

Content

[Print to PDF](#) ►

- [第一回：Matplotlib初相识](#)
 - [一、认识matplotlib](#)
 - [二、一个最简单的绘图例子](#)
 - [三、Figure的组成](#)
 - [四、两种绘图接口](#)
 - [参考资料](#)
- [第二回：艺术画笔见乾坤](#)
 - [一、概述](#)
 - [二、基本元素 - primitives](#)
 - [三、对象容器 - Object container](#)
 - [参考资料](#)
- [第三回：布局格式定方圆](#)
 - [一、子图](#)
 - [二、子图上的方法](#)
 - [参考资料](#)
- [第四回：文字图例尽眉目](#)
 - [一、Figure和Axes上的文本](#)
 - [二、Tick上的文本](#)
 - [三、legend（图例）](#)
 - [参考资料](#)
- [第五回：样式色彩秀芳华](#)
 - [一、matplotlib的绘图样式（style）](#)
 - [二、matplotlib的色彩设置（color）](#)
 - [参考资料](#)

By Datawhale

© Copyright © Copyright 2020.

第一回：Matplotlib初相识

一、认识matplotlib

Matplotlib是一个Python 2D绘图库，能够以多种硬拷贝格式和跨平台的交互式环境生成出版物质量的图形，用来绘制各种静态，动态，交互式的图表。

Matplotlib可用于Python脚本，Python和IPython Shell、Jupyter notebook，Web应用程序服务器和各种图形用户界面工具包等。

Matplotlib是Python数据可视化库中的泰斗，它已经成为python中公认的数据可视化工具，我们所熟知的pandas和seaborn的绘图接口其实也是基于matplotlib所作的高级封装。

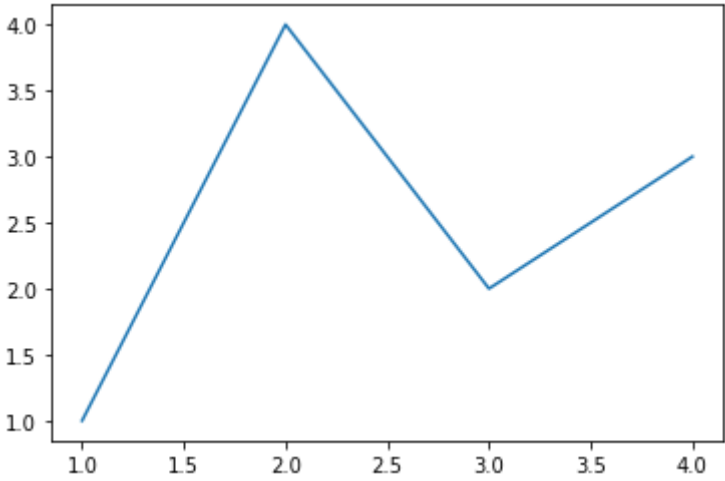
为了对matplotlib有更好的理解，让我们从一些最基本的概念开始认识它，再逐渐过渡到一些高级技巧中。

二、一个最简单的绘图例子

Matplotlib的图像是画在figure（如windows，jupyter窗体）上的，每一个figure又包含了一个或多个axes（一个可以指定坐标系的子区域）。最简单的创建figure以及axes的方式是通过`pyplot.subplots`命令，创建axes以后，可以使用`Axes.plot`绘制最简易的折线图。

```
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots() # 创建一个包含一个axes的figure
ax.plot([1, 2, 3, 4], [1, 4, 2, 3]); # 绘制图像
```



Trick: 在jupyter notebook中使用matplotlib时会发现，代码运行后自动打印出类似`<matplotlib.lines.Line2D at 0x23155916dc0>`这样一段话，这是因为matplotlib的绘图代码默认打印出最后一个对象。如果不想显示这句话，有以下三种方法，在本章节的代码示例中你能找到这三种方法的使用。

- 1. 在代码块最后加一个分号；
- 2. 在代码块最后加一句`plt.show()`
- 3. 在绘图时将绘图对象显式赋值给一个变量，如将`plt.plot([1, 2, 3, 4])` 改成`line =plt.plot([1, 2, 3, 4])`

和MATLAB命令类似，你还可以通过一种更简单的方式绘制图像，`matplotlib.pyplot`方法能够直接在当前axes上绘制图像，如果用户未指定axes，matplotlib会帮你自动创建一个。所以上面的例子也可以简化为以下这一行代码。

```
line =plt.plot([1, 2, 3, 4], [1, 4, 2, 3])
```

☰ Contents

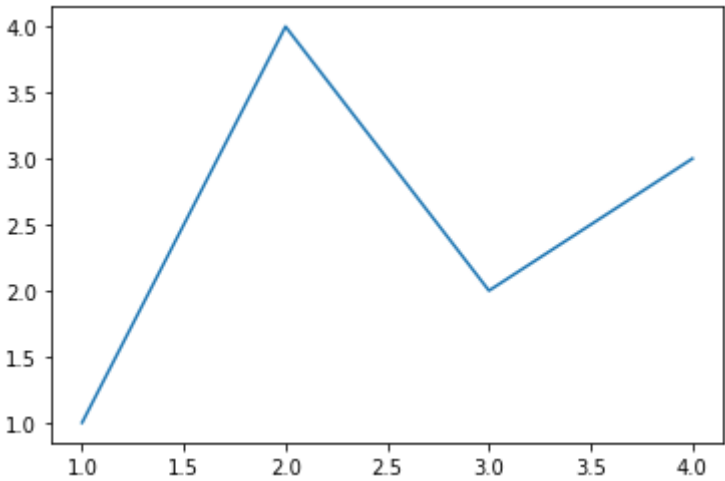
[二、认识matplotlib](#) [Print to PDF](#) ▶

[二、一个最简单的绘图例子](#)

[三、Figure的组成](#)

[四、两种绘图接口](#)

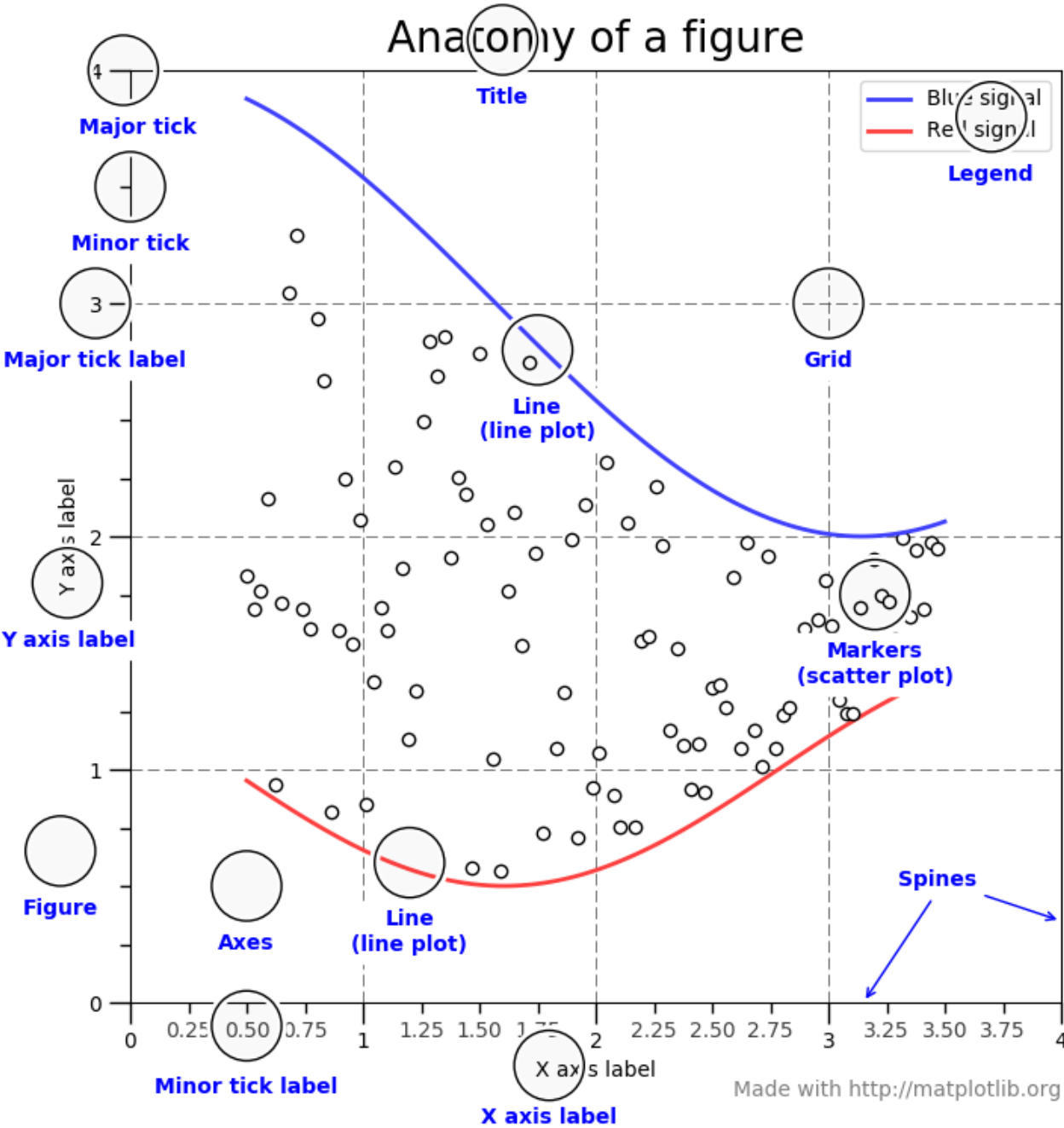
[参考资料](#)



三、Figure的组成

现在我们来深入看一下figure的组成。通过一张figure解剖图，我们可以看到一个完整的matplotlib图像通常会包括以下四个层级，这些层级也被称为容器（container），下一节会详细介绍。在matplotlib的世界中，我们将通过各种命令方法来操纵图像中的每一个部分，从而达到数据可视化的最终效果，一副完整的图像实际上是各类子元素的集合。

- **Figure**: 顶层级，用来容纳所有绘图元素
- **Axes**: matplotlib宇宙的核心，容纳了大量元素用来构造一幅幅子图，一个figure可以由一个或多个子图组成
- **Axis**: axes的下属层级，用于处理所有和坐标轴，网格有关的元素
- **Tick**: axis的下属层级，用来处理所有和刻度有关的元素



四、两种绘图接口

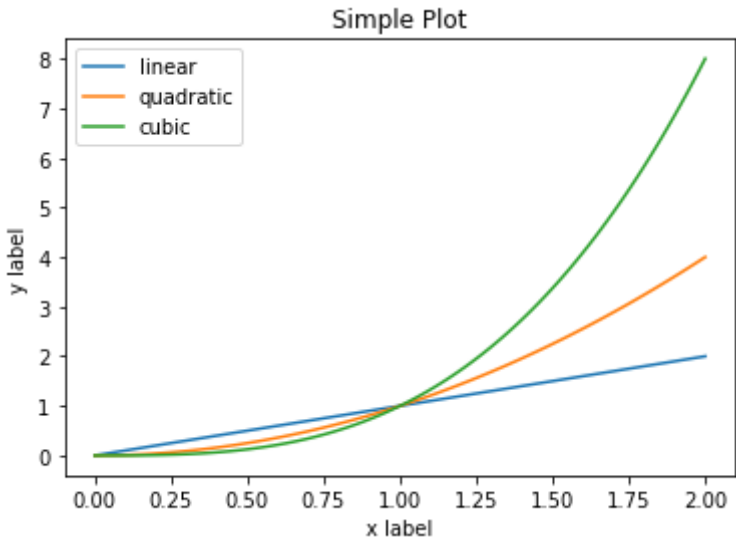
matplotlib提供了两种最常用的绘图接口

1. 显式创建figure和axes，在上面调用绘图方法，也被称为OO模式（object-oriented style）
2. 依赖pyplot自动创建figure和axes，并绘图

使用第一种绘图接口，是这样的：

```
x = np.linspace(0, 2, 100)

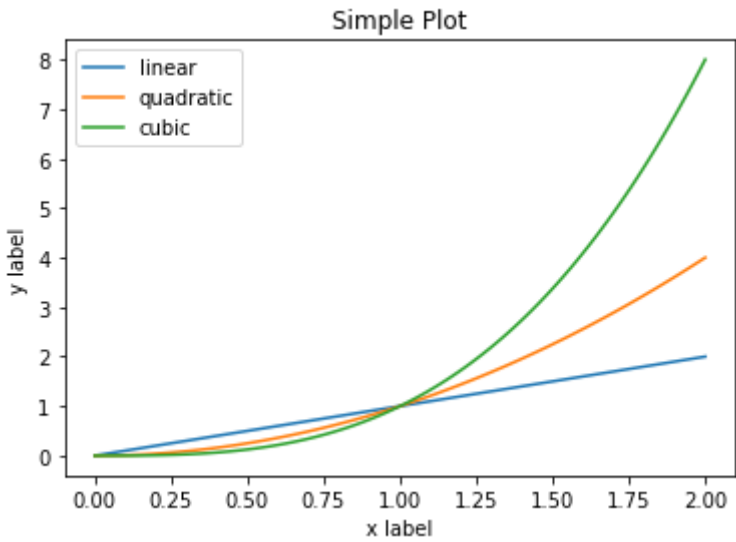
fig, ax = plt.subplots()
ax.plot(x, x, label='linear')
ax.plot(x, x**2, label='quadratic')
ax.plot(x, x**3, label='cubic')
ax.set_xlabel('x label')
ax.set_ylabel('y label')
ax.set_title("Simple Plot")
ax.legend()
plt.show()
```



而如果采用第二种绘图接口，绘制同样的图，代码是这样的：

```
x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Simple Plot")
plt.legend()
plt.show()
```



参考资料

[1.matplotlib官网用户指南](#)

By Datawhale
© Copyright © Copyright 2020.

第二回：艺术画笔见乾坤

一、概述

1. matplotlib的三层api

matplotlib的原理或者说基础逻辑是，用Artist对象在画布(canvas)上绘制(Render)图形。就和人作画的步骤类似：

- 1. 准备一块画布或画纸
- 2. 准备好颜料、画笔等制图工具
- 3. 作画

所以matplotlib有三个层次的API：

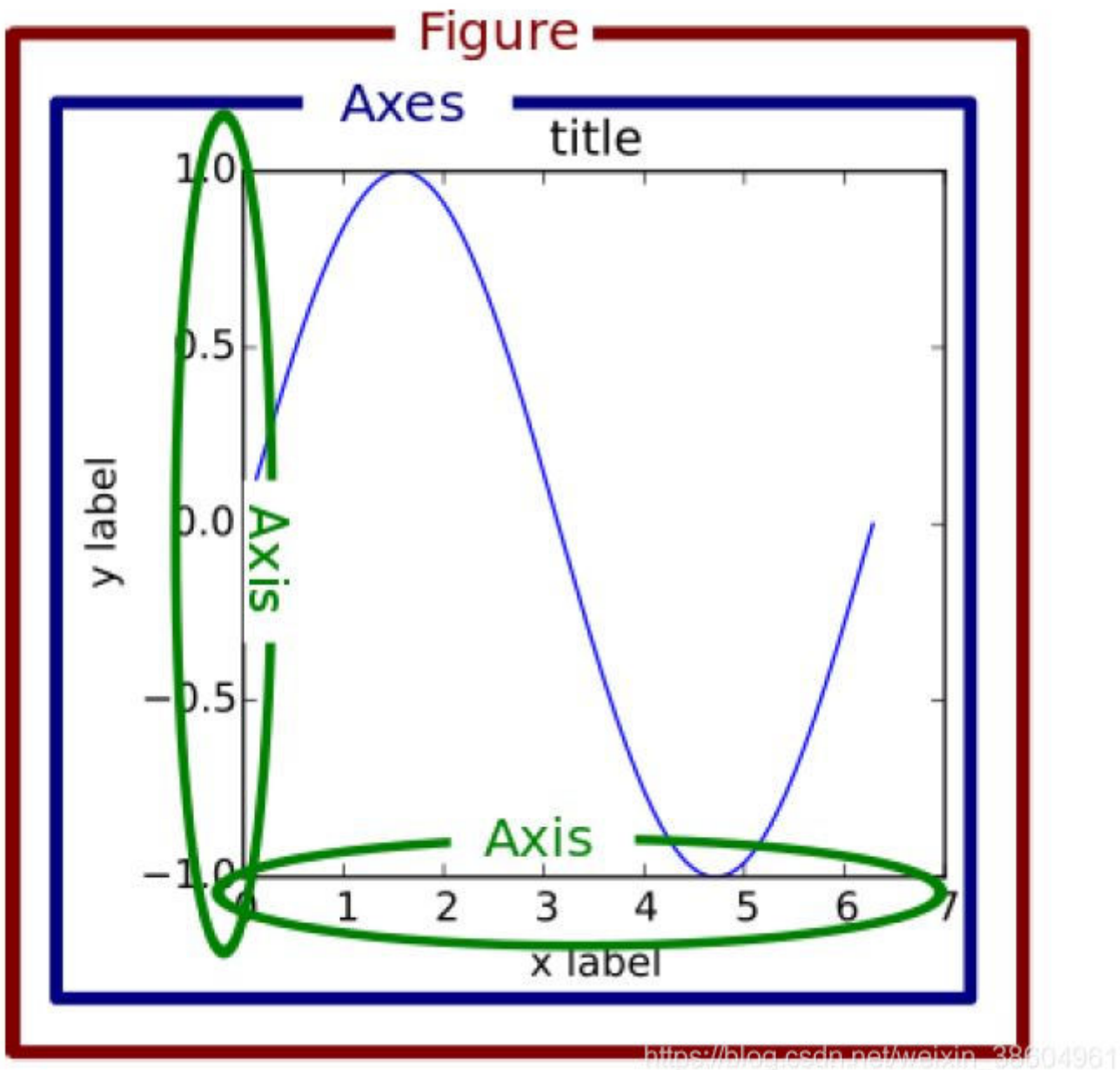
`matplotlib.backend_bases.FigureCanvas` 代表了绘图区，所有的图像都是在绘图区完成的
`matplotlib.backend_bases.Renderer` 代表了渲染器，可以近似理解为画笔，控制如何在 `FigureCanvas` 上画图。
`matplotlib.artist.Artist` 代表了具体的图表组件，即调用了`Renderer`的接口在`Canvas`上作图。
前两者处理程序和计算机的底层交互的事项，第三项`Artist`就是具体的调用接口来做出我们想要的图，比如图形、文本、线条的设定。所以通常来说，我们95%的时间，都是用来和`matplotlib.artist.Artist`类打交道的。

2. Artist的分类

Artist有两种类型：`primitives` 和 `containers`。

`primitive`是基本要素，它包含一些我们要在绘图区作图用到的标准图形对象，如**曲线Line2D**，**文字text**，**矩形Rectangle**，**图像image**等。

`container`是容器，即用来装基本要素的地方，包括**图形figure**、**坐标系Axes**和**坐标轴Axis**。他们之间的关系如下图所示：



Contents

二、概述

Print to PDF ▶

1. matplotlib的三层api

2. Artist的分类

3. matplotlib标准用法

二、基本元素 - primitives

1. 2DLines

2. patches

3. collections

4. images

三、对象容器 - Object container

1. Figure容器

2. Axes容器

3. Axis容器

4. Tick容器

参考资料

3. matplotlib标准用法

matplotlib的标准使用流程为：

- 1. 创建一个Figure实例
- 2. 使用Figure实例创建一个或者多个Axes或Subplot实例
- 3. 使用Axes实例的辅助方法来创建primitive

值得一提的是，Axes是一种容器，它可能是matplotlib API中最重要的类，并且我们大多数时间都花在和它打交道上。更具体的信息会在第三节容器小节说明。

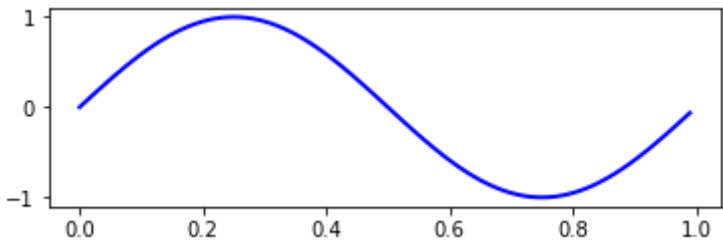
一个流程示例及说明如下：

```
import matplotlib.pyplot as plt
import numpy as np

# step 1
# 我们用 matplotlib.pyplot.figure() 创建了一个Figure实例
fig = plt.figure()

# step 2
# 然后用Figure实例创建了一个两行一列(即可以有两个subplot)的绘图区，并同时在第一个位置创建了一个subplot
ax = fig.add_subplot(2, 1, 1) # two rows, one column, first plot

# step 3
# 然后用Axes实例的方法画了一条曲线
t = np.arange(0.0, 1.0, 0.01)
s = np.sin(2*np.pi*t)
line, = ax.plot(t, s, color='blue', lw=2)
```



二、基本元素 - primitives

各容器中可能会包含多种基本要素-primitives, 所以先介绍下primitives，再介绍容器。

本章重点介绍下 primitives 的几种类型：**曲线-Line2D**，**矩形-Rectangle**，**图像-image**（其中文本-Text较为复杂，会在之后单独详细说明。）

1. 2DLines

在matplotlib中曲线的绘制，主要是通过类 matplotlib.lines.Line2D 来完成的。

它的基类: matplotlib.artist.Artist

matplotlib中**线-line**的含义：它表示的可以是连接所有顶点的实线样式，也可以是每个顶点的标记。此外，这条线也会受到绘画风格的影响，比如，我们可以创建虚线种类的线。

它的构造函数：

```
class matplotlib.lines.Line2D(xdata, ydata, linewidth=None, linestyle=None, color=None,
marker=None, markersize=None, markeredgewidth=None, markeredgewidth=None,
markerfacecolor=None, markerfacecoloralt='none', fillstyle=None, antialiased=None,
dash_capstyle=None, solid_capstyle=None, dash_joinstyle=None, solid_joinstyle=None,
pickradius=5, drawstyle=None, markevery=None, **kwargs)
```

其中常用的的参数有：

- **xdata**:需要绘制的line中点的在x轴上的取值，若忽略，则默认为range(1,len(ydata)+1)
- **ydata**:需要绘制的line中点的在y轴上的取值
- **linewidth**:线条的宽度
- **linestyle**:线型
- **color**:线条的颜色
- **marker**:点的标记，详细可参考[markers API](#)
- **markersize**:标记的size

