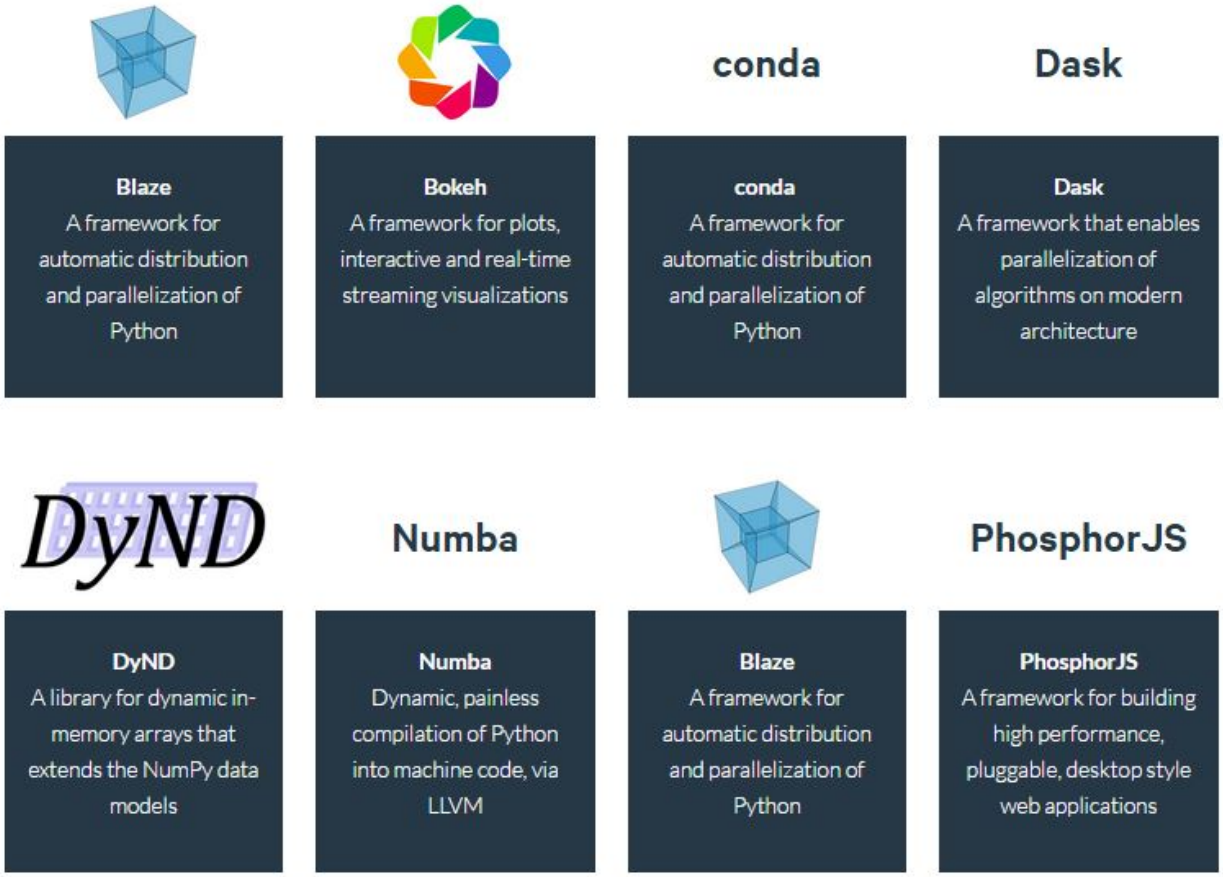
Bokeh in IPython Notebook

- Bokeh is a Python interactive visualization library for large datasets that natively uses the latest web technologies.
- Its goal is to provide elegant, concise construction of novel graphics in the style of Protovis/D3, while delivering high-performance interactivity over large data to thin clients.

- Bokeh is developed by **Continuum Analytics**.



