Numpy精简文档

0.1 NumPy 安装

- pip 安装命令行执行：pip install numpy
- Anaconda 平台安装：conda install numpy
- 安装完成测试：import numpy, 如果安装完成导入之后不报错，则证明安装成功。

0.2 NumPy 基本数据类型

- bool_ 布尔型数据类型（True 或者 False）
- int_ 默认的整数类型（类似于 C 语言中的 long，int32 或 int64）
- intc 与 C 的 int 类型一样，一般是 int32 或 int 64
- intp 用于索引的整数类型（类似于 C 的 ssize_t，一般情况下仍然是 int32 或 int64）
- int8 字节（-128 to 127）
- int16 整数（-32768 to 32767）
- int32 整数（-2147483648 to 2147483647）
- int64 整数（-9223372036854775808 to 9223372036854775807）
- uint8 无符号整数（0 to 255）
- uint16 无符号整数（0 to 65535）
- uint32 无符号整数（0 to 4294967295）
- uint64 无符号整数（0 to 18446744073709551615）
- float_ float64 类型的简写
- float16 半精度浮点数，包括：1 个符号位，5 个指数位，10 个尾数位
- float32 单精度浮点数，包括：1 个符号位，8 个指数位，23 个尾数位
- float64 双精度浮点数，包括：1 个符号位，11 个指数位，52 个尾数位
- complex_ complex128 类型的简写，即 128 位复数
- complex64 复数，表示双 32 位浮点数（实数部分和虚数部分）
- complex128 复数，表示双 64 位浮点数（实数部分和虚数部分）
  numpy 的数值类型实际上是 dtype 对象的实例，并对应唯一的字符，包括 np.bool_，np.int32，np.float32，等等。

0.3 NumPy 数组

- NumPy 数组是一个多维数组对象 Ndarray，由数组实际数据与描述这些数据的元数据组成。
- 一般来说，大部分操作仅针对于元数据，而不改变底层实际的数据。
- 需要注意的是，NumPy 数组的下标起始位 0，同一个 NumPy 数组中的所有元素类型必须相同，NumPy 数组的创建访问中 () 与 [] 功能没有什么区别。

0.4 NumPy 基础数组创建

NumPy 数组有很多种创建方式，一般常用的方式是将 list 转换为 np.array:

```
In [1]: import numpy as np
        a = [1,2,3]
        b = np.array(a)
        b

Out[1]: array([1, 2, 3])
```

或者是：

```
In [2]: a = np.array([1,2,3], float)
        a

Out[2]: array([1., 2., 3.])
```

这里的 float 用于指定数据类型，可以不写，也可以写为 float 等其它类型, 需要注意的是，使用 array() 函数创建数组时，参数必须是方括号括起来的列表。
下面创建多维数组：

```
In [3]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
        a

Out[3]: array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])
```

0.5 NumPy 特殊数组创建

NumPy 提供了一些使用占位符创建数组的函数，这些数组在机器学习中是比较常见的。在创建过程中我们同样可以使用 dtype = int 指定元素类型：

```
In [4]: a = np.zeros((2,3), dtype = float)
        a

Out[4]: array([[0., 0., 0.],
               [0., 0., 0.]])
```

创建一个全为 1 的数组：

```
In [5]: a = np.ones((3, 3))
        a

Out[5]: array([[1., 1., 1.],
               [1., 1., 1.],
               [1., 1., 1.]])
```

创建等差数列，从 1 开始，5 结束，0.5 为差的等差数列，最后一项一定小于 5：

```
In [6]: a = np.arange(1, 5, 0.5)
        a

Out[6]: array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

创建单位矩阵（矩阵的乘法中，有一种矩阵起着特殊的作用，如同数的乘法中的 1，这种矩阵被称为单位矩阵。它是个方阵，从左上角到右下角的对角线（称为主对角线）上的元素均为 1。），参数可以指定大小：

```
In [7]: a = np.eye(3)
        a

Out[7]: array([[1., 0., 0.],
               [0., 1., 0.],
               [0., 0., 1.]])
```

生成指定长度，在 [0,1) 之间平均分布的随机数组：

```
In [8]: np.random.random(5)

Out[8]: array([0.39448889, 0.34565971, 0.88672892, 0.94258034, 0.05633518])
```

生成指定长度，符合正太分布的随机数组，指定其均值为 0，标准差为 0.1：

```
In [9]: mu, sigma = 0, 0.1
        np.random.normal(mu,sigma,5)

Out[9]: array([-0.02168114,  0.08116314, -0.10573889, -0.0143433 , -0.10197173])
```

0.6 NumPy 数组的访问

和 list 的访问形式基本一致，支持切片操作，我们可以切片每一个维度，索引每一个维度：

```
In [10]: a = np.array([(1,2), (3,4), (5,6)]) a[0]
```

```
Out[10]: array([1, 2])

In [11]: a[1:]

Out[11]: array([[3, 4],
                [5, 6]])

In [12]: a[:, :1]

Out[12]: array([[1],
                [3],
                [5]])

In [13]: a[1][1]

Out[13]: 4
```

使用 take() 函数进行操作，take(indices[, axis, out, mode]) 提取指定索引位置的数据, 并以一维数组或者矩阵返回 (主要取决 axis)：

```
In [14]: a = np.array([1,2,3])
         a.take(1)

Out[14]: 2

In [15]: b = np.array([(1,2,3), (4,5,6)])
         b.take(1, axis = 1)

Out[15]: array([2, 5])
```

0.7 NumPy 数组的遍历

一维数组遍历：

```
In [16]: a = np.array([1,2,3])
         for i in a:
             print(i)
```

```
1
2
3
```

多维数组的遍历：

```
In [17]: a = np.array([(1,2), (3,4), (5,6)])
         for i,j in a:
             print(i*j)
```

2
12
30

0.8 NumPy 数组的常用属性

比较常用的属性有：* ndarray.ndim: 数组的维度（数组轴的个数），等于秩 * ndarray.shape: 数组的大小。为一个表示数组在每个维度上大小的整数元组。例如二维数组中，表示数组的” 行数” 和” 列数” * ndarray.size: 数组元素的总个数，等于 shape 属性中元组元素的乘积 * ndarray.dtype: 表示数组中元素类型的对象 *ndarray.itemsize:数组中每个元素的字节大小 *ndarray.data:包含实际数组元素的缓冲区，一般不用

```
In [18]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
         a.ndim
```

Out[18]: 2

```
In [19]: a.shape
```

Out[19]: (3, 3)

```
In [20]: a.size
```

Out[20]: 9

```
In [21]: a.dtype
```

Out[21]: dtype('int32')

```
In [22]: a.itemsize
```

Out[22]: 4

```
In [23]: a.data
```

Out[23]: <memory at 0x00000000058F8C18>

0.9 NumPy 数组的基本操作

in:检测数值是否在数组中

```
In [24]: a = np.array([(1,2), (3,4)])
         3 in a
```

Out[24]: True

```
In [25]: 5 in a

Out[25]: False
```

reshape: 数组的重排列，例如将一个 3 维数组转变为 1 维（元素数一定要保持不变）

```
In [26]: a = np.zeros([2,3,4])
         a

Out[26]: array([[[0., 0., 0., 0.],
                 [0., 0., 0., 0.],
                 [0., 0., 0., 0.]],

                [[0., 0., 0., 0.],
                 [0., 0., 0., 0.],
                 [0., 0., 0., 0.]]])

In [27]: a.reshape(24)

Out[27]: array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
                0., 0., 0., 0., 0., 0., 0., 0.])
```

insert: 插入元素，insert(arr, obj, values, axis) 其中，arr 是指数组，obj 是指插入轴的对应下标，values 是插入的值，axis 指插入的轴是哪一个，axis = 0 插入行，axis = 1 插入列

```
In [28]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
         a = np.insert(a, 0, 0, axis = 0)
         a

Out[28]: array([[0, 0, 0],
                [1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

delete: 删除元素参数如下

* arr：数组 * obj：删除某一轴的对应标号 * axis：axis=0 删除行，axis=1 删除列

```
In [29]: a

Out[29]: array([[0, 0, 0],
                [1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [30]: np.delete(a, 0, axis = 1)
```

```
Out[30]: array([[0, 0],
                [2, 3],
                [5, 6],
                [8, 9]])
```

copy：创建一个新的，内存分离的拷贝，赋值与 copy 的区别在于一个是浅拷贝，一个是深拷贝，赋值操作，两个变量指向同一个空间；copy 操作，两个指向不同的空间

```
In [31]: a = np.array([1,2,3])
         b = a
         b is a

Out[31]: True

In [32]: b = a.copy()
         b is a

Out[32]: False

In [33]: b

Out[33]: array([1, 2, 3])

In [34]: a = np.array([1])
         b

Out[34]: array([1, 2, 3])
```

transpose：转置（可以直接.T）

```
In [35]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
         a.transpose()

Out[35]: array([[1, 4, 7],
                [2, 5, 8],
                [3, 6, 9]])

In [36]: a.T

Out[36]: array([[1, 4, 7],
                [2, 5, 8],
                [3, 6, 9]])
```

fill：用一个值填充数组

```
In [37]: a = np.array([1,2,3])
         a.fill(0)
         a
```

```
Out[37]: array([0, 0, 0])
```

flatten: 把多维数组转换为一维数组，注意每个元组的长度是相同的

```
In [38]: a = np.array([(1,2), (3,4), (5,6)])
         a.flatten()

Out[38]: array([1, 2, 3, 4, 5, 6])
```

newaxis: 增加维度

```
In [39]: a = np.array([1,2,3])
         a.shape

Out[39]: (3,)

In [40]: a = a[ :, np.newaxis]
         a

Out[40]: array([[1],
                [2],
                [3]])

In [41]: a.shape

Out[41]: (3, 1)
```

tile：重复指定数组

```
In [42]: a = np.array([(1,2), (3,4)])
         np.tile(a, (2,3))

Out[42]: array([[1, 2, 1, 2, 1, 2],
                [3, 4, 3, 4, 3, 4],
                [1, 2, 1, 2, 1, 2],
                [3, 4, 3, 4, 3, 4]])
```

