一、matplotlib绘图基础

Python扩展库matplotlib是一套以来扩展库numpy和标准库tkinter的绘图工具包，可以绘制出满足出版级质量要求的图形。matplotlib为Python构建了一个类似MATLAB的绘图接口，广泛应用于数据可视化和科学计算可视化领域。

1.matplotlib绘图简介

Python扩展库matplotlib主要包括pyplot、pylab等绘图模块，也包含大量用于字体、颜色、图例等图形元素管理和控制的模块，支持线条样式、字体属性、轴属性等属性管理和控制，可以使用非常简洁的代码绘制出优美的图案。其中，pyplot绘图模块包含常用的matplotlib `API` 函数，可以使用numpy进行数组运算和表示。

使用扩展库matplotlib进行数据可视化之前，首先导入必要的工具包：

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

这样，后面的代码可以使用 `plt` 来使用 `pyplot` 模块。此外，若想代码能够访问matplotlib函数集合，也需要导入matplotlib库：

```
import matplotlib as mpl
```

1.1matplotlib绘图步骤

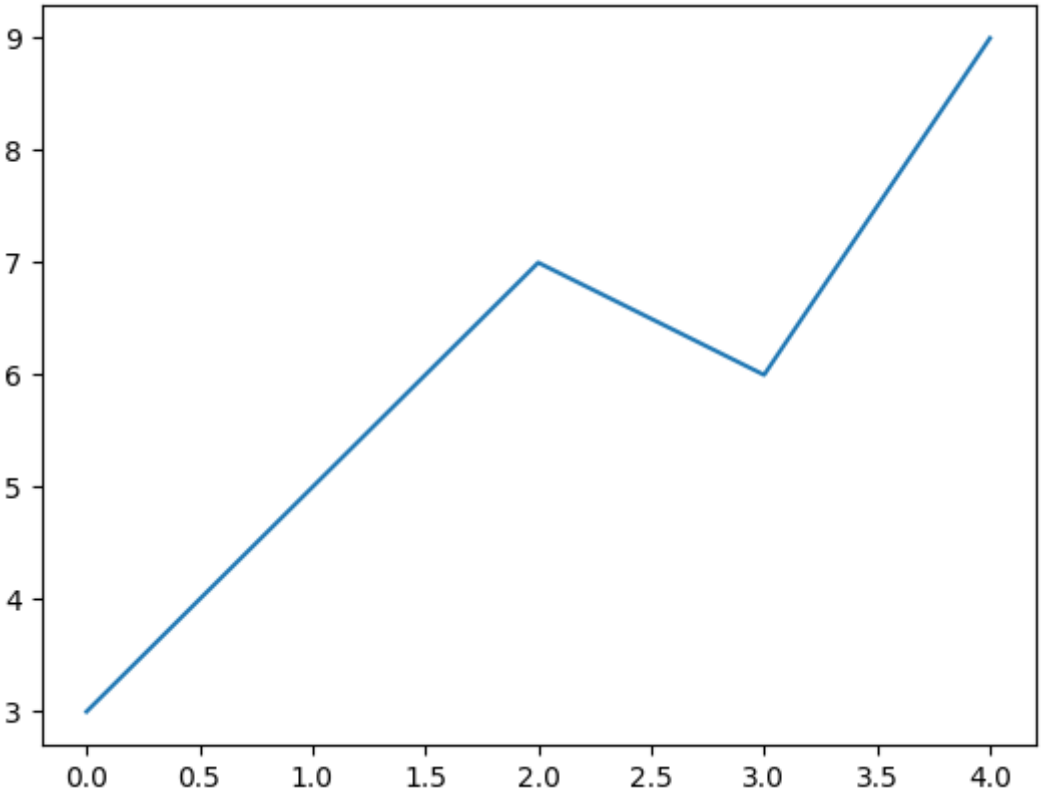
使用Python语句生成或读入数据后，通常根据数据类型和可视化要求选择合适的图形，以便于数据表示。然后可以使用pylab模块或者pyplot模块绘制图形，一般步骤如下：

- 第一步，创建画布（相当于绘画前准备图纸）
- 第二步，根据需要，在画布上创建一个或多个绘图区域。可以在一张画布上绘制一个图形，也可以将画布分成几个部分，在每个组成部分分别绘制相同或不同的图形。这时，图形的不同组成部分拥有各自独立的坐标系
- 第三步，在选定的区域描点、线、图形等
- 第四步，为绘图线或坐标轴添加刻度、范围及修饰标签等，使得图形更容易理解，更具表现力
- 第五步，根据需要为图形添加其他信息，例如旋转标签、图例、标题等等
- 最后，显示绘图结果

给出一个示例代码：

```
# 创建画布
plt.figure()
# 生成数据
data = [3, 5, 7, 6, 9]
# 图形绘制
plt.plot(data)
# 可视化结果
plt.show()
```

运行结果如下图所示：



1.2matplotlib基本元素

matplotlib基本元素如下表所示：

元素	含义
画布（Figure）	使用扩展库matplotlib绘图时，用户绘制的图形都成现在画布上
变量	变量是用于绘制图形的数据，生成或读入需要可视化展示的数据后，一般会将数据保存在变量中。用户调用matplotlib，使用相关的命令操作变量进行可视化
函数	matplotlib模块包含各种各样的绘图函数，调用这些绘图函数，将数据绘制成图形，以可视化形式展示出来
坐标轴（Axes）	包含水平（x轴）和垂直（y轴）的轴线

绘图时，用户通过变量和函数改变画布和坐标轴中的元素，例如标题（title）、标签（label）等等，描述画布和坐标轴上呈现的内容。

要注意，matplotlib基本元素之间呈**层级关系。**画布上可以创建多个绘图区域，因此画布`fig`内可以包含多个子图`ax`；每一个子图拥有自

己的数据`data`、标题`title`以及横轴`xaxis`和纵轴`yaxis`的坐标系；每个坐标轴包含单独的刻度`tick`和坐标轴标签`label`；刻度`tick`

还可以拥有自己的刻度标签`tick label`，即为每个刻度赋予实际意义。

2.matplotlib基本元素可视化

2.1matplotlib画布

用户绘制的matplotlib图形位于画布上，可以通过`plt.figure()`函数创建画布

```
plt.figure([num, figsize, dpi, facecolor, edgecolor, frameon])
```

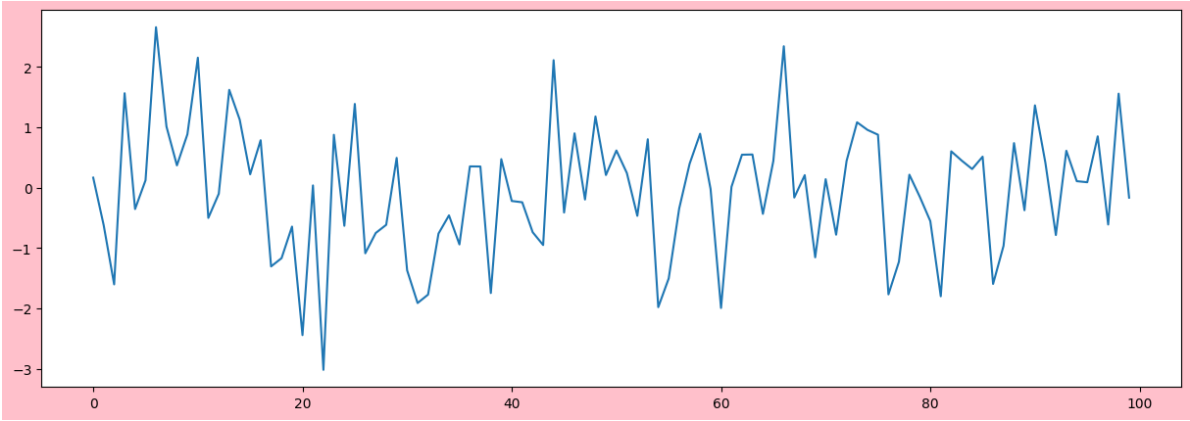
各个参数的意义：

参数	意义
`num`	指定图形编号或名称，数字表示图形编号，字符串表示图形名称
`figsize`	指定画布的宽和高，单位为英寸，1英寸等于2.5cm
`dpi`	指定绘图对象的分辨率，即每英寸多少个像素，默认值为80
`facecolor`	指定画布背景颜色
`edgecolor`	指定画布边框颜色
`frameon`	指定是否显示画布边框

示例代码：

```
# 创建画布
plt.figure('fig_test', figsize = (15, 5), facecolor = 'pink', frameon = True)
# 生成数据
random_data = np.random.randn(100)
# 绘制图形
plt.plot(random_data)
# 可视化结果
plt.show()
```

运行结果如下图所示：



如果用户省略画布创建语句，matplotlib会生成默认画布。图形绘制完毕后，需要用`plt.show()`来显示绘图结果。

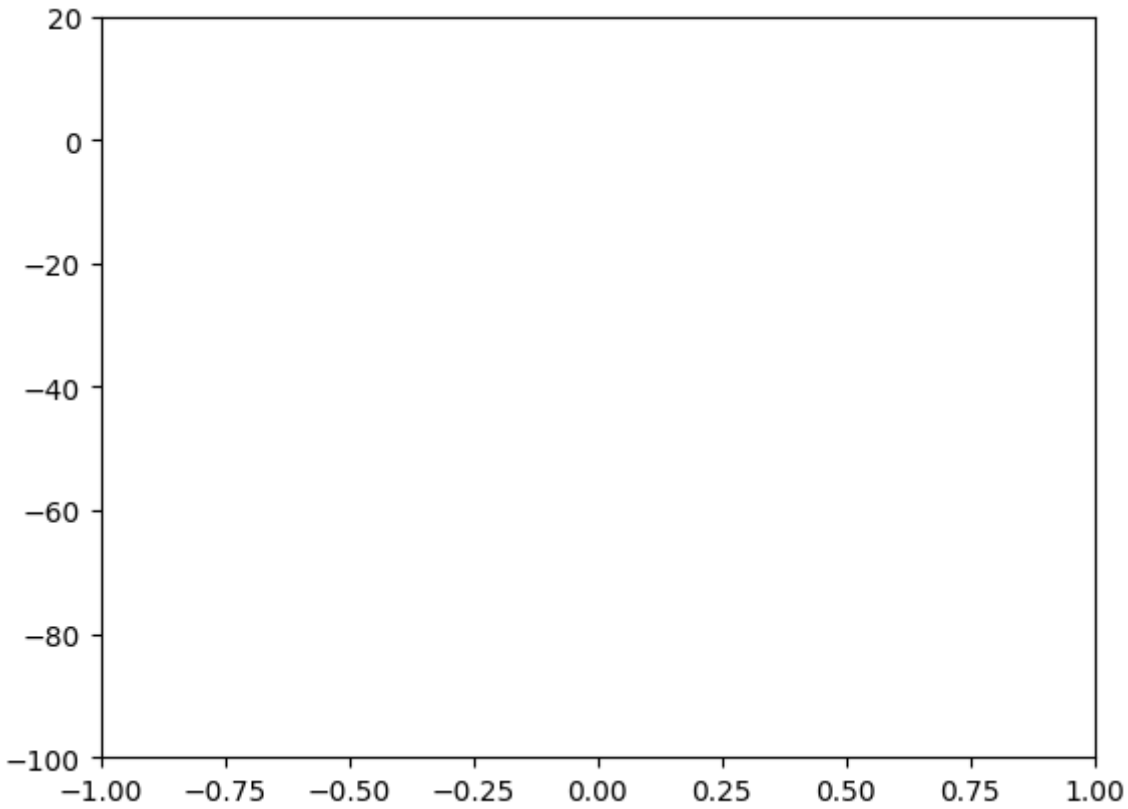
2.2设置坐标轴长度与范围

matplotlib绘图时，可以调用`plt.axis()`函数设置x轴和y轴的属性值。

示例代码：

```
# 设置坐标轴范围
ax = [-1, 1, -100, 100]
plt.axis(ax)
# 设置y轴
plt.axis(ymax = 20)
# 可视化结果
plt.show()
```

运行结果如下：



2.3设置图形的线型与颜色

matplotlib绘图时，可以设置线条的颜色、线宽、点标记等线条风格，满足个性化需求。若使用`plt.plot()`函数绘图，可以设置相关参数值确定图形的线型和颜色。

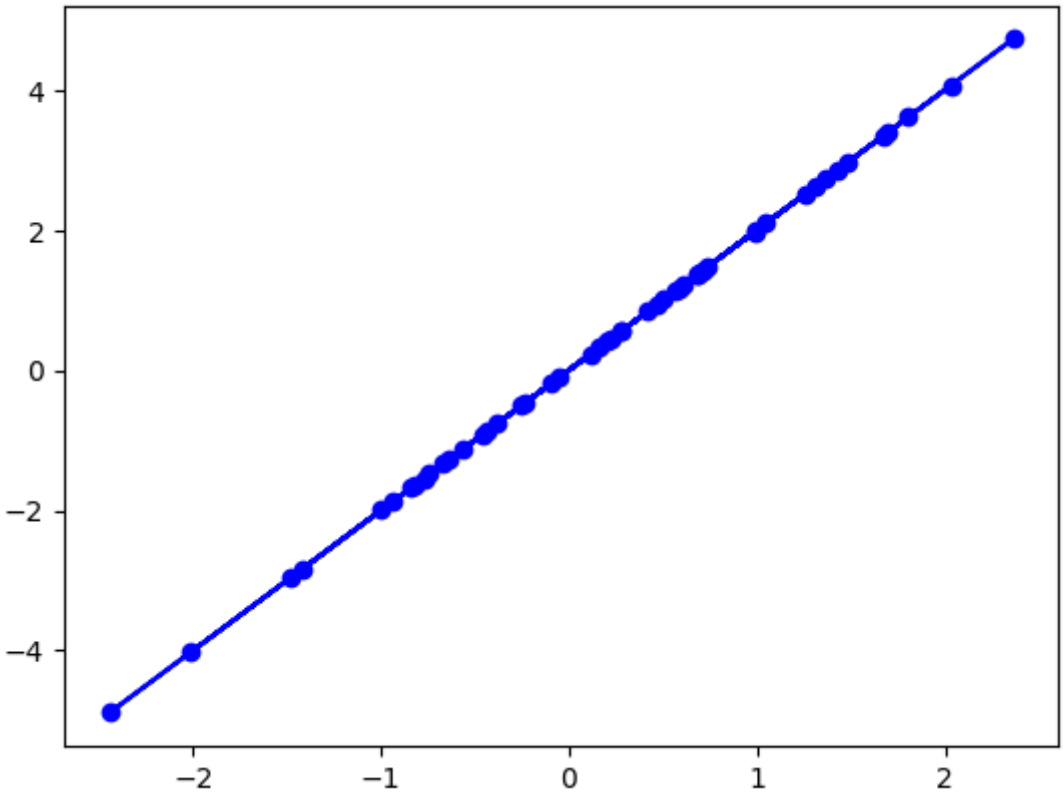
```
plt.plot([x, ]y[, fmt])
```

x和y分别表示绘制在横轴和纵轴上的数据

`fmt`：用一个字符串来定义图形的基本属性，例如颜色（color）、点标记（marker）、线型（linestyle）等等

示例代码：

```
x = np.random.randn(50)
y = x * 2
# 蓝色圆点实线
plt.plot(x, y, 'bo-')
plt.show()
```



