

---

# Bokeh 中文使用手册

靳军<sup>1</sup>

刘逸铭<sup>2</sup>

<sup>1</sup>华东师范大学, 上海, 中国

<sup>2</sup>华东师范大学, 上海, 中国

---

June 13, 2017

**B**okeh 是一个很美观实用、功能强大并能在前端完美嵌入的 **Python** 交互绘图库，但是他的官方文档并不十分好查阅，加上现在还没有完整的中文文档，所以我们在前人的初步成果上，将其用户指南完整翻译出来，并加以提炼，方便更多人学习。

## 1 用户指南 User Guide

这一章内容主要是了解一下整片用户指南的大体结构，根据你可能用到的情况，你可以挑选不同的章节去了解。如果想快速使用而不深究，可以只看快速入门。章节根据内容分为：

- 快速入门 Quickstart
- 设置 Getting Set Up
- 一些重要概念 Defining Key Concepts
- 用基础标志绘图 Plotting with Basic Glyphs
- 嵌入图表和 Apps Embedding Plots and Apps

我们将对这些内容分别作出详细介绍。

## 2 快速入门 Quickstart

### 2.1 介绍 Introduction

Bokeh 是致力于网页浏览器展示的 **Python** 交互式图表库。Bokeh 能读取巨大的数据集或者流数据以简单快捷的方式为网页提供优美、简洁、高交互性能的图形。

为了能让用户高度自定义简单、高性能、灵活的图表，Bokeh 开放三个层次的接口给用户：

- **bokeh.models**: 低级接口，能为 **app** 开发者提供高度灵活的图形表示（可以自定义一些顶层的组件）
- **bokeh.plotting**: 中级接口，该接口主要用于绘制曲线（默认加载一些低级的组件）
- **bokeh.charts**: 高级接口，能快速简单地构建复杂的图形

快速入门主要运用 **bokeh.plotting** 接口。

### 2.2 快速安装 Quick Installation

Bokeh 有多种安装方式，Bokeh 推荐在 Anaconda Python distribution 的命令窗口来安装，这是最简单的方法。

```
1 conda install bokeh
```

该命令会安装 Bokeh 所有的依赖包。

Anaconda 最小安装适用于包括 Windows 等大多数系统，在安装的同时也会在 Anaconda 的根目录中创建一个 `examples/` 的子目录，其中包含了一些例子。

如果你已经安装了 Bokeh 所有的依赖包，比如 Numpy，那么你也可以通过 pip 来安装：

```
1 pip install bokeh
```

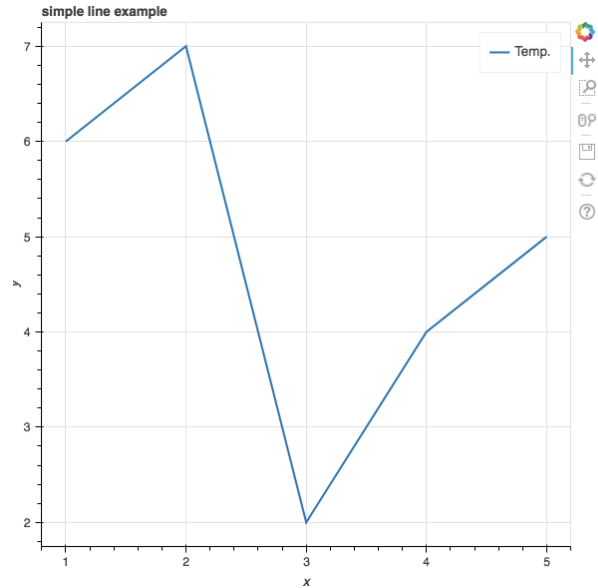


Figure 1: Bokeh 快速画图 1

## 2.3 开始 Getting Started

Bokeh 是一个很大的库，功能丰富，而这一章节中只是对 Bokeh 常用的示例和工作流进行简单讲解。若要了解更多功能，请参见官方英文 document: [User Guide](#)。

下面是一些示例（确保装好所有依赖）：

以 Python 列表 (List) 作为参数传入绘制带交互工具（缩放、画笔、自适应大小、保存等）的图。

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 # 生成数据
4 x = [1, 2, 3, 4, 5]
5 y = [6, 7, 2, 4, 5]
6 # 设定输出为HTML
7 output_file("start1.html")
8 # 在HTML上生成图层
9 p = figure(title="simple line example",
10            x_axis_label='x', y_axis_label='y')
11 # 在图层上画线
12 p.line(x, y, legend="Temp.", line_width=2)
13 # 打开网页，展示结果
14 show(p)
```

当运行这段代码之后，会在项目当前目录下创建一个 `start1.html` 文件，并且浏览器会自动打开一个新标签页来显示刚刚创建的图表，而且具有基本交互功能（为展示需要，会把代码生成的图形放在此文档中，以供参考）。

稍加总结，用 `bokeh.plotting` 接口创建图表的基本步骤如下：

- 准备一些数据（在上面的例子中，数据是一个简单的 Python 列表）
- 指定一个输出（在上面的例子中，用 `output_file()` 函数指定了输出文件名为 `"start1.html"`）
- 调用 `figure()` 函数创建图表容器并指定一些整体参数，比如 `title`、`tools` 和 `axes labels`
- 将数据传入渲染函数（在上面的例子中是 `Figure.line` 函数）并指定一些视觉参数，比如 `colors`、`legends` 和 `widths`
- 调用 `show()` 或者 `save()` 来显示或保存结果

当然，反复进行步骤三、四可以用来在一个图层（即一张图片）上绘制多个图形，与此同时，在初始化图层的第三步，可以很灵活便捷地设置坐标轴（比如对数坐标轴），并且自定义标签。另外，第四步的渲染过程也可以像 Matplotlib 一样个性化地加标签，生成多样的图例。可以参考下面的例子：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 # 生成数据
4 x = [0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0]
5 y0 = [i**2 for i in x]
6 y1 = [10**i for i in x]
7 y2 = [10**(i**2) for i in x]
8 # 设定输出为HTML
9 output_file("start2.html")
10 # 生成图层
11 p = figure(
```

```

12     tools="pan,box_zoom,reset,save",
13     y_axis_type="log",
14     y_range=[0.001, 10**11],
15     title="log axis example",
16     x_axis_label='sections',
17     y_axis_label='particles')
18 # 再次渲染
19 p.line(x, x, legend="y=x")
20 p.circle(x, x, legend="y=x",
21          fill_color="white", size=8)
22 p.line(x, y0, legend="y=x^2", line_width=3)
23 p.line(x, y1, legend="y=10^x",
24        line_color="red")
25 p.circle(x, y1, legend="y=10^x",
26          fill_color="red",
27          line_color="red", size=6)
28 p.line(x, y2, legend="y=10^x^2",
29        line_color="orange",
30        line_dash=[4, 4])
31 # 输出结果
32 show(p)

```

您将会看到如下结果：

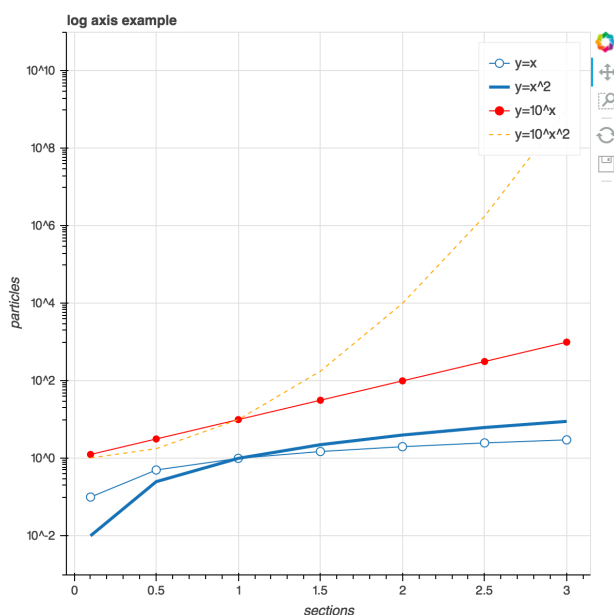


Figure 2: Bokeh 快速画图 2

## 2.4 多语言支持 Other Languages

Bokeh 的体系结构使得 Bokeh 能够非常容易的为其他语言提供使用接口，其中有一些小众语言的接口已经存在（比如 R, Scala, Julia）。尽管我们团队是 Python 的忠实粉丝，但是我们也同时在开发着不同语言的 Bokeh 库，可以通过以下链接查看已经编译完的一些多语言 Bokeh 库：

- [Bokeh for R](#)

- [Bokeh for Scala](#)
- [Bokeh for Julia](#)

## 2.5 示例数据 Sample Data

示例中所用的数据和示例是分开存放的，若要下载这些数据，可以在 Bash 或者 Windows 命令提示符后键入命令

```
1 bokeh sampledata
```

## 2.6 相关概念 Concepts

根据上面的一些例子，这里提出一些核心概念。

### 2.6.1 图层 Plot

图形是 Bokeh 的中心概念。图形在 Bokeh 中指的是容纳曲线、指示、数据、工具等几乎所有用来展示的元素的一个容器。具体的，Bokeh 中的 `bokeh.plotting` 接口提供了一个 `Figure` 类来聚集所有这些必要的元素，而 `Figure` 实例可以很方便的由 `figure()` 函数来创建。

### 2.6.2 标示 Glyphs

标示是 Bokeh 中的基础视觉元素。在 Bokeh 的底层，标示以 `glyph` 对象存在，比如 `Line`。如果你要使用低级接口 `bokeh.models` 来绘制图形，那么你还可能需要创建所有必要的 Bokeh 对象，其中包括上面提到的 `glyph` 对象和对应的数据集。为了使用更方便，`bokeh.plotting` 提供一个更高级的 `glyph` 方法像本章第一个示例中用的 `Figure.line` 就是其中一个。第二个示例中增加了 `Figure.circle` 来同时显示曲线和圆。除了曲线和圆，Bokeh 还有许多可选的 [标示](#) 和 [标志](#)。

### 2.6.3 指示 & 说明 Guides and Annotations

Bokeh 还提供一些别的视觉组件来帮助用户更好的展示或者让用户在图中做比较。这一类组件分为两类：

- 指示 (Guides)：指示的作用主要是帮助用户判断图中的距离，角度等。具体的包括格线、坐标轴等
- 说明 (Annotations)：说明主要包括图中的标签部分，具体的比如有标题，图例等

## 2.6.4 数据范围 Ranges

数据范围指的是绘图过程中所使用的数据的上界与下界。默认情况下，用 `bokeh.plotting` 接口绘制出的图像时，Bokeh 的 `DataRange` 对象会根据所用数据计算出数据范围，之后传给绘图函数。当然，数据范围也可以手动传入，只需要在绘图时传入如下参数，参数可以接受 Python 中的列表或者二元元组形式的赋值：

```
1 p = figure(x_range=[0,10],
2           y_range=(10, 20))
```

## 2.6.5 相关资源 Resources

若要在用户本地展示图形，要求所使用的浏览器加载 Bokeh 的 JS 和 CSS 文件。默认情况下，`output_file()` 函数会自动在生成的 HTML 中加载来自 <http://cdn.pydata.org> 的样式文件。但是，你也可以将文件下载下来，之后再本地载入，若要使用这种方式请在 `output_file()` 函数中添加参数 `mode="inline"`。

