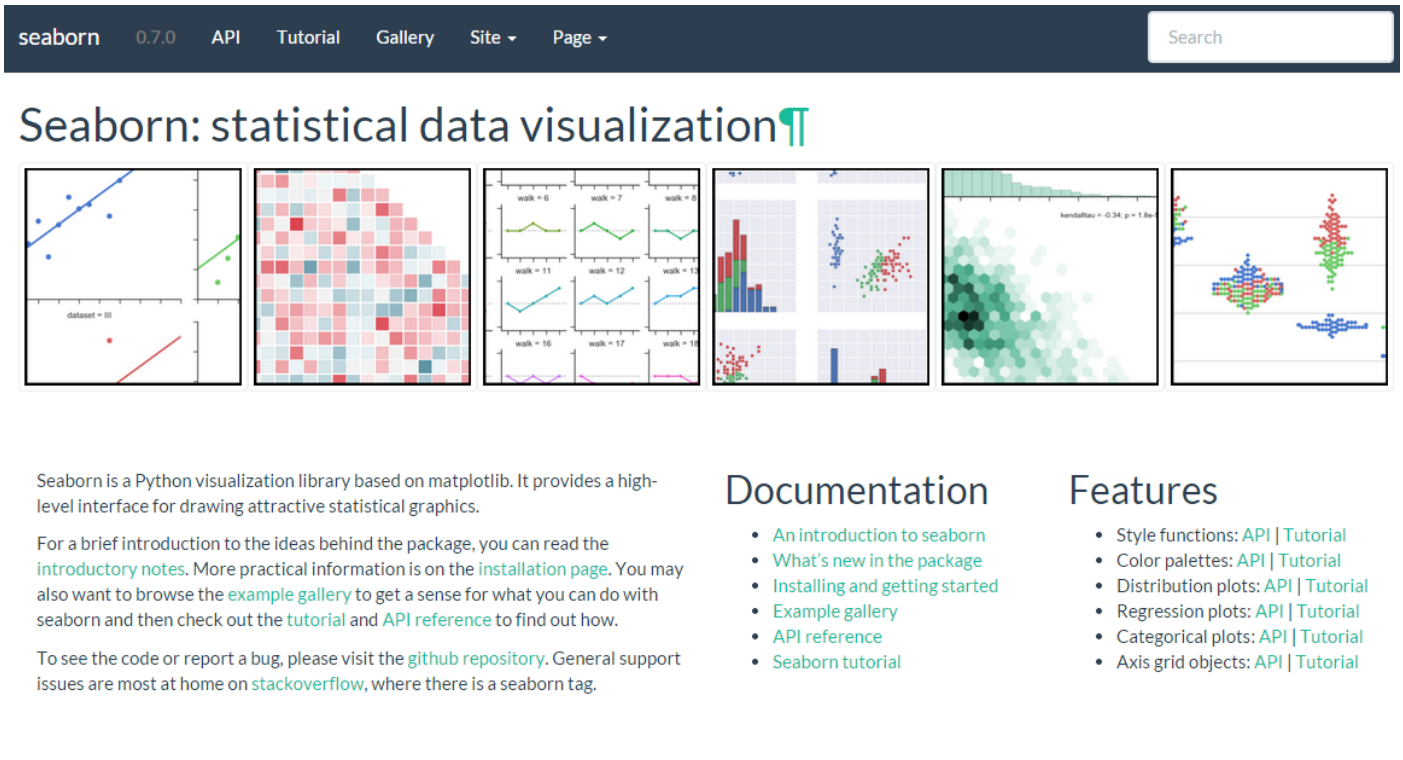
《SEABORN 统计绘图模块中文指南》

zw 开源量化团队 QQ 群：533233771

2016.05.01



摘自：《zw 量化实盘·开源课件》系列
更多量化资料,请浏览 zw 网站：http://ziwang.com
QQ 群：124134140 （zw 量化&大数据)

前言·团队的力量

元旦后才组建的 zw 量化开源组，是一只年轻的团队，是国内开源领域，不断创造奇迹的新锐团队。
从元旦筹建开始，短短几个月，团队成员扩展到 130 多人。
在团队磨合的同时，先后发布了一系列开源作品，填补了国内量化领域的多个空白：
《PYTHON 开源量化项目汇总·ZW 中文版》
《PYTHON 开源量化项目汇总·GIT 项目增强版》
《zwQuant》开源量化软件，及函数级中文注解
《zw-talib》开源金融函数包，函数级中文注解
zw-tick 分笔数据源 2010-2015，及配套开源程序，全功能：下载、分时转换、数据更新追加、去重
.....

尽管如此，可是，这次《seaborn 绘图模块中文指南》，还是再一次，给笔者带来了震撼。
seaborn 文档的翻译期间，正好与 zwQuant 的开发时间冲突，作为 zw 量化开源组的发起人，本人只是提出了一个设想，一个题目。
期间，没有给开源组更多的支持，包括：资料搜集、文档翻译、汇总、校稿、.....
可是，最终，交到笔者手中的作品，是历来最完美的一次。
漂亮的排版、精美的插图、流畅而专业的译文。
这一切，让笔者再一次，感受到，zw 量化开源组，这些年轻人的活力与未来。
也许，这就是，团队的力量。

让我们，再次感谢 zw 量化开源组各位成员的无私奉献。

这次参加《seaborn 绘图模块中文指南》的开源组成员有：

seaborn 翻译组名单：
iris,木东，周涛，木子文戎，O_Chaser，VAN，木荣，7777777
梵森多格，VAN，yonglesunny，我怎能不戏猫，兔牙，点点。

zw 开源量化团队 QQ 群：533233771

Seaborn 介绍

Seaborn is a library for making attractive and informative statistical graphics in Python. It is built on top of matplotlib and tightly integrated with the PyData stack, including support for numpy and pandas data structures and statistical routines from scipy and statsmodels.

Seaborn 是一个制作富于吸引力和信息丰富的统计绘图 python 库。他基于 matplotlib 上层并与 PyData 栈紧密结合，包括支持 numpy 和 pandas 数据结构和 scipy 和 statsmodels 统计事务。

Some of the features that seaborn offers are

- Several built-in themes that improve on the default matplotlib aesthetics
- Tools for choosing color palettes to make beautiful plots that reveal patterns in your data
- Functions for visualizing univariate and bivariate distributions or for comparing them between subsets of data
- Tools that fit and visualize linear regression models for different kinds of independent and dependent variables
- Functions that visualize matrices of data and use clustering algorithms to discover structure in those matrices
- A function to plot statistical timeseries data with flexible estimation and representation of uncertainty around the estimate
- High-level abstractions for structuring grids of plots that let you easily build complex visualizations

Seaborn 提供的特性有：

- 一些内置主题可促进默认的 matplotlib 美学。
- 选择调色板工具能制作漂亮的图并揭示数据的模式。
- 可视化单变量和双变量分布的功能，并可以在数据子集中比较它们。
- 为不同类型的自变量和因变量拟合和可视化线性回归模型的功能。
- 可视化数据矩阵功能，并且用聚类算法去发现那些矩阵的结构。
- 利用灵活的估计和估计不确定性表达式来绘制统计时间序列数据。
- 高层次抽象的结构化绘图网络，让你轻松构造复杂的可视化。

Seaborn aims to make visualization a central part of exploring and understanding data. The plotting functions operate on dataframes and arrays containing a whole dataset and internally perform the necessary aggregation and statistical model-fitting to produce informative plots. If matplotlib “tries to make easy things easy and hard things possible”, seaborn tries to make a well-defined set of hard things easy too.

Seaborn 旨在探索和理解数据的可视化成为核心部分。绘图功能用于数据框和数组包含的整个数据集，在内部执行需要的聚合和统计模型拟合来产生信息图。如果 matplotlib “试图让简单的事情简单，困难的事情也简单”，Seaborn 也试图使困难的事情易于成为意义明确的集合。

The plotting functions try to do something useful when called with a minimal set of arguments, and they expose a number of customizable options through additional parameters. Some of the functions plot directly into a matplotlib

axes object, while others operate on an entire figure and produce plots with several panels. In the latter case, the plot is drawn using a Grid object that links the structure of the figure to the structure of the dataset in an abstract way.

该绘图功能尝试用最少的参数来做一些有用的事情，他们通过附加参数提供了一些可定制的选项。一些绘图功能可直接进入 matplotlib 轴对象，而其他功能需要在整体图形操作和面板图来生成图。在后一种情况下，绘图可以通过网格对象来画图，将图的结构与抽象的方法中的数据集的结构联系起来。

Because seaborn uses matplotlib, the graphics can be further tweaked using matplotlib tools and rendered with any of the matplotlib backends to generate publication-quality figures. Seaborn can also be used to target web-based graphics through the mpld3 and Bokeh libraries.

因为 Seaborn 使用 Matplotlib，图形可以进一步调整使用 matplotlib 工具并为 matplotlib 后端提供服务来制作图书出版质量级别的图像。Seaborn 也可以通过 mpld3 和 Bokeh 库制作基于 Web 的图形的图像。

Seaborn should be thought of as a complement to matplotlib, not a replacement for it. When using seaborn, it is likely that you will often invoke matplotlib functions directly to draw simpler plots already available through the `pyplot` namespace. Further, while the seaborn functions aim to make plots that are reasonably "production ready" (including extracting semantic information from Pandas objects to add informative labels), full customization of the figures will require a sophisticated understanding of matplotlib objects.

Seaborn 应视作为对 matplotlib 的补充，而不是替代它。当我们使用 Seaborn 时，你可能会经常援引 matplotlib 功能通过现成的 pyplot 直接画简单的图像。但进一步的，Seaborn 功能旨在使绘图能适度"准备就绪"（包括从 pandas 对象提取语义信息来添加信息标签），全面定制图像需要对 matplotlib 对象有充足的理解。

For more detailed information and copious examples of the syntax and resulting plots, you can check out the example gallery, tutorial or API reference.

更详细的信息和丰富的关于语法和结果图的例子,你可以查看例子画廊 ，教程或 API 参考 。

v0.7 版包新特性

A catalog of new features, improvements, and bug-fixes in each release.

该包有什么新特性

以下是，各自发布的新特性、改进和修复 bug 的名目。

v0.7.0 (January 2016)¶

2016 年一月 v0.7.0

This is a major release from 0.6. The main new feature is `swarmplot()` which implements the beeswarm approach for drawing categorical scatterplots. There are also some performance improvements, bug fixes, and updates for compatibility with new versions of dependencies.

该发布主要来自 o.6 版本，主要的新特性在于 swarmplot()可以实现通过 beeswarm 方法来画出分类散点图。同样也有一些性能上改进，bug 修复，可更新对新版本依赖库的兼容性。

- Added the `swarmplot()` function, which draws beeswarm plots. These are categorical scatterplots, similar to those produced by `stripplot()`, but position of the points on the categorical axis is chosen to avoid overlapping points. See the categorical plot tutorial for more information.
- 增加 swarmplot()功能可绘制蜜蜂群图，它们是分类散点图，类似于 stripplot 产生的图，不过落在分类轴上的点将避免重叠。看一下 categorical plot tutorialg 获得更多信息。
- Added an additional rule when determining category order in categorical plots. Now, when numeric variables are used in a categorical role, the default behavior is to sort the unique levels of the variable (i.e they will be in proper numerical order). This can still be overridden by the appropriate `{*_}order` parameter, and variables with a `category` datatype will still follow the category order even if the levels are strictly numerical.
  - 在分类图中确定类别顺序时，增加了一个附加规则。现在，当数字变量被用于一个明确的分类角色时，默认行为是排序的变量的唯一的水平（即他们将按适当的数字排序）。适当的{ *_ }order 参数更为重要，如果水平严格是数值，那么类型变量仍然按照类别排序。
- Changed some of the `stripplot()` defaults to be closer to `swarmplot()` points are somewhat smaller, have no outlines, and are not split by default when using `hue`.
- 改变 stripplot()一些默认设置，可以是接近于 swarmplot()点更加小，没有轮廓，当使用色调时不会被默认分隔。
- Changed how `stripplot()` draws points when using `hue` nesting with `split=False` so that the different `hue` levels are not drawn strictly on top of each other.
- 改变 stripplot()画点方式，当使用色调巢 split=False 时，不同的色调层不能严格的画在其它色调层上面。
- Improve performance for large dendrograms in `clustermap()`.
- 改进 clustermap()大型系统树图的性能。
- Added `font.size` to the plotting context definition so that the default output from `plt.text` will be scaled appropriately.
- Fixed a bug in `clustermap()` when `fastcluster` is not installed.
- 修复 clustermap()当快速聚类没有安装上的时候的 bug。
- Fixed a bug in the zscore calculation in `clustermap()`.
  修复 clustermap()z 分循环的 bug。
- Fixed a bug in `distplot()` where sometimes the default number of bins would not be an integer.
- 修复 distplot()当有时候默认宽带不是一个整数时候的 bug。
- Fixed a bug in `stripplot()` where a legend item would not appear for a `hue` level if there were no observations in the first group of points.
- 修复 stripplot()当 legend 项不出现在色调层如果第一组点里没有观测。
- Heatmap colorbars are now rasterized for better performance in vector plots.
- 热力图色彩柱在向量图栅格化性能更好。
- Added workarounds for some matplotlib boxplot issues, such as strange colors of outlier points.
- 增加一些 matplotlib 盒子图问题的解决方案，诸如离群点上的古怪颜色。
- Added workarounds for an issue where violinplot edges would be missing or have random colors.
- 增加当小提琴图问题的解决方法，边可能会消失或产生随机颜色。
- Added a workaround for an issue where only one `heatmap()` cell would be annotated on some matplotlib backends.
- 增加当只有一个 heatmap()单元格时，会在 matplotlib 后端进行标注的解决方案。

- Fixed a bug on newer versions of matplotlib where a colormap would be erroneously applied to scatterplots with only three observations.
- 修复了一个 bug，关于在 matplotlib 新版本上，只有 3 个观测的时候 colormpa 会不正确的生成散点图。
- Updated seaborn for compatibility with matplotlib 1.5.
- 更新对 matplotlib1.5 兼容性。
- Added compatibility for various IPython (and Jupyter) versions in functions that use widgets.
- 增加对不同版本 Ipython（还用 Jupyter）版本使用组件的兼容性。

Seaborn 教程

纲要

风格管理

- 控制图像美学
  - ✧ 使用 axes_style() and set_style()样式图形
  - ✧ 用 despine()删除 spines
  - ✧ 临时设定图形风格
  - ✧ 重载 Seaborn 样式元素
  - ✧ 用 plotting_context()和 set_context()缩放图形元素
- 选择调色板
  - ✧ 利用 color_palette()构建调色板
  - ✧ 定性调色板
  - ✧ 连续调色板
  - ✧ 极端型调色板
  - ✧ 用 set_palette()改变默认调色板

绘图函数

- 可视化数据集的分布
  - ✧ 绘制单变量分布
  - ✧ 绘制双变量分布
  - ✧ 可视化数据中的成对关系

- 可视化线性关系
  - ✧ 绘制线性回归模型的函数
  - ✧ 拟合各种不同模型
  - ✧ 条件变量
  - ✧ 控制图的大小和形状
  - ✧ 在其它环境下绘制回归

- **对分类数据进行绘图**
  - ✧ 分类散点图
  - ✧ 分类观测的分布
  - ✧ 分类统计估计
  - ✧ 绘制宽型数据
  - ✧ 绘制多面板分类图

结构化网格

- 数据感知网格的绘制
  - ✧ 用 FaceGrid 构造子集
  - ✧ 用 PairGrid 和 pairplot()绘制成对关系

控制图像美学

绘制富于吸引力的图表非常重要，当你把研究的数据集制作成图表，观看会感到舒适。

可视化是沟通量化洞见的重要途径，在这种情况下更需要引人注意的图表来吸引观众。

Matplotlib 是高度可定制的，但它很难知道用什么设置来绘制出吸引人的图表，Seaborn 附带一些自定义的主题和控制 matplotlib 图表外观的高级接口。

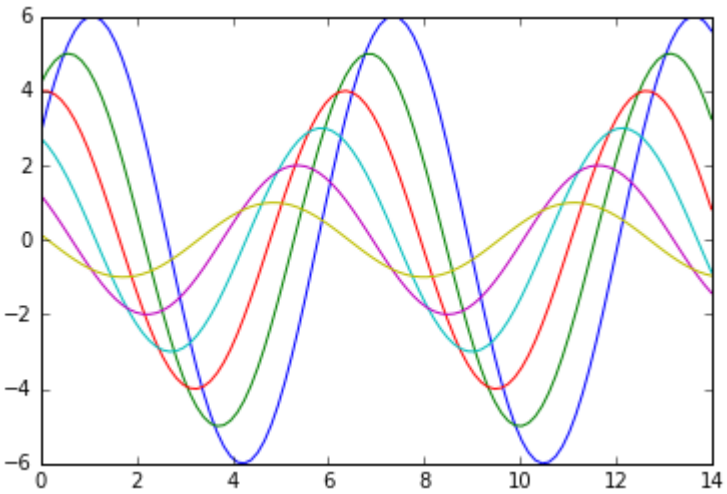
```
%matplotlib inline
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
np.random.seed(sum(map(ord, "aesthetics")))
```

让我们定义一个简单的函数来绘制一些偏移正弦波，这将有助于我们看到我们可以调整不同风格的参数。

```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
```

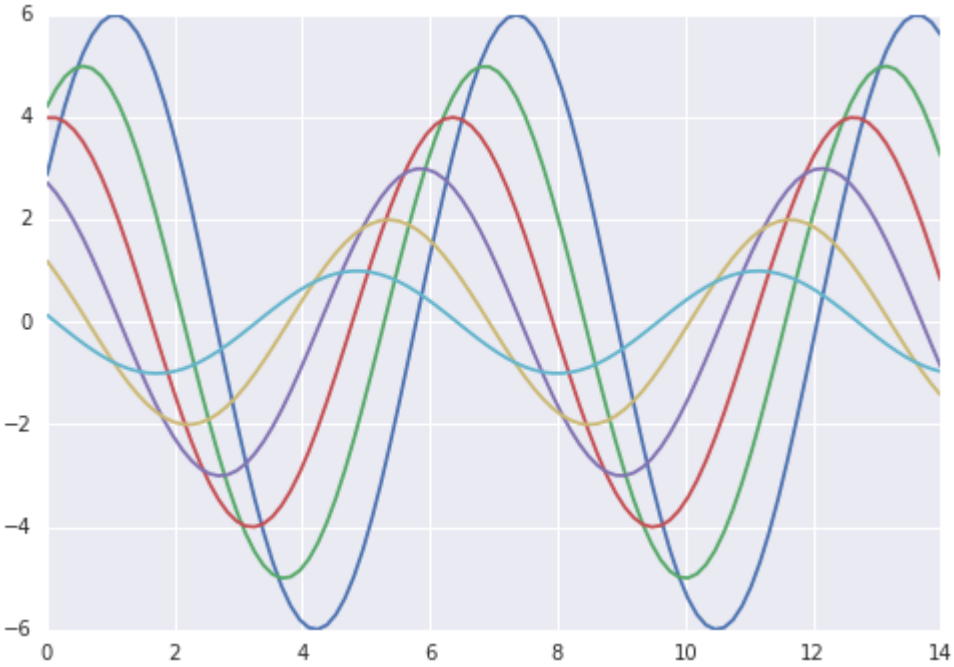
这是 matplotlib 默认的绘图样式。

sinplot()



换成 seaborn 默认项，只需要简单的导入库。

```
import seaborn as sns
sinplot()
```



Seaborn 默认终止 MATLAB 加载 matplotlib 中的 aesthetic，从而阻止把柔和的颜色覆盖在浅灰色背景的白色网格上。我们发现，几乎所有情况下，图像比表格更好。白字灰底在绘图使用中会默认强迫禁止。这种网格在从多分面把结构图像化是非常有帮助的，而这也是库中一些复杂工具的要点所在。

Seaborn 将 matplotlib 参数分割成两个单独的组。第一组设置 aesthetic 风格，第二组缩放图中各种不同元素，以便它可以容易适应不同情境。

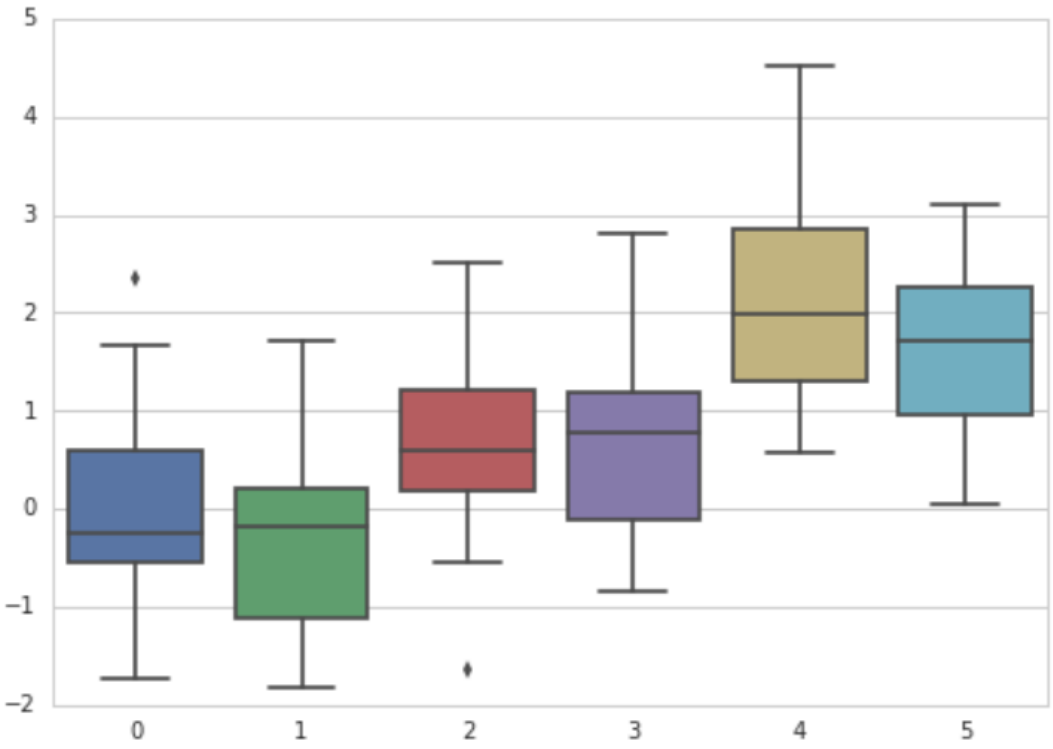
该接口为了操作这些参数提供两对函数，控制样式的 axes_style() 和 set_style() 函数，图表缩放的 plotting_context() and set_context()，在这两种情况下，第一个函数返回参数字典，第二个设置 matplotlib 的默认值。

使用 axes_style() and set_style()样式图形

seaborn 现在有五个主题: darkgrid, whitegrid, dark, white, 和 ticks。

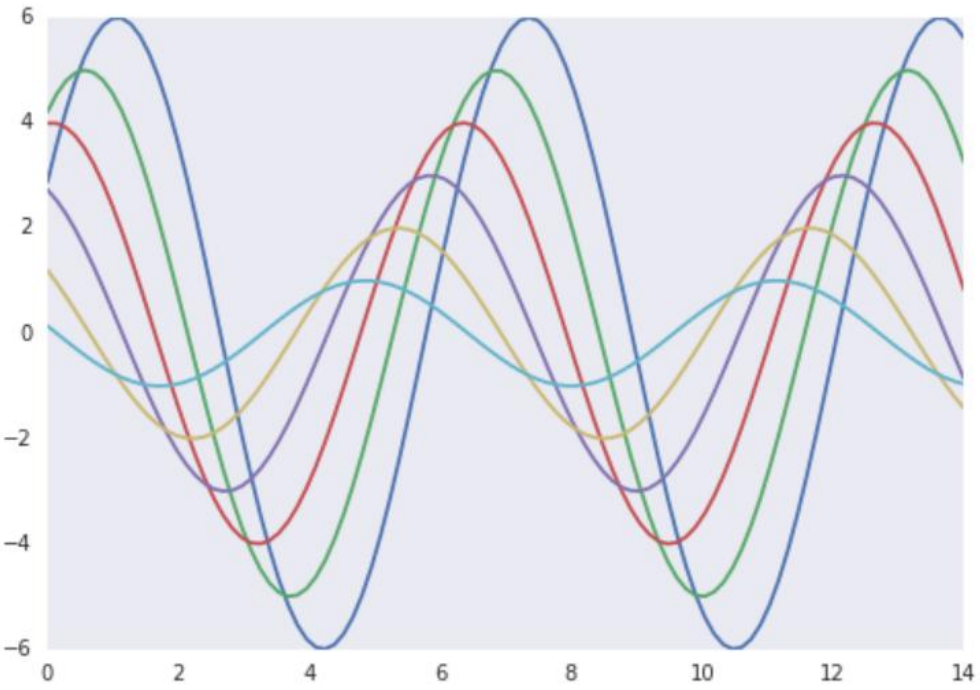
它们分别适用于不同的应用和个人喜好。默认主题是 darkgrid。如上所述，网格当作量化信息绘图的查找表是有帮助的，白灰色有助于避免网格与代表数据的线条之间产生冲突，whitegrid 主题类似，但它更适合于大量数据元素的绘制。

```
sns.set_style("whitegrid")
data = np.random.normal(size=(20, 6)) + np.arange(6) / 2
sns.boxplot(data=data);
```

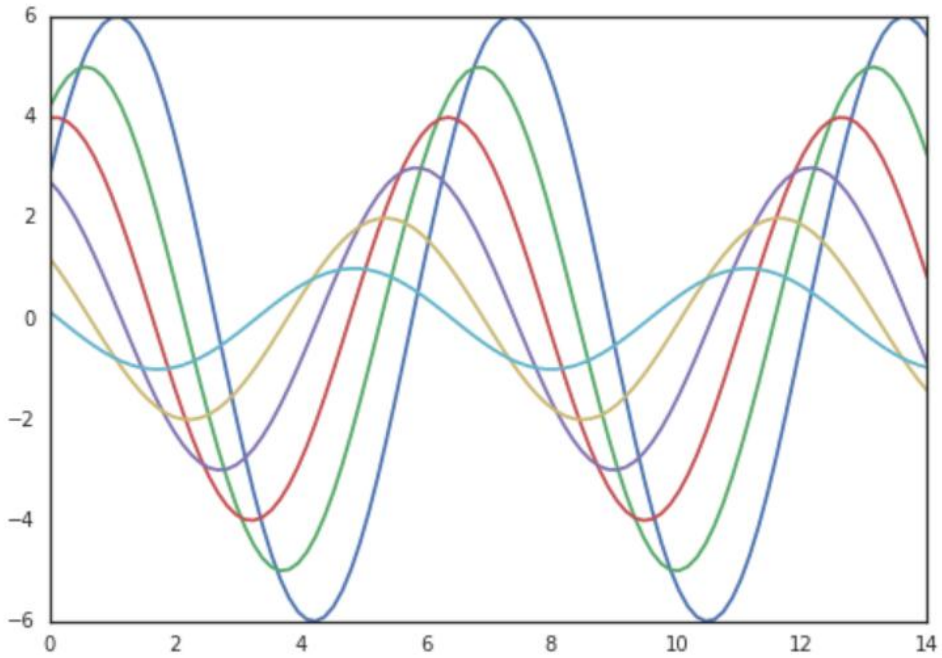


对于许多图形来说，（特别对于像会谈，在这里主要是想用图形来加深数据的印象），网格是不必要的。

```
sns.set_style("dark")
sinplot()
```

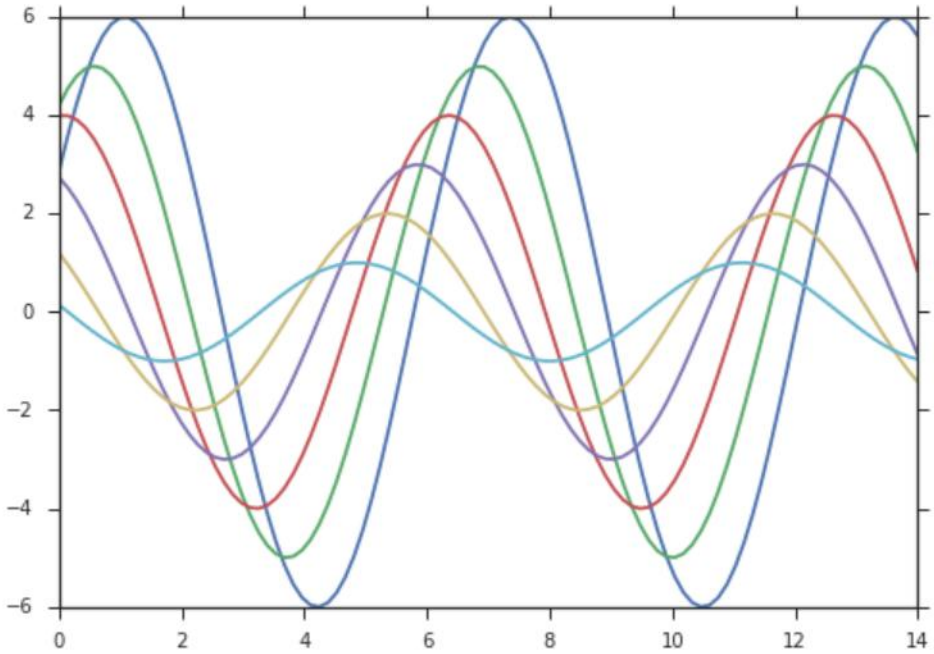



```
sns.set_style("white")
sinplot()
```



有时你可能想图形上增加一些结构，使用刻度很方便：

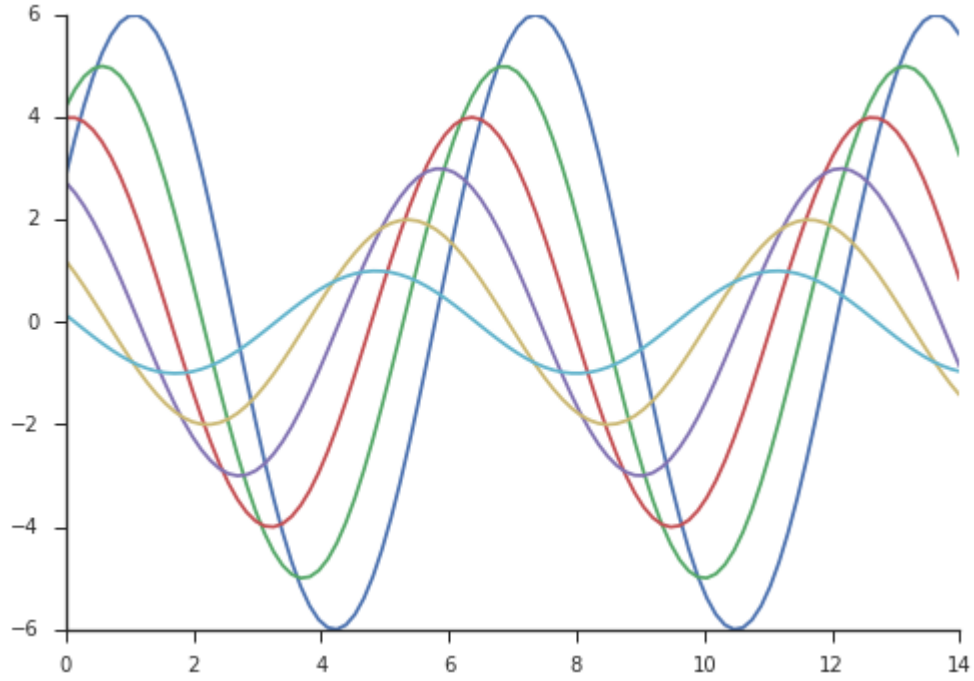
```
sns.set_style("ticks")
sinplot()
```



用 despine()删除 spines(脊线，这里应该指坐标轴的右侧和上侧的边框)

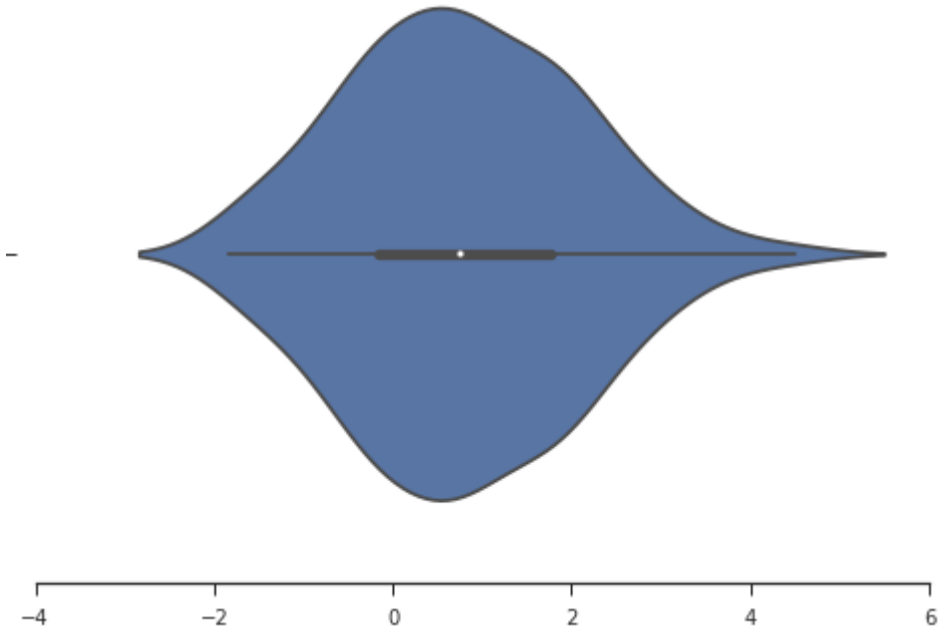
`white` 和 `ticks` 风格的类型都可以受益于移除顶部和右轴的脊线，实际上这是不必要的。 它不可能通过 matplotlib 参数来实现,但是你可以调用 seaborn 函数 `despine()`来删除他们:

```
sinplot()
sns.despine()
```



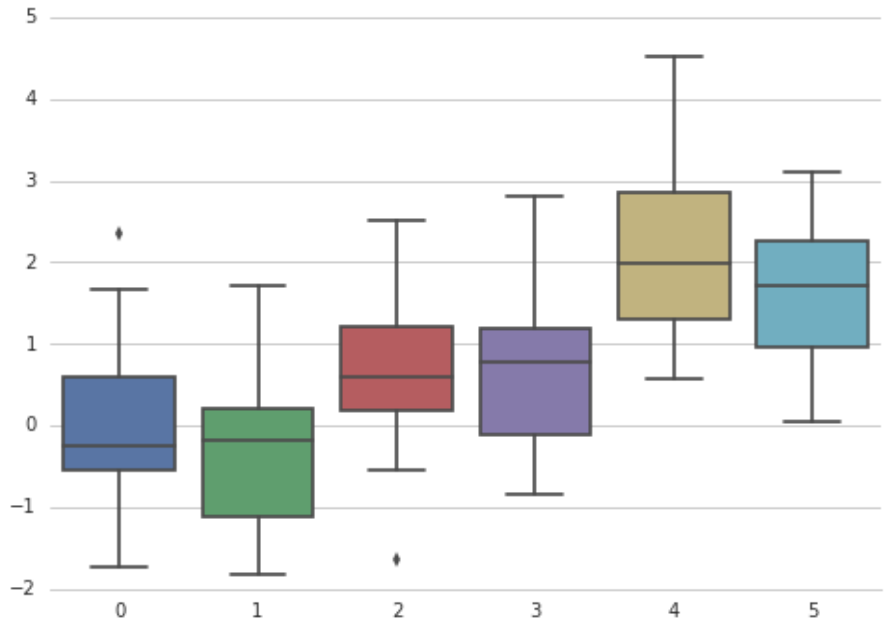
一些图像受益于从数据偏移脊线,它们也可以调用 **despine()**来完成。当刻度不能覆盖整个范围的轴，`trim` 参数会限制现有的脊线的范围。

```
f, ax = plt.subplots()
sns.violinplot(data)
sns.despine(offset=10, trim=True);
```



通过向 **despine()**添加参数，你还可以控制哪些 spines 可以被移除。

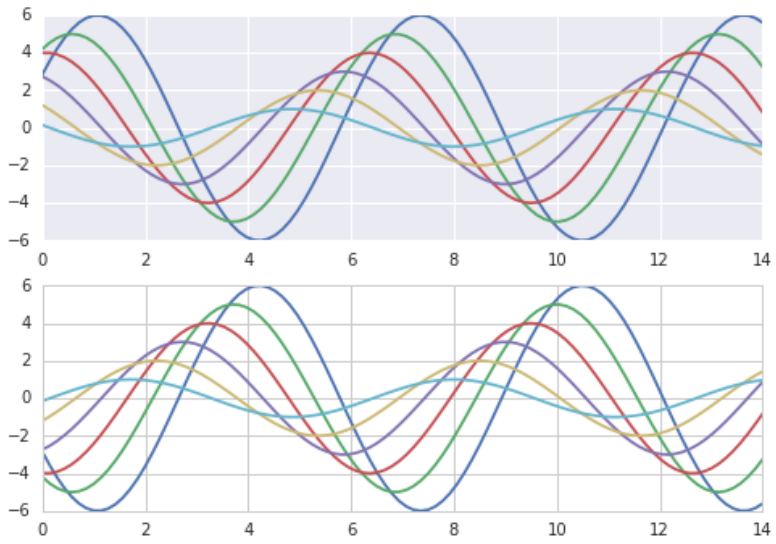
```
    sns.set_style("whitegrid")
    sns.boxplot(data=data, palette="deep")
    sns.despine(left=True)
```



临时设定图形风格

虽然很容易来回切换,你仍然可以在一个 `with` 语句中使用 `axes_style()`函数临时设置绘图参数。这也允许你用不同风格的轴来绘制图像:

```
    with sns.axes_style("darkgrid"):
        plt.subplot(211)
        sinplot()
    plt.subplot(212)
    sinplot(-1)
```