Bokeh scales visualization to Big Data

(From Bokeh Documentation)

- Interactive and real-time streaming visualization framework that scales to Big Data with data shading
- Bokeh is a Python data visualization library combining the ideas of the **Grammar of Graphics** and **Protovis**, with enhancements to support interactive visualization. Its primary output backend is HTML5 Canvas.
- There are many excellent plotting packages for Python, but they generally do not optimize for the particular needs of statistical plotting or

multidimensional datasets.

- Additionally, advanced visual customization is typically difficult for non-programmers, and most libraries do not build a reified data processing pipeline that supports rich interactivity like linked brushing.
- Bokeh addresses these problems at their core by using a declarative data transformation scheme, and is engineered to operate in a client/server model for the modern web.
- Bokeh can produce elegant and interactive visualization like ***D3.js*** with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications.

Benefits of Bokeh:

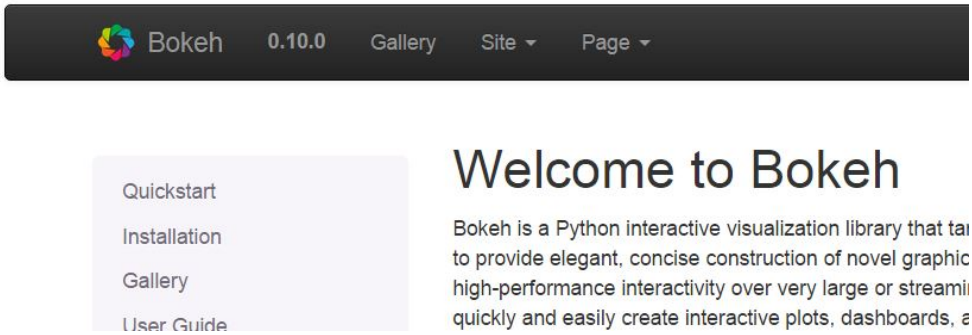
- Bokeh allows you to build complex statistical plots quickly and through simple commands
- Bokeh provides you output in various medium like html, notebook and server
- We can also embed Bokeh visualization to flask and django app
- Bokeh can transform visualization written in other libraries like matplotlib, seaborn, ggplot
- Bokeh has flexibility for applying interaction, layouts and different styling option to visualization

Installation

Usually they are already installed, but sometimes you may need to install a few Python packages :

```
conda install pandas
conda install bokeh
```

IMPORTANT : Version



- This workshop will be based on version 0.10.
- A lot of code for older version of Bokeh no longer work.
- To check what version you actually have installed, run the following code
- Bokeh is under active development. Expect the next version soon.

```
In [1]: import bokeh
        print(bokeh.__version__)
```

```
0.10.0
```

Challenges with Bokeh

- Like with any upcoming open source library, Bokeh is undergoing a lot of development. So, the code you write today may not be entirely reusable in future.
- It has relatively less visualization options, when compared to ***D3.js***. Hence, it is unlikely in near future that it will challenge ***D3.js*** for its crown.
- Given the benefits and the challenges, it is currently ideal to rapidly develop prototypes. However, if you want to create something for production environment, ***D3.js*** might still be your best bet.

Bokeh Graphics with Jupyter Notebooks

Displaying Inline Plots
To display Bokeh plots inline in an IPython/Jupyter notebook, use the `output_notebook()` function from `bokeh.io` instead of (or in addition to) the `output_file()`

- Open up a Jupyter Notebook, and enter the following code in over three or four cells.
- Run those cells. The output should come up on a new HTML page. (See next page).
- The output graphic has some controls (pan , zoom, resize etc). Try these controls out.
- Vary some of the values (width, either , title).

```
#Import library
from bokeh.charts import Bar, output_file, show


#use output_file to visualize it in html file
#use output_notebook to visualize it in notebook

# create a simple data set
myData = {"y": [1, 2, 3, 4, 5]}

# Output to Line.HTML
output_file("lines.html", title="line plot example")

# create a new line chat with a title and axis labels
p = Bar(myData, title="Simple Bar Chart Example",
label='x', ylabel='values',
width=400, height=400)

# show the results
show(p)
```