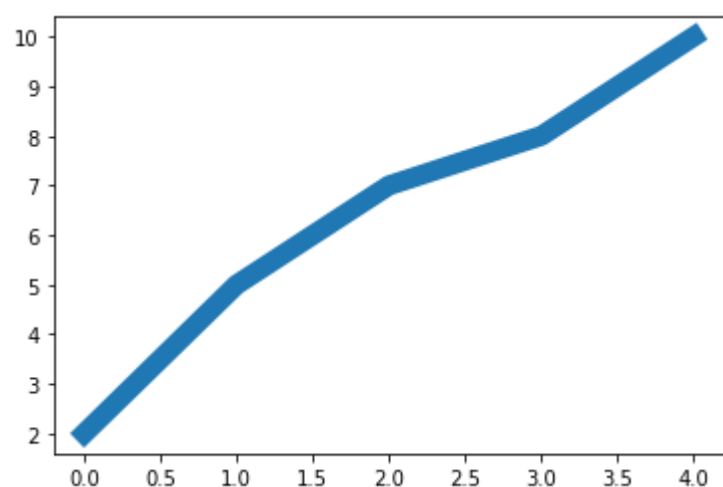
其他详细参数可参考[Line2D官方文档](#)

a. 如何设置Line2D的属性

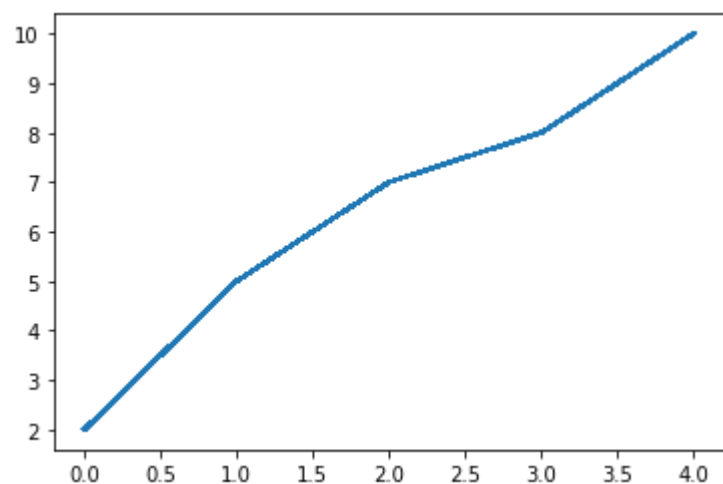
有三种方法可以用设置线的属性。

1. 直接在`plot()`函数中设置
2. 通过获得线对象，对线对象进行设置
3. 获得线属性，使用`setp()`函数设置

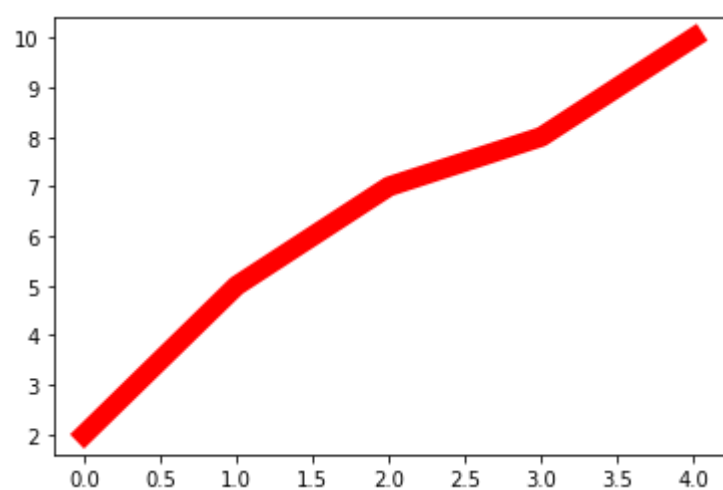
```
# 1) 直接在plot()函数中设置
import matplotlib.pyplot as plt
x = range(0,5)
y = [2,5,7,8,10]
plt.plot(x,y, linewidth=10); # 设置线的粗细参数为10
```



```
# 2) 通过获得线对象，对线对象进行设置
x = range(0,5)
y = [2,5,7,8,10]
line, = plt.plot(x, y, '-')
line.set_antialiased(False) # 关闭抗锯齿功能
```



```
# 3) 获得线属性，使用setp()函数设置
x = range(0,5)
y = [2,5,7,8,10]
lines = plt.plot(x, y)
plt.setp(lines, color='r', linewidth=10);
```



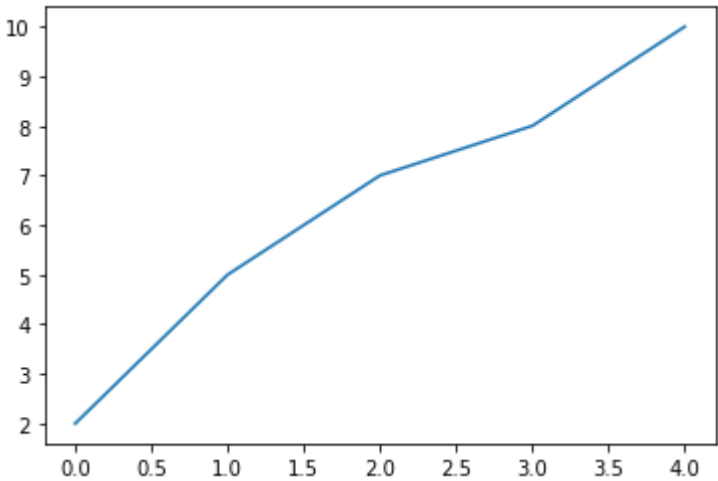
b. 如何绘制lines

1. 绘制直线line
2. errorbar绘制误差折线图

绘制直线line常用的方法有两种:

- **pyplot方法绘制**
- **Line2D对象绘制**

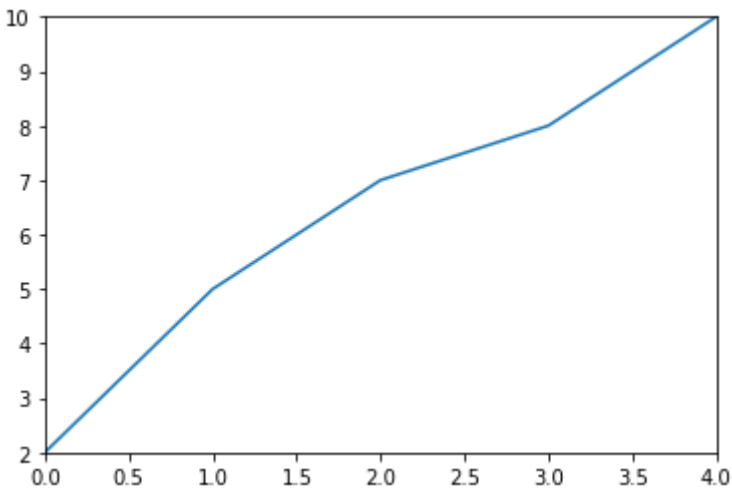
```
# 1. pyplot方法绘制
import matplotlib.pyplot as plt
x = range(0,5)
y = [2,5,7,8,10]
plt.plot(x,y);
```



```
# 2. Line2D对象绘制
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D

fig = plt.figure()
ax = fig.add_subplot(111)
line = Line2D(x, y)
ax.add_line(line)
ax.set_xlim(min(x), max(x))
ax.set_ylim(min(y), max(y))

plt.show()
```



2) **errorbar**绘制误差折线图

pyplot里有个专门绘制误差线的功能，通过**errorbar**类实现，它的构造函数：

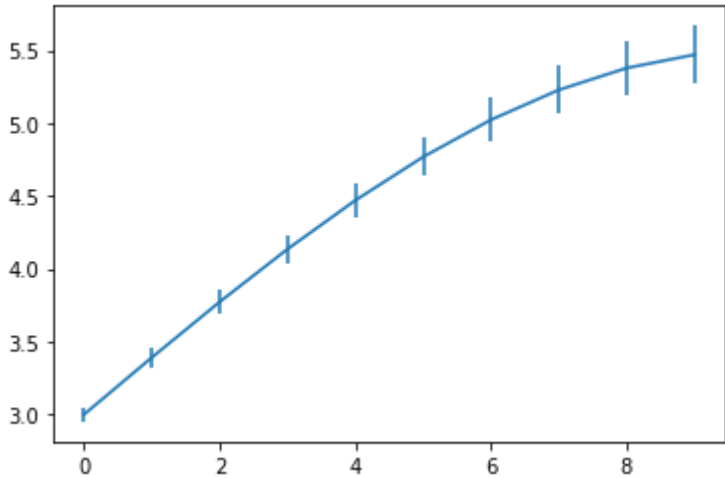
```
matplotlib.pyplot.errorbar(x, y, yerr=None, xerr=None, fmt="", ecolor=None, elinewidth=None,
capsize=None, barsabove=False, lolims=False, uplims=False, xlolims=False, xuplims=False,
errorevery=1, capthick=None, *, data=None, **kwargs)
```

其中最主要的参数是前几个:

- **x**: 需要绘制的line中点的在x轴上的取值
- **y**: 需要绘制的line中点的在y轴上的取值
- **yerr**: 指定y轴水平的误差
- **xerr**: 指定x轴水平的误差
- **fmt**: 指定折线图中某个点的颜色，形状，线条风格，例如'co--'
- **ecolor**: 指定error bar的颜色
- **elinewidth**: 指定error bar的线条宽度

绘制errorbar


```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
x = np.arange(10)
y = 2.5 * np.sin(x / 20 * np.pi)
yerr = np.linspace(0.05, 0.2, 10)
plt.errorbar(x, y + 3, yerr=yerr, label='both limits (default)');
```



2. patches

matplotlib.patches.Patch类是二维图形类。它的基类是matplotlib.artist.Artist，它的构造函数：

详细清单见 [matplotlib.patches API](#)

```
Patch(edgecolor=None, facecolor=None, color=None, linewidth=None, linestyle=None,
antialiased=None, hatch=None, fill=True, capstyle=None, joinstyle=None, **kwargs)
```

a. Rectangle-矩形

Rectangle矩形类在官网中的定义是：通过锚点xy及其宽度和高度生成。Rectangle本身的主要比较简单，即xy控制锚点，width和height分别控制宽和高。它的构造函数：

```
class matplotlib.patches.Rectangle(xy, width, height, angle=0.0, **kwargs)
```

在实际中最常见的矩形图是**hist直方图**和**bar条形图**。

1) hist-直方图

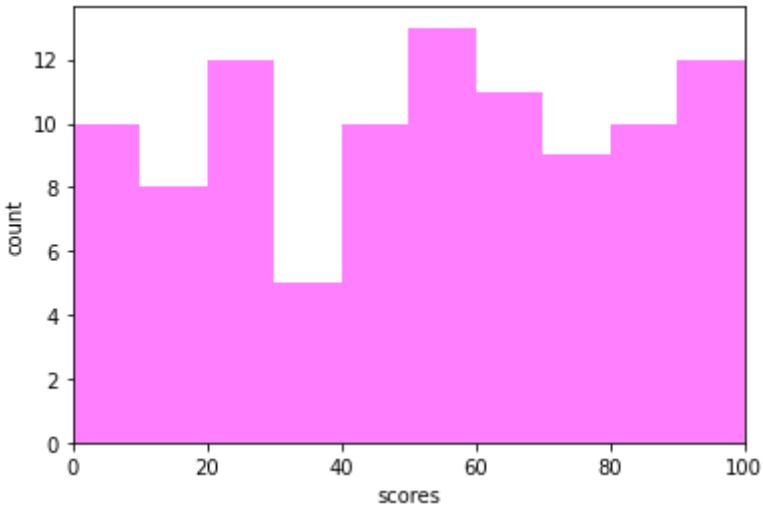
```
matplotlib.pyplot.hist(x,bins=None,range=None, density=None, bottom=None, histtype='bar',
align='mid', log=False, color=None, label=None, stacked=False, normed=None)
```

下面是一些常用的参数：

- **x**: 数据集，最终的直方图将对数据集进行统计
- **bins**: 统计的区间分布
- **range**: tuple, 显示的区间，range在没有给出bins时生效
- **density**: bool, 默认为false，显示的是频数统计结果，为True则显示频率统计结果，这里需要注意，频率统计结果=区间数目/(总数*区间宽度)，和normed效果一致，官方推荐使用density
- **histtype**: 可选{'bar', 'barstacked', 'step', 'stepfilled'}之一，默认为bar，推荐使用默认配置，step使用的是梯状，stepfilled则会对梯状内部进行填充，效果与bar类似
- **align**: 可选{'left', 'mid', 'right'}之一，默认为'mid'，控制柱状图的水平分布，left或者right，会有部分空白区域，推荐使用默认
- **log**: bool, 默认False,即y坐标轴是否选择指数刻度
- **stacked**: bool, 默认为False，是否为堆积状图

hist绘制直方图

```
import matplotlib.pyplot as plt
import numpy as np
x=np.random.randint(0,100,100) #生成[0-100)之间的100个数据,即 数据集
bins=np.arange(0,101,10) #设置连续的边界值，即直方图的分布区间[0,10),[10,20)...
plt.hist(x,bins,color='fuchsia',alpha=0.5)#alpha设置透明度，0为完全透明
plt.xlabel('scores')
plt.ylabel('count')
plt.xlim(0,100)#设置x轴分布范围
plt.show()
```



Rectangle矩形类绘制直方图

```
import pandas as pd
import re
df = pd.DataFrame(columns = ['data'])
df.loc[:, 'data'] = x
df['fenzu'] = pd.cut(df['data'], bins=bins, right = False,include_lowest=True)

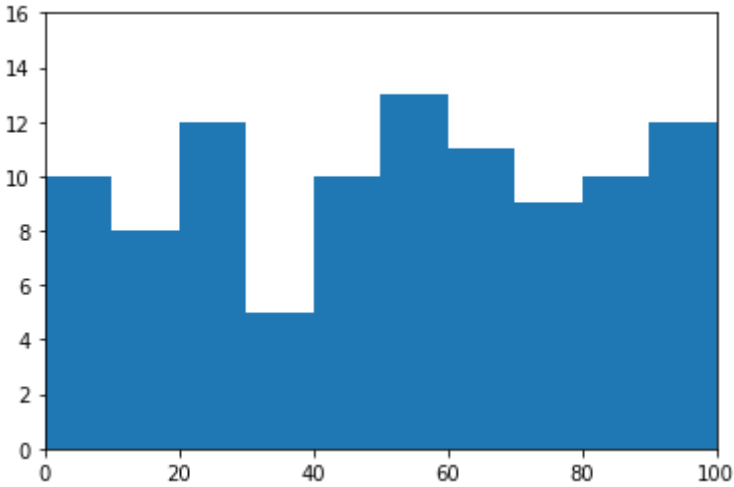
df_cnt = df['fenzu'].value_counts().reset_index()
df_cnt.loc[:, 'mini'] = df_cnt['index'].astype(str).map(lambda x:re.findall('\[(.*)\]',x)
[0]).astype(int)
df_cnt.loc[:, 'maxi'] = df_cnt['index'].astype(str).map(lambda x:re.findall('\,(.*)\)',x)
[0]).astype(int)
df_cnt.loc[:, 'width'] = df_cnt['maxi']- df_cnt['mini']
df_cnt.sort_values('mini',ascending = True,inplace = True)
df_cnt.reset_index(inplace = True,drop = True)

#用Rectangle把hist绘制出来
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

for i in df_cnt.index:
    rect =
plt.Rectangle((df_cnt.loc[i, 'mini'],0),df_cnt.loc[i, 'width'],df_cnt.loc[i, 'fenzu'])
    ax1.add_patch(rect)

ax1.set_xlim(0, 100)
ax1.set_ylim(0, 16)
plt.show()
```



2) bar-柱状图

```
matplotlib.pyplot.bar(left, height, alpha=1, width=0.8, color=, edgecolor=, label=, lw=3)
```

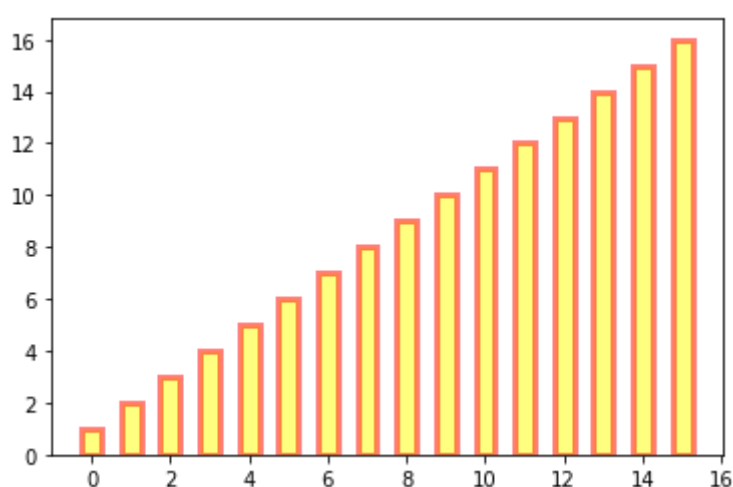
下面是一些常用的参数：

- **left**: x轴的位置序列，一般采用range函数产生一个序列，但是有时候可以是字符串
- **height**: y轴的数值序列，也就是柱形图的高度，一般就是我们需要展示的数据；
- **alpha**: 透明度，值越小越透明
- **width**: 为柱形图的宽度，一般这是为0.8即可；
- **color或facecolor**: 柱形图填充的颜色；
- **edgecolor**: 图形边缘颜色
- **label**: 解释每个图像代表的含义，这个参数是为legend()函数做铺垫的，表示该次bar的标签

有两种方式绘制柱状图

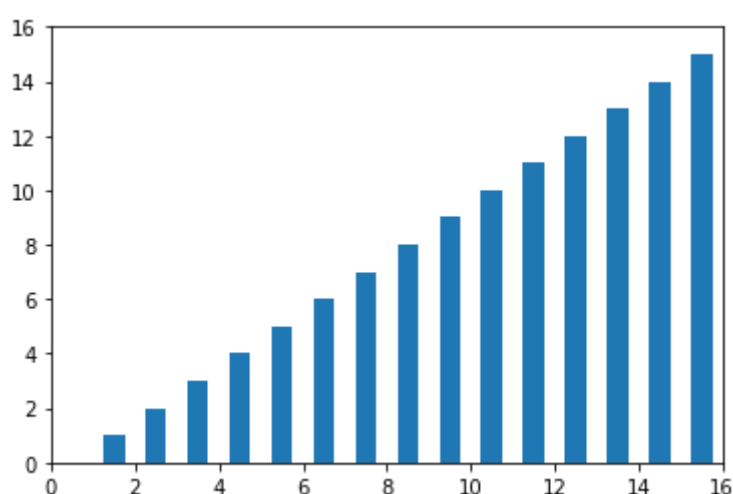
- bar绘制柱状图
- Rectangle矩形类绘制柱状图

```
# bar绘制柱状图
import matplotlib.pyplot as plt
y = range(1,17)
plt.bar(np.arange(16), y, alpha=0.5, width=0.5, color='yellow', edgecolor='red', label='The First Bar', lw=3);
```



```
# Rectangle矩形类绘制柱状图
import matplotlib.pyplot as plt
fig = plt.figure()
ax1 = fig.add_subplot(111)

for i in range(1,17):
    rect = plt.Rectangle((i+0.25,0),0.5,i)
    ax1.add_patch(rect)
ax1.set_xlim(0, 16)
ax1.set_ylim(0, 16)
plt.show()
```



b. Polygon-多边形

matplotlib.patches.Polygon类是多边形类。其基类是matplotlib.patches.Patch，它的构造函数：

```
class matplotlib.patches.Polygon(xy, closed=True, **kwargs)
```

xy是一个N×2的numpy array，为多边形的顶点。

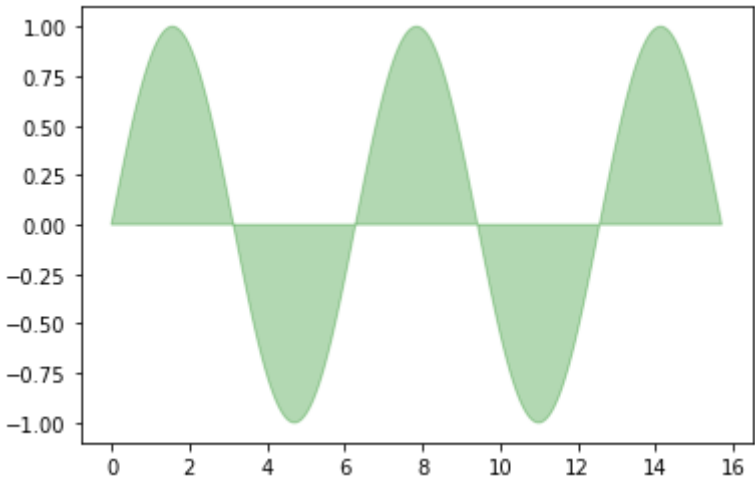
closed为True则指定多边形将起点和终点重合从而显式关闭多边形。

matplotlib.patches.Polygon类中常用的是fill类，它是基于xy绘制一个填充的多边形，它的定义：

```
matplotlib.pyplot.fill(*args, data=None, **kwargs)
```

参数说明：关于x、y和color的序列，其中color是可选的参数，每个多边形都是由其节点的x和y位置列表定义的，后面可以选择一个颜色说明符。您可以通过提供多个x、y、[颜色]组来绘制多个多边形。

```
# 用fill来绘制图形
import matplotlib.pyplot as plt
x = np.linspace(0, 5 * np.pi, 1000)
y1 = np.sin(x)
y2 = np.sin(2 * x)
plt.fill(x, y1, color = "g", alpha = 0.3);
```



c. Wedge-契形

matplotlib.patches.Polygon类是多边形类。其基类是matplotlib.patches.Patch，它的构造函数：

```
class matplotlib.patches.Wedge(center, r, theta1, theta2, width=None, **kwargs)
```

一个Wedge-契形 是以坐标x,y为中心，半径为r，从θ1扫到θ2(单位是度)。
如果宽度给定，则从内半径r - 宽度到外半径r画出部分楔形。wedge中比较常见的是绘制饼状图。

matplotlib.pyplot.pie语法：

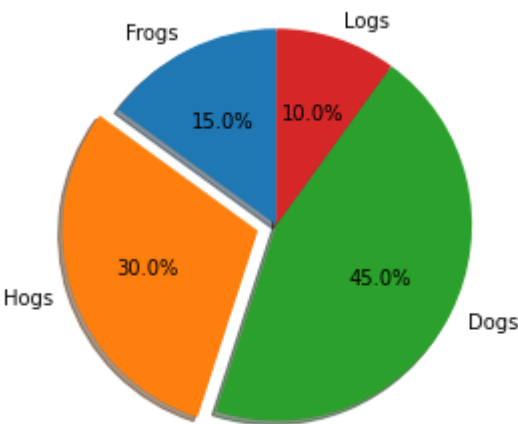
```
matplotlib.pyplot.pie(x, explode=None, labels=None, colors=None, autopct=None, pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1, counterclock=True, wedgeprops=None, textprops=None, center=0, 0, frame=False, rotatelabels=False, *, normalize=None, data=None)
```

