如何优雅而高效地使用Matplotlib实现数据可视化

By 路雪2018年1月16日 12:42

> Matplotlib 能创建非常多的可视化图表，它也有一个丰富的 Python 工具生态环境，很多更高级的可视化工具使用 Matplotlib 作为基础库。因此本文旨在提供一种高效的 Matplotlib 使用方法，并希望该方法可以帮助大家理解如何更有效地进行日常数据分析工作。

引言

对新手来说 Python 可视化实在有些令人挫败。有很多不同的选项，如何选择正确的选项是一个挑战。例如，两年前这篇文章《Overview of Python Visualization Tools》仍然吸引了大量读者。在那篇文章中，我否定了 Matplotlib。但是，在使用过 pandas、scikit-learn、seaborn 和其他 Python 数据科学包之后，我觉得之前否认 Matplotlib 的行为有点不成熟。坦白讲，当时我不是很了解 Matplotlib，也不懂如何在我的工作流中高效使用 Matplotlib。

现在我学习了一些工具，了解了如何基于 Matplotlib 使用这些工具，Matplotlib 逐渐变成了可视化工具的核心。本文将展示如何使用 Matplotlib。我坚定地认为 Matplotlib 是 Python 数据科学包必不可少的一部分，希望这篇文章可以帮助大家了解如何使用 Matplotlib 进行 Python 可视化。

为什么大家都在否定 Matplotlib？

我认为，Matplotlib 对于新手来说比较难存在几个原因。首先，Matplotlib 有两个界面。第一个界面基于 MATLAB，使用基于状态的接口。第二个界面是面向对象的接口。本文就不展开介绍 Matplotlib 有两个界面的原因，但了解这两种方法在使用 Matplotlib 绘图时会很重要。两个界面会引起混淆的原因可以通过 Stack Overflow 和谷歌搜索查找一些信息。此外，新用户将发现混淆问题有多个解决方案，但是这些问题看起来类似却不完全相同。从我的个人经验来讲，我们从以前的代码中可以看出有一些 Matplotlib 代码的混杂。

关键点

Matplotlib 新手应该学习和使用面向对象的接口。

使用 Matplotlib 的另一个历史性挑战是一些默认的样式缺乏吸引力。在 R 使用 ggplot 就可以生成相当不错的图，而 Matplotlib 相对来说有点丑。好消息是 Matplotlib 2.0 中的样式好看了很多，你可以用最小的努力生成可视化。

第三个挑战是你不确定什么时候该使用 Matplotlib，什么时候该使用基于 Matplotlib 构建的工具，如 pandas 或 seaborn。大部分时候做一件事都有多种选择，但是对于新手来说选择正确的道路有些困难。

为什么使用 Matplotlib？

尽管 Matplotlib 有这么多问题，我还是喜欢用它。因为它很强大，这个库允许你创建几乎所有的可视化图表。此外，围绕 Matplotlib 有一个丰富的 Python 工具生态环境，很多更高级的可视化工具使用 Matplotlib 作为基础库。因此如果你想在 Python 数据科学工具包中进行任何操作，你需要对如何使用 Matplotlib 有一些基础了解。这就是本文其余部分的重点，提供一种高效使用 Matplotlib 的基础方法。

