Basic Plotting with matplotlib

You can show matplotlib figures directly in the notebook by using the `%matplotlib notebook` and `%matplotlib inline` magic commands.

`%matplotlib notebook` provides an interactive environment.

```
%matplotlib notebook
```

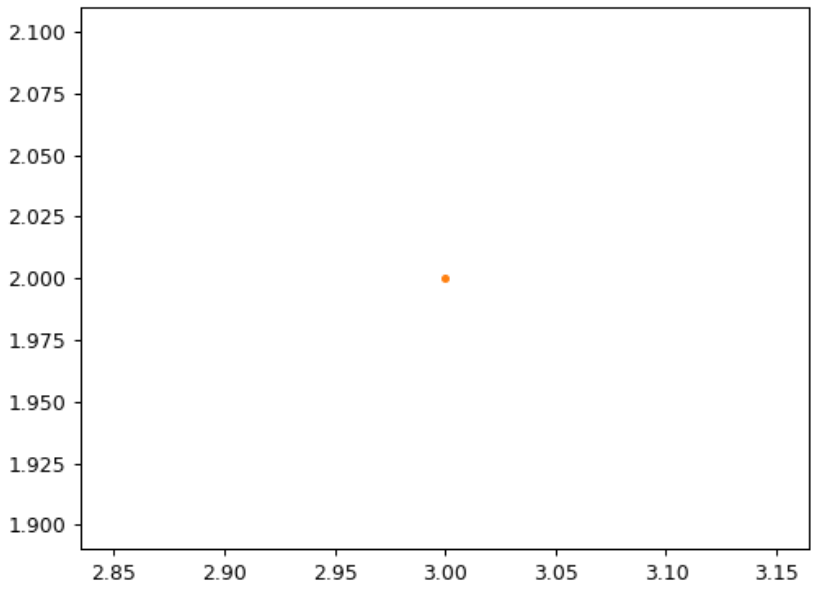
```
import matplotlib as mpl
mpl.get_backend()
```

```
'nbAgg'
```

```
import matplotlib.pyplot as plt
# plt.plot?
```

```
# because the default is the line style '-',
# nothing will be shown if we only pass in one point (3,2)
plt.plot(3, 2)
```

```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fbddb9c36a0>]
```

```
# we can pass in '.' to plt.plot to indicate that we want
# the point (3,2) to be indicated with a marker '.'
plt.plot(3, 2, '.')
```

```
[<matplotlib.lines.Line2D at 0x7fbddb9ce5f8>]
```

Let's see how to make a plot without using the scripting layer.

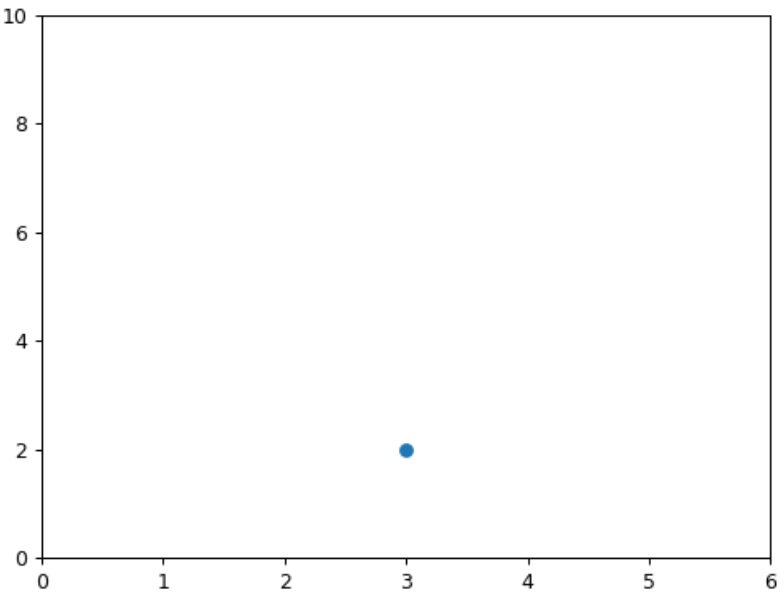
```
# First let's set the backend without using mpl.use() from the scripting layer
from matplotlib.backends.backend_agg import FigureCanvasAgg
from matplotlib.figure import Figure

# create a new figure
fig = Figure()

# associate fig with the backend
canvas = FigureCanvasAgg(fig)

# add a subplot to the fig
ax = fig.add_subplot(111)

# plot the point (3,2)
ax.plot(3, 2, '.')

# save the figure to test.png
# you can see this figure in your Jupyter workspace afterwards by going to
# https://hub.coursera-notebooks.org/
canvas.print_png('test.png')
```