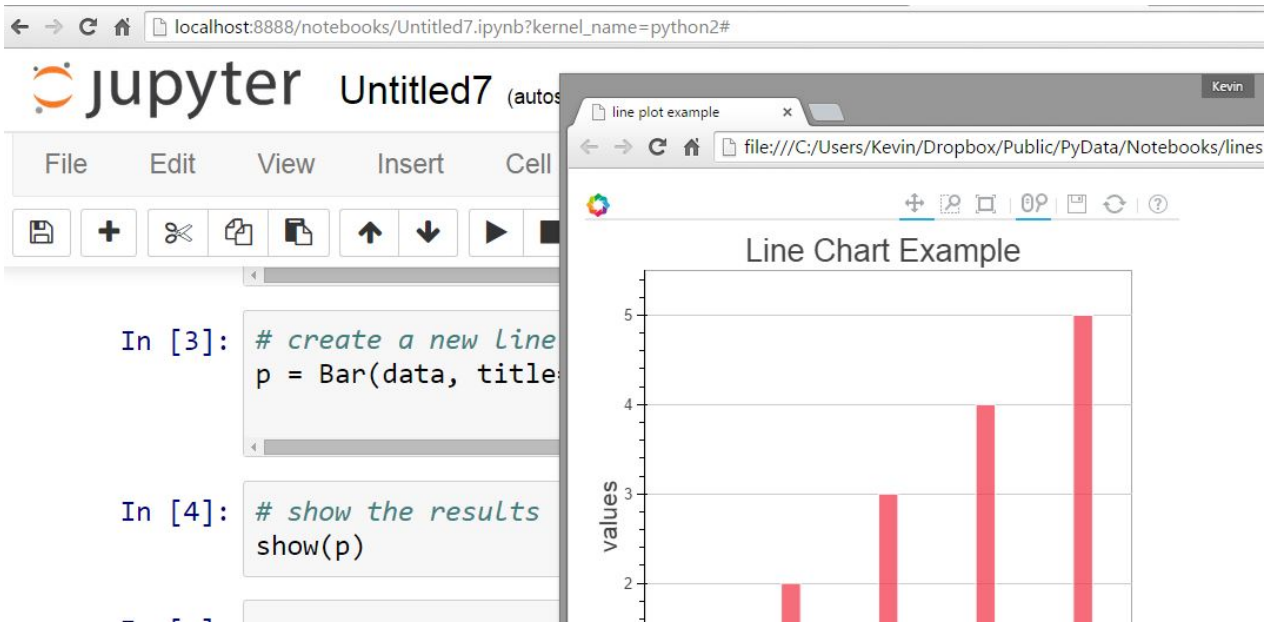


Figure 1: Output

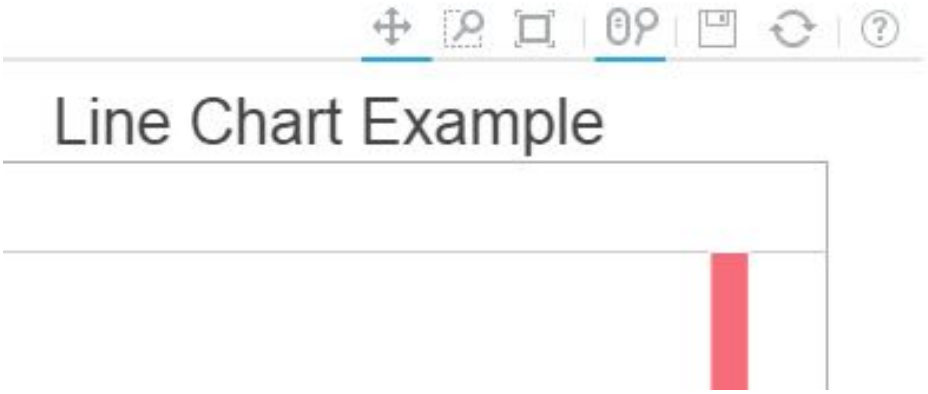
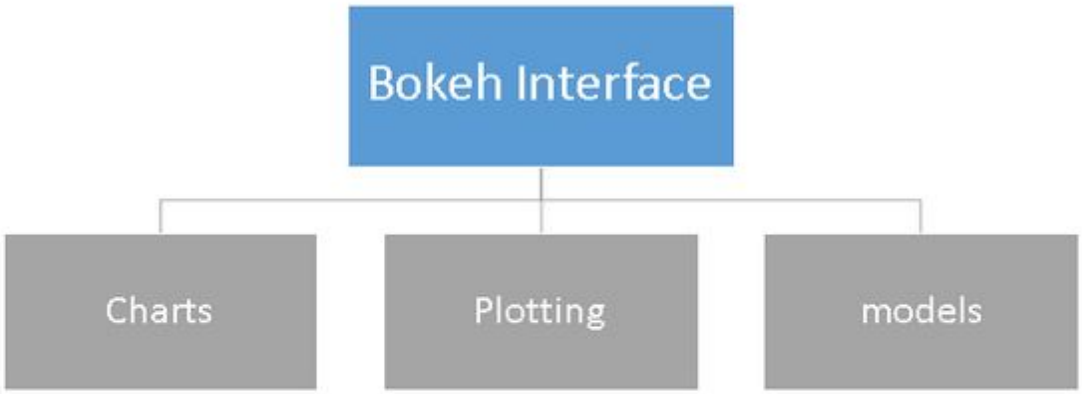