若`fmt`接收属性时用的是全名，则应该用关键字参数对耽搁属性赋值，不能使用单字母组合赋值。

2.4设置图形刻度、刻度标签和坐标轴标签等

为了提供更多图形信息，可以在图形中加入许多配置，例如坐标刻度、刻度标签和网格等，这里使用`pyplot`模块的相关函数设置图形刻度范围、刻度标签和坐标轴标签。

- 设置刻度范围

如果未设置坐标轴的范围，那么`matplotlib`默认按照数据的范围自动选择它自己认为合适的区间来展示所有的数据。我们也可以自己来设置数据范围：使用`xlim()`和`ylim()`函数分别设置`x`轴和`y`轴的刻度显示范围

```
x = np.linspace(5, 10, 100)
y = np.random.rand(100)
plt.plot(x, y)
# 设置x轴范围
plt.xlim(4, 11)
# 设置y轴范围
plt.ylim(-0.2, 1.2)
plt.show()
```

- 设置坐标轴刻度值

`xticks()`和`yticks()`函数分别设置 x 轴和 y 轴的刻度值，增加图形的可读性

```
x = np.random.randn(100)
y = x.cumsum()
# 累加和
plt.plot(y)
# 设置x轴的刻度
plt.xticks(np.linspace(0, 100, 5), list('abcde'), fontsize = 15)
# 设置y轴的刻度
# rotation可以设置刻度旋转的角度
plt.yticks(np.linspace(-10, 20, 3), ['max', 'min', 0], fontsize = 15., rotation
= 60)
```

- 设置其他信息

`xlabel()` 和 `ylabel()` 函数分别设置 x 轴和 y 轴的标签文本，`grid()` 函数用于绘制表示刻度线的网格，`axhline()` 函数和 `axvline()` 函数分别绘制平行 x 轴和 y 轴的水平参考线；`axhspan()` 函数和 `axvspan()` 函数分别绘制垂直 x 轴和 y 轴的参考区域。

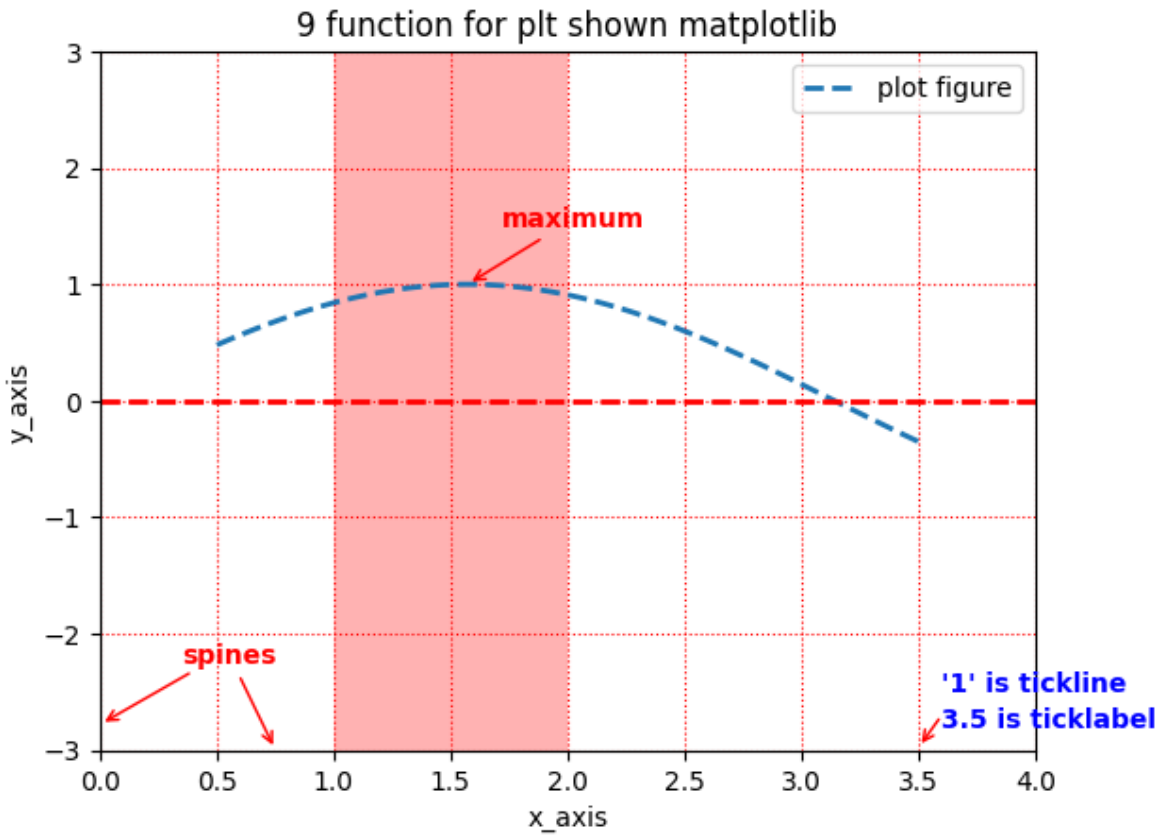
使用图例和注解可以为图形添加与给定数据相关的简短描述，方便用户理解图形的实际意义。`annotate()` 函数为图形内容的细节添加指向型注释文本，用于标记途中的重要细节。

- `text()` 函数为图形内容细节添加无指向型注释文本，用于解释图中的重要细节.
- `title()` 函数为图形内容添加标题.
- `legend()` 函数标识图形中不同变量的文本标签图例.

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

# 生成数据
x = np.linspace(0.5, 3.5, 100)
y = np.sin(x)
# 绘图
plt.plot(x, y, ls = '--', lw = 2, label = 'plot figure')

# 设置刻度范围
plt.xlim(0.0, 4.0)
plt.ylim(-3.0, 3.0)
plt.xlabel('x_axis')
plt.ylabel('y_axis')
# 设置网格线
plt.grid(ls = ':', color = 'r')
# 绘制平行于x轴的水平参考线
plt.axhline(y = 0.0, c = 'r', ls = '--', lw = 2)

# 绘制垂直于y轴的参考区域
plt.axvspan(xmin = 1.0, xmax = 2.0, facecolor = 'r', alpha = 0.3)
plt.annotate('maximum', xy = (np.pi / 2, 1.0), xytext = ((np.pi / 2) + 0.15,
1.5), weight = 'bold', color = 'r', arrowprops = dict(arrowstyle = '- >',
connectionstyle = 'arc3', color = 'r'))
plt.annotate('spines', xy = (0.75, -3), xytext = (0.35, -2.25), weight = 'bold',
color = 'r', arrowprops = dict(arrowstyle = '- >', connectionstyle = 'arc3',
color = 'r'))
plt.annotate('', xy = (0, -2.78), xytext = (0.4, -2.32), weight = 'bold', color
= 'r', arrowprops = dict(arrowstyle = '- >', connectionstyle = 'arc3', color =
'r'))
```

```
plt.annotate('', xy = (3.5, -2.98), xytext = (3.6, -2.7), weight = 'bold', color
= 'r', arrowprops = dict(arrowstyle = '- >', connectionstyle = 'arc3', color =
'r'))
plt.text(3.6, -2.5, "'1' is tickline", weight = 'bold', color = 'b')
plt.text(3.6, -2.8, "3.5 is ticklabel", weight = 'bold', color = 'b')

# 设置图形标题
plt.title("9 function for plt shown matplotlib")
# 设置图例
plt.legend(loc = 'upper right')

plt.show()
```

运行结果如下图所示：



3.matplotlib的ax图像绘图

使用`matplotlib`库绘制图形时，首先使用`figure`方法创建画布，然后图形的绘制可以使用`pyplot`模块的相关函数完成，其实，使用画布中坐标对象`ax`也可以实现队整个图形的绘制。

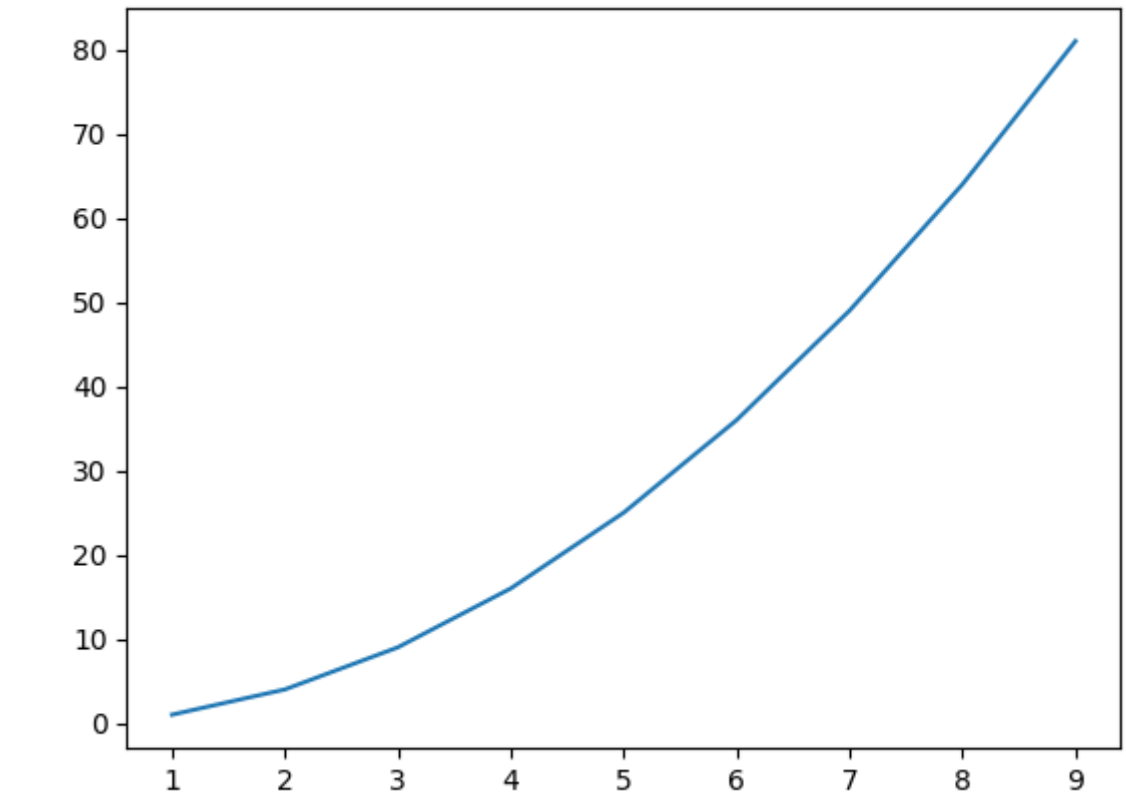
示例代码如下：

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

x = np.arange(1, 10)
y = x ** 2

plt.figure()
plt.plot(x, y)
plt.show()
```

```
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

二者的区别在于：`plt.plot()` 函数首先创建一个 `figure` 对象，然后在这个画布隐式生成的画图区域上绘图；而 `ax.plot()` 方法同时生成了 `fig` 对象和 `ax` 对象，然后使用 `ax` 对象在其区域上绘图。

运行结果如下图所示：



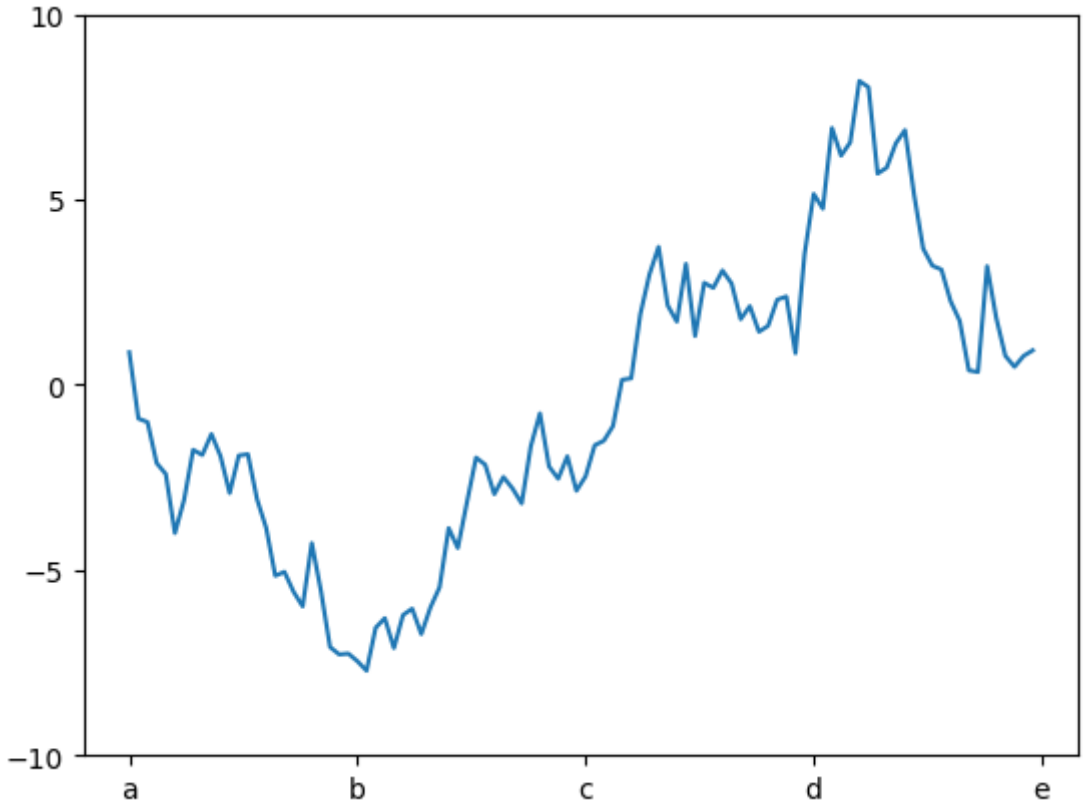
代码将生成两个与上图一模一样的图片

可以使用 `ax` 对象的 `set_xticklabels()` 方法和 `set_yticklabels()` 方法设置刻度标签

示例代码如下：