We can use html cell magic to display the image.

```
%%html
<img src='test.png' />
```

```
# create a new figure
plt.figure()

# plot the point (3,2) using the circle marker
plt.plot(3, 2, 'o')

# get the current axes
ax = plt.gca()

# Set axis properties [xmin, xmax, ymin, ymax]
ax.axis([0,6,0,10])
```

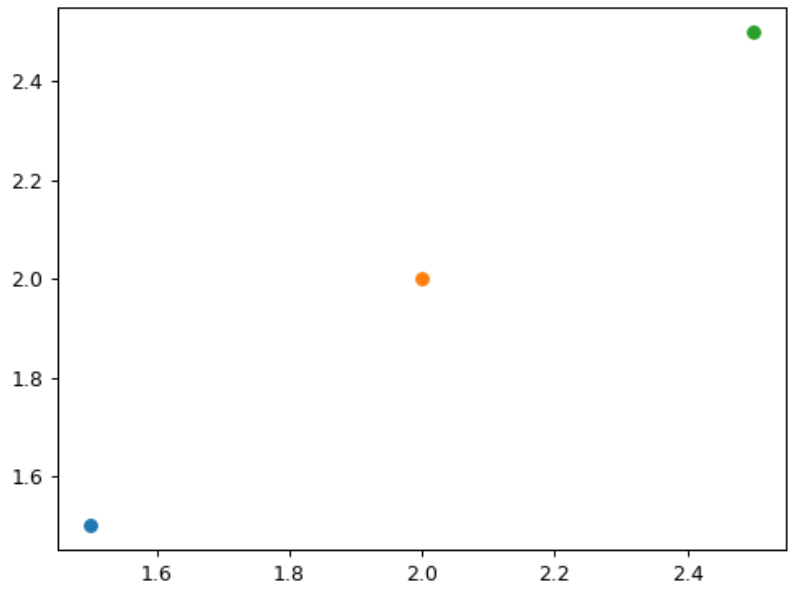
```
<IPython.core.display.Javascript object>
```



```
[0, 6, 0, 10]
```

```
# create a new figure
plt.figure()

# plot the point (1.5, 1.5) using the circle marker
plt.plot(1.5, 1.5, 'o')
# plot the point (2, 2) using the circle marker
plt.plot(2, 2, 'o')
# plot the point (2.5, 2.5) using the circle marker
plt.plot(2.5, 2.5, 'o')
```

```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fbddb4e9ba8>]
```