## 2.7 更多例子 More examples

下面给出更多在别的场合下经常会用到的例子，所有这些例子都是用 `bokeh.plotting` 接口绘制的。

### 2.7.1 参数向量化 Vectorized colors and sizes

这个例子主要演示如何传入一系列的数据给绘图参数如 `fill_color` 和 `radius`。你也可以在这个例子中找到以下情况的使用法：

- 传入一个包含工具名称的列表给 `figure()`
- 用 `mode` 参数从 CDN 获取 BokehJS 文件
- 自定义 `x_range` 和 `y_range` 参数
- 传入空值 `None` 来擦掉一条曲线
- 运用 Numpy 数组来传入数据

```
1 import numpy as np
2 from bokeh.plotting import figure, \
3   output_file, show
4 # 生成数据（随机位置与随机半径 -> 颜色分布）
5 N = 4000
6 x = np.random.random(size=N) * 100
7 y = np.random.random(size=N) * 100
8 radii = np.random.random(size=N) * 1.5
9 colors = [
10     "#%02x%02x%02x" % (int(r), int(g), 150)
11     for r, g in zip(50+2*x, 30+2*y)]
```

```
12 # 生成静态HTML（利用 CDN 资源）
13 output_file("start3.html",
14             title="color_scatter.py example",
15             mode="cdn")
16 # 制定交互工具
17 TOOLS="resize, crosshair, pan, wheel_zoom, \
18       box_zoom, reset, box_select, lasso_select"
19 # 创建图层
20 p = figure(tools=TOOLS, x_range=(0,100),
21           y_range=(0,100))
22 # 通过矢量进行圆形渲染（颜色和大小）
23 p.circle(x,y, radius=radii,
24         fill_color=colors,
25         fill_alpha=0.6, line_color=None)
26 # 显示结果
27 show(p)
```

其中，CDN 的全称是 Content Delivery Network，即内容分发网络。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。作为结果，您将看到以下图像：

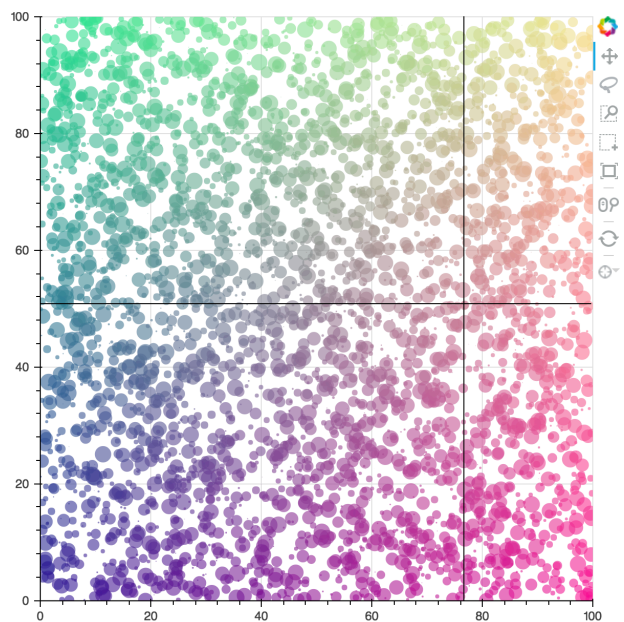


Figure 3: Bokeh 快速画图 3

### 2.7.2 数据联动 Linked panning and brushing

联动数据：数据联动指的是一个图形中的数据变化会影响到另一个图形中的数据。有时候在不同的图形中添加联动的数据曲线更有利于战术数据。在 Bokeh 中进行数据联动，一般只需在几个图形间传递将组件参数即可。下面的例子演示了在 Bokeh 的笔工具联动 \*\* (linked panning) \*\*，方法是在图形间传递几个组件参数。你也可以在下面的例子中找到以下情

况的用法:

- 通过多次调用 `figure()` 函数来创建多个图形
- 用 `gridplot()` 函数来排列多个图形
- 用 `Figure.triangle` 和 `Figure.square` 在图形中添加新的标识
- 通过调节 `toolbar_location` 的值为 `None` 来隐藏工具栏
- `line_color`、`fill_color`、`line_alpha`、`fill_alpha` 属性的用法

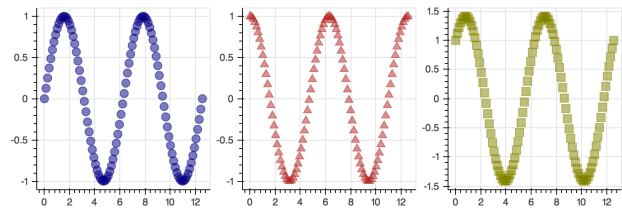


Figure 4: Bokeh 快速画图 4

**姿态联动 Postural linkage** 姿态联动是指当您拖动一个子图层时，其他子图层上的渲染会随着被拖动的图层的移动而移动，故称姿态联动。

```
1 import numpy as np
2 from bokeh.layouts import gridplot
3 from bokeh.plotting import figure, \
4     output_file, show
5 # 数据生成 (正余弦以及叠加)
6 N = 100
7 x = np.linspace(0, 4*np.pi, N)
8 y0 = np.sin(x)
9 y1 = np.cos(x)
10 y2 = np.sin(x) + np.cos(x)
11 # 生成静态HTML
12 output_file("start4.html")
13 # 创建图层
14 s1 = figure(width=250, plot_height=250,
15             title=None)
16 # 图层渲染
17 s1.circle(x, y0, size=10,
18           color="navy", alpha=0.5)
19 # 新知识: 叠加新图层共享数据范围, 再次渲染
20 s2 = figure(width=250, height=250,
21             x_range=s1.x_range,
22             y_range=s1.y_range,
23             title=None)
24 s2.triangle(x, y1, size=10,
25            color="firebrick", alpha=0.5)
26 # 新知识: 再次叠加图, 这次只共享一个数据范围
27 s3 = figure(width=250, height=250,
28             x_range=s1.x_range,
29             title=None)
30 s3.square(x, y2, size=10,
31           color="olive", alpha=0.5)
32 # 新知识: 三个图层排布, 隐藏工具栏
33 p = gridplot([[s1, s2, s3]],
34               toolbar_location=None)
35 # 显示结果
36 show(p)
```

您将看到以下结果并尝试拖动:

**选择联动 Selected linkage** 选择联动是指选择方形选择工具 (box select) 或者套索选择工具 (lasso select) 在其中一个图形中进行操作，这个操作会联动另一个图形。实现的方式是在两个图形中传递 `ColumnDataSource` 数据结构参数。

```
1 import numpy as np
2 from bokeh.plotting import *
3 from bokeh.models import ColumnDataSource
4 # 生成数据
5 N = 300
6 x = np.linspace(0, 4*np.pi, N)
7 y0 = np.sin(x)
8 y1 = np.cos(x)
9 # 输出静态HTML
10 output_file("start5.html")
11 # 新知识: 自定义一组数据属性用来分享
12 source = ColumnDataSource(
13     data=dict(x=x, y0=y0, y1=y1))
14 TOOLS = "pan,wheel_zoom,box_zoom,
15         reset,save,box_select,lasso_select"
16 # 生成一个图层并渲染
17 left = figure(tools=TOOLS, width=350,
18              height=350, title=None)
19 left.circle('x', 'y0', source=source)
20
21 # 生成另一个图层并渲染
22 right = figure(tools=TOOLS, width=350,
23               height=350, title=None)
24 right.circle('x', 'y1', source=source)
25 # 排布子图层
26 p = gridplot([[left, right]])
27 # 显示结果
28 show(p)
```

你将看到以下结果并尝试拖动:



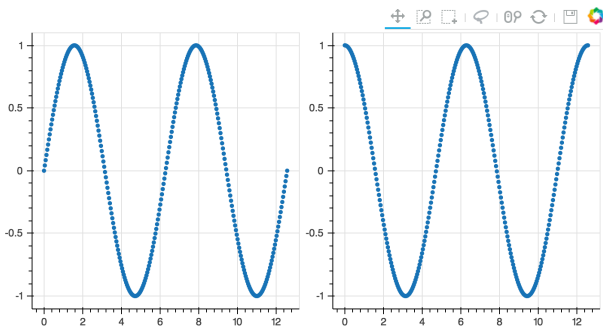


Figure 5: Bokeh 快速画图 5

### 2.7.3 时间序列图形 Datetime axes

处理时间序列数据是数据分析中另一常见的情形。Bokeh 有一个成熟的类 `DatetimeAxis`，可以根据现有的图形来生成时间序列图形。`DatetimeAxis` 的一些坐标轴的参数有默认值，当然你也可以不用 `DatetimeAxis` 而看情况直接将 `figure()` 中 `x_axis_type` 或 `y_axis_type` 的值设为 "datetime" 来说明绘图的类型是时间序列。

这个例子中也有一些有趣的用法：

- 设置 `figure()` 函数中的 `width` 和 `height` 参数
- 通过改变参数自定义图形和其他对象
- 通过设置 `figure` 的参数（如 `legend`、`grid`、`xgrid`、`ygrid`、`axis`、`xaxis`、`yaxis`）来调整图形中的注释。

```
1 import numpy as np
2 from bokeh.plotting import figure, \
3     output_file, show
4 from bokeh.sampledata.stocks import AAPL
5 # 生成数据
6 aapl = np.array(AAPL['adj_close'])
7 aapl_dates = np.array(AAPL['date'],
8     dtype=np.datetime64)
9 window_size = 30
10 window = np.ones(window_size)/\
11     float(window_size)
12 aapl_avg = np.convolve(aapl, window, 'same')
13 # 输出静态HTML
14 output_file("start6.html",
15     title="stocks.py example")
16 # 新建格式为datetime的图层
17 p = figure(width=800, height=350,
18     x_axis_type="datetime")
19 # 添加渲染
20 p.circle(aapl_dates, aapl, size=4,
21     color='darkgrey', alpha=0.2,
22     legend='close')
23 p.line(aapl_dates, aapl_avg,
24     color='navy', legend='avg')
```

```
25 # 新知识：通过设置属性自定义样式
26 p.title.text = "AAPL One-Month Average"
27 p.legend.location = "top_left"
28 p.grid.grid_line_alpha=0
29 p.xaxis.axis_label = 'Date'
30 p.yaxis.axis_label = 'Price'
31 p.ygrid.band_fill_color="olive"
32 p.ygrid.band_fill_alpha = 0.1
33 # 显示结果
34 show(p)
```

这里绘图需要下载一些官方资源，已经下载的上述代码运行不会出问题，如果上述代码运行出错请根据提示下载资源。

```
1 from bokeh.sampledata import download
2 download()
```

您将会看到结果如下：

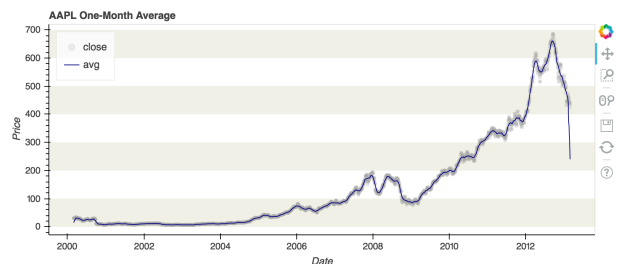


Figure 6: Bokeh 快速画图 6

## 2.8 Bokeh 图形服务器 Bokeh Plot Serve

Bokeh 服务器是一个可选的组件项。尽管没有 Bokeh 图形服务器，我们一样可以创建出有趣、可交互的可视化数据。但是 Bokeh 还有一些新颖、强大的能力或许你会想到：

- 用 UI 和选项来驱动图形的计算和更新
- 能够降低大型数据集采样率的智能服务端
- 通过流数据自动更新图形
- 适用于大数据的成熟图形重绘系统
- 发布更易于普通用户操作的可视化图形

由于空间有限，不能在快速入门中讲解所有的用法，读者可以现在下面这个例子中感受一些 Bokeh 图形服务器的强大。

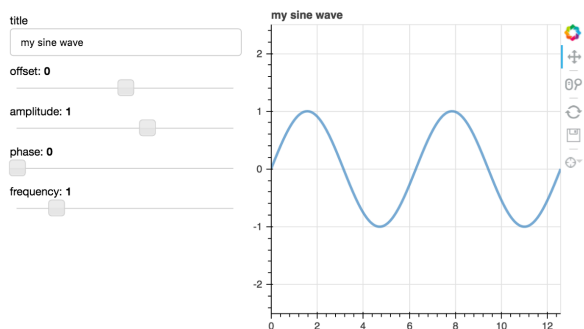


Figure 7: Bokeh 快速画图 7

你可以在 [Gallery](#) 的 [Server App Examples](#) 部分中找到更多有关图形服务器绘图例子。想要知道更多图形服务器的细节请移步 User Guide 的 [Running a Bokeh Server](#) 部分。

