第八讲
Matplotlib基础（I）
Lecture 8
Matplotlib Fundamentals (I)

明玉瑞 Yurui Ming

yrming@gmail.com

声明
Disclaimer

- 本讲义在准备过程中由于时间所限，所用材料来源并未规范标示引用来源。所引材料仅用于教学所用，作者无意侵犯原著者之知识版权，所引材料之知识版权均归原著者所有；若原著者介意之，请联系作者更正及删除。

  The time limit during the preparation of these slides incurs the situation that not all the sources of the used materials (texts or images) are properly referenced or clearly manifested. However, all materials in these slides are solely for teaching and the author is with no intention to infringe the copyright bestowed on the original authors or manufacturers. All credits go to corresponding IP holders. Please address the author for any concern for remedy including deletion.

Matplotlib起源
Source of Matplotlib

- Matplotlib是由John D. Hunter博士在2003年创建的，最初目的是方便地将脑电图数据可视化。在科学计算与可视化领域，流行的做法是基于MATLAB计算引擎进行计算，基于MATLAB图形引擎进行渲染。但MATLAB是商业软件，有昂贵的授权费用。Matplotlib最初的设想包括在不同的平台提供MATLAB的一个可替代选择，具有高质量的光栅和矢量输出，提供对数学表达式的支持以及可以从 shell 交互式工作。

  Matplotlib was originally created by John D. Hunter in 2003 in order to visualize electrocorticography data. In the fields of scientific computing and visualization, conventional treatments are performing computations based on MATLAB's computing engine and rendering based on MATLAB's graphic engine. However, the proprietary software incurs high licensing fees for scholars that could be unaffordable. The initial goal of Matplotlib included a provision of MATLAB substation on different platforms, to have high quality raster and vector output, to provide support for mathematical expressions and to work interactively from the shell.

