衍生而来。你将使用 conda 创建环境，以便分隔使用不同 Python 版本和不同程序包的项目。你还将使用它在环境中安装、卸载和更新包。通过使用 Anaconda，处理数据的过程将更加愉快。

包管理器用于在计算机上安装库和其他软件。你可能已经熟悉 pip，它是 Python 库的默认包管理器。conda 与 pip 相似，不同之处是可用的包以数据科学包为主，而 pip 适合一般用途。与此同时，conda 并非像 pip 那样专门适用于 Python，它也可以安装非 Python 的包。它是支持任何软件的包管理器。也就是说，虽然并非所有的 Python 库都能通过 Anaconda 发行版和 conda 获得，但同时它也支持非 Python 库的获得。在使用 conda 的同时，你仍可以使用 pip 来安装包。

除了管理包之外，conda 还是虚拟环境管理器。它类似于另外两个很流行的环境管理器，即 [virtualenv](https://virtualenv.pypa.io/en/stable/" \t "_blank) 和 [pyenv](https://github.com/yyuu/pyenv" \t "_blank)。

你可以使用conda环境管理器分隔不同项目的包。你常常要使用依赖于某个库的不同版本的代码。例如，你的代码可能使用了 Numpy 中的新功能，或者使用了已删除的旧功能。实际上，不可能同时安装两个 Numpy 版本。你要做的应该是，为每个 Numpy 版本创建一个环境，然后在项目的对应环境中工作。

你也可以将环境中的包列表导出为文件，然后将该文件与代码打包在一起。这能让其他人轻松加载代码的所有依赖项。pip 提供了类似的功能，即 pip freeze > requirements.txt。

可以在终端或命令提示符中键入 conda list，以查看你安装的内容。

在 Windows 上，会随 Anaconda 一起安装一批应用程序：

* Anaconda Navigator，它是用于管理环境和包的 GUI
* Anaconda Prompt 终端，它可让你使用命令行界面来管理环境和包
* Spyder，它是面向科学开发的 IDE

为了避免报错，我推荐在默认环境下更新所有的包。打开 Anaconda Prompt （或者 Mac 下的终端），键入：

**conda** upgrade --all

安装包

 conda install package\_name

你还可以同时安装多个包。类似 conda install numpy scipy pandas 的命令会同时安装所有这些包。还可以通过添加版本号（例如 conda install numpy=1.10）来指定所需的包版本。

Conda 还会自动为你安装依赖项。例如，scipy 依赖于 numpy，因为它使用并需要 numpy。如果你只安装 scipy (conda install scipy)，则 conda 还会安装 numpy（如果尚未安装的话）。

卸载包

conda remove package\_name

要更新包，请使用 conda update package\_name。如果想更新环境中的所有包（这样做常常很有用），请使用 conda update --all。最后，要列出已安装的包，请使用前面提过的 conda list。

如果不知道要找的包的确切名称，可以尝试使用 conda search search\_term 进行搜索。例如，我知道我想安装 [Beautiful Soup](https://www.crummy.com/software/BeautifulSoup/)，但我不清楚确切的包名称。因此，我尝试执行 conda search beautifulsoup。

管理环境：

如前所述，你可以使用 conda 创建环境以隔离项目。要创建环境，请在终端中使用 conda create -n env\_name list of packages。在这里，-n env\_name 设置环境的名称（-n是指名称），而 list of packages 是要安装在环境中的包的列表。例如，要创建名为 my\_env 的环境并在其中安装 numpy，请键入 conda create -n my\_env numpy。

创建环境时，可以指定要安装在环境中的 Python 版本。这在你同时使用 Python 2.x 和 Python 3.x 中的代码时很有用。要创建具有特定 Python 版本的环境，请键入类似于 conda create -n py3 python=3 或 conda create -n py2 python=2 的命令。实际上，我在我的个人计算机上创建了这两个环境。我将它们用作与任何特定项目均无关的通用环境，以处理普通的工作（可轻松使用每个 Python 版本）。这些命令将分别安装 Python 3 和 Python 2 的最新版本。要安装特定版本（例如 Python 3.3），请使用 conda create -n py python=3.3。

进入环境：

创建了环境后，在 OSX/Linux 上使用 source activate my\_env 进入环境。在 Windows 上，请使用 activate my\_env。

要离开环境，请键入 source deactivate（在 OSX/Linux 上）。在 Windows 上，请使用 deactivate。

保存和加载环境：

共享环境这项功能确实很有用，它能让其他人安装你的代码中使用的所有包，并确保这些包的版本正确。你可以使用 conda env export > environment.yaml 将包保存为 [YAML](http://www.yaml.org/)。命令的第一部分 conda env export 用于输出环境中的所有包的名称（包括 Python 版本）。

上图中，你可以看到环境的名称和所有依赖项及其版本。导出命令的第二部分 > environment.yaml 将导出的文本写入到 YAML 文件 environment.yaml 中。现在可以共享此文件，而且其他人能够用于创建和你项目相同的环境。

要通过环境文件创建环境，请使用 conda env create -f environment.yaml。这会创建一个新环境，而且它具有同样的在 environment.yaml 中列出的库。

**列出环境**

如果忘记了环境的名称（我有时会这样），可以使用 conda env list 列出你创建的所有环境。你会看到环境的列表，而且你当前所在环境的旁边会有一个星号。默认的环境（即当你不在选定环境中时使用的环境）名为 root。

**删除环境**

如果你不再使用某些环境，可以使用 conda env remove -n env\_name 删除指定的环境（在这里名为 env\_name）。

**共享环境**

在 GitHub 上共享代码时，最好同样创建环境文件并将其包括在代码库中。这能让其他人更轻松地安装你的代码的所有依赖项。对于不使用 conda 的用户，我通常还会使用 pip freeze（[在此处了解详情](https://pip.pypa.io/en/stable/reference/pip_freeze/)）将一个 pip requirements.txt 文件导出并包括在其中。

<https://pip.pypa.io/en/stable/reference/pip_freeze/>

## [Examples](https://pip.pypa.io/en/stable/reference/pip_freeze/#id6)

1. Generate output suitable for a requirements file.
2. $ pip freeze
3. docutils==0.11
4. Jinja2==2.7.2
5. MarkupSafe==0.19
6. Pygments==1.6
7. Sphinx==1.2.2
8. Generate a requirements file and then install from it in another environment.
9. $ env1/bin/pip freeze > requirements.txt

$ env2/bin/pip install -r requirements.txt

sudo pip install --upgrade keras