In [1]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
# 解决坐标轴刻度负号乱码
plt.rcParams['axes.unicode_minus'] = False

# 解决中文乱码问题
plt.rcParams['font.sans-serif'] = ['Simhei']
```

1 plt.plot绘图进阶

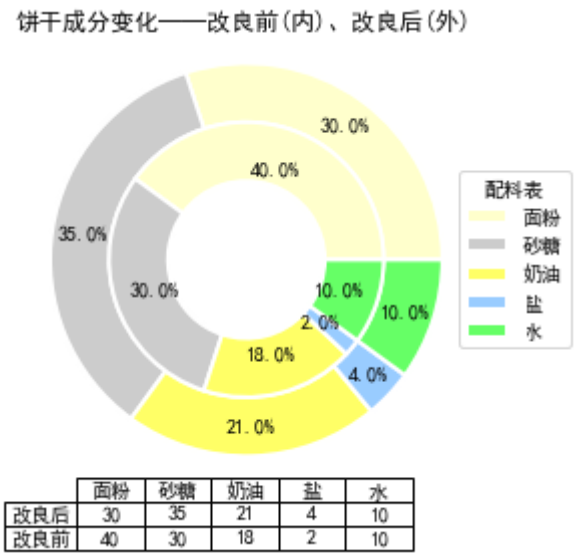
1.1 添加表格——table()

有时候为了更加全面地凸显数据的规律和特点，需要将统计图形和数据表格结合使用。

- cellText：表格的数值。
- cellLoc：表格中的数据对齐位置（center,left,right）。
- colWidths:表格每列的宽度。
- colLabels：表格每列的列名。
- colColours：表格每列的列名称所在单元格的颜色。
- rowLabels：表格每行的行名称。
- rowLoc:表格每行的行名称的对齐位置（center,left,right）。
- loc：表格在画布中的位置。

In [2]:

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

#数据集，x1,x2分别对应外部、内部百分比例
outer=[30,35,21,4,10]
inner=[40,30,18,2,10]

#设置饼状图各个区块的颜色
color=['#FFFFCC','#CCCCCC','#FFFF66','#99CCFF','#66FF66']

plt.pie(outer,autopct='%3.1f%%',radius=1,pctdistance=0.85,colors=color,wedgeprops=dict(linew
plt.pie(inner,autopct='%3.1f%%',radius=0.7,pctdistance=0.7,colors=color,wedgeprops=dict(line

#图例
legend_text=['面粉','砂糖','奶油',"盐",'水']
plt.legend(legend_text,title='配料表',loc='center right')
plt.axis('equal')
plt.title("饼干成分变化——改良前(内)、改良后(外)")

#添加表格
cellText=[outer,inner]
cellLoc="center"

colWidths=[0.1]*5
colLabels=legend_text

rowLabels=["改良后","改良前"]
rowLoc="center"
loc="bottom"

plt.table(cellText=cellText,cellLoc=cellLoc,colWidths=colWidths,colLabels=colLabels,rowLabel

plt.show();
```



	面粉	砂糖	奶油	盐	水
改良后	30	35	21	4	10
改良前	40	30	18	2	10

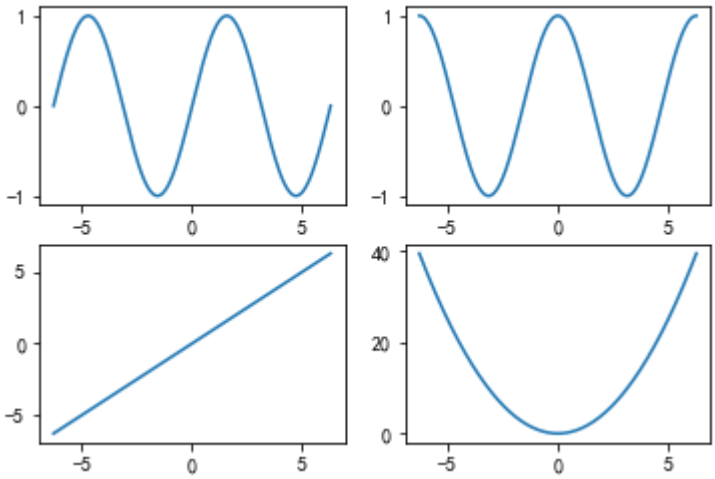
添加了表格之后，我们就可以从绝对和相对的角度比较饼干成分的变化。

1.2 子图plt.subplot()

- subplot(C,R,P)：划分C行R列，从最左上角往右数起，序号P依次增加，直到换一行，序号也是从左往右的增加。

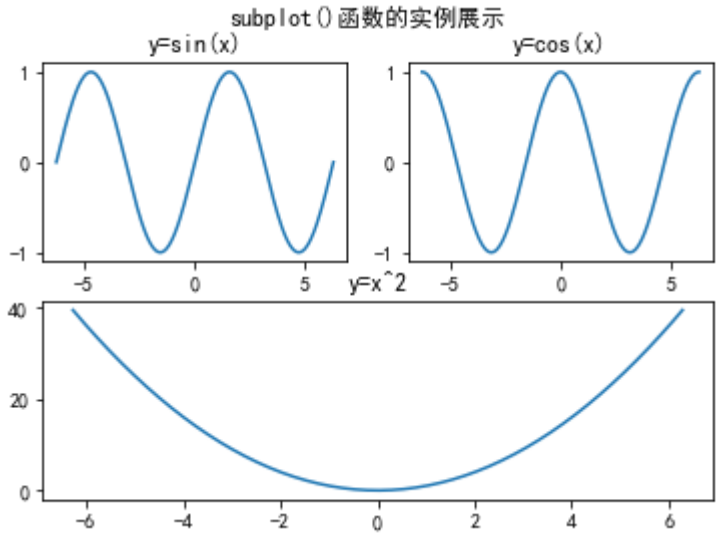
In [5]:

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

x=np.linspace(-2*np.pi,2*np.pi,200)
y_01=np.sin(x)
y_02=np.cos(x)
y_03=x
y_04=x**2

plt.subplot(221)
plt.plot(x,y_01,label="sin(x)")

plt.subplot(222)
plt.plot(x,y_02,label="cos(x)")

plt.subplot(223)
plt.plot(x,y_03,label="tan(x)");

plt.subplot(224)
plt.plot(x,y_04,label="x**2")

plt.show();
```



In [33]:

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

x=np.linspace(-2*np.pi,2*np.pi,200)
y_01=np.sin(x)
y_02=np.cos(x)
y_03=x**2


plt.subplot(221)
plt.plot(x,y_01)
plt.title("y=sin(x)")

plt.subplot(222)
plt.plot(x,y_02)
plt.title("y=cos(x)")

plt.subplot(212)    #两行一列，占其第二行
plt.plot(x,y_03);
plt.title("y=x^2")

plt.suptitle("subplot()函数的实例展示")
# plt.subplot(224)
# plt.plot(x,y_04,label="x**2");

plt.show();
```

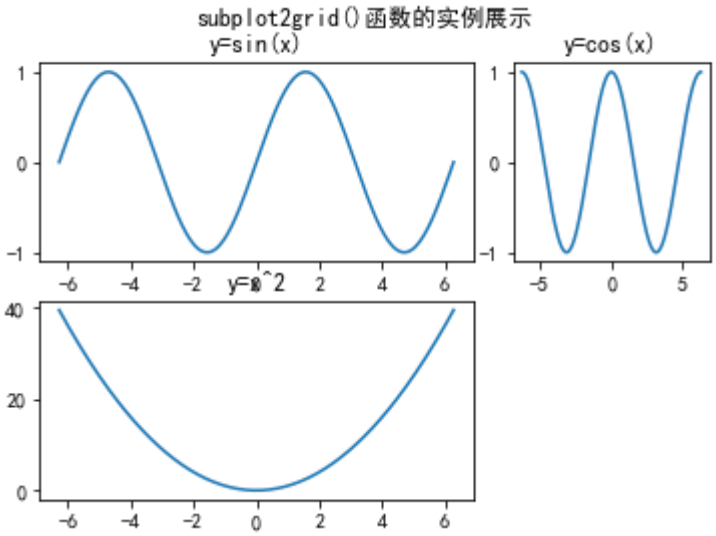


1.3 网格布局子图——subplot2grid()

- subplot2grid(shape,loc,colspan,rowspan)
  - shape如果是(2,3)的话，就是设置了一个2行3列的网格布局
  - loc表示元组的第一个和第二个数值的起点，如果loc为(0,1)就以为这图形从第1行第2列的位置开始绘制。
  - rowspan表示图形横跨多少行。
  - colspan表示图形横跨多少列。

In [10]:

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

x=np.linspace(-2*np.pi,2*np.pi,200)
y_01=np.sin(x)
y_02=np.cos(x)
y_03=x**2

plt.subplot2grid((2,3),(0,0),colspan=2)     #在一个2行3列的网格，从第1行第1列开始绘制，横跨2列
plt.plot(x,y_01)
plt.title("y=sin(x)")

plt.subplot2grid((2,3),(0,2))               #在一个2行3列的网格，从第1行第3列开始绘制，横跨2列
plt.plot(x,y_02)
plt.title("y=cos(x)")


plt.subplot2grid((2,3),(1,0),colspan=2)     #在一个2行3列的网格，从第2行第1列开始绘制，横跨3列
plt.plot(x,y_03);
plt.title("y=x^2")


plt.suptitle("subplot2grid()函数的实例展示")

plt.show();
```



2 面向对象接口绘图

In [10]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
type(plt.plot)
```