0.10 NumPy 数组的数学操作

加减乘除，*星乘表示矩阵内各对应位置相乘，点乘表示求矩阵内积，二维数组称为矩阵积（mastrix product）：

```
In [43]: a = np.ones((2,2))
         b = np.array([(-1,1),(-1,1)])
         a
```

```
Out[43]: array([[1., 1.],
                [1., 1.]])

In [44]: b

Out[44]: array([[-1,  1],
                [-1,  1]])

In [45]: a + b

Out[45]: array([[0., 2.],
                [0., 2.]])

In [46]: a - b

Out[46]: array([[2., 0.],
                [2., 0.]])

In [47]: a * b

Out[47]: array([[-1.,  1.],
                [-1.,  1.]])

In [48]: a / b

Out[48]: array([[-1.,  1.],
                [-1.,  1.]])

In [49]: a.dot(b)

Out[49]: array([[-2.,  2.],
                [-2.,  2.]])
```

NumPy 中的常量

```
In [50]: np.e

Out[50]: 2.718281828459045

In [51]: np.pi

Out[51]: 3.141592653589793

In [52]: np.Inf

Out[52]: inf

In [53]: np.NaN
```

```
Out[53]: nan
```

sum: 求和，prod: 求积，数组的成员函数也可以用于 NumPy 的标准函数

```
In [54]: a = np.array([1,2,1])
         a.sum()

Out[54]: 4

In [55]: a.prod()

Out[55]: 2
```

mean,var,std,max,min: 平均数，方差，标准差，最大值，最小值

```
In [56]: a = np.array([5,3,1])
         a.mean()

Out[56]: 3.0

In [57]: a.var()

Out[57]: 2.6666666666666665

In [58]: a.std()

Out[58]: 1.632993161855452

In [59]: a.max()

Out[59]: 5

In [60]: a.min()

Out[60]: 1
```

argmax,argmin: 最大与最小值对应的索引值;
ceil, floor, rint: 取元素值上限，下限，四舍五入

```
In [61]: a = np.array([1.2, 3.8, 4.9])
         a.argmax()

Out[61]: 2

In [62]: a.argmin()

Out[62]: 0

In [63]: np.ceil(a)
```

```
Out[63]: array([2., 4., 5.])

In [64]: np.floor(a)

Out[64]: array([1., 3., 4.])

In [65]: np.rint(a)

Out[65]: array([1., 4., 5.])
```

unique: 去除数组中重复的值，clip: 把数组里的元素限定到指定范围，并且超出范围的最大值与最小值变为对应的上下限

```
In [66]: a = np.array([1,1,1,2,2,3,4,4,5,5,6,6,6])
         np.unique(a)

Out[66]: array([1, 2, 3, 4, 5, 6])

In [67]: a.clip(2,5)

Out[67]: array([2, 2, 2, 2, 2, 3, 4, 4, 5, 5, 5, 5, 5])
```

sort: 排序

```
In [68]: a = np.array([16,31,12,28,22,31,48])
         a.sort()
         a

Out[68]: array([12, 16, 22, 28, 31, 31, 48])
```

diagonal: 取对角元素

```
In [69]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
         a.diagonal()

Out[69]: array([1, 5, 9])
```

除了上述函数外，还有很多数据运算操作，它们的使用方式基本都类似，例如：abs, sign, sqrt, log, log10, exp, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arcsinh, arccosh,arctanh 等等。

0.11 NumPy 线性代数

有关线性代数的运算均在numpy.linalg 中; 矩阵(matrix) 是 array 的分支，matrix 和 array 在很多时候都是通用的，用哪一个都一样。官方建议大家如果两个可以通用，那就选择 array，因为 array 更灵活，速度更快。

array的优势就是不仅仅表示二维，还能表示 3、4、5...维，而且在大部分Python程序里，array也是更常用的。

dot: 矩阵乘法，对于两个一维的数组，计算的是这两个数组对应下标元素的乘积和 (数学上称之为内积)； 对于二维数组，计算的是两个数组的矩阵乘积；
对于多维数组，它的通用计算公式如下，即结果数组中的每个元素都是：数组 a 的最后一维上的所有元素与数组 b 的倒数第二位上的所有元素的乘积和：dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m]) diag: 返回矩阵对角线元素，前文我们已经演示

```
In [70]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
         np.diag(a)
```

```
Out[70]: array([1, 5, 9])
```

trace: 对角线元素和

```
In [71]: a = np.array([(1,2,3), (4,5,6), (7,8,9)])
         np.trace(a)
```

```
Out[71]: 15
```

det: 行列式，函数计算输入矩阵的行列式，行列式在线性代数中是非常有用的值，它从方阵的对角元素计算。
对于 2Œ2 矩阵，它是左上和右下元素的乘积与其他两个的乘积的差，换句话说，对于矩阵[[a，b]，[c，d]]，行列式计算为 ad-bc。

```
In [72]: a = np.array([(1,2), (3,4)])
         np.linalg.det(a)
```

```
Out[72]: -2.0000000000000004
```

下面还有一些函数可以使用，这里不一一列出来作用，当我们碰到具体问题时，去查询相应的具体知识点就可以，这里我们只需大概了解有这么多函数可以使用。
* eig: 特征值、特征向量 * inv: 逆 * qr:QR 分解 * svd: 奇异值分解* solve: 解线性方程Ax=b * lstsq: 计算 Ax=b 的最小二乘解

0.12 NumPy 随机数

所有有关线性代数的运算均在 np.random 中
* rand: 均匀分布的样本值 * randint: 给定上下限的随机整数 * randn: 标准正态分布 * binomial: 二项分布 * normal: 正态分布 * chisquare: 卡方分布 * gamma Gamm: 分布 * uniform:[0,1] 之间的均匀分布 * poisson: 泊松分布 * shuffle: 洗牌

0.13 文件操作

二进制文件：np.save、np.load 文本文件：np.loadtxt、np.savetxt* 二进制文件会存储为.npy 的格式

0.14 NumPy 广播机制

广播 (Broadcast) 是 numpy 对不同形状 (shape) 的数组进行数值计算的方式，对数组的算术运算通常在相应的元素上进行。如果两个数组a 和 b 形状相同，即满足a.shape == b.shape，那么 a*b 的结果就是 a 与 b 数组对应位相乘。这要求维数相同，且各维度的长度相同。

```
In [73]: a = np.array([1,2,3])
         b = np.array([4,5,6])
         a + b
```

```
Out[73]: array([5, 7, 9])
```

当运算中的 2 个数组的形状不同时，numpy 将自动触发广播机制

```
In [74]: a = np.array([(1,2), (2,2), (3,3), (4,4)])
         b = np.array([-1,1])
         a + b
```

```
Out[74]: array([[0, 3],
                [1, 3],
                [2, 4],
                [3, 5]])
```

Matplotlib 教程

0.1 简介

- Matplotlib 是 Python 下基础的 2d 绘图库（也可以绘制 3d，但是需要额外安装工具包），它的起源是模仿MATLAB 的图形命令，尽管看起来与 MATLAB 很相似但是它们并不相关，它可以绘制高质种图形，包含条形图、盒图、直方图、散点图、饼图等等。
- 在机器学习中，通常使用 Matplotlib 来展现数据，观察数据，从而分析出数据模型。在数据分析领域它有很知名的地位，而且具有丰富的拓展，能够实现强大的功能。
- 在 API 方面，Matplotlib 提供了一个名为 matplotlib.pyplot 的工具集，开发者可以需要几行代码就可以绘制精致的图形，关于 matplotlib.pyplot 的更详细的说明可以参见官方文档：https://matplotlib.org/api/pyplot_api.html

0.2 安装

- Matplotlib 是 python 库，因此我们建议使用 3.6 版本的Python。通过 Python 自带的包管理工具 pip 可以很轻松的安装 Matplotlib，安装时我们可以在 cmd 输入并执行下面的命令：
- pip install –i https://pypi.tuna.tsinghua.edu.cn/simple matplotlib
- -i 及后面的链接代表我们从清华的pypi 镜像下载安装。通常情况下在国内使用镜像安装能有更快的下载速度。

0.3 导入

我们通过 import 语句导入 matplotlib.pyplot：

```
In [1]: import matplotlib.pyplot as plt
        plt
```

```
Out[1]: <module 'matplotlib.pyplot' from 'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\matplotl
```

如果是在 juypter notebook 中，则可以使用下面的命令将绘制的图像嵌入到 notebook 里:

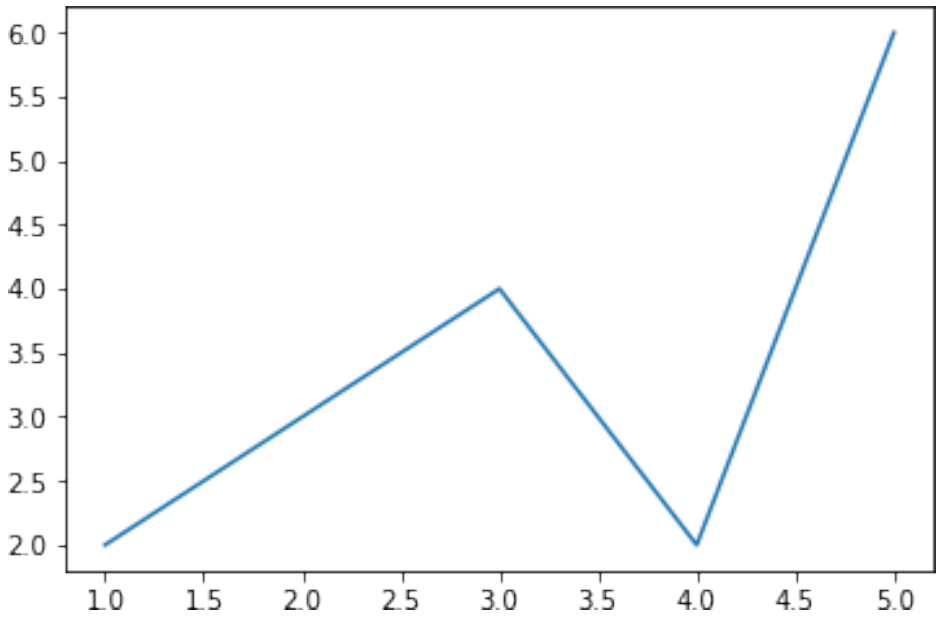
```
In [2]: %matplotlib inline
```

0.4 基本绘图

首先我们使用plot() 函数绘制一个折线图，在函数中我们需要传入 2 个数组，分别代表 x 轴与y 轴的取值，然后我们要使用 show() 函数来显示图像。

In [3]:
```
import matplotlib.pyplot as plt

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

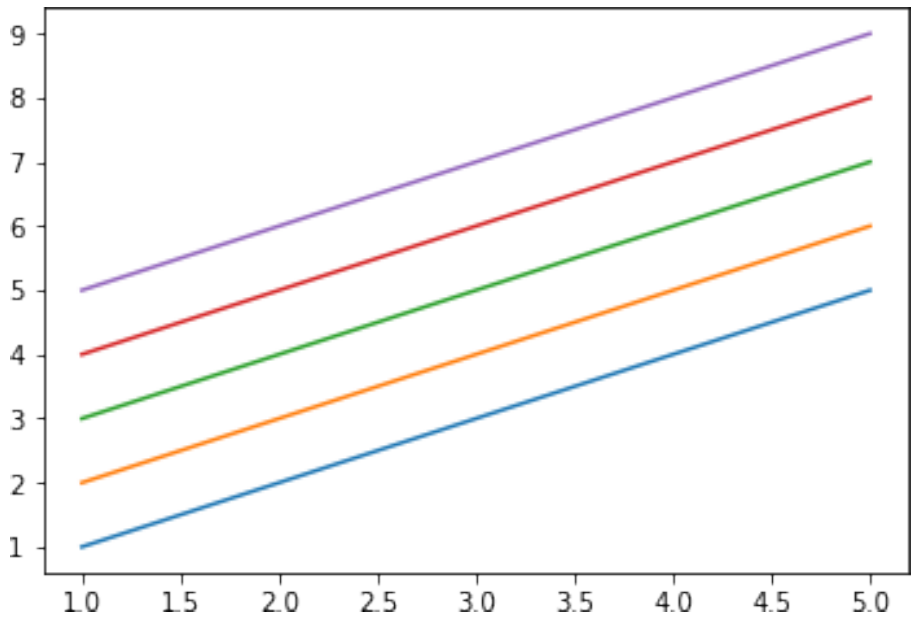
Out[3]: [<matplotlib.lines.Line2D at 0x7ed8978>]



0.5 绘制多个线条

我们也可以在一张图中绘制多个线条：

In [4]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(1,5,num = 5)
for i in range(5):
    plt.plot(x, x + i)
plt.show()
```

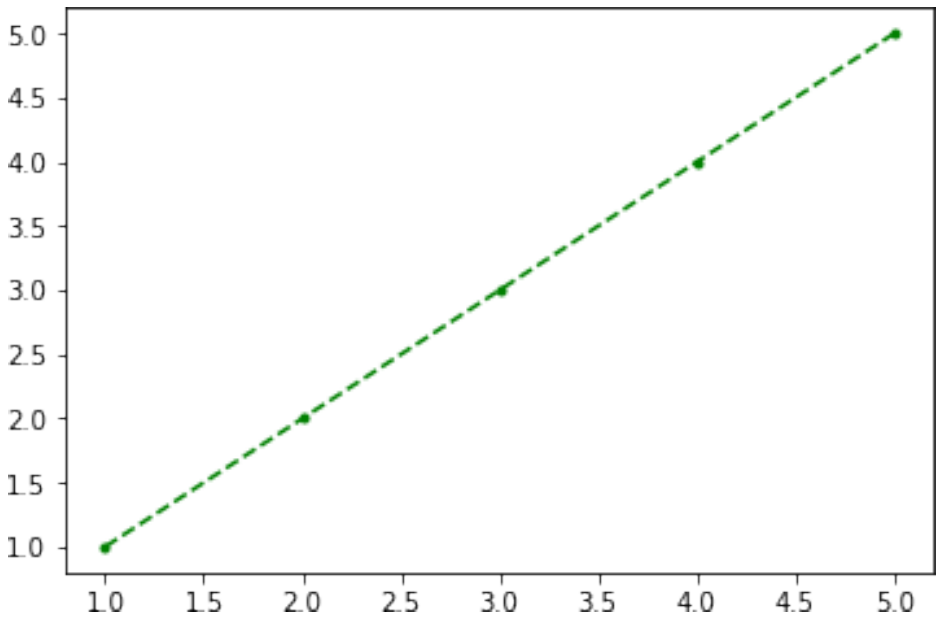


0.6 格式字符串

- 在 plot() 函数中前两个参数为数组或常量，那么第三个参数可以指定颜色标记与线条样式。
- 颜色指线条的颜色，标记是用指定的样式来标记数组中的点，线条样式则有实线、虚线、点线等等。
- 格式字符串由颜色，标记和线条的部分组成：fmt = '[color][marker][line]'
- 例如我们需要设置颜色为绿色（用 g 表示），标记为点（用. 表示），线条用虚线（用--表示），那么 fmt = 'g.--'
- 如果需要完整的格式字符串的列表，我们可以去参考官方文档：https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html