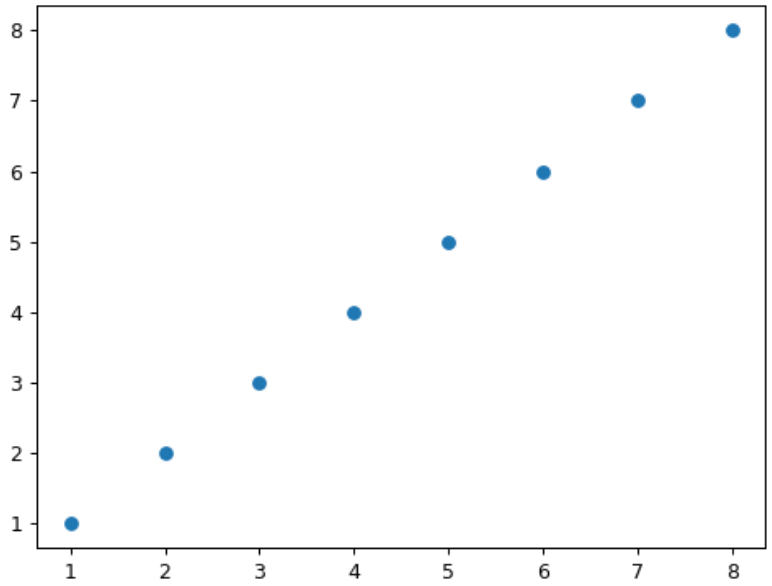
```
# get current axes
ax = plt.gca()
# get all the child objects the axes contains
ax.get_children()
```

```
[<matplotlib.lines.Line2D at 0x7fbddb4e9240>,
 <matplotlib.lines.Line2D at 0x7fbddb5124a8>,
 <matplotlib.lines.Line2D at 0x7fbddb4e9ba8>,
 <matplotlib.spines.Spine at 0x7fbddb4fee48>,
 <matplotlib.spines.Spine at 0x7fbddb4fe588>,
 <matplotlib.spines.Spine at 0x7fbddba1e518>,
 <matplotlib.spines.Spine at 0x7fbddba2ff98>,
 <matplotlib.axis.XAxis at 0x7fbddb5096a0>,
 <matplotlib.axis.YAxis at 0x7fbddb524b00>,
 <matplotlib.text.Text at 0x7fbddb4b68d0>,
 <matplotlib.text.Text at 0x7fbddb4b6940>,
 <matplotlib.text.Text at 0x7fbddb4b69b0>,
 <matplotlib.patches.Rectangle at 0x7fbddb4b69e8>]
```

Scatterplots

```
import numpy as np

x = np.array([1,2,3,4,5,6,7,8])
y = x

plt.figure()
plt.scatter(x, y) # similar to plt.plot(x, y, '.'), but the underlying child objects in the axes are not Line2D
```

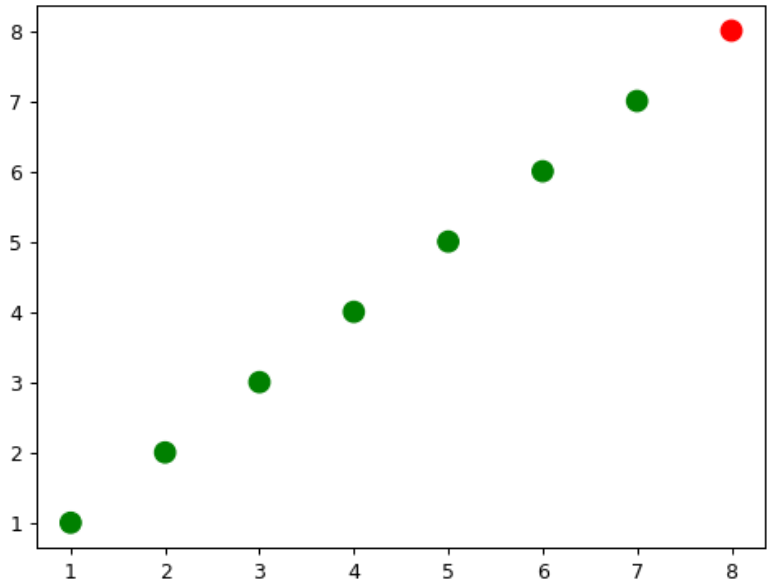
```
<IPython.core.display.Javascript object>
```




```
<matplotlib.collections.PathCollection at 0x7fbddb3fd4e0>
```

```
import numpy as np

x = np.array([1,2,3,4,5,6,7,8])
y = x

# create a list of colors for each point to have
# ['green', 'green', 'green', 'green', 'green', 'green', 'green', 'red']
colors = ['green']*(len(x)-1)
colors.append('red')

plt.figure()

# plot the point with size 100 and chosen colors
plt.scatter(x, y, s=100, c=colors)
```

```
<IPython.core.display.Javascript object>
```




```
<matplotlib.collections.PathCollection at 0x7fbddb3aa6d8>
```

```
# convert the two lists into a list of pairwise tuples
zip_generator = zip([1,2,3,4,5], [6,7,8,9,10])

print(list(zip_generator))
# the above prints:
# [(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]

zip_generator = zip([1,2,3,4,5], [6,7,8,9,10])
# The single star * unpacks a collection into positional arguments
print(*zip_generator)
# the above prints:
# (1, 6) (2, 7) (3, 8) (4, 9) (5, 10)
```

```
[(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]
(1, 6) (2, 7) (3, 8) (4, 9) (5, 10)
```

```
# use zip to convert 5 tuples with 2 elements each to 2 tuples with 5 elements each
print(list(zip((1, 6), (2, 7), (3, 8), (4, 9), (5, 10))))
# the above prints:
# [(1, 2, 3, 4, 5), (6, 7, 8, 9, 10)]
```

