**matplotlib编程模式**

**matplotlib的可视化结构**

Figure的对象与构造

FigureManager

|-Figure

|-Axes

|-Artist

|-Line2D

|-Patch

|-Legend

|-Text

|-\_BaseImage

|-Table

|-OffsetBox

|-AnnotationBbox

|-Collection

|-QuiverKey

* 说明：
  + 一般matplotlib使用FigureManager管理多个Figure（图表），这个对象我们一般不直接使用；
    - 直接调用全局函数可以
  + 每个Figure由一个或者多个坐标系Axes构成；
    - 每个坐标系由两个两个坐标轴构成
    - 坐标轴上面包含坐标刻度等
  + 在坐标系中提供绘制，每个被绘制的对象为Artist
    - Artist分成很多类型，形成丰富的图表元素。

In [ ]:

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

*# --------------------------*

*# Figure:Figure不建议使用构造器构造，而是使用pyplot提供的函数构造*

figure = plt.figure('窗体标题')

*# --------------------------*

*# 坐标轴*

ax = plt.Axes(

figure, *# 坐标轴所在图形*

[0, 0, 1, 1]) *# 坐标的区域*

*# --------------------------*

*# 图形*

line = plt.Line2D(

[0, 0.25, 0.5, 0.75, 1], *# 线条的所有点的X坐标*

[0, 1, 0, 1, 0]) *# 线条的所有点的Y坐标*

*# --------------------------*

*# Figure，Axes，Artist的容器关系 每个被绘制的对象为Artist*

ax.add\_artist(line)

figure.add\_axes(ax)

*# --------------------------*

*# 显示图形*

figure.show(warn=**False**)

*# 显式窗体*

plt.show()

**Axes提供直接绘制功能**

In [ ]:

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

*# --------------------------*

*# Figure:Figure不建议使用构造器构造，而是使用pyplot提供的函数构造*

figure = plt.figure('窗体标题')

*# --------------------------*

*# 坐标轴*

ax = plt.Axes(

figure, *# 坐标轴所在图形*

[0, 0, 1, 1]) *# 坐标的区域*

*# --------------------------*

*# 直接使用Axes提供的绘制函数 散点图*

ax.scatter(x=[0.1, 0.25, 0.5, 0.75], y=[0.1, 0.6, 0.5, 0.25], color='r', marker='.')

figure.add\_axes(ax)

*# --------------------------*

*# 显示图形*

figure.show(warn=**False**)

*# 显式窗体*

plt.show()

**绘制散点图**

In [ ]:

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

*# 数据挖掘模块*

**from** **sklearn** **import** datasets

data,target=datasets.load\_iris(return\_X\_y=**True**)

*# 1英寸(长度单位，等于2.54厘米*

figure = plt.figure('机器学习可视化',figsize=(6,4))

*# 位置0.1 0.1 0.8 0.8*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8], label='鸢尾花数据集')

*# 切换特征维度观察数据的线性可分性*

ax.scatter(x=data[0:50:,3], y=data[0:50,1],color='r',label='A类',marker='.')

ax.scatter(x=data[50:100,3], y=data[50:100,1],color='g',label='B类',marker='x' )

ax.scatter(x=data[100:150,3], y=data[100:150,1],color='b',label='C类',marker='v' )

ax.legend() *# 显示摘要*

plt.show()

**Figure的使用**

**构造Figure对象**

1. 构造器定义

**class** **matplotlib**.figure.Figure(

figsize=**None**,

dpi=**None**,

facecolor=**None**,

edgecolor=**None**,

linewidth=0.0,

frameon=**None**,

subplotpars=**None**,

tight\_layout=**None**,

constrained\_layout=**None**)

1. 构造器参数说明
   * |-figsize : 2元小数元组: rcParams["figure.figsize"]
     + |-英寸为单位的元组表示(width, height).
   * |-dpi : float: rcParams["figure.dpi"]
     + |-每英寸的点数（像素）
   * |-facecolor : 字符与4元元组: rcParams["figure.facecolor"]
     + |-设置的是Figure的patch(小区域块)的填充颜色facecolor.
   * |-edgecolor : 字符与4元元组: rcParams["figure.edgecolor"]
     + |-设置的是Figure的patch(小区域块)的边界颜色edgecolor.
   * |-linewidth : float
     + |-边框的线条宽度.
   * |-frameon : bool: rcParams["figure.frameon"]
     + |-True就绘制边框.
   * |-subplotpars : SubplotParams：rcParams["figure.subplot.\*"]
     + |-Subplot子图的参数.
   * |-tight\_layout : bool或者字典: rcParams["figure.autolayout"]
     + |-If False 就使用subplotpars.
     + |-If True 使用缺省的padding实现tight\_layout.
     + |-字典：使用key参数：pad, w\_pad, h\_pad, and rect实现tight\_layout.
   * |-constrained\_layout : bool：rcParams["figure.constrained\_layout.use"]
     + |-True：实现 constrained layout. 与tight\_layout类似,但比tight\_layout灵活.
     + |-该参数对subplot()与 subplot2grid()无效.

**Figure对象的属性与数据访问**

**属性**

* 上面的参数只有三个提供属性模式访问
  + |-number
  + |-dpi
  + |-frameon

**数据访问**

* Figure的属性都提供set/get函数设置与获取
  + set\_xxxx(value)
  + value=get\_xxxx()
  + |-get\_dpi()
    - |-set\_dpi(val)
  + |-get\_edgecolor()
    - |-set\_edgecolor(color)
  + |-get\_facecolor()
    - |-set\_facecolor(color)
  + |-get\_figheight()
    - |-set\_figheight(val, forward=True)
      * |-#forward用来控制：当设置尺寸过大，窗体是否自动调整大小。
  + |-get\_figwidth()
    - |-set\_figwidth(val, forward=True)
      * |-#forward用来控制：当设置尺寸过大，窗体是否自动调整大小。
  + |-get\_size\_inches()
    - |-set\_size\_inches(w, h=None, forward=True)
      * |-#forward用来控制：当设置尺寸过大，窗体是否自动调整大小。
  + |-get\_frameon()
    - |-set\_frameon(b)
  + |-get\_tight\_layout()
    - |-set\_tight\_layout(tight)
  + |-get\_constrained\_layout()
    - |-set\_constrained\_layout(constrained)
  + |-get\_constrained\_layout\_pads(relative=False）
    - |-set\_constrained\_layout\_pads(\*\*kwargs)

**Figure参数的rcParams设置：**

* Figure的参数还可以通过matplotlib.rcParams['xxxx']来设置

**使用figure函数创建Figure对象**

1. 函数定义如下

matplotlib.pyplot.figure(

num=**None**,

figsize=**None**,

dpi=**None**,

facecolor=**None**,

edgecolor=**None**,

frameon=**True**,

FigureClass=<**class** '**matplotlib**.figure.Figure'>,

clear=**False**, \*\*kwargs)

1. 函数的参数说明：
   * |- num : 整数或者字符串: None
     + |-Figure的内部标示ID，如果是字符串也会作为窗体标题
     + |-如果没有设置num，或者是整数，figure的ID会自动按序生成。
   * |- figsize : 2元小数元组: rcParams["figure.figsize"] = [6.4, 4.8]
     + |-窗体大小(英寸为单位)
   * |- dpi : 整数类型: rcParams["figure.dpi"] = 100
     + |-分辨率，每英寸的像素个数
   * |- facecolor :字符串或者4元元组: rcParams["figure.facecolor"] = 'w'
     + |-Figure的主体颜色
   * |- edgecolor :字符串或者4元元组: rcParams["figure.edgecolor"] = 'w'
     + |-Figure的边界颜色
   * |- frameon : bool:True
     + |-是否绘制Frame边框
       - |- FigureClass : Figure子类：Figure
     + |-一般指定figure函数创建Figure对象的类（必须是Figure的子类），默认是Figure
   * |- clear : bool: False
     + |-如果num指定的Figure存在，则清除。

**把Figure保存成图像**

* 可以使用savefig把Figure保存成图片
  + |- savefig(fname, \*, frameon=None, transparent=None, \*\*kwargs)

**Figure与FigureManager**

* 注意：
  + 除了在构造器外，Figure没有提供title，linewidth的设置方式
  + 阅读源代码知道，对窗体标题的处理有下面代码完成
    - |-figManager.set\_window\_title(figLabel)
    - |-figManager.canvas.figure.set\_label(figLabel) #这个是Figure的标签,来自Artist类
  + 获取figmanager可以使用pyplot函数实现
    - FigureManager=plt.get\_current\_fig\_manager()
      * |-返回对象类型：NSWindow
    - 如果不存在Figure，会自动返回，plt.figure一般会创建一个新的。所以该函数最好在Figure创建后调用

In [ ]:

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

*# ---------------------------------------------*

*# Figure*

figure = plt.figure(

num='Figure使用与管理', *# 窗体ID，字符串也是窗体标题*

figsize=(6.4, 4.8), *# 窗体大小（英寸）*

dpi=150, *# 分辨率*

frameon=**True**, *# 是否绘制边框*

linewidth=5 *# 来自Figure构造器的绘制变量的线条宽度*

)

FigureManager = plt.get\_current\_fig\_manager()

FigureManager.set\_window\_title('换一个窗体标题')

*# 当坐标轴填满Figure,上面的某些属性看不见效果，比如facecolor与edgecolor，linewidth*

*# 属性方式访问*

figure.dpi = 50

figure.number = 2

*# set/get方式访问*

*# ---------------------------------------------*

*# 坐标轴*

ax = plt.Axes(

figure, *# 坐标轴所在图形*

[0.1, 0.1, 0.8, 0.8]) *# 坐标的区域*

*# 图形*

line = plt.Line2D(

[0.25, 0.5, 0.75], *# 线条的所有点的X坐标*

[0.5, 0.5, 0.5]) *# 线条的所有点的Y坐标*

*# Figure，Axes，Artist的容器关系*

ax.add\_artist(line)

figure.add\_axes(ax)

*# 显示图形*

figure.show(warn=**False**)

*# 显式窗体*

plt.show()

**Axes坐标系**

* Axes坐标轴的使用与管理 ---- 之理解型add\_axes
  + 坐标轴通常包含Axis, Tick, Line2D, Text, Polygon等对象绘制完成,
  + 坐标轴的创建提供了两种方式：
    - |-Figure.add\_axes(\*args, \*\*kwargs) [理解型]
    - |-add\_subplot(\*args, \*\*kwargs) [快捷性]

**add\_axes创建坐标系**

* add\_axes函数使用的两种方式：
  + |-add\_axes(rect, projection=None, polar=False, \*\*kwargs)
    - |-[add\_axes函数利用参数构造Axes对象]
  + |-add\_axes(ax)
    - |-[调用者自己构造Axes对象]
* 上面两种方式本质都一样，因为参数都是一样的。

**创建Axes对象**

* Axes类型定义

.**class** **matplotlib**.axes.Axes(fig, rect, facecolor=**None**, frameon=**True**, sharex=**None**, sharey=**None**, label='', xscale=**None**, yscale=**None**, \*\*kwargs)

* 参数说明：
  + |-fig : Figure
    - |-坐标轴所属的Figure.
  + |-rect : [left, bottom, width, height]
    - |-坐标轴在Figure中的位置与大小
  + |-facecolor: 字符串与四元元组：None
    - |-周坐标轴填充色
  + |-frameon：bool：True
    - |-是否绘制坐标轴边框
  + |-sharex：Axes：None
    - |-使用sharex指定的坐标轴的属性
  + |-sharey：Axes：None
    - |-使用sharey指定的坐标轴的属性
  + |-label：：''
    - |-坐标轴的标签
  + |-xscale：字符串：None
    - |-x坐标比例尺的计算方式
    - |-取值：{"linear", "log", "symlog", "logit", ...}
  + |-yscale：字符串：None
    - |-y坐标比例尺的计算方式
    - |-取值：{"linear", "log", "symlog", "logit", ...}
* 还有其他属性可以在这儿使用。请参考官方文档。

