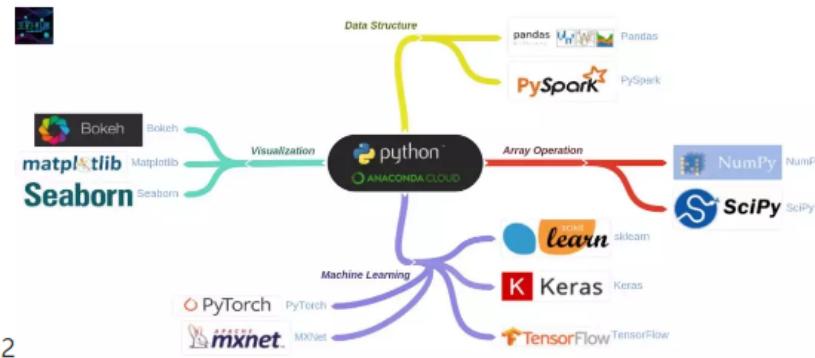


盘一盘 Python 系列 5 - Matplotlib

From:王圣元 王的机器 5/3



52

全文共 17354 字，167 幅图表截屏

预计阅读时间 44 分钟。

0 引言 <

本文是 Python 系列的第八篇

- [Python 入门篇 \(上\)](#)
- [Python 入门篇 \(下\)](#)
- [数组计算之 NumPy \(上\)](#)
- [数组计算之 NumPy \(下\)](#)
- [科学计算之 SciPy](#)
- [数据结构之 Pandas \(上\)](#)
- [数据结构之 Pandas \(下\)](#)
- **基本可视化之 Matplotlib**
- [统计可视化之 Seaborn](#)
- [交互可视化之 Bokeh](#)
- [炫酷可视化之 PyEcharts](#)
- [机器学习之 Sklearn](#)

- 深度学习之 TensorFlow
- 深度学习之 Keras
- 深度学习之 PyTorch
- 深度学习之 MXnet

Matplotlib 是 Python 中最基本的可视化工具，官网里 (<https://matplotlib.org/>) 有无数好资料，但这不是重点，本文肯定和市面上的所有讲解都不一样。

和 NumPy, SciPy, Pandas 一样，要用 Matplotlib，首先引用其库。

```
1 import matplotlib
```

先来类比一下人类和 Matplotlib 画图过程。



想想平时我们怎么画图，是不是分三步

- 找画板
- 用调色板
- 画画

Matplotlib 模拟了类似过程，也分三步

- FigureCanvas
- Renderer
- Artist

上面是 Matplotlib 里的三层 API:

1. FigureCanvas 帮你确定画图的地方
2. Renderer 帮你把想画的东西展示在屏幕上
3. Artist 帮你用 Renderer 在 Canvas 上画图

95% 的用户 (我们这些凡人) 只需用 Artist 就能自由的在电脑上画图了。

下面代码就是给 matplotlib 起了个别名 mpl, 由于用 matplotlib.plot 比较多, 也给它起了个别名 plt。

```
1 import matplotlib as mpl
2 import matplotlib.pyplot as plt
3 %matplotlib inline
```

而 %matplotlib inline 就是在 Jupyter notebook 里面内嵌画图的,

在画图中, 个人偏好百度 Echarts 里面的一组颜色, 因此将其 hex 颜色代码定义出来留在后面用。其中**红色的 r_hex** 和**深青色的 dt_hex** 是大爱。

```
1 r_hex = '#dc2624'      # red,          RGB = 220,38,36
2 dt_hex = '#2b4750'      # dark teal,    RGB = 43,71,80
3 tl_hex = '#45a0a2'      # teal,         RGB = 69,160,162
4 r1_hex = '#e87a59'      # red,          RGB = 232,122,89
5 tl1_hex = '#7dcaa9'     # teal,         RGB = 125,202,169
6 g_hex = '#649E7D'       # green,        RGB = 100,158,125
7 o_hex = '#dc8018'       # orange,       RGB = 220,128,24
8 tn_hex = '#C89F91'      # tan,          RGB = 200,159,145
9 g50_hex = '#6c6d6c'     # grey-50,     RGB = 108,109,108
10 bg_hex = '#4f6268'     # blue grey,   RGB = 79,98,104
11 g25_hex = '#c7cccc'    # grey-25,     RGB = 199,204,207
```

	Red:	RGB = 220, 38, 36	Hex = #dc2624
	Dark Teal:	RGB = 43, 71, 80	Hex = #2b4750
	Teal:	RGB = 69, 160, 162	Hex = #45a0a2
	Red:	RGB = 232, 122, 89	Hex = #e87a59
	Teal:	RGB = 125, 202, 169	Hex = #7dcaa9
	Green:	RGB = 100, 158, 125	Hex = #649e7d
	Orange:	RGB = 220, 128, 24	Hex = #dc8018
	Tan:	RGB = 200, 159, 145	Hex = #c89f91
	Grey-50:	RGB = 108, 109, 108	Hex = #6c6d6c
	Blue Grey:	RGB = 79, 98, 104	Hex = #4f6268
	Grey-25:	RGB = 199, 204, 207	Hex = #c7cccf

本章我们用以下思路来讲解：

- 第一章介绍 matplotlib 中的绘图逻辑，图包含的重要元素和他们之间的层级 (hierarchy)
- 第二章只关注折线图 (line chart)，但是一步步从最初的烂图完善到最终的美图。
这样可以把一种类型的图中的性质吃透，类比到其他类型的图一点也不难。
- 第三章从画图的四大目的出发，即**分布、联系、比较和构成**，介绍了相对应的直方图 (histogram chart)，散点图 (scatter chart)，折线图 (line chart) 和饼状图 (pie chart)。**这章偏向于用合适的图来实现不同的目的，没有在如何完善图的方面上下功夫，但在最后一节提到了如何画出使信息更有效的表达的图。**

注：第二、三章的例子都用金融数据举例，在公众号对话框输入 data 可下载数据。

本帖目录如下：

第一章 - Matplotlib 101

1.1 概览

目录

- 1.2 图
- 1.3 坐标系 & 子图
- 1.4 坐标轴
- 1.5 刻度
- 1.6 基础元素

第二章 - 画美感图

- 2.1 画第一幅图
- 2.2 图的默认设置
- 2.3 设置尺寸和 DPI
- 2.4 设置颜色-风格-宽度
- 2.5 设置边界
- 2.6 设置刻度标签
- 2.7 添加图例
- 2.8 添加第二幅图
- 2.9 两个坐标系 & 两幅子图
- 2.10 设置标注
- 2.11 设置透明度
- 2.12 完善细节

第三章 - 画有效图

- 3.1 概览
- 3.2 直方图
- 3.3 散点图
- 3.4 折线图
- 3.5 饼状图
- 3.6 同理心

总结





1 Matplotlib 101

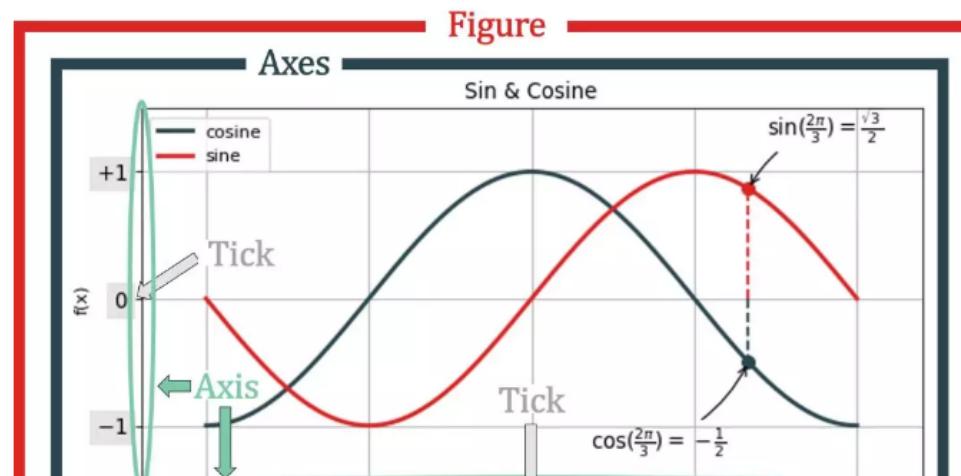
1.1 概览

Matplotlib 是一个巨无霸，乍一看无从下手，只能分解之后各点击破。总体来说，它包含两类元素：

- **基础 (primitives) 类**：线 (line), 点 (marker), 文字 (text), 图例 (legend), 网格 (grid), 标题 (title), 图片 (image) 等。
- **容器 (containers) 类**：图 (figure), 坐标系 (axes), 坐标轴 (axis) 和刻度 (tick)

基础类元素是我们想画出的标准对象，而容器类元素是基础类元素的寄居处，它们也有层级结构。

图 → 坐标系 → 坐标轴 → 刻度



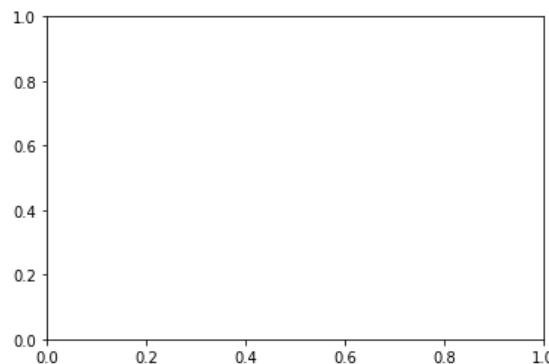


由上图看出：

- 图包含着坐标系 (多个)
- 坐标系由坐标轴组成 (横轴 `xAxis` 和纵轴 `yAxis`)
- 坐标轴上面有刻度 (主刻度 `MajorTicks` 和副刻度 `MinorTicks`)

Python 中万物皆对象，Matplotlib 里这些元素也都是对象。下面代码打印出坐标系、坐标轴和刻度。

```
1 fig = plt.figure()
2 ax = fig.add_subplot(1,1,1)
3 plt.show()
4
5 xax = ax.xaxis
6 yax = ax.yaxis
7
8 print('fig.axes:', fig.axes, '\n')
9 print('ax.xaxis:', xax )
10 print('ax.yaxis:', yax, '\n' )
11 print('ax.xaxis.majorTicks:', xax.majorTicks, '\n' )
12 print('ax.yaxis.majorTicks:', yax.majorTicks, '\n' )
13 print('ax.xaxis.minorTicks:', xax.minorTicks )
14 print('ax.yaxis.minorTicks:', yax.minorTicks )
```



```
fig.axes: []

ax.xaxis: XAxis(54.000000,36.000000)
ax.yaxis: YAxis(54.000000,36.000000)

ax.xaxis.majorTicks: [<matplotlib.axis.XTick object at 0x000001ED434A8860>,
<matplotlib.axis.XTick object at 0x000001ED434A8160>, <matplotlib.axis.XTick object at 0x000001ED434D72B0>, <matplotlib.axis.XTick object at 0x000001ED434D7550>, <matplotlib.axis.XTick object at 0x000001ED434D7C88>, <matplotlib.axis.XTick object at 0x000001ED434E21D0>]

ax.yaxis.majorTicks: [<matplotlib.axis.YTick object at 0x000001ED434BB5C0>,
<matplotlib.axis.YTick object at 0x000001ED434A8EB8>, <matplotlib.axis.YTick object at 0x000001ED434E26D8>, <matplotlib.axis.YTick object at 0x000001ED434E2B00>, <matplotlib.axis.YTick object at 0x000001ED434E2C88>, <matplotlib.axis.YTick object at 0x000001ED434E24E0>]

ax.xaxis.minorTicks: [<matplotlib.axis.XTick object at 0x000001ED434C60B8>]
ax.yaxis.minorTicks: [<matplotlib.axis.YTick object at 0x000001ED434CE208>]
```

从打印结果可看出坐标系、坐标轴和刻度都是对象。细看一下发现 xaxis 和 yaxis 上面都有 6 个主刻度 (majorTicks)。

此外，由坐标系和坐标轴指向同一个图 (侧面验证了图、坐标系和坐标轴的层级性)。

```
1 print('axes.figure:', ax.figure)
2 print('xaxis.figure:', xax.figure)
3 print('yaxis.figure:', yax.figure)
```

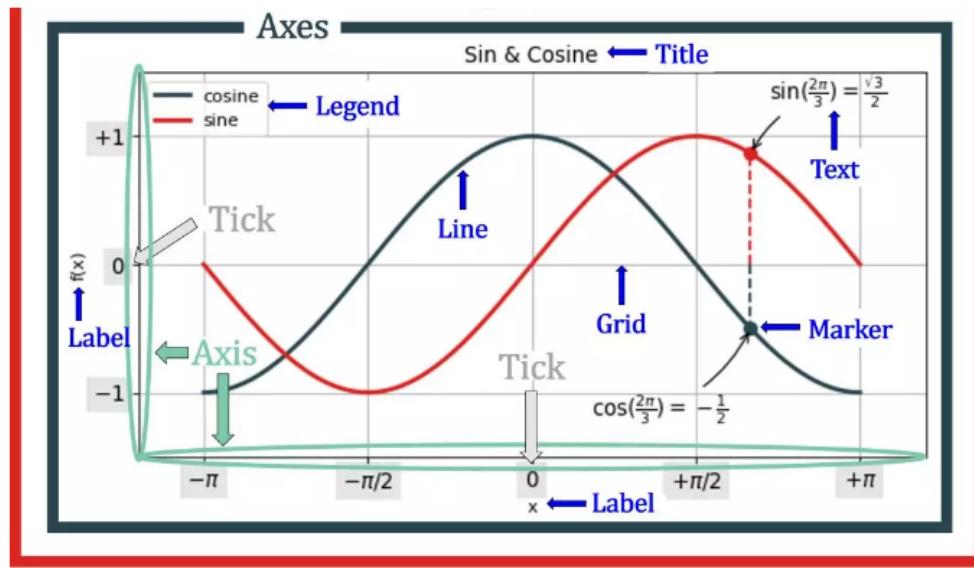
```
axes.figure: Figure(432x288)
xaxis.figure: Figure(432x288)
yaxis.figure: Figure(432x288)
```

创造完以上四个容器元素后，我们可在上面添加各种基础元素，比如：

- 在坐标轴和刻度上添加标签
- 在坐标系中添加线、点、网格、图例和文字
- 在图中添加图例

如下图所示：

Figure



接下来四节分别介绍四大容器，让我们先从「图」开始。

1.2 图

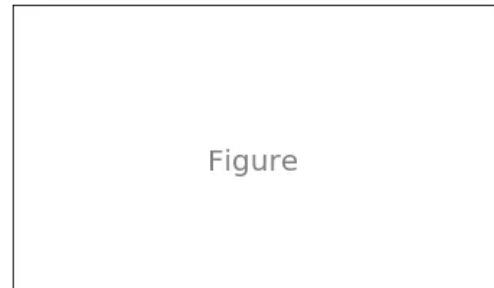
图是整个层级的顶部。

在图中可以添加基本元素「文字」。

```

1 plt.figure()
2 plt.text( 0.5, 0.5, 'Figure', ha='center',
3           va='center', size=20, alpha=.5 )
4 plt.xticks([]), plt.yticks([])
5 plt.show()

```



Figure

用 `plt.text()` 函数，其参数解释如下：

- 第一、二个参数是指横轴和纵轴坐标
- 第三个参数字符是指要显示的内容
- `ha`, `va` 是横向和纵向位置
- `size` 设置字体大小
- `alpha` 设置字体透明度 (0.5 是半透明)