制作数据x的饼图，每个楔子的面积用x/sum(x)表示。
其中最主要的参数是前4个：

- **x**：契型的形状，一维数组。
- **explode**：如果不是等于None，则是一个len(x)数组，它指定用于偏移每个楔形块的半径的分数。
- **labels**：用于指定每个契型块的标记，取值是列表或为None。
- **colors**：饼图循环使用的颜色序列。如果取值为None，将使用当前活动循环中的颜色。
- **startangle**：饼状图开始的绘制的角度。

pie绘制饼状图

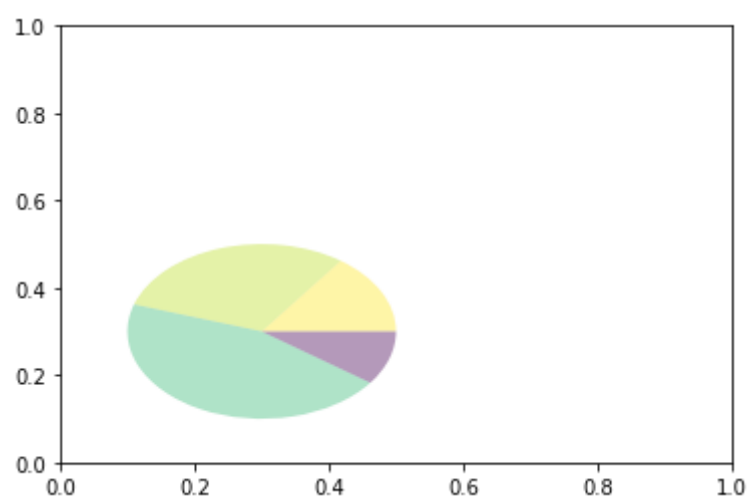
```
import matplotlib.pyplot as plt
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=True, startangle=90)
ax1.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



wedge绘制饼图

```
import matplotlib.pyplot as plt
from matplotlib.patches import Circle, Wedge
from matplotlib.collections import PatchCollection

fig = plt.figure()
ax1 = fig.add_subplot(111)
theta1 = 0
sizes = [15, 30, 45, 10]
patches = []
patches += [
    Wedge((0.3, 0.3), .2, 0, 54),           # Full circle
    Wedge((0.3, 0.3), .2, 54, 162),        # Full ring
    Wedge((0.3, 0.3), .2, 162, 324),       # Full sector
    Wedge((0.3, 0.3), .2, 324, 360),      # Ring sector
]
colors = 100 * np.random.rand(len(patches))
p = PatchCollection(patches, alpha=0.4)
p.set_array(colors)
ax1.add_collection(p)
plt.show()
```



3. collections

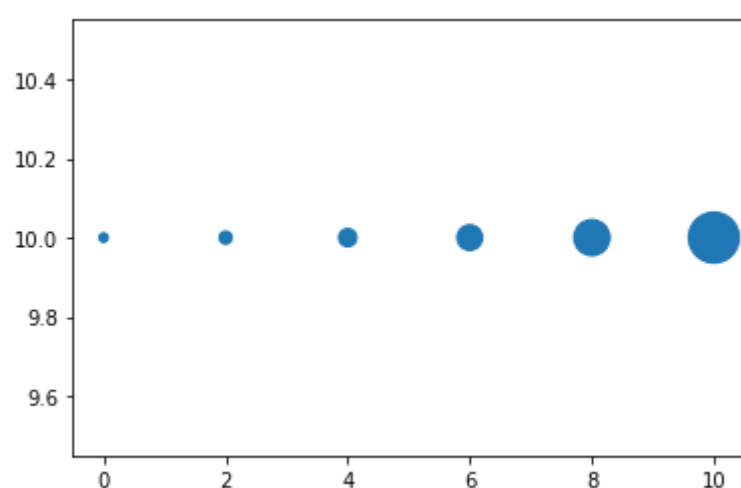
`collections`类是用来绘制一组对象的集合，`collections`有许多不同的子类，如`RegularPolyCollection`, `CircleCollection`, `Pathcollection`, 分别对应不同的集合子类型。其中比较常用的就是散点图，它是属于`PathCollection`子类，`scatter`方法提供了该类的封装，根据x与y绘制不同大小或颜色标记的散点图。它的构造方法：

```
Axes.scatter(self, x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None,
vmax=None, alpha=None, linewidths=None, verts=, edgecolors=None, *, plotnonfinite=False,
data=None, **kwargs)
```

其中最主要的参数是前5个：

- **x**: 数据点x轴的位置
- **y**: 数据点y轴的位置
- **s**: 尺寸大小
- **c**: 可以是单个颜色格式的字符串，也可以是一系列颜色
- **marker**: 标记的类型

```
# 用scatter绘制散点图
x = [0, 2, 4, 6, 8, 10]
y = [10]*len(x)
s = [20*2**n for n in range(len(x))]
plt.scatter(x, y, s=s)
plt.show()
```



4. images

images是matplotlib中绘制image图像的类，其中最常用的imshow可以根据数组绘制成图像，它的构造函数：

```
class matplotlib.image.AxesImage(ax, cmap=None, norm=None, interpolation=None, origin=None,
    extent=None, filternorm=True, filterrad=4.0, resample=False, **kwargs)
```

imshow根据数组绘制图像

```
matplotlib.pyplot.imshow(X, cmap=None, norm=None, aspect=None, interpolation=None,
    alpha=None, vmin=None, vmax=None, origin=None, extent=None, shape=, filternorm=1,
    filterrad=4.0, imlim=, resample=None, url=None, *, data=None, **kwargs)
```

使用imshow画图时首先需要传入一个数组，数组对应的是空间内的像素位置和像素点的值，interpolation参数可以设置不同的差值方法，具体效果如下。

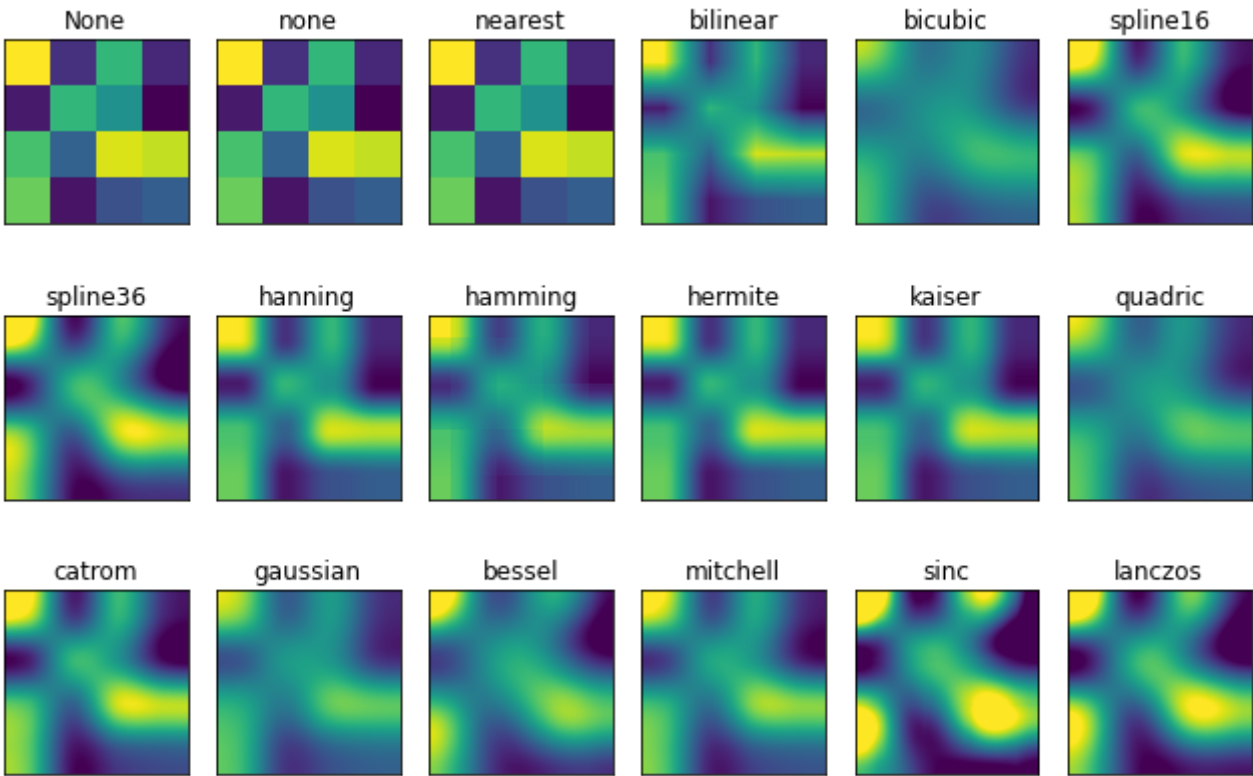
```
import matplotlib.pyplot as plt
import numpy as np
methods = [None, 'none', 'nearest', 'bilinear', 'bicubic', 'spline16',
           'spline36', 'hanning', 'hamming', 'hermite', 'kaiser', 'quadric',
           'catrom', 'gaussian', 'bessel', 'mitchell', 'sinc', 'lanczos']

grid = np.random.rand(4, 4)

fig, axs = plt.subplots(nrows=3, ncols=6, figsize=(9, 6),
    subplot_kw={'xticks': [], 'yticks': []})

for ax, interp_method in zip(axs.flat, methods):
    ax.imshow(grid, interpolation=interp_method, cmap='viridis')
    ax.set_title(str(interp_method))

plt.tight_layout()
plt.show()
```



三、对象容器 - Object container

容器会包含一些primitives，并且容器还有它自身的属性。

比如Axes Artist，它是一种容器，它包含了很多primitives，比如Line2D, Text；同时，它也有自身的属性，比如xscal，用来控制X轴是linear还是log的。

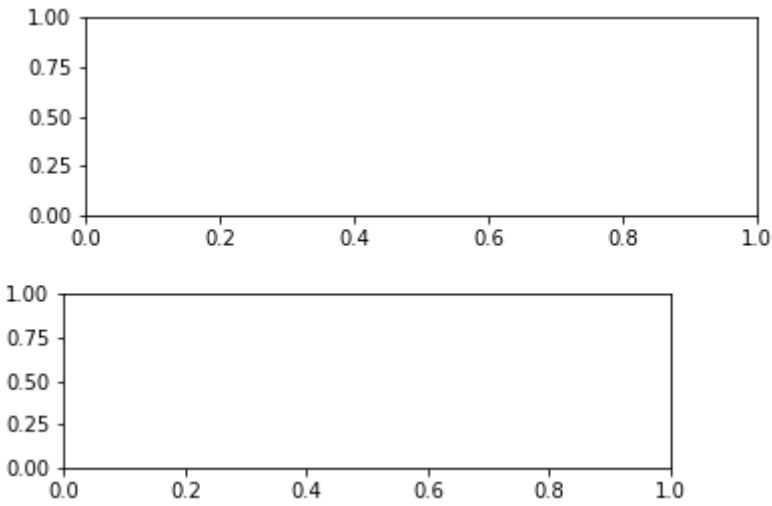
1. Figure容器

matplotlib.figure.Figure是Artist最顶层的container-对象容器，它包含了图表中的所有元素。一张图表的背景就是在Figure.patch的一个矩形Rectangle。

当我们向图表添加Figure.add_subplot()或者Figure.add_axes()元素时，这些都会被添加到Figure.axes列表中。


```
fig = plt.figure()
ax1 = fig.add_subplot(211) # 作一幅2*1的图，选择第1个子图
ax2 = fig.add_axes([0.1, 0.1, 0.7, 0.3]) # 位置参数，四个数分别代表了(left,bottom,width,height)
print(ax1)
print(fig.axes) # fig.axes 中包含了subplot和axes两个实例，刚刚添加的
```

```
AxesSubplot(0.125,0.536818;0.775x0.343182)
[<AxesSubplot:~>, <Axes:~>]
```

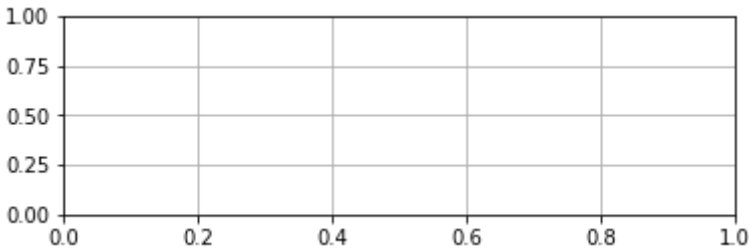


由于Figure维持了current axes，因此你不应该手动的从Figure.axes列表中添加删除元素，而是要通过Figure.add_subplot()、Figure.add_axes()来添加元素，通过Figure.delaxes()来删除元素。但是你可以迭代或者访问Figure.axes中的Axes，然后修改这个Axes的属性。

比如下面的遍历axes里的内容，并且添加网格线：

```
fig = plt.figure()
ax1 = fig.add_subplot(211)

for ax in fig.axes:
    ax.grid(True)
```



Figure也有它自己的text、line、patch、image。你可以直接通过add primitive语句直接添加。但是注意Figure默认的坐标系是以像素为单位，你可能需要转换成figure坐标系：(0,0)表示左下点，(1,1)表示右上点。

Figure容器的常见属性：

- Figure.patch属性：Figure的背景矩形
- Figure.axes属性：一个Axes实例的列表（包括Subplot）
- Figure.images属性：一个FigureImages patch列表
- Figure.lines属性：一个Line2D实例的列表（很少使用）
- Figure.legend属性：一个Figure Legend实例列表（不同于Axes.legend）
- Figure.texts属性：一个Figure Text实例列表

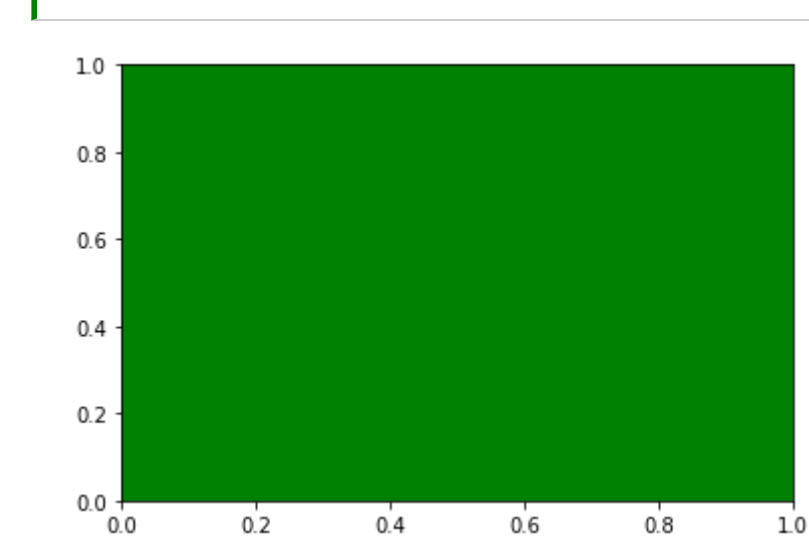
2. Axes容器

matplotlib.axes.Axes是matplotlib的核心。大量的用于绘图的Artist存放在它内部，并且它有许多辅助方法来创建和添加Artist给它自己，而且它也有许多赋值方法来访问和修改这些Artist。

和Figure容器类似，Axes包含了一个patch属性，对于笛卡尔坐标系而言，它是一个Rectangle；对于极坐标而言，它是一个Circle。这个patch属性决定了绘图区域的形状、背景和边框。

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

fig = plt.figure()
ax = fig.add_subplot(111)
rect = ax.patch # axes的patch是一个Rectangle实例
rect.set_facecolor('green')
```

`Axes`有许多方法用于绘图，如`.plot()`、`.text()`、`.hist()`、`.imshow()`等方法用于创建大多数常见的`primitive`(如 `Line2D`, `Rectangle`, `Text`, `Image`等等) 。在`primitives`中已经涉及，不再赘述。

`Subplot`就是一个特殊的`Axes`，其实例是位于网格中某个区域的`Subplot`实例。其实你也可以在任意区域创建 `Axes`，通过`Figure.add_axes([left,bottom,width,height])`来创建一个任意区域的`Axes`，其中`left,bottom,width,height`都是[0—1]之间的浮点数，他们代表了相对于`Figure`的坐标。

你不应该直接通过`Axes.lines`和`Axes.patches`列表来添加图表。因为当创建或添加一个对象到图表中时，`Axes`会做许多自动化的工作:
它会设置`Artist`中`figure`和`axes`的属性，同时默认`Axes`的转换;
它也会检视`Artist`中的数据，来更新数据结构，这样数据范围和呈现方式可以根据作图范围自动调整。

你也可以使用`Axes`的辅助方法`.add_line()`和`.add_patch()`方法来直接添加。

另外`Axes`还包含两个最重要的`Artist container`:

`ax.xaxis`: `XAxis`对象的实例，用于处理x轴tick以及label的绘制
`ax.yaxis`: `YAxis`对象的实例，用于处理y轴tick以及label的绘制
会在下面章节详细说明。

Axes容器的常见属性有：

`artists`: `Artist`实例列表 `patch`: `Axes`所在的矩形实例 `collections`: `Collection`实例 `images`: `Axes`图像 `legends`:
`Legend` 实例 `lines`: `Line2D` 实例 `patches`: `Patch` 实例 `texts`: `Text` 实例 `xaxis`: `matplotlib.axis.XAxis` 实例 `yaxis`:
`matplotlib.axis.YAxis` 实例

3. Axis容器

`matplotlib.axis.Axis`实例处理`tick line`、`grid line`、`tick label`以及`axis label`的绘制，它包括坐标轴上的刻度线、刻度`label`、坐标网格、坐标轴标题。通常你可以独立的配置y轴的左边刻度以及右边的刻度，也可以独立地配置x轴的上边刻度以及下边的刻度。

刻度包括主刻度和次刻度，它们都是`Tick`刻度对象。

`Axis`也存储了用于自适应，平移以及缩放的`data_interval`和`view_interval`。它还有`Locator`实例和`Formatter`实例用于控制刻度线的位置以及刻度`label`。

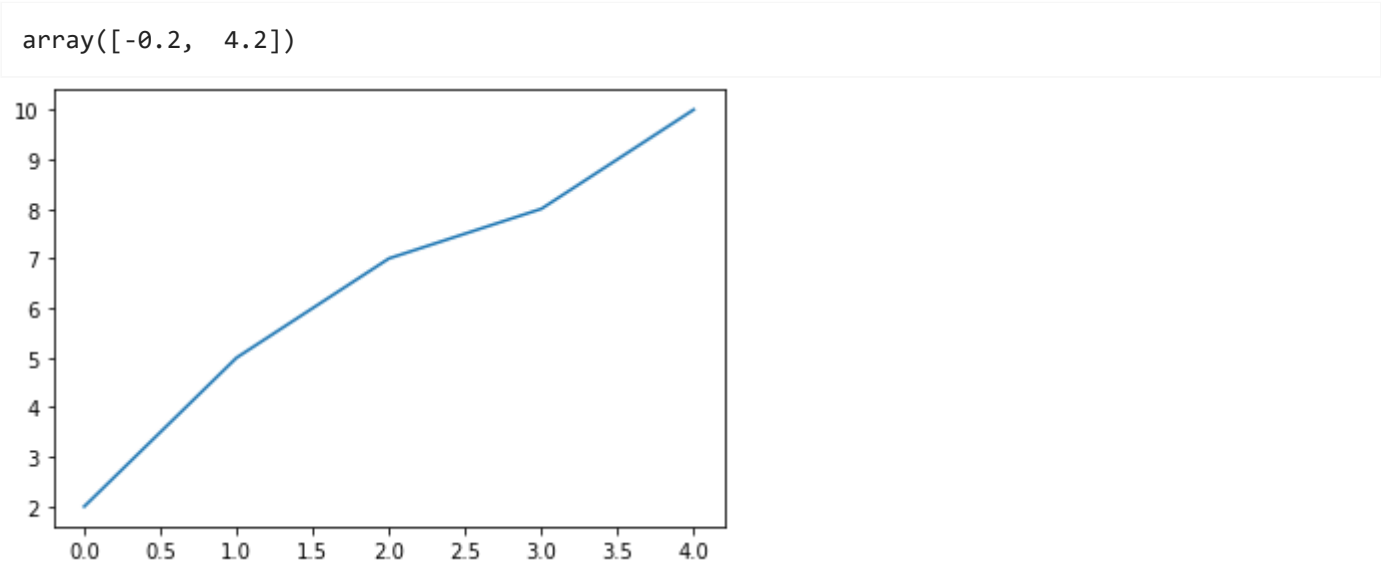
每个`Axis`都有一个`label`属性，也有主刻度列表和次刻度列表。这些`ticks`是`axis.XTick`和`axis.YTick`实例，它们包含着`line primitive`以及`text primitive`用来渲染刻度线以及刻度文本。

刻度是动态创建的，只有在需要创建的时候才创建（比如缩放的时候）。`Axis`也提供了一些辅助方法来获取刻度文本、刻度线位置等等：
常见的如下：

```
# 不用print, 直接显示结果
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

fig, ax = plt.subplots()
x = range(0,5)
y = [2,5,7,8,10]
plt.plot(x, y, '-')
```

axis = ax.xaxis # axis为X轴对象
axis.get_ticklocs() # 获取刻度线位置
axis.get_ticklabels() # 获取刻度Label列表(一个Text实例的列表)。 可以通过minor=True/False关键字
参数控制输出minor还是major的tick Label。
axis.get_ticklines() # 获取刻度线列表(一个Line2D实例的列表)。 可以通过minor=True/False关键字参
数控制输出minor还是major的tick Line。
axis.get_data_interval()# 获取轴刻度间隔
axis.get_view_interval()# 获取轴视角(位置)的间隔



下面的例子展示了如何调整一些轴和刻度的属性(忽略美观度，仅作调整参考)：

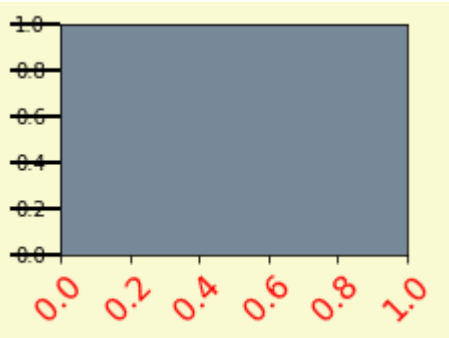
```
fig = plt.figure() # 创建一个新图表
rect = fig.patch # 矩形实例并将其设为黄色
rect.set_facecolor('lightgoldenrodyellow')

ax1 = fig.add_axes([0.1, 0.3, 0.4, 0.4]) # 创一个axes对象，从(0.1,0.3)的位置开始，宽和高都为0.4，
rect = ax1.patch # ax1的矩形设为灰色
rect.set_facecolor('lightslategray')
```

for label in ax1.xaxis.get_ticklabels():
 # 调用x轴刻度标签实例，是一个text实例
 label.set_color('red') # 颜色
 label.set_rotation(45) # 旋转角度
 label.set_fontsize(16) # 字体大小

for line in ax1.yaxis.get_ticklines():
 # 调用y轴刻度线条实例，是一个Line2D实例
 line.set_color('green') # 颜色
 line.set_markersize(25) # marker大小
 line.set_markedewidth(2)# marker粗细

plt.show()



4. Tick容器

matplotlib.axis.Tick是从Figure到Axes到Axis到Tick中最末端的容器对象。
Tick包含了tick、grid line实例以及对应的label。

所有的这些都可以通过Tick的属性获取，常见的tick属性有

- Tick.tick1line: Line2D实例
- Tick.tick2line: Line2D实例
- Tick.gridline: Line2D实例
- Tick.label1: Text实例