```
x = np.random.randn(100)
ax = plt.subplot(111)
ax.plot(x.cumsum())
'''
设置x轴和y轴刻度值
功能类似ply模块的xticks()和yticks()函数
'''
ax.set_xticks([0, 25, 50, 75, 100])
ax.set_yticks([-10, -5, 0, 5, 10])
'''
设置刻度标签
可以使用plt模块的xticks()和yticks()函数实现
'''
ax.set_xticklabels(list('abcde'))
plt.show()
```

运行结果如下图所示：

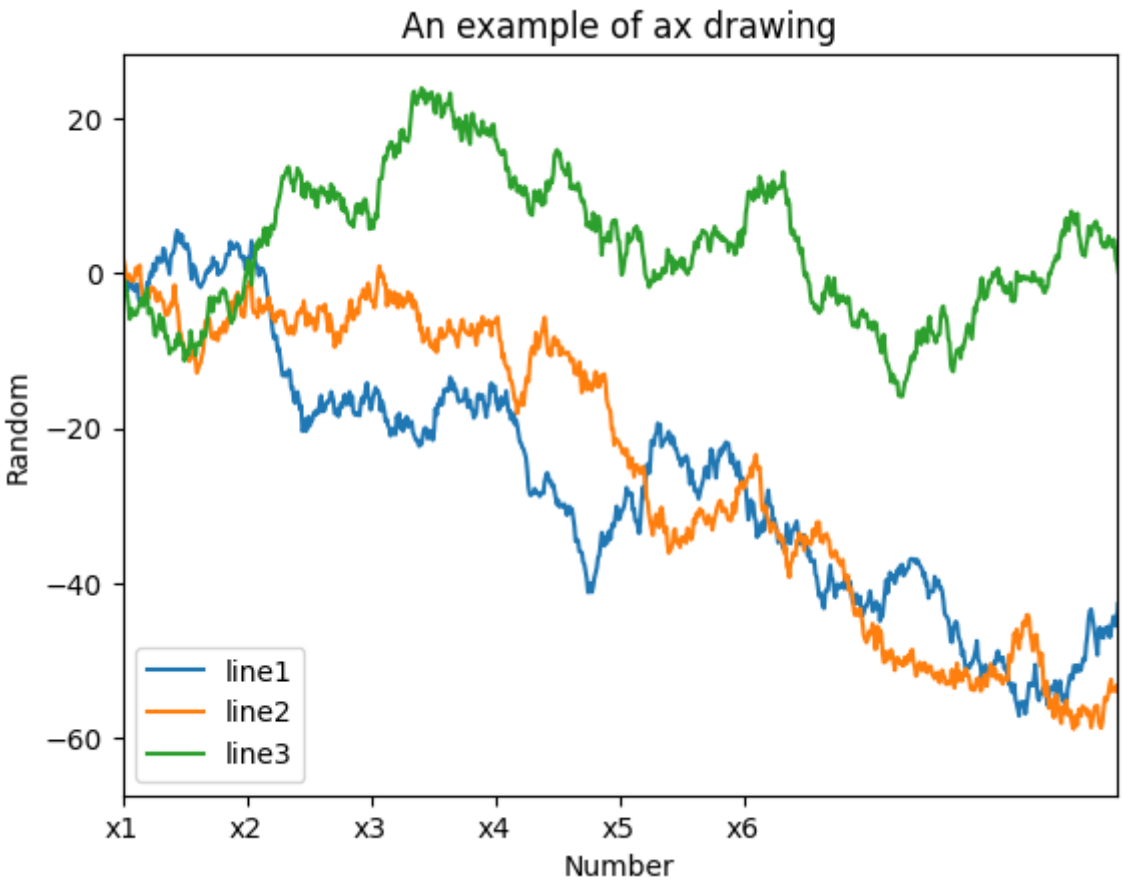


下面使用画布中坐标对象ax的相关方法完成与plt模块绘图相同的功能，代码如下：

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

# 生成数据
data1 = np.random.randn(1000).cumsum()
data2 = np.random.randn(1000).cumsum()
data3 = np.random.randn(1000).cumsum()

fig, ax = plt.subplots(1)
ax.plot(data1, label = 'line1')
ax.plot(data2, label = 'line2')
ax.plot(data3, label = 'line3')

# 设置刻度范围，功能类似plt模块的xlim()和ylim()函数
ax.set_xlim([0, 800])

# 设置刻度值，功能类似于plt模块的xticks()和yticks()函数
ax.set_xticks([0, 100, 200, 300, 400, 500])

# 设置刻度标签，可以使用plt模块的xticks()和yticks()函数实现
ax.set_xticklabels(['x1', 'x2', 'x3', 'x4', 'x5', 'x6'])

# 设置坐标轴标签，功能类似plt模块的xlabel()和ylabel()函数
ax.set_xlabel('Number')
ax.set_ylabel('Random')

# 设置标题，功能类似plt模块的title()函数
ax.set_title('An example of ax drawing')

# 图例
```

```
ax.legend(loc = 3)
plt.show()
```

运行结果如下图所示：



4.matplotlib绘制子图

默认情况下，`matplotlib`使用整个绘图区域绘制图形，多个图形叠加并共用同一套坐标系统。然而，优势我们需要将整个绘图区域划分为不同的子区域，在每个子区域分别绘制不同的图形，并且每个子区域使用各自独立的坐标系统。

我们使用`matplotlib.pyplot`中的`subplot()`函数划分绘图区域并创建子图，语法如下：

> subplot(numbRow, numbCol, plotNum)
>
> 或者：
>
> subplot(numbRow numbCol plotNum)

- numbRow：表示绘图区域中子图的行数
- numbCol：表示绘图区域中子图的列数
- plotNum：指定Axes对象所在的绘图区域

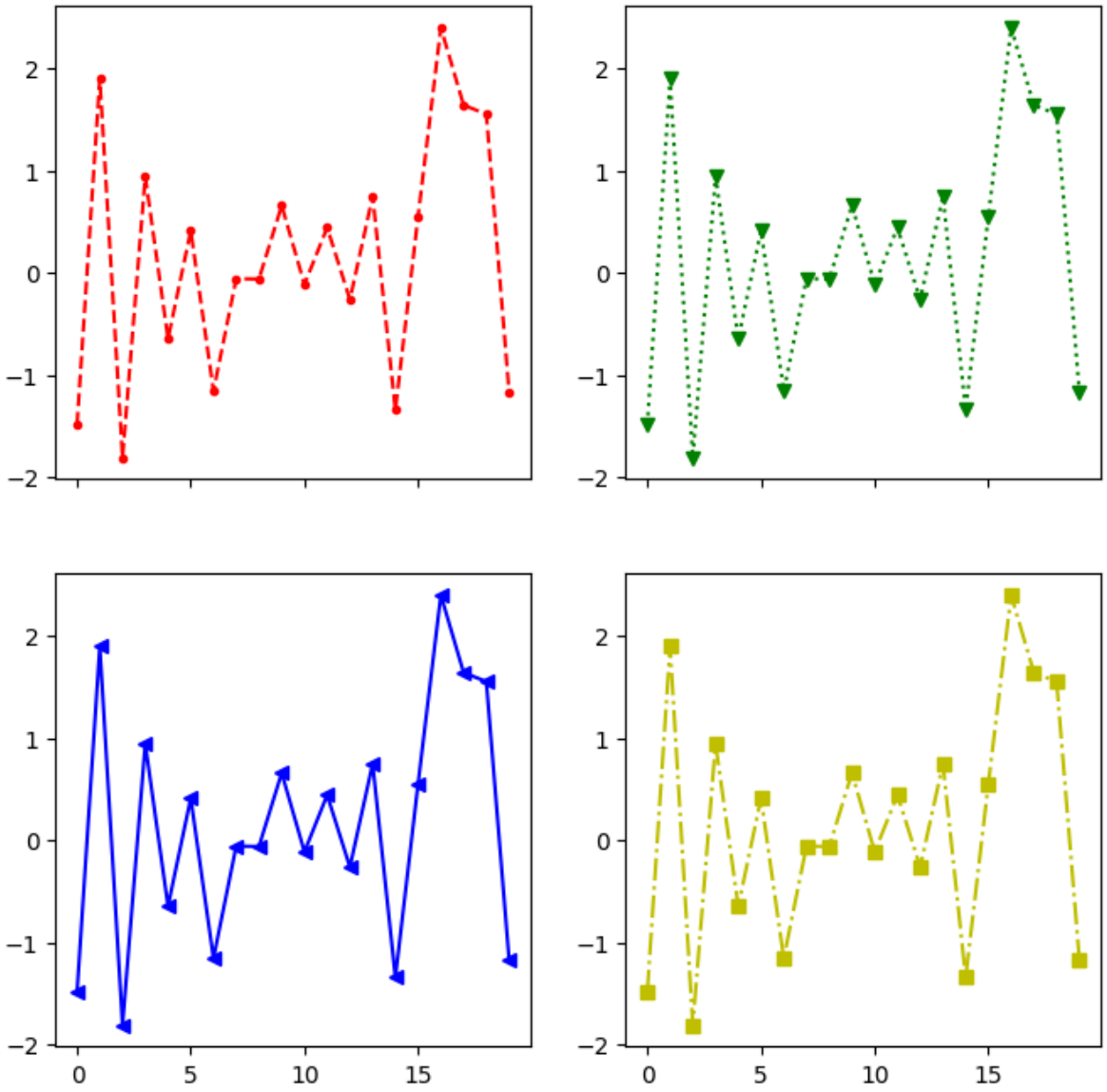
此外，`subplot()`函数可以使用可选参数设置图形的更多属性

`subplot()`函数的返回值包括一个绘图区和一系列Axes实例对象组成的数组，将整个绘图区域划分为numRows行和numbCols列个子区域，按照从左到右、从上到下的顺序队每个子区域一次编号。用户可以方便地访问每一个Axes对象，绘制并独立配置每一个子图。例如，函数`subplot(2,2,1)`表示将绘图区域划分为2x2个子区域，也就是共有4个子图。其中，1代表第一幅图所在的子区域，即左上方的子区域编号为1，也可以写成`subplot(221)`。

示例代码如下：

```
fig, subplot_arr = plt.subplots(2, 2, figsize = (8, 8), sharex = True)
data = np.random.randn(20)
subplot_arr[0, 0].plot(data, '--r.')
subplot_arr[0, 1].plot(data, 'gv:')
subplot_arr[1, 0].plot(data, 'b<-')
subplot_arr[1, 1].plot(data, 'ys-.')
```

运行结果如下图所示：

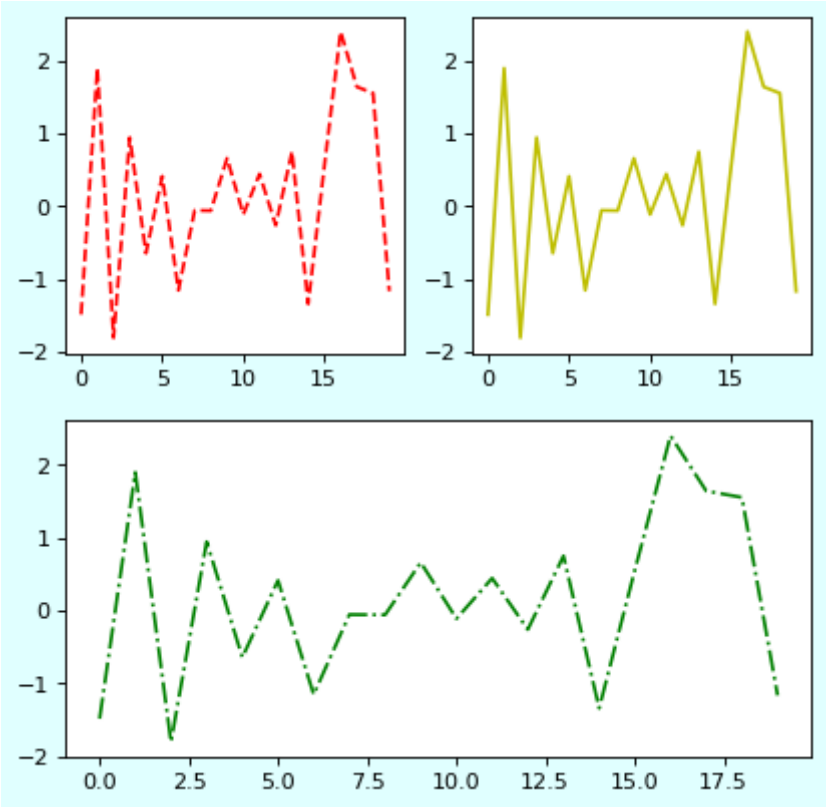


如果绘制的图形包含3幅图或者5幅图的话，需要对编号较大的子区域进行重新排列。例如，当绘制一幅包含3个子图的图形时，如果将整个绘图区按照2x2划分，那么前两个子区域分别是$(2,2,1)$和$(2,2,2)$，第三个子图需要占用$(2,2,3)$和$(2,2,4)$这两个子区域，因此需要再次使用`subplot()`函数对这个子区域按照2x1重新划分，使得前两个子图占用$(2,1,1)$的区域，第三个子图占用$(2,1,2)$的区域。

示例代码如下：

```
import matplotlib.pyplot as plt

plt.figure(figsize = (6, 6), dpi = 80, facecolor = 'lightcyan')
plt.figure(1)
ax1 = plt.subplot(221)
plt.plot(data, color = 'r', linestyle = '--')
ax1 = plt.subplot(222)
plt.plot(data, color = 'y', linestyle = '-')
ax1 = plt.subplot(212)
plt.plot(data, color = 'g', linestyle = '-.')
```

运行结果如下图所示：



5.matplotlib中文字体的显示

使用matplotlib绘制的图形在显示中文时经常出现乱码问题。这是因为matplotlib默认安装和使用的sans-serif字体并不支持中文显示，当用户将图形标题、标签或图例设置为中文字符串时，这些中文字体无法正常显示出来。针对该问题有两种解决方案：

方案一：使用matplotlib中的reParams属性设置

在运行绘图命令之前，添加下列代码：

```
# 设置中文字体为黑体，中文正常显示
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
```

