3.1 matplotlib简介

在气象、基因、政治经济趋势分析等科研领域，Python常常被数据科学家用于数据密集型研究。基于Python的可视化和分析工具有很多，最流行的之一是matplotlib，这是一个功能性和扩展性很强的代码库。其中，最常用的模组是matplotlib.pyplot，它能让matplotlib以类似MATLAB方式运行的命令式函数集合。使用pyplot函数可以进行图表的创建、指定绘图区域、线条绘制、标注图表等功能。在matplotlib.pyplot中，不同的状态信息通过函数调用来保存，以便记录诸如当前图表和绘图区域等信息。

接下来，我们将介绍matplotlib的一些常用功能。由于本章侧重于介绍matplotlib，而它又常常与numpy等数学扩展包结合使用，对于numpy等扩展包相关内容，本章将只作简要介绍，感兴趣的读者可查阅相关资料进一步拓展学习。

3.2 matplotlib安装与测试

在Windows操作系统中，首先需要安装微软的Visual Studio。在<http://dev.windows.com/>上，可以搜索、下载并安装Visual Studio Community，这是Visual Studio的社区免费版开发工具。

和许多扩展库一样，matplotlib的安装可以通过pip进行。如果你已经安装了Python 3.x或以上版本，则安装Python时就已经自带了pip功能。打开命令行窗口，输入pip install matplotlib命令，即可自动下载并安装matplotlib。

安装完成后，可以打开一个命令行窗口进入Python终端会话，对之前的安装进行测试。输入import matplotlib命令，尝试导入matplotlib，如果没有出现任何错误信息提示，说明说明matplotlib已经成功安装，可以使用。

如果你已经安装了Anaconda，则Anaconda中已经集成了matplotlib，无需重复安装。

3.3 折线图

**3.3.1 绘制基本折线图**

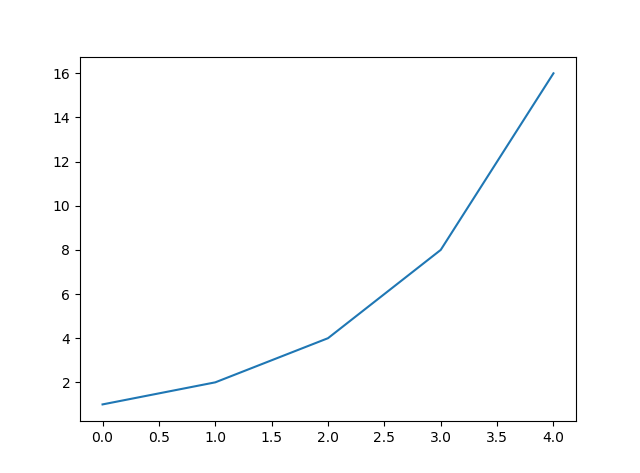
首先我们来试着绘制折线图。新建一个Python文件，并输入以下代码：

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  plt.plot([1, 2, 4, 8, 16]) plt.show() |

在这段代码中，我们首先导入了matplotlib库中的模组pyplot。为方便起见，我们为matplotlib.pyplot设置了在本程序中的别名plt，以简化后续程序的编写。

接下来，我们创建了一个包含有2的N次幂的列表，又将其作为参数，传递给了函数plot()，这个函数将根据给定的参数尝试绘制相应的图形。在第二行中，通过函数show()打开matplotlib查看器，并最终显示绘制结果，如下图所示。