**使用add\_axes直接创建**

* 函数定义： ```python

add\_axes(rect, projection=None, polar=False, \*\*kwargs)

```

* 参数说明：
  + |-rect : 四元元组或者列表
    - |-坐标轴位置与大小
  + |-projection : {None, 'aitoff', 'hammer', 'lambert', 'mollweide', 'polar', 'rectilinear', str},
    - |-坐标轴的类型
  + |-polar : boolean
    - |-使用极坐标，取值True等价于projection='polar'.
  + |-sharex, sharey : Axes
    - |-使用sharey指定的坐标轴的属性
  + |-label : str
    - |-坐标轴标签

还有其他属性可以在这儿使用。请参考官方文档。

**add\_subplot创建坐标系**

1. 函数定义：

add\_subplot(\*args, \*\*kwargs)

1. 该函数的所有属性基本上与add\_axes一样。可以创建axes，并指定位置，在坐标轴在figure中均匀分配，一共有三种使用方法
   * |-add\_subplot(nrows, ncols, index, \*\*kwargs)
     + |-nrows，坐标轴的行数（多个坐标轴）
     + |-ncols，坐标轴的列数（多个坐标轴）
     + |-index，坐标轴的位置（位置按照自然数计算，行：按照从上到下，列：按照从左到右）
     + |- \*\*kwargs 与add\_axes函数中参数一样，可以求助于文档
   * |-add\_subplot(pos, \*\*kwargs)
     + |-与上一种方法一样，只是当nrows，ncols， index不超过1位数的时候，可以合并成pos来表示
   * |-add\_subplot(ax)
     + |-与add\_axes一样
2. 例子代码

In [ ]:

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

figure = plt.figure('Figure的坐标轴管理', (5, 4))

*# ------------------------------------------*

ax1 = figure.add\_subplot(221, title='坐标轴1')

ax2 = figure.add\_subplot(2, 2, 2, title='坐标轴2')

ax3 = figure.add\_subplot(224, title='坐标轴4', position=[0.55, 0.05, 0.35, 0.3])

ax1.set\_xbound(lower=0, upper=10)

ax1.set\_ybound(lower=-1, upper=1)

line = plt.Line2D(

xdata=np.linspace(0, 10, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line2',

color=(1, 0, 0, 1)

)

ax1.add\_line(line)

*# 下面代码报错，多个坐标轴，不能公用一个Line2D对象*

*# ax2.add\_line(line)*

*# ax3.add\_line(line)*

*# ------------------------------------------*

plt.show()

In [ ]:

**Axes的属性**

* 属性太多，我们分成几个部分说明 Axes坐标轴的使用与管理 ---- 之坐标轴外观属性

下列属性使用方式

|-（1）可以在构造器

|-（2）创建函数参数

|-（3）属性访问（部分）

|-（4）set/get方法

|-（5）matplotlib.rcParams\['xxxx'\] 这个是全局，影响所有坐标轴

**外观基本属性**[**¶**](file:///C:\Users\11863\Desktop\02Axes%E4%B8%8E%E5%9B%BE%E5%BD%A2%E7%BB%98%E5%88%B6.html#外观基本属性)

|-facecolor：color

|-填充颜色

|-fc：color

|与facecolor一样

|-figure：Figure

|-所在Figure容器

|-frame\_on：bool

|-是否绘制边框

|-label：object

|-坐标轴的标签

|-position：[left, bottom, width, height] **or** Bbox

|-坐标轴的位置，大小

|-title：str

|-坐标轴标题

|-visible：bool

|-是否显示坐标轴

**外观x-坐标轴属性**

下面属性控制这坐标轴的范围，刻度的方式，标签，其中控制坐标轴的范围分成三组，相互之间有影响；

（xbound）（xlim）（xmargin+xscale）（xticks，xticklabels）

|-xbound：unknown

|-坐标轴范围

|-xlabel：str

|-坐标轴标签

|-xlim：(left: float, right: float)

|-坐标轴限制

|-xmargin：float greater than -0.5

|-坐标轴边界（中心点）

|-xscale：{"linear", "log", "symlog", "logit", ...}

|-坐标轴比例尺

|-xticklabels：List[str]

|-刻度标签

|-xticks：list

|-刻度

**外观y-坐标轴属性（与x-坐标轴一样）**

|-ybound：unknown

|-ylabel：str

|-ylim：(bottom: float, top: float)

|-ymargin：float greater than -0.5

|-yscale：{"linear", "log", "symlog", "logit", ...}

|-yticklabels：List[str]

|-yticks：list

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

figure = plt.figure(num='Axes坐标轴的属性')

*# 坐标轴*

ax = figure.add\_axes([0.2, 0.2, 0.6, 0.3])

*# 1.外观基本属性*

*# ---------------------------------------------*

ax.set\_fc((1, 0, 0))

*# 控制标题*

*# Axes.set\_title(label, fontdict=None, loc='center', pad=None, \*\*kwargs)*

ax.set\_title(

label='坐标轴标题', *# 显示的标题内容*

fontdict={ *# 可以使用Text所有的字体属性，也可以附加在kwargs参数后面。*

'fontsize': 10,

'fontweight': 900,

'fontstyle': 'italic',

'verticalalignment': 'baseline',

'horizontalalignment': 'center'}, *# 字体的相关属性*

loc='center', *# 取值{'center', 'left', 'right'}*

pad=10) *# 与坐标轴的间隔距离(单位是点 )*

*# \*\*kwargs 其他的文本属性（参考官方文档）*

*'''*

*字体*

*fontfamily {FONTNAME, 'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}*

*fontname {FONTNAME, 'serif', 'sans-serif', 'cursive', 'fantasy', 'monospace'}*

*fontproperties font\_manager.FontProperties*

*fontsize {size in points, 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}*

*fontstretch {a numeric value in range 0-1000, 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded', 'ultra-expanded'}*

*fontstyle {'normal', 'italic', 'oblique'}*

*fontvariant {'normal', 'small-caps'}*

*fontweight {a numeric value in range 0-1000, 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'}*

*'''*

ax.set\_visible(**True**) *# 设置坐标轴不可见*

*# Axes.set\_position(pos, which='both')*

ax.set\_position(

pos=[0.05, 0.25, 0.9, 0.5], *# 新的位置[left, bottom, width, height] or Bbox*

which='original' *# {'both', 'active', 'original'} 原来位置，还是绘制位置*

)

*# print(ax.get\_position(original=True)) # which通常下active与original都是一样的，除非使用set\_aspect改变方向*

*# ---------------------------------------------*

*# 2.外观x-坐标轴属性*

*# ---------------------------------------------*

*# Axes.set\_ylabel(ylabel, fontdict=None, labelpad=None, \*\*kwargs)[source]*

ax.set\_xlabel(

xlabel='X坐标', *# 标签的内容*

fontdict={ *# 控制标签的字体、位置等参数*

'fontsize': 12,

'fontweight': 'bold',

'fontstyle': 'italic',

'verticalalignment': 'baseline',

'horizontalalignment': 'center',

'color': (1, 0, 0, 1)

},

labelpad=30 *# 标签与x坐标轴的距离*

)

*# Axes.set\_xbound(lower=None, upper=None)*

ax.set\_xbound(

lower=-1, *# 坐标取值的范围（缺省是0-1）*

upper=1)

*# Axes.set\_xlim(left=None, right=None, emit=True, auto=False, \*, xmin=None, xmax=None)*

*# 设置坐标轴的范围，效果与xbound一样，只是使用方式的区别*

r = ax.set\_xlim(

left=-10,

right=10,

emit=**False**, *# 是否发出信号，告诉观察器，left与right参数是否改变*

auto=**True** *# ,*

) *# xmin=-1, xmax=1) 这两个参数未来会取消，与left与right作用一样*

print(r)

*# 下面两个属性结合在一起使用有效*

*# Axes.set\_xmargin(m)*

*# 设置边的大小: [0, 2],设置 0.1的结果是[-0.2, 2.2]，如果-0.1，则结果是[0.2, 1.8]*

ax.set\_xmargin(0.2)

*# Axes.set\_xscale(value, \*\*kwargs)*

ax.set\_xscale("linear")

*# print(ax.get\_xlim())*

*# Axes.set\_xticks(ticks, minor=False)*

ax.set\_xticks(

ticks=[0, 2, 3.5, 5, 7],

minor=**False**

)

*# Axes.set\_xticklabels(labels, fontdict=None, minor=False, \*\*kwargs)*

ax.set\_xticklabels(

labels=['|', 'x', 'y', 'z', 'w'],

fontdict={

'color': (0, 0, 1, 1)

}

)

*# ---------------------------------------------*

*# 图形*

line = plt.Line2D(

[0.25, 0.5, 0.75], *# 线条的所有点的X坐标*

[0.5, 0.5, 0.5]) *# 线条的所有点的Y坐标*

*# Figure，Axes，Artist的容器关系*

ax.add\_artist(line)

*# 显示图形*

figure.show(warn=**False**)

**Axis对象**

**Formatter**

操作最重要的对象Axis--Formatter

|-xaxis

|-Axes.get\_xaxis()返回XAxis对象

|-yaxis

|-Axes.get\_yaxis()返回YAxis对象

上述两个函数返回不同的对象

1.控制标签的格式化

|-NullFormatter

|-没有标签

|-IndexFormatter

|-从labels列表中取标签

|-FixedFormatter

|-从labels手动设置字符串

|-FuncFormatter

|-用函数的返回值设置标签

|-StrMethodFormatter

|-使用字符串format方法.

|-FormatStrFormatter

|-使用sprintf格式化方法.

|-ScalarFormatter

|-缺省的标量格式化

|-LogFormatter

|-log坐标的格式化

|-LogFormatterExponent

|-对数坐标格式化值指数为 exponent = log\_base(value).

|-LogFormatterMathtext

|-对数坐标格式化值指数为 exponent = log\_base(value)，使用数学文本

|-LogFormatterSciNotation

|-科学记数法的对数坐标值

|-LogitFormatter

|-概率格式化

|-EngFormatter

|-工程记号表示的标签格式化

|-前缀符号：

|-ENG\_PREFIXES = {-24: 'y', -21: 'z', -18: 'a', -15: 'f', -12: 'p', -9: 'n', -6: 'μ', -3: 'm', 0: '', 3: 'k', 6: 'M', 9: 'G', 12: 'T', 15: 'P', 18: 'E', 21: 'Z', 24: 'Y'}

|-PercentFormatter

|-百分格式化

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.ticker** **import** PercentFormatter

**from** **matplotlib.ticker** **import** EngFormatter

**from** **matplotlib.ticker** **import** FuncFormatter

figure = plt.figure(1, figsize=(4, 4))

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.4])

*# -------------------------------*

*# 1.PercentFormatter*

*# |-class matplotlib.ticker.PercentFormatter(xmax=100, decimals=None, symbol='%', is\_latex=False)*

xaxis = ax.get\_xaxis()

pfx = PercentFormatter(

xmax=10, *# 百分比的计算分母：输出值计算方式：x / xmax \* 100. 比如千分比，使用10*

decimals=2, *# 小数点维数：精度*

symbol='‰$a^2$', *# 符号*

is\_latex=**True**) *# 符号支持Latex语法*

xaxis.set\_major\_formatter(pfx)

*# 2.EngFormatter*

*# |-class matplotlib.ticker.EngFormatter(unit='', places=None, sep=' ')[source]*

yaxis = ax.get\_yaxis()

pfy = EngFormatter(

unit='Hz', *# 单位（后缀）*

places=1, *# 小数位数*

sep='\*') *# 前缀（常用符号见上面说明与文档说明）*

*# print(pfy.format\_eng(0.1)) # 100m*

yaxis.set\_major\_formatter(pfy)

ax2 = figure.add\_axes([0.1, 0.55, 0.8, 0.4])

*# 3.FuncFormatter*

*# class matplotlib.ticker.FuncFormatter(func)*

xaxis = ax2.get\_xaxis()

**def** my\_x\_formatter\_func(x, pos):

**if** x **and** pos:

*# print(type(x), x, type(pos), pos)*

**return** 'L:**%5.2f**(**%d**)' % (x, pos)

**else**:

**return** "o"

ff = FuncFormatter(my\_x\_formatter\_func)

xaxis.set\_major\_formatter(ff)

*# -------------------------------*

figure.show(warn=**False**)

*# plt.show()*

**Locator**

操作最重要的对象Axis--Locator

|-xaxis

|-Axes.get\_xaxis()返回XAxis对象

|-yaxis

|-Axes.get\_yaxis()返回YAxis对象

上述两个函数返回不同的对象

2.控制标签的位置

|-AutoLocator

|-自动定位器，大部分plot行为的默认定位器

|-MaxNLocator

|-计算出一个最佳的最大间隔，然后生成合适的位置.

|-LinearLocator

|-从最小到最大均匀计算定位

|-set\_params(numticks=**None**, presets=**None**)

|-tick\_values(vmin, vmax)

|-view\_limits(vmin, vmax)

|-LogLocator

|-从最小到最大按照对数计算定位.

|-MultipleLocator

|-按照倍数（可以整数与小数）计算定位

|-FixedLocator

|-固定的定位

|-IndexLocator

|-索引定位计算(e.g., where x = range(len(y))).

|-NullLocator

|-没有刻度.

|-SymmetricalLogLocator

|-按照对称对数计算定位

|-LogitLocator

|-按照分对数计算定位，计算公式log(x/1-x)

|-OldAutoLocator

|-选择一个乘法器，动态计算定位.

|-AutoMinorLocator

|-当坐标轴使线性并且主刻度是统一的空间，把大刻度按照4或者5等分均分的方式计算小刻度定位

*# 注意Locator可以用于Major，也可以用于Minor。*

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.ticker** **import** \*

figure = plt.figure(1, figsize=(4, 6))

*# -------------------------------*

ax1 = figure.add\_axes([0.1, 0.1, 0.8, 0.4])

*# class matplotlib.ticker.LinearLocator(numticks=None, presets=None)*

*# |-numticks=None 刻度个数*

*# |-presets=None*

*# 1.LinearLocator*

*# from matplotlib.ticker import LinearLocator*

ll = LinearLocator(numticks=6)

xaxis = ax1.get\_xaxis()

xaxis.set\_major\_locator(ll)

*# -------------------------------*

*# 2. AutoMinorLocator*

*# class matplotlib.ticker.AutoMinorLocator(n=None)*

*# |-n=None 设置等分的份数；默认4，5等分*

*# from matplotlib.ticker import AutoMinorLocator*

aml = AutoMinorLocator(5)

xaxis.set\_minor\_locator(aml)

*# -------------------------------*

*# 3.MultipleLocator*

*# class matplotlib.ticker.MultipleLocator(base=1.0)*

*# |-base=1.0 倍数*

*# from matplotlib.ticker import MultipleLocator*

mpl = MultipleLocator(0.15) *# 按照0.15的倍数设置刻度*

yaxis = ax1.get\_yaxis()

yaxis.set\_major\_locator(mpl)

yaxis.set\_minor\_locator(aml)

*# -------------------------------*

ax2 = figure.add\_axes([0.1, 0.55, 0.8, 0.4])

*# 4.IndexLocator*

*# class matplotlib.ticker.IndexLocator(base, offset)*

*# |-base*

*# |-offset*

*# 从plot等函数的数据中抽取标签刻度,取点计算公式是： (i-offset)%base*

*# from matplotlib.ticker import IndexLocator*

il = IndexLocator(0.2, 0.55) *# 不是按照下标计算的,而是按照data[0]~data[-1]从开始位置offset，按照base为等距计算。*

xaxis = ax2.get\_xaxis()

xaxis.set\_major\_locator(il)

ax2.plot(

[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9],

[0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]

)

*# -------------------------------*

*# 5.FixedLocator*

*# class matplotlib.ticker.FixedLocator(locs, nbins=None)*

*# |-locs 设置标签*

*# |-nbins(限制抽取的标签个数)ticks <= nbins +1*

*# |-其中采取的是抽样算法，从data中选取标签*

yaxis = ax2.get\_yaxis()

*# from matplotlib.ticker import FixedLocator*

fl = FixedLocator(

locs=[0, 0.1, 0.2, 0.47, 0.48, 0.5, 0.51, 0.52, 0.53, 0.56], nbins=10)

yaxis.set\_major\_locator(fl)

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**ticks**

操作最重要的对象Axis--Label

操作最重要的对象Axis--Locator

|-xaxis

|-Axes.get\_xaxis()返回XAxis对象

|-yaxis

|-Axes.get\_yaxis()返回YAxis对象

1.标签控制主要包含位置与文本（内容，文本样式等）

|-Axis.set\_label\_coords

|-设置标签的坐标系（就是坐标变换）

|-Axis.set\_label\_position

|-设置标签的位置

|-取值下面两个：{'top', 'bottom'}之一

|-Axis.set\_label\_text

|-设置标签的文本

|-Axis.get\_label\_position

|-返回标签的位置

|-Axis.get\_label\_text

|-返回标签的文本对象

2.tick

主刻度

|-Axis.get\_major\_ticks

|-Axis.get\_majorticklabels

|-Axis.get\_majorticklines

|-Axis.get\_majorticklocs

细刻度

|-Axis.get\_minor\_ticks

|-Axis.get\_minorticklabels

|-Axis.get\_minorticklines

|-Axis.get\_minorticklocs

通用属性

|-Axis.get\_offset\_text

|-Axis.get\_tick\_padding

|-Axis.get\_ticklabels

|-Axis.get\_ticklines

|-Axis.get\_ticklocs

|-Axis.get\_gridlines

|-Axis.grid

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

figure = plt.figure(1, figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.3, 0.8, 0.6])

*# ---------------------------------------------*

*# 1. labels也可以直接使用方法设置*

xaxis = ax.get\_xaxis()

xaxis.set\_label\_text(

label='x坐标轴',

fontdict={

'color': (1, 0, 0, 1)

})

xaxis.set\_label\_position('top')

*# -------------------------------*

*# 可以控制刻度与坐标轴的距离*

ticks = xaxis.get\_major\_ticks()

**for** tick **in** ticks:

tick.set\_pad(20) *# 刻度与x坐标轴的距离（单位像素）*

*# 控制刻度的文本样式*

labels = xaxis.get\_majorticklabels()

**for** label **in** labels:

label.set\_color((0, 1, 0, 1))

*# 控制刻度的线条样式*

lines = xaxis.get\_majorticklines()

**for** line **in** lines:

line.set\_color((0, 0, 1, 1))

line.set\_marker('v')

*# 线条的长度无法改变，Line2D的数据是1个点。*

*# 刻度的位置再这个返回，修改影响不了位置*

locs = xaxis.get\_majorticklocs()

locs[0] = 0.05

*# 控制坐标轴网格*

*# Axis.grid(b=None, which='major', \*\*kwargs)*

*# |- b : bool or None*

*# |- 是否显示网格线,如果设置了其他参数，默认就是True*

*# |- which : {'major', 'minor', 'both'}*

*# |-再哪儿显示网格线*

*# |- \*\*kwargs : Line2D properties*