重载 Seaborn 样式元素

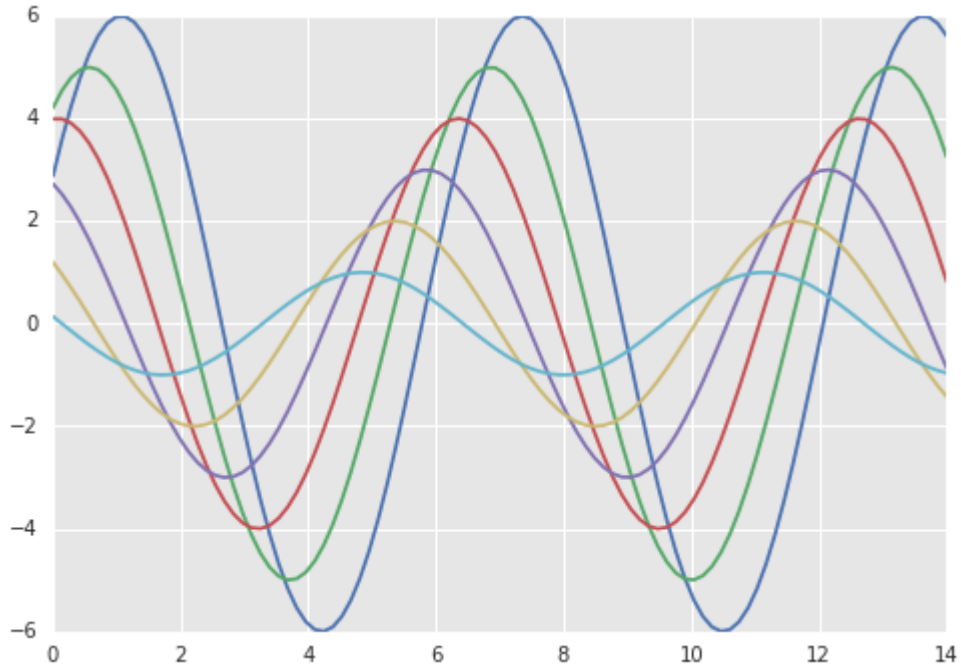
如果你想定制 seaborn 样式，你可以传递 `axes_style()` 和 `set_style()`的 `rc` 参数字典。注意你只能通过这种方式重载部分样式定义的参数。（但是，高级别的 `set()`函数可以使用 matplotlib 中的任何参数字典）。

如果你想查看有哪些参数，可以不需要输入直接调用，它将会返回当前的设置。

```
sns.axes_style()
{'axes.axisbelow': True,
 'axes.edgecolor': '.8',
 'axes.facecolor': 'white',
 'axes.grid': True,
 'axes.labelcolor': '.15',
 'axes.linewidth': 1.0,
 'figure.facecolor': 'white',
 'font.family': [u'sans-serif'],
 'font.sans-serif': [u'Arial',
  u'Liberation Sans',
  u'Bitstream Vera Sans',
  u'sans-serif'],
 'grid.color': '.8',
 'grid.linestyle': u'-',
 'image.cmap': u'Greys',
 'legend.frameon': False,
 'legend.numpoints': 1,
 'legend.scatterpoints': 1,
 'lines.solid_capstyle': u'round',
 'text.color': '.15',
 'xtick.color': '.15',
 'xtick.direction': u'out',
 'xtick.major.size': 0.0,
 'xtick.minor.size': 0.0,
 'ytick.color': '.15',
 'ytick.direction': u'out',
 'ytick.major.size': 0.0,
 'ytick.minor.size': 0.0}
```

接下来可以设置不同版本的参数。

```
sns.set_style("darkgrid", {"axes.facecolor": ".9"})
sinplot()
```



用 plotting_context()和 set_context()缩放图形元素

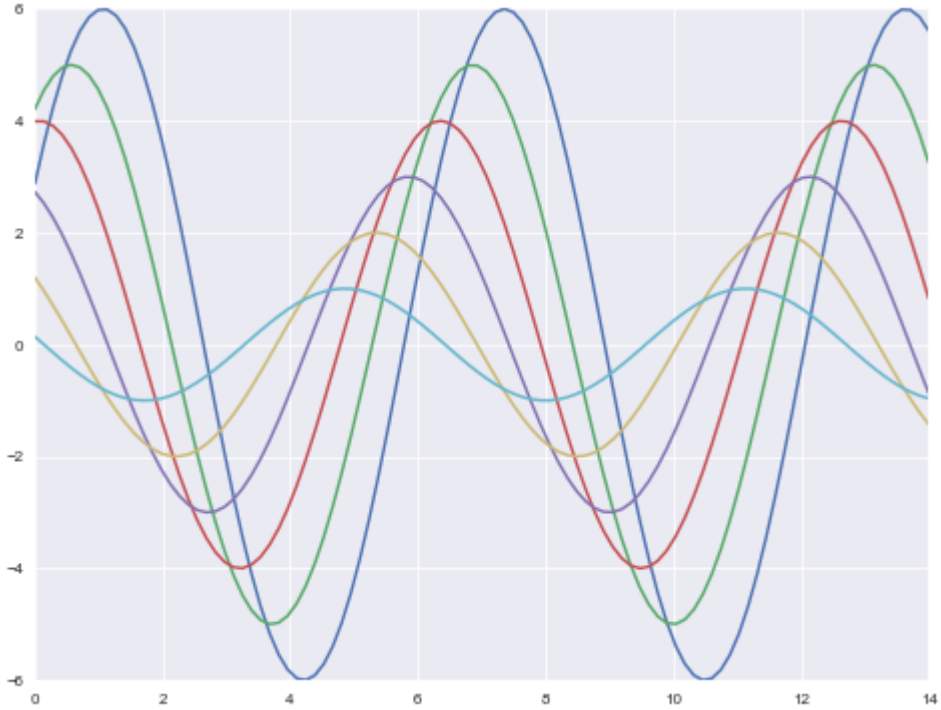
各个参数集用于控制绘图的缩放，在设置中使用同样的绘图代码对于大型的图还是小型的图都是适合的。

首先要通过调用 `set()`重置默认的参数：

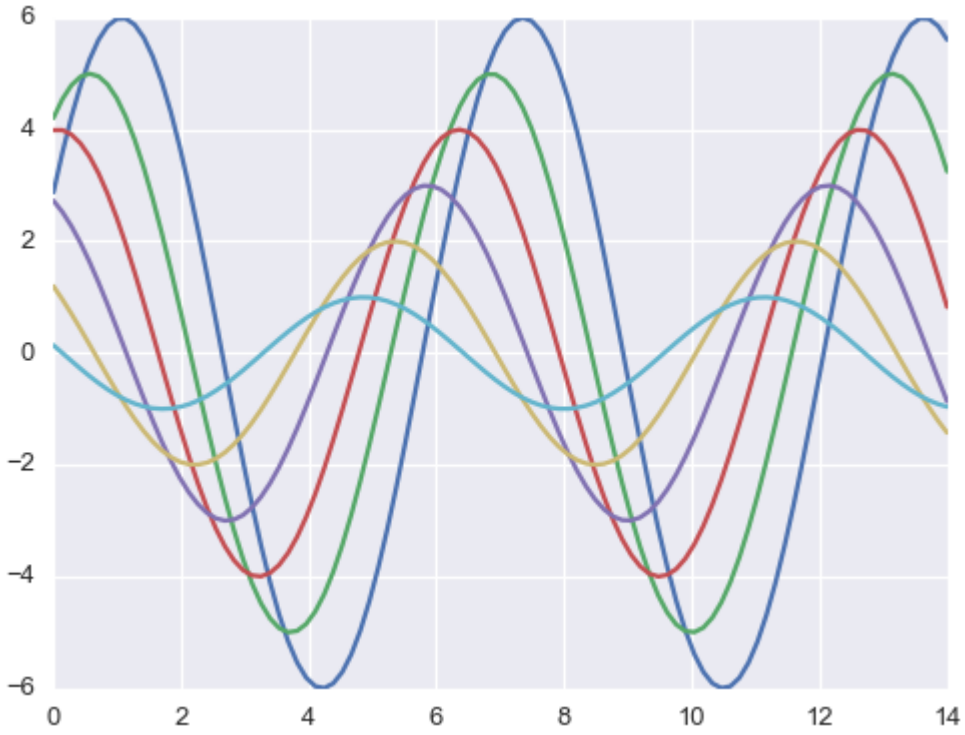
```
sns.set()
```

其中预设的四个环境，按照大小的排序分别是 `paper`, `notebook`, `talk`, 和 `poster`。默认的是 `notebook` 样式，并且上面的画图中也用到了。

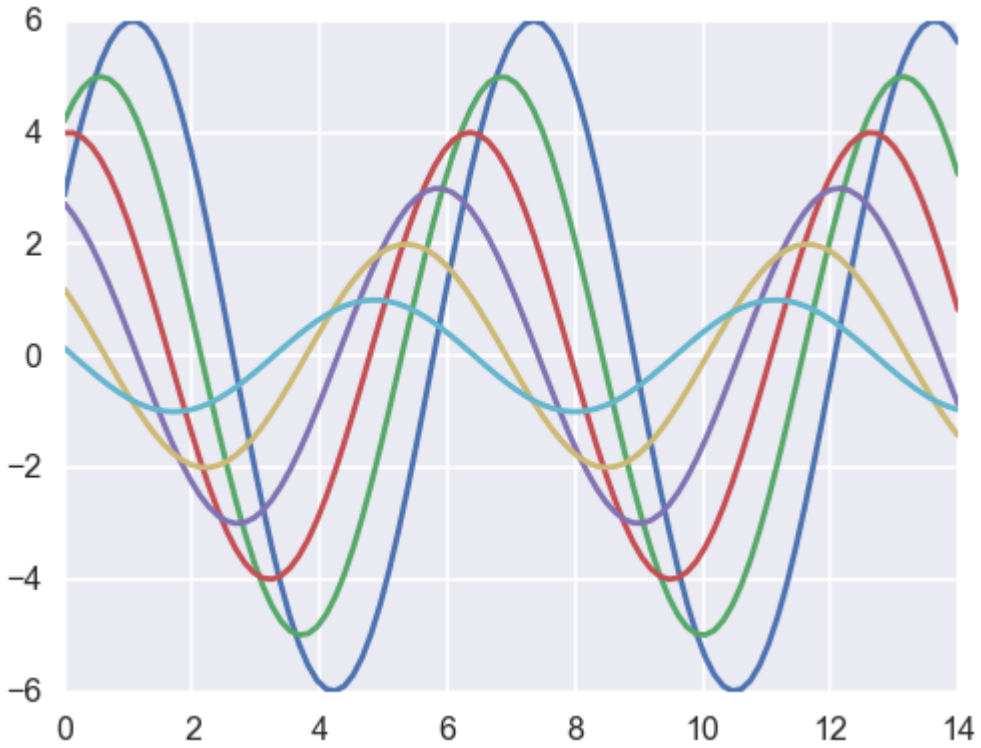
```
sns.set_context("paper")
plt.figure(figsize=(8, 6))
sinplot()
```




```
sns.set_context("talk")
plt.figure(figsize=(8, 6))
sinplot()
```




```
sns.set_context("poster")
plt.figure(figsize=(8, 6))
sinplot()
```

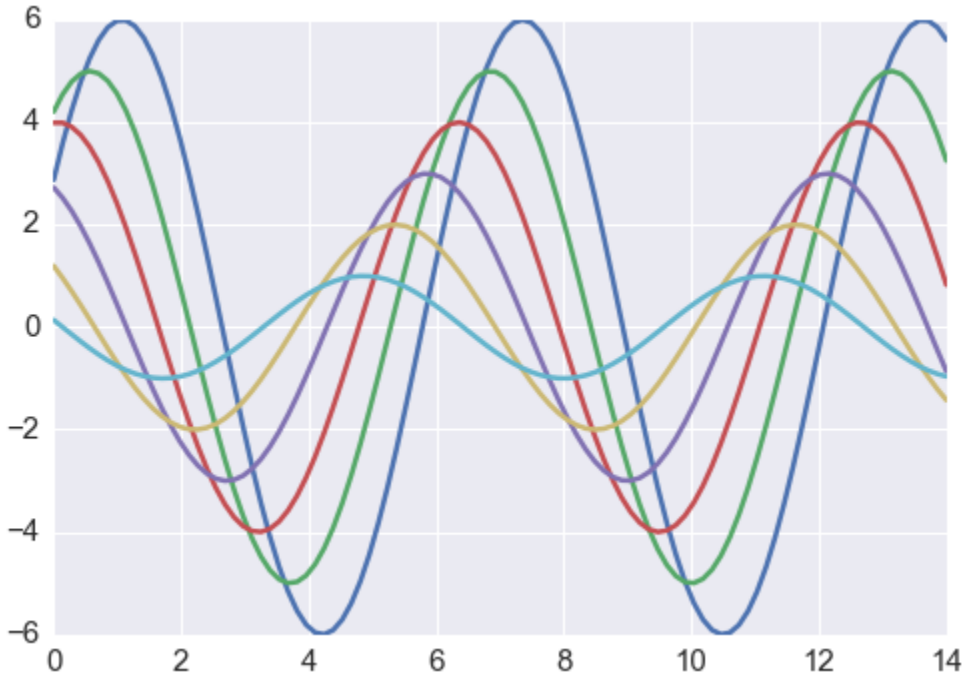


你所知道的大多数样式函数应该转化为环境函数。

你可以设置这些参数的一个名字来调用 **set_context()**，而且通过提供参数字典来重载参数。

当改变环境的时候，你还可以独立缩放字体元素的大小。（该选项也能通过最高级的 **set()** 函数设定）。

```
sns.set_context("notebook", font_scale=1.5, rc={"lines.linewidth": 2.5})
sinplot()
```



类似的（尽管可能用处不大），你可以暂时使用 `with` 声明来控制图形的缩放。

样式和环境都可以通过 set()函数快速配置。该函数同样也默认设置了调色板，但那是教程手册下一节中所要讲的内容。

选择调色板

颜色比图形样式的其他方面更为重要，这是因为如果通过有效运用颜色可以揭示数据中的模式，而如果运用不当则会掩盖数据中的模式。现存有许多很好的资源来学习如何在可视化中运用颜色的好技巧，而我偏向于 Rob Simmon 的 series of blog posts 以及 more technical paper。Matplotlib 文档现在也有了一个 nice tutorial，用以展示内置颜色表的一些知觉特性。

在可视化方面，Seaborn 使得在选取与运用调色板变得容易，适用于你正在处理的数据及你制定的目标。

```
%matplotlib inline
```

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

sns.set(rc={"figure.figsize": (6, 6)})
np.random.seed(sum(map(ord, "palettes")))
```

利用 color_palette()构建调色板

在使用离散的调色板时，其中最重要的函数是 color_palette()。该函数给许多（但不是所有）可能在 seaborn 中生成的颜色提供接口，它被含有 palette 参数的函数内部使用（当需要多种颜色的时候，只需改变 color 参数）。

color_palette() 将识别所有 seaborn 调色板或者 matplotib 颜色表的名称（除了 jet，任何时候你都不会用到）。color_palette()也可以识别任意有效的 matplotib 格式（RGB 元组，十六进制颜色码，或者 HTML color names）中指定的颜色列表。它的返回值总是一个 RGB 元组的列表。

最后，无参数调用 color_palette()会返回目前默认的颜色。

相应的函数，set_palette()，使用同样的参数并可以设置所有图的默认色相环。你也可以使用带有 with 声明的 color_palette()函数来暂时改变默认的调色板（见下文）。

在不知道数据特征的情况下，想知道是哪一种调色板或者颜色表最适合一组数据集，这通常是不可能的。我们通过使用三种配色方式：离散型，渐变型和极端型来使用 color_palette()和其它 seaborn 调色板功能。

定性调色板

当你想区分没有内在顺序的离散数据块时，定性的（或称分类的）调色板是最合适的。

导入 seaborn 之后，matplotlib 的默认色相环会变成一组 6 种颜色，目的为了使图像看起来更舒适。

```
current_palette = sns.color_palette()
sns.palplot(current_palette)
```

默认有 6 种不同的主题，称为：`deep`, `muted`, `pastel`, `bright`, `dark`, `colorblind`

使用循环颜色系统

当你要区分超过 6 种类别，最简单的方法就是从循环颜色系统中等间隔地选色绘图(这样在改变色相的同时又保证亮度和饱和度的一致性)。这是绝大部分 seaborn 函数在他们需要超过目前设置更多颜色的时候，默认使用的方法。

最常见的方法是使用 hls 色域，这是 RGB 值的简单变换。

```
sns.palplot(sns.color_palette("hls", 8))
```

`hls_palette()`函数的功能是调整颜色的亮度和饱和度。

```
sns.palplot(sns.hls_palette(8, l=.3, s=.8))
```

然而，由于人类视觉系统的工作方式，从 RGB 色阶来说，相同强度的颜色看起来并不必须同样强烈。我们发觉到黄色和绿色相对较亮，而蓝色相对较暗，这成为 hls 系统追求一致性的问题。

为了解决这个问题，seaborn 提供了一个和 husl 系统的接口，这个接口使得在选择均匀变化色相的同时保持亮度和饱和度的一致性。

```
sns.palplot(sns.color_palette("husl", 8))
```

还有一个类似的函数 `husl_palette()`，它能够提供 husl 系统的更灵活的接口。

使用分类的 Color Brewer 调色板

另一个好用的分类调色板来源于 Color Brewer 工具(该软件也有连续和极端型调色板，我们后面会看到)。这些在 matplotlib colormaps 里也有，但却不能很好地操作。在 seaborn，当你想要一个定性的 Color Brewer 调色板，你会得到一组离散的颜色，但这说明颜色组会从某一特定颜色开始循环。

Color Brewer 网站有一个很赞的功能，它提供了一些色盲安全调色板的指导。色盲分很多种(http://en.wikipedia.org/wiki/Color_blindness)，但是绝大部分色盲都难以区分红色和绿色。一般来说，当需要基于颜色来区分绘制元素时避免使用红色和绿色是个不错的主意。

```
sns.palplot(sns.color_palette("Paired"))
```

```
sns.palplot(sns.color_palette("Set2", 10))
```

函数 `choose_colorbrewer_palette()`可以帮助你从 Color Brewer 库中挑选调色板。这个函数必须在 IPython notebook 中使用，它将启动互动部件让你浏览各种选项和调整其参数。

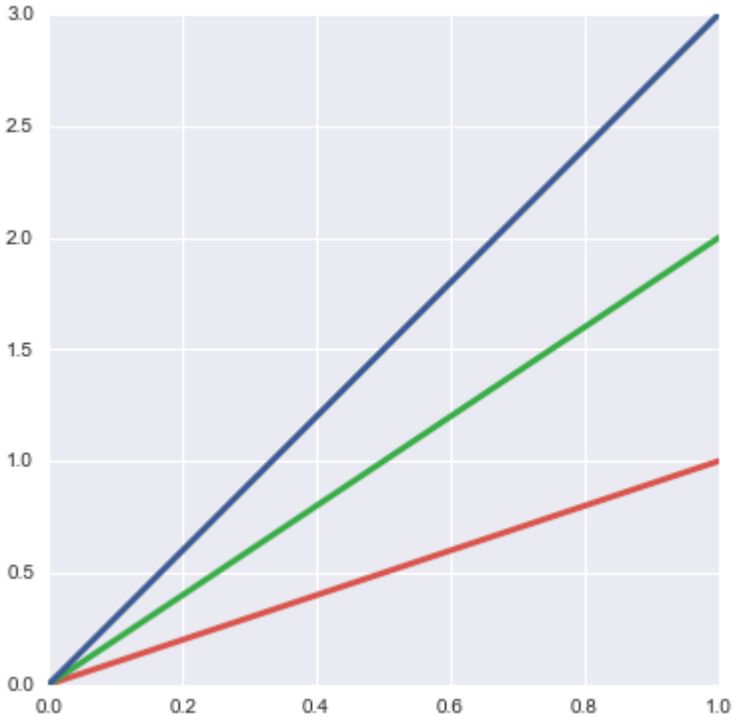
当然，你可能喜欢特定颜色放在一起成颜色组， `color_palette()`允许接收颜色列表，因此这很容易做到。

```
flatui = ["#9b59b6", "#3498db", "#95a5a6", "#e74c3c", "#34495e", "#2ecc71"]
sns.palplot(sns.color_palette(flatui))
```

从 xkcd 颜色调查项目中选择已命名的颜色

前一段时间，xkcd 通过众包方式为随机 RGB 颜色命名。从而产生了 954 个已命名的颜色，你可以在 seaborn 中用 `xkcd_rgb` 字典来引用。

```
plt.plot([0, 1], [0, 1], sns.xkcd_rgb["pale red"], lw=3)
plt.plot([0, 1], [0, 2], sns.xkcd_rgb["medium green"], lw=3)
plt.plot([0, 1], [0, 3], sns.xkcd_rgb["denim blue"], lw=3);
```



如果你想花一些时间来挑选颜色，那么下面这种交互式可视化可能是有用的。除了从 `xkcd_rgb` 字典中挑出单一的颜色，你也可以传送一系列名字到 **`xkcd_palette()`** 函数。

```
colors = ["windows blue", "amber", "greyish", "faded green", "dusty purple"]

sns.palplot(sns.xkcd_palette(colors))
```

连续调色板

调色板的第二个主要类别被称为“连续调色板”。当数据范围从相对较低或者不感兴趣的值过渡到相对较高或者感兴趣的值，使用连续的颜色映射是很合适的。尽管有时你想在连续调色板中使用极端型颜色，但是像 **`kdeplot()`** 或 **`corrplot()`** 函数的使用确实更普遍（功能类似于 matplotlib 函数）。

上述情况在色图中普遍使用 `jet`（或其它彩虹调色板），因为色调的渐变会提供数据的附加信息。然而色图中大幅度的色调变化会引入不连续性（这是数据中没有的），并且我们的视觉系统不能很自然地将彩虹色映射成“高”或者“低”这种定量的区别。结果是这些可视化过程最终更像是一个谜，它们让数据的规律变得更模糊而不是更清晰。`jet` 调色板尤其差，因为最亮的颜色，黄色和青色被用来表示中间数据值。这样使得在强调极值时人们不感兴趣的部分被强化了。

对于连续性数据，最好使用色调变化相对微小的调色板，而不是用亮度和饱和度跳变很大的。这种方法能够使目光自然地聚焦于数据中相对重要的部分。

Color Brewer 库中有大量这样的调色板。它们以主色调的名字来命名。

```
sns.palplot(sns.color_palette("Blues"))
```

和 matplotlib 一样，如果你想让亮度由递增变为递减，可以在调色板名字加上 `_r` 后缀。

```
sns.palplot(sns.color_palette("BuGn_r"))
```

还有一个小技巧，让你可以创建“暗色”的调色板，这种调色板没有那么大的动态范围。当你想映射连续的线或者点，但是亮色的线比较难以区分的时候，“暗色”调色板就派上用场了。

```
sns.palplot(sns.color_palette("GnBu_d"))
```

请记住，你可能想使用 `choose_colorbrewer_palette()` 函数来设置不同的选项，如果你想返回一个色图的颜色对象值，用于传递给 seaborn 或 matplotlib 函数，那么可以设置参数 `as_cmap` 为 `True`。

用 cubehelix_palette()产生连续调色板

对于色调的亮度或其他一些变化呈线性增加或减少，cubehelix 调色板系统创建了连续调色板。这意味着，当被转化成黑白色（打印的时候）或者有色盲个人查看时，色图信息将被保留。

Matplotlib 有一个默认的内置 cubehelix 版本:

```
sns.palplot(sns.color_palette("cubehelix", 8))
```

Seaborn 增加了一个与 cubehelix 系统的接口，这样你可以创建不同的调色板，并且在亮度渐变上都有很好的表现。

Seaborn 的 **cubehelix_palette()**函数默认返回的调色板与 matplotlib 有点不同，它并没有旋转色调更远或覆盖色调周围更广的强度，它还会把颜色的顺序调换，以使更重要的值颜色更深。

```
sns.palplot(sns.cubehelix_palette(8))
```

cubehelix_palette()的其他参数用于控制调色板的外观。两个主要参数是 `start`（0-3 之间的值）和 `rot`（旋转度，-1 到 1 之间的任意值）。

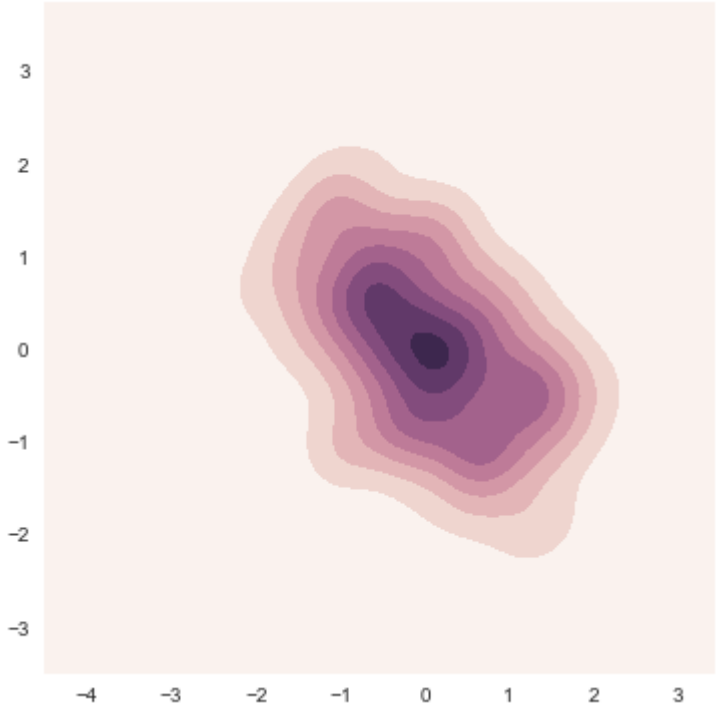
```
sns.palplot(sns.cubehelix_palette(8, start=.5, rot=-.75))
```

你还可以控制端点的暗度和亮度，甚至反转渐变的顺序。

```
sns.palplot(sns.cubehelix_palette(8, start=2, rot=0, dark=0, light=.95, reverse=True))
```

默认你能够得到一个颜色列表，就像其他 seaborn 调色板一样，但是通过设置参数 `as_cmap` 为 `True`，你可以返回调色板如色图对象，作为 seaborn 或 matplotlib 函数的参数。

```
x, y = np.random.multivariate_normal([0, 0], [[1, -.5], [-.5, 1]], size=300).T
cmap = sns.cubehelix_palette(light=1, as_cmap=True)
sns.kdeplot(x, y, cmap=cmap, shade=True);
```



使用该系统时为了帮助选择更好的调色板或色图，你可以在 notebook 使用 **choose_cubehelix_palette()**函数来启动交互式 APP，可以让你设置不同参数。如果你想像 hexbin 类似函数一样返回色图（而不是列表），那么可以将 `as_cmap` 设置为 `True`。

用 light_palette() 和 dark_palette()产生自定义连续调色板

对于自定义连续调色板的简单接口，你可以使用 **light_palette()**或 **dark_palette()**，这两个函数都能基于单一颜色产生调色板，调色板从亮度或深度的不饱和一直到饱和。这两个函数还可以配合 **choose_light_palette()**和 **choose_dark_palette()**函数，后者能够生成交互式的部件。

```
sns.palplot(sns.light_palette("green"))
```

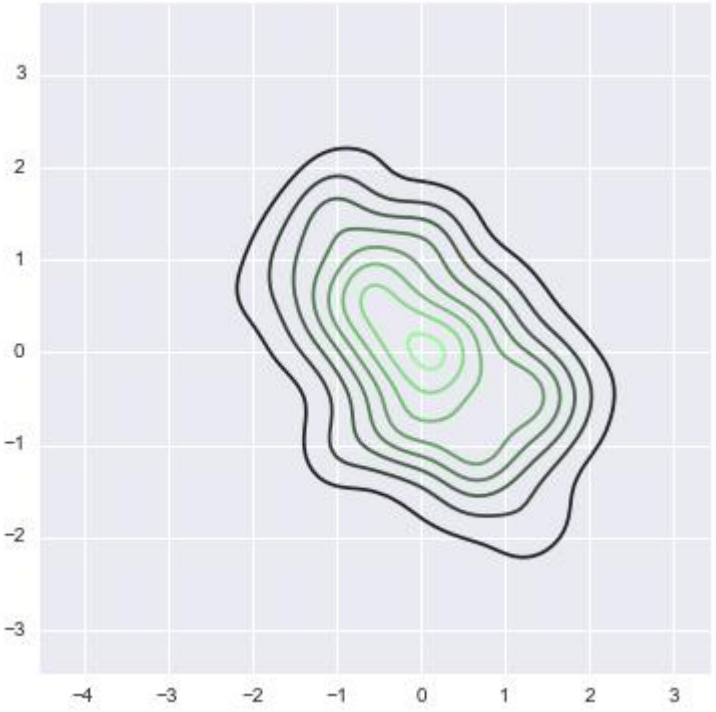
```
sns.palplot(sns.dark_palette("purple"))
```

这些调色板同样可以反转。

```
sns.palplot(sns.light_palette("navy", reverse=True))
```

这些调色板也能够用于创建色图对象而不是颜色列表。

```
pal = sns.dark_palette("palegreen", as_cmap=True)
sns.kdeplot(x, y, cmap=pal);
```



默认情况下，输入可以是任意有效的 matplotlib 颜色。通过调整输入值可以产生不同的演绎。目前，你可以提供 `hls` 或 `husl`（还有默认 RGB 的值）颜色元组，也可以输入从 `xkcd` 库中基于任意有效的颜色产生的调色板。

```
sns.palplot(sns.light_palette((210, 90, 60), input="husl"))
```

```
sns.palplot(sns.dark_palette("muted purple", input="xkcd"))
```

注意，这两个交互式调色板部件的默认输入值是 `husl` 颜色组，这虽然和函数本身的默认项不同，但是 `husl` 颜色组在这里是很有用的。

极端型调色板

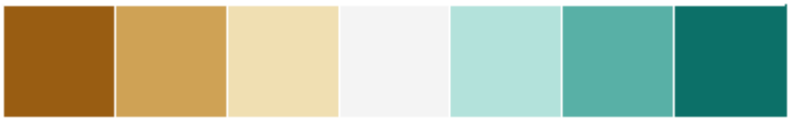
第三类调色板叫做“极端型调色板”，用于对高、低值感兴趣的数据，且这些数据通常都有一个定义明确的数据中点。例如，当你正在基线时间点开始绘制温度的变化时，最好是使用不同的配色来显示温度相对降低或升高的区域。

现在除了在中点以明确的色调起始出发到不需要强调的颜色之间微妙的变化外，选择适合的极端型调色板的规则与连续型调色板相似。同样重要的是，起始值是相似的亮度和饱和度。

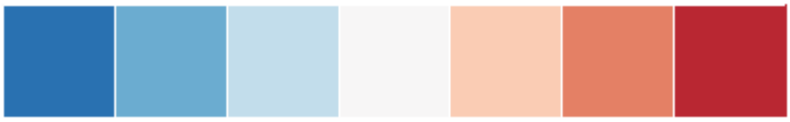
同样需要强调的是应该避免使用红色和绿色，因为有许多人无法分辨它们。

令人惊奇的是 Color Brewer 颜色库附带了一套可供选择的极端型调色方案。

```
sns.palplot(sns.color_palette("BrBG", 7))
```



```
sns.palplot(sns.color_palette("RdBu_r", 7))
```



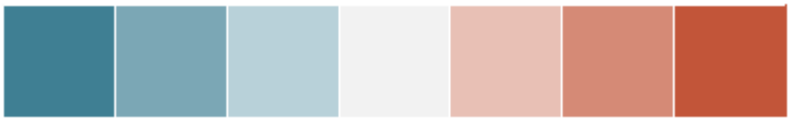
另一可供选择的颜色库是 matplotlib 自带的 coolwarm 调色板。注意此调色板缺少中间值和极值的对比。

```
sns.palplot(sns.color_palette("coolwarm", 7))
```

使用 diverging_palette()自定义极端型调色板

可以使用 seaborn 中的 **`diverging_palette()`**函数创建自定义的极端型调色方案。（自然也存在伴随交互式部件，**`choose_diverging_palette()`**）。该函数使用 `husl` 颜色系统制作极端型调色板。你可以传递两端的色调（色度）值或亮度和饱和度值来生成极值。使用 `husl` 意味着从极值到中点的产生的颜色有很好的平衡性。

```
sns.palplot(sns.diverging_palette(220, 20, n=7))
```



```
sns.palplot(sns.diverging_palette(145, 280, s=85, l=25, n=7))
```

参数 sep 控制调色板中间区域两个颜色带的分隔宽度。

```
sns.palplot(sns.diverging_palette(10, 220, sep=80, n=7))
```

也可以将调色板的中点设置为暗色而不是亮色。

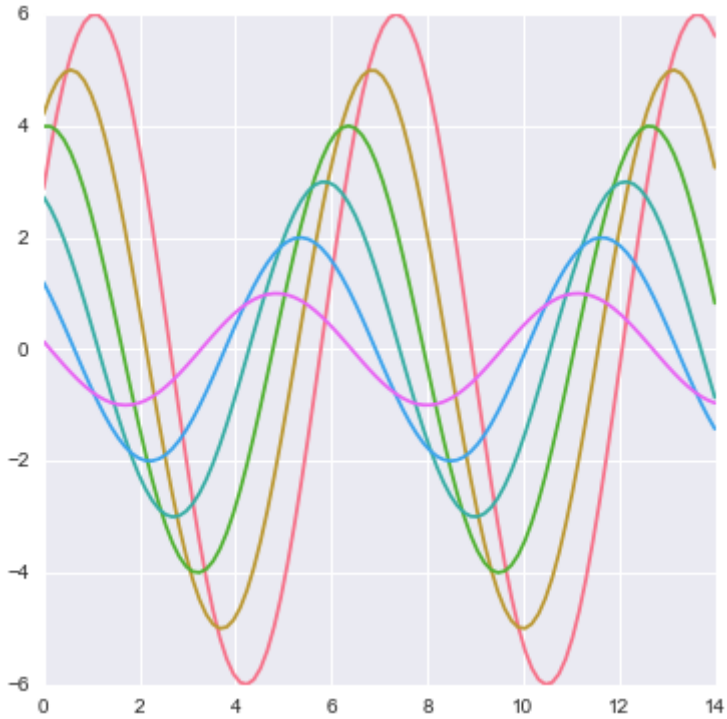
```
sns.palplot(sns.diverging_palette(255, 133, l=60, n=7, center="dark"))
```

使用 set_palette()变更默认调色板

`color_palette()`函数有一个伴随函数 `set_palette()`。它们之间的关系类似于 aesthetics tutorial。`set_palette()`接受与 `color_palette()`相同的参数，但是它会改变 matplotlib 的默认参数，使所有的绘图都使用这个调色板。

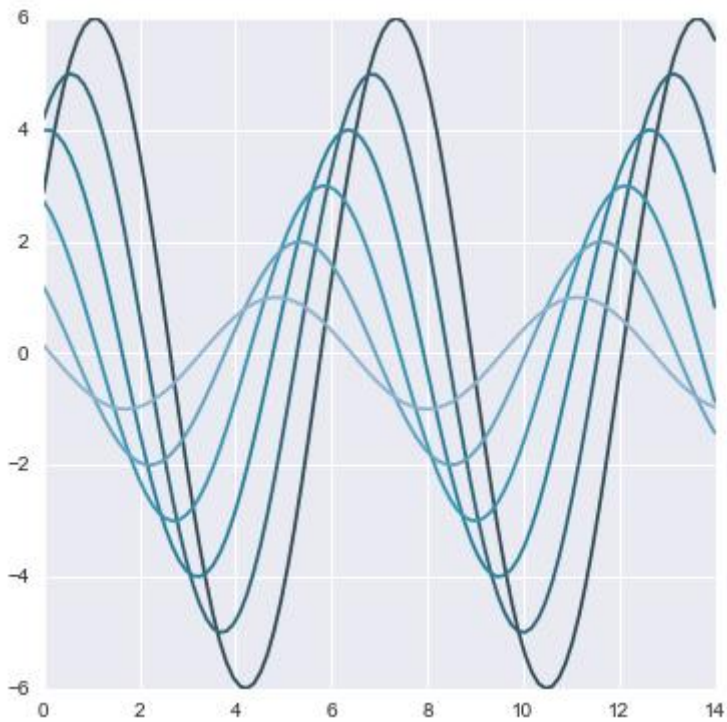
```
def sinplot(flip=1):
    x = np.linspace(0, 14, 100)
    for i in range(1, 7):
        plt.plot(x, np.sin(x + i * .5) * (7 - i) * flip)
sns.set_palette("husl")
```

```
sinplot()
```



`color_palette()` 函数也可用在 with 声明中，暂时改变调色板。

```
with sns.color_palette("PuBuGn_d"):
    sinplot()
```



绘图函数

可视化数据集的分布

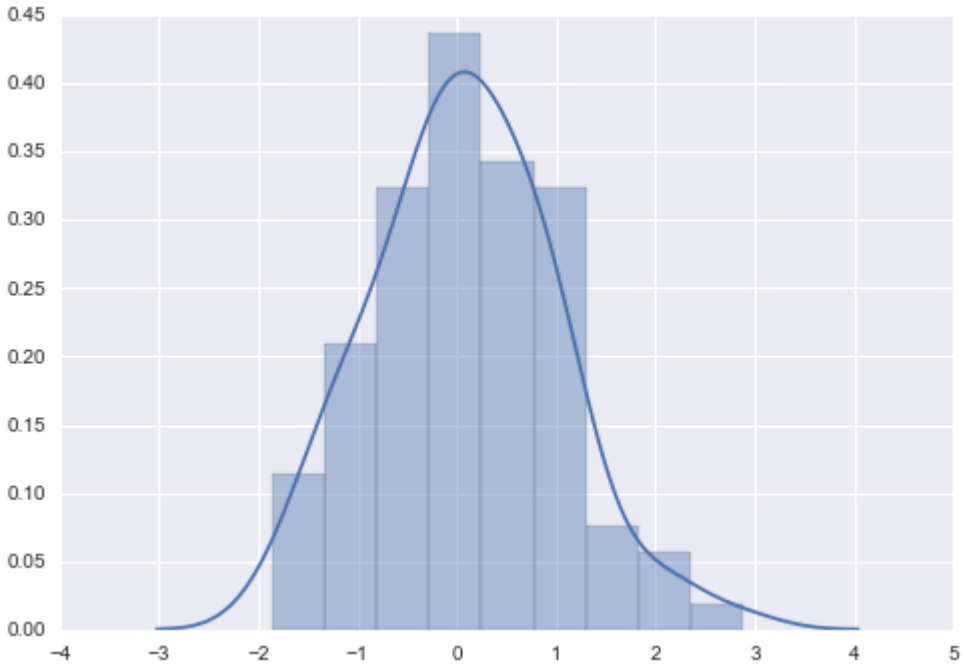
在处理一组数据时，通常首先要做的第一件事就是知道数据如何分布。本章教程将简要介绍的一些工具，在 seborn 检查单变量和双变量分布。你可能还想看一个分类绘图章节的一些函数例子，使得在比较一个变量在其它变量不同水平下的分布变得容易。

```
%matplotlib inline
import numpy as np
import pandas as pd
from scipy import stats, integrate
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)
np.random.seed(sum(map(ord, "distributions")))
```

绘制单变量分布

采取在 seaborn 单变量分布的快速浏览最方便的方式是 distplot()功能。默认情况下，它将画出一个直方图和拟合内核密度估计（KDE）

