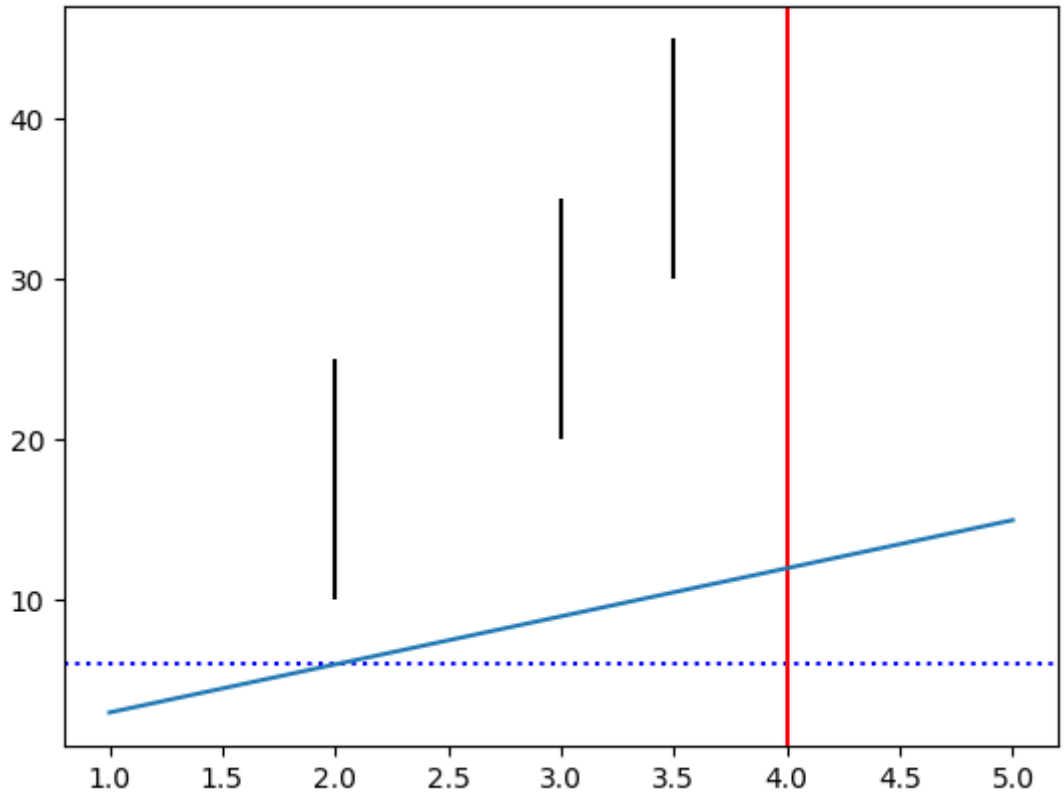
数据分析DAY05

matplotlib基本功能详解

基本绘图

1）绘图核心API

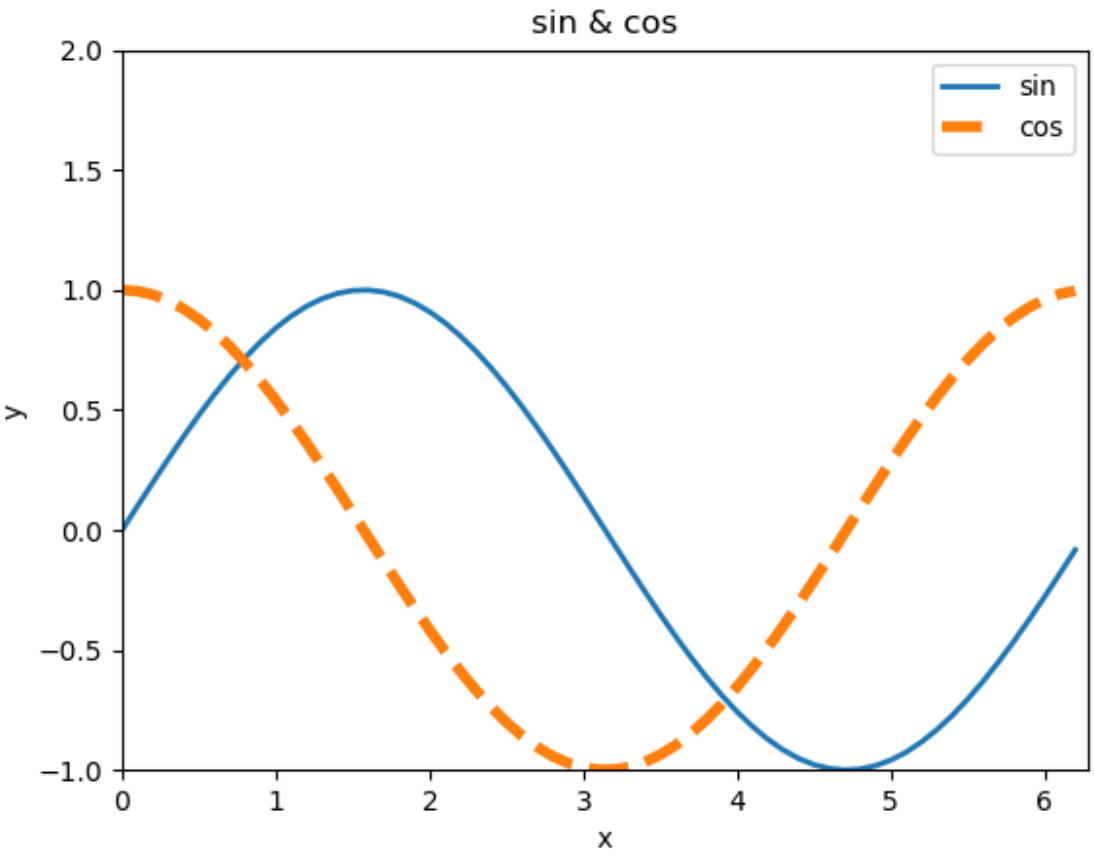
案例： 绘制简单直线




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

# 绘制简单直线
x = np.array([1, 2, 3, 4, 5])
y = np.array([3, 6, 9, 12, 15])

# 绘制水平线、垂线
plt.axhline(y=6, ls=":", c="blue")  # 添加水平直线
plt.axvline(x=4, ls="-", c="red")  # 添加垂直直线

# 绘制多段垂线
plt.vlines([2, 3, 3.5],  # 垂线的x坐标值
           [10, 20, 30],  # 每条垂线起始y坐标
           [25, 35, 45])  # 每条垂线结束y坐标
```

```
plt.plot(x, y)
plt.show() # 显示图片，阻塞方法
```

2）设置线型、线宽



linestyle: 设置线型，常见取值有实线（'-'）、虚线（'--'）、点虚线（'-.'）、点线（':'）

linewidth：线宽

color：颜色（red, blue, green）

alpha: 设置透明度（0~1之间）

案例：绘制正弦、余弦曲线，并设置线型、线宽、颜色、透明度

```
# 绘制正弦曲线
import numpy as np
import matplotlib.pyplot as plt
import math

x = np.arange(0, 2 * np.pi, 0.1)  # 以0.1为单位，生成0~6的数据
print(x)
y1 = np.sin(x)
y2 = np.cos(x)