前提

推荐以下步骤学习如何使用 Matplotlib：

1. 学习 Matplotlib 的基本术语，具体来说就是什么是 Figure 和 Axes。

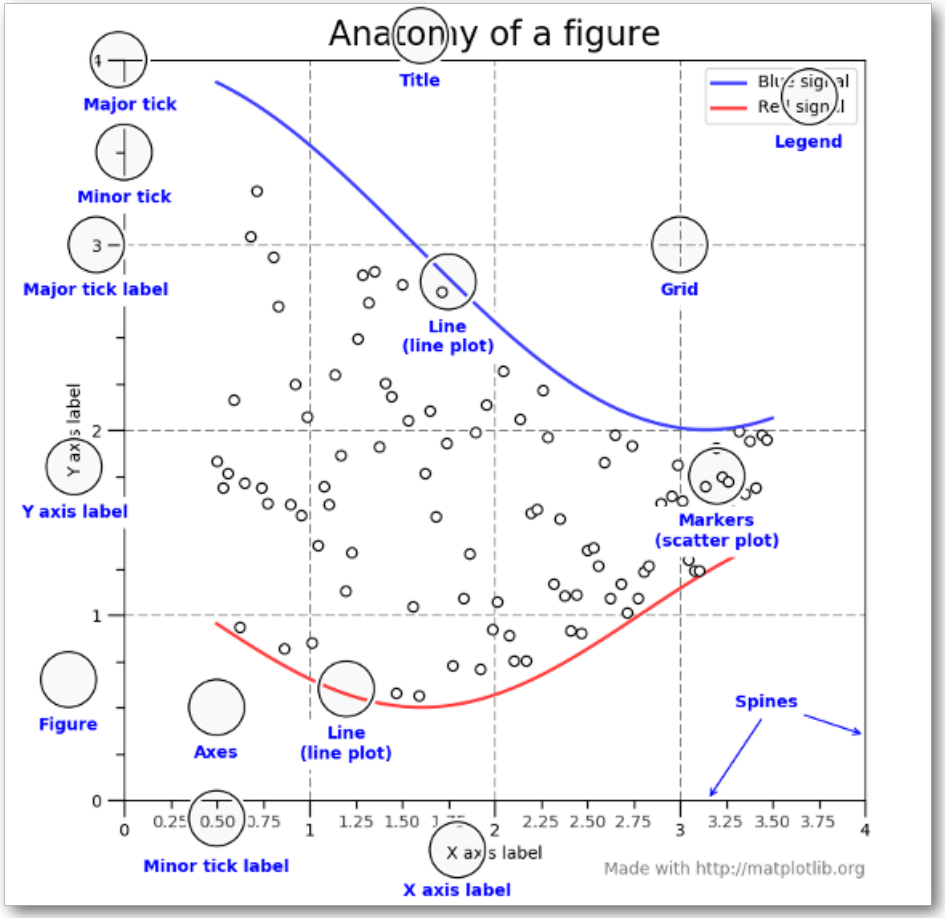
2. 一直使用面向对象的界面，养成习惯。

3. 用基础的 pandas 绘图开始可视化。

4. 使用 seaborn 进行稍微复杂的数据可视化。

5. 使用 Matplotlib 自定义 pandas 或 seaborn 可视化。

下图非常重要，有助于理解图的不同术语。



大部分术语很直接易懂，需要牢记的是 Figure 是可能包含一或多个 axes 的最终图像。Axes 代表单个图。一旦你理解这些是什么以及如何通过面向对象的 API 评估它们，其余步骤就很简单了。

了解这个知识还有一个好处，就是当你在网络上看东西的时候有一个出发点。如果你花时间了解了这个点，那么其他的 Matplotlib API 才有意义。此外，很多高级 Python 包，如 seaborn 和 ggplot 依赖于 Matplotlib 构建，因此理解了基础，学习更强大的框架才更加容易。

最后，我不是说你应该逃避其他优秀选项，如 ggplot（又名 ggpy）、bokeh、plotly 或 altair。我只是认为你需要对 matplotlib + pandas + seaborn 有一个基础的了解。了解基础可视化栈之后，你就可以探索其他优秀工具，根据需求做出合适的选择。

开始

下面主要介绍如何在 pandas 中创建基础的可视化以及使用 Matplotlib 定制最常用的项。了解基础流程有助于更直观地进行自定义。

我主要关注最常见的绘图任务，如标注轴、调整图形界限（limit）、更新图标题、保存图像和调整图例。

开始，我打算设置输入，读取一些数据：

```
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

数据包括 2014 年的销售交易额。为简短起见，我将总结这些数据，列出前十名客户的采购次数和交易额。绘图时我将对各列进行重命名。

