

Matplotlib 数据可视化分析

介绍

Matplotlib 是支持 Python 语言的开源绘图库，因为其支持丰富的绘图类型、简单的绘图方式以及完善的接口文档，深受 Python 工程师、科研学者、数据工程师等各类人士喜欢。本次实验课程中，我们将学会使用 Matplotlib 绘图的方法和技巧。

知识点

- 二维图形绘制
- 子图及组合图形
- 兼容 MATLAB 风格 API



数据分析过程中，一定会遇到需要针对数据进行绘图的场景。Matplotlib 是支持 Python 语言的开源绘图库，因为其支持丰富的绘图类型、简单的绘图方式以及完善的接口文档，深受 Python 工程师、科研学者、数据工程师等各类人士喜欢。Matplotlib 拥有着十分活跃的社区以及稳定的版本迭代，当我们在学习数据分析的课程时，掌握 Matplotlib 的使用无疑是最重要的准备工作之一。

在使用 Notebook 环境绘图时，需要先运行 Jupyter Notebook 的魔术命令 `%matplotlib inline`。这条命令的作用是将 Matplotlib 绘制的图形嵌入在当前页面中。而在桌面环境中绘图时，不需要添加此命令，而是在全部绘图代码之后追加 `plt.show()`。

In []:

```
%matplotlib inline
```

In [1]:

```
%matplotlib inline
```

简单图形绘制

使用 Matplotlib 提供的面向对象 API，需要导入 `pyplot` 模块，并约定简称为 `plt`。

In []:

```
from matplotlib import pyplot as plt
```

In [2]:

```
from matplotlib import pyplot as plt
```

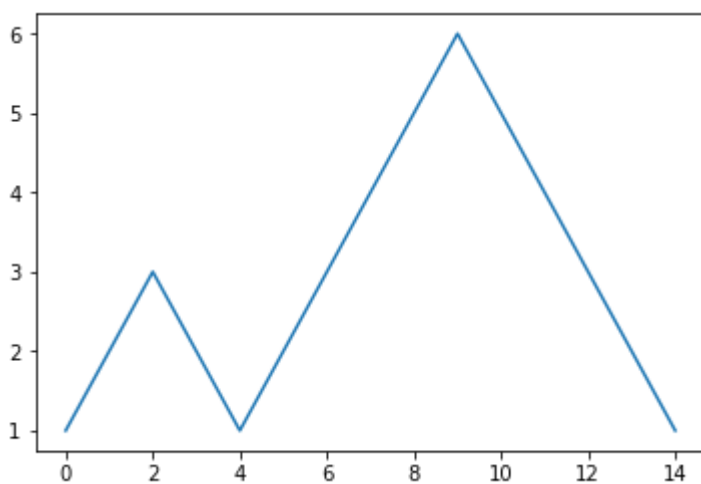
我们都说了，Matplotlib 是一个非常简单而又完善的开源绘图库。那么它到底有多简单呢？下面，我们通过 1 行代码绘制一张简单的折线图。

In [3]:

```
plt.plot([1, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1])
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x7fcc3e99b630>]
```



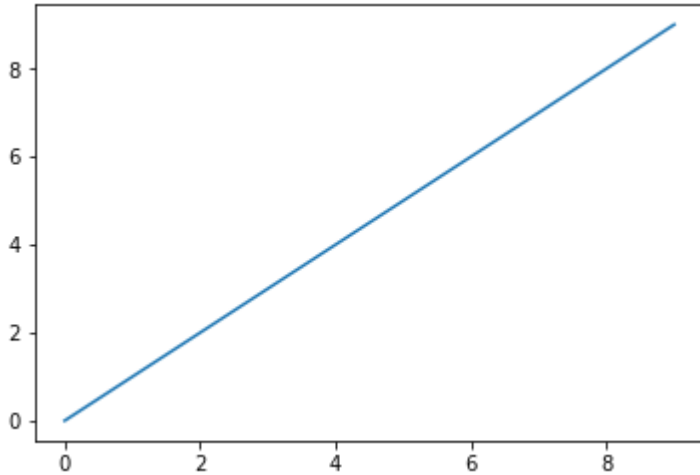
In [9]:

```
import numpy as np

plt.plot(np.arange(10))
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x7fcc3e80d898>]
```



可以看到，一张和山峰样式相似的折线图就绘制出来了。

前面，我们从 Matplotlib 中导入了 `pyplot` 绘图模块，并将其简称为 `plt`。`pyplot` 模块是 Matplotlib 最核心的模块，几乎所有样式的 2D 图形都是经过该模块绘制出来的。这里简称其为 `plt` 是约定俗成的，希望你也这样书写代码，以便拥有更好的可读性。

`plt.plot()` 是 `pyplot` 模块下面的直线绘制（折线图）方法类。示例中包含了一个 `[1, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1]` 列表，Matplotlib 会默认将该列表作为 `y` 值，而 `x` 值会从 0 开始依次递增。

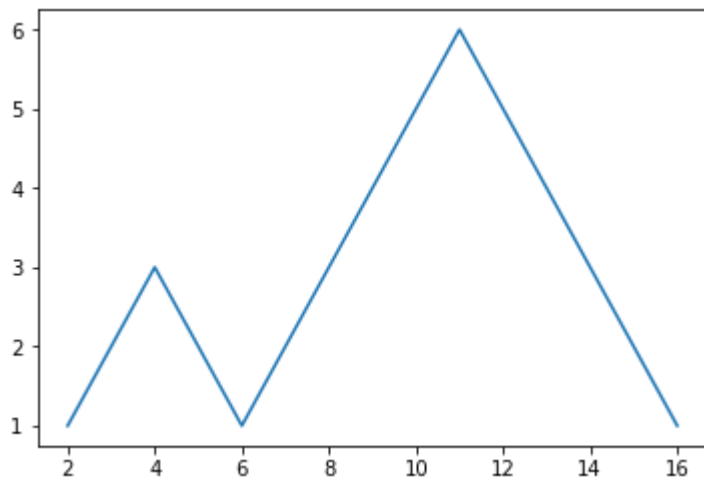
当然，如果你需要自定义横坐标值，只需要传入两个列表即可。如下方代码，我们自定义横坐标刻度从 2 开始。

In [12]:

```
plt.plot([2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],  
         [1, 2, 3, 2, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1])
```

Out[12]:

[<matplotlib.lines.Line2D at 0x7fcc3e711668>]

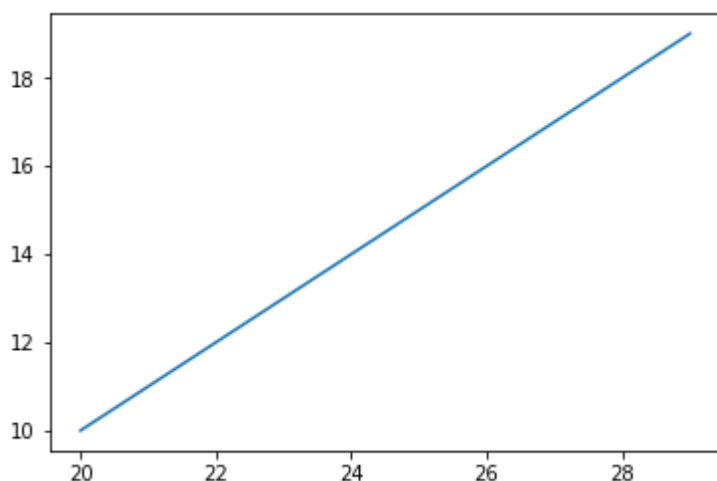


In [11]:

```
plt.plot(np.arange(20,30),np.arange(10,20))
```

Out[11]:

[<matplotlib.lines.Line2D at 0x7fcc3e736cf8>]



上面演示了如何绘制一个简单的折线图。那么，除了折线图，我们平常还要绘制柱状图、散点图、饼状图等等。这些图应该怎样绘制呢？

`pyplot` 模块中 `pyplot.plot` 方法是用来绘制折线图的。你应该会很容易联想到，更改后面的方法类名就可以更改图形的样式。的确，在 `Matplotlib` 中，大部分图形样式的绘制方法都存在于 `pyplot` 模块中。例如：