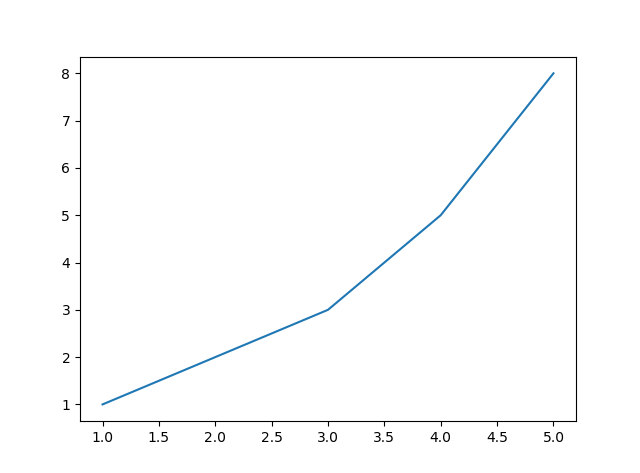
在上例中，虽然我们只向plot()函数传递了一个列表或者数组，但仍然绘制出了包含x轴和y轴两个坐标轴信息的图表。这是因为在plot()函数只接收到一个列表作为参数时，将会默认这个列表存储的是y轴的数据，而x轴则默认从0开始，长度与y轴相同。因此，对于上例中给定了单个数组[1, 2, 4, 8, 16]作为参数的情形，x轴的数值为[0, 1, 2, 3, 4]。



plot()函数非常灵活，可以支持任意数量的参数输入。例如，若想同时指定x和y轴的数值，可用如下格式向plot()输入参数：

|  |
| --- |
| plt.plot([1, 2, 3, 4, 5], [1, 2, 3, 5, 8]) |

此时，plot()接受了两个参数，[1, 2, 3, 4, 5]列表参数指定了横坐标的数值，第二个参数为列表[1, 2, 3, 5, 8]，指定了纵坐标的数值。于是可以得到以下图形。注意它与上一个图在坐标刻度上的区别：



**3.3.2 设置线条属性**

我们可以对图表中的线条进行线宽、线型、开启/关闭抗锯齿效果等多种属性的设置。设置线条属性的常用方法有若干种。这里介绍一种最常用的方法，即使用关键词参数进行设置。采用如下格式调用plot()函数：

|  |
| --- |
| plt.plot(x, y, linewidth=1.5) |

即在参数中明确地指出要修改的属性名称并传值。在上面这行代码中，我们明确指出，将线宽（linewidth）属性的值设置为1.5。

下表给出了常用的线条对象（Line2D对象）属性。对于未明确指定取值的参数，plot()将取其默认值：

|  |  |  |
| --- | --- | --- |
| **属性** | **含义** | **数据类型** |
| alpha | 不透明度（范围[0, 1]） | float |
| animated | 是否开启动态模式（需要相应函数不断更新数据） | True/False |
| antialiased或aa | 是否开启抗锯齿 | True/False |
| color或c | 线条颜色 | （部分常用格式）  1）表示RGB（红绿蓝）的三元组或表示RGBA（红绿蓝透明度）的四元组（每个颜色分量取值范围为[0, 1]）  2）十六进制的RGB或RGBA字符串（如’#0F0F0F’）  3）颜色关键词缩写之一（b,g,r,c,m,y,k,w） |
| data | 线条绘制数据来源 | numpy数组构成的元组  (np.array xdata, np.array ydata) |
| label | 线条标签 | String |
| linestyle或ls | 线条样式 | ‘-‘ ‘--‘ ‘-.’ ’:’ ’steps’等 |
| linewidth或lw | 线条宽度 | float |
| marker | 数据点形状 | ‘+’ ‘,’ ‘.’ ‘1’ ‘2’ ‘3’ ‘4’ |
| markeredgecolor或mec | 数据点边缘颜色 | 参见color项 |
| markeredgewidth或mew | 数据点边缘宽度 | float |
| markerfacecolor或mfc | 数据点颜色 | 参加color项 |
| markersize或ms | 数据点大小 | float |
| visible | 线条是否可见 | True/False |
| xdata | x轴数据 | numpy数组np.array |
| ydata | y轴数据 | numpy数组np.array |