Tick.label2: Text实例

y轴分为左右两个，因此tick1对应左侧的轴；tick2对应右侧的轴。
x轴分为上下两个，因此tick1对应下侧的轴；tick2对应上侧的轴。

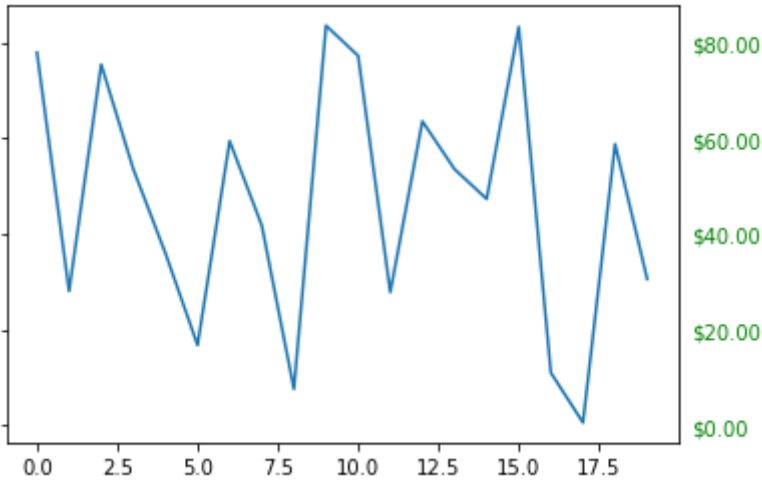
下面的例子展示了，如何将Y轴右边轴设为主轴，并将标签设置为美元符号且为绿色：

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib

fig, ax = plt.subplots()
ax.plot(100*np.random.rand(20))

# 设置ticker的显示格式
formatter = matplotlib.ticker.FormatStrFormatter('%$1.2f')
ax.yaxis.set_major_formatter(formatter)

# 设置ticker的参数，右侧为主轴，颜色为绿色
ax.yaxis.set_tick_params(which='major', labelcolor='green',
                        labelleft=False, labelright=True);
```



参考资料

- [1. matplotlib设计的基本逻辑](#)
- [2. matplotlib.artist api](#)
- [3. matplotlib官方教程](#)
- [4. AI算法工程师手册](#)

By Datawhale
© Copyright © Copyright 2020.

第三回：布局格式定方圆

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus'] = False #用来正常显示负号
```

[Contents](#)[二、子图](#)[Print to PDF](#)[1. 使用 plt.subplots 绘制均匀状态下的子图](#)[2. 使用 GridSpec 绘制非均匀子图](#)[二、子图上的方法](#)[参考资料](#)

一、子图

1. 使用 plt.subplots 绘制均匀状态下的子图

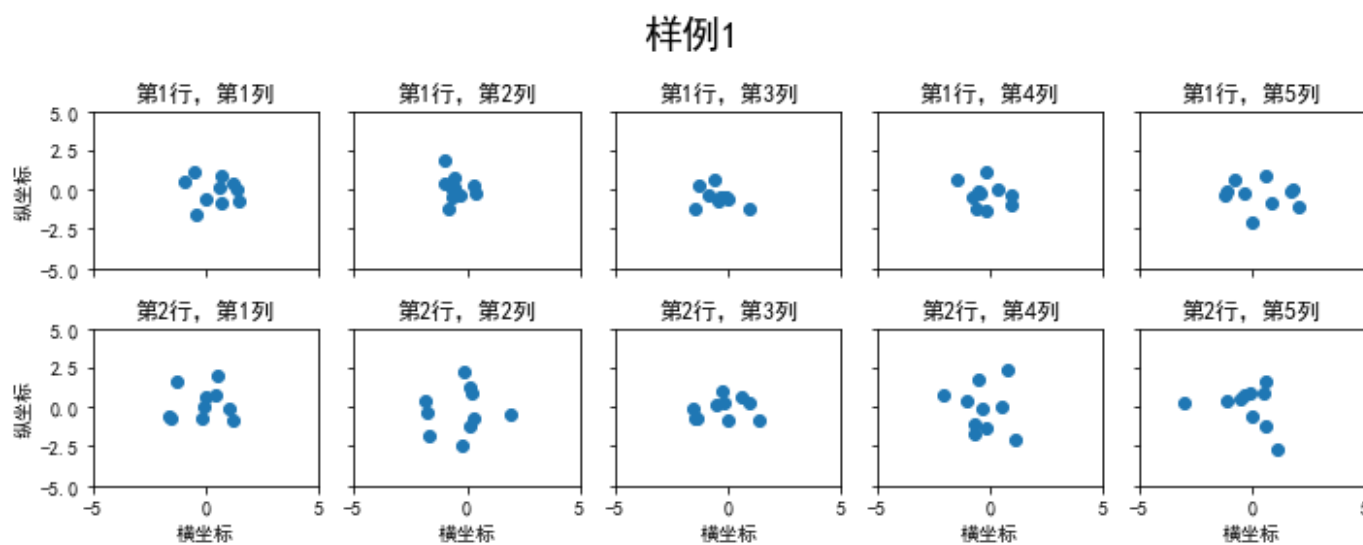
返回元素分别是画布和子图构成的列表，第一个数字为行，第二个为列

`figsize` 参数可以指定整个画布的大小

`sharex` 和 `sharey` 分别表示是否共享横轴和纵轴刻度

`tight_layout` 函数可以调整子图的相对大小使字符不会重叠

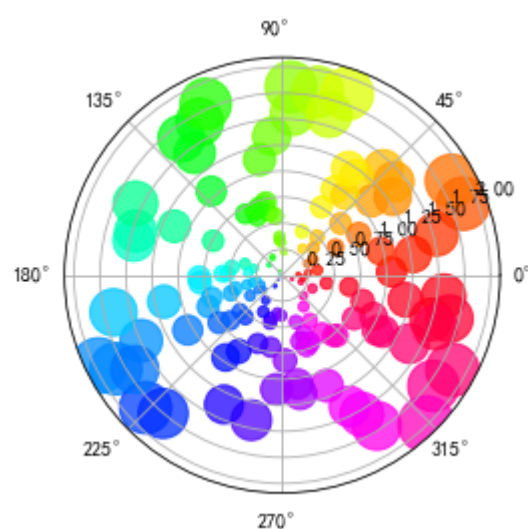
```
fig, axs = plt.subplots(2, 5, figsize=(10, 4), sharex=True, sharey=True)
fig.suptitle('样例1', size=20)
for i in range(2):
    for j in range(5):
        axs[i][j].scatter(np.random.randn(10), np.random.randn(10))
        axs[i][j].set_title('第%d行, 第%d列'%(i+1,j+1))
        axs[i][j].set_xlim(-5,5)
        axs[i][j].set_ylim(-5,5)
        if i==1: axs[i][j].set_xlabel('横坐标')
        if j==0: axs[i][j].set_ylabel('纵坐标')
fig.tight_layout()
```



除了常规的直角坐标系，也可以通过`projection`方法创建极坐标系下的图表

```
N = 150
r = 2 * np.random.rand(N)
theta = 2 * np.pi * np.random.rand(N)
area = 200 * r**2
colors = theta

plt.subplot(projection='polar')
plt.scatter(theta, r, c=colors, s=area, cmap='hsv', alpha=0.75);
```



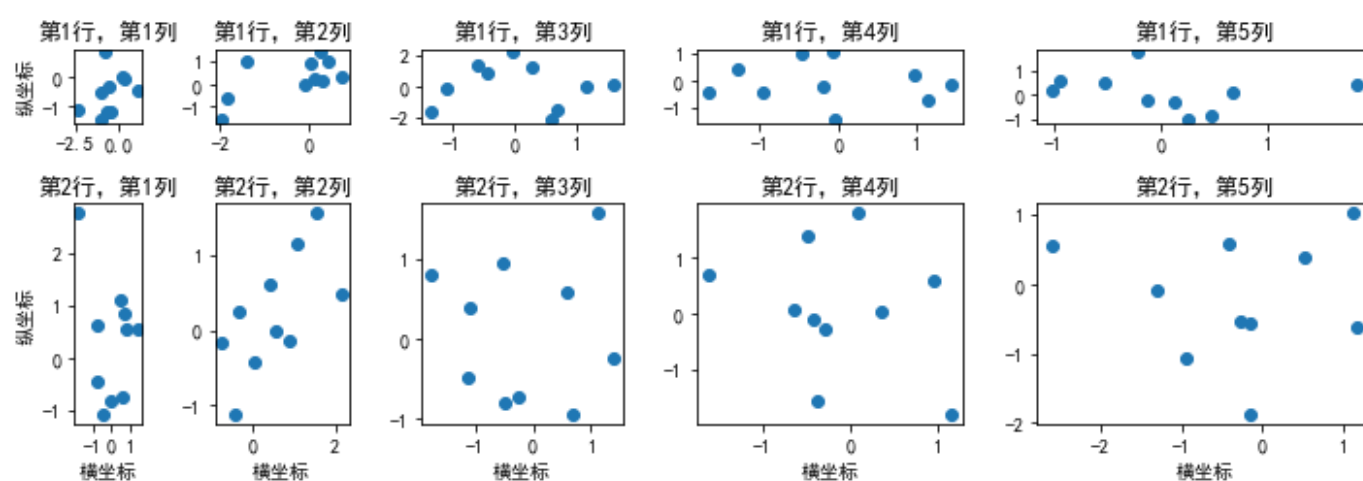
2. 使用 GridSpec 绘制非均匀子图

所谓非均匀包含两层含义，第一是指图的比例大小不同但没有跨行或跨列，第二是指图为跨列或跨行状态

利用 `add_gridspec` 可以指定相对宽度比例 `width_ratios` 和相对高度比例参数 `height_ratios`

```
fig = plt.figure(figsize=(10, 4))
spec = fig.add_gridspec(nrows=2, ncols=5, width_ratios=[1,2,3,4,5], height_ratios=[1,3])
fig.suptitle('样例2', size=20)
for i in range(2):
    for j in range(5):
        ax = fig.add_subplot(spec[i, j])
        ax.scatter(np.random.randn(10), np.random.randn(10))
        ax.set_title('第%d行, 第%d列'%(i+1,j+1))
        if i==1: ax.set_xlabel('横坐标')
        if j==0: ax.set_ylabel('纵坐标')
fig.tight_layout()
```

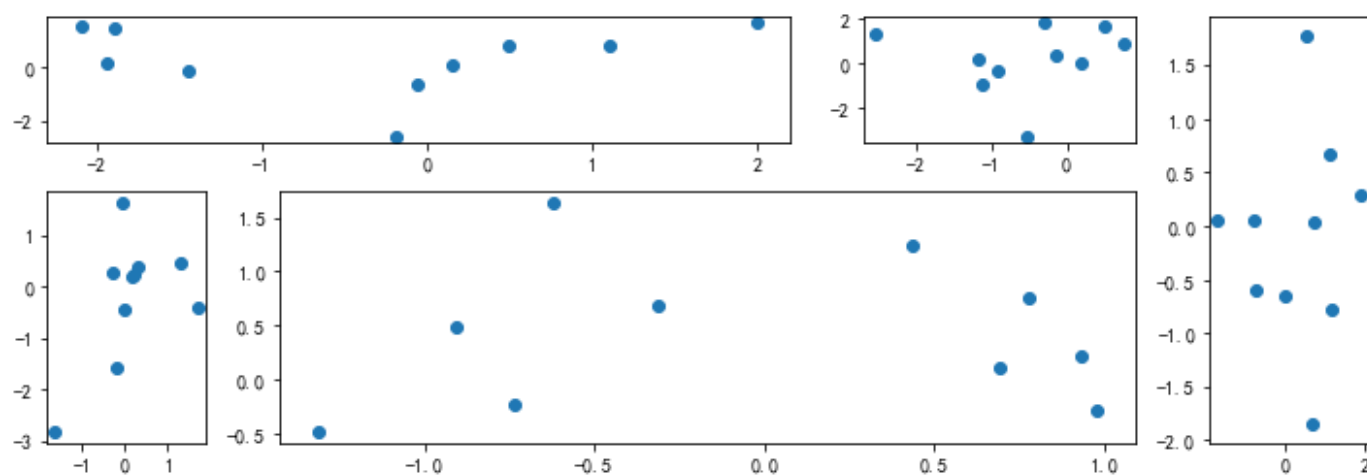
样例2



在上面的例子中出现了 `spec[i, j]` 的用法，事实上通过切片就可以实现子图的合并而达到跨图的共能

```
fig = plt.figure(figsize=(10, 4))
spec = fig.add_gridspec(nrows=2, ncols=6, width_ratios=[2,2.5,3,1,1.5,2], height_ratios=[1,2])
fig.suptitle('样例3', size=20)
# sub1
ax = fig.add_subplot(spec[0, :3])
ax.scatter(np.random.randn(10), np.random.randn(10))
# sub2
ax = fig.add_subplot(spec[0, 3:5])
ax.scatter(np.random.randn(10), np.random.randn(10))
# sub3
ax = fig.add_subplot(spec[:, 5])
ax.scatter(np.random.randn(10), np.random.randn(10))
# sub4
ax = fig.add_subplot(spec[1, 0])
ax.scatter(np.random.randn(10), np.random.randn(10))
# sub5
ax = fig.add_subplot(spec[1, 1:5])
ax.scatter(np.random.randn(10), np.random.randn(10))
fig.tight_layout()
```

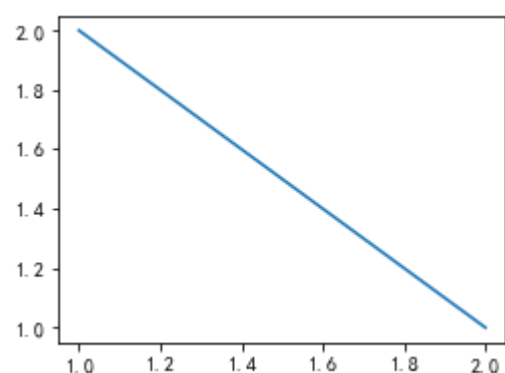
样例3



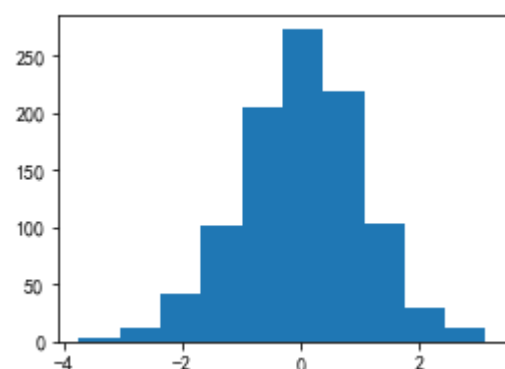
二、子图上的方法

在 `ax` 对象上定义了和 `plt` 类似的图形绘制函数，常用的有： `plot`, `hist`, `scatter`, `bar`, `barh`, `pie`

```
fig, ax = plt.subplots(figsize=(4,3))
ax.plot([1,2],[2,1]);
```

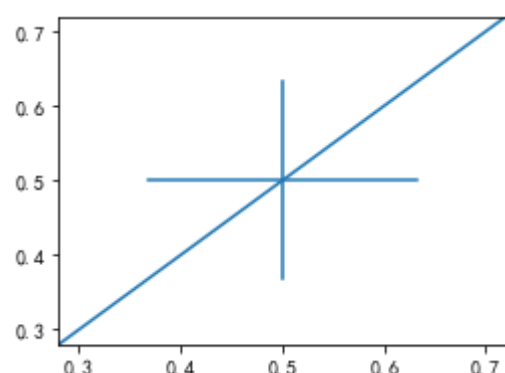


```
fig, ax = plt.subplots(figsize=(4,3))
ax.hist(np.random.randn(1000));
```



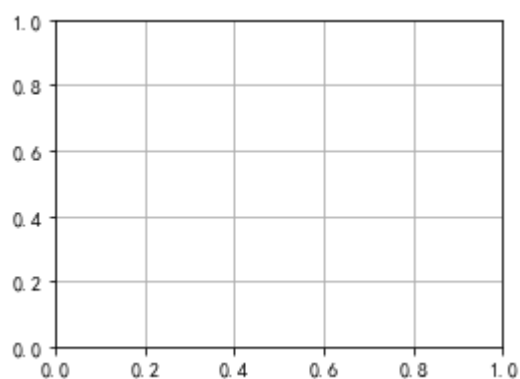
常用直线的画法为： `axhline`, `axvline`, `axline` （水平、垂直、任意方向）

```
fig, ax = plt.subplots(figsize=(4,3))
ax.axhline(0.5,0.2,0.8)
ax.axvline(0.5,0.2,0.8)
ax.axline([0.3,0.3],[0.7,0.7]);
```



使用 `grid` 可以加灰色网格

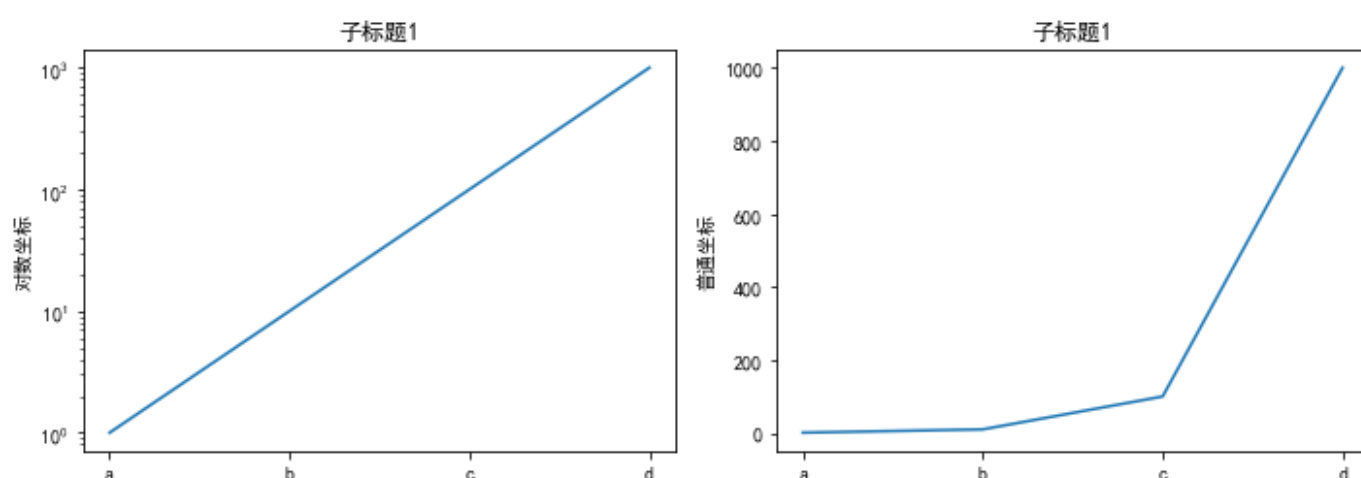
```
fig, ax = plt.subplots(figsize=(4,3))
ax.grid(True)
```



使用 `set_xscale`, `set_title`, `set_xlabel` 分别可以设置坐标轴的规格（指对数坐标等）、标题、轴名

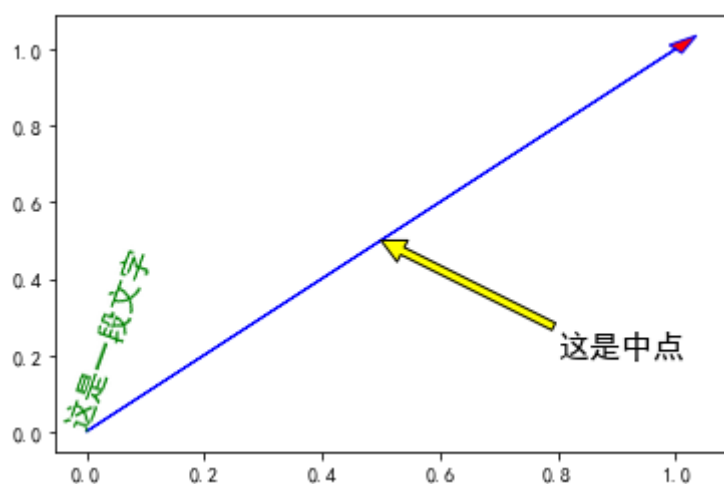
```
fig, axs = plt.subplots(1, 2, figsize=(10, 4))
fig.suptitle('大标题', size=20)
for j in range(2):
    axs[j].plot(list('abcd'), [10**i for i in range(4)])
    if j==0:
        axs[j].set_yscale('log')
        axs[j].set_title('子标题1')
        axs[j].set_ylabel('对数坐标')
    else:
        axs[j].set_title('子标题1')
        axs[j].set_ylabel('普通坐标')
fig.tight_layout()
```

大标题

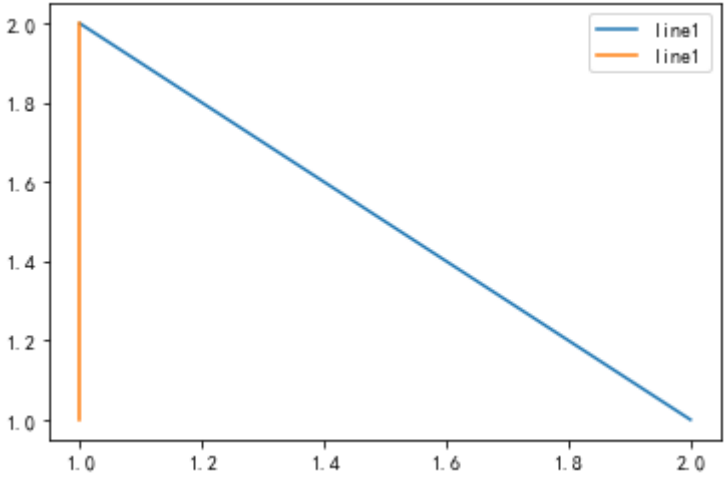


与一般的 `plt` 方法类似，`legend`, `annotate`, `arrow`, `text` 对象也可以进行相应的绘制

```
fig, ax = plt.subplots()
ax.arrow(0, 0, 1, 1, head_width=0.03, head_length=0.05, facecolor='red', edgecolor='blue')
ax.text(x=0, y=0, s='这是一段文字', fontsize=16, rotation=70, rotation_mode='anchor',
color='green')
ax.annotate('这是中点', xy=(0.5, 0.5), xytext=(0.8, 0.2), arrowprops=dict(facecolor='yellow',
edgecolor='black'), fontsize=16);
```



```
fig, ax = plt.subplots()
ax.plot([1,2],[2,1],label="line1")
ax.plot([1,1],[1,2],label="line1")
ax.legend(loc=1);
```

其中，图例的 `loc` 参数如下：

string	code
best	0
upper right	1
upper left	2
lower left	3
lower right	4
right	5
center left	6
center right	7
lower center	8
upper center	9
center	10

参考资料

[1.matplotlib官网布局使用指南](#)

第四回：文字图例尽眉目

一、Figure和Axes上的文本

Matplotlib具有广泛的文本支持，包括对数学表达式的支持、对栅格和矢量输出的TrueType支持、具有任意旋转的换行分隔文本以及Unicode支持。

下面的命令是介绍了通过pyplot API和objected-oriented API分别创建文本的方式。

pyplot API	OO API	description
text	text	在 Axes的任意位置添加text
title	set_title	在 Axes添加title
figtext	text	在Figure的任意位置添加text.
suptitle	suptitle	在 Figure添加title
xlabel	set_xlabel	在Axes的x-axis添加label
ylabel	set_ylabel	在Axes的y-axis添加label

1.text

pyplot API: matplotlib.pyplot.text(x, y, s, fontdict=None, **kwargs)

OO API:Axes.text(self, x, y, s, fontdict=None, **kwargs)

