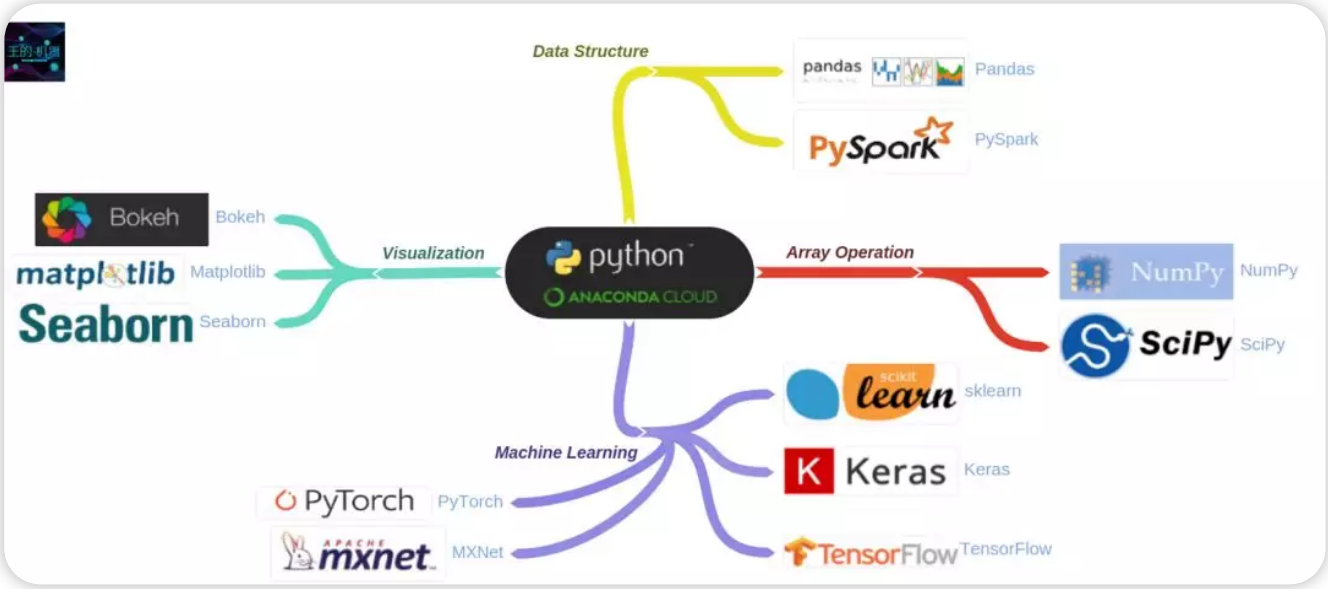
盘一盘 Python 系列 7 - PyEcharts (v1.0)

原创 王圣元 王的机器 2019-12-08

来自专辑

Python



本文含 5835 字，33 图表截屏
建议阅读 30 分钟

0 引言

有读者说〖**PyEcharts**〗一贴里的图美如画，但是版本是 pyecharts v0.5，用现在 v1.0 来运行会出错，建议我再写篇 pyecharts v1.0 的。我最不喜欢让读者失望，这不我就来了。

用 v1.0 来运行 v0.5 的代码是肯定会报错的，v0.5 和 v1 间不兼容，v1 是一个全新的版本。首先来回顾 v0.5 的方法总结。

PyEcharts v0.5 方法总结

对 pyecharts 中的所有原件，都是先创建 (可以带些必要属性，比如标题和尺寸)，再用 add 方法添加额外属性。其通用化流程为

```
object = Object( 必要属性 )
object.add(  额外属性 )
```

在画图之前，你应该对那些原件可以干嘛有个大概印象，比如 Kline 是画 K 线图，Heatmap 是画热力图，WordCloud 是图词云图等等。对于那些装饰原件所需的必要属性和额外属性，上官网一查便知，跟着例子看理解更快。

画多个坐标系用 Grid 对象；叠加多个原件用 Overlap 对象；随着时间轴展示不同时点的数据关系用 Timeline 对象，等等。

首先用 pip 安装 pyecharts

```
pip install pyecharts
```

引入 pyecharts 并打印出它的版本

```
import pyecharts as pye
print('pyecharts: %s' % pye.__version__)
```

```
pyecharts: 1.3.1
```

本文首先对比 pyecharts v0.5 和 v1.0 的区别，之后举三个从简单到复杂的例子来学习 v1.0 的用法。

1 v0.5 Vs v1.0

引入基本元件

在 v0.5 中，引入 Line (线)、Kline (K 线)、Bar (柱状图)、Pie (饼状图)、Grid (多坐标系)、Overlap (叠加对象)、Timeline (时间轴轮播图)、TreeMap (树状图) 和 WordCloud (词云

图) 的代码如下：

```
from pyecharts
import Line, Kline, Pie,
       Grid, Overlap, Timeline,
       TreeMap, WordCloud
```