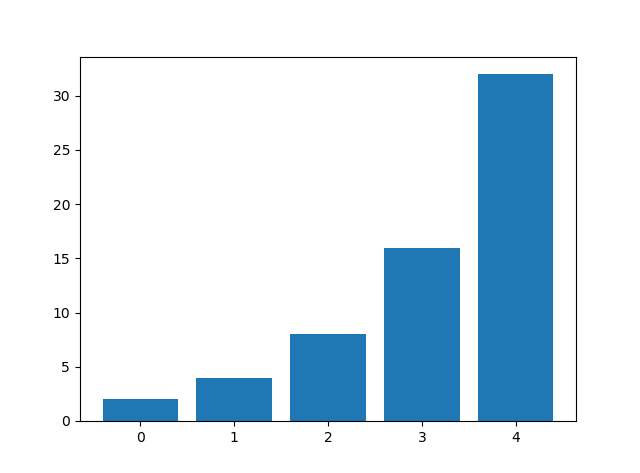
3.4 柱状图

**3.4.1 绘制基本柱状图**

除折线图外，最常用的统计图类型之一就是柱状图。绘制柱状图的基本函数为plt.bar()。下面绘制一幅基本的柱状图：

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  data = [2, 4, 8, 16, 32]  plt.bar(range(len(data)), data) plt.show() |

在上述代码中，我们向plt.bar()传递了两个参数，分别是柱状图下标和柱体的高度。可以发现，这和折线图的绘制有相似之处：在调用plt.plot()绘制折线图时，若只传递两个参数，则第一个参数表示数据点的横坐标，第二个参数表示数据点的纵坐标即数据点高度。



**3.4.2 调整柱体属性**

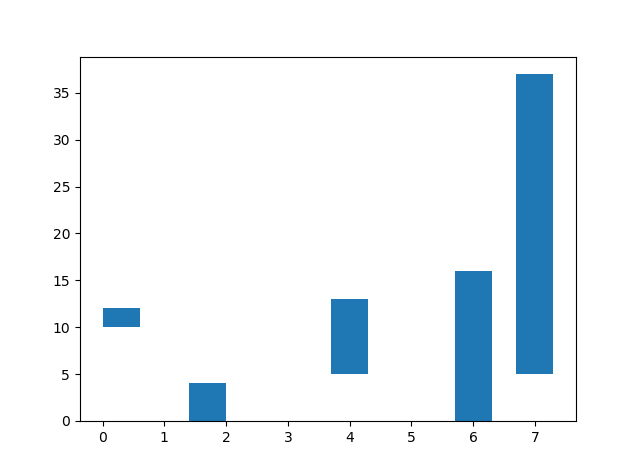
和plot()相似，我们可以通过向bar()传递参数来修改柱体的位置、大小或颜色等属性：

|  |
| --- |
| bar(left, height, width=1.0, bottom=**None**, \*\*kwargs) |

四个参数：：left、height、width、bottom共同确定了柱体的位置和大小。left指定了主体的居中位置，因此，左下角坐标为(left-width/2, bottom)，而右上角坐标为(left+width/2, bottom+height)。例如，我们可以对柱体位置作如下调整：

|  |
| --- |
| **import** numpy **as** np **import** matplotlib.pyplot **as** plt  data = [2, 4, 8, 16, 32]  plt.bar([0.3,1.7,4,6,7], data, width=0.6, bottom=[10, 0, 5, 0, 5]) plt.show() |

注意到我们此次调用plt.bar()时，传递了四个参数，此时四个参数依次为left、height、width和bottom。因此，柱体的位置发生了变化。作者可进一步尝试向width传递一个四元素数组，赋以不同的柱体宽度值，观察柱状图的形态变化情况。



**3.4.3 为柱状图添加文本**

在前面生成柱状图的过程中，我们的横坐标使用的是数值。但事实上对柱状图而言，很多时候往往需要在横坐标处添加文本标签，说明每个柱体的含义。此时只需要在调用bar()函数的过程中，传入由标签构成的字符串列表，即可实现图中柱体刻度标签的添加。

如果希望对x轴或y轴整体添加标签，则需要调用xlabel()和ylabel()函数添加，将标签文本字符串作为参数传入即可。如果想要为柱状图添加标题，则可以类似地调用title()函数并传入参数以实现。

