高效使用 Python 可视化工具 Matplotlib

伯乐专栏/李大萌 Python开发者 2017-07-03

> 编译：伯乐在线 - 李大萌
> 如有好文章投稿，请点击 → 这里了解详情

Matplotlib是Python中最常用的可视化工具之一,可以非常方便地创建海量类型的2D图表和一些基本的3D图表。本文主要介绍了在学习Matplotlib时面临的一些挑战，为什么要使用Matplotlib，并推荐了一个学习使用Matplotlib的步骤。

简介

对于新手来说，进入Python可视化领域有时可能会令人感到沮丧。Python有很多不同的可视化工具，选择一个正确的工具有时是一种挑战。 例如，即使两年过去了，这篇《Overview of Python Visualization Tools》是引导人们到这个网站的顶级帖子之一。 在那篇文章中，我对matplotlib留下了一些阴影，并在分析过程中不再使用。 然而，在使用诸如pandas，scikit-learn，seaborn和其他数据科学技术栈的python工具后，觉得丢弃matplotlib有点过早了。说实话，之前我不太了解matplotlib，也不知道如何在工作流程中有效地使用。

现在我花时间学习了其中的一些工具，以及如何使用matplotlib，已经开始将matplotlib看作是不可或缺的工具了。这篇文章将展示我是如何使用matplotlib的，并为刚入门的用户或者没时间学习matplotlib的用户提供一些建议。我坚信matplotlib是python数据科学技术栈的重要组成部分，希望本文能帮助大家了解如何将matplotlib用于自己的可视化。

为什么对matplotlib都是负面评价?

在我看来，新用户学习matplotlib之所以会面临一定的挑战，主要有以下几个原因。

首先，matplotlib有两种接口。第一种是基于MATLAB并使用基于状态的接口。第二种是面向对象的接口。为什么是这两种接口不在本文讨论的范围之内，但是知道有两种方法在使用matplotlib进行绘图时非常重要。

两种接口引起混淆的原因在于，在stack overflow社区和谷歌搜索可以获得大量信息的情况下，新用户对那些看起来有些相似但不一样的问题，面对多个解决方案会感到困惑。从我自己的经历说起。回顾一下我的旧代码，一堆matplotlib代码的混合——这对我来说非常混乱（即使是我写的）。

> 关键点
>
> matplotlib的新用户应该学习使用面向对象的接口。

matplotlib的另一个历史性挑战是，一些默认风格选项相当没有吸引力。 在R语言世界里，可以用ggplot生成一些相当酷的绘图，相比之下，matplotlib的选项看起来有点丑。令人欣慰的是matplotlib 2.0具有更美观的样式，以及非常便捷对可视化的内容进行主题化的能力。

使用matplotlib我认为第三个挑战是，当绘制某些东西时，应该单纯使用matplotlib还是使用建立在其之上的类似pandas或者seaborn这样的工具，你会感到困惑。任何时候都可以有多种方式来做事，对于新手或不常用matplotlib的用户来讲，遵循正确的路径是具有挑战性的。将这种困惑与两种不同的API联系起来，是解决问题的秘诀。

为什么坚持要用matplotlib?

尽管有这些问题，但是我庆幸有matplotlib，因为它非常强大。这个库允许创建几乎任何你可以想象的可视化。此外，围绕着它还有一个丰富的python工具生态系统，许多更先进的可视化工具用matplotlib作为基础库。如果在python数据科学栈中进行任