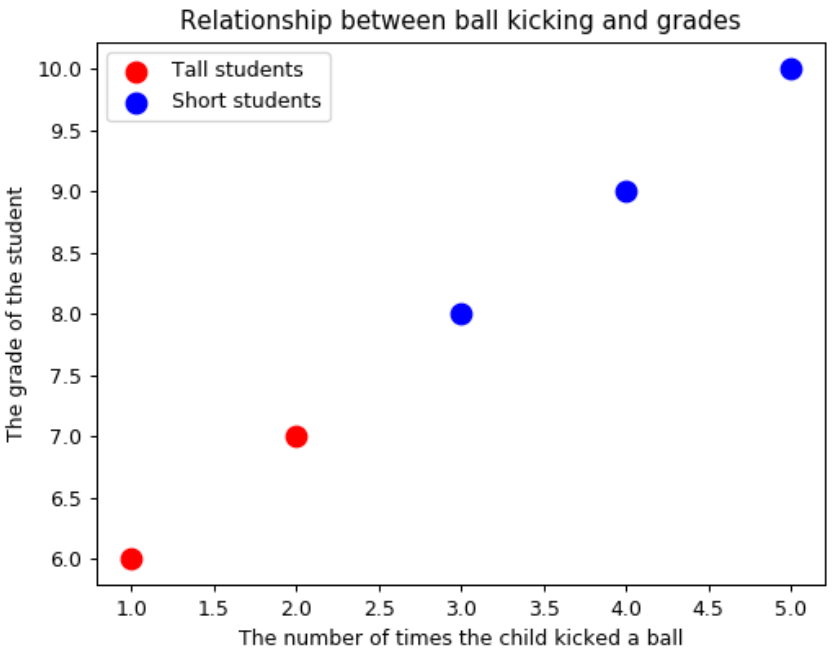
```
zip_generator = zip([1,2,3,4,5], [6,7,8,9,10])
# let's turn the data back into 2 lists
x, y = zip(*zip_generator) # This is like calling zip((1, 6), (2, 7), (3, 8), (4, 9), (5, 10))
print(x)
print(y)
# the above prints:
# (1, 2, 3, 4, 5)
# (6, 7, 8, 9, 10)
```

```
[(1, 2, 3, 4, 5), (6, 7, 8, 9, 10)]
(1, 2, 3, 4, 5)
(6, 7, 8, 9, 10)
```

```
plt.figure()
# plot a data series 'Tall students' in red using the first two elements of x and y
plt.scatter(x[:2], y[:2], s=100, c='red', label='Tall students')
```

```
# plot a second data series 'Short students' in blue using the last three elements of x and y
plt.scatter(x[2:], y[2:], s=100, c='blue', label='Short students')
```

```
<IPython.core.display.Javascript object>
```




```
<matplotlib.collections.PathCollection at 0x7fbddb30c400>
```

```
# add a label to the x axis
plt.xlabel('The number of times the child kicked a ball')
# add a label to the y axis
plt.ylabel('The grade of the student')
# add a title
plt.title('Relationship between ball kicking and grades')
```

```
<matplotlib.text.Text at 0x7fbddb2be630>
```

```
# add a legend (uses the labels from plt.scatter)
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fbddb5611d0>
```

```
# add the legend to loc=4 (the lower right hand corner), also gets rid of the frame and adds a title
plt.legend(loc=4, frameon=False, title='Legend')
```

```
<matplotlib.legend.Legend at 0x7fbddb241390>
```

```
# get children from current axes (the legend is the second to last item in this list)
plt.gca().get_children()
```

```
[<matplotlib.collections.PathCollection at 0x7fbddb30c128>,
 <matplotlib.collections.PathCollection at 0x7fbddb30c400>,
 <matplotlib.spines.Spine at 0x7fbddb300908>,
 <matplotlib.spines.Spine at 0x7fbddb300b00>,
 <matplotlib.spines.Spine at 0x7fbddb300cf8>,
```

```
 <matplotlib.spines.Spine at 0x7fbddb300ef0>,
 <matplotlib.axis.XAxis at 0x7fbddb3090f0>,
 <matplotlib.axis.YAxis at 0x7fbddb3131d0>,
 <matplotlib.text.Text at 0x7fbddb2be630>,
 <matplotlib.text.Text at 0x7fbddb2be6a0>,
 <matplotlib.text.Text at 0x7fbddb2be710>,
 <matplotlib.legend.Legend at 0x7fbddb241390>,
 <matplotlib.patches.Rectangle at 0x7fbddb2b6320>]
```

```
# get the legend from the current axes
legend = plt.gca().get_children()[-2]
```

```
# you can use get_children to navigate through the child artists
legend.get_children()[0].get_children()[1].get_children()[0].get_children()
```

```
[<matplotlib.offsetbox.HPacker at 0x7fbddb241f98>,
 <matplotlib.offsetbox.HPacker at 0x7fbddb24e128>]
```