### 3 设置 Getting Set Up

这一章的第一部分会指引你快速安装一下 Bokeh 并做一些小测试来检测一下是否安装正常。

#### 3.1 安装 Bokeh 库 Installing the Bokeh Library

安装 Bokeh 最简单的方法就是用诸如 pip、conda 等包管理工具来安装。如果你用过 Anaconda 那么你可以用 conda 来安装。

```
1 conda install bokeh
```

否则，就用 pip 工具来安装

```
1 pip install bokeh
```

如果要查看更多安装方面的细节可以参考 Bokeh 文档的 [安装](#) 部分。

#### 3.2 安装确认 Verifying your installation

您可以在 Python 中先运行 `import bokeh`，再通过 `bokeh.__version__` 调出 bokeh 版本以确定您正确安装了 bokeh (不报错即正确安装)。如果您在最原始的 Python 编译器上运行，您会看到以下结果：

```
Python 3.4.3 [Continuum Analytics, Inc.] (default, Mar 6 2015, 12:07:41)
[GCC 4.2.1 (Apple Inc. build 5577)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import bokeh
>>> bokeh.__version__
'0.8.2'
>>>
```

Figure 8: Bokeh 设置 1

进一步确认，您可以尝试着绘制一下最初级的图形，运行以下代码，可以在解释器中运行也可以在 IDE 中运行：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 output_file("setting1.html")
4 p = figure()
5 p.line([1, 2, 3, 4, 5], [6, 7, 2, 4, 5]
6         , line_width=2)
7 show(p)
```

这会在本地产生一个 `setting1.html` 文件，然后弹出一个包含了绘图结果网页，您将会看到以下结果：

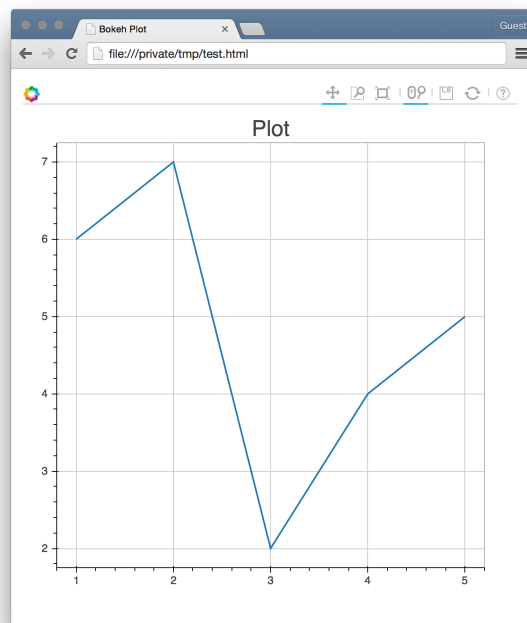


Figure 9: Bokeh 设置 2

当然，您也可以在 Python 的 notebook——Jupyter 上验证您的安装，不同的是，您需要将上述代码中的 `output_file` 变为 `output_notebook`，您将会看到：

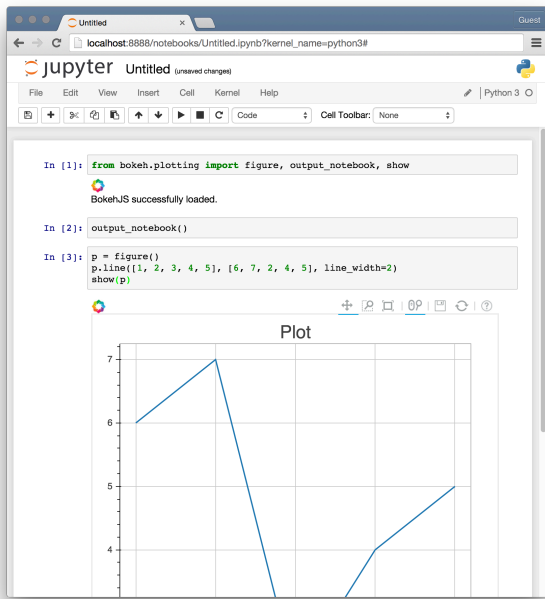


Figure 10: Bokeh 设置 3

## 4 一些重要概念 Defining Key Concepts

### 4.1 术语表 Glossary

为了更好的使用用户指南，了解相关高级概念很重要。下面是一份小术语表，介绍了一些 Bokeh 的重要概念。

- **应用-Application** : Bokeh 应用是一个从本地提交的，在网页上运行的 bokeh 文档。
- **BokehJS 文件-BokehJS** : Bokeh 的 Javascript 文件主要用于渲染图形和 UI 中的交互工具。一般用户不需要考虑 JavaScript 文件中的内容，但如果你知道一些关于 JavaScript 和 Bokeh 之间的联系的知识，那是最好不过的。更多细节可以查看[开发者指南](#)中的 **BokehJS**。
- **图表-Charts** : 示例中的静态图如柱状图，水平延伸的图和时间序列图等都可以由 Bokeh 提供的 **bokeh.charts** 高级接口来快速构建。更多示例及用法请查[构建高级图表](#)。
- **文件-Documents** : 文件指的是 Bokeh 应用所使用的数据集。文件中包含 Bokeh 应用渲染图表和交互工具所要用的所有模型和数据。
- **嵌入-Embedding** : 嵌入指的是在 Web 应用、网页或者 Jupyter (IPython) Notebook 中插入 Bokeh 的图表。更多细节请查看[插入图表或 Apps](#)。

- **标志-Glyphs** : 标志是构建 Bokeh 图形的基础元素，如曲线、三角形、方形、楔形、图示等都属于标志。**bokeh.plotting** 接口提供了便捷的方法创建自定义标志。更多细节请查看[Plotting with Basic Glyphs](#)。
- **模型-Models** : 模型是 Bokeh 中最底层的类，模型的作用就是组成 Bokeh 应用的整个“轮廓 (scene-graphs)”。这些类在 **bokeh.models** 接口里。大多数的用户不会直接接触到这些底层的类。然而最终的 Bokeh 应用是由这些类的集合组成的，所以理解它也是非常重要的，这样你可以更随心所欲的绘制 Bokeh 图形。更多信息请查看[Styling Visual Attributes](#)。
- **服务器-Server** : Bokeh 服务器是一个可选项，即使不使用 Bokeh 可以在浏览器中渲染交互图形。Bokeh 服务器主要用于发布或分享交流 Bokeh 图形或者应用，它的特点是可以处理大型流数据集，即提供实时图形服务。更多信息请查看[Running a Bokeh Server](#)。
- **挂件-Widgets** : 挂件指的是在图形外围的那些元素或服务。如滚动条、下拉式菜单、按钮等都属于挂件。与挂件交互会出发后台的一些函数，从而影响图形中的数据。挂件可以用在独立的 Bokeh 应用中也可以搭配 Bokeh 服务器一起使用。更多信息请查看[Adding Interactions](#)。

### 4.2 输出方式 Output Methods

我们可以看到用户指南中遍布的示例中有很多输出文件的方式，最常用的主要是以下几种：

- **output\_file** 用于生成独立的 Bokeh 图表 HTML 文件。
- **output\_notebook** 用于在 Jupyter notebook 上嵌入 Bokeh 图形
- **output\_server** 用于在一个运行着的 Bokeh 服务器安装 Bokeh 应用

这些函数经常回合 **show** 或者 **save** 搭配使用。描述一下下述代码的运行过程：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 output_file("output.html")
4 p = figure()
5 p.line(x=[1, 2, 3], y=[4,6,2])
6 show(p)
```



假设这个文件叫 `example`，在 `terminal` 里执行 `python example.py` 会生成一个包含 Bokeh 图形的 HTML 文件。这些函数搭配在交互式设置和从 Web 应用（如 Flask、Django 等）中分离出单独的 HTML 图形文件非常有用。

Bokeh 还提供一个强大的命令行工具——`bokeh`——可以用来生成多种输出：

- `bokeh html`：根据任何一种 Bokeh 应用的资源文件（如 Python 脚本、Bokeh 应用目录、JSON 文件）生成独立的 HTML 文件。
- `bokeh json`：根据任何一种 Bokeh 应用的资源文件生成相当于 Bokeh 文件的序列化 JSON 文件。
- `bokeh serve`：将 Bokeh 文件作为 Web 应用发布。

用 `bokeh` 命令（而不是直接运行 `.py` 文件）的好处是不需要在 Python 代码中指定输出方式，而通过命令来获得多种格式的输出。也就是一次编写，多次输出。上述的代码也可以简化（等效）为

```
1 from bokeh.plotting import figure
2 p = figure()
3 p.line(x=[1, 2, 3], y=[4,6,2])
```

现在您可以通过在命令行输入 `bokeh html example.py` 来生成独立的 HTML 文件，或者用 `bokeh serve example.py` 将 Bokeh 图形以 Web 应用的形式发布。更多信息请查看 [Using bokeh Commands](#)。

## 4.3 接口 Interfaces

Bokeh 不仅为数据科学家及相关领域的专家提供了快速便捷的接口，还为软件开发者和工程师们提供了更加丰富的接口来配置更多 Bokeh 成熟的特性。正因为这个特点，Bokeh 的方法是分等级的，不同等级的需求、特性对应不同等级的接口方法，而这些方法的使用方法基本都是相同的，这样就可以提高代码的复用率。这一节中主要向用户概述一些用户级的可用接口和一些中心概念。如果你想直接绘制图形，可以跳过这一章，转向 [Plotting with Basic Glyphs](#) 和 [Making High-level Charts](#)。

### 4.3.1 bokeh.models

Bokeh 实际上是由两个组件库构成的。

第一部分是在浏览器中运行的 JavaScript 库，BokehJS。这个库负责所有的页面渲染和用户交互。BokehJS 的输入是一个用来构成整个网页“轮廓”的 JSON 对象集。这些对象包括图形、挂件中你能看见的

所有东西。这些 JSON 对象在浏览器中最终会被转换成 Backbone 模型，并渲染成对应的 Backbone 可视化模型。

第二个部分是 Python 库（或者其他语言类型的库，这里以 Python 作为例子），这些库可以生成上述说的 JSON 对象。这些都是在库的底层完成的，这些 Python 类可以将代码中设置的内容和属性序列化 JSON 对象。所有这些低级模型都可以在低级接口 `bokeh.models` 中找到。大多数类都很简单，只有一些属性，设置没有方法。这些属性可以在创建模型的时候传入，或者在之后指定使用该属性的模型。下面以 `Rect` 标志对象为例给出例子：

```
1 # 模型建立时属性可以被确定
2 glyph = Rect(x="x", y="y2", w=10, h=20,\
3             line_color=None)
4
5 # 或者在后期指定属性
6 glyph.fill_alpha = 0.5
7 glyph.fill_color = "navy"
```

类似这样的配置活儿可以普遍的适用于所有 Bokeh 模型。加上 Bokeh 接口最终生成的都是模型集，所以所有图形、挂件的风格化和配置都可以通过这一方法来实现，并且不限制于使用的是哪个接口。

使用 `bokeh.models` 接口可以随意搭配图形和挂件，而这样做的坏处就是结果有时候可能并没有意义或者一塌糊涂，但至少，这可以让开发者亲自参与到构建“轮廓”的过程中来。因为这个原因，大多数用户可能更乐意去使用高级接口（下面会介绍到），除非他们有特殊的需求，而高级接口不提供这一需求。关于更多 Bokeh 模型的知识请移步 [Reference Guide](#)。