在 v1.0 中，引入它们（除了 `Overlap`）的代码如下：

```
from pyecharts.charts
import Line, Kline, Pie,
       Grid, Timeline,
       TreeMap, WordCloud
```

在 v1.0 中，我们从 `pyecharts.charts` 中引入元件，而不是从 `pyecharts` 引入。此外，v1.0 已经没有用于组合元件的 `Overlap` 了，它有一种更简单的组合方法。对于两个元件，K 线 kline 和线 line，v0.5 和 v1.0 的代码如下：

v0.5：需要先创建一个 Overlap 对象，再把 kline 和 line 一个个添加进去。

```
overlap = Overlap()
overlap.add(kline)
overlap.add(line)
```

v1.0：每个元件都有 overlap() 函数，可以另外元件，比如先创建 kline 再添加 line.

```
kline.overlap(line)
```

一切皆配置

在 pyecharts v1.0 中，一切皆配置（options）。配置项有两种：**全局配置项**和**系列配置项**。

全局配置项有以下 16 小项：

系列配置项有以下 14 小项：

配置项越细就能画出更多细节。在后面几节我们会重点说明，尤其是全局配置项，它可通过 `set_global_options` 方法来设置。

引入 pyecharts 里的 options 代码如下：

```
from pyecharts import options as opts
```

1 K 线图

数据

首先用 `YahooFinancials` API 来下载外汇的三年半历史数据，安装该 API 用一行代码：

```
pip install yahoofinancials
```

数据的描述如下

- 起始日：2016-01-01
- 终止日：2019-05-13
- 四个外汇：欧元美元、美元日元、美元人民币，英镑美元

其中货币用的不是市场常见格式，比如「欧元美元」用 EURUSD=X，而不是 EURUSD，而「美元日元」用 JPY=X 而不是 USDJPY

下面代码就是从 API 获取数据：

```
from yahoofinancials import YahooFinancials
```

```
start_date = '2016-01-01'
end_date = '2019-05-13'

currencies = ['EURUSD=X', 'JPY=X', 'CNY=X', 'GBPUSD=X']
cryptocurrencies = ['BTC-USD', 'ETH-USD', 'XRP-USD']
```

```
FX_obj = YahooFinancials( currencies )
CRX_obj = YahooFinancials( cryptocurrencies )

FX_daily = FX_obj.get_historical_price_data( start_date, end_date, 'daily' )
CFX_daily = CRX_obj.get_historical_price_data( start_date, end_date, 'daily' )
```

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

```
1 FX_daily
```


```
{'EURUSD=X': {'eventsData': {},
  'firstTradeDate': {'formatted_date': '2003-12-01', 'date': 1070236800},
  'currency': 'USD',
  'instrumentType': 'CURRENCY',
  'timeZone': {'gmtOffset': 3600},
  'prices': [{'date': 1451606400,
    'high': 1.0866966247558594,
    'low': 1.0859060287475586,
    'open': 1.0859060287475586,
    'close': 1.0859060287475586,
    'volume': 0,
    'adjclose': 1.0859060287475586,
    'formatted_date': '2016-01-01'},
   {'date': 1451865600,
    'high': 1.094599723815918,
    'low': 1.0805997848510742,
    'open': 1.0855052471160889,
    'close': 1.0853991508483887,
    'volume': 0,
```


数据样子虽丑，但还满齐全，画 K 线需要的开盘价 (open)、最高价 (high)、最低价 (low)、收盘价 (close) 都有。将上面的「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'
5
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     return data
```

第 3 行完全是为了 YahooFinancial 里面的输入格式准备的。如果 Asset 是加密货币，直接用其股票代码；如果 Asset 是汇率，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，看看 USDCNY。

```
curr = 'USDCNY'
data = data_converter( FX_daily, curr, 'FX' )
data.head(3).append(data.tail(3))
```

OHLC	open	close	low	high
date				
2016-01-01	6.4837	6.4837	6.4837	6.4837
2016-01-04	6.4837	6.4837	6.4837	6.5256
2016-01-05	6.5254	6.5254	6.5095	6.5254
2019-05-08	6.7820	6.7820	6.7819	6.8288
2019-05-09	6.8175	6.8265	6.7953	6.8276
2019-05-12	6.8141	6.8231	6.8141	6.8808

PyEcharts v0.5

PyEcharts 0.5 里画 K 线用到 `Kline`对象，除此之外我们添加最高价和最低价两条线 `Line` 对象，再用 `Overlap` 对象来「叠加」它们。

