#21

Using Bokeh for Visualization

MIS561 Data Visualization
Original Author: Lusi Yang

A little history behind the Python libraries

Rise of Big Data and Web 2.0:

- Interactive websites that uses JavaScript
- Data Visualization tools such as Tableau and PowerBI

On the Python side:

- Matplotlib - developed by John Hunter (based on existing tools such as ggplot2 and Matlab)
- Seaborn – developed by Michael Waskom (improves usability of the Matplotlib)

Matplotlib and Seaborn library can't create interactive visuals.

Introduction to Bokeh

Bokeh has been around since 2013, with version 3.0.0 being released in 2022 (Top contributor: Sarah Bird).

It targets modern web browsers to present interactive visualizations to users rather than static images.

Some of the features of Bokeh:

- Simple visualizations
- Excellent animated visualizations
- Inter-visualization interactivity
- Supports multiple languages
- Multiple ways to perform a task
- Beautiful chart styling

Interfaces in Bokeh

The interface-based approach provides different levels of complexity for users that either simply want to create some basic plots with very few customizable parameters or the users who want full control over their visualizations and want to customize every single element of their plots.

This layered approach is divided into two levels:

Plotting — This layer is customizable.

Models interface — This layer is complex and provides an open approach to designing charts.

Output

Three ways of output:

- The **.show()** method:

Display the plot in an HTML page

- Inline **.show()** method:

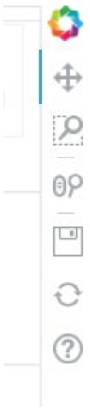
Display the chart inside your notebook (when using inline plotting).

- The **.output_file()** method

Directly save the visualization to a file without any overhead using the **.output_file()** method.

Presentation – interactions in Bokeh

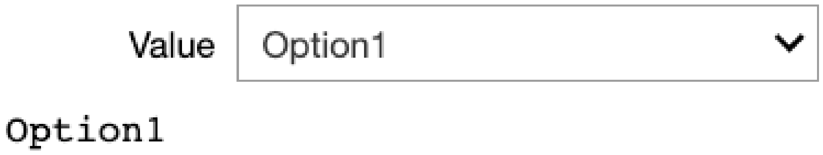
Passive interactions



- Passive interactions are actions that the users can take that neither change the data nor the displayed data.
- In Bokeh, this is called the **Inspector**. The inspector contains attributes such as zooming, panning, and hovering over data.

Active interactions

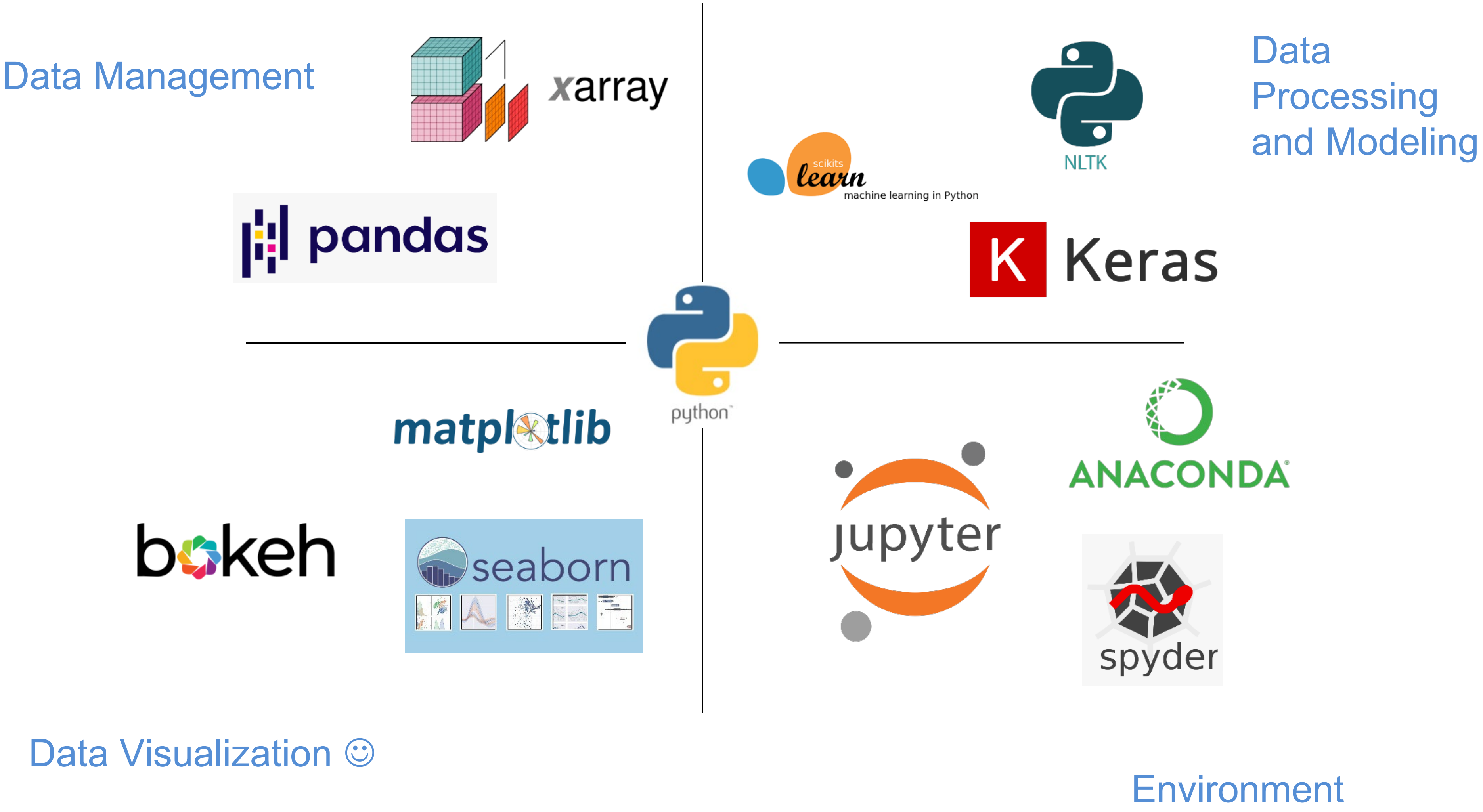
- Active interactions are actions that directly change the displayed data. This incorporates actions such as selecting subsets of data or filtering the dataset based on parameters.
- **Widgets** are the most prominent of active interactions.



Value Option1

```
Option1
```

Motivation for using Python to Analyze Data



Data Management

Data Processing and Modeling

Data Visualization ☺

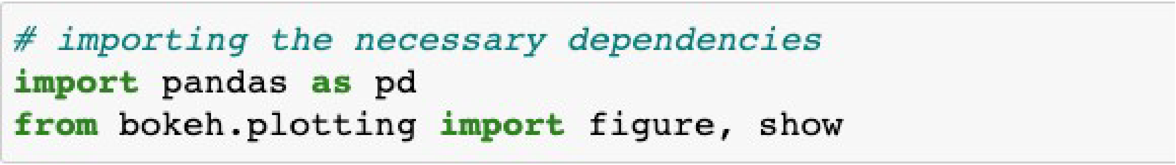
Environment

Activity I

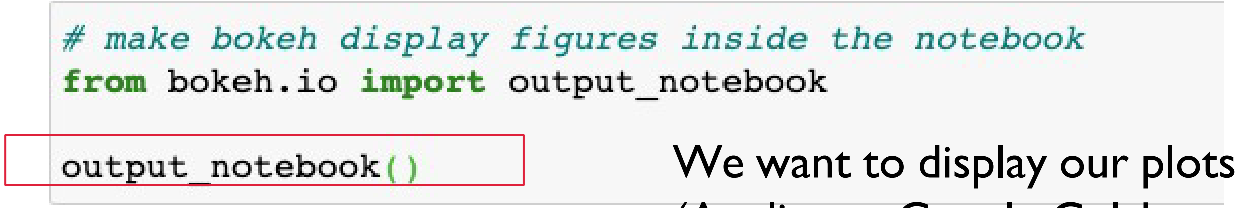
Plotting with Bokeh

Objective

In this activity, we will use higher-level interface in Bokeh to get some insights into the population density development of Germany and Switzerland.