在图中可以添加基本元素「图片」。

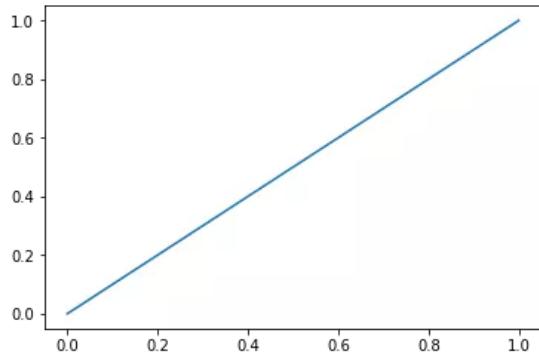
```
1 from PIL import Image
2 plt.figure()
3 plt.xticks([]), plt.yticks([])
4 im = np.array(Image.open('Houston Rockets.png'))
5 plt.imshow(im)
6 plt.show()
```



用 `Image.open()` 将图片转成像素存在 `ndarray` 中，再用 `plt.imshow()` 展示。

在图中可以添加基本元素「折线」。

```
1 plt.figure()
2 plt.plot([0,1],[0,1])
3 plt.show()
```



`plt.plot()` 函数是用来画折线图的，前两个参数分别是 `x` 和 `y`，该函数会在第二节细讲。

当我们每次说画东西，看起来是在图 (Figure) 里面进行的，实际上是在坐标系 (Axes) 里面进行的。一幅图中可以有多个坐标系，因此在坐标系里画东西更方便 (有些设置使用起来也更灵活)。

下面来看看层级中排名第二的「坐标系」。

1.3 坐标系 & 子图

一幅图 (Figure) 中可以有多个坐标系 (Axes)，那不是说一幅图中有多幅子图 (Subplot)，因此坐标系和子图是不是同样的概念？

在绝大多数情况下是的，两者有一点细微差别：

- 子图在母图中的网格结构一定是规则的
- 坐标系在母图中的网格结构可以是不规则的

由此可见，子图是坐标系的一个特例，来我们先研究特例。

子图

把图想成矩阵，那么子图就是矩阵中的元素，因此可像定义矩阵那样定义子图 - (行数、列

数、第几个子图)。

```
subplot(rows, columns, i-th plots)
```

文字解释起来有些晦涩，看代码和图就好懂了。

1×2 子图

```
1 plt.subplot(2,1,1)
2 plt.xticks([]), plt.yticks([])
3 plt.text( 0.5, 0.5, 'subplot(2,1,1)', ha='center', va='center', size=20, alpha=.5 )
4
5 plt.subplot(2,1,2)
6 plt.xticks([]), plt.yticks([])
7 plt.text( 0.5, 0.5, 'subplot(2,1,2)', ha='center', va='center', size=20, alpha=.5 )
8
9 plt.show()
```



```
subplot(2,1,1)
```



```
subplot(2,1,2)
```

这两个子图类似于一个列向量

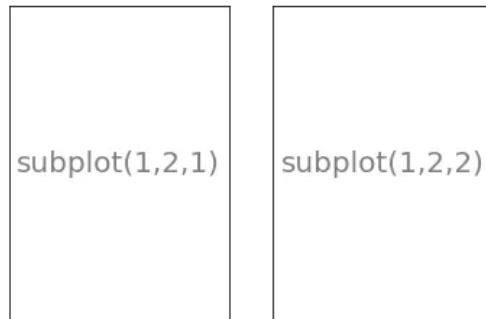
- `subplot(2, 1, 1)` 是第一幅
- `subplot(2, 1, 2)` 是第二幅

声明完子图后，下面所有代码就只在这幅子图上生效，直到声明下一幅子图。

2×1 子图

```
1 plt.subplot(1,2,1)
2 plt.xticks([]), plt.yticks([])
3 plt.text( 0.5, 0.5, 'subplot(1,2,1)', ha='center', va='center', size=20, alpha=.5 )
4
```

```
5 plt.subplot(1,2,2)
6 plt.xticks([]), plt.yticks([])
7 plt.text( 0.5, 0.5, 'subplot(1,2,2)', ha='center', va='center', size=20, alpha=.5 )
8
9 plt.show()
```



这两个子图类似于一个行向量

- `subplot(1, 2, 1)` 是第一幅
- `subplot(1, 2, 2)` 是第二幅

2x2 子图

```
1 fig, axes = plt.subplots(nrows=2, ncols=2)
2
3 for i, ax in enumerate(axes.flat):
4     ax.set( xticks=[], yticks=[] )
5     s = 'subplot(2,2,' + str(i) + ')'
6     ax.text( 0.5, 0.5, s, ha='center', va='center', size=20, alpha=.5 )
7
8 plt.show()
```



这次我们用过坐标系来生成子图 (子图是坐标系的特例嘛), 第 1 行

```
fig, axes = plt.subplots(nrows=2, ncols=2)
```

得到的 `axes` 是一个 2×2 的对象。在第 3 行的 `for` 循环中用 `axes.flat` 将其打平，然后在每个 `ax` 上生成子图。

坐标系

坐标系比子图更通用，有两种生成方式

- 用 `gridspec` 包加上 `subplot()`
- 用 `plt.axes()`

不规则网格

```
1 import matplotlib.gridspec as gridspec
2 G = gridspec.GridSpec(3, 3)
3
4 ax1 = plt.subplot(G[0, :])
5 plt.xticks([]), plt.yticks([])
6 plt.text( 0.5, 0.5, 'Axes 1', ha='center', va='center', size=20, alpha=.5 )
7
8 ax2 = plt.subplot(G[1, :-1])
9 plt.xticks([]), plt.yticks([])
10 plt.text( 0.5, 0.5, 'Axes 2', ha='center', va='center', size=20, alpha=.5 )
11
12 ax3 = plt.subplot(G[1:, -1])
13 plt.xticks([]), plt.yticks([])
14 plt.text( 0.5, 0.5, 'Axes 3', ha='center', va='center', size=20, alpha=.5 )
15
16 ax4 = plt.subplot(G[-1, 0])
17 plt.xticks([]), plt.yticks([])
18 plt.text( 0.5, 0.5, 'Axes 4', ha='center', va='center', size=20, alpha=.5 )
19
20 ax5 = plt.subplot(G[-1, -2])
21 plt.xticks([]), plt.yticks([])
22 plt.text( 0.5, 0.5, 'Axes 5', ha='center', va='center', size=20, alpha=.5 )
23
24 plt.show()
```

Axes 1



第 2 行将整幅图分成 3×3 份赋值给 G, 第 4, 8, 12, 16, 20 行分别用

```
plt.subplot(G[1])
```

生成五个坐标系。G[] 里面的切片和 Numpy 数组用法一样:

- $G[0, :] =$ 图的第一行 (Axes 1)
- $G[1, :-1] =$ 图的第二行, 第二三列 (Axes 2)
- $G[1:, -1] =$ 图的第二三行, 第三列 (Axes 3)
- $G[-1, 0] =$ 图的第三行, 第一列 (Axes 4)
- $G[-1, -2] =$ 图的第三行, 第二列 (Axes 5)

大图套小图

```
1 plt.axes([0.1,0.1,0.8,0.8])
2 plt.xticks([]), plt.yticks([])
3 plt.text( 0.6, 0.6, 'axes([0.1,0.1,0.8,0.8])', ha='center', va='center', size=20, alpha=.5 )
4
5 plt.axes([0.2,0.2,0.3,0.3])
6 plt.xticks([]), plt.yticks([])
7 plt.text( 0.5, 0.5, 'axes([0.2,0.2,0.3,0.3])', ha='center', va='center', size=11, alpha=.5 )
8
9 plt.show()
```



第 1 和 5 行分别用

```
plt.axes([l, b, w, h])
```

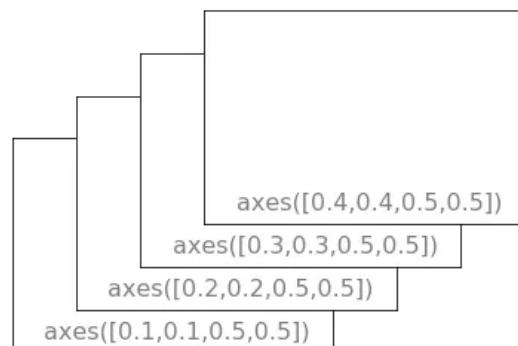
其中 [l, b, w, h] 可以定义坐标系

- l 代表坐标系左边到 Figure 左边的水平距离
- b 代表坐标系底边到 Figure 底边的垂直距离
- w 代表坐标系的宽度
- h 代表坐标系的高度

如果 l, b, w, h 都小于 1，那它们是标准化 (normalized) 后的距离。比如 Figure 底边长度为 10，坐标系底边到它的垂直距离是 2，那么 $b = 2/10 = 0.2$ 。

重叠图

```
1 plt.axes([0.1,0.1,0.5,0.5])
2 plt.xticks([]), plt.yticks([])
3 plt.text( 0.1, 0.1, 'axes([0.1,0.1,0.5,0.5])', ha='left', va='center', size=16, alpha=.5 )
4
5 plt.axes([0.2,0.2,0.5,0.5])
6 plt.xticks([]), plt.yticks([])
7 plt.text( 0.1, 0.1, 'axes([0.2,0.2,0.5,0.5])', ha='left', va='center', size=16, alpha=.5 )
8
9 plt.axes([0.3,0.3,0.5,0.5])
10 plt.xticks([]), plt.yticks([])
11 plt.text( 0.1, 0.1, 'axes([0.3,0.3,0.5,0.5])', ha='left', va='center', size=16, alpha=.5 )
12
13 plt.axes([0.4,0.4,0.5,0.5])
14 plt.xticks([]), plt.yticks([])
15 plt.text( 0.1, 0.1, 'axes([0.4,0.4,0.5,0.5])', ha='left', va='center', size=16, alpha=.5 )
16
17 plt.show()
```



不解释，懂了 [l, b, w, h] 的意思这幅重叠图应该知道怎么生成了。

在本小节最后，总结一下两种生成单个坐标系的方法（生成多个坐标系可以类推）。

两种生成坐标系的推荐代码

代码 1

同时生成图和坐标系。

```
1 fig, ax = plt.subplots()
2 ax.set( xticks=[], yticks=[] )
3 s = 'Style 1\n\nfig, ax = plt.subplots()\nax.plot()'
4 ax.text( 0.5, 0.5, s, ha='center', va='center', size=16, alpha=.5 );
```

Style 1

```
fig, ax = plt.subplots()
ax.plot()
```

代码 2

先生成图，再添加坐标系。

```
1 fig = plt.figure()
2 ax = fig.add_subplot(1,1,1)
3 ax.set( xticks=[], yticks=[] )
4 s = 'Style 2\n\nfig = plt.subplots()\nax = fig.add_subplot(1,1,1)\nax.plot()'
5 ax.text( 0.5, 0.5, s, ha='center', va='center', size=16, alpha=.5 );
```

Style 2

```
fig = plt.subplots()  
ax = fig.add_subplot(1,1,1)  
ax.plot()
```

超级细心的读者可能会发现，上面所有的图都看不到坐标轴和刻度啊，是的，我是故意这样做的，在深度研究坐标系和子图时，剔除不必要的信息，用的是以下代码 (将刻度设为空集 []):

```
plt.xticks([]), plt.yticks([])
```

或

```
ax.set(xticks=[], yticks=[])
```

现在是时候来看看层级中排名第三的「坐标轴」。

1.4 坐标轴

一个坐标系 (Axes)，通常是二维，有两条坐标轴 (Axis):

- 横轴: XAxis
- 纵轴: YAxis

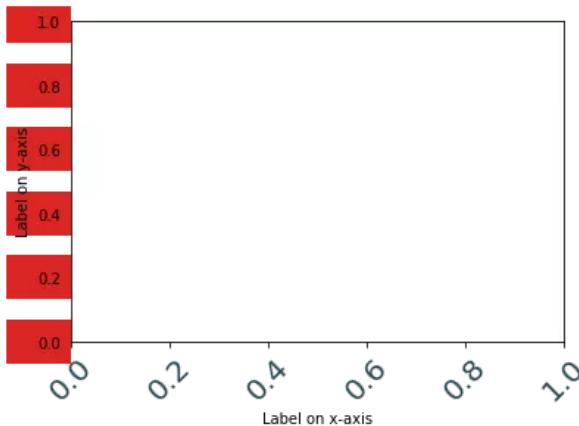
每个坐标轴都包含两个元素

- 容器类元素「刻度」，该对象里还包含**刻度本身**和**刻度标签**
- 基础类元素「标签」，该对象包含的是**坐标轴标签**

「刻度」和「标签」都是对象，下面代码通过改变它们一些属性值来进行可视化。

```
1 fig, ax = plt.subplots()  
2 ax.set_xlabel('Label on x-axis')  
3 ax.set_ylabel('Label on y-axis')  
4  
5 for label in ax.xaxis.get_ticklabels():  
6     # label is a Text instance
```

```
7     label.set_color( dt_hex )
8     label.set_rotation(45)
9     label.set_fontsize(20)
10
11    for line in ax.yaxis.get_ticklines():
12        # line is a Line2D instance
13        line.set_color( r_hex )
14        line.set_markersize(500)
15        line.set_markeredgewidth(30)
16
17    plt.show()
```



第 2 和 3 行打印出 x 轴和 y 轴的标签。

第 5 到 9 行处理「刻度」对象里的刻度标签，将它颜色设定为深青色，字体大小为 20，旋转度 45 度。

第 11 到 15 行处理「标签」对象的刻度本身 (即一条短线)，将它颜色设定为红色，标记长度和宽度为 500 和 30 (夸张了些，但就为大家看清楚这条代表刻度的短线！)。

万物皆对象，坐标轴也不例外，下面代码打印出 x 轴的标签、刻度位置点、刻度标签、刻度线，刻度标签位置、主刻度。

```
1 print( ax.xaxis.get_label() )
2 print( ax.xaxis.get_ticklocs() )
3 print( ax.xaxis.get_ticklabels() )
```

```
4 print( ax.xaxis.get_ticklines() )
5 print( ax.xaxis.get_ticks_position() )
6 print( ax.xaxis.get_major_ticks() )
```

```
Text(0.5, 17.20000000000003, 'Label on x-axis')
[0. 0.2 0.4 0.6 0.8 1.]
<a list of 6 Text major ticklabel objects>
<a list of 12 Line2D ticklines objects>
bottom

[<matplotlib.axis.XTick object at 0x000001ED438A95C0>,
 <matplotlib.axis.XTick object at 0x000001ED438A1F28>,
 <matplotlib.axis.XTick object at 0x000001ED43886940>,
 <matplotlib.axis.XTick object at 0x000001ED438C92E8>,
 <matplotlib.axis.XTick object at 0x000001ED438C98D0>,
 <matplotlib.axis.XTick object at 0x000001ED438C9DAO>]
```

问题：其他都好懂，比如 6 个刻度标签，但为什么有 12 条刻度线？不应该是 6 条吗？我查了半天资料都查不到，知道答案的同学可留言。

我的猜测是，当每条刻度线加粗后的像是一个刻度矩形，由 2 条刻度线组成，那么总共就是 12 条？

下面来看看层级中排名第四也是最后的「刻度」。

1.5 刻度

刻度 (Tick) 其实在坐标轴那节已经讲过了，它核心内容就是

- 一条短线 (刻度本身)
- 一串字符 (刻度标签)