方法	含义
<code>matplotlib.pyplot.angle_spectrum</code>	绘制电子波谱图
<code>matplotlib.pyplot.bar</code>	绘制柱状图
<code>matplotlib.pyplot.barh</code>	绘制直方图
<code>matplotlib.pyplot.broken_barh</code>	绘制水平直方图
<code>matplotlib.pyplot.contour</code>	绘制等高线图
<code>matplotlib.pyplot.errorbar</code>	绘制误差线
<code>matplotlib.pyplot.hexbin</code>	绘制六边形图案
<code>matplotlib.pyplot.hist</code>	绘制柱形图
<code>matplotlib.pyplot.hist2d</code>	绘制水平柱状图
<code>matplotlib.pyplot.pie</code>	绘制饼状图
<code>matplotlib.pyplot.quiver</code>	绘制量场图
<code>matplotlib.pyplot.scatter</code>	散点图
<code>matplotlib.pyplot.specgram</code>	绘制光谱图

下面，我们参考折线图的绘制方法，尝试绘制几个简单的图形。

`matplotlib.pyplot.plot(*args, **kwargs)` 方法严格来讲可以绘制线形图或者样本标记。其中，`*args` 允许输入单个 `y` 值或 `x, y` 值。

例如，我们这里绘制一张自定义 `x, y` 的正弦曲线图。

In [13]:

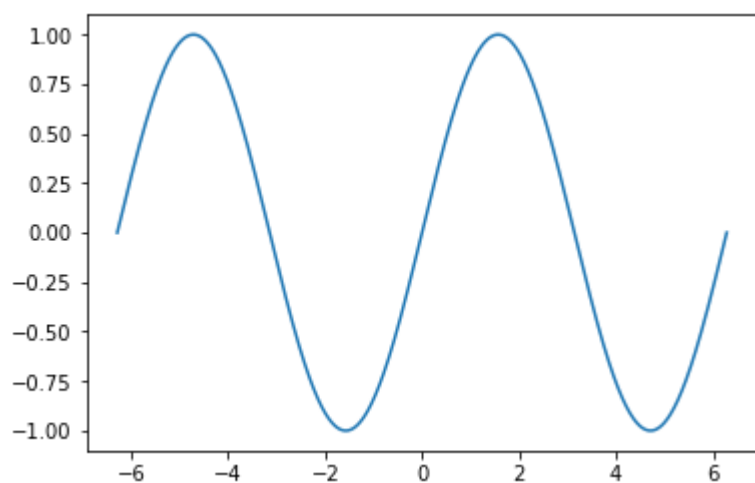
```
import numpy as np # 载入数值计算模块

# 在  $-2\pi$  和  $2\pi$  之间等间距生成 1000 个值, 也就是  $x$  坐标
X = np.linspace(-2*np.pi, 2*np.pi, 1000)
# 计算  $y$  坐标
y = np.sin(X)

# 向方法中 `*args` 输入  $x, y$  坐标
plt.plot(X, y)
```

Out[13]:

[<matplotlib.lines.Line2D at 0x7fcc3e670a90>]



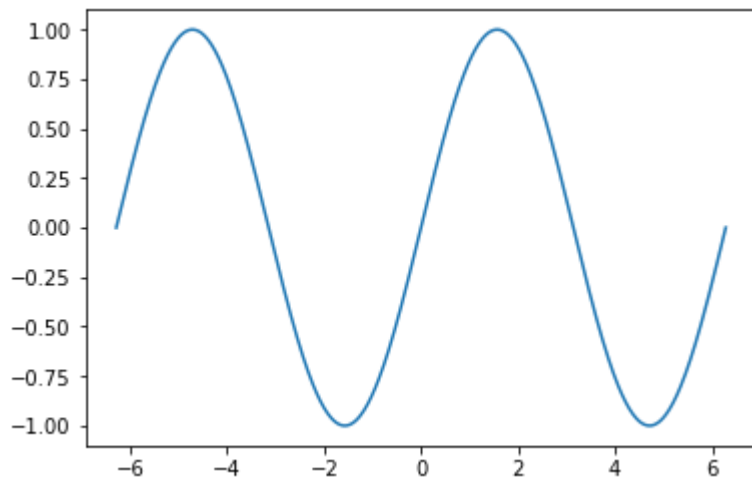
In [15]:

```
import numpy as np

x = np.linspace(-2*np.pi, 2*np.pi, 500)
y = np.sin(x)
plt.plot(x,y)
```

Out[15]:

[<matplotlib.lines.Line2D at 0x7fcc3e5b58d0>]



正弦曲线就绘制出来了。但值得注意的是，`pyplot.plot` 在这里绘制的正弦曲线，实际上不是严格意义上的曲线图，而在两点之间依旧是直线。这里看起来像曲线是因为样本点相互挨得很近。

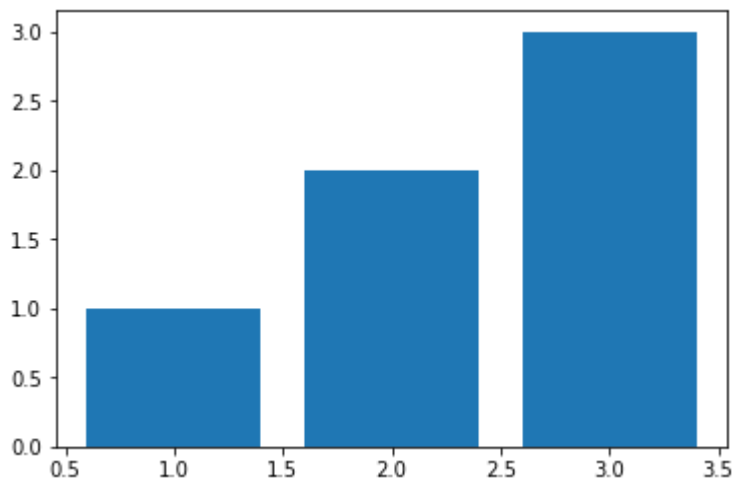
柱形图 `matplotlib.pyplot.bar(*args, **kwargs)` 大家应该都非常了解了。这里，我们直接用上面的代码，仅把 `plt.plot(x, y)` 改成 `plt.bar(x, y)` 试一下。

In [20]:

```
plt.bar([1, 2, 3], [1, 2, 3])
```

Out[20]:

<BarContainer object of 3 artists>

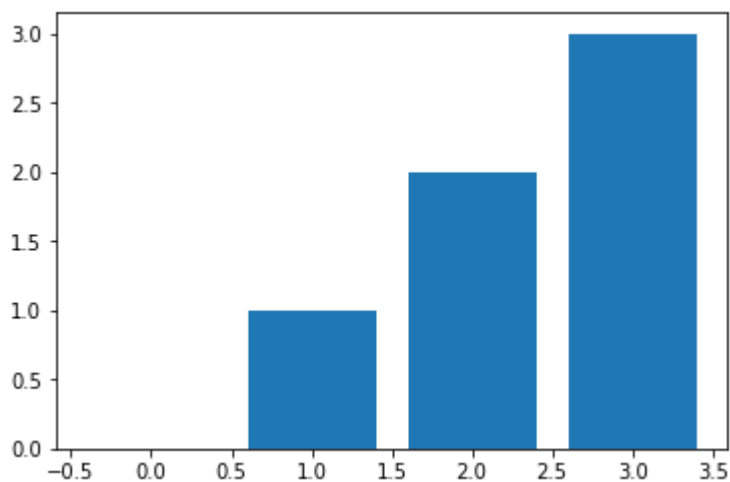


In [21]:

```
plt.bar(range(4), range(4))
```

Out[21]:

<BarContainer object of 4 artists>



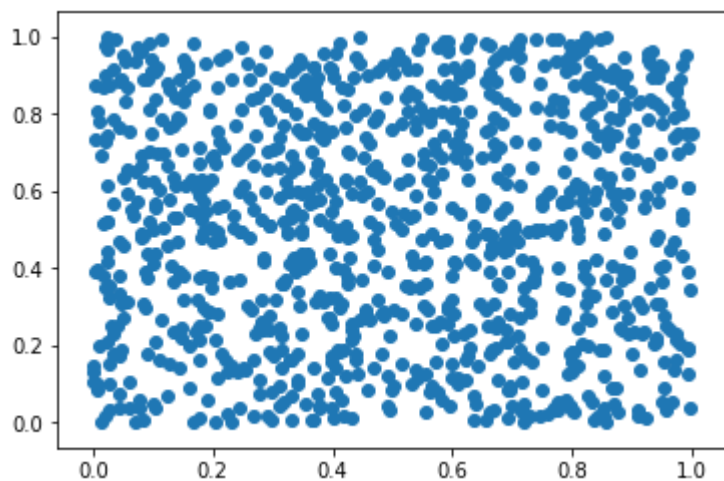
散点图 `matplotlib.pyplot.scatter(*args, **kwargs)` 就是呈现在二维平面的一些点，这种图像的需求也是非常常见的。比如，我们通过 GPS 采集的数据点，它会包含经度以及纬度两个值，这样的情况就可以绘制成散点图。