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  labels = [**'Monday'**, **'Tuesday'**, **'Wednesday'**, **'Thursday'**] num\_list = [3.2, 1.6, 6.8, 3.6] plt.bar(range(len(num\_list)), num\_list, tick\_label=labels)  *# 添加标签和标题* plt.xlabel(**'x-axis'**) plt.ylabel(**'y-axis'**) plt.title(**'bar graph'**)  plt.show() |

注意到，在调用bar()时的第三个参数，我们将存有标签的字符串列表labels赋值给了tick\_label，从而添加了刻度标签。

此外，若想添加含有中文的文本，需要在传入中文字符串之前，添加以下代码：

|  |
| --- |
| plt.rcParams[**'font.sans-serif'**] = [**'SimHei'**] plt.rcParams[**'axes.unicode\_minus'**] = **False** |

这里的字符串SimHei代表黑体。

在本小节中介绍的为x轴、y轴添加标签、添加标题，以及支持中文输入的代码，同样适用于其他类型的统计图。

图片包含 屏幕截图

自动生成的说明

**3.4.4 创建堆叠柱状图**

如果需要表示某个柱体中不同组成部分的占比（如在一个柱体中，用不同颜色分别表示某年的新生儿中男婴和女婴），或是并列对比两种组成部分的数值大小（如在同一个标签位置，用两个柱体分别表示某年新生儿中的男婴和女婴数量），可以制作堆叠柱状图或并列柱状图来实现。

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  labels = [**'Monday'**, **'Tuesday'**, **'Wednesday'**, **'Thursday'**, **'Friday'**] list1 = [3.2, 1.6, 6.8, 2.6, 4] list2 = [2.3, 1.2, 3.3, 5.5, 3]  plt.bar(range(len(list1)), list1, label=**'part1'**, color=**'g'**) plt.bar(range(len(list1)), list2, bottom=list1, label=**'part2'**, tick\_label=labels, color=**'r'**)  *# 添加标签、标题和图例* plt.xlabel(**'x-axis'**) plt.ylabel(**'y-axis'**) plt.title(**'bar graph'**)  plt.legend()  plt.show() |

注意观察第二次调用plt.bar()时参数的不同，我们设置了参数bottom的值，使第二次绘制时添加的柱体正好位于第一次绘制的柱体上方，从而实现了堆叠柱状图。我们还在两次调用时通过color设置了不同的颜色进行区分，与此同时，还调用了函数legend()，创建一个图例，帮助读图者更好地辨认每种颜色对应的类别。

图片包含 屏幕截图

自动生成的说明

3.5 饼图

在数据可视化中，饼图做为一种常用的统计图，能清晰地表示一个整体中各项的大小与在总体中的占比。使用matplotlib绘制饼图的函数是pie()。

**3.5.1 绘制基本饼图**

和其他统计图类似，绘制饼图的基本方法就是将存储数据的列表作为参数传给函数pie()。

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  labels = **'Andy'**,**'Mary'**,**'Tom'**,**'Susan'** data = [30,40,45,20]  plt.pie(data, labels=labels) plt.show() |

我们将标签文本和数据列表作为参数传入函数plt.pie()，即可绘制一个基本的饼图。

图片包含 矢量图形

自动生成的说明

可以看到，在默认设置下，饼图绘制的是一个椭圆形的“饼”。若想绘制圆形饼图，需要在调用plt.show()前，调用如下函数：

|  |
| --- |
| plt.axis(**'equal'**) |

之后再次运行程序，可以看到生成了一个圆形的饼图。

图片包含 矢量图形

自动生成的说明

**3.5.2 凸出显示饼图中的某一块**

我们经常可以看到这样一种饼图：为了特别显示饼图中某一块，让这一块与整体稍微分离。在matplotlib中绘制饼图时，可以通过调整explode参数实现这一效果。

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  labels = **'Andy'**,**'Mary'**,**'Tom'**,**'Susan'** data = [30,40,45,20]  explode = (0,0.2,0,0) plt.pie(data, explode=explode, labels=labels)  plt.axis(**'equal'**) plt.show() |