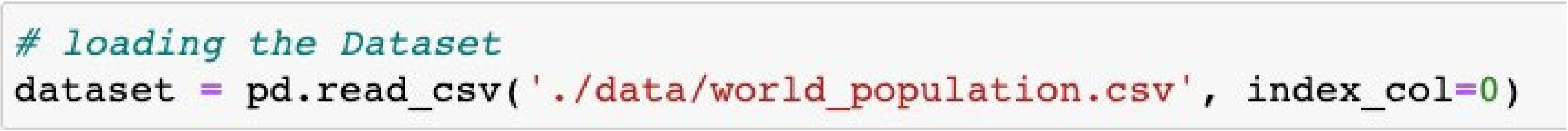


```
# importing the necessary dependencies
import pandas as pd
from bokeh.plotting import figure, show
```

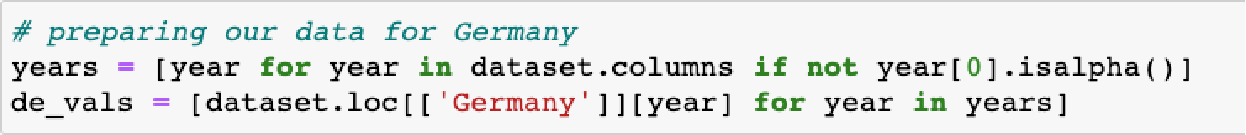


```
# make bokeh display figures inside the notebook
from bokeh.io import output_notebook

output_notebook()
```

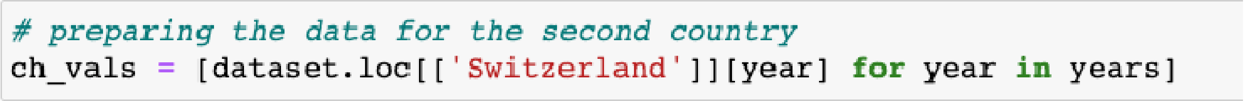
We want to display our plots inside a Jupyter Notebook (Applies to Google Colab as well)



```
# loading the Dataset
dataset = pd.read_csv('./data/world_population.csv', index_col=0)
```

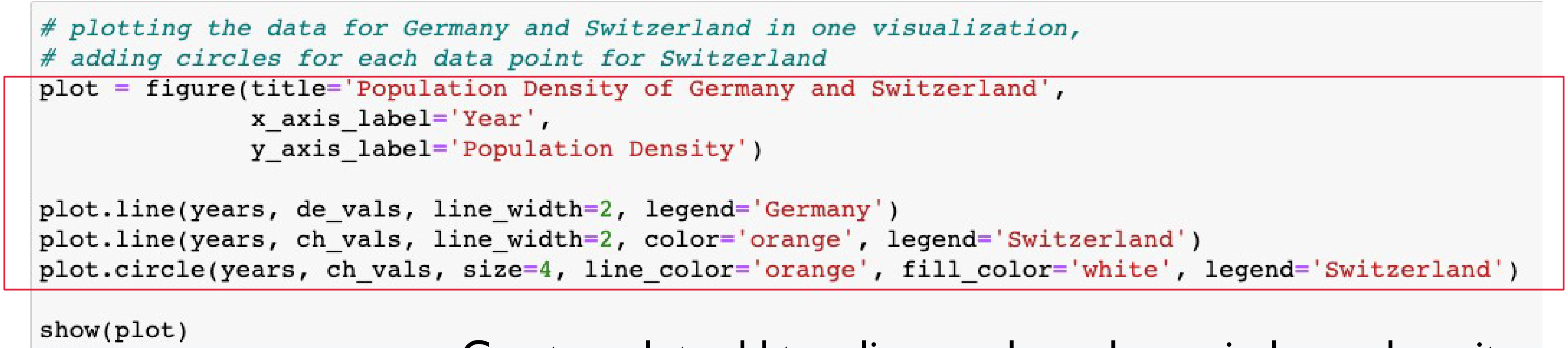


```
# preparing our data for Germany
years = [year for year in dataset.columns if not year[0].isalpha()]
de_vals = [dataset.loc[['Germany']][year] for year in years]
```



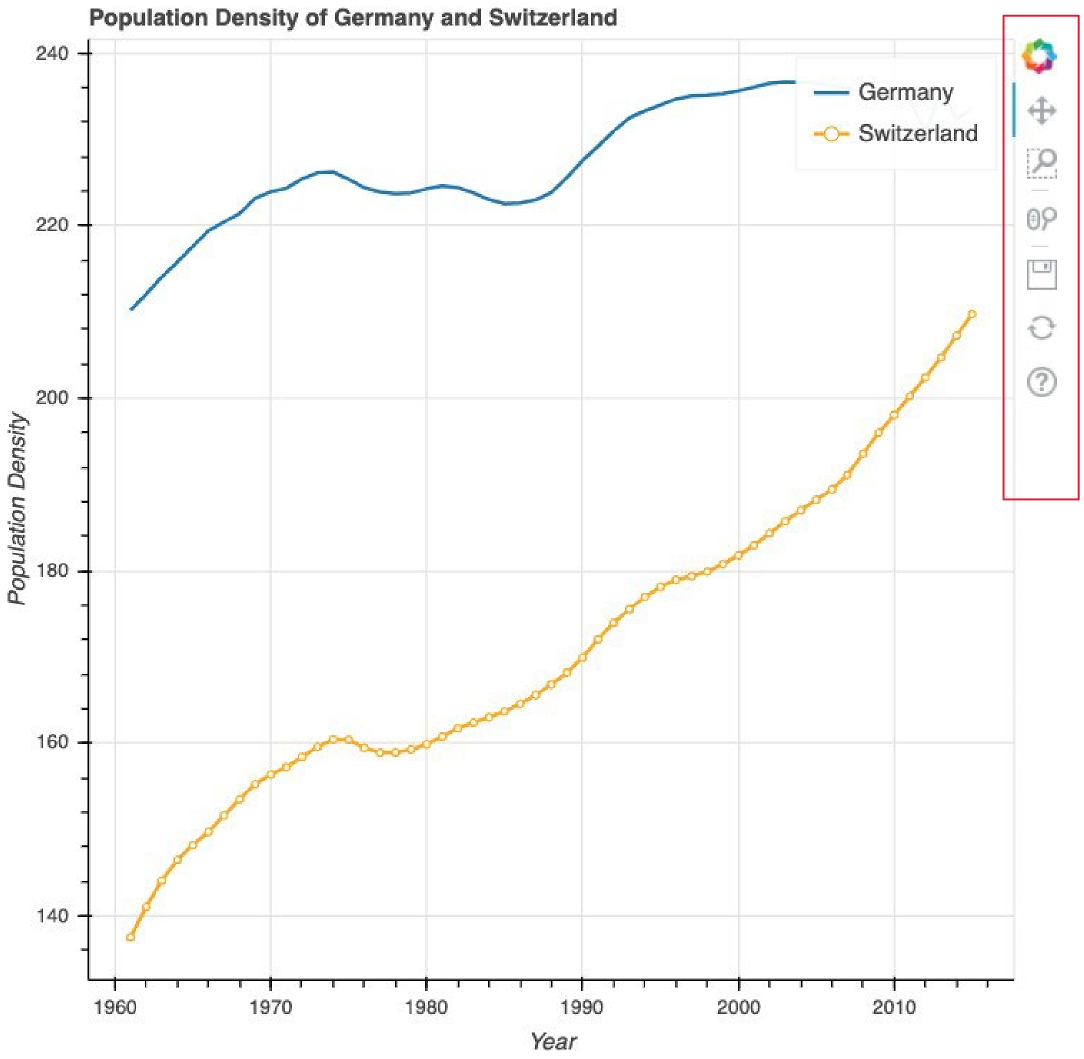
```
# preparing the data for the second country
ch_vals = [dataset.loc[['Switzerland']][year] for year in years]
```

Extract data population data for Germany and Switzerland



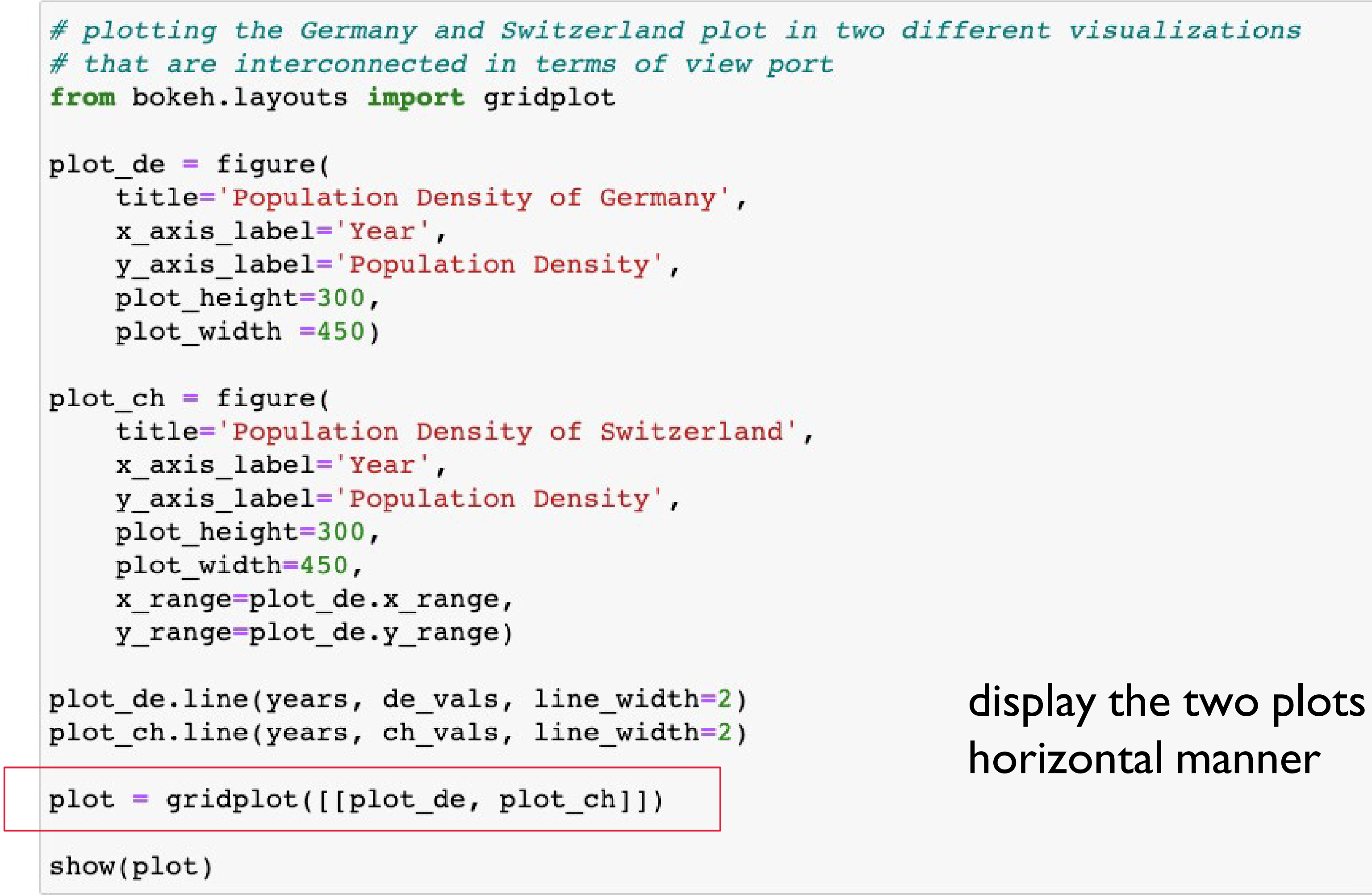
```
# plotting the data for Germany and Switzerland in one visualization,
# adding circles for each data point for Switzerland
plot = figure(title='Population Density of Germany and Switzerland',
              x_axis_label='Year',
              y_axis_label='Population Density')

plot.line(years, de_vals, line_width=2, legend='Germany')
plot.line(years, ch_vals, line_width=2, color='orange', legend='Switzerland')
plot.circle(years, ch_vals, size=4, line_color='orange', fill_color='white', legend='Switzerland')

show(plot)
```