*# |-网格线的属性，与Line2D的属性一样*

xaxis.grid(b=**True**, which='major', color='r', linestyle='--')

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**aspect**

设置x与y的坐标单位比例

|-Axes.set\_aspect(aspect, adjustable=**None**, anchor=**None**, share=**False**)

|-aspect : {'auto', 'equal'} **or** num

|-'auto'

|-使用Axes的区域，自动决定绘制区域的比例。

|-'equal'

|-绘制区域比例相等，等于num=1

|-num

|-设置坐标轴的绘制比例

注意：aspect与xlim与ylim是有关系的(如果不考虑adjustable)

|-aspect=1，xlim的长度为10，ylim的长度为1，则坐标轴就是10：1的比例绘制

|-aspect=10，xlim的长度为10，ylim的长度为1，则坐标轴就是1：1的比例绘制

|-adjustable : **None** **or** {'box', 'datalim'}, 可选

|-box:调整物理维度，调整绘制区域，达到aspect的值

|-物理区域可能有空余

|-datalim：调整x的limits或者y的limits，达到aspect的值。

|-物理区域总是充分利用的

|-anchor : **None** **or** str **or** 2-tuple of float, optional

|-锚点：这个参数在box才有效，因为根据datalim来计算比例，物理空间永远是填充满的，锚点效果不大。

|-'C' 中间

|-'SW' 西南角

|-'S' 南面中间

|-'SE' 东南

|-etc. 其他方向组合

|-share : bool, optional

|-是否使用改变到所有共享的坐标轴

|-Axes.get\_aspect()

|-返回字符串

**坐标翻转**

坐标轴翻转

|-翻转坐标轴

| |-Axes.invert\_xaxis()

| |-Axes.invert\_yaxis()

|-判定坐标轴是否翻转

| |-Axes.xaxis\_inverted()

| |-Axes.yaxis\_inverted()

**循环属性Cycler**

循环属性：

|-Axes.set\_prop\_cycle(\*args, \*\*kwargs)

该属性只对数组这样的数据有效，可以按照列反复使用循环属性。

其中有三种使用方式：

|-set\_prop\_cycle(cycler)

|-set\_prop\_cycle(label=values[, label2=values2[, ...]])

|-set\_prop\_cycle(label, values)

参数说明如下：

|-cycler : Cycler

|-cycler对象，可以使用函数构造

|-label : str

|-一个字符串，用来指定属性名

|-values : iterable

|-对应属性名的属性值（类型是可迭代类型）

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

**from** **cycler** **import** cycler

figure = plt.figure('循环属性', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_xbound(lower=0, upper=10) *# 会被plot等函数改变*

ax.set\_xlim(left=-2, right=10)

*# 生成a坐标（n,）*

x = np.linspace(0, 2 \* np.pi)

*# 相位(4,)*

phases = [np.pi\*0/2,

np.pi\*1/2,

np.pi\*2/2,

np.pi\*3/2]

*# 根据相位产生(n,4)的矩阵*

y = np.transpose(

[np.sin(x + phase) **for** phase **in** phases]

)

*# ------------------------------------------*

*# 属性设置在前后都无所谓，在最后绘制才有效。*

c = cycler(

color=[

(1, 0, 0, 1),

(0, 1, 0, 1),

(0, 0, 1, 1),

(1, 1, 0, 1)]

)

m = cycler(

marker=['+', 'v', '.', 'x']

)

*# Cycler 支持+ \* += \*=等运算*

*# 使用方式1：使用Cycle第对象*

*# ax.set\_prop\_cycle(c+m)*

*# 使用方式2：使用label=值*

*# ax.set\_prop\_cycle(color=['y', 'r', 'b', 'g'])*

*# 使用方式3：多个label属性设置*

ax.set\_prop\_cycle(

color=['y', 'r', 'b', 'g'],

marker=['.', 'v', '^', '+']

)

ax.plot(x, y)

*# ax.set\_xbound(lower=0, upper=10) # 绘制后设置bound才有效*

*# ------------------------------------------*

figure.show(warn=**False**)

plt.show()

**Artist对象**

**Line2D**

**绘制Line2D**

**class** **matplotlib**.lines.Line2D(

|-xdata,

|-绘制的x坐标，与下面的ydata对应形成坐标点

|-ydata,

|-绘制的y坐标，与上面的xdata对应形成坐标点

|-linewidth=**None**,

|-线条宽度

|-linestyle=**None**,

|-线条样式

|-实线 :'-' 或者 'solid'

|-虚线 :'--' 或者 'dashed'

|-虚点线 :'-.' 或者 'dashdot'

|-点线 :':' 或者 'dotted'

|-不绘制 :'None'

|-不绘制 :' '

|-不绘制 :''

|-定制虚线 :(offset, on-off-seq)

|-color=**None**,

|-线条颜色

|-marker=**None**,

|-节点标记（节点就是xdata与ydata构成的点，就是线条的连接点）

|-该值可以有很多字符串样式('.','v'等)，也可以使用整数编号(1-11)

|-markersize=**None**,

|-节点的大小

|-markeredgewidth=**None**,

|-节点宽度边界

|-markeredgecolor=**None**,

|-节点边界颜色（端点边界宽度大一点才效果明显）

|-markerfacecolor=**None**,

|-节点填充颜色（端点边界宽度大一点才效果明显）

|-markerfacecoloralt='none',

|-节点的可选颜色

|-当fillstyle填充分成2个部分的时候，可选颜色就有效果

|-fillstyle=**None**,

|-节点填充样式： {'full', 'left', 'right', 'bottom', 'top', 'none'}

|-antialiased=**None**,

|-控制线条的锯齿处理效果

|-dash\_capstyle=**None**,

|-虚线的端点样式(当线条很粗的时候，效果会明显点)

|- {'butt', 'round', 'projecting'}

|-solid\_capstyle=**None**,

|-实线的端点样式 (当线条很粗的时候，效果会明显点)

|- {'butt', 'round', 'projecting'}

|-dash\_joinstyle=**None**,

|- 虚线的链接样式（折线处的样式）(当线条很粗的时候，效果会明显点)

|-{'miter', 'round', 'bevel'}

|-solid\_joinstyle=**None**,

|- 实线的链接样式（折线处的样式）(当线条很粗的时候，效果会明显点)

|- {'miter', 'round', 'bevel'}

|-pickradius=5,

|-选取的检测半径

|-drawstyle=**None**,

|-绘制样式,缺省的是直接连接节点。

|-{'default', 'steps', 'steps-pre', 'steps-mid', 'steps-post'}

|-steps=steps-pre,点在前steps-pre，中steps-mid，后steps-post，

|-markevery=**None**,

|-对节点的标记方式，

|-every=**None**, 所有点都绘制.

|-every=N, 0开始，整除N的点都绘制

|-every=(start, N), 从start开始，整除N的点都绘制

|-every=slice(start, end, N)指定区间，间隔N的点.被绘制

|-every=[i, j, m, n], 指定位置的点被绘制.

|-every=0.1,指定间隔距离附近的点被绘制.

|-every=(0.5, 0.1) 从0.5开始，按照距离附件点被绘制

**Line2D的picker使用**

picker的使用

|-set\_picker(p)

|-picker两种方式：

|-**True**，用来交给Figure中的canvas的事件处理

|-float作用与pickradius一样

|-直接绑定一个处理函数，这个函数

|- Tuple[bool, dict] callable[[Artist, Event]]

|-set\_pickradius(d)

In [ ]:

*#!/usr/bin/python*

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

**import** **random**

figure = plt.figure(num='Line2D')

*# 坐标轴*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_xticks([x **for** x **in** range(-100, 101, 20)])

*# 显示图形*

*# ----------------------------------------------*

line = plt.Line2D(

xdata=[x **for** x **in** range(-100, 101, 20)],

ydata=[random.random() **for** x **in** range(-100, 101, 20)],

linewidth=1,

linestyle=(0, (1, 2, 2, 1)),

color=(1, 0, 0, 1),

marker='.',

markersize=20,

markeredgewidth=2,

markeredgecolor=(0, 1, 0, 1),

markerfacecolor=(0, 0, 1, 1),

markerfacecoloralt=(1, 0, 1, 1),

fillstyle='bottom',

antialiased=**True**,

dash\_capstyle='round',

solid\_capstyle='round',

dash\_joinstyle='bevel',

solid\_joinstyle='bevel',

pickradius=5,

drawstyle='steps-mid',

markevery=0.2,

label='Hello'

)

*# 2. picker*

*'''*

*# picker方式一*

*def pick\_line(event):*

*print(event)*

*line.set\_picker(True)*

*figure.canvas.mpl\_connect('pick\_event', pick\_line)*

*'''*

*# picker方式二*

**def** line\_picker(obj, mouseevent):

print(obj, mouseevent)

**return** **True**, {} *# 第一只返回bool，判定是否点击了Artist，第二个是一个数据属性字典*

line.set\_picker(line\_picker)

ax.add\_artist(line)

*# ----------------------------------------------*

figure.show(warn=**False**)

*# 显式窗体*

plt.show()

**Rectangle**

矩形图形绘制

**class** **matplotlib**.patches.Rectangle(xy, width, height, angle=0.0, \*\*kwargs)[source]

|-xy : (float, float)

|-width : float

|-height : float

|-angle : float, optional

|-fill : bool, optional

其他属性：

|-hatching

|-/ - diagonal hatching

|-\ - back diagonal

|-| - vertical

|-- - horizontal

|-+ - crossed

|-x - crossed diagonal

|-o - small circle

|-O - large circle

|-. - dots

|-\* - stars

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

figure = plt.figure(1, figsize=(4, 4))

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

*# -------------------------------*

rect = plt.Rectangle(

xy=(0.2, 0.2),

width=0.3,

height=0.3,

angle=30,

fill=**True**,

hatch='/',

snap=**True**)

ax.add\_artist(rect)

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**plot折线绘制**

矩形图形绘制--Axes除了使用add\_artist添加构造好的图形外，还可以使用一系列函数直接绘制

|-Axes.plot(\*args, scalex=**True**, scaley=**True**, data=**None**, \*\*kwargs)

|-返回：line数组

|-x, y : 标量或者向量

|-绘制线段的点坐标

|-不使用x=，或者y=的参数名字

|-fmt : str

|-用来控制线段的颜色与样式，格式是：fmt = '[color][marker][line]'

|-其中color，marker，line的取值可以参考官方文档

|-如果fmt只有颜色，颜色增加了16进制表示:*#RRGGBBAA*

|-data : 索引类型，通常是字典

|-设置data就会替代x，y（x，y就用来作为线条的标签）

|-data字典的key是x，y指定的标签

|-scalex, scaley : bool: **True**

|-设置为True，会根据x，y属性设置limits。

|-设置为False不会自动scale。（观察x，y轴随x、y参数改变的变化）

|-\*\*kwargs : Line2D的属性

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.markers** **import** MarkerStyle

figure = plt.figure(1, figsize=(4, 4))

ax = figure.add\_axes([0.05, 0.05, 0.9, 0.9])

*# 1.plot*

*# -------------------------------*

lines1 = ax.plot(

[0.2, 0.7],

[0.5, 0.5],

'#FF0000',

scalex=**False**,

scaley=**False**,

)

lines2 = ax.plot(

'x标签',

'y标签',

'#FF0000',

data={

'x标签': [0.2, 0.7],

'y标签': [0.7, 0.7]

},

scalex=**False**,

scaley=**False**,

)

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**scatter散点图**

|-Axes.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**, \*, data=**None**, \*\*kwargs)

返回paths

|-x, y : 形状为 (n, )的数组,

|-s : 标量 或者 形状为(n, )数组,

|-marker的大小，一般是点的2次方，缺省是rcParams['lines.markersize'] \*\* 2.

|-c : 颜色或者颜色序列,

|-marker : MarkerStyle(包含marker与markerstyle)

|-cmap : Colormap: **None**

|-颜色空间

|-norm : Normalize: **None**

|-当c是浮点数数组的时候使用，一把使用缺省值： colors.Normalize

|-vmin, vmax : scalar: **None**

|-与norm一起使用

|-alpha : scalar: **None**

|-linewidths : scalar 或者 array\_like: **None**

|-marker边界的线条宽度

|-edgecolors : color **or** sequence of color: 'face'

|-'face': 边界颜色与face颜色相同.

|-'none': 不绘制边界.

|-颜色与颜色序列.

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.markers** **import** MarkerStyle

figure = plt.figure(1, figsize=(4, 4))

*# 2.scatter*

*# -------------------------------*

ax2 = figure.add\_axes([0.05, 0.55, 0.9, 0.9])

ax2.set\_autoscale\_on(**False**)

ax2.scatter(

x=[0.3, 0.7],

y=[0.5, 0.8],

s=[100, 50],

c=[(1, 0, 0, 1), (0, 0, 1, 1)],

marker=MarkerStyle('v', 'full'),

)

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**bar柱状图**

|-Axes.bar(x, height, width=0.8, bottom=**None**, \*, align='center', data=**None**, \*\*kwargs)

返回：container : BarContainer

|-x : sequence of scalars

|-height : scalar **or** sequence of scalars

|-width : scalar **or** array-like:0.8

|-bottom : scalar **or** array-like

|-align : {'center', 'edge'}: 'center'

|-tick\_label : string **or** array-like

|-orientation : {'vertical', 'horizontal'}, optional

|-内部使用，如果使用horizontal，则请使用barh函数。

|-color : scalar **or** array-like

|-edgecolor : scalar **or** array-like

|-linewidth : scalar **or** array-like(像素)

|-xerr, yerr : scalar **or** array-like of shape(N,) **or** shape(2,N), optional

|-log : bool, optional, default: **False**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **matplotlib.markers** **import** MarkerStyle

figure = plt.figure(1, figsize=(4, 4))

ax2 = figure.add\_axes([0.05, 0.55, 0.9, 0.9])

*# 3.bar*

br = ax2.bar(

x=[0.2, 0.5, 0.8],

height=[0.2, 0.3, 0.4],

width=0.05,

bottom=[0.5, 0.1, 0.3],

align='edge',

tick\_label=['x', 'y', 'z'],

color=[(1, 0, 0, 1), (0, 1, 0, 1), (0, 0, 1, 1)],

edgecolor=[(1, 1, 0, 1), (0, 1, 1, 1), (1, 0, 1, 1)],

linewidth=[1, 2, 5], *# 边界线宽度*

xerr=[0.02, 0.02, 0.05], *# 误差横线*

yerr=[

[0.05, 0, 0.00], *# 误差竖线（上面部分）*

[0.05, 0.0, 0.0] *# 误差竖线（下面面部分）*

],

ecolor='b', *# 错误条的颜色*

*# orientation='horizontal' # 误差线方向*

)

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**pie饼图**

矩形图形绘制--Axes除了使用add\_artist添加构造好的图形外，还可以使用一系列函数直接绘制

|-Axes.pie(x, explode=**None**, labels=**None**, colors=**None**, autopct=**None**, pctdistance=0.6, shadow=**False**, labeldistance=1.1, startangle=**None**, radius=**None**, counterclock=**True**, wedgeprops=**None**, textprops=**None**, center=(0, 0), frame=**False**, rotatelabels=**False**, \*, data=**None**)

返回值：

|-patches : list

matplotlib.patches.Wedge对象的列表

|-texts : list

matplotlib.text.Text对象列表

|-autotexts : list

标签的Text对象，当autopct非None的时候返回.

参数说明：

|-x : array-like

|-饼图上的数据份额，最后自定转换成百分比。

|-explode : array-like, optional, default: **None**

|-控制饼图中，每个饼偏离中心的距离

|-labels : list, optional, default: **None**

|-每个饼对应的文字标签（可显示）

|-colors : array-like, optional, default: **None**

|-每个饼对应的颜色

|-autopct : **None** (default), string, **or** function, optional

|-现在在饼内的文本,如果有%字符格式，会自定填充饼的百分比。

|-pctdistance : float, optional, default: 0.6

|-autopct与中心点的距离

|-shadow : bool, optional, default: **False**

|-是否绘制阴影

|-labeldistance : float, optional, default: 1.1

|-标签到中心点的距离

|-startangle : float, optional, default: **None**

|-开始角度（单位：角度，非弧度）

|-radius : float, optional, default: **None**

|-饼的半径

|-counterclock : bool, optional, default: **True**

|-饼的绘制方向（顺时针False））

|-wedgeprops : dict, optional, default: **None**

|-使用字典设置边界属性

|-字典的key，需要参考wedge object对象的帮助文档(在patch模块Wedge类中)

|-textprops : dict, optional, default: **None**

|-文本属性，可以参考Text的官方文档。(在matplotlib.text.Text中有描述)

|-center : list of float, optional, default: (0, 0)

|-中心点（注意：frame=**True**，center改变不了比例尺）

|-frame : bool, optional, default: **False**

|-显示坐标轴

|-rotatelabels : bool, optional, default: **False**

|-对labels的文本进行对应的旋转。

注意:这个函数也支持data替代x。

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

figure = plt.figure(1, figsize=(4, 4))

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

*# 1. pie*

*# -------------------------------*

pe = ax.pie(

x=[3, 2, 2],

explode=[0.01, 0.01, 0.01],

labels=['第一个饼', '第二个饼', '第三个饼'],

colors=['r', 'g', 'b'],

autopct='份额:**%8.2f%%**',

pctdistance=1.3,

shadow=**False**,

labeldistance=0.5,

startangle=45,

radius=0.5,

counterclock=**False**,

wedgeprops={

'linewidth': 1,

'edgecolor': (1, 1, 0, 1),

'hatch': '.'

},

textprops={

'color': (0, 1, 1, 1)

},

center=(0.5, 0.5), *# 外观上位置都在窗体中间，可以设置为（500，500）但改变了坐标轴的参照点（可以使用光标观察坐标变化）*

frame=**True**,

rotatelabels=**True**

)

*# -------------------------------*

figure.show(warn=**False**)

plt.show()

**hist直方图**

- 直方图，也称频率图或者密度图，用来显示频率统计。

Axes.hist(

x,

|- (n,) array **or** sequence of (n,) arrays

|- 直方图数据（用来统计），类型是一维数组与一维数组序列

|- 直方图数据表示x位置,如果是数组序列，每个数组的长度可以不同。

bins=**None**,

|- int **or** sequence **or** str, optional

|- 整数表示箱子个数，类型可以是整数与序列或者字符串

range=**None**,

|- 绘制的范围，用来调整箱子的绘制范围，这个范围最好根据x坐标的范围一致。

density=**None**,

|- 类型True与False或者None,True表示概率，False使用统计计数表示

weights=**None**,

|- 每个数据的权重，类型与x一样，可以用来控制统计的结果,形状与x对应，weights与x的形状必须一致