```
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)
```

下图是数据。

	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

现在数据以简单的表格形式呈现，我们再来看一下如何将数据绘制成条形图。如前所述，Matplotlib 具备多种不同风格，可用于渲染图表。你可以使用 plt.style.available 查看你的系统可用的风格。

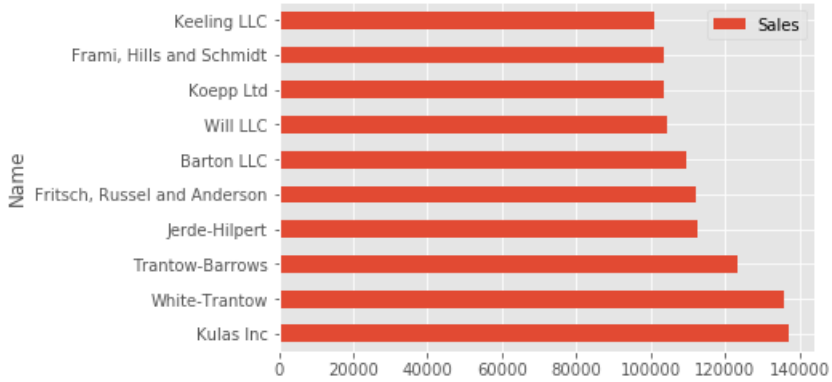
```
plt.style.available
['seaborn-dark',
 'seaborn-dark-palette',
 'fivethirtyeight',
 'seaborn-whitegrid',
 'seaborn-darkgrid',
 'seaborn',
 'bmh',
 'classic',
 'seaborn-colorblind',
 'seaborn-muted',
 'seaborn-white',
 'seaborn-talk',
 'grayscale',
 'dark_background',
 'seaborn-deep',
 'seaborn-bright',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

使用如下简单风格：

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

现在我们有了好看的风格，第一步就是使用标准 pandas 绘图函数绘制数据：

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



推荐使用 pandas 绘图的原因在于它是一种快速便捷地建立可视化原型的方式。

自定义图表

如果你对该图表的重要部分都很满意，那么下一步就是对它执行自定义。一些自定义（如添加标题和标签）可以使用 pandas plot 函数轻松搞定。但是，你可能会发现自己需要在某个时刻跳出来。这就是我推荐你养成以下习惯的原因：

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

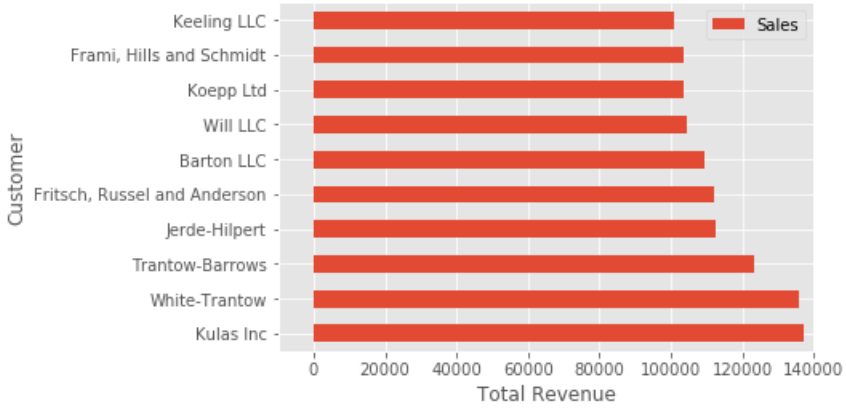
生成的图表和原始图表基本一样，不过我们向 plt.subplots() 添加了一个额外的调用，并将 ax 传输至绘图函数。因此，通过 ax 或 fig 对象可以执行任何自定义。

我们利用 pandas 实现快速绘图，现在利用 Matplotlib 获取所有功能。通过使用命名惯例，调整别人的解决方案适应自己的需求变得更加直接简单了。

假设我们想调整一些轴标签，且 ax 变量中有多个轴，可以进行一些操作：

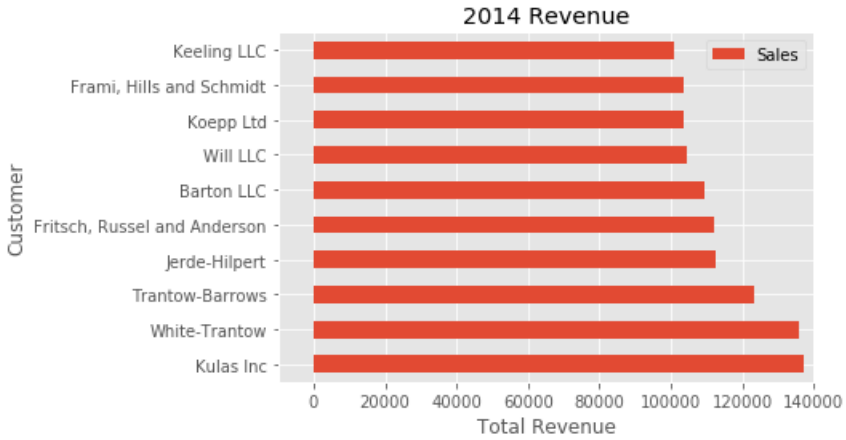
```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
```

```
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer');
```