Create a plot, add two line graphs and one circle graph on it



Inspector

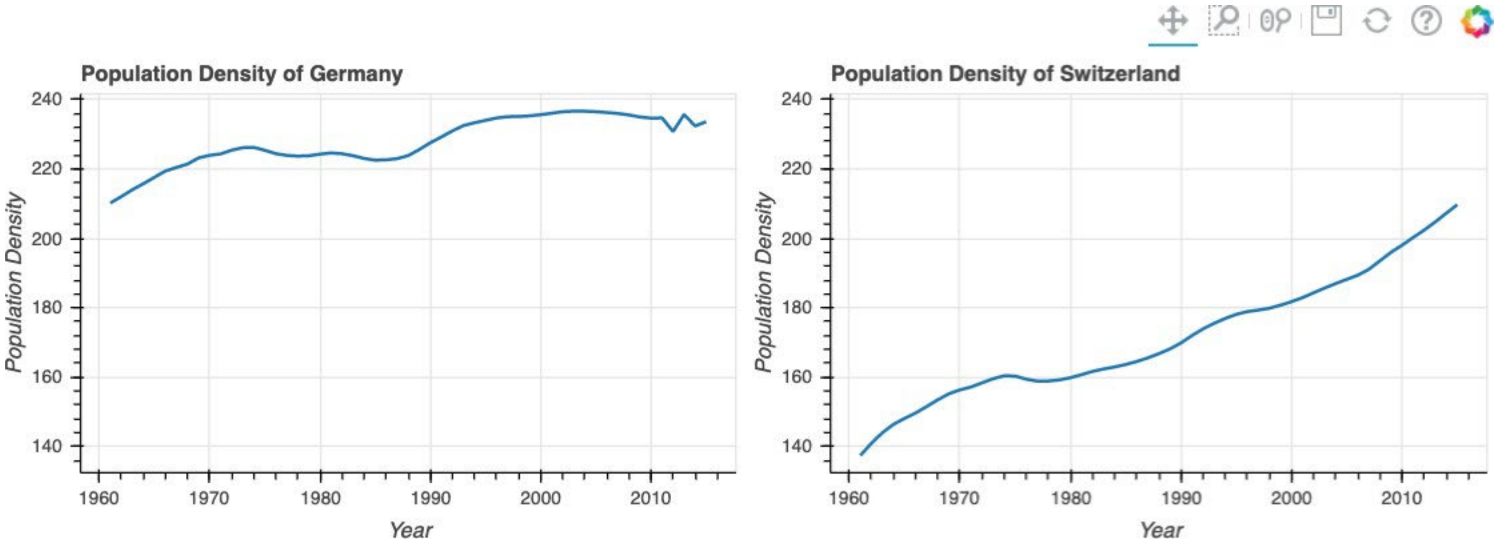
- pan
- box zoom
- wheel zoom
- save
- reset



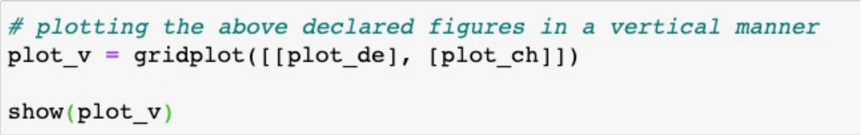
```
# plotting the Germany and Switzerland plot in two different visualizations
# that are interconnected in terms of view port
from bokeh.layouts import gridplot

plot_de = figure(
    title='Population Density of Germany',
    x_axis_label='Year',
    y_axis_label='Population Density',
    plot_height=300,
    plot_width =450)

plot_ch = figure(
    title='Population Density of Switzerland',
    x_axis_label='Year',
    y_axis_label='Population Density',
    plot_height=300,
    plot_width=450,
    x_range=plot_de.x_range,
    y_range=plot_de.y_range)

plot_de.line(years, de_vals, line_width=2)
plot_ch.line(years, ch_vals, line_width=2)

plot = gridplot([[plot_de, plot_ch]])

show(plot)
```

display the two plots in a horizontal manner

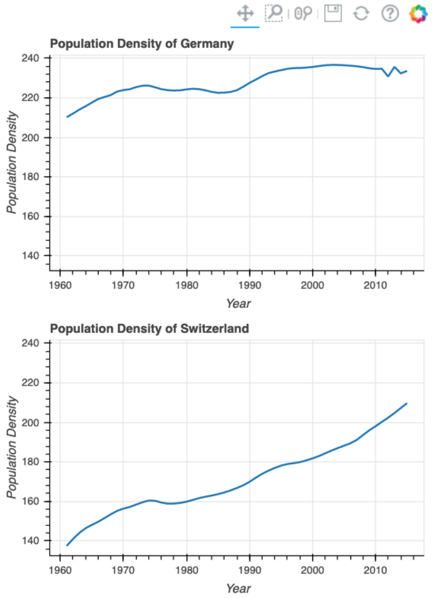
display the two plots in a horizontal manner



display the two plots in a vertical manner



```
# plotting the above declared figures in a vertical manner
plot_v = gridplot([[plot_de], [plot_ch]])

show(plot_v)
```



Interfaces in Bokeh

bokeh.plotting

- The composition of sub-elements such as axes, grids, and the **Inspector** (they provide basic ways of exploring your data through zooming, panning, and hovering) is done without additional configuration.

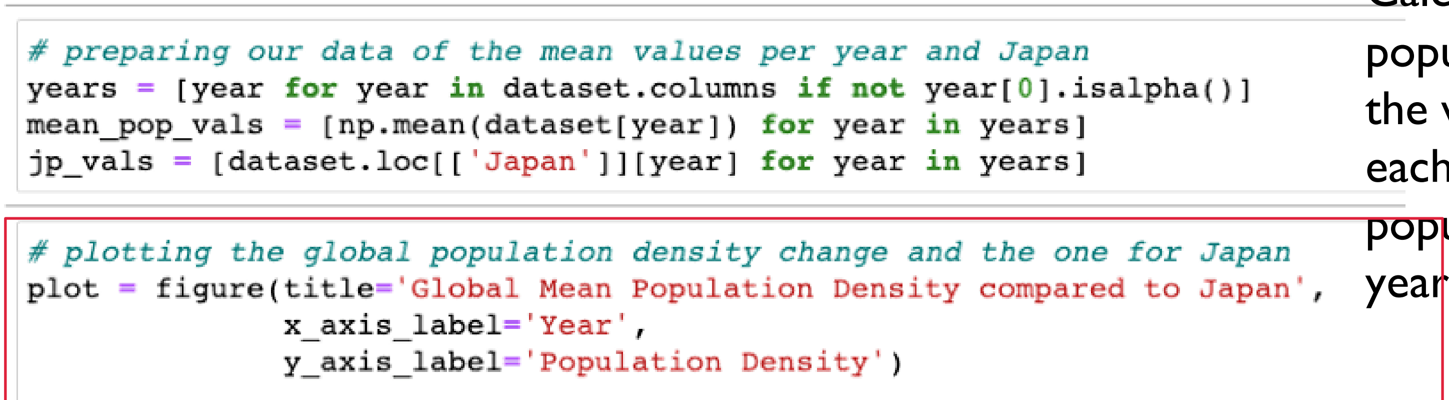
bokeh.models

- This low-level interface is composed of two libraries: the JavaScript library called **BokehJS**, which gets used for displaying the charts in the browser and the Python library, which provides the developer interface.
- The models interface exposes complete control over how Bokeh plots and **widgets** are assembled and configured.