# 绘制图形
plt.plot(x, y1, label="sin", linewidth=2)  # 实线，线宽2像素
plt.plot(x, y2, label="cos", linestyle="--", linewidth=4)  # 虚线，线宽4像素
```

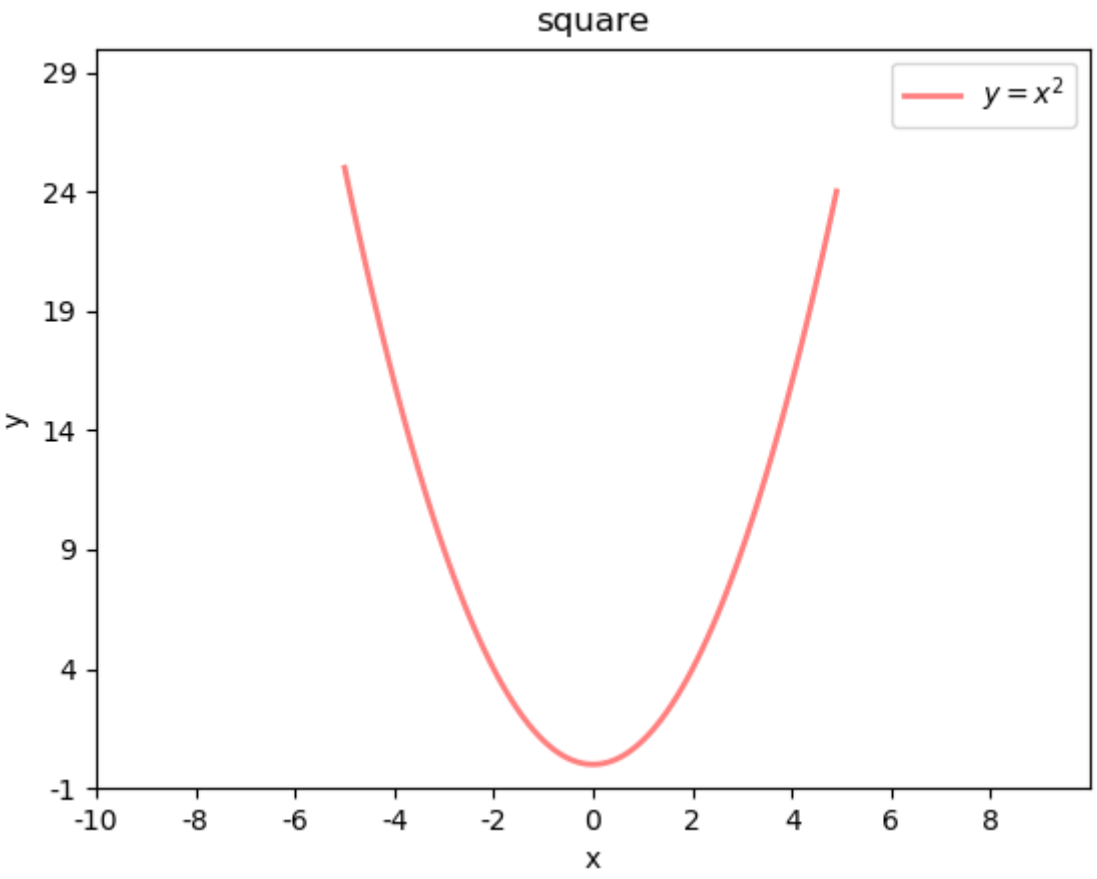
```
plt.xlabel("x")  # x轴文字
plt.ylabel("y")  # y轴文字

# 设置坐标轴范围
plt.xlim(0, 2 * math.pi)
plt.ylim(-1, 2)

plt.title("sin & cos")  # 图标题
plt.legend()  # 图例
plt.show()
```

3）设置坐标轴范围

语法：

```
#x_limt_min:    <float> x轴范围最小值
#x_limit_max:   <float> x轴范围最大值
plt.xlim(x_limt_min, x_limit_max)
#y_limt_min:    <float> y轴范围最小值
#y_limit_max:   <float> y轴范围最大值
plt.ylim(y_limt_min, y_limit_max)
```

4）设置坐标刻度



语法：

```
#x_val_list:    x轴刻度值序列
#x_text_list:   x轴刻度标签文本序列 [可选]
plt.xticks(x_val_list , x_text_list )
#y_val_list:    y轴刻度值序列
#y_text_list:   y轴刻度标签文本序列 [可选]
plt.yticks(y_val_list , y_text_list )
```

案例：绘制二次函数曲线

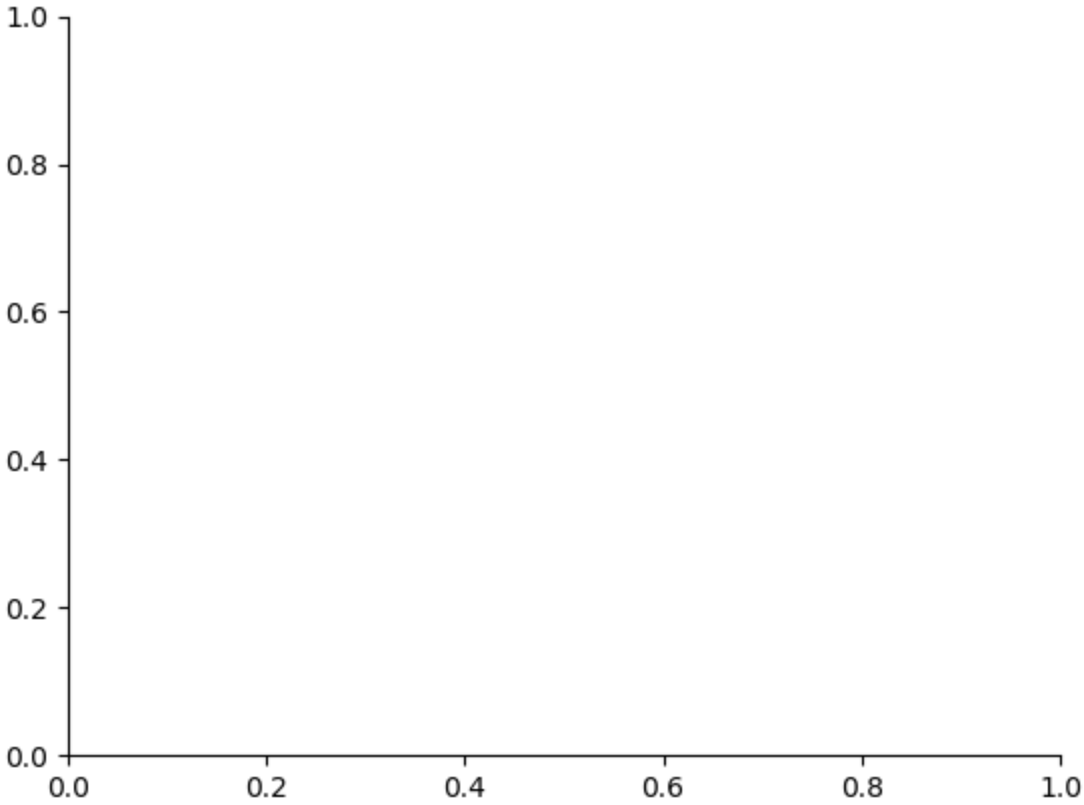
```
# 绘制二次函数曲线
import numpy as np
import matplotlib.pyplot as plt
import math

x = np.arange(-5, 5, 0.1)  # 以0.1为单位，生成-5~5的数据
print(x)
y = x ** 2

# 绘制图形
plt.plot(x, y, label="$y = x ^ 2$",
         linewidth=2,  # 线宽2像素
         color="red",  # 颜色
         alpha=0.5)  # 透明度

plt.xlabel("x")  # x轴文字
plt.ylabel("y")  # y轴文字

# 设置坐标轴范围
plt.xlim(-10, 10)
plt.ylim(-1, 30)

# 设置刻度
x_tck = np.arange(-10, 10, 2)
x_txt = x_tck.astype("U")
plt.xticks(x_tck, x_txt)

y_tck = np.arange(-1, 30, 5)
y_txt = y_tck.astype("U")
plt.yticks(y_tck, y_txt)

plt.title("square")  # 图标题
plt.legend(loc="upper right")  # 图例 upper right, center
plt.show()
```

刻度文本的特殊语法 -- LaTex排版语法字符串

```
r'$x^n+y^n=z^n$',   r'$\int\frac{1}{x} dx = \ln |x| + C$',     r'$-\frac{\pi}{2}$'
```

$$x^n + y^n = z^n, \int \frac{1}{x} dx = \ln |x| + C, -\frac{\pi}{2}$$

5）设置坐标轴



坐标轴名：left / right / bottom / top