方案二：使用matplotlib的`font_manager`工具

这里主要分为三个步骤：

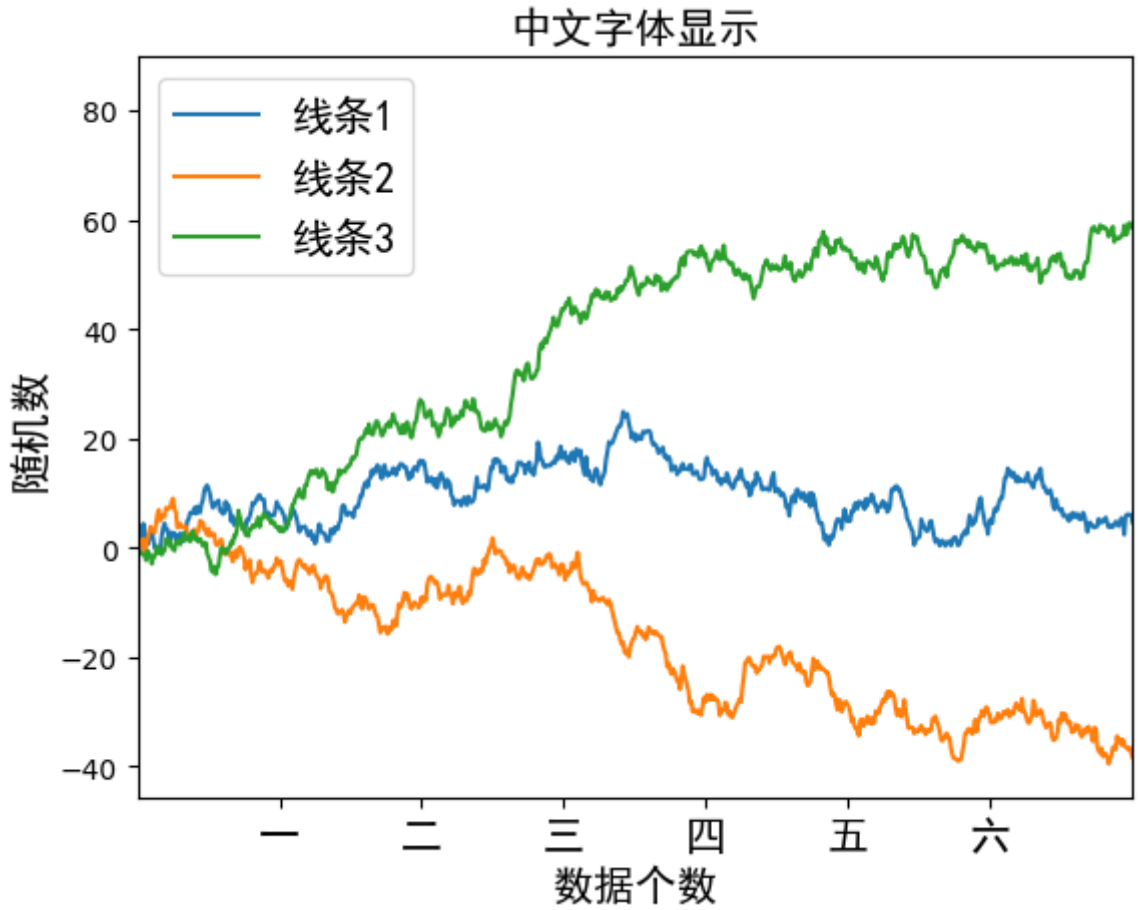
- 首先，导入`font_manager`模块；
- 然后，找到中文字体在计算机中的位置，导入中文字体库
- 接着，根据绘图时中文字体出现的不同位置，采取适当的解决方案。例如，在图例中出现中文字体，就去设置`legend`函数的`prop`属性；标题或者横纵坐标出现中文，就去设置对应函数的`fontproperties`属性；若显示希腊字母，则需要采用`latex`格式。

示例代码：

```
# 生成数据
data1 = np.random.randn(1000).cumsum()
data2 = np.random.randn(1000).cumsum()
data3 = np.random.randn(1000).cumsum()
# 导入库
import matplotlib.font_manager as fm
# 定位中文字体文件
zhfont1 = fm.FontProperties(fname = "C:/Windows/Fonts/simhei.ttf", size = 15)
# 绘图
plt.figure()
plt.plot(data1, label = "线条1")
plt.plot(data2, label = "线条2")
plt.plot(data3, label = "线条3")
# 设置刻度
plt.xlim([0, 700])
# 设置刻度标签：中文字体
plt.xticks((100, 200, 300, 400, 500, 600), ('一', '二', '三', '四', '五', '六'),
fontproperties = zhfont1)
# 设置坐标轴标签：中文字体
plt.xlabel('数据个数', fontproperties = zhfont1)
plt.ylabel('随机数', fontproperties = zhfont1)
# 设置中文标题
plt.title('中文字体显示', fontproperties = zhfont1)
# 图例
plt.legend(loc = 'upper left', prop = zhfont1)
```

这里注意，中文字体文件一般在`C:/Windows/Fonts`目录下

上述代码运行结果如下图所示：



二、定性数据可视化

1.条形图

条形图或柱状图适用于中小规模的二维数据集，且只有一个维度的数据需要进行比较的情况。条形图时用同宽度条形的高度或长短来表示数据变动的图形。绘制条形图时，各类别可以放在纵轴，称为条形图；也可以放在横轴，称为柱状图。

函数`bar()`可以根据给定的数据绘制条形图或柱状图，语法格式如下：

```
bar(left, height, width = 0.8, bottom = None, hold = None, data = None, *
*kwargs)
```

- left表示每根柱子的横坐标
- height表示每根柱子的高度，即待比较数据的统计值
- width：表示每根柱子的宽度，默认为0.8
- bottom：表示每根柱子底部边框的y坐标值

函数`bar()`的可选参数如下：

- color：表示每根柱子的颜色，可以指定一个颜色值，使所有的主子呈现相同的颜色；也可以指定包含不同颜色的列表，使柱子呈现不同的颜色
- edgecolor：表示每根柱子边框的颜色
- linewidth：表示每根柱子边框的宽度，默认为无边框
- align：表示每根柱子的对齐方式，以柱状图为例：参数align = 'edge'且width > 0表示柱子左侧边框与给定的横坐标对齐；参数align = 'edge'且width < 0表示柱子右侧边框与给定的横坐标对齐；align‘ = 'center'表示给定的横坐标对齐柱子的中间位置。

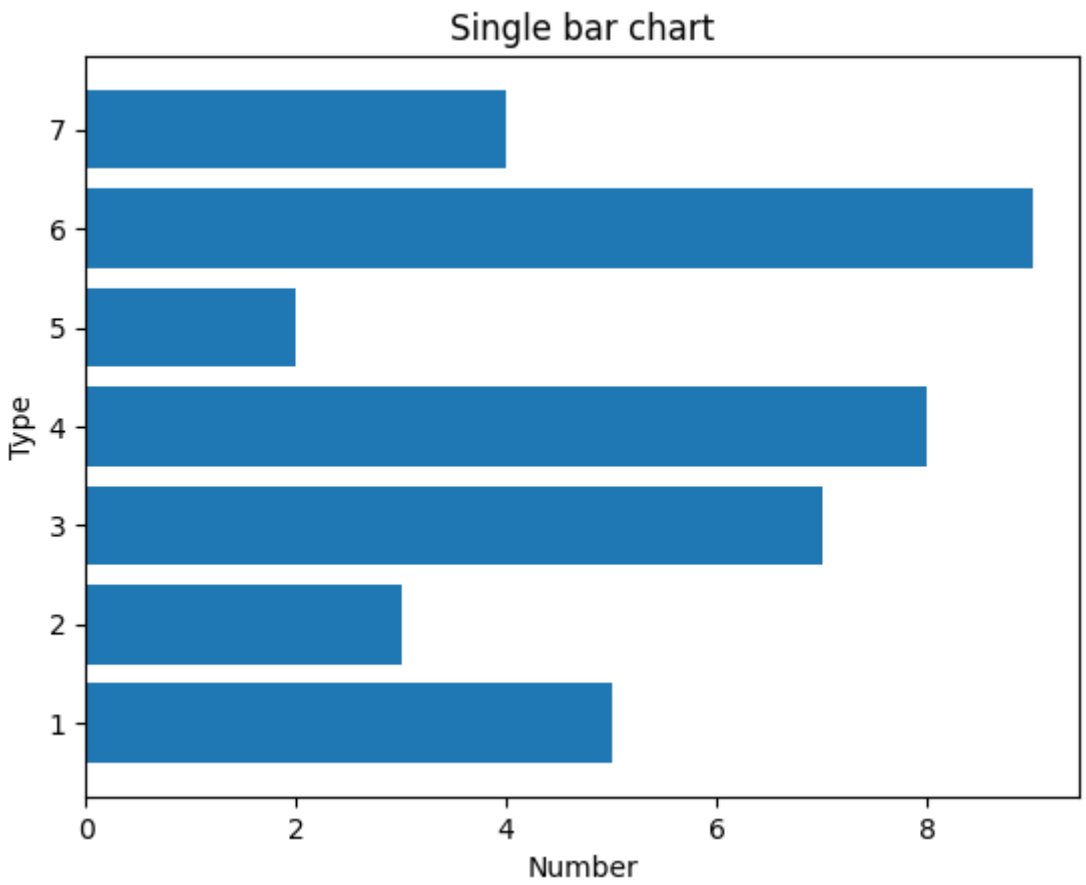
- orientation：设置柱子的显示方式，当orientation = 'vertical'时绘制柱状图；当orientation = 'horizontal'时绘制条形图。但是不建议设置此参数绘制条形图，绘制条形图一般采用函数`barh()`
- alpha：设置柱子的透明度
- antialiased：设置是否启用抗锯齿功能
- fill：设置是否填充
- hatch：指定内部填充符号，可选值有`/ \\ | - + x o O . *`
- label：指定图例中显示的文本标签
- linestyle：指定边框的类型
- visible：设置绘制的柱子是否可见

下面给出两段示例代码：

单式条形图

```
x = [1, 2, 3, 4, 5, 6, 7]
data = [5, 3, 7, 8, 2, 9, 4]
plt.barh(x, data)

plt.xlabel('Number')
plt.ylabel('Type')
plt.title('Example of single bar chart')
plt.show()
```

运行结果如图所示：

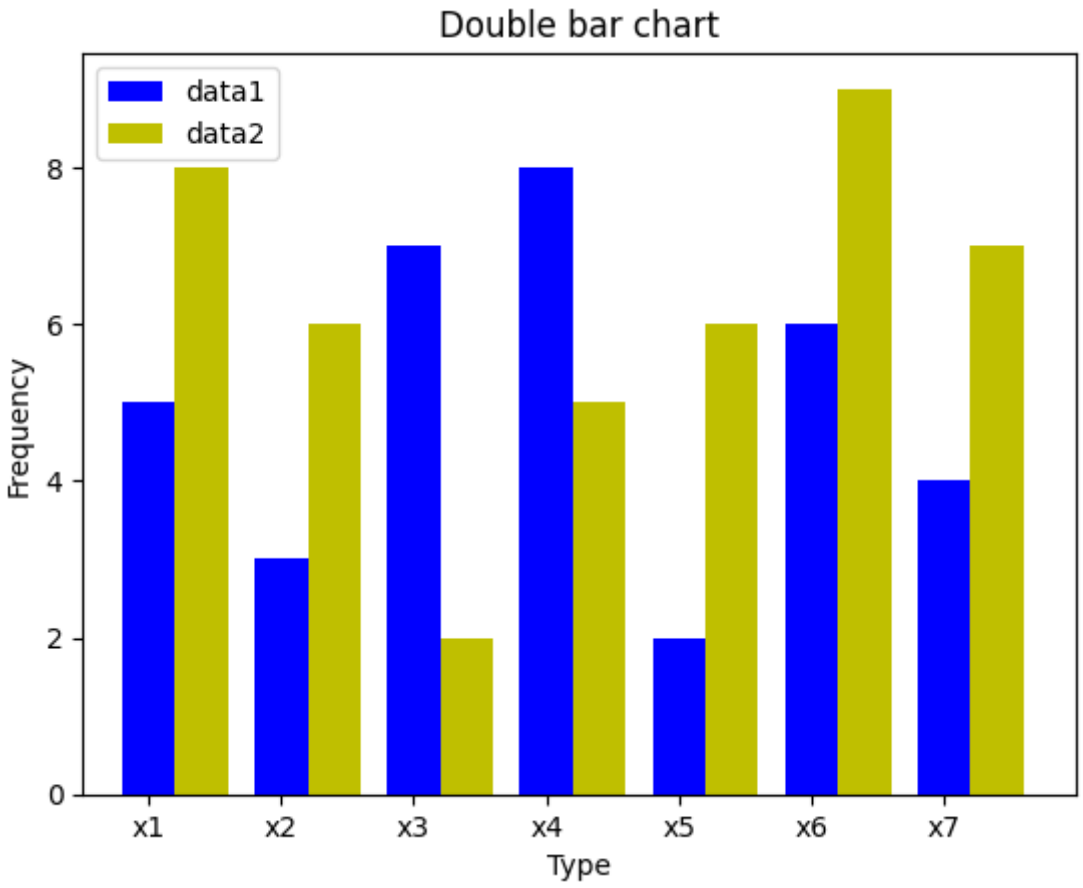


复式条形图

```
x1 = [1, 3, 5, 7, 9, 11, 13]
data1 = [5, 3, 7, 8, 2, 6, 4]
plt.bar(x1, data1, color = 'b', label = 'data1')
```

```
x2 = np.array(x1) + 0.8
data2 = [8, 6, 2, 5, 6, 9, 7]
plt.bar(x2, data2, color = 'y', label = 'data2')

plt.xlim([0, 15])
plt.xticks([1, 3, 5, 7, 9, 11, 13], ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7'])

plt.xlabel('Type')
plt.ylabel('Frequency')
plt.title('Double bar chart')
plt.legend()
plt.show()
```

运行结果如图所示：

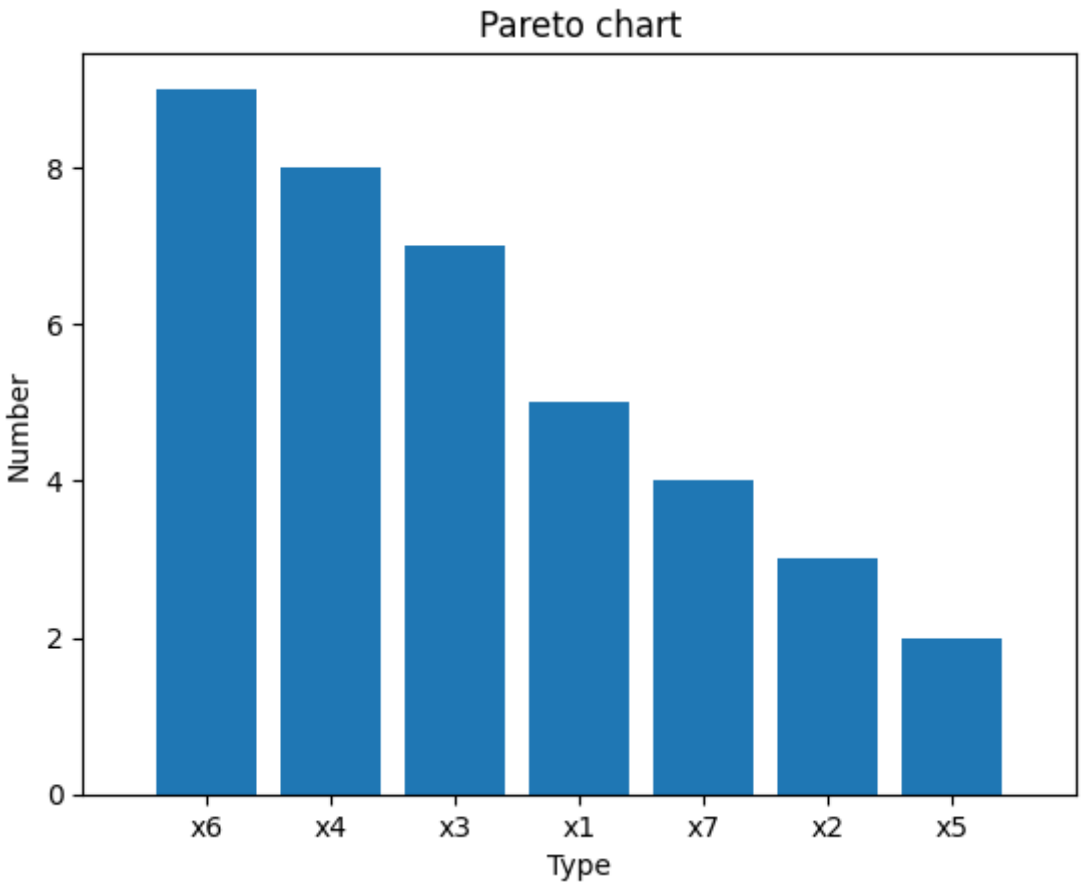


2.帕累托图

帕累托图是按照各类别数据出现的频率高低排序后绘制的条形图。通过条形的排序，很容易看出哪类数据出现较多，哪类数据出现较少。

```
x = [1, 2, 3, 4, 5, 6, 7]
data = [5, 3, 7, 8, 2, 9, 4]
data1 = sorted(data, reverse = True)
plt.bar(x, data1)

# 设置刻度范围
plt.xlim([0, 8])
plt.xticks([1, 2, 3, 4, 5, 6, 7], ['x6', 'x4', 'x3', 'x1', 'x7', 'x2', 'x5'])
plt.xlabel('Type')
plt.ylabel('Number')
plt.title('Pareto chart')
plt.show()
```