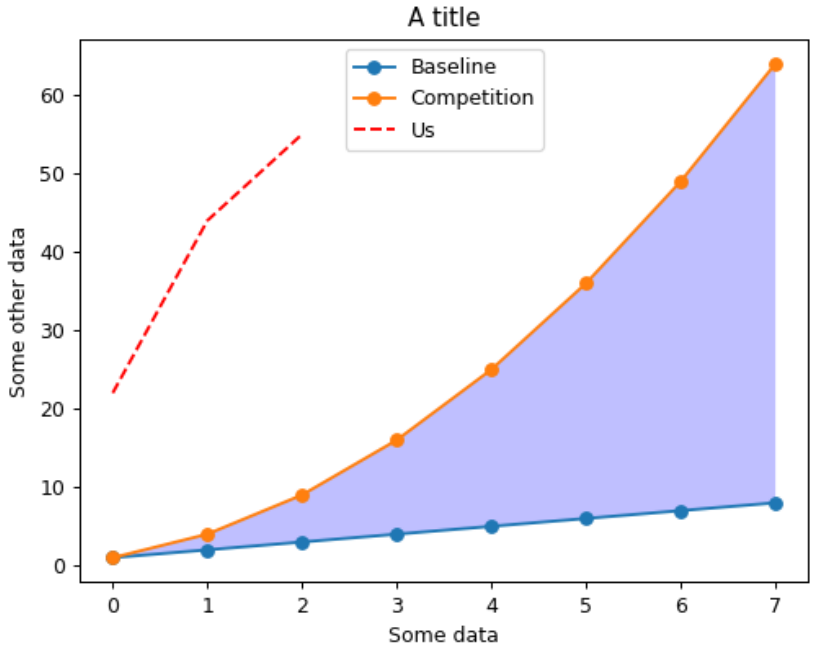
```
# import the artist class from matplotlib
from matplotlib.artist import Artist

def rec_gc(art, depth=0):
    if isinstance(art, Artist):
        # increase the depth for pretty printing
        print("  " * depth + str(art))
        for child in art.get_children():
            rec_gc(child, depth+2)

# Call this function on the legend artist to see what the legend is made up of
rec_gc(plt.legend())
```

```
Legend
    <matplotlib.offsetbox.VPacker object at 0x7fbddb256be0>
        <matplotlib.offsetbox.TextArea object at 0x7fbddb256978>
            Text(0,0,'None')
        <matplotlib.offsetbox.HPacker object at 0x7fbddb24eeb8>
            <matplotlib.offsetbox.VPacker object at 0x7fbddb24eef0>
                <matplotlib.offsetbox.HPacker object at 0x7fbddb256860>
                    <matplotlib.offsetbox.DrawingArea object at 0x7fbddb2561d0>
                        <matplotlib.collections.PathCollection object at 0x7fbddb256358>
                    <matplotlib.offsetbox.TextArea object at 0x7fbddb24ef28>
                        Text(0,0,'Tall students')
                <matplotlib.offsetbox.HPacker object at 0x7fbddb256940>
                    <matplotlib.offsetbox.DrawingArea object at 0x7fbddb2565f8>
                        <matplotlib.collections.PathCollection object at 0x7fbddb2567f0>
                    <matplotlib.offsetbox.TextArea object at 0x7fbddb2563c8>
                        Text(0,0,'Short students')
    FancyBboxPatch(0,0;1x1)
```