### 4.3.2 bokeh.plotting

`plotting` 接口是 Bokeh 的中级接口，这一接口的使用和 Matplotlib 还有 Matlab 中的绘制函数很像。他主要用于让用户选择合适于所用数据的标志，若不指定标志，则图形使用默认的坐标轴、网格线和工具来绘图。所有构建“轮廓”的艰难工序在这一接口中将自动完成。

`bokeh.plotting` 接口的中心类是 `Figure` 类。`Figure` 继承自 `Plot`，所以也能非常轻松的在图形中添加各类标志。除此之外，`Figure` 也能不费吹灰之力就生成默认坐标轴、网格线还有工具。一般来说，用户会通过调用 `figure()` 函数来创建一个 `Figure` 对象。

`bokeh.plotting` 一个典型的用法如下，在代码的下方附带了对应的图形结果：

```

1 from bokeh.plotting import figure, \
2   output_file, show
3 # 创建图层
4 p = figure(width=300, height=300, \
5   tools="pan,reset,save")
6 # 添加圆形渲染
7 p.circle([1, 2.5, 3, 2], [2, 3, 1, 1.5], \
8   radius=0.3, alpha=0.5)
9 # 指定文件
10 output_file("./concept1.html")
11 # 绘制图像
12 show(p)

```

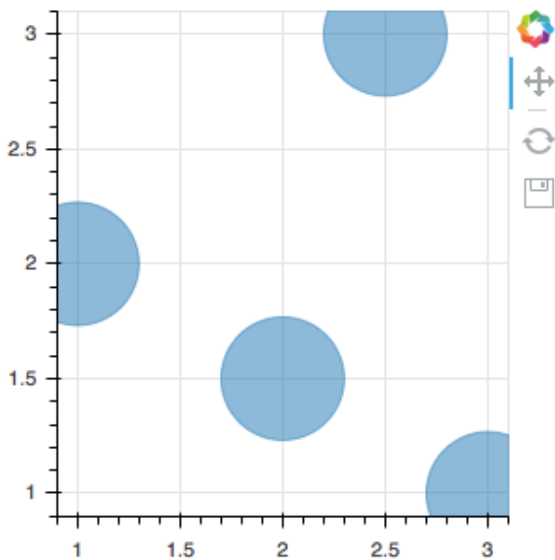


Figure 11: Bokeh 概念 1

注意观察代码，代码中用了 `figure()` 创建了图形，之后用 `Figure.circle` 在图中添加了圆标志。可以看见我们并不需要为配置坐标轴或者网格线而操心（如果需要，也可以自行配置），而且添加工具栏上的工具也只需要传入相应的名字即可。在代码的最后我们用了一些输出函数来将图形展示出来。

除了上面的，其实还有许多用法这里没有展示出来，比如不展示转而保存该图形、风格化或去掉坐标轴和网格线、增加别的曲线并将它们显示在一个图形中等。这些用法的示例你都可以在 [User Guide 的 Plotting with Basic Glyphs](#) 这一章中看到。

### 4.3.3 bokeh.charts

`bokeh.charts` 是 Bokeh 的一个高级接口，主要可以快速的绘制一些静态图表。跟 `bokeh.plotting` 一样，`bokeh.charts` 是通过整合一些较底层的类来简化创建图表的过程。虽然是简化了过程，但是你还是需要提

供必要的一些数据，一般情况下，这一接口中的函数是用来创建普通的示意静态图的，`bokeh.charts` 中的函数会自动为不同组的数据标记不同的颜色和其他一些标识。

`bokeh.charts` 中快速绘图的类型包括 `Bar()`、`Box-Plot()`、`Histogram()`、`TimeSeries()`（分别对应柱状图、箱图、直方图、时间序列图）等，下面以 `Scatter()` 作散点图为例给出代码：

```

1 from bokeh.charts import Scatter, \
2   output_file, show
3 # 准备数据，一个pandas的分组数据
4 from bokeh.sampledata.autompg import autmpg as df
5 # 建立散点图
6 p = Scatter(df, x='mpg', y='hp',
7   color='cyl',
8   title="MPG vs HP",
9   legend='top_right',
10  xlabel="Miles Per Gallon",
11  ylabel="Horsepower")
12 # 指明保存位置
13 output_file("./concept2.html")
14 # 绘图
15 show(p)

```

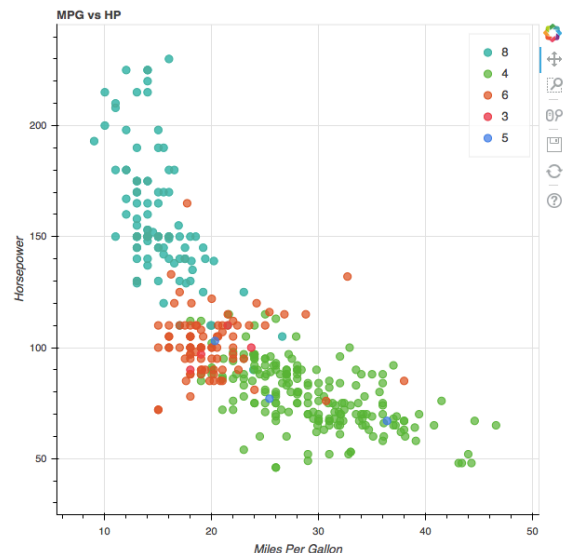


Figure 12: Bokeh 概念 2

需要指出的是输出函数像 `output_file()` 和 `show()` 之类的被普遍的用在各个绘图接口中，就像 `bokeh.plotting` 中也有 `output_file()` 和 `show()` 一样。他虽然是定义在 `bokeh.io` 模块中的，但是为了方便也可以从 `bokeh.charts` 中导入。

#### 4.3.4 其他接口 Other Interfaces

Bokeh 与 Matplotlib 具有一定的共存性，你可以用第三方库 `mplexporter` 绘制图形，之后再转化成 Bokeh 图形，虽然不能 100% 利用 Matplotlib 的特性，但是有时候 Bokeh 这一特点非常实用。除了刚刚说的 Matplotlib 图形，你也可以非常轻易的将用 Seaborn 和 `ggplot.py` 绘制出来的图形转化成 Bokeh 图形。下面这个例子只用了一行代码将 Seaborn 图形转化成了 Bokeh 图形。

```
1 from seaborn import violinplot
2 import seaborn as sns
3 from bokeh import mpl
4 from bokeh.plotting import \
5     output_file, show
6 tips = sns.load_dataset("tips")
7 sns.set_style("whitegrid")
8 ax = violinplot(x="day", y="total_bill",
9                hue="sex", data=tips,
10                palette="Set2", split=True,
11                scale="count",
12                inner="stick")
13 output_file("./images/concept3.html")
14 show(mpl.to_bokeh())
```

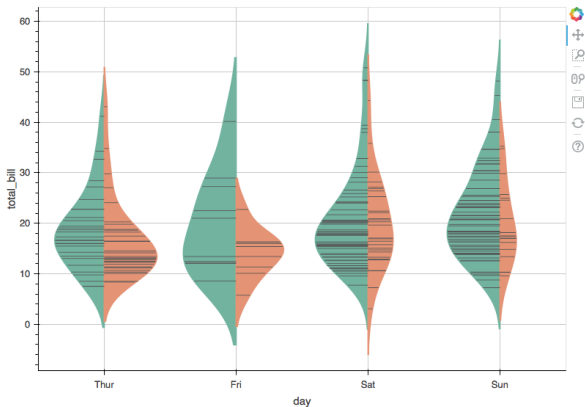


Figure 13: Bokeh 概念 3

## 5 用基础标志绘图 Plotting with Basic Glyphs

### 5.1 创建标志 Creating Figures

需要指出的是利用 `bokeh.plotting` 创建出的图形子带默认的工具和可视化风格。如果要风格化自己的图形请看 [Styling Visual Attributes](#)，若要组合不同工具请看 [Configuring Plot Tools](#)。

#### 5.1.1 散点图标 Scatter Markers

**Circle Scatter** 圆形散点图 要绘制散点图可以用 Figure 中的 `circle()` 函数：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 output_file("basic_glyphs1.html")
4 p = figure(plot_width=400, plot_height=400)
5 p.circle([1, 2, 3, 4, 5],
6          [6, 7, 2, 4, 5],
7          size=20, color="navy", alpha=0.5)
8 show(p)
```

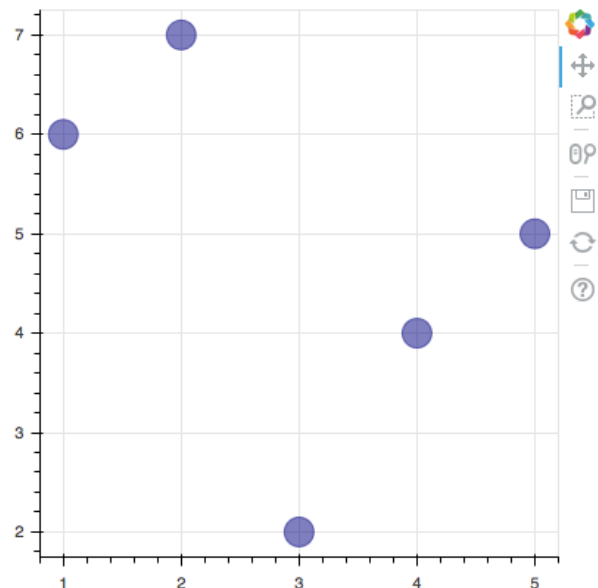


Figure 14: Bokeh 基础作图 1

**Square Scatter** 方形散点图 照着葫芦画瓢，如果要绘制方形的散点图可以用 Figure 中的 `square()` 函数：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 output_file("basic_glyphs2.html")
4 p = figure(plot_width=400, plot_height=400)
5 p.square([1, 2, 3, 4, 5],
6          [6, 7, 2, 4, 5],
7          size=20, color="olive", alpha=0.5)
8 show(p)
```

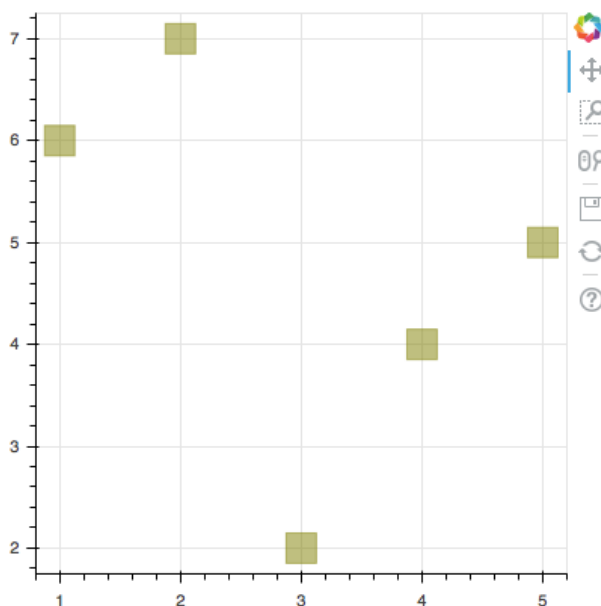


Figure 15: Bokeh 基础作图 2

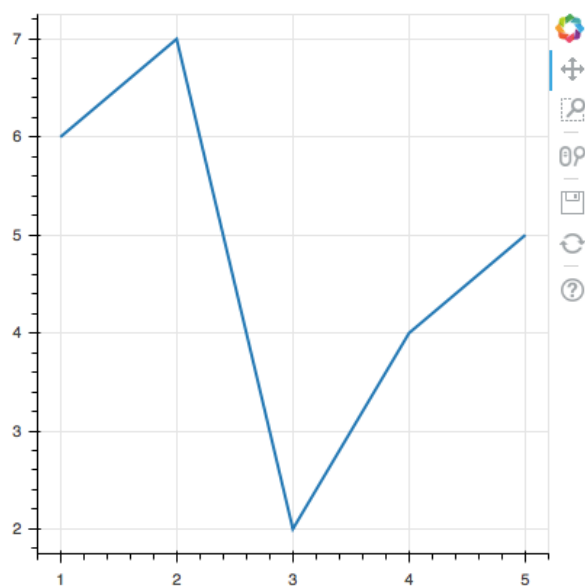


Figure 16: Bokeh 基础作图 3

**More** 更多散点图 还有许多类型的散点图可供选择，具体见下表：

Table 1: 散点形状

Function	Meaning
<code>x()</code>	十字散点图
<code>asterisk()</code>	交叉双十字散点图
<code>diamond()</code>	菱形散点图
<code>inverted_triangle()</code>	倒三角散点图