```
1  date = data.index
2  price = data.values
3
4  kline = Kline( curr+' Chart', title_pos='center' )
5  kline.add( 'K-Line', date, price, tooltip_tragger='axis', is_datazoom_show=True,
6             legend_pos='right', legend_orient='vertical', legend_text_size=10 )
7
8  line2 = Line()
9  features = ['high', 'low']
10
11 for feature in features:
12     line2.add( feature, date, data[feature], tooltip_tragger='axis' )
13
14 overlap = Overlap( width=1000, height=400 )
15 overlap.add(kline)
16 overlap.add(line2)
17 #overlap.render(path=u'USDCNY Chart.html')
```

第 1-2 行获取日期和汇率。

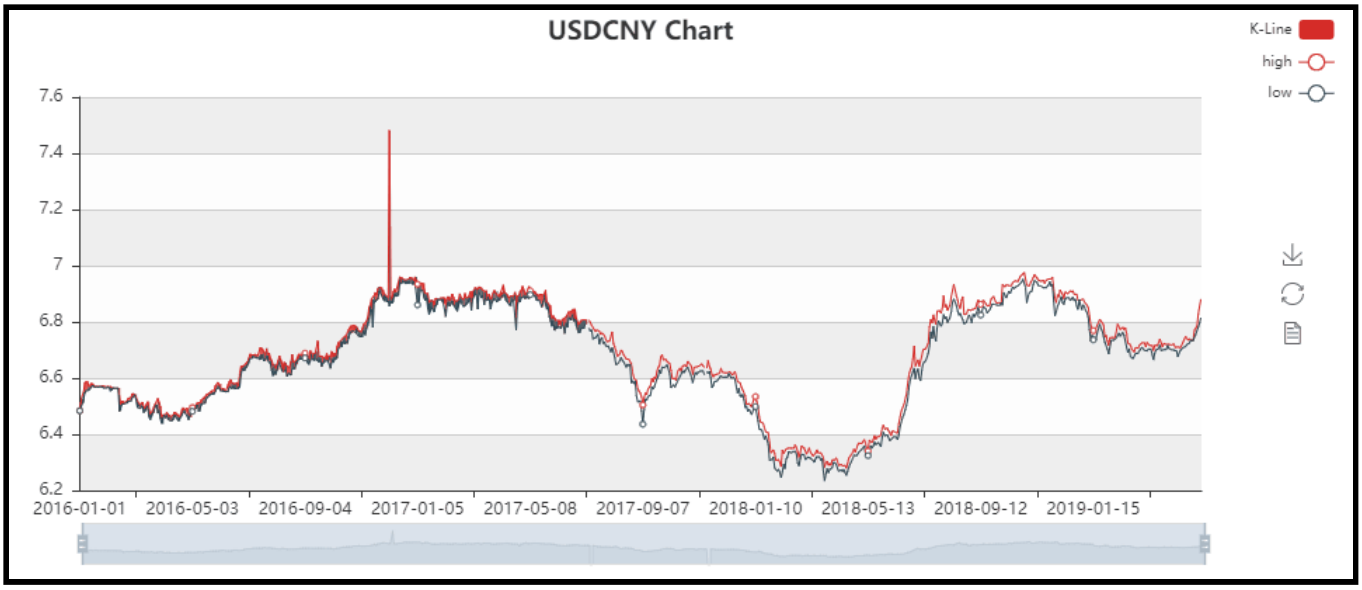
第 4 行创建 K 线对象 `Kline`，设置好标题 "xxx Chart" 和位置 center。第 5-6 行在 `Kline` 上添加属性

- **图例**：'K-Line',
- **x 坐标轴数据**：日期
- **y 坐标轴数据**：一定要按 [开盘值, 收盘值, 最低值, 最高值] 的顺序，之前处理数据特意按这个顺序设定 DataFrame 的列标签的
- **x 坐标轴可拉伸**：**True**
- **图例位置**：右边
- **图例排序**：竖直
- **图例文字大小**：10

第 8 行创建折线对象 `Line`。第 9-13 行在 `Line` 上添加两条折线，一条是最高价，一条是最低价。

第 14 行创建叠加对象 `Overlap`。第 15-16 行在 `Overlap` 上分别添加之前的 `Kline` 和 `Line`，这样就把所有对象整合在一起了。

第 17 行如果被运行，该动态图被生成到 USDCNY Chart.html 网页文件里；如果没被运行，该动态图将显示在 Jupyter Notebook 中。



PyEcharts 1.0

PyEcharts 1.0 里画 K 线用到 `Kline` 对象，除此之外我们添加最高价和最低价两条线 `Line` 对象，然后直接把两条线添加到 K 线上去。

```
1 date = data.index
2 price = data.values
3
4 date_list = pd.to_datetime(date).strftime('%Y/%m/%d').tolist()
```

需要把日期转成 'Y/m/d' 格式，在转成列表形式。在 v1.0 中，所有数据都需要转成**列表**形式。