运行结果如图所示：



3.饼图

饼图是用圆形以及扇形的角度来表示数值大小的图形，主要用于表示一个样本（或总体）中各组成部分的数据占全部数据的比例，可以使用matplotlib库中的`pie()`函数，根据给定的数据绘制饼图，其用法如下：

```
pie(x, explode = None, labels = None, colors = None, autopct = None, pctdistance
= 0.6, shadow = False, labeldistance = 1.1, startangle = None, radius = None,
counterclock = True, wedgeprops = None, textprops = None, center = (0, 0), frame
= False, hold = None, data = None)
```

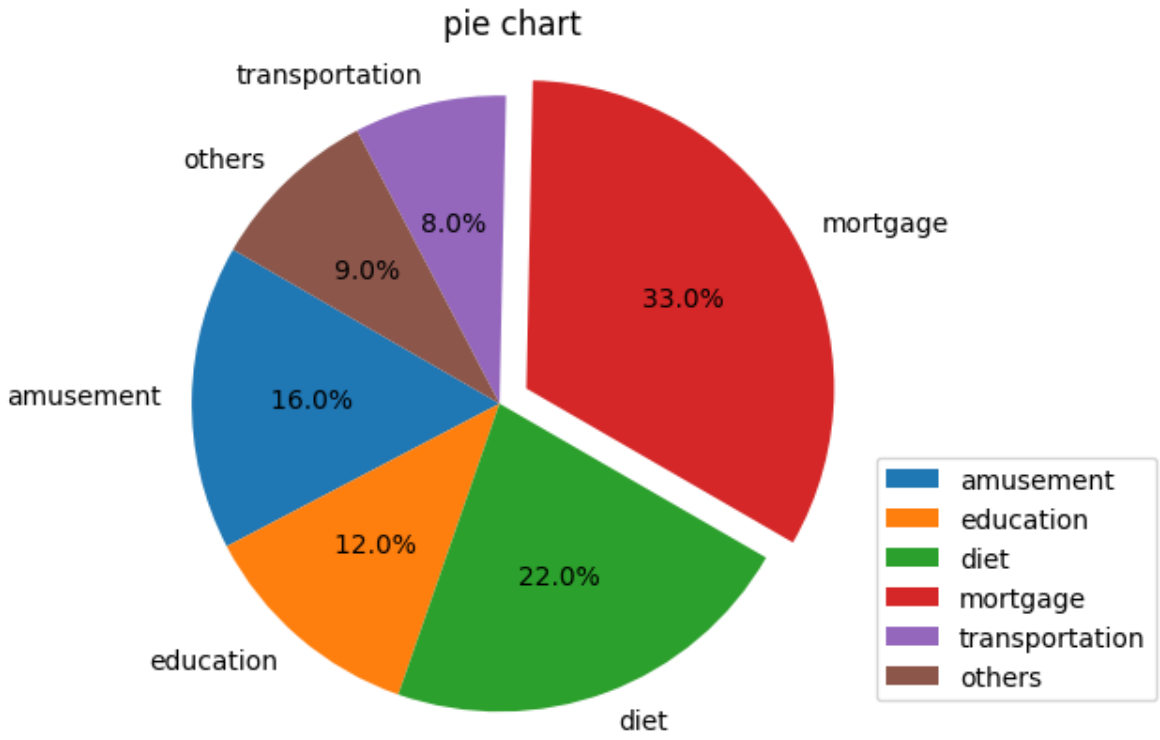
参数含义如下表所示：

参数名称	含义
x	取值为数据序列，自动计算每个数据占的百分比并确定对应的扇形面积
explode	取值为None或者与x登场的数据序列，用于指定每个扇形沿半径方向偏离圆心的距离，取值为None表示无偏移，正数表示远离圆心
colors	取值为None或元素为颜色值的数据序列，指定每个扇形的颜色；如果颜色值序列长度小于扇形数量，则循环使用颜色值
labels	与x长度相同的字符串序列，用于指定每个扇形对应的文本标签
autopct	设置扇形内部标签的显示格式，一般为一个格式化字符串，表示标签显示为数据在总体中占有的百分比
pctdistance	设置每个扇形中心与autopct指定文本的距离，默认值为0.6
labeldistance	设置每个饼标签与圆心的径向距离，1.1表示1.1倍半径
shadow	布尔类型的数据，设置是否显示阴影，默认值是False
startangle	设置第一个扇形的起始角度，沿x轴逆时针方向计算，默认值是0
radius	设置饼的半径，默认值为1
counterclock	布尔类型的数据，设置饼图中每个扇形的绘制方向
center	形式为(x,y)的元组数据，设置饼的圆心位置
frame	布尔类型的数据，显示是否显示边框

示例代码如下：

```
labels = ['amusement', 'education', 'diet', 'mortgage', 'transportation',
'others']
sizes = [16, 12, 22, 33, 8, 9]
explode = (0, 0, 0, 0.1, 0, 0)
plt.pie(sizes, explode = explode, labels = labels, autopct = '%1.1f% % ', shadow
= False, startangle = 150)

plt.title("pie chart")
plt.axis('equal')

plt.legend(loc = "lower right", fontsize = 10, bbox_to_anchor = (1.2, 0.05),
borderaxespad = 0.3)
plt.show()
```

运行结果如图所示：



4.环形图

饼图只能显示一个样本中各个部分的比例。如果要显示多个样本的构成比例，可以把多个饼图叠加在一起，挖去中间的部分，这就形成了环形图。环形图中间为空，每个样本用一个环来表示，样本中的每一部分数据用环中的一段来表示。环形图可以显示多个样本各部分所占的比例，有利于展示构成的比较研究结果。

`pie()`函数可以绘制饼图，也可以绘制环形图，主要参数如下：

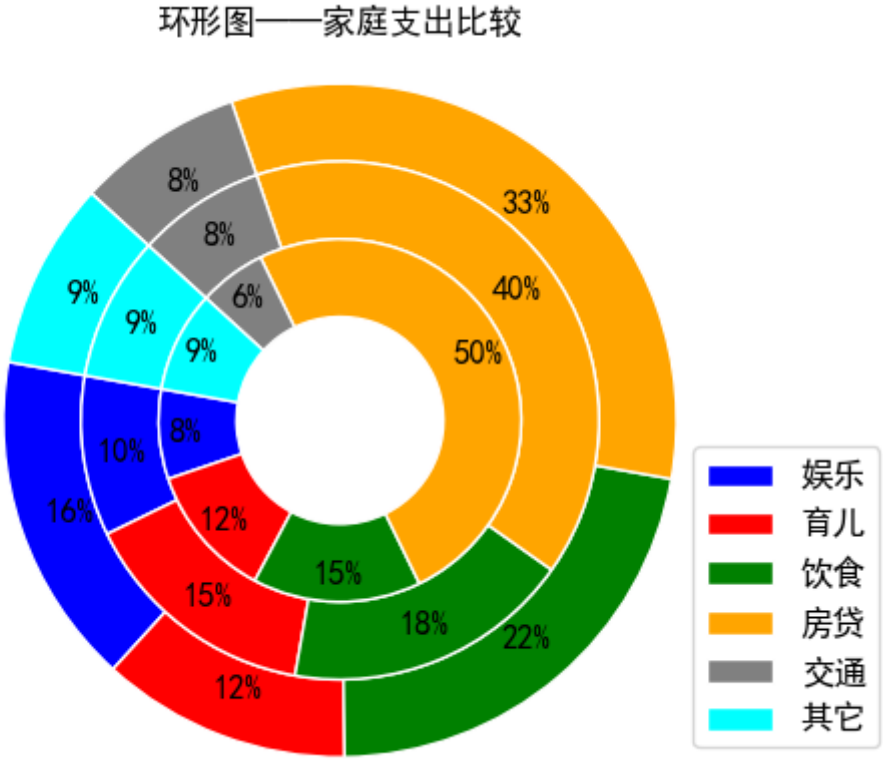
- radius：表示圆环的半径，据此区分不同的圆环
- pctdistance：表示每个扇形中心与autopct指定文本的距离
- textprops：表示标签文本的颜色，值为字典类型
- wedgeprops：表示环形的宽度和边框颜色，值为字典类型

示例代码

```
# 不同年份的家庭支出构成比较研究
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm

plt.rcParams['font.sans-serif'] = ['SimHei']  # 用黑体显示中文
plt.rcParams['axes.unicode_minus'] = False  # 解决保存图像是负号'-'显示为方块的问题

labels = ['娱乐', '育儿', '饮食', '房贷', '交通', '其它']
sizes3 = [16, 12, 22, 33, 8, 9]
sizes2 = [10, 15, 18, 40, 8, 9]
sizes1 = [8, 12, 15, 50, 6, 9]
colors = ('blue', 'red', 'green', 'orange', 'gray', 'cyan')
explode = (0, 0, 0, 0, 0, 0)

plt.figure()
_, texts3, autotexts3 = plt.pie(sizes3, explode=explode, radius=1.3,
autopct='%1.0f%%', pctdistance=0.85, colors=colors,
```

```
                                    shadow=False, startangle=170, wedgeprops=
{'width': 0.3, 'edgecolor': 'w'})
_, texts2, autotexts2 = plt.pie(sizes2, explode=explode, radius=1.0,
autopct='%1.0f%%', pctdistance=0.85, colors=colors,
                                    shadow=False, startangle=170, wedgeprops=
{'width': 0.3, 'edgecolor': 'w'})
_, texts1, autotexts1 = plt.pie(sizes1, explode=explode, radius=0.7,
autopct='%1.0f%%', pctdistance=0.85, colors=colors,
                                    shadow=False, startangle=170, wedgeprops=
{'width': 0.3, 'edgecolor': 'w'})

# 相等的纵横比确保饼图绘制为圆形
plt.axis('equal')

# 重新设置字体大小
proptease = fm.FontProperties(fname = "C:/Windows/Fonts/simhei.ttf",
size='large')

# 应用字体设置到文本和自动文本上
plt.setp(texts1 + texts2 + texts3, fontproperties=proptease)
plt.setp(autotexts1 + autotexts2 + autotexts3, fontproperties=proptease)

# 修正标题字体属性引用
plt.title("环形图--家庭支出比较", fontproperties=proptease)

# 图例
plt.legend(
    loc="lower right",
    labels=labels,
    fontsize=10,
    bbox_to_anchor=(1.05, 0.05),
    borderaxespad=0.2,
    prop=proptease
)

plt.show()
```

运行结果如图所示：



三、定量数据可视化

以上品质数据的可视化方法同样适用于定量数据的可视化。但是数值型数据还有一些特有的展示方法，它们不适用于分类数据和顺序数据。对于数值型单变量数据可视化，常用直方图、茎叶图和箱线图等。对于两个及以上变量，可以采用多变量数据可视化方法，常用散点图、气泡图和雷达图等。

1.直方图

直方图是用于展示连续型数据分布特征的统计图形，它用矩形的宽度和高度（即面积）表示频数分布。在平面直角坐标系中，用横轴表示数据分组，纵轴表示频数或频率，这样，每个数据分组与相应的频数就形成了一个矩形，即直方图。

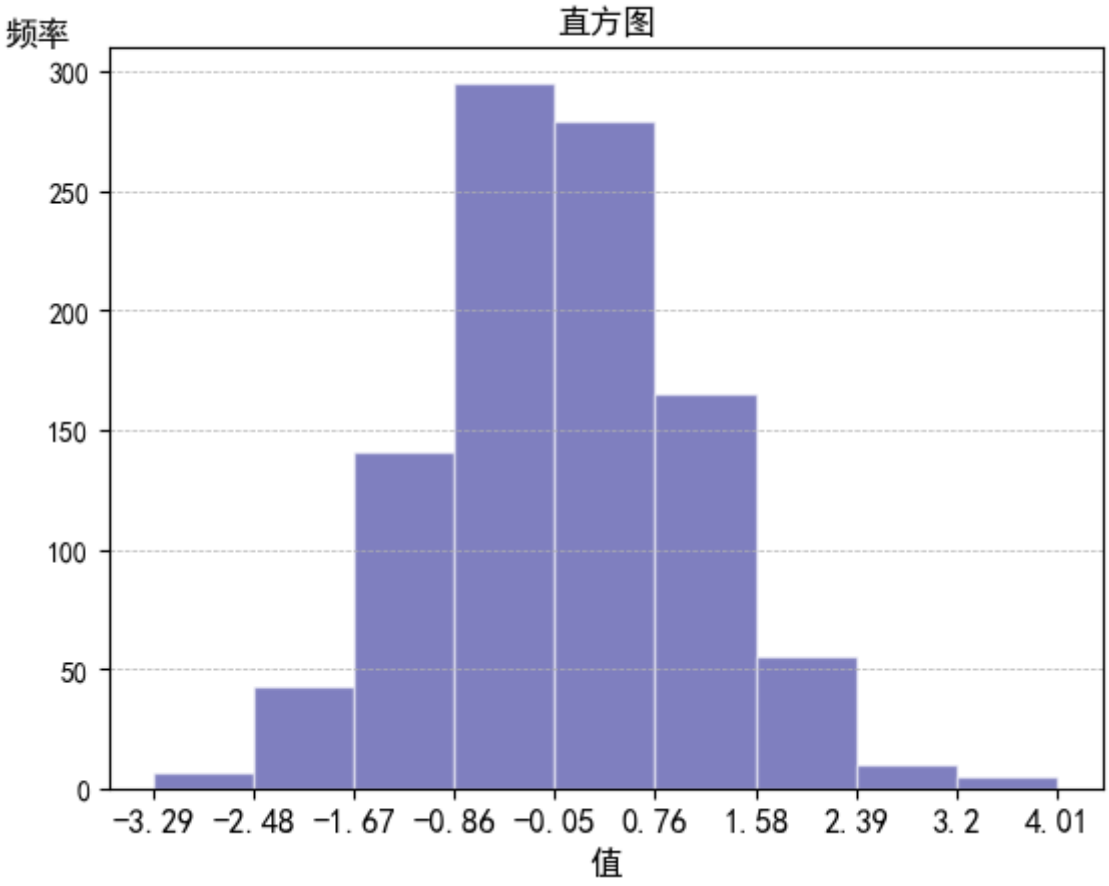
直方图与条形图不同，主要有以下三点：

- 首先，条形图使用条星的长度表示个类别频数的多少，宽度是固定的；直方图是用面积表示各组频数的多少，矩形的高度表示每一组的频数或频率，宽度表示各组的组距
- 其次，由于分组数据具有连续性，直方图的各矩形通常连续排列，而条形图是分开排列
- 最后，条形图主要用于展示品质数据，而直方图主要用于展示数值型数据

