Python进行优雅的画图

Wechat:skyloving123 作者：朝天椒 公众号：辣椒哈哈

关于公众号：用于记录吃着辣椒的学习旅程

画图与数据挖掘（python）

> - 在实际工作中，处理数据挖掘相关的问题时，基本步骤有：数据分析、数据处理、特征分析与提取、模型评价与选择以及模型的融合等步骤组成。如果数据量不是那种大到无法进行画图分析的话，画图分析将是数据挖掘分析与建模中最有用的一种技能，对于数据的探索与特征的提取有着魔法的魅力，关于数据分析与挖掘这个岗位，朝天椒后期对出一篇自己工作后的理解，这里不进行具体叙述。因此，要成为一名能在数据中找到价值和变现能力的数据分析与挖掘工程师，画图是不可或缺的能力。朝天椒收集了各路大神使用python进行优雅的画图的方法以及自己的一些总结，故想通过本文展示给大家。

python画图的工具介绍

> 使用python进行数据分析常用的两个代码工具s’p’yder、jupyter notebook。个人比较喜欢使用jupyter notebook进行数据分析，使用jupyter notebook进行画图分析时，有很多的插件能够帮助我们提升开发的效率，下面介绍jupyter notebook中对数据分析有用的一些插件：
>
> - 对于jupyter notebook的使用就不进行介绍了，如果不懂的自行百度学习，安装jupyter notebook后，在cmd下依次运行如下命令即可：

```
1: pip install jupyter_contrib_nbextensions
2: jupyter contrib nbextension install --user
```

- 有时候当我们使用jupyter notebook过程久了回出现控制插件不显示在我们的网页当中，出现这个问题的原因一般都是其相关的依赖包不匹配，具体做法就是将相关的依赖包进行更新即可：

```
pip install upgrade jupyter-contrib-nbextensions
pip install upgrade jupyter-nbextensions-configurator
```

- 比较好用的插件：table of contents（目录）， hinterland（代码提示）， snippets menu（代码重复）， split cell notebook（核的并行排版），在插件上将上面这些插件给选上对后续的数据分析有很大的效率提升作用。

- 其中qgrid包和autopep8包是比较好用的包， 但是安装时，一定要注意它所依赖的包的版本，如果使用各个包安装完成了，并没有报相关的错误，但是先jupyter notebook上面无法进行显示，可以换一个默认的jupyter notebook浏览器，这里推荐使用chrome，朝天椒使用ie时会出现显示不出来的问题。
- **qgrid的安装**

```
# qgird依赖三个包，pandas， ipywidgets， notebook
pip install --upgrade pandas

pip install ipywidgets
jupyter nbextension enable --py widgetsnbextension

pip install qgrid
jupyter nbextension enable --py --sys-prefix qgrid
```

- **qgrid的使用**

```
from ipywidgets import interact, interact_manual
import ipywidgets as widgets
import qgrid
qgrid.show_grid(df, show_toolbar=True)
```

python画图中好用的包

- matplotlib
- pandas
- seaborn
- plotly
- pyecharts

画图的基本设置

我们在对数据进行图像展示的时候，由于数据的原因，经常会有一些是中文的标签，为了能够显示中文，需要在画图的前面调加两句基本设置：

```
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号
```

matplotlib

在python各类画图工具中这个包是最古老的，画图的难度也是最大的，画图不仅步骤繁琐，封装的不是很好，而且参数的介绍也不够友好。但是很多的包又是在其基础上进行扩展而来的，因此，了解该包的画图方法还是有必要的，但是实际工作中一般用不到那么高深的画图，因此，尽可能的不要使用该包进行画图分析。

- 使用mat进行画图，主要包括如下几个步骤：
    1. 准备好数据

```
2. 构建一张图
3.  选用什么样的图标对数据进行展示
4.  对图的一些基本信息进行设置
5.  对特定的位置添加注释：plt.text(x,y,c)
5.  显示，保持图片
```