```
# 获取当前坐标轴字典，{'left':左轴,'right':右轴,'bottom':下轴,'top':上轴 }
ax = plt.gca()
# 获取其中某个坐标轴
axis = ax.spines['坐标轴名']
# 设置坐标轴的位置。 该方法需要传入2个元素的元组作为参数
# type: <str> 移动坐标轴的参照类型  一般为'data' (以数据的值作为移动参照值)
# val:  参照值
axis.set_position((type, val))
# 设置坐标轴的颜色
# color: <str> 颜色值字符串
axis.set_color(color)
```

案例：设置坐标轴格式

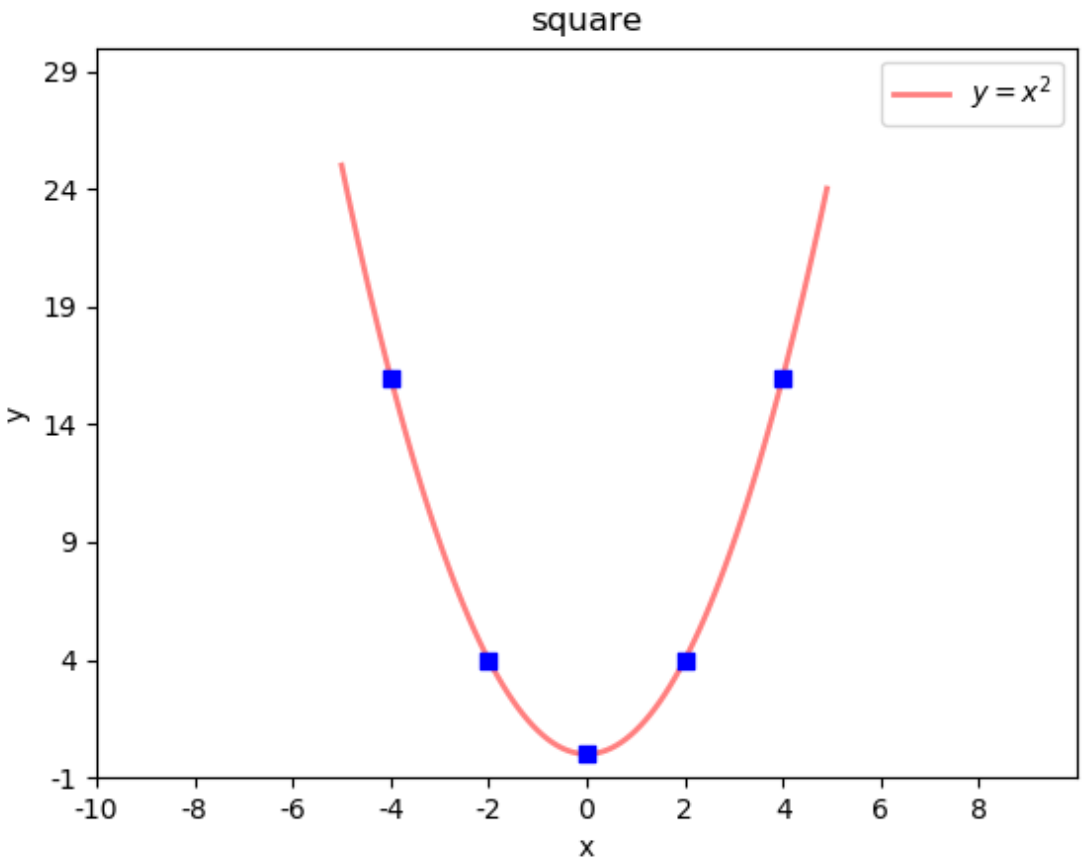
```
# 设置坐标轴
import matplotlib.pyplot as plt

ax = plt.gca()
axis_b = ax.spines['bottom']  # 获取下轴
axis_b.set_position(('data', 0))  # 设置下轴位置，以数据作为参照值

axis_l = ax.spines['left']  # 获取左轴
axis_l.set_position(('data', 0))  # 设置左轴位置，以数据作为参照值

ax.spines['top'].set_color('none')  # 设置顶部轴无色
ax.spines['right'].set_color('none')  # 设置右部轴无色

plt.show()
```

6）图例

显示两条曲线的图例，并测试loc属性。

```
# 再绘制曲线时定义曲线的label
# label: <关键字参数 str> 支持LaTex排版语法字符串
plt.plot(xarray, yarray ... label='', ...)
# 设置图例的位置
# loc: <关键字参数> 制定图例的显示位置 (若不设置loc，则显示默认位置)
#    ===============   =============
#    Location String   Location Code
#    ===============   =============
#    'best'            0
#    'upper right'     1
#    'upper left'      2
#    'lower left'      3
#    'lower right'     4
#    'right'           5
#    'center left'     6
#    'center right'    7
#    'lower center'    8
#    'upper center'    9
#    'center'          10
#    ===============   =============
plt.legend(loc='')
```

7）特殊点



语法：

```
# xarray: <序列> 所有需要标注点的水平坐标组成的序列
# yarray: <序列> 所有需要标注点的垂直坐标组成的序列
plt.scatter(xarray, yarray,
            marker='',        #点型 ~ matplotlib.markers
            s='',             #大小
            edgecolor='',     #边缘色
            facecolor='',     #填充色
            zorder=3          #绘制图层编号 （编号越大，图层越靠上）
)
```

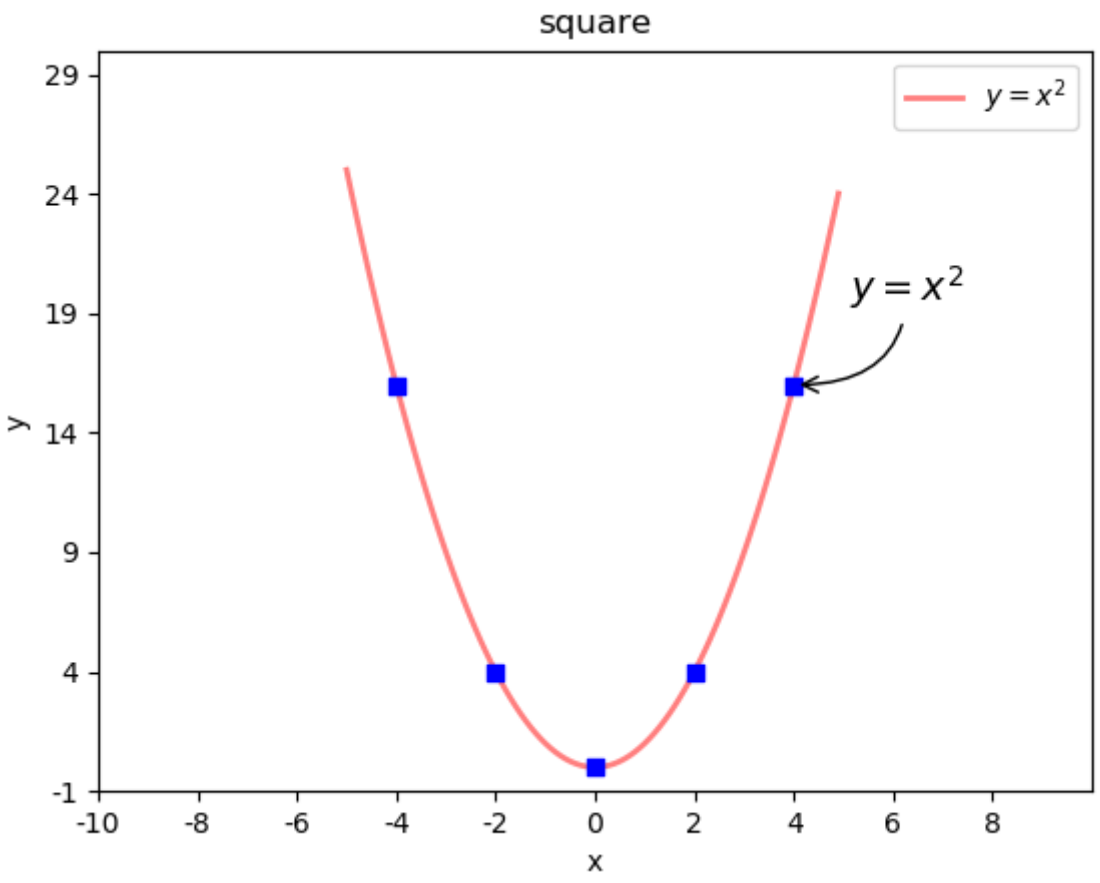
示例：在二次函数图像中添加特殊点