Line Plots

```
import numpy as np

linear_data = np.array([1,2,3,4,5,6,7,8])
exponential_data = linear_data**2

plt.figure()
# plot the linear data and the exponential data
plt.plot(linear_data, '-o', exponential_data, '-o')
```

```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fbddb3bf780>,
 <matplotlib.lines.Line2D at 0x7fbddb34a978>]
```

```
# plot another series with a dashed red line
plt.plot([22,44,55], '--r')
```

```
[<matplotlib.lines.Line2D at 0x7fbddb177780>]
```

```
plt.xlabel('Some data')
plt.ylabel('Some other data')
plt.title('A title')
# add a legend with legend entries (because we didn't have labels when we plotted the data series)
plt.legend(['Baseline', 'Competition', 'Us'])
```

```
<matplotlib.legend.Legend at 0x7fbddb183828>
```

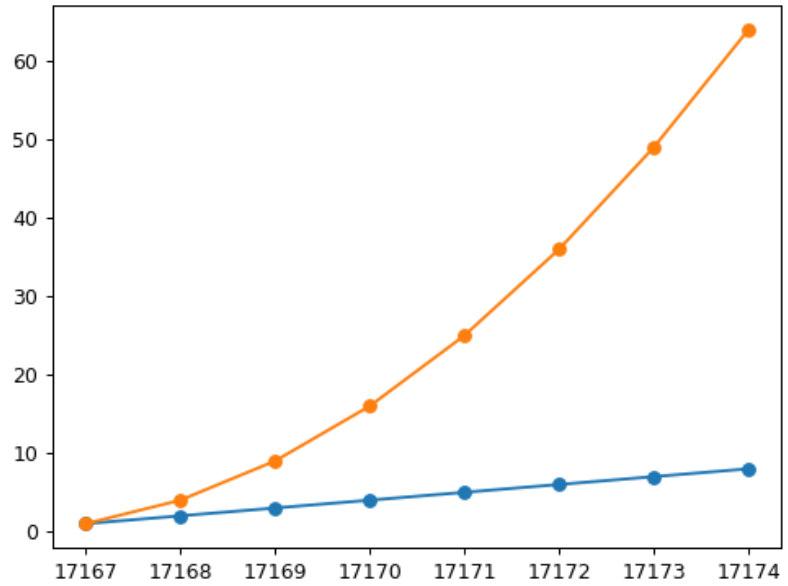
```
# fill the area between the linear data and exponential data
plt.gca().fill_between(range(len(linear_data)),
                       linear_data, exponential_data,
                       facecolor='blue',
                       alpha=0.25)
```

```
<matplotlib.collections.PolyCollection at 0x7fbddb18a7f0>
```