Figure 2: Controls

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.

Visualization with Bokeh

- Bokeh offers both powerful and flexible features which imparts simplicity and highly advanced customization.
- It provides multiple visualization interfaces to the user as shown below:`Bokeh_Interface`

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.

Charts - Example 1

- As mentioned above, it is a high level interface used to present information in standard visualization form.
- These forms include box plot, bar chart, area plot, heat map, donut chart and many others.
- You can generate these plots just by passing data frames, numpy arrays and dictionaries.

Lets look at the common methodology to create a chart:

- Import the library and functions/ methods
- Prepare the data
- Set the output mode (Notebook, Web Browser or Server)
- Create chart with styling option (if required)
- Visualize the chart

```
#Import library

from bokeh.charts import Bar, output_file, show
#use output_notebook to visualize it in notebook

# prepare data (dummy data)
data = {"y": [1, 2, 3, 4, 5]}

# Output to Line.HTML
output_file("lines.html", title="line plot example")

#put output_notebook() for notebook

# create a new line chat with a title and axis labels
p = Bar(data, title="Line Chart Example", xlabel='x',
ylabel='values', width=400, height=400)

# show the results
show(p)
```

- **Important:** In the chart above, you can see the tools at the top (zoom, resize, reset, wheel zoom) and these tools allows you to interact with chart.
- You can also look at the multiple chart options (legend, xlabel, ylabel, xgrid, width, height and many other) and various example of charts here.

Charts Example 2

- Compare the distribution of sepal length and petal length of IRIS data set using Box plot on notebook
- To create this visualization, we can import the iris data set using ***sklearn library***.
- Then, follow the steps as discussed above to visualize chart in ipython notebook.

```
#IRIS Data Set

from sklearn.datasets import load_iris
import pandas as pd
iris = load_iris()

df=pd.DataFrame(iris.data)
df.columns=['petal_width','petal_length',
'sepal_width','sepal_length']

#Import library
from bokeh.charts import BoxPlot,
output_notebook, show
data=df[['petal_length','sepal_length']]

# Output to Notebook
output_notebook()

# create a new line chat with a title and axis labels
p = BoxPlot(data, width=400, height=400)

# show the results
show(p)
Bokeh_Box_Plot
```

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.

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.

Box Plots

The `BoxPlot` can be used to summarize the statistical properties of different groups of data. The label specifies a column in the data to group by, and a box plot is generated for each group:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl',
title="MPG Summary (grouped by CYL)")

output_file("boxplot.html")

show(p)
```

The label can also accept a list of column names, in which case the data is grouped by all the groups in the list:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label=['cyl', 'origin'],
title="MPG Summary (grouped by CYL, ORIGIN)")

output_file("boxplot.html")

show(p)
```