```
# 绘制特殊点
plt.scatter(x_tck,  # x坐标数组
            x_tck ** 2,  # y坐标数组
            marker="s",  # 点形状 s:square
            s=40,  # 大小
            facecolor="blue",  # 填充色
            zorder=3)  # 图层编号
```

marker点型可参照：help(matplotlib.markers)

也可参照附录： matplotlib point样式

8）备注



语法：

```
# 在图表中为某个点添加备注。包含备注文本，备注箭头等图像的设置。
plt.annotate(
    r'$\frac{\pi}{2}$',          #备注中显示的文本内容
    xycoords='data',             #备注目标点所使用的坐标系（data表示数据坐标系）
    xy=(x, y),                   #备注目标点的坐标
    textcoords='offset points', #备注文本所使用的坐标系（offset points表示参照点的偏移坐标系）
    xytext=(x, y),               #备注文本的坐标
    fontsize=14,                 #备注文本的字体大小
    arrowprops=dict()            #使用字典定义文本指向目标点的箭头样式
)
```

arrowprops参数使用字典定义指向目标点的箭头样式

```
#arrowprops字典参数的常用key
arrowprops=dict(
    arrowstyle='',      #定义箭头样式
    connectionstyle=''  #定义连接线的样式
)
```

箭头样式（arrowstyle）字符串如下

```
============   ============================================
Name           Attrs
============   ============================================
  '-'          None
  '->'         head_length=0.4,head_width=0.2
  '-['         widthB=1.0,lengthB=0.2,angleB=None
  '|-|'        widthA=1.0,widthB=1.0
  '-|>'        head_length=0.4,head_width=0.2
  '<-'         head_length=0.4,head_width=0.2
  '<->'        head_length=0.4,head_width=0.2
  '<|-'        head_length=0.4,head_width=0.2
  '<|-|>'      head_length=0.4,head_width=0.2
  'fancy'      head_length=0.4,head_width=0.4,tail_width=0.4
  'simple'     head_length=0.5,head_width=0.5,tail_width=0.2
  'wedge'      tail_width=0.3,shrink_factor=0.5
============   ============================================
```

连接线样式（connectionstyle）字符串如下

```
============   ============================================
Name           Attrs
============   ============================================
  'angle'       angleA=90,angleB=0,rad=0.0
  'angle3'      angleA=90,angleB=0`
  'arc'         angleA=0,angleB=0,armA=None,armB=None,rad=0.0
  'arc3'        rad=0.0
  'bar'         armA=0.0,armB=0.0,fraction=0.3,angle=None
============   ============================================
```

示例：在二次函数图像中添加备注

```
# 设置备注
plt.annotate(
    r'$y = x ^ 2$',         #备注中显示的文本内容
    xycoords='data',            #备注目标点所使用的坐标系（data表示数据坐标系）
    xy=(4, 16),                 #备注目标点的坐标 (4,16)
    textcoords='offset points', #备注文本所使用的坐标系（offset points表示参照点的偏移坐标系）
    xytext=(20, 30),                #备注文本的坐标
    fontsize=14,                #备注文本的字体大小
    arrowprops=dict(
        arrowstyle="->", connectionstyle="angle3"
    )           #使用字典定义文本指向目标点的箭头样式
)
```

高级绘图

语法：绘制两个窗口，一起显示。

```
# 手动构建 matplotlib 窗口
plt.figure(
    'sub-fig',                  #窗口标题栏文本
    figsize=(4, 3),     #窗口大小 <元组>
    facecolor=''        #图表背景色
)
plt.show()
```

plt.figure方法不仅可以构建一个新窗口，如果已经构建过title='xxx'的窗口，又使用figure方法构建了title='xxx' 的窗口的话，mp将不会创建新的窗口，而是把title='xxx'的窗口置为当前操作窗口。

设置当前窗口的参数

语法：测试窗口相关参数

```
# 设置图表标题 显示在图表上方
plt.title(title, fontsize=12)
# 设置水平轴的文本
plt.xlabel(x_label_str, fontsize=12)
# 设置垂直轴的文本
plt.ylabel(y_label_str, fontsize=12)
# 设置刻度参数   labelsize设置刻度字体大小
plt.tick_params(..., labelsize=8, ...)
# 设置图表网格线  linestyle设置网格线的样式
    #   -  or solid 粗线
    #   -- or dashed 虚线
    #   -. or dashdot 点虚线
    #   :  or dotted 点线
plt.grid(linestyle='')
# 设置紧凑布局，把图表相关参数都显示在窗口中
plt.tight_layout()
```

示例：绘制两个图像窗口

```
# 绘制两个图像窗口
import matplotlib.pyplot as plt

plt.figure("FigureA", facecolor="lightgray")
plt.grid(linestyle="-.")  # 设置网格线

plt.figure("FigureB", facecolor="gray")
plt.xlabel("Date", fontsize=14)
plt.ylabel("Price", fontsize=14)
plt.grid(linestyle="--")  # 设置网格线
plt.tight_layout()  # 设置紧凑布局

plt.show()
```

执行结果：

1）子图

矩阵式布局

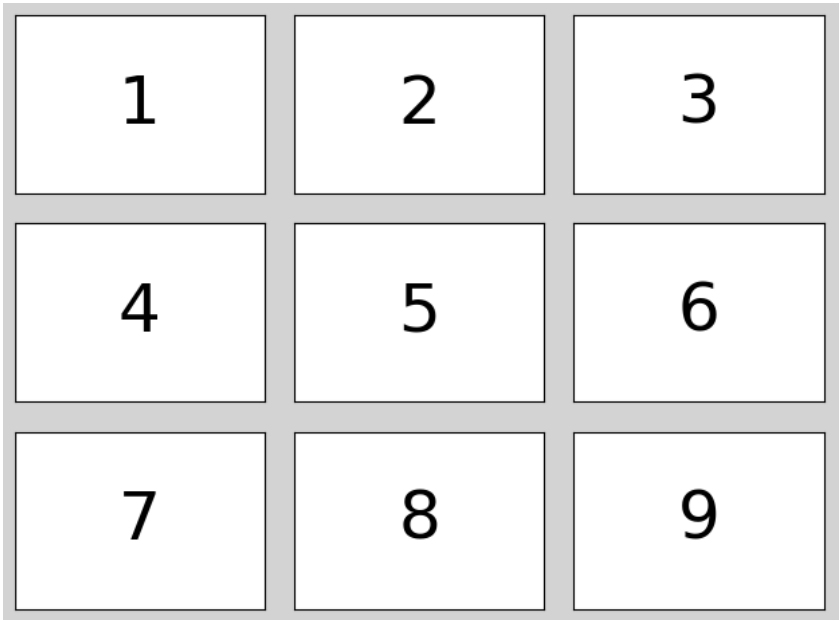
绘制矩阵式子图布局相关API：

```
plt.figure('Subplot Layout', facecolor='lightgray')
# 拆分矩阵
    # rows: 行数
    # cols: 列数
    # num:  编号
plt.subplot(rows, cols, num)
    #   1 2 3
    #   4 5 6
    #   7 8 9
plt.subplot(3, 3, 5)        #操作3*3的矩阵中编号为5的子图
plt.subplot(335)            #简写
```

案例：绘制9宫格矩阵式子图，每个子图中写一个数字。

```
plt.figure('Subplot Layout', facecolor='lightgray')

for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.text(
        0.5, 0.5, i+1,
        ha='center',
        va='center',
        size=36,
        alpha=0.5,
        withdash=False
    )
    plt.xticks([])
    plt.yticks([])

plt.tight_layout()
plt.show()
```

执行结果：



网格式布局(很少使用)

网格式布局支持单元格的合并。

绘制网格式子图布局相关API：