本节想继续深挖刻度，即看看不同类型的刻度，需要用到各种各样的 Tick Locators。

前期工作

为了显示不同类型的刻度，首先定义一个 `setup(ax)` 函数，主要功能有

- 去除左纵轴 (y 轴)、右纵轴和上横轴
- 去除 y 轴上的刻度
- 将 x 轴上的刻度位置定在轴底
- 设置主刻度和副刻度的长度和宽度
- 设置 x 轴和 y 轴的边界
- 将图中 patch 设成完全透明

```
1 import matplotlib.ticker as ticker
2
3 def setup(ax):
4     ax.spines['right'].set_color('none')
5     ax.spines['left'].set_color('none')
6     ax.spines['top'].set_color('none')
7     ax.yaxis.set_major_locator(ticker.NullLocator())
8     ax.xaxis.set_ticks_position('bottom')
9
10    ax.tick_params(which='major', width=2.00)
11    ax.tick_params(which='major', length=10)
12    ax.tick_params(which='minor', width=0.75)
13    ax.tick_params(which='minor', length=2.5)
14
15    ax.set_xlim(0, 5)
16    ax.set_ylim(0, 1)
17
18    ax.patch.set_alpha(0.0)
```

注：这些操作都是为了下面显示刻度用，往下看就知道为什么这么做的。

为了感受一下上面每个操作对原图产生的效果，我们画出 6 个子图，其中

- 第一幅是原图
- 第二幅处理左、右、上轴
- 第三幅处理刻度标签
- 第四幅处理刻度尺寸
- 第五幅处理坐标轴边界
- 第六幅处理颜色和透明度

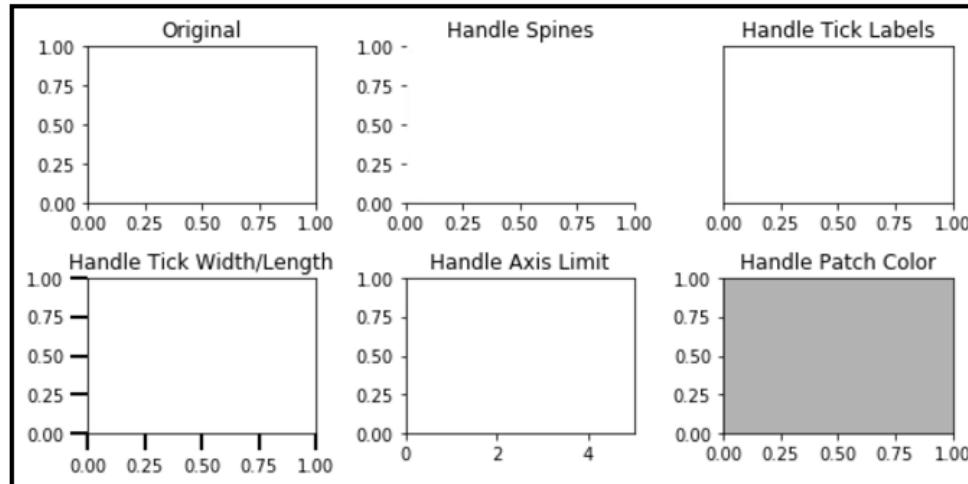
代码如下：

```
1 fig, axes = plt.subplots( nrows=2, ncols=3, figsize=(8, 4) )
2
3 axes[0,0].set_title('Original')
```

```

4
5     axes[0,1].spines['right'].set_color('none')
6     axes[0,1].spines['left'].set_color('none')
7     axes[0,1].spines['top'].set_color('none')
8     axes[0,1].set_title('Handle Spines')
9
10    axes[0,2].yaxis.set_major_locator(ticker.NullLocator())
11    axes[0,2].xaxis.set_ticks_position('bottom')
12    axes[0,2].set_title('Handle Tick Labels')
13
14    axes[1,0].tick_params(which='major', width=2.00)
15    axes[1,0].tick_params(which='major', length=10)
16    axes[1,0].tick_params(which='minor', width=0.75)
17    axes[1,0].tick_params(which='minor', length=2.5)
18    axes[1,0].set_title('Handle Tick Width/Length')
19
20    axes[1,1].set_xlim(0, 5)
21    axes[1,1].set_ylim(0, 1)
22    axes[1,1].set_title('Handle Axis Limit')
23
24    axes[1,2].patch.set_color('black')
25    axes[1,2].patch.set_alpha(0.3)
26    axes[1,2].set_title('Handle Patch Color')
27
28    plt.tight_layout()
29    plt.show()

```



将上面效果全部合并，这个 `setup(ax)` 就是把坐标系里所有元素都去掉，只留 x 轴来添加各种刻度。

刻度展示

不同的 `locator()` 可以生成不同的刻度对象，我们来研究以下 8 种：

1. `NullLocator()`: 空刻度

2. `MultipleLocator(a)`: 刻度间隔 = 标量 a
3. `FixedLocator(a)`: 刻度位置由数组 a 决定
4. `LinearLocator(a)`: 刻度数目 = a, a 是标量
5. `IndexLocator(b, o)`: 刻度间隔 = 标量 b, 偏移量 = 标量 o
6. `AutoLocator()`: 根据默认设置决定
7. `MaxNLocator(a)`: 最大刻度数目 = 标量 a
8. `LogLocator(b, n)`: 基数 = 标量 b, 刻度数目 = 标量 n

代码如下：

```

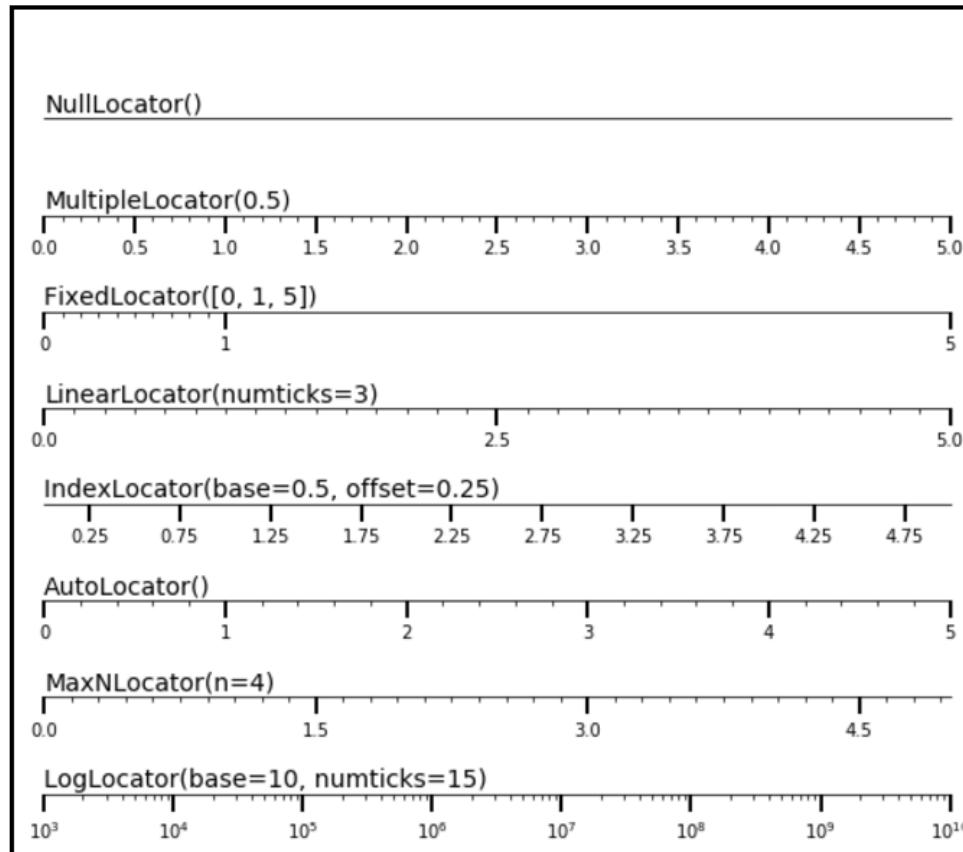
1 plt.figure(figsize=(8, 6))
2 n = 8
3
4 # Null Locator
5 ax = plt.subplot(n, 1, 1)
6 setup(ax)
7 ax.xaxis.set_major_locator(ticker.NullLocator())
8 ax.xaxis.set_minor_locator(ticker.NullLocator())
9 ax.text(0.0, 0.1, "NullLocator()", fontsize=14, transform=ax.transAxes)
10
11 # Multiple Locator
12 ax = plt.subplot(n, 1, 2)
13 setup(ax)
14 ax.xaxis.set_major_locator(ticker.MultipleLocator(0.5))
15 ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.1))
16 ax.text(0.0, 0.1, "MultipleLocator(0.5)", fontsize=14, transform=ax.transAxes)
17
18 # Fixed Locator
19 ax = plt.subplot(n, 1, 3)
20 setup(ax)
21 majors = [0, 1, 5]
22 ax.xaxis.set_major_locator(ticker.FixedLocator(majors))
23 minors = np.linspace(0, 1, 11)[1:-1]
24 ax.xaxis.set_minor_locator(ticker.FixedLocator(minors))
25 ax.text(0.0, 0.1, "FixedLocator([0, 1, 5])", fontsize=14, transform=ax.transAxes)
26
27 # Linear Locator
28 ax = plt.subplot(n, 1, 4)
29 setup(ax)
30 ax.xaxis.set_major_locator(ticker.LinearLocator(3))
31 ax.xaxis.set_minor_locator(ticker.LinearLocator(31))
32 ax.text(0.0, 0.1, "LinearLocator(numticks=3)", fontsize=14, transform=ax.transAxes)
33
34 # Index Locator
35 ax = plt.subplot(n, 1, 5)
36 setup(ax)
37 ax.plot(range(0, 5), [0]*5, color='White')
38 ax.xaxis.set_major_locator(ticker.IndexLocator(base=.5, offset=.25))
39 ax.text(0.0, 0.1, "IndexLocator(base=0.5, offset=0.25)", fontsize=14, transform=ax.transAxes)
40
41 # Auto Locator
42 ax = plt.subplot(n, 1, 6)
43 setup(ax)
44 ax.xaxis.set_major_locator(ticker.AutoLocator())
45 ax.xaxis.set_minor_locator(ticker.AutoMinorLocator())
46 ax.text(0.0, 0.1, "AutoLocator()", fontsize=14, transform=ax.transAxes)
47

```

```

48 # MaxN Locator
49 ax = plt.subplot(n, 1, 7)
50 setup(ax)
51 ax.xaxis.set_major_locator(ticker.MaxNLocator(4))
52 ax.xaxis.set_minor_locator(ticker.MaxNLocator(48))
53 ax.text(0.0, 0.1, "MaxNLocator(n=4)", fontsize=14, transform=ax.transAxes)
54
55 # Log Locator
56 ax = plt.subplot(n, 1, 8)
57 setup(ax)
58 ax.set_xlim(10**3, 10**10)
59 ax.set_xscale('log')
60 ax.xaxis.set_major_locator(ticker.LogLocator(base=10.0, numticks=15))
61 ax.text(0.0, 0.1, "LogLocator(base=10, numticks=15)", fontsize=14, transform=ax.transAxes)
62
63 # Push the top of the top axes outside the figure because we only show the bottom spine.
64 plt.subplots_adjust( left=0.05, right=0.95, bottom=0.05, top=1.05 )
65
66 plt.show()

```



上图其实包含 8 个子图，但只含有 x 轴，这也是为什么要先定一个 `setup(ax)` 函数来只保留 x 轴。另外将 patch 的 alpha 设为 0 使其完全透明，就是为了有些标签在子图交界处还能显示出来。感兴趣的同学可以把 alpha 设成 1 看看上图变成什么样子了。

目前，我们已经介绍四个最重要的容器以及它们之间的层级

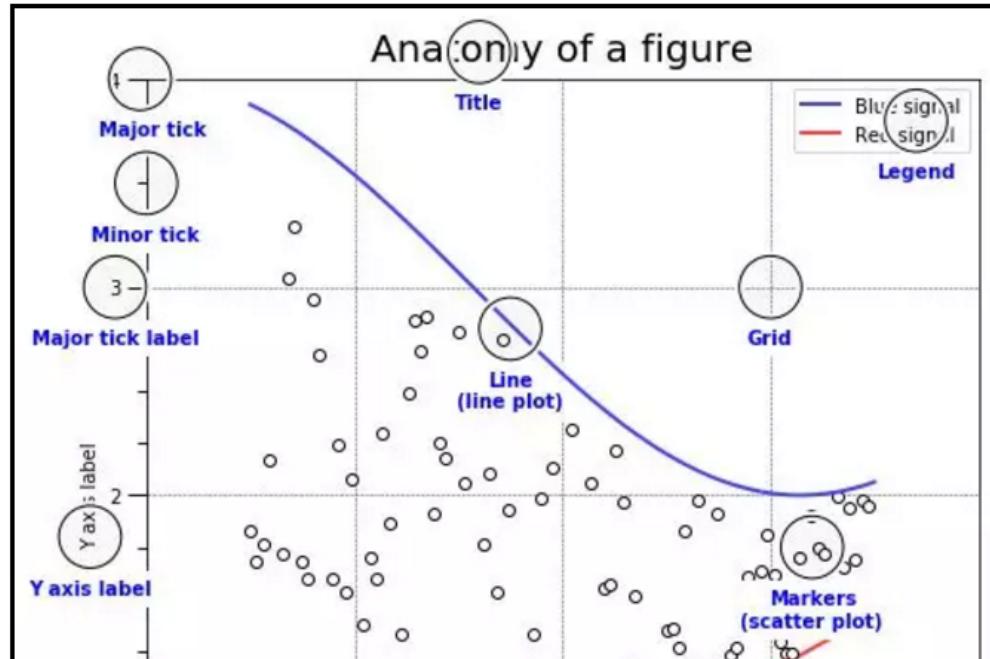
Figure → Axes → Axis → Ticks

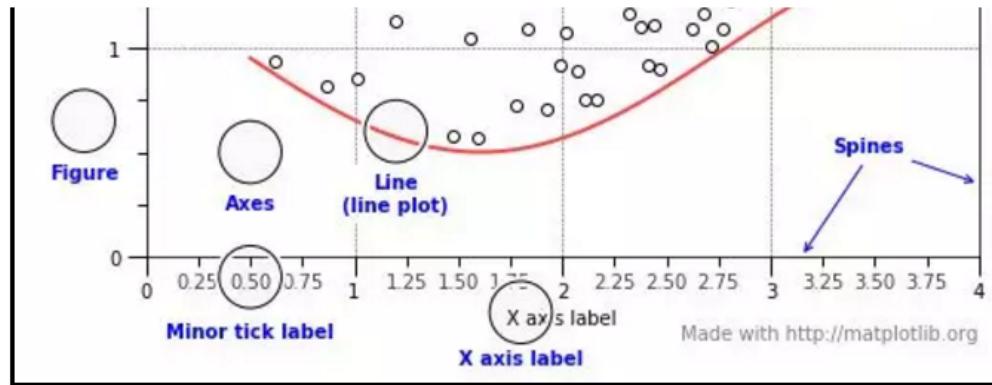
图 → 坐标系 → 坐标轴 → 刻度

但要画出一幅有内容的图，还需要在容器里添加基础元素比如线 (line)，点 (marker)，文字 (text)，图例 (legend)，网格 (grid)，标题 (title)，图片 (image) 等，具体来说