Box Color

The colour of the box in a `BoxPlot` can be set to a fixed colour using the `color` parameter:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl', color='#00cccc',
title="MPG Summary (grouped by CYL)")

output_file("boxplot.html")

show(p)
```

As with `BarCharts`, the color can also be given a column name, in which case the boxes are shaded automatically according to the group:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl', color='cyl',
title="MPG Summary (grouped and shaded by CYL)")

output_file("boxplot.html")

show(p)
```

Whisker Color

The colour of the whiskers can be similary controlled using the `whisker_color` parameter. For a single colour:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl', whisker_color='goldenrod',
title="MPG Summary (grouped by CYL, shaded whiskers)")

output_file("boxplot.html")

show(p)
```

Or shaded automatically according to a column grouping:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl', whisker_color='cyl',
title="MPG Summary (grouped and whiskers shaded by CYL)")

output_file("boxplot.html")

show(p)
```

Outliers

- By default, BoxPlot charts show outliers above and below the whiskers.
- However, the display of outliers can be turned on or off with the outliers parameter:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl', outliers=False,
title="MPG Summary (grouped by CYL, no outliers)")

output_file("boxplot.html")

show(p)
```

Markers

The marker used for displaying outliers is controlled by the marker parameter:

```
from bokeh.charts import BoxPlot, output_file, show
from bokeh.sampledata.autompg import autompg as df

p = BoxPlot(df, values='mpg', label='cyl', marker='square',
title="MPG Summary (grouped by CYL, square marker)")

output_file("boxplot.html")

show(p)
```

Scatterplot for iris data set

```
In [1]: from bokeh.sampledata.iris import flowers
```

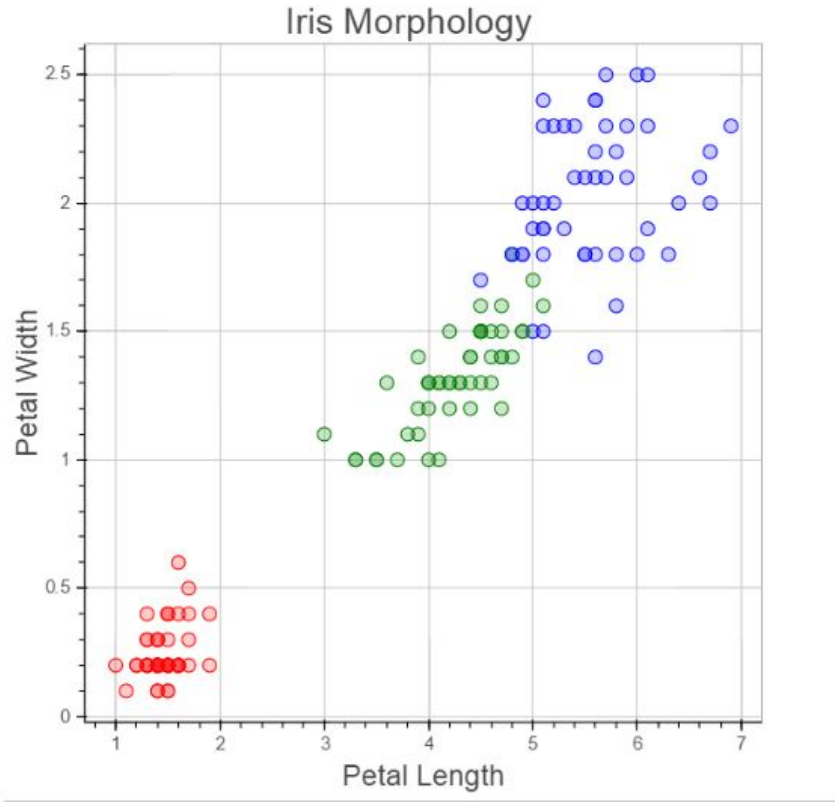
```
In [2]: flowers
```

Out[2]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa

```
from bokeh.sampledata.iris import flowers
from bokeh.plotting import figure, show, output_file

colormap = {'setosa': 'red',
'versicolor': 'green',
'virginica': 'blue'}
flowers['color'] =
flowers['species'].map(lambda x: colormap[x])

output_file("iris.html",
title="iris.py example")

p = figure(title = "Iris Morphology")
p.xaxis.axis_label = 'Petal Length'
```

```
p.yaxis.axis_label = 'Petal Width'

p.circle(flowers["petal_length"],
     flowers["petal_width"],
    color=flowers["color"],
    fill_alpha=0.2, size=10, )

show(p)
```



Exercises

- Try this plot out for different combinations of predictors variables.
- Try out some different colour combinations for the `colormap`.
- Try this out with a different data set : autompg. (*See the graphic below.*) You can use `cyl` and `origin` as grouping variables.
  - The levels of cyl are 4,6 and 8.
  - The levels of origin are 1,2 and 3.