参数：此方法接受以下描述的参数：

s:此参数是要添加的文本。

xy:此参数是放置文本的点(x, y)。

fontdict:此参数是一个可选参数，并且是一个覆盖默认文本属性的字典。如果fontdict为None，则由rcParams确定默认值。

返回值：此方法返回作为创建的文本实例的文本。

fontdict主要参数具体介绍，更多参数请参考[官网说明](#)：

Contents

Print to PDF ▶

- 一、Figure和Axes上的文本
 - [1.text](#)
 - [2.title和set_title](#)
 - [3.figtext和text](#)
 - [4.suptitle](#)
 - [5.xlabel和ylabel](#)
 - [6.字体的属性设置](#)
 - [7.数学表达式](#)
- 二、Tick上的文本
 - [1.简单模式](#)
 - [2.Tick Locators and Formatters](#)
- 三、legend（图例）
- [参考资料](#)

Property	Description
alpha	float or None 该参数指透明度，越接近0越透明，越接近1越不透明
backgroundcolor	color
bbox	dict with properties for patches.FancyBboxPatch 这个是用来设置text周围的box外框
color or c	color 指的是字体的颜色
fontfamily or family	{FONTNAME, 'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'} 该参数指的是字体的类型
fontproperties or font or font_properties	font_manager.FontProperties or str or pathlib.Path
fontsize or size	float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'} 该参数指字体大小
fontstretch or stretch	{a numeric value in range 0-1000, 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded', 'ultra-expanded'} 该参数是指从字体中选择正常、压缩或扩展的字体
fontstyle or style	{'normal', 'italic', 'oblique'} 该参数是指字体的样式是否倾斜等
fontweight or weight	{a numeric value in range 0-1000, 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}
horizontalalignment or ha	{'center', 'right', 'left'} 该参数是指选择文本左对齐右对齐还是居中对齐
label	object
linespacing	float (multiple of font size)
position	(float, float)
rotation	float or {'vertical', 'horizontal'} 该参数是指text逆时针旋转的角度，“horizontal”等于0，“vertical”等于90。我们可以根据自己设定来选择合适角度
verticalalignment or va	{'center', 'top', 'bottom', 'baseline', 'center_baseline'}

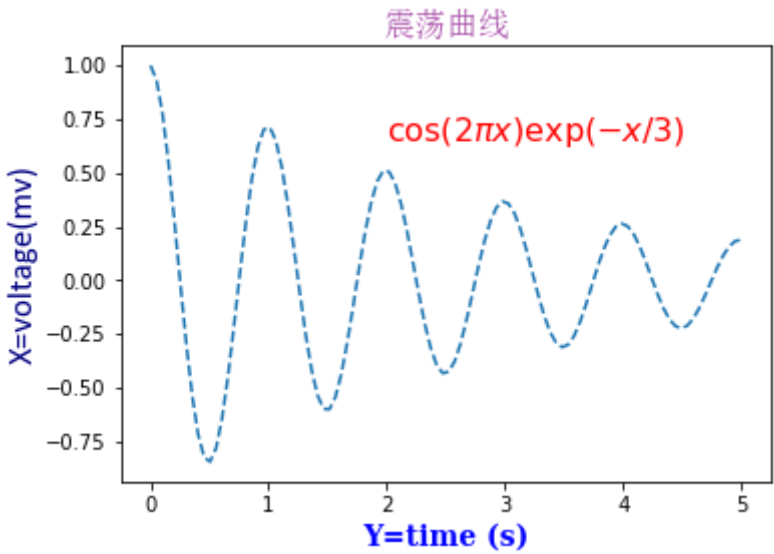
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.font_manager import FontProperties
import numpy as np
```

```
#fontdict学习的案例
#学习的过程中请尝试更换不同的fontdict字典的内容，以便于更好的掌握
#-----设置字体样式，分别是字体，颜色，宽度，大小
font1 = {'family': 'SimSun', #华文楷体
        'alpha':0.7, #透明度
        'color': 'purple',
        'weight': 'normal',
        'size': 16,
        }
font2 = {'family': 'Times New Roman',
        'color': 'red',
        'weight': 'normal',
        'size': 16,
        }
font3 = {'family': 'serif',
        'color': 'blue',
        'weight': 'bold',
        'size': 14,
        }
font4 = {'family': 'Calibri',
        'color': 'navy',
        'weight': 'normal',
        'size': 17,
        }

#-----四种不同字体显示风格-----

#-----建立函数-----
x = np.linspace(0.0, 5.0, 100)
y = np.cos(2*np.pi*x) * np.exp(-x/3)
#-----绘制图像，添加标注-----
plt.plot(x, y, '--')
plt.title('震荡曲线', fontdict=font1)
#-----添加文本在指定的坐标处-----
plt.text(2, 0.65, r'$\cos(2 \pi x) \exp(-x/3)$', fontdict=font2)
#-----设置坐标标签
plt.xlabel('Y=time (s)', fontdict=font3)
plt.ylabel('X=voltage(mv)', fontdict=font4)

# 调整图像边距
plt.subplots_adjust(left=0.15)
plt.show()
```



2.title和set_title

pyplot API: matplotlib.pyplot.title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)

OO API: Axes.set_title(self, label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)

该命令是用来设置axes的标题。

参数：此方法接受以下描述的参数:

label: str, 此参数是要添加的文本

fontdict: dict, 此参数是控制title文本的外观，默认fontdict如下：

```
{'fontsize': rcParams['axes.titlesize'],
'fontweight': rcParams['axes.titleweight'],
'color': rcParams['axes.titlecolor'],
'verticalalignment': 'baseline',
'horizontalalignment': loc}
```

loc: str, {'center', 'left', 'right'}默认为center

pad: float,该参数是指标题偏离图表顶部的距离，默认为6。

y: float, 该参数是title所在axes垂向的位置。默认值为1，即title位于axes的顶部。

kwargs: 该参数是指可以设置的一些奇特文本的属性。

返回值：此方法返回作为创建的title实例的文本。

3.figtext和text

pyplot API: matplotlib.pyplot.figtext(x, y, s, fontdict=None, **kwargs)

OO API:text(self, x, y, s, fontdict=None,**kwargs)

参数：此方法接受以下描述的参数:

x,y: float, 此参数是指在figure中放置文本的位置。一般取值是在[0,1]范围内。使用transform关键字可以更改坐标系。

s:str,此参数是指文本

fontdict:dict,此参数是一个可选参数，并且是一个覆盖默认文本属性的字典。如果fontdict为None，则由rcParams确定默认值。

返回值：此方法返回作为创建的文本实例的文本。

4.suptitle

pyplot API: matplotlib.pyplot.suptitle(t, **kwargs)

OO API:suptitle(self, t, **kwargs)

参数：此方法接受以下描述的参数:

t: str,标题的文本

x: float,默认值是0.5.该参数是指文本在figure坐标系下的x坐标

y: float,默认值是0.95.该参数是指文本在figure坐标系下的y坐标

horizontalalignment, ha:该参数是指选择文本水平对齐方式，有三种选择{'center', 'left', 'right'}，默认值是 'center'

verticalalignment, va: 该参数是指选择文本垂直对齐方式，有四种选择{'top', 'center', 'bottom', 'baseline'}，默认值是 'top'

fontsize, size: 该参数是指文本的大小，默认值是依据rcParams的设置： rcParams["figure.titlesize"] (default: 'large')

fontweight, weight: 该参数是用来设置字重。默认值是依据rcParams的设置： rcParams["figure.titleweight"] (default: 'normal')

fontproperties:None or dict,该参数是可选参数，如果该参数被指定，字体的大小将从该参数的默认值中提取。

返回值：此方法返回作为创建的title实例的文本。

5.xlabel和ylabel

pyplot API: matplotlib.pyplot.xlabel(xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)

matplotlib.pyplot.ylabel(ylabel, fontdict=None, labelpad=None,*, loc=None, **kwargs)

OO API: Axes.set_xlabel(self, xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)

Axes.set_ylabel(self, ylabel, fontdict=None, labelpad=None,*, loc=None, **kwargs)

参数：此方法接受以下描述的参数:

xlabel或者ylabel: label的文本

labelpad:设置label距离轴(axis)的距离

loc: {'left', 'center', 'right'},默认为center

**kwargs:[文本属性](#)

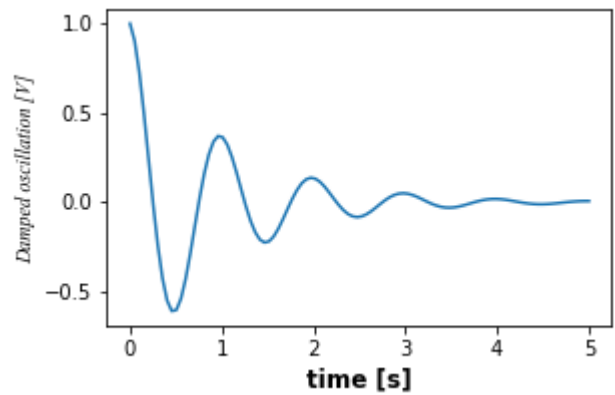
返回值：此方法返回作为创建的xlabel和ylabel实例的文本。

```
#文本属性的输入一种是通过**kwargs属性这种方式，一种是通过操作
matplotlib.font_manager.FontProperties 方法
#该案例中对于x_label采用**kwargs调整字体属性， y_label则采用
matplotlib.font_manager.FontProperties 方法调整字体属性
#该链接是FontProperties方法的介绍
https://matplotlib.org/api/font_manager_api.html#matplotlib.font_manager.FontProperties
x1 = np.linspace(0.0, 5.0, 100)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)

font = FontProperties()
font.set_family('serif')
font.set_name('Times New Roman')
font.set_style('italic')

fig, ax = plt.subplots(figsize=(5, 3))
fig.subplots_adjust(bottom=0.15, left=0.2)
ax.plot(x1, y1)
ax.set_xlabel('time [s]', fontsize='large', fontweight='bold')
ax.set_ylabel('Damped oscillation [V]', fontproperties=font)

plt.show()
```



6.字体的属性设置

字体设置一般有全局字体设置和自定义局部字体设置两种方法。

```
#首先可以查看matplotlib所有可用的字体
from matplotlib import font_manager
font_family = font_manager.fontManager.ttflist
font_name_list = [i.name for i in font_family]
for font in font_name_list[:10]:
    print(f'{font}\n')
```

```
STIXGeneral
DejaVu Sans Mono
DejaVu Sans
DejaVu Sans Mono
STIXGeneral
DejaVu Serif Display
STIXSizeOneSym
DejaVu Serif
STIXSizeFourSym
STIXGeneral
```

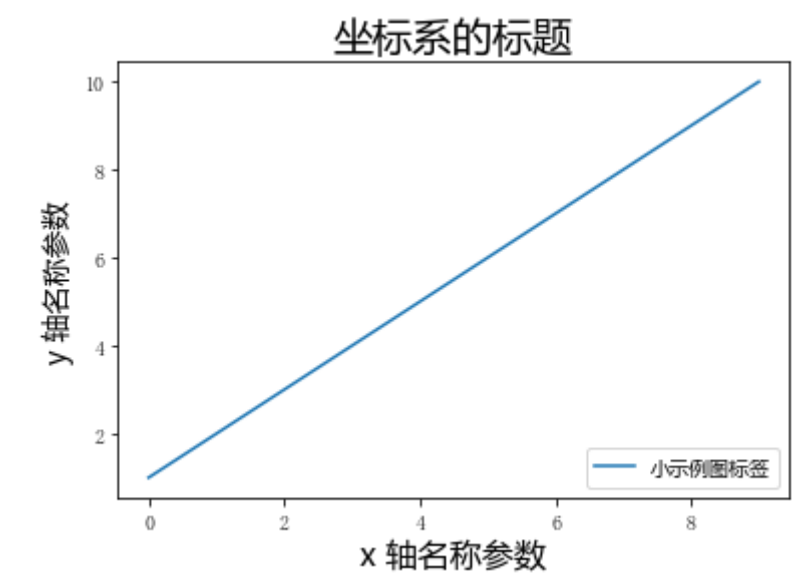
[为方便在图中加入合适的字体，可以尝试了解中文字体的英文名称,该链接告诉了常用中文的英文名称](#)

```
#该block讲述如何在matplotlib里面，修改字体默认属性，完成全局字体的更改。
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimSun'] # 指定默认字体为新宋体。
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像时 负号 '-' 显示为方块和报错的问题。
```

```
#局部字体的修改方法1
import matplotlib.pyplot as plt
import matplotlib.font_manager as fontmg

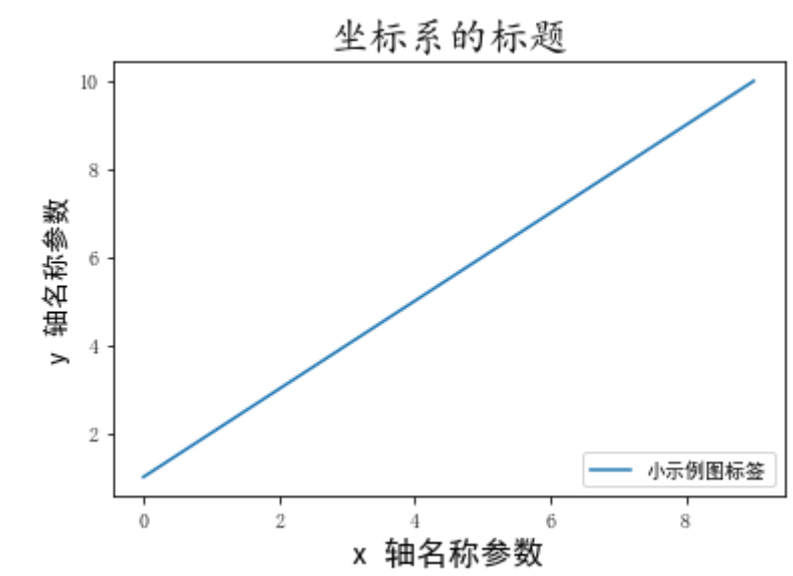
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
plt.plot(x, label='小示例图标签')

# 直接用字体的名字。
plt.xlabel('x 轴名称参数', fontproperties='Microsoft YaHei', fontsize=16) # 设置x轴名称，采用微软雅黑字体
plt.ylabel('y 轴名称参数', fontproperties='Microsoft YaHei', fontsize=14) # 设置Y轴名称
plt.title('坐标系的标题', fontproperties='Microsoft YaHei', fontsize=20) # 设置坐标系标题的字体
plt.legend(loc='lower right', prop={"family": 'Microsoft YaHei'}, fontsize=10); # 小示例图的字体设置
```



```
#局部字体的修改方法2
import matplotlib.pyplot as plt
import matplotlib.font_manager as fontmg

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
plt.plot(x, label='小示例图标签')
#fname为你系统中的字体库路径
my_font1 = fontmg.FontProperties(fname=r'C:\Windows\Fonts\simhei.ttf')      # 读取系统中的 黑体
字体。
my_font2 = fontmg.FontProperties(fname=r'C:\Windows\Fonts\simkai.ttf')      # 读取系统中的 楷体
字体。
# fontproperties 设置中文显示, fontsize 设置字体大小
plt.xlabel('x 轴名称参数', fontproperties=my_font1, fontsize=16)          # 设置x轴名称
plt.ylabel('y 轴名称参数', fontproperties=my_font1, fontsize=14)          # 设置y轴名称
plt.title('坐标系的标题', fontproperties=my_font2, fontsize=20)          # 标题的字体设置
plt.legend(loc='lower right', prop=my_font1, fontsize=10);                # 小示例图的字体设置
```

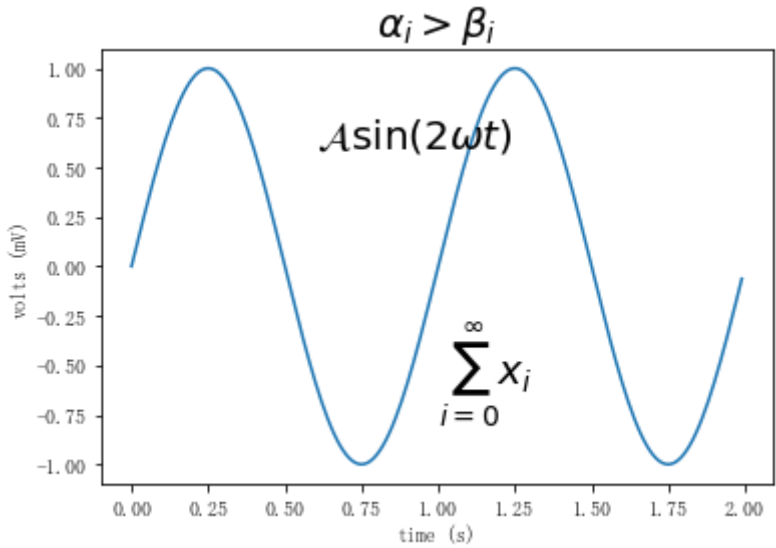


7.数学表达式

在文本标签中使用数学表达式。有关MathText的概述，请参见 [写数学表达式](#),但由于数学表达式的练习想必我们都在markdown语法和latex语法中多少有接触，故在此不继续展开，愿意深入学习的可以参看官方文档.下面是一个官方案例，供参考了解。

```
import numpy as np
import matplotlib.pyplot as plt
t = np.arange(0.0, 2.0, 0.01)
s = np.sin(2*np.pi*t)

plt.plot(t, s)
plt.title(r'$\alpha_i > \beta_i$', fontsize=20)
plt.text(1, -0.6, r'$\sum_{i=0}^{\infty} x_i$', fontsize=20)
plt.text(0.6, 0.6, r'$\mathcal{A}\mathrm{sin}(2 \omega t)$',
         fontsize=20)
plt.xlabel('time (s)')
plt.ylabel('volts (mV)')
plt.show()
```

```
#这是对前七节学习内容的总结案例
import matplotlib
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111)
fig.subplots_adjust(top=0.85)

# 分别在figure和subplot上设置title
fig.suptitle('bold figure suprtitle', fontsize=14, fontweight='bold')
ax.set_title('axes title')

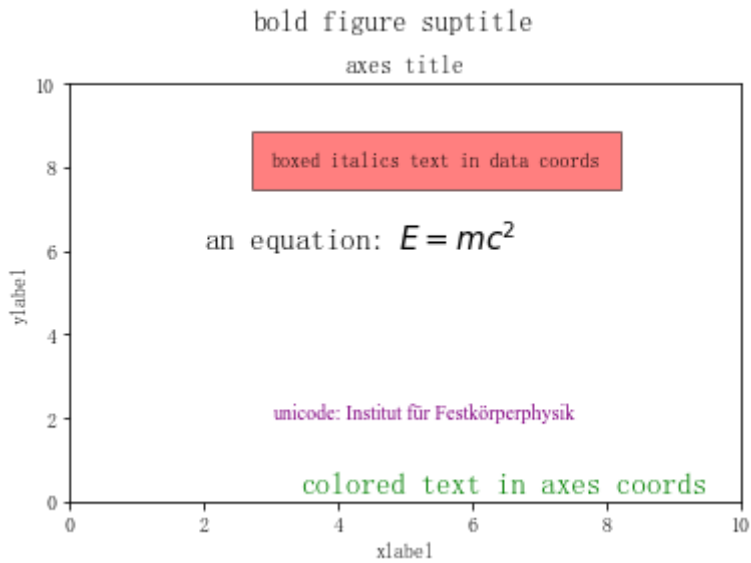
ax.set_xlabel('xlabel')
ax.set_ylabel('ylabel')

# 设置x-axis和y-axis的范围都是[0, 10]
ax.axis([0, 10, 0, 10])

ax.text(3, 8, 'boxed italics text in data coords', style='italic',
        bbox={'facecolor': 'red', 'alpha': 0.5, 'pad': 10})

ax.text(2, 6, r'an equation: $E=mc^2$', fontsize=15)
font1 = {'family': 'Times New Roman',
         'color': 'purple',
         'weight': 'normal',
         'size': 10,
        }
ax.text(3, 2, 'unicode: Institut für Festkörperphysik', fontdict=font1)
ax.text(0.95, 0.01, 'colored text in axes coords',
        verticalalignment='bottom', horizontalalignment='right',
        transform=ax.transAxes,
        color='green', fontsize=15)

plt.show()
```



二、Tick上的文本

设置tick（刻度）和ticklabel（刻度标签）也是可视化中经常需要操作的步骤，matplotlib既提供了自动生成刻度和刻度标签的模式（默认状态），同时也提供了许多让使用者灵活设置的方式。

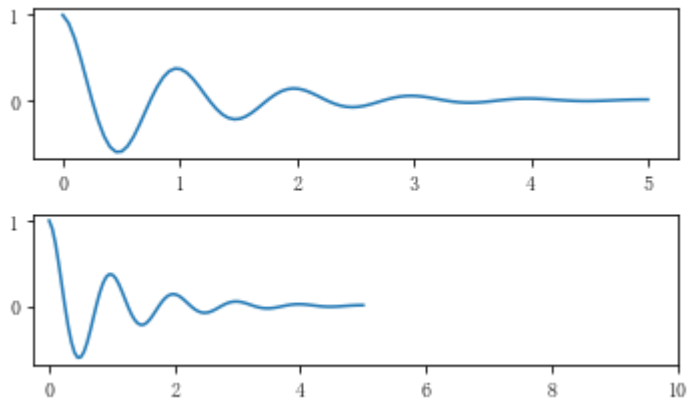
1.简单模式

可以使用axis的set_ticks方法手动设置标签位置，使用axis的set_ticklabels方法手动设置标签格式

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
x1 = np.linspace(0.0, 5.0, 100)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
```

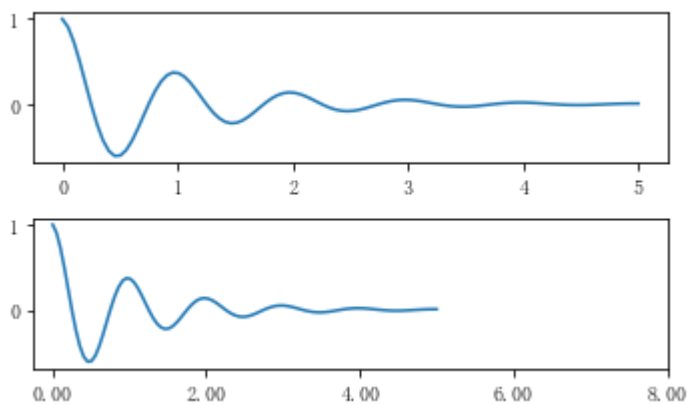
使用`axis`的`set_ticks`方法手动设置标签位置的例子，该案例中由于`tick`设置过大，所以会影响绘图美观，不建议用此方式进行设置`tick`

```
fig, axs = plt.subplots(2, 1, figsize=(5, 3), tight_layout=True)
axs[0].plot(x1, y1)
axs[1].plot(x1, y1)
axs[1].xaxis.set_ticks(np.arange(0., 10.1, 2.))
plt.show()
```



使用`axis`的`set_ticklabels`方法手动设置标签格式的例子

```
fig, axs = plt.subplots(2, 1, figsize=(5, 3), tight_layout=True)
axs[0].plot(x1, y1)
axs[1].plot(x1, y1)
ticks = np.arange(0., 8.1, 2.)
tickla = [f'{tick:1.2f}' for tick in ticks]
axs[1].xaxis.set_ticks(ticks)
axs[1].xaxis.set_ticklabels(tickla)
plt.show()
```