Activity 2

Comparing the Plotting and Models Interfaces

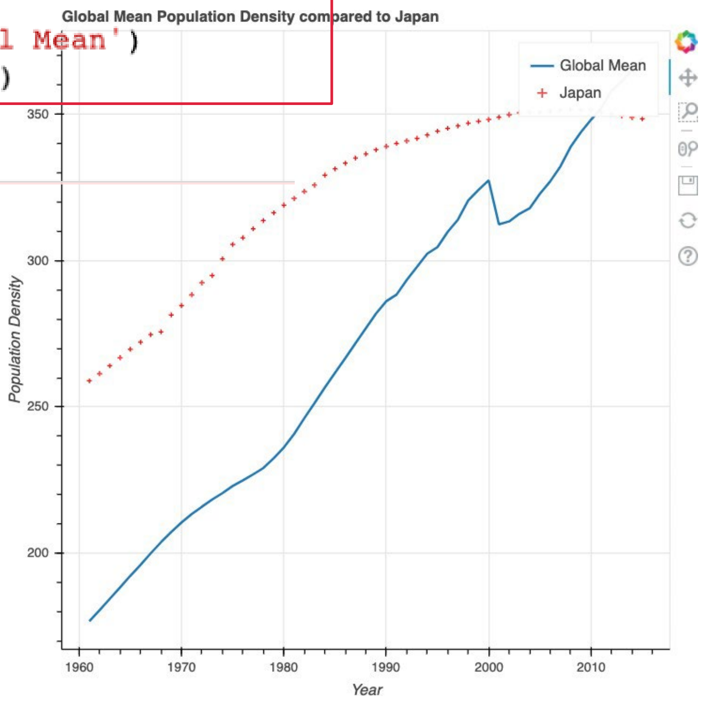
a) using the plotting interface



```
# preparing our data of the mean values per year and Japan
years = [year for year in dataset.columns if not year[0].isalpha()]
mean_pop_vals = [np.mean(dataset[year]) for year in years]
jp_vals = [dataset.loc[['Japan']][year] for year in years]
```

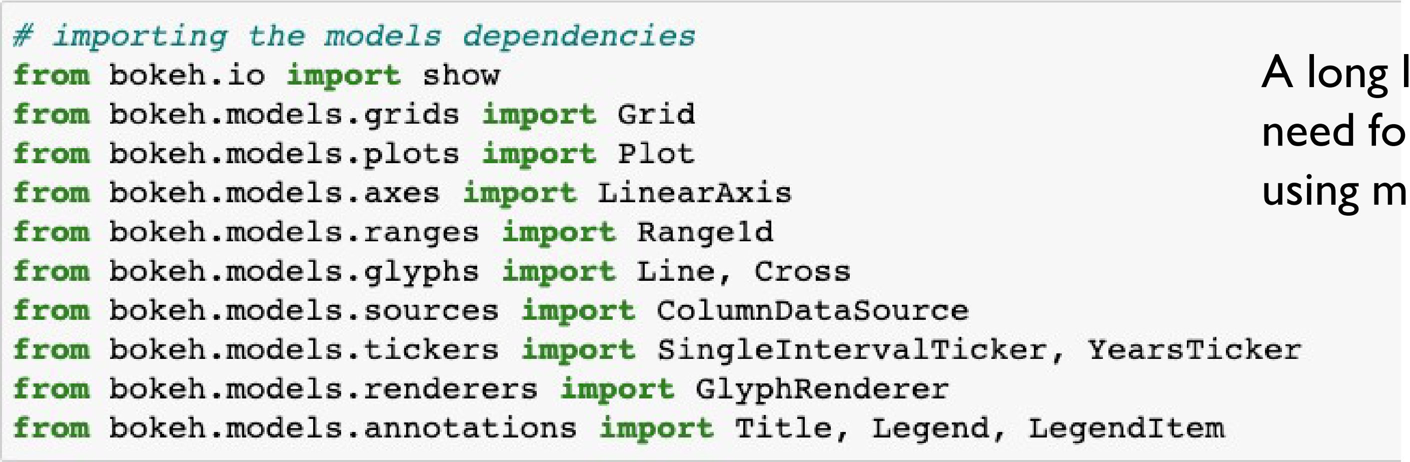
```
# plotting the global population density change and the one for Japan
plot = figure(title='Global Mean Population Density compared to Japan',
              x_axis_label='Year',
              y_axis_label='Population Density')

plot.line(years, mean_pop_vals, line_width=2, legend='Global Mean')
plot.cross(years, jp_vals, legend='Japan', line_color='red')

show(plot)
```

Calculate the mean population density for the whole dataset for each year, and the mean population density per year for **Japan**



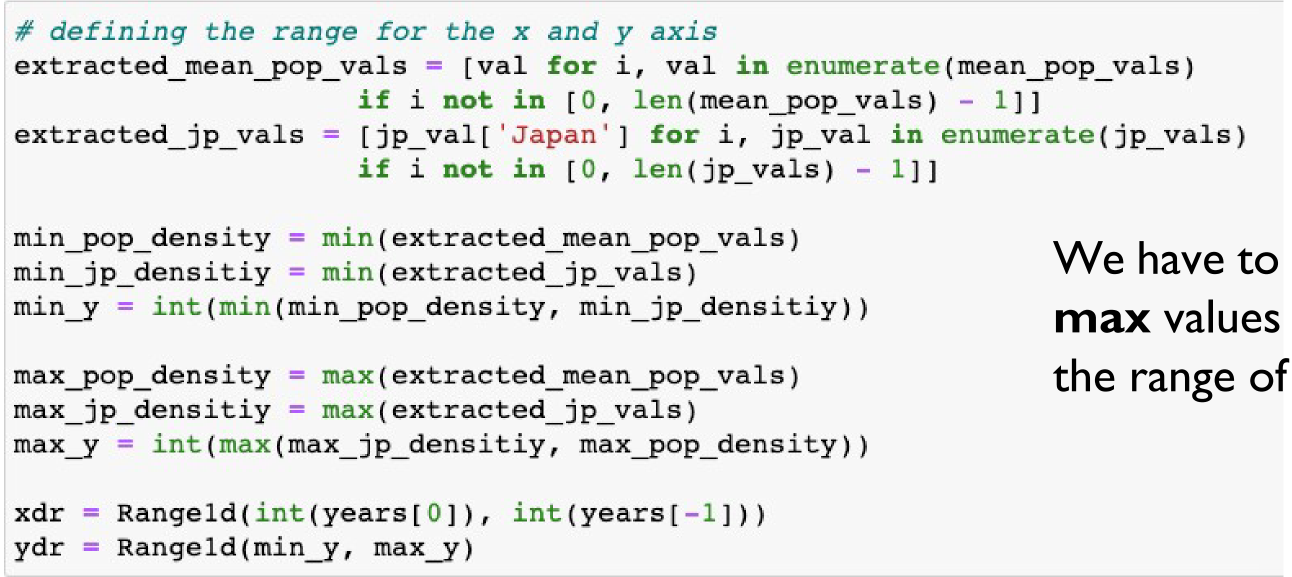
Using the **plotting** interface, we can create a plot element, and plot the global mean with a line and the mean of **Japan** with crosses

b) using the models interface



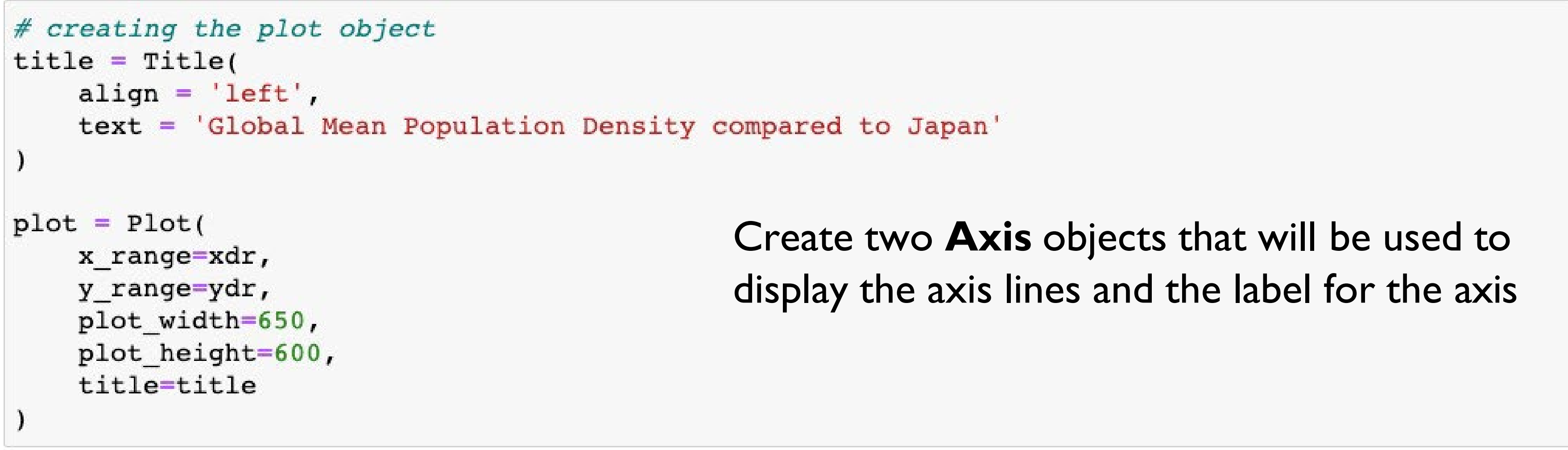
```
# importing the models dependencies
from bokeh.io import show
from bokeh.models.grids import Grid
from bokeh.models.plots import Plot
from bokeh.models.axes import LinearAxis
from bokeh.models.ranges import Range1d
from bokeh.models.glyphs import Line, Cross
from bokeh.models.sources import ColumnDataSource
from bokeh.models.tickers import SingleIntervalTicker, YearsTicker
from bokeh.models.renderers import GlyphRenderer
from bokeh.models.annotations import Title, Legend, LegendItem
```