```
x = np.random.normal(size=100)
sns.distplot(x);
```

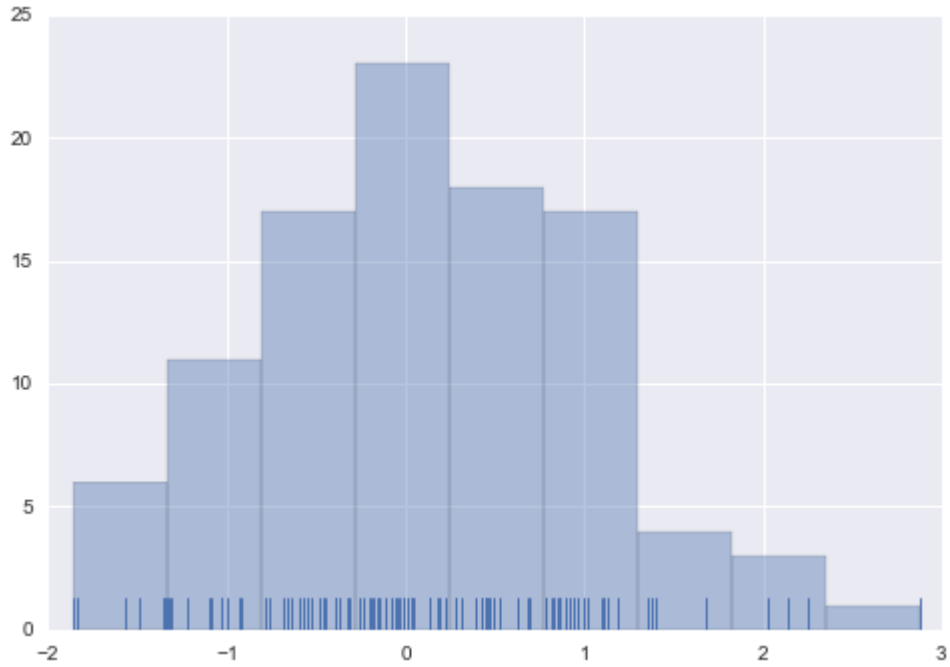


直方图

直方图大家很熟悉，而且已经存在于 matplotlib 中的 hist 函数。直方图通过一定范围的数据来生成分箱和绘制长条来显示进入每个分箱的观测数据。

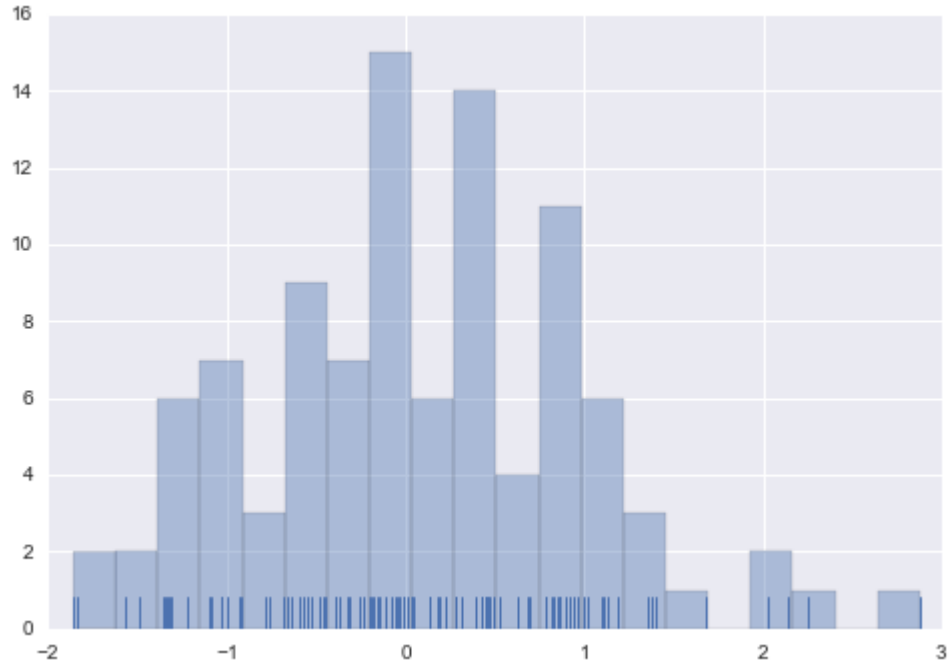
为了说明这一点，我们消除密度曲线并增加地毯图。它会在每个观测点都画了一个很小的垂直刻度。你可使用 **rugplot()**函数来绘制地毯图，但它在 **distplot()**函数也能用。

```
sns.distplot(x, kde=False, rug=True);
```



绘制直方图时，主要选择设置分箱的数量和把它们放在哪里。**distplot()**函数在默认情况下使用简单的规则来猜测什么是合适的数量。但尝试使用更多或更少的分箱来确定是否可能揭示数据中的其他特性。

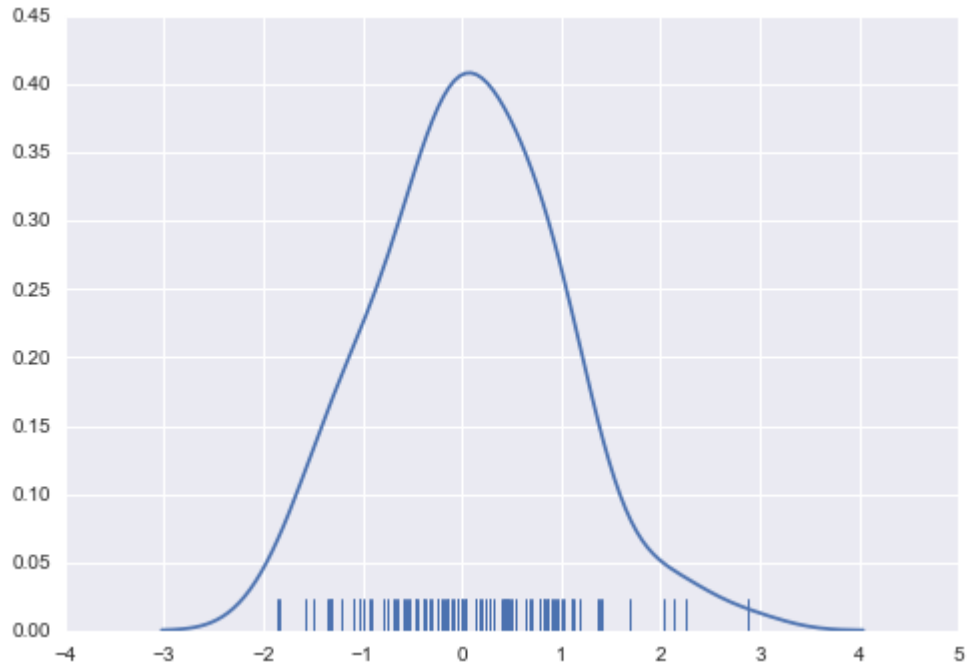
```
sns.distplot(x, bins=20, kde=False, rug=True);
```



核密度估计

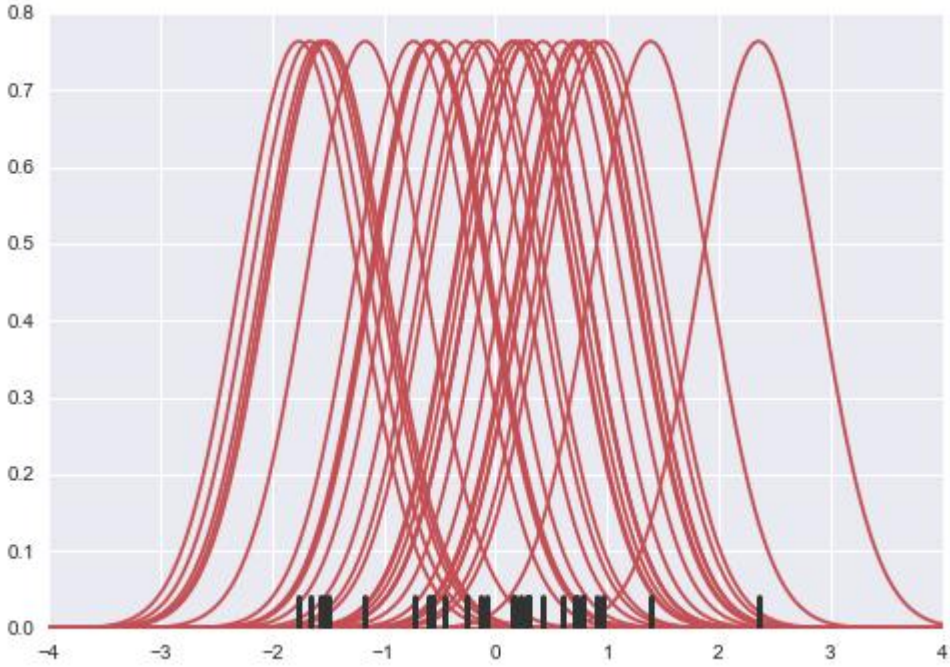
核密度估计可能大家不太熟悉，但它可以是绘制分布形状有用的工具。就像直方图，KDE 绘图对沿着一个高度轴和其它轴的观测点密度进行了编码。

```
sns.distplot(x, hist=False, rug=True);
```



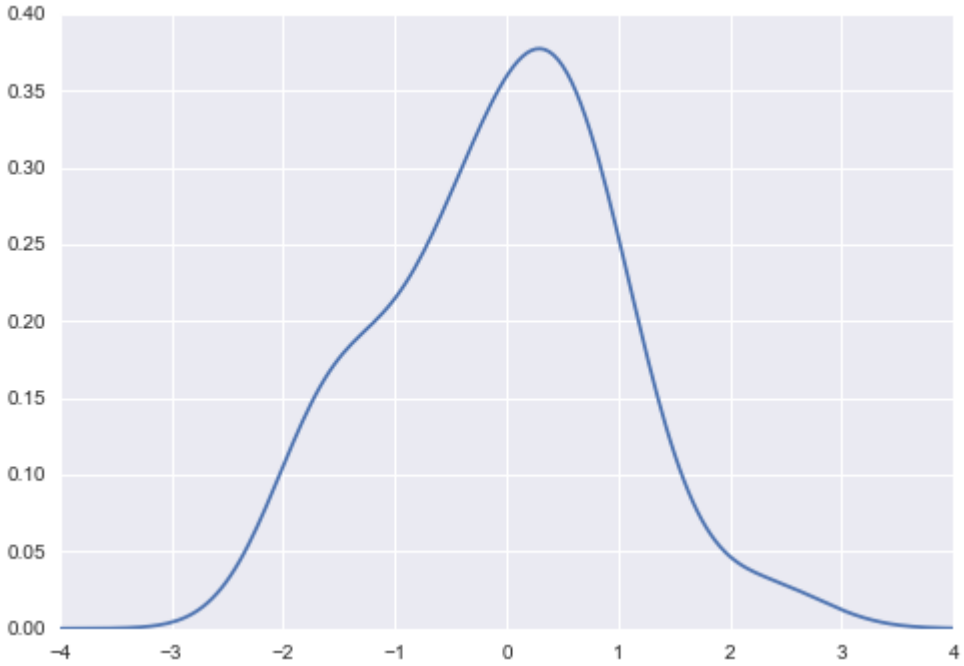
绘制一个 KDE 比绘制一个直方图需要更多的计算。这是因为每个观测点首先被正态（高斯）曲线中间的那个值所替代。

```
x = np.random.normal(0, 1, size=30)
bandwidth = 1.06 * x.std() * x.size ** (-1 / 5.)
support = np.linspace(-4, 4, 200)

kernels = []
for x_i in x:

    kernel = stats.norm(x_i, bandwidth).pdf(support)
    kernels.append(kernel)
    plt.plot(support, kernel, color="r")

sns.rugplot(x, color=".2", linewidth=3);
```



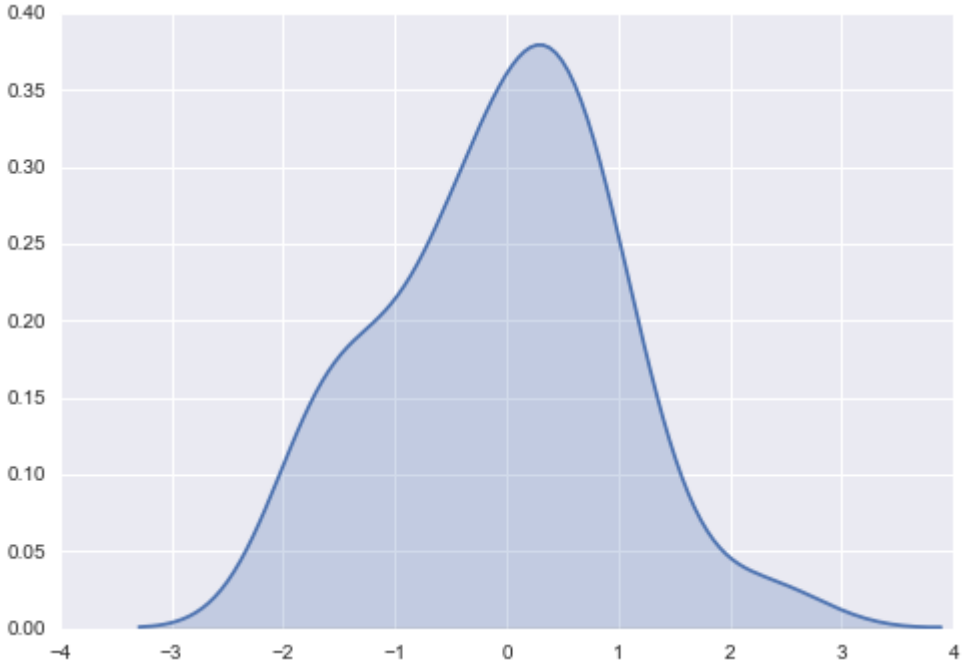
接下来，这些曲线是计算网格中每个点的密度值之和，由此产生的曲线归一化，曲线下的面积等于 1。

```
density = np.sum(kernels, axis=0)
density /= integrate.trapz(density, support)
plt.plot(support, density);
```



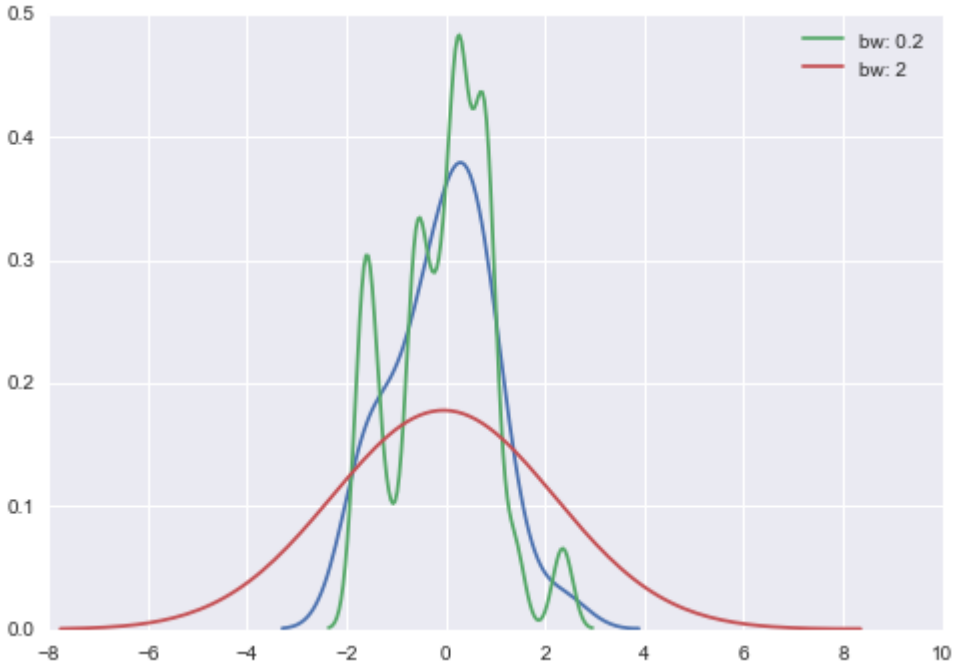
我们看到，如果我们在 seaborn 使用 `kdeplot()`函数，我们可以得到同样的曲线。该函数使用了 `distplot()`函数，但当你只是想密度估计它能提供一个更方便的直接接口来获得其他选项。

```
sns.kdeplot(x, shade=True);
```



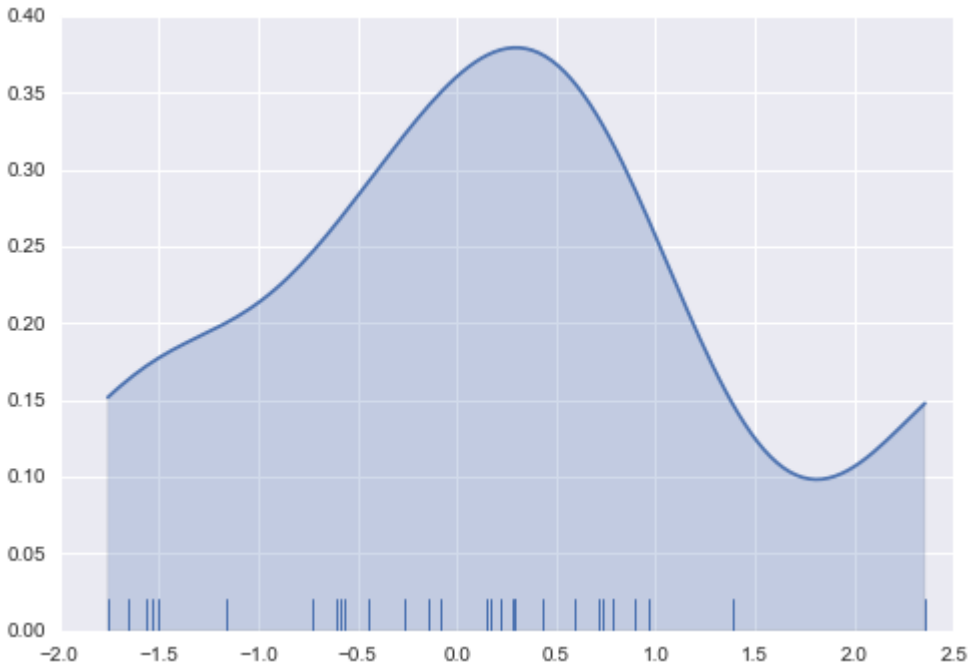
KDE 的宽度(bw)参数控制估计拟合数据的紧密程度，就像直方图中分箱大小。它对应于我们之前绘制的核宽度。默认行为是尝试使用共同参考规则去猜测一个好的数值，但它可能有利于尝试更大或更小的值。

```
sns.kdeplot(x)
sns.kdeplot(x, bw=.2, label="bw: 0.2")
sns.kdeplot(x, bw=2, label="bw: 2")
plt.legend();
```



正如上面你所看到的，高斯 KDE 过程的本质意味着估计扩展了以前数据集的的最大值和最小值。可以用 `Cut` 参数控制曲线中极端值的追溯程度。然而，这仅影响曲线如何绘制而不是影响如何拟合。

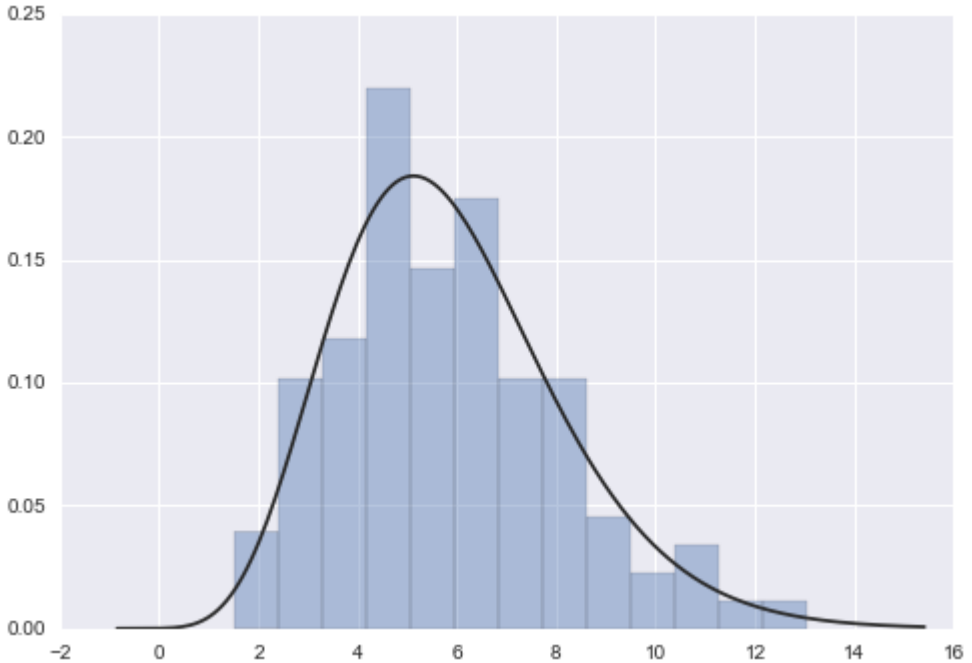
```
sns.kdeplot(x, shade=True, cut=0)
sns.rugplot(x);
```



拟合参数分布

你还可以使用 `distplot()` 来拟合数据集的参数分布，直观评估它如何贴合对应观测数据。

```
x = np.random.gamma(6, size=200)
sns.distplot(x, kde=False, fit=stats.gamma);
```



绘制双变量分布

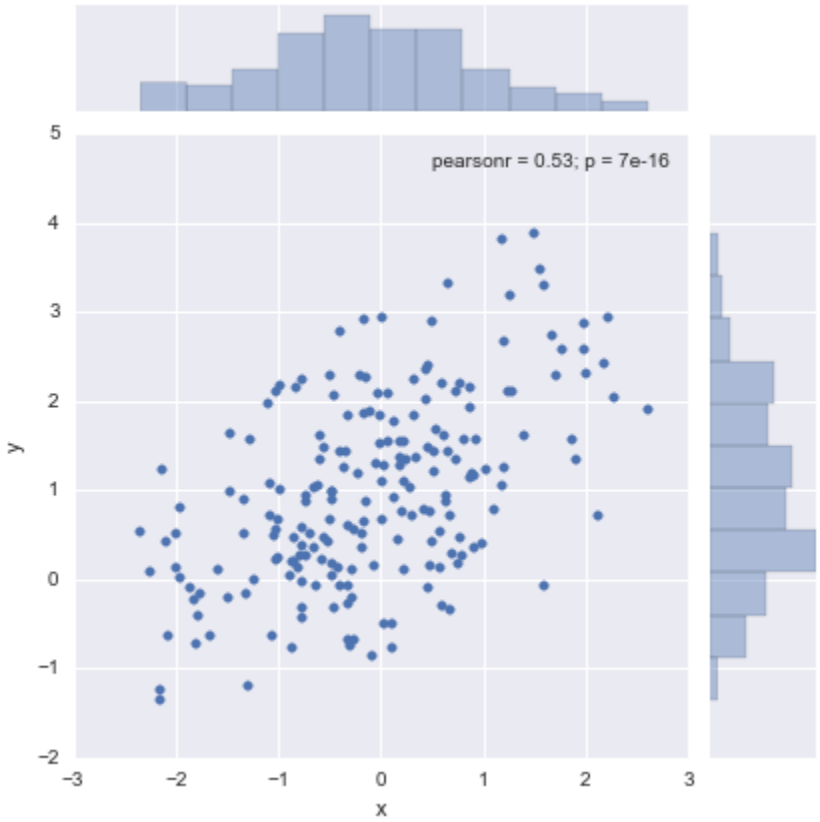
它对于可视化两个变量的双变量分布也是有用的。Seaborn 最简单的方法是调整 `jointplot()`函数，该函数创建了一个多层面板图，能显示两个变量之间的双变量（或联合）关系与每个单独轴的单变量（或边际）分布。

```
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
```

散点图

最熟悉的可视化双变量方式是散点图，每个观测点显示 x 和 y 值，这与两个维度上的地毯图相似。你可以使用 matplotlib `plt.scatter` 函数画散点图，并且它也用 `jointplot()`函数的默认显示类型。

```
sns.jointplot(x="x", y="y", data=df);
```

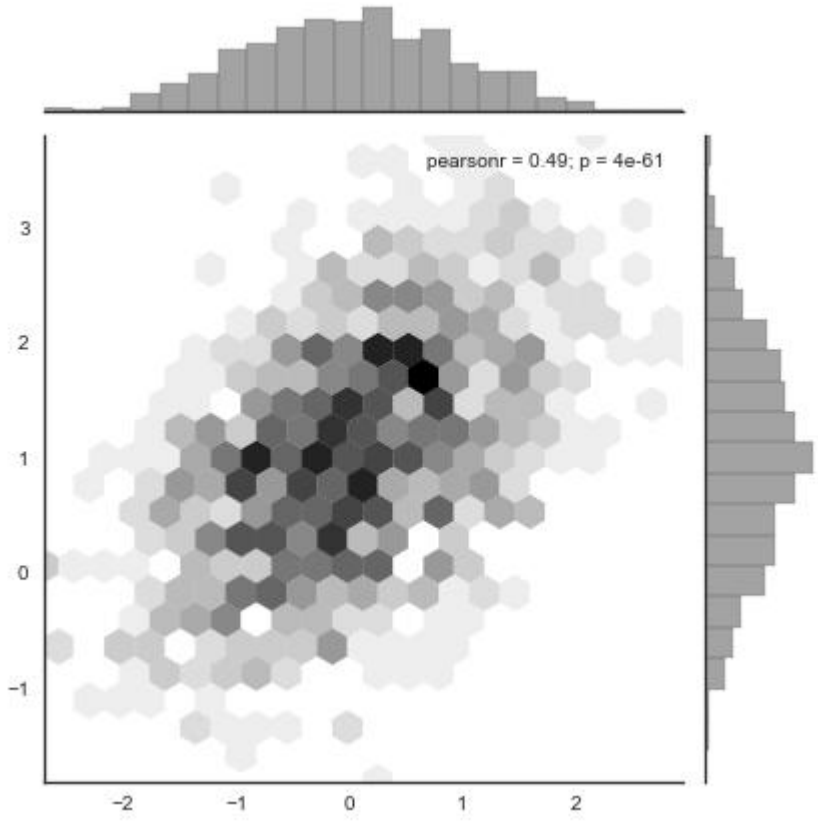


六边分箱绘图

与直方图的相似的还有被称为“hexbin”的绘图，因为它显示了落入六边形分箱观测点的计数值。这有利于相

对大型数据集的绘制。它可以通过 matplotlib `plt.hexbin` 函数和 **`jointplot()`**函数的样式来绘图，用白色背景看起来最好。

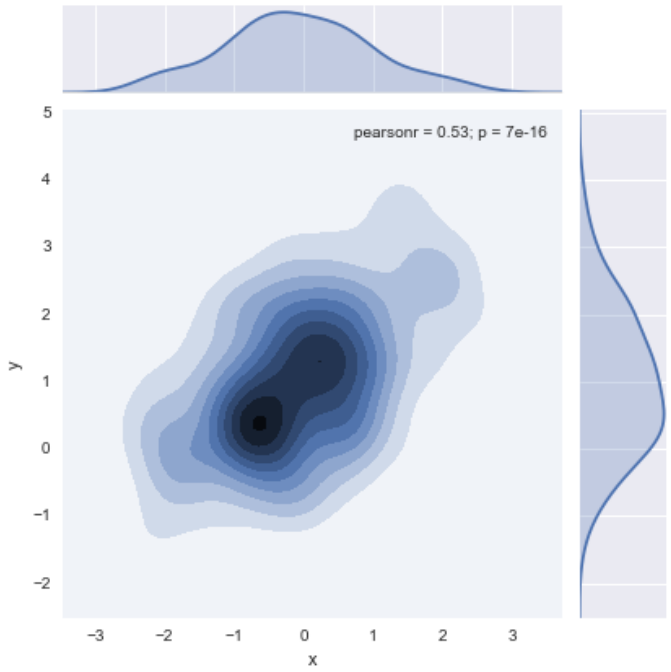
```
    x, y = np.random.multivariate_normal(mean, cov, 1000).T
    with sns.axes_style("white"):
        sns.jointplot(x=x, y=y, kind="hex", color="k");
```



核密度估计

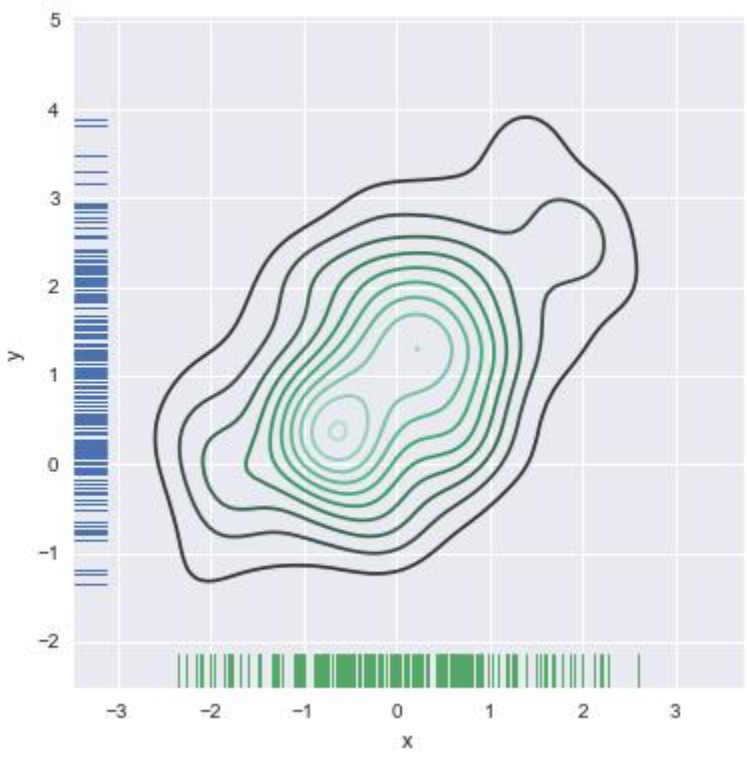
它也可能使用上述的核密度估计来更好的双变量分布。在 seaborn 中，这类绘图采用等高线图来显示，可以使用 **`jointplot()`**的风格项。

```
sns.jointplot(x="x", y="y", data=df, kind="kde");
```



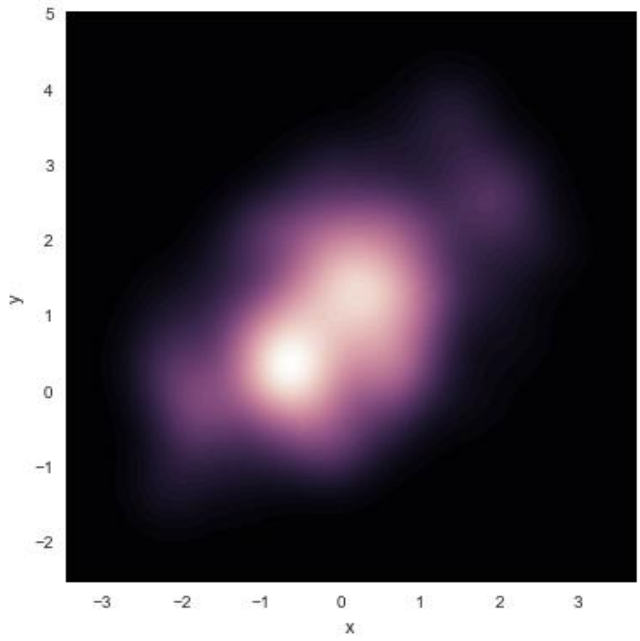
你也可以使用 **kdeplot()**函数画一个二维的内核密度图。它允许你在指定（并可能已经存在）的 matplotlib 轴画这种图，而 **jointplot()**函数会管理自己的图。

```
f, ax = plt.subplots(figsize=(6, 6))
sns.kdeplot(df.x, df.y, ax=ax)
sns.rugplot(df.x, color="g", ax=ax)
sns.rugplot(df.y, vertical=True, ax=ax);
```



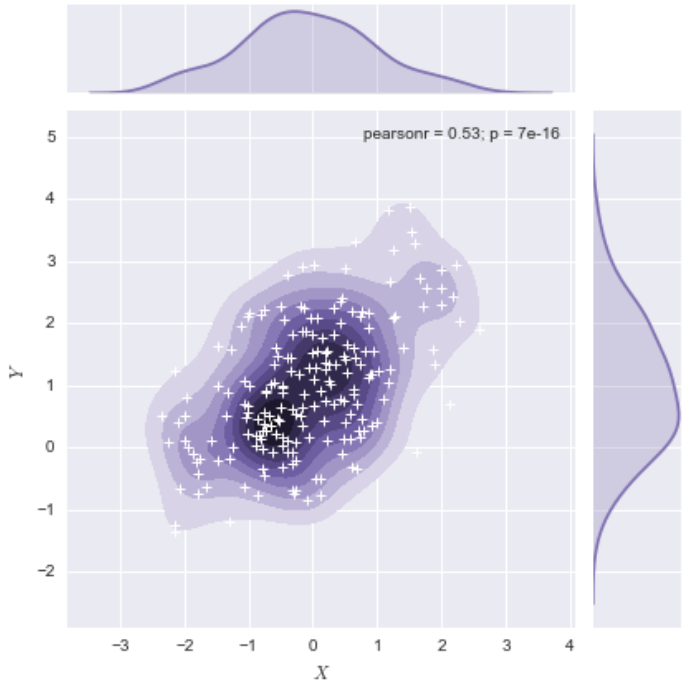
如果你希望更多连续不断地显示双变量密度，你可以简单地增加等高水平的数量。

```
f, ax = plt.subplots(figsize=(6, 6))
cmap = sns.cubehelix_palette(as_cmap=True, dark=0, light=1, reverse=True)
sns.kdeplot(df.x, df.y, cmap=cmap, n_levels=60, shade=True);
```



`jointplot()` 函数使用 `JointGrid` 来管理图形。为了增加灵活性，你可能想直接使用 `JointGrid` 来绘图 。在绘图后 `jointplot()` 函数返回 `JointGrid` 对象，如此你可以使用它来添加更多的层或微调可视化的其他方面。

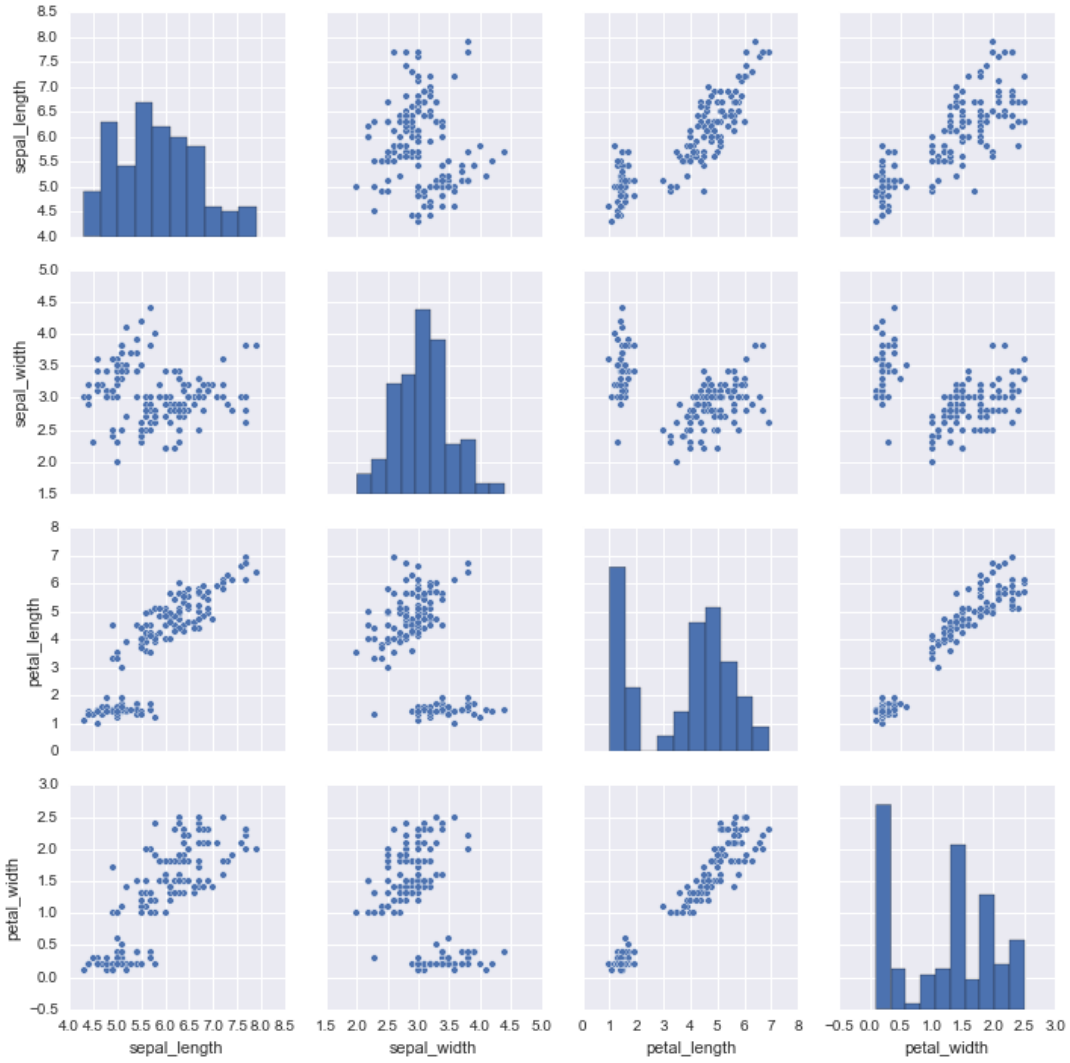
```
g = sns.jointplot(x="x", y="y", data=df, kind="kde", color="m")
g.plot_joint(plt.scatter, c="w", s=30, linewidth=1, marker="+")
g.ax_joint.collections[0].set_alpha(0)
g.set_axis_labels("$X$", "$Y$");
```



可视化数据中成对关系

在一个数据集中绘制多成对双变量分布，你可以使用 **pairplot()** 函数。该函数创建了矩阵的轴并且显示数据框中每对列的关系。默认情况下，该函数也画出单变量分布，该分布每个变量位于对角线。