Let's try working with dates!

```
plt.figure()

observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')

plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')
```

```
<IPython.core.display.Javascript object>
```

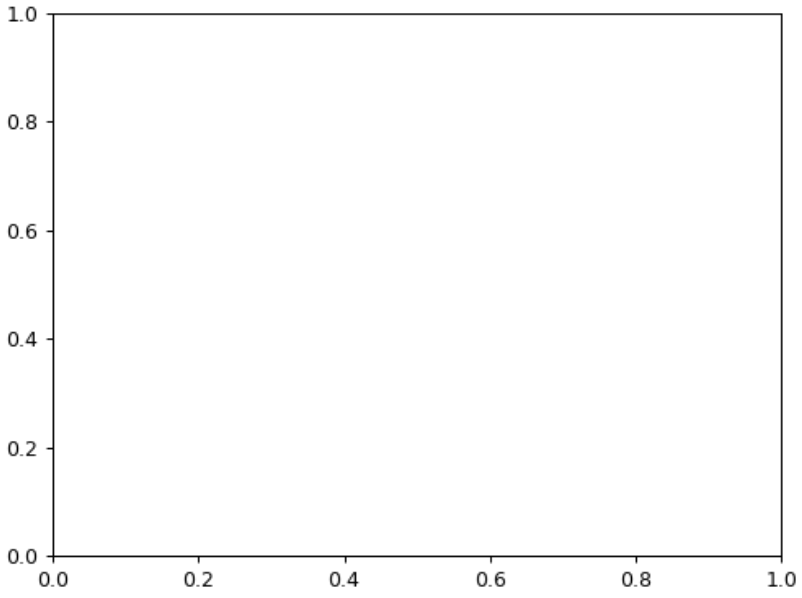



```
[<matplotlib.lines.Line2D at 0x7fbddb0f9d30>,
 <matplotlib.lines.Line2D at 0x7fbddb0f9eb8>]
```