A long list of imports we need for a comparable plot using models interface



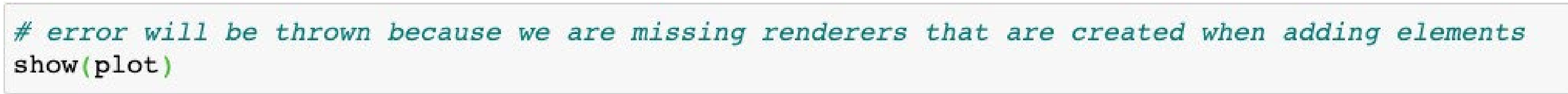
```
# defining the range for the x and y axis
extracted_mean_pop_vals = [val for i, val in enumerate(mean_pop_vals)
                          if i not in [0, len(mean_pop_vals) - 1]]
extracted_jp_vals = [jp_val['Japan'] for i, jp_val in enumerate(jp_vals)
                          if i not in [0, len(jp_vals) - 1]]

min_pop_density = min(extracted_mean_pop_vals)
min_jp_densitiy = min(extracted_jp_vals)
min_y = int(min(min_pop_density, min_jp_densitiy))

max_pop_density = max(extracted_mean_pop_vals)
max_jp_densitiy = max(extracted_jp_vals)
max_y = int(max(max_jp_densitiy, max_pop_density))

xdr = Range1d(int(years[0]), int(years[-1]))
ydr = Range1d(min_y, max_y)
```

We have to find out the **min** and **max** values for the y-axis to decide the range of axis



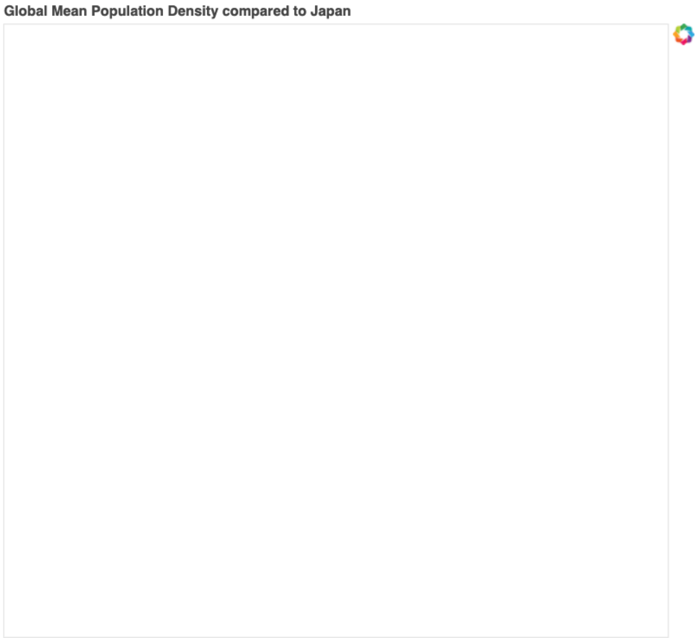
```
# creating the plot object
title = Title(
    align = 'left',
    text = 'Global Mean Population Density compared to Japan'
)

plot = Plot(
    x_range=xdr,
    y_range=ydr,
    plot_width=650,
    plot_height=600,
    title=title
)
```

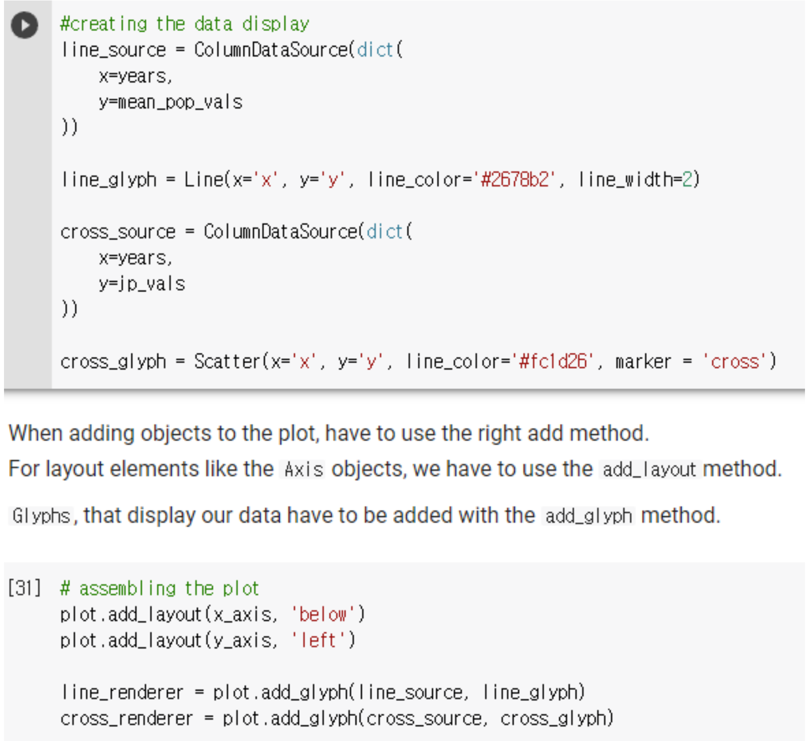
Create two **Axis** objects that will be used to display the axis lines and the label for the axis



```
# error will be thrown because we are missing renderers that are created when adding elements
show(plot)
```

If we try to display our plot now using the show method, we will get an error, because we have no renderers defined at the moment.





```
#creating the data display
line_source = ColumnDataSource(dict(
    x=years,
    y=mean_pop_vals
))

line_glyph = Line(x='x', y='y', line_color='#2678b2', line_width=2)

cross_source = ColumnDataSource(dict(
    x=years,
    y=jp_vals
))

cross_glyph = Scatter(x='x', y='y', line_color='#fc1d26', marker = 'cross')
```

When adding objects to the plot, have to use the right add method.

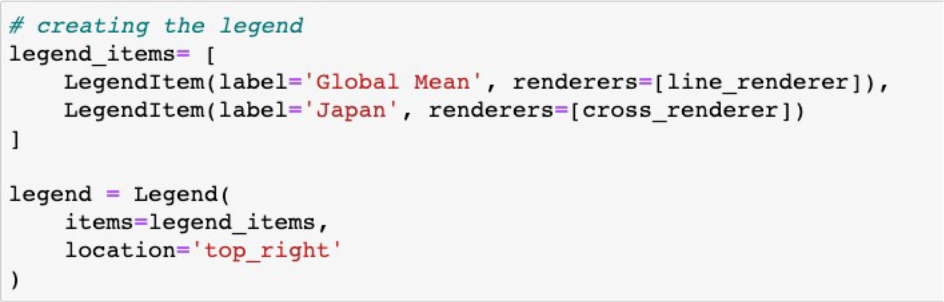
For layout elements like the `Axis` objects, we have to use the `add_layout` method.

`Glyphs`, that display our data have to be added with the `add_glyph` method.

```
[31] # assembling the plot
     plot.add_layout(x_axis, 'below')
     plot.add_layout(y_axis, 'left')

     line_renderer = plot.add_glyph(line_source, line_glyph)
     cross_renderer = plot.add_glyph(cross_source, cross_glyph)
```

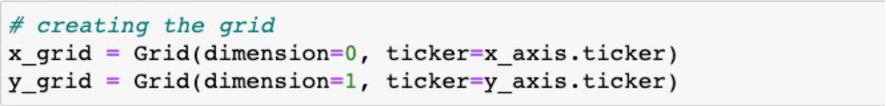
Insert data into a DataSource object that will be displayed in a plot

Add objects to the plot

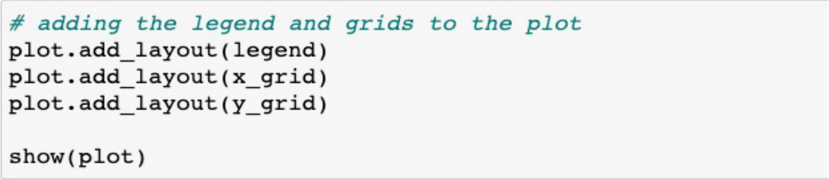


```
# creating the legend
legend_items= [
    LegendItem(label='Global Mean', renderers=[line_renderer]),
    LegendItem(label='Japan', renderers=[cross_renderer])
]

legend = Legend(
    items=legend_items,
    location='top_right'
)
```