```
1  kline = (
2      Kline()
3      .add_xaxis(date_list)
4      .add_yaxis("K Line", price.tolist())
5      .set_global_opts(
6          yaxis_opts=opts.AxisOpts(
7              is_scale=True,
8              splitarea_opts=opts.SplitAreaOpts(
9                  is_show=True,
10                 areastyle_opts=opts.AreaStyleOpts(opacity=1)
11             ),
12         ),
13         xaxis_opts=opts.AxisOpts(is_scale=True),
14         title_opts=opts.TitleOpts(title=curr+' Chart'),
15         datazoom_opts=[opts.DataZoomOpts()],
16     )
17 )
```

第 2 行用 `Kline()` 构造函数生成 **K 线对象**，然后

- 用 add_xaxis() 来修饰 x 轴（第 3 行）传入**日期列表**
- 用 add_yaxis() 来修饰 y 轴（第 4 行）传入**价格列表**
- 用 set_global_opts() 来设置全局配置（第 5-16 行），主要配置包括：
    - y 轴可缩放，且颜色交错（第 6-12 行）
    - x 轴可缩放（第 13 行）
    - 设置标题（第 14 行）
    - 数据局域缩放（第 15 行）

```
19 line = (
20     Line()
21     .add_xaxis(date_list)
22     .add_yaxis(
23             series_name='high',
24             y_axis=data['high'].tolist(),
25             is_smooth=True,
26             is_hover_animation=False,
27             linestyle_opts=opts.LineStyleOpts(width=1, opacity=0.5),
28             label_opts=opts.LabelOpts(is_show=False),
29     )
30     .add_yaxis(
31             series_name='low',
32             y_axis=data['low'].tolist(),
33             is_smooth=True,
34             is_hover_animation=False,
35             linestyle_opts=opts.LineStyleOpts(width=1, opacity=0.5),
36             label_opts=opts.LabelOpts(is_show=False),
37     )
38     .set_global_opts(datazoom_opts=[opts.DataZoomOpts()],)
39 )
```

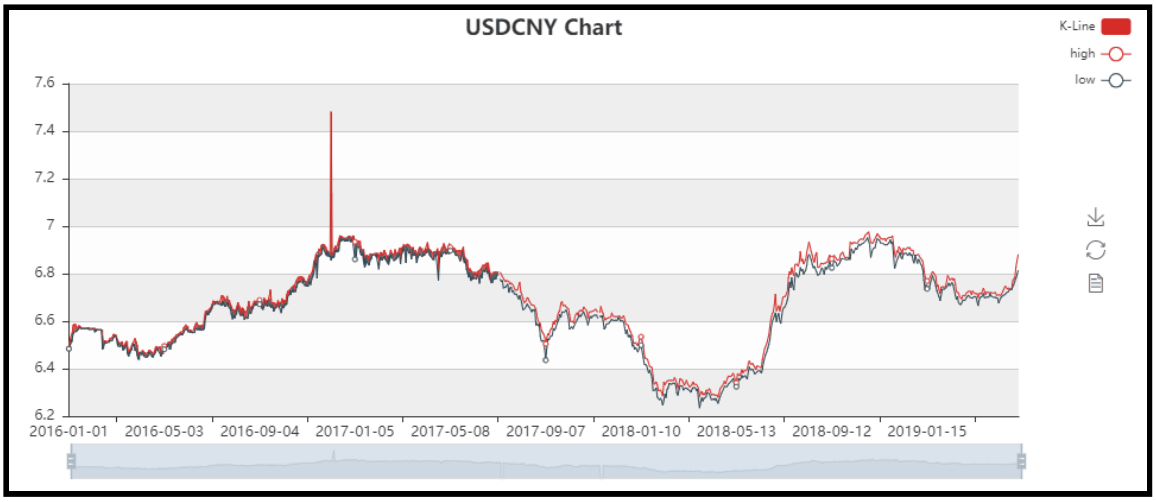
第 20 行用 `Line()` 构造函数生成**线对象**，然后

- 用 add_xaxis() 来修饰 x 轴（第 21 行）传入**日期列表**
- 用 add_yaxis() 来修饰 y 轴（第 22-29 行）传入**最高价列表**
- 用 add_yaxis() 来修饰 y 轴（第 30-37 行）传入**最低价列表**
- 用 set_global_opts() 来设置全局配置，主要设置数据局域缩放（第 38 行）

在修饰 y 轴时，我们还设置了线的宽度和透明度、已经不打印出 y 轴对应的图示。

```
41 overlap_kline_line = kline.overlap(line)
42 overlap_kline_line.render_notebook()
```

最后将 **K 线**和**两条线**组合在一起，在 notebook 里展现 (render_notebook)。



在 v1.0 中，通用代码长得以下这个样子

```
obj  = (
    Object(...)
    .add_xaxis(...)
    .add_yaxis(...)
    .set_global_options(...)
)
```

其中 `Object` 可以是任何常见元件，比如 `Kline`，`Line` 和 `Bar` 等等。三点省略号 `...` 就代表各种配置了，具体是什么那就要读文档了。

2 股价 K 线图 + 折线图

数据

本小节使用 5 个股票数据，描述如下：

- **5 只股票**：AAPL, JD, BABA, FB, GS
- **1 年时期**：从 2018-02-26 到 2019-02-26

再加上同时期的标准普尔 500 指数 (SPX)，和恐慌指数 (VIX)。数据如下：