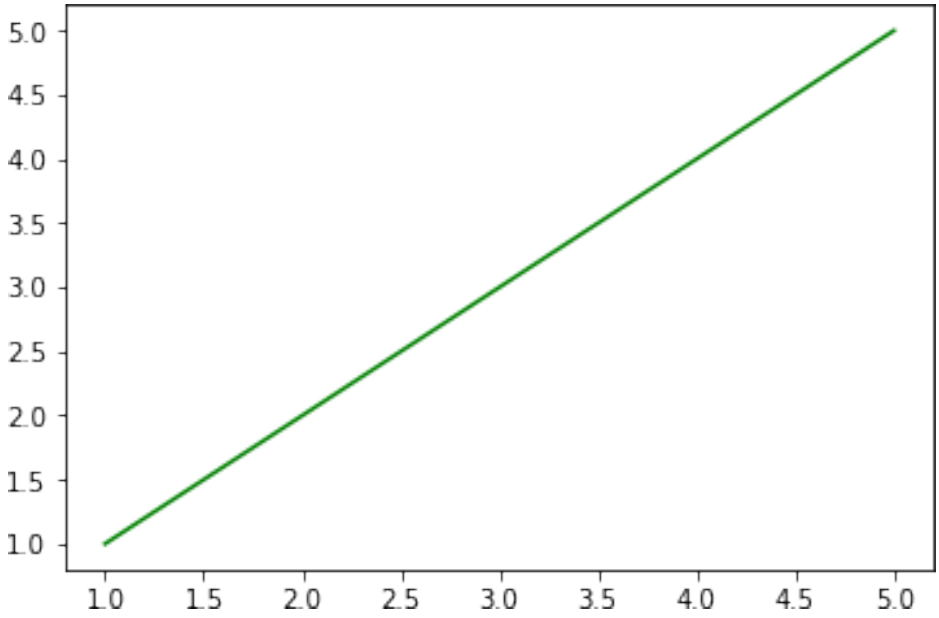
上述程序如下：

In [5]:
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5], [1,2,3,4,5], 'g.--')
plt.show()
```



当然我们可以省略格式字符串中某些项，那么映射到图上同样也省略了该项，不指定线类型时，默认是直线：

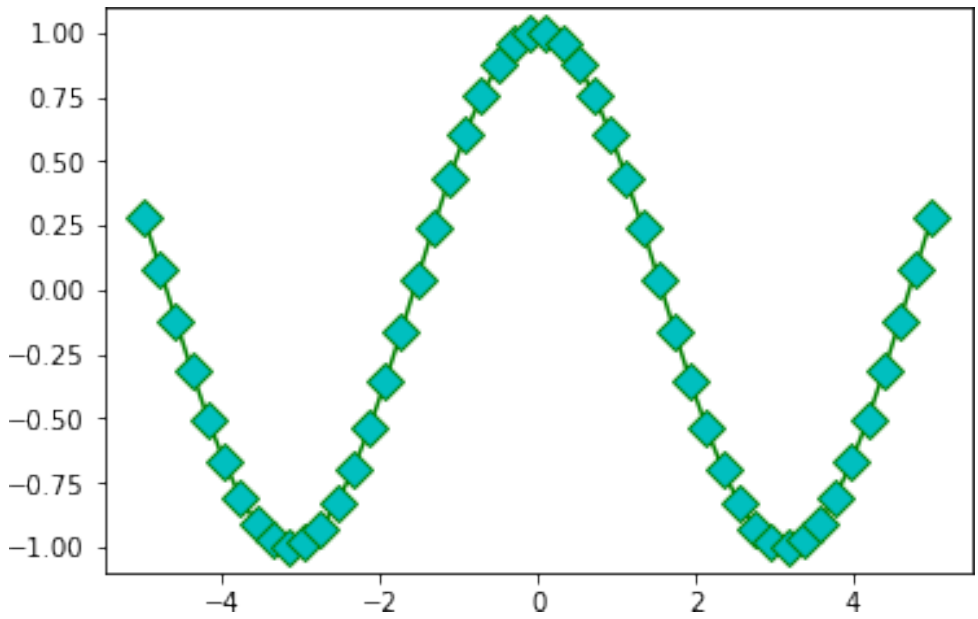
In [6]:
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5], [1,2,3,4,5], 'g')
plt.show()
```



0.7 标记可以单独指定

marker 指定标记的形状，markerfacecolor 指定标记颜色，marksize 指定标记大小:

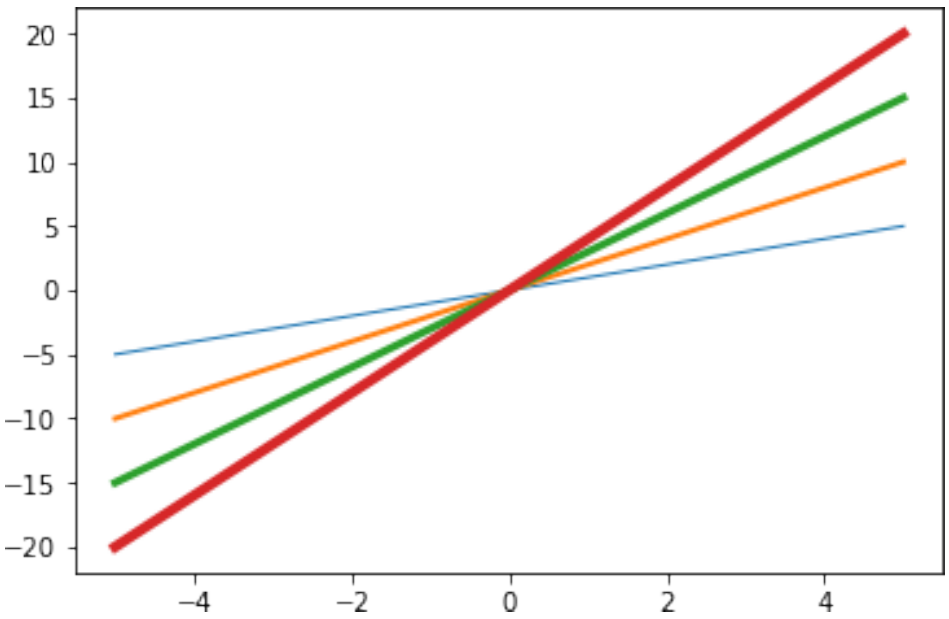
In [7]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5,5,50)
y = np.cos(x)
plt.plot(x, y, 'g', marker = 'D', markerfacecolor = 'c', markersize = 10)
plt.show()
```



0.8 线宽

通过 linewidth 参数调整线的宽度:

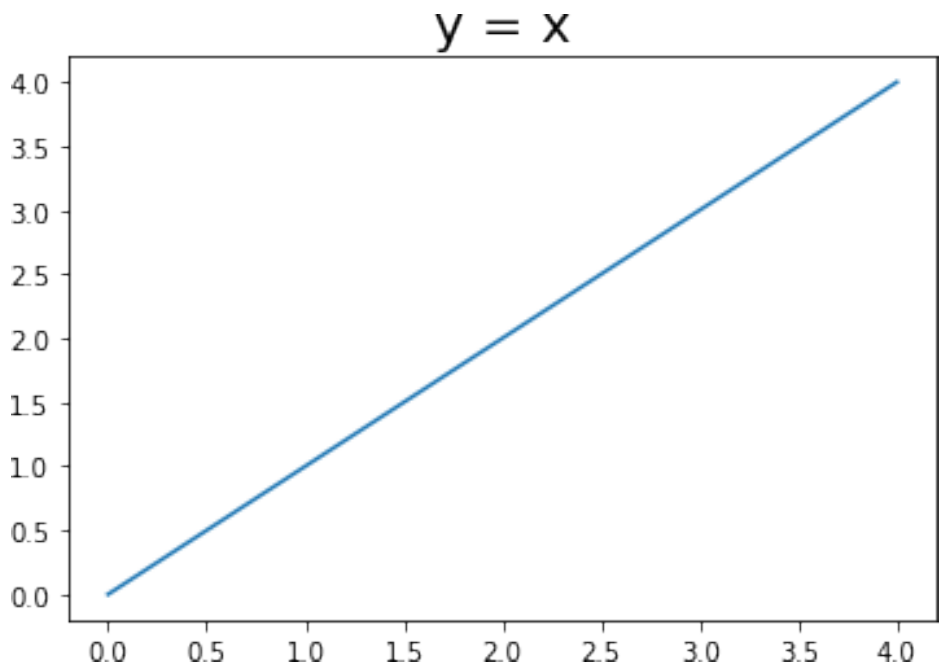
In [8]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5, 5)
for i in range(1,5):
    plt.plot(x, i*x, linewidth = i)
plt.show()
```



0.9 标题

通过plt.title() 函数给图片加上标题，fontsize 参数设置字体大小（中文显示，请尝试自行搜索解决方法）：

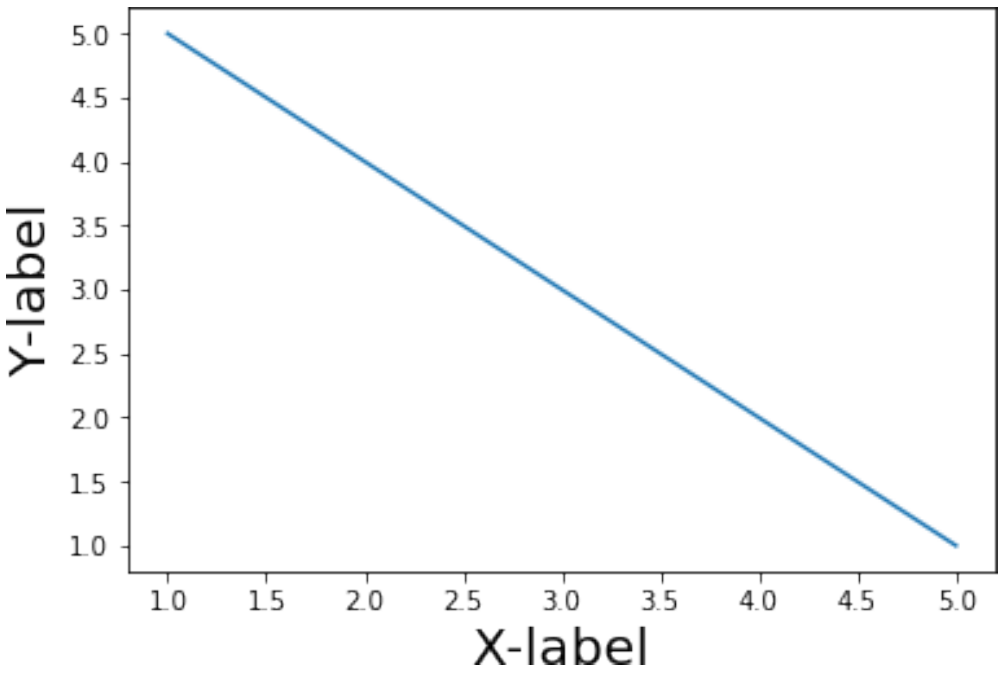
In [9]:
```
import matplotlib.pyplot as plt

x = range(5)
plt.plot(x,x)
plt.title('y = x',fontsize = 20)
plt.show()
```



0.10 轴标

通过 plt.xlabel() 和 plt.ylabel() 可以给横纵坐标轴加上标签，同样 fontsize 可以设置字体大小:

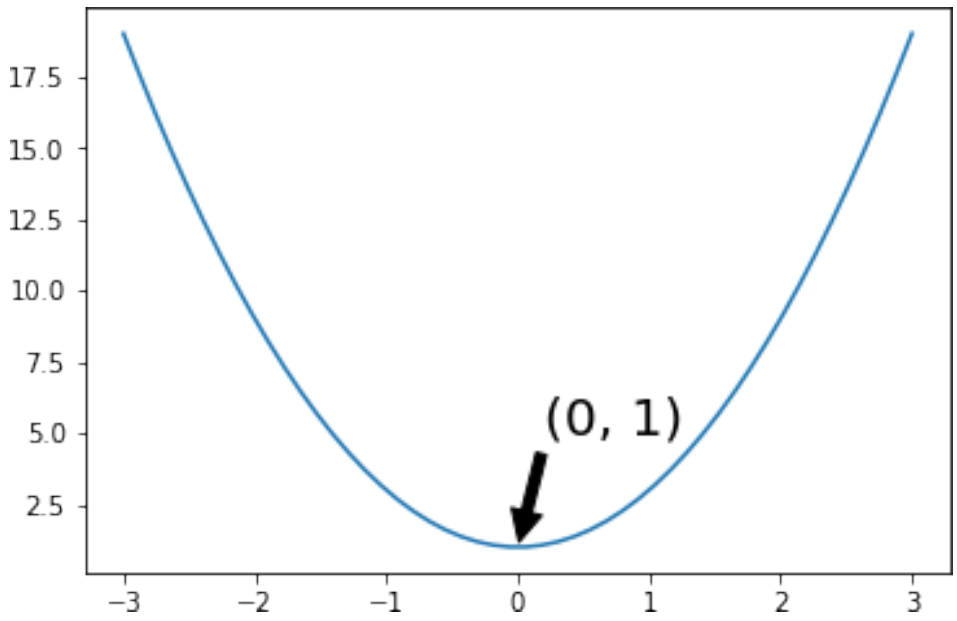
In [10]:
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5], [5,4,3,2,1])
plt.xlabel('X-label', fontsize = 20)
plt.ylabel('Y-label', fontsize = 20)
plt.show()
```



0.11 标注

通过 plt.annotate() 函数进行标注，参数 xy 与 xytext 表示标注的点与标注的文本所在的位置，arrowprops 设置标注箭头的属性，fontsize 设置字体大小:

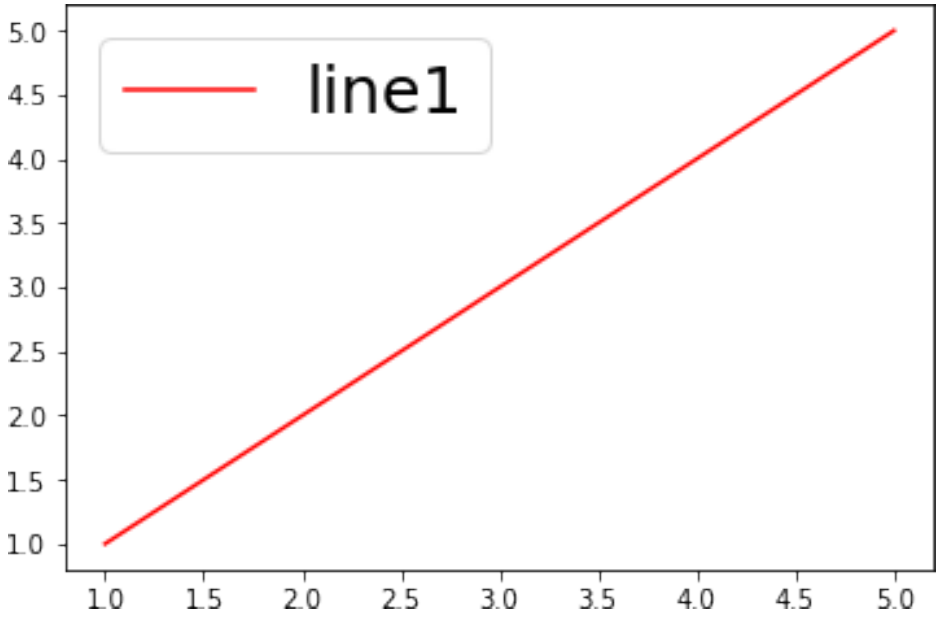
In [11]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-3, 3)
y = 2*x**2+1 plt.plot(x,
y)
plt.annotate('(0, 1)', xy = (0, 1), xytext = (0.2, 5),
             arrowprops = dict(facecolor = 'black', shrink = 0.05), fontsize = 20)
plt.show()
```



0.12 图例

在 Matplotlib 中，图例用来描述所绘制的对象，图例由若干个条目组成，每个条目又分为左边的标记和右边的文本：

```
In [12]: import matplotlib.pyplot as plt
         import numpy as np

         x = (1,2,3,4,5)
         y = x
         plt.plot(x, y, 'r', label="line1")
         plt.legend(loc = 'best',fontsize = 25)
         plt.show()
```

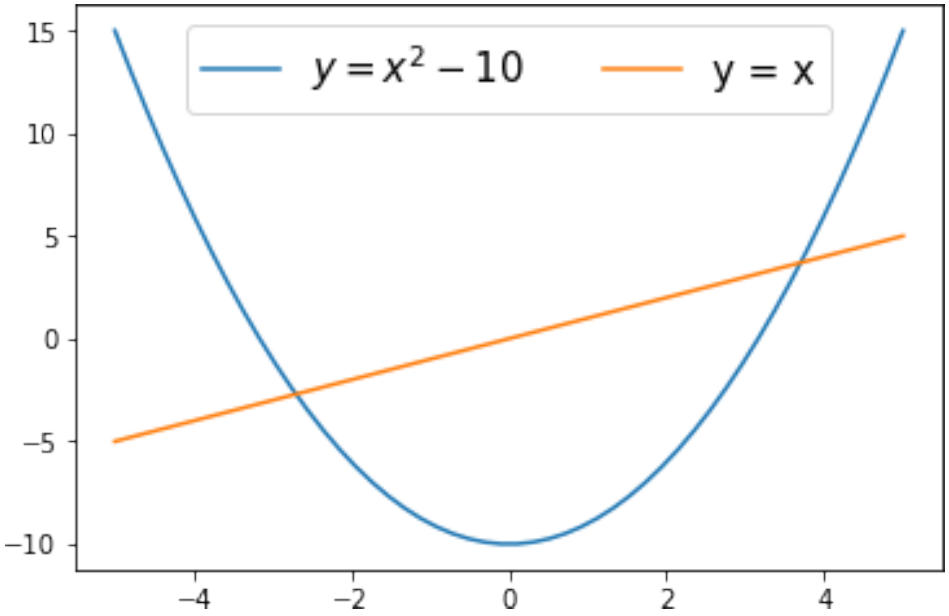


在 Matplotlib 中，通过 legend() 函数绘制图例，它有多种用法，最常用的做法是：

在调用 plt.plot() 时通过指定 label 参数给正在绘制的对象增加标签通过 plt.legend() 绘制图例，Matplotlib 会根据所有对象的特征和标签自动生成图例条目，进而生成整个图例 legend() 有很多参数，常用的参数有：

通过 loc 参数指定图例的位置，如果取值为 0 或 best，则会放置到最合适的位置, 取值 1 为右上角，2 为左上角通过ncol 定义图例有几列通过fontsize 指定图例中文本的大小更多lengend() 的用法，请参见官方文档 https://matplotlib.org/users/legend_guide.html

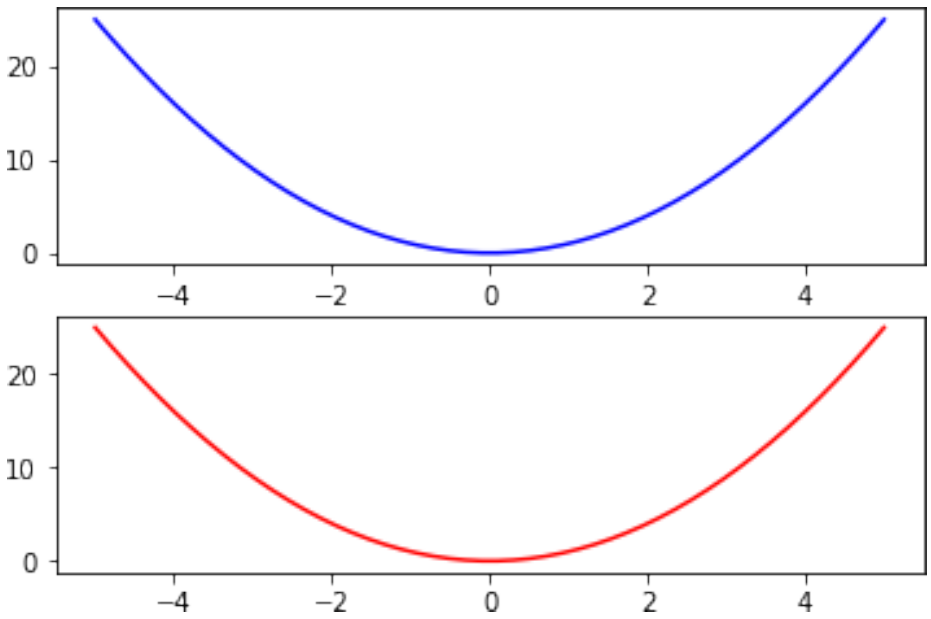
In [13]:
```
import matplotlib.pyplot as plt
import numpy as np
import   pandas as  pd

x = np.linspace(-5, 5)
plt.plot(x,x**2-10,label='$y = x^2-10$')# 标签内可以使用 LaTeX 语法
plt.plot(x,x,label='y = x')
plt.legend(loc=0,ncol=2,fontsize = 15)
plt.show()
```