cumulative=**False**,

|- 是否累加，类型是True与False，或者None

bottom=**None**,

|- 每个箱子的底边高度,类型是标量（统一高度）或者数组（对应每个箱子）

|- 注意：标量也要采用数组。比如[1]

histtype='bar',

|- 直方图类型,取值为：{'bar', 'barstacked', 'step', 'stepfilled'},

align='mid',

| -对齐方式，bar在箱子中的对齐方式,取值为：{'left', 'mid', 'right'},

orientation='vertical',

|- 直方图的绘制方向，取值为 {'horizontal', 'vertical'},

rwidth=**None**,

|- bar在箱子中的相对宽度，数据类型是：scalar **or** **None**，histtype类型为step与stepfilled无效

log=**False**,

|- x坐标采用log类型的scale,取值类型：bool, optional

color=**None**,

|- 颜色

label=**None**,

|- 显示在legend中的信息 ，类型维字符串

stacked=**False**,

|- 堆叠方式，类型是True与False

normed=**None**,

|- 已经不推荐使用，等价于density

\*,

data=**None**,

\*\*kwargs)

返回值：

n :

返回：array **or** list of arrays,表示统计数据列表

bins :

返回：array，箱子的边界位置

patches :

返回：list **or** list of lists，创建的Patch Artist(Bar)对象列表

'''

**一维数据直方图**

In [ ]:

% matplotlib inline

*#!/usr/bin/python*

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

**import** **random**

**import** **csv**

figure = plt.figure(num='直方图')

*# 坐标轴*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_xlim(0,12)

*# 显示图形*

*# ----------------------------------------------*

fields = ['工作年限', '学历', '职位', '薪水', '城市', '发布时间']

data = []

city = set()

**with** open('jobs.csv', 'r') **as** fd:

*# 1.2. 构建DictReader*

reader = csv.DictReader(fd, fieldnames=fields)

next(reader)

*# 1.3. 读取（顺便统计）*

**for** row **in** reader:

data.append(row['城市'])

city.add(row['城市'])

r = ax.hist(

x=data,

bins=len(city), *# 箱子个数*

density=**False**,

label='直方图',

range=(0, 12), *# 与城市个数相同，用来调整箱子绘制的宽度，最好与坐标范围一致ax.set\_xlim(0,12)*

cumulative=**False**, *# 累加*

bottom=[-1], *# 箱子的底边高,-1 表示绘制到y的-1位置,注意不能使用-1，必须是[]*

histtype='bar', *# {'bar', 'barstacked', 'step', 'stepfilled'},*

align='left', *# 微调bar的位置*

orientation='vertical', *# {'horizontal', 'vertical'} 一般没有必要使用horizontal横过来*

rwidth=0.5, *# histtype类型为step与stepfilled无效*

color='r',

stacked=**True**, *# 多个数据才有效果*

)

print(r) *# 返回的数据*

ax.legend()

*# ----------------------------------------------*

figure.show(warn=**False**)

*# 显式窗体*

plt.show()

**多维数据直方图**

In [ ]:

**Legend对象**

操作并管理Axes中的Legend。

1、绘制主题：

|-Axes.legend(\*args, \*\*kwargs)

返回matplotlib.legend.Legend对象

|-handles : sequence of Artist, optional

|-labels : sequence of strings, optional

|-loc : int **or** string **or** pair of floats, default: rcParams["legend.loc"] ('best' **for** axes, 'upper right' **for** figures)

|-bbox\_to\_anchor : BboxBase, 2-tuple, **or** 4-tuple of floats

|-ncol : integer

|-prop : **None** **or** matplotlib.font\_manager.FontProperties **or** dict

|-fontsize : int **or** float **or** {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}

|-numpoints : **None** **or** int

|-scatterpoints : **None** **or** int

|-scatteryoffsets : iterable of floats

|-markerscale : **None** **or** int **or** float

|-markerfirst : bool

|-frameon : **None** **or** bool

|-fancybox : **None** **or** bool

|-shadow : **None** **or** bool

|-framealpha : **None** **or** float

|-facecolor : **None** **or** "inherit" **or** a color spec

|-edgecolor : **None** **or** "inherit" **or** a color spec

|-mode : {"expand", **None**}

|-bbox\_transform : **None** **or** matplotlib.transforms.Transform

|-title : str **or** **None**

|-title\_fontsize: str **or** **None**

|-borderpad : float **or** **None**

|-labelspacing : float **or** **None**

|-handlelength : float **or** **None**

|-handletextpad : float **or** **None**

|-borderaxespad : float **or** **None**

|-columnspacing : float **or** **None**

|-handler\_map : dict **or** **None**

2、返回主题：

|-Axes.get\_legend()

|-返回Legend对象

|-Axes.get\_legend\_handles\_labels(legend\_handler\_map=**None**)

|-返回Legend句柄与标签

**Line2D主题**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

line1 = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(0, 1, 0, 1)

)

line2 = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line2',

color=(1, 0, 0, 1)

)

line3 = ax.plot(

np.linspace(0, 1, 20),

np.random.uniform(-1, 1, size=20),

'#0000FF',

label='Line3'

)

*# ax.add\_artist(line1)*

*# add\_artist不会自动显示label。*

ax.add\_line(line1)

ax.add\_line(line2)

*# 注意上面的添加顺序*

*# ---------------------------------------------*

lg = ax.legend()

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**Pie主题**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

pie = ax.pie([2, 3, 4], labels=['X', 'Y', 'Z'])

*# ---------------------------------------------*

lg = ax.legend()

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**混合主题**

In [ ]:

*#!/usr/bin/python*

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

ax.set\_xlim(-1, 1)

line1 = plt.Line2D(

xdata=np.linspace(-1, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(0, 1, 0, 1)

)

line2 = plt.Line2D(

xdata=np.linspace(-1, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line2',

color=(1, 0, 0, 1)

)

line3 = ax.plot(

np.linspace(-1, 1, 20),

np.random.uniform(-1, 1, size=20),

'#0000FF',

label='Line3'

)

*# ax.add\_artist(line1)*

*# add\_artist不会自动显示label。*

ax.add\_line(line1)

ax.add\_line(line2)

pie = ax.pie([2, 3, 4], labels=['X', 'Y', 'Z'], frame=**True**)

*# 注意上面的添加顺序*

*# ---------------------------------------------*

lg = ax.legend()

*# ax.set\_axis\_on()*

*# ax.set\_frame\_on(b=True) # 控制可见*

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**Legend属性**

操作并管理Axes中的Legend。

1、绘制主题：

|-Axes.legend(\*args, \*\*kwargs)

返回matplotlib.legend.Legend对象

|-handles : sequence of Artist, optional

|-labels : sequence of strings, optional

|-loc : int **or** string **or** pair of floats, : rcParams["legend.loc"]

指定主题的位置，可以使用字符串：codes代码。也可以使用二元的绝对坐标

|-loc='lower right'

|-loc=4

|-loc=(0.9,0.1)

上述三种方式都在下右位置

当使用坐标形式的时候，bbox\_to\_anchor失效。

|-('best' **for** axes, 'upper right' **for** figures)

|-'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

|-bbox\_to\_anchor : BboxBase, 2-tuple, **or** 4-tuple of floats

|-绝对控制主题的位置，前提是loc不要使用浮点数坐标形式

|-loc用来控制在一个区域中的位置方式

|-bbox\_to\_anchor用来确定一个区域，默认是axis或者figure的坐标系

|-loc='upper center', bbox\_to\_anchor=(0.5,0.5),表示以（0.5，0.5）为锚点的上中位置（既主题的下中点就在（0.5，0.5））

|-loc与锚点之间略有一点间隔

|-ncol : integer

|-控制列数

|-prop : **None** **or** matplotlib.font\_manager.FontProperties **or** dict

|-字体控制

|-family: A list of font names **in** decreasing order of priority. The items may include a generic font family name, either 'serif', 'sans-serif', 'cursive', 'fantasy', **or** 'monospace'. In that case, the actual font to be used will be looked up **from** **the** associated rcParam.

|-style: Either 'normal', 'italic' **or** 'oblique'.

|-variant: Either 'normal' **or** 'small-caps'.

|-stretch: A numeric value **in** the range 0-1000 **or** one of 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' **or** 'ultra-expanded'

|-weight: A numeric value **in** the range 0-1000 **or** one of 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'

|-size: Either an relative value of 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' **or** an absolute font size, e.g., 12

|-fontsize : int **or** float **or** {'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large'}

|-numpoints : **None** **or** int

|-在主题中控制marker的显示格式（就是主题中的图形也显示marker，如果线条设置了marker得话）也可以使用参数设置：rcParams["legend.numpoints"]

|-scatterpoints : **None** **or** int

|-在主题中显示散点的marker数量：，可以使用rcParams["legend.scatterpoints"]设置

|-scatteryoffsets : iterable of floats

|-控制散点在主题中的垂直偏离位置，缺省值是[0.375, 0.5, 0.3125]

|-0与主题文本对齐，1上对齐，0.5同一高度

|-markerscale : **None** **or** int **or** float

|-marker的方法比例，可以使用rcParams["legend.markerscale"]设置

|-markerfirst : bool

|-控制marker绘制的啥位置，True在左，False在右

|-frameon : **None** **or** bool

|-控制边框的绘制，True就绘制

|-可以使用 rcParams["legend.frameon"].设置

|-fancybox : **None** **or** bool

|-与frameon一起使用才有效果，就是控制是否圆角矩形

|- rcParams["legend.fancybox"].

|-shadow : **None** **or** bool

|-rcParams["legend.shadow"]

|-framealpha : **None** **or** float

|-控制主题的背景透明度

|-rcParams["legend.framealpha"]

|-facecolor : **None** **or** "inherit" **or** a color spec

|-rcParams["legend.facecolor"]

|-因为继承，也会从rcParams["axes.facecolor"]获取

|-edgecolor : **None** **or** "inherit" **or** a color spec

|-rcParams["legend.edgecolor"],也可以设置 rcParams["axes.edgecolor"]

|-mode : {"expand", **None**}

|-绘制的区域模式,在水平方向填充

|-bbox\_transform : **None** **or** matplotlib.transforms.Transform

|-bbox\_to\_anchor的坐标系

|-title : str **or** **None**

|-标题

|-title\_fontsize: str **or** **None**

|-标题的字体

|-borderpad : float **or** **None**

|-边界空白(单位按照字体大小度量,1表示一个字符宽度或者高度)

|- rcParams["legend.borderpad"]

|-labelspacing : float **or** **None**

|-主题中标签的间隔(单位按照字体大小度量,1表示一个字符宽度或者高度)

|-rcParams["legend.labelspacing"]

|-handlelength : float **or** **None**

|-rcParams["legend.handlelength"]

|-handle的长度（单位按照字体大小度量）

|-handletextpad : float **or** **None**

|-rcParams["legend.handletextpad"].

|-控制handle与标签文本间的距离（单位按照字体大小度量）

|-borderaxespad : float **or** **None**

|-rcParams["legend.borderaxespad"].

|-坐标轴与主题之间的距离（单位按照字体大小度量）

|-columnspacing : float **or** **None**

|-rcParams["legend.columnspacing"]

|-多列之间的距离（单位按照字体大小度量）

|-handler\_map : dict **or** **None**

|-可以使用其他对象映射到handle，产生不同的handle效果

2、返回主题：

|-Axes.get\_legend()

|-返回Legend对象

|-Axes.get\_legend\_handles\_labels(legend\_handler\_map=**None**)

|-返回Legend句柄与标签

In [ ]:

*#!/usr/bin/python*

*# coding=utf-8*

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

ax.set\_xlim(-1, 1)

line1 = plt.Line2D(

xdata=np.linspace(-1, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(1, 0, 0, 1),

marker='.'

)

line2 = plt.Line2D(

xdata=np.linspace(-1, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line2',

color=(0, 1, 0, 1),

marker='v'

)

ax.add\_artist(line1)

ax.add\_line(line2)

*# ---------------------------------------------*

*# 用法一：自动映射标签（根据添加的顺序自动映射）*

*# lg1 = ax.legend(labels=['x', 'y'])*

*# 用法二：定制映射标签*

*# lg2 = ax.legend(handles=[line1, line2], labels=['x', 'y'])*

*# 常见属性使用：*

lg2 = ax.legend(

handles=[line1, line2],

labels=['x', 'y'], *# 定制标签，默认是Artist的label属性提供的标签，调用get\_label获取*

loc='upper left', *# 标签的为位置，三种方式4, 'lower right' (0.85, 0.00)*

bbox\_to\_anchor=(0.5, 0.5, 0.5, 0.5), *# 控制位置的锚点与区域，位置：(x, y)说明loc以改点为锚点，位置与大小：(x, y, width, height)说明loc再该区域来设置位置。*

ncol=1, *# 主题的列数(默认一列，行数自动计算)*

prop={ *# 控制主题的字体*

'size': 9

},

numpoints=3, *# 主题中显示marker的个数，前提是Line2D需要设置marker*

markerfirst=**False**, *# marker在右，默认在左*

frameon=**True**,

fancybox=**True**, *# 与frameon一起使用才有效果，就是控制是否圆角矩形*

mode=**None**, *# "expand", # x方向扩展填充整个区域*

title='主题',

borderpad=1, *# 边界宽度，以一个字大小为单位*

labelspacing=2, *# 显示内容见的间隔,以一个字大小为单位*

handletextpad=4, *# 控制handle与text的距离*

borderaxespad=0 *# 控制坐标轴与主题之间的间距*

)

ax.grid(b=**True**, which='both') *# 为了更好显示loc与bbox\_to\_anchor的结合控制位置*

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**标注与文本**

操作并管理Text与Annotations。

|-Axes.annotate

|-使用文本标注点xy.

|-Axes.text

|-添加文本到坐标轴

|-Axes.table

|-添加表格到坐标轴

|-Axes.arrow

|-添加箭头到坐标轴

|-Axes.inset\_axes

|-添加内部子坐标轴到坐标轴

|-Axes.indicate\_inset

|-添加内部标示器到坐标轴

|-Axes.indicate\_inset\_zoom

|-添加内部标示器到坐标轴，标示器被限制在一个内部坐标轴

**annotate标注**

1.annotate

|-Axes.annotate(s, xy, \*ags, \*\*kwargs)

|-s : str

|-标注的文本

|-xy : (float, float)

|- 标注目标的位置

|-xytext : (float, float), optional

|-标准文本的位置

|-xycoords : str, Artist, Transform, callable **or** tuple, optional

|-xy所在的坐标系

|-textcoords : str, Artist, Transform, callable **or** tuple, optional

|-文本所在坐标系

|-arrowprops : dict, optional

|-从xytext到xy的箭头属性，该属性存在，箭头就存在

|-如果在arrowprops中使用arrowstyle，下面几个属性失效:

|-width 箭头线条宽度

|-headwidth 头部宽度

|-headlength 头部长度

|-shrink 端点之间总长度的收缩因子

|-annotation\_clip : bool **or** **None**, optional

|-标注是否裁剪

|-\*\*kwargs

|-其他文本属性

----------------------

可思议使用 FancyArrowPatch 的属性：

|-arrowstyle the arrow style

|-connectionstyle the connection style

|-relpos default **is** (0.5, 0.5)

|-patchA default **is** bounding box of the text

|-patchB default **is** **None**

|-shrinkA default **is** 2 points

|-shrinkB default **is** 2 points

|-mutation\_scale default **is** text size (**in** points)

|-mutation\_aspect default **is** 1.

|-还可以使用matplotlib.patches.PathPatch的属性（参考文档）

----------------------

arrowstyle

|-'-' **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

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

line = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(1, 0, 0, 1)

)

ax.add\_line(line)

ax.legend()

*# ---------------------------------------------*

*# 1. Annotation*

ax.annotate(

s='标注', *# 标注的文本内容:必须*

xy=(0.9, 0.5), *# 坐标位置：必须参数，单位按照坐标单位*

xytext=(0.1, 0.1),

arrowprops={

'width': 2, *# 箭头线条粗细*

'headwidth': 5,

'headlength': 5,

'shrink': 0.1, *# 头尾的收缩因子(0-1之间)，该因子会改变线段长度*

'connectionstyle': 'angle3' *# 如果要定制链接类型，可以使用ConnectionStyle对象*

}

)

ax.annotate(

s='标注', *# 标注的文本内容:必须*

xy=(0.9, 0.1), *# 坐标位置：必须参数，单位按照坐标单位*

xytext=(0.1, 0.9),

arrowprops={

'arrowstyle': 'wedge', *# 使用该属性，就不能使用width等几个属性。*

'connectionstyle': 'angle3'

},

color=(0, 1, 0, 1)

)

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**text文本**

2. Text

|-Axes.text(x, y, s, fontdict=**None**, withdash=**False**, \*\*kwargs)