- 画一条线，用 `plt.plot()` 或 `ax.plot()`
- 画个记号，用 `plt.scatter()` 或 `ax.scatter()`
- 添加文字，用 `plt.text()` 或 `ax.text()`
- 添加图例，用 `plt.legend()` 或 `ax.legend()`
- 添加图片，用 `plt.imshow()` 或 `ax.imshow()`

最后用 Matplotlib 官网的图来总结所有元素。





现在你基本理解了 Matplotlib 里面的绘图逻辑和元素，下两节分别从不同维度 (深度和广度) 研究如何画图：

- 第二节只研究一种类型的图「折线图」，但从头到尾不断根据需求添加元素完善它。
深度研究做到完美！
- 第三节研究四种类型的图 (展示数据的分布、联系、对比和组成)，却没在美感上下功夫，广度研究满足目的！

但读完后两节后，你应该可以在各种类型的图上做到完美。



2 画美感图 <

本节记录着老板让斯蒂文绘图不断提需求直到把他逼疯的一段对话。



我是老板，我要求很高。



我是斯蒂文，我能力很强。

2.1 画第一幅图



画一幅标准普尔 500 指数在 2007-2010 的走势图。



小菜一碟，用 Pandas 读取数据存成 DataFrame，再用 Matplotlib 里的 plt.plot() 画图。

首先用 `pd.read_csv` 函数从存好的 S&P500.csv，截屏如下：

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Adj Close	Volume
2	03/01/1950	16.66	16.66	16.66	16.66	16.66	1260000
3	04/01/1950	16.85	16.85	16.85	16.85	16.85	1890000
4	05/01/1950	16.93	16.93	16.93	16.93	16.93	2550000
5	06/01/1950	16.98	16.98	16.98	16.98	16.98	2010000
6	09/01/1950	17.08	17.08	17.08	17.08	17.08	2520000
7	10/01/1950	17.03	17.03	17.03	17.03	17.03	2160000
8	11/01/1950	17.09	17.09	17.09	17.09	17.09	2630000
9	12/01/1950	16.76	16.76	16.76	16.76	16.76	2970000
10	13/01/1950	16.67	16.67	16.67	16.67	16.67	3330000

该函数中三个参数代表：

- `index_col = 0` 是说把第一列 Date 当成行标签 (index)
- `parse_dates = True` 是说把行标签转成 date 对象
- `dayFirst = True` 是说日期是 DD/MM/YYYY 这样的格式

```
1 data = pd.read_csv('S&P500.csv',
2                     index_col=0,
3                     parse_dates=True,
4                     dayfirst=True)
5 data.head(3).append(data.tail(3))
```

Open High Low Close Adj Close Volume

Date

1950-01-03	16.660000	16.660000	16.660000	16.660000	16.660000	1260000
1950-01-04	16.850000	16.850000	16.850000	16.850000	16.850000	1890000
1950-01-05	16.930000	16.930000	16.930000	16.930000	16.930000	2550000
2019-04-22	2898.780029	2909.510010	2896.350098	2907.969971	2907.969971	2997950000
2019-04-23	2909.989990	2936.310059	2908.530029	2933.679932	2933.679932	3635030000
2019-04-24	2934.000000	2936.830078	2926.050049	2927.250000	2927.250000	3448960000

S&P 500 的数据从 1950 年 1 月 3 号开始，老板只需要 2007 年 1 月 1 日到 2010 年 1 月 1 日的数据。做个切片即可，存储成 spx。

```
1 spx = data[['Adj Close']]
2     .loc['2007-01-01':'2010-01-01']
3 spx.head(3).append(spx.tail(3))
```

```
Adj Close
Date
2007-01-03 1416.599976
2007-01-04 1418.339966
2007-01-05 1409.709961
2009-12-29 1126.199951
2009-12-30 1126.420044
2009-12-31 1115.099976
```

spx 是个 DataFrame，将它的值一个个画出折线图只需用 `plt.plot()` 函数，展示在屏幕需用 `plt.show()`。

```
1 plt.plot( spx.values )
2 plt.show()
```





在 `plot()` 函数里面只有变量 `y` 时 (`y = spx.values`)，那么自变量就是默认赋值为 `range(len(y))`。

此外我们没有设置图的尺寸、像素、线的颜色宽度、坐标轴的刻度和标签、图例、标题等等，所有设置都用的是 `matplotlib` 的默认设置。

此图虽丑，但也满足了老板的需求，即标准普尔 500 指数在 2007-2010 的走势图。斯蒂文提交给了老板。

2.2 图的默认设置



这图也太丑了，而且横轴不是日期，我怎么知道金融危机发生在什么时候？再改一次！

要求合理，没问题。



要修改图就必须知道它的属性，用 `plt.rcParams` 可查看上图的所有默认属性（非常多的属性值）。

```
1 plt.rcParams
```

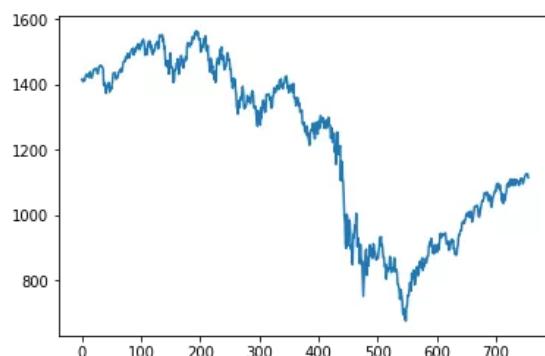
```
RcParams({_internal.classic_mode': False,
          'agg.path.chunksize': 0,
          'animation.avconv_args': [],
          'animation.avconv_path': 'avconv',
          'animation.bitrate': -1,
          'animation.codec': 'h264',
          'animation.convert_args': [],
          'animation.convert_path': 'convert',
          'animation.embed_limit': 20.0,
          'animation.ffmpeg_args': [],
          'animation.ffmpeg_path': 'ffmpeg',
          'animation.frame_format': 'png',
          'animation.html': 'none',
          'animation.html_args': [],
          'animation.writer': 'ffmpeg',
```

```
'axes.autolimit_mode': 'data',
'axes.axisbelow': 'line',
'axes.edgecolor': 'black',
```

看完上面的属性值后，斯蒂文决定在图表尺寸 (figsize)，每英寸像素点 (dpi)，线条颜色 (color)，线条风格 (linestyle)，线条宽度 (linewidth)，横纵轴刻度 (xticks, yticks)，横纵轴边界 (xlim, ylim) 做改进。那就先看看它们的默认属性值是多少。

```
1 print( 'figure size:', plt.rcParams['figure.figsize'] )
2 print( 'figure dpi:', plt.rcParams['figure.dpi'] )
3 print( 'line color:', plt.rcParams['lines.color'] )
4 print( 'line style:', plt.rcParams['lines.linestyle'] )
5 print( 'line width:', plt.rcParams['lines.linewidth'] )
6
7 fig = plt.figure()
8 ax = fig.add_subplot(1, 1, 1)
9 ax.plot( spx['Adj Close'].values )
10
11 print( 'xticks:', ax.get_xticks() )
12 print( 'yticks:', ax.get_yticks() )
13 print( 'xlim:', ax.get_xlim() )
14 print( 'ylim:', ax.get_ylim() )
```

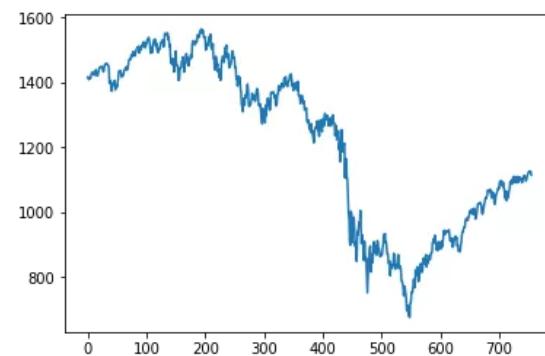
```
figure size: [6.0, 4.0]
figure dpi: 72.0
line color: c0
line style: -
line width: 1.5
xticks: [-100.    0.   100.   200.   300.   400.   500.   600.   700.   800.]
yticks: [ 600.   800.  1000.  1200.  1400.  1600.  1800.]
xlim: (-37.75, 792.75)
ylim: (632.0990292500001, 1609.58102375)
```



将属性值打印结果和图一起看一目了然。现在我们知道这张图大小是 6×4 , 每英寸像素有 72 个, 线颜色 C0 代表是蓝色, 风格 - 是连续线, 宽度 1.5, 等等。

斯蒂文现在有个“大胆”的想法, 把这些默认属性值显性的在代码中写出来, 画出来的跟什么设置都不写生成的图应该是一样的。来验证一下:

```
1 # Create a new figure of size 6x4 points, using 72 dots per inch
2 plt.figure( figsize=(6, 4), dpi=72 )
3
4 # Plot using blue color (C0) with a continuous line of width 1.5 (pixels)
5 plt.plot( spx.values, color='C0', linewidth=1.5, linestyle='-' )
6
7 # Set x ticks
8 plt.xticks( np.linspace(-100,800,10) )
9
10 # Set y ticks
11 plt.yticks( np.linspace(600,1800,7) )
12
13 # Set x limits
14 plt.xlim(-37.75, 792.75)
15
16 # Set y limits
17 plt.ylim(632.0990292500001, 1609.58102375)
18
19 # Show result on screen
20 plt.show()
```



一模一样!



... 你这不是瞎耽误时间吗?

不这样把默认属性值的显性写出来, 我怎么知道如何改进?



老板将信将疑。。。

2.3 设置尺寸和 DPI



图的尺寸差，3年数据这么长图宽点比较好。



属性名称在手，改 `figsize` 就行了。

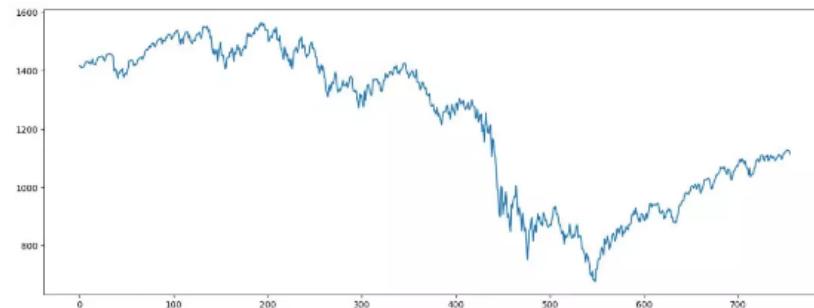
用 `figsize` 和 `dpi` 一起可以控制图的大小和像素。

函数 `figsize(w, h)` 决定图的宽和高 (单位是英寸)，而属性 `dpi` 全称 `dots per inches`，测量每英寸多少像素。两个属性一起用，那么得到的图的像素为

$(w*dpi, h*dpi)$

套用在下面代码中，我们其实将图的大小设置成 16×6 平方英寸，而像素设置成 $(1600, 600)$ ，因为 $dpi = 100$ 。

```
1 plt.figure( figsize=(16,6), dpi=100 )
2 plt.plot( spx.values )
3 plt.show()
```



运行代码生成大宽屏图！

2.4 设置颜色-风格-宽度



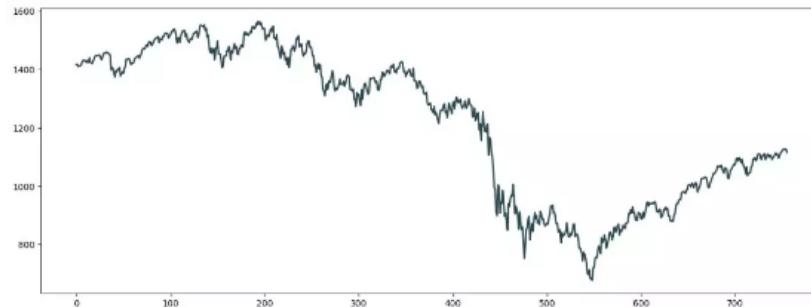
线条有点纤细，这种蓝也不高雅。



纤细？不高雅？Excuse me？

在 `plt.plot()` 用 `color`, `linewidth` 和 `linestyle` 属性一起可以控制折线的颜色 (上面定义的深青色)、宽度 (2 像素) 和风格 (连续线)。

```
1 plt.figure( figsize=(16,6), dpi=100 )
2 plt.plot( spx.values, color=dt_hex,
3           linewidth=2, linestyle='-' )
4 plt.show()
```



现在线条更明显了，而深青色看起来也比较有品位。

2.5 设置边界



横轴怎么是数字，不应该是日期么？另外这条线快碰到天花板了。

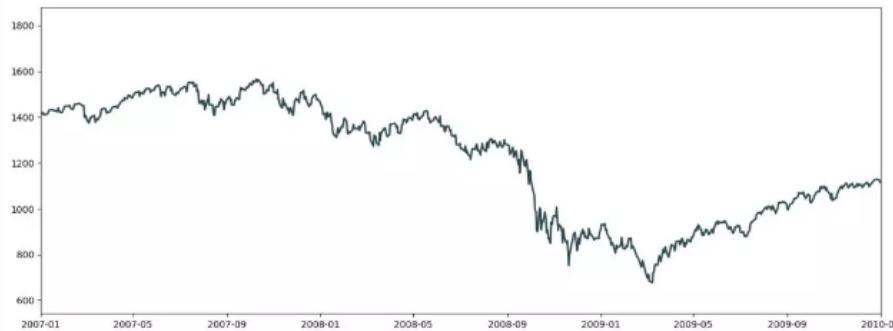


这个需求合理，设置下横轴和纵轴的边界就可以了。

下面代码第 2 行在图中 (fig) 添加了一个坐标系 (ax)，然后所有操作都在 ax 里面完成，比如用

- `ax.plot()` 来画折线
- `ax.set_xlim()`, `ax.set_ylim()` 来设置横轴和纵轴的边界

```
1 fig = plt.figure( figsize=(16,6), dpi=100 )
2 ax = fig.add_subplot(1,1,1)
3 x = spx.index
4 y = spx.values
5 ax.plot( x, y, color=dt_hex, linewidth=2, linestyle='-' )
6 ax.set_xlim(['1/1/2007', '1/1/2010'])
7 ax.set_ylim( y.min()*0.8, y.max()*1.2 );
```



第 3 行的 x 是日期 (回顾 spx 是一个 DataFrame，行标签是日期)。

第 6 行将横轴的上下边界设为 2007-01-01 和 2010-01-01，只好是整个时间序列的起始日和终止日。

第 7 行将纵轴的上下边界设为 spx 的最小值的 0.8 倍和最大值的 1.2 倍。

现在横轴的刻度标签都是日期，比数字刻度带来的信息多；而 spx 图离顶部也有空间，看起来没那么挤。

2.6 设置刻度和标签



横轴日期隔得有点开，而且只有年月，没有日。

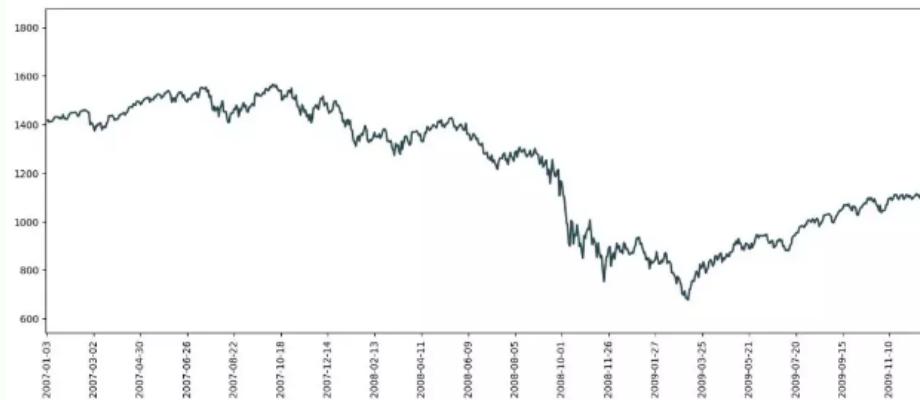
没有日？Interesting



上图横轴的刻度个数（老板说日期隔得有点开）和标签显示（老板说只有年月）都是默认设置。为了满足老板的要求，斯蒂文只能手动设置，用以下两个函数：

- 先用 `ax.set_ticks()` 设置出数值刻度
- 再用 `ax.set_xticklabels()` 在对应的数值刻度上写标签

```
1 fig = plt.figure( figsize=(16,6), dpi=100 )
2 ax = fig.add_subplot(1,1,1)
3 x = spx.index
4 y = spx.values
5 ax.plot( y, color=dt_hex, linewidth=2, linestyle='-' )
6
7 ax.set_xlim(-1, len(x)+1)
8 ax.set_ylim( y.min()*0.8, y.max()*1.2 )
9
10 ax.set_xticks( range(0,len(x),40) )
11 ax.set_xticklabels( [x[i].strftime('%Y-%m-%d') for i in ax.get_xticks()], \
12 rotation=90 );
```



第 7 行设置横轴的边界，下界是 -1，上界是 len(x) +1。

第 10 行先设置横轴「数值刻度」为 range(0,len(x), 40)，即 0, 40, 80, ...