0.13 子图

- 通过 matplotlib.pyplot.subplot() 创建一个子图
- 参数nrows, ncols, index 表示将整个绘图区视为nrows x ncols 个区域，当前要绘制到第 index 个区域
- 三个参数可以写在一起，比如 221 表示 2x2 个区域中的第 1 个例如绘制两行一列的图组

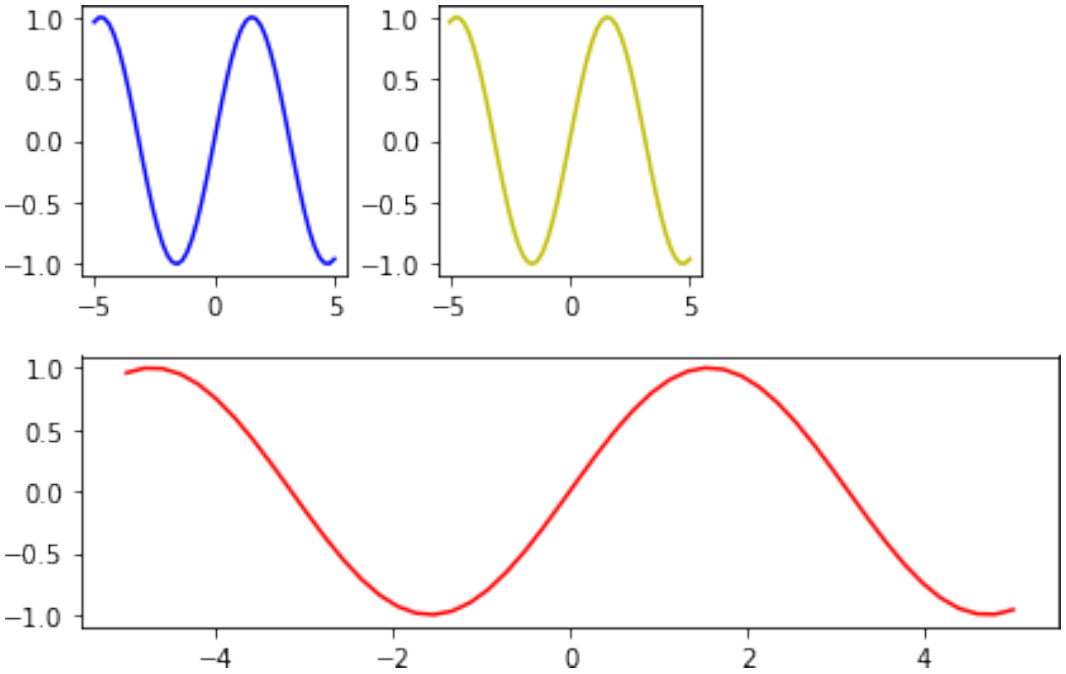
In [14]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-5, 5)
y = x**2

# 将整个画布视为 2x1 的区域，绘制第 1 个区域
plt.subplot(211)
plt.plot(x, y, 'b')

# 将整个画布视为 2x1 的区域，绘制第 2 个区域
plt.subplot(212)
plt.plot(x, y, 'r')

plt.show()
```



下面是比较复杂的实例：

```
In [15]: import matplotlib.pyplot as plt
         import numpy as np

         x = np.linspace(-5, 5)
         y = np.sin(x)

         # 将整个画布视为 2x3 的区域，绘制第一个区域
         plt.subplot(231)
         plt.plot(x, y, 'b')

         # 将整个画布视为 2x3 的区域，绘制第三个区域
         plt.subplot(232)
         plt.plot(x, y, 'y')

         # 将整个画布视为 2x1 的区域，绘制第二个区域
         plt.subplot(212)
         plt.plot(x, y, 'r')

         plt.tight_layout()      # 调整子图的间距
```

```
plt.show()
```

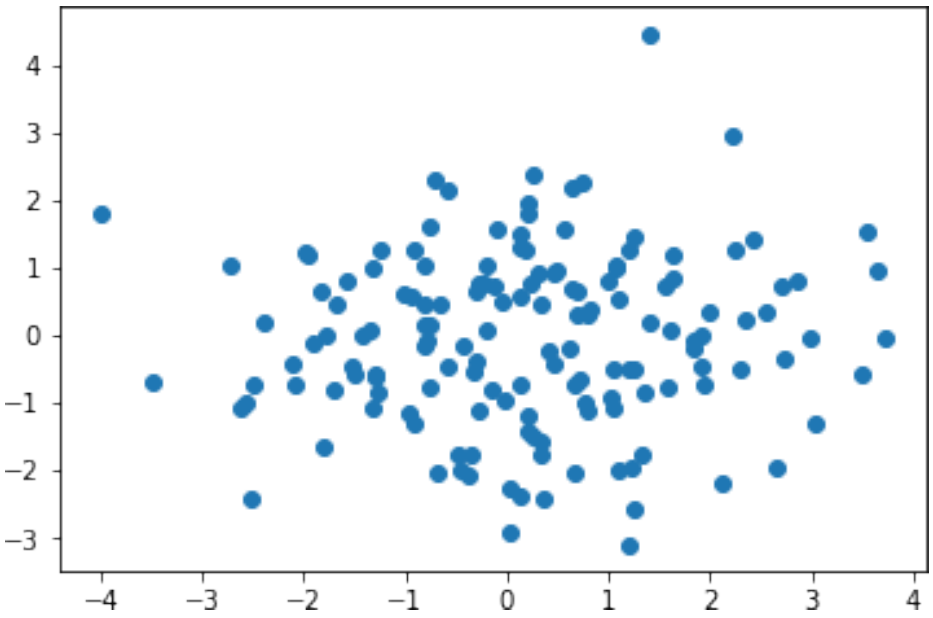


0.14 散点图

- scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None, vmin=None, vmax=None, alpha=None, linewidths=None, verts=None, edgecolors=None, hold=None, data=None, **kwargs)
- 常用的参数与基本绘图的使用方法相同

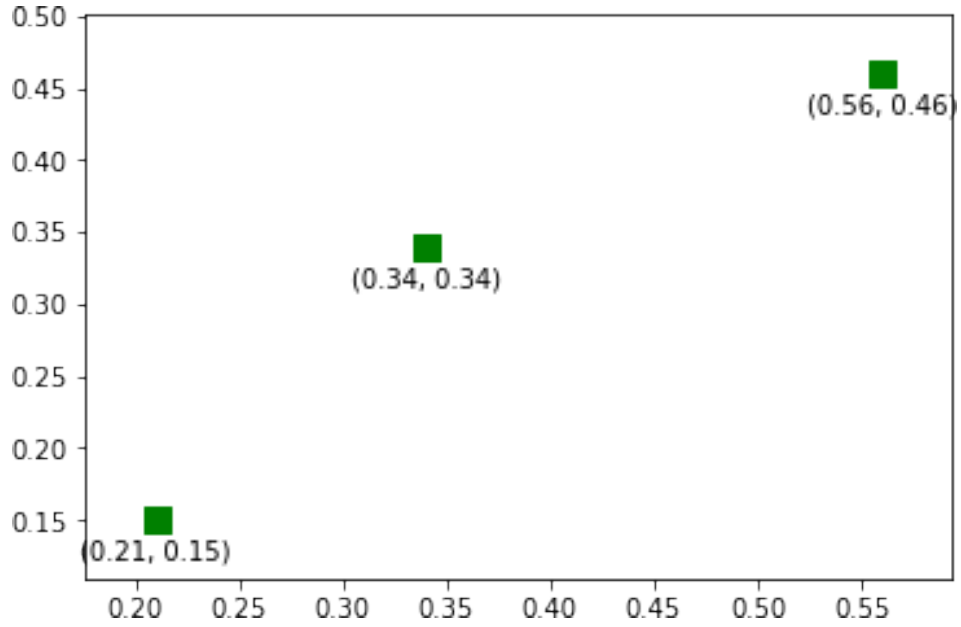
In [16]:
```
import matplotlib.pyplot as plt
import numpy as np

# 这里使用 multivariate_normal 方法多元正态分布矩阵，
# np.random.multivariate_normal 方法用于根据实际情况生成一个多元正态分布矩阵
# multivariate_normal(mean, cov, size=None, check_valid=None, tol=None)
# mean：mean 是多维分布的均值维度为 1；
# cov：协方差矩阵，注意：协方差矩阵必须是对称的且需为半正定矩阵；
# size：指定生成的正态分布矩阵的维度（例：若 size=(1, 1, 2)，则输出的矩阵的-#
shape 即形状为 1X1X2XN（N 为 mean 的长度））；
# check_valid：这个参数用于决定当 cov 即协方差矩阵不是半正定矩阵时程序的处理-
# 方式，它一共有三个值：warn，raise 以及 ignore。当使用 warn 作为传入的参数时，
```