```
import matplotlib.gridspec as mg
plt.figure('Grid Layout', facecolor='lightgray')
# 调用GridSpec方法拆分网格式布局
# rows: 行数
# cols: 列数
# gs = mg.GridSpec(rows, cols)  拆分成3行3列
gs = mg.GridSpec(3, 3)
# 合并0行与0、1列为一个子图表
plt.subplot(gs[0, :2])
plt.text(0.5, 0.5, '1', ha='center', va='center', size=36)
plt.show()
```

案例：绘制一个自定义网格布局。

```
import matplotlib.gridspec as mg
plt.figure('GridLayout', facecolor='lightgray')
gridsubs = plt.GridSpec(3, 3)
# 合并0行、0/1列为一个子图
plt.subplot(gridsubs[0, :2])
plt.text(0.5, 0.5, 1, ha='center', va='center', size=36)
plt.tight_layout()
plt.xticks([])
plt.yticks([])
```

自由式布局(很少使用)

自由式布局相关API：

```
plt.figure('Flow Layout', facecolor='lightgray')
# 设置图标的位置，给出左下角点坐标与宽高即可
# left_bottom_x: 坐下角点x坐标
# left_bottom_x: 坐下角点y坐标
# width:         宽度
# height:        高度
# plt.axes([left_bottom_x, left_bottom_y, width, height])
plt.axes([0.03, 0.03, 0.94, 0.94])
plt.text(0.5, 0.5, '1', ha='center', va='center', size=36)
plt.show()
```

案例：测试自由式布局，定位子图。

```
plt.figure('FlowLayout', facecolor='lightgray')

plt.axes([0.1, 0.2, 0.5, 0.3])
plt.text(0.5, 0.5, 1, ha='center', va='center', size=36)
plt.show()
```

2）散点图

可以通过每个点的坐标、颜色、大小和形状表示不同的特征值。

身高	体重	性别	年龄段	种族
180	80	男	中年	亚洲
160	50	女	青少	美洲

绘制散点图的相关API：

```
plt.scatter(
    x,                  # x轴坐标数组
    y,                  # y轴坐标数组
    marker='',          # 点型
    s=10,               # 大小
    color='',           # 颜色
    edgecolor='',       # 边缘颜色
    facecolor='',       # 填充色
    zorder=''           # 图层序号
)
```

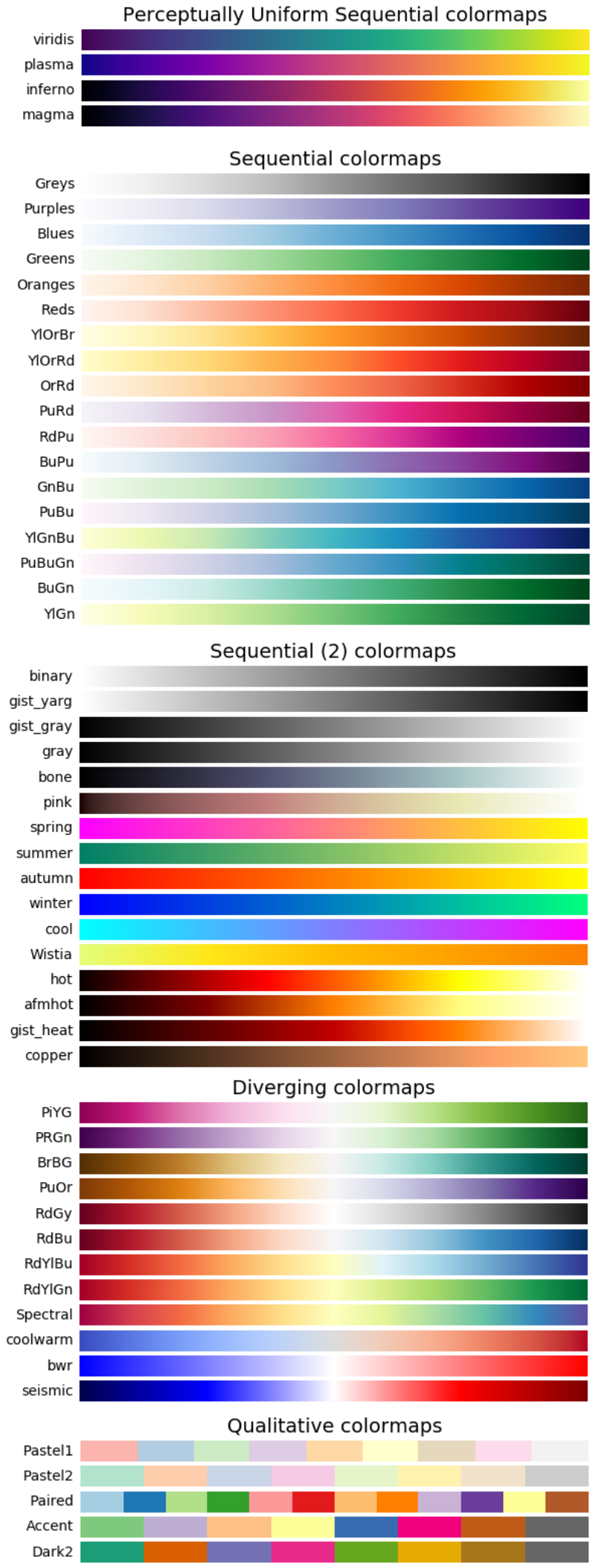
numpy.random提供了normal函数用于产生符合 正态分布 的随机数

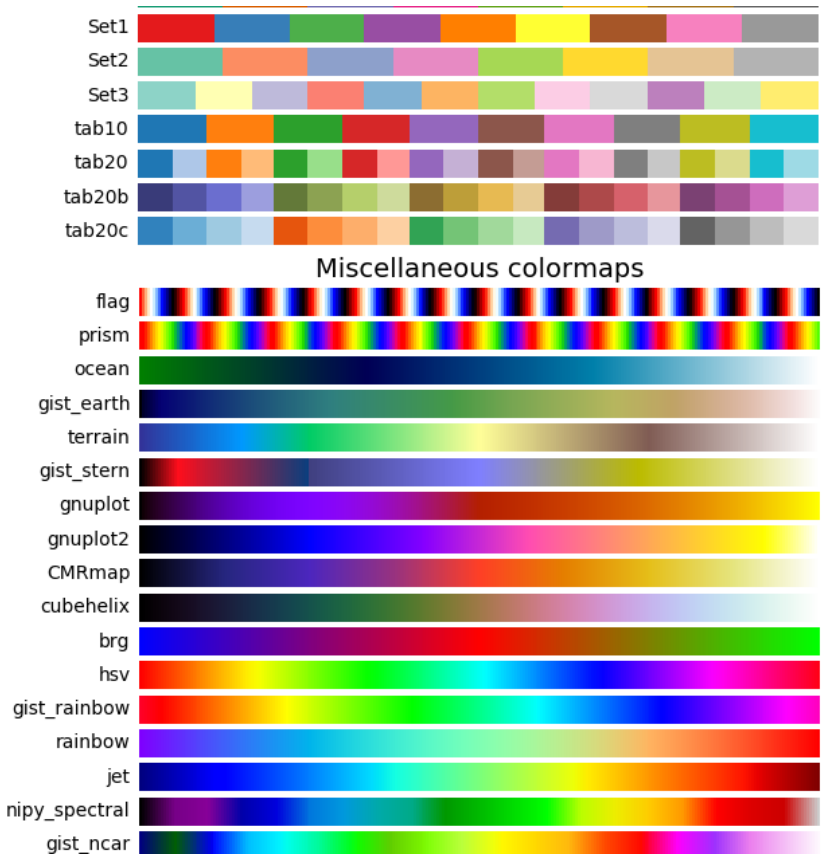
```
n = 100
# 172:  期望值
# 10:   标准差
# n:    数字生成数量
x = np.random.normal(172, 20, n)
y = np.random.normal(60, 10, n)
```

案例：绘制平面散点图。

