Activity 7.02: Visualizing Stock Prices with Bokeh

This activity will combine most of what you have already learned about Bokeh. You will also need the skills you have acquired while working with Pandas.

We will create an interactive visualization that displays a candle stick plot, which is often used when handling stock price data.
We will be able to compare two stocks with each other by selecting them from dropdowns.
A RangeSlider will allow us to restrict the displayed date range in the requested year 2016.
Depending on what graph we choose, we will either see the candle stick visualization or a simple line plot displaying the volume of the selected stock.

The dataset of this exercise contains temporal stock price data.
This means we'll be looking at data over a range of time.

Loading our dataset

In [1]:
```python
# importing the necessary dependencies
import pandas as pd
from bokeh.io import output_notebook

output_notebook()
```

```
(https://bokeh.org) BokehJS 2.0.2 successfully loaded.
```

In [2]:
```python
# loading the Dataset with geoplotlib
dataset = pd.read_csv('../../Datasets/stock_prices.csv')
```

In [3]:
```python
# looking at the dataset
dataset.head()
```

Out[3]:

	date	symbol	open	close	low	high	volume
0	2016-01-05 00:00:00	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06 00:00:00	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07 00:00:00	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08 00:00:00	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11 00:00:00	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

Just as in the previous exercise, we want to map the date column to another column with the shortened date that only contains the year, month, and day.

In [4]:
```
# mapping the date of each row to only the year-month-day format
from datetime import datetime

def shorten_time_stamp(timestamp):
    shortened = timestamp[0]

    if len(shortened) > 10:
        parsed_date=datetime.strptime(shortened, '%Y-%m-%d %H:%M:%S')
        shortened=datetime.strftime(parsed_date, '%Y-%m-%d')

    return shortened

dataset['short_date'] = dataset.apply(lambda x: shorten_time_stamp(x),
```

Note:
The exectuion of the cell will take a moment since it's a fairly large dataset.
Please be patient.

In [5]:
```
# looking at the dataset with shortened dat
dataset.head()
```

Out[5]:

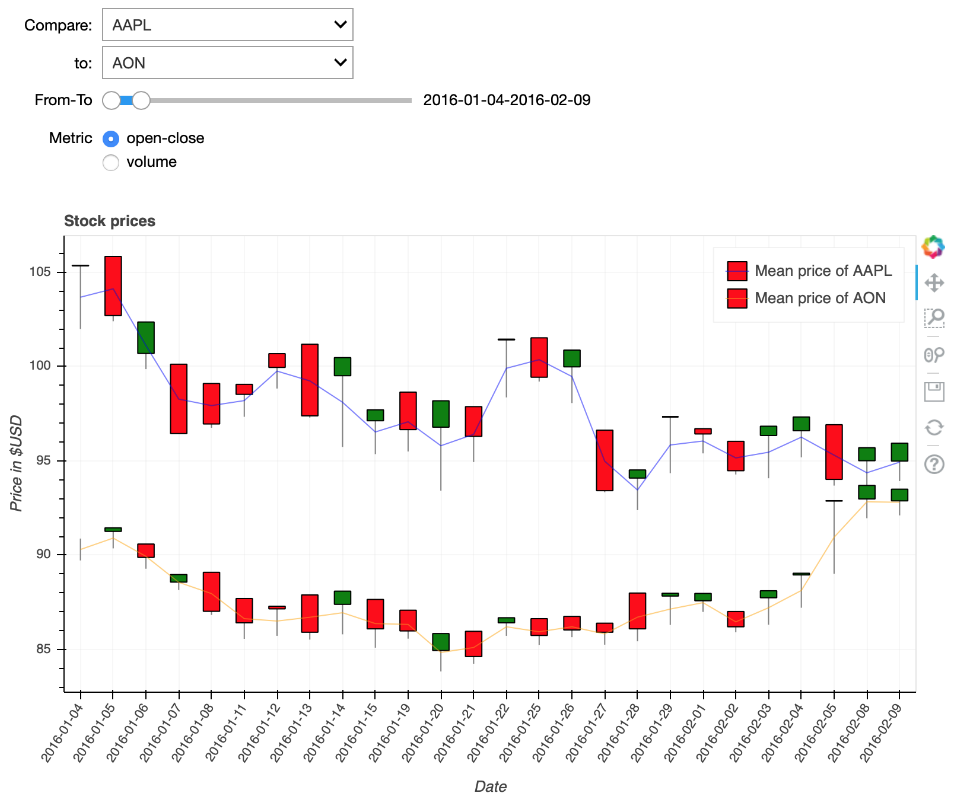
	date	symbol	open	close	low	high	volume	short_date
0	2016-01-05 00:00:00	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0	2016-01-05
1	2016-01-06 00:00:00	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0	2016-01-06
2	2016-01-07 00:00:00	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0	2016-01-07
3	2016-01-08 00:00:00	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0	2016-01-08
4	2016-01-11 00:00:00	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0	2016-01-11

