

< [更多资源](#) | [目录](#) | [简单的折线图](#) >

 [Open in Colab](#)

# Visualization with Matplotlib

## 使用Matplotlib进行可视化

We'll now take an in-depth look at the Matplotlib package for visualization in Python. Matplotlib is a multi-platform data visualization library built on NumPy arrays, and designed to work with the broader SciPy stack. It was conceived by John Hunter in 2002, originally as a patch to IPython for enabling interactive MATLAB-style plotting via gnuplot from the IPython command line. IPython's creator, Fernando Perez, was at the time scrambling to finish his PhD, and let John know he wouldn't have time to review the patch for several months. John took this as a cue to set out on his own, and the Matplotlib package was born, with version 0.1 released in 2003. It received an early boost when it was adopted as the plotting package of choice of the Space Telescope Science Institute (the folks behind the Hubble Telescope), which financially supported Matplotlib's development and greatly expanded its capabilities.

本章中我们会介绍在Python中使用Matplotlib包进行可视化的知识。Matplotlib是一个基于NumPy数组构建的多平台数据可视化程序库，在SciPy技术栈中被广泛使用。Matplotlib是John Hunter在2002年开始构思，最早时候是作为IPython的一个补充，通过gnuplot用来在IPython命令行中实现MATLAB风格的交互式的图表展示。IPython的作者Fernando Perez那时正忙于完成他的博士学位，John意识到他可能在几个月内都无法抽出时间来检查这个补丁的代码。于是John决定将这个构思实现成独立的软件包，于是MatPlotlib诞生了，0.1版本发布于2003年。最早时候在空间望远镜研究院（也就是哈勃望远镜背后的工作小组）得到应用来绘图，从中获得了很大的推动力，研究院从经济上支持Matplotlib项目的发展并极大的扩展了其功能。

One of Matplotlib's most important features is its ability to play well with many operating systems and graphics backends. Matplotlib supports dozens of backends and output types, which means you can count on it to work regardless of which operating system you are using or which output format you wish. This cross-platform, everything-to-everyone approach has been one of the great strengths of Matplotlib. It has led to a large user base, which in turn has led to an active developer base and Matplotlib's powerful tools and ubiquity within the scientific Python world.

Matplotlib最重要的特性之一就是它对很多操作系统以及后端图形引擎的广泛支持。Matplotlib能在大量的图形引擎上工作并输出多种不同的格式，这意味着你可以认为无论使用那种操作系统输出何种格式，它都能良好工作。这种特性为Matplotlib带来了大量的用户基础，从而也吸引了活跃的开发社区，使其发展称为在整个科学Python社区中无处不在的强大绘图工具。

In recent years, however, the interface and style of Matplotlib have begun to show their age. Newer tools like ggplot and ggvis in the R language, along with web visualization toolkits based on D3js and HTML5 canvases, often make Matplotlib feel clunky and old-fashioned. Still, I'm of the opinion that we cannot ignore Matplotlib's strength as a well-tested, cross-platform graphics engine. Recent Matplotlib versions make it relatively easy to set new global plotting styles (see [Customizing Matplotlib: Configurations and Style Sheets](#)), and people have been developing new packages that build on its powerful internals to drive Matplotlib via cleaner, more modern APIs—for example, Seaborn (discussed in [Visualization With Seaborn](#)), ggpy, [HoloViews](#), [Altair](#), and even Pandas itself can be used as wrappers around Matplotlib's API. Even with wrappers like these, it is still often useful to dive into Matplotlib's syntax to adjust the final plot output. For this reason, I believe that Matplotlib itself will remain a vital piece of the data visualization stack, even if new tools mean the community gradually moves away from using the Matplotlib API directly.

然而最近几年，Matplotlib显得有点过时了。R语言中的ggplot和ggvis这些新工具广泛应用了类似D3js和HTML5画布这样的Web技术，让Matplotlib显得相形见绌。近来的Matplotlib版本将设置新的图表风格变得相对简单了一些（参见[自定义matplotlib：配置和样式单](#)），而且开发者在实现MATLAB风格的基础上开发了很多新的包，使得可视化过程能够通过更清晰和现代的API来实现，例如Seaborn（参见[使用Seaborn进行可视化](#)）、[ggpy](#)、[HoloViews](#)和[Altair](#)，而且Pandas本身也提供了对Matplotlib的API封装。但是即使使用封装后的API，深入研究Matplotlib的语法对于更精确的调整图表的输出也是非常有帮助的。正因为如此，作者深信Matplotlib仍然会在数据可视化技术栈中占有不可或缺的地位，即使近期，社区已经逐步不再直接调用Matplotlib的API的情况下。

## General Matplotlib Tips

### 通用提示

Before we dive into the details of creating visualizations with Matplotlib, there are a few useful things you should know about using the package.