```
iris = sns.load_dataset("iris")
sns.pairplot(iris);
```

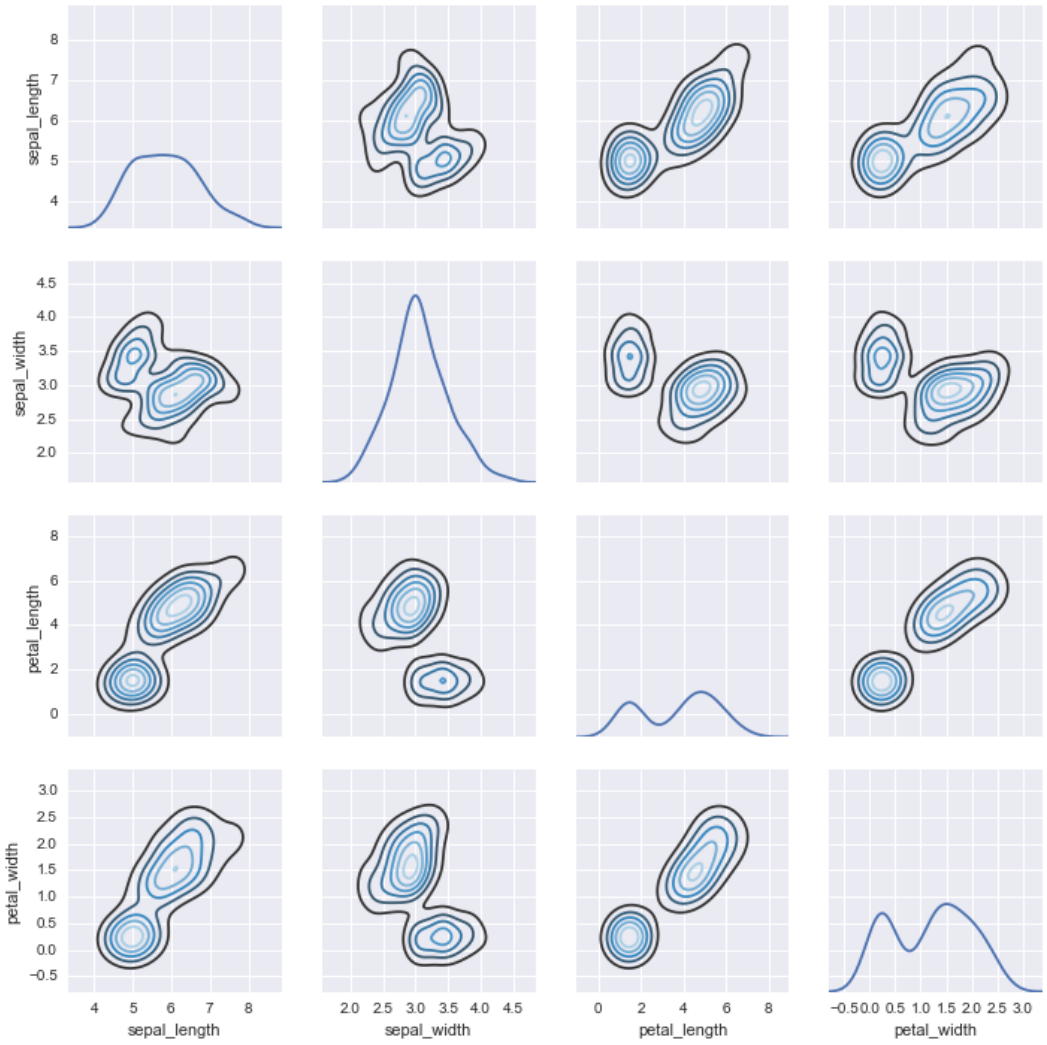


很像 **jointplot()** 函数和 **JointGrid** 之间的关系，**pairplot()** 函数是建立在 **PairGrid** 对象之上的，为了增加灵活性直接使用。

```
g = sns.PairGrid(iris)
g.map_diag(sns.kdeplot)
g.map_offdiag(sns.kdeplot, cmap="Blues_d", n_levels=6);
```

```
/Users/mwaskom/anaconda/lib/python2.7/site-packages/matplotlib/axes/_axes.py:519: UserW
arning: No labelled objects found. Use label='...' kwarg on individual plots.
```

```
warnings.warn("No labelled objects found. "
```



可视化线性关系

许多数据集包含许多的定量变量，分析的目的往往是为了找出这些变量之间的关系。我们之前讨论的函数，是显示两个变量的联合分布来实现。然而，利用统计模型估计含有噪音的两个观测集的简单关系是有帮助的。本章所讨论的函数，将采取线性回归的通用框架来实现。

本着图基（美国著名统计学家）的精神，在探索性数据分析时，seaborn的回归图主要通过增加视觉引导来强化数据模式。也就是说，seaborn自身并非只是一个统计分析工具。为了使用量化手段来拟合相关的回归模型时，你需要使用statsmodels。然而，seaborn的目的是通过可视化快速、简单的探索性分析一个数据集，跟通过统计数据表格探索一个数据集同样（不亚于）重要。

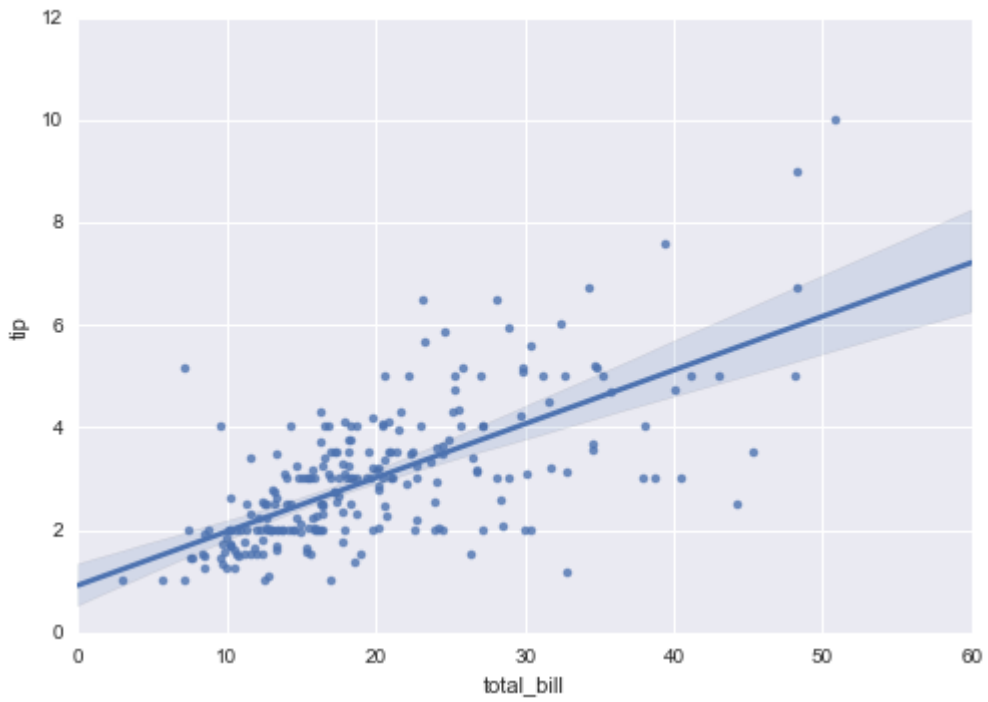
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)
np.random.seed(sum(map(ord, "regression")))
tips = sns.load_dataset("tips")
```

绘制线性回归模型的函数

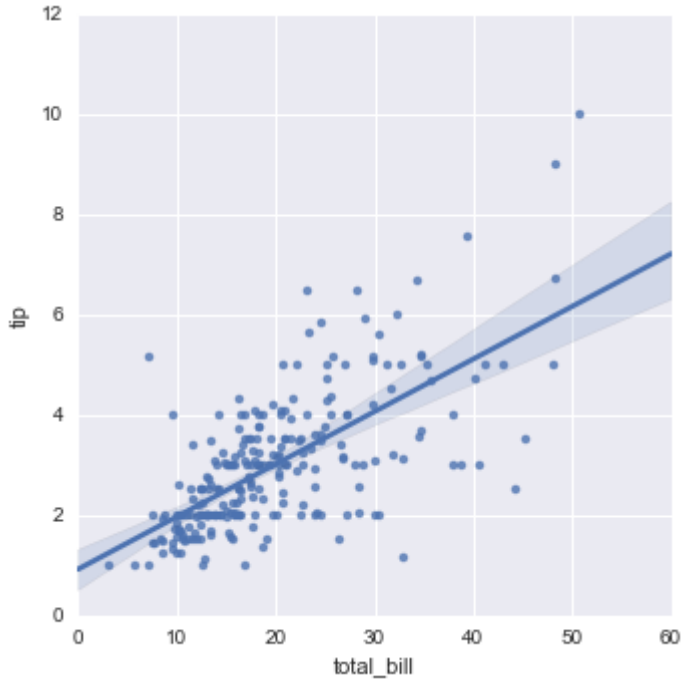
Seaborn的两个主要功能是通过回归分析来实现可视化线性关系。这两个函数是regplot()和lmplot()，互相紧密联系，并共享各自的核心功能。但是，搞清楚他们之间的差异十分重要，这使你能为某项工作快速选择出正确的工具。

最简单的调用方式是，两个函数画出x和y两个变量的散点图，然后拟合y ~ x的回归模型，绘制最终的回归线，该回归线的置信空间为95%。

```
sns.regplot(x="total_bill", y="tip", data=tips);
```



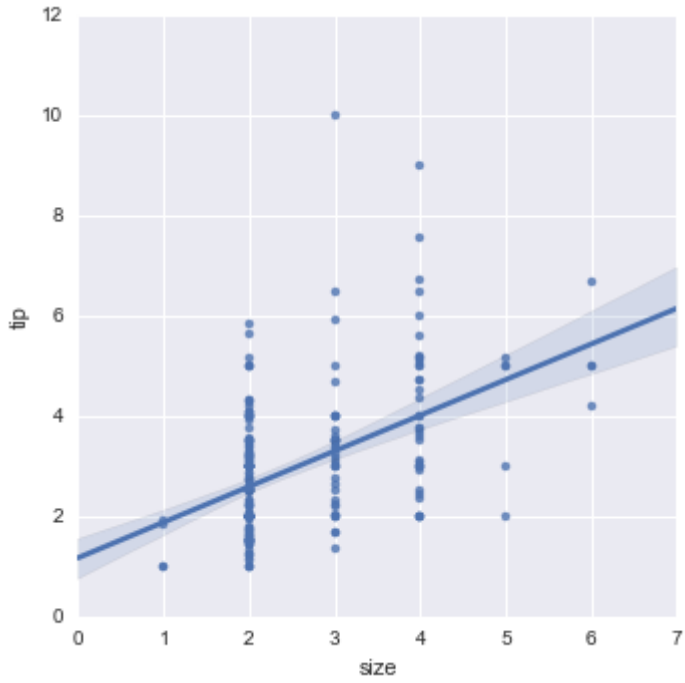

```
sns.lmplot(x="total_bill", y="tip", data=tips);
```



你应该注意到绘图结果是相同的，除了图的形状不同以外。不久我们将解释原因。现在，其它最主要的区别是regplot()接受各种格式的变量x和变量y，包括简单numpy数组，pandas中Series对象，或作为pandas的DataFrame对象传递给data。相反，lmplot()的data参数有要求变量x和变量y必须是特定的字符串。这种数据格式称为“长型”或“规整”的数据。除了输入的便利性，regplot()包含lmplot()的一个特征子集，稍后我们将做演示。

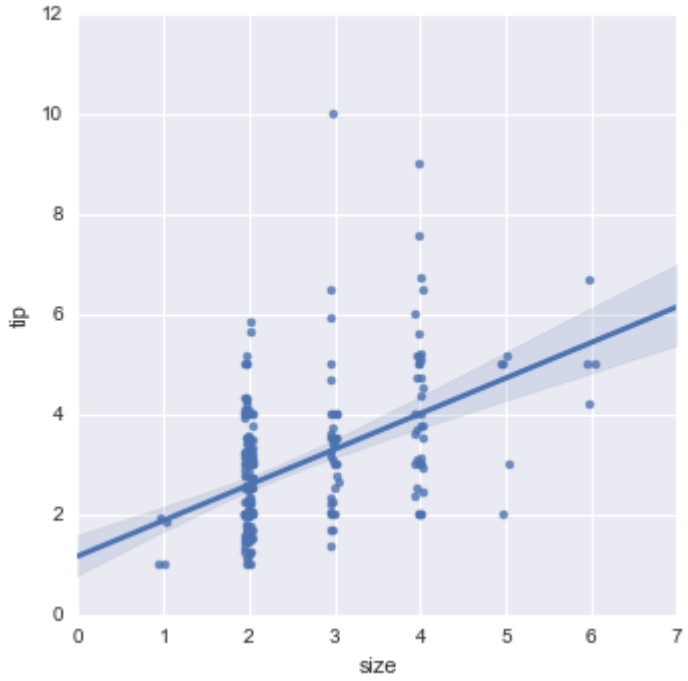
当其中一个变量含有离散值时，拟合出一个线性回归是可能的，但是，这种离散数据集简单生成散点图时常并非最佳：

```
sns.lmplot(x="size", y="tip", data=tips);
```



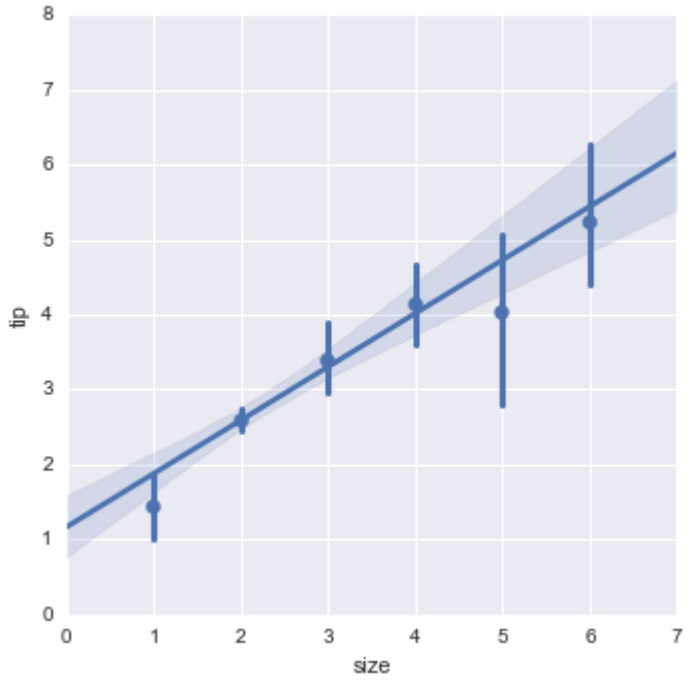
一种选择是在离散值中增加一些随机噪声（“抖动”），使得离散值的分布变得更加清晰。注意“抖动”仅用于散点图的数据，并不影响拟合的回归线：

```
sns.lmplot(x="size", y="tip", data=tips, x_jitter=.05);
```



第二种选择，合并每个离散分箱的观测，沿着置信区间绘制集中趋势估计：

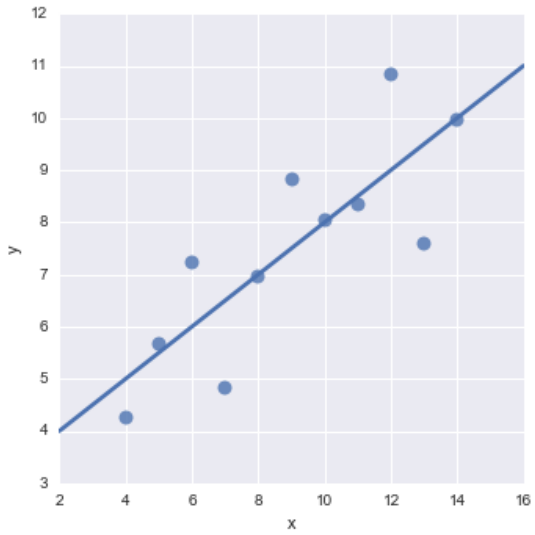
```
sns.lmplot(x="size", y="tip", data=tips, x_estimator=np.mean);
```



拟合不同种类的模型

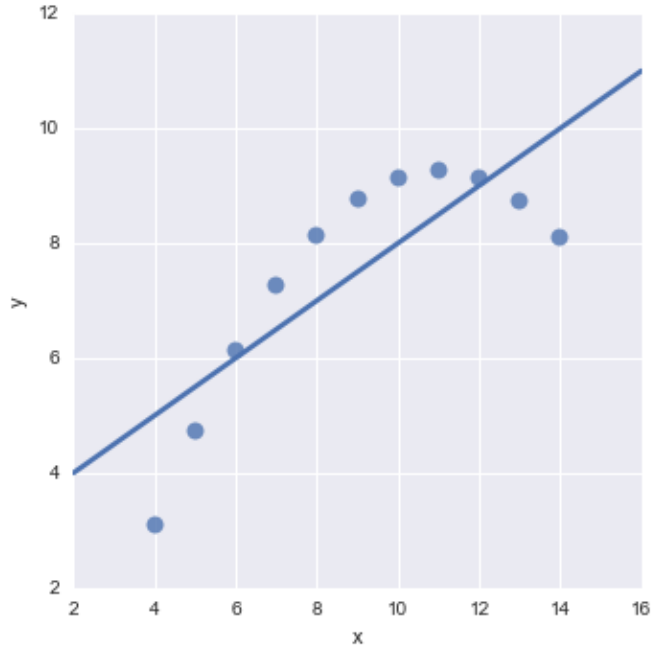
简单的线性回归模型，用的是非常简单的拟合，然而,它并不适合多种类型的数据集。统计学家 F.J. Anscombe 构造出的四组奇特的数据展示了几个例子，简单的线性回归对关系提供了一个相同的估计，但我们用肉眼就能清楚的看出他们的差异。例如，在第一种情况下，线性回归是一个很好的模型：

```
anscombe = sns.load_dataset("anscombe")
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'I'"),
           ci=None, scatter_kws={"s": 80});
```



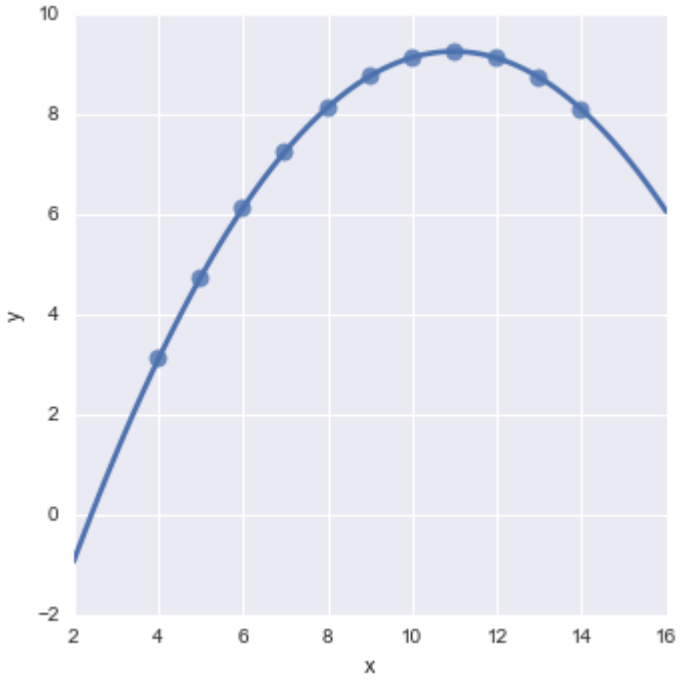
对于第二个数据集来说，他们的线性关系是相同的，但图清楚地表明，这不是一个很好的模型：

```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
           ci=None, scatter_kws={"s": 80});
```



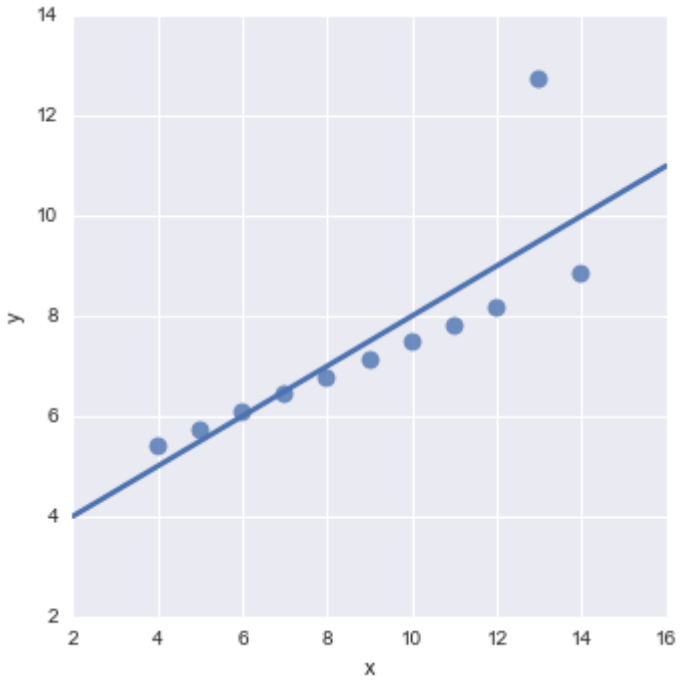
在表达这些高阶关系时，lmplot()和 regplot()可以在简单的非线性数据集中拟合多项式回归模型：

```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
           order=2, ci=None, scatter_kws={"s": 80});
```



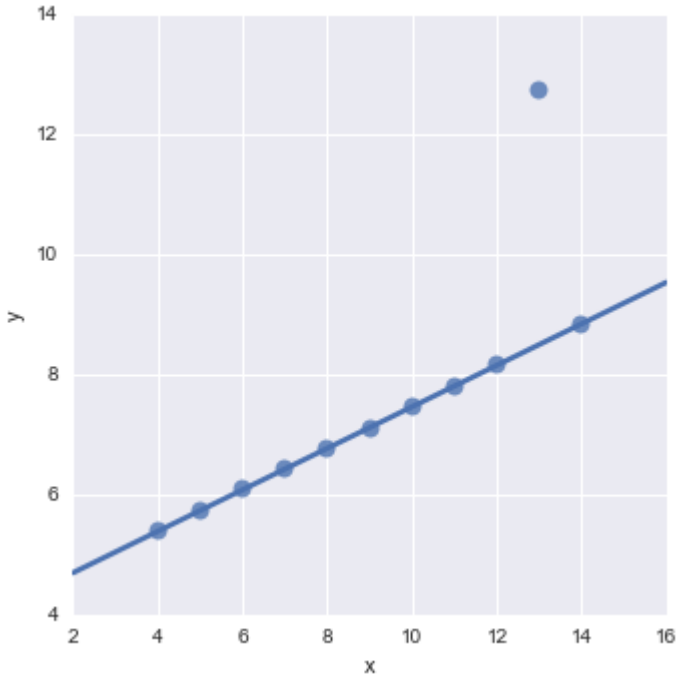
构成“离群”观测的不同问题是研究一些偏离主要关系以外的某些原因：

```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'III'"),
           ci=None, scatter_kws={"s": 80});
```



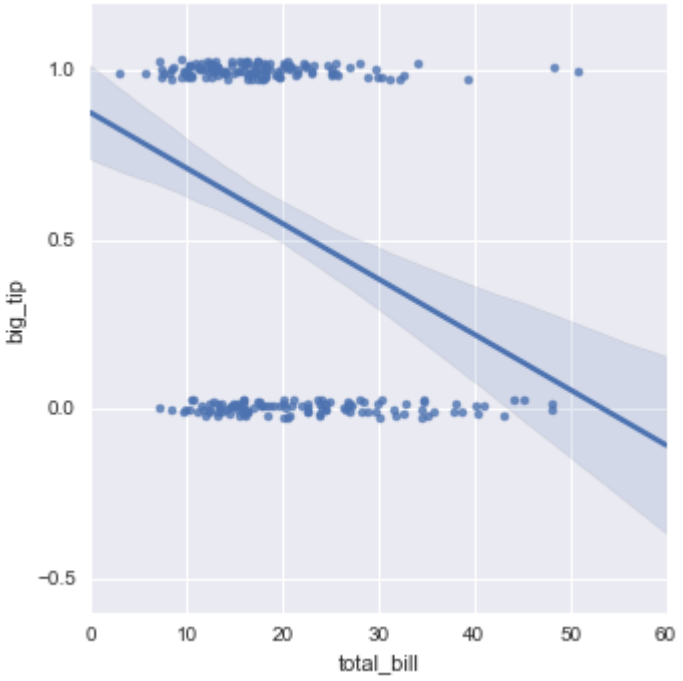
为了表达离群值，使用不同损失函数来降低比较大的误差的稳健回归是有用的：

```
sns.lmplot(x="x", y="y", data=anscombe.query("dataset == 'III'"),
           robust=True, ci=None, scatter_kws={"s": 80});
```



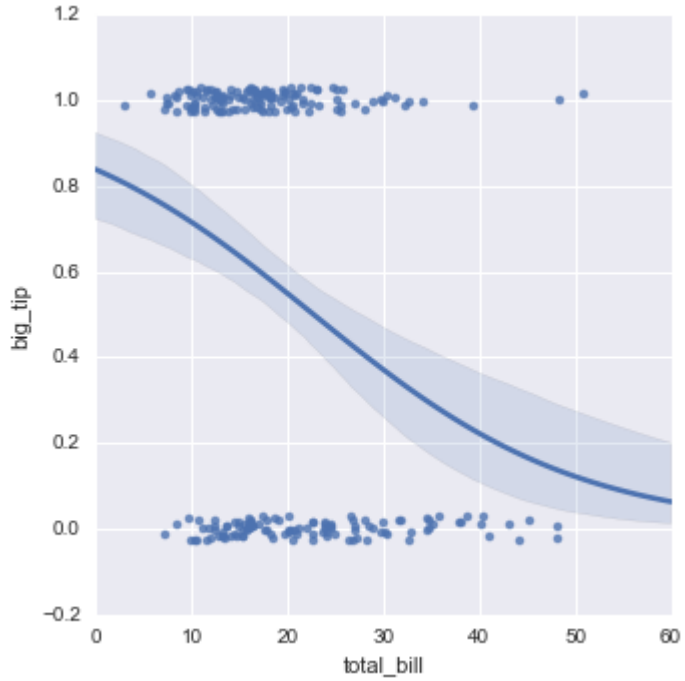
当 Y 变量是二分类的时候，简单线性回归也能“工作”，但提供了不合情理的预测。

```
tips["big_tip"] = (tips.tip / tips.total_bill) > .15
sns.lmplot(x="total_bill", y="big_tip", data=tips,
           y_jitter=.03);
```



这种情况下的解决方案是拟合逻辑回归，在给定 x 值的回归线显示 y 的概率估计。

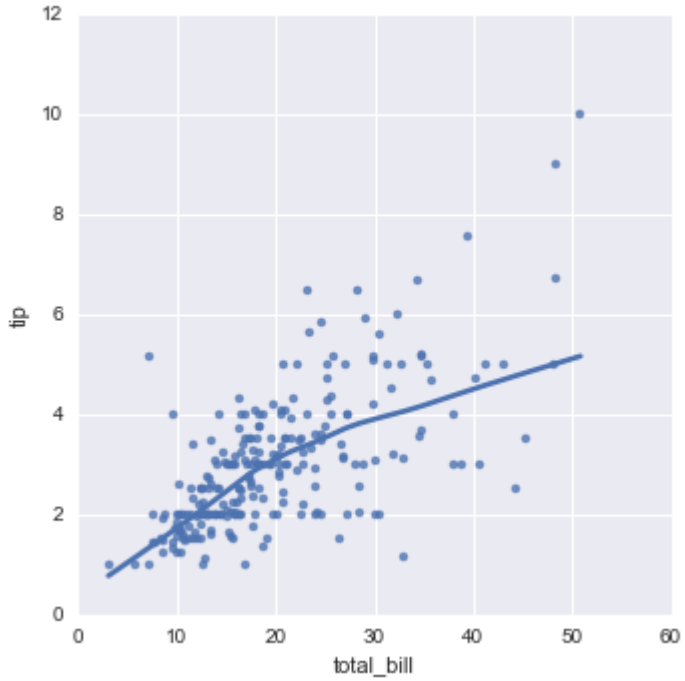
```
sns.lmplot(x="total_bill", y="big_tip", data=tips,
           logistic=True, y_jitter=.03);
```



请注意逻辑回归估计比简单的回归更需要大量计算（稳健回归也是一样），使用 bootstrap 方法计算回归线的置信区间，你可能希望为了更快迭代而关闭它（使用 ci=None）。

一个完全不同的方法是使用一个 lowess 去拟合非参数回归，这种方法具有最少的假设，虽然它是计算密集型的，但目前置信区间是不需要计算的。

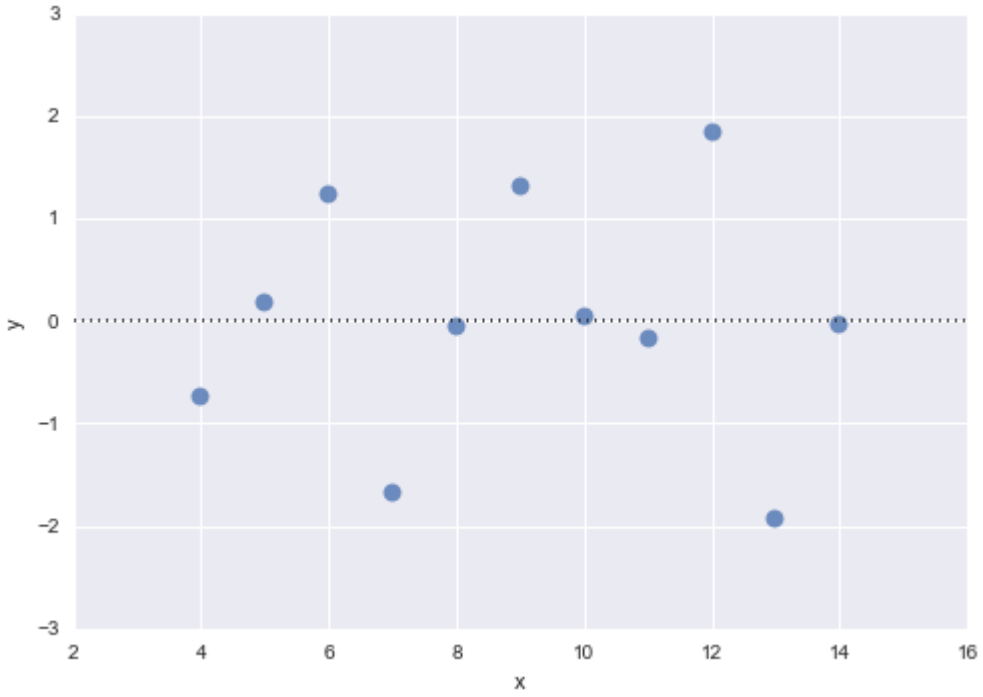
```
sns.lmplot(x="total_bill", y="tip", data=tips,
           lowess=True);
```



residplot()方法作为检测简单回归模型是否适用一个数据集的工具是有用的，它拟合并删除一个简单的线性回归，

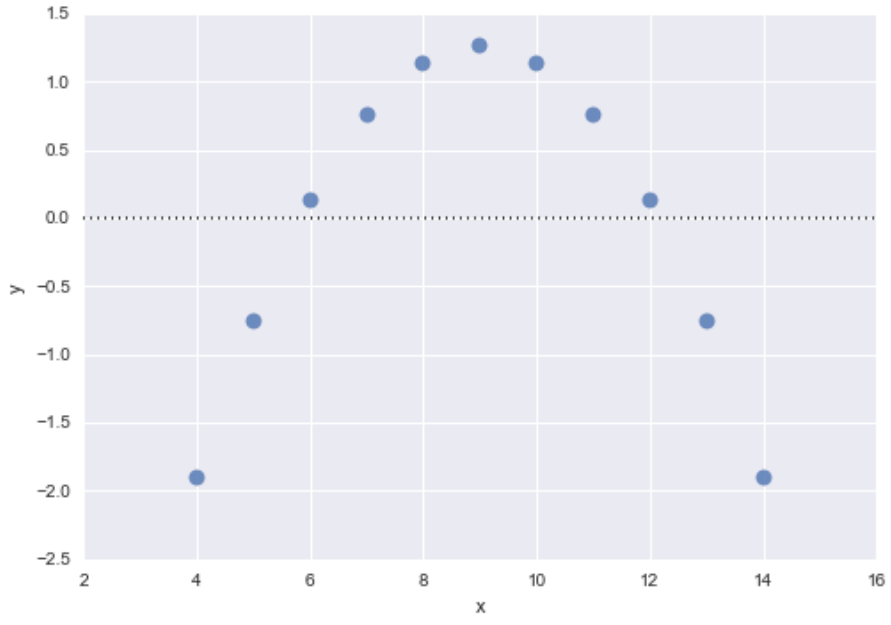
然后绘制出每个观测的残差值。理想情况下，这些值应随机散落在 y = 0 附近：

```
sns.residplot(x="x", y="y", data=anscombe.query("dataset == 'I'"),
              scatter_kws={"s": 80});
```



如果误差中有结构，它表明，简单线性回归是不合适的：

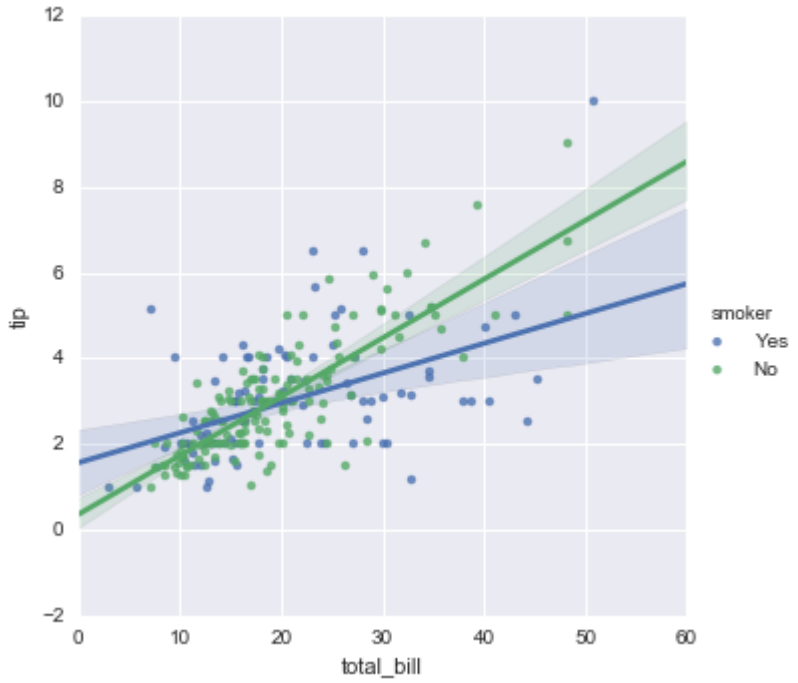
```
sns.residplot(x="x", y="y", data=anscombe.query("dataset == 'II'"),
              scatter_kws={"s": 80});
```



条件变量

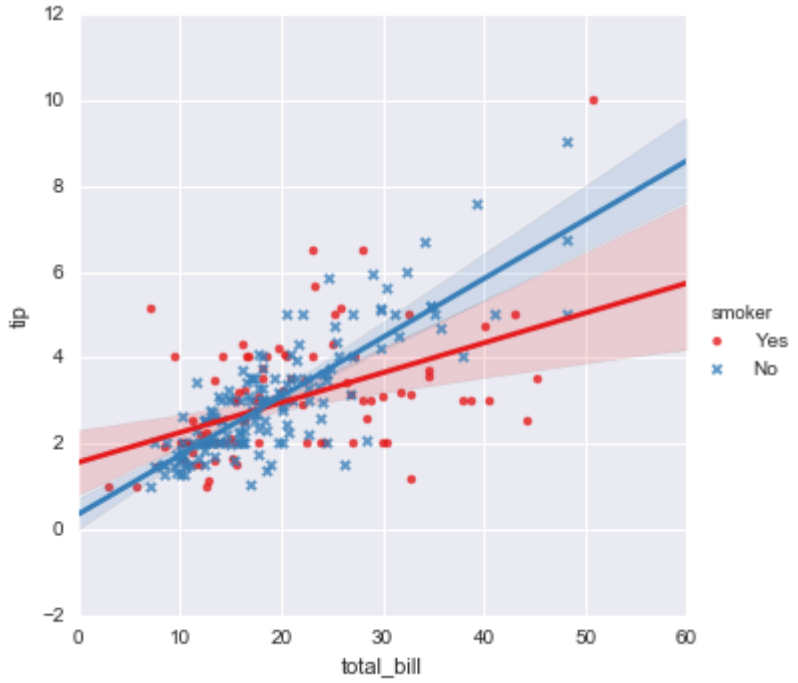
前述图显示了探索成对变量之间关系的许多方法。然而，通常情况下，一个更有趣的问题是“如何将这两个变量之间的关系的改变作为第三个变量的函数呢？”这是 regplot()和 lmplot()之间的差异。regplot()总是显示一个单一的关系，lmplot()结合 regplot()与 facetgrid 提供一个简单的接口来显示一个分面线性回归图，允许你探索这三个附加分类变量的相互作用。分离出这种关系的最好方法是绘制在相同坐标轴上的两个水平，并使用颜色来区分它们：

```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips);
```



除了颜色，更好地使用不同的散点图标记代替黑白色是可能的。你对使用颜色拥有更多的全权控制：

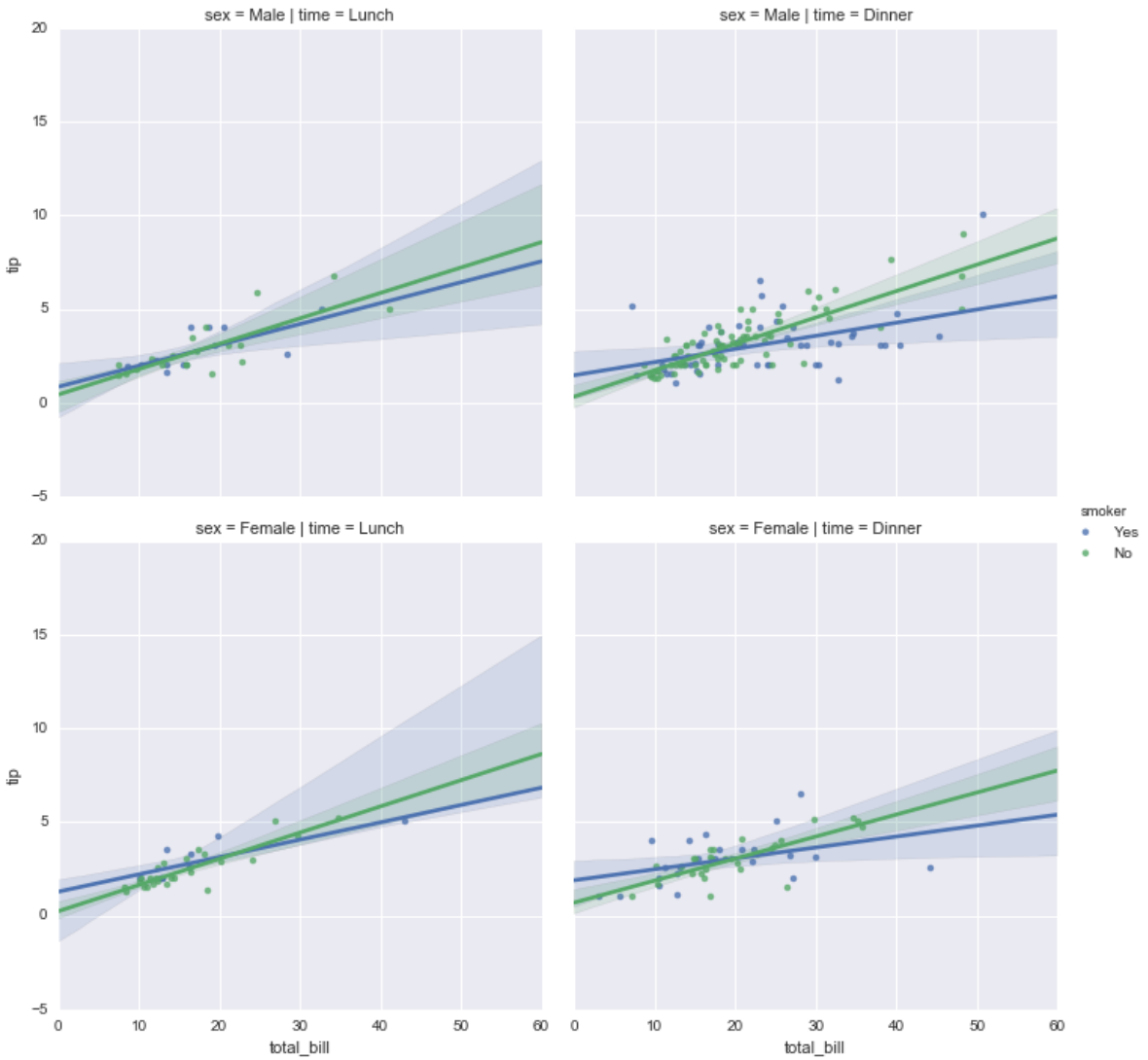
```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=tips,
           markers=["o", "x"], palette="Set1");
```