```
In [6]: from bokeh.sampledata.autompg import autompg

In [7]: autompg
```

Out[7]:

	mpg	cyl	displ	hp	weight	accel	yr	origin	name
0	18	8	307	130	3504	12.0	70	1	chevrolet chevelle malibu
1	15	8	350	165	3693	11.5	70	1	buick skylark 320
2	18	8	318	150	3436	11.0	70	1	plymouth satellite
3	16	8	304	150	3433	12.0	70	1	amc rebel sst
4	17	8	302	140	3449	10.5	70	1	ford torino
5	15	8	429	198	4341	10.0	70	1	ford galaxie 500

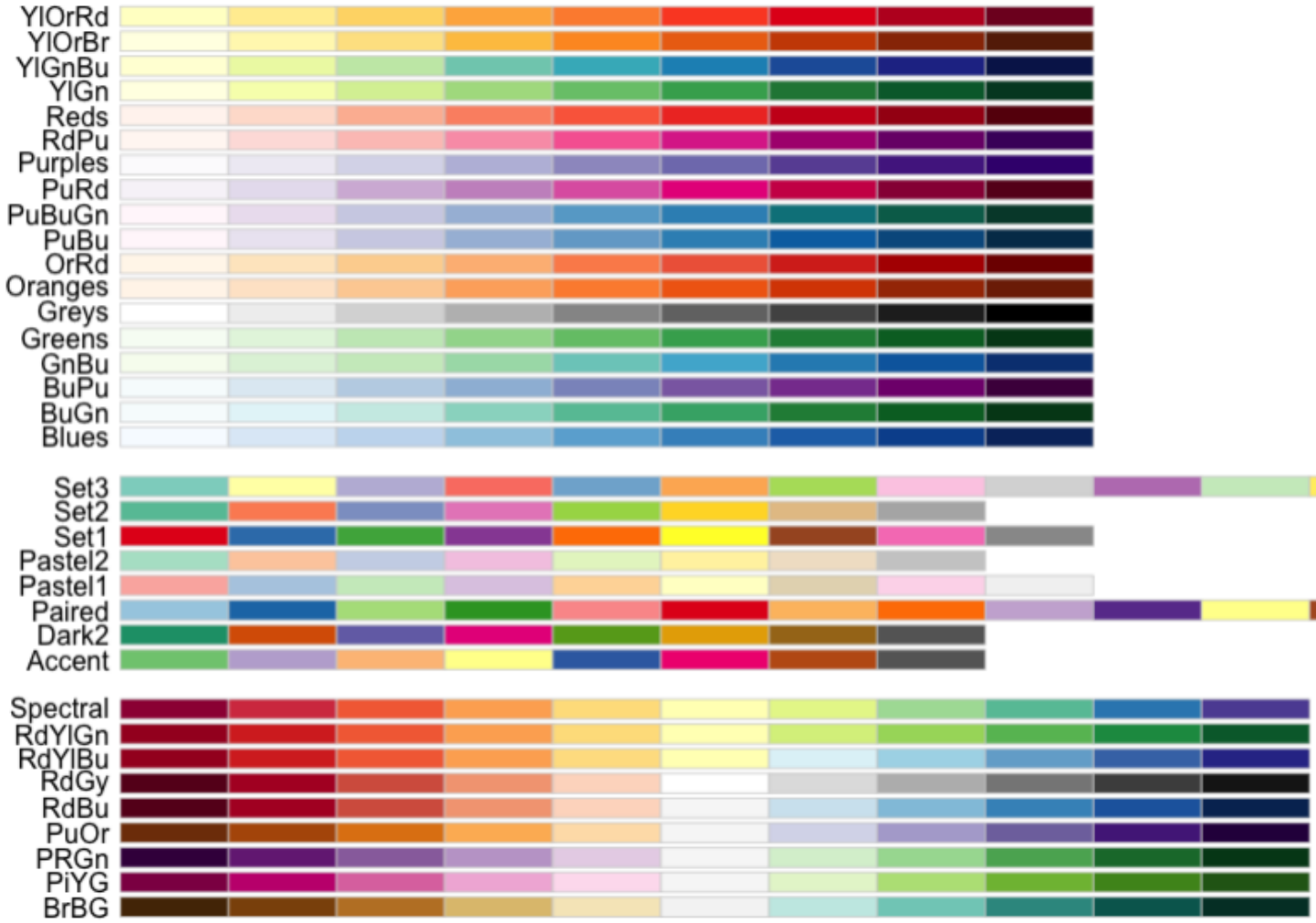
Other interfaces

- 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())
```