Matplotlib起源
Source of Matplotlib

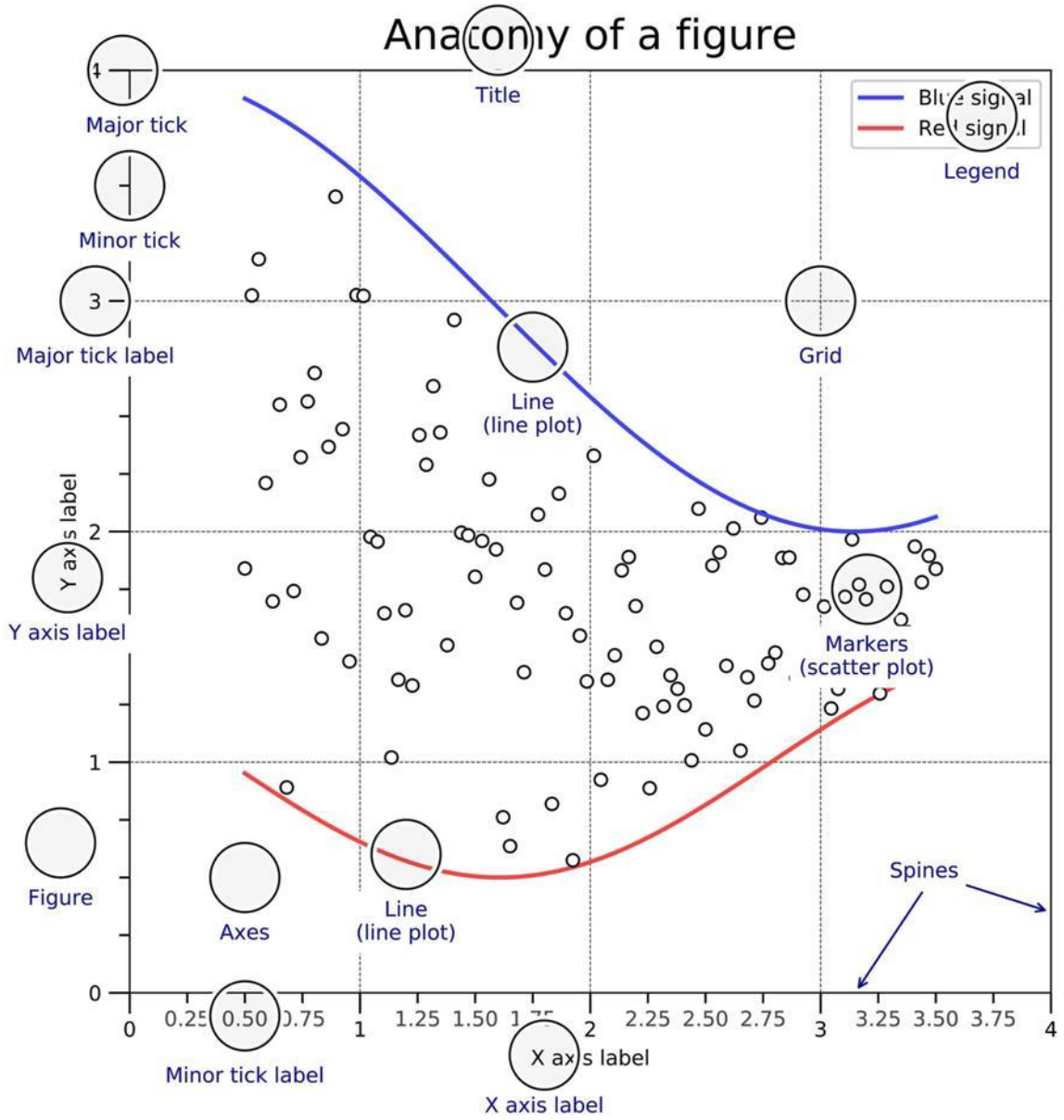
- Matplotlib提供了一个简单直观的界面 (pyplot) 以及一个面向对象的架构，允许调整图形中的任何内容。凭借强大的功能，它既可以设计非常高质量的图形科学出版，也可以用作常规图形库来设计非科学图形。时至今日，Matplotlib 库已是 Python 科学可视化的事实上的标准。

  Matplotlib offers both a simple and intuitive interface (pyplot) as well as an object-oriented architecture that allows users to tweak anything within a figure. Such a versatile and powerful library one way allows you to design very high-quality figures, suitable for scientific publishing, on the other hand, it can also be used as a regular graphic library in order to design non-scientific figures. Today, the Matplotlib library is a de facto standard for Python scientific visualization.

图示解析
Anatomy of Figure

- Matplotlib图形由具有层次结构的元素组成，当这些元素组合在一起时，就形成了如图所示的实际图形。

  A matplotlib figure is composed of a hierarchy of elements that, when put together, forms the actual figure as shown on figure.



图示解析
Anatomy of Figure

- 大多数情况下，图形元素并不是由用户明确创建的，而是从各种绘图命令的处理中衍生出来的。例如，考虑可以编写的最简单的 matplotlib 脚本：

  Most of the time, these elements are not created explicitly by the user but derived from the processing of the various plot commands. For example, let us consider the simplest matplotlib script the user can write:

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(range(10))
[<matplotlib.lines.Line2D object at 0x000001E5C77CBBC8>]
>>> plt.show()
```

- 为了显示结果，matplotlib 需要创建上图中所示的大部分元素。确切的元素列表取决于默认设置（以后会讲），但最低限度是创建一个图形，作为所有绘图元素的顶级容器，一个包含大部分图形元素的 Axes，当然还有实际绘制，在这种情况下是一条线。

  In order to display the result, matplotlib needs to create most of the elements shown on the previous figure. The exact list depends on the default settings (cover later), but the bare minimum is the creation of a Figure that is the top-level container for all the plot elements, an Axes that contains most of the figure elements and of course the actual plot, a line in this case.

图示解析
Anatomy of Figure

- 不必指定所有内容的方式可能很方便，但与此同时，它限制了选择的丰富性，因为缺少的元素会使用默认值自动创建。例如，在前面的示例中，拥护无法控制初始图形大小，因为在创建过程中已隐式选择它。如果要更改图形大小或轴方面，则需要更明确。

  The possibility of not necessary to specifying everything might be convenient. However, in the meantime, it limits the users' choices because missing elements are created automatically, using default values. For example, in the previous example, the user have no control of the initial figure size since is has been chosen implicitly during creation. If the user want to change the figure size or the axes aspect, it is needed to be more explicit specified.