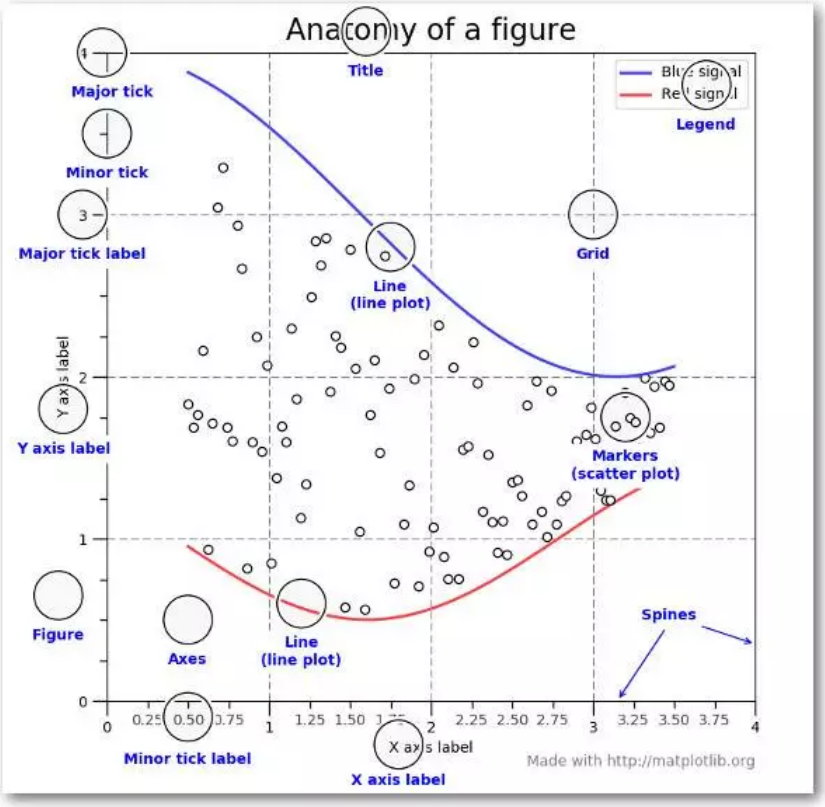
何工作，都将需要对如何使用matplotlib有一个基本的了解。这是本文的其余部分的重点——介绍一种有效使用matplotlib的基本方法。

基本前提

如果你除了本文之外没有任何基础，建议用以下几个步骤学习如何使用matplotlib：

1. 学习基本的matplotlib术语，尤其是什么是图和坐标轴
2. 始终使用面向对象的接口，从一开始就养成使用它的习惯
3. 用基础的pandas绘图开始你的可视化学习
4. 用seaborn进行更复杂的统计可视化
5. 用matplotlib来定制pandas或者seaborn可视化

这幅来自matplotlib faq的图非常经典，方便了解一幅图的不同术语。



大多数术语都非常直接，但要记住的要点是，Figure是最终的图像，可能包含一个或多个坐标轴。坐标轴代表一个单独的划分。一旦你了解这些内容，以及如何通过面向对象的API访问它们，下面的步骤才能开始进行。

这些术语知识有另一个好处，当你在网上看某些东西时，就有了一个起点。如果你花时间了解了这一点，才会理解matplotlib API的其余部分。此外，许多python的高级软件包，如seaborn和ggplot都依赖于matplotlib。因此，了解这些基础知识后再学那些功能更强大的框架会容易一些。

最后，我不是说你应该避免选择例如ggplot（aka ggpy），bokeh，plotly或者altair等其他更好的工具。我只是认为你需要从对matplotlib + pandas + seaborn 有一个基本了解开始。一旦理解了基本的可视化技术，就可以探索其他工具，并根据自己的需要做出明智的选择。

入门

本文的其余部分将作为一个入门教程，介绍如何在pandas中进行基本的可视化创建，并使用matplotlib自定义最常用的项目。一旦你了解了基本过程，进一步的定制化创建就相对比较简单。

重点讲一下我遇到的最常见的绘图任务，如标记轴，调整限制，更新绘图标题，保存图片和调整图例。如果你想跟着继续学习，在链接https://github.com/chris1610/pbpython/blob/master/notebooks/Effectively-Using-Matplotlib.ipynb 中包含附加细节的笔记，应该非常有用。

准备开始，我先引入库并读入一些数据：

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

df = pd.read_excel("https://github.com/chris1610/pbpython/blob/master/data/sample-salesv3.xlsx?raw=true")
df.head()
```

	account number	name	sku	quantity	unit price	ext price	date
0	740150	Barton LLC	B1-20000	39	86.69	3380.91	2014-01-01 07:21:51
1	714466	Trantow-Barrows	S2-77896	-1	63.16	-63.16	2014-01-01 10:00:47
2	218895	Kulas Inc	B1-69924	23	90.70	2086.10	2014-01-01 13:24:58
3	307599	Kassulke, Ondricka and Metz	S1-65481	41	21.05	863.05	2014-01-01 15:05:22
4	412290	Jerde-Hilpert	S2-34077	6	83.21	499.26	2014-01-01 23:26:55

这是2014年的销售交易数据。为了使这些数据简短一些，我将对数据进行聚合，以便我们可以看到前十名客户的总购买量和总销售额。为了清楚我还会在绘图中重新命名列。