Note:
The last, newly added, column now holds the timestamp without the hour, minute, and second information.

Building an interactive visualization

There are many options when it comes to choosing which interactivity to use.
Since the goal of this activity is to be able to compare two stocks with each other in terms of traded volume and the high/low and open/close prices over a time range, we will need widgets to select elements and a slider to select a given range.
Considering that we have to options of display, we also need a way to select either one or the other.

At the end of this task, you will have something comparable to image below implemented and ready to compare data from the dataset.


Compare: AAPL

to: AON

From-To 2016-01-04-2016-02-09

Metric [✓ open-close] volume

In [6]:
```python
# importing the necessary dependencies
from bokeh.plotting import figure, show
from ipywidgets import interact, widgets
```

Before we go in and implement the plotting methods, we want to set up the interactivity widgets. **Please scroll down** to the cell that says `# extracing the necessary data` before implementing the plotting.

Still make sure to execute the cells below that will simply `pass` and do nothing for now.

At the moment, our `show()` in the last cell will not render any elements into our visualization.
We will start with the, so called, candle stick visualization which is often used with stock price data.

The already defined method below gets our `plot` object, a `stock_name`, a `stock_range` containing the data for the, with the widgets, selected range, and a color for the line.
We will use those arguments to create the candle sticks. They basically contain a `segment` that creates the vertical line, and either a green or red `vbar` to color code whether the close price is lower than the open price.

Once you have created the candles, we also want to have a continuous line running through the mean (high, low) point of each candle. So you have to calculate the mean for every high, low pair and then plot those data points with a line with the given `color`.

Make sure to reference the example provided in the Bokeh library here, you can adapt the code in there to our arguments. https://bokeh.pydata.org/en/latest/docs/gallery/candlestick.html (https://bokeh.pydata.org/en/latest/docs/gallery/candlestick.html)

After you are done implementing the `add_candle_plot` method, scroll down and run the `@interact` cell again.
You will now see the candles being displayed for the two selected stocks.

Note:
Since we are providing the `plot` as a reference to the method, we don't need to return anything since we are mutating our passed in plot.

In [7]:
```
def add_candle_plot(plot, stock_name, stock_range, color):
    inc_1 = stock_range.close > stock_range.open
    dec_1 = stock_range.open > stock_range.close
    w = 0.5

    plot.segment(stock_range['short_date'], stock_range['high'],
                 stock_range['short_date'], stock_range['low'],
                 color="grey")

    plot.vbar(stock_range['short_date'][inc_1], w,
              stock_range['high'][inc_1], stock_range['close'][inc_1],
              fill_color="green", line_color="black",
              legend_label=('Mean price of ' + stock_name), muted_alpha

    plot.vbar(stock_range['short_date'][dec_1], w,
              stock_range['high'][dec_1], stock_range['close'][dec_1],
              fill_color="red", line_color="black",
              legend_label=('Mean price of ' + stock_name), muted_alpha

    stock_mean_val=stock_range[['high', 'low']].mean(axis=1)
    plot.line(stock_range['short_date'], stock_mean_val,
              legend_label=('Mean price of ' + stock_name), muted_alpha
              line_color=color, alpha=0.5)
```

The last missing step is implementing the plotting of the lines if the `volume` value is selected. We've created simple lines in the previous exercise, so this should not be a problem.

One additional interaction feature is to have an interactive legend that allows us to "mute", meaning grey out, each stock in the visualization.
To make our legend interactive please take a look at the documentation for the legend feature. https://bokeh.pydata.org/en/latest/docs/user_guide/interaction/legends.html (https://bokeh.pydata.org/en/latest/docs/user_guide/interaction/legends.html)

Note:
Don't forget to update your `add_canlde_plot` `vbar` s and `segment` to also include the `muted_alpha` parameter. Otherwise you won't be able to mute the stocks in the "open-close" visualization.