```
>>> import matplotlib.pyplot as plt
>>>
>>> fig = plt.figure(figsize=(6,6))
>>> ax = plt.subplot(aspect=1)
>>> ax.plot(range(10))
[<matplotlib.lines.Line2D object at 0x000001E5C733B048>]
>>> plt.show()
>>>
```

图示解析
Anatomy of Figure

- 读者可能已经注意到，前面的两个简单的示例稍有不同。在第二个实例中，plot命令附加到ax上而不是plt上。直接使用plt.plot实际上是告诉matplotlib我们要在当前轴上绘图，即隐式或显式创建的最后一个轴。 在Python的哲学中，显式是优于隐式（import this）的。因此，当读者有选择时，最好准确地指定想要做什么。为此，了解图形的不同元素是很重要的。

  The user may have noticed the slightly difference between the previous two examples. In the second example, the plot command is attached to ax instead of plt. The use of plt.plot is actually to tell matplotlib that we want to plot on the current axes, that is, the last axes that has been created, implicitly or explicitly. No need to remind that explicit is better than implicit as in the The Zen of Python. When you have choice, it is thus preferable to specify exactly what you want to do. Consequently, it is important to know what are the different elements of a figure.

图示解析
Anatomy of Figure

- 图形：对图形而言，最重要的元素显然是图形本身。其是在用户调用figure方法时创建的，可以指定大小，也可以指定背景颜色（facecolor）以及标题（suptitle）等。这里提示一点，在保存图形时不会使用背景颜色，因为savefig函数还有一个facecolor参数（默认为白色），它将覆盖图形的背景颜色。如果不想要任何背景，可以在保存图形时指定 transparent=True 。

  Figure: The most important element of a figure is the figure itself. It is created when you call the figure method. You can specify its size, a background color (facecolor) as well as a title (suptitle). It is important to know that the background color won't be used when you save the figure because the savefig function has also a facecolor argument (that is white by default) that will override your figure background color. If you don't want any background, you can specify transparent=True when you save the figure.

图示解析
Anatomy of Figure

- 坐标系统：坐标轴或坐标系统是第二个最重要的元素，因为它对应于将要呈现的数据的实际区域。它也被称为子图。每个图形可以有一个到多个坐标系统，即一个或多个子图。每个坐标系统通常被称为脊的四个边缘（左、上、右和下）包围。这些脊中的每一个都可以用主要刻度和次要刻度（可以指向内部或外部）、刻度标签和标签来修饰。默认情况下，matplotlib 只修饰左侧和底部的脊。