示例代码如下：

```
# 1000个正态分布的数据
x = np.random.randn(1000)
bins = np.around(np.linspace(min(x), max(x), 10, endpoint = True), 2)

h = plt.hist(x, bins = bins, color = 'navy', edgecolor = 'white', alpha = 0.5)
plt.xlabel('值', fontproperties = proptease)
plt.ylabel('频率', rotation = 'horizontal', y = 1, fontproperties = proptease)
plt.xticks(bins, bins, fontproperties = proptease)
# 网格线
plt.grid(axis = 'y', linewidth = 0.5, linestyle = '--')
plt.title("直方图", fontproperties = font)
plt.show()
```

运行结果如下图所示：



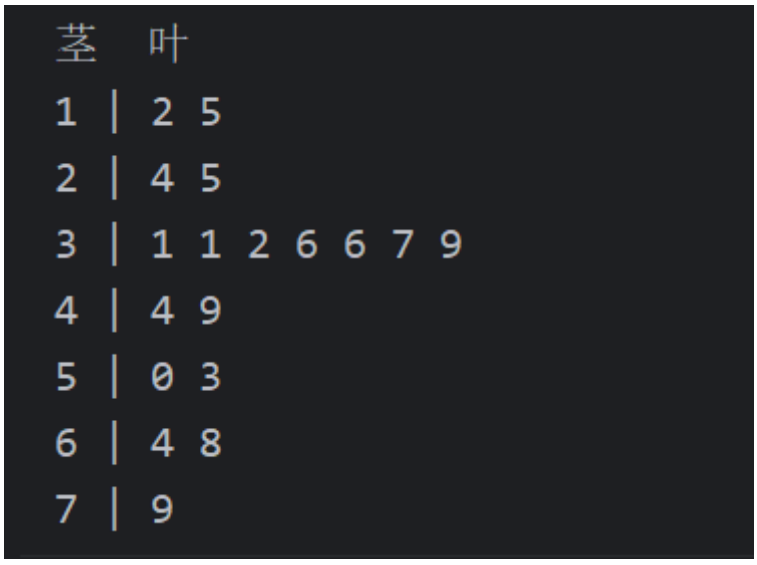
2.茎叶图

对于未分组的数值型数据，可以用茎叶图或箱线图来表示。茎叶图是反应原始数据分析特征的图形。它由茎和叶两部分组成，其图形是由数字组成的。通过茎叶图，可以看出数据的分布形状以及离散状况，例如分布是否对称、数据是否集中、是否有了离散点等等。制作茎叶图时，首先把一个数值分成两部分，通常以该组数据的高位数值作为树茎，叶子上保留该数值的最后一个数字。

Python不能直接调用函数来绘制茎叶图，这里自定义一个`stem()`函数来绘制茎叶图：

```
import math
from itertools import groupby
# 参数data为数据集，n = 10表示以个位数为叶子
# lambda x为一个匿名函数，x表示sorted(data)中的元素
# math.floor(x / n)表示取x的百位和十位数，并向下取整作为茎；lst中为数据的个位数，作为叶子
def stem(data, n):
    for k, g in groupby(sorted(data), key = lambda x : math.floor(x / n)):
        lst = map(str, [d % n for d in list(g)])
        print(k, '|', ''.join(lst))
# 调用函数
weight = [12, 15, 24, 25, 53, 64, 31, 31, 36, 36, 37, 39, 44, 49, 50, 79, 68,
32, ]
print('茎', '', '叶 ')
stem(weight, 10)
```

运行结果如下图所示：



可见，茎叶图保留了数据的全部信息，能够直观地显示出数据的分布状况。如果将茎叶图逆时针旋转90°，可以得到一个类似于频数直方图的图形。与横置的直方图相比，茎叶图在展示数据分布状况的同时，完整保留了原始数据的信息。直方图适合展示数据的分布，但是无法保留每一个原始数据。因此，直方图适用于大批量数据的展示，茎叶图更适合展示小批量数据。

3.箱线图

箱线图由一组数据的最大值、最小值、中位数、两个四分位数这5个特征值绘制而成，主要反应原始数据的分布特征。对于多组数据，可以将个组数据的 箱线图并列起来，进行多组数据分布特征的比较研究。matplotlib的函数`boxplot()`可以根据给定的数据绘制箱线图，语法格式如下：

```
boxplot(x, notch = None, sym = None, vert = None, whis = None, positions = None,
widths = None, patch_artist = None, bootstrap = None, usermedians = None,
conf_intervals = None, meanline = None, showmeans = None, showcaps = None,
showbox = None, showfliers = None, boxprops = None, labels = None, flierprops =
None, medianprops = None, meanprops = None, capprops = None, whiskerprops =
None, manage_xticks = None, autorange = False, zorder = None, hold = None, data
= None)
```

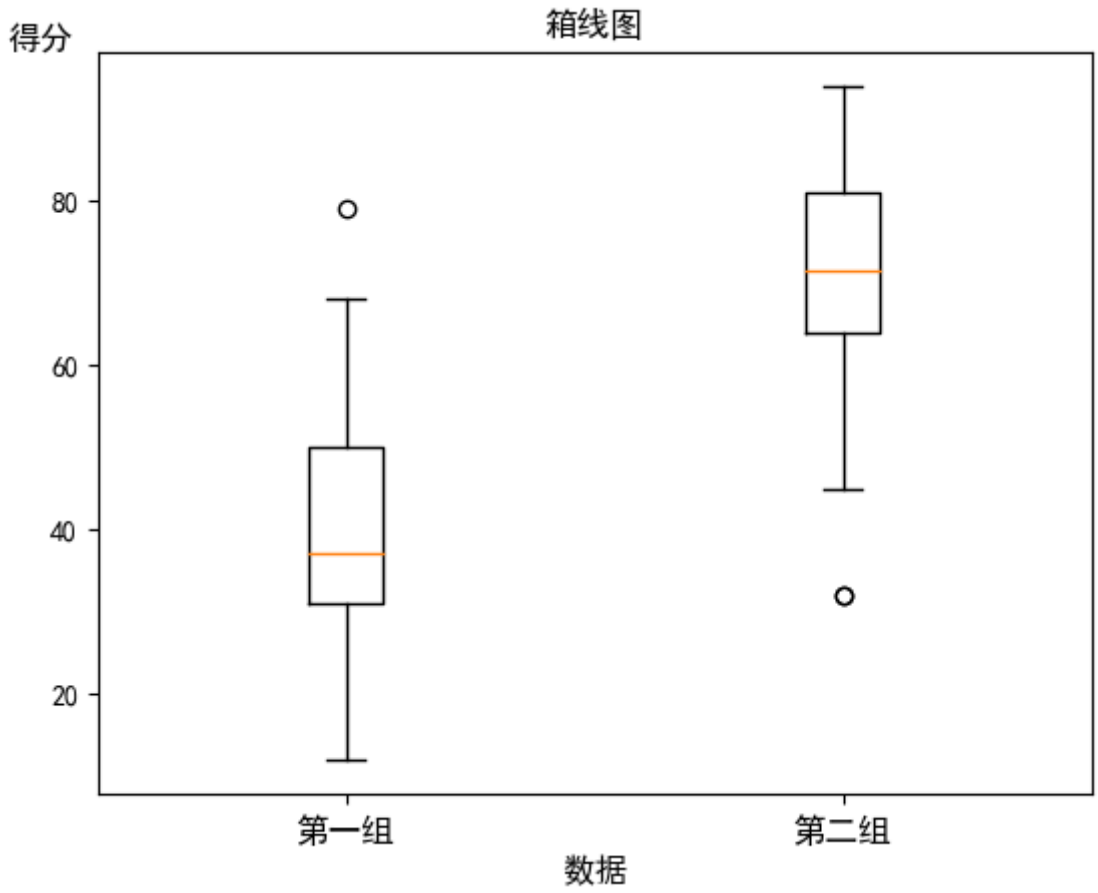
参数及其含义如下表所示：

参数名称	含义
x	指定要绘制箱线图的数据
notch	是否以凹口形式展示箱线图
sym	指定异常点的形状，默认为圆圈展示
vert	是否将箱线图垂直摆放，默认值为True，即垂直摆放
whis	指定上下须与上下四分位的距离，默认值为1.5倍的四分位差
positions	指定箱线图的位置
widths	指定箱线图的宽度
patch_artist	是否填充箱体的颜色
meanline	是否用线的形式表示均值
showmeans	是否显示均值
showfliers	是否显示异常值，默认显示
showbox	是否显示箱线图的箱体
showcaps	是否显示箱线图顶端和末端的两条线
boxprops	设置箱体的属性，如边框色，填充色等
capprops	设置箱线图顶端（最大值）和末端（最小值）线条的属性
filerprops	设置异常值的属性
filerprops	设置中位数的属性
meanprops	设置均值的属性
labels	为箱线图添加标签
whiskerprops	设置须的属性

示例代码如下：

```
weight1 = [12, 15, 24, 25, 53, 64, 31, 31, 36, 37, 39, 44, 49, 50, 79, 68, 32]
weight2 = [32, 45, 64, 65, 73, 64, 81, 81, 86, 86, 77, 69, 94, 79, 90, 79, 68,
32, 70, 46]
weight = [weight1, weight2]
labels = ['第一组', '第二组']
plt.boxplot(weight, labels = labels)
plt.xticks(fontproperties = proptease)
plt.xlabel('数据', fontproperties = proptease)
plt.ylabel('得分', rotation = 'horizontal', y = 1, fontproperties = proptease)
plt.title('箱线图', fontproperties = proptease)
plt.show()
```

运行结果如图所示：



4.折线图

折线图比较适合刻画多组数据随时间变化的趋势，或描述一组数据对另一组数据的依赖关系。matplotlib.pyplot中的 `plot()` 函数可以根据给定的数据绘制折线图，其常用参数如下表所示：

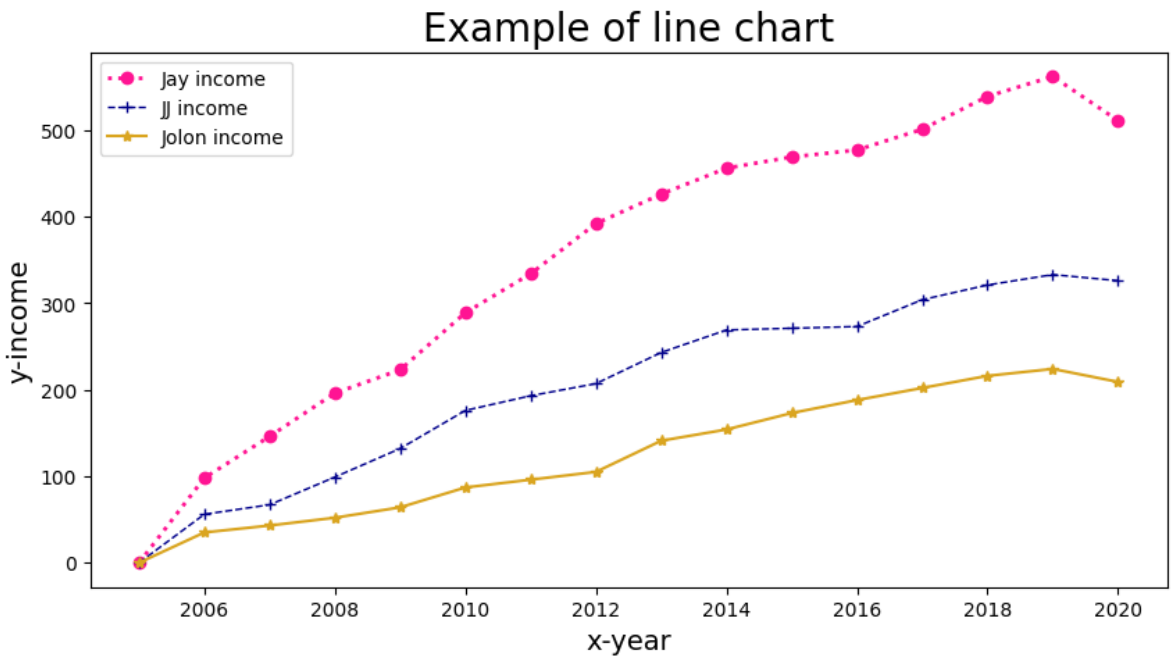
参数	接收值	说明	默认值
`x,y`	列表或数组	表示x轴与y轴对应的数据	无
`color`	字符串	表示折线的颜色	`None`
`marker`	字符串	表示折线上数据点的形状	`None`
`linestyle`	字符串	表示折线的类型	—
`linewidth`	数值	表示线条宽度，单位为像素	1
`alpha`	`[0,1]` 区间的小数	表示点的透明度，默认值1表示完全不透明	1
`label`	字符串	图例内容，显示指定的线条宽度	`None`

示例代码如下：

```
import matplotlib.pyplot as plt

# 生成数据
time = [2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016,
2017, 2018, 2019, 2020]
data1 = [0, 98, 146, 196, 223, 289, 334, 392, 426, 456, 469, 477, 501, 539, 562,
511]
```

```
data2 = [0, 56, 67, 99, 132, 176, 193, 207, 243, 269, 271, 273, 304, 321, 333,
326]
data3 = [0, 35, 43, 52, 64, 87, 96, 105, 141, 154, 173, 188, 202, 216, 224, 209]

# 设置画布的尺寸
plt.figure(figsize = (10, 5))
# 设定标题和字号
plt.title('Example of line chart', fontsize = 20)
# 设置x轴和字号大小
plt.xlabel(u'x-year', fontsize = 14)
plt.ylabel(u'y-income', fontsize = 14)

plt.plot(time, data1, color = "deeppink", linewidth = 2, linestyle = ":", label
= 'Jay income', marker = 'o')
plt.plot(time, data2, color = "darkblue", linewidth = 1, linestyle = "--", label
= 'JJ income', marker = '+')
plt.plot(time, data3, color = "goldenrod", linewidth = 1.5, linestyle = "-",
label = 'Jolon income', marker = '*')

plt.legend(loc = 2)
plt.show()
```