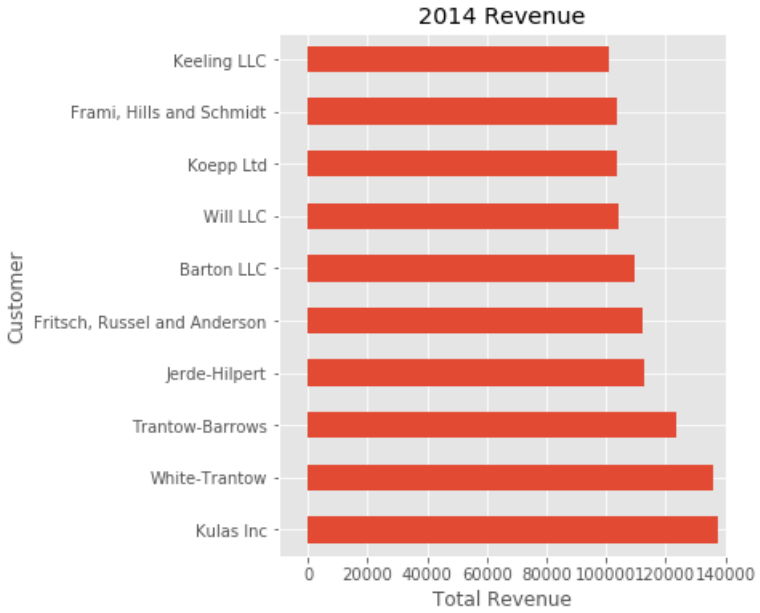
这是另一种改变标题和标签的简单方式：

```
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')
```



为了进一步展示该方法，我们还可以使用 plt.subplots() 函数可以定义图像尺寸，一般以英寸为单位。我们还可以使用 ax.legend().set_visible(False) 移除图例。

```
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)
```



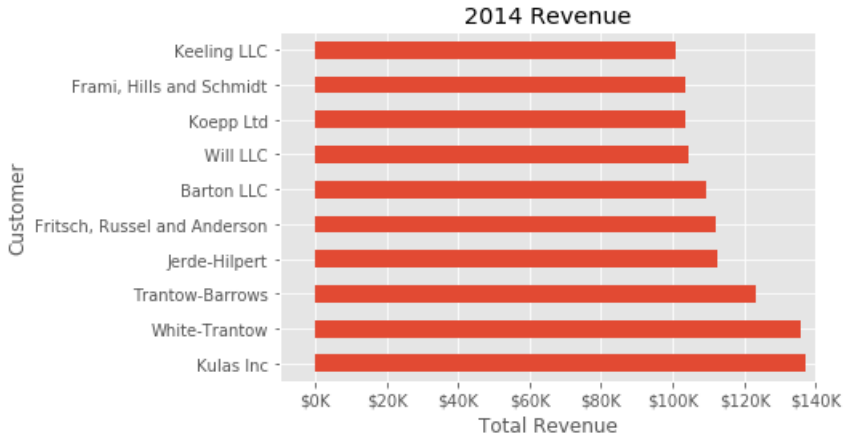
要想修改这个图像，你可能需要执行很多操作。图中最碍眼的可能是总收益额的格式。Matplotlib 可以使用 FuncFormatter 解决这一问题。该函数用途多样，允许用户定义的函数应用到值，并返回格式美观的字符串。

以下是货币格式化函数，用于处理数十万美元区间的数值：

```
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)
```