In [22]:

```
# X,y 的坐标均有 numpy 在 0 到 1 中随机生成 1000 个值
X = np.random.rand(1000)
y = np.random.rand(1000)
# 向方法中 `*args` 输入 X, y 坐标
plt.scatter(X, y)
```

Out[22]:

<matplotlib.collections.PathCollection at 0x7fcc3de7e0f0>

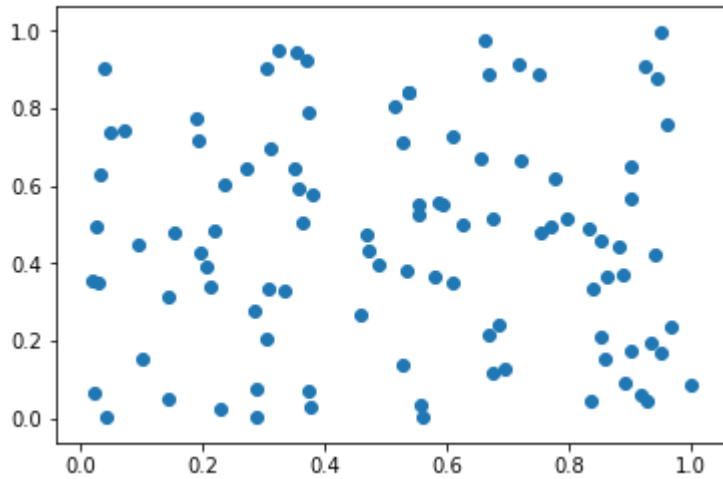


In [30]:

```
x = np.random.rand(100)
y = np.random.rand(100)
plt.scatter(x,y)
```

Out[30]:

<matplotlib.collections.PathCollection at 0x7fcc3dbf5cc0>



饼状图 `matplotlib.pyplot.pie(*args, **kwargs)` 在有限列表以百分比呈现时特别有用，你可以很清晰地看出来各类别之间的大小关系，以及各类别占总体的比例。

In []:

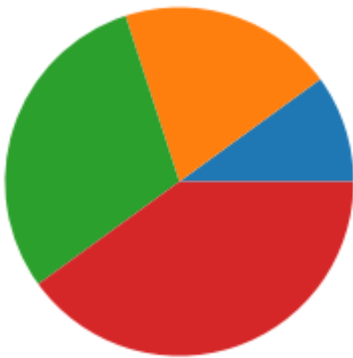
```
plt.pie([1, 2, 3, 4, 5])
```

In [31]:

```
plt.pie(range(1,5))
```

Out[31]:

```
(<matplotlib.patches.Wedge at 0x7fcc3dbd0780>,  
<matplotlib.patches.Wedge at 0x7fcc3dbd0c50>,  
<matplotlib.patches.Wedge at 0x7fcc3dbda160>,  
<matplotlib.patches.Wedge at 0x7fcc3dbda630>],  
[Text(1.0461621663333946, 0.3399186987098808, ''),  
Text(0.33991867422268784, 1.0461621742897658, ''),  
Text(-1.0461621902025062, 0.3399186252483017, ''),  
Text(0.3399188211458418, -1.0461621265515308, '')] )
```



量场图 `matplotlib.pyplot.quiver(*args, **kwargs)` 就是由向量组成的图像，在气象学等方面被广泛应用。从图像的角度来看，量场图就是带方向的箭头符号。

In []:

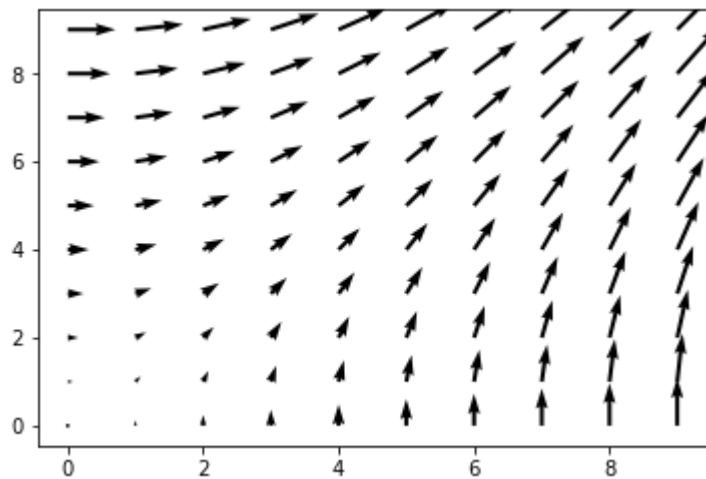
```
X, y = np.mgrid[0:10, 0:10]  
plt.quiver(X, y)
```

In [44]:

```
x, y = np.mgrid[0:10, 0:10]
plt.quiver(x,y)
```

Out[44]:

<matplotlib.quiver.Quiver at 0x7fcc3db2a4a8>



中学学习地理的时候，我们就知道等高线了。等高线图 `matplotlib.pyplot.contourf(*args, **kwargs)` 是工程领域经常接触的一类图，它的绘制过程稍微复杂一些。

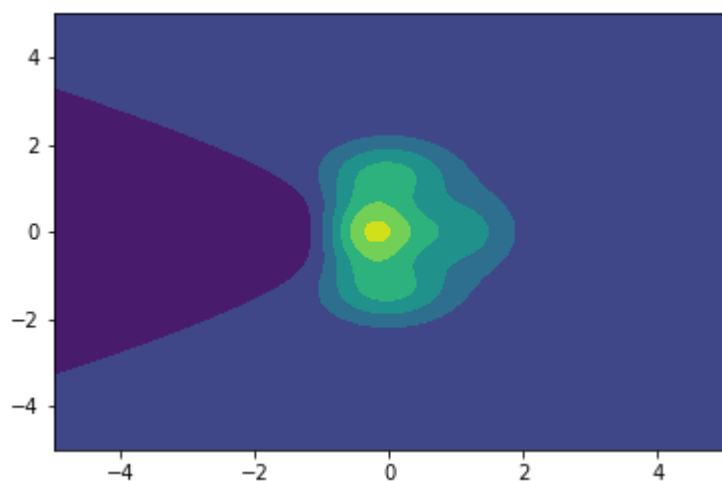
In [45]:

```
# 生成网格矩阵
x = np.linspace(-5, 5, 500)
y = np.linspace(-5, 5, 500)
X, Y = np.meshgrid(x, y)
# 等高线计算公式
Z = (1 - X / 2 + X ** 3 + Y ** 4) * np.exp(-X ** 2 - Y ** 2)

plt.contourf(X, Y, Z)
```

Out[45]:

<matplotlib.contour.QuadContourSet at 0x7fcc3da86b00>

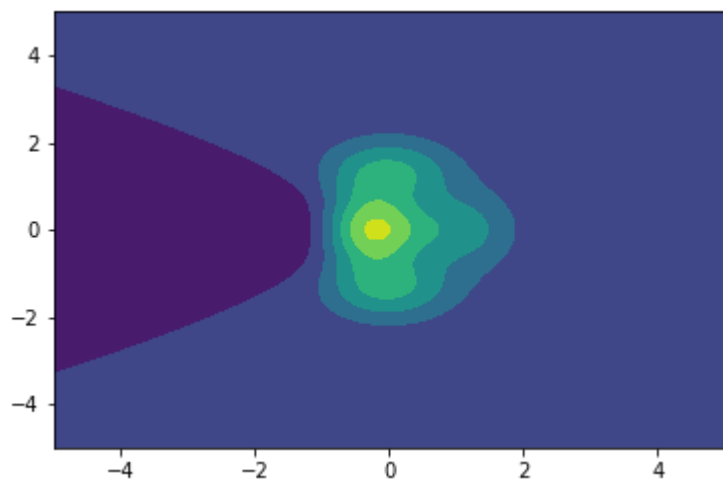


In [47]:

```
x = np.linspace(-5, 5, 500)
y = np.linspace(-5, 5, 500)
x, y = np.meshgrid(x, y)
z = (1 - X / 2 + X ** 3 + Y ** 4) * np.exp(-X ** 2 - Y ** 2)
plt.contourf(x, y, z)
```

Out[47]:


<matplotlib.contour.QuadContourSet at 0x7fcc3d622e10>



定义图形样式

上面，我们绘制了简单的基础图形，但这些图形都不美观。你可以通过更多的参数来让图形变得更漂亮。

我们已经知道了，线形图通过 `matplotlib.pyplot.plot(*args, **kwargs)` 方法绘出。其中，`args` 代表数据输入，而 `kwargs` 的部分就是用于设置样式参数了。

二维线形图  包含的参数 (https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.plot.html) 超过 40 余项，其中常用的也有 10 余项，选取一些比较有代表性的参数列举如下：

参数	含义
<code>alpha=</code>	设置线型的透明度，从 0.0 到 1.0
<code>color=</code>	设置线型的颜色
<code>fillstyle=</code>	设置线型的填充样式
<code>linestyle=</code>	设置线型的样式
<code>linewidth=</code>	设置线型的宽度
<code>marker=</code>	设置标记点的样式

.....