在我们开始深入介绍使用Matplotlib进行可视化之前，有一些使用该软件包的基本知识需要了解。

### Importing Matplotlib

#### 载入Matplotlib

Just as we use the `np` shorthand for NumPy and the `pd` shorthand for Pandas, we will use some standard shorthands for Matplotlib imports:

NumPy的载入使用惯例别名 `np`，Pandas的载入使用惯例别名 `pd`，下面是载入Matplotlib的惯例别名：

```
In [1]: import matplotlib as mpl
import matplotlib.pyplot as plt
```

The `plt` interface is what we will use most often, as we shall see throughout this chapter.

`plt` 是我们最常用的模块，本章中我们会一直看到它。

### Setting Styles

#### 设置风格

We will use the `plt.style` directive to choose appropriate aesthetic styles for our figures. Here we will set the `classic` style, which ensures that the plots we create use the classic Matplotlib style:

使用 `plt.style` 属性用来给我们的图表设置视觉的风格。下面我们设置使用 `classic` 风格，这让我们之后的图表都会保持使用经典Matplotlib风格：

```
In [2]: plt.style.use('classic')
```

Throughout this section, we will adjust this style as needed. Note that the stylesheets used here are supported as of Matplotlib version 1.5; if you are using an earlier version of Matplotlib, only the default style is available. For more information on stylesheets, see [Customizing Matplotlib: Configurations and Style Sheets](#).

在本章中，我们会根据需要进行调整风格设置。这里要说明的是，只有Matplotlib 1.5及之后的版本支持风格设置；如果你在使用更早期的版本，那么Matplotlib只能使用默认风格。要获取关于样式单的更多内容，参见[自定义matplotlib：配置和样式单](#)。

### show() or No show() ? How to Display Your Plots

#### show() 或是不需要 show() ？ 如何显示你的图表

A visualization you can't see won't be of much use, but just how you view your Matplotlib plots depends on the context. The best use of Matplotlib differs depending on how you are using it: roughly, the three applicable contexts are using Matplotlib in a script, in an IPython terminal, or in an IPython notebook.

一张你看不到的图表不会有什么用处，但是显示图表的方法根据使用环境会有所不同。Matplotlib的最佳实践取决于你在什么环境中使用它；通常有三种应用场景，在脚本文件中使用，在IPython终端中使用以及在IPython notebook中使用。

#### Plotting from a script

在脚本文件中作图

If you are using Matplotlib from within a script, the function `plt.show()` is your friend. `plt.show()` starts an event loop, looks for all currently active figure objects, and opens one or more interactive windows that display your figure or figures.

如果你是在脚本文件中使用Matplotlib，`plt.show()` 是你显示图表的函数。`plt.show()` 会启动一个事件循环，找到所有激活的图表对象，然后打开一个或多个交互的窗口来显示你的图表。

So, for example, you may have a file called *myplot.py* containing the following:

因此，假设你有一个*myplot.py*文件包含以下代码：

```
# ----- file: myplot.py -----
import matplotlib.pyplot as plt
import numpy as np

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

plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))

plt.show()
```

You can then run this script from the command-line prompt, which will result in a window opening with your figure displayed:

你可以在命令行中运行这个脚本文件，运行结果会打开一个窗口里面显示你的图表：

```
$ python myplot.py
```

The `plt.show()` command does a lot under the hood, as it must interact with your system's interactive graphical backend. The details of this operation can vary greatly from system to system and even installation to installation, but matplotlib does its best to hide all these details from you.

`plt.show()` 函数在底层做了许多工作，因为它需要和你系统的交互式图形引擎通信。这个操作的细节根据系统不同甚至不同安装方式会有区别，Matplotlib尽最大可能为用户屏蔽了这些底层实现细节。

One thing to be aware of: the `plt.show()` command should be used *only once* per Python session, and is most often seen at the very end of the script. Multiple `show()` commands can lead to unpredictable backend-dependent behavior, and should mostly be avoided.