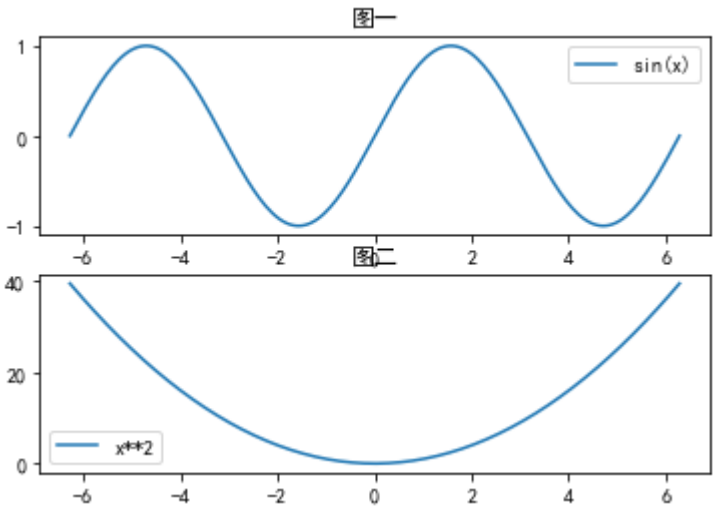
Out[10]: function

2.1 为什么要这样画

我们上面一直通过matplotlib.pyplot模块中的plot()函数的反复调用来进行画图，其实有时候需要对图形进一步自定义，往往需要通过面向对象接口，两种绘图方式有什么不同呢？我们首先看看之前的画图方式：

In [16]:

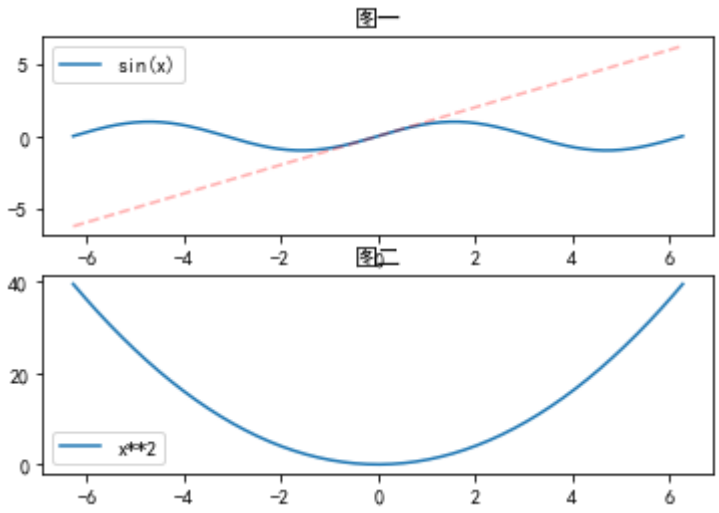
```
import matplotlib as mpl
import matplotlib.pyplot as plt

x=np.linspace(-2*np.pi,2*np.pi,200)
y_01=np.sin(x)
y_02=x**2

plt.subplot(211)
plt.plot(x,y_01,label="sin(x)")
plt.legend()
plt.title("图一")

plt.subplot(212)
plt.plot(x,y_02,label="x**2")
plt.legend()
plt.title("图二")

plt.show();
```



- 我们想一下，上面的代码运行到最后一行的时候，如果此时想在代码后续再画一条折线图，添加到“图一”中，实现起来就比较麻烦。因为此时两幅子图已经画完。那如何让Python再回到“图一”中去绘图呢？
- 另外一个问题就是，如果我想对图形的细节作进一步优化，比如上面图二的图标题，就和图一的横轴标签重叠了，有办法对这些小的细节作优化吗？
- 此时我们就需要引入另外一种绘图方式：通过面向对象绘图。我们首先阅读以下代码，对比上面的绘图方式，看看这种绘图方式的特点：

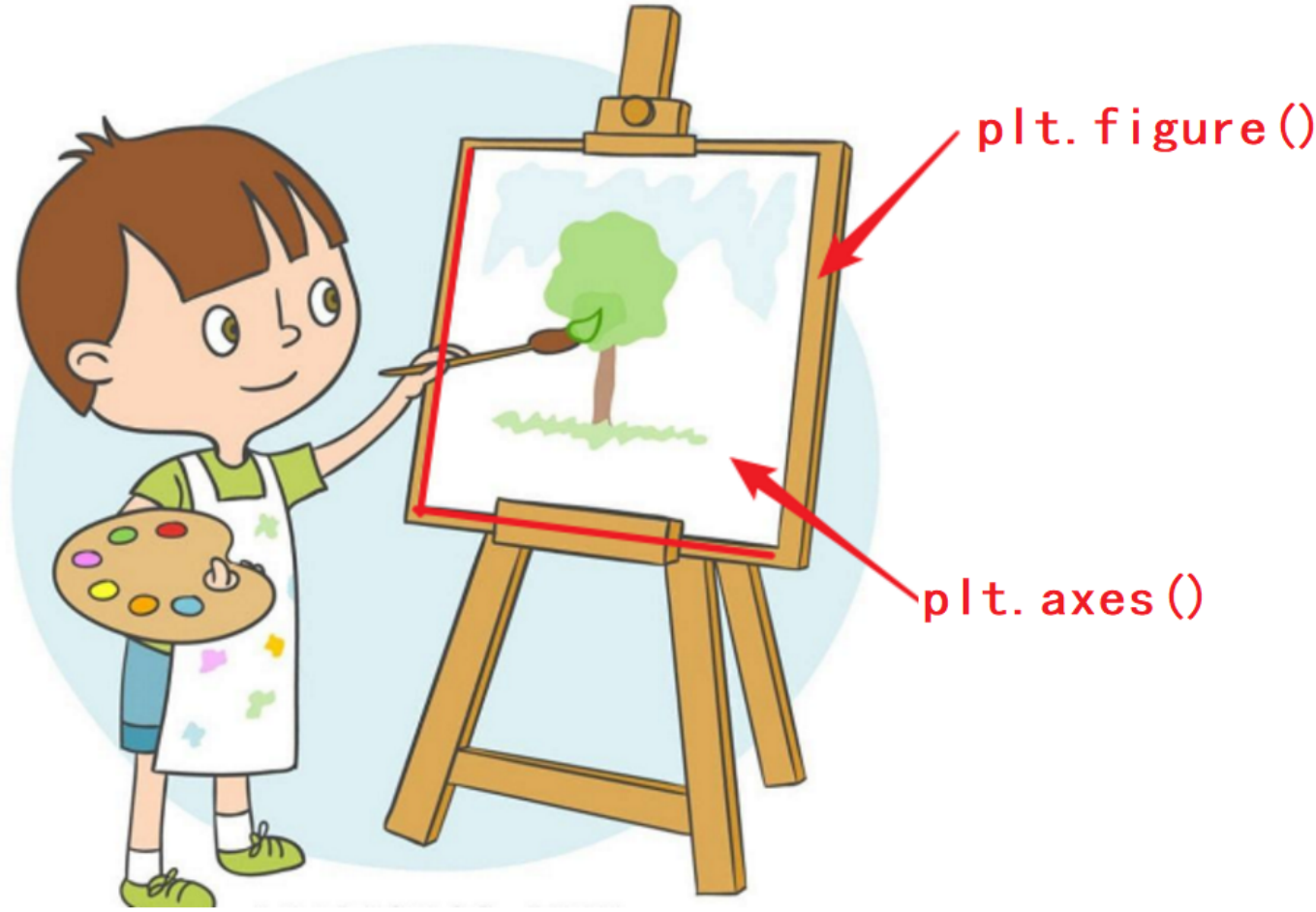
In [11]:

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

fig,ax=plt.subplots(2,1)

x=np.linspace(-2*np.pi,2*np.pi,200)
y_01=np.sin(x)
y_02=x**2


ax[0].plot(x,y_01,label="sin(x)")
ax[0].legend()
ax[0].set_title("图一")


ax[1].plot(x,y_02,label="x**2")
ax[1].legend()
ax[1].set_title("图二")

#如果在画完两幅子图之后，在这个时候想再往图一添加折线图y=x，可以直接通过实例ax[0].plot()来画
ax[0].plot(x,x,color="r",ls="--",alpha=0.3)