In [8]:
```
# method to build the plot
def get_plot(stock_1, stock_2, date, value):
    stock_1 = dataset[dataset['symbol'] == stock_1]
    stock_2 = dataset[dataset['symbol'] == stock_2]

    stock_1_name=stock_1['symbol'].unique()[0]
    stock_1_range=stock_1[(stock_1['short_date'] >= date[0]) & (stock_1
    stock_2_name=stock_2['symbol'].unique()[0]
    stock_2_range=stock_2[(stock_2['short_date'] >= date[0]) & (stock_2

    plot=figure(title='Stock prices',
                    x_axis_label='Date',
                    x_range=stock_1_range['short_date'],
                    y_axis_label='Price in $USD',
                    plot_width=800,
                    plot_height=500)

    plot.xaxis.major_label_orientation = 1
    plot.grid.grid_line_alpha=0.3

    if value == 'open-close':
        add_candle_plot(plot, stock_1_name, stock_1_range, 'blue')
        add_candle_plot(plot, stock_2_name, stock_2_range, 'orange')

    if value == 'volume':
        plot.line(stock_1_range['short_date'], stock_1_range['volume'],
                  legend_label=stock_1_name, muted_alpha=0.2)
        plot.line(stock_2_range['short_date'], stock_2_range['volume'],
                  legend_label=stock_2_name, muted_alpha=0.2,
                  line_color='orange')

    plot.legend.click_policy="mute"

    return plot

```

We want to **start implementing our visualization here**.

In the following cells, we will extract the necessary data which will be provided to the widget elements.
In the first cell we want to extract the following information:

- a list of unique stock names that are present in the dataset
- a list of all short_dates that are in 2016
- a sorted list of unique dates generated from the previous list of dates from 2016
- a list with the values `open-close` and `volume`

Once we have this information in place, we can start building our widgets.

In [9]:
```python
# extracing the necessary data
stock_names=dataset['symbol'].unique()
dates_2016=dataset[dataset['short_date'] >= '2016-01-01']['short_date']
unique_dates_2016=sorted(dates_2016.unique())
value_options=['open-close', 'volume']
```

Given the extracted information from the cell above, we can now define the widgets and provide the available options to it.

As mentioned in the introduction, we want to have several interactive features including:

- two `Dropdown` s with which we can select two stocks that should be compared to each other
  - the first dropdown by default should have the `AAPL` stock selected, named "Compare: "
  - the second dropdown by default should have the `AON` stock selected, named "to: "
- a `SelectionRange` which will allow us to select a range of dates from the extracted list of unique 2016 dates
  - by default, the first 25 dates should be selected, named "From-To"
  - make sure to disable the `continuous_update` parameter here
  - adjust the layout width to 500px to make sure the dates are displayed correctly
- a `RadioButton` group that provides the options "open-close" and "volume"
  - by default, "open-close" should be selected, named "Metric"

In [10]:
```python
# setting up the interaction elements
drp_1=widgets.Dropdown(options=stock_names,
                       value='AAPL',
                       description='Compare:')

drp_2=widgets.Dropdown(options=stock_names,
                       value='AON',
                       description='to:')

range_slider=widgets.SelectionRangeSlider(options=unique_dates_2016,
                                          index=(0,25),
                                          continuous_update=False,
                                          description='From-To',
                                          layout={'width': '500px'})

value_radio=widgets.RadioButtons(options=value_options,
                                 value='open-close',
                                 description='Metric')
```

Note:
As mentioned in the previous exercise, we can also make use of the widgets described here:
https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html
(https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html)

After setting up the widgets, we can the method that will be called with each update of the interaction widgets.
As seen in the previous exercise, we will use the `@interact` decorator for this.

Instead of value ranges or lists, we will provide the variable names of our already created widgets in the decorator.
The method will get 4 arguments, `stock_1` , `stock_2` , `date` , and `value` .
Since we have already set up the empty method that will return a plot above, we can call `show()` with the method call inside to show the result once it is returned from the `get_stock_for_2016` method.

Once you've build the widgets, upon execution, you will see them being displayed below the cell.
We are now ready to to **scroll up and implement the plotting** with Bokeh.

