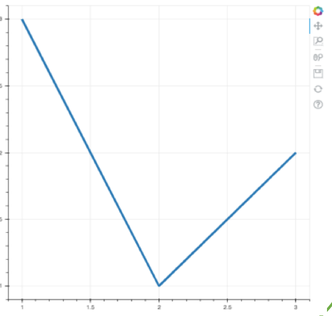
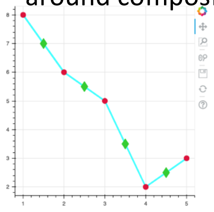
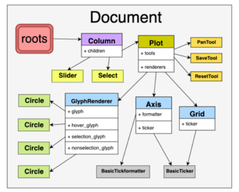
Bokeh is an interactive python library that provides elegant visualizations for large or streaming datasets on web browsers for quick and easy presentation.
Basic code example to plot a line graph using Bokeh and display the output in a local html file –

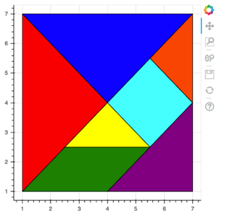
```
from bokeh.plotting import
figure, output_file, show

output_file("test.html")
plot = figure()
plot.line([1, 2, 3, 4],
          [6, 7, 2, 4],
          line_width=2)
show(plot)
```



Bokeh exposes two interface levels to users:
bokeh.models **-** A low-level interface that provides the most flexibility to application developers.
bokeh.plotting - A higher-level interface centered around composing visual glyphs.







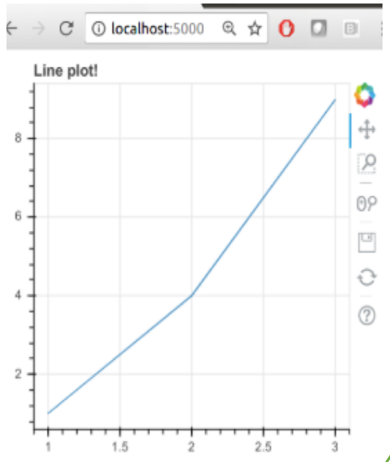
```
plot.line(x, y, line_width=3, line_color='aqua')
plot.circle(x, y, size=10, color='crimson')
plot.diamond([1.5, 2.5, 3.5, 4.5], [7, 5.5, 3.5, 2.5], color='limegreen', size=20)
```

```
# Tangram
t_xs = [ [1, 4, 1], [1, 4, 7], [1, 2.5, 5.5, 4], [2.5, 4, 5.5], [4, 5.5, 7, 5.5], [5.5, 7, 7], [4, 7, 7] ]
t_ys = [ [1, 4, 7], [7, 4, 7], [1, 2.5, 2.5, 1], [2.5, 4, 2.5], [4, 5.5, 4, 2.5], [5.5, 7, 4], [1, 4, 1] ]
t_color = [ 'red', 'blue', 'green', 'yellow', 'cyan', 'orangered', 'purple' ]
plot = figure(plot_width=400, plot_height=400)
plot.patches(t_xs, t_ys, fill_color=t_color, line_color='black')
```

All the rectangular boxes are Bokeh models. These are low-level objects that comprise a Bokeh scene graph. The scene graph is serialized and used to render the plot

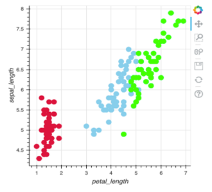
Bokeh is best used to live-stream visualizations that update with real time data. The code below shows how to start a ***Bokeh server*** to plot a line graph.

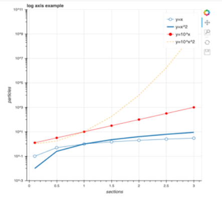
```
from bokeh.server.server import Server
from bokeh.application import Application
from bokeh.application.handlers.function
import FunctionHandler
from bokeh.plotting
import figure, ColumnDataSource

def make_document(doc):
    fig = figure(title='Line plot!',
                 sizing_mode='scale_width')
    fig.line(x=[1, 2, 3], y=[1, 4, 9])

    doc.title = "Hello, world!"
    doc.add_root(fig)

apps = {'/': Application
       (FunctionHandler(make_document))}

server = Server(apps, port=5000)
server.start()
```

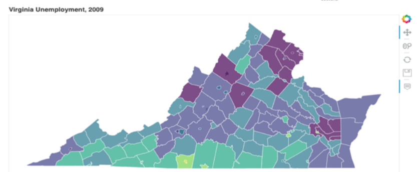


All the glyphs are drawn by a Figure object. Glyphs can be displayed alone and together. Plotting also provide flexibility to customize the glyphs.

```
mapper = CategoricalColorMapper(
            factors=['setosa', 'virginica', 'versicolor'],
            palette=['crimson', 'lime', 'skyblue'])
plot = figure(plot_width=400, plot_height=400,
             x_axis_label='petal_length', y_axis_label='sepal_length')
plot.circle('petal_length', 'sepal_length',
            size=10, source=source,
            color={'field': 'species',
                   'transform': mapper})
```

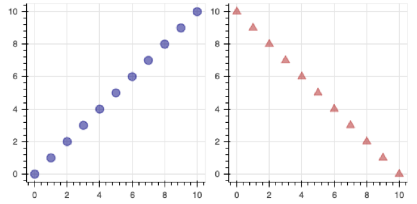






Linking Behaviors: In different figures, enable x_range, y_range to share range objects:

```
s1 = figure(…)
s2 = figure(… x_range=s1.x_range,
y_range=s1.y_range) #share x, y with s1
```



Interactive Legend: clicking or tapping on the legend entries will hide or mute the corresponding glyph

```
p.legend.location = "top_left"
p.legend.click_policy="hide" / "mute"
```

Widget: Provide different common widgets for interaction usage.

```
from bokeh.models.widgets import Button,
CheckboxButtonGroup, ColorPicker,
DataTable, Dropdown, FileInput,
MultiSelect, RangeSlider, Toggle, Div,
Paragraph, PreText
```

Integrate with JavaScript: custom JavaScript event with Bokeh models

```
from bokeh.models.callbacks import CustomJS

callback = CustomJS(args=dict(xr=plot.x_range),
code="""  // JavaScript code goes here  """)
# attach to property change events
js_on_change('start', callback)
# callback receiving from cb_obj variable
p.js_on_event('tap', callback)
```