运行结果如图所示：



5.散点图

散点图是用二维坐标展示两个变量之间关系的图形。其中，横州代表变量x，纵轴代表变量y，每组数据(x_i, y_i)在坐标系中用一个点来表示，n组数据在坐标系中形成的n个点称为散点，由坐标和散点形成的二维数据图称为散点图。散点图比较适合描述数据在平面或空间中的分布和聚合情况，有助于分析数据之间的联系。

matplotlib中的`scatter()`函数可以根据给定的数据绘制散点图，用法如下：

```
scatter(x, y, s = None, marker = None, cmap = None, norm = None, vmin = None,
vmax = None, alpha = None, linewidths = None, verts = None, edgecolors = None,
data = None, **kwargs)
```

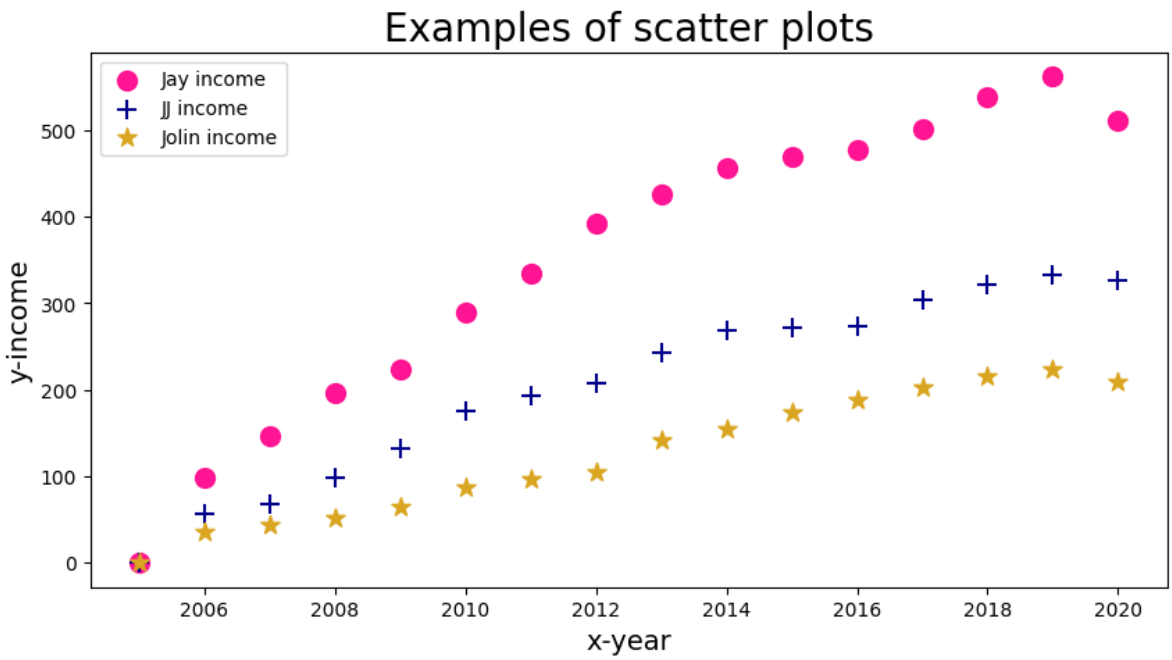
参数及其含义如下表所示：

参数名称	含义
x,y	分别用于指定绘制散点的x轴、y轴输入数据，可以是标量或数组形式的数据
s	数值或一维数组，指定散点的大小，若是一维数组，则表示图中每个点的大小
c	字符串或一维数组，表示散点的颜色，若是一维数组，则表示图中每个点的颜色
marker	字符串类型的数据，用于表示散点的类型
alpha	用于表示散点的透明度，取值范围为0~1的小数
edgecolors	用于指定散点符号的边线颜色，可以是颜色值或包含若干颜色值的序列

示例代码如下：

```
import matplotlib.pyplot as plt

plt.figure(figsize = (10, 5))
# 标题，并设定字号大小
plt.title('Examples of scatter plots', fontsize = 20)
plt.xlabel(u'x-year', fontsize=14)
plt.ylabel(u'y-income', fontsize=14)

# 生成数据
time = [2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016,
2017, 2018, 2019, 2020]
data1 = [0, 98, 146, 196, 223, 289, 334, 392, 426, 456, 469, 477, 501, 539, 562,
511]
data2 = [0, 56, 67, 99, 132, 176, 193, 207, 243, 269, 271, 273, 304, 321, 333,
326]
data3 = [0, 35, 43, 52, 64, 87, 96, 105, 141, 154, 173, 188, 202, 216, 224, 209]

plt.scatter(time, data1, s = 100, c = 'deeppink', marker = 'o')
plt.scatter(time, data2, s = 100, c = 'darkblue', marker = '+')
plt.scatter(time, data3, s = 100, c = 'goldenrod', marker = '*')

plt.legend(['Jay income', 'JJ income', 'Jolin income'])
plt.show()
```

运行结果如下：



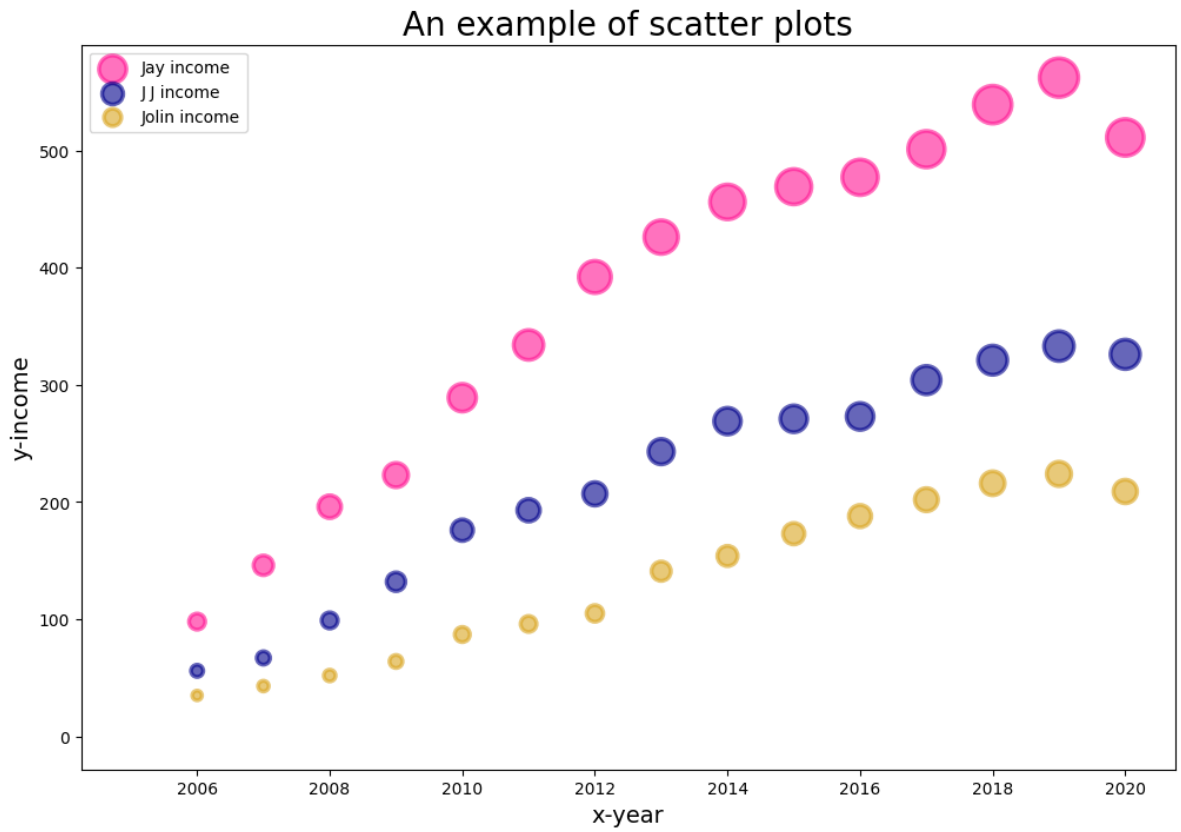
6.气泡图

散点图只能展示两个维度的数据：x轴和y轴。气泡图比散点图增加了一个维度，即使用表几点的大小代表第三个维度。根据数值的大小动态调整气泡的大小，因此气泡图可以看做散点图的衍生，用于展示三个变量之间的关系。与散点图了诶死，绘制气泡图时将一个变量放在横轴，另一个变量放在纵轴，第三个变量则用气泡的大小表示，使用matplotlib中的`scatter()`函数绘制气泡图。

示例代码如下：

```
import matplotlib.pyplot as plt

plt.figure(figsize = (12, 8))
# 标题，并设定字号大小
plt.title('An example of scatter plots', fontsize = 20)
plt.xlabel(u'x-year', fontsize=14)
plt.ylabel(u'y-income', fontsize=14)

# 生成数据
time = [2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016,
2017, 2018, 2019, 2020]
data1 = [0, 98, 146, 196, 223, 289, 334, 392, 426, 456, 469, 477, 501, 539, 562,
511]
data2 = [0, 56, 67, 99, 132, 176, 193, 207, 243, 269, 271, 273, 304, 321, 333,
326]
data3 = [0, 35, 43, 52, 64, 87, 96, 105, 141, 154, 173, 188, 202, 216, 224, 209]

plt.scatter(time, data1, s = data1, c = 'deeppink', marker = 'o', linewidth =
2.5, alpha = 0.6)
plt.scatter(time, data2, s = data2, c = 'darkblue', marker = 'o', linewidth =
2.5, alpha = 0.6)
plt.scatter(time, data3, s = data3, c = 'goldenrod', marker = 'o', linewidth =
2.5, alpha = 0.6)

plt.legend(['Jay income', 'J J income', 'Jolin income'])
plt.show()
```

运行结果如图所示：



7.雷达图

雷达图是显示多个变量的常用图形，也成为蜘蛛图。雷达图在显示或对比各变量的数值总和时十分有用。假设各变量的取值具有相同的正负号，则总的绝对值与图形围成的区域成正比。利用雷达图也可以研究多个样本间的相似程度。

可以使用matplotlib中的`polar()`函数根据给定的数据绘制雷达图，用法如下：

```
polar(theta, r, **kwargs)
```

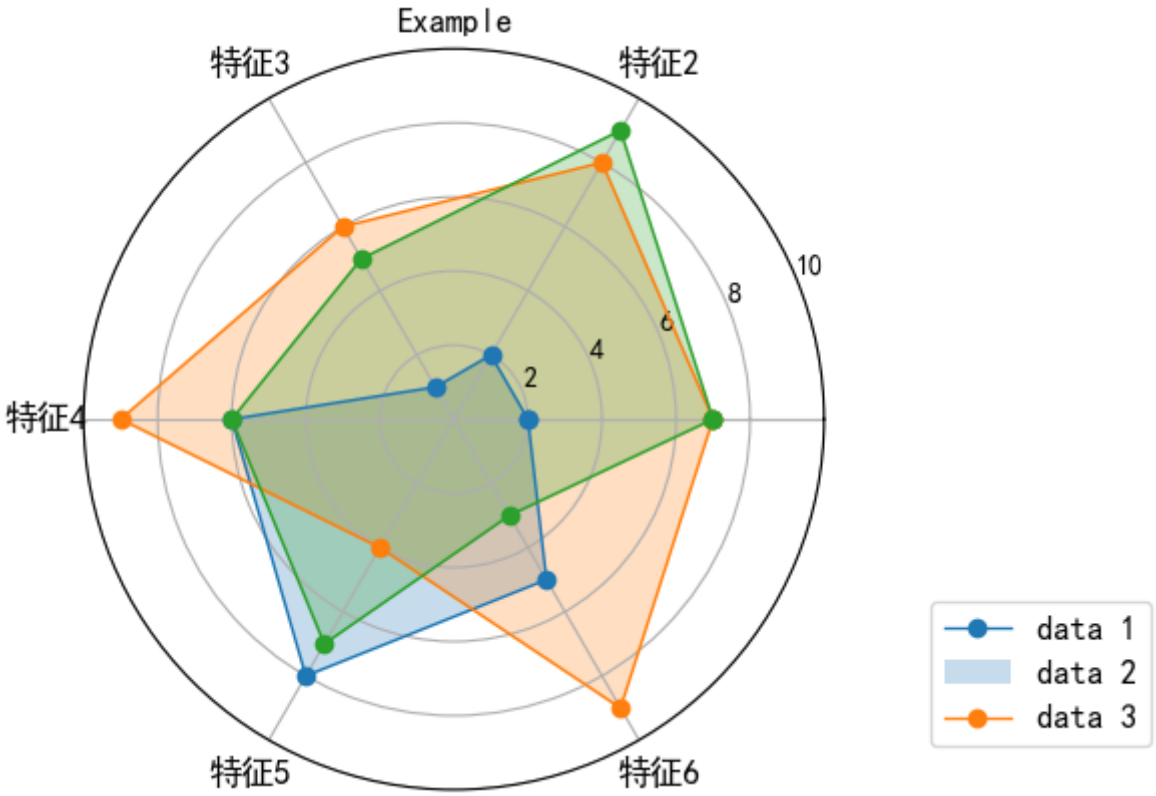
theta表示极角，r表示极径；其他参数与`plot()`函数的参数类似。

示例代码如下：

```
import numpy as np
from matplotlib import font_manager as fm
import matplotlib.pyplot as plt
import matplotlib as mpl

# 初始化字体属性
proptease = fm.FontProperties(size=12)

# 指定中文字体
mpl.rcParams['font.sans-serif'] = ['SimHei']  # 使用黑体字体，确保系统中有该字体
mpl.rcParams['axes.unicode_minus'] = False  # 解决坐标轴负号显示问题

# 标签
labels = ['特征1', '特征2', '特征3', '特征4', '特征5', '特征6', '']

# 数据长度
```

```
dataLenth = 6

# 随机数据
data1 = np.random.randint(1, 10, dataLenth)
data2 = np.random.randint(2, 10, dataLenth)
data3 = np.random.randint(3, 10, dataLenth)

# 分割圆周长
angles = np.linspace(0, 2 * np.pi, dataLenth, endpoint=False)
data1 = np.concatenate((data1, [data1[0]]))
data2 = np.concatenate((data2, [data2[0]]))
data3 = np.concatenate((data3, [data3[0]]))
angles = np.concatenate((angles, [angles[0]]))

# 绘制雷达图
plt.polar(angles, data1, 'o-', linewidth=1)
plt.fill(angles, data1, 'o-', alpha=0.25)
plt.polar(angles, data2, 'o-', linewidth=1)
plt.fill(angles, data2, 'o-', alpha=0.25)
plt.polar(angles, data3, 'o-', linewidth=1)
plt.fill(angles, data3, 'o-', alpha=0.25)

# 设置网格、标签
plt.thetagrids(angles * 180 / np.pi, labels, fontproperties=proptease)
plt.ylim(0, 10)

# 设置图例和标题
plt.legend(['data 1', 'data 2', 'data 3'], loc="lower right", fontsize=10,
bbox_to_anchor=(1.45, 0.05), borderaxespad=0.2, prop=proptease)
plt.title('Example', fontproperties=proptease)

# 显示图表
plt.show()
```