所有类型的图标都有 `x`、`y`、`size`（定义在 `screen units`）和 `angle`（默认单位是弧度）。除此之外，`circle()` 函数具有 `radius` 参数，用来调节 `data-space units`。

### 5.1.2 线形标志 Line Glyphs

**Single Lines** 单个直线 下面直接给出例子，向 `line()` 函数传入一系列 `x` 和 `y` 值来绘制一根直线：

```
1 from bokeh.plotting import figure,\
2     output_file, show
3 output_file("basic_glyphs3.html")
4 p = figure(plot_width=400,
5     plot_height=400)
6 p.line([1, 2, 3, 4, 5],[6, 7, 2, 4, 5],
7     line_width=2)
8 show(p)
```

```
1 from bokeh.plotting import figure,\
2     output_file, show
3 output_file("basic_glyphs4.html")
4 p = figure(plot_width=400,
5     plot_height=400)
6 p.multi_line([[1, 3, 2],[3, 4, 6, 6]],
7     [[2, 1, 4], [4, 7, 8, 5]],
8     color=["firebrick", "navy"],
9     alpha=[0.8, 0.3], line_width=4)
10 show(p)
```

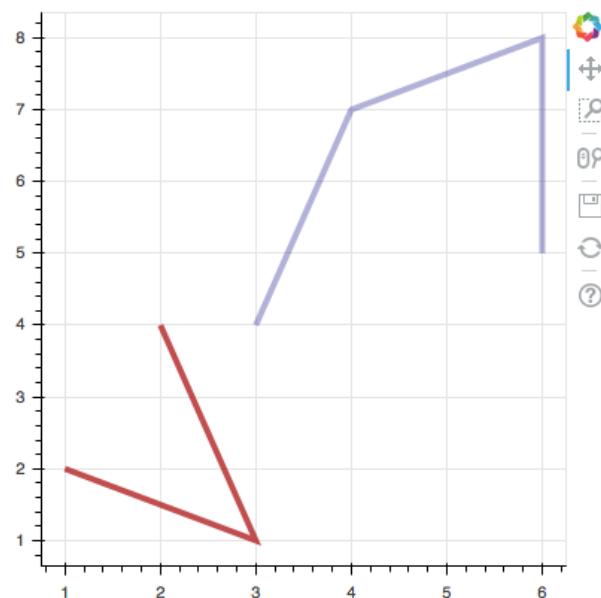


Figure 17: Bokeh 基础作图 4

**Missing Points** 隐藏直线段 NaN 值可以传入 `line()` 和 `multi_line()` 函数。在下面的例子中我们将逻辑上为一条的直线中的一部分隐藏，使其实际上变成两条直线。（值为 NaN 的那一段直线就不进行插值了，所以就没有直线）：

```
1 from bokeh.plotting import figure,\
2     output_file, show
3 output_file("basic_glyphs5.html")
4 p = figure(plot_width=400,
5     plot_height=400)
6 nan = float('nan')
7 p.line([1, 2, 3, nan, 4, 5],
8     [6, 7, 2, 4, 4, 5], line_width=2)
9 show(p)
```

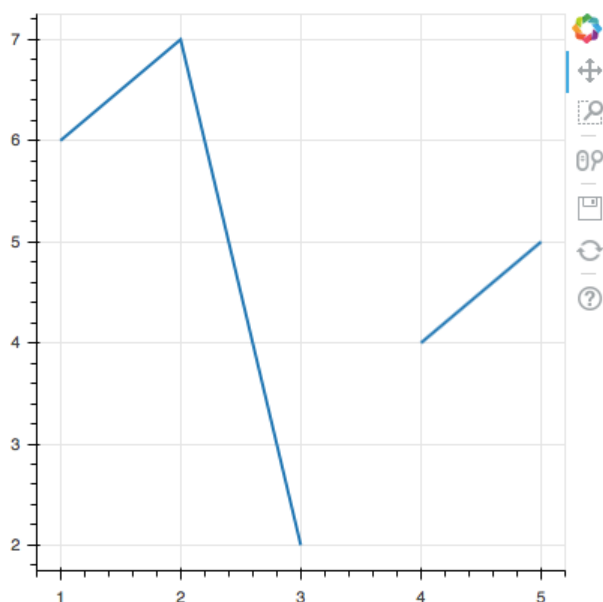


Figure 18: Bokeh 基础作图 5

### 5.1.3 贴图标志 Patch Glyphs

**Single Patches** 单个贴图 下面的例子示范的是如何向 `patch()` 函数传入两组一维 Python 列表来生成单个多边形的贴图。其中两个一维 Python 列表中的值对应位组成一个点的坐标。

```
1 from bokeh.plotting import figure,\
2     output_file, show
3 output_file("basic_glyphs6.html")
4 p = figure(plot_width=400,
5     plot_height=400)
6 p.patch([1, 2, 3, 4, 5],
7     [6, 7, 8, 7, 3], alpha=0.5,
8     line_width=2)
9 show(p)
```

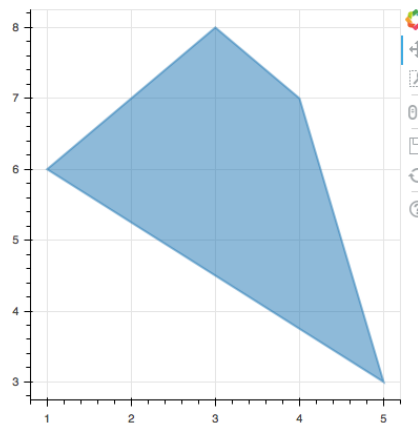


Figure 19: Bokeh 基础作图 6

**Multiple Patches** 多个贴图 如果要在一个图形中绘制多个贴图，可以使用 `patches()` 方法，方式如下：

```
1 from bokeh.plotting import figure,\
2     output_file, show
3 output_file("basic_glyphs7.html")
4 p = figure(plot_width=400,
5     plot_height=400)
6 p.patches([[1, 3, 2], [3, 4, 6, 6]],
7     [[2, 1, 4], [4, 7, 8, 5]],
8     color=["firebrick", "navy"],
9     alpha=[0.8, 0.3], line_width=2)
10 show(p)
```

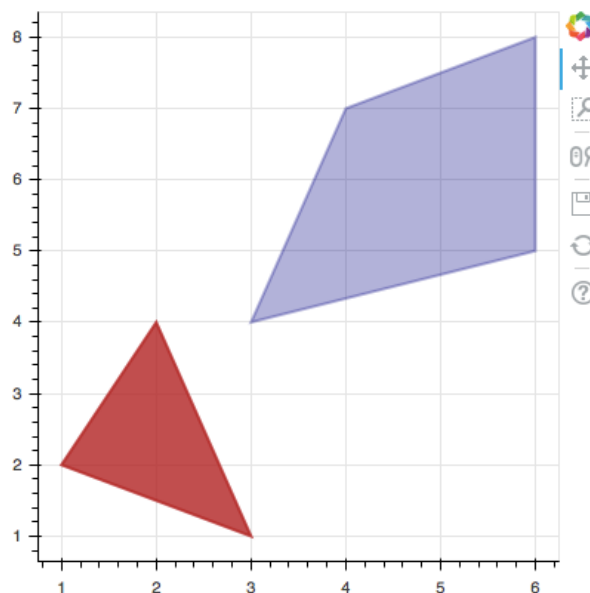


Figure 20: Bokeh 基础作图 7

**Missing Points** 隐藏部分贴图 和 `line()`、`multi_line()` 一样，NaN 值也可以被传入 `patch()`、`patches()` 中。在下面的例子中，我们将一个逻辑上为



一块的贴图分成两块（实际上相当于将 NaN 两边的数据分开绘制贴图，只不过这两个贴图颜色一样）：

```
1 from bokeh.plotting import figure,\
2   output_file, show
3 output_file("basic_glyphs8.html")
4 p = figure(plot_width=400,plot_height=400)
5 nan = float('nan')
6 p.patch([1, 2, 3, nan, 4, 5, 6],
7         [6, 7, 5, nan, 7, 3, 6],
8         alpha=0.5,
9         line_width=2)
10 show(p)
```

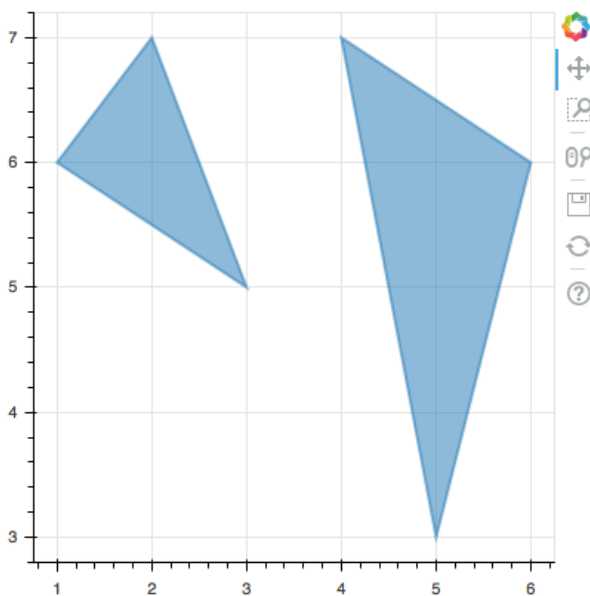


Figure 21: Bokeh 基础作图 8

**Rectangles, Ovals and Ellipses** 长方形与椭圆形要在图形中绘制和网格对齐的四边形可以使用 `quad()` 标志函数，`quad()` 函数接受 `left`、`right`、`top`、`bottom` 这四个参数，分别表示四边形的四个边对应的坐标。

```
1 from bokeh.plotting import figure,\
2   show, output_file
3 output_file('basic_glyphs9.html')
4 p = figure(width=400, height=400)
5 p.quad(top=[2, 3, 4],bottom=[1, 2, 3],
6        left=[1, 2, 3],
7        right=[1.2, 2.5, 3.7],
8        color="#B3DE69")
9 show(p)
```

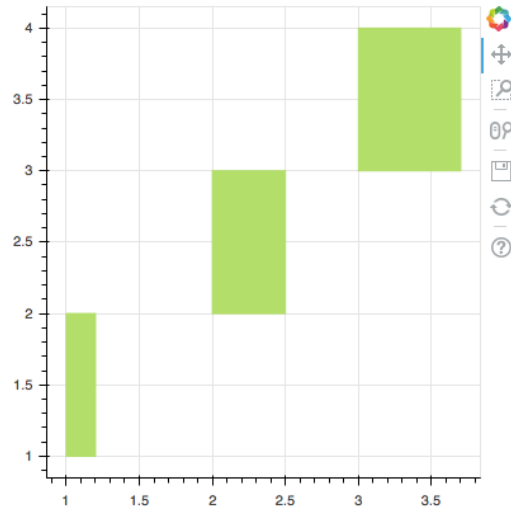


Figure 22: Bokeh 基础作图 9

如果要绘制任意一种四边形，可以用 `rect()` 函数并指定需要的长、宽、角度来画出需要的标志。

```
1 from math import pi
2 from bokeh.plotting import figure,\
3   show, output_file
4 output_file('basic_glyphs10.html')
5 p = figure(width=400, height=400)
6 p.rect(x=[1, 2, 3], y=[1, 2, 3],
7        width=0.2, height=40, color="#CAB2D6",
8        angle=pi/3, height_units="screen")
9 show(p)
```