```
stock_data = pd.read_csv( '1Y Stock Data.csv',
                          parse_dates=[0],
                          dayfirst=True )
stock_data.head().append(stock_data.tail())
```

	Date	Symbol	Open	High	Low	Close	Adj Close	Volume
0	2018-02-26	AAPL	176.350006	179.389999	176.210007	178.970001	176.285675	38162200
1	2018-02-27	AAPL	179.100006	180.479996	178.160004	178.389999	175.714386	38928100
2	2018-02-28	AAPL	179.259995	180.619995	178.050003	178.119995	175.448410	37782100
3	2018-03-01	AAPL	178.539993	179.779999	172.660004	175.000000	172.375214	48802000
4	2018-03-02	AAPL	172.800003	176.300003	172.449997	176.210007	173.567078	38454000
1255	2019-02-20	GS	198.729996	199.300003	197.509995	198.600006	198.600006	2266000
1256	2019-02-21	GS	198.970001	199.449997	195.050003	196.360001	196.360001	2785900
1257	2019-02-22	GS	196.600006	197.750000	195.199997	196.000000	196.000000	2626600
1258	2019-02-25	GS	198.000000	201.500000	197.710007	198.649994	198.649994	3032200
1259	2019-02-26	GS	198.470001	200.559998	196.550003	198.899994	198.899994	2498000

```
data = pd.read_csv( 'S&P500.csv',
                    index_col=0,
                    parse_dates=True,
                    dayfirst=True )
spx = data[['Adj Close']].loc['2018-02-26':'2019-02-26']
spx.head(3).append(spx.tail(3))
```

	Adj Close
Date	
2018-02-26	2779.600098
2018-02-27	2744.280029
2018-02-28	2713.830078
2019-02-22	2792.669922
2019-02-25	2796.110107
2019-02-26	2793.899902

```
data = pd.read_csv( 'VIX.csv',
                    index_col=0,
                    parse_dates=True,
```

```
                        dayfirst=True )
vix = data[['Adj Close']].loc['2018-02-26':'2019-02-26']
vix.head(3).append(vix.tail(3))
```

Date	Adj Close
2018-02-26	15.80
2018-02-27	18.59
2018-02-28	19.85
2019-02-22	13.51
2019-02-25	14.85
2019-02-26	15.17

PyEcharts v0.5

我们想把苹果股票的 K 线图，和 SPX 和 VIX 折线图放在一起看。如果再用 `Overlap` 来叠加它们会显得图很乱，这时可以借用 `pyecharts` 里的 `Grid` 对象，它是将上面三个图放在三个坐标系中。

代码如下：

```
code = 'AAPL'
stock = stock_data[ stock_data['Symbol']==code ]

date = stock['Date'].dt.strftime('%d-%b-%Y')
price = stock[['Open','Close','Low','High']].values

kline = Kline( code, title_pos='left' )
kline.add( '', date, price, tooltip_tragger='axis', is_datazoom_show=True )

line1 = Line( 'SPX', title_top='55%' )
line1.add( '', date, spx.values, yaxis_min=2200, yaxis_max=3000,
               mark_point=['min'], is_datazoom_show=True,
               datazoom_xaxis_index=[2,1,0] )

line2 = Line( 'VIX', title_top='75%' )
line2.add( '', date, vix.values, yaxis_min=0, yaxis_max=40,
               mark_point=['max'], is_datazoom_show=True,
               datazoom_xaxis_index=[2,1,0] )

grid = Grid( width=1000, height=600 )
grid.add( line2, grid_top="75%" )
grid.add( line1, grid_top="55%", grid_bottom="30%" )
grid.add( kline, grid_top="5%", grid_bottom="50%" )
#grid.render(path='AAPL&VIX.html')
```

第 1-5 行用 code 获取股票数据，并获取日期和价格。为了画 K 线，价格数组的列必须按 pyecharts 里 API 要求的顺序 - [开盘价, 收盘价, 最低价, 最高价]。

第 6 -7 行创建 Kline 对象 (标题放左边)，并添加 x 轴数据、y 轴数据和"允许横轴拉伸"。

第 10 -12 行创建 Line 对象 (标题为 SPX，位置离顶 55%) 并起名为 line1，再添加若干属性，比如 y 轴范围、标识最小值、允许横轴拉伸。最关键的是 datazoom_xaxis_index=[2,1,0]，就说伸缩功能控制三个轴，AAPL 一个，SPX 一个，VIX 一个。这样拉伸 x 轴三幅子图可以同时动，非常酷!

第 15 -18 行创建 Line 对象 (标题为 VIX，位置离顶 75%) 并起名为 line2，再添加若干属性。

第 20 行创建 Grid 对象，宽 1000，高 600 (这些数值是不断尝试看效果设置的)。

第 20-23 行将三幅图加在 Grid 中，关键点是如何设置里面的 grid_top 和 grid_bottom 里的百分数而使得图看起来好看，这个没有标准的，不停地尝试到你最终满意为止。本例中 AAPL 占了 5% 到 50% 的位置，SPX 占了 55% 到 70% 的位置，VIX 占了 75% 到 90% 的位置 (还有 10% 位置留给了拉缩轴)。

第 24 行如果被运行，该动态图被生成到 APPL&VIX.html 网页文件里；如果没被运行，该动态图将显示在 Jupyter Notebook 中。

从图上可以看到在 2018 年底 SPX 和 VIX 同时到达最低点和最高点，对应的苹果 K 线看，在那一点前后苹果股价有一个大跌和大涨。

PyEcharts v1.0

直接上代码。