运行结果如图所示：



8.矩阵图

矩阵图通过可视化的方式呈现矩阵数据，可以用于展示三维信息。matplotlib中的`imshow()`函数可以根据给定的数据绘制矩阵图：

```
plt.imshow(X, cmap)
```

X：表示用于绘图的矩阵数据，为二维数据形式

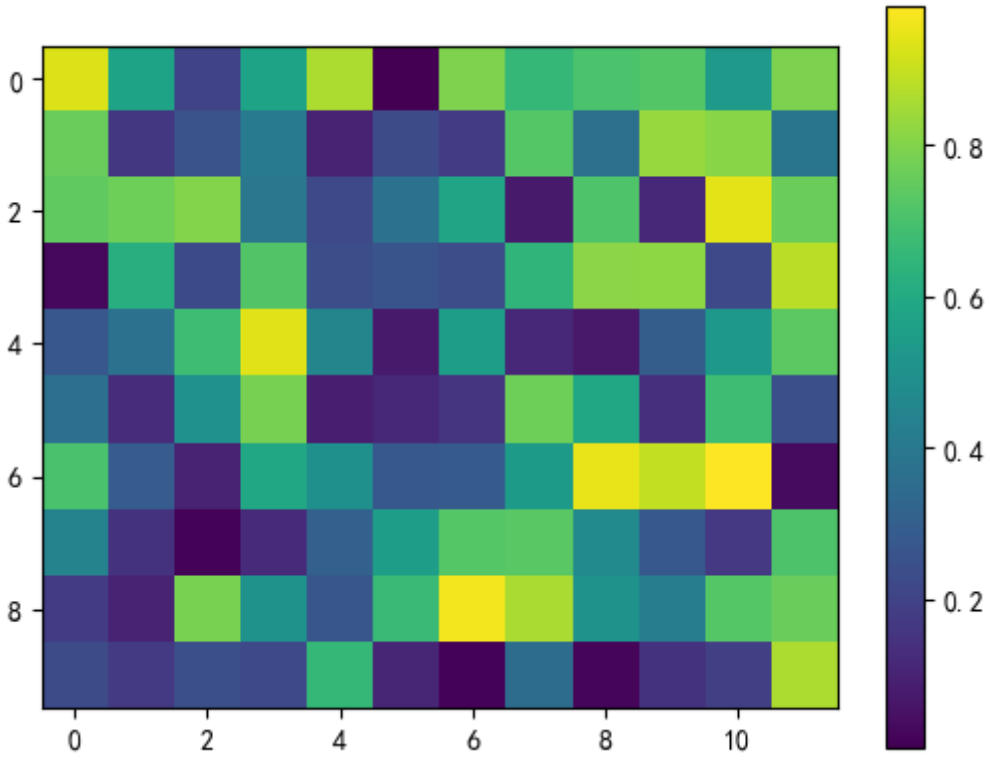
cmap：表示用于绘图的颜色主题，以该主题中颜色的深浅表示数据的大小

`plt.colorbar()`函数为图形添加颜色条

示例代码如下：

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

m = np.random.rand(10, 12)
plt.imshow(m)
plt.colorbar()
plt.show()
```

运行结果如图所示：



四、使用matplotlib绘制三维图形

一般情况下，基于`mpl_toolkits.mplot3d`模块的`Axes3D()`函数，绘制三维图形之前，要导入模块：

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

使用`plt.figure()`函数创建画布，并且将此画布变量作为参数传入`Axes3D()`函数，返回代表三维坐标轴的`Axes3D`对象`ax`，通过`ax`对象绘制三维图形：

```
fig = plt.figure()
ax = Axes3D(fig)
```

可以使用`ax`对象的`plot()`方法绘制三维图像，`plot_surface()`方法绘制三维曲面，使用`scatter()`方法绘制三维散点图，使用`bar3d()`方法绘制三维柱状图等等。

1.3D曲线

matplotlib中的`Axes3D.plot()`方法可以根据给定的数据绘制3D曲线：

```
Axes3D.plot(xs, ys, zs, zdir)
```

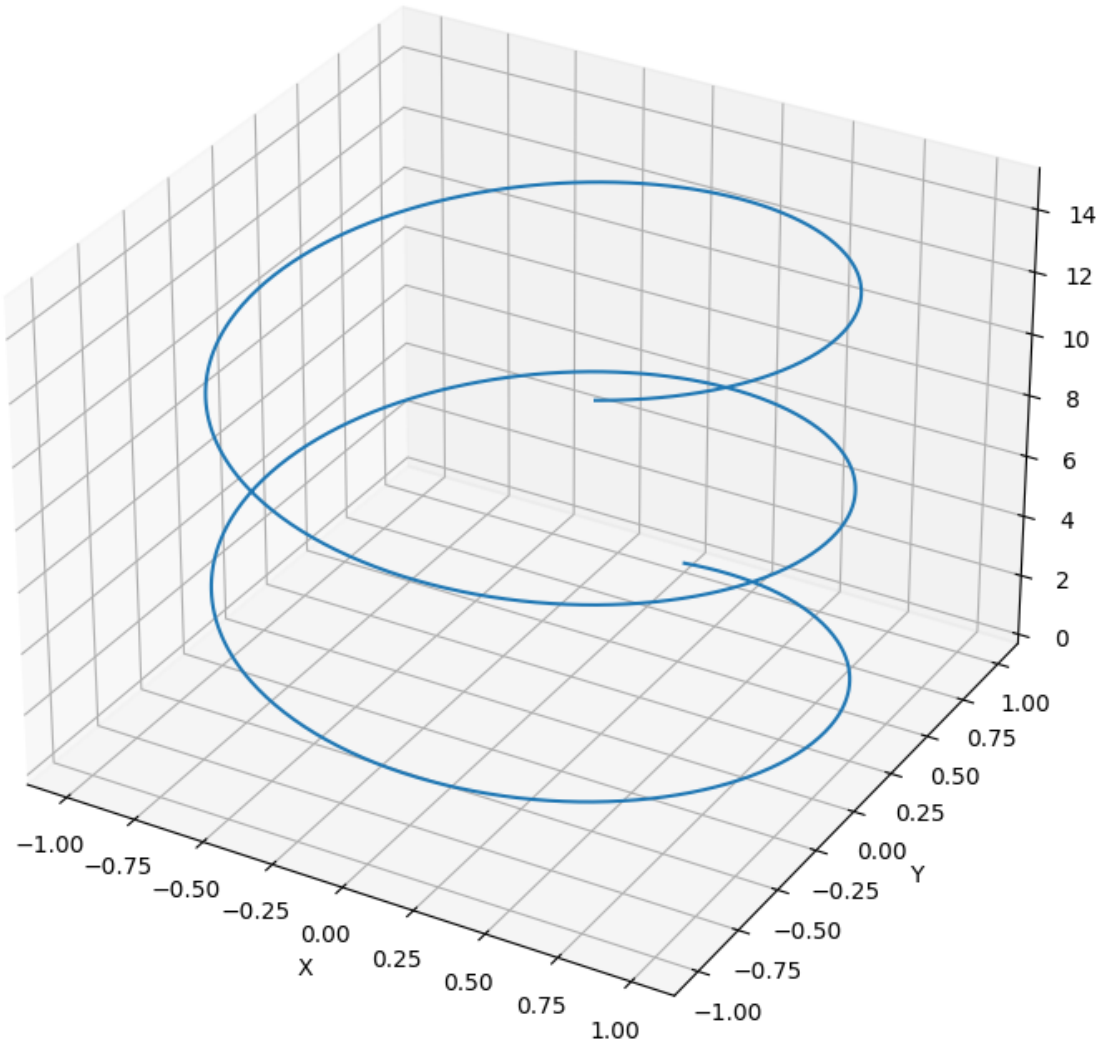
这里的`xs`,`ys`,`zs`代表点的三维坐标；`zdir`表示竖直轴，默认为`z`

示例代码如下：

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

# 准备数据
```

```
zline = np.linspace(0, 15, 1000)
xline = np.sin(zline)
yline = np.cos(zline)

# 创建3D图形对象，推荐使用add_subplot方法
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')

ax.plot(xline, yline, zline)
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

# 展示图像
plt.show()
```

运行结果如图所示：



2.3D散点图

matplotlib中的`Axes3D.scatter()`函数可以根据给定的数据绘制3D散点图：

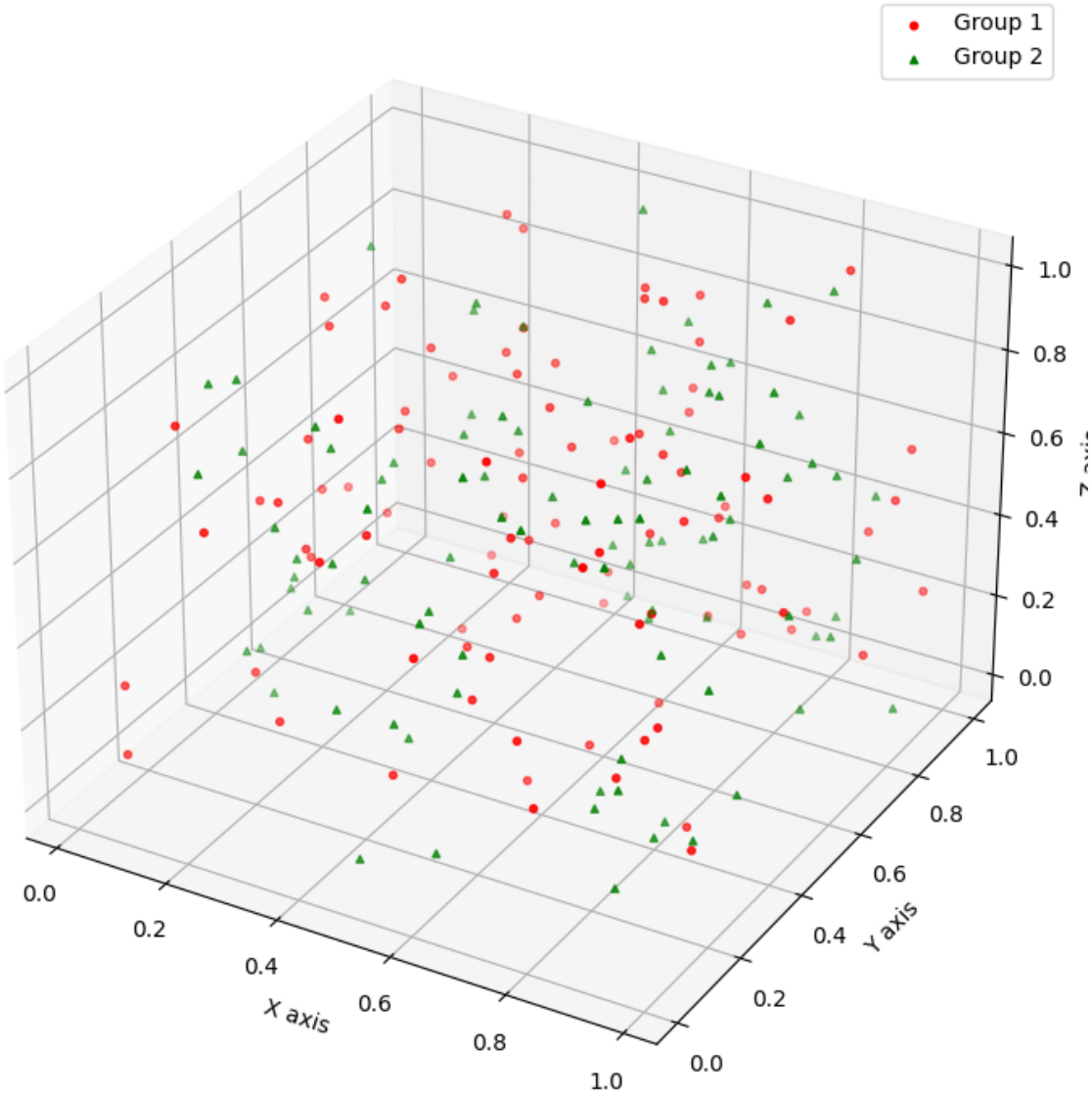
```
Axes3D.scatter(xs, ys, zs, zdir, s, c, marker)
```

- xs, ys, zs表示三点符号的三维坐标
- zdir表示竖直轴，默认为z轴
- s表示散点符号的大小
- c表示散点符号的颜色
- marker表示散点符号的形状

示例代码如下：

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

# 准备数据
x1 = np.random.rand(100)
y1 = np.random.rand(100)
z1 = np.random.rand(100)
x2 = np.random.rand(100)
y2 = np.random.rand(100)
z2 = np.random.rand(100)

# 创建3D图形对象，推荐使用add_subplot方法
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')

# 绘制3D散点图
scatter1 = ax.scatter(x1, y1, z1, s=10, c='r', marker='o', label='Group 1')
scatter2 = ax.scatter(x2, y2, z2, s=10, c='g', marker='^', label='Group 2')

# 添加图例
ax.legend(handles=[scatter1, scatter2])

# 添加轴标签
ax.set_xlabel('X axis')
ax.set_ylabel('Y axis')
ax.set_zlabel('Z axis')

# 展示图像
plt.show()
```

运行结果如图所示：



3.3D柱状图

`matplotlib`中的`Axes3D.bar()`函数可以根据给定的数据绘制3D散点图：

```
Axes3D.bar(left, height, zs, zdir)
```

- left表示柱子处于左边坐标轴的位置
- height表示柱子的高度
- zs表示柱子处于z坐标轴的位置
- zdir表示竖直轴，默认为z

示例代码如下：

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

# 准备数据
x = np.arange(10)
y1 = np.random.rand(10)
y2 = np.random.rand(10)

# 创建3D图像对象，这里推荐使用projection='3d'的方式
```

```
fig = plt.figure(figsize = (12, 9))
ax = fig.add_subplot(111, projection='3d')

# 绘制图形
ax.bar(x, y1, zs=0, zdir='y', color='b', alpha=0.6)
ax.bar(x, y2, zs=1, zdir='y', color='r', alpha=0.6)

# 设置y轴刻度，注意要与zs设置的值对应
ax.set_yticks([0, 1])
ax.set_yticklabels(['Set1', 'Set2'])  # 可选，给y轴刻度命名

# 添加标签和标题
ax.set_xlabel('X Axis')
ax.set_ylabel('Y Axis')
ax.set_zlabel('Z Axis')
plt.title('3D Bar Chart Example')

# 展示图像
plt.show()
```

运行结果如图所示：