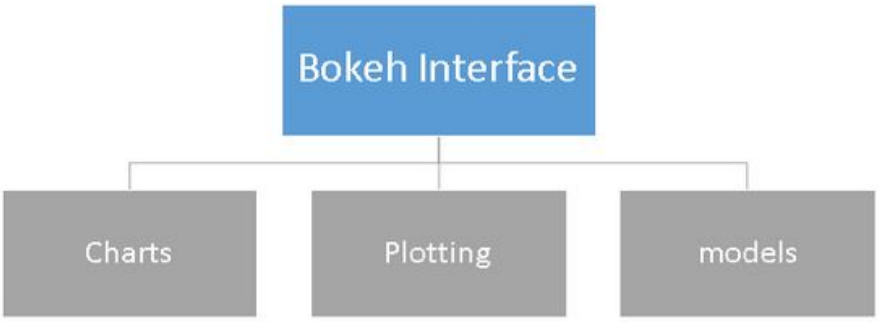
Components

Bokeh is actually composed of two library components.

- The first component is a JavaScript library, `BokehJS`, that runs in the browser. This library is responsible for all of the rendering and user interaction. Its input is a collection of declarative JSON objects that comprise a scenegraph. The objects in this scenegraph describe everything that BokehJS should handle: what plots and widgets are present and in what arrangement, what tools and renderers and axes the plots will have, etc. These JSON objects are converted into Backbone Models in the browser, and are rendered by corresponding Backbone Views.
- The second component is a library in Python (or other languages) that can generate the JSON described above. In the Python Bokeh library, this is accomplished at the lowest level by exposing a set of model classes that exactly mirror the set of Backbone Models that are created in the browser. These python model classes know how to validate their content and attributes, and also how to serialize themselves to JSON.

Interfaces



- Charts: a *high-level* interface that is used to build complex statistical plots as quickly and in a simplistic manner.
- Plotting: an *intermediate-level* interface that is centered around composing visual glyphs.
- Models: a *low-level* interface that provides the maximum flexibility to application developers.

- Bokeh is intended to be useful to data-scientists and domain experts, working at a very high level, as well as to application developers and software engineers, who may want more control or access to more sophisticated features.
- Because of this, Bokeh takes a layered approach and offers programming interfaces appropriate to different levels, as well as some compatibility interfaces to make use of existing code from other libraries.
- This section provides an overview of the different interfaces that are available to Bokeh users, as well as more context about the most important concepts central to the library.

`bokeh.models`

- All of the ***low level*** models live in the low-level `bokeh.models` interface.
- Most of the models are very simple, usually consisting of a few property attributes and no methods.
- Model attributes can either be configured when the model is created, or later by setting attribute values on the model object.

Here are some examples for a `Rect glyph` object:

```
# properties can be configured when a
# model object is initialized
glyph = Rect(x="x", y="y2",
     w=10, h=20, line_color=None)

# or by assigning values to attributes
# on the model later
glyph.fill_alpha = 0.5
glyph.fill_color = "navy"
```

- These methods of configuration work in general for all Bokeh models. Because of that, and because all Bokeh interfaces ultimately produce collections of Bokeh models, styling and configuring plots and widgets is accomplished in basically the same way, regardless of which interface is used.

- Using the `bokeh.models` interface provides complete control over how Bokeh plots and Bokeh widgets are put together and configured.
  (hence: ***Low Level***)

- However, this provides no help with assembling the models in meaningful or correct ways. It is entirely up to developers to build the visualization piece by piece.

- Most users will probably want to use one of the higher level interfaces described, unless they have specialized requirements that necessitate finer control.

`bokeh.plotting`

- Bokeh provides a ***mid-level*** general purpose `bokeh.plotting` interface, which is similar in specificity to Matplotlib or Matlab style plotting interfaces.
- It is centered around having users relate the visual glyphs they would like to have displayed to their data, and otherwise taking care of putting together plots with sensible default axes, grids, and tools.
- All the hard work to assemble the appropriate Bokeh Models to form a visualization that ***BokehJS*** can render is handled automatically.
- The main class in the `bokeh.plotting` interface is the `Figure` class. This is a subclass of the basic `Plot` model, that includes methods for easily adding different kinds of glyphs to a plot.
- Additionally it composes default axes, grids, and tools in the proper way without any extra effort. Typically, users will want to create `Figure` objects by using the `figure()` function.

A prototypical example of the **bokeh.plotting** usage is show below, along with the resulting plot:

```
from bokeh.plotting import figure, output_file, show

# create a Figure object
p = figure(width=300, height=300,
tools="pan,reset,save")

# add a Circle renderer to this figure
p.circle([1, 2.5, 3, 2], [2, 3, 1, 1.5], radius=0.3,
 alpha=0.5)

# specify how to output the plot(s)
output_file("output.html")

# display the figure
show(p)
```

- The main observation is that the typical usage involves creating plots objects with the `figure()` function, then using the glyph methods like `Figure.circle` to add renderers for our data.
- We do not have to worry about configuring any axes or grids (although we can configure them if we need to), and specifying tools is done simply with the names of tools to add.
- Finally we use some output functions to display our plot. The output functions `output_file()` and `show()`, etc. are defined in the `bokeh.io` module, but are also importable from `bokeh.plotting` for convenience.
- There are many other possibilities: saving our plot instead of showing it, styling or removing the axes or grids, adding additional renderers, and laying out multiple plots together.

`bokeh.charts`

- Bokeh also provides a very high-level bokeh.charts interface for quickly creating statistical charts.
- As with **`bokeh.plotting`**, the main purpose of the interface is to help simplify the creation of Bokeh object graphs by encapsulating patterns of assembling Bokeh models.
- The **`bokeh.charts`** interface may also take the additional step of performing necessary statistical or data processing for the user.
- The interface presents functions for common, schematic statistical charts.
- Additionally, the chart functions can take care of automatically coloring and faceting based on group structure.

Charts example

The interface includes chart types such as: `Bar()`, `BoxPlot()`, `Histogram()`, `TimeSeries()`, and many others.

One simple example using `Scatter()` is shown below:

```
from bokeh.charts import Scatter, output_file, show

# prepare some data

from bokeh.sampledata.autompg import autompg as df

# create a scatter chart
p = Scatter(df, x='mpg', y='hp', color='cyl',
            title="MPG vs HP (colored by CYL)",
            legend='top_right',
            xlabel="Miles Per Gallon",
            ylabel="Horsepower")

# specify how to output the plot(s)
output_file("chart.html")

# display the figure
show(p)
```

Important to note is that the same output functions are used across different interfaces. As with `bokeh.plotting`, the output functions `output_file()` and `show()`, etc. that are defined in bokeh.io, are also importable from `bokeh.charts` as a convenience.

Other interfaces

- Bokeh provides some level of Matplotlib compatibility, by using the third-party mplexporter library.
- Although it does not provide 100% coverage of Matplotlib capabilities, it is still quite useful.
- For instance, in addition to many Matplotlib plots, it is often possible to convert plots created using the python Seaborn and `ggplot.py` libraries into Bokeh plots very easily.
- Here is a quick example that shows a Seaborn plot converted to a Bokeh plot with just one additional line of code:

```
import seaborn as sns

from bokeh import mpl
from bokeh.plotting import output_file, show

tips = sns.load_dataset("tips")

sns.set_style("whitegrid")

ax = sns.violinplot(x="day", y="total_bill",
                    hue="sex",
                    data=tips, palette="Set2",
                    split=True,
                    scale="count", inner="stick")

output_file("violin.html")

show(mpl.to_bokeh())
```