|-x, y : scalars

|-文本位置

|-s : str

|-文本内容

|-fontdict : dictionary, optional, default: **None**

|-文本的字体控制

|-withdash : boolean, optional, default: **False**

|-使用dash

|-\*\*kwargs : Text properties.

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

line = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(1, 0, 0, 1)

)

ax.add\_line(line)

ax.legend()

*# 2 .Text*

t1 = ax.text(

x=1,

y=0,

s='Hello World',

fontdict={

'size': 20

},

withdash=**True** *# 使用Line2D绘制字体,返回TextWithDash(0.5,0.5,'Hello')对象*

)

t1.set\_dashlength(100) *# 像素单位*

t1.set\_dashdirection(0) *# dash的方向0在前，1在后*

t1.set\_dashrotation(45) *# 旋转，单位是度。 (以文本中心旋转)*

t1.set\_dashpad(10) *# dash 与文本间宽度*

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**arrow箭头**

3. Arrow:

|-Axes.arrow(x, y, dx, dy, \*\*kwargs)

|-\*\*kwargs

|- FancyArrow 对象的构造器参数都可以使用,Patch的属性等，参考文档

构造器还允许使用的参数

Constructor arguments

|- width: float (default: 0.001)

|-width of full arrow tail

|-length\_includes\_head: bool (default: **False**)

|-**True** **if** head **is** to be counted **in** calculating the length.

|-head\_width: float **or** **None** (default: 3\*width)

|-total width of the full arrow head

|-head\_length: float **or** **None** (default: 1.5 \* head\_width)

|-length of arrow head

|-shape: ['full', 'left', 'right'] (default: 'full')

|-draw the left-half, right-half, **or** full arrow

|-overhang: float (default: 0)

|-fraction that the arrow **is** swept back (0 overhang means triangular shape). Can be negative **or** greater than one.

|-head\_starts\_at\_zero: bool (default: **False**)

|-**if** **True**, the head starts being drawn at coordinate 0 instead of ending at coordinate 0.

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

line = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(1, 0, 0, 1)

)

ax.add\_line(line)

ax.legend()

*# ---------------------------------------------*

ax.arrow(

x=0,

y=0,

dx=0.5,

dy=0.5,

width=0.001,

length\_includes\_head=**True**,

head\_width=0.03,

overhang=0.1,

head\_starts\_at\_zero=**False**

)

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**table表格**

4.table

返回 matplotlib.table.Table 对象

|-Axes.table(\*\*kwargs)

该函数调用的是matplotlib.table.table函数

|-**class** **matplotlib**.table.Table(ax, loc=**None**, bbox=**None**, \*\*kwargs)

|-matplotlib.table.table(cellText=**None**, cellColours=**None**, cellLoc='right', colWidths=**None**, rowLabels=**None**, rowColours=**None**, rowLoc='left', colLabels=**None**, colColours=**None**, colLoc='center', loc='bottom', bbox=**None**, edges='closed')

|-必须提供cellText **or** cellColours

|-loc取值

|-best

|-upper right

|-upper left

|-lower left

|-lower right

|-center left

|-center right

|-lower center

|-upper center

|-center

|-top right

|-top left

|-bottom left

|-bottom right

|-right

|-left

|-top

|-bottom

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

line = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(1, 0, 0, 1)

)

ax.add\_line(line)

ax.legend()

*# ---------------------------------------------*

*# 4. table*

ax.table(

cellText=[

['1', '2'],

['3', '4']],

cellColours=[

['r', 'b'],

['g', 'w']],

loc='bottom left', *# 位置，*

bbox=[0.5, 0.5, 0.3, 0.3] *# 配合loc使用控制表格的位置*

)

*# ---------------------------------------------*

figure.show(warn=**False**)

plt.show()

**inset\_axes 内部坐标系**

5.inset Axes

*# 该函数在matplotlib3.0使用的，请更新下matplotlib模块*

|-pip install matplotlib -U

|-Axes.inset\_axes(bounds, \*, transform=**None**, zorder=5, \*\*kwargs)[source]

|-bounds : [x0, y0, width, height]

|-transform : Transform

|-zorder : number

|-\*\*kwargs

|-axes.Axes的属性，用来控制位置

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **numpy** **as** **np**

figure = plt.figure('Legend使用', figsize=(5, 4))

*# 可以直接在add\_axes函数中设置*

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

ax.set\_ylim(-1, 1)

line = plt.Line2D(

xdata=np.linspace(0, 1, 20),

ydata=np.random.uniform(-1, 1, size=20),

label='Line1',

color=(1, 0, 0, 1)

)

ax.add\_line(line)

ax.legend()

*# ---------------------------------------------*

*# 5. inset Axes(坐标轴中嵌套坐标轴)*

in\_ax = ax.inset\_axes(

bounds=[0.8, 0.1, 0.15, 0.15]

)

in\_ax.plot(

[0.1, 0.9],

[0.5, 0.5],

color='r'

)

*# ---------------------------------------------*

figure.show(warn=**False**)

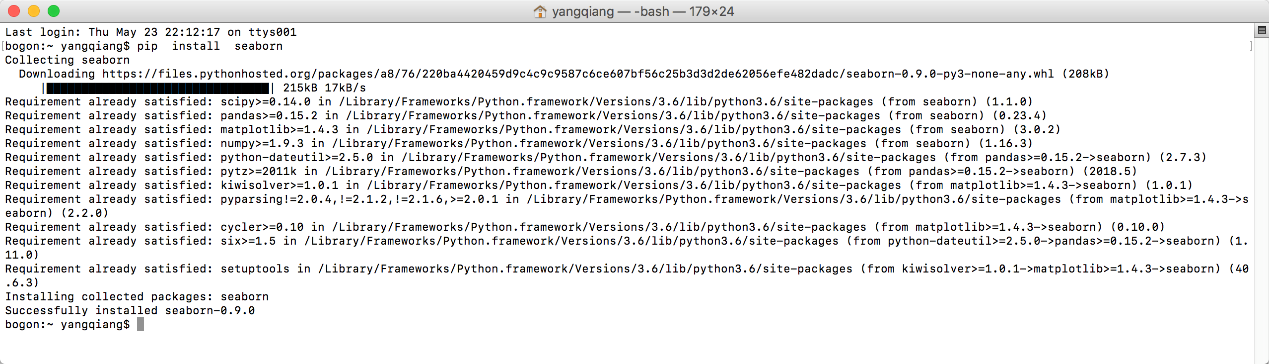
plt.show()

**matplotlib的可视化方式**

* matplotlib实际上有多种使用方式：
  + 直接使用原生的matplotlib；
  + 在Pandas中对matplotlib的封装；
    - 用于简单的统计数据可视化
  + matplotlib的高级封装Seaborn；
    - 用于统计数据可视化
* 使用场景：
  + Web应用
  + 桌面应用
  + 移动端应用等。

**Seaborn 入门与编程模式**

**安装**

* Python的安装还是那么简单
  + pip install seaborn -

**Seaborn的编程模式**

* Seaborn的优点：
  + 就是绘制简单；
  + 更多更简单的样式控制

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns** *#加载会导致matplotlib的默认作图风格变成seaborn*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:10]

*# 直接绘制*

sns.lineplot(data.index,data['price\_change'])

*# 显示*

plt.show()

**Seaborn的优点体验**

**样式的优点**

In [ ]:

%**matplotlib** inline

**import** **numpy** **as** **np**

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

sns.set() *# 可以注释这段代码体验默认样式，当然样式的设置不仅仅这么简单，也可以这么简单*

*# 样式主要包含主题与颜色*

figure = plt.figure('图形', figsize=(12,3))

ax = figure.add\_axes([0.1, 0.1, 0.8, 0.8])

x = np.linspace(0, 4\*np.pi, 100)

ax.set\_xlim(0, 4\*np.pi)

**for** i **in** np.linspace(0, np.pi, 10):

ax.plot(x, np.sin(x + i \* .5) \* (np.pi - i + 1)\*2)

plt.show()

**图形的多样性的便捷性**

* 提供更加多样的图形。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns** *#加载会导致matplotlib的默认作图风格变成seaborn*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:100]

*# 绘制小提琴图*

sns.violinplot(x= data['price\_change']) *# 箱线图与密度图的结合（密度图还可以分成上下两个曲线）*

*# 显示*

plt.show()

*# ------------------*

*# 提示：如果出现警告，请更新scipy的版本, 一般低于1.1.0的会出现警告，我写这个文档的时候，更新到1.3.0。*

**数据的内置性**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:100]

data['p\_change'] = data['p\_change']>=0

*# 绘制小提琴图*

sns.violinplot(x='price\_change', hue='p\_change', data=data,) *# 箱线图与密度图的结合（密度图还可以分成上下两个曲线）*

*# 显示*

plt.show()

*# data[0:3]*

**与matplotlib的接口调用**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change'], color=(1,0,0,1)) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

**Seaborn的样式与颜色**

**样式与颜色函数API说明**

* 样式分成两组函数
  1. 设置主题样式
  2. 设置相关参数
* 样式与参数设置：set([context, style, palette, font, …])
  1. 样式
     + 返回样式：axes\_style([style, rc])
     + 设置样式：set\_style([style, rc])
  2. 参数
     + 返回参数：plotting\_context([context, font\_scale, rc])
     + 设置参数：set\_context([context, font\_scale, rc])
  3. 其他函数：
     + set\_color\_codes([palette])：更改Matplotlib颜色简写的的说明。
     + reset\_defaults()：恢复到RC的缺省设置（RC是matplotlib的参数设置）
     + reset\_orig()：恢复到RC的原始设置
* 颜色调色板的函数
  1. 设置调色板：set\_palette(palette[, n\_colors, desat, …])
  2. 使用调色板获取颜色：color\_palette([palette, n\_colors, desat])

**set样式设置与参数设置**

* set函数定义：

seaborn.set(

context='notebook',

style='darkgrid',

palette='deep',

font='sans-serif',

font\_scale=1,

color\_codes=**True**,

rc=**None**)

* color\_codes:
  + 当palette是seaborn颜色模板，并且color\_codes=True，则支持'r','g','b'等颜色代码。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*# ----------------------*

sns.set() *# 默认都是缺省设置*

*# ----------------------*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change'], color=(1,0,0,1)) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

**context参数**

* context参数类型可以两种方式：
  + RC参数字典；
  + RC参数预先设置名，名字只能是如下字符串：paper, notebook, talk, poster；默认是notebook

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*# ----------------------*

sns.set(context='talk') *# 设置talk风格*

*# ----------------------*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change'], color=(1,0,0,1)) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

**使用plotting\_context设置**

- `seaborn.plotting\_context(context=None, font\_scale=1, rc=None)`

- context：参数；

- font\_scale：字体的独立方法缩放参数；

- rc：设置matplotlib参数；

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*# ----------------------*

sns.reset\_defaults() *# 恢复到缺省设置*

sns.set\_context(context='talk', font\_scale=0.5, rc={"lines.linewidth": 5}) *# 有可能被后面的设置覆盖*

*# ----------------------*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change'], color=(1,0,0,1)) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

sns.plotting\_context('paper')

**style参数**

* 用来设置坐标轴的样式，类型是字典类型或者设置好的字符串样式名
  + 预置的样式名包含：{darkgrid, whitegrid, dark, white, ticks}

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*# ----------------------*

sns.reset\_defaults() *# 恢复到缺省设置*

sns.set(

context='talk',

style='whitegrid',

font='Microsoft YaHei',

font\_scale=0.5,

rc={"lines.linewidth": 5}) *# 有可能被后面的设置覆盖*

*# ----------------------*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change'], color=(1,0,0,1)) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

sns.axes\_style('ticks')

**使用axes\_style坐标轴的样式：**

- `seaborn.axes\_style(style=None, rc=None)'

**palette参数**

* 用来指名颜色模板名，或者颜色列表。
  + seaborn预置颜色模板包含：deep, muted, bright, pastel, dark, colorblind
* 使用color\_palette获取颜色
  + seaborn.color\_palette(palette=None, n\_colors=None, desat=None)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*# ----------------------*

sns.reset\_defaults() *# 恢复到缺省设置*

sns.set(

context='talk',

style='ticks',

font\_scale=0.4,

palette='cool' *# 调用set\_palette：set\_palette(palette, color\_codes=color\_codes)，这里使用的是matplotlib的颜色*

) *# 有可能被后面的设置覆盖*

*# ----------------------*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change']) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

**使用set\_palette函数设置**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*# ----------------------*

sns.reset\_defaults() *# 恢复到缺省设置*

sns.set(

context='talk',

style='ticks',

font\_scale=0.4

) *# 有可能被后面的设置覆盖*

sns.set\_palette('cool', n\_colors=6, desat=0.5)

*# ----------------------*

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数据*

data = data[0:20]

*# 直接绘制*

ax = sns.lineplot(data.index,data['price\_change']) *# matplotlib参数使用变参自由设置*

*# 通过返回的坐标系，直接设置*

ax.set\_xticklabels(labels=data.index, rotation=90) *# 刻度标签*

ax.grid(b=**False**) *# 直接控制网格*

*# 显示e*

plt.show()

**Seaborn的图形 - 网格图**

* 某些数据需要分成多个坐标些绘制（把数据分成多个子集绘制），网格图提供了自动分组绘制的功能。网格图分成3类：
  + Facet grids
    - 用于绘制分成多个方面表示的数据。
  + Pair grids
    - 用于绘制成对关系的数据。
  + Joint grids
    - 用于绘制二变量图的。
* 网格图提供对数据的自动处理
* **注意：**
  + 下面网格的重点是理解数据分组，因为网格的作用就是把数据分成若干子集显示
  + 每个不同的网格，子集的划分方式也不同。

**Facet grids网格**

**Facet grids的使用**

* Facet grids
  + 把数据集分成多个子集，数据子集的个数由Facet grids的参数确定：
    - row=None,
    - col=None,
    - hue=None,
* 类构造器定义说明

**class** **seaborn**.FacetGrid(

data,

row=**None**,

col=**None**,

hue=**None**,

col\_wrap=**None**, *# 列自动换行（一行绘制不下的时候）,col\_wrap与row参数不兼容。*

sharex=**True**,

sharey=**True**,

height=3, *# 每个facet的高度，单位是inch。*

aspect=1, *# 高宽比例，用来计算宽度= aspect \* height*

palette=**None**, *# hue使用的颜色。palette name, list, or dict, optional*

row\_order=**None**, *# row的顺序，分类的值得顺序，比如鸢尾花分类0，1，2的顺序。*

col\_order=**None**,

hue\_order=**None**,

hue\_kws=**None**, *# 参数字典，用来控制hue的属性，比如marker的样式等，包含颜色*

dropna=**True**, *# 是否删除NaN值。*

legend\_out=**True**, *# 绘制网格线到外面*

despine=**True**, *# 从绘图中移除顶部和右侧的轴。*

margin\_titles=**False**, *# 使用row列名绘制标题，有的时候无效。*

xlim=**None**, *# 坐标轴的上下限*

ylim=**None**,

subplot\_kws=**None**, *# subplot的参数，用来控制坐标轴Axes对象的属性参数*

gridspec\_kws=**None**, *# 网格的控制参数*

size=**None** *# 这个参数被升级为height参数了，使用的时候，警告会告诉你。*

* 此类将数据集映射到多个坐标系上，坐标系的多少由相应数据集中列名的值的类别多少确定；
  + row：指定行
  + col：指定列
  + hue：指定由不同颜色区别数据值
  + 这三个列名的指定的数据集的应该是离散的，并且类别统计数量有限，对连续数据不太现实。

-上面三个参数需要依赖data参数的指定数据集（数据集需要是DataFrame，ndarray没有列名）。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, col=4, height=5) *# 使用标签的值来决定列数（因为鸢尾花只有三类，所以坐标系有三个）*

g = sns.FacetGrid(iris, row=4) *# 使用标签的值来决定行数，还可以使用hue在分子集。*

**在Facet grids网格中绘制图形**

* 绘制函数的定义

FacetGrid.map(func, \*args, \*\*kwargs) *# 应用图形绘制函数func到每个facet的数据子集；*

FacetGrid.map\_dataframe(func, \*args, \*\*kwargs) *# 类似FacetGrid.map，但传递参数（args）与数据（kwargs）*

**map函数的应用**

* map函数args与kwargs直接传递给func。数据内部自己处理。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, col=4)

g.map(plt.scatter, 0, 2, s=2\*\*2) *# 0是数据集中列名为0的列，2式数据集中列名为2的列，其他属性*

**map\_dataframe函数的应用**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, col=4)

*# 绘制函数*

**def** draw(x, y, \*\*kwargs): *# kwargs额外传递了color与data*

*# kwargs 包含两个额外的参数color与data，data是按照FacetGrid中参数分组的。*

ax = plt.gca()

*# 取出数据，取出颜色*

data = kwargs.pop('data')

color = kwargs.pop('color')

*# ax.scatter(data[x], data[y], color=color, \*\*kwargs) # 直接使用剩余的参数*

ax.scatter(data[x], data[y], color=(1,0,0,1), \*\*kwargs)

g.map\_dataframe(draw, 0, 2, s=3\*\*2, marker='.')

**FacetGrid参数的使用**

**data, row，col，hue参数**

* data ：绘制数据
* row=None, 指定用来指定行数的分组（分组的数据行数决定行数）
* col=None, 指定用来指定列数的分组（分组的数据行数决定列数）
* hue=None, 指定用来指定分颜色显示的分组（分组的数据行数决定颜色的数量）

1. 单独使用row，col

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, col=4) *# 可以把col换成row*

g.map(plt.scatter, 0, 2, s=2\*\*2) *# 0是数据集中列名为0的列，2式数据集中列名为2的列，其他属性*

1. 单独使用hue参数

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, hue=4) *# 对4列进行颜色分组*

g.map(plt.scatter, 0, 2, s=2\*\*2) *# 0是数据集中列名为0的列，2式数据集中列名为2的列，其他属性*