要添加另一个变量，您可以绘制多个“分面”，其中变量每一个水平出现在网格的行或列中：

```
sns.lmplot(x="total_bill", y="tip", hue="smoker", col="time", data=tips);
```


```
sns.lmplot(x="total_bill", y="tip", hue="smoker",
           col="time", row="sex", data=tips);
```



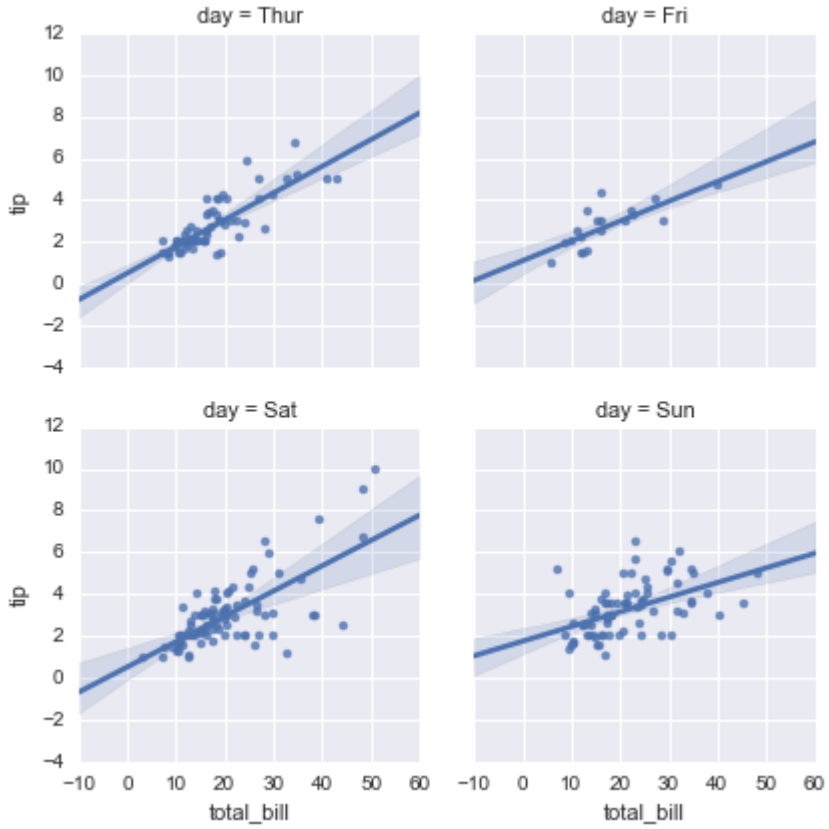
控制图的大小和形状

之前我们注意到 **regplot()**和 **lmplot()**制作的默认图表看起来是一样的，但是坐标轴却是不同的大小和形状。这是因为 **regplot** 函数是绘制在明确的坐标轴上的“坐标轴层级”的函数。这意味着你能使用多面板图形，并且精确控制回归图画在哪里。如果没有提供坐标轴，函数将使用当前活跃坐标轴，这就是为什么默认图和大部分的其他 matplotlib 函数产生的图有相同的大小和形状。你可以通过创建一个图形对象来控制大小。

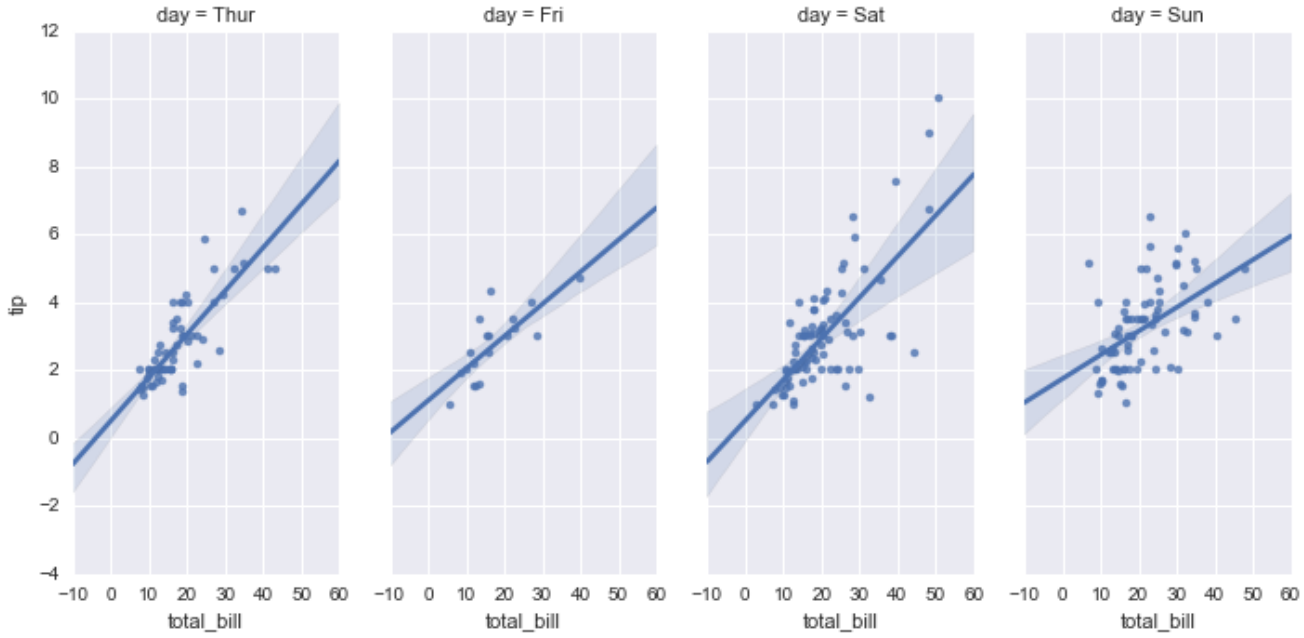
```
f, ax = plt.subplots(figsize=(5, 6))
sns.regplot(x="total_bill", y="tip", data=tips, ax=ax);
```

相比之下，`lmplot()`通过 `FacetGrid` 接口使用 `size` 和 `aspect` 两个参数来控制大小和形状，这两个参数的值可以应用到图表中的每一个分面中，而不是应用到整体的图形中。

```
sns.lmplot(x="total_bill", y="tip", col="day", data=tips,
                    col_wrap=2, size=3);
```



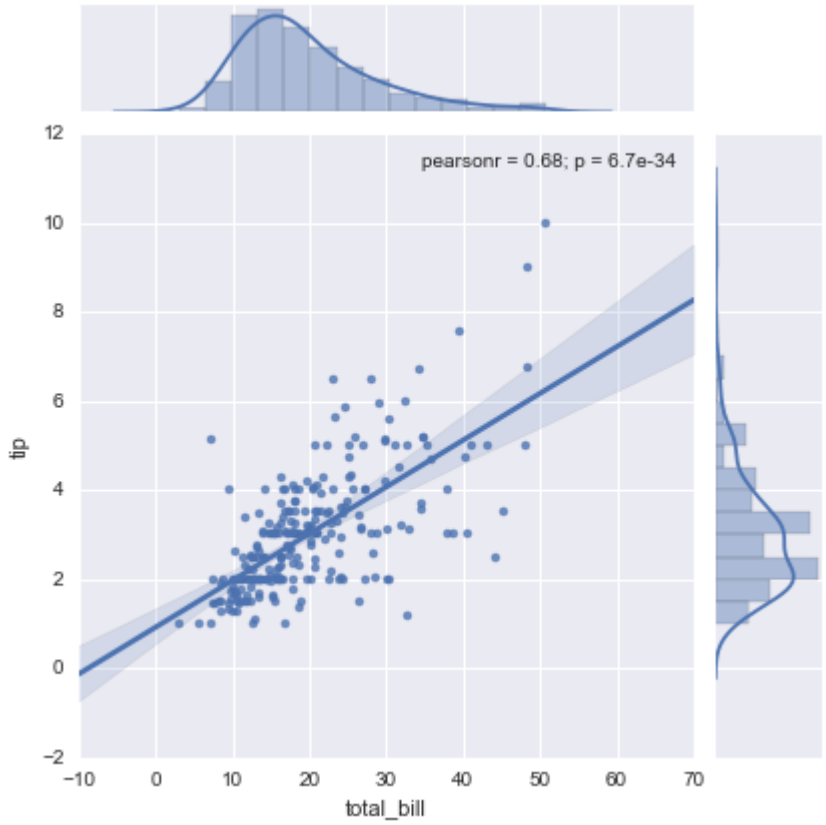

```
sns.lmplot(x="total_bill", y="tip", col="day", data=tips,
           aspect=.5);
```



在其它环境下绘制回归

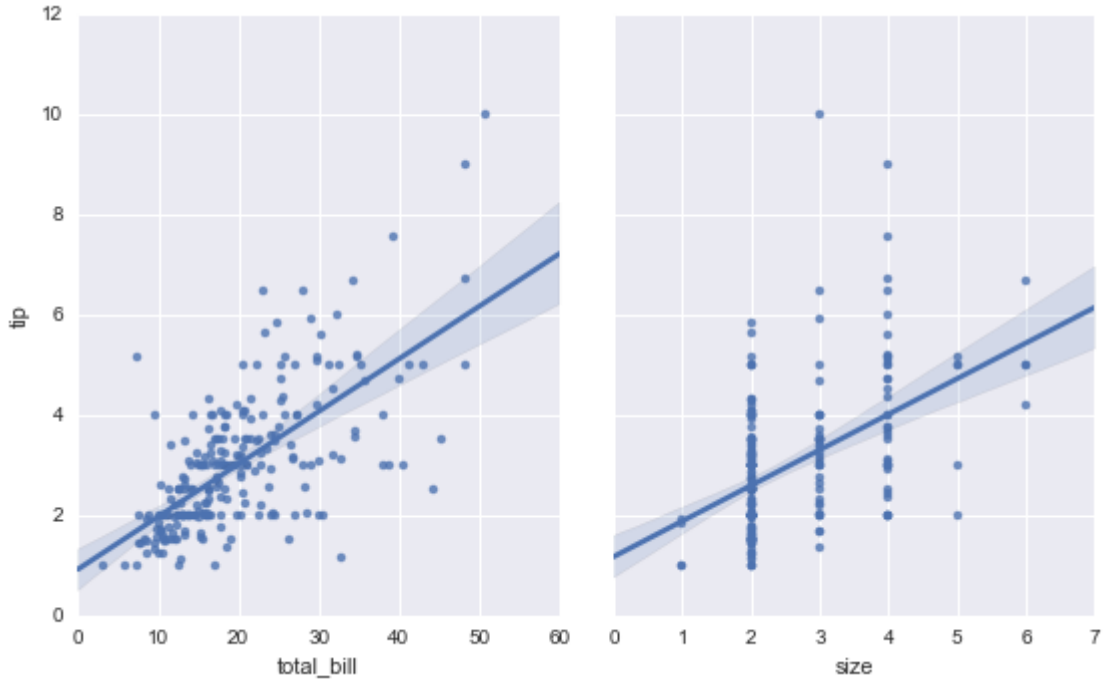
seaborn 函数使用 **regplot()**在更大更复杂的环境中。我们在分布教程开始就介绍过 **jointplot()**函数，除了先前讨论的绘图样式之外，**jointplot()**能通过传递参数 kind="reg"使用 **regplot()**展示拟合联合坐标轴的线性回归。

```
sns.jointplot(x="total_bill", y="tip", data=tips, kind="reg");
```



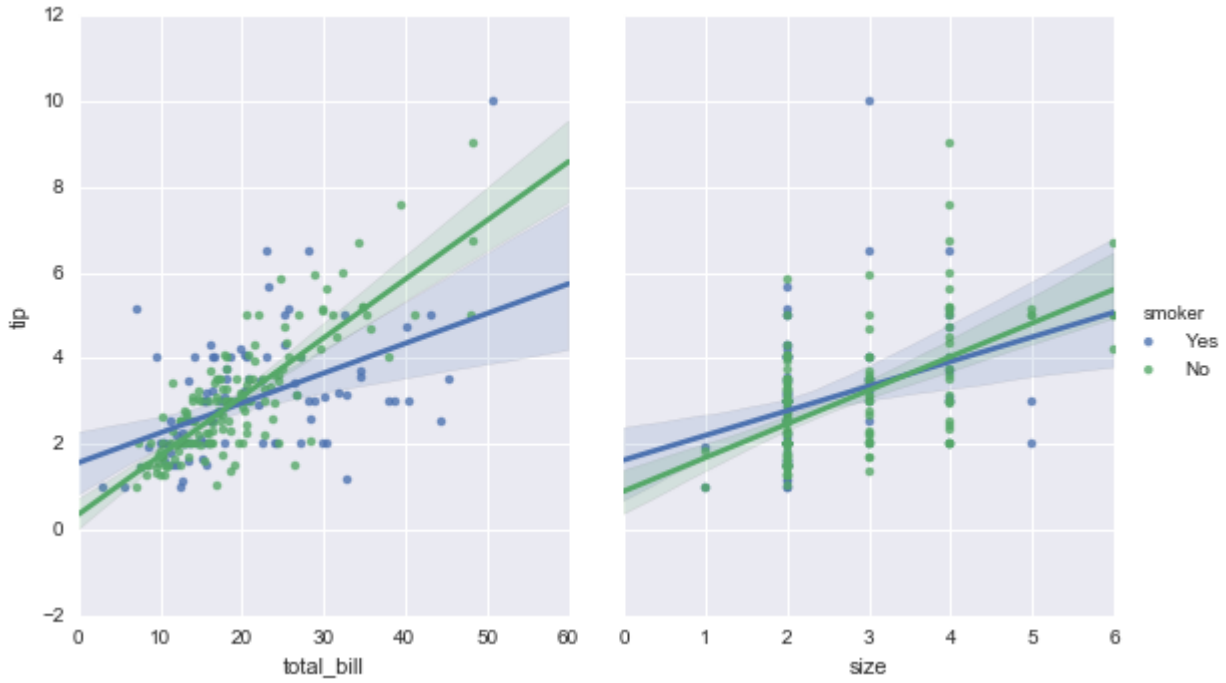
使用带 kind="reg"的 **pairplot()**函数，结合 **regplot()**和 **PairGrid** 展示数据集中的变量关系。注意这个与 **lmplot()**不同，在下图中，两个坐标轴在第三个变量的两个水平条件下并不会显示相同的关系，而是，**PairGrid()**常用来显示在数据集中不同成对的变量下的多样关系。

```
sns.pairplot(tips, x_vars=["total_bill", "size"], y_vars=["tip"],
             size=5, aspect=.8, kind="reg");
```



类似 **lmplot()**，但是不像 **jointplot()**，在附加类别变量条件下使用 **pairplot()**函数的内置参数 hue 。

```
sns.pairplot(tips, x_vars=["total_bill", "size"], y_vars=["tip"],
             hue="smoker", size=5, aspect=.8, kind="reg");
```



用分类数据进行绘图

我们之前已经了解如何使用散点图和回归模型拟合来可视化两个变量之间关系以及如何改变其他分类变量的水平。然而，如果你感兴趣的主要变量之一是分类数据该怎么做？在这种情况下，散点图和回归模型方法并不合适。有几个选项，然而，对于这种关系的可视化，我们将在教程里进行讨论。

将 seaborn 的分类图分成三组是很有用的，这显示分类变量每个水平下的每个观测数据，每个观测分布的抽象表示，那些静态统计估计显示集中化趋势和置信区间。第一类包括 swarmplot()和 stripplot()函数，第二类包括 boxplot()和 violinplot()函数和第三类 barplot()和 pointplot()函数。这些函数都共享一个用于接收数据的基本数据 API 接口，尽管每个都有指定参数，当应用于数据时控制可视化上具体部分。

就像 regplot()和 lmplot()之间的关系，在 seaborn 中有低阶和高阶方法制作分类图。上述函数在向具体 matplotlib 轴绘图时实现低阶功能。而 factorplot()则属于较高阶实现，其结合 FaceGrid 在图像面板网络上应用分类图的绘制。

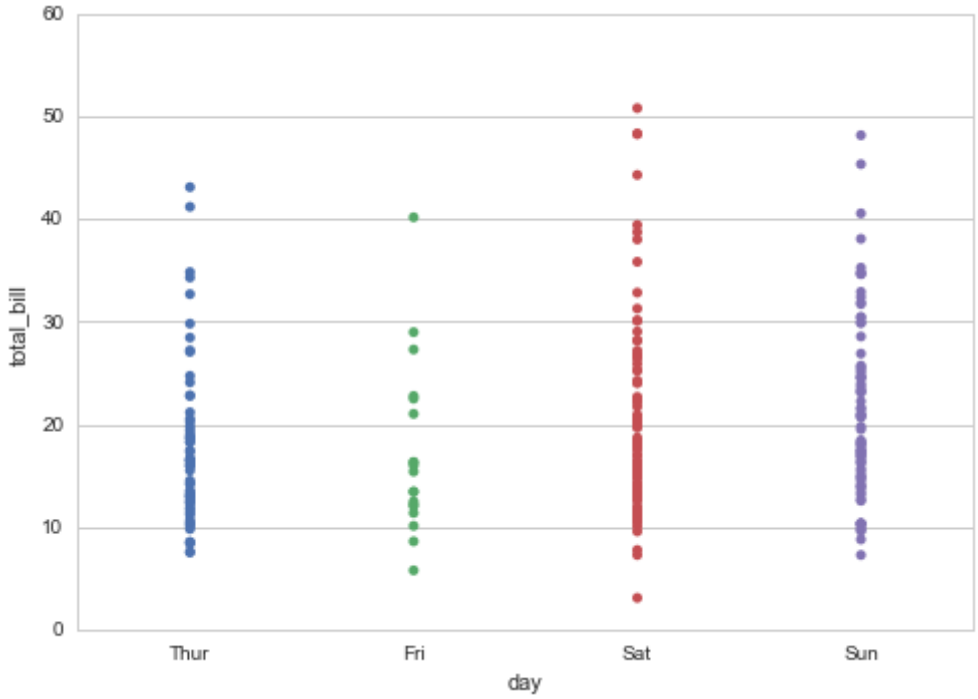
最好最方便的是用数据框在规整的格式下调用这些函数，上述低阶功能也能应用在宽格式的数据框格式或者简单的观测向量。见下面的例子。

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="whitegrid", color_codes=True)
np.random.seed(sum(map(ord, "categorical")))
titanic = sns.load_dataset("titanic")
tips = sns.load_dataset("tips")
iris = sns.load_dataset("iris")
```

分类散点图

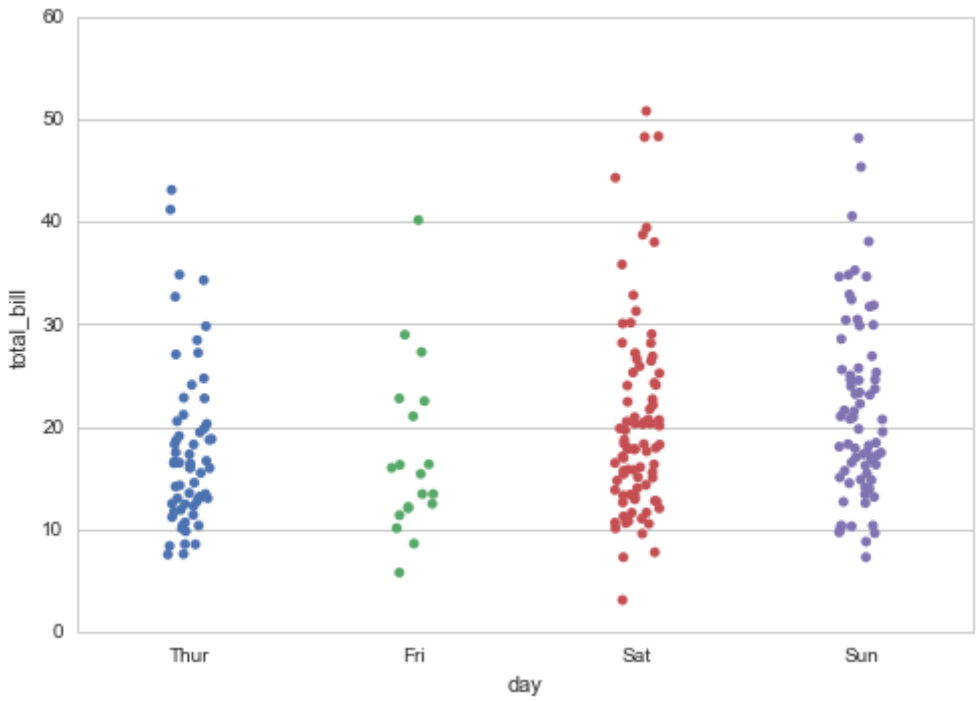
展示某些定量变量的值最简单的方法是使用 stripplot()，当变量是散点数据时其能够生成变量的散点图

```
sns.stripplot(x="day", y="total_bill", data=tips);
```



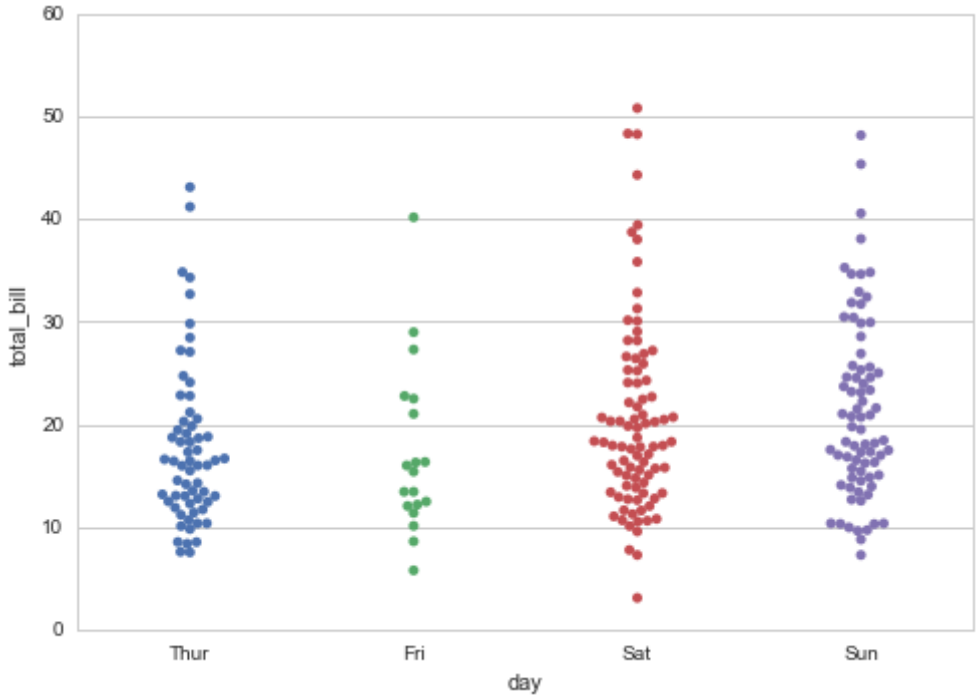
在这带形图中，散点图的点通常会重叠，很难看清数据的完整分布情况，一个简单的解决方案是调整分类轴的位置（仅沿分类轴的方向）使用一些随机抖动。

```
sns.stripplot(x="day", y="total_bill", data=tips, jitter=True);
```



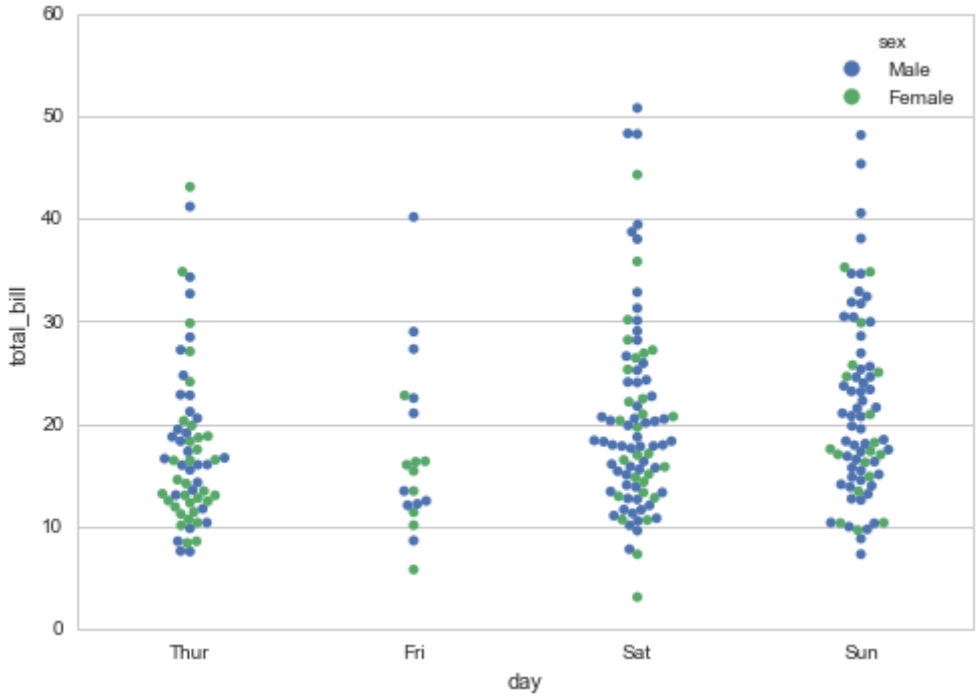
有个不同的方法是使用 swarmplot() 函数，能定位每个分类轴上的散点，通过算法来避免这些点的重叠：

```
sns.swarmplot(x="day", y="total_bill", data=tips);
```



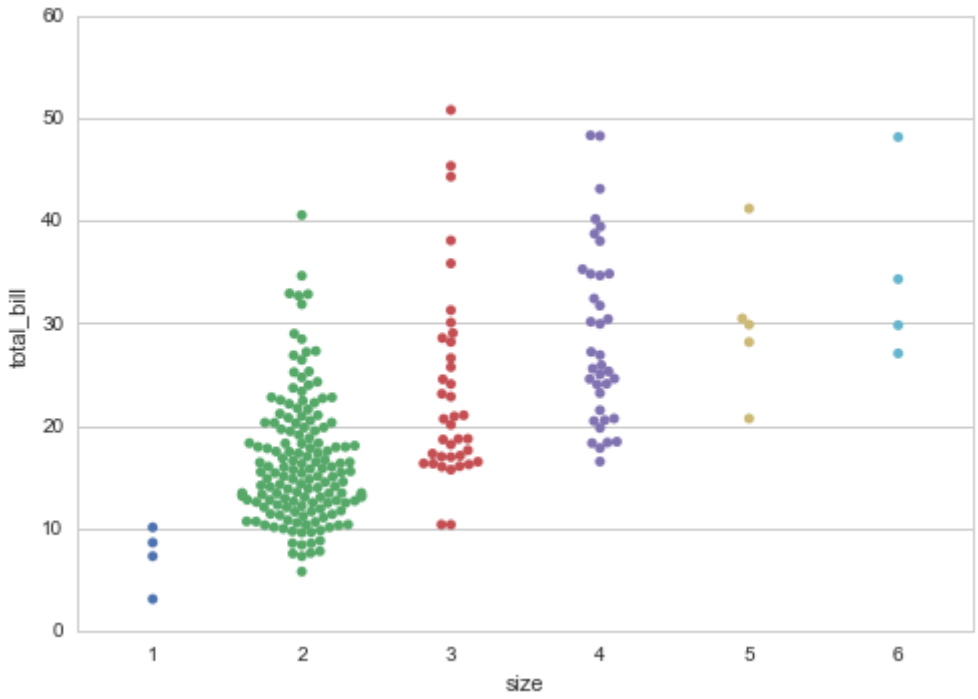
它也可以添加一个使用色调参数的嵌套的分类变量。在分类轴上颜色和位置本来是多余的，但现在每个散点图都能提供颜色和位置这两个信息：

```
sns.swarmplot(x="day", y="total_bill", hue="sex", data=tips);
```



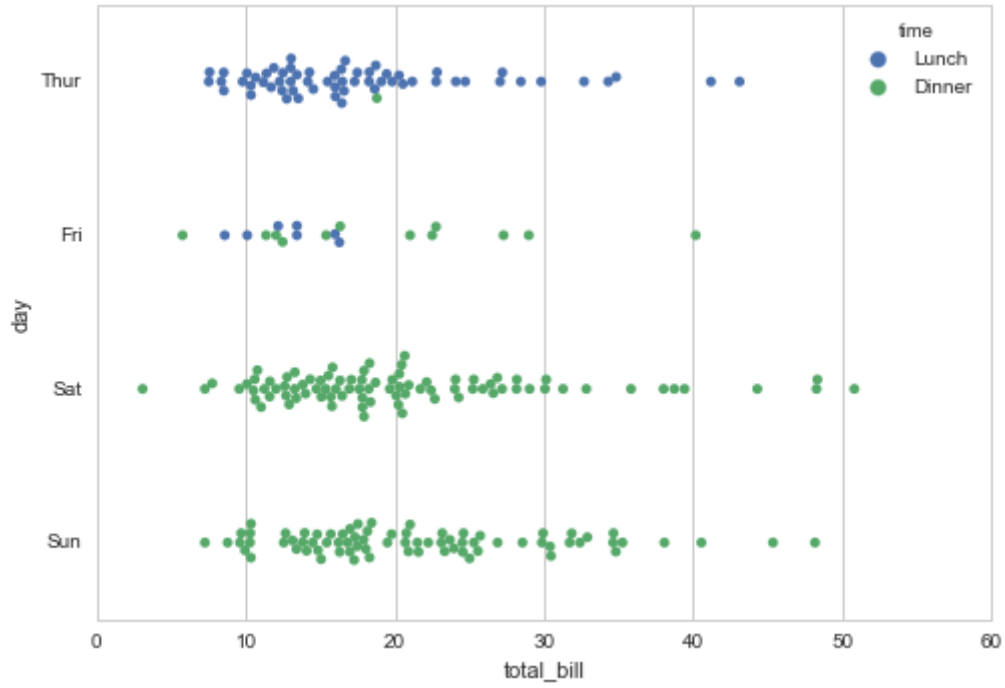
在一般情况下，seaborn 分类绘图函数会从数据中试图推断类别顺序。如果你的数据是 pandas 的分类数据格式，那么该类别的默认顺序可以直接在 seaborn 中设置，而其他数据类型、字符串类型的分类将按照它们在数据框中出现的次序来进行绘制，但是分类为数值型的数据会排序。

```
sns.swarmplot(x="size", y="total_bill", data=tips);
```



这些图有助于把分类变量放在纵轴（这非常有用尤其当分类名称很长或有很多分类变量时）。你能量使用 orient 关键字强制方向，但通常情况下方向可以通过 X 和 Y 变量的数据类型来推断：

```
sns.swarmplot(x="total_bill", y="day", hue="time", data=tips);
```



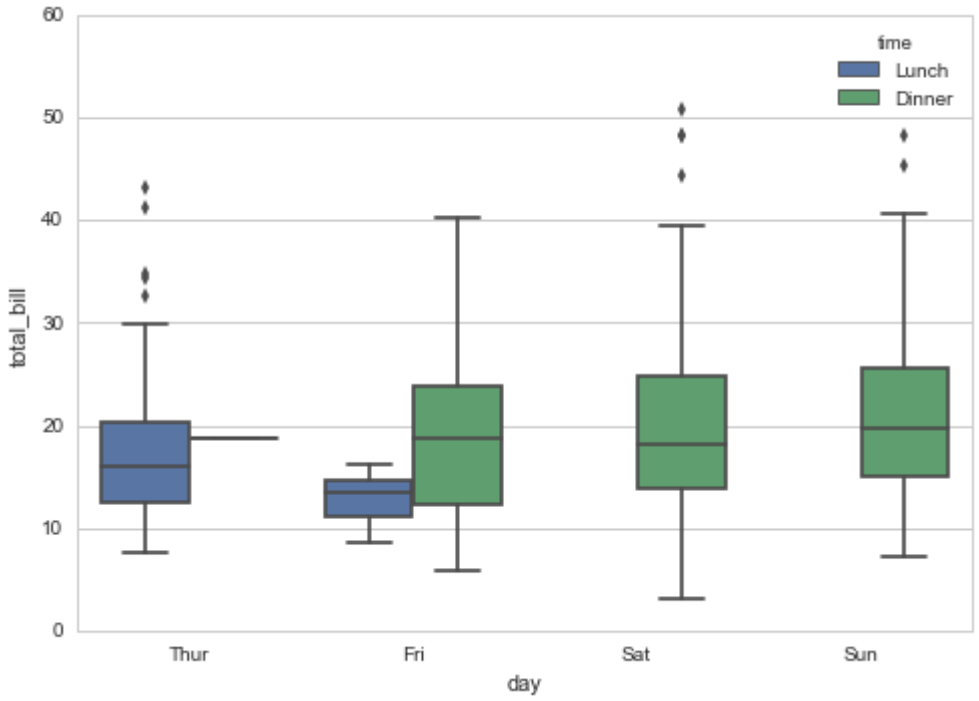
分类观测的分布

在某点上，该分类散点图方法在提供每个分类值的分布信息将变得有限。这有许多总结信息的方法使得比较跨分类水平变得容易。这些我们在章节讨论过的一般化方法，就用在我们需要快速比较跨多个分布的情况。

箱形图

首先是大家熟悉的 boxplot()。该图显示了分布的三个四分位值和极值，盒须延伸至与上下四分位数距离 1.5 个 IQR 的点上，观测点落在范围之外的将独立显示。重要的是，这意味着箱形图的每个值对应的实际的数据观测：

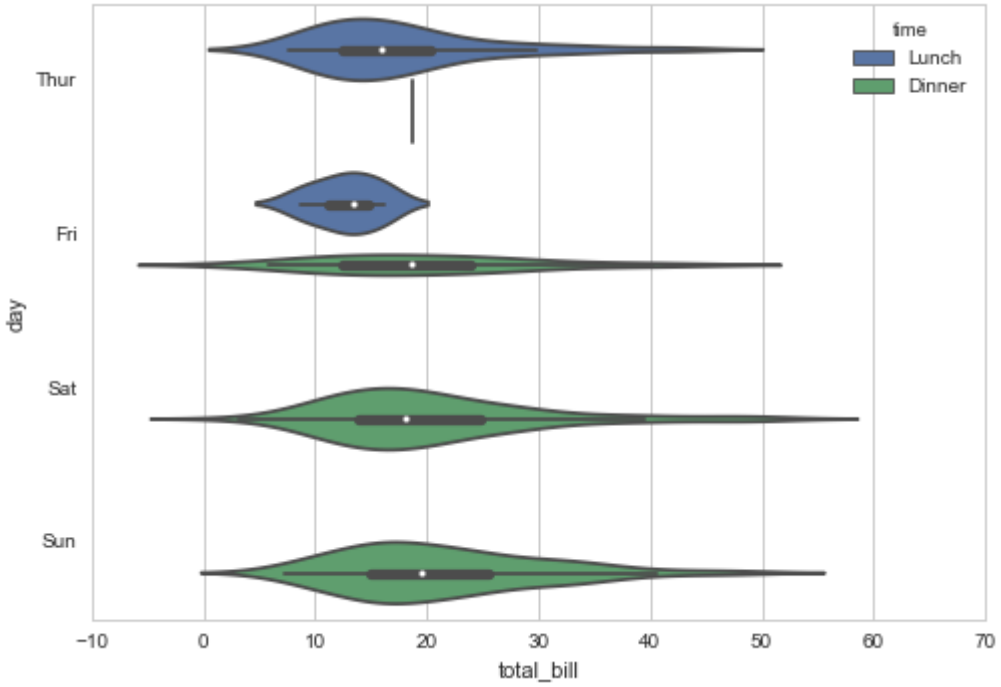
```
sns.boxplot(x="day", y="total_bill", hue="time", data=tips);
```



提琴图

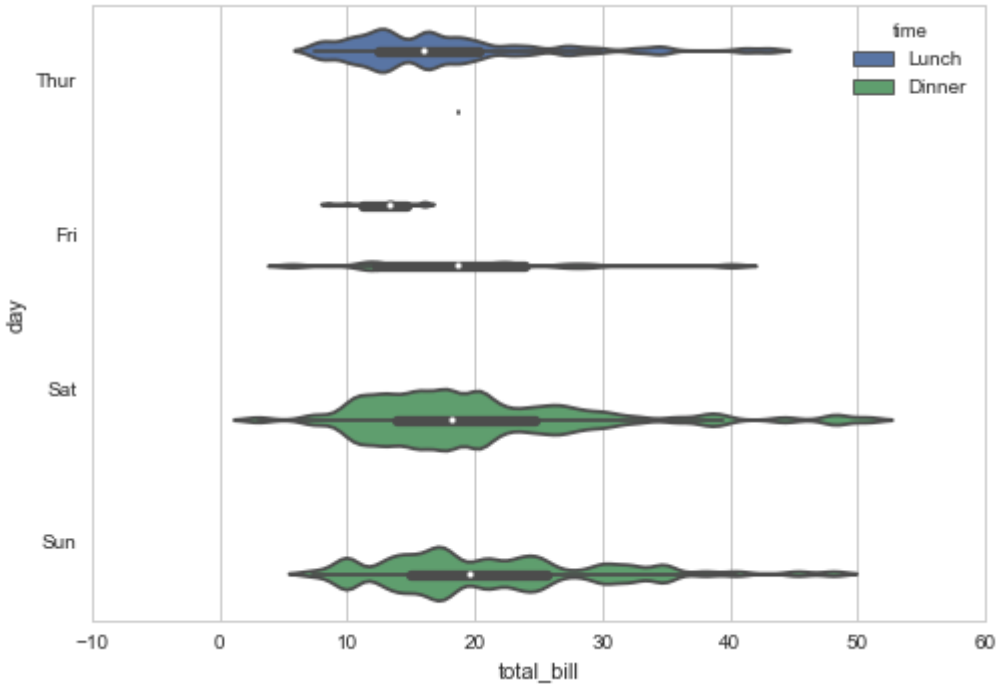
另一种方法是 violinplot()，它结合了在分布教程中描述过的箱形图和核密度估计过程：