```
code = 'AAPL'
stock = stock_data[ stock_data['Symbol']==code ]

date = stock['Date'].dt.strftime('%d-%b-%Y').tolist()
price = stock[['Open','Close','Low','High']].values.tolist()
```

日期和 OLHC 价格所有数据都需要转成**列表**形式。

```
kline = (
    Kline()
    .add_xaxis(date)
    .add_yaxis('', price)
    .set_global_opts(
        yaxis_opts=opts.AxisOpts(
            is_scale=True,
            splitarea_opts=opts.SplitAreaOpts(
                is_show=True,
                areastyle_opts=opts.AreaStyleOpts(opacity=1)
            ),
        ),
        xaxis_opts=opts.AxisOpts(is_scale=True),
        title_opts=opts.TitleOpts(title=code, pos_top='25%'),
        datazoom_opts=[opts.DataZoomOpts(xaxis_index=[0, 1, 2],
                                         range_start=0,
                                         range_end=len(date),)],
    )
)
```

用 `Kline()` 构造函数生成 **K 线对象**上节已经讲过，需要注意的是第 21 行中的 xaxis_index=[0,1,2]，这个设置太关键了。本图含三个子图

1. 苹果股票的 K 线图（index 0）
2. 标普 500 的折线图 (index 1)
3. 恐慌指数的折线图 (index 2)

上面设置是 index为 1 和 2 的两幅图的数据局部伸缩跟着 index 0 那幅图，这样就实现了用一根 x 轴的 slider 可以任意缩放三幅图的数据。

```
27 line1 = (
28     Line()
29     .add_xaxis(date)
30     .add_yaxis(
31             series_name='',
32             y_axis=spx.values.tolist(),
33             markpoint_opts=opts.MarkPointOpts(
34                 data=[opts.MarkPointItem(type_='min')]),
35             is_smooth=True,
36             is_hover_animation=False,
37             linestyle_opts=opts.LineStyleOpts(width=3, opacity=0.5),
38             label_opts=opts.LabelOpts(is_show=False),
39     )
40     .set_global_opts(
41         xaxis_opts=opts.AxisOpts(is_scale=True),
42         yaxis_opts=opts.AxisOpts(is_scale=True),
43         title_opts=opts.TitleOpts(title='SPX', pos_top='55%'),
44     )
45 )
```

```
47 line2 = (
48     Line()
49     .add_xaxis(date)
50     .add_yaxis(
51             series_name='',
52             y_axis=vix.values.tolist(),
53             markpoint_opts=opts.MarkPointOpts(
54                 data=[opts.MarkPointItem(type_='max')]),
55             is_smooth=True,
56             is_hover_animation=False,
57             linestyle_opts=opts.LineStyleOpts(width=3, opacity=0.5),
58             label_opts=opts.LabelOpts(is_show=False),
59     )
60     .set_global_opts(
61         xaxis_opts=opts.AxisOpts(is_scale=True),
62         yaxis_opts=opts.AxisOpts(is_scale=True),
63         title_opts=opts.TitleOpts(title='VIX', pos_top='75%'),
64     )
65 )
```

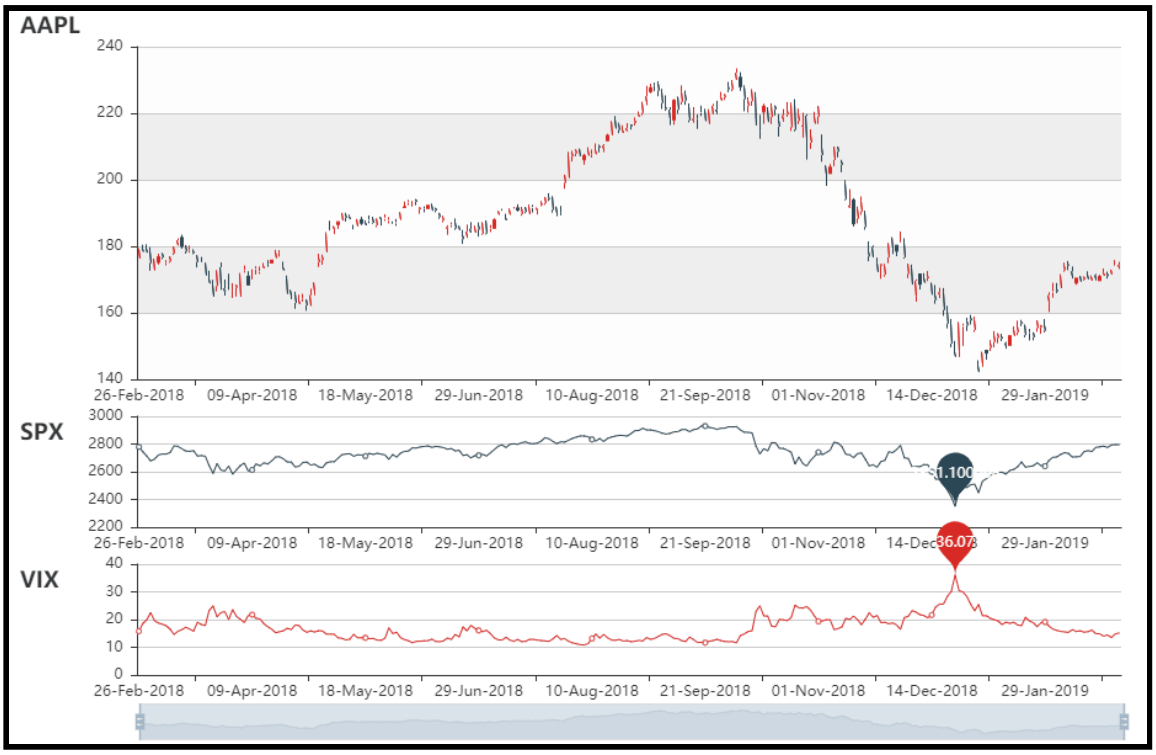
用 `Line()` 构造函数生成**线对象**上节已经讲过，需要注意的是第 34 和 54 行，用 MarkPointOpts 选项标识出 SPX 的最小值和 VIX 的最大值。