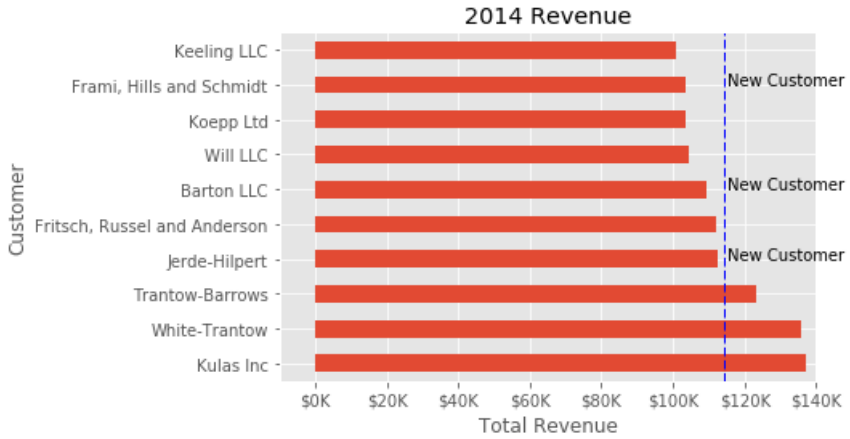
现在我们有了格式化程序函数，就需要定义它，并将其应用到 x 轴。完整代码如下：

```
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)
```



这张图美观多了，非常好地展示了自定义问题解决方案的灵活性。最后要说的自定义特征是向图表添加注释。你可以使用 ax.axvline() 画垂直线，使用 ax.text() 添加自定义文本。就以上示例，我们可以画一条表示平均值的线，包括代表 3 个新客户的标签。以下是完整代码：

```
# 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)
```



图表

目前，我们所做的所有改变都是针对单个图表。我们还能够在图像上添加多个表，使用不同的选项保存整个图像。

如果我们确定要在同一个图像上放置两个表，那么我们应该对如何做有一个基础了解。首先，创建图像，然后创建轴，再将它们绘制成图表。使用 plt.subplots() 可以完成该操作：

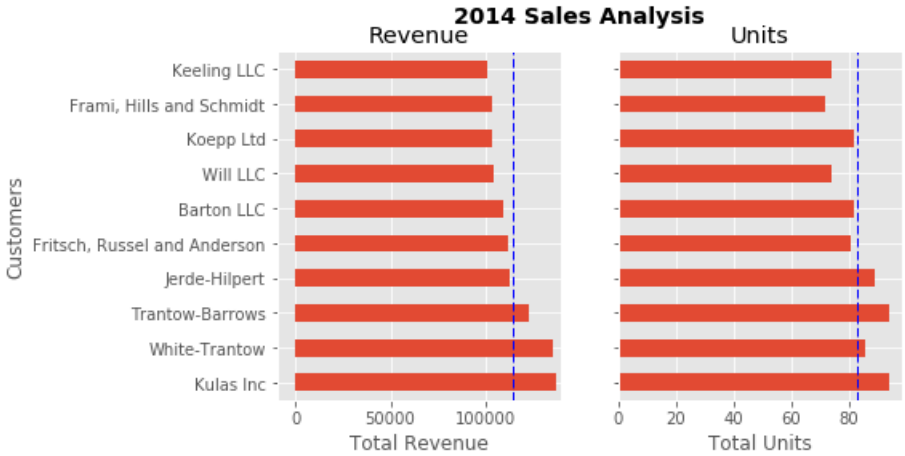
```
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(7, 4))
```

在这个例子中，我使用 nrows 和 ncols 指定大小，这对新用户来说比较清晰易懂。我还使用 sharey=True 以使 y 轴共享相同的标签。

该示例很灵活，因为不同的轴可以解压成 ax0 和 ax1。现在我们有了这些轴，就可以像上述示例中那样绘图，然后把一个图放在 ax0 上，另一个图放在 ax1。

```
# 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)
```



现在，我已经在 jupyter notebook 中用 %matplotlib inline 展示了很多图像。但是，在很多情况下你需要以特定格式保存图像，将其和其他呈现方式整合在一起。