Create legend objects



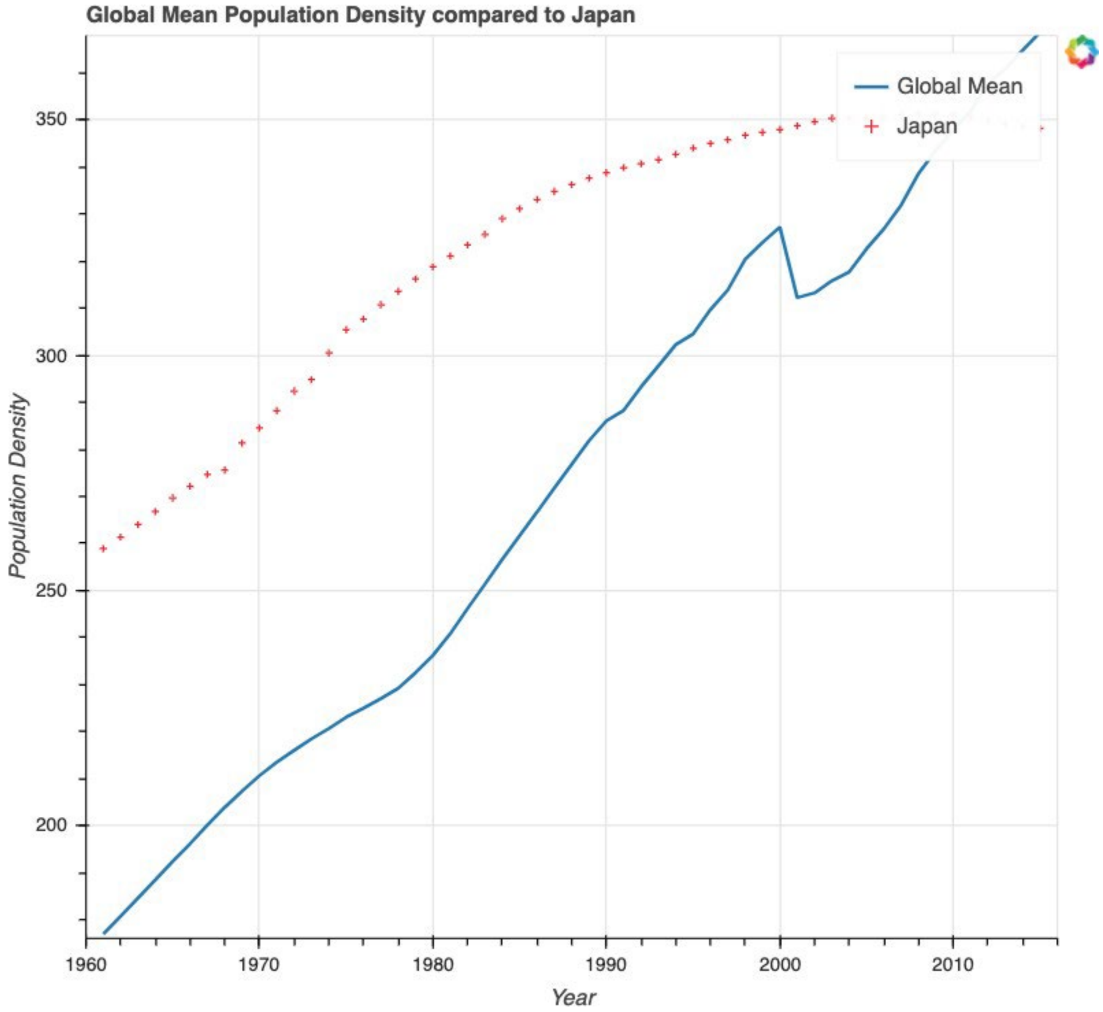
```
# creating the grid
x_grid = Grid(dimension=0, ticker=x_axis.ticker)
y_grid = Grid(dimension=1, ticker=y_axis.ticker)
```



```
# adding the legend and grids to the plot
plot.add_layout(legend)
plot.add_layout(x_grid)
plot.add_layout(y_grid)

show(plot)
```

Add grid and legend objects to the plot

Models interface-based plot displaying the lines and axes

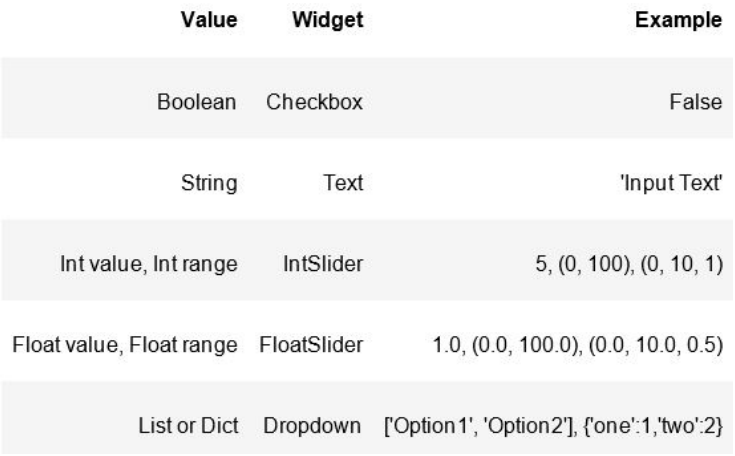


Adding widgets

One of the most powerful features of Bokeh is its ability to use **widgets** to interactively change the data that's displayed in the visualization.

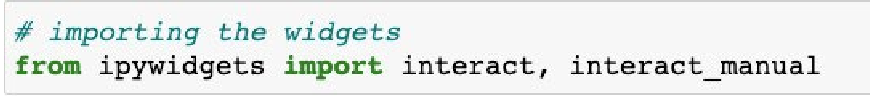
Using widgets, you can guide the user by restricting values and only displaying what you want them to see.

Developing a story behind your visualization is very important, and doing this is much easier if the user has ways of interacting with the data with widgets.

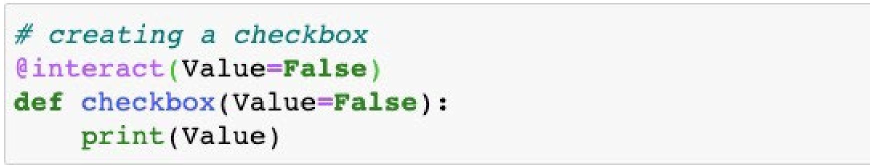


Value	Widget	Example
Boolean	Checkbox	False
String	Text	'Input Text'
Int value, Int range	IntSlider	5, (0, 100), (0, 10, 1)
Float value, Float range	FloatSlider	1.0, (0.0, 100.0), (0.0, 10.0, 0.5)
List or Dict	Dropdown	['Option1', 'Option2'], {'one':1,'two':2}

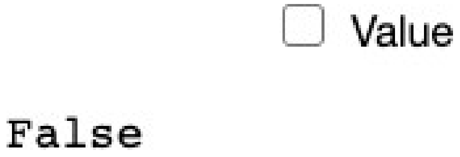
Interactive checkbox



```python
# importing the widgets
from ipywidgets import interact, interact_manual
```



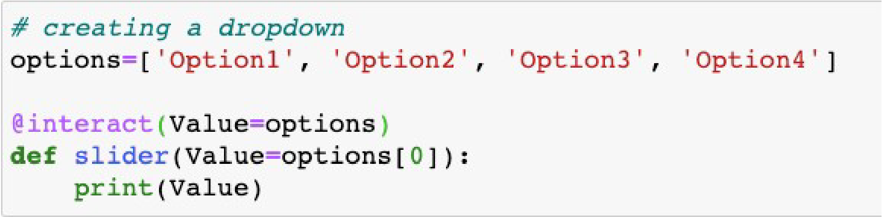
```python
# creating a checkbox
@interact(Value=False)
def checkbox(Value=False):
    print(Value)
```



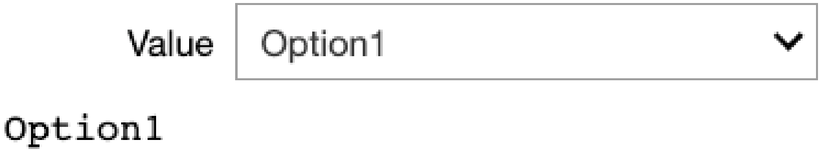
□ Value

```
False
```

Interactive dropdown



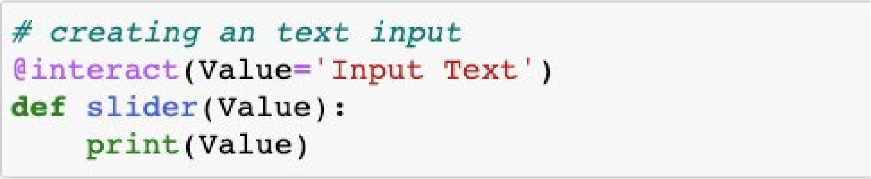
```python
# creating a dropdown
options=['Option1', 'Option2', 'Option3', 'Option4']

@interact(Value=options)
def slider(Value=options[0]):
    print(Value)
```



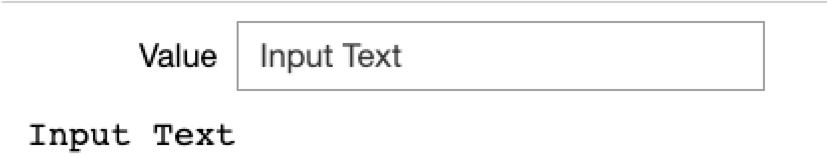
Value Option1

```
Option1
```

Interactive text input



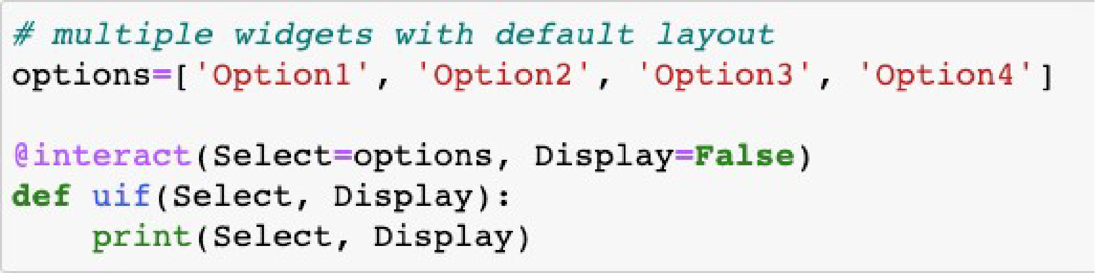
```python
# creating an text input
@interact(Value='Input Text')
def slider(Value):
    print(Value)
```