```python
# 散点图示例
import matplotlib.pyplot as plt
import numpy as np

n = 40
# 期望值：期望值是该变量输出值的平均数
# 标准差：是反映一组数据离散程度最常用的一种量化形式，是表示精确度的重要指标
x = np.random.normal(172, 20 ,n ) # 期望值，标准差，生成数量
y = np.random.normal(60, 10, n) # 期望值，标准差，生成数量

x2 = np.random.normal(180, 20 ,n ) # 期望值，标准差，生成数量
y2 = np.random.normal(70, 10, n) # 期望值，标准差，生成数量

plt.figure("scatter", facecolor="lightgray")
plt.title("Scatter Demo")
plt.scatter(x, y, c="red", marker="D")
plt.scatter(x2, y2, c="blue", marker="v")

plt.xlim(100, 240)
plt.ylim(0, 100)
plt.show()
```

cmap颜色映射表参照附件：cmap颜色映射表


Perceptually Uniform Sequential colormaps
viridis
plasma
inferno
magma
Sequential colormaps
Greys
Purples
Blues
Greens
Oranges
Reds
YlOrBr
YlOrRd
OrRd
PuRd
RdPu
BuPu
GnBu
PuBu
YlGnBu
PuBuGn
BuGn
YlGn
Sequential (2) colormaps
binary
gist_yarg
gist_gray
gray
bone
pink
spring
summer
autumn
winter
cool
Wistia
hot
afmhot
gist_heat
copper
Diverging colormaps
PiYG
PRGn
BrBG
PuOr
RdGy
RdBu
RdYlBu
RdYlGn
Spectral
coolwarm
bwr
seismic
Qualitative colormaps
Pastel1
Pastel2
Paired
Accent
Dark2



3）填充

以某种颜色自动填充两条曲线的闭合区域。

```
plt.fill_between(
    x,              # x轴的水平坐标
    sin_x,          # 下边界曲线上点的垂直坐标
    cos_x,          # 上边界曲线上点的垂直坐标
    sin_x<cos_x,    # 填充条件，为True时填充
    color='',       # 填充颜色
    alpha=0.2       # 透明度
)
```

案例：绘制两条曲线： sin_x = sin(x) cos_x = cos(x / 2) / 2 [0-8π]

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

n = 1000
x = np.linspace(0, 8 * np.pi, n)  # 返回指定间隔上的等距数字

sin_y = np.sin(x)  # 计算sin函数值
cos_y = np.cos(x / 2) / 2  # 计算cos函数值

plt.figure('Fill', facecolor='lightgray')
plt.title('Fill', fontsize=20)
plt.xlabel('x', fontsize=14)  # x轴标签
plt.ylabel('y', fontsize=14)  # y轴
plt.tick_params(labelsize=10)  # 刻度
plt.grid(linestyle=':')

plt.plot(x, sin_y, c='dodgerblue', label=r'$y=sin(x)$')
plt.plot(x, cos_y, c='orangered', label=r'$y=\frac{1}{2}cos(\frac{x}{2})$')
```

```
# 填充cos_y < sin_y的部分
plt.fill_between(x, cos_y, sin_y, cos_y < sin_y, color='dodgerblue', alpha=0.5)
# 填充cos_y > sin_y的部分
plt.fill_between(x, cos_y, sin_y, cos_y > sin_y, color='orangered', alpha=0.5)

plt.legend()
plt.show()
```

4）条形图（柱状图）

绘制柱状图的相关API：

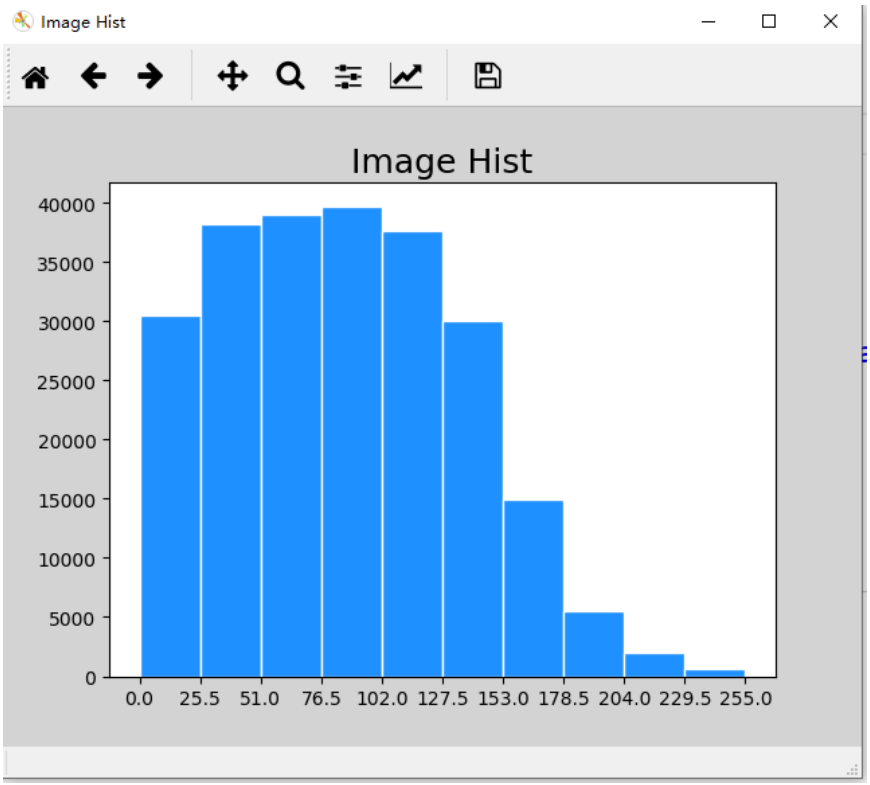
```
# 设置使中文显示完整
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False
plt.figure('Bar', facecolor='lightgray')
plt.bar(
    x,              # 水平坐标数组
    y,              # 柱状图高度数组
    width,          # 柱子的宽度
    color='',       # 填充颜色
    label='',       #
    alpha=0.2       #
)
```

案例：先以柱状图绘制苹果12个月的销量，然后再绘制橘子的销量。

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

apples = np.array([30, 25, 22, 36, 21, 29, 20, 24, 33, 19, 27, 15])
oranges = np.array([24, 33, 19, 27, 35, 20, 15, 27, 20, 32, 20, 22])

plt.figure('Bar', facecolor='lightgray')
plt.title('Bar', fontsize=20)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Price', fontsize=14)
plt.tick_params(labelsize=10)
plt.grid(axis='y', linestyle=':')
plt.ylim((0, 40))

x = np.arange(len(apples))  # 产生均匀数组，长度等同于apples

plt.bar(x - 0.2,  # 横轴数据
        apples,  # 纵轴数据
        0.4,  # 柱体宽度
        color='dodgerblue',
        label='Apple')
plt.bar(x + 0.2,  # 横轴数据
        oranges,  # 纵轴数据
        0.4,  # 柱体宽度
        color='orangered', label='Orange', alpha=0.75)

plt.xticks(x, ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct',
'Nov', 'Dec'])

plt.legend()
plt.show()
```

5）直方图

执行结果：



绘制直方图相关API：

```
plt.hist(
    x,                  # 值列表
    bins,               # 直方柱数量
    color,              # 颜色
    edgecolor           # 边缘颜色
)
```

案例：绘制统计直方图显示图片像素亮度分布：

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.misc as sm

img = sm.imread('../data/forest.jpg', True)
print(img.shape)

pixes = img.ravel()
plt.figure('Image Hist', facecolor='lightgray')
plt.title('Image Hist', fontsize=18)
plt.xticks(np.linspace(0, 255, 11))
plt.hist(x=pixes, bins=10, color='dodgerblue', range=(0, 255), edgecolor='white',
normed=False)
plt.show()
```

扩展：随机数模块与概率分布

numpy提供了random模块生成服从特定统计规律的随机数序列。

一组随机数可能呈现如下分布：

```
统计班级同学体重：[63.2, 76.5, 65.7, 68.9, 59.4 ... ]
统计班级同学身高：[163.2, 176.5, 165.7, 168.9, 159.4 ... ]
统计班级同学到班时间：['07:20:22','07:30:48','07:21:23','07:24:58' ...]
```