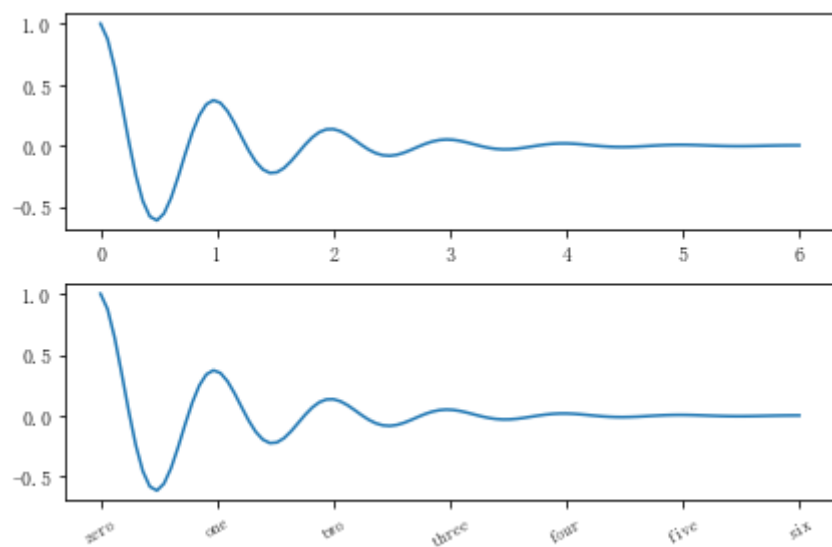
#一般绘图时会自动创建刻度，而如果通过上面的例子使用`set_ticks`创建刻度可能会导致`tick`的范围与所绘制图形的范围不一致的问题。

#所以在下面的案例中，`axs[1]`中`set_xtick`的设置要与数据范围所对应，然后再通过`set_xticklabels`设置刻度所对应的标签

```
import numpy as np
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2, 1, figsize=(6, 4), tight_layout=True)
x1 = np.linspace(0.0, 6.0, 100)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
axs[0].plot(x1, y1)
axs[0].set_xticks([0,1,2,3,4,5,6])

axs[1].plot(x1, y1)
axs[1].set_xticks([0,1,2,3,4,5,6])#要将x轴的刻度放在数据范围中的哪些位置
axs[1].set_xticklabels(['zero','one', 'two', 'three', 'four', 'five','six'],#设置刻度对应的标签
                        rotation=30, fontsize='small')#rotation选项设定x刻度标签倾斜30度。
axs[1].xaxis.set_ticks_position('bottom')#set_ticks_position()方法是用来设置刻度所在的位置，常用的参数有bottom、top、both、none
print(axs[1].xaxis.get_ticklines())
plt.show()
```

```
<a list of 14 Line2D ticklines objects>
```



2.Tick Locators and Formatters

除了上述的简单模式，还可以使用 **Tick Locators and Formatters** 完成对于刻度位置和刻度标签的设置。其中 [Axis.set_major_locator](#) 和 [Axis.set_minor_locator](#) 方法用来设置标签的位置，[Axis.set_major_formatter](#) 和 [Axis.set_minor_formatter](#) 方法用来设置标签的格式。这种方式的好处是不用显式地列举出刻度值列表。

`set_major_formatter` 和 `set_minor_formatter` 这两个 `formatter` 格式命令可以接收字符串格式（`matplotlib.ticker.StrMethodFormatter`）或函数参数（`matplotlib.ticker.FuncFormatter`）来设置刻度值的格式。

a) Tick Formatters

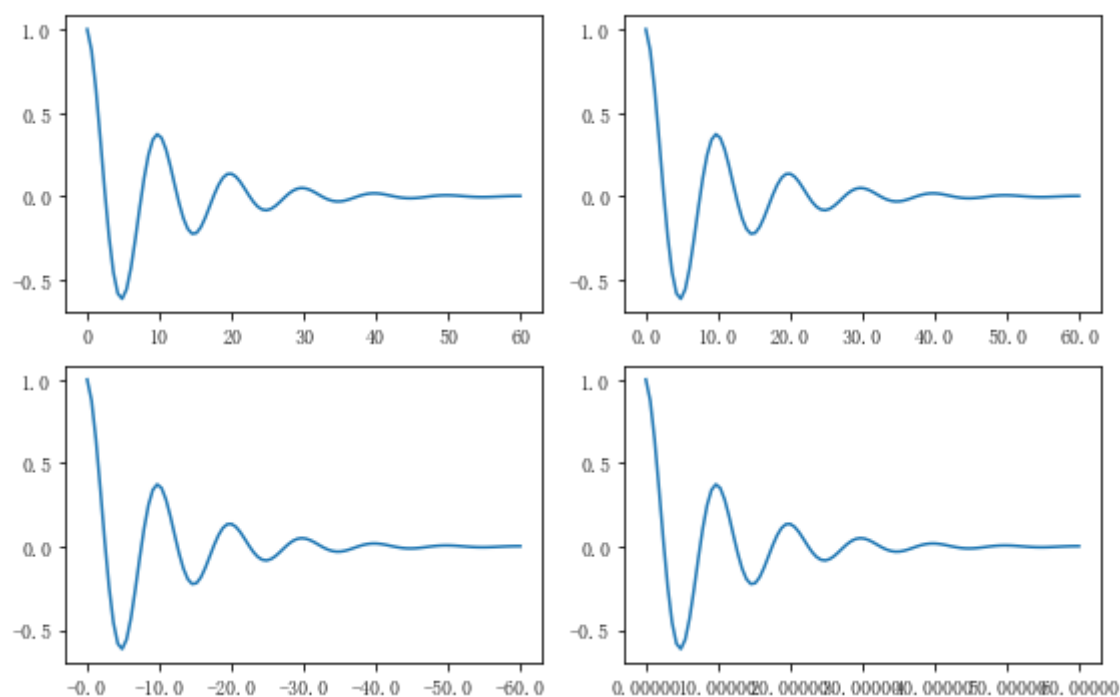
```
# 接收字符串格式的例子
fig, axs = plt.subplots(2, 2, figsize=(8, 5), tight_layout=True)
for n, ax in enumerate(axs.flat):
    ax.plot(x1*10., y1)

formatter = matplotlib.ticker.FormatStrFormatter('%1.1f')
axs[0, 1].xaxis.set_major_formatter(formatter)

formatter = matplotlib.ticker.FormatStrFormatter('%-1.1f')
axs[1, 0].xaxis.set_major_formatter(formatter)

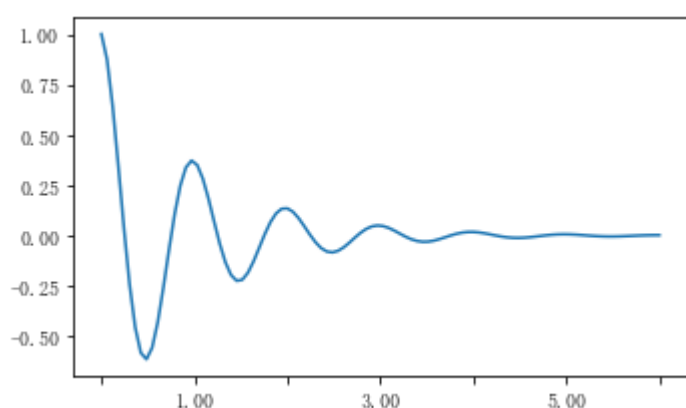
formatter = matplotlib.ticker.FormatStrFormatter('%1.5f')
axs[1, 1].xaxis.set_major_formatter(formatter)

plt.show()
```



```
# 接收函数的例子
def formatoddticks(x, pos):
    """Format odd tick positions."""
    if x % 2:
        return f'{x:1.2f}'
    else:
        return ''

fig, ax = plt.subplots(figsize=(5, 3), tight_layout=True)
ax.plot(x1, y1)
ax.xaxis.set_major_formatter(formatoddticks)
plt.show()
```



b) Tick Locators

在普通的绘图中，我们可以直接通过上图的`set_ticks`进行设置刻度的位置，缺点是需要自己指定或者接受matplotlib默认给定的刻度。当需要更改刻度的位置时，matplotlib给了常用的几种locator的类型。如果要绘制更复杂的图，可以先设置locator的类型，然后通过`ax.xaxis.set_major_locator(locator)`绘制即可

```
locator=plt.MaxNLocator(nbins=7)
```

```
locator=plt.FixedLocator(locs=[0,0.5,1.5,2.5,3.5,4.5,5.5,6])#直接指定刻度所在的位置
```

```
locator=plt.AutoLocator()#自动分配刻度值的位置
```

```
locator=plt.IndexLocator(offset=0.5, base=1)#面元间距是1，从0.5开始
```

```
locator=plt.MultipleLocator(1.5)#将刻度的标签设置为1.5的倍数
```

```
locator=plt.LinearLocator(numticks=5)#线性划分5等分，4个刻度
```

```
# 接收各种Locator的例子
fig, axs = plt.subplots(2, 2, figsize=(8, 5), tight_layout=True)
for n, ax in enumerate(axs.flat):
    ax.plot(x1*10., y1)

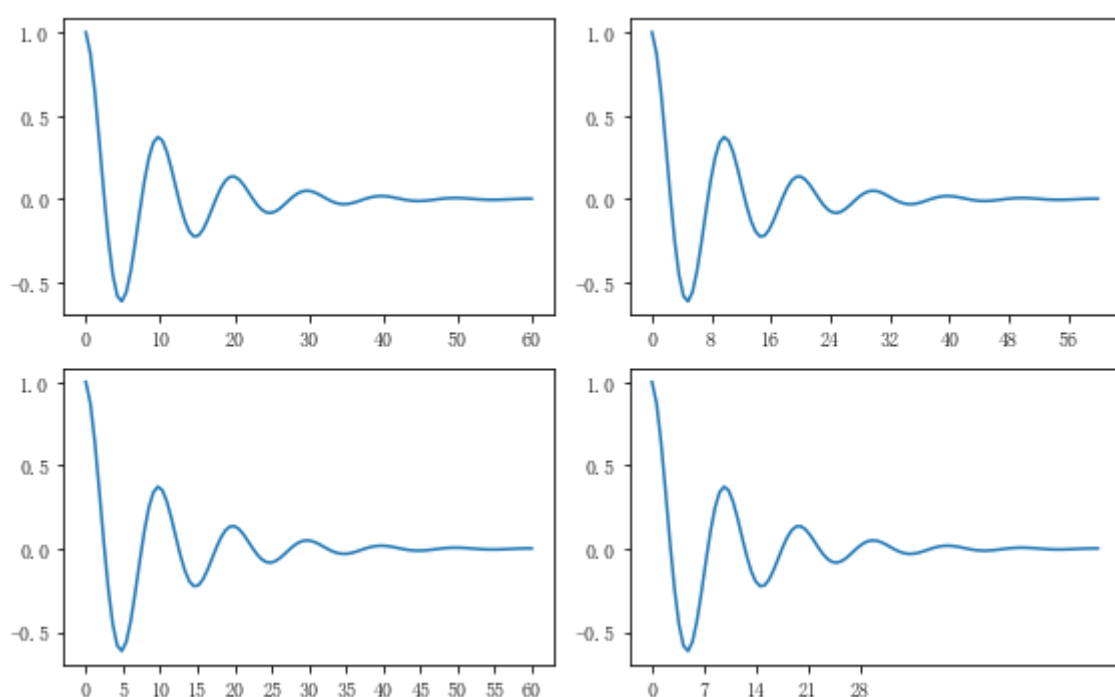
locator = matplotlib.ticker.AutoLocator()
axs[0, 0].xaxis.set_major_locator(locator)

locator = matplotlib.ticker.MaxNLocator(nbins=10)
axs[0, 1].xaxis.set_major_locator(locator)

locator = matplotlib.ticker.MultipleLocator(5)
axs[1, 0].xaxis.set_major_locator(locator)

locator = matplotlib.ticker.FixedLocator([0, 7, 14, 21, 28])
axs[1, 1].xaxis.set_major_locator(locator)

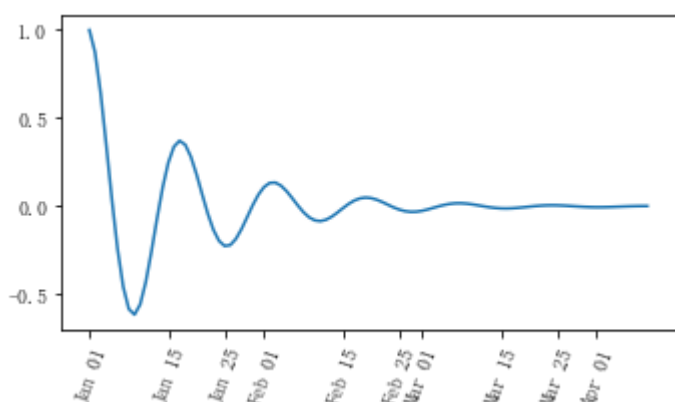
plt.show()
```



此外`matplotlib.dates` 模块还提供了特殊的设置日期型刻度格式和位置的方式

```
import matplotlib.dates as mdates
import datetime
# 特殊的日期型Locator和formatter
locator = mdates.DayLocator(bymonthday=[1,15,25])
formatter = mdates.DateFormatter('%b %d')

fig, ax = plt.subplots(figsize=(5, 3), tight_layout=True)
ax.xaxis.set_major_locator(locator)
ax.xaxis.set_major_formatter(formatter)
base = datetime.datetime(2017, 1, 1, 0, 0, 1)
time = [base + datetime.timedelta(days=x) for x in range(len(x1))]
ax.plot(time, y1)
ax.tick_params(axis='x', rotation=70)
plt.show()
```



其他进阶案例

```
#这个案例中展示了如何进行坐标轴的移动，如何更改刻度值的样式
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(-3,3,50)
y1 = 2*x+1
y2 = x**2
plt.figure()
plt.plot(x,y2)
plt.plot(x,y1,color='red',linewidth=1.0,linestyle = '--')
plt.xlim((-3,5))
plt.ylim((-3,5))
plt.xlabel('x')
plt.ylabel('y')
new_ticks1 = np.linspace(-3,5,5)
plt.xticks(new_ticks1)
plt.yticks([-2,0,2,5],[r'$one\ shu$',r'$\alpha$',r'$three$',r'four']])
'''
```

上一行代码是将y轴上的小标改成文字,其中, 空格需要增加\, 即'\'', \$可将格式更改成数字模式, 如果需要输入数学形式的 α , 则需要用\转换, 即\alpha

如果使用面向对象的命令进行画图, 那么下面两行代码可以实现与 `plt.yticks([-2,0,2,5],[r'$one\ shu$',r'α',r'$three$',r'four']])` 同样的功能

```
ax.set_yticks([-2,0,2,5])
ax.set_yticklabels([r'$one\ shu$',r'$\alpha$',r'$three$',r'four'])
'''
```

```
ax = plt.gca()#gca = 'get current axes' 获取现在的轴
'''
```

`ax = plt.gca()`是获取当前的axes, 其中gca代表的是get current axes。
`fig=plt.gcf`是获取当前的figure, 其中gcf代表的是get current figure。

许多函数都是对当前的Figure或Axes对象进行处理,

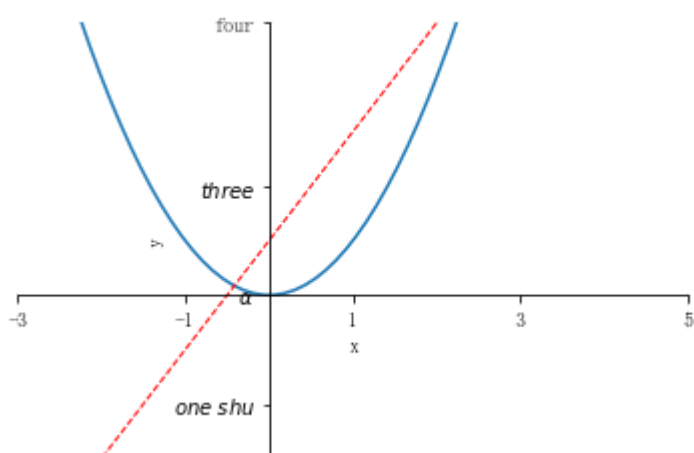
例如`plt.plot()`实际上会通过`plt.gca()`获得当前的Axes对象ax, 然后再调用`ax.plot()`方法实现真正的绘图。

而在本例中则可以通过`ax.spines`方法获得当前顶部和右边的轴并将其颜色设置为不可见

然后将左边轴和底部的轴所在的位置重新设置

最后再通过`set_ticks_position`方法设置ticks在x轴或y轴的位置, 本示例中因所设置的bottom和left是ticks在x轴或y轴的默认值, 所以这两行的代码也可以不写

```
'''
ax.spines['top'].set_color('none')
ax.spines['right'].set_color('none')
ax.spines['left'].set_position(('data',0))
ax.spines['bottom'].set_position(('data',0))#axes 百分比
ax.xaxis.set_ticks_position('bottom')    #设置ticks在x轴的位置
ax.yaxis.set_ticks_position('left')      #设置ticks在y轴的位置
plt.show()
```



三、[legend](#)（图例）

图例的设置会使用一些常见术语，为了清楚起见，这些术语在此处进行说明： **legend entry**（图例条目） **图例**有一个或多个**legend entries**组成。一个**entry**由一个**key**和一个**label**组成。 **legend key**（图例键） **每个 legend label**左面的**colored/patterned marker**（彩色/图案标记） **legend label**（图例标签） **描述**由**key**来表示的**handle**的文本 **legend handle**（图例句柄） **用于在图例中生成适当图例条目的原始对象**

以下面这个图为例，右侧的方框中的共有两个**legend entry**；两个**legend key**，分别是一个蓝色和一个黄色的**legend key**；两个**legend label**，一个名为‘Line up’和一个名为‘Line Down’的**legend label**

常用的几个参数：

(1)设置图列位置

plt.legend(loc='upper center') 等同于plt.legend(loc=9)，对应关系如下表。

loc by number	loc by text
0	'best'
1	'upper right'
2	'upper left'
3	'lower left'
4	'lower right'
5	'right'
6	'center left'
7	'center right'
8	'lower center'
9	'upper center'
10	'center'

(2)设置图例字体大小

fontsize : int or float or {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}

(3)设置图例边框及背景

```
plt.legend(loc='best',frameon=False) #去掉图例边框
plt.legend(loc='best',edgecolor='blue') #设置图例边框颜色
plt.legend(loc='best',facecolor='blue') #设置图例背景颜色,若无边框,参数无效
```

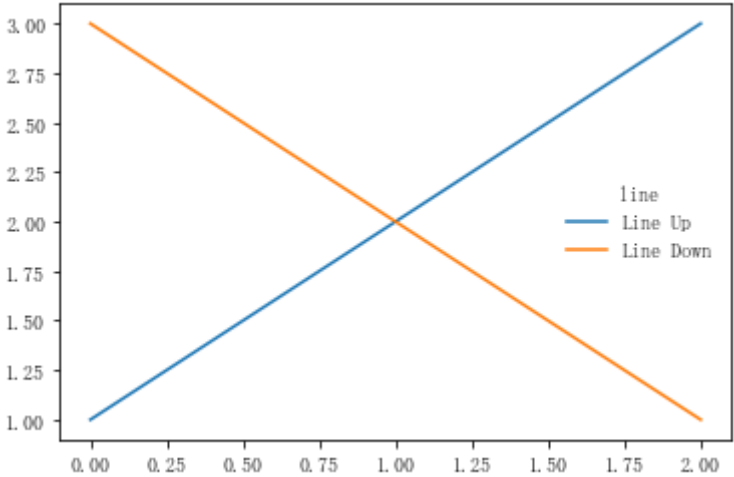
(4)设置图例标题

```
legend = plt.legend(["CH", "US"], title='China VS Us')
```

(5)设置图例名字及对应关系

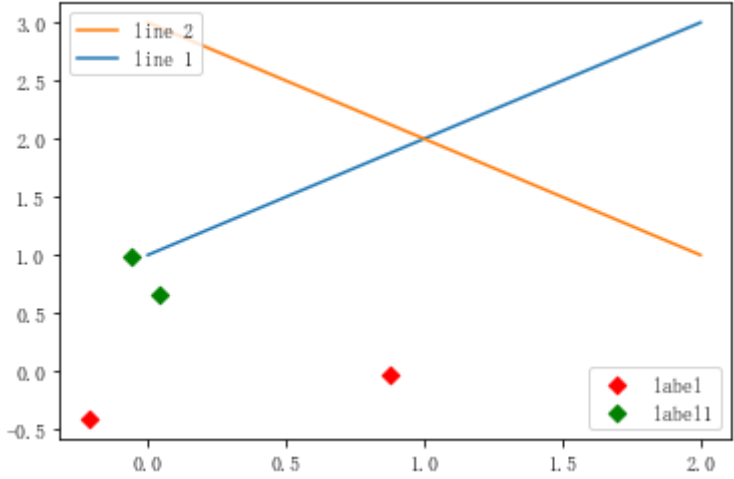
```
legend = plt.legend([p1, p2], ["CH", "US"])
```

```
line_up, = plt.plot([1, 2, 3], label='Line 2')
line_down, = plt.plot([3, 2, 1], label='Line 1')
plt.legend([line_up, line_down], ['Line Up', 'Line Down'],loc=5,
title='line',frameon=False);#Loc参数设置图例所在的位置, title设置图例的标题, frameon参数将图例边框
给去掉
```



```
#这个案例是显示多图例Legend
import matplotlib.pyplot as plt
import numpy as np
x = np.random.uniform(-1, 1, 4)
y = np.random.uniform(-1, 1, 4)
p1, = plt.plot([1,2,3])
p2, = plt.plot([3,2,1])
l1 = plt.legend([p2, p1], ["line 2", "line 1"], loc='upper left')

p3 = plt.scatter(x[0:2], y[0:2], marker = 'D', color='r')
p4 = plt.scatter(x[2:], y[2:], marker = 'D', color='g')
# 下面这行代码由于添加了新的Legend, 所以会将l1从Legend中给移除
plt.legend([p3, p4], ['label', 'label1'], loc='lower right', scatterpoints=1)
# 为了保留之前的l1这个Legend, 所以必须要通过plt.gca()获得当前的axes, 然后将l1作为单独的artist
plt.gca().add_artist(l1);
```



参考资料

[1.matplotlib官网文字使用指南](#)

By Datawhale
© Copyright © Copyright 2020.