- 具体的流程如下所示：

```
x = np.arange(0, 100)
fig = plt.figure(figsize=(10,10)) # 构建图
plt.plot(x, x, label='hh', c='red', ls='--', marker='8') # 画线性图
plt.xlabel('x') # 设置x轴标签
plt.ylabel('y') # 设置y轴标签
```

```
plt.xlim([0,120]) # 设置x轴的大小范围
plt.ylim([0,120]) # 设置y轴的大小范围
plt.xticks([i for i in range(100) if i % 10 == 0],rotation=35)
# 设置x轴的刻度
plt.yticks([i for i in range(100) if i % 10 == 0]) # 设置y轴的刻度
#其中对于yticks还有其它的一种用法就是将本来的显示标称进行特殊化的映射
plt.yticks(old_name, new_name)
plt.text(x=10, y=10, s='hello', fontsize=11) # 给某个具体的位置进行标注
plt.legend(loc = 'upper left') # 显示label以及label的位置，如果不设置这个
# 参数图中  不会出现label
plt.show() #显示图片
plt.savefig('./aa.png') # 保持图片
```

- 如果想要对图片的刻度与大小进行设置，则需要通过axis容器进行设置：调用容器xaxis有两种方法，一种是通过定义的图的属性，另一个是通过mat自带的属性进行调用：

```
fig = plt.figure(figsize=(10, 5))

axis = plt.gca().xaxis
axis.get_ticklabels()    #得到刻度标签;
axis.get_ticklines()    # 得到刻度线;
axis.get_ticklines(minor = True)    #得到次刻度线;
# 举个例子:就像我们的尺子上的厘米的为主刻度线,毫米的为次刻度线;
for label in axis.get_ticklabels():
    label.set_color('red')         #设置每个刻度标签的颜色;
    label.set_rotation(45)     #旋转45度;
    label.set_fontsize(16)      #设置字体大小;
for line in axis.get_ticklines():
    line.set_color('green')
    line.set_markersize(15)       #设置刻度线的长短;
    line.set_markeredgewidth(3)       #设置线的粗细
plt.show()
```

- 如果想要画出多个图，有多种实现的方法，本人习惯于比较容易理解的那种写法：
  1. 首先创建一张图表：plt.figure()
  2. 然后将图表一张一张的添加到这张大的图表上面：plt.add_subplot()
  3. 在这张小的图表上进行画图操作例如：ax1 = plt.plot(x, x)
  4. 设置这张小的图表的一些基本的图像属性：ax1.set_xlabel('x1')等。这里需要注意一点的是，进行小图表的设置的时候，有两种方法，一种是通过每个小图表自带的属性接口进行设置，还有一些就是通过plt所自带的图表属性的接口进行设置，这个看个人的习惯，比较好理解的是通过小图表自带的属性接口进行设置。

5. 重复上述的2-4的操作过程，直到添加的图片的个数满足所想要的结果

```
fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(221)
```

```
    ax1.plot(x, x)
    # 1. 通过添加的小图标自带的属性接口进行图表的设置的写法
    ax1.set_xlabel('x1'), ax1.set_ylabel('y1'), ax1.set_xlim([0, 100])
    # 设置x和y轴上的刻度属性
    ax1.set_xticklabels(
            [i for i in range(100) if i %10 == 0],rotation=35)
    # 2. 通过mat原始的设置图表基本属性的写法
    plt.xlabel('x1'), plt.ylabel('y1'), plt.xlim([0, 100])
    plt.xticks([i for i in range(100) if i%10==0])

    # 接着对对二张图片进行设置
    ax2 = fig.add_subplot(222)
    ax2.plot(x, -x)
    ax3 = fig.add_subplot(223)
    ax3.plot(x, x ** 2)
    ax4 = fig.add_subplot(224)
    ax4.plot(x, np.log(x))
    plt.legend(loc='best')
    plt.show()
```

- 其它一些经常会用到的图：
  1. 直方图： plt.hist()
  2. 散点图： plt.scatter()
  3. 柱状图：plt.bar(), plt.barh()

- 对于mat上述的画图方法，如果数据的格式是DataFrame类型，就不需要这么复杂，下面介绍如果数据是pandas中的series和dataframe格式的数据，如何进行优雅的画图。