第 11 行在这些「数值刻度」上写标签，即格式为 %Y-%m-%d 的日期。由于日期个数比较多，而且日期字符比较长，直接在图中显示出来会相互重叠非常难看。这里调节参数 rotation = 90 使得日期逆时针转了 90 度，看上图效果好多了。

现在横轴的刻度标签是带「年-月-日」的日期，而且标签的间隔刚刚好。

2.7 添加图例



怎么没有图例？能不能专业一点。



这不是 S&P 500 吗？一条时间序列要啥图例？难道接下来要画两条序列？

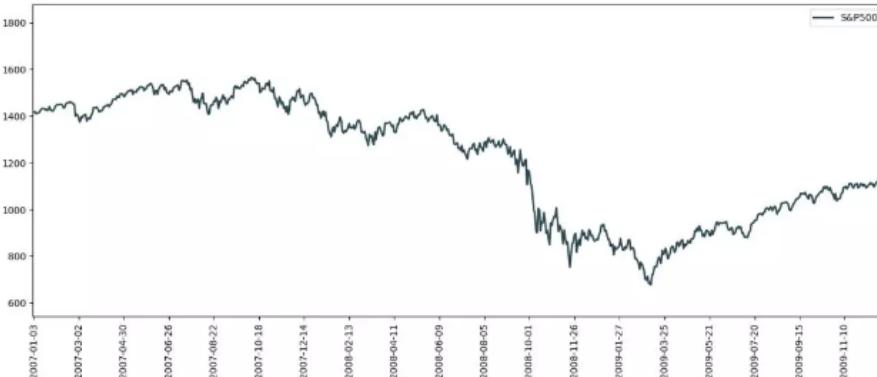
添加图例 (legend) 非常简单，只需要在 `ax.plot()` 里多设定一个参数 `label`，然后用

```
ax.legend()
```

其中 `loc = 0` 表示 matplotlib 自动安排一个最好位置显示图例，而 `frameon = True` 给图例加了外框。

```
1 fig = plt.figure( figsize=(16,6), dpi=100 )
2 ax = fig.add_subplot(1,1,1)
3 x = spx.index
4 y = spx.values
5 ax.plot( y, color=dt_hex, linewidth=2, linestyle='-', label='S&P500' )
6 ax.legend( loc=0, frameon=True )
7
8 ax.set_xlim(-1, len(x)+1)
9 ax.set_ylim( y.min()*0.8, y.max()*1.2 )
10
```

```
11 ax.set_xticks( range(0,len(x),40) )
12 ax.set_xticklabels( [x[i].strftime('%Y-%m-%d') for i in ax.get_xticks()], \
13                      rotation=90 );
```



注意图的右上角多了图例 **S&P500**。

2.8 添加第二幅图



对，就是要画两条序列，把恐慌指数 VIX 也加进去。



不扶墙就服你，但这个需求太简单。

在改进代码之前，先介绍一下 VIX 指数。

知识点

VIX 指数是芝加哥期权交易所 (CBOE) 市场波动率指数的交易代号，常见于衡量 S&P500 指数期权的隐含波动性，通常被称为「恐慌指数」，它是了解市场对未来30天市场波动性预期的一种衡量方法。

由其定义可知，S&P500 指数涨时，VIX 跌，而 S&P500 指数暴跌时，VIX 暴涨。

和之前一样，首先用 `pd.read_csv` 函数从存好的 VIX.csv 读取数据存成 DataFrame。

```
1 data = pd.read_csv('VIX.csv', index_col=0,
2                     parse_dates=True,
3                     dayfirst=True)
4 vix = data[['Adj Close']]
5 .loc['2007-01-01':'2010-01-01']
6 vix.head(3).append(vix.tail(3))
```

Adj Close	
Date	
2007-01-03	12.040000
2007-01-04	11.510000
2007-01-05	12.140000
2009-12-29	20.010000
2009-12-30	19.959999
2009-12-31	21.680000

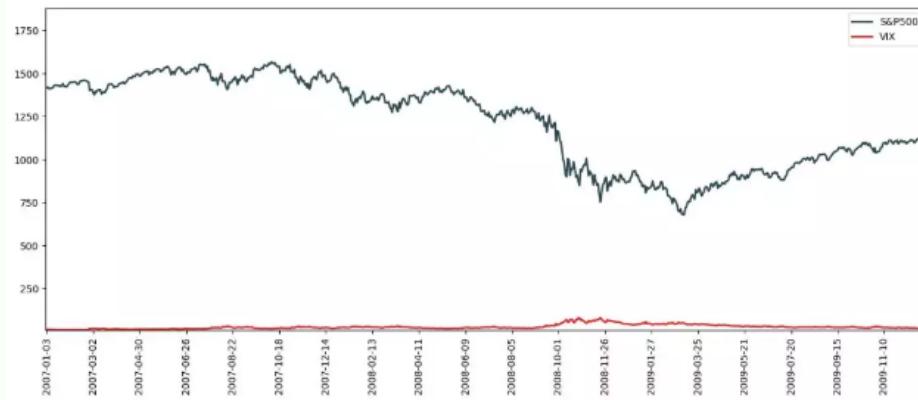
添加第二幅图也很简单，用两次 `plt.plot()` 或者 `ax.plot()` 即可。这里面用的是 `plt` 没用 `ax`，没有特殊原因，在本例中两者可以随意使用，但两者在使用「`.methods`」时有个小细节不知道大家注意到没有，

- `plt.xlim`
- `plt.ylim`
- `plt.xticks`

- `ax.set_xlim`
- `ax.set_ylim`
- `ax.set_xticks`

```
1 plt.figure(figsize=(16,6), dpi=100)
2 x = spx.index
3 y1 = spx.values
4 y2 = vix.values
5 plt.plot(y1, color=dt_hex, linewidth=2, linestyle='-', label='S&P500')
6 plt.plot(y2, color=r_hex, linewidth=2, linestyle='-', label='VIX')
7 plt.legend(loc=0, frameon=True)
8
9 plt.xlim(-1, len(x)+1)
10 plt.ylim(np.vstack([y1,y2]).min()*0.8, np.vstack([y1,y2]).max()*1.2)
```

```
11
12 x_tick = range(0,len(x),40)
13 x_label = [x[i].strftime('%Y-%m-%d') for i in x_tick]
14 plt.xticks( x_tick, x_label, rotation=90 )
15 plt.show()
```



这图怎么成这样？？？VIX 怎么是一条平线？

2.9 两个坐标系 & 两幅子图



图画成这个鬼样子还好意思给我看？



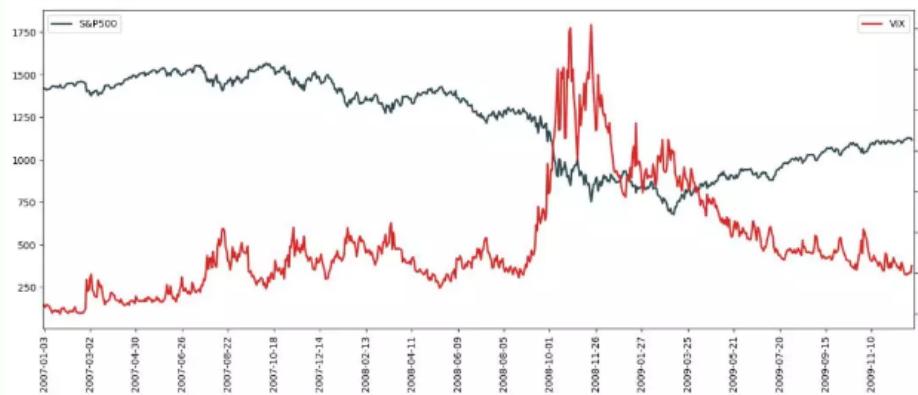
我知道问题出哪儿了，两个序列的量纲不一样，有两种解决方案：1. 用两个坐标系；2. 用两幅子图。

S&P500 的量纲都是千位数，而 VIX 的量纲是两位数，两者放在一起，那可不是 VIX 就像一条水平线一样。两种改进方式：

1. 用两个坐标系 (two axes)
2. 用两幅子图 (two subplots)

两个坐标系

```
1 fig = plt.figure( figsize=(16,6), dpi=100 )
2 ax1 = fig.add_subplot(1,1,1)
3
4 x = spx.index
5 y1 = spx.values
6 y2 = vix.values
7
8 ax1.plot( y1, color=dt_hex, linewidth=2, linestyle='-', label='S&P500' )
9 ax1.set_xlim(-1, len(x)+1)
10 ax1.set_ylim( np.vstack([y1,y2]).min()*0.8, np.vstack([y1,y2]).max()*1.2 )
11
12 x_tick = range(0,len(x),40)
13 x_label = [x[i].strftime('%Y-%m-%d') for i in x_tick]
14 ax1.set_xticks( x_tick )
15 ax1.set_xticklabels( x_label, rotation=90 )
16 ax1.legend( loc='upper left', frameon=True )
17
18 # Add a second axes
19 ax2 = ax1.twinx()
20 ax2.plot( y2, color=r_hex, linewidth=2, linestyle='-', label='VIX' )
21 ax2.legend( loc='upper right', frameon=True );
```



用 ax1 和 ax2 就能在两个坐标系上画图，代码核心部分是第 19 行的

```
ax2 = ax1.twinx()
```

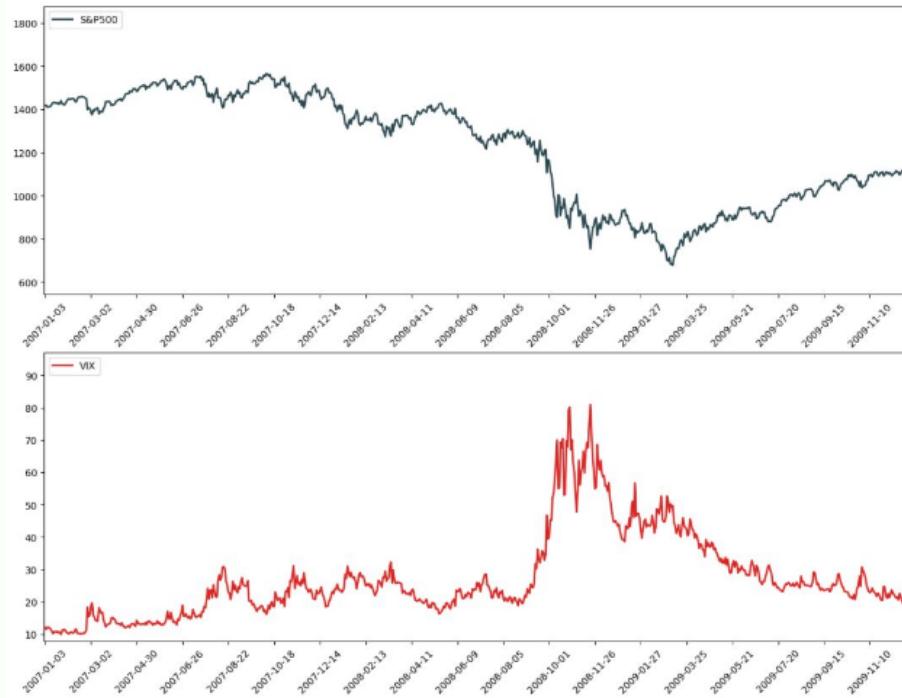
在每个坐标系下画图以及各种设置前面都讲的很清楚了。

两幅子图

```

1 plt.figure( figsize=(16,12), dpi=100 )
2
3 # subplot 1
4 plt.subplot(2,1,1)
5 x = spx.index
6 y1 = spx.values
7
8 plt.plot( y1, color=dt_hex, linewidth=2, linestyle='-', label='S&P500' )
9 plt.xlim(-1, len(x)+1)
10 plt.ylim( y1.min()*0.8, y1.max()*1.2 )
11
12 x_tick = range(0,len(x),40)
13 x_label = [x[i].strftime('%Y-%m-%d') for i in x_tick]
14 plt.xticks( x_tick, x_label, rotation=45 )
15 plt.legend( loc='upper left', frameon=True )
16
17 # subplot 2
18 plt.subplot(2,1,2)
19 y2 = vix.values
20
21 plt.plot( y2, color=r_hex, linewidth=2, linestyle='-', label='VIX' )
22 plt.xlim(-1, len(x)+1)
23 plt.ylim( y2.min()*0.8, y2.max()*1.2 )
24
25 plt.xticks( x_tick, x_label, rotation=45 )
26 plt.legend( loc='upper left', frameon=True )
27
28 plt.show()

```



定义 subplot(2,1,1) 和 subplot(2,1,2) 就能实现再两幅子图上画图。

在每幅子图上画图以及各种设置前面都讲的很清楚了。

这两种方法都可用，但在本例中，S&P500 和 VIX 放在一起（用两个坐标系）更能看出它们之间的关系，比如 2008 年 9 月到 2009 年 3 月的金融危机期间，S&P 500 在狂泻和 VIX 在飙升 😱😱😱。

2.10 设置标注

在金融危机时期，市场发生了 5 件大事，分别是

- 2017-10-11: 牛市顶点
 - 2008-03-12: 贝尔斯登倒闭
 - 2008-09-15: 雷曼兄弟倒闭
 - 2009-01-20: 苏格兰皇家银行股票抛售
 - 2009-04-02: G20 峰会