图片包含 矢量图形

自动生成的说明

对应饼图中的四个数据，我们创建了一个四元组，并将其传递给参数explode。四元组中每个元素的取值表示与饼整体的分离程度，数值越大，分离程度越高。此处我们只想凸出显示Mary的那一部分，因此将四元组的第二个元素设为0.2，其他元素保持为0不变。读者可自行尝试调整explode参数的各项值，观察元素全部取非0值、甚至取负值时，饼图的各组成部分位置变化情况。

3.5.3 为饼图添加文本

类似折线图和柱状图，我们可以用label()为饼图各部分添加标签文本，用title()添加标题。特别地，在饼图中，通过设置pie()函数的autopct参数，可以为饼图各部分标识其所占百分比。

|  |
| --- |
| **import** matplotlib.pyplot **as** plt  labels = **'Andy'**,**'Mary'**,**'Tom'**,**'Susan'** data = [30,40,45,20]  explode = (0.1,0.2,0.3,0.1) plt.pie(data, explode=explode, labels=labels, autopct=**'%.2f%%'**)  plt.title(**'pie chart'**) plt.axis(**'equal'**) plt.show() |

此次调用pie()时，我们设置了参数autopct为’%.2f%%’。其中，第一个%表示格式化输出；后两个%表示输出百分号；而.2f表示保留两位小数。因此，如果将autopct设为’%.3f’，则在饼图上每一块都将显示保留三位小数的百分比，且不含百分号。

图片包含 名片, 文字

自动生成的说明

3.6 其他常用设置

matplotlib允许用户以多种方式修改绘图样式，以根据不同的需求和数据，针对性地增强图表可读性。

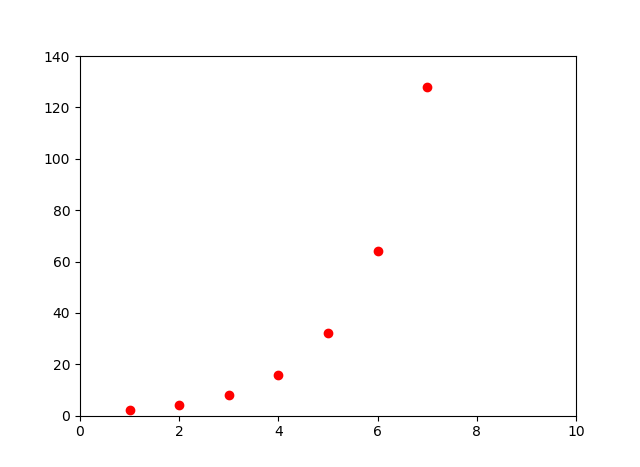
**3.6.1 向绘图函数传递参数以修改图表样式**

对于每一对x，y参数而言，都有一个可选的第三参数，用于格式化该对x，y所表示的点的颜色和线条类型。格式化字符串的字母和符号来源于MATLAB，你可以将颜色字符串与线条类型字符串联合使用。默认的格式字符串是“b-”，即上图所示的蓝色实线。例如，若想用绘制折线图的函数plot()绘制一幅红色散点图，可以采用以下代码定制输出图形的格式：

|  |
| --- |
| plt.plot([1, 2, 3, 4, 5, 6, 7], [2, 4, 8, 16, 32 ,64, 128], **'ro'**)  plt.axis([0, 10, 0, 140]) plt.show() |

可以看到，在对plot()函数的调用中，我们除了给出了两个列表作为横纵坐标数值以外，还给出了一个字符串‘ro’作为第三个参数。正是这个参数，让默认的蓝色折线图变为了红色散点图。其中，r指定了点的颜色为红色；o指定了图表格式为散点图。读者可以尝试修改该字符串，例如删去r、删去o，或者将ro变为go、bo，观察图表会怎样发生变化。

在代码的第二行，我们调用了另一个函数axis()。该函数接受一个长度为4的列表，从而指定了坐标轴的显示范围。具体而言，该列表的格式为[xmin, xmax, ymin, ymax],即依次指定了x轴的最小值、最大值和y轴的最小值、最大值。