  Axes : The axes is the second most important element, since it corresponds to the actual area where your data will be rendered. It is also called a subplot. You can have one to many axes per figure, or one to many subplots. Each axes is usually surrounded by four edges (left, top, right and bottom) that are called spines. Each of these spines can be decorated with major and minor ticks (that can point inward or outward), tick labels and a label. By default, matplotlib decorates only the left and bottom spines.

图示解析
Anatomy of Figure

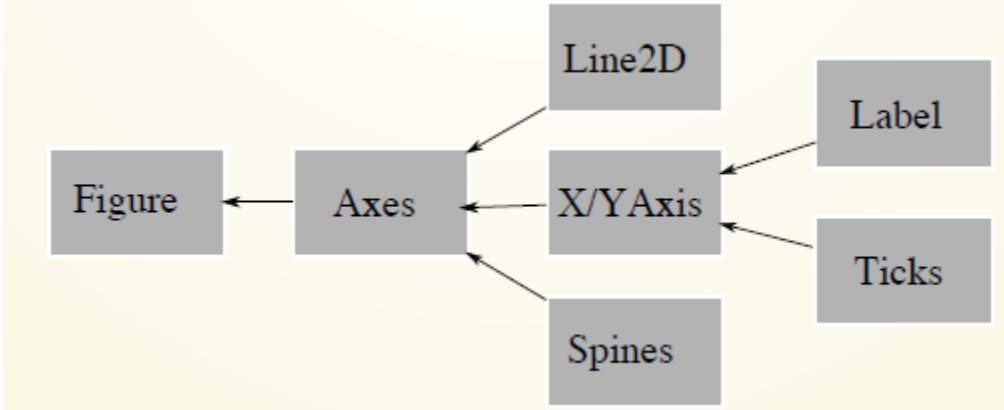
- 轴：经修饰的脊称为轴。水平方向即 x 轴，垂直方向即 y 轴。它们中的每一个都由脊、主要和次要刻度、主要和次要刻度标签和轴标签组成。

  Axis : The decorated spines are called axis. The horizontal one is the x-axis and the vertical one is the y-axis. Each of them are made of a spine, major and minor ticks, major and minor ticks labels and an axis label.

- 脊：脊是连接轴刻度线并示意数据区域边界的线。它们可以放置在任意位置，可以是可见的或不可见的。

  Spines : Spines are the lines connecting the axis tick marks and noting the boundaries of the data area. They can be placed at arbitrary positions and may be visible or invisible.

图示解析
Anatomy of Figure



- 艺术元：图形上的所有东西，包括图形、轴和单个坐标轴对象，都是艺术元。这包括Text对象、Line2D对象、集合对象、Patch对象。渲染图形时，所有艺术元都被绘制到画布上。给定的艺术元只能在一个轴中。

  Artist : Everything on the figure, including Figure, Axes, and Axis objects, is an artist. This includes Text objects, Line2D objects, collection objects, Patch objects. When the figure is rendered, all of the artists are drawn to the canvas. A given artist can only be in one Axes.

- 图形原语（或基元）：不管具体为何，绘制一般由块、线条和文本等图形原语组成。块可以非常小（例如标记）或非常大（例如条）并且具有一系列形状（圆形、矩形、多边形等）。 线条可以很小很细（例如刻度线）或很粗（例如影线）。文本可以使用系统上可用的任何字体，也可以使用latex引擎来呈现数学。

  Graphic primitives: A plot, independently of its nature, is made of patches, lines and texts. Patches can be very small (e.g., markers) or very large (e.g., bars) and have a range of shapes (circles, rectangles, polygons, etc.). Lines can be very small and thin (e.g., ticks) or very thick (e.g., hatches). Text can use any font available on your system and can also use a latex engine to render maths.

图示解析
Anatomy of Figure

- 图形原语中的每一个还具有许多其他属性，例如颜色（facecolor和edgecolor）、透明度（从0到1）、图案（例如破折号）、样式（例如帽子样式）、特殊效果（例如阴影或轮廓），抗锯齿（真或假）等。大多数时候，用户不应直接操作这些原语。相反，读者通过调用函数，在建构渲染的过程中，使用此类原语的集合。例如，当向图形添加新轴时，matplotlib将为脊和刻度构建线段集合，并且还将为刻度标签和轴标签添加标签集合。 即使这对用户来说是完全透明的，用户仍然可以在必要时单独访问这些元素。

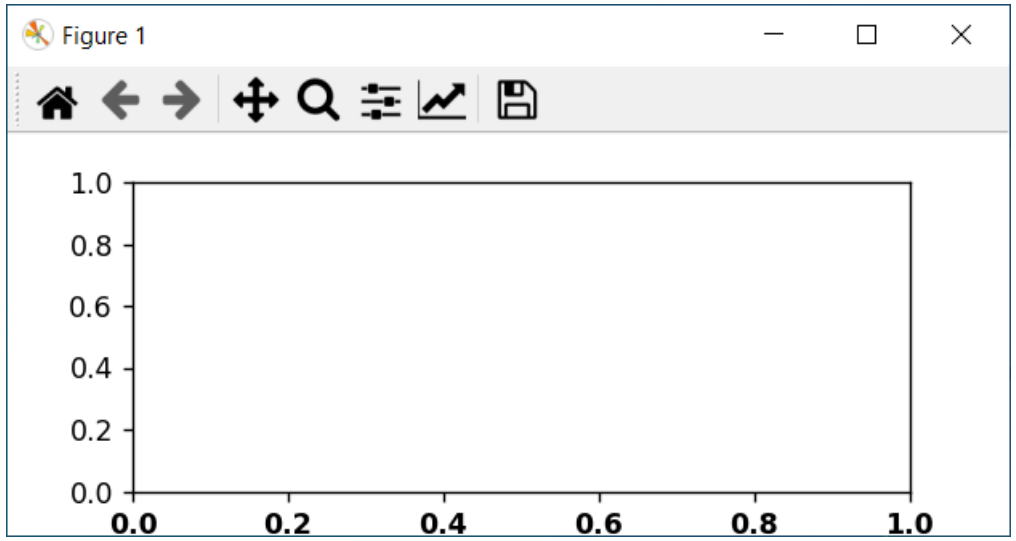