```
top_10 = (df.groupby('name')['ext price', 'quantity'].agg({'ext price': 'sum', 'quantity': 'count'})
         .sort_values(by='ext price', ascending=False))[:10].reset_index()
top_10.rename(columns={'name': 'Name', 'ext price': 'Sales', 'quantity': 'Purchases'}, inplace=True)
```

下面是数据的处理结果。

	Name	Purchases	Sales
0	Kulas Inc	94	137351.96
1	White-Trantow	86	135841.99
2	Trantow-Barrows	94	123381.38
3	Jerde-Hilpert	89	112591.43
4	Fritsch, Russel and Anderson	81	112214.71
5	Barton LLC	82	109438.50
6	Will LLC	74	104437.60
7	Koepp Ltd	82	103660.54
8	Frami, Hills and Schmidt	72	103569.59
9	Keeling LLC	74	100934.30

现在，数据被格式化成一个简单的表格，我们来看如何将这些结果绘制成条形图。

如前所述，matplotlib有许多不同的样式可用于渲染绘图，可以用plt.style.available查看系统中有哪些可用的样式。

```
plt.style.available
```

```
['seaborn-dark',
'seaborn-dark-palette',
'fivethirtyeight',
'seaborn-whitegrid',
'seaborn-darkgrid',
'seaborn',
'bmh',
'classic',
'seaborn-colorblind',
'seaborn-muted',
'seaborn-white',
'seaborn-talk',
'grayscale',
'dark_background',
'seaborn-deep',
'seaborn-bright',
'ggplot',
'seaborn-paper',
'seaborn-notebook',
'seaborn-poster',
'seaborn-ticks',
'seaborn-pastel']
```

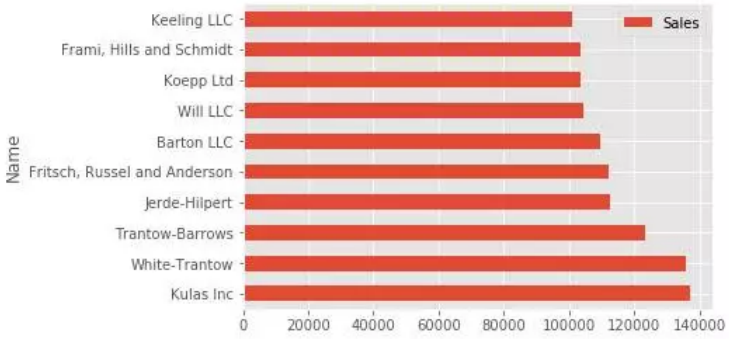
这样简单使用一个样式：

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

我鼓励大家尝试不同的风格，看看你喜欢哪些。

现在我们准备好了一个更美观的样式，第一步是使用标准的pandas绘图功能绘制数据：