我想看到这五个事件标注在 S&P 500 的走势图上。



加标注的代码略长，新内容为

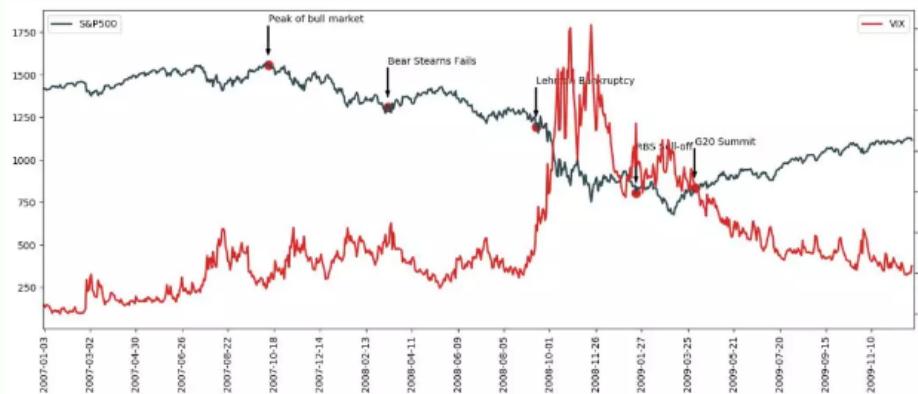
- 第 3-7 行的定义危机事件，以元组的列表存储
 - 第 26-34 行的事件标注，用到 `annotate()` 函数

```
1 fig = plt.figure( figsize=(16,6), dpi=100 )
2
3 crisis_data = [(datetime(2007, 10, 11), 'Peak of bull market'),
4                 (datetime(2008, 3, 12), 'Bear Stearns Fails'),
5                 (datetime(2008, 9, 15), 'Lehman Bankruptcy'),
6                 (datetime(2009, 1, 20), 'RBS Sell-off'),
7                 (datetime(2009, 4, 2), 'G20 Summit')]
```

```

9  ax1 = fig.add_subplot(1,1,1)
10
11 x = spx.index
12 y1 = spx.values
13 y2 = vix.values
14
15 ax1.plot( y1, color=dt_hex, linewidth=2, linestyle='-', label='S&P500' )
16 ax1.set_xlim(-1, len(x)+1)
17 ax1.set_ylimit( np.vstack([y1,y2]).min()*0.8, np.vstack([y1,y2]).max()*1.2 )
18
19 x_tick = range(0,len(x),40)
20 x_label = [x[i].strftime('%Y-%m-%d') for i in x_tick]
21 ax1.set_xticks( x_tick )
22 ax1.set_xticklabels( x_label, rotation=90 )
23
24 ax1.legend( loc='upper left', frameon=True )
25
26 for date, label in crisis_data:
27     date = date.strftime('%Y-%m-%d')
28     xi = x.get_loc(date)
29     yi = spx.asof(date)
30     ax1.scatter( xi, yi, 80, color=r_hex )
31     ax1.annotate( label, xy=(xi, yi + 60),
32                   xytext=(xi, yi + 300),
33                   arrowprops=dict(facecolor='black', headwidth=4, width=1, headlength=6),
34                   horizontalalignment='left', verticalalignment='top' )
35
36 # Add a second axes
37 ax2 = ax1.twinx()
38 ax2.plot( y2, color=r_hex, linewidth=2, linestyle='-', label='VIX' )
39 ax2.legend( loc='upper right', frameon=True );

```



从第 26 行开始，用 for 循环读取 crisis_data 里面每个日期 date 和事件 label。

第 28 和 29 行是获取每一个 date 在整个日期数组中的索引 xi，以及对应的 spx 值 yi。

第 30 行用 scatter() 函数画出一个圆点，标注事件在 spx 折现上的位置。

第 31 和 34 行是重头戏，在 annotate() 函数里设置了事件，箭头坐标，事件打印的坐标，箭头性质，以及对齐属性。

事件的确标注在图上了，但是效果像一坨💩。

2.11 设置透明度



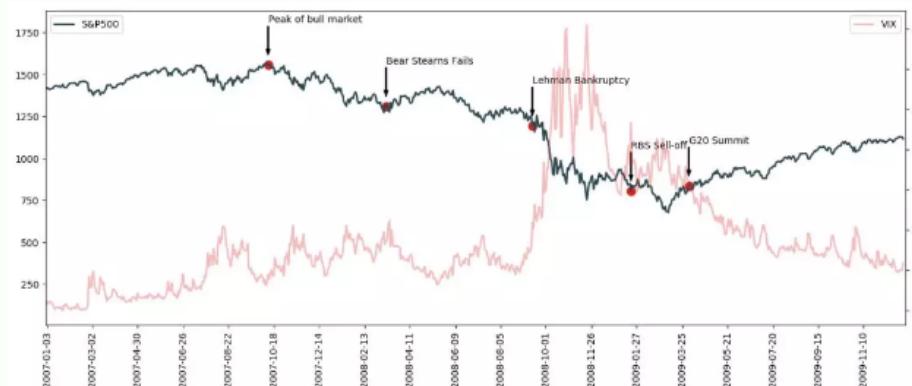
S&P 500 和 VIX 两条线画在一起太混乱了，而且事件标注也看不清楚。再信息不能少的情况下务必改进！



S&P 500 是主线，VIX 是副线，把副线的透明度调高点不就行了。

为了把 VIX 折线弄得透明些，只用设置 `ax2.plot()` 里的 `alpha` 参数为 0.3，具体设什么值看你想要多透明，`alpha` 在 0 和 1 之间，0 是完全透明，1 是完全不透明。

```
1 ax2.plot( y2, color=r_hex,
2             linewidth=2,
3             linestyle='--',
4             label='VIX',
5             alpha=0.3 )
```



美如画！雷曼兄弟倒闭（事件 3）后 S&P 暴跌最厉害，而同期的 VIX 也飙到天际。在 G20

峰会(事件5)过后，大国领导者一起解决金融危机问题，从那个点开始，S&P500上涨VIX下跌。

经济总体平稳！风险总体可控！



2.12 完善细节



基本满意，你自由发挥看看还有什么可以改进的地方。

别介，我最怕这种开放式要求。 . .



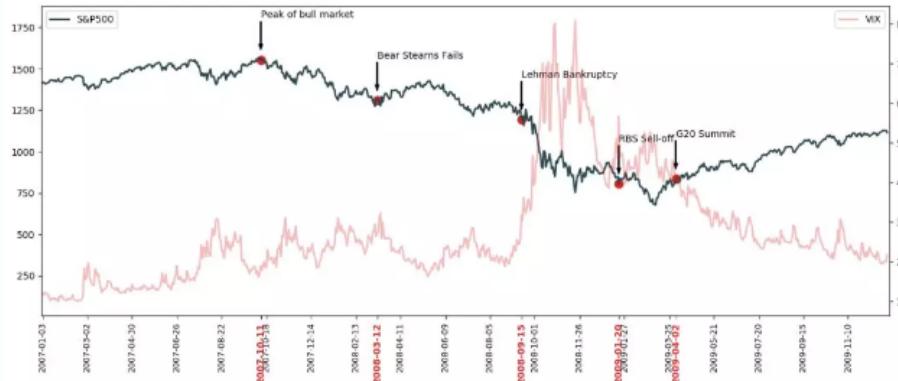
既然老板关注这五个事件，而它们发生的日期可能没有落在横轴标签上，那老板不是在图上还是找不到他们发生的具体时间么？把它们加上去怎么样？

```
1 fig = plt.figure( figsize=(16,6), dpi=100 )
2
3 crisis_data = [(datetime(2007, 10, 11), 'Peak of bull market'),
4                  (datetime(2008, 3, 12), 'Bear Stearns Fails'),
5                  (datetime(2008, 9, 15), 'Lehman Bankruptcy'),
6                  (datetime(2009, 1, 20), 'RBS Sell-off'),
7                  (datetime(2009, 4, 2), 'G20 Summit')]
8
9 ax1 = fig.add_subplot(1,1,1)
10
11 x = spx.index
12 y1 = spx.values
13 y2 = vix.values
14
15 ax1.plot( y1, color=dt_hex, linewidth=2, linestyle='-', label='S&P500' )
16 ax1.set_xlim(-1, len(x)+1)
17 ax1.set_ylim( np.vstack([y1,y2]).min()*0.8, np.vstack([y1,y2]).max()*1.2 )
18 ax1.legend( loc='upper left', frameon=True )
19
```

```

20 init_tick = list( range(0,len(x),40) )
21 impt_tick = []
22 impt_date = []
23
24 for date, label in crisis_data:
25     date = date.strftime('%Y-%m-%d')
26     impt_date.append(date)
27
28     xi = x.get_loc(date)
29     impt_tick.append(xi)
30     yi = spx.asof(date)
31
32     ax1.scatter( xi, yi, 80, color=r_hex )
33     ax1.annotate( label, xy=(xi, yi + 60),
34                   xytext=(xi, yi + 300),
35                   arrowprops=dict(facecolor='black', headwidth=4, width=1, headlength=6),
36                   horizontalalignment='left', verticalalignment='top' )
37
38 x_tick = init_tick+impt_tick
39 x_label = [x[i].strftime('%Y-%m-%d') for i in x_tick]
40 ax1.set_xticks( x_tick )
41 ax1.set_xticklabels( x_label, rotation=90 )
42
43 for i, label in enumerate(ax1.get_xticklabels()):
44     if i >= len(init_tick):
45         label.set_color(r_hex)
46         label.set_fontweight('bold')
47     else:
48         label.set_fontsize(9)
49
50 # Add a second axes
51 ax2 = ax1.twinx()
52 ax2.plot( y2, color=r_hex, linewidth=2, linestyle='--', label='VIX', alpha=0.3 )
53 ax2.legend( loc='upper right', frameon=True );

```



新添加的代码在第 20-22 行和第 43-48 行。主要就是把日期分成两类：

- 常规日期标签 `init_tick`
- 五个事件日期标签 `impt_tick`



现在才像话嘛，晚上下面可以加个蛋。

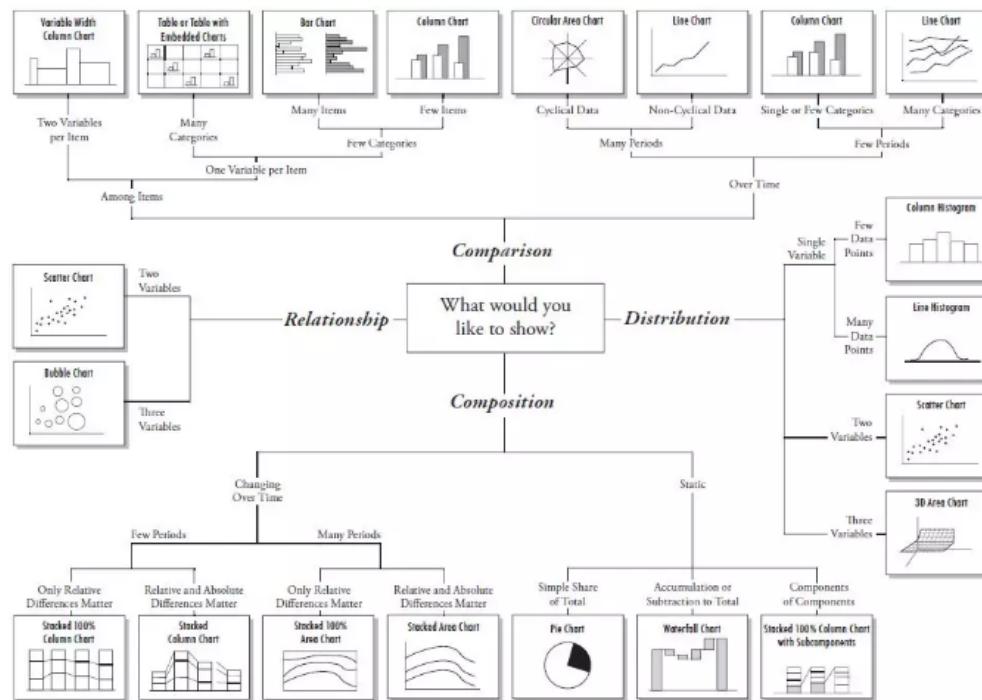


3 画有效图 <

3.1 概览

在做图表设计时候经常面临着怎么选用合适的图表，图表展示的关系分为四大类 (点击下图放大):

1. 分布 (distribution)
2. 联系 (relationship)
3. 比较 (comparison)
4. 构成 (composition)



在选用图表前首先要想清楚：你要表达什么样的数据关系。上面的图表分类太过繁多，接下来我们只讨论在量化金融中用的最多的几种类型，即

- 用**直方图**来展示股票价格和收益的**分布**
- 用**散点图**来展示两支股票之间的**联系**
- 用**折线图**来**比较**汇率在不同窗口的移动平均线
- 用**饼状图**来展示股票组合的**构成**成分

首先用 `YahooFinancials` API 来下载若干资产的一年历史数据 (安装该 API 用 `pip install yahoofinancials`)：

- 起始日：2018-04-29
- 终止日：2019-04-29
- 五只股票：英伟达、亚马逊、阿里巴巴、脸书、苹果
- 三个汇率：欧元美元、美元日元、美元人民币

下面代码就是从 API 获取数据，股票用的是股票代号 (stock code)，而货币用的该 API 要求的格式，比如「欧元美元」用 `EURUSD=X`，而不是市场常见的 `EURUSD`，而「美元日元」用 `JPY=X` 而不是 `USDJPY`。

```
1 from yahoofinancials import YahooFinancials
2
3 start_date = '2018-04-29'
4 end_date = '2019-04-29'
5
6 stock_code = ['NVDA', 'AMZN', 'BABA', 'FB', 'AAPL']
7 currency_code = ['EURUSD=X', 'JPY=X', 'CNY=X']
8
9 stock = YahooFinancials( stock_code )
10 currency = YahooFinancials( currency_code )
11
12 stock_daily = stock.get_historical_price_data( start_date, end_date, 'daily' )
13 currency_daily = currency.get_historical_price_data( start_date, end_date, 'daily' )
```

该 API 返回结果 `stock_daily` 和 `currency_daily` 是「字典」格式，样子非常丑陋，感受一下。

```
1 stock_daily
```

```
{'NVDA': {'eventsData': {'dividends': {'2018-05-23': {'amount': 0.15,
    'date': 1527082200,
    'formatted_date': '2018-05-23'},
    '2018-08-29': {'amount': 0.15,
    'date': 1535549400,
    'formatted_date': '2018-08-29'},
    '2018-11-29': {'amount': 0.16,
    'date': 1543501800,
    'formatted_date': '2018-11-29'},
    '2019-02-28': {'amount': 0.16,
    'date': 1551364200,
    'formatted_date': '2019-02-28'}}},
    'firstTradeDate': {'formatted_date': '1999-01-22', 'date': 916995600},
    'currency': 'USD',
    'instrumentType': 'EQUITY',
    'timeZone': {'gmtOffset': -14400},
    'prices': [{{'date': 1525095000,
        'high': 229.0,
        'low': 224.1199951171875,
        'open': 224.1199951171875,
        'close': 229.0,
        'volume': 0,
        'adjclose': 228.99951171875,
        'formatted_date': '2018-05-23'}}]}},
```

1 currency_daily

```
{'EURUSD=X': {'eventsData': {},
    'firstTradeDate': {'formatted_date': '2003-12-01', 'date': 1070236800},
    'currency': 'USD',
    'instrumentType': 'CURRENCY',
    'timeZone': {'gmtOffset': 3600},
    'prices': [{{'date': 1525042800,
        'high': 1.2138574123382568,
        'low': 1.2066364288330078,
        'open': 1.2128562927246094,
        'close': 1.2122827768325806,
        'volume': 0,
        'adjclose': 1.2122827768325806,
        'formatted_date': '2018-04-29'}}],
    {'date': 1525129200,
        'high': 1.2084592580795288,
        'low': 1.1983511447906494,
        'open': 1.208313226699829,
        'close': 1.2081234455108643,
        'volume': 0,
```