又或者呈现如下分布：

```
统计班级同学体重级别：[偏轻, 中等, 偏重, 超重, 中等, 偏重, 超重, 中等, 偏重...]
统计班级同学身高级别：[偏低, 中等, 中等, 中等, 中等, 偏高, 中等, 中等, 偏高...]
统计最近班级同学迟到人数（共10人）：[0, 1, 3, 0, 0, 1, 2, 0, 0, 0 ....]
```

二项分布（**binomial**）

二项分布就是重复n次独立事件的伯努利试验（Bernoulli experiment）。在每次试验中只有两种可能的结果，而且两种结果发生与否互相对立，并且相互独立，事件发生与否的概率在每一次独立试验中都保持不变，例如抛硬币。

```
# 产生size个随机数，每个随机数来自n次尝试中的成功次数，其中每次尝试成功的概率为p
np.random.binomial(n, p, size)
```

二项分布可以用于求如下场景的概率的近似值：

1. 某人投篮命中率为0.3，投10次，进5个球的概率。

```
sum(np.random.binomial(10, 0.3, 200000) == 5) / 200000
```

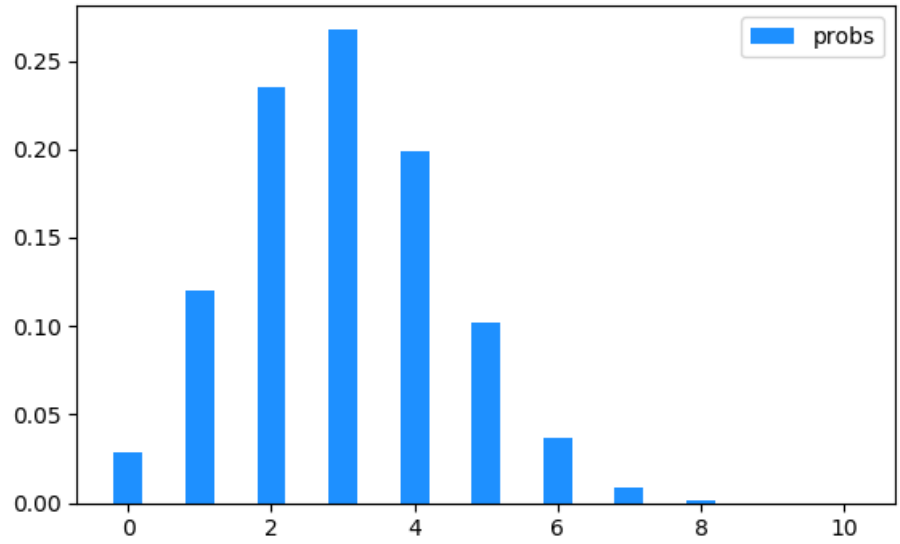
1. 某人打客服电话，客服接通率是0.6，一共打了3次，都没人接的概率。

```
sum(np.random.binomial(3, 0.6, 200000) == 0) / 200000
```

示例：模拟某人以30%命中率投篮，每次投10个，计算并打每种进球可能的概率

```
# 二项式分布示例
import numpy as np
import matplotlib.pyplot as mp

# binomial: 从二项分布中抽取样本
# n:尝试次数  p:概率
r = np.random.binomial(10, 0.5, 200000)
mp.hist(r, 11, edgecolor='white')
mp.legend()
mp.show()
```

执行结果：



超几何分布(hypergeometric)

超几何分布是统计学上一种离散概率分布。它描述了从有限N个物件（其中包含M个指定种类的物件）中抽出n个物件，成功抽出该指定种类的物件的次数（不放回）。以下是一组超几何分布的示例：

（1）10件产品中含有3件次品，从中任意取4件产品，所取出的次品件数服从超几何分布；
（2）袋中有8红球4白球，从中任意摸出5个球，摸出红球个数服从超几何分布；
（3）某班45个学生，女生20人，现从中选7人做代表，代表中所含女生的人数服从超几何分布；
（4）15张卡片中含有5件写有“奖”字，从中任意取3件产品，所取出的卡片中含有奖字的卡片张数服从超几何分布；

（5）10位代表中有5位支持候选人A，随机采访3人，其中支持候选人A的人数服从超几何分布；
（6）盘中装有10个粽子，豆沙粽2个，肉粽3个，白粽5个，从中任选3个，取到的豆沙粽的个数服从超几何分布。

API介绍：

```
# 产生size个随机数，每个随机数t为在总样本中随机抽取nsample个样本后好样本的个数，总样本由ngood个好样本和nbad个坏样本组成
np.random.hypergeometric(ngood, nbad, nsample, size)
```

示例一：从6个好苹果、4个坏苹果中抽取3个苹果，返回好球的数量（执行10次）

```
import numpy as np

# 从6个好球、4个坏球中抽取3个球，返回好球的数量（执行10次）
n = np.random.hypergeometric(6, 4, 3, 10)
print(n)
print(n.mean())
```

执行结果：

```
[2 2 3 1 2 2 1 3 2 2]
2.0
```

正态分布(normal)

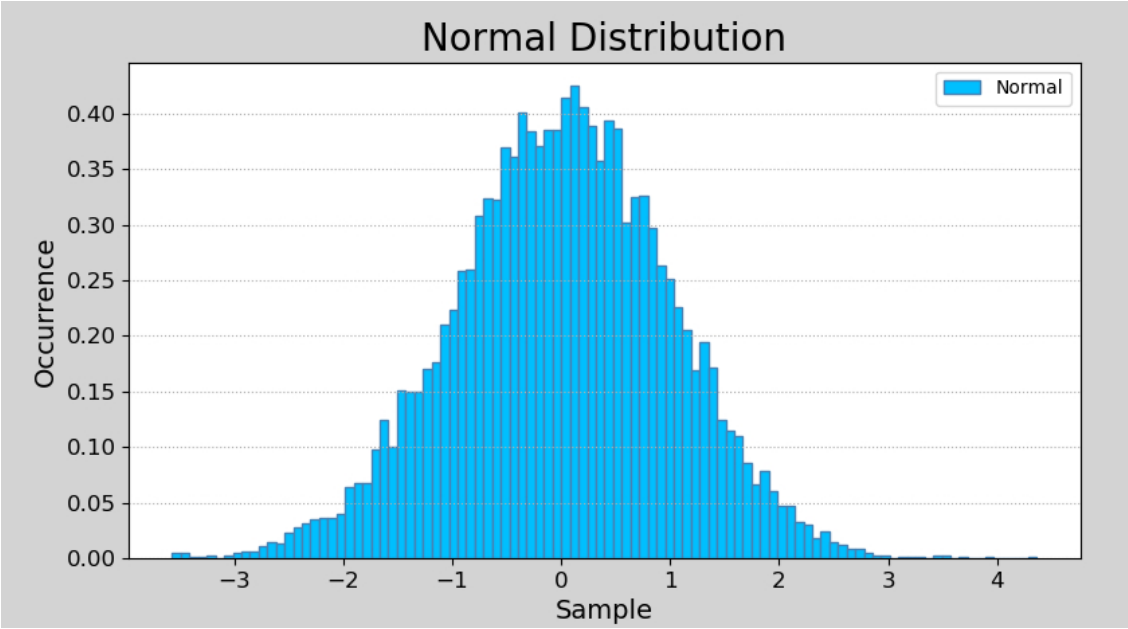
```
# 产生size个随机数，服从标准正态(期望=0, 标准差=1)分布。
np.random.normal(size)
# 产生size个随机数，服从正态分布(期望=1, 标准差=10)。
np.random.normal(loc=1, scale=10, size)
```