```
top_10.plot(kind='barh', y="Sales", x="Name")
```



我推荐先使用pandas绘图，是因为它是一种快速简便构建可视化的方法。 由于大多数人可能已经在pandas中进行过一些数据处理/分析，所以请先从基本的绘图开始。

定制化绘图

假设你对这个绘图的要点很满意，下一步就是定制它。使用pandas绘图功能定制（如添加标题和标签）非常简单。但是，你可能会发现自己的需求在某种程度上超越该功能。这就是我建议养成这样做的习惯的原因：

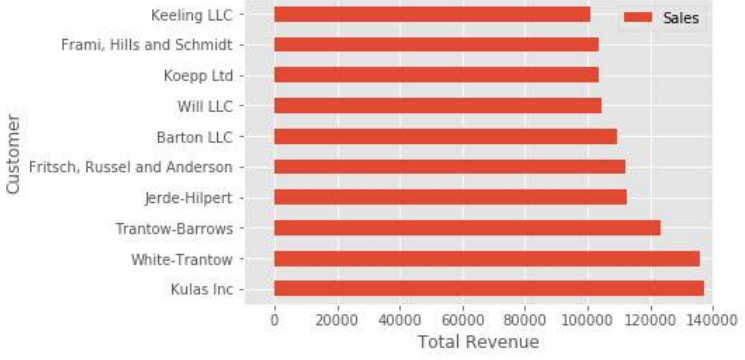
```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
```

得到的图看起来与原始图看起来相同，但是我们向plt.subplots() 添加了一个额外的调用，并将ax传递给绘图函数。为什么要这样做？ 记得当我说在matplotlib中要访问坐标轴和数字至关重要吗？这就是我们在这里完成的工作。将来任何定制化都将通过ax或fig对象完成。

我们得益于pandas快速绘图，获得了访问matplotlib的所有权限。我们现在可以做什么呢？用一个例子来展示。另外，通过命名约定，可以非常简单地把别人的解决方案改成适合自己独特需求的方案。

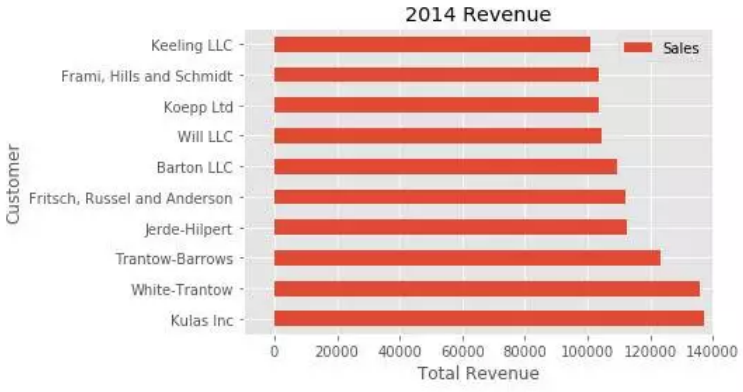
假设我们要调整x限制并更改一些坐标轴的标签？现在坐标轴保存在ax变量中，我们有很多的控制权：

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer');
```



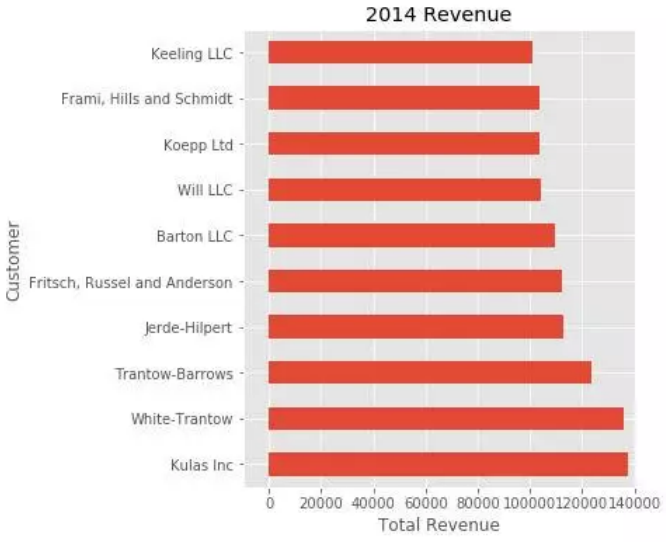
下面是一个快捷方式，可以用来更改标题和两个标签：

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')
```



为了进一步验证这种方法，还可以调整图像的大小。通过plt.subplots() 函数，可以用英寸定义figsize。也可以用ax.legend().set_visible（False）来删除图例。

```
fig, ax = plt.subplots(figsize=(5, 6))
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue')
ax.legend().set_visible(False)
```



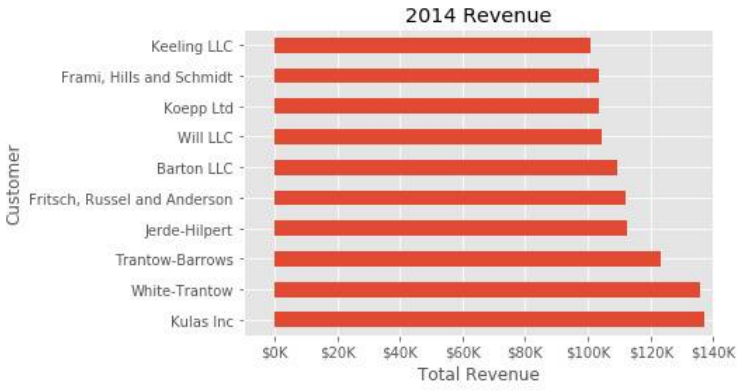
基于很多原因你可能想要调整一下这个图。看着最别扭的地方是总收入数字的格式。 Matplotlib可以通过FuncFormatter来帮我们实现。这个功能可以将用户定义的函数应用于值，并返回一个格式整齐的字符串放置在坐标轴上。

下面是一个货币格式化函数，可以优雅地处理几十万范围内的美元格式：