Pandas

Pandas为了能够更加有效和方便的进行数据分析，将mat进行了封装，因此，如果数据是dataframe格式，可以调用自动的接口进行画图数据分析。

- pandas进行画图的基本操作属性如下所示：
  1. 线条的形状：linestyle
  2. 标题： title
  3. 字体设置：fontsize
  4. 颜色的设置：color color
  5. 点的形状：marker
  6. 集合style(linestyle, color, marker) =’–o’
  7. 图表的透明度：alpha
  8. 坐标轴刻度的旋转：rotation=35,

9. 是不是网格形状：grid ()
10. 颜色映射于画板的选择：colormap='summer' cmpas
11. 什么类型的图表：kind(默认图表的形状是line)

```
# 线性图的构建方法
fig = df['a', 'b'].plot(figsize=(10, 5), kind='line')
```

对于其它类型的图的构建方法有两种写法，一种事通过plot种的kind参数进行控制，另一种方法是通过自带的接口进行控制，这个在性能上没有什么大的不同，看个人习惯就好：

```
# 方法一：
fig = df.plot(figsize=(10,5), kind='scatter', x='a', y='b')
# 方法二：
fig = df.plot.scatter(x='a', y='b', figsize=(10, 5),
                      alpha=0.8, colormap='summer_r')
```

- 对于数据格式的Dataframe形状的数值型数据如果想要对其进行线性图的描述，和mat的步骤差不多，不同的就是 构建一个图的时候，pandas可以通过将数据直接导入到图表种，而后面的操作方法和mat的方式基本没什么差别，pandas有个好处就是可以将格式化中的多列数据直接进行展示， 但是画出来的的x轴默认是index，这个需要注意，将需要为x轴的设置为index，pandas数据格式可以很方便的进行构建，使用pandas对多列数据进行画图时，可以控制子图参数进行分开：
  1. use_index：将索引设置引用为刻度标签
  2. stacked：堆砌操作将多个一起进行比较 针对的是柱状图操作
  3. subplots：将多个图分开操作 针对的是线性图操作 结合layout进行填充
  4. layout：控制子图的大小
- pandas多个图的优雅画法：plot

```
# 线性图
df.plot(kind='line', alpha=0.8, figsize=(10,6),
        subplots=True, layout=(1, 4),
        use_index=True,
        legend=True)
# 比如时间序列的线性图可以将train和test分开，在放到一起画图
train.Count.plot(figsize=(15, 8),
                 title='Daily ridership', fontsize=14)
test.Count.plot(figsize=(15, 8),
                title='Daily ridership', fontsize=14)
# 柱状图堆砌图
df.plot(kind='bar', alpha=0.8, figsize=(10,6),
        grid=True, stacked=True,
        facecolor='r', #柱状图的填充颜色
```

```
            edgecolor='b'   #柱状图边框的填充颜色
            yerr=y*0.1      #xerr/yerr为柱状图种bar的大小设置
            )
    # 一个在柱状图上进行文件的添加
    for i, j in zip(x, y):
        plt.text(i-0.2, j-0.15, '%.2f' % j, color='white')
    # 饼图
    s = pd.Series(x*np.random.rand(3), index=['a', 'b', 'c', 'd'],
                    name='seres')
    plt.axis('equal') #设置是否为一个圆
    s.plot.pie(
               explode=[0.1, 0, 0, 0], #指定每部分偏移圆的大小
               labels=s.index, #圆各部分的标签
               colors=['r','g','b','c'], #颜色设置
               autopct='%.2f%%', #圆种各个比例的设置
               pctdistance=0.6, #每个饼切片中心和通过autopct生成的文本之间的比例
               labeldistance=1.2, #
               shadow=True, #阴影
               startangle=0, #开始角度
               radius=1.5, #半径
               frame=False) #图框
    # 散点矩阵图:
    df = pd.DataFrame(np.random.randn(40, 4), columns=list('abcd'))
    pd.scatter_matrix(df, figsize=(3,4),marker='o',
                      diagonal='kde', #是否为直方图于核密度图，'hist'
                      alpha=0.5,
                      range_padding=0.5 #对靠近x和y轴的进行留白天从，
                      #值越大，留白距离越大
                      )
    # 其它的图如箱型图之内的画法上述的基本一致就不一一展示
```

seaborn

- seaborn也被称为sb（因为封装的很好的原因，调用非常的简单），但是简写一般为sn，sn是在mat的基础上进行了一层更加简单的封装，因此，使用sn进行画图可以更加的简单和方便， 在有了上述mat的基础上再去理解sn的画图就很简单了，不过感觉sn比较好用的就是它的`联动性和分组画图：可以在画图的时候按照某个字段进行区分以及可以一次对不同的字段进行多张图的展示`。