```
# 如果 cov 不是半正定的程序会输出警告但仍旧会得到结果；当使用 raise 作为传入的-
# 参数时，如果 cov 不是半正定的程序会报错且不会计算出结果；
# 当使用 ignore 时忽略这个问题即无论 cov 是否为半正定的都会计算出结果；
# tol：检查协方差矩阵奇异值时的公差，float 类型；
x = np.random.multivariate_normal([0, 0], [[2, 0], [0, 2]], 150)
plt.scatter(x[:, 0], x[ :, 1 ])# 绘制散点图
plt.show()
```



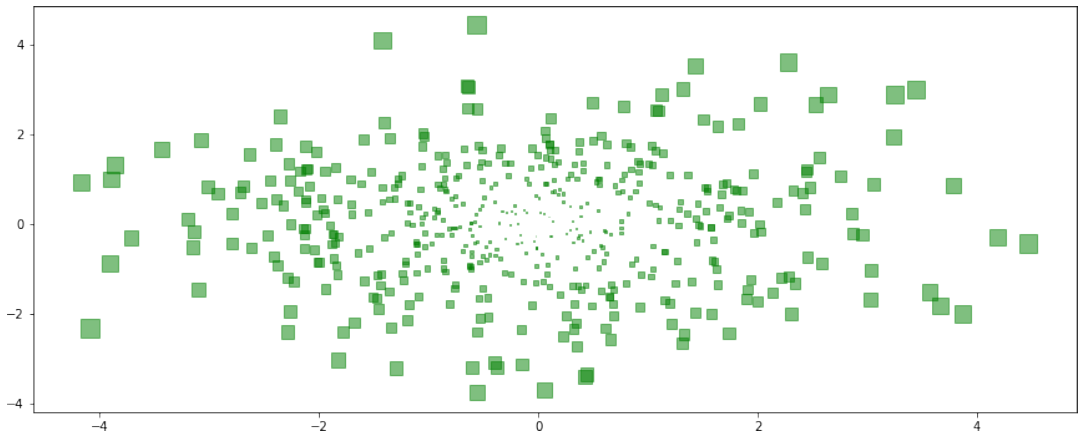
0.15 点的标注

In [17]:
```
import matplotlib.pyplot as plt

xs = [0.21, 0.56, 0.34]
ys = [0.15, 0.46, 0.34]
plt.scatter(xs, ys, marker = 's', color = 'g', s = 100)
for x,y in zip(xs, ys):
    plt.annotate('(%s, %s)'%(x, y), xy = (x, y), xytext = (0, -15),
                 textcoords = 'offset points', ha = 'center', fontsize = 10)
plt.show()
```



0.16 点的大小

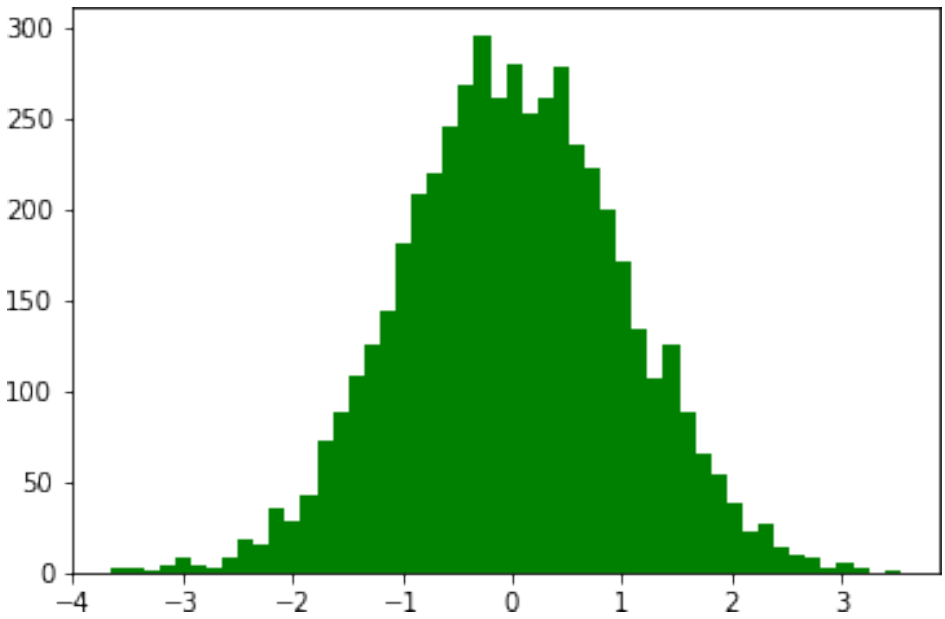
In [18]:
```
import matplotlib.pyplot as plt
import numpy as np

# 这里使用 multivariate_normal 方法多元正态分布矩阵
x = np.random.multivariate_normal([0, 0], [[2, 0], [0, 2]], 500)
fig = plt.figure(figsize=(15,6))

R = x**2
R_sum = R.sum(axis = 1)
#s 定义点的 size，可以取标量或者数组，alpha 指点的透明度
plt.scatter(x[:,0], x[:,1],  s=10*R_sum, color='g', marker='s', alpha=0.5)
plt.show()
```



0.17 直方图

- plt.hist(arr, bins=10, normed=0, facecolor='black', edgecolor='black',alpha=1，histtype='bar')
- arr: 需要计算直方图的一维数组
- bins: 直方图的柱数，可选项，默认为 10
- normed: 是否将得到的直方图向量归一化。默认为 0
- facecolor: 直方图颜色
- edgecolor: 直方图边框颜色
- alpha: 透明度
- histtype: 直方图类型，'bar', 'barstacked', 'step', 'stepfilled'

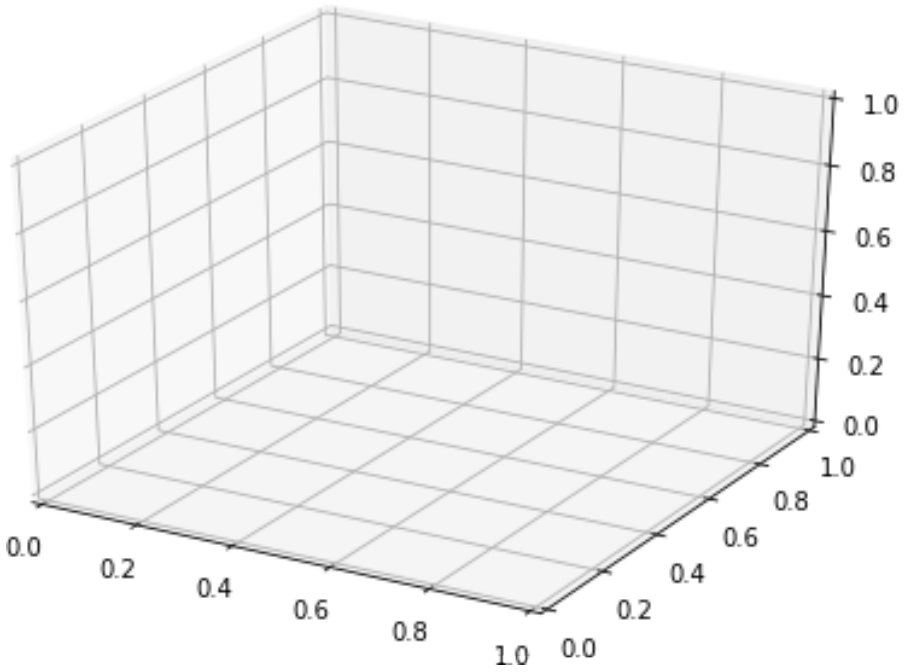
In [19]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(size=5000)

# 绘制直方图，通过图形的长相，就可以快速的判断数据是否近似服从正态分布
plt.hist(x, bins=50, facecolor = 'green')
plt.show()
```



0.18 3D 图

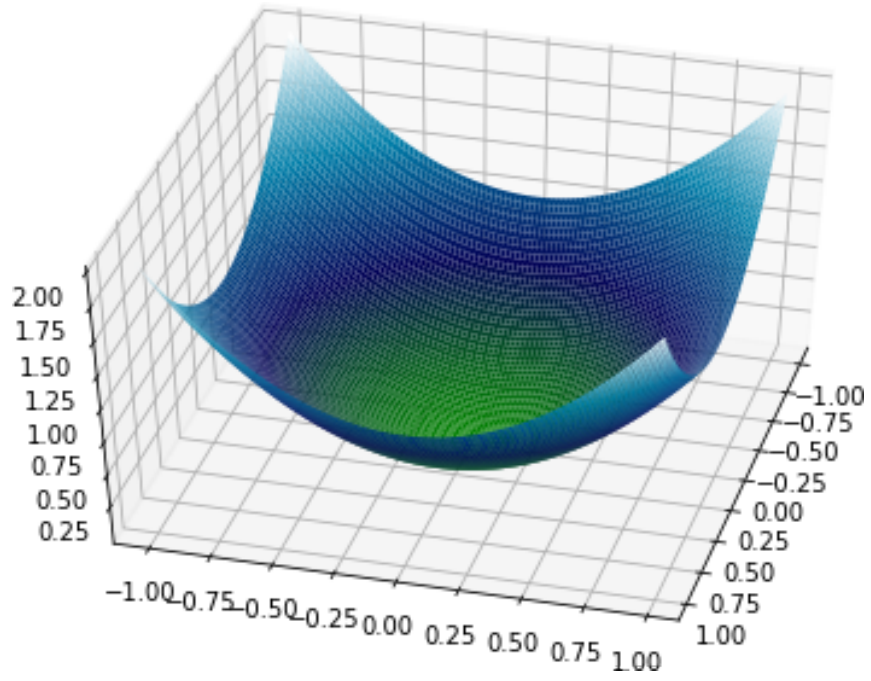
- 使用 matplotlib 绘制 3D 图形，需要通过创建一个新的 axes 对象——Axes3D 来实现，与二维图像不同的是，绘制三维图像主要通过mplot3d 模块实现。但是，使用Matplotlib 绘制三维图像实际上是在二维画布上展示，所以一般绘制三维图像时，同样需要载入pyplot 模块。如果需要更详细的了解 mplot3d 可以访问官方文档。