```
def currency(x, pos):
    'The two args are the value and tick position'
    if x >= 1000000:
        return '${:1.1f}M'.format(x*1e-6)
    return '${:1.0f}K'.format(x*1e-3)
```

现在我们有一个格式化函数，需要定义它并将其应用到x轴。以下是完整的代码：

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')
formatter = FuncFormatter(currency)
ax.xaxis.set_major_formatter(formatter)
ax.legend().set_visible(False)
```

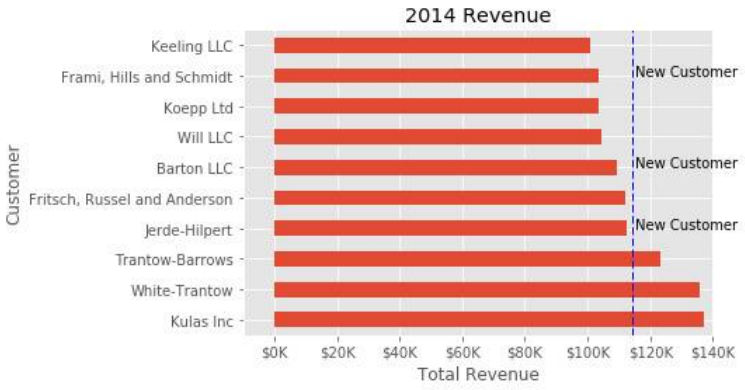


这样更美观，也是一个很好的例子，展示如何灵活地定义自己的问题解决方案。

我们最后要去探索的一个自定义功能是通过添加注释到绘图。绘制一条垂直线，可以用ax.axvline()。添加自定义文本，可以用ax.text()。

在这个例子中，我们将绘制一条平均线，并显示三个新客户的标签。 下面是完整的代码和注释，把它们放在一起。

```
# Create the figure and the axes
fig, ax = plt.subplots()

# Plot the data and get the averaged
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
avg = top_10['Sales'].mean()

# Set limits and labels
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')

# Add a line for the average
ax.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

# Annotate the new customers
for cust in [3, 5, 8]:
    ax.text(115000, cust, "New Customer")

# Format the currency
formatter = FuncFormatter(currency)
ax.xaxis.set_major_formatter(formatter)

# Hide the legend
ax.legend().set_visible(False)
```



虽然这可能不是让人感到兴奋（眼前一亮）的绘图方式，但它展示了你在用这种方法时有多大权限。

图形和图像

到目前为止，我们所做的所有改变都是单个图形。幸运的是，我们也有能力在图上添加多个图形，并使用各种选项保存整个图像。

如果决定要把两幅图放在同一个图像上，我们应对如何做到这一点有基本了解。 首先，创建图形，然后创建坐标轴，然后将其全部绘制在一起。我们可以用plt.subplots()来完成：

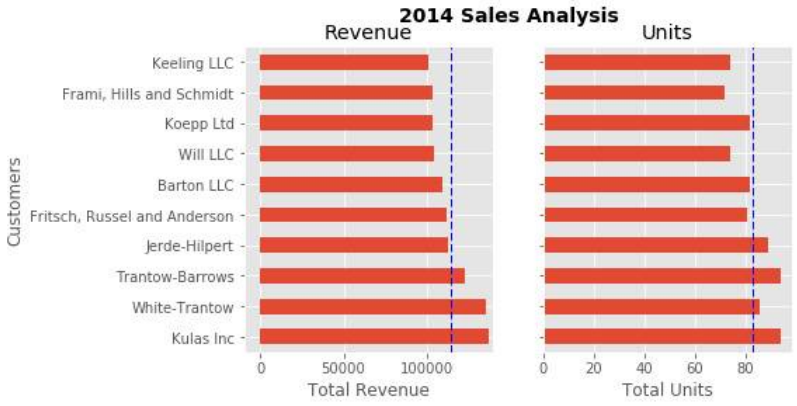
```
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(7, 4))
```

在这个例子中，用nrows和ncols来指定大小，这样对新用户来说比较清晰。在示例代码中，经常看到像1,2这样的变量。我觉得使用命名的参数，之后在查看代码时更容易理解。