- 下面介绍一个关于sn画出相关性特别漂亮的code:

```
    import seaborn as sns

    import matplotlib.pyplot as plt
    def plot_heat(train):
```

```
    corr = train.corr()#计算各变量的相关性系数
    xticks = list(corr.index)#x轴标签
    yticks = list(corr.index)#y轴标
    fig = plt.figure(figsize=(13,10))
    ax1 = fig.add_subplot(1, 1, 1)
    sns.heatmap(corr, annot=True,
         cmap="rainbow",ax=ax1,linewidths=5,
         annot_kws={'size': 9, 'weight': 'bold', 'color': 'blue'})
    ax1.set_xticklabels(xticks, rotation=35, fontsize=15)
    ax1.set_yticklabels(yticks, rotation=0, fontsize=15)

    plt.show()
```

- 由于sn基本都是一些api函数的调用，因此，不需要进行理解，只要去找到能够实现想要功能的图表的接口函数就好。下面直接就撸上sn的api和一个博客介绍
- sn_api
- sn_zh_blog
- all

plotly

- **在使用plotly的时候最好连接到外网，不然查询api速度有点慢。**
- 上述的画图都有一些不好的地方，就是画出来的图不怎么好看，而且在jupyter notebook上的显示是静态的（当鼠标点在图上不会有数字的变化）， 而plotly就可以完美的解决上述的问题，这个包是一个在us读博士的同事玩的时候学到的，虽然这个包画出来的图很美观，但是相对于其它的两个包画出来的出会占用更大的内存：
- plotly的画图分为两种，在线和离线两种，在线的注册一个账号可以保存到云端，朝天椒大部分时间都是使用jupyter notebook做数据分析，因此，大部分实际是通过离线的导包方式，`如果读者想要将分次的图片保存到云端，可以自己注册账号进行研究`,使用plotly的常用基本步骤和mat差不多，甚至更加的方便一些如下所示：
    1. 画一个想要的图表，可以一次画多个
    2. 设置图表的基本属性
    3. 对图表进行打包
    4. 将各图表打包到一张大图上
    5. 画图显示
- 上面的5个步骤分分别对应plotly的画图步骤，通过上述步骤结合jupyter notebook就能画出十分漂亮的图表出来，真的画出来的图十分的漂亮，该包的api文档介绍做的非常的友好，每种图形都有相应的code例子说明，而且如果对mat有一定的理解后，上手这个包的操作分常容易理解。

- **plot画图的基本过程**
    1. 定义一个图形的形状go.Scatter()
    2. 设置图形的基本属性go.Layout()

3. 将图层进行叠加go.Figure()

- **plotly参数的基本介绍**

```
# 图形的基本设置
      marker=dict(color='#483D8B',#设置条形图的颜色
      line=dict(color='rgb(256, 256, 256)',width=1.0,)),#设置条形图边框
      name='总次数',#设置这个图的名字，和图例对应
      orientation='h',#如果水平条形图需设置，竖直条形图不用设置
      opacity=0.9)#条形图颜色的不透明度
# 散点图的特有属性
      mode='text',#设置画图的种类，
      有'markers+text'、 mode='lines+markers+text',等各种组合
      textfont=dict(size=32,color='balck'),#设置标签的字体
      marker=dict(size=32,color='black',
                      line=dict(width=1,color='black'),),
# 图层的一些基本设置
    plot_bgcolor='#E6E6FA',#图的背景颜色
    paper_bgcolor='#F8F8FF',#图像的背景颜色
    autosize=False,width=1450,height=800,#设置图像的大小
    #设置图离图像四周的边距
    margin=go.Margin(l=480,r=60,b=50,t=60,pad=0),#pad参数是刻度与标签的距离
    #设置y轴的刻度和标签
    yaxis=dict(title='人均会议申请次数',#设置坐标轴的标签
        titlefont=dict(color='rgb(148, 103, 189)',size=24),
        #设置坐标轴标签的字体及颜色
        tickfont=dict(color='rgb(148, 103, 189)',size = 24,),
        #设置刻度的字体大小及颜色
        showticklabels=False,#设置是否显示刻度
            #设置刻度的范围及刻度
        autorange=False,range=[-0.05674507980728292,   -0.052731042093320
4],type='linear',
),
```