第五回： 样式色彩秀芳华

第五回详细介绍matplotlib中样式和颜色的使用，绘图样式和颜色是丰富可视化图表的重要手段，因此熟练掌握本章可以让可视化图表变得更美观，突出重点和凸显艺术性。

关于绘图样式，常见的有4种方法，分别是修改预定义样式，自定义样式，rcparams和matplotlibrc文件。

关于颜色使用，本章介绍了常见的5种表示单色颜色的基本方法，以及colormap多色显示的方法。

一、matplotlib的绘图样式（style）

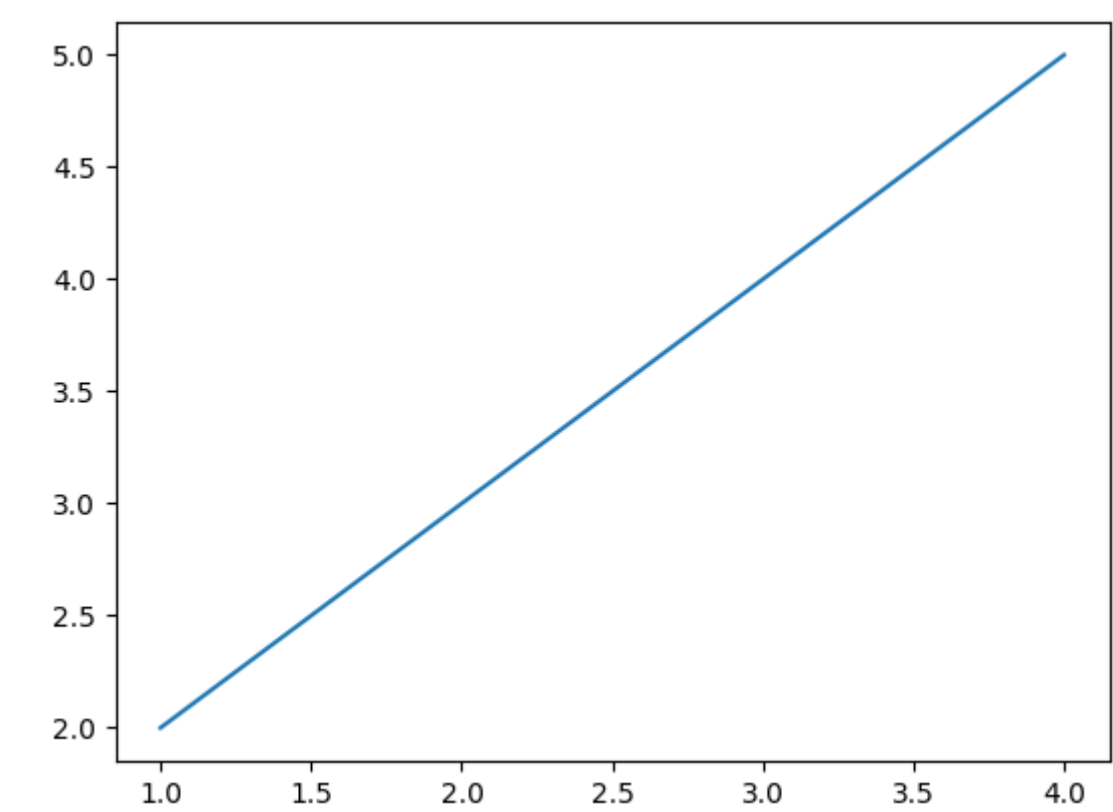
在matplotlib中，要想设置绘制样式，最简单的方法是在绘制元素时单独设置样式。但是有时候，当用户在做专题报告时，往往会希望保持整体风格的统一而不用对每张图一张张修改，因此matplotlib库还提供了四种批量修改全局样式的方式

1.matplotlib预先定义样式

matplotlib贴心地提供了许多内置的样式供用户使用，使用方法很简单，只需在python脚本的最开始输入想使用style的名称即可调用，尝试调用不同内置样式，比较区别

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

```
plt.style.use('default')
plt.plot([1,2,3,4],[2,3,4,5]);
```



```
plt.style.use('ggplot')
plt.plot([1,2,3,4],[2,3,4,5]);
```

Contents

Print to PDF ▶

二、matplotlib的绘图样式 (style)

1.matplotlib预先定义样式

2.用户自定义stylesheet

3.设置rcparams

4.修改matplotlibrc文件

二、matplotlib的色彩设置 (color)

1.RGB或RGBA

2.HEX RGB 或 RGBA

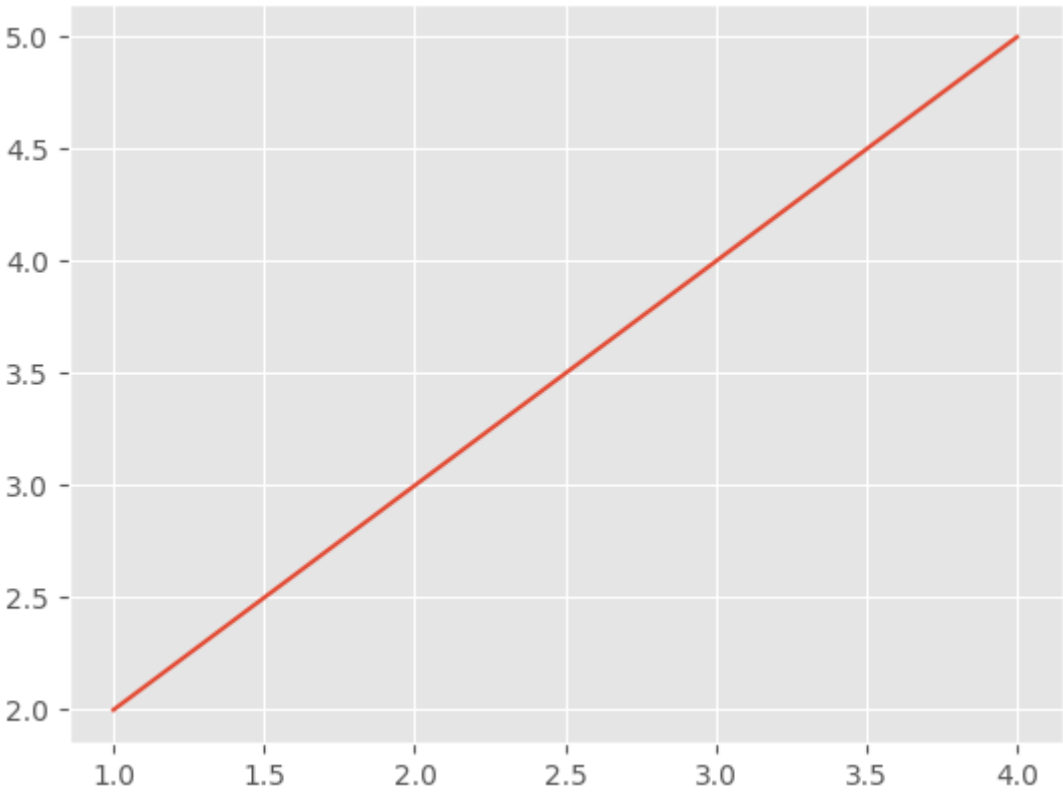
3.灰度色阶

4.单字符基本颜色

5.颜色名称

6.使用colormap设置一组颜色

参考资料



那么matplotlib究竟内置了那些样式供使用呢？总共以下26种丰富的样式可供选择。

```
print(plt.style.available)

['Solarize_Light2', '_classic_test_patch', 'bmh', 'classic', 'dark_background', 'fast',
'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn', 'seaborn-bright', 'seaborn-colorblind',
'seaborn-dark', 'seaborn-dark-palette', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted',
'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk',
'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'tableau-colorblind10']
```

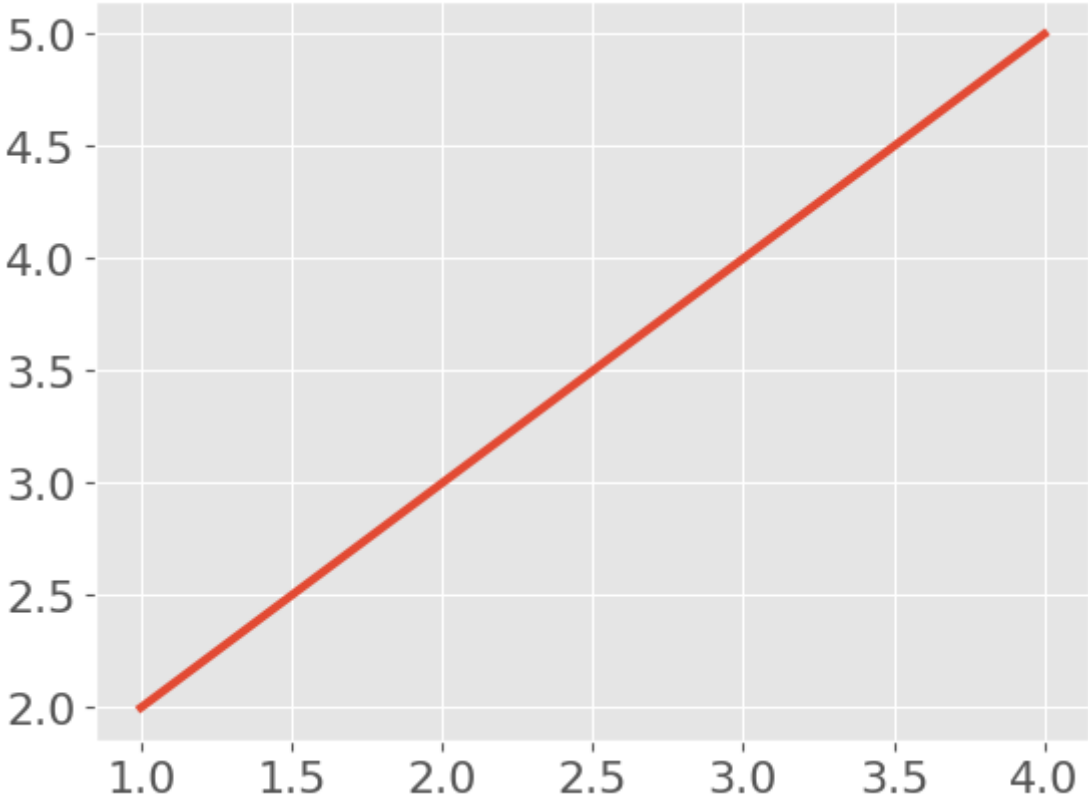
2.用户自定义stylesheet

在任意路径下创建一个后缀名为mplstyle的样式清单，编辑文件添加以下样式内容

```
axes.titlesize : 24
axes.labelsize : 20
lines.linewidth : 3
lines.markersize : 10
xtick.labelsize : 16
ytick.labelsize : 16
```

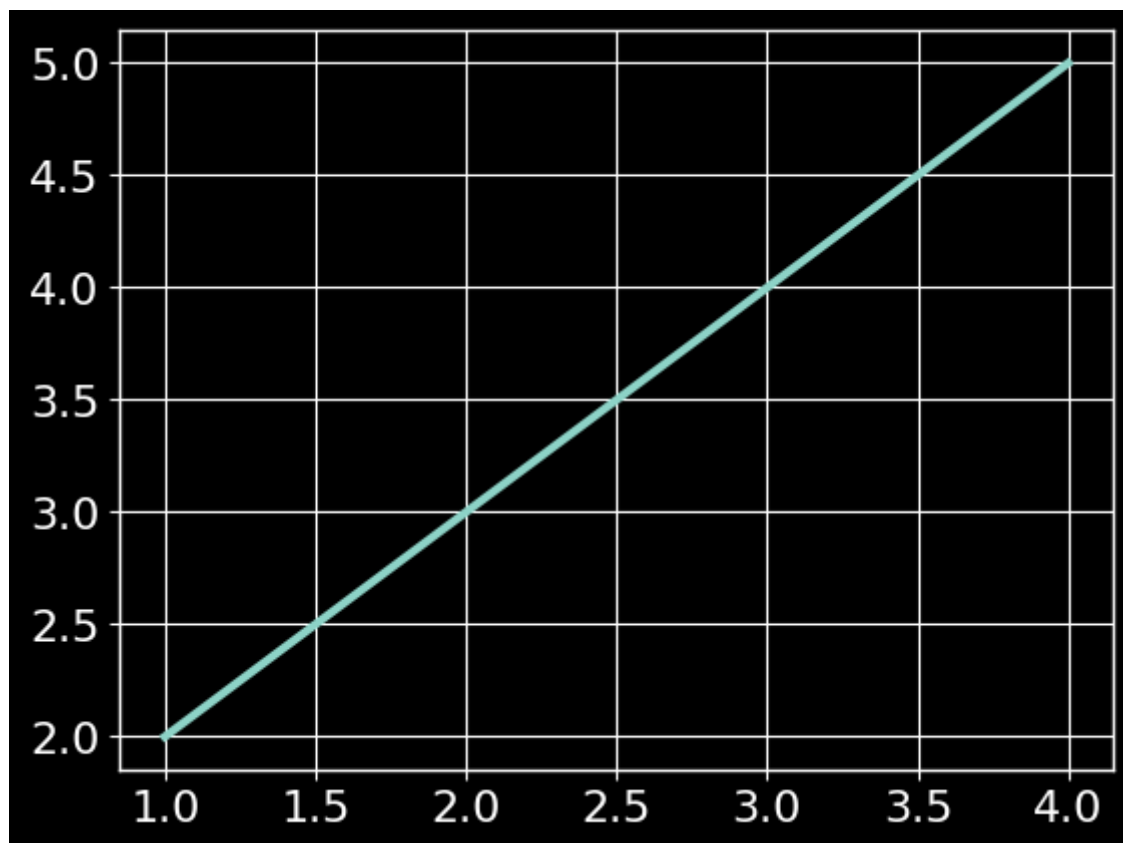
引用自定义stylesheet后观察图表变化。

```
plt.style.use('file/presentation.mplstyle')
plt.plot([1,2,3,4],[2,3,4,5]);
```



值得特别注意的是，matplotlib支持混合样式的引用，只需在引用时输入一个样式列表，若是几个样式中涉及到同一个参数，右边的样式表会覆盖左边的值。

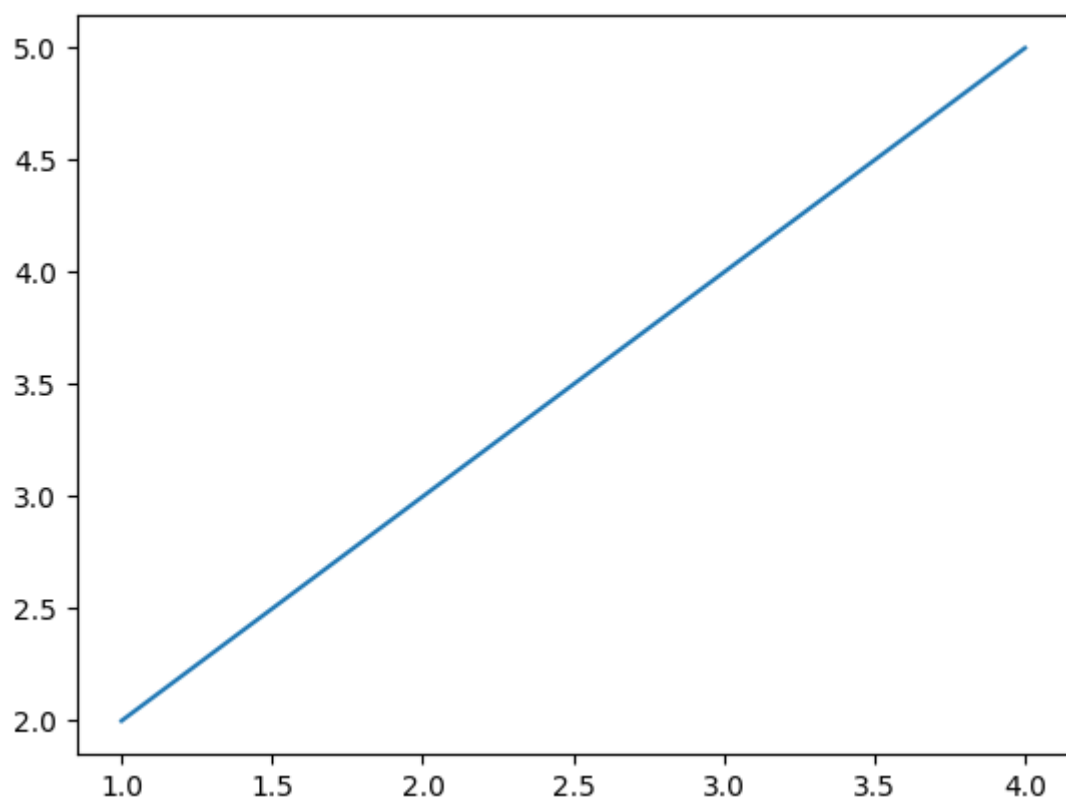
```
plt.style.use(['dark_background', 'file/presentation.mplstyle'])
plt.plot([1,2,3,4],[2,3,4,5]);
```



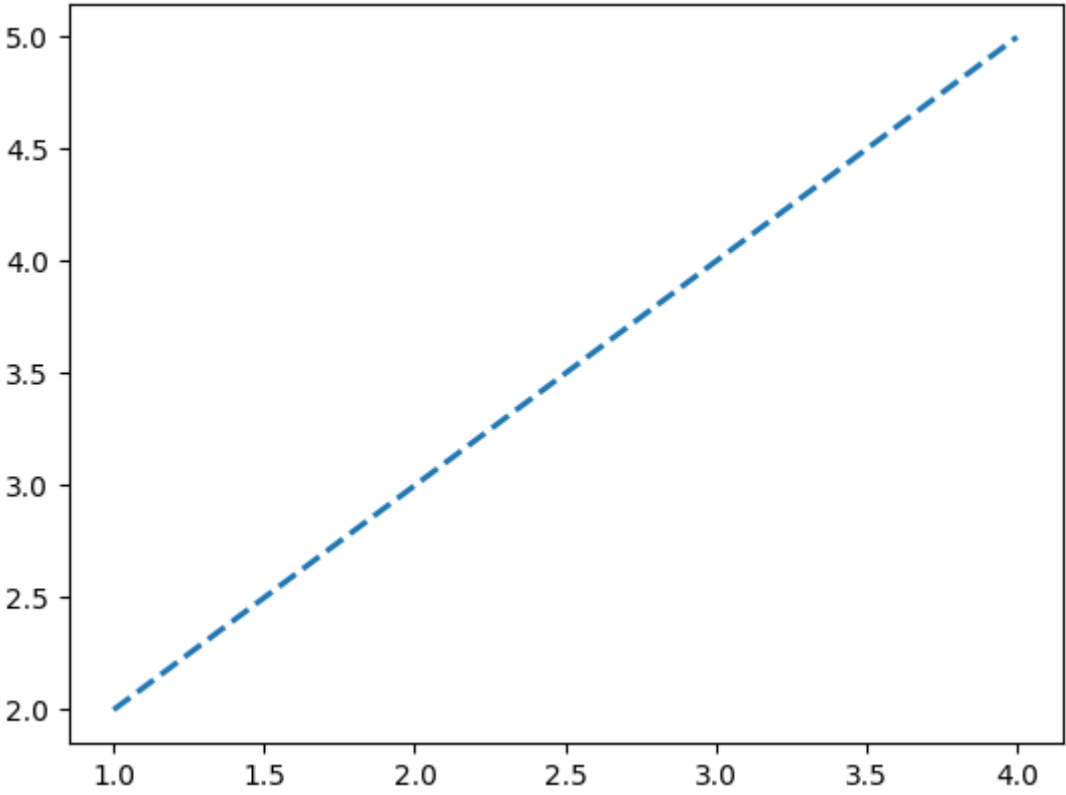
3.设置rcparams

我们还可以通过修改默认rc设置的方式改变样式，所有rc设置都保存在一个叫做 matplotlib.rcParams的变量中。修改过后再绘图，可以看到绘图样式发生了变化。

```
plt.style.use('default') # 恢复到默认样式
plt.plot([1,2,3,4],[2,3,4,5]);
```

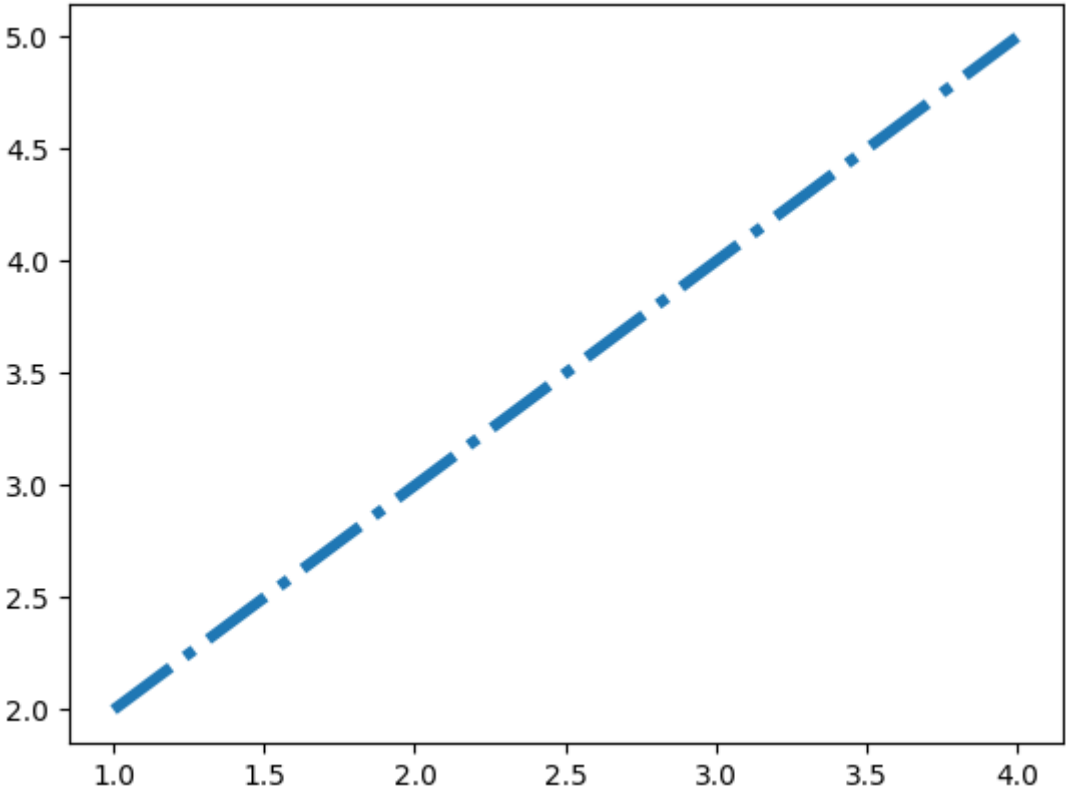


```
mpl.rcParams['lines.linewidth'] = 2
mpl.rcParams['lines.linestyle'] = '--'
plt.plot([1,2,3,4],[2,3,4,5]);
```



另外matplotlib也还提供了了一种更便捷的修改样式方式，可以一次性修改多个样式。

```
mpl.rcParams['lines', linewidth=4, linestyle='-.']
plt.plot([1,2,3,4],[2,3,4,5]);
```



4.修改matplotlibrc文件

由于matplotlib是使用matplotlibrc文件来控制样式的，也就是上一节提到的rc setting，所以我们还可以通过修改matplotlibrc文件的方式改变样式。

```
# 查找matplotlibrc文件的路径
mpl.matplotlib_fname()

'c:\\users\\skywater\\pycharmprojects\\personal\\demo\\lib\\site-packages\\matplotlib\\mpl-
data\\matplotlibrc'
```