  Each of these graphic primitives also has couple of other properties such as color (facecolor and edgecolor), transparency (from 0 to 1), patterns (e.g., dashes), styles (e.g., cap styles), special effects (e.g., shadows or outline), antialiased (True or False), etc. Most of the time, users do not manipulate these primitives directly. Instead, they call methods that build a rendering using a collection of such primitives. For example, when you add a new axes to a figure, matplotlib will build a collection of line segments for the spines and the ticks and will also add a collection of labels for the tick labels and the axis labels. Even though this is totally transparent for the user, those elements can still be accessed individually if necessary.

图示解析
Anatomy of Figure

- 例如，要使 X 轴刻度变粗，我们可以这样写：

  For example, to make the X axis tick to be bold, we would write:

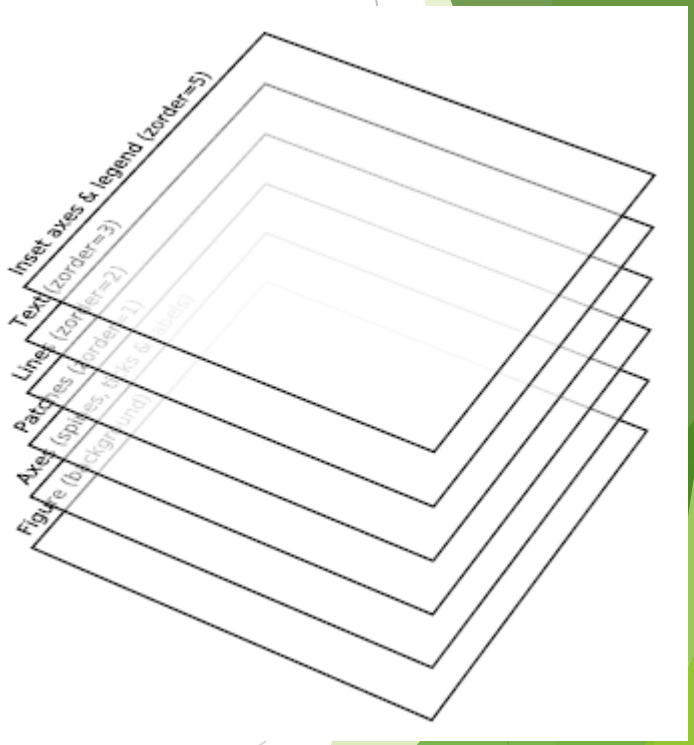
```
>>> fig, ax = plt.subplots(figsize=(5,2))
>>> for label in ax.get_xaxis().get_ticklabels():
...     label.set_fontweight("bold")
...
>>> plt.show()
```



图示解析
Anatomy of Figure

- 图元的一个重要属性是zorder属性，它反应了图元的（虚拟）深度，如图所示。 此zorder值用于在渲染图元之前，将图元从最低到最高排序。这用于控制谁先谁后的问题。大多数艺术元都拥有一个默认的zorder值，以便正确地渲染事物。例如，脊、刻度和刻度标签通常位于实际绘图的后面 。

  One important property of any primitive is the zorder property that indicates the virtual depth of the primitives as shown on figure. This zorder value is used to sort the primitives from the lowest to highest before rendering them. This allows to control what is behind what. Most artists possess a default zorder value such that things are rendered properly. For example, the spines, the ticks and the tick label are generally behind your actual plot.



图示解析
Anatomy of Figure

- 后端：后端是负责实际绘图的渲染器和允许与图形交互的可选用户界面的组合。到目前为止，我们一直在使用默认的渲染器和界面，以便在调用plt.show()方法时会显示一个窗口。 要知道当前使用的默认后端是什么，可以键入：

  Backends: A backend is the combination of a renderer that is responsible for the actual drawing and an optional user interface that allows to interact with a figure. Until now, we've been using the default renderer and interface resulting in a window being shown when the plt.show() method was called. To know what is your default backend, you can type:

```
>>> import matplotlib
>>> print(matplotlib.get_backend())
QtAgg
>>>
```

图示解析
Anatomy of Figure

- Matplotlib支持的渲染器和用户界面如下表所示：

  Supported renderers and user interfaces supported by matplotlib are as in the tables:

Renderer	Type	Filetype
Agg	raster	Portable Network Graphic (PNG)
PS	vector	Postscript (PS)
PDF	vector	Portable Document Format (PDF)
SVG	vector	Scalable Vector Graphics (SVG)
Cairo	raster / vector	PNG / PDF / SVG

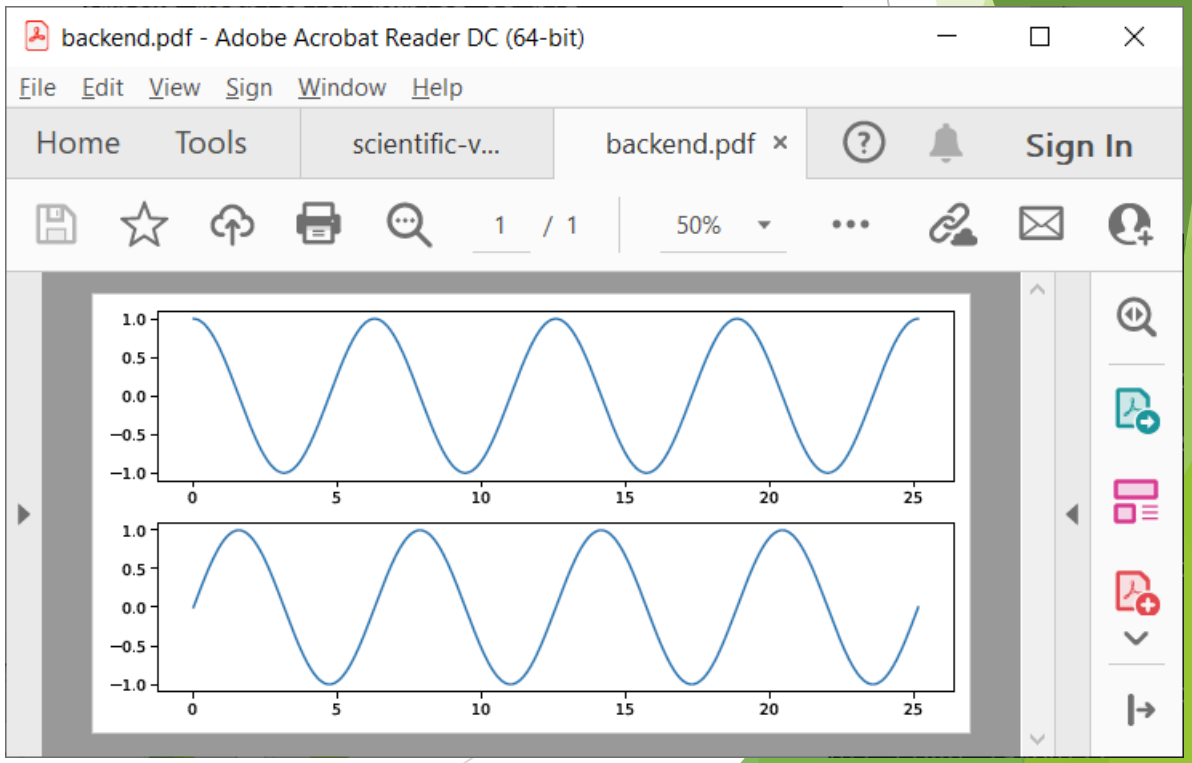
Interface	Renderer	Dependencies
GTK3	Agg or Cairo	PyGObject & Pycairo
Qt4	Agg	PyQt4
Qt5	Agg	PyQt5
Tk	Agg	TkInter
Wx	Agg	wxPython
MacOSX	—	OSX (obviously)
Web	Agg	Browser

图示解析 Anatomy of Figure

- 可以替换默认的渲染引擎，但某些引擎不具有对应的用户界面，因此不再能显示图形，但可以直接将图形存成文件。

  You can switch the default rendering engine to an alternative one. Note some rendering engine doesn't come with a user interface, which means you can not display and interact with the figure, however, you can directly save it to a file.