Matplotlib 支持多种不同文件保存格式。你可以使用 fig.canvas.get_supported_filetypes() 查看系统支持的文件格式：

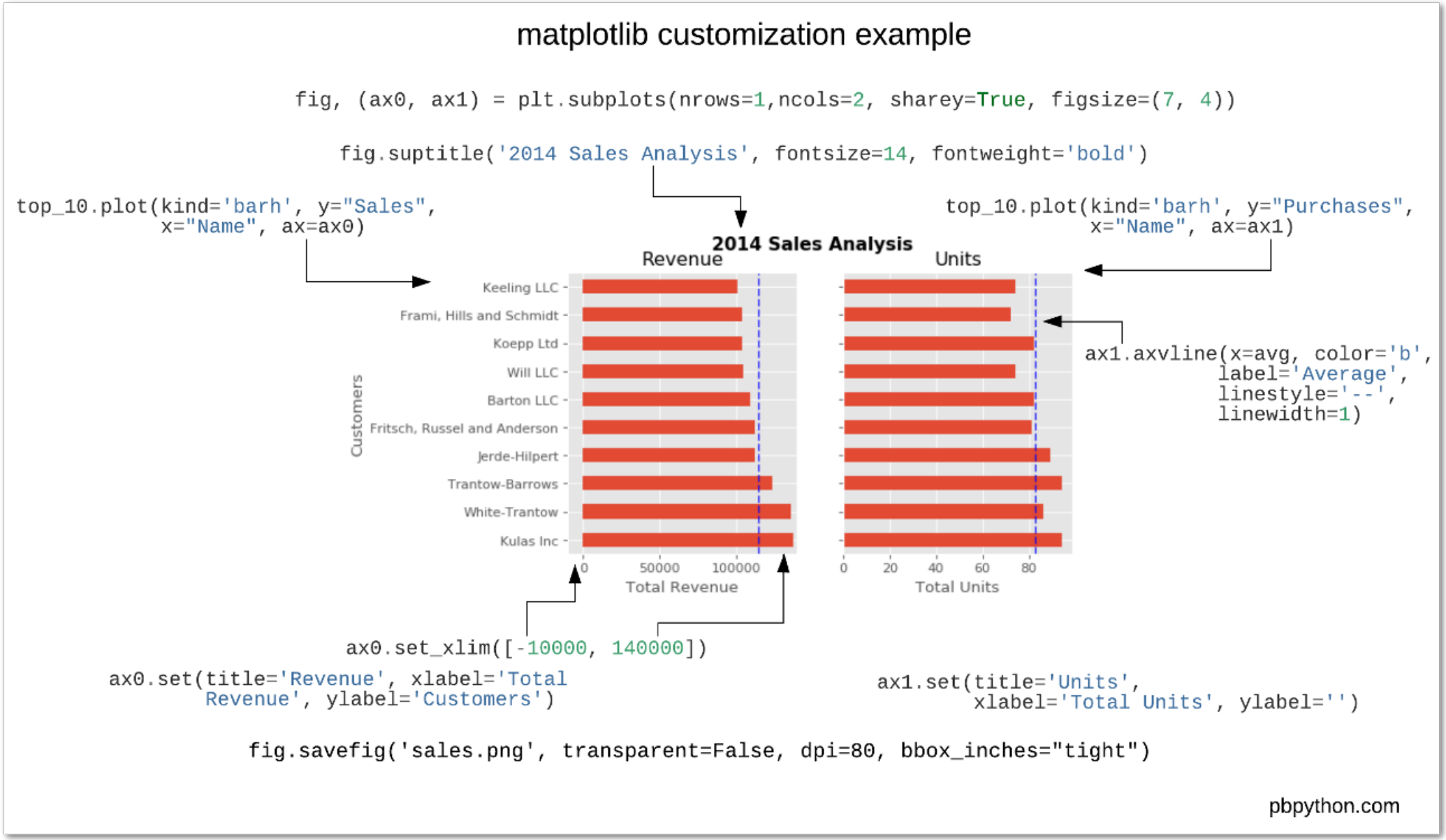
```
fig.canvas.get_supported_filetypes()
{'eps': 'Encapsulated Postscript',
 'jpeg': 'Joint Photographic Experts Group',
 'jpg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw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

我们有 fig 对象，因此我们可以将图像保存成多种格式：

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

结论

该版本将图表保存为不透明背景的 png 文件。我还指定 dpi 和 bbox_inches="tight" 以最小化多余空白。最后，希望该方法可以帮助大家理解如何更有效地使用 Matplotlib 进行日常数据分析。



工程数据可视化

提交评论

登录后参与评论去登录