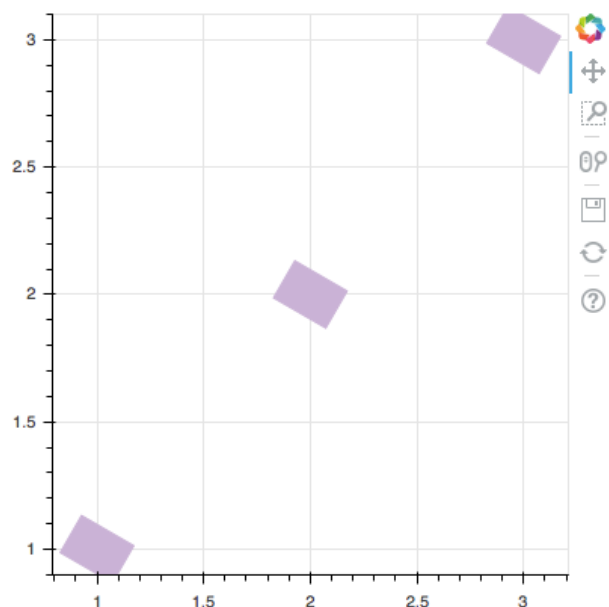


Figure 23: Bokeh 基础作图 10

`oval()` 标志函数可接受的参数与 `rect()` 一样：

```

1 from math import pi
2 from bokeh.plotting import figure,\
3     show, output_file
4 output_file('basic_glyphs11.html')
5 p = figure(width=400, height=400)
6 p.oval(x=[1, 2, 3], y=[1, 2, 3],
7        width=0.2, height=40, color="#CAB2D6",
8        angle=pi/3, height_units="screen")
9 show(p)

```

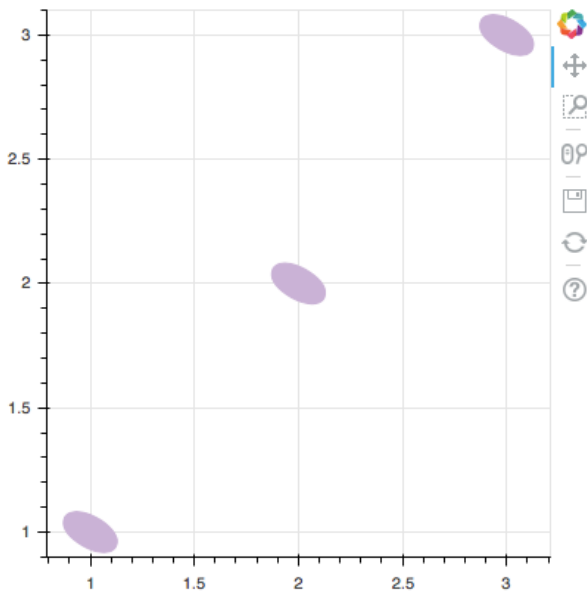


Figure 24: Bokeh 基础作图 11

`ellipse()` 标志函数接受的参数和 `oval()`、`rect()` 并无二样, `ellipse()` 和 `oval()` 函数绘制的都是椭圆形, 只是 `ellipse()` 绘制的椭圆形更胖一点:

```

1 from math import pi
2 from bokeh.plotting import figure,\
3     show, output_file
4 output_file('basic_glyphs12.html')
5 p = figure(width=400, height=400)
6 p.ellipse(x=[1, 2, 3], y=[1, 2, 3],
7           width=[0.2, 0.3, 0.1], height=0.3,
8           angle=pi/3, color="#CAB2D6")
9 show(p)

```

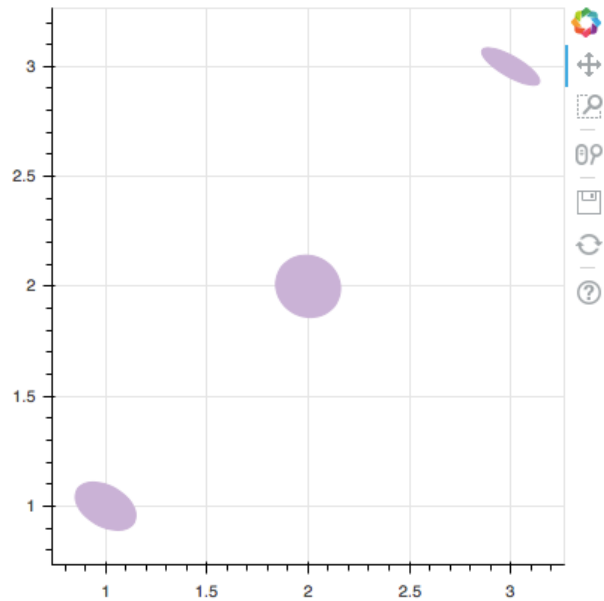


Figure 25: Bokeh 基础作图 12

**Images** 图片 要在图形中显示自己的图片, 可以用 `image()`、`image_rgba()` 和 `image_url()` 标志函数。

下面这个例子演示如何用 `image_rgba()` 在图形中嵌入 raw RGBA 数据图。(注意: 这个例子中使用 Numpy 库来简化数据的生成。)

```

1 from __future__ import division
2 import numpy as np
3 from bokeh.plotting import figure,\
4     output_file, show
5 # 创建一组RGBA数据
6 N = 20
7 img = np.empty((N, N), dtype=np.uint32)
8 view = img.view(dtype=np.uint8).\
9     reshape((N, N, 4))
10 for i in range(N):
11     for j in range(N):
12         view[i, j, 0] = int(255 * i / N)
13         view[i, j, 1] = 158
14         view[i, j, 2] = int(255 * j / N)
15         view[i, j, 3] = 255
16 output_file("basic_glyphs13.html")
17 p = figure(plot_width=400, plot_height=400,
18           x_range=(0, 10), y_range=(0, 10))
19 p.image_rgba(image=[img], x=[0],
20             y=[0], dw=[10], dh=[10])
21 show(p)

```

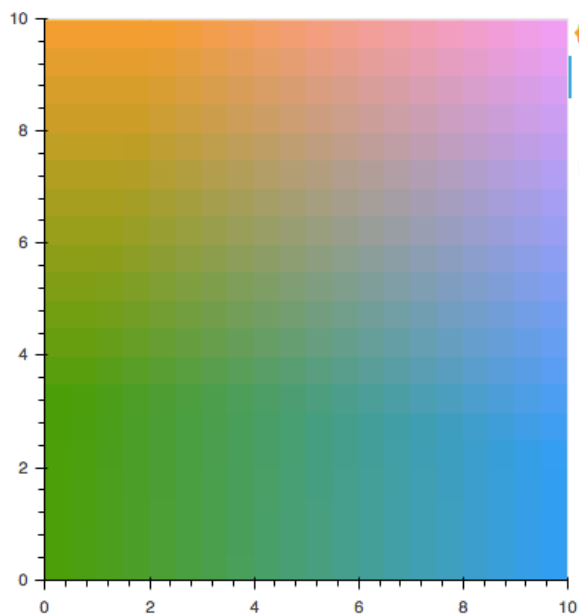


Figure 26: Bokeh 基础作图 13

**Segments and Rays** 线段与射线 有时候在一个图形中绘制多个线段或者射线会非常有用，Bokeh 提供了 `segment()` 和 `ray()` 标志函数来完成这一需求。

`segment()` 函数接受 `x0`、`y0`、`x1`、`y1`（分别对应起始 `x`、`y` 坐标和终点 `x`、`y` 坐标）：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 p = figure(width=400, height=400)
4 output_file('basic_glyphs14.html')
5 p.segment(x0=[1, 2, 3], y0=[1, 2, 3],
6           x1=[1.2, 2.4, 3.1],
7           y1=[1.2, 2.5, 3.7],
8           color="#F4A582", line_width=3)
9 show(p)
```

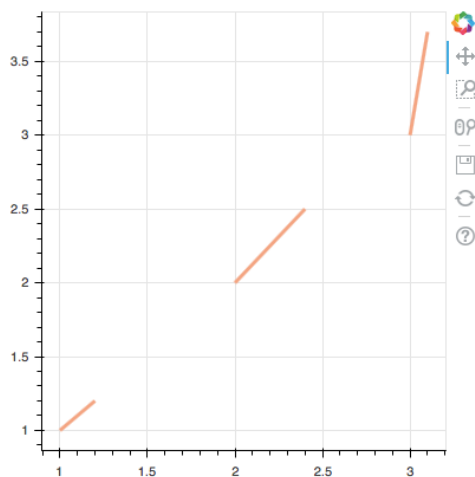


Figure 27: Bokeh 基础作图 14

`ray()` 函数接受 `x`、`y`、`length`、`angle`（分别表示起始 `x`、`y` 坐标、长度、角度）这几个参数。其中角度的默认单位为“弧度”（但也可以用角度），长度单位为屏幕像素，如果要绘制无限长的射线，将 0 传给 `length` 参数。

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 p = figure(width=400, height=400)
4 output_file('basic_glyphs15.html')
5 p.ray(x=[1, 2, 3], y=[1, 2, 3],
6       length=45, angle=[30, 45, 60],
7       angle_units="deg",
8       color="#FB8072", line_width=2)
9 show(p)
```

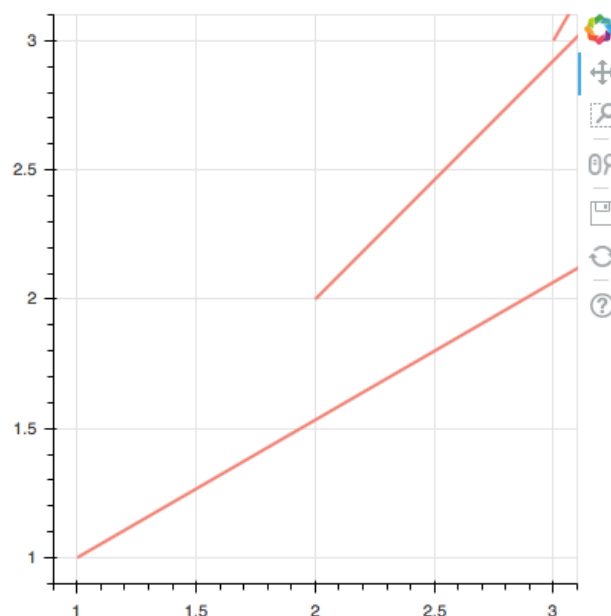


Figure 28: Bokeh 基础作图 15

**Wedges and Arcs** 楔形与弧形 `arc()` 标志函数能够绘制简单的弧形，他接受 `radius`、`start_angle` 和 `end_angle`（分别表示半径、起始角度、终点角度，上面例子中的角度的单位是弧度），该函数还有个可选参数 `direction`，可以传入“clock”（顺时针绘制）或者“anticlock”（逆时针绘制）参数。