plt.show();
```



首先要说明的是fig,ax=plt.subplots(2,1)是什么意思：

- plt.subplots()其实会返回两个对象，一个是画布实例figure，一个是坐标轴实例axes:
  - figure可以看做是一个图形实例，用来包含坐标轴、图形、文字标签等。
  - axes表示一个坐标轴实例，是一个带有刻度和标签的矩阵。
  - 先有了figure、axes，就可以使用ax.plot来绘图了
- 因为ax[0]表示第一个子图的实例，ax[1]表示第二个子图的实例，所以可以通过ax[0]回到第一个子图绘图，画出红色的折线图。



2.2 plt.figure

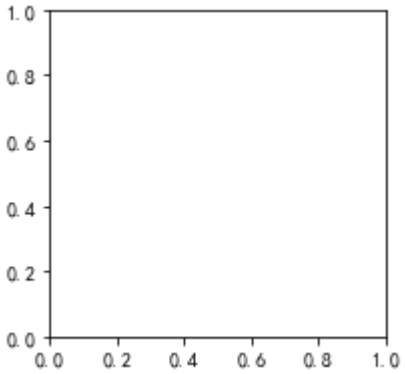
但是一般来说我们是通过以下方式来创建实例figure和实例axes：

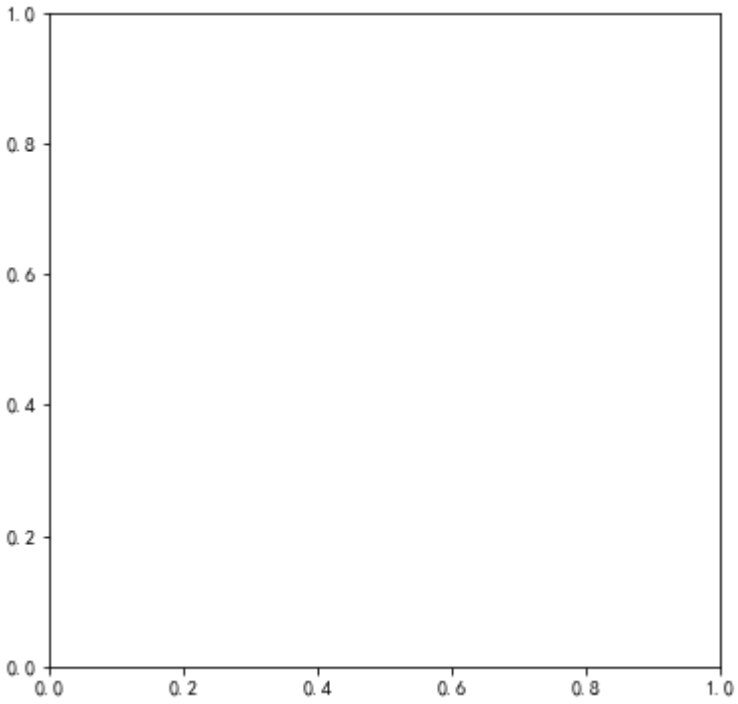
plt.figure(num, figsize, dpi, facecolor, edgecolor, frameon）

- num：整数或字符串，可选，默认值：无
  - 如果未提供，将创建新图形和图形编号。图形对象将此数字保存在“数字”中
  - 如果提供了num，将在该编号的图形中绘图。
  - 如果num是一个字符串，该字符串作为图标题。
- figsize：整数元组，可选，默认值：无
  - [宽度，高度]（英寸）
  - 如果未提供，则默认为[6.4,4.8]
- dpi：整数，可选，默认值：无
  - 这个数字的分辨率。如果未提供，则默认为100
- facecolor：背景颜色。如果未提供，则默认为"w"。
- edgecolor：边框颜色。如果未提供，则默认为"w"。
- frameon：布尔型，可选，默认值：True。如果为False，则禁止绘制图框。

In [15]:

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


fig_01=plt.figure(1,figsize=[3,3])  #建立“画布1”
ax_01=plt.axes()    #小心写成axis

fig_02=plt.figure(2,figsize=[6,6])  #建立“画布2”
ax_02=plt.axes()    #小心写成axis

plt.show();
```





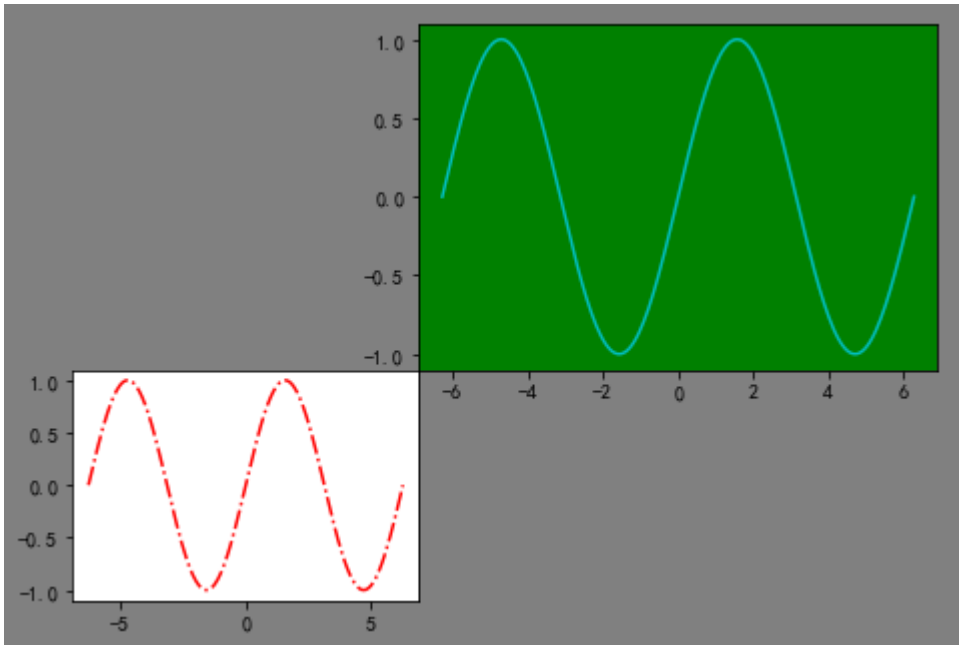
2.3 plt.axes

plt.axes用来创建一个新的全窗口轴:

- plt.axes([left, bottom, width, height]...)
- 以左下角为原点，右上角为1，设置轴域的[左坐标，底坐标，宽度，高度]
  - 上面的轴域取值范围是左下角（原点）为0，右上角为1。
  - 左坐标和底坐标两个参数对应的就是——该坐标轴的原点位置，为该图形宽度和高度比例。
  - 宽度和高度的参数就是——该坐标轴原点往外扩展，宽度和高度为图形的比例长度。

In [17]:

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

x=np.linspace(-2*np.pi,2*np.pi,200)
y=np.sin(x)

fig=plt.figure(1,facecolor="gray")  #建立“画布1”，颜色设置为灰色

ax_1=plt.axes([0,0,0.4,0.4],facecolor="w")    #轴域颜色设置成白色
ax_1.plot(x,y,ls="-.",color="r")

ax_2=plt.axes([0.4,0.4,0.6,0.6],facecolor="g")    #轴域颜色设置成绿色
ax_2.plot(x,y,ls="-",color="c")

plt.show();
```

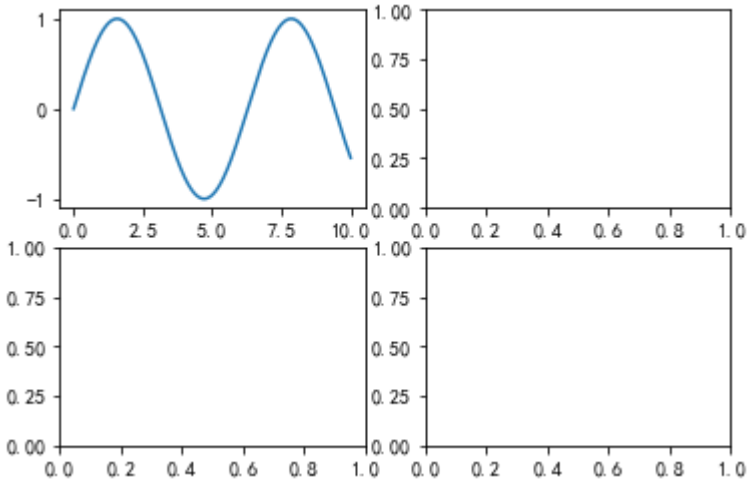


2.4 常见的建立figure和axes的方法

2.4.1 通过plt.subplots()

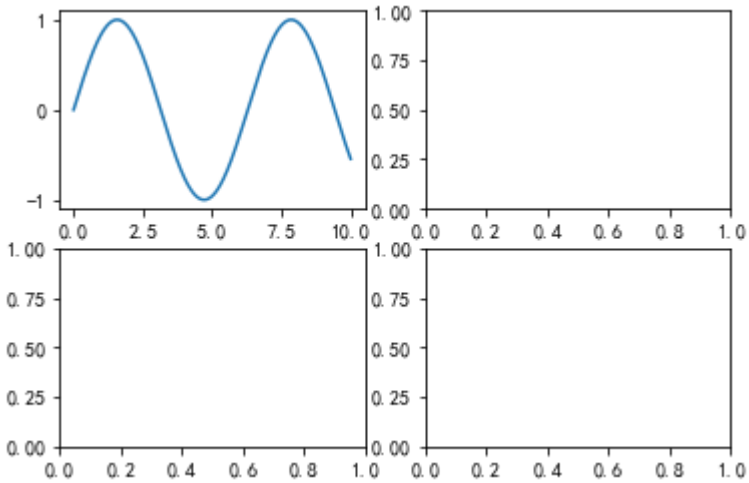
In [18]:

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

x=np.linspace(0,10,1000)
y=np.sin(x)

fig,((ax_01,ax_02),(ax_03,ax_04))=plt.subplots(2,2)

ax_01.plot(x,y)

plt.show();
# # ax_02.plot(x,y)

# # ax_03.plot(x,y)

# # ax_04.plot(x,y)
```



In [19]:

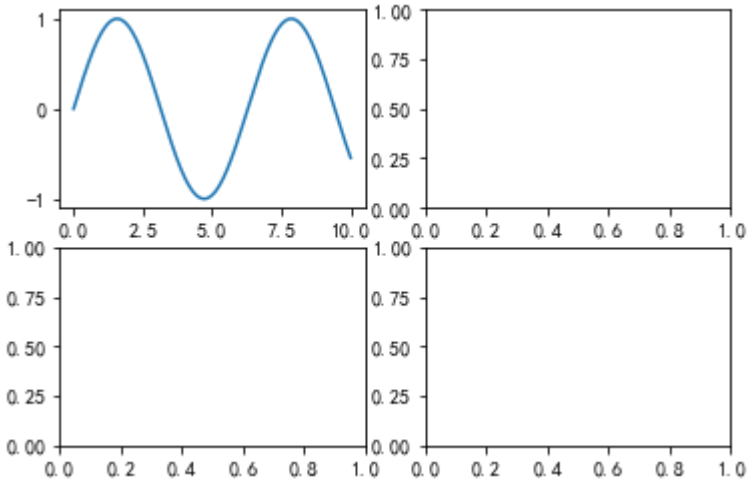
```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(0,10,1000)
y=np.sin(x)

fig,ax=plt.subplots(2,2)

ax_01=ax[0][0]
ax_01.plot(x,y)

ax_02=ax[0][1]
# ax_02.plot(x,y)

ax_03=ax[1][0]
# ax_03.plot(x,y)

ax_04=ax[1][1]
# ax_04.plot(x,y)

plt.show();
```