找到路径后，就可以直接编辑样式文件了，打开后看到的文件格式大致是这样的，文件中列举了所有的样式参数，找到想要修改的参数，比如lines.linewidth: 8，并将前面的注释符号去掉，此时再绘图发现样式以及生效了。

```
## *****
## * LINES *
## *****
## See https://matplotlib.org/api/artist_api.html#module-matplotlib.lines
## for more information on line properties.
#lines.linewidth: 1.5          # line width in points
#lines.linestyle: -           # solid line
#lines.color: C0              # has no affect on plot(); see axes.prop_cycle
#lines.marker: None           # the default marker
#lines.markerfacecolor: auto   # the default marker face color
#lines.markeredgecolor: auto   # the default marker edge color
#lines.markeredgewidth: 1.0    # the line width around the marker symbol
#lines.markersize: 6           # marker size, in points
#lines.dash_joinstyle: round    # {miter, round, bevel}
#lines.dash_capstyle: butt      # {butt, round, projecting}
#lines.solid_joinstyle: round    # {miter, round, bevel}
#lines.solid_capstyle: projecting # {butt, round, projecting}
#lines.antialiased: True        # render lines in antialiased (no jaggies)

## The three standard dash patterns. These are scaled by the linewidth.
#lines.dashed_pattern: 3.7, 1.6
#lines.dashdot_pattern: 6.4, 1.6, 1, 1.6
#lines.dotted_pattern: 1, 1.65
#lines.scale_dashes: True
```

二、matplotlib的色彩设置（color）

在可视化中，如何选择合适的颜色和搭配组合也是需要仔细考虑的，色彩选择要能够反映出可视化图像的主旨。从可视化编码的角度对颜色进行分析，可以将颜色分为色相、亮度和饱和度三个视觉通道。通常来说：
色相：没有明显的顺序性、一般不用来表达数据量的高低，而是用来表达数据列的类别。
明度和饱和度：在视觉上很容易区分出优先级的高低、被用作表达顺序或者表达数据量视觉通道。
具体关于色彩理论部分的知识，不属于本教程的重点，请参阅有关拓展材料学习。

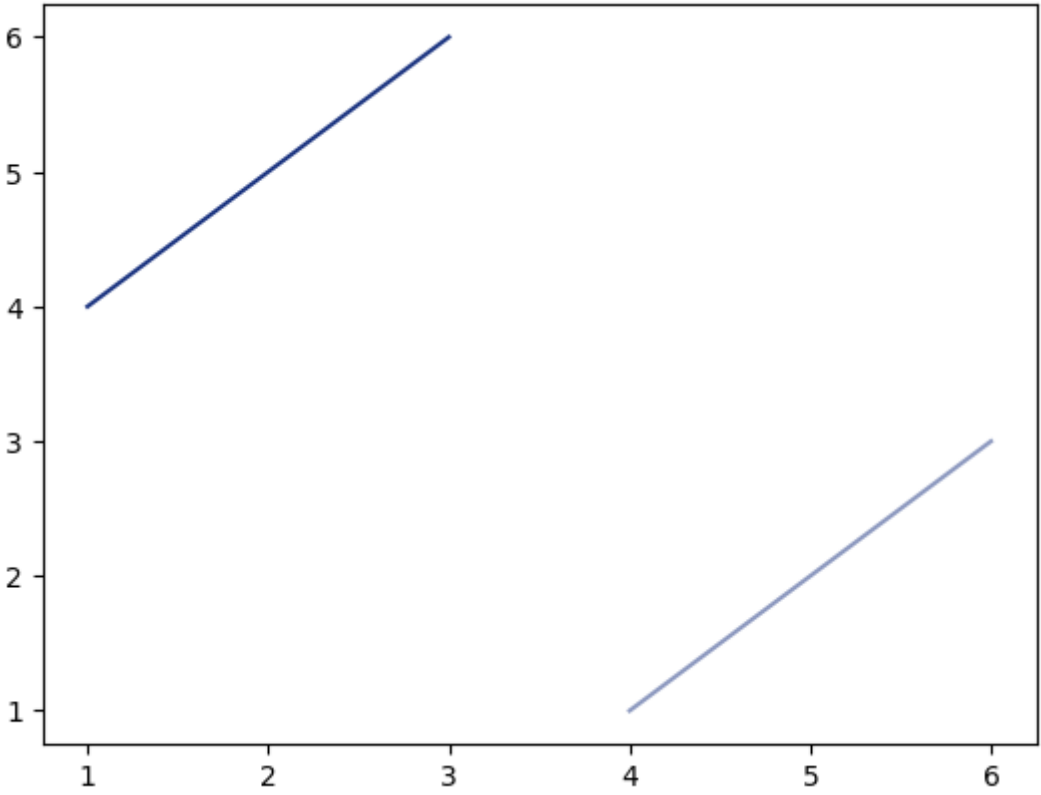
[ECharts数据可视化实验室](#)
[学会这6个可视化配色基本技巧，还原数据本身的意义](#)

在matplotlib中，设置颜色有以下几种方式：

1.RGB或RGBA

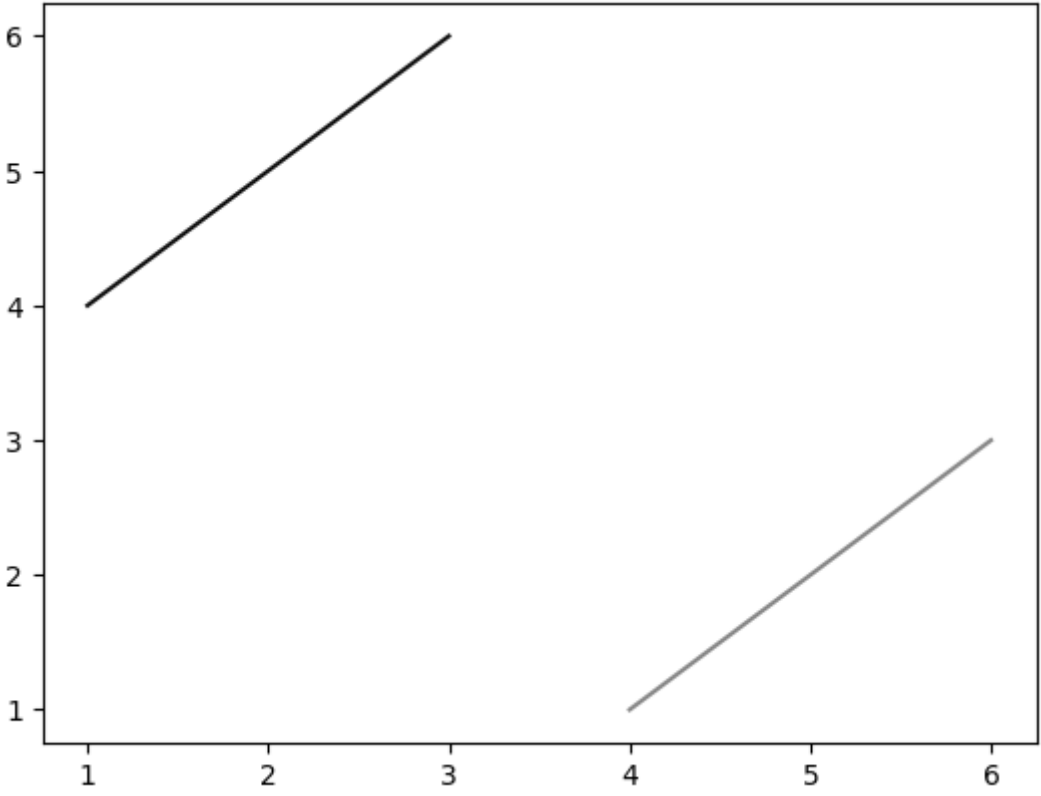
```
plt.style.use('default')

# 颜色用[0,1]之间的浮点数表示，四个分量按顺序分别为(red, green, blue, alpha)，其中alpha透明度可省略
plt.plot([1,2,3],[4,5,6],color=(0.1, 0.2, 0.5))
plt.plot([4,5,6],[1,2,3],color=(0.1, 0.2, 0.5, 0.5));
```



2.HEX RGB 或 RGBA

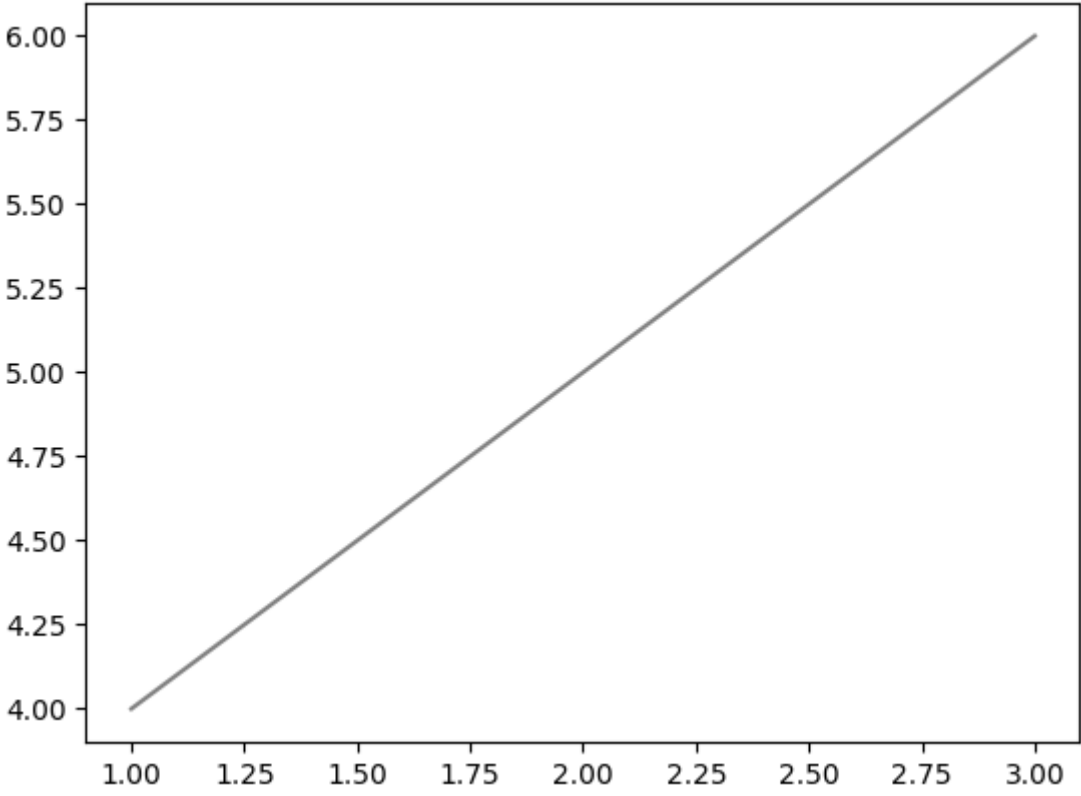
```
# 用十六进制颜色码表示，同样最后两位表示透明度，可省略
plt.plot([1,2,3],[4,5,6],color='#0f0f0f')
plt.plot([4,5,6],[1,2,3],color='#0f0f80');
```



RGB颜色和HEX颜色之间是可以一一对应的，以下网址提供了两种色彩表示方法的转换工具。
<https://www.colorhexa.com/>

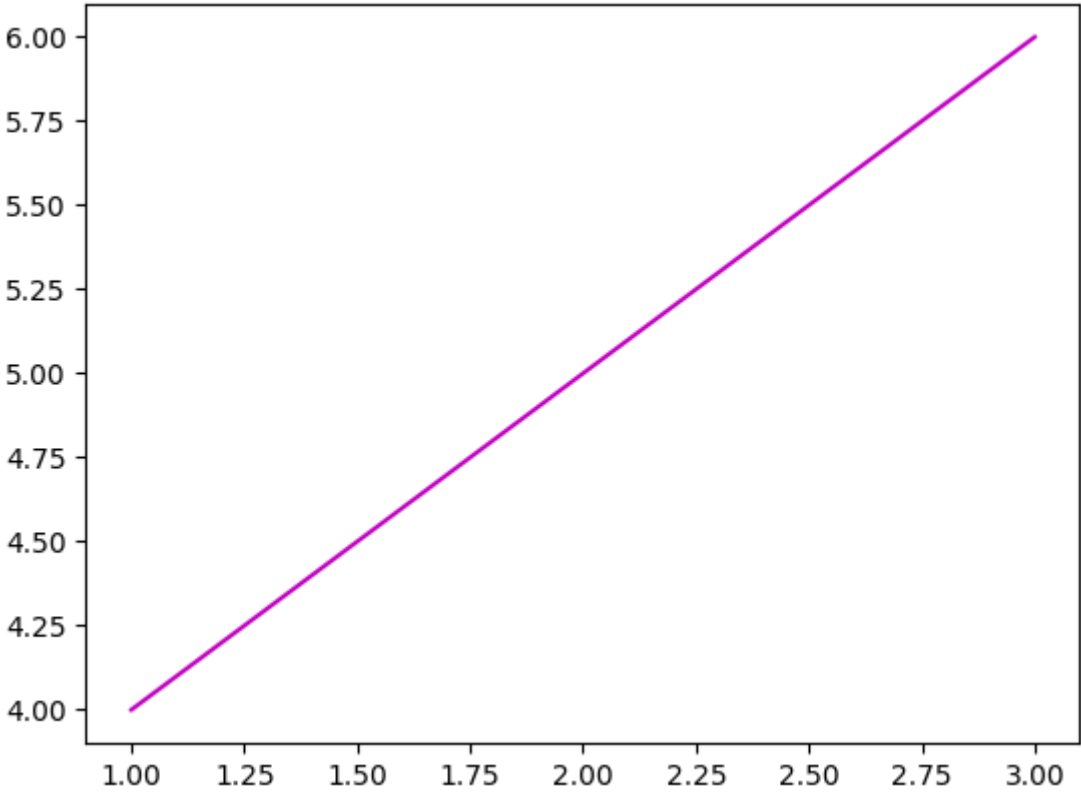
3.灰度色阶

```
# 当只有一个位于[0,1]的值时，表示灰度色阶
plt.plot([1,2,3],[4,5,6],color='0.5');
```



4.单字符基本颜色

```
# matplotlib有八个基本颜色，可以用单字符串来表示，分别是'b', 'g', 'r', 'c', 'm', 'y', 'k', 'w', 对应的是blue, green, red, cyan, magenta, yellow, black, and white的英文缩写
plt.plot([1,2,3],[4,5,6],color='m');
```



5.颜色名称

```
# matplotlib提供了颜色对照表，可供查询颜色对应的名称
plt.plot([1,2,3],[4,5,6],color='tan');
```

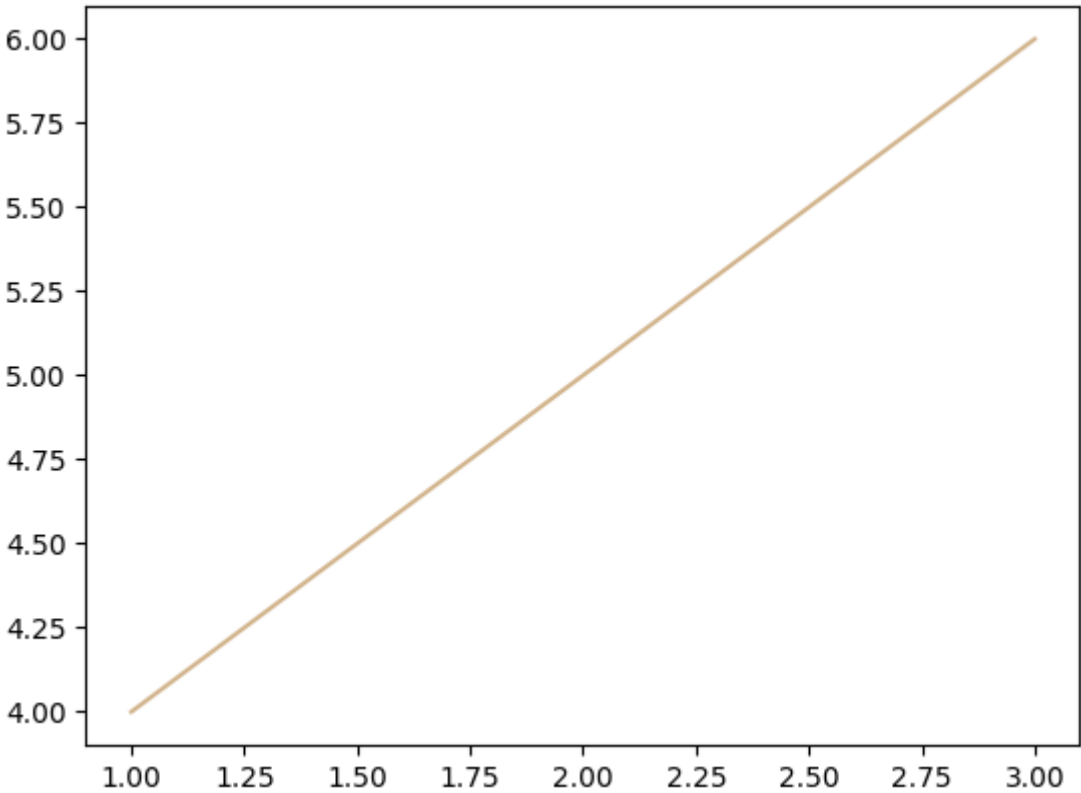







Tableau Palette

	tab:blue		tab:brown
	tab:orange		tab:pink
	tab:green		tab:gray
	tab:red		tab:olive
	tab:purple		tab:cyan

CSS Colors

	black		bisque		forestgreen		slategrey
	dimgray		darkorange		limegreen		lightsteelblue
	dimgrey		burlywood		darkgreen		cornflowerblue
	gray		antiquewhite		green		royalblue
	grey		tan		lime		ghostwhite
	darkgray		navajowhite		seagreen		lavender
	darkgrey		blanchedalmond		mediumseagreen		midnightblue
	silver		papayawhip		springgreen		navy
	lightgray		moccasin		mintcream		darkblue
	lightgrey		orange		mediumspringgreen		mediumblue
	gainsboro		wheat		mediumaquamarine		blue
	whitesmoke		oldlace		aquamarine		slateblue
	white		floralwhite		turquoise		darkslateblue
	snow		darkgoldenrod		lightseagreen		mediumslateblue
	rosybrown		goldenrod		mediumturquoise		mediumpurple
	lightcoral		cornsilk		azure		rebeccapurple
	indianred		gold		lightcyan		blueviolet
	brown		lemonchiffon		paleturquoise		indigo
	firebrick		khaki		darkslategray		darkorchid
	maroon		palegoldenrod		darkslategrey		darkviolet
	darkred		darkkhaki		teal		mediumorchid
	red		ivory		darkcyan		thistle
	mistyrose		beige		aqua		plum
	salmon		lightyellow		cyan		violet
	tomato		lightgoldenrodyellow		darkturquoise		purple
	darksalmon		olive		cadetblue		darkmagenta
	coral		yellow		powderblue		fuchsia
	orangered		olivedrab		lightblue		magenta
	lightsalmon		yellowgreen		deepskyblue		orchid
	sienna		darkolivegreen		skyblue		mediumvioletred
	seashell		greenyellow		lightskyblue		deeppink
	chocolate		chartreuse		steelblue		hotpink
	saddlebrown		lawngreen		aliceblue		lavenderblush
	sandybrown		honeydew		dodgerblue		palevioletred
	peachpuff		darkseagreen		lightslategray		crimson
	peru		palegreen		lightslategrey		pink
	linen		lightgreen		slategray		lightpink

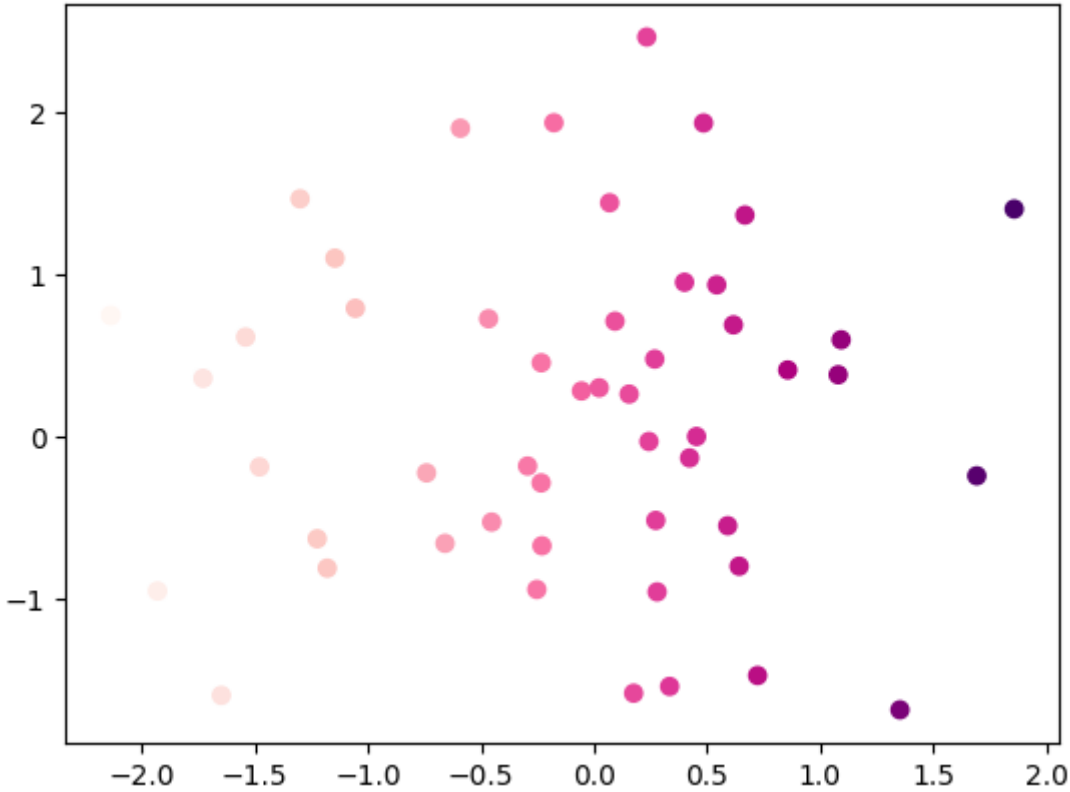
6.使用colormap设置一组颜色

有些图表支持使用colormap的方式配置一组颜色，从而在可视化中通过色彩的变化表达更多信息。

在matplotlib中，colormap共有五种类型:

- 顺序（Sequential）。通常使用单一色调，逐渐改变亮度和颜色渐渐增加，用于表示有顺序的信息
- 发散（Diverging）。改变两种不同颜色的亮度和饱和度，这些颜色在中间以不饱和的颜色相遇;当绘制的信息具有关键中间值（例如地形）或数据偏离零时，应使用此值。
- 循环（Cyclic）。改变两种不同颜色的亮度，在中间和开始/结束时以不饱和的颜色相遇。用于在端点处环绕的值，例如相角，风向或一天中的时间。
- 定性（Qualitative）。常是杂色，用来表示没有排序或关系的信息。
- 杂色（Miscellaneous）。一些在特定场景使用的杂色组合，如彩虹，海洋，地形等。

```
x = np.random.randn(50)
y = np.random.randn(50)
plt.scatter(x,y,c=x,cmap='RdPu');
```



在以下官网页面可以查询上述五种colormap的字符串表示和颜色图的对应关系
<https://matplotlib.org/stable/tutorials/colors/colormaps.html>

参考资料

- [1.matplotlib官网样式使用指南](#)
- [2.matplotlib官网色彩使用指南](#)

By Datawhale
© Copyright © Copyright 2020.