学过 Pandas 之后，我们应该可以把上面的「原始数据」转换成 DataFrame，代码如下：

```
1 def data_converter( price_data, code, asset ):
2     # convert raw data to dataframe
3     if asset == 'FX':
4         code = str(code[3:]) if code[:3]=='USD' else code) + '=X'
```

```

6     columns = ['open', 'close', 'low', 'high' ]
7     price_dict = price_data[code]['prices']
8     index = [ p['formatted_date'] for p in price_dict ]
9     price = [ [p[c] for c in columns] for p in price_dict ]
10
11    data = pd.DataFrame( price,
12                           index=pd.Index(index, name='date'),
13                           columns=pd.Index(columns, name='OHLC') )
14

```

第 3 行完全是为了 YahooFinancial 里面的输入格式准备的。如果 Asset 是股票类，直接用其股票代码；如果 Asset 是汇率类，一般参数写成 EURUSD 或 USDJPY

- 如果是 EURUSD，转换成 EURUSD=X
- 如果是 USDJPY，转换成 JPY=X

第 6 行定义好开盘价、收盘价、最低价和最高价的标签。

第 7 行获取出一个「字典」格式的数据。

第 8, 9 行用列表解析式 (list comprehension) 将日期和价格获取出来。

第 11 到 13 行定义一个 DataFrame

- 值为第 9 行得到的 price 列表
- 行标签为第 8 行得到的 index 列表
- 列标签为第 6 行定义好的 columns 列表

处理过后的数据格式美如画，不信你看 (用 EURUSD 和 NVDA 举例)

```

1 EURUSD = data_converter( currency_daily, 'EURUSD', 'FX' )
2 EURUSD.head(3).append(EURUSD.tail(3))

```

OHLC	open	close	low	high
date				
2018-04-29	1.212856	1.212283	1.206636	1.213857
2018-04-30	1.208313	1.208123	1.198351	1.208459
2018-05-01	1.199213	1.199156	1.195414	1.203109
2019-04-24	1.115698	1.115349	1.112174	1.116400
2019-04-25	1.113710	1.113685	1.112298	1.117281
2019-04-28	1.114902	1.115026	1.114554	1.116994

```
1 NVDA = data_converter( stock_daily, 'NVDA', 'EQ' )
2 NVDA.head(3).append(NVDA.tail(3))
```

date	OHLC	open	close	low	high
2018-04-30	226.990005	224.899994	224.119995	229.000000	
2018-05-01	224.570007	227.139999	222.199997	227.250000	
2018-05-02	227.000000	226.309998	225.250000	228.800003	
2019-04-24	191.089996	191.169998	188.639999	192.809998	
2019-04-25	189.550003	186.910004	183.699997	190.449997	
2019-04-26	180.710007	178.089996	173.300003	180.889999	

3.2 直方图

直方图 (histogram chart), 又称质量分布图, 是一种统计报告图, 由一系列高度不等的纵向条纹或线段表示数据分布的情况。一般用横轴表示数据类型, 纵轴表示分布情况。在 Matplotlib 里的语法是

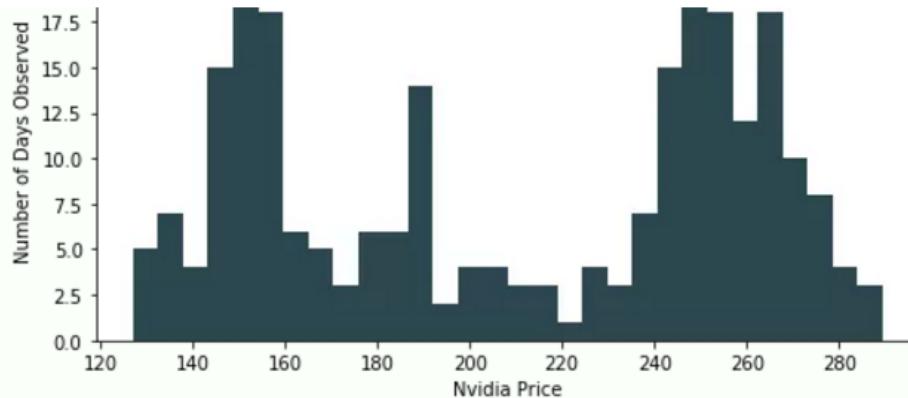
- plt.hist()
- ax.hist()

我们先看看英伟达 (NVDA) 的价格分布。

```
1 p_NVDA = NVDA[ 'close' ]
2
3 fig = plt.figure( figsize=(8,4) )
4 plt.hist( p_NVDA, bins=30, color=dt_hex )
5 plt.xlabel('Nvidia Price')
6 plt.ylabel('Number of Days Observed')
7 plt.title('Frequency Distribution of Nvidia Prices, Apr-2018 to Apr-2019')
8 plt.show()
```

Frequency Distribution of Nvidia Prices, Apr-2018 to Apr-2019





在本例中函数 `hist()` 里的参数有

- `p_NVDA`: Series, 也可以是 list 或者 ndarray
- `bins`: 分成多少堆
- `colors`: 用之前定义的深青色

从上图可看出, NVDA 的价格分布在有 220 划分的两个范围 (regime)。在 2018 年 11 月 16 日 (星期五), 英伟达第三季度的报表低于预期, 那么股价暴跌 19%, 在之后的星期一, 又跌 12%, 两个交易日股价一下子从原来的 220 左右跌到 150。

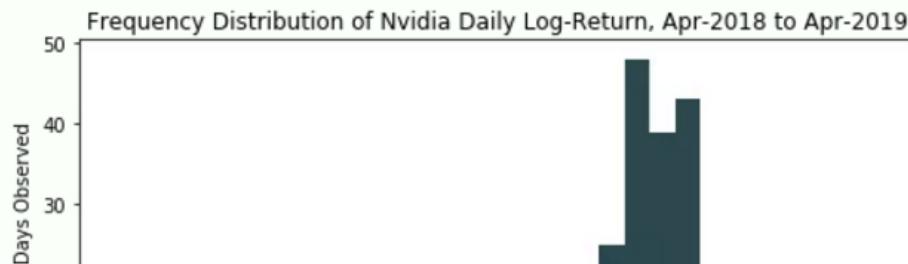
在研究股票价格序列中, 由于收益率有些好的统计性质, 我们对其更感兴趣, 接下来再看看英伟达 (NVDA) 的对数收益 (log-return) 的分布。

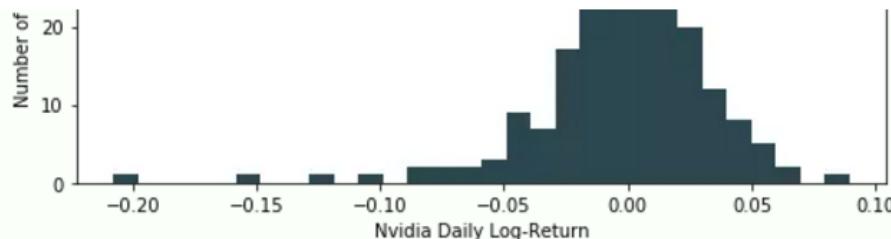
```

1 date = p_NVDA.index
2 price = p_NVDA.values
3 r_NVDA = pd.Series( np.log(price[1:]/price[:-1]), index=date[1:] )

1 fig = plt.figure( figsize=(8,4) )
2
3 plt.hist( r_NVDA, bins=30, color=dt_hex )
4 plt.xlabel('Nvidia Daily Log-Return')
5 plt.ylabel('Number of Days Observed')
6 plt.title('Frequency Distribution of Nvidia Daily Log-Return, Apr-2018 to Apr-2019')
7
8 plt.show()

```





首先对数收益的计算公式为

$$r(t) = \ln(P(t)/P(t-1))$$

得到 r_{NVDA} 。计算一天的收益率需要两天的价格，因此用 p_{NVDA} 计算 r_{NVDA} 时，会丢失最新一天的数据，因此我们用 $date[1:]$ 作为 r_{NVDA} 的行标签 (index)。

不考虑在 -0.20 和 -0.15 那两个极端值，对数收益率的分布像一个正态分布 (人人都喜欢正态分布)。

3.3 散点图

散点图 (scatter chart) 用两组数据构成多个坐标点，考察坐标点的分布，判断两变量之间是否存在某种联系的分布模式。在 Matplotlib 里的语法是

- `plt.scatter()`
- `ax.scatter()`

我们看看中美两大电商亚马逊 (AMZN) 和阿里巴巴 (BABA) 之间的价格和对数收益率的联系。

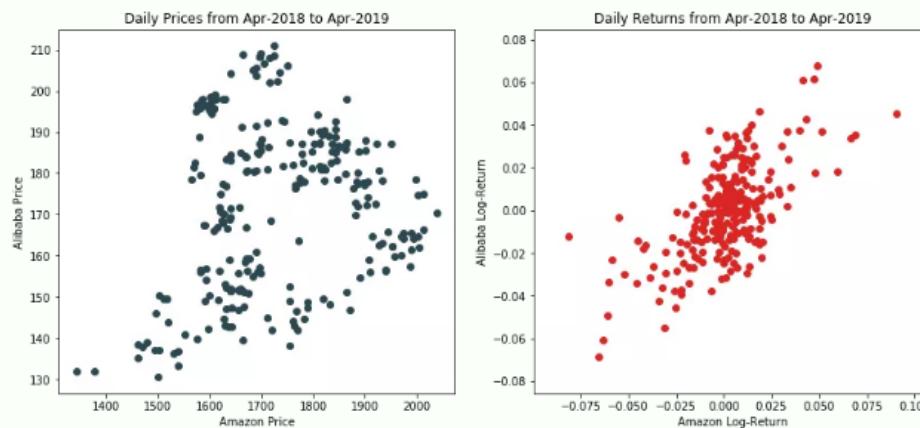
首先计算价格和对数收益率。

```

1 AMZN = data_converter( stock_daily, 'AMZN', 'EQ' )
2 BABA = data_converter( stock_daily, 'BABA', 'EQ' )
3
4 p_AMZN = AMZN['close']
5 p_BABA = BABA['close']
6
7 date = p_AMZN.index
8 price = p_AMZN.values
9 r_AMZN = pd.Series( np.log(price[1:]/price[:-1]), index=date[1:] )
10
11 date = p_BABA.index
12 price = p_BABA.values
13 r_BABA = pd.Series( np.log(price[1:]/price[:-1]), index=date[1:] )
```

用两个子图分别展示「价格」和「收益率」的散点图。

```
1 fig, axes = plt.subplots( nrows=1, ncols=2, figsize=(14,6) )
2
3 axes[0].scatter( p_AMZN, p_BABA, color=dt_hex )
4 axes[0].set_xlabel('Amazon Price')
5 axes[0].set_ylabel('Alibaba Price')
6 axes[0].set_title('Daily Prices from Apr-2018 to Apr-2019')
7
8 axes[1].scatter( r_AMZN, r_BABA, color=r_hex )
9 axes[1].set_xlabel('Amazon Log-Return')
10 axes[1].set_ylabel('Alibaba Log-Return')
11 axes[1].set_title('Daily Returns from Apr-2018 to Apr-2019')
12
13 plt.show()
```



在本例中函数 `scatter()` 里的参数有

- `p_AMZN (r_AMZN)`: Series, 也可以是 list 或者 ndarray
- `p_BABA (r_BABA)`: Series, 也可以是 list 或者 ndarray
- `colors`: 用之前定义的深青色和红色

从右图来看，亚马逊和阿里巴巴在这端时期的表现正相关，如果做线性回归是一条斜率为正的线。

折线图 (line chart) 显示随时间而变化的连续数据，因此非常适用于显示在相等时间间隔下数据的趋势。在 Matplotlib 里的语法是

- `plt.plot()`
- `ax.plot()`

我们来看看如何画 EURUSD 的 20 天和 60 天移动平均 (moving average, MA) 线。

首先获取 EURUSD 的收盘价。

```
1 curr = 'EURUSD'
2 EURUSD = data_converter( currency_daily, curr, 'FX' )
3 rate = EURUSD['close']
```

用 Pandas 里面的 `rolling()` 函数来计算 MA，在画出收盘价，MA20 和 MA60 三条折线。

```
1 fig = plt.figure(figsize=(16,6))
2 ax = fig.add_subplot(1,1,1)
3
4 ax.set_title( curr + ' - Moving Average' )
5 ax.set_xticks( range(0,len(rate.index),10) )
6 ax.set_xticklabels( [rate.index[i] for i in ax.get_xticks()], rotation=90 );
7
8 ax.plot( rate, color=dt_hex, linewidth=2, label='Close' )
9
10 MA_20 = rate.rolling(20).mean()
11 MA_60 = rate.rolling(60).mean()
12
13 ax.plot( MA_20, color=r_hex, linewidth=2, label='MA20' )
14 ax.plot( MA_60, color=g_hex, linewidth=2, label='MA60' )
15
16 ax.legend(loc=0);
```



在本例中函数 `plot()` 里的参数有

- `rate, MA_20, MA_60`: Series, 也可以是 list 或者 ndarray
- `colors`: 用之前定义的深青色, 红色, 绿色
- `linewidth`: 像素 2
- `label`: 用于显示图例

上面代码最关键的就是第 10 和 11 行, 用 `rolling(n)` 函数对 `rate` 求 n 天移动均值。从图中注意到绿色的 MA60 最短, 红色的 MA20 其次。原因很简单, 假如一年有 252 个交易日, 那么第 1 个 MA60 值需要第 1 到 60 个汇率, 第 2 个 MA60 值需要第 2 到 61 个汇率, 第 193 个 MA60 值需要第 193 到 252 个汇率。最终只有 193 个 MA60。同理可得到只有 223 个 MA20。

双均线策略如下: MA60 和 MA20 必有交点, 若 20 天平均线「上穿越」60 天均线, 则为买入点; 反之为卖出点。该策略基于不同天数均线的交叉点抓住股票的强势和弱势时刻进行交易。

3.5 饼状图

饼状图 (pie chart) 是一个划分为几个扇形的圆形统计图表, 用于描述量、频率或百分比之间的相对关系。在饼状图中, 每个扇区面积大小为其所表示的数量的比例。在 Matplotlib 里的语法是

- `plt.pie()`
- `ax.pie()`

我们来看看如何画出一个股票投资组合在 2019 年 4 月 26 日的饼状图, 假设组合里面有 100 股英伟达, 20 股亚马逊, 50 股阿里巴巴, 30 股脸书和 40 股苹果 (一个科技股爱好者的组合 😎)。

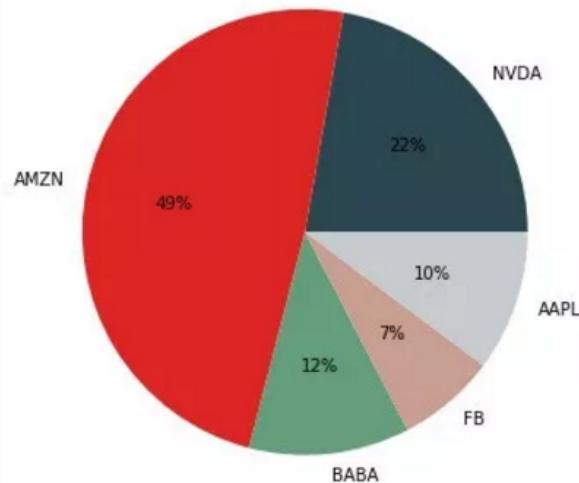
首先计算组合里五支股票在 2019 年 4 月 26 日的市值 (market value, MV)。

```
1 stock_list = ['NVDA', 'AMZN', 'BABA', 'FB', 'AAPL']
2 date = '2019-04-26'
3
4 MV = [ data_converter(stock_daily, code, 'EQ')['close'][date] for code in stock_list ]
5 MV = np.array(MV) * np.array([100, 20, 50, 30, 40])
```