2.4.2 通过fig.add_subplot()

In [20]:

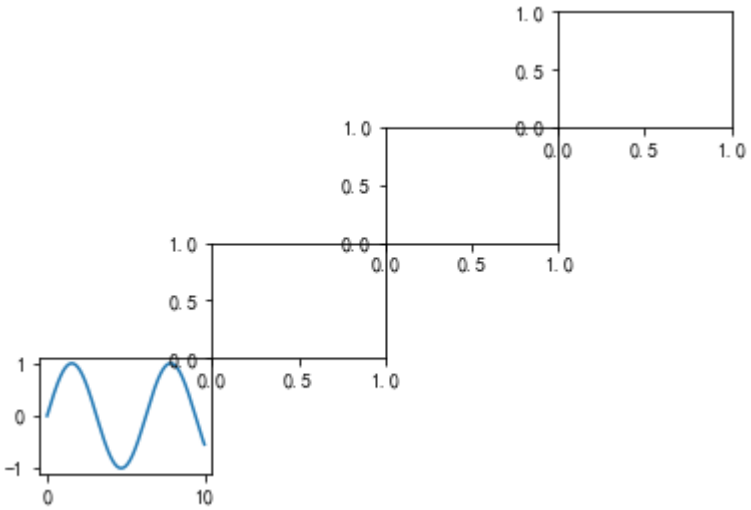
```
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()

ax_01 = fig.add_subplot(221)
ax_01.plot(x,y)

ax_02 = fig.add_subplot(222)
# ax_02.plot(x,y)

ax_03 = fig.add_subplot(223)
# ax_03.plot(x,y)
ax_04 = fig.add_subplot(224)
# ax_04.plot(x,y)

plt.show();
```



2.4.3 通过fig.add_axes()

In [21]:

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

fig = plt.figure()
ax_01 = fig.add_axes([0.1,0.1,0.2,0.2])  #记住括号里面是列表
ax_01.plot(x,y)

ax_02 = fig.add_axes([0.3,0.3,0.2,0.2])
# ax_02.plot(x,y)

ax_03 = fig.add_axes([0.5,0.5,0.2,0.2])
# ax_03.plot(x,y)

ax_04 = fig.add_axes([0.7,0.7,0.2,0.2])
# ax_04.plot(x,y)

plt.show();
```



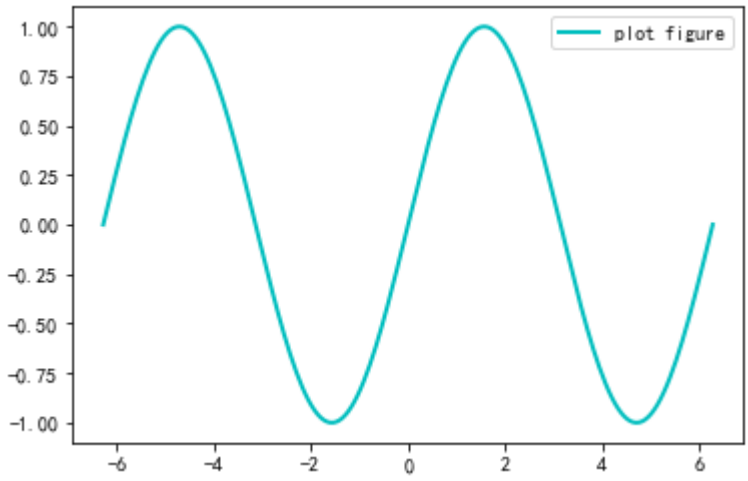
2.5 绘图

使用面线对象接口绘图。

2.5.1 折线图ax.plot

In [24]:

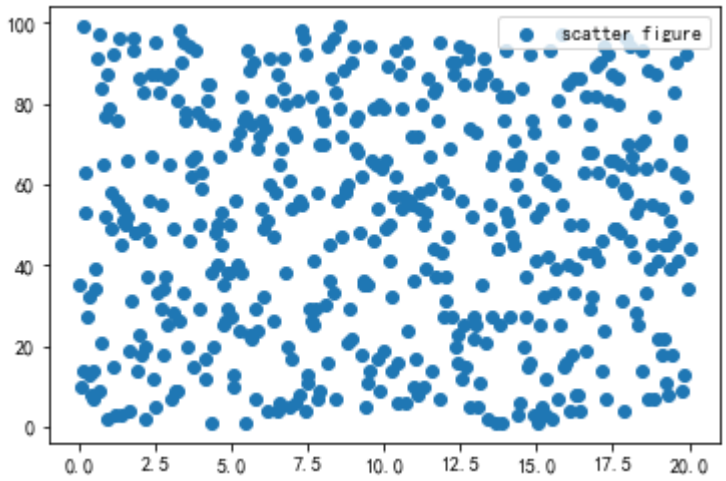
```
import matplotlib.pyplot as plt
import numpy as np

x=np.linspace(-2*np.pi,2*np.pi,200)
y=np.sin(x)

fig=plt.figure()  #建立“画布”
ax=plt.axes()    #小心写成axis

ax.plot(x,y,ls="-",lw=2,label="plot figure",color="c")
ax.legend()

# plt.plot(x,y,ls="-",lw=2,label="plot figure",color="c")
# plt.legend()        #将标签显示出来
plt.show();
```



2.5.2 散点图ax.scatter

In [31]:

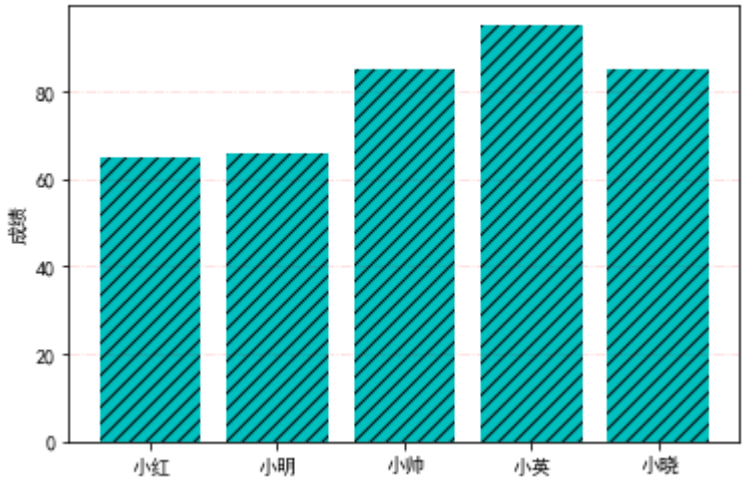
```
import matplotlib.pyplot as plt
import numpy as np

fig=plt.figure()  #建立“画布”
ax=plt.axes()    #小心写成axis

x=np.linspace(0.05,20,500)
y=np.random.randint(1,100,500)        #从0到1之间，生成1500个

ax.scatter(x,y,label="scatter figure")
ax.legend()
# plt.scatter(x,y,label="scatter figure")
# plt.legend()
plt.show();
```



2.5.3 柱状图ax.bar

In [32]:

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

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

x=[i for i in range(1,6)]
y= np.random.randint(45,100,5)
z=("小红","小明","小帅","小英","小晓")

ax.set_ylabel("成绩")
ax.grid(linestyle="-.",color="r",axis="y",alpha=0.15)
ax.bar(x,y,align="center",color="c",tick_label=z,hatch="///")

# plt.ylabel("成绩")
# plt.grid(linestyle="-.",color="r",axis="y",alpha=0.15)
# plt.bar(x,y,align="center",color="c",tick_label=z,hatch="///")

plt.show();
```

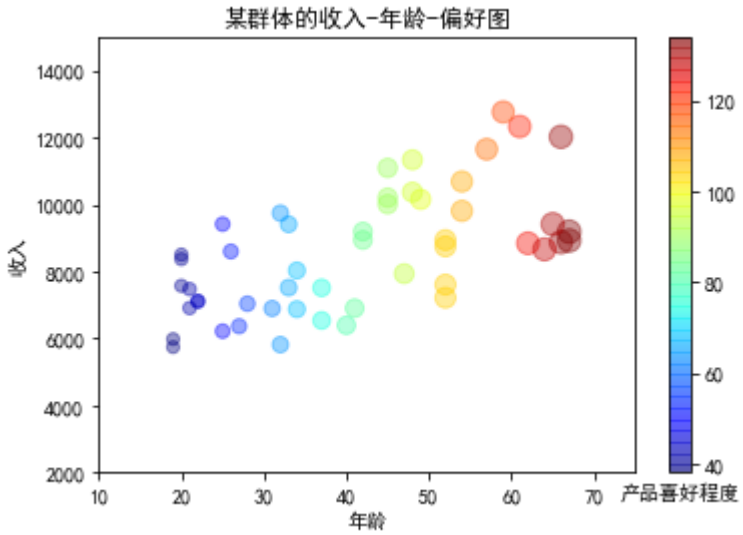


2.6 Matplotlib陷阱

- 绝大多数plt函数都可以直接转换成ax方法，比如plt.plot()-->ax.plot()、plt.legend()-->ax.legend()等。
- 在设置坐标轴上下限、坐标轴标题和图形标题方面，会稍有不同：
  - 设置坐标轴标题
    - plt.xlabel()-->ax.set_xlabel()
    - plt.ylabel()-->ax.set_ylabel()
  - 设置坐标轴上下限
    - plt.xlim()-->ax.set_xlim()
    - plt.ylim()-->ax.set_ylim()
  - 设置图形标题
    - plt.title()-->ax.set_title()
- 用面向对象接口画图的另外一个好处就是，你可以使用ax.set()对图形的属性进行一次性定义：