```
sns.violinplot(x="total_bill", y="day", hue="time", data=tips);
```



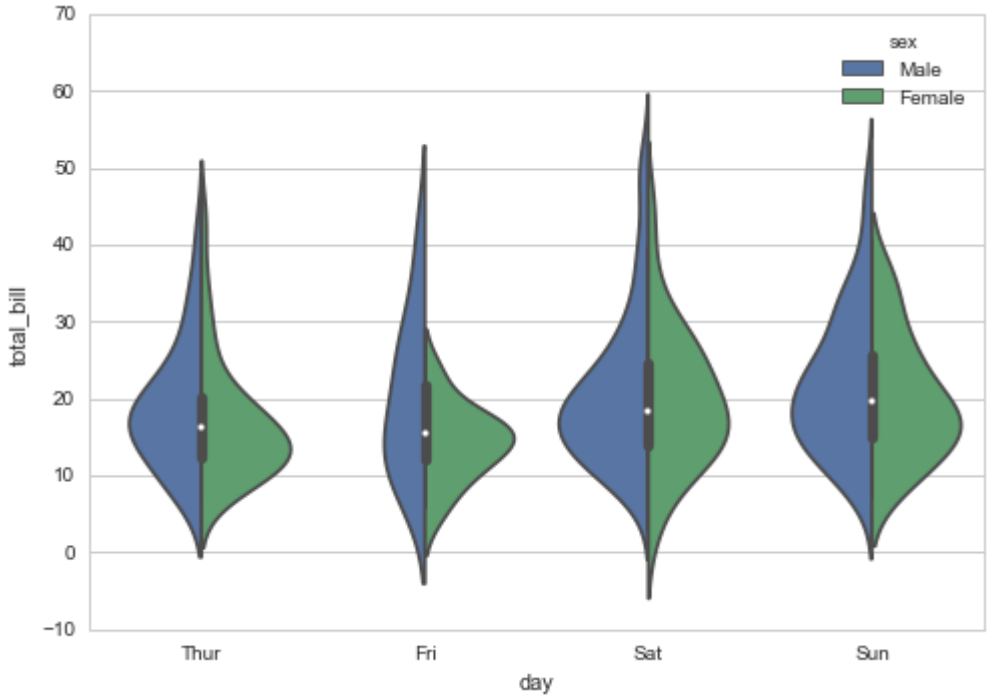
这种方法使用内核密度估计提供值分布更好的描述。此外，从箱线图的四分位和盒须值在小提琴里显示。因为 violinplot 使用了 KDE，有可能需要调整其它参数，增加本来简单箱形图的复杂性。

```
sns.violinplot(x="total_bill", y="day", hue="time", data=tips,
               bw=.1, scale="count", scale_hue=False);
```



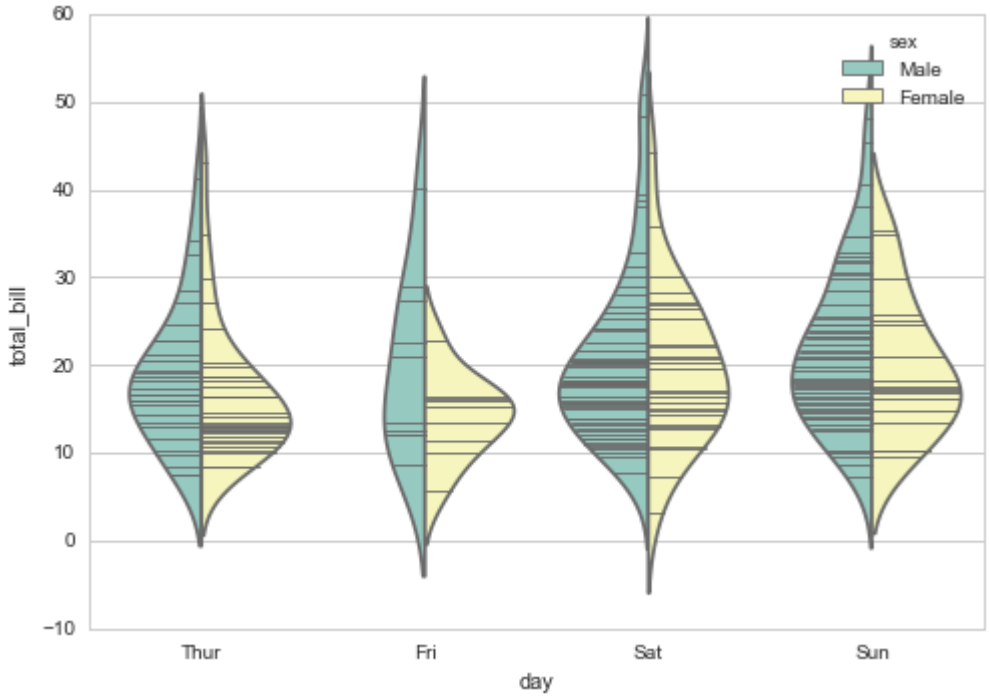
当色调参数只有两个水平时，分割小提琴是有可能的，它允许有效利用空间。

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips, split=True);
```



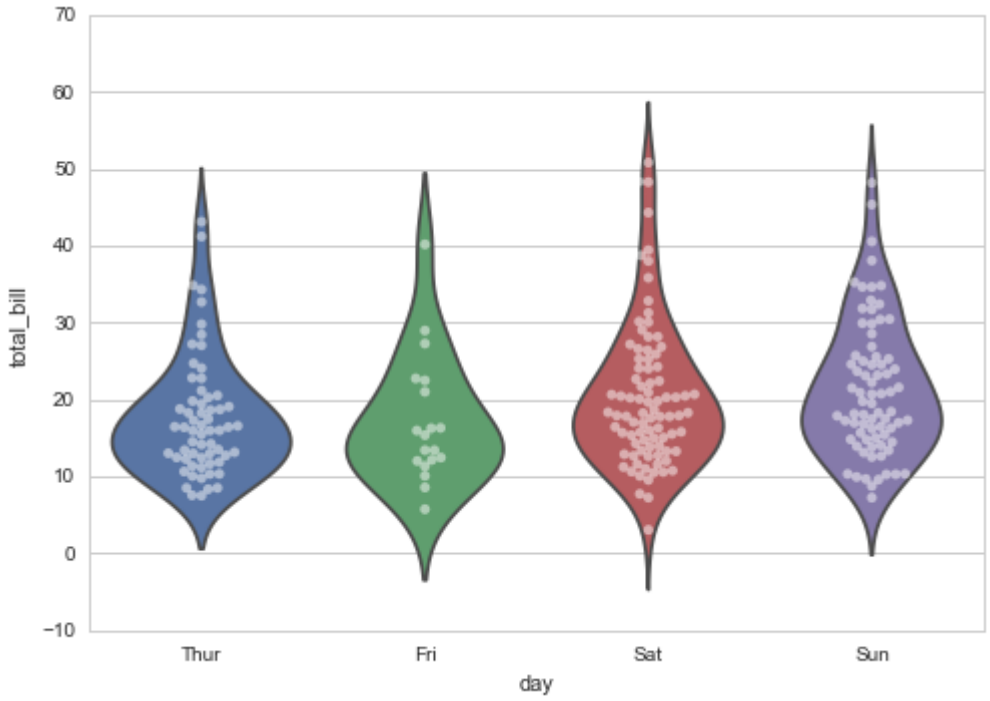
最后，在小提琴内部绘制时有几个选项可用，包括显示各自观测而不是箱形图值的摘要。

```
sns.violinplot(x="day", y="total_bill", hue="sex", data=tips,
               split=True, inner="stick", palette="Set3");
```



结合swarmplot() 或 swarmplot() 和 violinplot() 或boxplot()可以显示各自观测和分布摘要：

```
sns.violinplot(x="day", y="total_bill", data=tips, inner=None)
sns.swarmplot(x="day", y="total_bill", data=tips, color="w", alpha=.5);
```



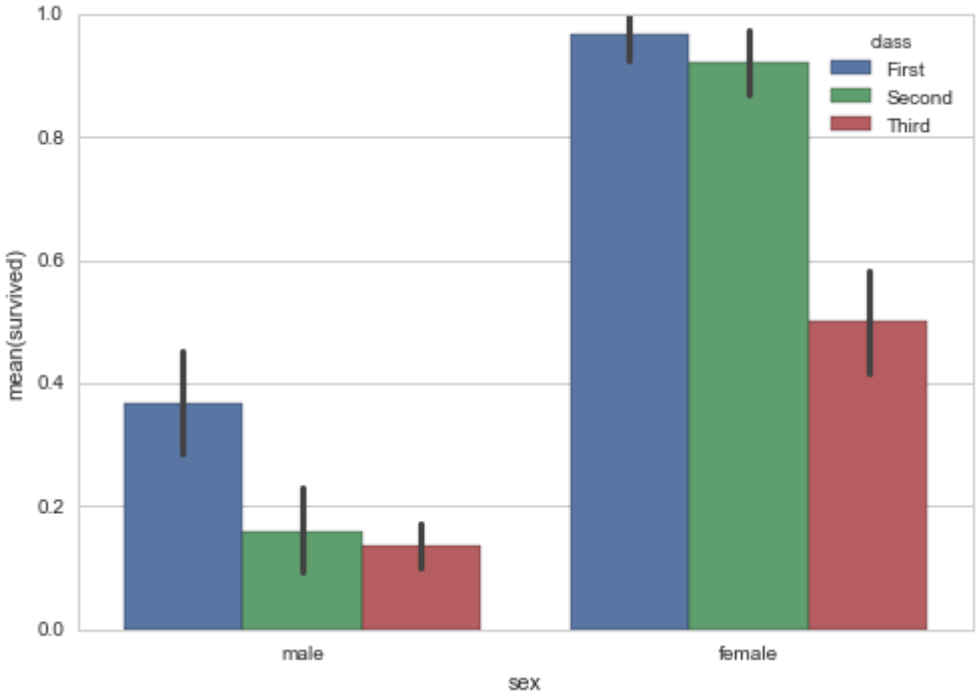
分类统计估计

通常情况下，除了显示每个类别中分布，你可能想要显示这些值的集中趋势。 Seaborn 有两个主要的方式来显示这些信息，但是重要的是，这些函数的基础 API 跟前面所讨论的是一样的。

条形图¶

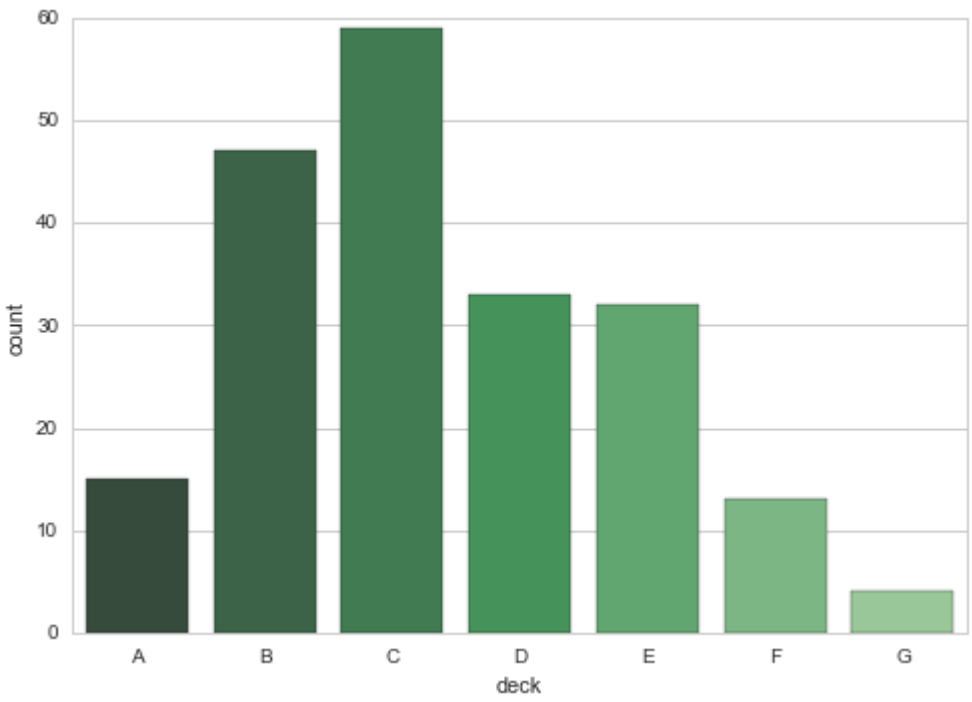
条形图为大家所熟知，在 Seaborn，barplot()函数使用默认方法操作数据集并显示任意估计。当在每个分类下存在多元观测时，它也可以使用 bootstrapping 围绕估计来计算置信空间和绘制误差线。

```
sns.barplot(x="sex", y="survived", hue="class", data=titanic);
```



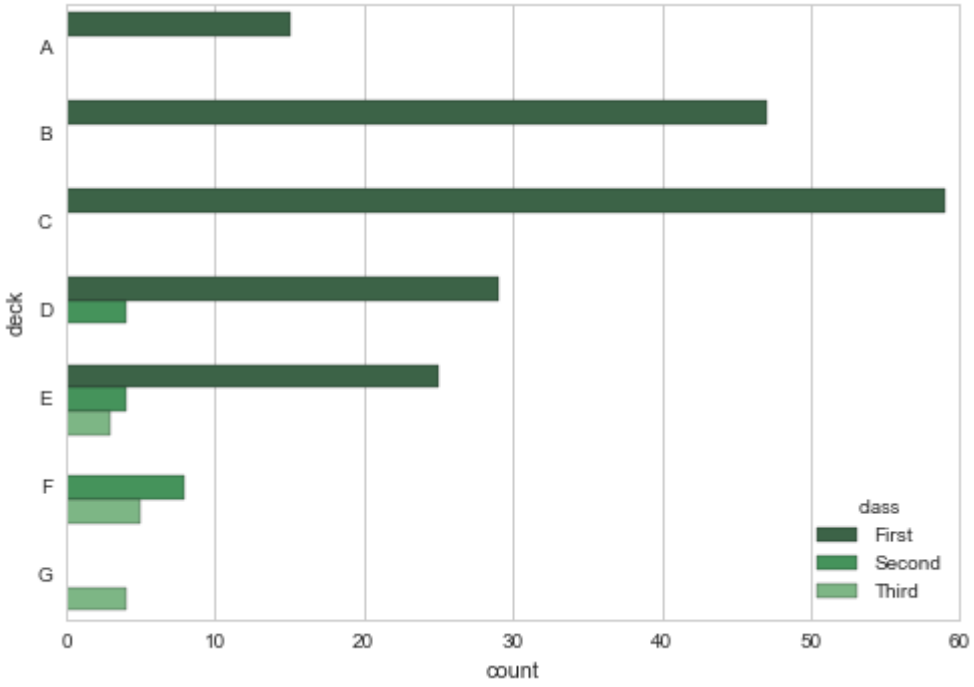
一个条形图特殊的例子，你要显示每个分类的观测数量而不是计算第二个变量的统计信息。这类似于分类直方图，而不是定量，变量。在 seaborn，用 countplot()函数很容易做到。

```
sns.countplot(x="deck", data=titanic, palette="Greens_d");
```



Barplot()和 countplot()可以来调用上面讨论过的选项，其它更细节的将在每个函数的详细文档中论述。

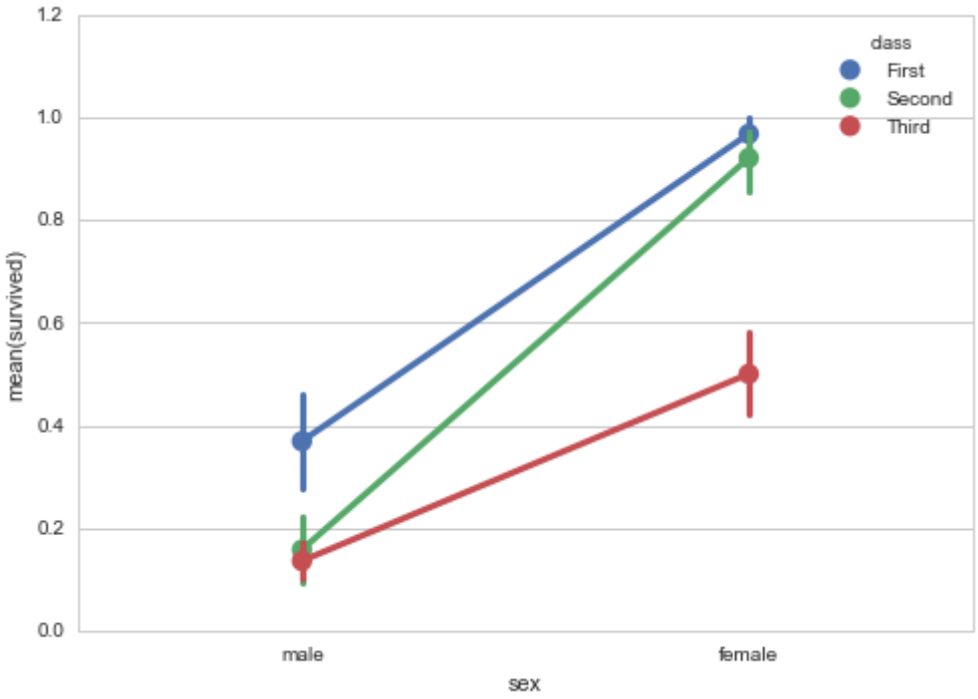
```
sns.countplot(y="deck", hue="class", data=titanic, palette="Greens_d");
```



点图

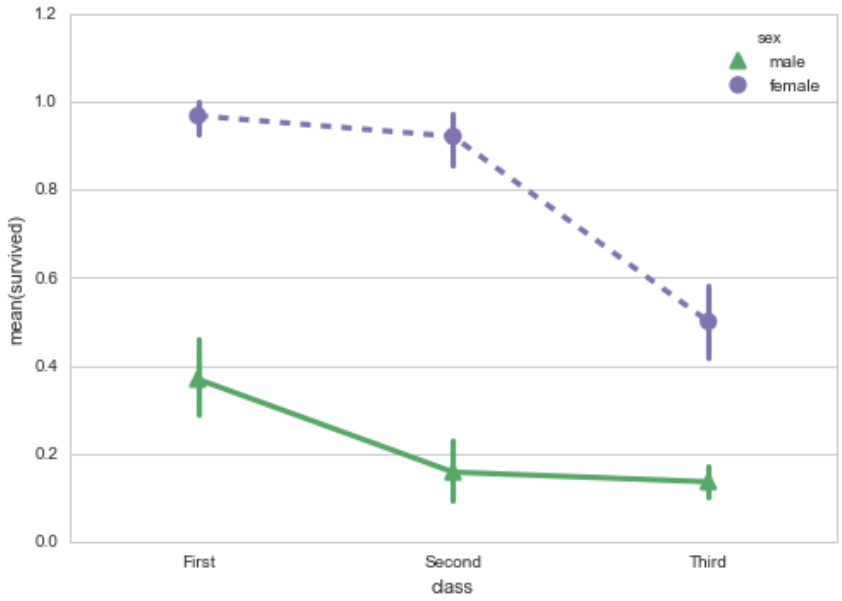
还有一个同样信息的可视化样式由 pointplot()函数提供。该函数用其它轴的高度对估计值编码，但不显示完整的条形，只是画出点估计和置信空间。此外，点图连接相同色调分类下的点。这使得它很容易看到主要关系的改变作为第二个变量的函数，因为人眼很适于在不同的斜率进行选择。

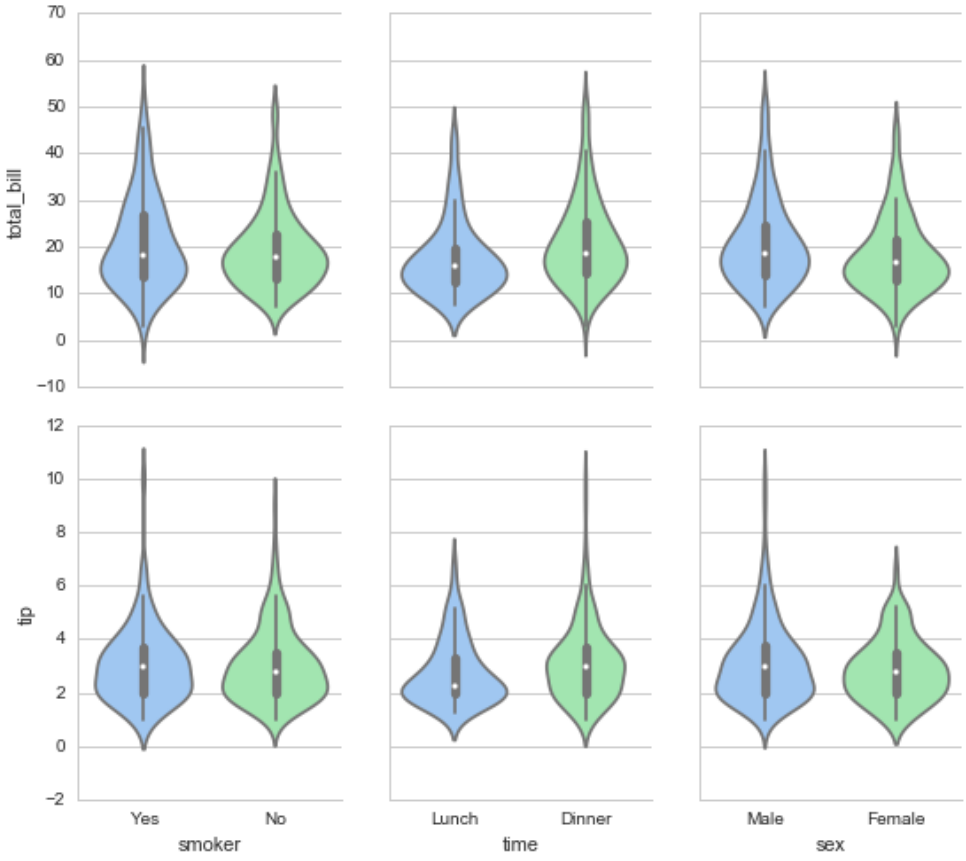
```
sns.pointplot(x="sex", y="survived", hue="class", data=titanic);
```



要画好黑白色图像，对色调分类水平用不同的标记和线型样式是好主意。

```
sns.pointplot(x="class", y="survived", hue="sex", data=titanic,
              palette={"male": "g", "female": "m"},
              markers=["^", "o"], linestyles=["-", "--"]);
```

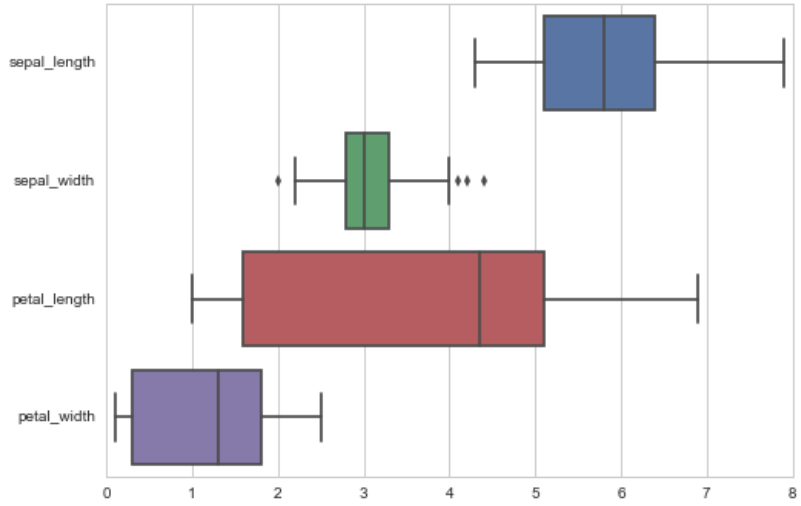




绘制宽型数据

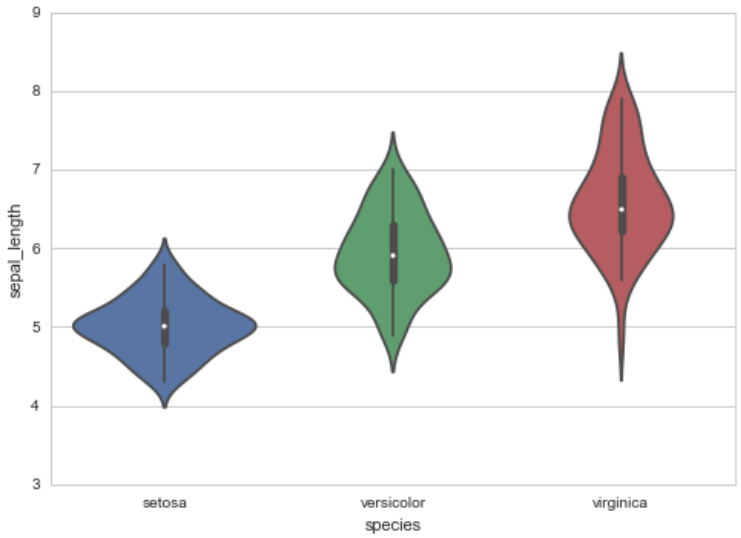
虽然我们推荐使用“长型”的或“规整型”的数据，但是这些函数也可以用于各种不同格式的“宽型”的数据，包括 pandas 的 DataFrame 或 numpy 的二维数组。这些对象可以直接传递 data 参数：

```
sns.boxplot(data=iris, orient="h");
```



此外，除了 DataFrame 中的变量，Pandas 的向量或者 numpy 的对象也可以在上述的函数中使用，例如：

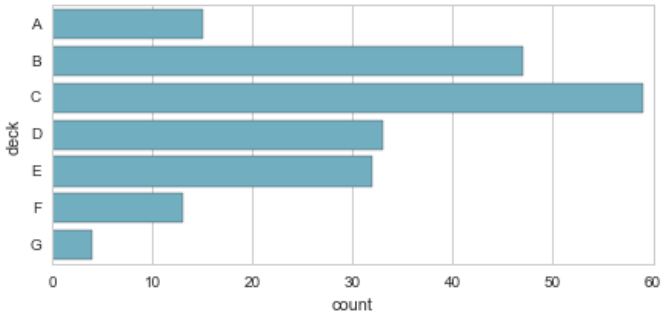
```
sns.violinplot(x=iris.species, y=iris.sepal_length);
```



为了控制上述函数绘制图形的尺寸和形状，你必须使用 matplotlib 的命令设置图形的大小和形状。当然，这也意味着这些图可以在多面板图形中与其它各种图形共存，例如：

```
f, ax = plt.subplots(figsize=(7, 3))
```

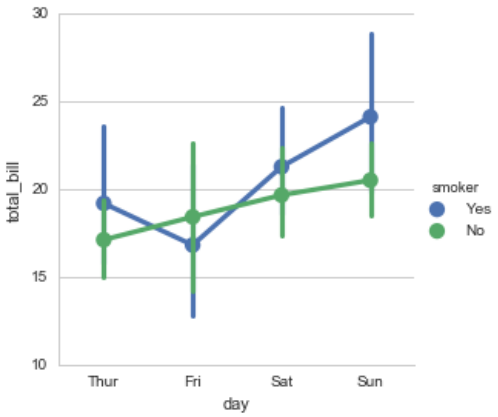
```
sns.countplot(y="deck", data=titanic, color="c");
```



绘制多面板分类图

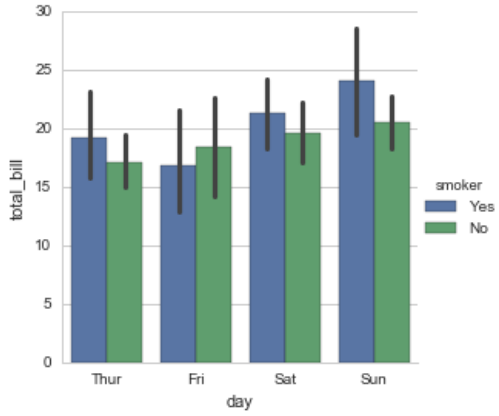
正如我们上面所提到的，在 seaborn 中一共有两种绘制分类图的方法。与回归绘图的对偶性相似，既可以采用前面介绍的方法进行绘制，同样也可以使用高级函数 factorplot()。该函数加入了 FacetGrid()增加检查大型数据结构图表中额外分类的能力。默认情况下，factorplot()产生一个 pairplot()：

```
sns.factorplot(x="day", y="total_bill", hue="smoker", data=tips);
```



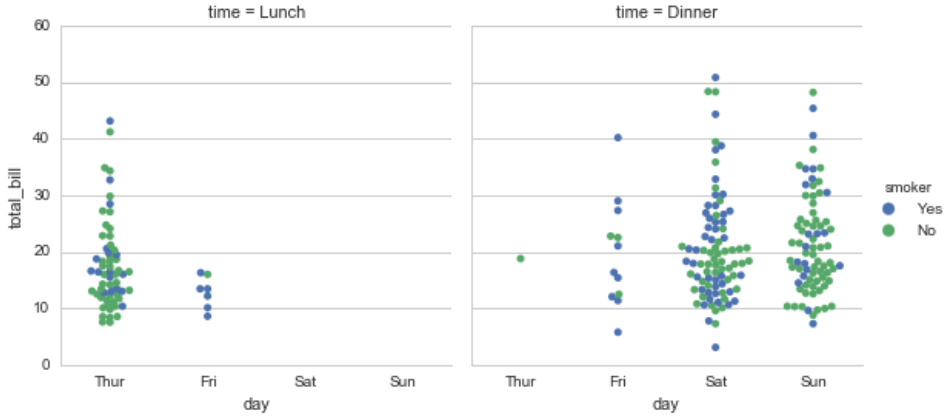
然而，kind 参数可以任意地选择前面讨论过的绘图方式，如：

```
sns.factorplot(x="day", y="total_bill", hue="smoker", data=tips, kind="bar");
```



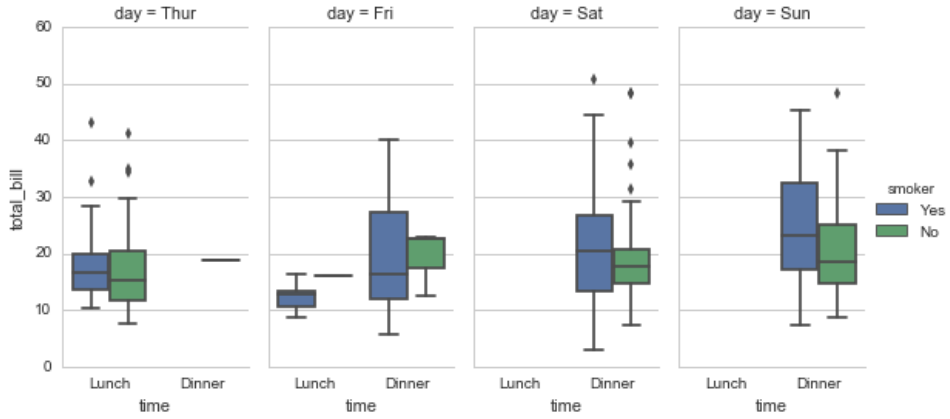
使用 factorplot()函数的主要好处是，它易于“分面”图和研究其他分类变量的角色：

```
sns.factorplot(x="day", y="total_bill", hue="smoker",
               col="time", data=tips, kind="swarm");
```



可以绘制任意类型的图形。由于 FacetGrid 的工作原理，你必须指定 size 和 aspect 参数（用于每个分面）来改变图形的大小和形状，例如：

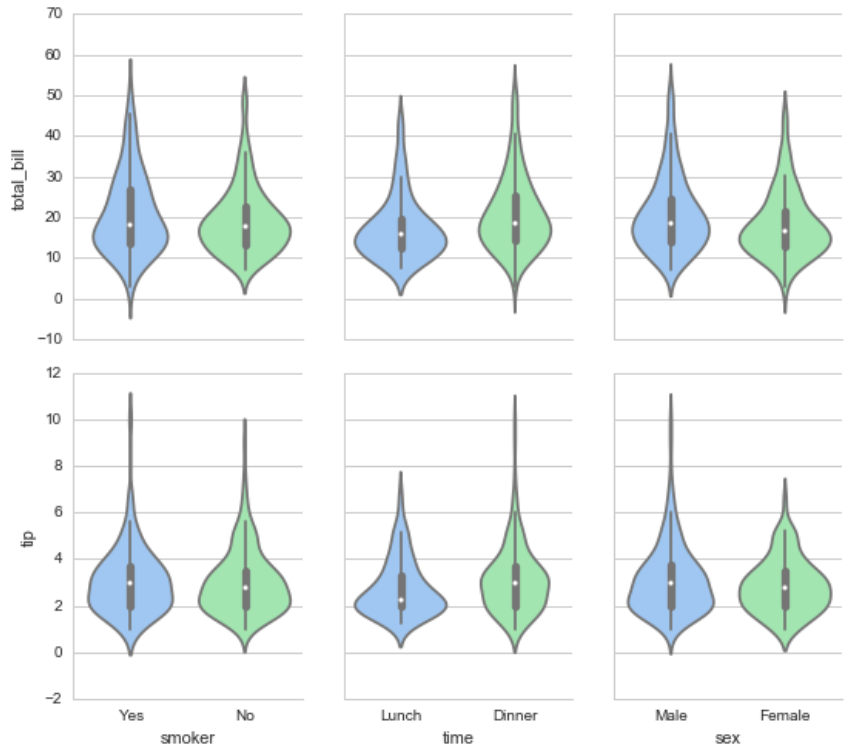
```
sns.factorplot(x="time", y="total_bill", hue="smoker",
               col="day", data=tips, kind="box", size=4, aspect=.5);
```



注意你也可以通过 boxplot()和 FacetGrid 来进行绘图，这事也是很重要的。但是必须注意的是，在每个分面中，分类变量的顺序必须保持一致，也可以通过传入 Categorical 数据类型或者传入 order 和 hue_order 来让它们保持一致。

由于分类绘图函数保持了通用的 API，我们可以在更复杂的情况下使用它们。例如，它们可以和 PairGrid 一起使用来展示不同变量之间的分类关系：

```
g = sns.PairGrid(tips,
                 x_vars=["smoker", "time", "sex"],
                 y_vars=["total_bill", "tip"],
                 aspect=.75, size=3.5)
g.map(sns.violinplot, palette="pastel");
```



结构化网格

数据感知网格的绘制

当在中等维度探索数据时，一个有用的方法是在数据集的不同子集上绘制多个相同的图的实例。这种技术有时被称为“网格”，或“格子”的绘制，它与“小多组图”思想有关。它允许受众者快速提取大量的复杂数据的信息。matplotlib 对多轴制图提供良好支持；Seaborn 在此基础上直接连接数据结构和绘图结构。

为了使用这些特性，数据必须放在 pandas 数据框中，而且必须使用所谓的 Hadley Whickam 所说的紧凑数据形式。简而言之，这意味着数据框应该结构化诸如每列存在且每行是一个观测。

对于高级使用，您可以直接使用本教程中所讨论的对象，它将提供最大的灵活性。一些 Seaborn 函数（如 lmplot()，factorplot()，和 pairplot()）在后台也使用它们。不同于其他的 Seaborn 函数，“轴一层”吸附到特定的（可能已经存在）没有另外操作 matplotlib 轴图，这些更高级别的函数能创建一个图形，当调用时通常如何设置更严格。

在某些情况下，无论是函数或对它们所依赖的类的构造函数将提供不同的接口属性如图尺寸参数，如 lmplot()你可以设置高度和高宽比的各个方面而不是图形的整体尺寸。使用其中一个对象的任何函数都会在绘图之后返回它，但大多数这些对象都有方便的方法来改变图的绘制，通常以更抽象和更容易的方式来进行。

```
%matplotlib inline
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib as mpl
import matplotlib.pyplot as plt
sns.set(style="ticks")
np.random.seed(sum(map(ord, "axis_grids")))
```

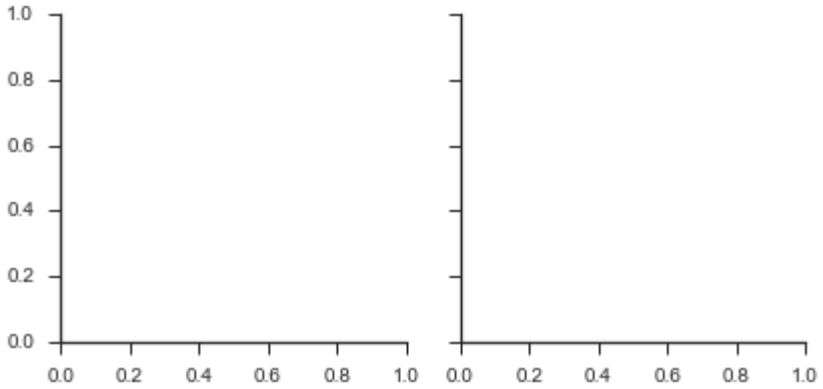
用 FacetGrid 构造子集

当你需要用数据子集分别可视化一个变量分布或者多变量关系，FaceGrid 类相当有用。FaceGrid 可以画出 3 个维度：行、列和色调。前两者与产生的数组的轴有明显对应，设想一下色调变量沿着深度轴作为第三维度，不同层次用不同颜色画出。

此类初始化 FaceGrid 对象，用数据库和变量名构成网格的行、列和色调维度。这些变量应该是分类或离散的，之后每一层的变量沿着轴的分面来使用。举个例子，我们想要检查在 tips 数据集中中饭和晚饭的差别。

另外，lmplot()和 factorplot()两者在内部使用这个对象，当他们完成后将返回这个对象，这样它可以进一步微调。

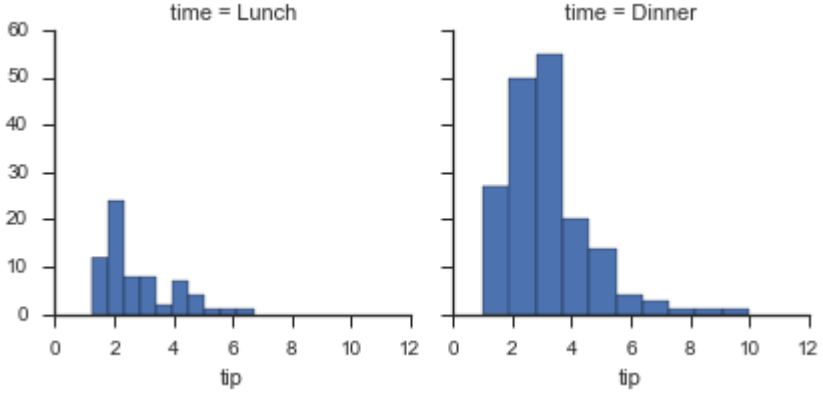
```
tips = sns.load_dataset("tips")

g = sns.FacetGrid(tips, col="time")
```



像这样设置 matplotlib 图和轴来初始化网格，但是现在还不能在这上面画东西。

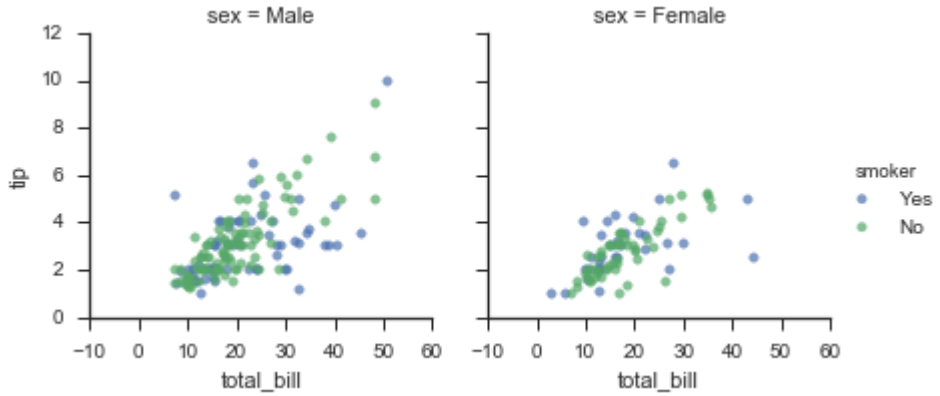
用 FaceGrid.map()主要的方法是在网格中可视化数据。提供画图功能和绘制数据框类型变量的命名。让我们看一下 tips 在这些子集中的分布，用直方图。

```
g = sns.FacetGrid(tips, col="time")

g.map(plt.hist, "tip");
```