除了可以处理Python默认的列表格式外，matplotlib还可以处理numpy数组，这也大大拓展了matplotlib的功能性。下例就通过数组输入数据，并用不同格式进行了可视化表示。

|  |
| --- |
| **import** matplotlib.pyplot **as** plt **import** numpy **as** np  *# 每隔0.2距取值* t = np.arange(0, 5, 0.2)  *# 红色虚线，蓝色方块，绿色三角* plt.plot(t, t, **'r--'**, t, t\*\*2, **'bs'**, t, t\*\*3, **'g^'**) plt.show() |

首先，通过import numpy as np导入numpy库。随后调用numpy的arange()函数，创建从0到5，差为0.2的等差数组。arange()是一个用于创建一位数组的非常灵活的函数，其参数可以有多种格式。当参数为3个时，分别表示数组的下限、上限、间隔值。

随后，调用plot()函数，确定要绘制的散点图格式。可以看到，上面这段代码的plot()函数一共有9个参数，其中出现的t即为上面arange()生成的数组。其中每3个变量绘制了一系列散点：“t, t, ‘r--”绘制了一条y=x的红色虚线图像；“t, t\*\*2, ‘bs’”绘制了y=x2的蓝色方点图像；“t, t\*\*3, ‘g^’”绘制了y=x3的绿色三角点图像。

图片包含 屏幕截图

自动生成的说明

**3.4.2 使用关键字符串绘点**

有些时候，你的数据存储格式用字符串访问特定变量，例如字典、numpy.recarray或pandas.DataFrame类型。matplotlib允许用户使用这些对象中的关键字作为描点参数，此时，matplotlib将根据这些字符串对应的变量进行描点。

|  |
| --- |
| **import** numpy **as** np **import** matplotlib.pyplot **as** plt  data = {**'a'**: np.arange(50),  **'c'**: np.random.randint(0, 50, 50),  **'d'**: np.random.randn(50)} data[**'b'**] = data[**'a'**] + 10 \* np.random.randn(50) data[**'d'**] = np.abs(data[**'d'**]) \* 100  plt.scatter(**'a'**, **'b'**, c=**'c'**, s=**'d'**, data=data) plt.xlabel(**'entry a'**) plt.ylabel(**'entry b'**) plt.show() |

在上面这段代码中，我们创建了这样一幅散点图：各点的位置随机选取，而大小、颜色代码也随机选取，且服从正态分布。那么，如何实现这样在描点时各种属性均按一定规律服从随机分布的效果？

首先我们创建了一个字典data。data在初始化时有三个键值对：

键‘a’对应的值为一个包含从0到49整数的数组（函数np.arange()在接收一个参数x时，将创建一个从0开始，步长为1，元素个数为x个的数组）；

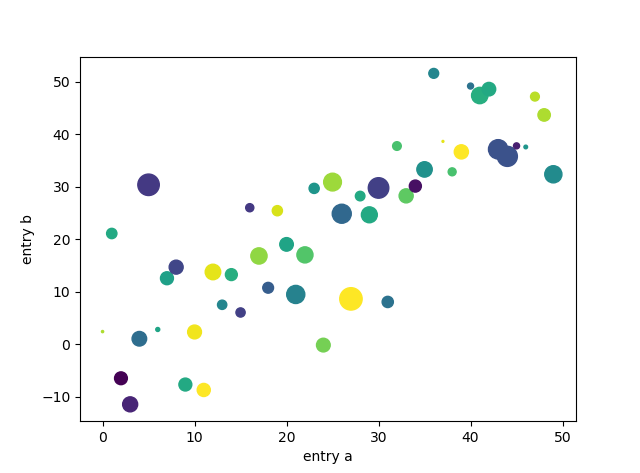
键‘c’对应的值为一个长度为50的随机数组，随机数在0到50内随机取得（函数np.random.randint()可接收三个参数，分别为low、high、size，即随机数组的取值下限、取值上限、数组大小）；

键‘d’对应的值为一个长度为50的随机数组，其中数组元素满足标准正态分布。

在初始化data之后，我们添加了一个新的键值对，键为‘b’。‘b’对应的值为键‘a’对应的数组加上一个10倍大小的标准正态分布数组。

最后，我们再更新键‘d’对应的值，将初始化值取绝对值后乘以100.