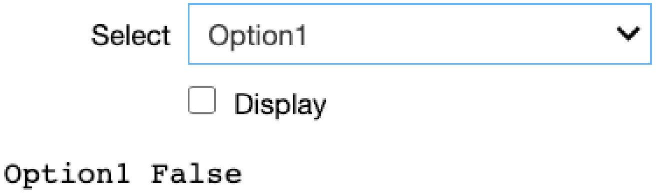
Value Input Text

```
Input Text
```

Two widgets displayed vertically by default



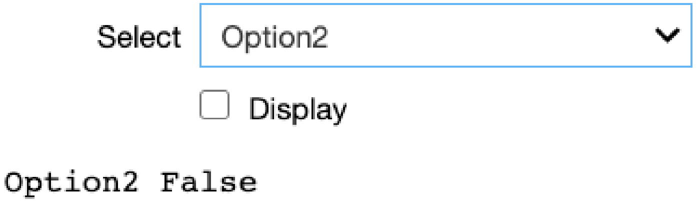
```
# multiple widgets with default layout
options=['Option1', 'Option2', 'Option3', 'Option4']

@interact(Select=options, Display=False)
def uif(Select, Display):
    print(Select, Display)
```



Select: Option1

☐ Display

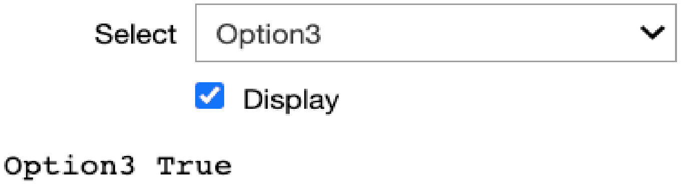
```
Option1 False
```



Select: Option2

☐ Display

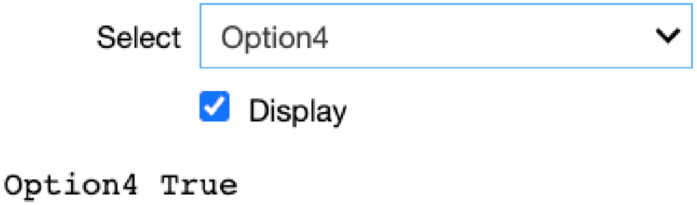
```
Option2 False
```



Select: Option3

☑ Display

```
Option3 True
```

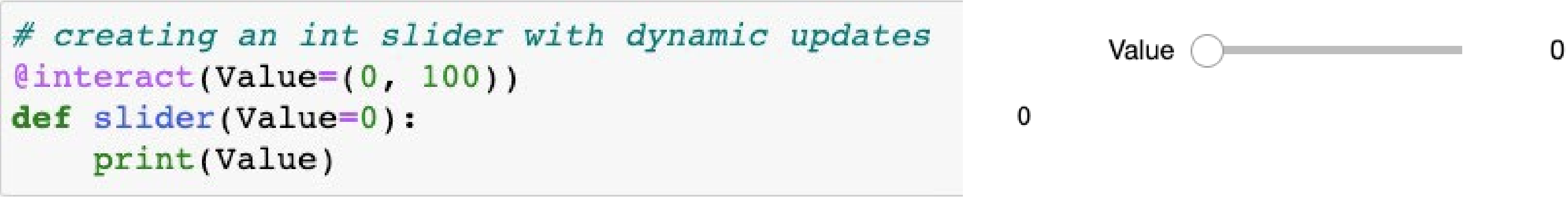


Select: Option4

☑ Display

```
Option4 True
```

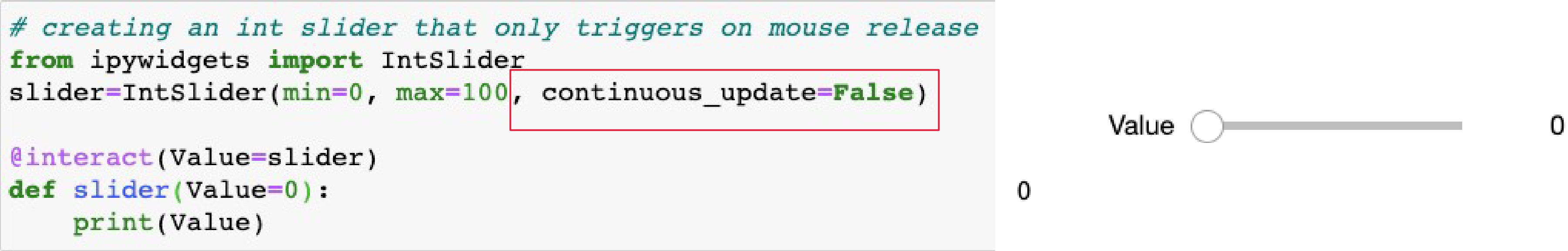
Interactive int slider



```
# creating an int slider with dynamic updates
@interact(Value=(0, 100))
def slider(Value=0):
    print(Value)
```

```
0
```

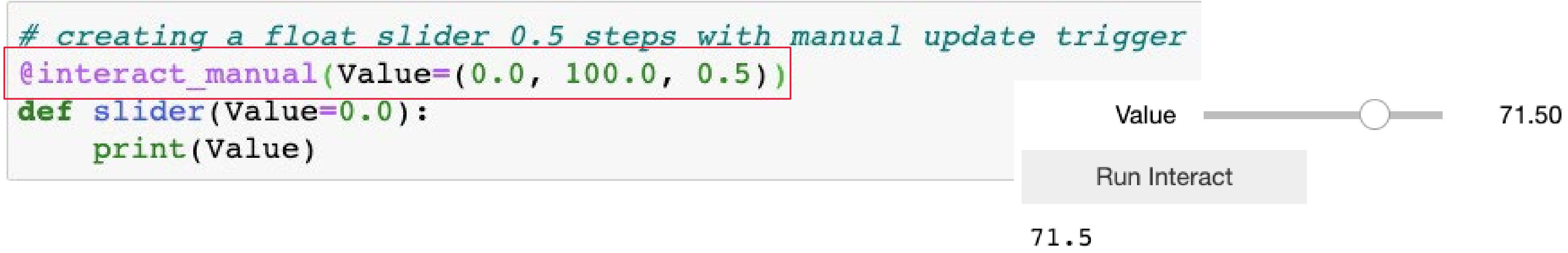
Interactive int slider that only triggers upon mouse release



```
# creating an int slider that only triggers on mouse release
from ipywidgets import IntSlider
slider=IntSlider(min=0, max=100, continuous_update=False)

@interact(Value=slider)
def slider(Value=0):
    print(Value)
```

```
0
```

Interactive int slider with a manual update trigger



```
# creating a float slider 0.5 steps with manual update trigger
@interact_manual(Value=(0.0, 100.0, 0.5))
def slider(Value=0.0):
    print(Value)
```