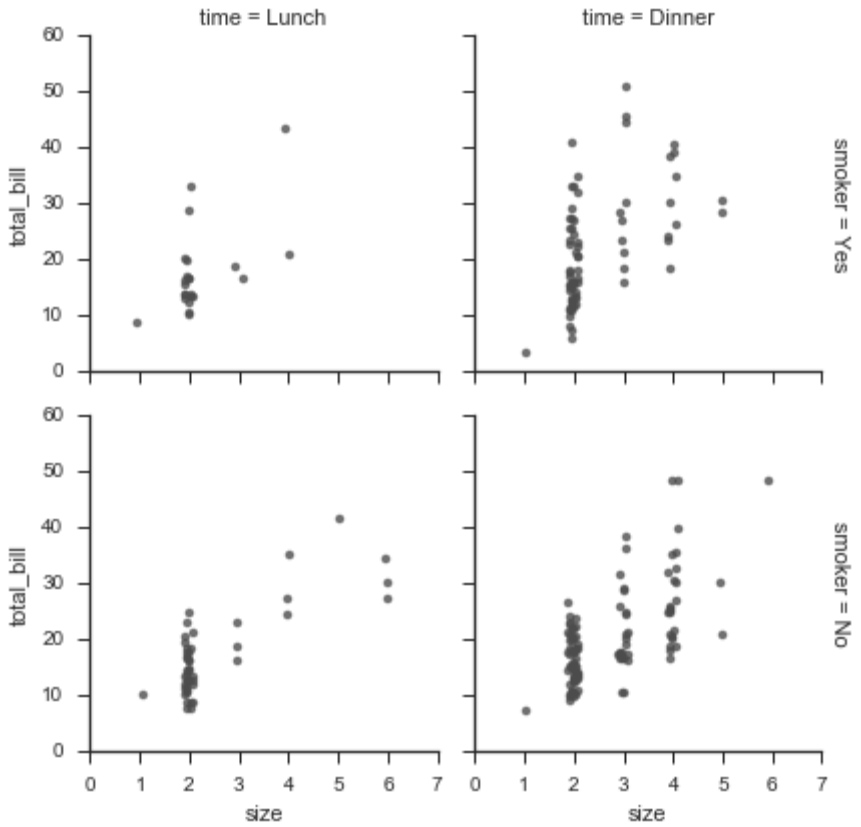
该功能可画出图和注解轴，希望一步生成到位。为了生成相关图，只要传递多个变量名。你也可以提供关键字参数，这样就可以传递到绘图功能。

```
g = sns.FacetGrid(tips, col="sex", hue="smoker")
g.map(plt.scatter, "total_bill", "tip", alpha=.7)
g.add_legend();
```



这里有一些选项用来控制网格外观，这样可以传递类构造式。

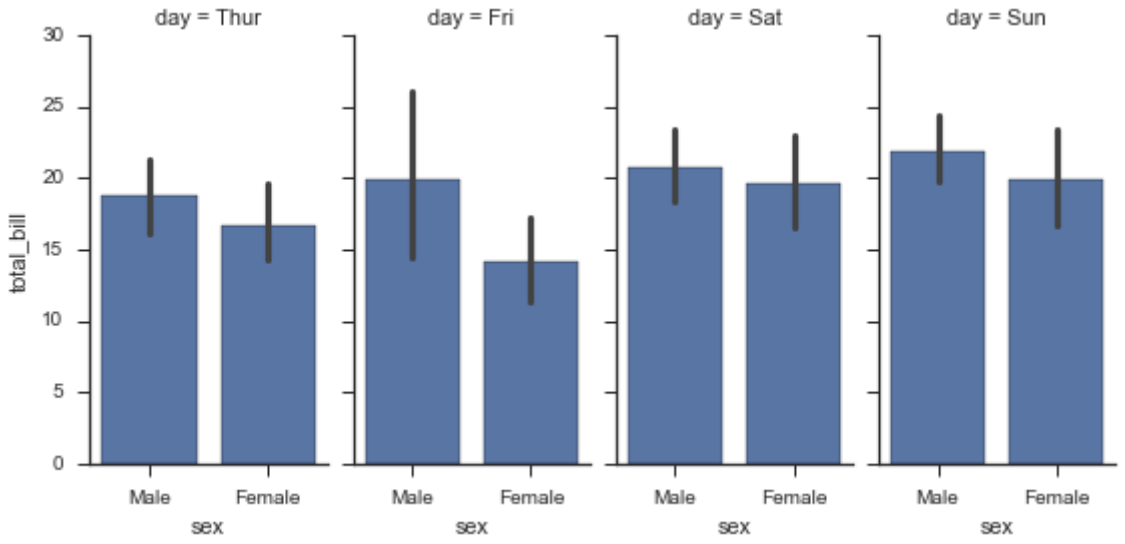
```
g = sns.FacetGrid(tips, row="smoker", col="time", margin_titles=True)
g.map(sns.regplot, "size", "total_bill", color=".3", fit_reg=False, x_jitter=.1);
```



即使边界标题不能正式支持 matplotlib API，在这些情况下可能不能正常运行。在特殊情况下，它目前不能用于放置在图外面的 legend。

图像大小可以通过每一分面的高度以及纵横比。

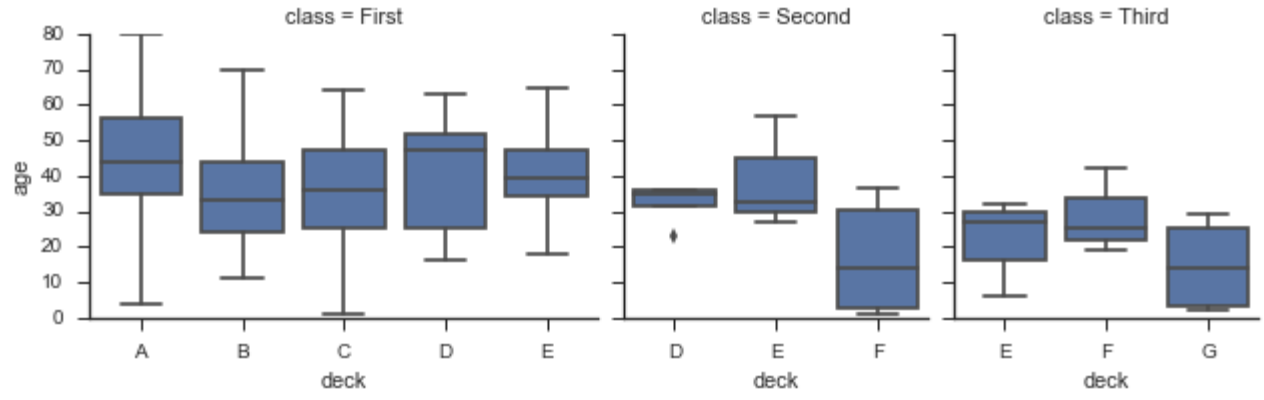
```
g = sns.FacetGrid(tips, col="day", size=4, aspect=.5)
g.map(sns.barplot, "sex", "total_bill");
```



大于 1.4 版本的 matplotlib，你可以传递参数用于 gridspec 模块，可用于通过增加其大小来吸引注意某一特定方面的。当可视化不均匀数量群体分布的数据集时，这是特别有用的。

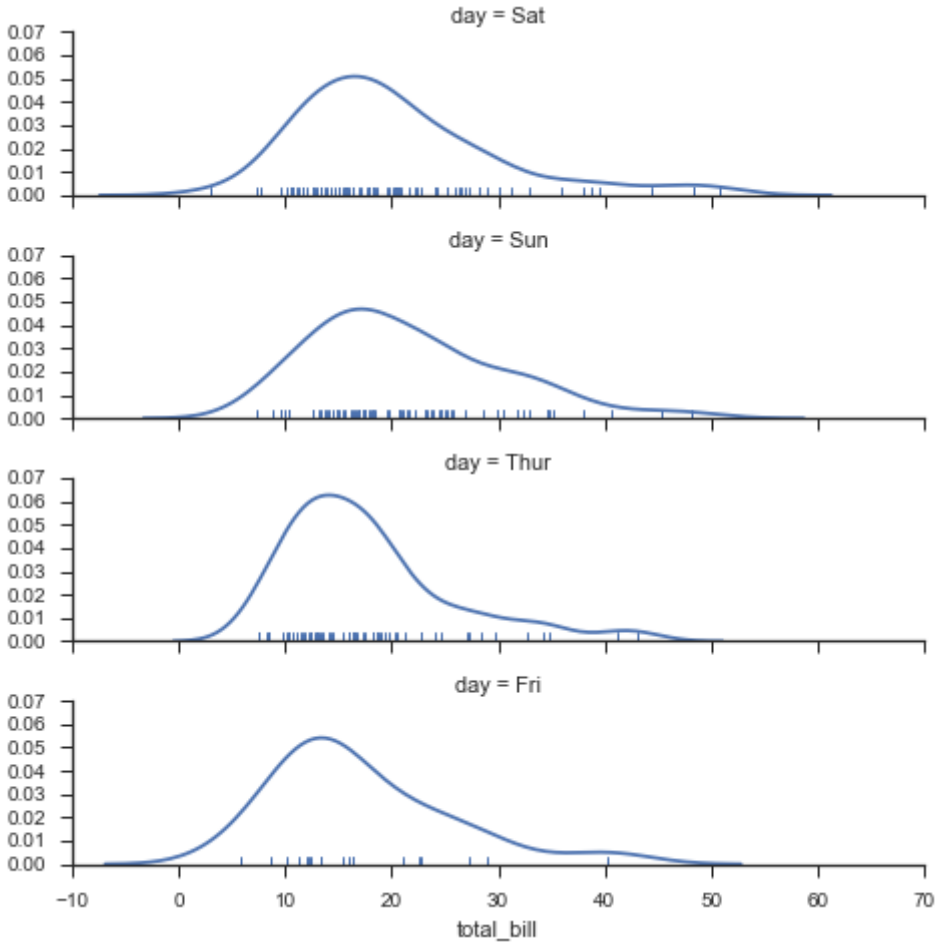
```
titanic = sns.load_dataset("titanic")
titanic = titanic.assign(deck=titanic.deck.astype(object)).sort("deck")
g = sns.FacetGrid(titanic, col="class", sharex=False,
                  gridspec_kws={"width_ratios": [5, 3, 3]})
g.map(sns.boxplot, "deck", "age");
```

```
/Users/mwaskom/anaconda/lib/python2.7/site-packages/ipykernel/__main__.py:2: FutureWarning: sort(columns=....) is deprecated, use sort_values(by=.....)
  from ipykernel import kernelapp as app
```



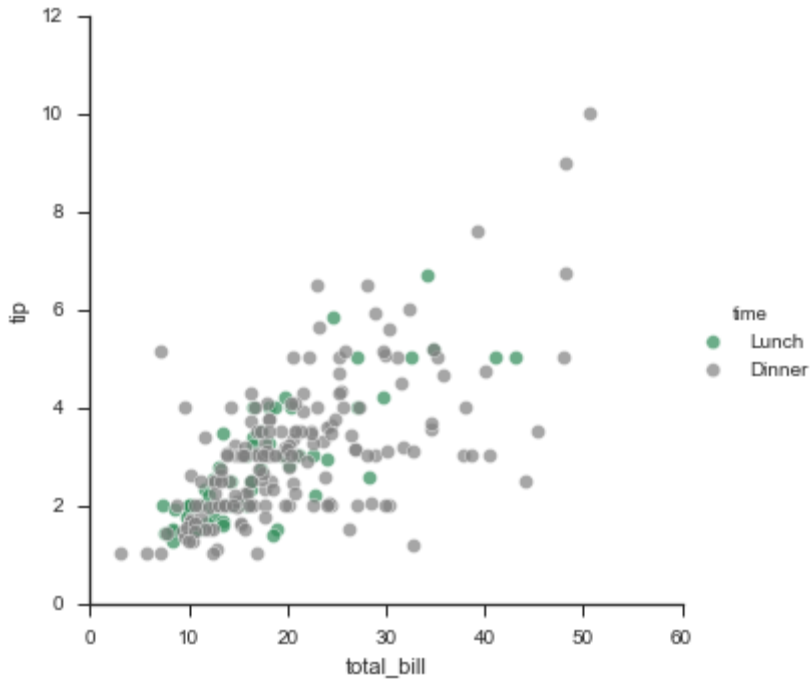
对于分面的排序是采数据框信息，如果这个变量曾定义分面的分类类型，那么就使用该分类排序。否则，这分面会依据分类水平来排序。这是可能的，但是，为了指定分面维度的顺序，要用合适的*_order参数。

```
ordered_days = tips.day.value_counts().index
g = sns.FacetGrid(tips, row="day", row_order=ordered_days,
                  size=1.7, aspect=4,)
g.map(sns.distplot, "total_bill", hist=False, rug=True);
```



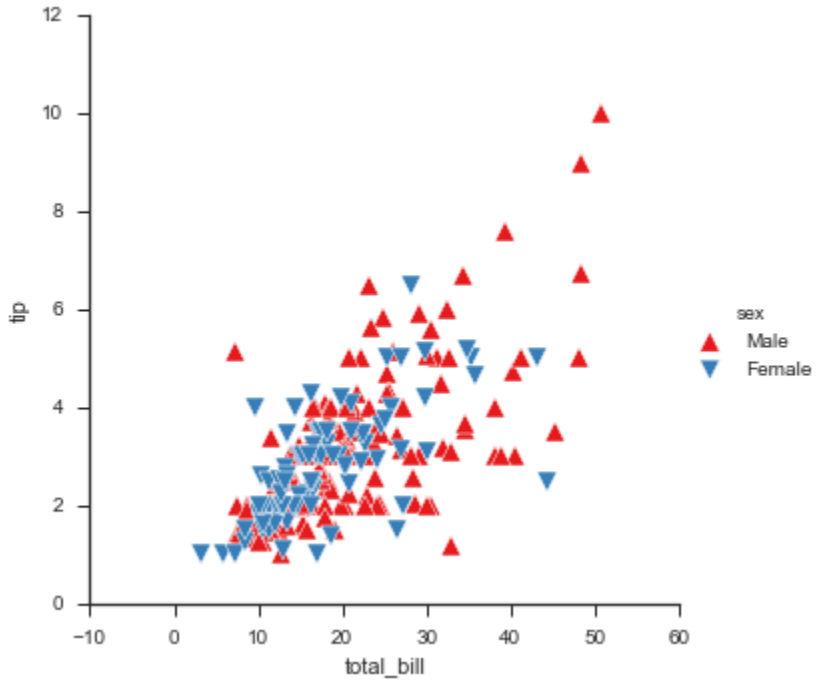
任何 seaborn 画板（如，有一些是 color_palette()可以提供的，你可以使用字典去映射色调中的值到现成的 matplotlib 色彩：

```
pal = dict(Lunch="seagreen", Dinner="gray")
g = sns.FacetGrid(tips, hue="time", palette=pal, size=5)
g.map(plt.scatter, "total_bill", "tip", s=50, alpha=.7, linewidth=.5, edgecolor="white")
g.add_legend();
```



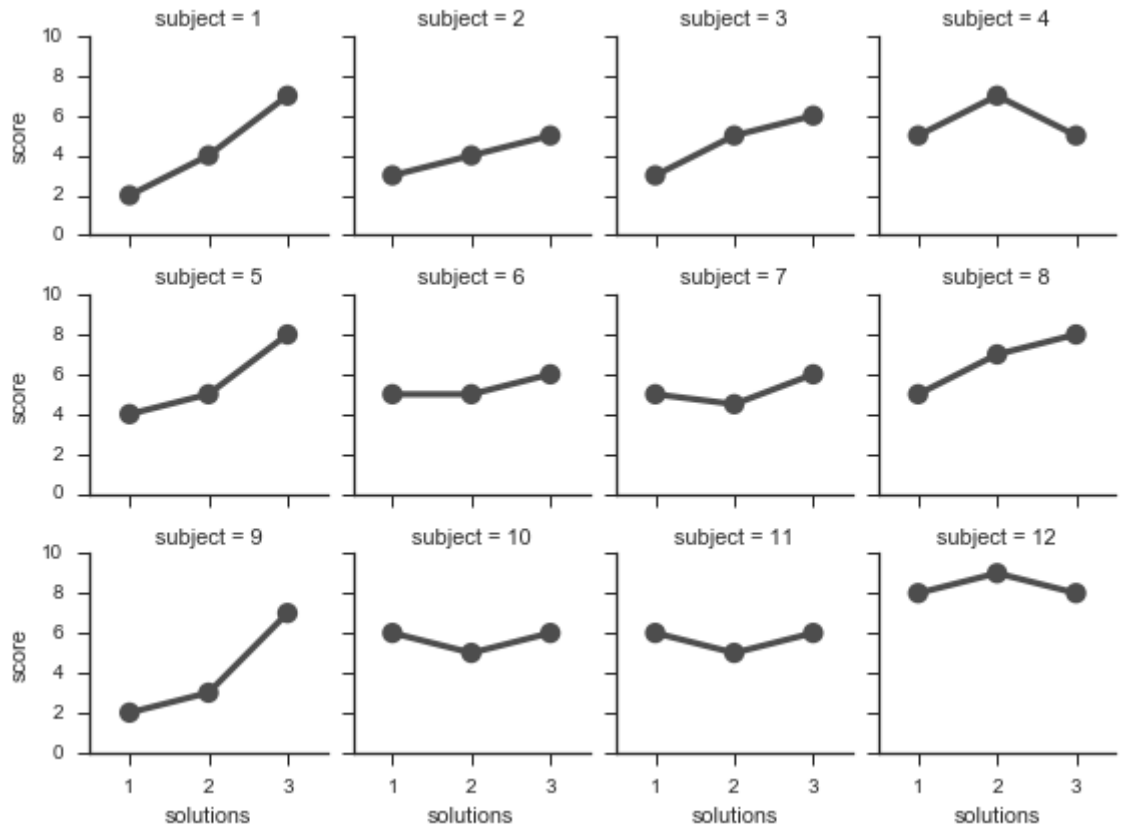
你同样也可以允许 plot 其它方面依据不同色调变量的水平有所不同，他可以帮助作画将变的更容易理解当打印成黑白时。为了做到这个，向 hue_kws 传递字典当 plottting 功能关键字参数的名字和关键字是关键字值列表时，每一个色调的水平都有一个。

```
g = sns.FacetGrid(tips, hue="sex", palette="Set1", size=5, hue_kws={"marker": ["^", "v"]})
g.map(plt.scatter, "total_bill", "tip", s=100, linewidth=.5, edgecolor="white")
g.add_legend();
```



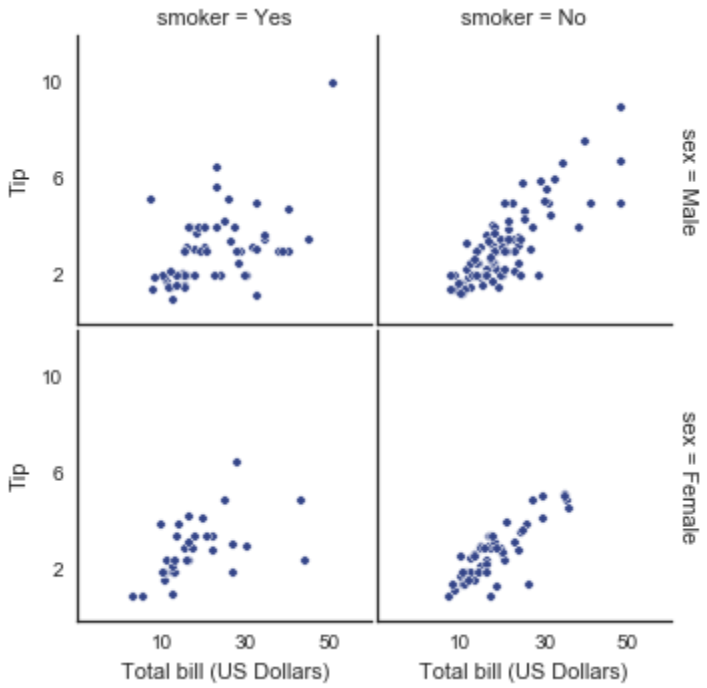
你可以有一个不同水平的变量，你可以画出它，并且列可以封装这样可以横跨多个行。当你这么做了，你不需要使用行变量。

```
attend = sns.load_dataset("attention").query("subject <= 12")
g = sns.FacetGrid(attend, col="subject", col_wrap=4, size=2, ylim=(0, 10))
g.map(sns.pointplot, "solutions", "score", color=".3", ci=None);
```



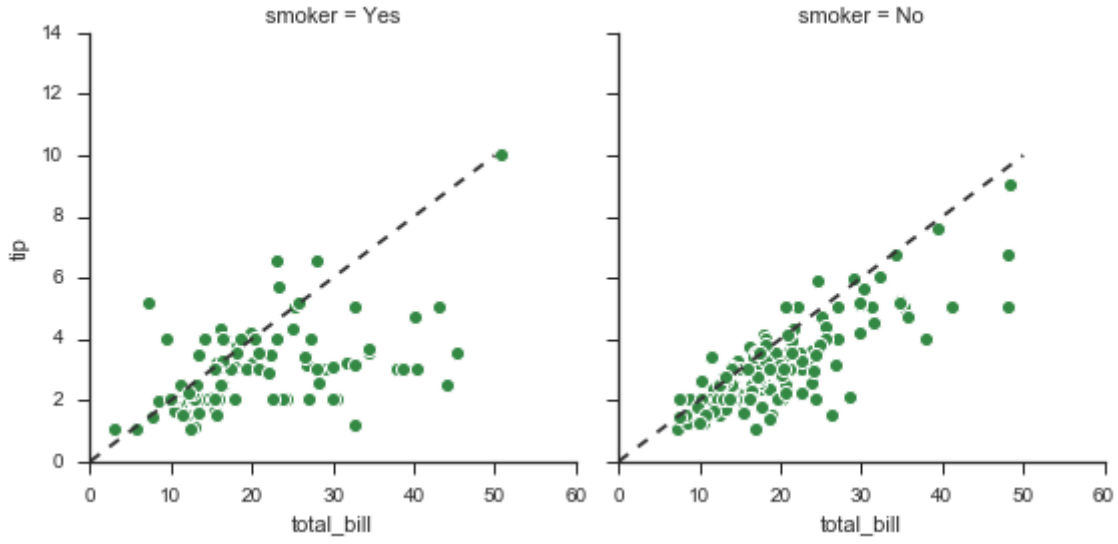
一旦你已经使用 FaceGrid.map()画出一张图（可以调用很多次），你可能想要调整图的某些方面，FacetGrid 对象有一些方法来在更高抽象层上来操作图像。最常用的是 FaceGrid.set()，还有一些特定方法像 FaceGride.set_axis_labels(),用来表示分面内部不需要轴标签。看例子。

```
with sns.axes_style("white"):
    g = sns.FacetGrid(tips, row="sex", col="smoker", margin_titles=True, size=2.5)
g.map(plt.scatter, "total_bill", "tip", color="#334488", edgecolor="white", lw=.5);
g.set_axis_labels("Total bill (US Dollars)", "Tip");
g.set(xticks=[10, 30, 50], yticks=[2, 6, 10]);
g.fig.subplots_adjust(wspace=.02, hspace=.02);
```



甚至自定义，你可以直接基于 matplotlib 的 Figuer 和 Axes 对象进行工作，它可以分别存为成员属性 fig 和 axes(两维数组)。当绘图不需要行或列分面时，你可以使用 ax 属性去直接访问单轴。

```
g = sns.FacetGrid(tips, col="smoker", margin_titles=True, size=4)
g.map(plt.scatter, "total_bill", "tip", color="#338844", edgecolor="white", s=50, lw=1)
for ax in g.axes.flat:
    ax.plot((0, 50), (0, .2 * 50), c=".2", ls="--")
g.set(xlim=(0, 60), ylim=(0, 14));
```



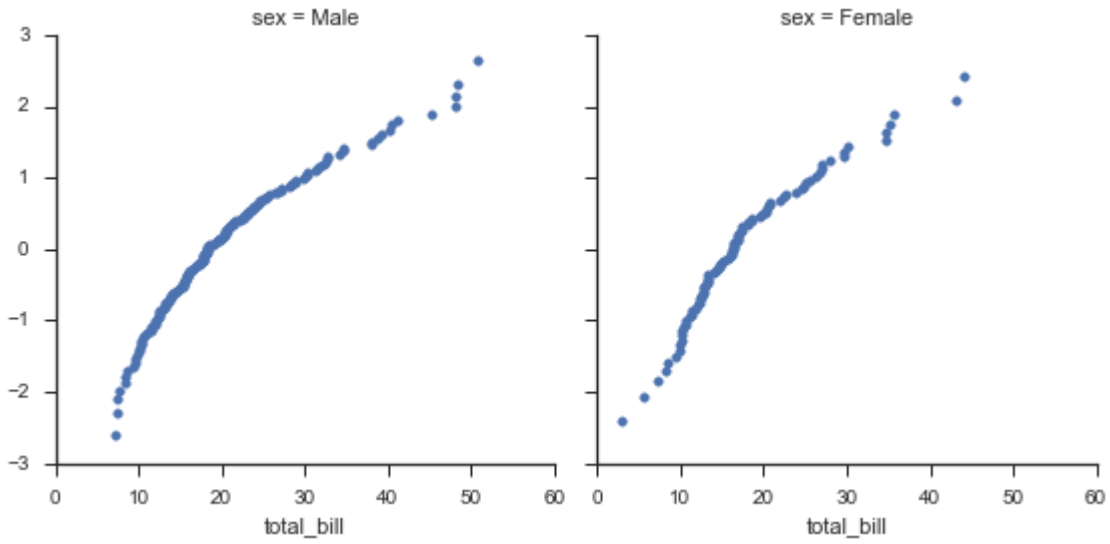
在网格上映射自定义功能

当使用 FaceGrid 时，不限于已有的 matplotlib 和 seaborn 功能。但是为了工作恰当，任何你使用的功能应当遵循一些规则。

1. 必须在画在当前激活的 matplotlib 轴。这才正真使用到 matplotlib.pyplot 命名空间，你可以调用 plt.gca 得到当前轴的参考如果你想要用它的方法直接工作的话。
2. 在绘制位置参数时必须接收数据，FaceGrid 在内部为每个命名的位置参数将传递数据序列至 FacetGrid.map()。
3. 必须能接受 color 和 label 关键字参数，理想情况下它能做一些有用的事情。在大多数情况下，抓取**kwargs 的通用字典是最容易的，根据底层绘图功能来传递它们。

让我们来看一下画图功能的最小例子，该功能可以使用每个分面数据的一个向量。

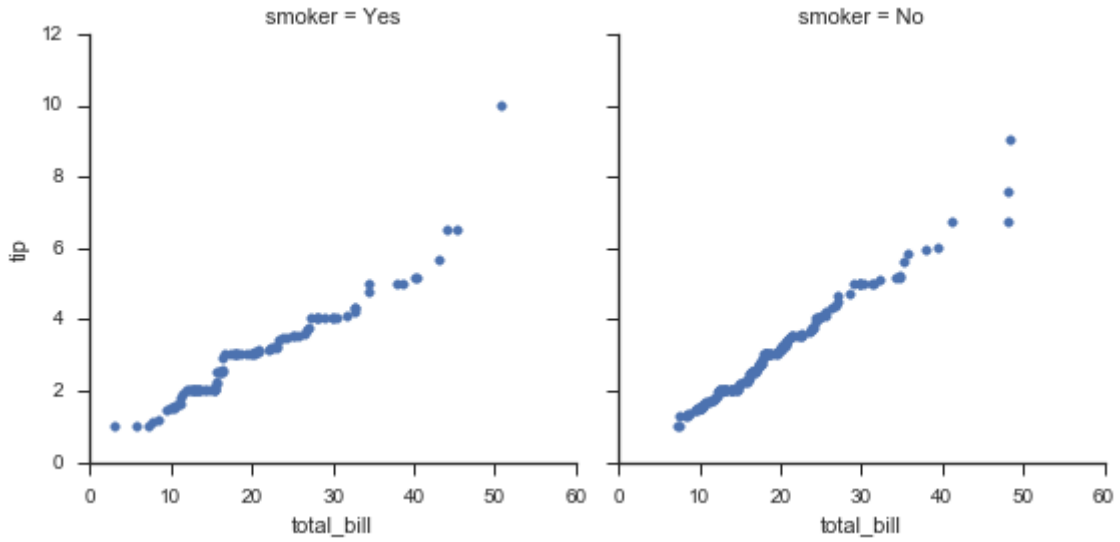
```
def quantile_plot(x, **kwargs):
    qntls, xr = stats.probplot(x, fit=False)
    plt.scatter(xr, qntls, **kwargs)

g = sns.FacetGrid(tips, col="sex", size=4)
g.map(quantile_plot, "total_bill");
```



如果我们想要做一个双变量画图，你应该写这个函数它才能先接收 x 轴变量再 y 轴变量。

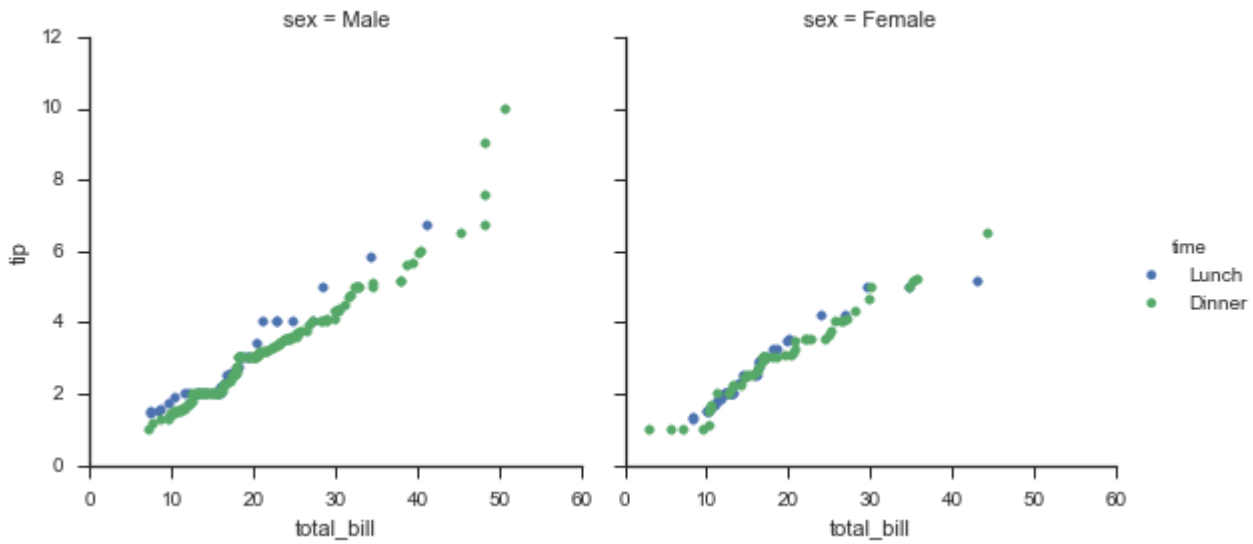
```
def qqplot(x, y, **kwargs):
    _, xr = stats.probplot(x, fit=False)
```

```
    _, yr = stats.probplot(y, fit=False)
    plt.scatter(xr, yr, **kwargs)

g = sns.FacetGrid(tips, col="smoker", size=4)
g.map(qqplot, "total_bill", "tip");
```



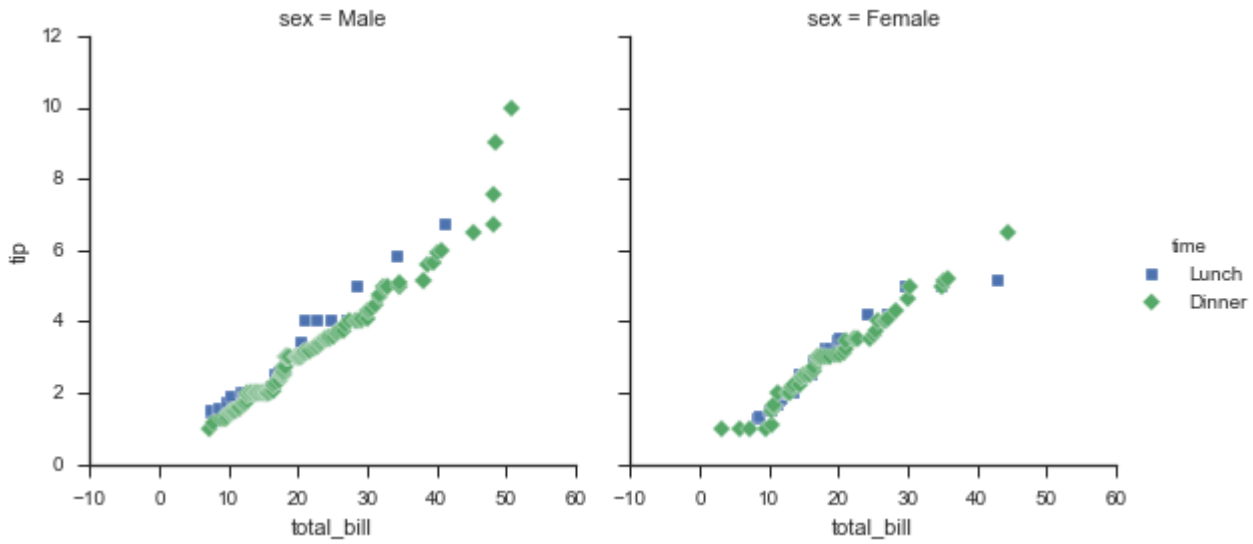
因为 plt.scatter 接收 color 和 label 关键字参数，如果正确运行它们，我们可以毫无困难的增加色调分面。

```
g = sns.FacetGrid(tips, hue="time", col="sex", size=4)
g.map(qqplot, "total_bill", "tip")
g.add_legend();
```



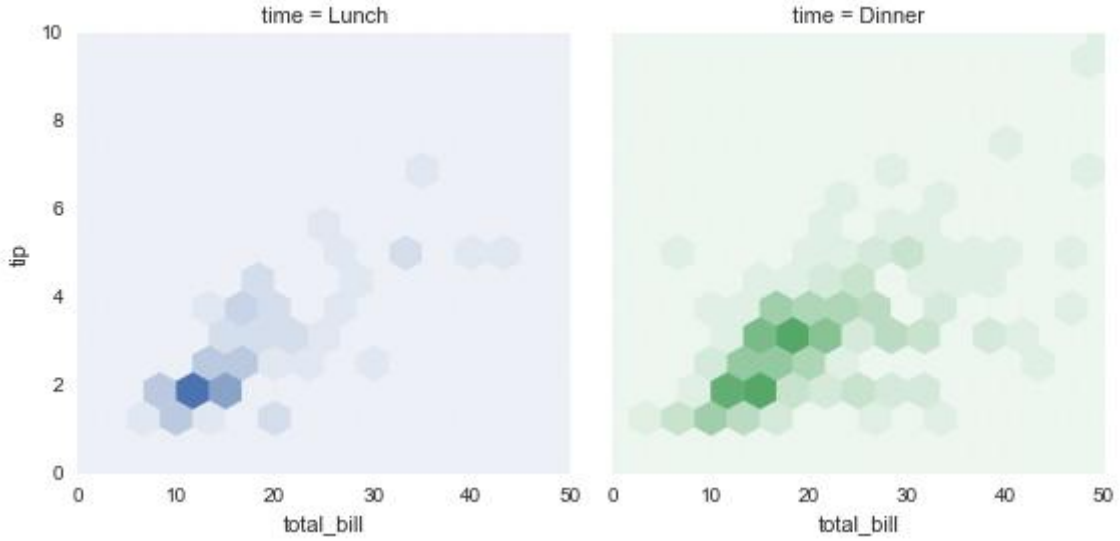
这个方法同样允许我们使用额外的 aesthetics 去区分色调变量的不同水平和关键字参数，这样就不依赖于 faceting 变量了。

```
g = sns.FacetGrid(tips, hue="time", col="sex", size=4,
                  hue_kws={"marker": ["s", "D"]})
g.map(qqplot, "total_bill", "tip", s=40, edgecolor="w")
g.add_legend();
```



有时候，即使你想要去用 color 和 label 关键字参数映射函数时但不如你预期那样运行良好，在这种情况，你想要在你自定义函式逻辑中明确抓取它们并处理它们。举个例子，这种方法允许你映射 plt.hexbin，否则 FaceGrid API 使用不佳。

```
def hexbin(x, y, color, **kwargs):
    cmap = sns.light_palette(color, as_cmap=True)
    plt.hexbin(x, y, gridsize=15, cmap=cmap, **kwargs)

with sns.axes_style("dark"):
    g = sns.FacetGrid(tips, hue="time", col="time", size=4)
g.map(hexbin, "total_bill", "tip", extent=[0, 50, 0, 10]);
```