In [20]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d  import Axes3D # 绘制 3D 坐标的函数

# 创建一个绘图对象
fig1=plt.figure()
# 用这个绘图对象创建一个 Axes 对象（有 3D 坐标）
ax=Axes3D(fig1)
```



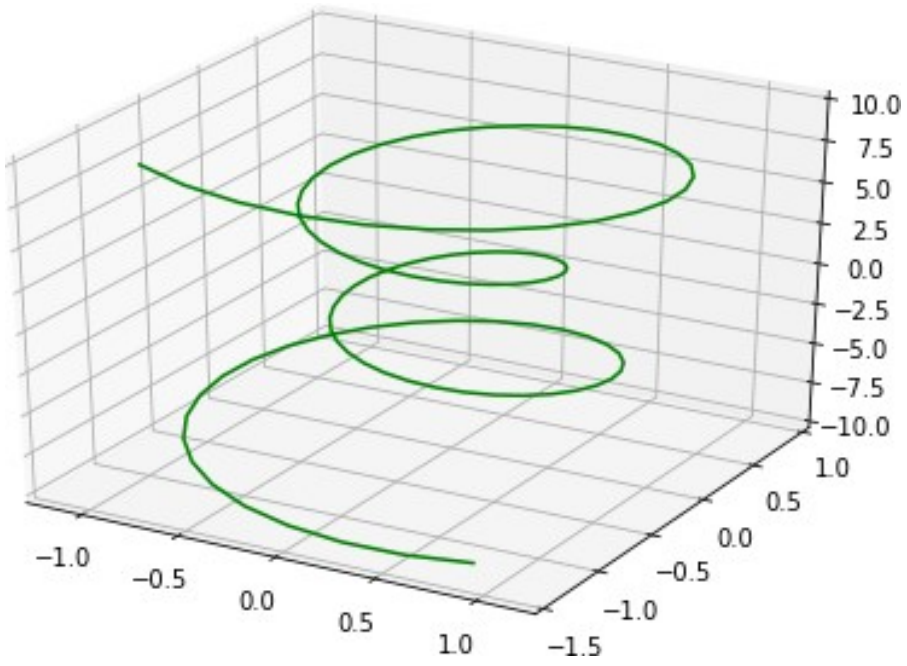
In [21]:
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

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

x = np.arange(-1, 1, 0.02)
y = np.arange(-1, 1, 0.02)
# meshgrid 函数将两个输入的数组 x 和 y 进行扩展，前一个的扩展与后一个有关，
# 后一个的扩展与前一个有关，前一个是竖向扩展，后一个是横向扩展。
X,Y = np.meshgrid(x,y)


Z = X**2 + Y**2
# cmap 为色彩映射，可选值参见官方文档
ax.plot_surface(X, Y, Z, rstride=1, cstride=1, cmap='ocean')
# 第一个参数为仰角，第二个为方位角
ax.view_init(45, 15)
```

```
plt.show()
```



0.19 3D 曲线图

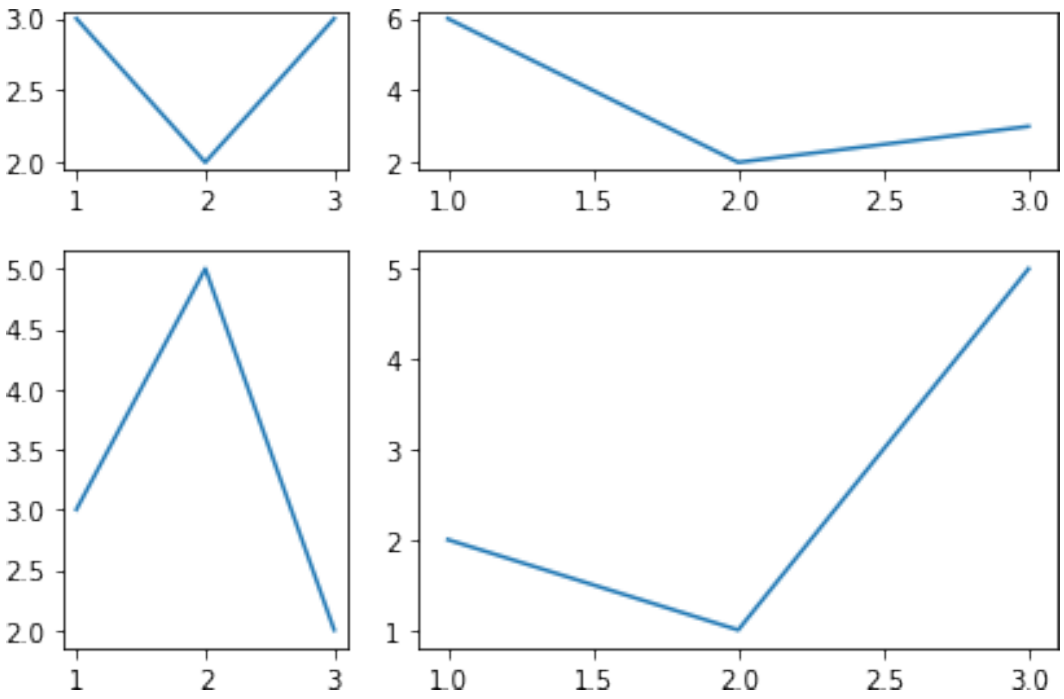
In [22]:
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

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

c = np.linspace(-1,1,100)**2 + 0.5
x = c*np.sin(np.linspace(-10, 10, 100))
y = c*np.cos(np.linspace(-10, 10, 100))
z = np.linspace(-10, 10, 100)
# 给定 3 个坐标数组
ax.plot(x,y,z,color = 'g')
plt.show()
```



0.20 子图与布局

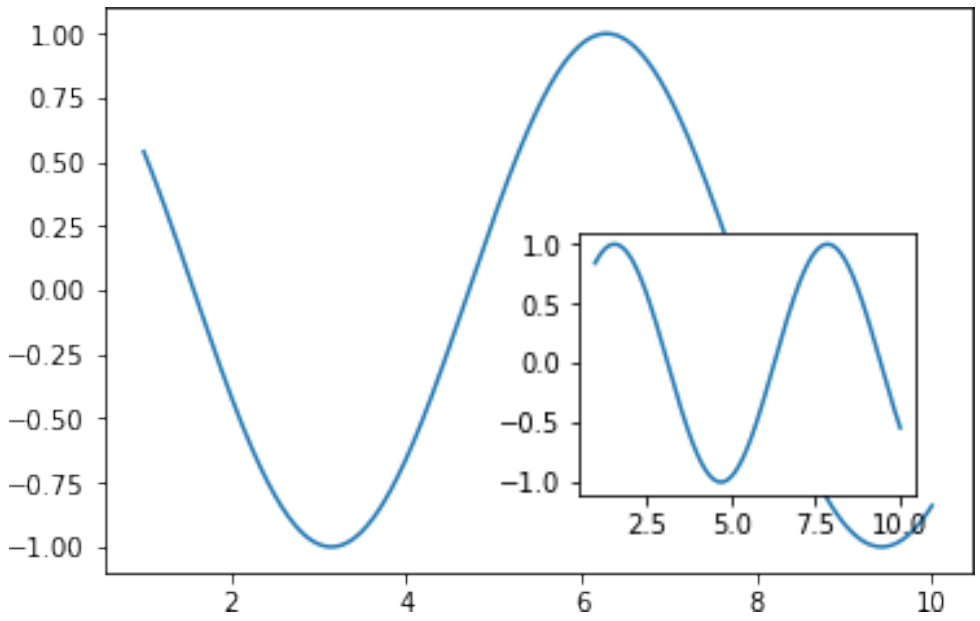
In [23]: **import matplotlib.pyplot as plt**

```
# 第一个元组参数代表将画布分为 3 行 3 列，第二个元组代表在第一行第一列开始绘制。
# 第一个参数从 1 开始，第二个从 0 开始取值
ax1 = plt.subplot2grid((3, 3), (0, 0))
#colspan 代表占几行，rowspan 代表占几列
ax2 = plt.subplot2grid((3, 3), (0, 1), colspan=2)
ax3 = plt.subplot2grid((3, 3), (1, 0), rowspan=2)
ax4 = plt.subplot2grid((3, 3), (1, 1), rowspan=2, colspan=2)

ax1.plot([1, 2, 3], [3, 2, 3], label = 'ax1')
ax2.plot([1, 2, 3], [6, 2, 3], label = 'ax2')
ax3.plot([1, 2, 3], [3, 5, 2], label = 'ax3')
ax4.plot([1, 2, 3], [2, 1, 5], label = 'ax4')

plt.tight_layout()      # 调整子图间距
plt.show()
```



0.21 子图

In [24]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(1,10,1000)
y1 = np.cos(x)
y2 = np.sin(x)

fig, ax1 = plt.subplots()

left,bottom,width,height = [0.55, 0.23, 0.3, 0.35]
ax2 = fig.add_axes([left,bottom,width,height])

ax1.plot(x,y1)
ax2.plot(x,y2)
plt.show()
```



0.22 结尾

- 本文档是对机器学习领域中常用的绘图进行总结，在实际应用中我们应当了解常用的方法，并且针对具体案例时再具体查找官方文档等相关的资料，达到融会贯通的工作状态，工具是其次，解决问题才是你的主要目的。