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 p = figure(width=400, height=400)
4 output_file('basic_glyphs16.html')
5 p.arc(x=[1, 2, 3], y=[1, 2, 3],
6       radius=0.1, start_angle=0.4,
7       end_angle=4.8, color="navy")
8 show(p)
```

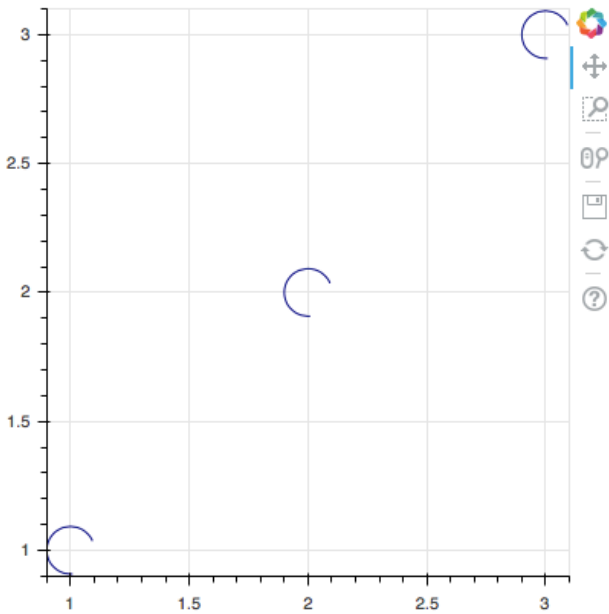


Figure 29: Bokeh 基础作图 16

wedge() 函数接受的参数和 arc() 一样，但是其绘制的是扇形：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 p = figure(width=400, height=400)
4 output_file('basic_glyphs17.html')
5 p.wedge(x=[1, 2, 3], y=[1, 2, 3],
6         radius=0.2, start_angle=0.4,
7         end_angle=4.8, color="firebrick",
8         alpha=0.6, direction="clock")
9 show(p)
```

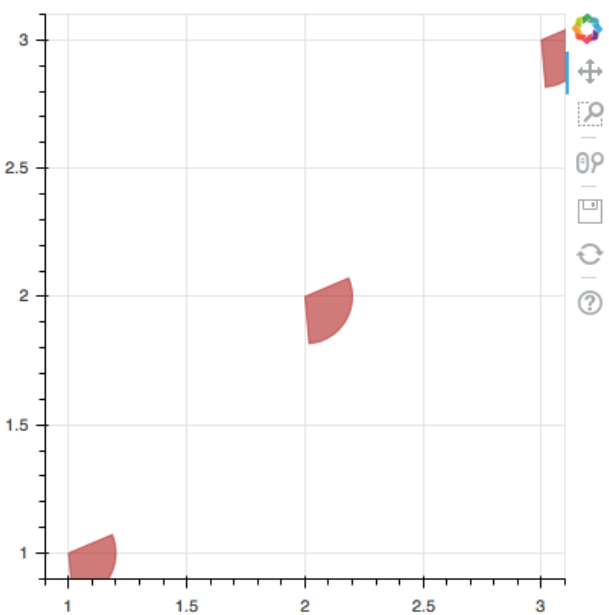


Figure 30: Bokeh 基础作图 17

annular\_wedge() 函数和 arc() 函数很像，但绘制的图形是环形，annular\_wedge() 接受 inner\_radius 和 outer\_radius 参数（分别对应内半径和外半径），而非 radius：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 p = figure(width=400, height=400)
4 output_file('basic_glyphs18.html')
5 p.annular_wedge(x=[1, 2, 3], y=[1, 2, 3],
6                inner_radius=0.1, outer_radius=0.25,
7                start_angle=0.4, end_angle=4.8,
8                color="green", alpha=0.6)
9 show(p)
```

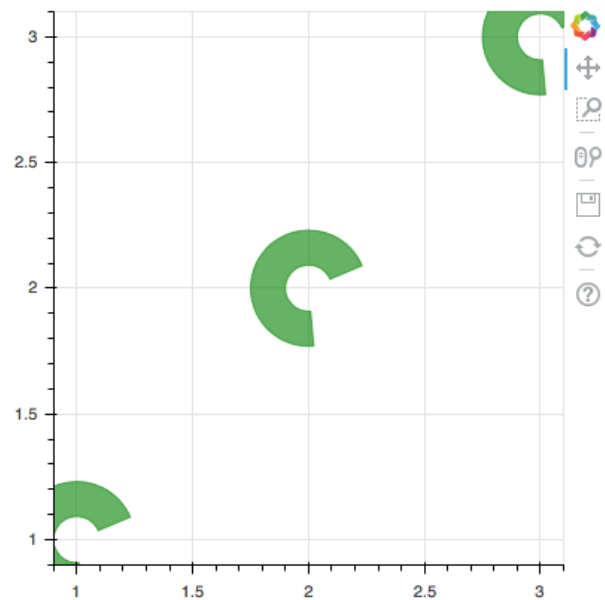


Figure 31: Bokeh 基础作图 18

最后是 annulus() 标志函数，该函数只能绘制整个环形，用法如下：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 p = figure(width=400, height=400)
4 output_file('basic_glyphs19.html')
5 p.annulus(x=[1, 2, 3], y=[1, 2, 3],
6           inner_radius=0.1, outer_radius=0.25,
7           color="orange", alpha=0.6)
8 show(p)
```

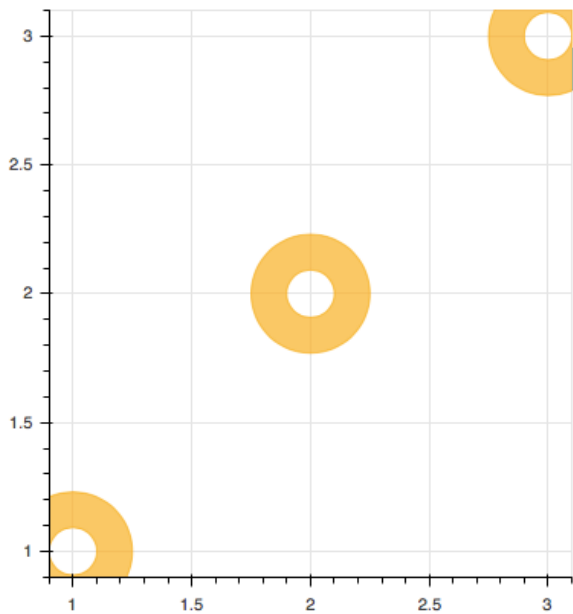


Figure 32: Bokeh 基础作图 19

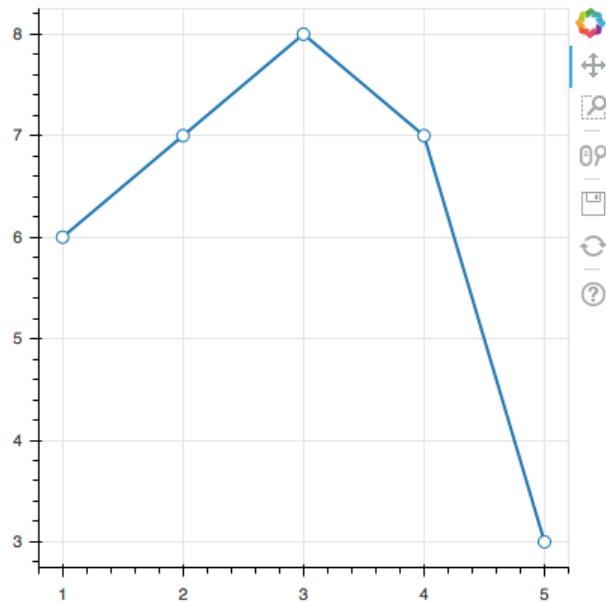


Figure 33: Bokeh 基础作图 20

所有在 `bokeh.plotting` 中的标志函数都遵循这一原则，同一个图形中可以添加任意多个标志。

**Specialized Curves** 特殊曲线 Bokeh 还提供了 `quadratic()` 和 `bezier()` 标志函数来绘制二次和三次曲线。有时候还有一些特例情况，详情请看 [reference documentation](#)。

## 5.2 组合标志 Combining Multiple Glyphs

要在同一图形中绘制多种标志可以直接在图形对象 (Figure) 中添加:

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 x = [1, 2, 3, 4, 5]
4 y = [6, 7, 8, 7, 3]
5 output_file("basic_glyphs20.html")
6 p = figure(plot_width=400,
7     plot_height=400)
8 # 同时添加线和点
9 p.line(x, y, line_width=2)
10 p.circle(x, y, fill_color="white",
11     size=8)
12 show(p)
```

## 5.3 设置坐标范围 Setting Ranges

一般情况下, Bokeh 会根据曲线标志自适应的设置坐标范围, 但你也可以用 `Range1d` 方法来改变 `x_range` 和 `y_range` 来设置坐标范围, `Range1d` 方法接受两个参数, 分别为坐标的起始点和终点。

```
1 p.x_range = Range1d(0, 100)
```

为方便起见, `figure()` 方法也能接受形式为 (起始点, 终点) 的元组作为属性 `x_range` 或 `y_range` 的值, 而这两个属性能改变图形的坐标轴范围, 下面的例子演示了两种修改坐标轴范围的方法:

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 from bokeh.models import Range1d
4 output_file("basic_glyphs21.html")
5 # 修改x轴范围
6 p = figure(plot_width=400,
7     plot_height=400, x_range=(0, 20))
8 # 修改y轴范围
9 p.y_range = Range1d(0, 15)
10 p.circle([1, 2, 3, 4, 5],
11     [2, 5, 8, 2, 7], size=10)
12 show(p)
```



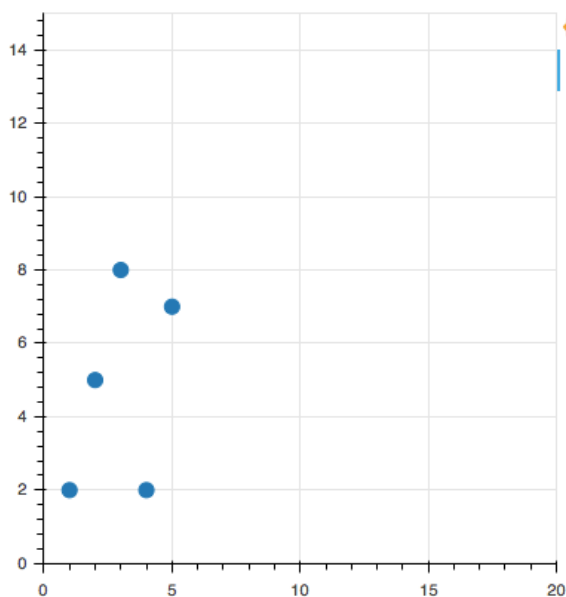


Figure 34: Bokeh 基础作图 21

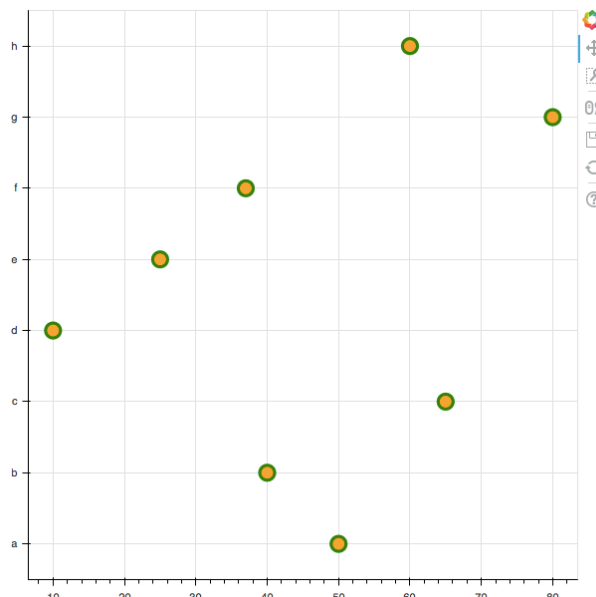


Figure 35: Bokeh 基础作图 22

你也可以设置一个范围，限制用户使用 `pan` 或 `zoom` 工具的范围。

## 5.4 配置坐标轴 Specifying Axis Types

上面所有的例子所使用的坐标轴都是线性的。线性坐标轴适用于大多数图形，但有时候时间轴和对数轴更有用，这一节中将讲述如何配置 `bokeh.plotting` 图形的坐标轴类型。

**Categorical Axes 类轴** 顾名思义，即以类别作为值得坐标轴，可以先定义 `factor` 列表作为“y”值进行输入：

```
1 from bokeh.plotting import figure, \
2     output_file, show
3 factors = ["a", "b", "c", "d",
4     "e", "f", "g", "h"]
5 x = [50, 40, 65, 10, 25, 37, 80, 60]
6 output_file("basic_glyphs22.html")
7 p = figure(y_range=factors)
8 p.circle(x, factors, size=15,
9     fill_color="orange", line_color="green",
10    line_width=3)
11 show(p)
```