In [37]:

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

np.random.seed(100)
fig=plt.figure()  #建立“画布”
ax=plt.axes()    #小心写成axis

age=np.random.randint(18,70,50)
income=age*100+np.random.randint(2000,7000,50)
edu=age*2
prefer=age*2

plt.scatter(age,income,edu,prefer,cmap="jet",alpha=0.4,marker="o")

ax.set(xlim=(10,75),ylim=(2000,15000),xlabel="年龄",ylabel="收入",title="某群体的收入-年龄-

aa=plt.colorbar()

aa.ax.set_xlabel('产品喜好程度')

plt.show();
```

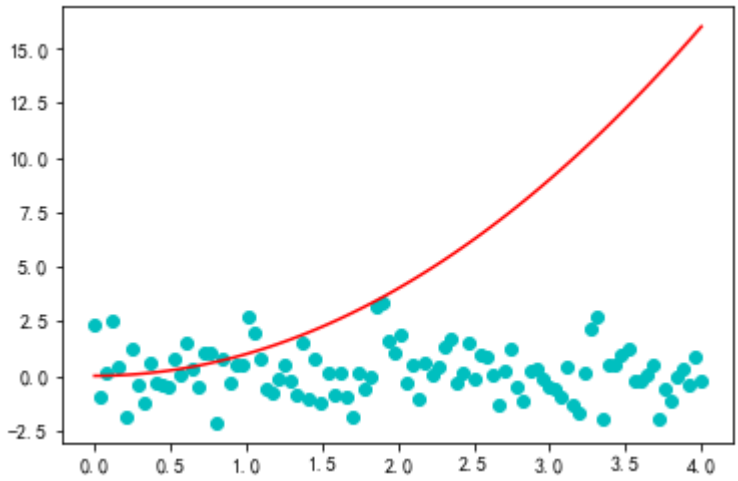


3 共享绘图区域

如何将多张图画在同一个坐标轴？

In [40]:

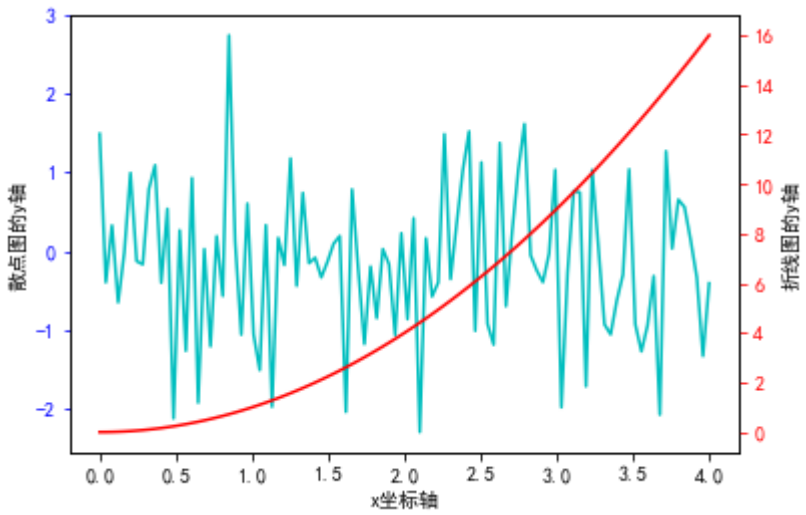
```
import matplotlib as mpl
import matplotlib.pyplot as plt

x=np.linspace(0.0,4.0,100)
y_01=np.random.randn(100)
plt.scatter(x,y_01,c="c")


x=np.linspace(0.0,4.0,100)
y_02=x**2
plt.plot(x,y_02,c="r")

plt.show();
```



如果只想共享x轴，y轴各自保留呢？

In [41]:

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

fig,ax_01=plt.subplots()        #生成一个画布对象fig，以及一个坐标轴实例数组ax_01



#通过实例ax_01绘制x、y坐标轴标签
x=np.linspace(0.0,4.0,100)
y_01=np.random.randn(100)
ax_01.plot(x,y_01,c="c")        #通过实例ax_01来绘图

ax_01.set_xlabel("x坐标轴")
ax_01.set_ylabel("散点图的y轴")
ax_01.tick_params("y",colors="b")  # 绘制ax_01的y坐标轴的主刻度线和刻度标签颜色(蓝色)


#通过ax.twins()来“克隆”ax_01的双胞胎轴域ax_02
ax_02=ax_01.twinx()

x=np.linspace(0.0,4.0,100)
y_02=x**2
ax_02.plot(x,y_02,c="r")

ax_02.set_ylabel("折线图的y轴")
ax_02.tick_params("y",colors="r")  # 绘制ax_02的y坐标轴的主刻度线和刻度标签颜色（红色）

plt.show();
```



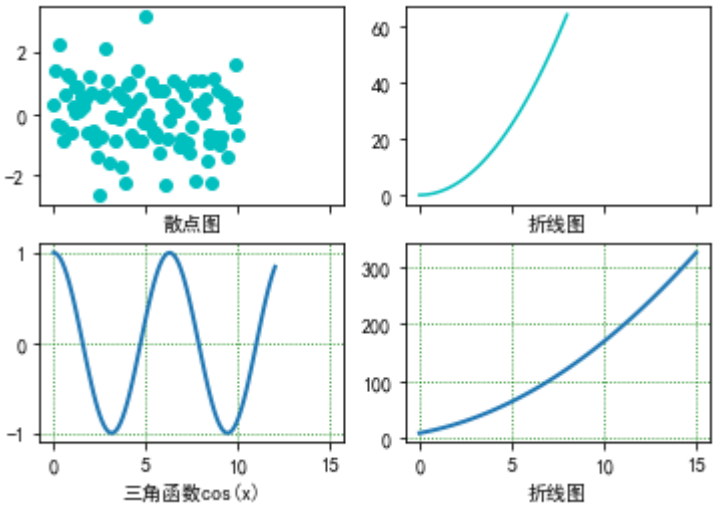
3.1 全部子图共享某坐标轴

有时候，我们将多个子图放在一起显示，是想要某些子图之间做出对比，这个时候，我们可能想将子图的坐标轴设置得一样，此时该怎么办？

- 此时我们可以通过修改subplots(shape,sharex,sharey)里面的sharex,sharey两个参数，以sharex为例：
  - sharex="all"：所有的x轴都相同，以子图中x轴范围最大的作为共享x轴
  - sharex="row"：同一行的子图，所有的x轴都相同
  - sharex="col"：同一列的子图，所有的x轴都相同
  - sharex="none"：坐标轴不变
- sharex,sharey两个参数都可以同时在subplots()中使用。

In [57]:

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

def hehe():
fig,ax=plt.subplots(2,2,sharex="all")          #修改sharex,sharey


ax_01=ax[0,0]
x_01=np.linspace(0.0,10.0,100)
y_01=np.random.randn(100)
ax_01.scatter(x_01,y_01,c="c")
ax_01.set_xlabel("散点图")

ax_02=ax[0,1]
x_02=np.linspace(0.0,8.0,100)
y_02=x_02**2
ax_02.plot(x_02,y_02,c="c")
ax_02.set_xlabel("折线图")

ax_03=ax[1,0]
x_03=np.linspace(0.0,12.0,100)
y_03=np.cos(x_03)
ax_03.plot(x_03,y_03,lw=2,ls="-")
ax_03.grid(True,ls=":",c="g")
ax_03.set_xlabel("三角函数cos(x)")

ax_04=ax[1,1]
x_04=np.linspace(0.0,15.0,100)
y_04=(x_04+3)**2
ax_04.plot(x_04,y_04,lw=2,ls="-")
ax_04.grid(True,ls=":",c="g")
ax_04.set_xlabel("折线图")

# plt.suptitle("子图共享同一坐标的展示");

plt.show();
```

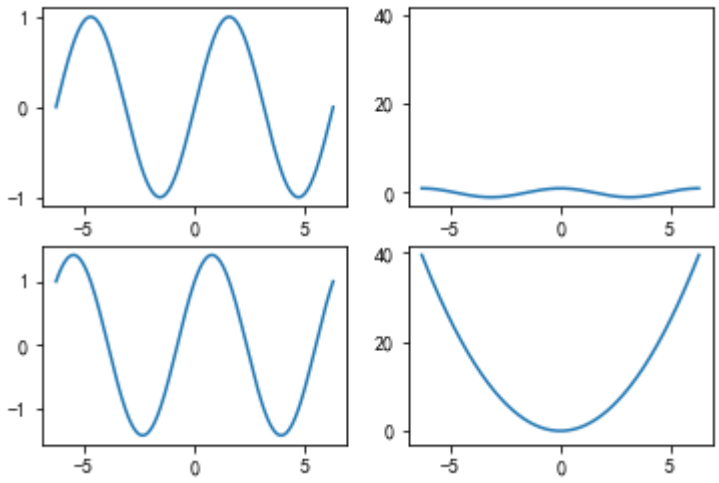


3.2 与某子图共享坐标轴

在使用subplot()制图方法的时候，创建子图轴的实例的时候，用sharex或sharey参数指定与那个子图共享坐标轴。

In [52]:

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

x=np.linspace(-2*np.pi,2*np.pi,200)
y_01=np.sin(x)
y_02=np.cos(x)
y_03=np.sin(x)+np.cos(x)
y_04=x**2


fig,ax=plt.subplots(2,2)

ax_01=plt.subplot(221)
ax_01.plot(x,y_01)

ax_02=plt.subplot(222)
ax_02.plot(x,y_02)

ax_03=plt.subplot(223)
ax_03.plot(x,y_03)

ax_04=plt.subplot(224,sharey=ax_02)    #用sharex或sharey参数指定与那个子图共享坐标轴
ax_04.plot(x,y_04)

plt.show();
```