.....

至于每一项参数包含的设置选项，大家需要通过 [🔗 官方文档](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html) (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html) 详细了解。

下面，我们重新绘制一个三角函数图形。

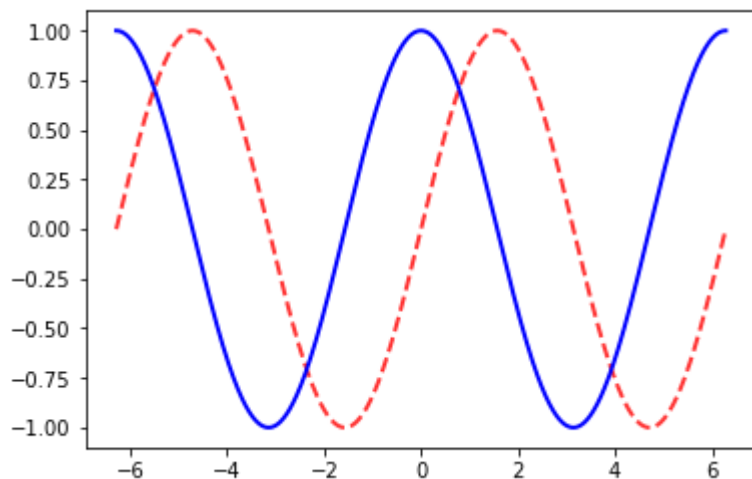
In [48]:

```
# 在  $-2\pi$  和  $2\pi$  之间等间距生成 1000 个值，也就是  $x$  坐标
X = np.linspace(-2 * np.pi, 2 * np.pi, 1000)
# 计算  $\sin()$  对应的纵坐标
y1 = np.sin(X)
# 计算  $\cos()$  对应的纵坐标
y2 = np.cos(X)

# 向方法中 `*args` 输入  $x, y$  坐标
plt.plot(X, y1, color='r', linestyle='--', linewidth=2, alpha=0.8)
plt.plot(X, y2, color='b', linestyle='-', linewidth=2)
```

Out[48]:

[<matplotlib.lines.Line2D at 0x7fcc3d658be0>]

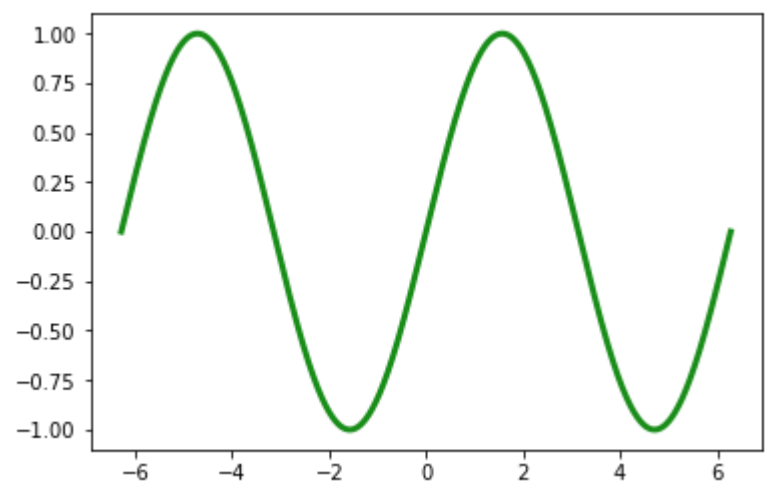


In [56]:

```
plt.plot(X,y1, color = 'g', linestyle = '-', linewidth =3, alpha = 0.9 )
```

Out[56]:

[<matplotlib.lines.Line2D at 0x7fcc3ce566d8>]



散点图也是相似的，它们的很多样式参数都是大同小异，需要大家阅读 [官方文档](https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html) (https://matplotlib.org/api/_as_gen/matplotlib.pyplot.scatter.html) 详细了解。

参数	含义
s=	散点大小
c=	散点颜色
marker=	散点样式
cmap=	定义多类别散点的颜色
alpha=	点的透明度
edgecolors=	散点边缘颜色

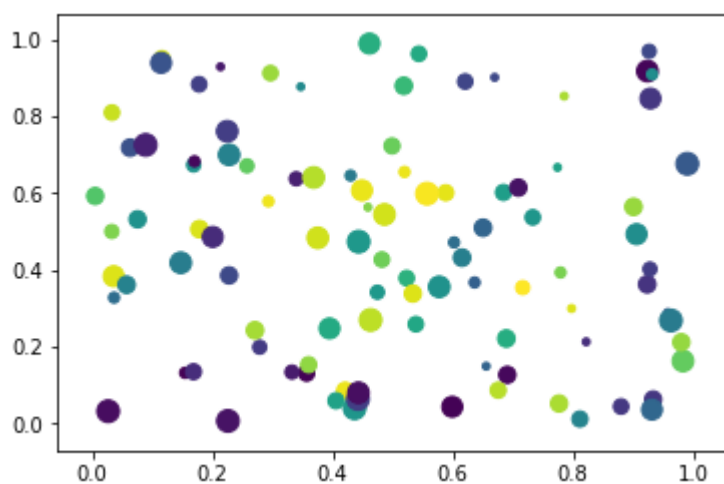
In [57]:

```
# 生成随机数据
x = np.random.rand(100)
y = np.random.rand(100)
colors = np.random.rand(100)
size = np.random.normal(50, 60, 10)

plt.scatter(x, y, s=size, c=colors) # 绘制散点图
```

Out[57]:

<matplotlib.collections.PathCollection at 0x7fcc3ce3dc88>



In [66]:

```
plt.scatter(x, y, s=np.random.normal(100,200,300), c=colors) # 绘制散点图
```

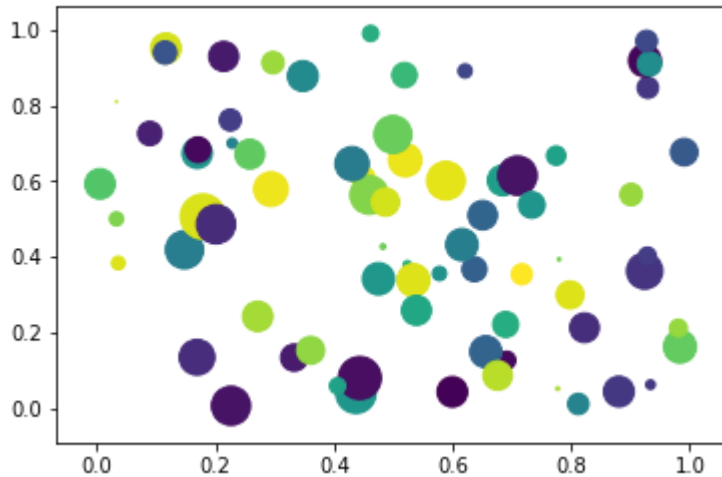
/opt/conda/lib/python3.6/site-packages/matplotlib/collections.py:874:

RuntimeWarning: invalid value encountered in sqrt

```
scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
```

Out[66]:

<matplotlib.collections.PathCollection at 0x7fcc3cb5bf28>



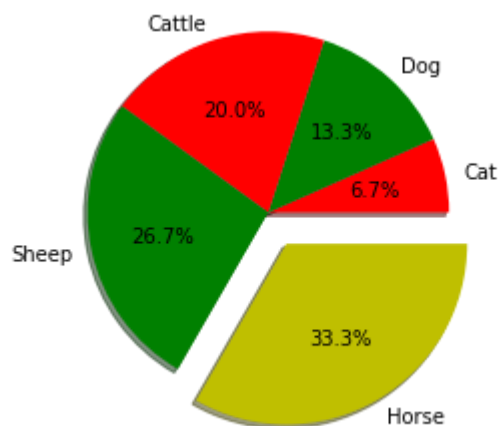
饼状图通过 `matplotlib.pyplot.pie()` 绘出。我们也可以进一步设置它的颜色、标签、阴影等各类样式。下面就绘出一个示例。

In [74]:

```
label = 'Cat', 'Dog', 'Cattle', 'Sheep', 'Horse' # 各类别标签
color = 'r', 'g', 'r', 'g', 'y' # 各类别颜色
size = [1, 2, 3, 4, 5] # 各类别占比
explode = (0, 0, 0, 0, 0.2) # 各类别的偏移半径
# 绘制饼状图
plt.pie(size, colors=color, explode=explode,
        labels=label, shadow=True, autopct='%1.1f%%')
# 饼状图呈正圆
plt.axis('equal')
```