1. 同时使用row，col与hue参数

In [ ]:

*# 同时使用row，col与hue的情况（因为iris的分组数目多，不适合使用，我们换成Seaborn的官方数据）*

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data = pd.read\_csv('tips.csv')

data[0:2]

In [ ]:

*# tips.csv的数据：otal\_bill tip sex smoker day time size*

*# 数据的shape：(244, 7)*

*# 其中sex， smoker， day，time非常适合分组*

g = sns.FacetGrid(data, row='sex', col='time', hue='day')

*# 上面的参数解释按照sex显示2行，按照time显示4列，每个图中数据按照day显示为7中颜色*

g.map(plt.scatter, 'total\_bill','tip', s=2\*\*2)

**sharex与sharey参数**

- 设置为True使用一样的坐标参数。否则根据数据情况调整。

In [ ]:

*# tips.csv的数据：otal\_bill tip sex smoker day time size*

*# 数据的shape：(244, 7)*

*# 其中sex， smoker， day，time非常适合分组*

g = sns.FacetGrid(data, row='sex', col='time', hue='day',sharex=**False**,sharey=**False**)

*# 上面的参数解释按照sex显示2行，按照time显示4列，每个图中数据按照day显示为7中颜色*

g.map(plt.scatter, 'total\_bill','tip', s=2\*\*2)

**height（size），aspect，参数**

* size已经被height替代。
* 宽度由height \* aspect 决定

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, col=4, height=3, aspect=2)

g.map(plt.scatter, 0, 2, s=2\*\*2) *# 0是数据集中列名为0的列，2式数据集中列名为2的列，其他属性*

**row\_order,col\_order, hue\_order参数**

* 因为row，col，huer按值分组，所以其中的值在显示的时候会有顺序
  + 比如鸢尾花的3个类别，按照0，1，2显示？还是1，0，2显示？可以使用指定列表指定。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(iris, col=4, height=3, aspect=1, col\_order=[2, 0, 1]) *# 可观察每个坐标轴的标题中的内容*

g.map(plt.scatter, 0, 2, s=2\*\*2)

**palette参数**

* 用来指定调色板，数据类型可以是如下类型：
  + palette name,
  + list,
  + dict

1. 使用list与dict指定的颜色，这个需要意义对应。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

hue=4,

height=3,

aspect=1,

*# palette= [(1,0,0,1),(0,1,0,1),(0,0,1,1)])*

palette= {

0.0: (1,0,0,1), *# key对应分组的值。比如：0.0，1.0，2.0*

1.0: (0,1,0,1),

2.0: (0,0,1,1)

})

g.map(plt.scatter, 0, 2, s=2\*\*2)

1. 使用调色板指定颜色
   * 后面专门讲解下seaborn的颜色；这里简单使用。
   * sns.color\_palette(palette='cool', n\_colors=3, desat=.5)
     + desat：饱和度，0-1之间的数值。
     + 饱和度是指色彩的鲜艳程度，也称色彩的纯度。
     + 饱和度取决于该色中含色成分和消色成分（灰色）的比例。含色成分越大，饱和度越大；消色成分越大，饱和度越小。
     + 纯的颜色都是高度饱和的，如鲜红，鲜绿。混杂上白色，灰色或其他色调的颜色，是不饱和的颜色，如绛紫，粉红，黄褐等。
     + 完全不饱和的颜色根本没有色调，如黑白之间的各种灰色

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

*# 从cool调色板中取3个颜色构成调色板（颜色列表）*

palette = sns.color\_palette(palette='cool', n\_colors=3, desat=1) *# 取1表示饱和，颜色彩色度更好。*

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

hue=4,

height=3,

aspect=1,

palette=palette

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

**hue\_kws参数**

* 该参数主要根据绘制图形不同，用来指定用颜色表示分组数据的图形样式。
  + 比如散点图就指定点的样式，折线图就是表示线段的样式。
  + kws就是kwargs的意思。
  + 数据类型是字典表示。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

*# 从cool调色板中取3个颜色构成调色板（颜色列表）*

palette = sns.color\_palette(palette='cool', n\_colors=3, desat=1) *# 取1表示饱和，颜色彩色度更好。*

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

hue=4,

height=3,

aspect=1,

palette=palette,

hue\_kws={

'marker':['o','>','<'],

'color':[(1,0,0,1),(0,1,0,1),(0,0,1,1)], *# 会覆盖palette*

'edgecolor':[(0,1,1),(1,0,1),(1,1,0)] *# 点的边界颜色等*

}

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

**legend\_out参数**

* 控制legend的绘制位置
  + True：扩大Figure，早图的外面绘制主题，否则在图中绘制主题，当然前提是设置显示主题才有效果。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.font\_manager** **import** FontProperties

*# 解决汉字问题*

font=FontProperties(fname=r'msyh.ttf',size=14)

*# sns.set(font=font.get\_name())*

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target])

g = sns.FacetGrid(

iris,

hue=4,

legend\_out=**False** *# False在内部，True在外部*

)

g.map(plt.scatter, 0,2, s=2\*\*2).add\_legend(title='标题') *# 添加主题*

**despine参数**

* 取值类型：True或者False，用来控制top与right的坐标系边框的显示。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.font\_manager** **import** FontProperties

*# 解决汉字问题*

font=FontProperties(fname=r'msyh.ttf',size=14)

*# sns.set(font=font.get\_name())*

sns.set(font='Microsoft YaHei')

sns.set\_style("ticks") *# 用来演示despine的参数的。*

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target])

g = sns.FacetGrid(

iris,

hue=4,

legend\_out=**False**, *# False在内部，True在外部z*

despine=**True** *# 坐标系top与right的边框，这个需要sns.set\_style("ticks")设置显示坐标系才行。默认是False*

)

g.map(plt.scatter, 0,2, s=2\*\*2)

**margin\_titles参数**

* 用来控制row变量指定的行标题显示的位置：
  + True : 显示在右边
  + False：显示在上面

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

row=4,

margin\_titles=**True**

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

**xlim, ylim参数**

- 控制坐标轴的大小，类型是元组

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

col=4,

xlim=(0,10),

ylim=(-2, 10)

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

**subplot\_kws参数**

* 用来传递给subplot函数的参数。
  + projection
  + 用来创建子坐标系的参数

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

col=4,

subplot\_kws={

'facecolor':(0,0,1,0.5),

}

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

**FacetGrid的成员函数**

**add\_legend绘制主题**

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

col=4,

xlim=(0,10),

ylim=(-2, 10)

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

*# legends = {*

*# 0:mpl.patches.Patch(color=(1,0,0), label='Hell'),*

*# 1:mpl.patches.Patch(color=(0,1,0), label='word'),*

*# }*

legends = {

'主题一': mpl.lines.Line2D(xdata=[0, 1], ydata=[0, 0], color=(1,0,0), label='Hell'),

'主题二': mpl.patches.Patch(color=(0,1,0), label='word'),

}

g.add\_legend(legend\_data=legends, title='标题')

**despine控制坐标轴**

* 存在全局函数提供坐标轴的控制

seaborn.despine(

fig=**None**,

ax=**None**,

top=**True**, *# 坐标框架的位置：上*

right=**True**,

left=**False**,

bottom=**False**,

offset=**None**, *# 两个坐标轴的偏离距离*

trim=**False**) *# 当边框没有覆盖整个数据轴的范围的时候，trim参数会限制留存的边框范围。*

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

sns.set\_style("ticks")

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

col=4,

xlim=(0,10),

ylim=(-2, 10)

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

g.despine(top=**False**, bottom=**False**, right=**False**, left=**False**, offset=-20, trim=**False**)

**facet\_axis访问每个坐标系**

* 使用ax属性与axes属性也可以访问。
  + facet\_axis(row\_i, col\_j)
  + ax一个坐标系
  + axes在多个坐标系使用

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.FacetGrid(

iris,

col=4,

xlim=(0,10),

ylim=(-2, 10)

)

g.map(plt.scatter, 0, 2, s=2\*\*2)

*# 按照位置访问*

ax2 = g.facet\_axis(0, 1)

ax2.set\_title('facet\_axis的修改的标题')

ax3 = g.axes[0,2]

print(type(ax3),g.axes.shape)

ax3.set\_title('axes属性修改的标题')

*# 备注：实际上还能使用fig属性访问呢Figure，这个属性在文档中没有出现，但应该想得到。*

**facet\_data访问每个子集数据**

* facet\_data()

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

g = sns.FacetGrid(data, row='sex', col='time', hue='day',sharex=**False**,sharey=**False**)

*# 上面的参数解释按照sex显示2行，按照time显示4列，每个图中数据按照day显示为7中颜色*

g.map(plt.scatter, 'total\_bill','tip', s=2\*\*2).add\_legend()

f\_data = g.facet\_data() *# 生成器*

*# print(len(list(f\_data)))*

**for** fd **in** f\_data:

print('分组------------')

print(fd[0], fd[1].shape) *# fd是元组：数据的索引（第一个）与数据（第二个）*

*# 注意下面输出的分组*

**savefig保存为图像**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

g = sns.FacetGrid(data, row='sex', col='time', hue='day',sharex=**False**,sharey=**False**)

*# 上面的参数解释按照sex显示2行，按照time显示4列，每个图中数据按照day显示为7中颜色*

g.map(plt.scatter, 'total\_bill','tip', s=2\*\*2).add\_legend()

g.savefig('fig.png',format='png')

**set设置坐标轴标签**

* 这个函数代表系列坐标轴设置函数：
  + set(\*\*kwargs) 设置坐标轴属性Axes的所有属性
  + set\_axis\_labels([x\_var, y\_var]) 设置标签，位置在left column 与 bottom row of the grid.
  + set\_titles([template, row\_template, …]) 绘制标题（可以采用模板，模板指定参数）：
    - row\_var,col\_var：用于分组的的数据集列名
    - row\_name, col\_name：分组以后的分组数据值（每个组是一样的）。
  + set\_xlabels([label]) 设置x轴标签
  + set\_xticklabels([labels, step]) 设置x刻度标签
  + set\_ylabels([label]) 设置x轴标签
  + set\_yticklabels([labels]) 设置x刻度标签

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

g = sns.FacetGrid(data, row='sex', col='time', hue='day',sharex=**False**,sharey=**False**)

*# 上面的参数解释按照sex显示2行，按照time显示4列，每个图中数据按照day显示为7中颜色*

g.map(plt.scatter, 'total\_bill','tip', s=2\*\*2).add\_legend()

g.set\_axis\_labels('X标签', 'Y标签')

*# g.set(xlabel='X标签', ylabel='Y标签') # 等价于上面的代码*

g.set\_titles('参数：**{row\_var}**=**{row\_name}**')

*# help(g.set\_titles)*

g.set\_xticklabels(labels=['X1','X2','X3','X4','X5','X6'],rotation=90) *# 支持其他相关参数*

help(g.set\_xticklabels)

**PairGrid网格**

* PairGrid
  + 对数据特征两两显示关系。
* PairGrid构造器定义说明：

**class** **seaborn**.PairGrid(

data, *# 数据*

hue=**None**, *# 颜色分类显示的数据的列名*

hue\_order=**None**,

palette=**None**,

hue\_kws=**None**,

vars=**None**, *#*

x\_vars=**None**,

y\_vars=**None**,

diag\_sharey=**True**,

height=2.5,

aspect=1,

despine=**True**,

dropna=**True**,

size=**None**)

**PairGrid的数据参数**

**vars参数**

* 使用vars指定需要两两组合显示关系的特征。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.PairGrid(

iris,

hue=4,

vars=[0,1,2] *# 默认是所有特征*

)

g.map(plt.scatter) *# 没有其他参数，只能一个参数。*

**x\_vars与y\_vars参数**

* x\_vars与y\_vars与vars一样，但可以行列单独指定特征，这样的好处就不会显示特征与自己的关系。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.PairGrid(

iris,

hue=4,

x\_vars=[0, 1],

y\_vars=[2, 3]

)

g.map(plt.scatter) *# 没有其他参数，只能一个参数。*

**PairGrid的函数**

add\_legend([legend\_data, title, label\_order]) *# 绘制主题。*

map(func, \*\*kwargs) *# 使用一样的函数在绘制么个坐标系的图。*

map\_diag(func, \*\*kwargs) *# 使用单变量函数绘制对角线上的图；*

map\_lower(func, \*\*kwargs) *# 使用双变量函数绘制下三角的图。*

map\_offdiag(func, \*\*kwargs) *# 使用双变量函数绘制非对角的图。*

map\_upper(func, \*\*kwargs) *# 使用双变量函数绘制上三角的图。*

set(\*\*kwargs) *# 设置坐标轴的属性。*

**map\_diag函数**

- 第一个参数必须是单变量的绘制函数

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.PairGrid(

iris,

hue=4,

vars=[0,1,2,3]

*# x\_vars=[0, 1],*

*# y\_vars=[2, 3]*

)

g = g.map\_diag(plt.hist) *# 对单变量数据统计，或者密度函数等*

**map\_lower与map\_upper与map\_offdiag函数**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.PairGrid(

iris,

hue=4,

vars=[0,1,2,3]

*# x\_vars=[0, 1],*

*# y\_vars=[2, 3]*

)

g.map\_diag(sns.kdeplot) *# 对单变量数据统计或者密度函数等*

*# g.map\_diag(plt.plot) # 可以自动生成整数的x坐标的。*

*# g.map\_lower(plt.scatter)*

*# g.map\_upper(plt.scatter)*

*# g.map\_offdiag(plt.scatter ) # 等于上面两个效果*

g.map\_lower(sns.kdeplot, cmap="cool")

g.map\_upper(plt.scatter, marker='.', s=1)

**JointGrid网格图**

* 绘制二元变量图，但在边界绘制单变量图。
* 类定义说明：

**class** **seaborn**.JointGrid(

x,

y,

data=**None**,

height=6,

ratio=5,

space=0.2,

dropna=**True**,

xlim=**None**,

ylim=**None**,

size=**None**)

**JointGrid网格图结构**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.JointGrid(x=0, y=2, data=iris)

**图形绘制**

annotate(func[, template, stat, loc]) *#关系标注*

plot(joint\_func, marginal\_func[, annot\_func]) *# 调用函数绘制图.*

plot\_joint(func, \*\*kwargs) *# 绘制x与y的二元图func支持两个参数*

plot\_marginals(func, \*\*kwargs) *# 分别绘制x与y的单变量图*

savefig(\*args, \*\*kwargs) *# 保存图形*

set\_axis\_labels([xlabel, ylabel]) *# 设置两个坐标轴的的标签*

**plot绘制**

JointGrid.plot(

joint\_func,

marginal\_func,

annot\_func=**None**)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.JointGrid(x=0, y=1, data=iris)

**def** ann(p1, p2):

*# print(p1) #返回x与y的数据列表*

**return** 20 *# 需要显示的标注*

g = g.plot(plt.scatter, sns.distplot, annot\_func=ann) *# 已经不推荐使用：annot\_func*

**annotate**

annotate(

func[, *# 绘制函数*

template, *# 模板*

stat, *# 用模板的字符串*

loc]) *# 位置*

* 注意，还有其他与标注相关参数都可以在最后添加。
* 参数说明：
  + loc的设置：
    - best
    - upper right
    - upper left
    - lower left
    - lower right
    - right
    - center left
    - center right
    - lower center
    - upper center
    - center

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.JointGrid(x=0, y=1, data=iris)

**def** ann(p1, p2):

*# print(p1) #返回x与y的数据列表*

**return** 20 *# 需要显示的标注*

g = g.annotate(ann, template='**{stat}**: **{val:04d}**', stat='$X^2 + Y^2=$', loc='upper left', fontsize=12) *# 不推荐使用*

**plot\_joint绘制**

* 绘制二元变量图。
  + plot\_joint(func, \*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.JointGrid(x=0, y=1, data=iris)

**def** ann(p1, p2):

*# print(p1) #返回x与y的数据列表*

**return** 20 *# 需要显示的标注*

g.plot\_joint(plt.scatter)

**plot\_marginals绘制**

* 绘制边界上的单元变量图
  + JointGrid.plot\_marginals(func, \*\*kwargs)
* 参数说明：
  + func函数
  + - 两个参数：
  + - 第一个用来接收绘制的数组数据；
  + - 第二个用来指定方向，参数名必须是vertical。

- 满足这个条件的不多，一般都在seaborn中存在。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.JointGrid(x=0, y=1, data=iris)

**def** ann(p1, p2):

*# print(p1) #返回x与y的数据列表*

**return** 20 *# 需要显示的标注*

g.plot\_marginals(sns.kdeplot)

**Seaborn图形-关系图**

* 关系统计图提供三个函数实现：

relplot([x, y, hue, size, style, data, row, …])

- Figure级接口，绘制在FacetGrid上。

- 绘制关系。

scatterplot([x, y, hue, style, size, data, …])

- 绘制一个可能有几个语义分组的散点图。

lineplot([x, y, hue, size, style, data, …])

- 画一个可能有几个语义分组的线图。

**relplot绘制函数**

* 函数定义说明：

seaborn.relplot(

x=**None**,

y=**None**,

hue=**None**,

size=**None**,

style=**None**,

data=**None**,

row=**None**,

col=**None**,

col\_wrap=**None**,

row\_order=**None**,

col\_order=**None**,

palette=**None**,

hue\_order=**None**,

hue\_norm=**None**,

sizes=**None**,

size\_order=**None**,

size\_norm=**None**,

markers=**None**,

dashes=**None**,

style\_order=**None**,

legend='brief',

kind='scatter',

height=5,

aspect=1,

facet\_kws=**None**,

\*\*kwargs)

**x,y与data参数**

* 绘制data中x与y的关系
  + x,y必须是数值型。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.relplot(x=0, y=1, data=iris)

**kind参数与line的意义**

* scatter散点坐标关系，默认是scatter。
* line曲线关系