还要提醒一下：`plt.show()` 函数在每个Python会话中*仅能使用一次*，最常见的情况就是在脚本的末尾使用它。多次调用 `show()` 函数会导致不可预料的结果，应该避免。

#### Plotting from an IPython shell

在IPython终端中作图

It can be very convenient to use Matplotlib interactively within an IPython shell (see [IPython: Beyond Normal Python](#)). IPython is built to work well with Matplotlib if you specify Matplotlib mode. To enable this mode, you can use the `%matplotlib` magic command after starting `ipython` :

在IPython终端（参见[IPython：超越Python解释器](#)）中交互式使用Matplotlib是非常方便的。IPython内建有支持Matplotlib的模式。要激活这个模式，你只需要在IPython终端输入 `%matplotlib` 魔术指令即可：

```
In [1]: %matplotlib
Using matplotlib backend: TkAgg

In [2]: import matplotlib.pyplot as plt
```

At this point, any `plt` plot command will cause a figure window to open, and further commands can be run to update the plot. Some changes (such as modifying properties of lines that are already drawn) will not draw automatically: to force an update, use `plt.draw()`. Using `plt.show()` in Matplotlib mode is not required.

这之后任何 `plt` 的作图命令都会打开一个窗口包含作出的图表，后续运行的命令还能更新图表。某些改变（例如修改已经画好的线条的属性）不会自动更新，这时可以使用 `plt.draw()` 来强制更新窗口。在Matplotlib模式下是不需要使用 `plt.show()` 的。

#### Plotting from an IPython notebook

在IPython notebook中作图

The IPython notebook is a browser-based interactive data analysis tool that can combine narrative, code, graphics, HTML elements, and much more into a single executable document (see [IPython: Beyond Normal Python](#)).

IPython notebook是一个基于浏览器的交互式开发工具，能将说明、代码、图像、HTML和其他内容都合成在一个可执行文档中（参见[IPython：超越Python解释器](#)）。

Plotting interactively within an IPython notebook can be done with the `%matplotlib` command, and works in a similar way to the IPython shell. In the IPython notebook, you also have the option of embedding graphics directly in the notebook, with two possible options:

- `%matplotlib notebook` will lead to *interactive* plots embedded within the notebook
- `%matplotlib inline` will lead to *static* images of your plot embedded in the notebook

在IPython notebook中交互式的作图也可以使用 `%matplotlib` 魔术指令，其工作方式类似于在IPython终端中一样。而且在IPython notebook中，你还可以通过指定该魔术指令的参数让作出的图直接在内联在notebook中显示。两个参数可以指定不同的作图模式：

- `%matplotlib notebook`：在notebook中作出具有交互控制功能的内联图表
- `%matplotlib inline`：在notebook中作出静态内联图表

For this book, we will generally opt for `%matplotlib inline` :

本书中，我们通常使用 `%matplotlib inline`：


```
In [3]: %matplotlib inline
```

After running this command (it needs to be done only once per kernel/session), any cell within the notebook that creates a plot will embed a PNG image of the resulting graphic:

运行了这条魔术指令后（只需要在每个jupyter内核中运行一次即可），后续notebook中任何创建图表的代码都会输出一个内嵌的PNG图像，作出图表：

```
In [4]: import numpy as np
x = np.linspace(0, 10, 100)

fig = plt.figure()
plt.plot(x, np.sin(x), '-b')
plt.plot(x, np.cos(x), '--g');
```



## Saving Figures to File

### 将图表保存到文件

One nice feature of Matplotlib is the ability to save figures in a wide variety of formats. Saving a figure can be done using the `savefig()` command. For example, to save the previous figure as a PNG file, you can run this:

Matplotlib还有一个非常棒的功能，那就是将图表保存成很多种不同的文件格式。保存图表可以通过 `savefig()` 函数实现。例如，如果我们需要将上面的图表保存成一个PNG文件，只需要执行下面的代码：

```
In [5]: fig.savefig('my_figure.png')
```

We now have a file called `my_figure.png` in the current working directory:

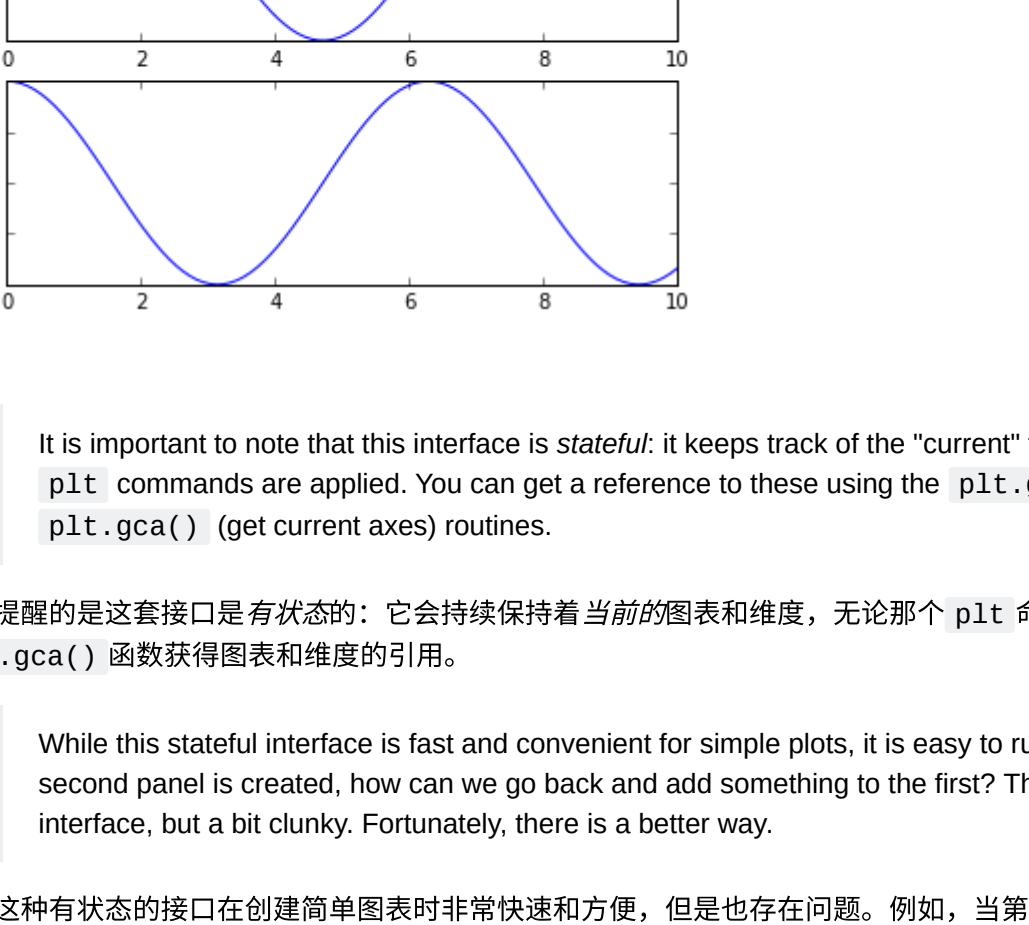
然后在当前工作目录下就可以看到这个文件：

```
In [6]: !ls -lh my_figure.png
-rw-rw-r-- 1 wangy wangy 26K 12月 6 15:47 my_figure.png
```

To confirm that it contains what we think it contains, let's use the IPython `Image` object to display the contents of this file:

我们可以使用IPython的 `Image` 对象将这个图像文件显示出来，验证一下保存的文件是否和图表一致：

```
In [7]: from IPython.display import Image
Out[7]: Image('my_figure.png')
```



In `savefig()`, the file format is inferred from the extension of the given filename. Depending on what backends you have installed, many different file formats are available. The list of supported file types can be found for your system by using the following method of the figure canvas object:

在 `savefig()` 函数中，保存文件的格式是通过文件的扩展名决定的。根据系统的图形引擎不同，支持的文件格式略有不同。你可以通过下面的代码得到系统支持的所有文件格式：

```
In [8]: fig.canvas.get_supported_filetypes()
Out[8]: {'ps': 'Postscript',
'eps': 'Encapsulated Postscript',
'pdf': 'Portable Document Format',
'pgf': 'PGF code for LaTeX',
'png': 'Portable Network Graphics',
'raw': 'Raw RGBA bitmap',
'rgba': 'Raw RGBA bitmap',
'svg': 'Scalable Vector Graphics',
'svgz': 'Scalable Vector Graphics',
'jpg': 'Joint Photographic Experts Group',
'tif': 'Joint Photographic Experts Group',
'tiff': 'Tagged Image File Format',
'tifff': 'Tagged Image File Format'}
```

Note that when saving your figure, it's not necessary to use `plt.show()` or related commands discussed earlier.