Using Palettes

- Palettes are sequences (lists or tuples) of RGB(A) hex strings that define a colormap and be can set as the palette attribute of all chart types from `bokeh.charts` and as the color attribute of many plot objects from `bokeh.plotting`.
- Bokeh offers many of the standard Brewer palettes, which can be imported from the `bokeh.palettes` module.
- For example, importing `Spectral6` gives a six element list of RBG(A) hex strings from the Brewer Spectral colormap.




```
>>> from bokeh.palettes import Spectral6
>>> Spectral6
[ '#3288bd', '#99d594', '#e6f598',
'#fee08b', '#fc8d59', '#d53e4f']
```

- All of the standard palettes included in bokeh can be found at Standard Palettes.

- Custom palettes can be made by creating sequences of RGB(A) hex strings.

Colourblind Friendly Palettes

The default Palettes, that are used by high level charts, contain red and green are inappropriate for any colourblind users.

```
from bokeh.palettes import brewer
palette = brewer["Blues"][3]
```

pass palette into chart call e.g.

```
bar = Bar(medals, countries, title="grouped, dict_input",
xlabel="countries",
ylabel="medals",
legend=True, width=800, height=600,
notebook=True, palette=palette)
```

Palette can actually be a list of any values you want, but there's inbuilt palettes listed in the Bokeh documentation.

Glossary

In order to make the best use of this User Guide, it is important to have context for some high level concepts and terms. Here is a small glossary of some of the most important concepts in Bokeh.

BokehJS

The JavaScript client library that actually renders the visuals and handles the UI interactions for Bokeh plots and widgets in the browser. Typically, users will not have to think about this aspect of Bokeh much (We write the JavaScript, so you dont have to!) but it is good to have basic knowledge of this dichotomy. For full details, see the BokehJS chapter of the Developer Guide.

Charts

Schematic statistical plots such as bar charts, horizon plots, time series, etc. that may include faceting, grouping, or stacking based on the structure of the data. Bokeh provides a high level bokeh.charts interface to quickly construct these kinds of plots. See Using High-level Charts for examples and usage.

Embedding

Various methods of including Bokeh plots and widgets into web apps and pages, or the IPython notebook.

Glyphs

The basic visual building blocks of Bokeh plots, e.g. lines, rectangles, squares, wedges, patches, etc. The `bokeh.plotting` interface provides a convenient

way to create plots centered around glyphs.

Models

The lowest-level objects that comprise Bokeh scenegraphs. These live in the `bokeh.models` interface. Most users will not use this level of interface to assemble plots directly.

However, ultimately all Bokeh plots consist of collections of models, so it is important to understand them enough to configure their attributes and properties.

Server

The bokeh-server is an optional component that can be used for sharing and publishing Bokeh plots and apps, for handling streaming of large data sets, or for enabling sophisticated user interactions based off of widgets and selections.

Widgets

User interface elements outside of a Bokeh plot such as sliders, drop down menus, buttons, etc. Events and updates from widgets can inform additional computations, or cause Bokeh plots to update.