  + 阴影部分是由于纵坐标上多个值导致的, 取值为均值, 阴影部分是置信区间（95%作为通用的置信概率）
  + 阴影部分的面积使用置信区间参数控制ci, ci取值0-100表示置信度（置信概率）
  + 异常点不在阴影范围内（异常点的检测方式应该在线箱图概念中一样）。
  + ci参数在文档中没有说明。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.relplot(x=0, y=1, data=iris, kind='line', ci=100)

*# 返回的是FacetGrid，可以再绘制。*

g.map(plt.scatter,0,1, color=(1, 0, 0, 1), s=3\*\*2, marker= '.')

**分组变量参数hue,size,style**

* 这三个参数用来指定数据集中的列名来分组，分组的数据使用不同的方式显示：
  + hue：使用颜色区分显示不同的组；
  + size：使用大小区分显示不同的组；
  + style：使用样式区分显示不同的组；（比如marker样式等）

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

g = sns.relplot(x='total\_bill', y='tip', hue='sex', size='size', style='day', data=data, kind='scatter')

**hue,size是字母与数值的区别**

* 使用hue与size指定的列的数据可以是字母与数值
  + 字母就直接用来作为类别
  + 数值:
    - hue就用来作为颜色索引
    - size就作为点的大小。size的数值类型必须是整数，小数就无效。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

g = sns.relplot(x='total\_bill', y='tip', hue='tip', size='total\_bill', style='sex', data=data, kind='scatter',palette='cool')

**hue,size作为列表**

* hue,size,style参数可以使用列表指定对应行的颜色，大小与样式
* hue,size本身还可以是数值。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

*# 数据行数为244*

g = sns.relplot(x='total\_bill', y='tip', hue=range(244), size=range(244), data=data, kind='scatter',palette='cool')

*# style.tolist()*

**hue\_norm与size\_norm参数**

* hue\_norm与size\_norm规范化范围，也可以使用函数来转换；
  + hue\_norm=(0,50) 如果指定total\_bill作为hue指定的变量。
  + 或者使用规范化对象，比如对数转换对象（一种可调用对象）。
    - 可以直接实现，也可以调用matplotlib中已经实现好的函数。

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

g = sns.relplot(x='total\_bill', y='tip', hue='total\_bill', size='total\_bill',

hue\_norm=(0, 50), *# 0-50之间线性同比缩小*

data=data, kind='scatter',palette='cool')

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

**class** **log\_norm**(Normalize): *# 其实就是0-1标准规范化实现。*

**def** \_\_call\_\_(self, v):

**if** isinstance(v, np.float64):

**return** v / (self.vmax - self.vmin)

**else**:

**return** np.array(v) / (self.vmax - self.vmin)

*# 字段： otal\_bill tip sex smoker day time size*

g = sns.relplot(x='total\_bill', y='tip', hue='total\_bill', size='total\_bill',

hue\_norm=log\_norm(0, 50), *# 0-50之间线性同比缩小*

data=data, kind='scatter',palette='cool')

**sizes参数**

* 用来指定每个分类的值对应的显示的大小, 类型是：
  + list, dict, or tuple,

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

g = sns.relplot(x='total\_bill', y='tip', hue='total\_bill',

size='total\_bill', sizes=(0, 5\*\*2), *# 指定大小的*

data=data, kind='scatter',palette='cool')

* size是分类的时候，使用列表与字典

In [ ]:

*# row*

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

g = sns.relplot(x='total\_bill', y='tip', hue='total\_bill',

size='day',

sizes={'Fri':2\*\*2, 'Thur':2\*\*3, 'Sun':2\*\*4, 'Sat':2\*\*5},

*# sizes=[2\*\*2, 4\*\*2,8\*\*2, 16\*\*2], # day的4个值对应的大小*

data=data, kind='scatter',palette='cool')

**scatterplot散点图**

seaborn.scatterplot(

x=**None**,

y=**None**,

hue=**None**,

style=**None**,

size=**None**,

data=**None**,

palette=**None**,

hue\_order=**None**,

hue\_norm=**None**,

sizes=**None**,

size\_order=**None**,

size\_norm=**None**,

markers=**True**, *# 用来指定marker样式，类型：boolean, list, or dictionary, optional*

style\_order=**None**,

x\_bins=**None**, *# 当前没有实现*

y\_bins=**None**,

units=**None**, *# 没有实现*

estimator=**None**, *# 分组数据的聚合方式（估计量）,对散点没有实现*

ci=95, *# 置信度 或者指定使用标准方差 int or “sd” ，对散点没有实现*

n\_boot=1000, *# 计算置信度的采样次数。采用bootstraps采样：有放回采样*

alpha='auto', *# 点的透明度*

x\_jitter=**None**, *# 没有使用*

y\_jitter=**None**,

legend='brief', *# 主题的绘制方式“brief”, “full”, or False,*

ax=**None**, *# 绘制到指定的坐标轴*

\*\*kwargs)

* 其中data用来指定数据源；
* markers参数

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

*# 字段： otal\_bill tip sex smoker day time size*

*# g = sns.relplot(x='total\_bill', y='tip', hue='total\_bill', style='day', markers=['>','<','^','v'],*

*# data=data, kind='scatter',palette='cool')*

g = sns.scatterplot(x='total\_bill', y='tip', hue='total\_bill', style='day', markers=['>','<','^','v'],

data=data,palette='cool') *# 与上面一样。*

**lineplot线段图**

seaborn.lineplot(

x=**None**,

y=**None**,

hue=**None**,

size=**None**,

style=**None**,

data=**None**,

palette=**None**,

hue\_order=**None**,

hue\_norm=**None**,

sizes=**None**,

size\_order=**None**,

size\_norm=**None**,

dashes=**True**, *# 线条样式：boolean, list, or dictionary, optional*

markers=**None**, *# 点的样式*

style\_order=**None**,

units=**None**, *# 单位*

estimator='mean', *# 聚合方法*

ci=95, *# 置信概率*

n\_boot=1000, *# bootstrap采样方法*

sort=**True**, *# 绘制的线段的点是否按照x排序。*

err\_style='band', *# 是否用半透明误差带或离散误差条绘制置信区间。*

err\_kws=**None**, *# err绘制的参数：err\_band : dict of keyword arguments*

legend='brief',

ax=**None**, \*\*kwargs)

**estimator参数与ci参数**

* 采样数与置信概率的设置

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

*# 数据预处理*

ax = sns.lineplot(x='ym', y='open', data=data,n\_boot=10000,ci=95, sort=**False**)

sns.scatterplot(x='ym',y='open', hue='open', data=data, ax=ax, s=2\*\*3,palette='cool', markers='o')

sns.lineplot(x='ym',y='open', data=data, ax=ax,estimator='max', ci=0, sort=**False**)

sns.lineplot(x='ym',y='open', data=data, ax=ax,estimator='min', ci=0, sort=**False**)

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

ax.set\_xticklabels(labels=g\_data['ym'], rotation=90)

ax.figure.set\_size\_inches(12,6)

plt.show()

**err\_style与err\_band参数**

* err\_style参数取值：
  + band : 默认
  + bars
* err\_band
  + 与err\_style有关的参数包含：
    - ax.fill\_between
    - ax.errorbar

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

*# 数据预处理*

ax = sns.lineplot(x='ym', y='open', data=data,n\_boot=100,ci=95, err\_style='bars', sort=**False**)

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

ax.set\_xticklabels(labels=g\_data['ym'], rotation=90)

ax.figure.set\_size\_inches(12,6)

**Seaborn图形-分类图**

* Seaborn提供的分类图有：

catplot([x, y, hue, data, row, col, …])

*# 绘制分类图到FacetGrid。下面的都是这个函数的特列，使用kind指定不同类型*

stripplot([x, y, hue, data, order, …])

*# 绘制的散点图中有一个数据是类别变量。*

swarmplot([x, y, hue, data, order, …])

*# 绘制点不重叠的分类散点图。*

boxplot([x, y, hue, data, order, hue\_order, …])

*# 绘制与类别分布有关的方框图。*

violinplot([x, y, hue, data, order, …])

*# 绘制箱线图和核密度估计图结合的图。*

boxenplot([x, y, hue, data, order, …])

*# 为容量较大数据集的绘制方框图，有所增强。*

pointplot([x, y, hue, data, order, …])

*# 绘制散点图，使用散点图的符号显示点估计和置信区间。*

barplot([x, y, hue, data, order, hue\_order, …])

*# 绘制柱状图，将点估计和置信区间显示为矩形柱状条。*

countplot([x, y, hue, data, order, …])

*# 绘制柱状图，其中的柱状表示每个分类箱中的观察计数*

* catplot函数的使用分成很多类型
  + 返回的是FacetGrid对象

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.catplot(x='ym',y='open',data=data)

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.ax.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.fig.set\_size\_inches(12,6)

*# data*

**stripplot单变量抖动散点图**

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.stripplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

jitter=0.2,

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

**swarmplot不重叠散点图**

* 与stripplot类似，但对重叠点进行处理。处理方式不是抖动。
* 函数定义
* seaborn.swarmplot(
* x=**None**,
* y=**None**,
* hue=**None**,
* data=**None**,
* order=**None**,
* hue\_order=**None**,
* dodge=**False**,
* orient=**None**,
* color=**None**,
* palette=**None**,
* size=5,
* edgecolor='gray',
* linewidth=0,
* ax=**None**,
* \*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.swarmplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

**boxplot分类箱线统计图**

* 函数定义说明

seaborn.boxplot(

x=**None**,

y=**None**,

hue=**None**,

data=**None**,

order=**None**,

hue\_order=**None**,

orient=**None**,

color=**None**,

palette=**None**,

saturation=0.75,

width=0.8,

dodge=**True**,

fliersize=5,

linewidth=**None**,

whis=1.5,

notch=**False**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.boxplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

**violinplot箱线与核密度估计图**

* 函数定义说明

seaborn.violinplot(

x=**None**,

y=**None**,

hue=**None**,

data=**None**,

order=**None**,

hue\_order=**None**,

bw='scott',

cut=2,

scale='area',

scale\_hue=**True**,

gridsize=100,

width=0.8,

inner='box',

split=**False**,

dodge=**True**,

orient=**None**,

linewidth=**None**,

color=**None**,

palette=**None**,

saturation=0.75,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.violinplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

g = sns.violinplot(

x="day",

y="total\_bill",

hue="smoker",

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**,

split=**True**

)

g.figure.set\_size\_inches(12,6)

**boxenplot大样本箱线图**

* 体现了样本数量的细节。
* 函数说明

seaborn.boxenplot(

x=**None**,

y=**None**,

hue=**None**,

data=**None**,

order=**None**,

hue\_order=**None**,

orient=**None**,

color=**None**,

palette=**None**,

saturation=0.75,

width=0.8,

dodge=**True**,

k\_depth='proportion',

linewidth=**None**,

scale='exponential',

outlier\_prop=**None**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.boxenplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

g = sns.boxenplot(

x="day",

y="total\_bill",

hue="smoker",

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

)

g.figure.set\_size\_inches(12,6)

**pointplot 估计与置信区间点图**

* 类似箱线图，但使用点替代。
* 函数定义说明

seaborn.pointplot(

x=**None**,

y=**None**,

hue=**None**,

data=**None**,

order=**None**,

hue\_order=**None**,

estimator=<function mean>,

ci=95,

n\_boot=1000,

units=**None**,

markers='o',

linestyles='-',

dodge=**False**,

join=**True**,

scale=1,

orient=**None**,

color=**None**,

palette=**None**,

errwidth=**None**,

capsize=**None**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.pointplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

markers='.',

capsize=0.1, *# 端点大小*

errwidth=1, *# 错误线粗细*

estimator=np.max, *# 统计函数 np.std等函数*

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

**barplot 估计与置信区间柱状图**

* 函数说明

seaborn.barplot(

x=**None**,

y=**None**,

hue=**None**,

data=**None**,

order=**None**,

hue\_order=**None**,

estimator=<function mean>,

ci=95,

n\_boot=1000,

units=**None**,

orient=**None**,

color=**None**,

palette=**None**,

saturation=0.75,

errcolor='.26',

errwidth=**None**,

capsize=**None**,

dodge=**True**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data = ts.get\_hist\_data('000001') *#一次性获取全部日k线数年月日*

*# 数据预处理*

data['date'] = data.index

dt = pd.to\_datetime(data.index, format='%Y-%m-**%d**')

data['ym'] = dt.strftime('%Y-%m')

g = sns.barplot(

x='ym',

y='open',

*# hue='ym',*

data=data,

palette='cool',

capsize=0.1, *# 端点大小*

errwidth=1, *# 错误线粗细*

errcolor=(1,0,0,1), *# 误差颜色*

saturation=0.1, *# 颜色饱和度*

estimator=np.min, *# 统计函数 np.std等函数*

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g\_data = data.groupby('ym').sum()

g\_data['ym'] = g\_data.index

g.set\_xticklabels(labels=g\_data['ym'], rotation=90)

g.figure.set\_size\_inches(12,6)

*# data*

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

sns.set(style="ticks", font='Microsoft YaHei')

data = pd.read\_csv('tips.csv')

g = sns.barplot(

x="day",

y="total\_bill",

hue="smoker",

data=data,

palette='cool',

*# linewidth=1,*

*# edgecolor=(1,0,0,1),*

dodge=**True**

)

g.figure.set\_size\_inches(12,6)

**countplot分类直方图**

* 函数说明

seaborn.countplot(

x=**None**,

y=**None**,

hue=**None**,

data=**None**,

order=**None**,

hue\_order=**None**,

orient=**None**,

color=**None**,

palette=**None**,

saturation=0.75,

dodge=**True**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.countplot(

x=1,

hue=4, *# 分类显示直方图*

data=iris,

palette='cool',

capsize=0.1, *# 端点大小*

saturation=1, *# 颜色饱和度*

dodge=**True**

) *# jitter控制振动方法*

*# g = sns.stripplot(x='ym', y='open', hue='ym', data=data, palette='cool',jitter=0.2) # jitter控制振动方法*

g.figure.set\_size\_inches(12,6)

*# data*

**Seaborn图形-分布图**

jointplot(x, y[, data, kind, stat\_func, …])

*# 一元变量与二元变量图合并绘制二元图。*

pairplot(data[, hue, hue\_order, palette, …])

*# 数据集中两两关系图。*

distplot(a[, bins, hist, kde, rug, fit, …])

*# 单变量分布图*

kdeplot(data[, data2, shade, vertical, …])

*# 拟合并绘制核密度图（一元与二元）*

rugplot(a[, height, axis, ax])

*# 将数组中的数据点绘制为刻度图。*

**jointplot合并图**

* 函数说明：

seaborn.jointplot(

x,

y,

data=**None**,

kind='scatter',

stat\_func=**None**, *# 统计函数，已经不推荐使用。*

color=**None**,

height=6,

ratio=5,

space=0.2,

dropna=**True**,

xlim=**None**,

ylim=**None**,

joint\_kws=**None**,

marginal\_kws=**None**,

annot\_kws=**None**,

\*\*kwargs)

* kind可以指定的图的类型
  + kind的取值: { “scatter” | “reg” | “resid” | “kde” | “hex” }

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.jointplot(

x=1,

y=3,

data=iris,

kind='kde',

color=(1,0,0,1))

g.fig.set\_size\_inches(10,6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

* 使用JointGrid控制每个部分的绘制

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.jointplot(

x=1,

y=3,

data=iris,

kind='kde',

color=(1, 1, 0, 1)

)

g.plot\_joint(sns.kdeplot, zorder=0, n\_levels=6)

g.fig.set\_size\_inches(10,6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

**pairplot数据集两两关系图**

* 函数说明：

seaborn.pairplot(

data,

hue=**None**,

hue\_order=**None**,

palette=**None**,

vars=**None**,

x\_vars=**None**,

y\_vars=**None**,

kind='scatter',

diag\_kind='auto',

markers=**None**,

height=2.5,

aspect=1,

dropna=**True**,

plot\_kws=**None**,

diag\_kws=**None**,