3.3 修改子图之间的空隙

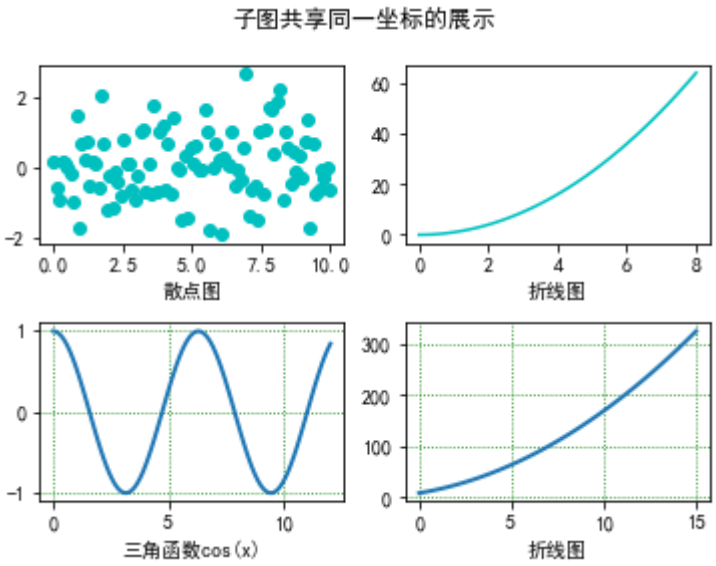
从上图我们看到前面一行的子图的x轴标签，并没有显示出来，这是因为第一行的子图和第二行的子图间隙太小的缘故，怎么修改子图的间隙呢？

主要通过fig.subplots_adjust(wspace,hspace)中的wspace和hspace修改。

- wspace：子图之间的宽距
- hspace：子图之间的高距

In [62]:

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


fig,ax=plt.subplots(2,2)
fig.subplots_adjust(hspace=0.45)    #可以在开头就添加

ax_01=ax[0,0]
x_01=np.linspace(0.0,10.0,100)
y_01=np.random.randn(100)
ax_01.scatter(x_01,y_01,c="c")
ax_01.set_xlabel("散点图")

ax_02=ax[0,1]
x_02=np.linspace(0.0,8.0,100)
y_02=x_02**2
ax_02.plot(x_02,y_02,c="c")
ax_02.set_xlabel("折线图")

ax_03=ax[1,0]
x_03=np.linspace(0.0,12.0,100)
y_03=np.cos(x_03)
ax_03.plot(x_03,y_03,lw=2,ls="-")
ax_03.grid(True,ls=":",c="g")
ax_03.set_xlabel("三角函数cos(x)")

ax_04=ax[1,1]
x_04=np.linspace(0.0,15.0,100)
y_04=(x_04+3)**2
ax_04.plot(x_04,y_04,lw=2,ls="-")
ax_04.grid(True,ls=":",c="g")
ax_04.set_xlabel("折线图")

plt.suptitle("子图共享同一坐标的展示")


plt.show();
```



3.4 自动调整坐标轴范围

如果你觉得调整坐标轴范围好数据范围太麻烦，可以使用函数autoscale()，坐标轴范围会自适应调整，用法：

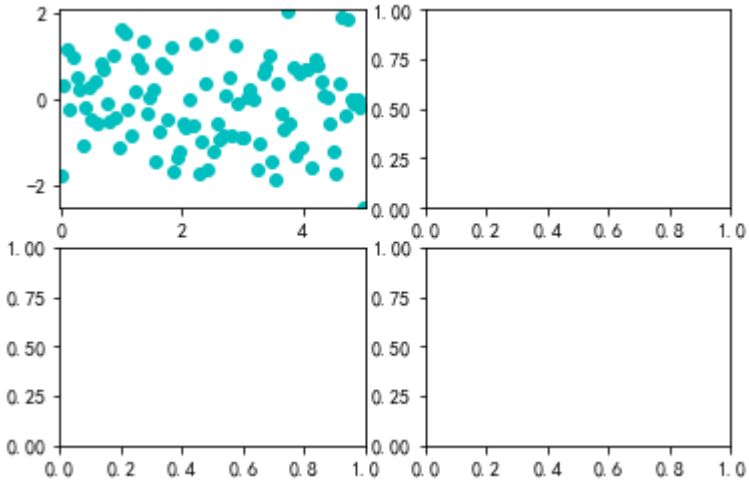
- ax.autoscale(enable=True,axis="both",tight=True)
  - enable:对坐标轴范围进行自适应调整。

- axis：使x,y轴都进行自适应调整。
- tight：让坐标轴的范围调整到数据的范围上。

- 其实将ax.autoscale(enable=True,axis="both",tight=True)

In [63]:

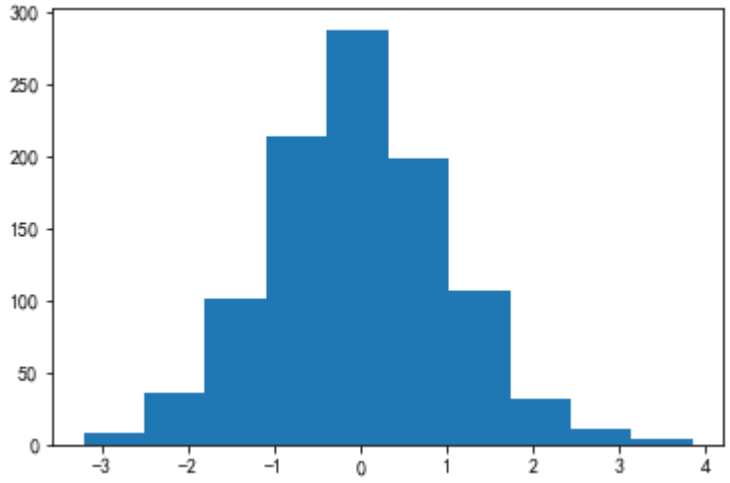
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

np.random.seed(100)

fig,ax=plt.subplots(2,2)

ax_01=ax[0,0]
x_01=np.linspace(0.0,5.0,100)
y_01=np.random.randn(100)
ax_01.scatter(x_01,y_01,c="c")
ax_01.set_xlabel("散点图")
ax_01.autoscale(enable=True,axis="both",tight=True)     #比如你觉得子图1的坐标轴范围不够紧凑

plt.show();
```



4 配置文件与样式表

4.1 修改默认配置：rcParams

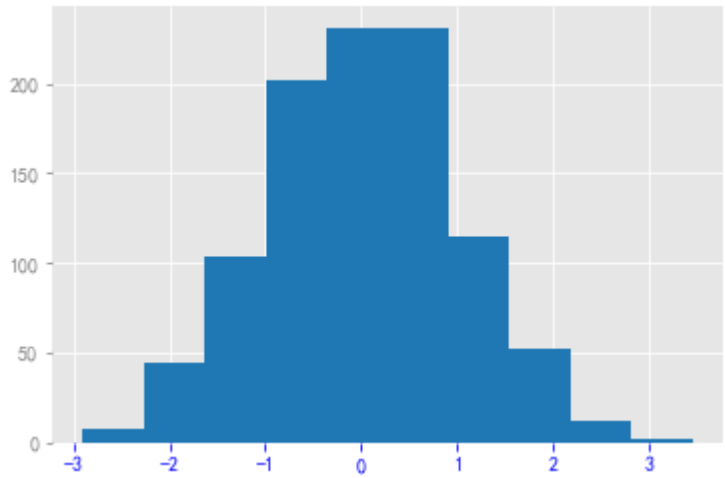
In [64]:
```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.randn(1000)
plt.hist(x) #创建一个图形看看效果

plt.show();
```



In [65]:
```
# 先复制一下目前的rcParams字典，这样可以在修改之后再还原回来
import matplotlib.pyplot as plt

hehe = plt.rcParams.copy()
```

In [84]:
```
plt.rc('axes', facecolor='#E6E6E6', edgecolor='none',   #facecolor设置图形底色、edgecolor外
       axisbelow=True, grid=True)

plt.rc('grid', color='w', linestyle='solid')
plt.rc('xtick', direction='out', color='blue')  #设置x坐标刻度颜色
plt.rc('ytick', direction='out', color='gray')  #设置y坐标刻度颜色
plt.rc('patch', edgecolor='#3498DB')
plt.rc('lines', linewidth=2)
```