在调用plt.scatter()进行描点时，我们输入了五个参数：首先注意到最后一个参数data，它指定了前四个参数所提及的字符串应该在字典data中找到其对应值；第一、二个参数通过字符串‘a’‘b’指定了点的横纵坐标；第三、四个参数通过字符串‘c’‘d’指定了各点的颜色和大小。最终，调用plt.show()，我们就得到了这样一幅点的位置、颜色和大小按一定规律服从随机分布的散点图，而这里的“一定规律”则由我们事先设定的字典data所确定。我们在plt.scatter()中所做的，就是调用字典data中的键字符串作为参数，指定描点时的各项属性。



**3.4.3 在一个绘图区域上创建多个子图**

有时我们需要同时创建多个子图来实现数据之间的对比。在matplotlib中，可以使用subplot()函数实现子图的创建。通常，在使用subplot()时，会按subplot(nrows, ncols, index)的格式输入三个参数。其中，参数nrows和ncols分别为number of rows和number of columns的简写，表示将绘图区域划分为nrows\*ncols个子区域；第三个参数index则是指定即将绘图的子区域编号。

以如下代码为例：

|  |
| --- |
| **import** numpy **as** np **import** matplotlib.pyplot **as** plt  names = [**'sub\_a'**, **'sub\_b'**, **'sub\_c'**] values = [5, 25, 125]  plt.figure(figsize=(9, 3))  plt.subplot(1,3,1) plt.bar(names, values) plt.subplot(1,3,2) plt.scatter(names, values) plt.subplot(1,3,3) plt.plot(names, values) plt.suptitle(**'Subplots'**) plt.show() |

这段代码创建了一个包含三个不同类型子图的总图。首先，我们定义了两个列表names和values，其中values用于存储具体数据，而names则用于存储标示数据的标签字符串。names和values这一组标签和数据将为三个子图所共同使用。

接下来，我们调用plt.figure()函数设置画布尺寸，虽然在本例中，我们只向figure()函数传递了参数figsize的取值，但事实上figure()的设置选项非常丰富。figure()函数的调用格式为：

figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True)

|  |  |
| --- | --- |
| **参数** | **功能** |
| num | 指定图像文件的编号或名称。当传递整数型参数时，图像标题被命名为“Figure X”；  当传递字符串参数时，图像标题被命名为参数字符串。默认标题：“Figure 1” |
| figsize | 指定图像的宽和高。单位：英寸 |
| dpi | 指定图像的分辨率（每英寸包含的像素数量）。默认值：80 |
| facecolor | 指定图像的背景色 |
| edgecolor | 指定图像边框颜色 |
| frameon | 指定边框是否显示 |

其常用参数和功能如上表所示。在本例中，我们只设置了画布的尺寸，传递了(9, 3)作为参数，表示设置画布尺寸为宽9英寸，高3英寸。

接下来就是通过plt.subplot()与不同的绘图函数组合，绘制各个子图的过程了。我们每绘制一个子图之前，就需要调用一次subplot()函数，指定我们要绘制哪个子图。

第一次调用plt.subplot()时，代入参数为1，3，1。三个参数分别表示接下来要绘制的是一个1行3列的总图中的第1个子图。接下来调用plt.bar()绘制条形图；

第二次调用plt.subplot()时，代入参数为1，3，2。三个参数分别表示接下来要绘制的是一个1行3列的总图中的第2个子图。接下来调用plt.scatter()绘制散点图；

第三次调用plt.subplot()时，代入参数为1，3，1。三个参数分别表示接下来要绘制的是一个1行3列的总图中的第3个子图。接下来调用plt.plot()绘制折线图。

以上三组调用，完成了子图的绘制。随后，我们调用plt.suptitle()，在图中添加一个标题。注意区分该函数与figure()函数的num参数的不同之处。

图片包含 屏幕截图

自动生成的说明最后，调用plt.show()，完成绘图并显示结果。

#end