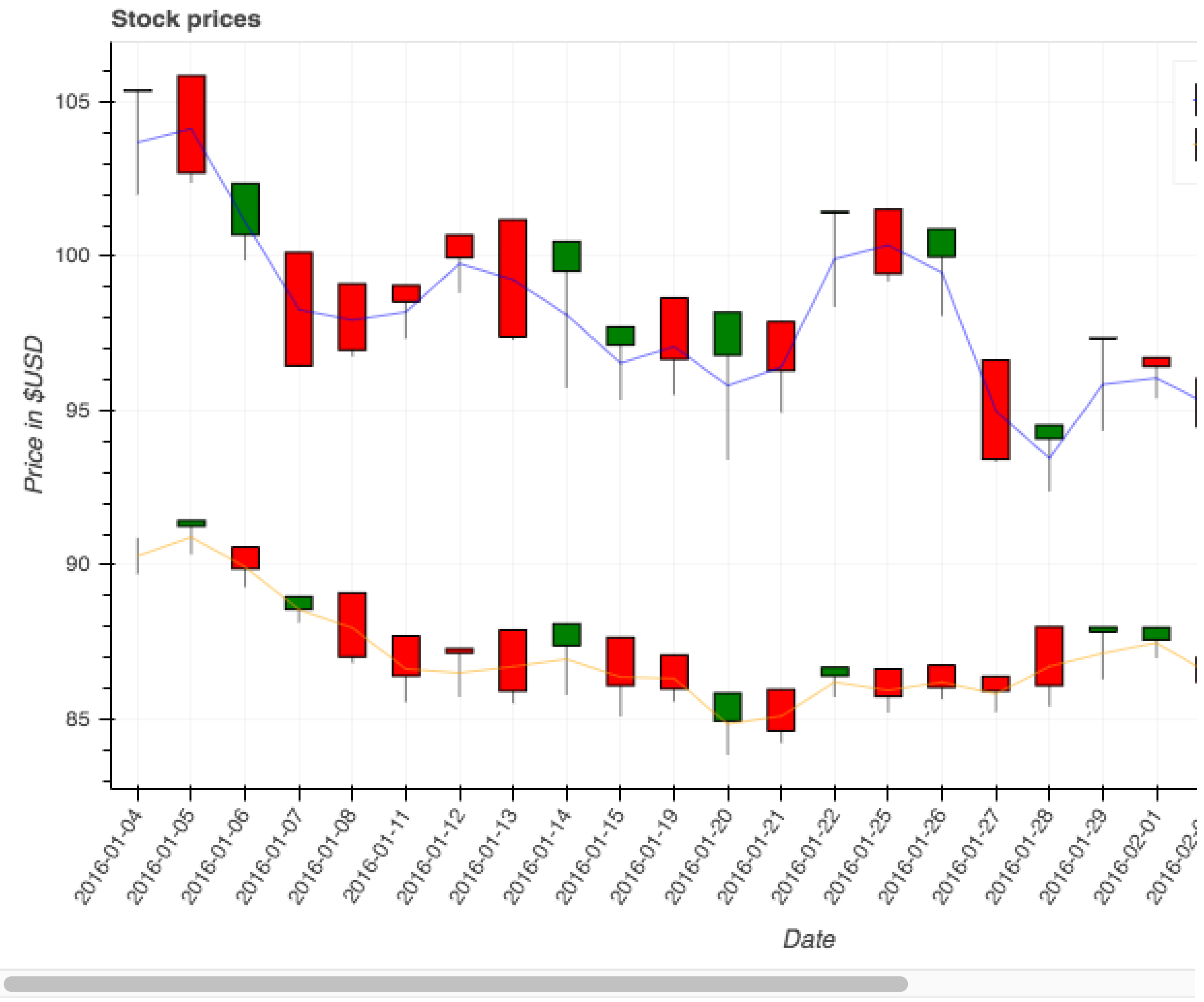
In [11]:
```
# creating the interact method
@interact(stock_1=drp_1, stock_2=drp_2, date=range_slider, value=value_
def get_stock_for_2016(stock_1, stock_2, date, value):
    show(get_plot(stock_1, stock_2, date, value))
```

Compare: AAPL

to: AON

From-To 2016-01-04-2016-02-09

Metric [✓ open-close] volume



This is a nice example that shows us how much interaction we can add to a visualization with very simple techniques such as using the interact functionality.

Note:
Think about what else you could add/change for this visualization. Maybe we don't only want to display 2016 but be able to select the year we want to display. Maybe we want to compare different years with each other.
There are endless options.