Out[74]:

```
(-1.1126474248725045, 1.205364242938969, -1.282680566307163, 1.1257798
39003857)
```

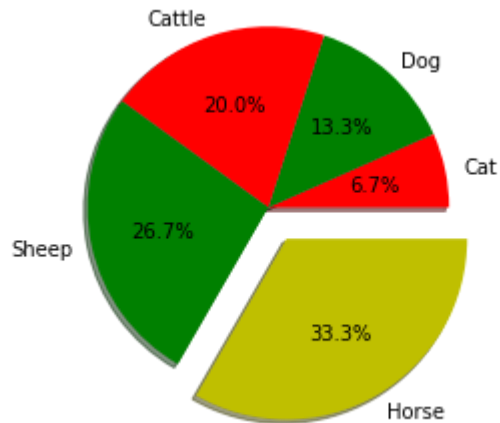


In [71]:

```
plt.pie(size, colors=color, explode=explode,  
        labels=label, shadow=True, autopct='%1.1f%%')  
plt.axis('equal')
```

Out[71]:

```
(-1.1126474248725045, 1.205364242938969, -1.282680566307163, 1.1257798  
39003857)
```



组合图形样式

上面演示了单个简单图像的绘制。实际上，我们往往会遇到将几种类型的一样的图放在一张图内显示，也就是组合图的绘制。其实很简单，你只需要将所需图形的代码放置在一起就可以了，比如绘制一张包含柱形图和折线图的组合图。

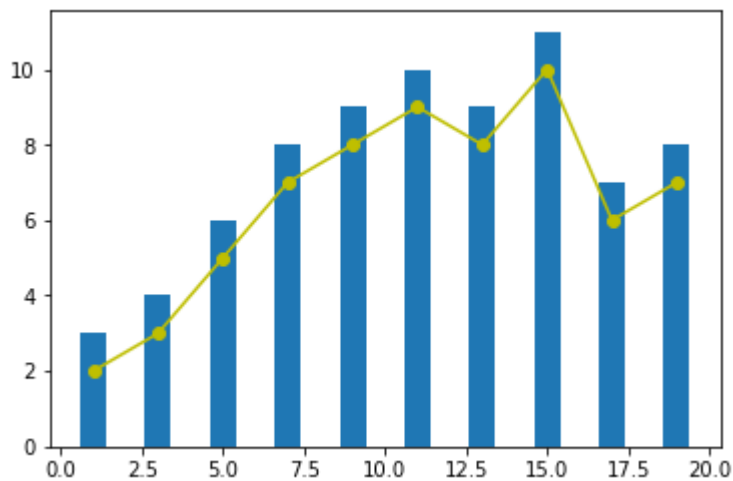
In [75]:

```
x = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
y_bar = [3, 4, 6, 8, 9, 10, 9, 11, 7, 8]
y_line = [2, 3, 5, 7, 8, 9, 8, 10, 6, 7]

plt.bar(x, y_bar)
plt.plot(x, y_line, '-o', color='y')
```

Out[75]:

[<matplotlib.lines.Line2D at 0x7fcc3cac17b8>]

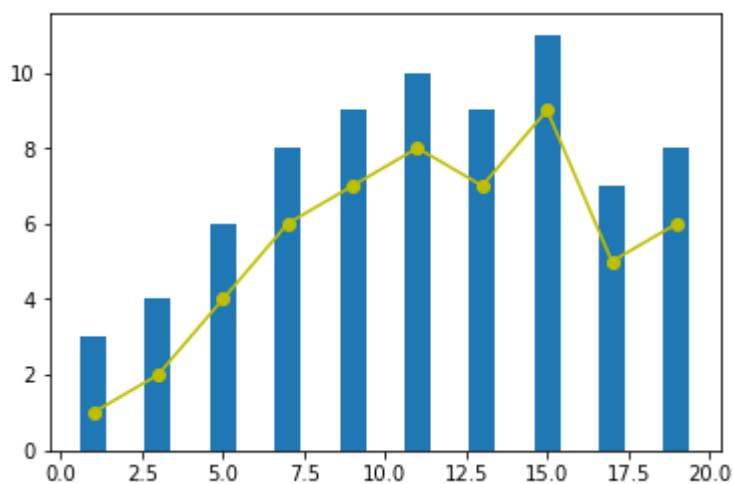


In [89]:

```
plt.bar(x, y_bar)
plt.plot(x, np.array(y_bar) - 2, '-o', color='y')
```

Out[89]:

[<matplotlib.lines.Line2D at 0x7fcc3c794a20>]



当然，并不是任何的代码放在一起都是组合图。上面，两张图的横坐标必须共享，才能够被 Matplotlib 自动判断为组合图效果。

定义图形位置

在图形的绘制过程中，你可能需要调整图形的位置，或者把几张单独的图形拼接在一起。此时，我们就需要引入 `plt.figure` 图形对象了。

下面，我们绘制一张自定义位置的图形。

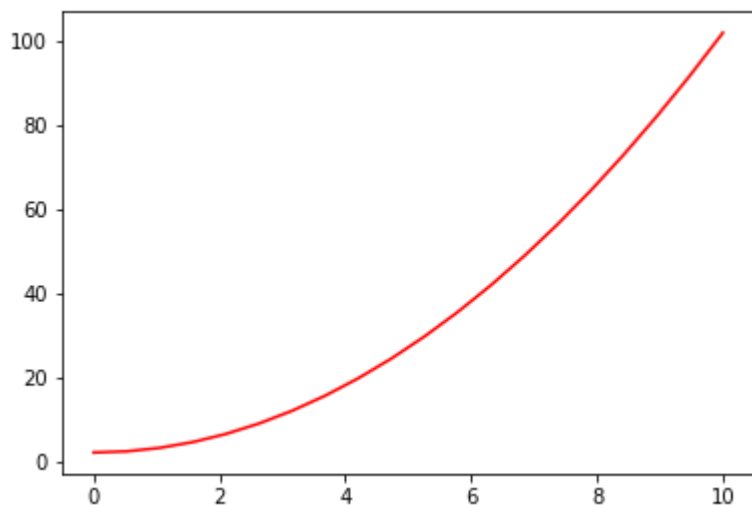
In [90]:

```
x = np.linspace(0, 10, 20) # 生成数据
y = x * x + 2

fig = plt.figure() # 新建图形对象
axes = fig.add_axes([0.5, 0.5, 0.8, 0.8]) # 控制画布的左, 下, 宽度, 高度
axes.plot(x, y, 'r')
```

Out[90]:

[<matplotlib.lines.Line2D at 0x7fcc3c794da0>]

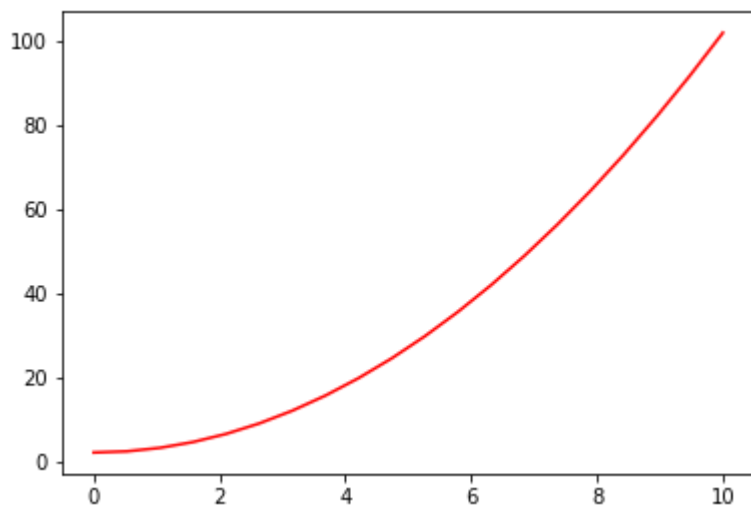


In [102]:

```
fig = plt.figure() # 新建图形对象
axes = fig.add_axes([0, 0, 0.8, 0.8])
axes.plot(x, y, 'r')
```

Out[102]:

[<matplotlib.lines.Line2D at 0x7fcc3c89e630>]



上面的绘图代码中，你可能会对 **figure** 和 **axes** 产生疑问。Matplotlib 的 API 设计的非常符合常理，在这里，**figure** 相当于绘画用的画板，而 **axes** 则相当于铺在画板上的画布。我们将图像绘制在画布上，于是就有了 **plot**，**set_xlabel** 等操作。