In [85]:
```
x=np.random.randn(1000)
plt.hist(x)  #创建一个图形看看效果

plt.show();
```



In [ ]:
```
# 重置 rcParams
plt.rcParams.update(hehe)
```

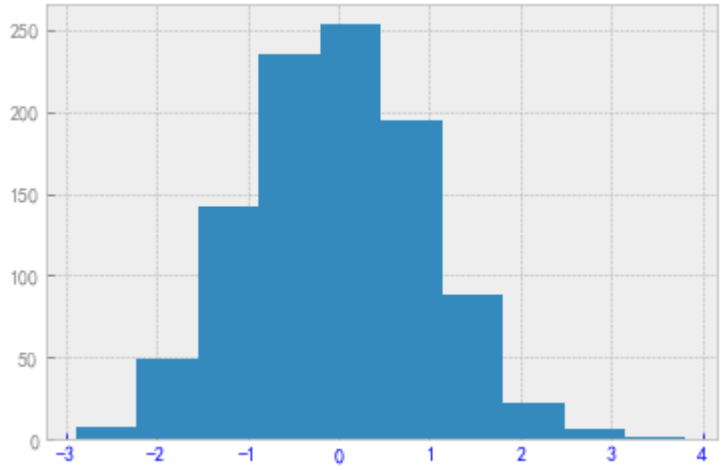
4.2 样式表

In [69]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt

# 查看所以风格
plt.style.available[:]
```

Out[69]:
```
['bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark-palette',
 'seaborn-dark',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'seaborn',
 'Solarize_Light2',
 'tableau-colorblind10',
 '_classic_test']
```

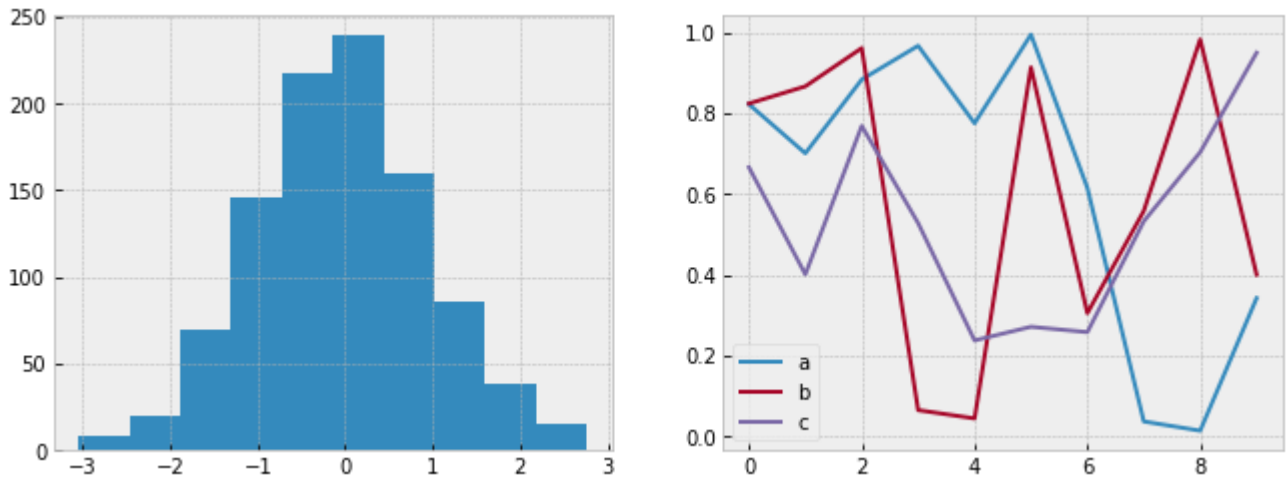
In [74]:
```
# 使用某种样式表的基本方法如下：
plt.style.use("bmh")
#但是这样做会改变后面所有的风格
```

In [75]:
```
import matplotlib.pyplot as plt
import numpy as np

x=np.random.randn(1000)
plt.hist(x); #创建一个图形看看效果
```



In [16]:

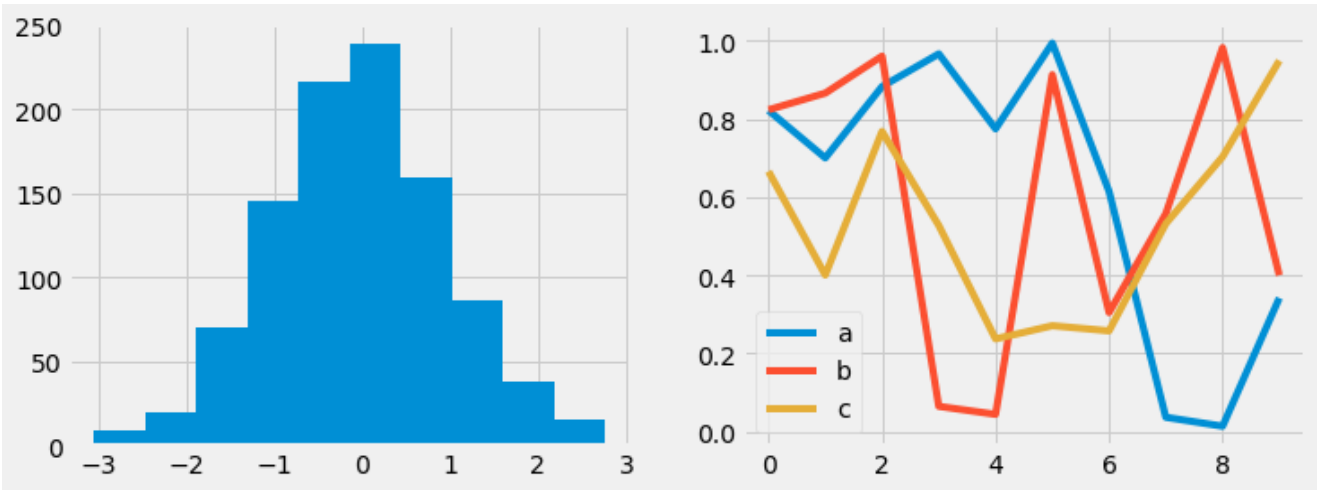
```
# 创建一个可以画两种基本图形的函数
def hehe():
    np.random.seed(0)
    fig, ax = plt.subplots(1, 2, figsize=(11, 4))
    ax[0].hist(np.random.randn(1000))
    for i in range(3):
        ax[1].plot(np.random.rand(10))
    ax[1].legend(['a', 'b', 'c'], loc='lower left')
#下面就用这个函数来演示不同风格的显示效果

# 可以使用风格上下文管理器（context manager）临时更换至另一种风格
with plt.style.context('bmh'):
    hehe()  #这里是写你的定义的图形函数
```



4.2.1 FiveThiryEight style风格

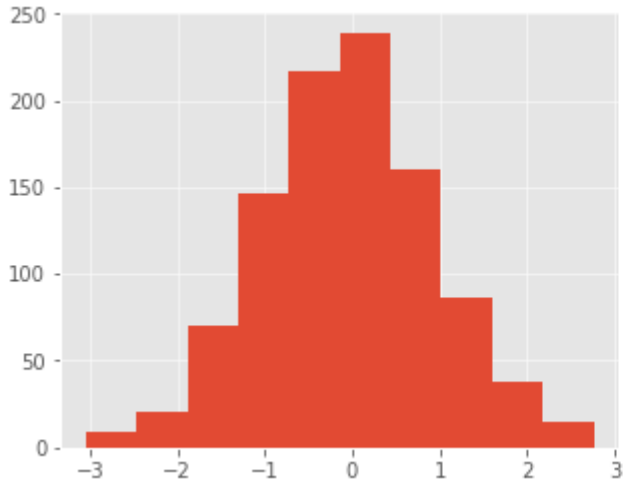
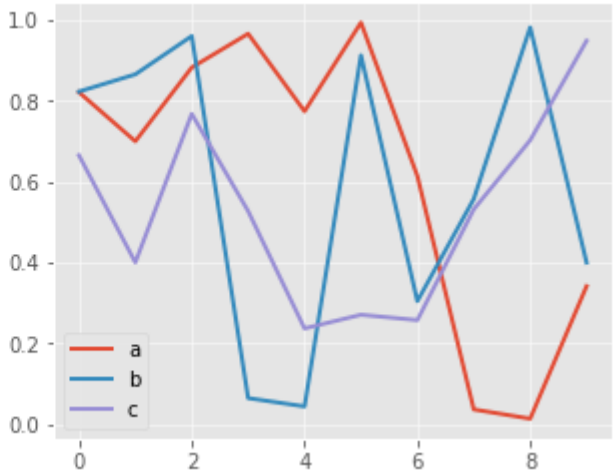
In [18]:

```
with plt.style.context('fivethirtyeight'):
    hehe()
```



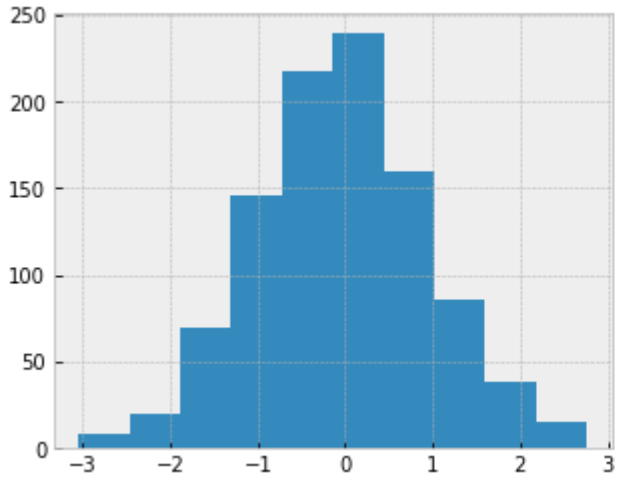
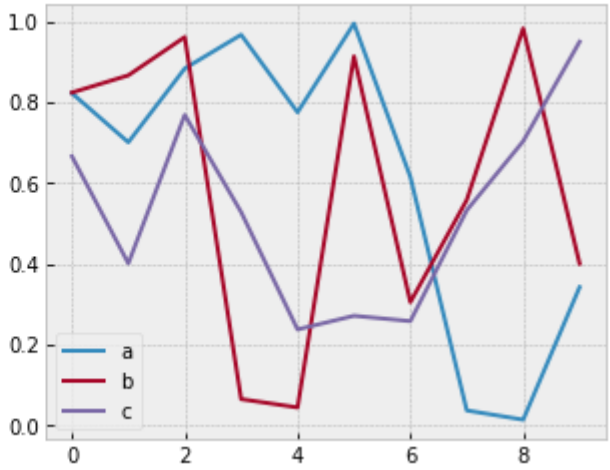
4.2.2 ggplot风格

In [19]:
```
with plt.style.context('ggplot'):
    hehe()
```

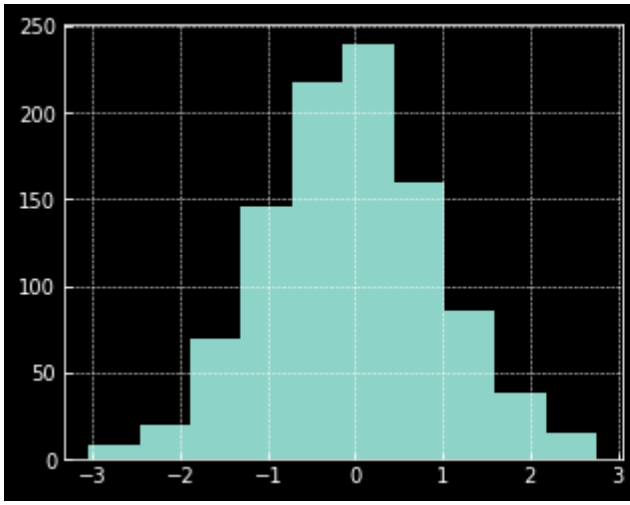
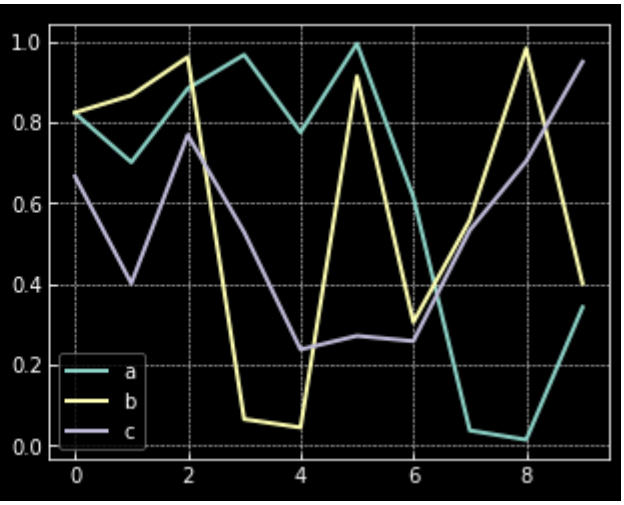
4.2.3 bmh风格

In [20]:
```
with plt.style.context('bmh'):
    hehe()
```

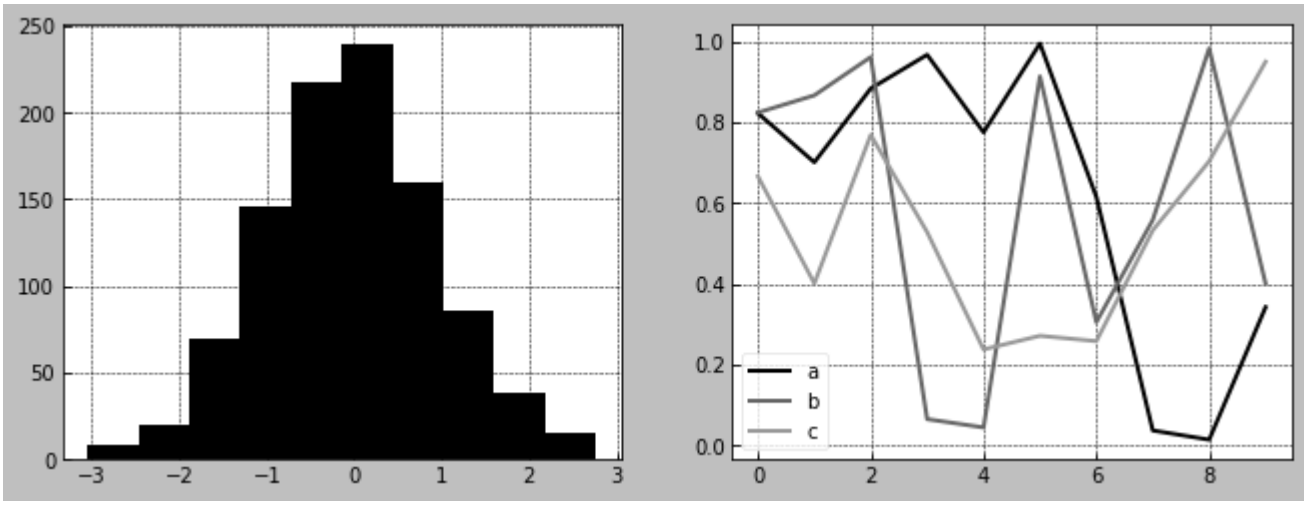
4.2.4 黑色背景风格

In [21]:
```
with plt.style.context('dark_background'):
    hehe()
```

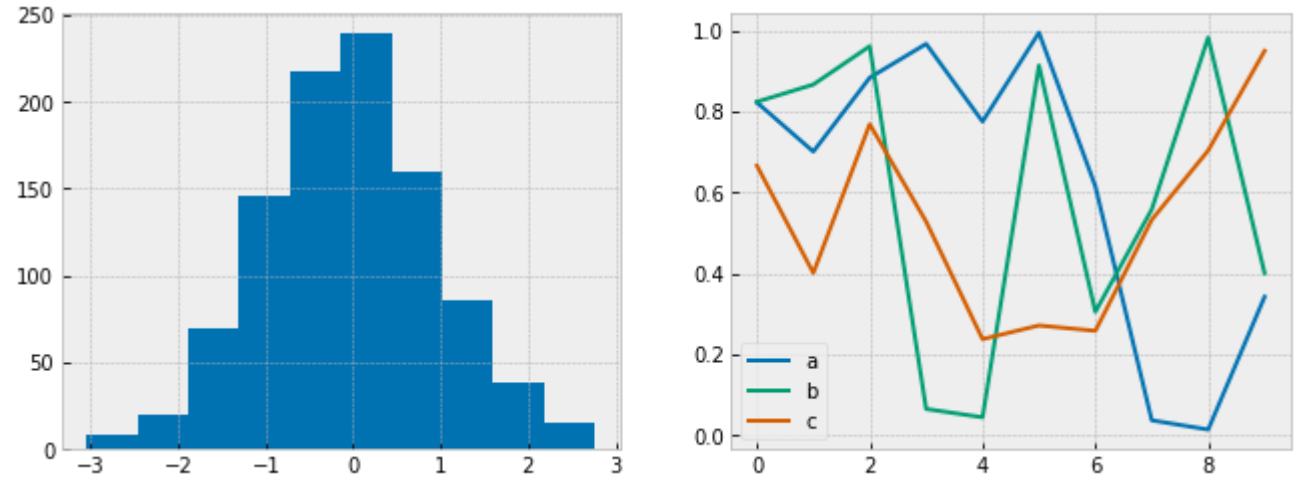
4.2.5 灰度风格

In [22]:
```
with plt.style.context('grayscale'):
    hehe()
```

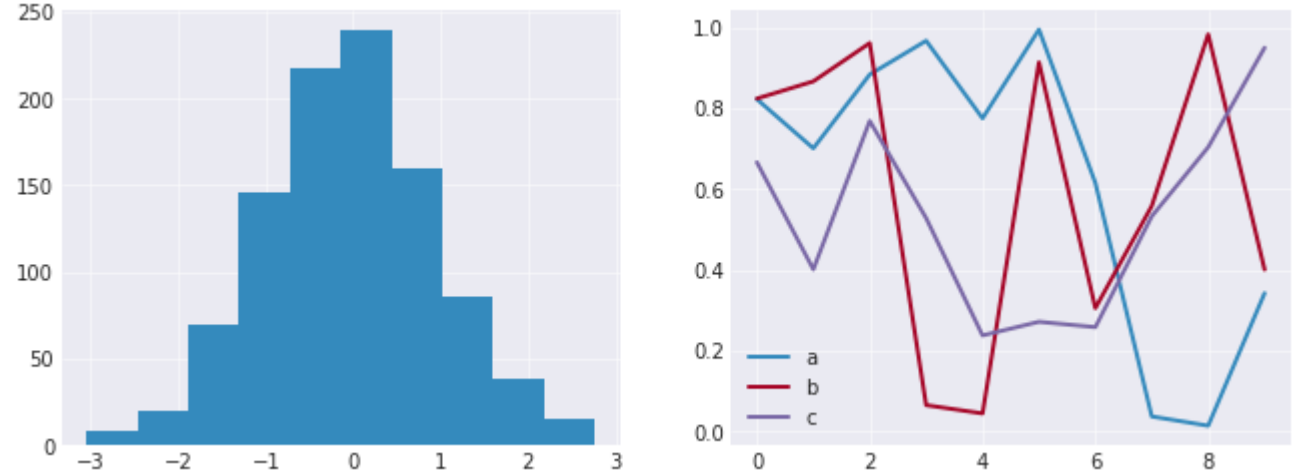


4.2.6 seaborn风格

In [23]:
```
with plt.style.context('seaborn-colorblind'):
    hehe()
```



In [24]:
```
with plt.style.context('seaborn-darkgrid'):
    hehe()
```



https://matplotlib.org/gallery (https://matplotlib.org/gallery)