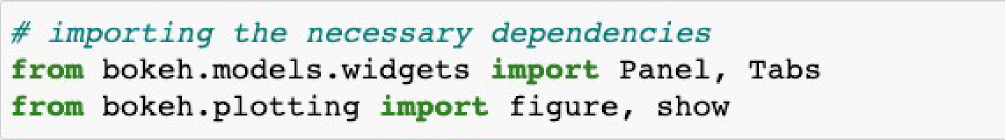
```
71.5
```

Activity 3

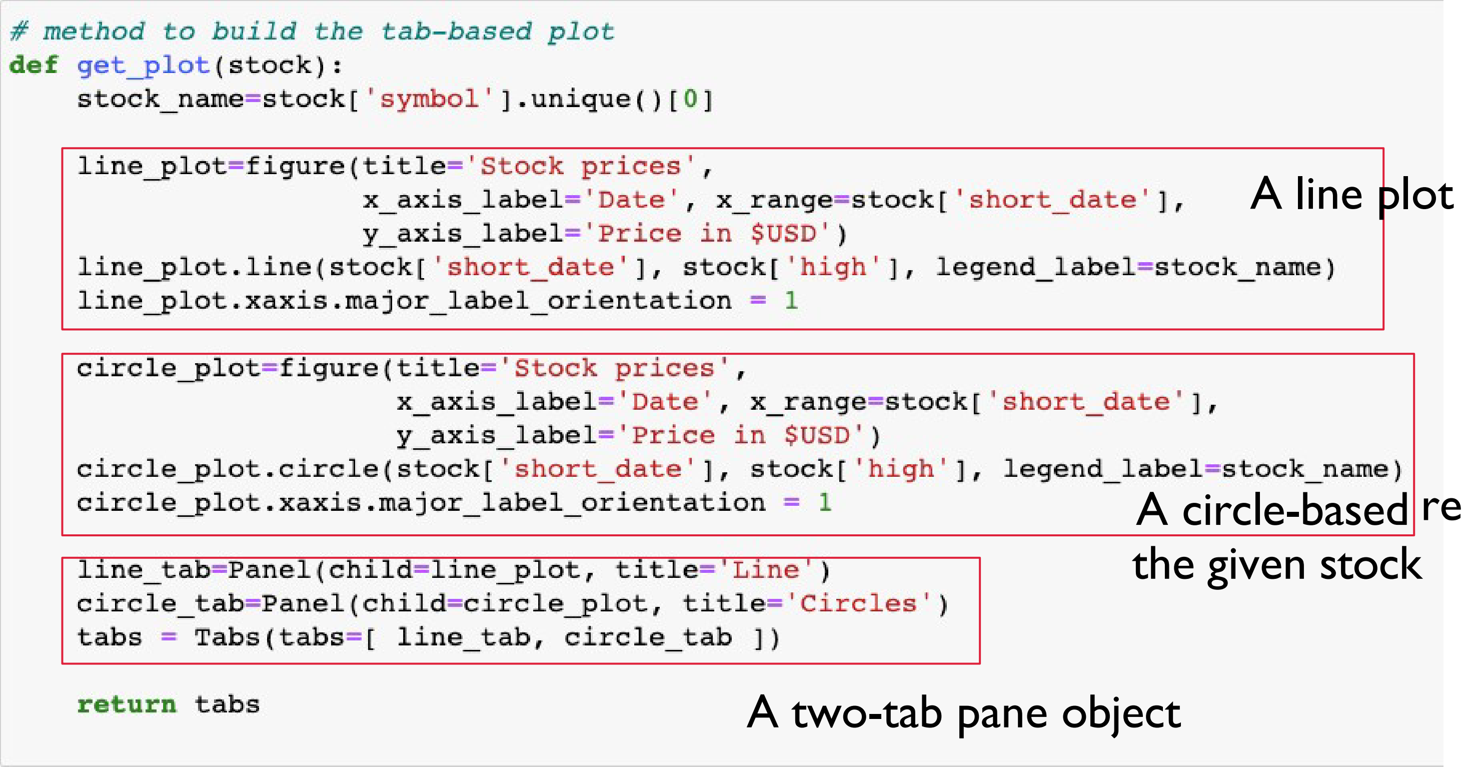
Basic Interactivity with Widgets

Objective

In this activity, we will create an interactive visualization with the stock price dataset.



```python
# importing the necessary dependencies
from bokeh.models.widgets import Panel, Tabs
from bokeh.plotting import figure, show
```

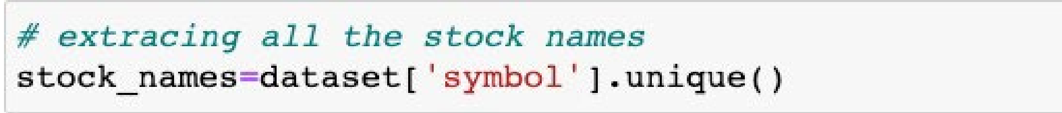


```python
# method to build the tab-based plot
def get_plot(stock):
    stock_name=stock['symbol'].unique()[0]

    line_plot=figure(title='Stock prices',
                     x_axis_label='Date', x_range=stock['short_date'],
                     y_axis_label='Price in $USD')
    line_plot.line(stock['short_date'], stock['high'], legend_label=stock_name)
    line_plot.xaxis.major_label_orientation = 1

    circle_plot=figure(title='Stock prices',
                       x_axis_label='Date', x_range=stock['short_date'],
                       y_axis_label='Price in $USD')
    circle_plot.circle(stock['short_date'], stock['high'], legend_label=stock_name)
    circle_plot.xaxis.major_label_orientation = 1

    line_tab=Panel(child=line_plot, title='Line')
    circle_tab=Panel(child=circle_plot, title='Circles')
    tabs = Tabs(tabs=[ line_tab, circle_tab ])

    return tabs
```

A line plot of a given stock

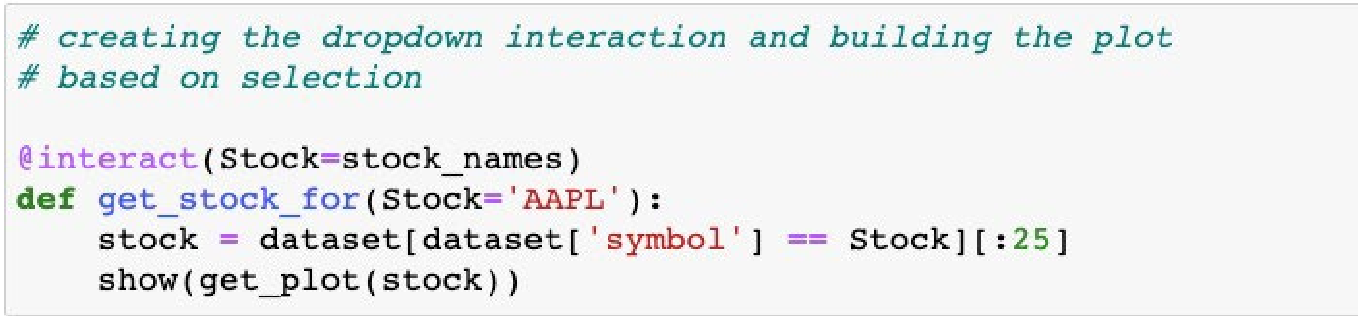
A circle-based representation of the given stock

A two-tab pane object



```python
# extracing all the stock names
stock_names=dataset['symbol'].unique()
```

Get a list of stock names in the data

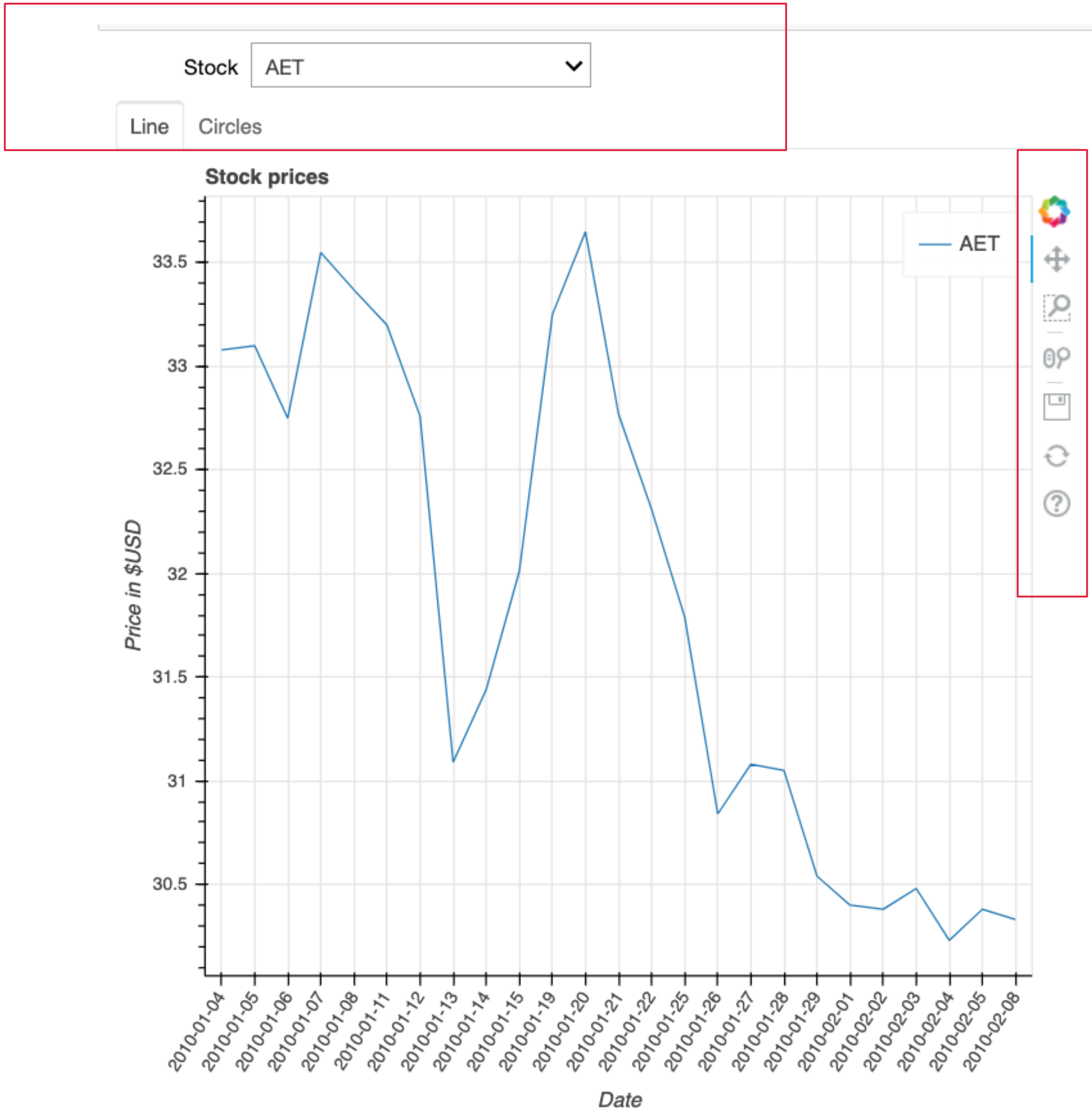


```python
# creating the dropdown interaction and building the plot
# based on selection

@interact(Stock=stock_names)
def get_stock_for(Stock='AAPL'):
    stock = dataset[dataset['symbol'] == Stock][:25]
    show(get_plot(stock))
```

Use the list above as an input for the interact element

Add a dropdown widget

Tabs and dropdown menu



Stock: AET

Inspector