grid\_kws=**None**,

size=**None**)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.pairplot(

data=iris,

hue=4,

x\_vars=[0,1],

y\_vars=[2,3],

palette='cool'

)

g.fig.set\_size\_inches(10,6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

**distplot分布图**

* 分布图使用多种方式显示分布信息。
  + 这个函数不支持数据源绑定。
* 函数说明

seaborn.distplot(

a,

bins=**None**,

hist=**True**,

kde=**True**,

rug=**False**,

fit=**None**, *# 具有fit函数的对象*

hist\_kws=**None**,

kde\_kws=**None**,

rug\_kws=**None**,

fit\_kws=**None**,

color=**None**,

vertical=**False**,

norm\_hist=**False**,

axlabel=**None**,

label=**None**,

ax=**None**)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

**from** **scipy.stats** **import** norm

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

*# 拟合模型*

g = sns.distplot(

a=iris[0],

bins=10,

color=(0, 0, 1, 1),

hist=**True**,

kde=**True**,

rug=**True**,

fit = norm, *# 拟合成正态分布*

hist\_kws={

"histtype": "step",

"linewidth": 3,

"alpha": 1,

"color": "g"}

)

g.figure.set\_size\_inches(10, 6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

**kdeplot密度分布图**

* 函数说明：

seaborn.kdeplot(

data,

data2=**None**,

shade=**False**,

vertical=**False**,

kernel='gau',

bw='scott',

gridsize=100,

cut=3,

clip=**None**,

legend=**True**,

cumulative=**False**,

shade\_lowest=**True**, *# 最低部分不绘制*

cbar=**False**,

cbar\_ax=**None**,

cbar\_kws=**None**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.kdeplot(

data=iris[0],

)

g = sns.kdeplot(

data=iris[0],

data2=iris[1],

shade=**True**,

n\_levels=5,

cmap='cool',

shade\_lowest=**False**

*# ax = g*

)

g.figure.set\_size\_inches(10,6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="whitegrid", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.kdeplot(

data=iris[0][:50],

data2=iris[1][:50],

shade=**True**,

n\_levels=5,

cmap='cool',

shade\_lowest=**False**,

*# ax = g*

)

g = sns.kdeplot(

data=iris[0][100:150],

data2=iris[1][100:150],

shade=**True**,

n\_levels=5,

cmap='Reds',

kernel='cos',

shade\_lowest=**False**, *# 最低的部分不绘制。*

*# ax = g*

)

g.figure.set\_size\_inches(10,6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

**rugplot刻度图**

* 函数说明

seaborn.rugplot(

a,

height=0.05,

axis='x',

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

g = sns.rugplot(

a=iris[0],

height=0.5,

)

g.figure.set\_size\_inches(10,6)

*# g.\_\_dict\_\_*

*# # data*

*# 可以通过返回的JointGrid可以控制每个部分的图。*

**Seaborn图形-回归图**

* 回归图一共三个：

lmplot(x, y, data[, hue, col, row, palette, …])

*# 绘制数据与回归模型。*

regplot(x, y[, data, x\_estimator, x\_bins, …])

*# 绘制数据与线性回归模型。*

residplot(x, y[, data, lowess, x\_partial, …])

*# 绘制线性回归的残差；*

**lmplot回归模型图**

* 函数说明：

seaborn.lmplot(

x, y, data,

hue=**None**,

col=**None**, row=**None**,

palette=**None**,

col\_wrap=**None**,

height=5, aspect=1,

markers='o',

sharex=**True**, sharey=**True**,

hue\_order=**None**, col\_order=**None**, row\_order=**None**,

legend=**True**, legend\_out=**True**,

x\_estimator=**None**,

x\_bins=**None**,

x\_ci='ci',

scatter=**True**,

fit\_reg=**True**,

ci=95, n\_boot=1000,

units=**None**,

order=1,

logistic=**False**,

lowess=**False**,

robust=**False**,

logx=**False**,

x\_partial=**None**,

y\_partial=**None**,

truncate=**False**,

x\_jitter=**None**, y\_jitter=**None**,

scatter\_kws=**None**,

line\_kws=**None**,

size=**None**)

* 返回FacetGrid对象

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

iris.rename(

columns={

0: '特征1',

1: '特征2',

2: '特征3',

3: '特征4',

4: '标签'

},

inplace=**True**

)

g = sns.lmplot(x='特征1', y='特征2', data=iris)

g.fig.set\_size\_inches(10,6)

* 按照多类数据分类回归

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

iris.rename(

columns={

0: '特征1',

1: '特征2',

2: '特征3',

3: '特征4',

4: '标签'

},

inplace=**True**

)

*# g = sns.lmplot(*

*# x='特征1', y='特征3',*

*# hue='标签',*

*# data=iris,*

*# markers=['<', '.', '>'],*

*# palette='cool')*

g = sns.lmplot(

x='特征1', y='特征3',

hue='标签',

data=iris,

markers=['<', '.', '>'],

palette={

0: (1, 0, 0),

1: (0, 1, 0),

2: (0, 0, 1)

})

g.fig.set\_size\_inches(10,6)

* 网格显示

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

iris.rename(

columns={

0: '特征1',

1: '特征2',

2: '特征3',

3: '特征4',

4: '标签'

},

inplace=**True**

)

*# g = sns.lmplot(*

*# x='特征1', y='特征3',*

*# hue='标签',*

*# data=iris,*

*# markers=['<', '.', '>'],*

*# palette='cool')*

g = sns.lmplot(

x='特征1', y='特征3',

hue='标签',

col='标签',

data=iris,

ci=100,

markers=['<', '.', '>'],

palette={

0: (1, 0, 0),

1: (0, 1, 0),

2: (0, 0, 1)

})

g.fig.set\_size\_inches(10,6)

* 对数据做估计处理

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

iris.rename(

columns={

0: '特征1',

1: '特征2',

2: '特征3',

3: '特征4',

4: '标签'

},

inplace=**True**

)

*# g = sns.lmplot(*

*# x='特征1', y='特征3',*

*# hue='标签',*

*# data=iris,*

*# markers=['<', '.', '>'],*

*# palette='cool')*

g = sns.lmplot(

x='特征1', y='特征3',

hue='标签',

col='标签',

x\_estimator=np.mean,

data=iris,

ci=100,

markers=['<', '.', '>'],

palette={

0: (1, 0, 0),

1: (0, 1, 0),

2: (0, 0, 1)

})

g.fig.set\_size\_inches(10,6)

**regplot线性回归模型图**

* 函数定义：
* 该函数与lmplot类似，但不支持网格处理。
* seaborn.regplot(
* x, y, data=**None**,
* x\_estimator=**None**, x\_bins=**None**, x\_ci='ci',
* scatter=**True**,
* fit\_reg=**True**,
* ci=95,
* n\_boot=1000,
* units=**None**,
* order=1,
* logistic=**False**,
* lowess=**False**,
* robust=**False**,
* logx=**False**,
* x\_partial=**None**, y\_partial=**None**,
* truncate=**False**,
* dropna=**True**,
* x\_jitter=**None**, y\_jitter=**None**,
* label=**None**,
* color=**None**,
* marker='o',
* scatter\_kws=**None**,
* line\_kws=**None**,
* ax=**None**)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

data = data[0:100]

target = target[0:100]

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

iris.rename(

columns={

0: '特征1',

1: '特征2',

2: '特征3',

3: '特征4',

4: '标签'

},

inplace=**True**

)

g = sns.regplot(

x='特征1', y='标签',

*# x\_estimator=np.mean,*

data=iris,

ci=100,

logistic=**True**,

y\_jitter=0.03

)

g.figure.set\_size\_inches(10,6)

**residplot线性回归残差图**

* 函数定义说明：

seaborn.residplot(

x, y, data=**None**,

lowess=**False**, *# 残差进行平滑拟合*

x\_partial=**None**, *# 混淆变量，并在绘图前从x或y变量中删除。*

y\_partial=**None**,

order=1,

robust=**False**, *# 鲁棒性*

dropna=**True**,

label=**None**,

color=**None**,

scatter\_kws=**None**,

line\_kws=**None**,

ax=**None**)

* 对（x, y）进行线性回归， 并使用把回归线平行x轴，改变x点的位置。

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

*# data = data [0:100]*

*# target = target[0:100]*

iris = pd.DataFrame(np.c\_[data, target]) *# 原鸢尾花的标签合并到第四列，列名自动生成为0，1，2，3，4*

iris.rename(

columns={

0: '特征1',

1: '特征2',

2: '特征3',

3: '特征4',

4: '标签'

},

inplace=**True**

)

g = sns.residplot(

x='特征1', y='特征3',

data=iris,

)

g.figure.set\_size\_inches(10,6)

**Seaborn图形-矩阵图**

* 矩阵图包含两个：
  + 热图与聚类图

heatmap(data[, vmin, vmax, cmap, center, …])

*# 热图*

clustermap(data[, pivot\_kws, method, …])

*# 聚类图*

**heatmap热图**

* 本质上是2D直方图
* 函数定义

seaborn.heatmap(

data, *# 2d数据*

vmin=**None**, vmax=**None**,

cmap=**None**,

center=**None**,

robust=**False**,

annot=**None**,

fmt='.2g',

annot\_kws=**None**,

linewidths=0,

linecolor='white',

cbar=**True**,

cbar\_kws=**None**,

cbar\_ax=**None**,

square=**False**,

xticklabels='auto',

yticklabels='auto',

mask=**None**,

ax=**None**,

\*\*kwargs)

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data = np.random.randn(10, 10)

g = sns.heatmap(

data = data,

*# annot=True,*

fmt="5.2f",

cmap="cool"

)

g.figure.set\_size\_inches(10,6)

**clustermap聚类图**

* 本质也是热图，只是在边界增加了结构。
* 函数定义

seaborn.clustermap(

data,

pivot\_kws=**None**,

method='average',

metric='euclidean',

z\_score=**None**,

standard\_scale=**None**,

figsize=**None**,

cbar\_kws=**None**,

row\_cluster=**True**,

col\_cluster=**True**,

row\_linkage=**None**,

col\_linkage=**None**,

row\_colors=**None**,

col\_colors=**None**,

mask=**None**, \*

\*kwargs)

* 返回ClusterGrid对象

In [ ]:

%**matplotlib** inline

**import** **tushare** **as** **ts**

**import** **matplotlib.pyplot** **as** **plt**

**import** **matplotlib** **as** **mpl**

**import** **seaborn** **as** **sns**

**import** **sklearn.datasets** **as** **ds**

**import** **pandas** **as** **pd**

**from** **matplotlib.colors** **import** Normalize

**import** **tushare** **as** **ts**

sns.set(style="ticks", font='Microsoft YaHei')

data,target= ds.load\_iris(return\_X\_y=**True**)

iris = pd.DataFrame(data)

g = sns.clustermap(

data=iris,

)

g.fig.set\_size\_inches(10,6)

**Matplotlib 3D可视化的编程模式**

1. Matplotlib 3D帮助文档
   * Matplotlib中提供了3D图形绘制接口
   * 
2. 编程模式
   * 3D可视化编程模式与2D非常类似：
     1. 创建Figure；
     2. 创建3D坐标系；(在Figure.add\_axes函数的projection参数中指定'3d'值即可，必须小写)
     3. 绘制3D图形（调用3D绘制接口函数）

* Axes3D的定义：

**class** **mpl\_toolkits**.mplot3d.axes3d.Axes3D(

fig,

rect=**None**,

\*args,

azim=-60,

elev=30,

zscale=**None**,

sharez=**None**,

proj\_type='persp',

\*\*kwargs)

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# ax = figure.add\_subplot(111, projection='3d') # 或者这种方式*

ax.set\_xlim(0,100)

ax.set\_ylim(0,100)

ax.set\_zlim(0,100)

ax.set\_title('3D图形')

ax.set\_xlabel('X轴', labelpad=20,color=(1,0,0,1))

ax.set\_ylabel('Y轴', labelpad=30)

ax.set\_zlabel('Z轴', labelpad=40)

ax.set\_xticks([0,10,20,30,50,80,100])

ax.set\_yticks([0,10,20,30,50,80,100])

ax.set\_zticks([0,10,20,30,50,80,100])

ax.grid(b=**False**) *# 网格线*

plt.show()

**常见3D图形绘制**

**plot线条**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:20j,-5:5:20j];

z = (x\*\*2 - y\*\*2)/2

ax.plot(x.flat, y.flat, z.flat, label='3D线条', color=(1,0,0,1), linewidth=1)

ax.grid(b=**False**) *# 网格线*

plt.show()

**scatter散点图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:100j,-5:5:100j];

z = (x\*\*2 - y\*\*2)/2

colors = plt.cm.get\_cmap('cool')

ax.scatter(x.flat, y.flat, z.flat, label='3D点', s=1, c=z.flat, cmap=colors)

ax.grid(b=**False**) *# 网格线*

plt.show()

**plot\_wireframe网格图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:200j,-5:5:200j];

z = (x\*\*2 - y\*\*2)/2

*# rcount=20, ccount=20 与rstride=5, cstride=5不能同时指定*

ax.plot\_wireframe(x, y, z, label='3D网格', colors= '#0000ff',linewidth=1, rstride=5, cstride=5)

ax.grid(b=**False**) *# 网格线*

plt.show()

**plot\_surface曲面图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:200j,-5:5:200j];

z = (x\*\*2 - y\*\*2)/2

*# rcount=20, ccount=20 与rstride=5, cstride=5不能同时指定*

ax.plot\_surface(x, y, z, label='3D曲面', cmap=plt.cm.get\_cmap('cool') )

ax.grid(b=**False**) *# 网格线*

plt.show()

**plot\_trisurf三角面图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:20j,-5:5:20j];

z = (x\*\*2 - y\*\*2)/2

ax.plot\_trisurf(x.flat, y.flat, z.flat, label='3D三角面', cmap=plt.cm.get\_cmap('cool') )

ax.grid(b=**False**) *# 网格线*

plt.show()

**contour与contourf等高线图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(18, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax1= figure.add\_subplot(121, projection='3d') *# 或者这种方式*

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:20j,-5:5:20j];

z = (x\*\*2 - y\*\*2)/2

*# 等高线不支持标签*

*# ax1.plot\_surface(x, y, z, label='3D曲面', cmap=plt.cm.get\_cmap('cool') )*

ax1.contour(x, y, z, levels=[-10,-9, -8, -7, -6, -5,-4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) *# levels必须递增数列*

ax1.grid(b=**False**) *# 网格线*

ax2= figure.add\_subplot(122, projection='3d')

*# ax2.plot\_surface(x, y, z, label='3D曲面', cmap=plt.cm.get\_cmap('cool') )*

ax2.contourf(x, y, z, cmap=plt.cm.get\_cmap('cool'),

levels=[-10,-9, -8, -7, -6, -5,-4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9] )

plt.show()

**bar柱状图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(8, 6))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

x = [1,2,3,4,5]

y = [5,4,3,2,1]

z = [1,2,3]

*# 使用线条绘制马鞍面*

ax.bar(x, y, z[0], label='3D柱状图', color=(1, 0, 0, 0.7),zdir='y') *# zdir是y，则柱状图在xz平面上*

ax.bar(x, y[::-1], z[1], label='3D柱状图', color=(0, 1, 0, 0.7),zdir='y') *# zdir是y，则柱状图在xz平面上*

ax.bar(x, y, z[2], label='3D柱状图', color=(0, 0, 1, 0.7),zdir='y') *# zdir是y，则柱状图在xz平面上*

ax.grid(b=**False**) *# 网格线*

plt.show()

**quiver箭头图**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(16,8))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:20j,-5:5:20j];

z = (x\*\*2 - y\*\*2)/2

ax.quiver(x, y, z, 1, 0, 0, length=0.5, linewidth=1,color=(1,0,0,1), arrow\_length\_ratio=0.5,normalize=**False**)

*# normalize控制箭头的长短*

*# 1,0,0决定了绘制的每个坐标的方向，这儿所有点方向一样（方向是圆点到这三个参数的连线方向）*

ax.grid(b=**False**) *# 网格线*

plt.show()

**text文本**

In [ ]:

% matplotlib inline

**import** **matplotlib.pyplot** **as** **plt**

**from** **mpl\_toolkits.mplot3d** **import** axes3d

**import** **numpy** **as** **np**

*# 1. 创建图(绘制环境)*

figure = plt.figure('3D图形', figsize=(16,8))

*# 2. 创建3D坐标系（直接创建，使用Figure中的函数创建：这里使用函数）*

ax = figure.add\_axes([0.1,0.1,0.8,0.8], projection='3d')

*# 使用线条绘制马鞍面*

x, y=np.mgrid[-5:5:20j,-5:5:20j];

z = (x\*\*2 - y\*\*2)/2

ax.plot\_trisurf(x.flat, y.flat, z.flat, label='3D三角面', cmap=plt.cm.get\_cmap('cool') )

**for** x\_,y\_,z\_ **in** zip(x.flat, y.flat, z.flat):

ax.text(x\_, y\_, z\_, 'A',color=(1,0,0,1))

ax.grid(b=**False**) *# 网格线*

plt.show()