**Datetime Axes 时间坐标轴** 当处理的数据与时间有关时，使用时间坐标轴是个很好的选择。之前我们已经学习过了如何使用 `bokeh.plotting` 的 `figure()` 函数来绘制图形，其实 `figure()` 函数还接受 `x_axis_type` 和 `y_axis_type` 参数，分别表示 x 和 y 轴的类型。要配置成时间坐标轴，传入 `"datetime"` 参数即可。（注意：下面的例子需要联网才能正常运行，并且，为了更方便的展示数据，会用到第三方 `Tushare` 库。）

```
1 import tushare as ts
2 from bokeh.plotting import figure, \
3     output_file, show
4 PuFa = ts.get_k_data('600000')
5 PuFa['date'] = pd.to_datetime(PuFa['date'])
6 output_file("basic_glyphs23.html")
7 # 设定时间坐标轴
8 p = figure(width=800, height=250,
9     x_axis_type="datetime")
10 p.line(PuFa['date'], PuFa['close'],
11     color='navy', alpha=0.5)
12 show(p)
```



Figure 36: Bokeh 基础作图 23

**Log Scale Axes** 对数坐标轴 当处理指数级增长等快速增长的数据时，用指数坐标轴能更准确方便的表示图形，有的情况数据增长的很快，以致于两个轴都要用指数坐标轴来表示。

和上面的方法类似，要配置指数坐标轴，向 figure() 函数的 x\_axis\_type 和 y\_axis\_type 属性传入 "log" 值即可：

```
1 from bokeh.plotting import figure,\
2     output_file, show
3 x = [0.1, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0]
4 y = [10**xx for xx in x]
5 output_file("basic_glyphs24.html")
6 # 创建对数轴
7 p = figure(plot_width=400,
8           plot_height=400,
9           y_axis_type="log",
10          y_range=(10**-1, 10**4))
11 p.line(x, y, line_width=2)
12 p.circle(x, y,
13         fill_color="white", size=8)
14 show(p)
```

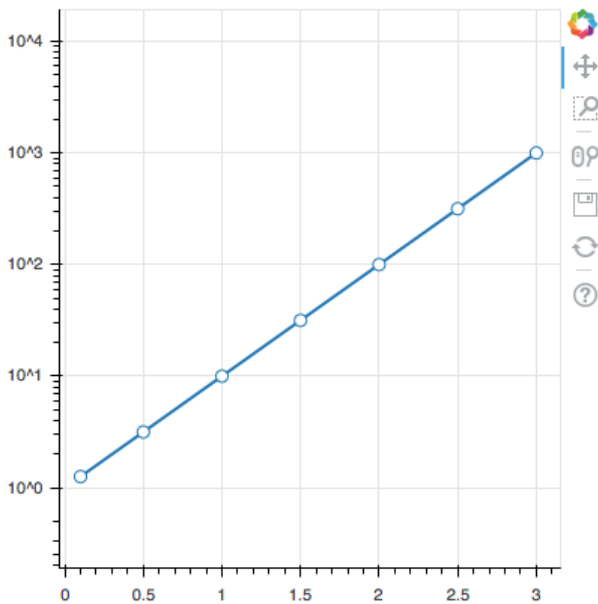


Figure 37: Bokeh 基础作图 24

**Twin Axes** 双轴 有的情况下要将两个范围不同的曲线表示在同一图形中，这时就需要两个坐标轴，也就是双轴。可以通过图形对象的 extra\_x\_range 和 extra\_y\_range 属性来创建额外的坐标轴，再通过 add\_layout 函数布置到图形中，例子如下：

```
1 from numpy import pi, \
2     arange, sin, linspace
```

```
3 from bokeh.plotting import figure, \
4     output_file, show
5 from bokeh.models import LinearAxis, \
6     Range1d
7 x = arange(-2*pi, 2*pi, 0.1)
8 y = sin(x)
9 y2 = linspace(0, 100, len(y))
10 output_file("basic_glyphs25.html")
11 p = figure(x_range=(-6.5, 6.5), \
12          y_range=(-1.1, 1.1))
13 p.circle(x, y, color="red")
14 p.extra_y_ranges = {"foo": \Range1d(
15     start=0, end=100)}
16 p.circle(x, y2, color="blue",
17         y_range_name="foo")
18 p.add_layout(LinearAxis(
19     y_range_name="foo"), 'left')
20 show(p)
```

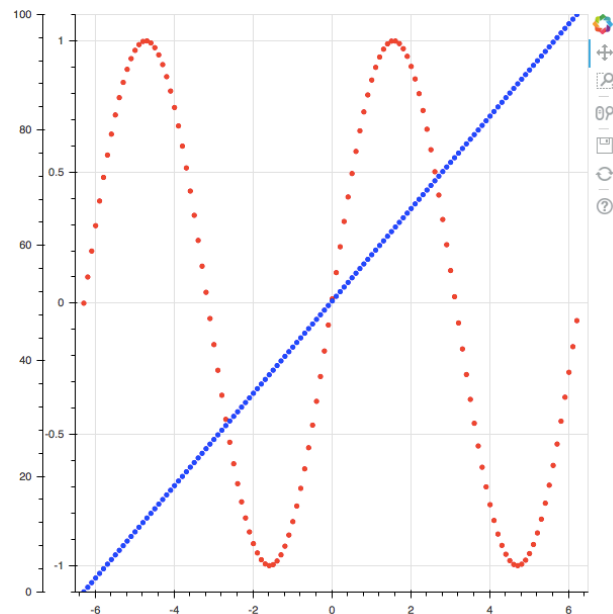


Figure 38: Bokeh 基础作图 25

## 6 嵌入图表和 Apps Embedding Plots and Apps

Bokeh 提供了多种方式来为 HTML 插入图表

### 6.1 生成单独的 HTML 文件

Bokeh 的 file\_html() 函数能生成包含图表的独立 HTML 文件，该文件的模板能用 Bokeh 自带的，也可以用自己的（具体自己查阅 file\_html() 函数的文档，例子 gapminder example plot）。HTML 文件包含了图

表的所有信息并且是可移植的，同时每张图表旁边的交互工具不会消失。例子：

```
1 from bokeh.plotting import figure
2 from bokeh.resources import CDN
3 from bokeh.embed import file_html
4
5 plot = figure()
6 plot.circle([1,2], [3,4])
7
8 html = file_html(plot, CDN, "my plot")
9 # 然后可以在flask或其他框架中引用html
```

这个方法相对比较低级、直接，可以用 `bokeh.plotting` 和 `bokeh.charts` 中的接口配合 `output_file()`、`show()`、`save()` 函数来达到相同的效果。

## 6.2 作为组件嵌入到 HTML 中

Bokeh 提供 `components()` 函数来生成组件形式的图标代码。`components()` 返回：

- 一个 `<script>` 标签代码段（包含所有图表数据）
- 一个 `<div>` 标签代码段（包含引用 `<script>` 的代码）

例子：

```
1 from bokeh.plotting import figure
2 from bokeh.embed import components
3
4 plot = figure()
5 plot.circle([1,2], [3,4])
6
7 script, div = components(plot)
```

其中 `script` 如下所示：

```
<script type="text/javascript">
$(function() {
  var modelid = "fba97329-a355-499e-9252-0adc64b19d2e";
  var modeltype = "Plot";
  var elementid = "8ed68feb-d258-4953-9dfb-fb1c13326509";
  Bokeh.logger.info("Realizing plot:");
  Bokeh.logger.info(" - modeltype: Plot");
  Bokeh.logger.info(" - modelid: fba97329-a355-499e-9252-0adc64b19d2e");
  Bokeh.logger.info(" - elementid: 8ed68feb-d258-4953-9dfb-fb1c13326509");

  var all_models = [ JSON PLOT MODELS AND DATA ARE HERE ];

  Bokeh.load_models(all_models);
  var model = Bokeh.Collections(modeltype).get(modelid);
  var view = new model.default_view({
    model: model, el: '#8ed68feb-d258-4953-9dfb-fb1c13326509'
  });
  Bokeh.index[modelid] = view
});
</script>
```

Figure 39: Bokeh 网页嵌入 1

`div` 如下所示：

```
<div class="plotdiv" id="8ed68feb-d258-4953-9dfb-fb1c13326509"></div>
```

Figure 40: Bokeh 网页嵌入 2

得到了 `script` 和 `div` 之后还不能够显示图表，还需将 Bokeh 官方的 `.css` 和 `.js` 文件引用过来才行，也可以将官方的文件下载到本地来加载。引用只需要在 `<head>` 标签中加入如下代码：

```
<link
href="http://cdn.pydata.org/bokeh/release/bokeh-x.y.z.min.css"
rel="stylesheet" type="text/css">
<link
href="http://cdn.pydata.org/bokeh/release/bokeh-widgets-x.y.z.min.css"
rel="stylesheet" type="text/css">

<script src="http://cdn.pydata.org/bokeh/release/bokeh-x.y.z.min.js"></script>
<script src="http://cdn.pydata.org/bokeh/release/bokeh-widgets-x.y.z.min.js"></script>
```

Figure 41: Bokeh 网页嵌入 3

其中文件名中有“-widget”的引用文件只有在你需要用到 Bokeh widgets 时才会用到，平时可以不引用来节省页面加载的时间。的 `x.y.z` 指的是版本号，比如现在要引用 0.12.0 版本的 Bokeh，则代码应如下所示：

```
<link
href="http://cdn.pydata.org/bokeh/release/bokeh-0.12.0.min.css"
rel="stylesheet" type="text/css">
<link
href="http://cdn.pydata.org/bokeh/release/bokeh-widgets-0.12.0.min.css"
rel="stylesheet" type="text/css">

<script src="http://cdn.pydata.org/bokeh/release/bokeh-0.12.0.min.js"></script>
<script src="http://cdn.pydata.org/bokeh/release/bokeh-widgets-0.12.0.min.js"></script>
```

Figure 42: Bokeh 网页嵌入 4

一个完整的 HTML 文件示例：

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8">
5     <title>Bokeh Scatter Plots</title>
6     <link rel="stylesheet"
7       href="http://cdn.pydata.org/bokeh/
8       release/bokeh-0.12.0.min.css"
9     type="text/css" />
10    <script type="text/javascript"
11      src="http://cdn.pydata.org/bokeh/
12      release/bokeh-0.12.0.min.js">
13    </script>
14
15    <!-- SCRIPT 文本放这COPY -->
16
17  </head>
18  <body>
19    <!-- DIV 文本放这 -->
20
21  </body>
22 </html>
```

`component()` 函数能接受单个 Bokeh 图表模型作为参数，也能接受包含 Bokeh 图表模型的列表 (List)、元组 (tuple)、字典 (dictionary) 作为参数，其返回值是包含 `script` 和 `div` 文本的对应数据结构。

### 6.3 自动加载的嵌入方式

Bokeh 提供了一种自动加载的方式来嵌入图表，自动加载的形式是将一段 `<script>` 标签文本插入到 `<body>` 标签中，运行时 `<script>` 文本会自动替换成 `<div>` 文本。（这种方式需要引用官方的 `.css` 和 `.js` 文件，引用方式上面提到过了）下面有两个不同的例子：

#### 6.3.1 从服务器获取数据

这种方法是将数据用 `autoload_server()` 函数存进 Bokeh 服务器 (Bokeh server) 中：

```
1 from bokeh.client import push_session
2 from bokeh.embed import autoload_server
3 from bokeh.plotting import figure, curdoc
4
5 # figure() 函数自动添加图片到 curdoc() 中
6 plot = figure()
7 plot.circle([1,2], [3,4])
8
9 session = push_session(curdoc())
10 script = autoload_server(plot,
11     session_id=session.id)
```

`script` 文本如下所示：