第 4 行用了列表解析式来获取 `stock_list` 每支股票的价格, 第 5 行将价格乘上数量得到市值。

设定好五种颜色和百分数格式 %.0f%% (小数点后面保留 0 位), 画出饼状图。

```
1 fig = plt.figure(figsize=(16,6))
2 ax = fig.add_subplot(1,1,1)
3
4 ax.pie(MV, labels=stock_list, colors=[dt_hex,r_hex,g_hex,tn_hex,g25_hex], \
5         autopct='%.0f%%')
6
7 plt.show()
```



在本例中函数 `pie()` 里的参数有

- MV: 股票组合市值, ndarray
- labels: 标识, list
- colors: 用之前定义的一组颜色, list
- autopct: 显示百分数的格式, str

虽然画出了饼状图, 但看起来有些别扭, 且听下节分解如何改进。

3.6 同理心

为用户习惯考虑

把饼当成钟, 大多数人习惯顺时针的看里面的内容, 因此把**面积最大的那块的一条边**(见下图)放在**12点的位置**最能突显其重要性, 之后按面积从大到小顺时针排列。

在画饼状图前，我们需要额外做两件事：

1. 按升序排列 5 只股票的市值
2. 设定 `pie()` 的相关参数达到上述「最大块放 12 点位置」的效果

首先按市值大小按升序排序。

```
1 idx = MV.argsort()[::-1]
2 MV = MV[idx]
3 stock_list = [ stock_list[i] for i in idx ]
4 print( MV )
5 print( stock_list )
```

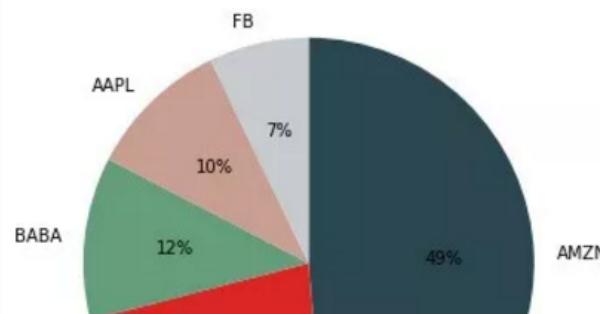
```
[39012.60009766 17808.99963379
9354.49981689 8172.00012207
5744.70016479]

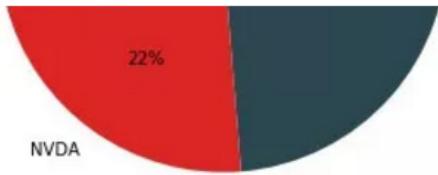
['AMZN', 'NVDA', 'BABA', 'AAPL', 'FB']
```

设定参数

- `startangle = 90` 是说第一片扇形 (`AMZN` 深青色那块) 的左边在 90 度位置
- `counterclock = False` 是说顺时针拜访每块扇形

```
1 fig = plt.figure(figsize=(16,6))
2 ax = fig.add_subplot(1,1,1)
3
4 ax.pie( MV, labels=stock_list, colors=[dt_hex,r_hex,g_hex,tn_hex,g25_hex], \
5         autopct='%.0f%%', startangle=90, counterclock=False )
6
7 plt.show()
```





和上节最后的图相比，现在这饼状图看上去是不是顺眼多了。你承不承认你第一眼就注意到 12 点那个位置的扇形？

为图表信息考虑

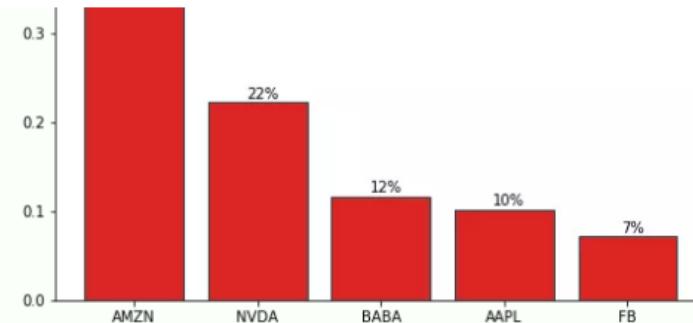
当饼状图里面扇形多过 5 个时，面积相近的扇形大小并不容易一眼辨别出来，不信看上图的 BABA 和 APPL，没看到数字很难看出那个面积大。但绝大多数人是感官动物，图形和数字肯定先选择看图形，这个时候用柱状图 (bar chart) 来代替饼状图，每个市值成分大小一目了然 (好图就是能让用户能最快的抓住核心信息)。

用 `ax.bar()` 函数来画柱状图，为了和饼状图的信息一致，几个关键操作为

- 第 4 行计算出市值的百分数 `pct_MV`
- 第 8, 9 行设置横轴刻度 (0,1,2,3,4) 和标签 (`stock_list`)
- 第 12, 13 行在特定位置上 ($x+0.04$, $y+0.05/100$) 将 `pct_MV` 以 `{0:.0%}` 的格式 (不保留小数点) 写出来，这些位置试几次看图的效果就可以确定下来。

```
1 fig = plt.figure(figsize=(8,6))
2 ax = fig.add_subplot(1,1,1)
3
4 pct_MV = MV / np.sum(MV)
5 index = np.arange(len(pct_MV))
6
7 ax.bar(index, pct_MV, facecolor=r_hex, edgecolor=dt_hex )
8 ax.set_xticks(index)
9 ax.set_xticklabels(stock_list)
10 ax.set_ylim(0, np.max(pct_MV)*1.1)
11
12 for x,y in zip(index,pct_MV):
13     ax.text(x+0.04, y+0.05/100, '{0:.0%}'.format(y), ha='center', va='bottom')
14
15 plt.show()
```





在本例中函数 `bar()` 里的参数有

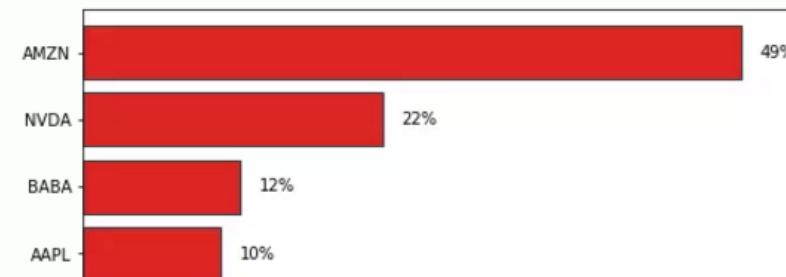
- `index`: 横轴刻度, ndarray
- `pct_MV`: 股票组合市值比例, ndarray
- `facecolor`: 柱状颜色, 红色
- `edgecolor`: 柱边颜色, 深青色

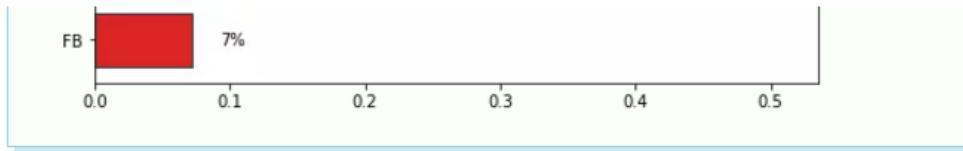
如果柱状很多时, 或者标签名字很长时, 用横向柱状图 (horizontal bar chart), 函数为 `ax.bart()`。代码和上面非常类似, 就是把横轴和纵轴的调换了一下。

```

1 fig = plt.figure(figsize=(8,4))
2 ax = fig.add_subplot(1,1,1)
3
4 pct_MV = MV[::-1] / np.sum(MV)
5 index = np.arange(len(pct_MV))
6
7 ax.bart(index, pct_MV, facecolor=r_hex, edgecolor=dt_hex )
8 ax.set_yticks(index)
9 ax.set_yticklabels(stock_list[::-1])
10 ax.set_xlim(0, np.max(pct_MV)*1.1)
11
12 for x,y in zip(pct_MV,index):
13     ax.text(x+0.04, y, '{0:.0%}'.format(x), ha='right', va='center')
14
15 plt.show()

```





为色盲用户考虑

世界上有 1/12 的男人和 1/200 的女人都有不同程度的色盲症状。因此当你将结果展示给重要客户时，最好考虑到这一点，我相信对方会非常欣赏你这种「同理心」。

幸运的是，Matplotlib 里面有专门为色盲考虑的色彩风格，首先用下列语句看查看所有的色彩风格。

```
1 print(plt.style.available)

['bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot',
'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palett
e', 'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted',
'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster',
'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'se
aborn', 'Solarize_Light2', 'tableau-colorblind10', '_classic_test'] >
```

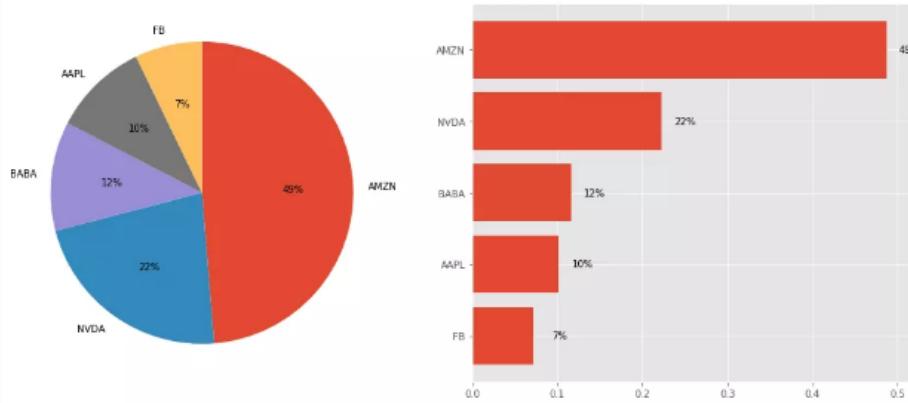
不难发现 `seaborn-colorblind` 和 `tableau-colorblind10` 就是我们所需要的。下面我们看看不同色彩风格下的「饼状图」和「柱状图」。

首先看从 R 中借用过来的大名鼎鼎的 `ggplot` 的效果。

```
1 plt.style.use('ggplot')

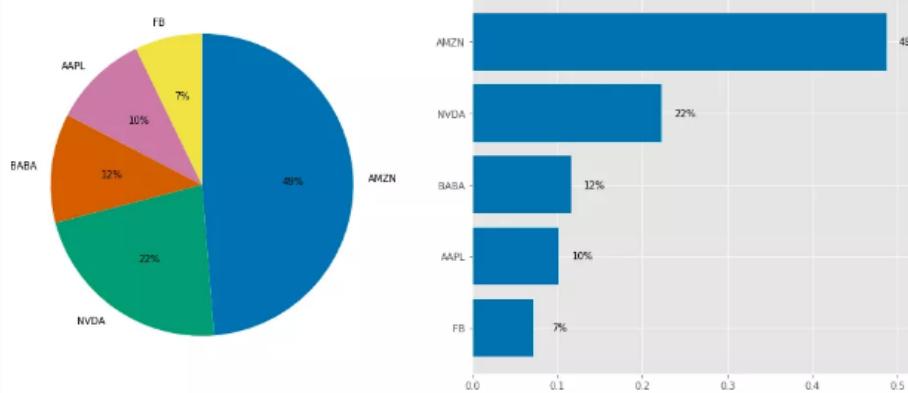
1 fig, axes = plt.subplots( nrows=1, ncols=2, figsize=(14,6) )
2
3 axes[0].pie( MV, labels=stock_list, autopct='%.0f%%', \
4             startangle=90, counterclock=False )
5
6 pct_MV = MV[::-1] / np.sum(MV)
7 index = np.arange(len(pct_MV))
8
9 axes[1].barh( index, pct_MV )
10 axes[1].set_yticks( index )
11 axes[1].set_yticklabels( stock_list[::-1] )
12 axes[1].set_xlim( 0, np.max(pct_MV)*1.1 )
13
```

```
14 for x,y in zip(pct_MV,index):
15     axes[1].text( x+0.04, y, '{0:.0%}'.format(x), ha='right', va='center' )
16
17 plt.tight_layout()
18 plt.show()
```



再看 `seaborn-colorblind` 的效果。

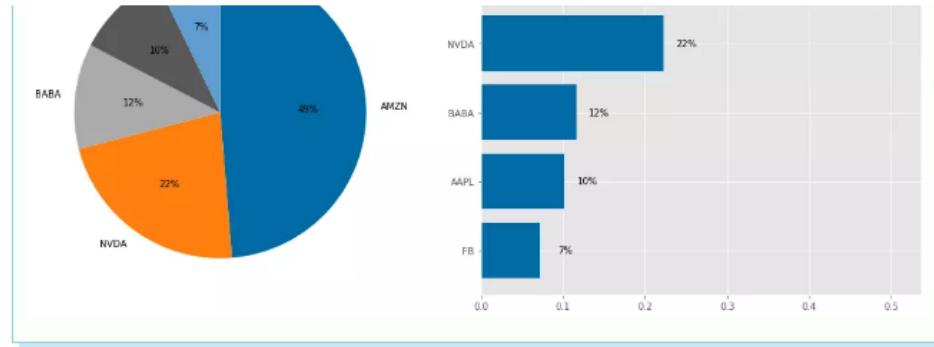
```
1 plt.style.use('seaborn-colorblind')
```



最后看 `tableau-colorblind10` 的效果。

```
1 plt.style.use('tableau-colorblind10')
```





4 总结

本贴的思路非常清晰：

- 第一章了解 Matplotlib 的绘图逻辑，以及里面包含的画图元素以及它们之间的层级。
- 第二章深度学 Matplotlib，只研究折线图，通过研究它的属性，一步步改进图的尺寸、像素、线条颜色宽度风格、坐标轴边界、刻度标签、图例、多图、多坐标系、标注、透明度等等，画出了一幅美图。
- 第三章广度学 Matplotlib，通过数据的**分布**、**联系**、**比较**和**构成**研究了直方图、散点图、折线图和饼状图，最后还为用户着想（习惯、色盲等等）画出更能有效表达信息的图。

我们现在处于一个大数据的时代，制图能力现在和写作能力一样重要。任何人现在都可以用各种制图工具或者编程语言来画图，但是很少人懂得画出好图。

好图不是指的绚烂的颜色 (fancy colors) 和复杂的层级 (complex layers)，当一张图里的信息能够以最清晰和有效的方式传递给使用者，那么这张图就是好图。

到此，我突然决定不写交互式的 Bokeh 了，因为使用 Matplotlib 和 PyEcharts 已经足够。另外用于统计可视化的 Seaborn 要不要写（因为学会了 Matplotlib 学 Seaborn 很容易），看看投票结果吧（我已投**跳吧跳吧**）。

投票已过期

每篇写的好累，要写 seaborn 还是直接跳到 pyechart? (单选)

- 要写，不要跳 :(
- 跳吧跳吧 :)

下篇讨论用于统计制图的 Seaborn (看投票结果) 还是直奔炫酷制图的 PyEcharts?

Stay Tuned!

• END •