```
import numpy as np
import matplotlib

matplotlib.use("PDF")
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8, 4), frameon=False)

ax = plt.subplot(2, 1, 1)
X = np.linspace(0, 4 * 2 * np.pi, 500)
(line,) = ax.plot(X, np.cos(X))

ax = plt.subplot(2, 1, 2)
X = np.linspace(0, 4 * 2 * np.pi, 500)
(line,) = ax.plot(X, np.sin(X))

plt.tight_layout()
plt.savefig("backend.pdf")
```



图示解析
Anatomy of Figure

- 每英寸点数（DPI，或 dpi[1]）是度量打印、视频或图像扫描仪点密度的单位，即在 1 英寸（2.54 厘米）的跨度内可以排成一行的单个点的数量。值越大，表示解析越精细，或分辨率越高。

  Dots per inch (DPI, or dpi) is a measure of spatial printing, video or image scanner dot density, in particular the number of individual dots that can be placed in a line within the span of 1 inch (2.54 cm). The higher the value, the finer the details, or the greater the resolution.

- 尺寸和分辨率：在matplotlib中，默认的dpi为100。在该默认设定值下，一个大小为（6,6）图形，它对应于 6 英寸（宽）乘 6 英寸（高）的大小。当显示时在屏幕上，点对应于像素，我们可以推断出图形本身大小（即没有工具栏的窗口大小）正好是 600×600 像素。如果将图形保存为位图格式，例如 png（便携式网络图形），则可体现这一点。

  Dimensions & resolution: In matplotlib, the default dpi is 100. Under the default value, a figure of specified size (6,6) corresponds to a size of 6 inches (width) by 6 inches (height). When displayed on a screen, dots corresponds to pixels, and we can immediately deduce that the figure size (i.e., window size without the toolbar) will be exactly 600×600 pixels. This is manifest if the figure is saved in a bitmap format such as png (Portable Network Graphics).

图示解析
Anatomy of Figure

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure(figsize=(6,6))
>>> plt.savefig("output.png")
>>>
```

```
C:\Users\yrming\Desktop\备课\Python程序设计>magick identify -verbose output.png
Image:
  Filename: output.png
  Format: PNG (Portable Network Graphics)
  Mime type: image/png
  Class: DirectClass
  Geometry: 600x600+0+0
  Resolution: 39.37x39.37
  Print size: 15.24x15.24
  Units: PixelsPerCentimeter
  Colorspace: sRGB
  Type: Bilevel
  Base type: Undefined
  Endianness: Undefined
  Depth: 8/1-bit
```