用sharey = True这个参数，以便yaxis共享相同的标签。

这个例子也很好，因为各个坐标轴被解压缩到ax0和ax1。有这些坐标轴轴，你可以像上面的例子一样绘制图形，但是在ax0和ax1上各放一个图。

```
# Get the figure and the axes
fig, (ax0, ax1) = plt.subplots(nrows=1,ncols=2, sharey=True, figsize=(7, 4))
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax0)
ax0.set_xlim([-10000, 140000])
ax0.set(title='Revenue', xlabel='Total Revenue', ylabel='Customers')

# Plot the average as a vertical line
avg = top_10['Sales'].mean()
ax0.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

# Repeat for the unit plot
top_10.plot(kind='barh', y="Purchases", x="Name", ax=ax1)
avg = top_10['Purchases'].mean()
ax1.set(title='Units', xlabel='Total Units', ylabel='')
ax1.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

# Title the figure
fig.suptitle('2014 Sales Analysis', fontsize=14, fontweight='bold');

# Hide the legends
ax1.legend().set_visible(False)
ax0.legend().set_visible(False)
```



到目前为止，我一直用jupyter notebook，借助%matplotlib内联指令来显示图形。但是很多时候，需要以特定格式保存数字，和其他内容一起展示。

Matplotlib支持许多不同格式文件的保存。 你可以用fig.canvas.get_supported_filetypes（）查看系统支持的格式：

```
fig.canvas.get_supported_filetypes()
```

```
{'eps': 'Encapsulated Postscript',
'jpeg': 'Joint Photographic Experts Group',
'jpg': 'Joint Photographic Experts Group',
'pdf': 'Portable Document Format',
'pgf': 'PGF code for LaTeX',
'png': 'Portable Network Graphics',
'ps': 'Postscript',
'raw': 'Raw RGBA bitmap',
'rgba': 'Raw RGBA bitmap',
'svg': 'Scalable Vector Graphics',
'svgz': 'Scalable Vector Graphics',
'tif': 'Tagged Image File Format',
'tiff': 'Tagged Image File Format'}
```

由于我们有fig对象，我们可以用多个选项来保存图像：

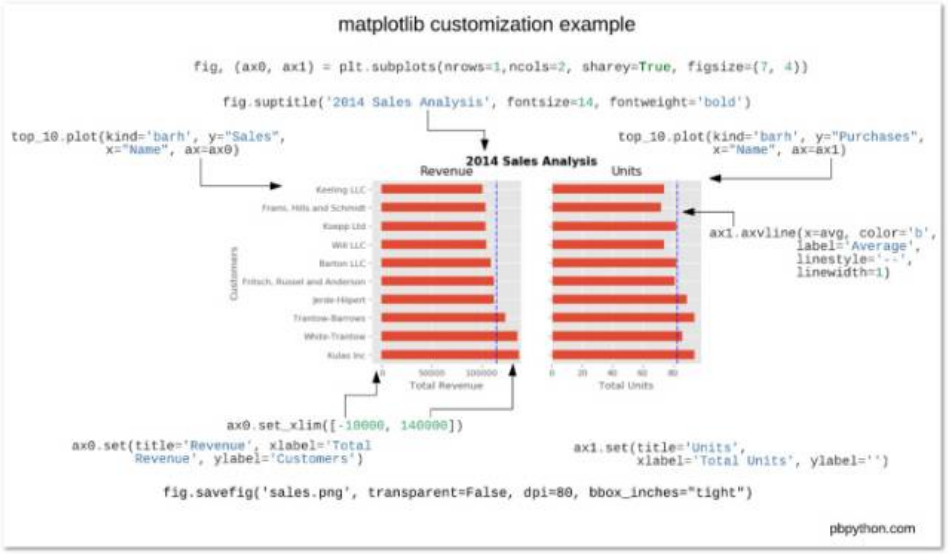
```
fig.savefig('sales.png', transparent=False, dpi=80, bbox_inches="tight")
```

上面的代码把图像保存为背景不透明的png。还指定了分辨率dpi和bbox_inches =“tight”来尽量减少多余的空格。

结论

希望这个过程有助于你了解如何在日常的数据分析中更有效地使用matplotlib。 如果在做分析时养成使用这种方法的习惯，你应该可以快速定制出任何你需要的图像。

作为最后的福利，我引入一个快速指南来总结所有的概念。希望这有助于把这篇文章联系起来，并为今后使用参考提供方便。



阅读原文