$$标准正态分布概率密度：\frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}$$

案例：生成10000个服从正态分布的随机数并绘制随机值的频数直方图。

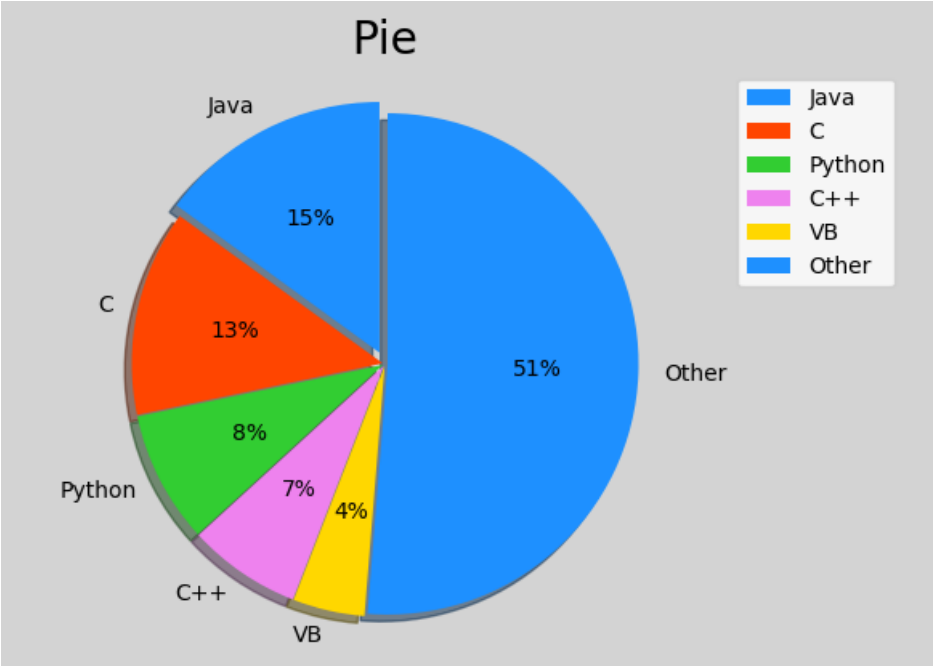
```
import numpy as np
import matplotlib.pyplot as mp

samples = np.random.normal(size=10000)

mp.figure('Normal Distribution',facecolor='lightgray')
mp.title('Normal Distribution', fontsize=20)
mp.xlabel('Sample', fontsize=14)
mp.ylabel('Occurrence', fontsize=14)
mp.tick_params(labelsize=12)
mp.grid(axis='y', linestyle=':')
mp.hist(samples, 100, edgecolor='steelblue',
        facecolor='deepskyblue', label='Normal')
mp.legend()
mp.show()
```

执行结果：



##

6）饼图



绘制饼状图的基本API：

```
plt.pie(
    values,          # 值列表
    spaces,          # 扇形之间的间距列表
    labels,          # 标签列表
    colors,          # 颜色列表
    '%d%%',          # 标签所占比例格式
    shadow=True,     # 是否显示阴影
    startangle=90    # 逆时针绘制饼状图时的起始角度
    radius=1         # 半径
)
```

案例：绘制饼状图显示6门编程语言的流行程度：

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

plt.figure('pie', facecolor='lightgray')
plt.title('Pie', fontsize=20)
# 整理数据
values = [15, 13.3, 8.5, 7.3, 4.62, 51.28]
spaces = [0.05, 0.01, 0.01, 0.01, 0.01, 0.01]
labels = ['Java', 'C', 'Python', 'C++', 'VB', 'Other']
colors = ['dodgerblue', 'orangered', 'limegreen', 'violet', 'gold','blue']
# 等轴比例
plt.axis('equal')
plt.pie(
    values,  # 值列表
    spaces,  # 扇形之间的间距列表
    labels,  # 标签列表
    colors,  # 颜色列表
    '%d%%',  # 标签所占比例格式
    shadow=True,  # 是否显示阴影
    startangle=90,  # 逆时针绘制饼状图时的起始角度
    radius=1  # 半径
)
plt.legend()
plt.show()
```

pandas可视化

基本绘图

Series数据可视化

Series提供了plot方法以index作为x，以value作为y，完成数据可视化：

```
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range('1/1/2000', periods=1000))
ts = ts.cumsum()
ts.plot()
```

DataFrame数据可视化

DataFrame提供了plot方法可以指定某一列作为x，某一列作为y，完成数据可视化：

```
df3 = pd.DataFrame(np.random.randn(1000, 2),
                   columns=['B', 'C']).cumsum()
df3['A'] = np.arange(len(df3))
df3.plot(x='A', y='B')
```

高级绘图

plot()方法可以通过kind关键字参数提供不同的图像类型，包括：

类型	说明
bar or barh	柱状图
hist	直方图
box	箱线图
scatter	散点图
pie	饼状图

相关API如下：

```
# 柱状图
series.plot.bar()
dataFrame.plot.bar()
dataFrame.plot.barh()
```

直方图

```
# 直方图
series.plot.hist(alpha=0.5, bins=5)
dataFrame.plot.hist(alpha=0.5, bins=5)
```

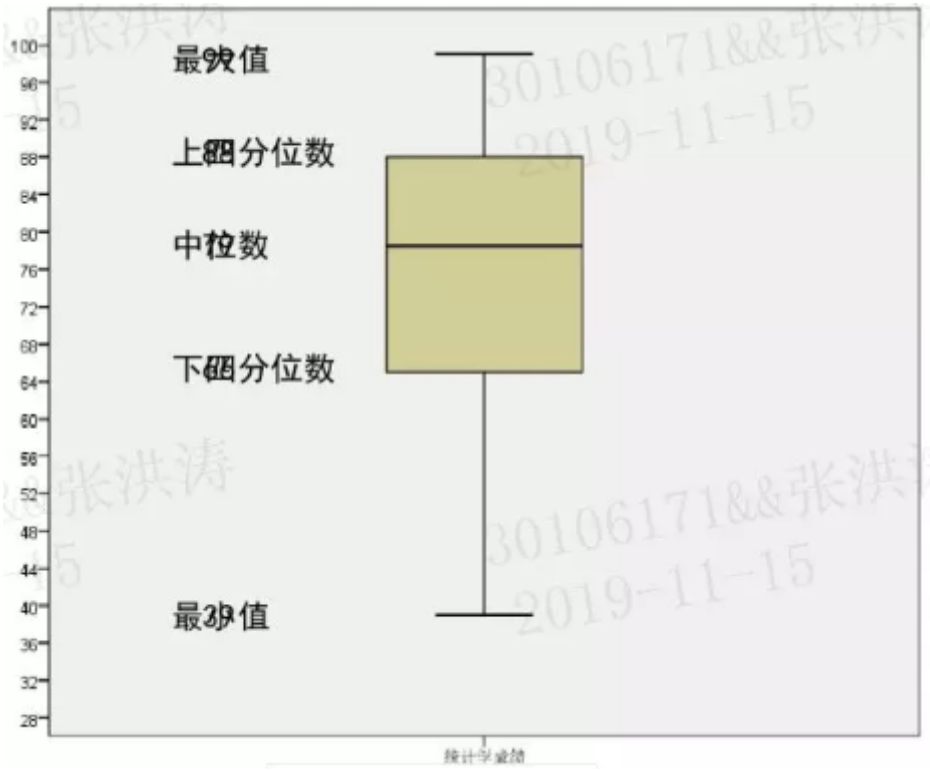
散点图

```
# 散点图
df.plot.scatter(x='a', y='b', c=col, colormap='');
```

饼状图

```
# 饼状图
series.plot.pie(figsize=(6, 6))
dataFrame.plot.pie(subplots=True, figsize=(6, 6), layout=(2, 2))
```

箱线图




```
# 箱线图
# 先找出一组数据的上边缘、下边缘、中位数和两个四分位数；然后， 连接两个四分位数画出箱体；再将上边缘和下边缘与箱体相连接，中位数在箱体中间
df.plot.box()
# 分组箱线图
df.boxplot(by='X')
```

箱线图反应一组数据的集中趋势，四分位数的差可以反映一组数据的离散情况：

1. 中位数高，表示平均水平较高；反之则表示平均水平较低。
2. 箱子短，表示数据集中；箱子长，表示数据分散。