```
grid_chart = Grid( init_opts=opts.InitOpts(width="1000px", height="600px") )

grid_chart.add(
    kline,
    grid_opts=opts.GridOpts(pos_left="10%", pos_right="8%", pos_top="5%", height="40%")
)
grid_chart.add(
    line1,
    grid_opts=opts.GridOpts(pos_left="10%", pos_right="8%", pos_top="50%", height="15%")
)
grid_chart.add(
    line2,
    grid_opts=opts.GridOpts(pos_left="10%", pos_right="8%", pos_top="70%", height="15%")
)

grid_chart.render_notebook()
```

第 57 行用 `Grid()` 构造函数来生成网格对象 grid_chart，用来组合上面的三幅图。

接下来一个个加上 AAPL K 线（第 69-72 行）、 SPX 折线（第 73-76 行）和 VIX 折线（第 77-80 行），注意里面 GridOpts 选项里的位置参数。

最后（第 82 行）在 notebook 里展现 grid_chart。



3 股价 K 线图 + 交易量柱状图

数据

本小节使用标准普尔 500 指数 (SPX) 在 2018-02-26 到 2019-02-26 的数据。

```
data = pd.read_csv( 'S&P500.csv', index_col=0, parse_dates=True, dayfirst=True )
data = data.loc['2018-02-26':'2019-02-26']
data.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
2018-02-26	2757.370117	2780.639893	2753.780029	2779.600098	2779.600098	3424650000
2018-02-27	2780.449951	2789.149902	2744.219971	2744.280029	2744.280029	3745080000
2018-02-28	2753.780029	2761.520020	2713.540039	2713.830078	2713.830078	4230660000
2018-03-01	2715.219971	2730.889893	2659.649902	2677.669922	2677.669922	4503970000
2018-03-02	2658.889893	2696.250000	2647.320068	2691.250000	2691.250000	3882450000

PyEcharts v1.0

首先整理一下数据，比如将它们转换成列表形式，等等。

```
date = pd.to_datetime(data.index).strftime('%Y/%m/%d').tolist()
price = data[['Open','Close','Low','High']].values.tolist()
close = data['Close']
volume = data['Volume'].values.tolist()
```