借助于图形对象，我们可以实现大图套小图的效果。

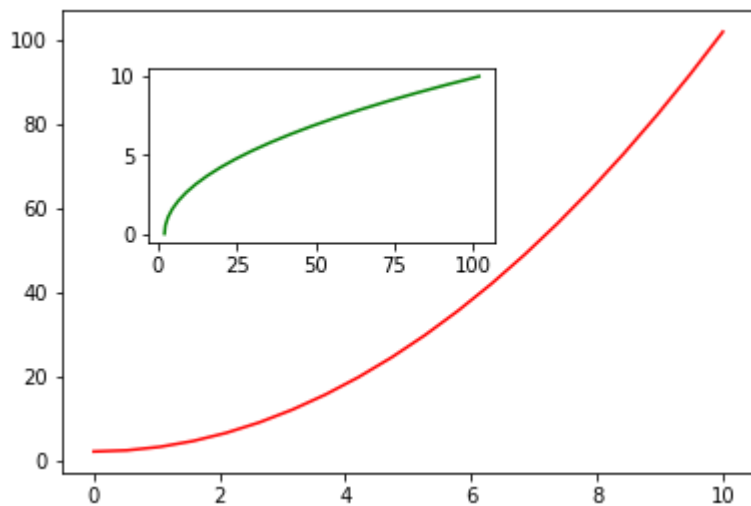
In [103]:

```
fig = plt.figure() # 新建画板
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # 大画布
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # 小画布

axes1.plot(x, y, 'r') # 大画布
axes2.plot(y, x, 'g') # 小画布
```

Out[103]:

[<matplotlib.lines.Line2D at 0x7fcc3c93ff98>]



In []:

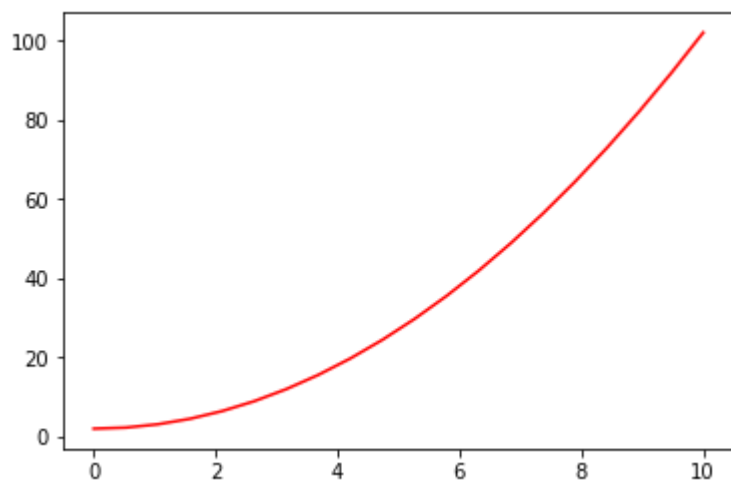
上面的绘图代码中，你已经学会了使用 `add_axes()` 方法向我们设置的画板 `figure` 中添加画布 `axes`。在 Matplotlib 中，还有一种添加画布的方式，那就是 `plt.subplots()`，它和 `axes` 都等同于画布。

In [111]:

```
fig, axes = plt.subplots()
axes.plot(x, y, 'r')
```

Out[111]:

[<matplotlib.lines.Line2D at 0x7fcc3c3ab160>]



In [110]:

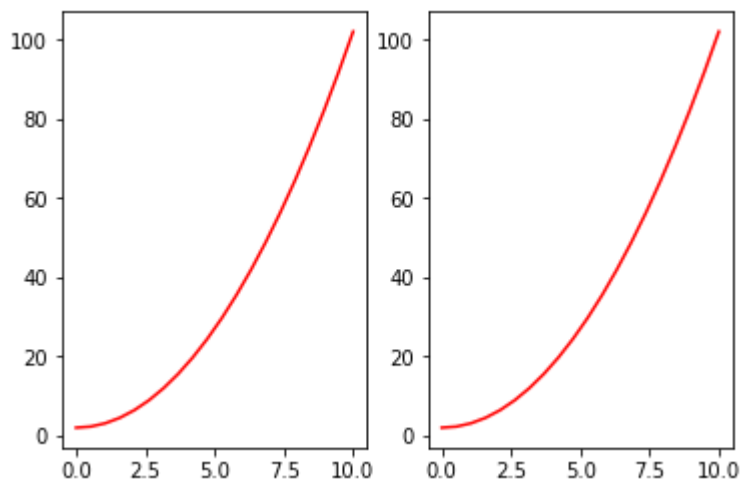
Out[110]:

[<matplotlib.lines.Line2D at 0x7fcc3c40ecc0>]

借助于 `plt.subplots()`，我们就可以实现子图的绘制，也就是将多张图按一定顺序拼接在一起。

In [105]:

```
fig, axes = plt.subplots(nrows=1, ncols=2) # 子图为 1 行, 2 列
for ax in axes:
    ax.plot(x, y, 'r')
```



In []:

通过设置 `plt.subplots` 的参数，可以实现调节画布尺寸和显示精度。

In []:

```
fig, axes = plt.subplots(
    figsize=(16, 9), dpi=50) # 通过 figsize 调节尺寸, dpi 调节显示精度
axes.plot(x, y, 'r')
```

In []:

规范绘图方法

上面，我们已经入门了 Matplotlib 的绘图方法。由于 Matplotlib 的灵活性，很多方法都可以画出图形

来。但为了避免「想怎么画，就怎么画」的问题，我们需要根据自己的习惯，约定一套比较规范的绘图方法。

首先，任何图形的绘制，都建议通过 `plt.figure()` 或者 `plt.subplots()` 管理一个完整的图形对象。而不是简单使用一条语句，例如 `plt.plot(...)` 来绘图。

管理一个完整的图形对象，有很多好处。在图形图形的基础上，给后期添加图例，图形样式，标注等预留了很大的空间。除此之外。代码看起来也更加规范，可读性更强。

接下来，我们就通过几组例子来演示规范的绘图方法。

添加图标题、图例

绘制包含图标题、坐标轴标题以及图例的图形，举例如下：

In [115]:

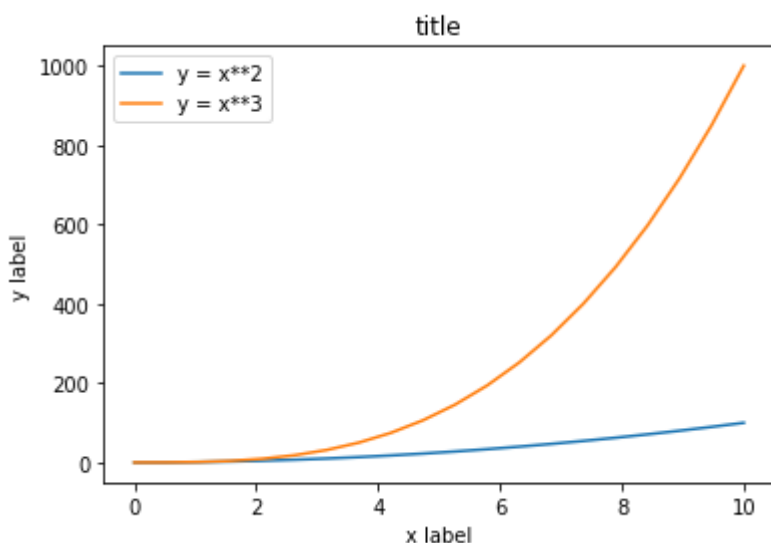
```
fig, axes = plt.subplots()

axes.set_xlabel('x label') # 横轴名称
axes.set_ylabel('y label')
axes.set_title('title') # 图形名称

axes.plot(x, x**2)
axes.plot(x, x**3)
axes.legend(["y = x**2", "y = x**3"], loc=0) # 图例
```

Out[115]:

<matplotlib.legend.Legend at 0x7fcc3c1e88d0>



In []:

图例中的 `loc` 参数标记图例位置，`1, 2, 3, 4` 依次代表：右上角、左上角、左下角，右下角；`0` 代表自适应

线型、颜色、透明度

在 Matplotlib 中，你可以设置线的颜色、透明度等其他属性。

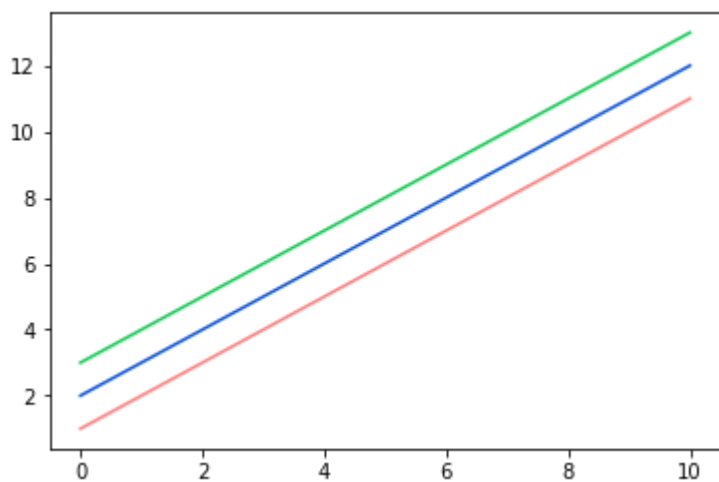
In [116]:

```
fig, axes = plt.subplots()

axes.plot(x, x+1, color="red", alpha=0.5)
axes.plot(x, x+2, color="#1155dd")
axes.plot(x, x+3, color="#15cc55")
```

Out[116]:

[<matplotlib.lines.Line2D at 0x7fcc3c14e2b0>]



In []:

而对于线型而言，除了实线、虚线之外，还有很多丰富的线型可供选择。

In [117]:

```
fig, ax = plt.subplots(figsize=(12, 6))

# 线宽
ax.plot(x, x+1, color="blue", linewidth=0.25)
ax.plot(x, x+2, color="blue", linewidth=0.50)
ax.plot(x, x+3, color="blue", linewidth=1.00)
ax.plot(x, x+4, color="blue", linewidth=2.00)

# 虚线类型
ax.plot(x, x+5, color="red", lw=2, linestyle='-')
ax.plot(x, x+6, color="red", lw=2, ls='-.')
ax.plot(x, x+7, color="red", lw=2, ls=':')

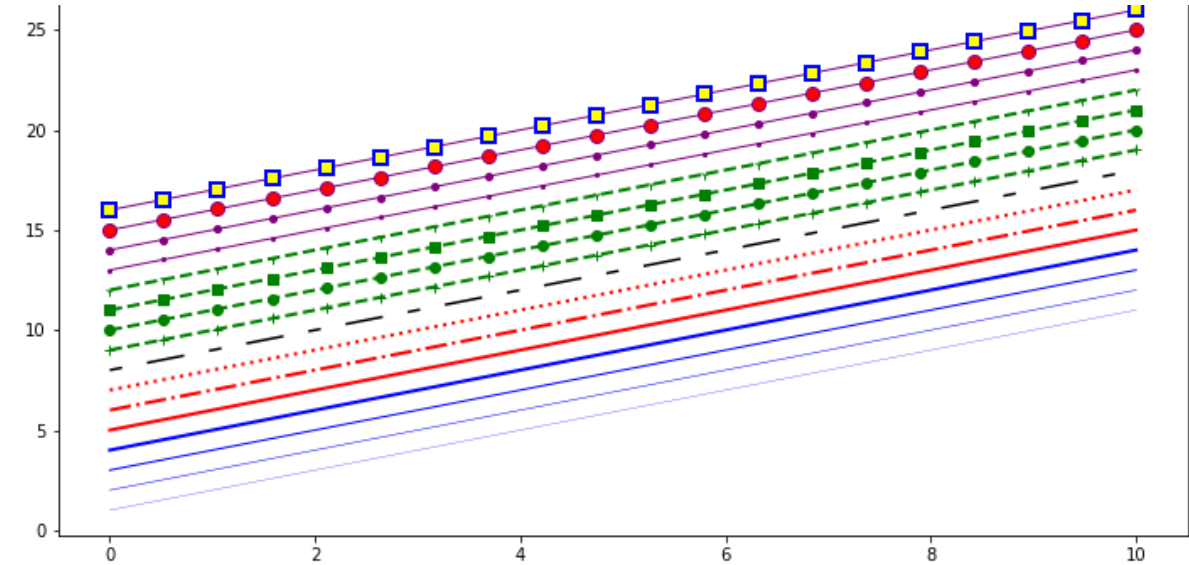
# 虚线交错宽度
line, = ax.plot(x, x+8, color="black", lw=1.50)
line.set_dashes([5, 10, 15, 10])

# 符号
ax.plot(x, x + 9, color="green", lw=2, ls='--', marker='+')
ax.plot(x, x+10, color="green", lw=2, ls='--', marker='o')
ax.plot(x, x+11, color="green", lw=2, ls='--', marker='s')
ax.plot(x, x+12, color="green", lw=2, ls='--', marker='1')

# 符号大小和颜色
ax.plot(x, x+13, color="purple", lw=1, ls='-', marker='o', markersize=2)
ax.plot(x, x+14, color="purple", lw=1, ls='-', marker='o', markersize=4)
ax.plot(x, x+15, color="purple", lw=1, ls='-',
        marker='o', markersize=8, markerfacecolor="red")
ax.plot(x, x+16, color="purple", lw=1, ls='-', marker='s', markersize=8,
        markerfacecolor="yellow", markeredgewidth=2, markeredgecolor="blue")
```

Out[117]:

[<matplotlib.lines.Line2D at 0x7fcc3c0cf940>]



In []:

画布网格、坐标轴范围

有些时候，我们可能需要显示画布网格或调整坐标轴范围。设置画布网格和坐标轴范围。这里，我们通过指定 `axes[0]` 序号，来实现子图的自定义顺序排列。

In [118]:

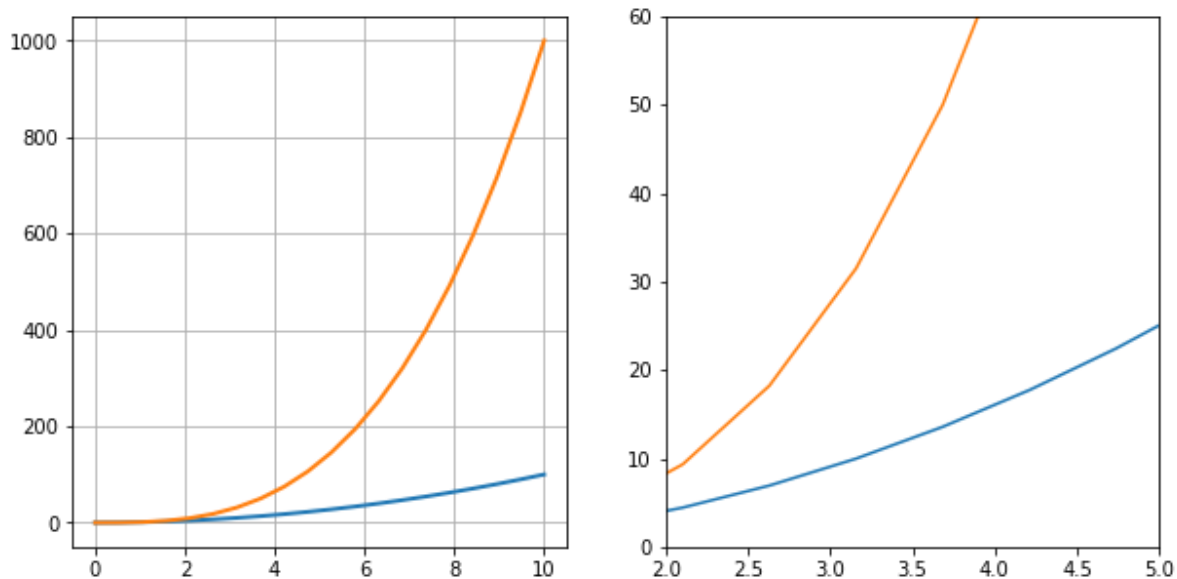
```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# 显示网格
axes[0].plot(x, x**2, x, x**3, lw=2)
axes[0].grid(True)

# 设置坐标轴范围
axes[1].plot(x, x**2, x, x**3)
axes[1].set_ylim([0, 60])
axes[1].set_xlim([2, 5])
```

Out[118]:

(2, 5)



In []:

除了折线图，Matplotlib 还支持绘制散点图、柱状图等其他常见图形。下面，我们绘制由散点图、梯度图、条形图、面积图构成的子图。

In [119]:

```
n = np.array([0, 1, 2, 3, 4, 5])

fig, axes = plt.subplots(1, 4, figsize=(16, 5))

axes[0].scatter(x, x + 0.25*np.random.randn(len(x)))
axes[0].set_title("scatter")

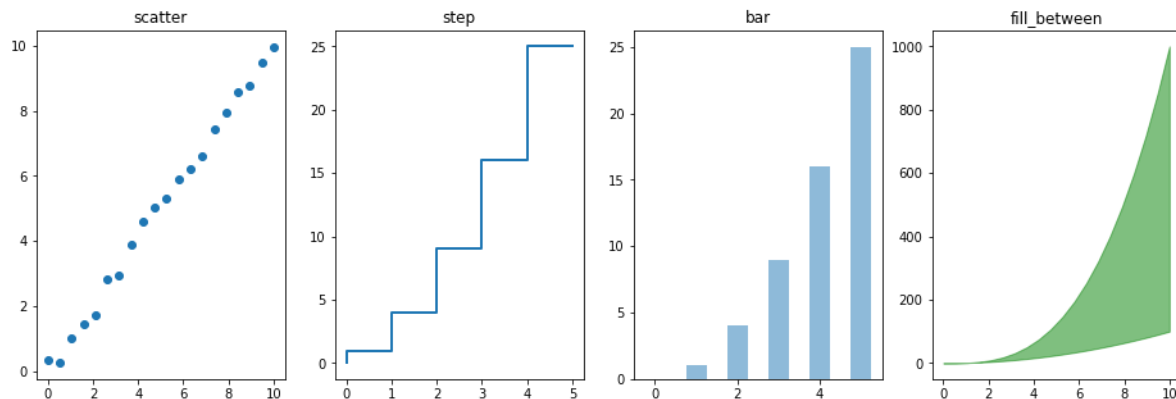
axes[1].step(n, n**2, lw=2)
axes[1].set_title("step")

axes[2].bar(n, n**2, align="center", width=0.5, alpha=0.5)
axes[2].set_title("bar")

axes[3].fill_between(x, x**2, x**3, color="green", alpha=0.5)
axes[3].set_title("fill_between")
```

Out[119]:

Text(0.5, 1.0, 'fill_between')



In []:

图形标注方法

当我们绘制一些较为复杂的图像时，阅读对象往往很难全面理解图像的含义。而此时，图像标注往往会起到画龙点睛的效果。图像标注，就是在画面上添加文字注释、指示箭头、图框等各类标注元素。

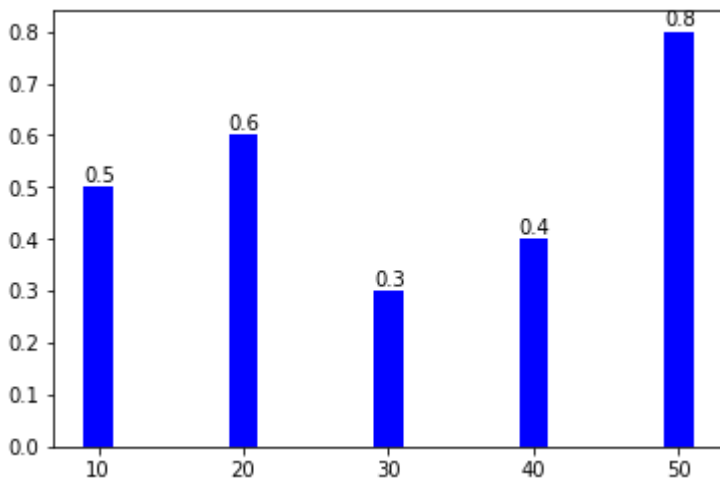
Matplotlib 中，文字标注的方法由 `matplotlib.pyplot.text()` 实现。最基本的样式为 `matplotlib.pyplot.text(x, y, s)`，其中 `x, y` 用于标注位置定位，`s` 代表标注的字符串。除此之外，你还可以通过 `fontsize=`，`horizontalalignment=` 等参数调整标注字体的大小，对齐样式等。

下面，我们举一个对柱形图进行文字标注的示例。

In [121]:

```
fig, axes = plt.subplots()

x_bar = [10, 20, 30, 40, 50] # 柱形图横坐标
y_bar = [0.5, 0.6, 0.3, 0.4, 0.8] # 柱形图纵坐标
bars = axes.bar(x_bar, y_bar, color='blue', label=x_bar, width=2) # 绘制柱形图
for i, rect in enumerate(bars):
    x_text = rect.get_x() # 获取柱形图横坐标
    y_text = rect.get_height() + 0.01 # 获取柱子的高度并增加 0.01
    plt.text(x_text, y_text, '%.1f' % y_bar[i]) # 标注文字
```



In []:

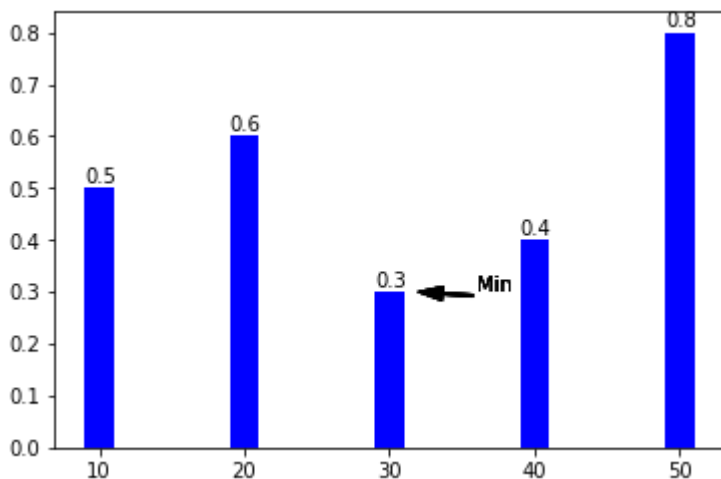
除了文字标注之外，还可以通过 `matplotlib.pyplot.annotate()` 方法向图像中添加箭头等样式标注。接下来，我们向上面的例子中增添一行增加箭头标记的代码。

In [123]:

```
fig, axes = plt.subplots()

bars = axes.bar(x_bar, y_bar, color='blue', label=x_bar, width=2) # 绘制柱形图
for i, rect in enumerate(bars):
    x_text = rect.get_x() # 获取柱形图横坐标
    y_text = rect.get_height() + 0.01 # 获取柱子的高度并增加 0.01
    plt.text(x_text, y_text, '%.1f' % y_bar[i]) # 标注文字

# 增加箭头标注
plt.annotate('Min', xy=(32, 0.3), xytext=(36, 0.3),
            arrowprops=dict(facecolor='black', width=1, headwidth=7))
```



In []:

上面的示例中，`xy=()` 表示标注终点坐标，`xytext=()` 表示标注起点坐标。在箭头绘制过程中，`arrowprops=()` 用于设置箭头样式，`facecolor=` 设置颜色，`width=` 设置箭尾宽度，`headwidth=` 设置箭头宽度，可以通过 `arrowstyle=` 改变箭头的样式。

兼容 MATLAB 代码风格接口

提示： 本部分内容适合于之前有 MATLAB 基础的用户了解，其他读者可以直接跳过。

相信很多学理工科的同学都使用过 MATLAB，它是一种用于算法开发、数据可视化、数据分析以及数值计算的高级技术计算语言和交互式环境。而在 Matplotlib 中，也提供了和 MATLAB 相似的 API。对

于使用过 MATLAB 的同学而言，这将是入门 Matplotlib 最快的方式。

使用 Matplotlib 提供的兼容 MATLAB API，需要导入 pylab 模块：

In [124]:

```
from matplotlib import pylab
```

In []:

使用 NumPy 生成随机数据：

In []:

```
x = np.linspace(0, 10, 20)
y = x * x + 2
```

In []:

只需要 1 句命令就可以完成绘图：

In []:

```
pylab.plot(x, y, 'r') # 'r' 代表 red
```

In []:

如果我们要绘制子图，就可以使用 `subplot` 方法绘制子图：

In []:

```
pylab.subplot(1, 2, 1) # 括号中内容代表 (行, 列, 索引)
pylab.plot(x, y, 'r--') # '' 中的内容确定了颜色和线型

pylab.subplot(1, 2, 2)
pylab.plot(y, x, 'g*-')
```

In []:

使用兼容 MATLAB 风格的 API 的好处在于，如果熟悉 MATLAB，那么将很快上手使用 Python 绘图。不过，除了一些简单的图形之外，并不鼓励使用兼容 MATLAB 的 API。

实验更加建议学习和使用前面介绍的 Matplotlib 提供的面向对象 API，它更加强大和好用。

实验总结

通过这节实验课程的学习，相信你已经初步掌握了使用 Matplotlib 绘图的方法和技巧。当然，如果你对 Matplotlib 非常感兴趣，也可以通过实验楼其他课程学习 Matplotlib 的更多内容。

➡ 继续学习

-  Seaborn 数据可视化基础课程_(<https://www.shiyanlou.com/courses/892>)

© 本课程内容，由作者授权实验楼发布，未经允许，禁止转载、下载及非法传播。