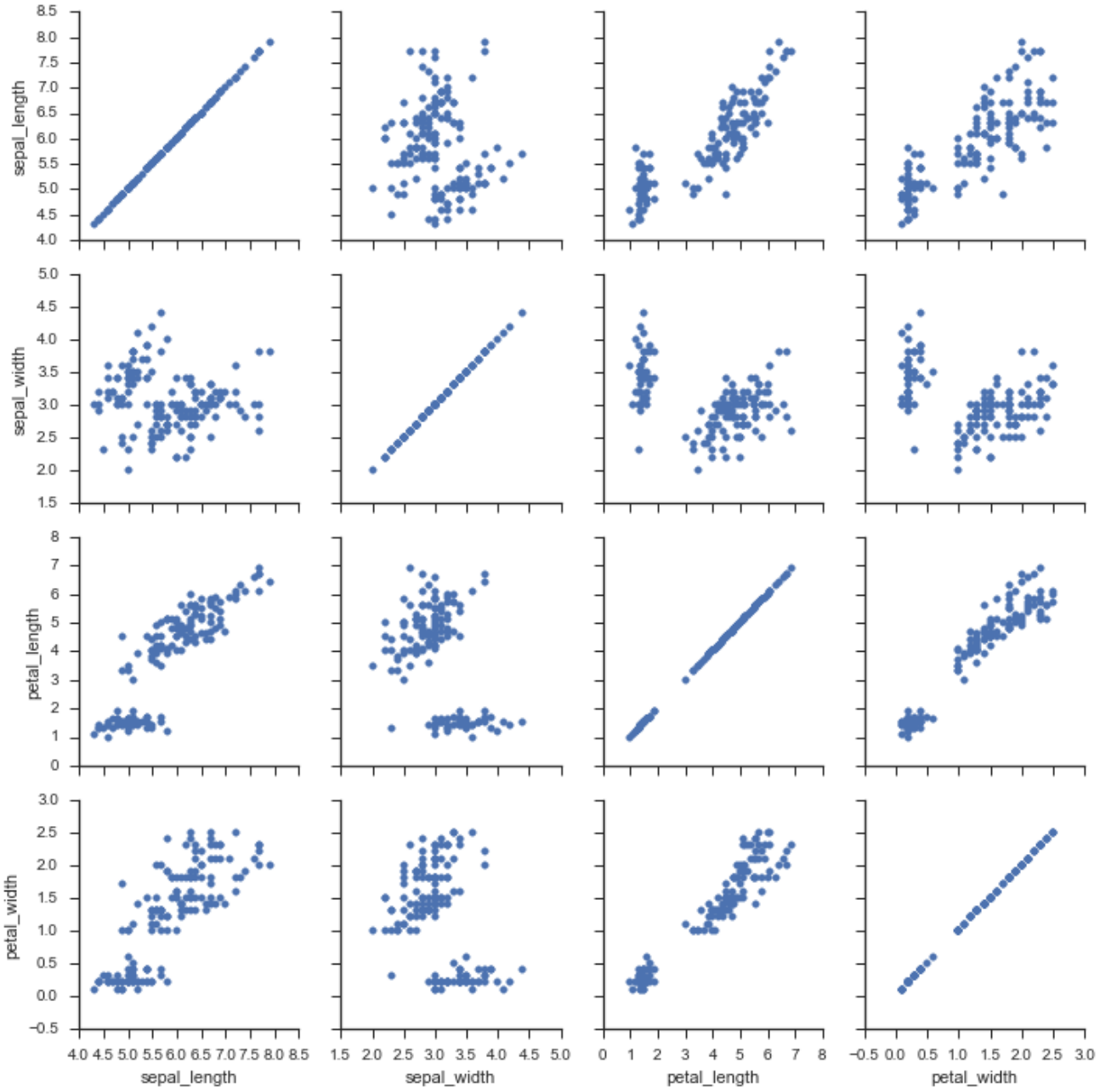
用 PairGrid 和 pairplot()绘制成对关系

PairGrid 也允许你用同样的 plot 类型快速绘制小子集的网格来可视化数据。在 PairGrid 中，每一个行和列分配给不同变量。

去理解 FaceGrid 和 PairGrid 的不同是很重要的，在前者，每个分面在不同其它变量层级上显示同样的关系条件。在后者，每张图显示不同的关系（即使上三角和下三角有镜像图）。使用 PairGrid 可以给你生成非常快速、非常高阶数据的有趣的关系摘要。

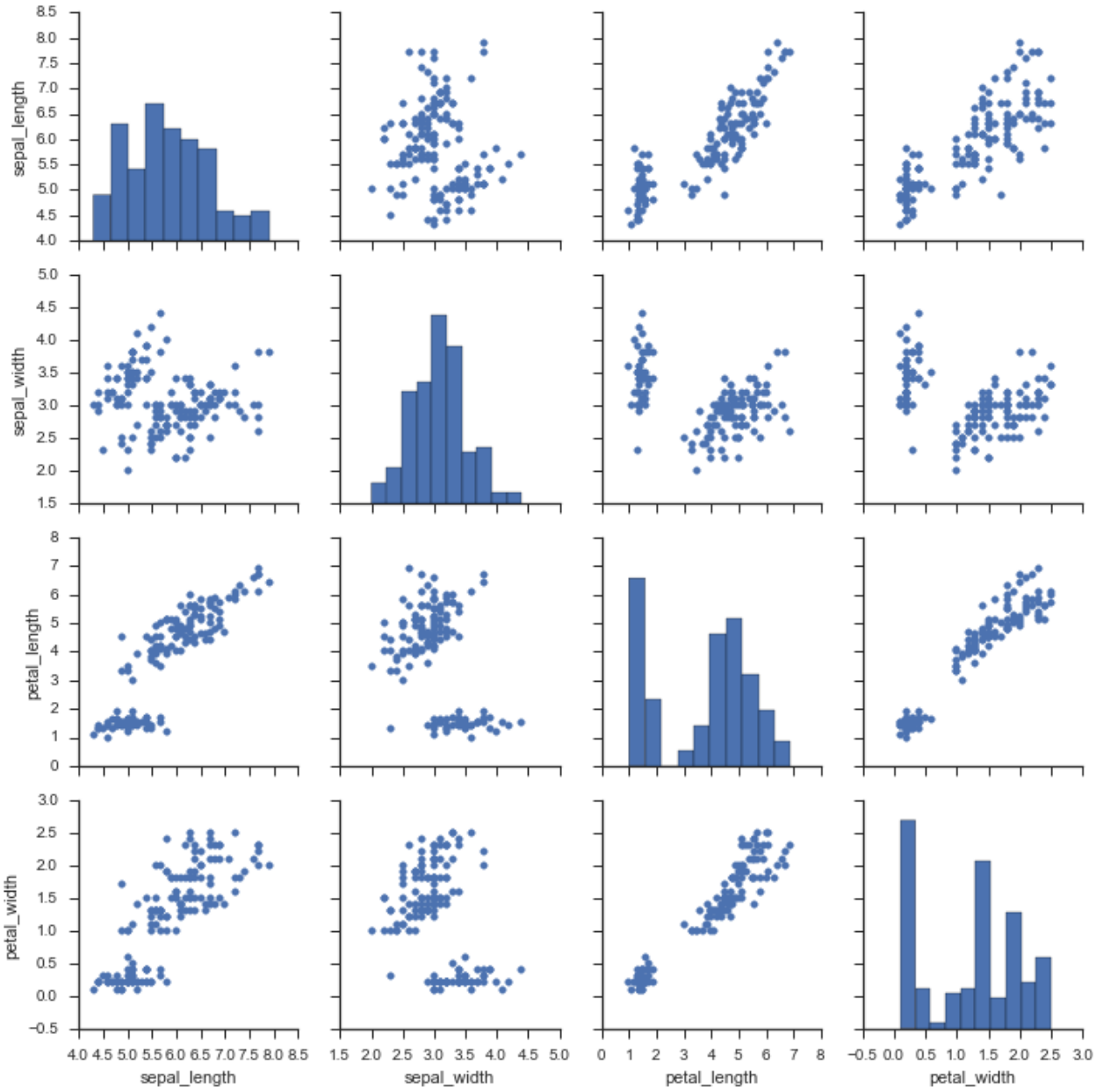
该类最基础的用法与 FacetGrid 非常相似。首先你初始化该网格，然后你传递

```
iris = sns.load_dataset("iris")
g = sns.PairGrid(iris)
g.map(plt.scatter);
```



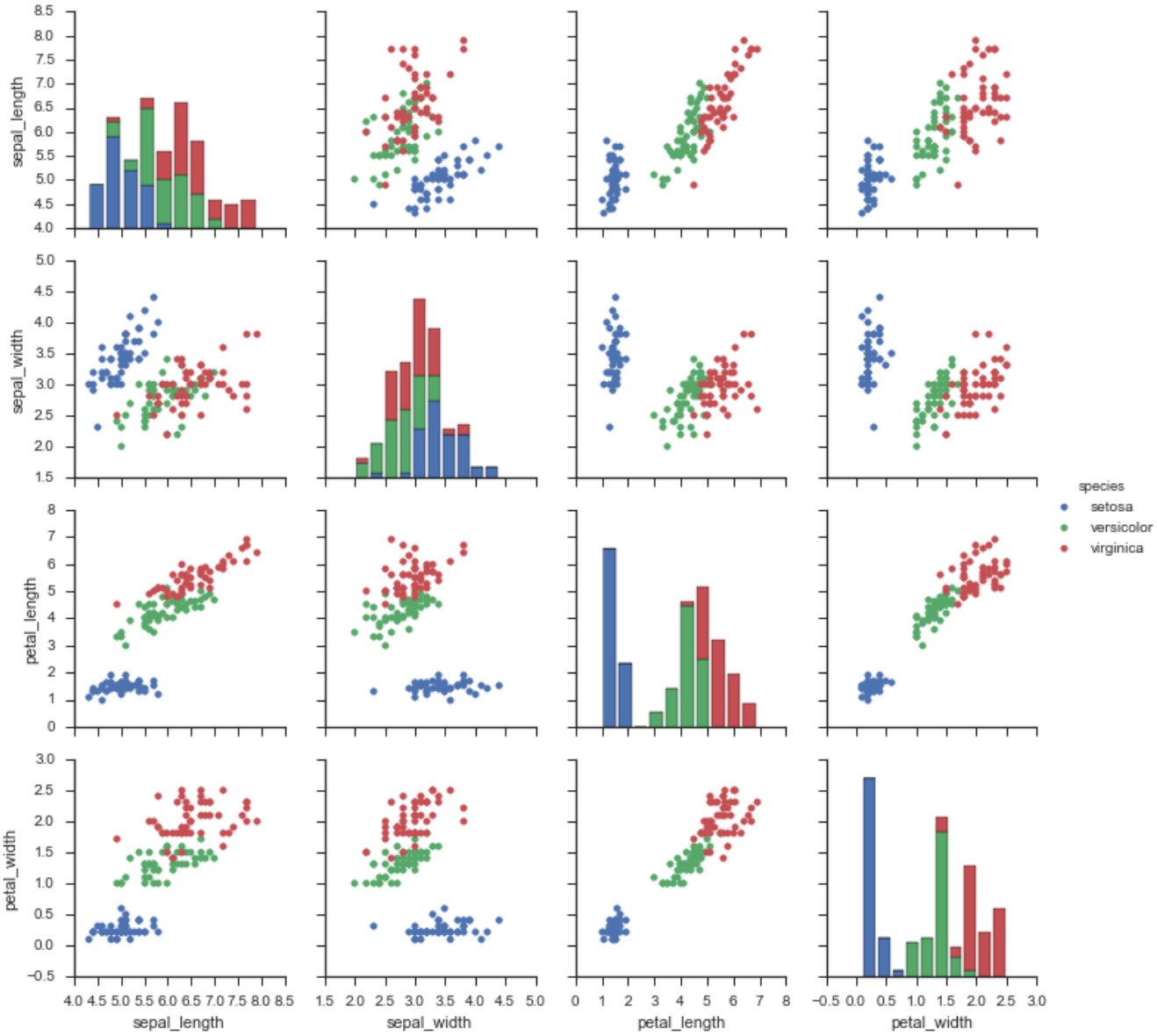
对角显示每一列变量的单变量分布来绘制不同的功能是可能的。留意轴刻度不能对应计数或轴密度。

```
g = sns.PairGrid(iris)
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter);
```



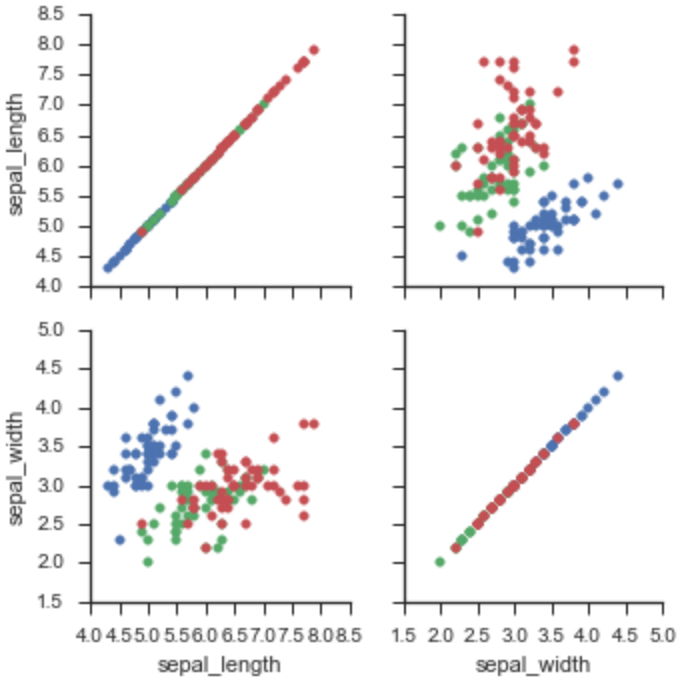
绘制这些观测颜色的常用方法，是通过各自类别变量。如，这鸢尾花数据集有 4 个度量，每个度量有 3 个不同的种类，所以你可以看到他们如何不同的。

```
g = sns.PairGrid(iris, hue="species")
g.map_diag(plt.hist)
g.map_offdiag(plt.scatter)
g.add_legend();
```



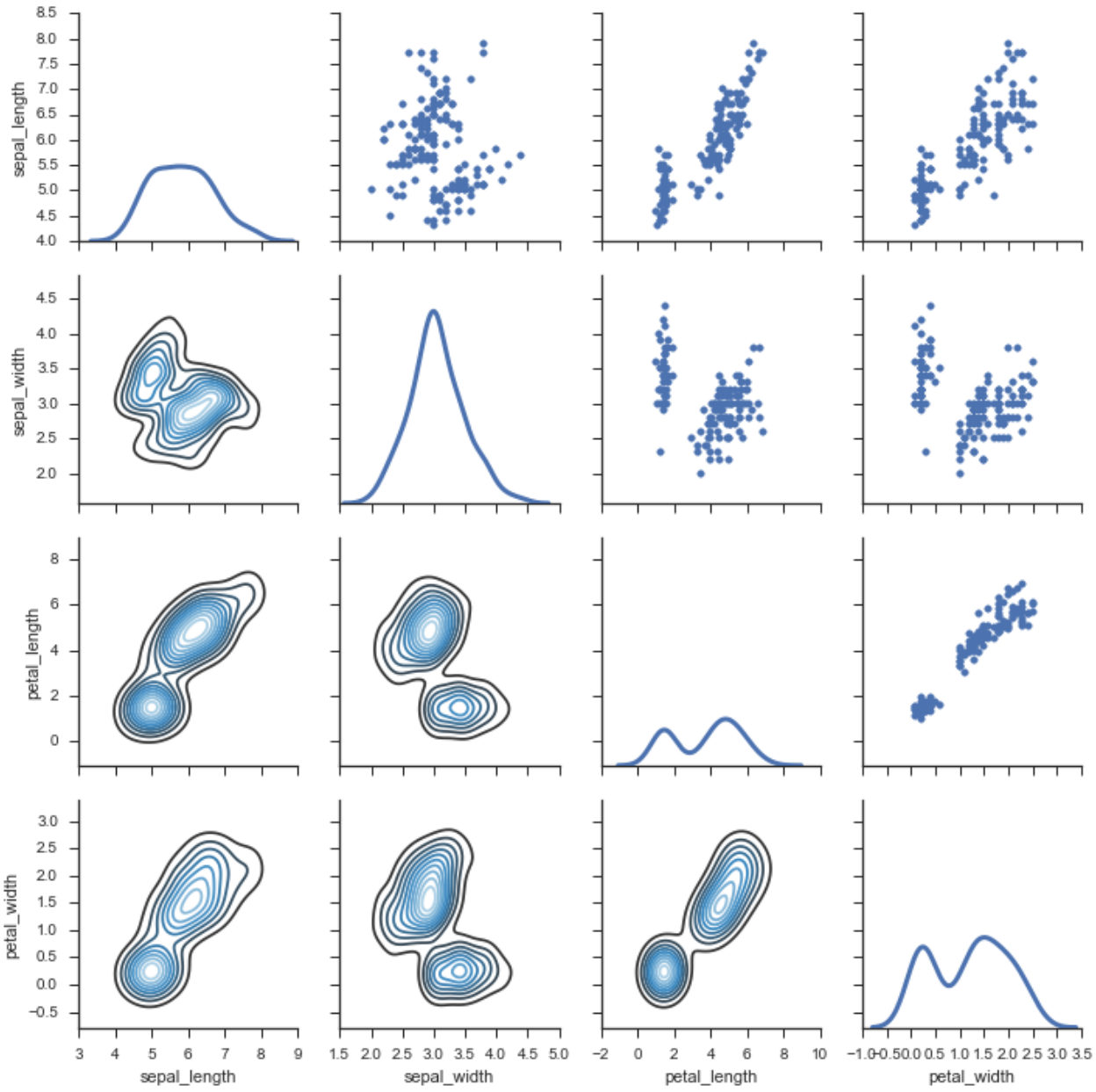
默认在数据集中的所有数值列，但是你可以关注你想要的特定关系。

```
g = sns.PairGrid(iris, vars=["sepal_length", "sepal_width"], hue="species")
g.map(plt.scatter);
```



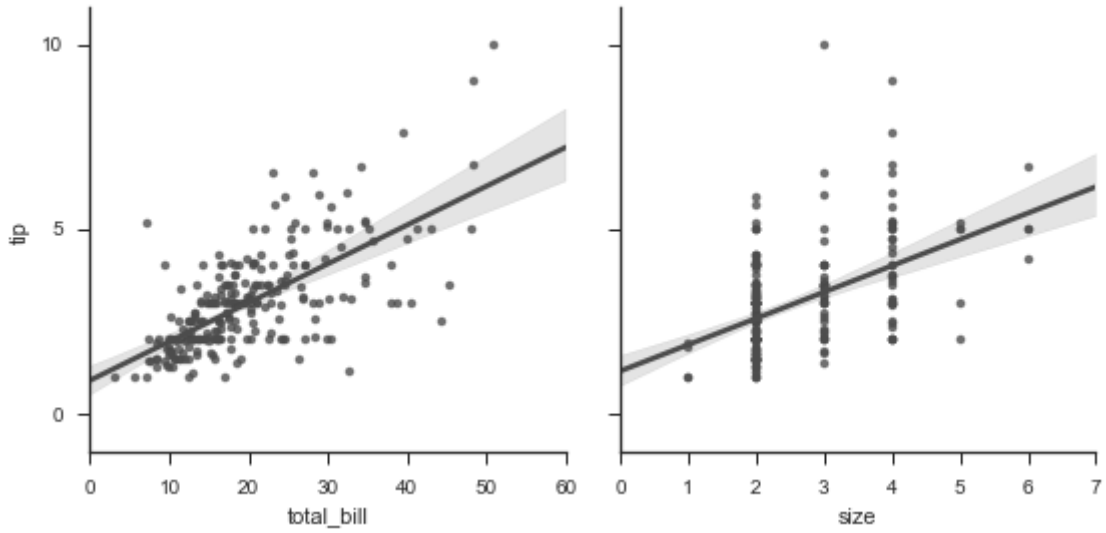
使用上三角或下三角强调关系的不同方面也是可能的。

```
g = sns.PairGrid(iris)
g.map_upper(plt.scatter)
g.map_lower(sns.kdeplot, cmap="Blues_d")
g.map_diag(sns.kdeplot, lw=3, legend=False);
```



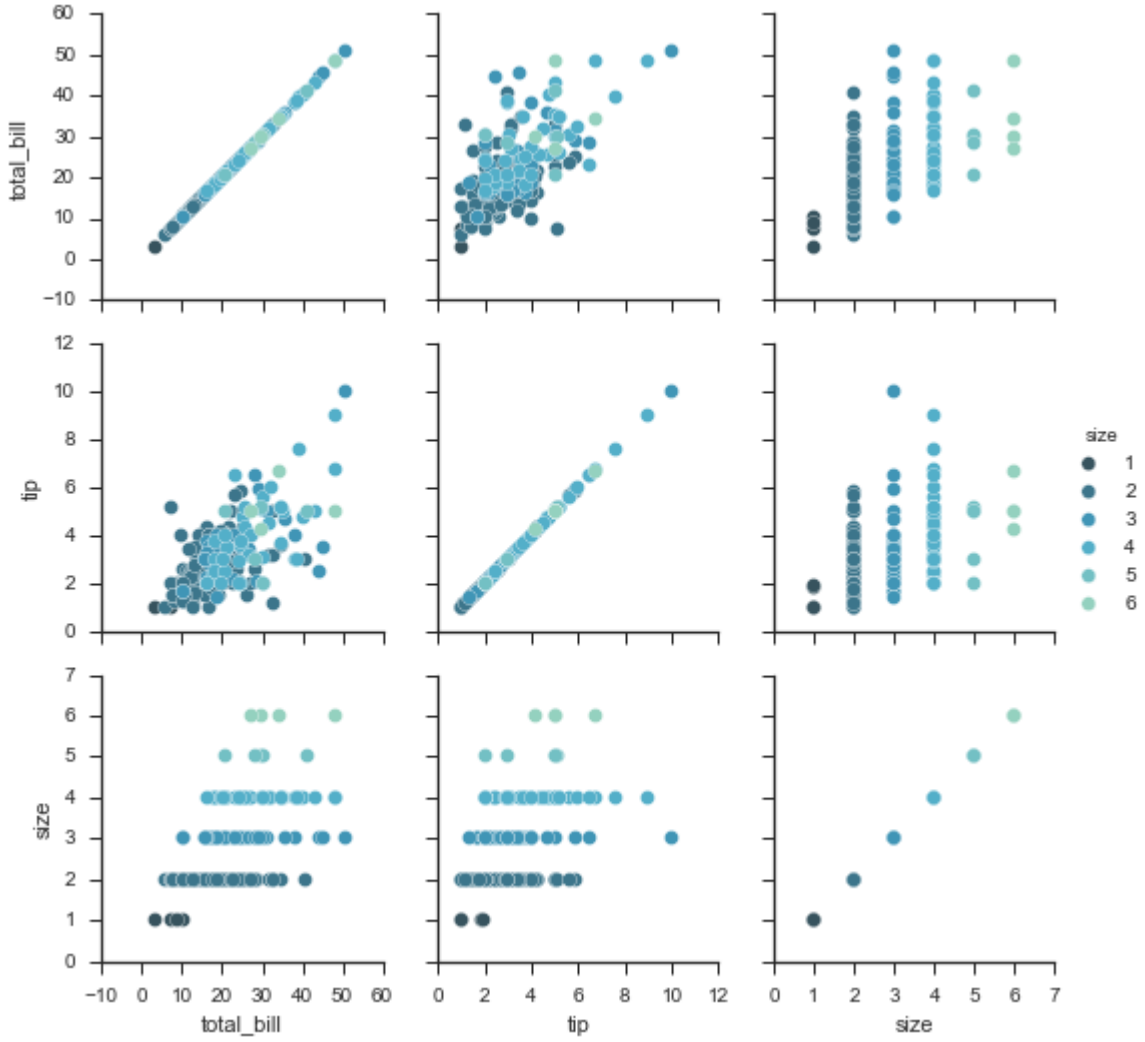
对角线上的方格子与身份关系实际上只是一种特殊情况，你可以在行和列中绘制不同的变量。

```
g = sns.PairGrid(tips, y_vars=["tip"], x_vars=["total_bill", "size"], size=4)
g.map(sns.regplot, color=".3")
g.set(ylim=(-1, 11), yticks=[0, 5, 10]);
```



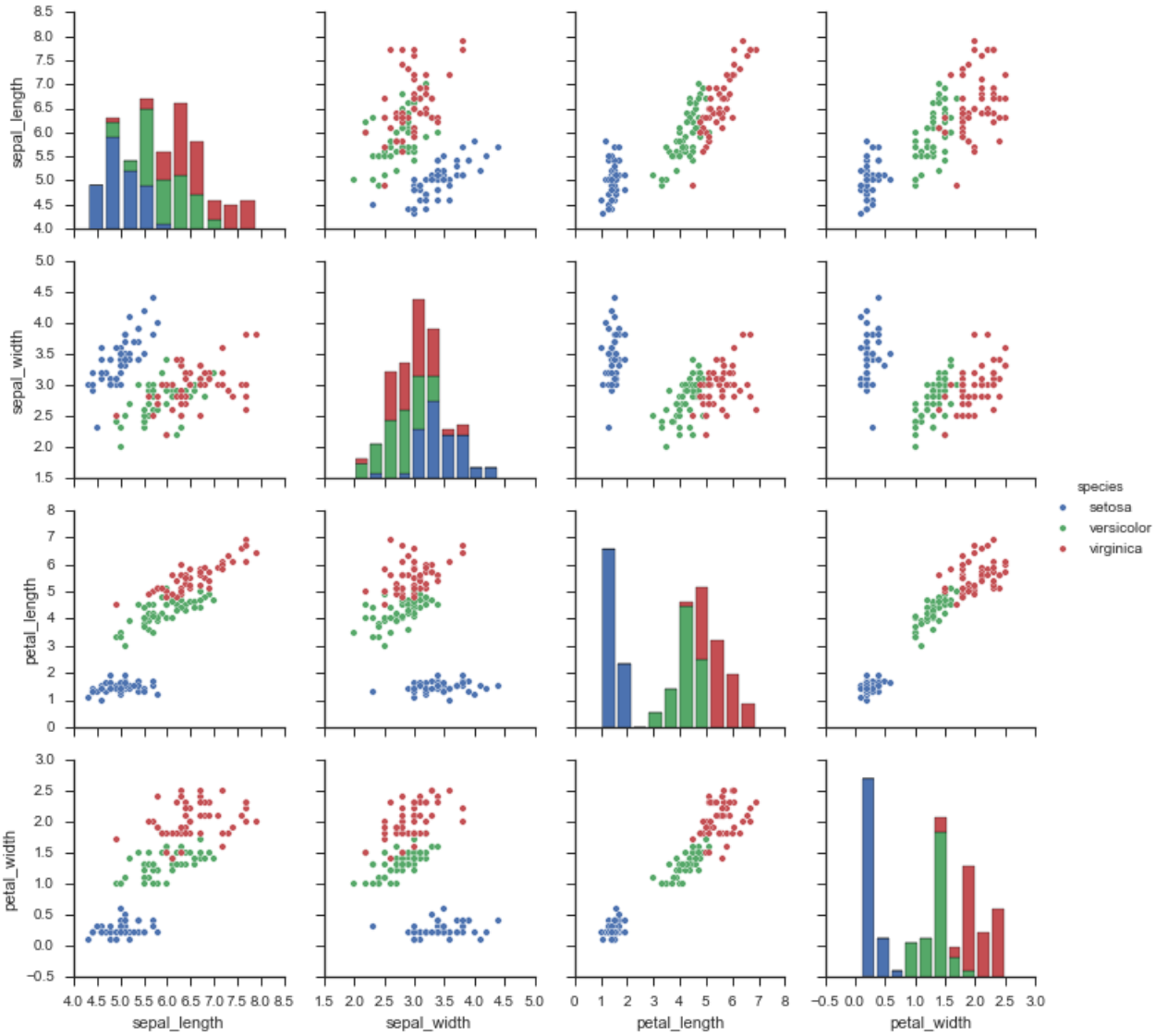
当然，aesthetic 属性是可以配置的。举个例子，你可以在这个绘制函数中使用不同的调色板（比如说，显示色调变量的顺序）和忽略关键字参数。

```
g = sns.PairGrid(tips, hue="size", palette="GnBu_d")
g.map(plt.scatter, s=50, edgecolor="white")
g.add_legend();
```



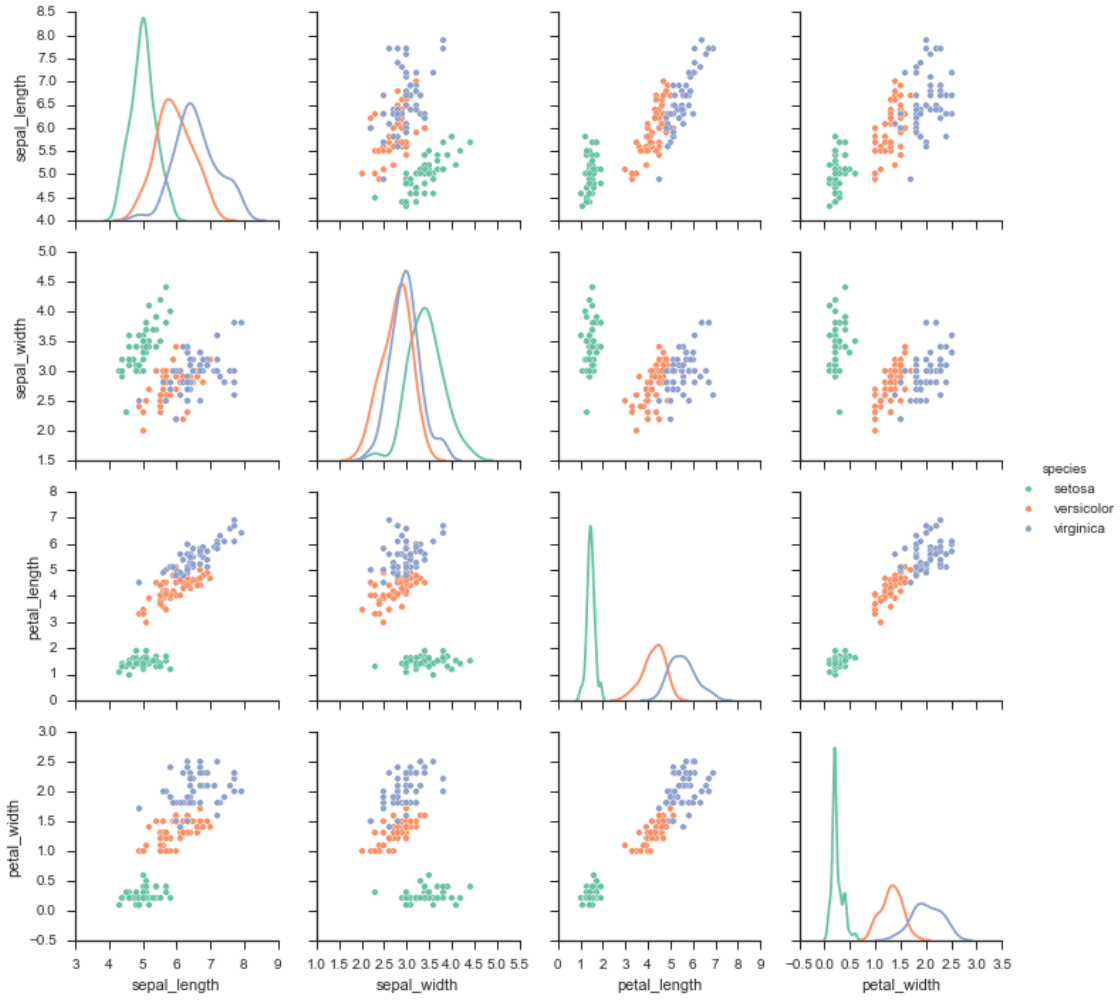
PairGrid 很灵活，但为了要迅速浏览一下数据集，用 pairplot()显得更容易使用。该函数默认使用散点图和直方图，即使仅仅增加少量其它种类。（目前，你可以在对角线上各 KDE 对角线上绘制回归图）

```
sns.pairplot(iris, hue="species", size=2.5);
```



你也可以通过关键字参数来控制图像的 aesthetics，它将返回更进一步微调 PariGrid 实例。

```
g = sns.pairplot(iris, hue="species", palette="Set2", diag_kind="kde", size=2.5)
```


sepal_length
sepal_width
petal_length
petal_width
species
setosa
versicolor
virginica

API reference ¶

Distribution plots¶分布绘图

`jointplot`(x, y[, data, kind, stat_func, ...])	Draw a plot of two variables with bivariate and univariate graphs. 绘制单因素或双因素两个变量图。
`pairplot`(data[, hue, hue_order, palette, ...])	Plot pairwise relationships in a dataset. 绘制数据集成对关系。
`distplot`(a[, bins, hist, kde, rug, fit, ...])	Flexibly plot a univariate distribution of observations. 灵活绘制观测数据的单变量分布。
`kdeplot`(data[, data2, shade, vertical, ...])	Fit and plot a univariate or bivariate kernel density estimate. 拟合和绘制单因素或双因素核密度估计
`rugplot`(a[, height, axis, ax])	Plot datapoints in an array as sticks on an axis. 绘制数组的数据点成轴上的柱。

Regression plots¶回归绘图

`lmplot`(x, y, data[, hue, col, row, palette, ...])	Plot data and regression model fits across a FacetGrid. 通过 FacetGrid 绘制数据和回归模型拟合
`regplot`(x, y[, data, x_estimator, x_bins, ...])	Plot data and a linear regression model fit. 绘制数据和线性回归拟合。
`residplot`(x, y[, data, lowess, x_partial, ...])	Plot the residuals of a linear regression. 绘制回归模型的残差
`interactplot`(x1, x2, y[, data, filled, ...])	Visualize a continuous two-way interaction with a contour plot. 通过轮廓图绘制两水平交互项。
`coefplot`(formula, data[, groupby, ...])	Plot the coefficients from a linear model. 绘制回归模型的系数。

Categorical plots¶分类绘图

`factorplot`([x, y, hue, data, row, col, ...])	Draw a categorical plot onto a FacetGrid. 在 FacetGrid 绘制分类图。
`boxplot`([x, y, hue, data, order, hue_order, ...])	Draw a box plot to show distributions with respect to categories. 绘制盒形图显示分类相关的分布
`violinplot`([x, y, hue, data, order, ...])	Draw a combination of boxplot and kernel density estimate. 绘制盒形图和核密度估计的组合。
`stripplot`([x, y, hue, data, order, ...])	Draw a scatterplot where one variable is categorical.

	绘制散点图当一个变量为类别。
swarmplot([x, y, hue, data, order, ...])	Draw a categorical scatterplot with non-overlapping points. 绘制无重叠点的分类散点图
pointplot([x, y, hue, data, order, ...])	Show point estimates and confidence intervals using scatter plot glyphs. 用散点图符号显示点估计和置信区间。
barplot([x, y, hue, data, order, hue_order, ...])	Show point estimates and confidence intervals as rectangular bars. 显示点估计和置信区间为矩形条。
countplot([x, y, hue, data, order, ...])	Show the counts of observations in each categorical bin using bars. 在每个类别组距显示观测条数

Matrix plots¶矩阵绘图

heatmap(data[, vmin, vmax, cmap, center, ...])	Plot rectangular data as a color-encoded matrix. 将图矩形数据作为颜色编码矩阵
clustermap(data[, pivot_kws, method, ...])	Plot a hierarchically clustered heatmap of a pandas DataFrame Pandas 数据框数据绘制层次聚类热点图。

Timeseries plots¶时间序列绘图

tsplot(data[, time, unit, condition, value, ...])	Plot one or more timeseries with flexible representation of uncertainty. 使用灵活的不确定性表达式绘制一个或多个时间序列图

Miscellaneous plots¶杂项绘图

palplot(pal[, size])	Plot the values in a color palette as a horizontal array. 将调色板中的值作为一个水平数组

Axis grids¶轴网络

FacetGrid(data[, row, col, hue, col_wrap, ...])	Subplot grid for plotting conditional relationships. 绘制条件关系的子网格。
PairGrid(data[, hue, hue_order, palette, ...])	Subplot grid for plotting pairwise relationships in a dataset. 绘制成对关系的子网格
JointGrid(x, y[, data, size, ratio, space, ...])	Grid for drawing a bivariate plot with marginal univariate plots.

Style frontend¶风格前端

set([context, style, palette, font, ...])	Set aesthetic parameters in one step. 一步设置 aesthetic 参数。
axes_style([style, rc])	Return a parameter dict for the aesthetic style of the plots. 返回该图的 aesthetic 风格参数字典。
set_style([style, rc])	Set the aesthetic style of the plots. 设置该图的 aesthetic 风格。
plotting_context([context, font_scale, rc])	Return a parameter dict to scale elements of the figure. 返回该图尺度元素的参数字典。
set_context([context, font_scale, rc])	Set the plotting context parameters. 设置绘图文本参数。
set_color_codes([palette])	Change how matplotlib color shorthands are interpreted. 改变 matplotlib 颜色简写注释的方式。
reset_defaults()	Restore all RC params to default settings. 恢复所有 RC 参数为默认设置
reset_orig()	Restore all RC params to original settings (respects custom rc). 恢复所有 RC 参数为初始设置（关于定制 rc）

Color palettes¶调色板

set_palette(palette[, n_colors, desat, ...])	Set the matplotlib color cycle using a seaborn palette. 使用 seaborn 调色板设置 matplotlib 色循环。
color_palette([palette, n_colors, desat])	Return a list of colors defining a color palette. 返回调色板颜色定义列表。
husl_palette([n_colors, h, s, l])	Get a set of evenly spaced colors in HUSL hue space. 得到一组间隔均匀在 HUSL 的色调空间。
hls_palette([n_colors, h, l, s])	Get a set of evenly spaced colors in HLS hue space. 得到一组间隔均匀在 HLS 的色调空间。
cubehelix_palette([n_colors, start, rot, ...])	Make a sequential palette from the cubehelix system. 生成立方螺旋系统的连续调色板
dark_palette(color[, n_colors, reverse, ...])	Make a sequential palette that blends from dark to color. 生成连续的调色板，混合深色和颜色。
light_palette(color[, n_colors, reverse, ...])	Make a sequential palette that blends from light to color. 生成连续的调色板，混合浅色和颜色。
diverging_palette(h_neg, h_pos[, s, l, sep, ...])	Make a diverging palette between two HUSL colors. 生成两种 HUSL 颜色的离散调色板
blend_palette(colors[, n_colors, as_cmap, in	Make a palette that blends between a list of colors.

put])

生成两种颜色表的混合调色板。

`xkcd_palette`(colors)

Make a palette with color names from the xkcd color survey.

生成从 xkcd 颜色测量颜色命名的调色板

`crayon_palette`(colors)

Make a palette with color names from Crayola crayons.

生成 Crayola 蜡笔颜色命名的调色板

`mpl_palette`(name[, n_colors])

Return discrete colors from a matplotlib palette.

返回 matplotlib 调色板的离散颜色。

Palette widgets¶调色板组件

`choose_colorbrewer_palette`(data_type[, as_cmap])

Select a palette from the ColorBrewer set.

为 ColorBrewer 套件选择调色板

`choose_cubehelix_palette`([as_cmap])

Launch an interactive widget to create a sequential cubehelix palette.

启动交互组件创建序列立方螺旋调色板

`choose_light_palette`([input, as_cmap])

Launch an interactive widget to create a light sequential palette.

启动交互组件创建轻量级序列调色板

`choose_dark_palette`([input, as_cmap])

Launch an interactive widget to create a dark sequential palette.

启动交互组件创建深度序列调 色板。

`choose_diverging_palette`([as_cmap])

Launch an interactive widget to choose a diverging color palette.

启动交互组件选择离散调色板。

Utility functions¶工具类函数

`despine`([fig, ax, top, right, left, bottom, ...])

Remove the top and right spines from plot(s).

`desaturate`(color, prop)

Decrease the saturation channel of a color by some percent.

通过百分比来减少一个颜色通道的饱和度

`saturate`(color)

Return a fully saturated color with the same hue.

返回一个与色调相同的完全饱和颜色。

`set_hls_values`(color[, h, l, s])

Independently manipulate the h, l, or s channels of a color.

独立实现颜色 h,l,s 通道

`ci_to_errsize`(cis, heights)

Convert intervals to error arguments relative to plot heights.

转换时间间隔与绘图高度相关的错误参数

`axlabel`(xlabel, ylabel, **kwargs)

Grab current axis and label it.

抓取当前轴并标注它