先用 `Kline()` 构建 K 线，这里面的内容最丰富。

```
kline = (
    Kline()
    .add_xaxis(xaxis_data=date)
    .add_yaxis(
        series_name="S&P index",
        y_axis=price,
        itemstyle_opts=opts.ItemStyleOpts(color="#ec0000", color0="#00da3c"),
    )
    .set_global_opts(
        title_opts=opts.TitleOpts(
            title="Kline + Line + Bar",
            subtitle="MA5, MA20",
        ),
        xaxis_opts=opts.AxisOpts(type_="category"),
        yaxis_opts=opts.AxisOpts(
            is_scale=True,
            splitarea_opts=opts.SplitAreaOpts(
                is_show=True, areastyle_opts=opts.AreaStyleOpts(opacity=1)
            ),
        ),
        legend_opts=opts.LegendOpts(is_show=False),
        datazoom_opts=[
            opts.DataZoomOpts(
                is_show=False,
                type_="inside",
                xaxis_index=[0, 1],
                range_start=0,
                range_end=100,
            ),
            opts.DataZoomOpts(
                is_show=True,
                xaxis_index=[0, 1],
                type_="slider",
                pos_top="90%",
                range_start=0,
                range_end=100,
            ),
        ],
```

```
39          tooltip_opts=opts.TooltipOpts(
40              trigger="axis",
41              axis_pointer_type="cross",
42              background_color="rgba(245, 245, 245, 0.8)",
43              border_width=1,
44              border_color="#ccc",
45              textstyle_opts=opts.TextStyleOpts(color="#000"),
46          ),
47          visualmap_opts=opts.VisualMapOpts(
48              is_show=False,
49              dimension=2,
50              series_index=3,
51              is_piecewise=True,
52              pieces=[
53                  {"value": -1, "color": "#ec0000"},
54                  {"value": 1, "color": "#00da3c"},
55              ],
56          ),
57          axispointer_opts=opts.AxisPointerOpts(
58              is_show=True,
59              link=[{"xAxisIndex": "all"}],
60              label=opts.LabelOpts(background_color="#777"),
61          ),
62          brush_opts=opts.BrushOpts(
63              x_axis_index="all",
64              brush_link="all",
65              out_of_brush={"colorAlpha": 0.1},
66              brush_type="lineX",
67          ),
68      )
69  )
```

重点：

第 7 行 - 添加用 Hex 字符串表示的红和绿两种颜色，对应着 K线涨和跌的颜色。

第 22-37 行 - 添加两个「数据区域缩放」功能，一个看的到（用鼠标拉缩图最下面的 slider），一个看不到（用鼠标直接在图中拉缩），并且设置 xaxis_index =[0,1]，表示用 K 线图（index 0）来控制柱状图（index 1）。

第 39-46 行 - 将两幅图的提示框合并在一起（第 41 行这个设置太牛逼）。

第 57-67 行 - **坐标轴指示器**配置和**区域选择组件**配置使得数据和轴可以一起联动。

再用 `Line()` 构建两条移动平均线，没什么可说的，用 pandas 里面的 rolling() 函数计算了 MA5 和 MA20。

```python
line = (
    Line()
    .add_xaxis(xaxis_data=date)
    .add_yaxis(
        series_name="MA5",
        y_axis=close.rolling(5).mean().tolist(),
        is_smooth=True,
        is_hover_animation=False,
        linestyle_opts=opts.LineStyleOpts(width=1, opacity=0.5),
        label_opts=opts.LabelOpts(is_show=False),
    )
    .add_yaxis(
        series_name="MA20",
        y_axis=close.rolling(20).mean().tolist(),
        is_smooth=True,
        is_hover_animation=False,
        linestyle_opts=opts.LineStyleOpts(width=1, opacity=0.5),
        label_opts=opts.LabelOpts(is_show=False),
    )
    .set_global_opts(xaxis_opts=opts.AxisOpts(type_="category"))
    )
```

再用 `Bar()` 构建交易量柱状图，注意第 112-115 行代码，这些设置为了不显示柱状图的 x 轴上的信息。

```python
bar = (
    Bar()
    .add_xaxis(xaxis_data=date)
    .add_yaxis(
        series_name="Volume",
        yaxis_data=[
            [i, volume[i], 1 if price[i][0] > price[i][1] else -1]
            for i in range(len(data))
        ],
        xaxis_index=1,
        yaxis_index=1,
        label_opts=opts.LabelOpts(is_show=False),
    )
    .set_global_opts(
        xaxis_opts=opts.AxisOpts(
            type_="category",
            is_scale=True,
            grid_index=1,
            boundary_gap=False,
            axisline_opts=opts.AxisLineOpts(is_on_zero=False),
            axistick_opts=opts.AxisTickOpts(is_show=False),
            splitline_opts=opts.SplitLineOpts(is_show=False),
            axislabel_opts=opts.LabelOpts(is_show=False),
            split_number=20,
            min_="dataMin",
            max_="dataMax",
        ),
        yaxis_opts=opts.AxisOpts(
            grid_index=1,
            is_scale=True,
            split_number=2,
        ),
        legend_opts=opts.LegendOpts(is_show=False),
    )
)
```

最后将 K 线图、两条移动均线图和交易量柱状图组合。

```
# Kline And Line
overlap_kline_line = kline.overlap(line)

# Grid Overlap + Bar
grid_chart = Grid( init_opts=opts.InitOpts(width="1000px", height="600px") )

grid_chart.add(
    overlap_kline_line,
    grid_opts=opts.GridOpts(pos_left="10%", pos_right="8%", height="50%")
    )
grid_chart.add(
    bar,
    grid_opts=opts.GridOpts(pos_left="10%", pos_right="8%", pos_top="70%", height="16%")
)

grid_chart.render_notebook()
```

看效果吧。

4 总结

太累了不想总结了，对 pyecharts v1.0 记着一点就行了：

一切皆配置（options），细节都在里面。

其他的都可以查文档，或者在函数中按“shift + tab”来查看有那些参数。