- **画单个图以及单个图放到一个大图的操作步骤**

```
    # 以散点图为例
    import plotly.offline as offline
    offline.init_notebook_mode(connected=False)
    from plotly import tools
    import plotly.graph_objs as go

    trace0 = go.Scatter(
                        x=[1,2,3,4],
                        y=[10,15,13,17],
                        mode='markers+lines',
                        name='figure_1'
    )
```

```
    layout = go.Layout(
                        xaxis=dict(title='x'),
                        yaxis=dict(title='y')
    )
    data = go.Data([trace0])
    fig = go.Figure(data=data, lay
out=layout)
    offline.iplot(fig, show_link=False)
```

plotly中将一张大图种各个小图分开画有两种写法，有一种由于写法比较容易混淆，朝天椒比较喜欢那种比较容易理解的写法， 与上面的单个图的画法不同之处在于，plotly通过一个tools类将子图等分布布局与操作封装起来，

- 其中启动子图的api函数基本包括了子图的基本属性的设置方法，比如共享x和y等

```
    x = ['1970-01-01', '1970-01-01', '1970-02-01', '1970-04-01',
         '1970-01-02', '1972-01-31', '1970-02-13', '1971-04-19']
    trace0 = go.Histogram(

        x=x,
        nbinsx = 4,
      )

    trace1 = go.Histogram(
        x=x,
        xbins=dict(
            start='1969-11-15',
            end='1972-03-31',
            size= 'M18'),
        autobinx = False
    )
    trace2 = go.Scatter(
        x=x,
        y=[i for i in range(len(x))]
      )
    # 启动子图窗口的函数设置和标题的设置方法
    fig = tools.make_subplots(rows=3, cols=2,  subplot_titles=(
                              'plot1', 'plot2', 'plot3', 'plot4'))
    fig.append_trace(trace0, 1, 1)
    fig.append_trace(trace1, 1, 2)
    fig.append_trace(trace2, 2, 1)
    # 对图的基本的框架属性的重新配置方法
    fig['layout'].update(height=600, width=1200,
                title='This is a multiple figure',
                xaxis=dict(domain=(0, 0.7), anchor='x2'),
                )# 用于控制图片在母图中的比例大小
```

- 一个对于时序问题变化进行很好的画图方法

```
# 这里关键之处在于，df的索引必须为datatime形式的时间索引值
    trace = go.Scatter(x=list(df.idnex), y=list(df.High))
    data = [trace]
    layout = dict(
    title='time seriers with range slider and selectors'),
    xaxis=dict(
        rangeselector=dict(
            buttons=list([
                dict(count=1,
                    label='1m',
                    step='month',
                    stepmode='backward'),
                dict(count=1,
                    label='6m',
                    step='month',
                    stepmode='backward'),
                dict(count=6,
                    label='1m',
                    step='month',
                    stepmode='backward'),
                dict(count=1,
                    label='YID',
                    step='year',
                    stepmode='backward'),
                dict(count=1,
                    label='1y',
                    step='year',
                    stepmode='backward'),
                dict(step='all')
            ])
        ),
        rangeslider=dict(visible=True),
        type='date'
    )
)
fig = dict(data=data, layout=layout)
offline.iplot(fig)
```

plotly相对于mat来说看起来没那么简单，但是其实是更加的简单，因为很多的东西都已经封装的很好了，这和世上的道理十分的相似，看起来简单的东西其实学习都不简单，看起来复杂的东西其实学习来都是简单的，因此，设计的原理就是这样。所以如果大家想要什么样的图直接打开plotluy的传送门就可：plotly_api。