图示解析
Anatomy of Figure

- 上例证实了图像的几何尺寸为600×600，而分辨率为39.37ppc（每厘米像素），则对应于 39.37*2.54 ≈ 100 dpi（每英寸点数）。如果要在保持相同 dpi 的情况下将此图像包含在文档中，则需要将图像的大小设置为 15.24 厘米 x 15.24 厘米。如果减小文档中图像的大小，例如缩小 3 倍，在这种特定情况下，这会机械地将图形dpi增加到300。对于一篇科学类文章，出版商通常会要求图像的dpi介于300到600之间。为了使事情正确，最好知道插入到文档中时图形的物理尺寸是多少。

  The above example confirms that the image geometry is 600×600 while the resolution is 39.37 ppc (pixels per centimeter), which corresponds to 39.37*2.54 ≈ 100 dpi (dots per inch). If you were to include this image inside a document while keeping the same dpi, you would need to set the size of the image to 15.24cm by 15.24cm. If you reduce the size of the image in your document, let’s say by a factor of 3, this will mechanically increase the figure dpi to 300 in this specific case. For a scientific article, publishers will generally request figures dpi to be between 300 and 600. To get things right, it is thus good to know what will be the physical dimension of your figure once inserted into your document.

图示解析
Anatomy of Figure

- 上例证实了图像的几何尺寸为600×600，而分辨率为39.37ppc（每厘米像素），则对应于 39.37*2.54 ≈ 100 dpi（每英寸点数）。如果要在保持相同 dpi 的情况下将此图像包含在文档中，则需要将图像的大小设置为 15.24 厘米 x 15.24 厘米。如果减小文档中图像的大小，例如缩小 3 倍，在这种特定情况下，这会机械地将图形dpi增加到300。对于一篇科学类文章，出版商通常会要求图像的dpi介于300到600之间。为了使事情正确，最好知道插入到文档中时图形的物理尺寸是多少。

  The above example confirms that the image geometry is 600×600 while the resolution is 39.37 ppc (pixels per centimeter), which corresponds to 39.37*2.54 ≈ 100 dpi (dots per inch). If you were to include this image inside a document while keeping the same dpi, you would need to set the size of the image to 15.24cm by 15.24cm. If you reduce the size of the image in your document, let’s say by a factor of 3, this will mechanically increase the figure dpi to 300 in this specific case. For a scientific article, publishers will generally request figures dpi to be between 300 and 600. To get things right, it is thus good to know what will be the physical dimension of your figure once inserted into your document.

图示解析
Anatomy of Figure

- 虽然matplotlib在绘制时指定了默认分辨率，但在保存图像时，可以重新指定分辨率。下面例子展示了以不同的分辨率保存包含文字的图像时的实际效果。注意，matplotlib 中默认使用 10 磅或 10 点大小的字体。但究竟什么是点？一点在 matplotlib 中它对应于 1/72 英寸。

  Although matplotlib uses default resolution for rendering, however, the resolution can be re-configured upon saving. The following example demonstrates the actual situations for images saving under different DPIs. Note by default matplotlib figure uses a font size of 10 points. But what is a point exactly? In matplotlib it corresponds to 1/72 inches.