注意当你保存图表的时候，是不需要使用 `plt.show()` 或类似的显示图表命令的。

## Two Interfaces for the Price of One

### 两套不同接口

A potentially confusing feature of Matplotlib is its dual interfaces: a convenient MATLAB-style state-based interface, and a more powerful object-oriented interface. We'll quickly highlight the differences between the two here.

Matplotlib一个很令人迷惑的地方是它具有两套接口：一套是很方便的MATLAB风格的接口，还有一套是更强大的面向对象的接口。我们在这里简单的介绍一下它们的区别。

#### MATLAB-style Interface

##### MATLAB 风格接口

Matplotlib was originally written as a Python alternative for MATLAB users, and much of its syntax reflects that fact. The MATLAB-style tools are contained in the pyplot (`plt`) interface. For example, the following code will probably look quite familiar to MATLAB users:

Matplotlib最早是用来为MATLAB用户提供一套Python环境下的替代品，因此很多的语法反映了这一点。MATLAB风格的接口（函数）都封装在pyplot (`plt`) 模块中。例如，下面的代码对于MATLAB用户来说不会陌生：

```
In [9]: plt.figure() # 创建图表
```

# 创建上面第一行的子图表，并设置x, y轴的数据  
plt.subplot(2, 1, 1) # (行、列、子图表序号)  
plt.plot(x, np.sin(x))

# 创建下面第二行的子图表，并设置x, y轴的数据  
plt.subplot(2, 1, 2)  
plt.plot(x, np.cos(x));



It is important to note that this interface is *stateful*: it keeps track of the "current" figure and axes, which are where all `plt` commands are applied. You can get a reference to these using the `plt.gcf()` (get current figure) and `plt.gca()` (get current axes) routines.

需要提醒的是这套接口是有状态的：它会持续保持着当前的图表和维度，无论那个 `plt` 命令改变了它。你可以通过 `plt.gcf()` 和 `plt.gca()` 函数获得图表和维度的引用。

While this stateful interface is fast and convenient for simple plots, it is easy to run into problems. For example, once the second panel is created, how can we go back and add something to the first? This is possible within the MATLAB-style interface, but a bit clunky. Fortunately, there is a better way.

虽然这种有状态的接口在创建简单图表时非常快速和方便，但是也存在问题。例如，当第二个子图表创建了之后，我们如何回去在第一个子图表中增加内容呢？虽然在MATLAB风格接口中也可以实现，但是非常别扭。幸运的是，我们有更好的办法。

#### Object-oriented interface

##### 面向对象接口

The object-oriented interface is available for these more complicated situations, and for when you want more control over your figure. Rather than depending on some notion of an "active" figure or axes, in the object-oriented interface the plotting functions are *methods* of explicit `Figure` and `Axes` objects. To re-create the previous plot using this style of plotting, you might do the following:

面向对象接口更适合于这种复杂的场景和你需要对图表更多控制权的场景。与其依赖于“当前活跃的”图表和维度，面向对象接口提供的是具体 `Figure` 和 `Axes` 对象的方法。采用这种接口重新创建上面的图表的代码如下：

```
In [10]: # 首先创建两个子图表
# 返回fig是两个子Figure对象，ax是两个子Axes对象
fig, ax = plt.subplots(2)
```


# 在两个不同的Axes对象上调用plot方法分别作图  
ax[0].plot(x, np.sin(x))  
ax[1].plot(x, np.cos(x));



For more simple plots, the choice of which style to use is largely a matter of preference, but the object-oriented approach can become a necessity as plots become more complicated. Throughout this chapter, we will switch between the MATLAB-style and object-oriented interfaces, depending on what is most convenient. In most cases, the difference is as small as switching `plt.plot()` to `ax.plot()`, but there are a few catches that we will highlight as they come up in the following sections.

对于简单的图表来说，采用哪种风格的接口取取决于个人喜好，但是对于复杂的图表来说，面向对象接口是必须的。在本章中，我们会在两者之间进行切换，依据哪种方式更加方便来决定。在大多数情况下，`plt.plot()` 切换到 `ax.plot()` 之间的区别会很小，但是里面会有些坑，我们会在后续小节中着重指出。

< [更多资源](#) | [目录](#) | [简单的折线图](#) >

 [Open in Colab](#)