Let's try using pandas

```
import pandas as pd

plt.figure()
observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')
observation_dates = map(pd.to_datetime, observation_dates) # trying to plot a map will result in an error
plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')
```

```
<IPython.core.display.Javascript object>
```




```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
```

```
/opt/conda/lib/python3.6/site-packages/matplotlib/units.py in get_converter(self, x)
    144                 # get_converter
--> 145                 if not np.all(xravel.mask):
    146                     # some elements are not masked


AttributeError: 'numpy.ndarray' object has no attribute 'mask'


During handling of the above exception, another exception occurred:


TypeError                                 Traceback (most recent call last)

<ipython-input-28-31d150774667> in <module>()
      4 observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')
      5 observation_dates = map(pd.to_datetime, observation_dates) # trying to plot a map will result in an error
----> 6 plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')


/opt/conda/lib/python3.6/site-packages/matplotlib/pyplot.py in plot(*args, **kwargs)
   3316                     mplDeprecation)
   3317     try:
-> 3318         ret = ax.plot(*args, **kwargs)
   3319     finally:
   3320         ax._hold = washold


/opt/conda/lib/python3.6/site-packages/matplotlib/__init__.py in inner(ax, *args, **kwargs)
   1890                     warnings.warn(msg % (label_namer, func.__name__),
   1891                                   RuntimeWarning, stacklevel=2)
-> 1892             return func(ax, *args, **kwargs)
   1893         pre_doc = inner.__doc__
   1894         if pre_doc is None:


/opt/conda/lib/python3.6/site-packages/matplotlib/axes/_axes.py in plot(self, *args, **kwargs)
   1404         kwargs = cbook.normalize_kwargs(kwargs, _alias_map)
   1405
-> 1406         for line in self._get_lines(*args, **kwargs):
   1407             self.add_line(line)
   1408             lines.append(line)


/opt/conda/lib/python3.6/site-packages/matplotlib/axes/_base.py in _grab_next_args(self, *args, **kwargs)
    414                 isplit = 2
    415
--> 416             for seg in self._plot_args(remaining[:isplit], kwargs):
    417                 yield seg
    418             remaining = remaining[isplit:]


/opt/conda/lib/python3.6/site-packages/matplotlib/axes/_base.py in _plot_args(self, tup, kwargs)
    383             x, y = index_of(tup[-1])
    384
--> 385         x, y = self._xy_from_xy(x, y)
    386
    387         if self.command == 'plot':


/opt/conda/lib/python3.6/site-packages/matplotlib/axes/_base.py in _xy_from_xy(self, x, y)
    215     def _xy_from_xy(self, x, y):
    216         if self.axes.xaxis is not None and self.axes.yaxis is not None:
--> 217             bx = self.axes.xaxis.update_units(x)
    218             by = self.axes.yaxis.update_units(y)
    219


/opt/conda/lib/python3.6/site-packages/matplotlib/axis.py in update_units(self, data)
   1411         """
   1412
-> 1413         converter = munits.registry.get_converter(data)
   1414         if converter is None:
   1415             return False
```

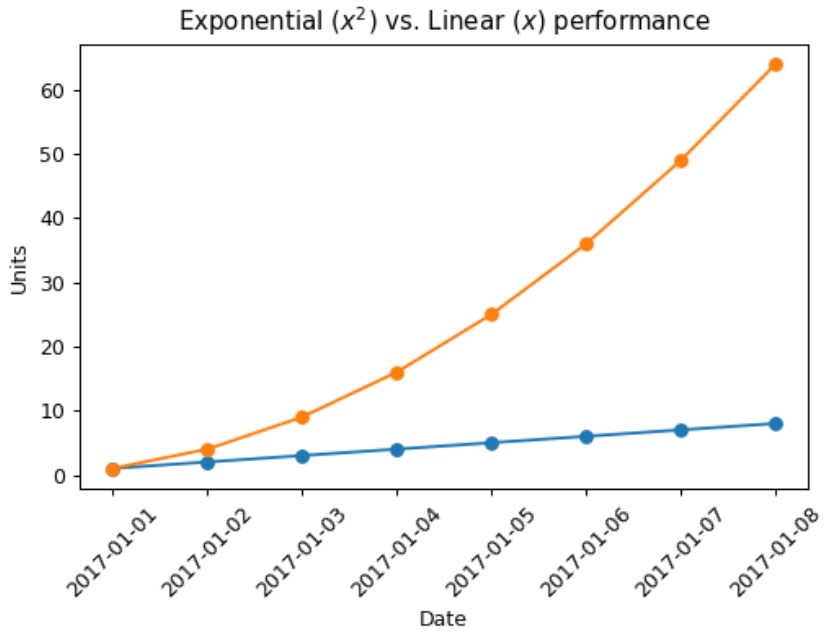
```
/opt/conda/lib/python3.6/site-packages/matplotlib/units.py in get_converter(self, x)
    156                 if (not isinstance(next_item, np.ndarray) or
    157                     next_item.shape != x.shape):
--> 158                     converter = self.get_converter(next_item)
    159                 return converter
    160
```

```
/opt/conda/lib/python3.6/site-packages/matplotlib/units.py in get_converter(self, x)
    159                 return converter
    160
--> 161         if converter is None and iterable(x) and (len(x) > 0):
    162             thisx = safe_first_element(x)
    163             if classx and classx != getattr(thisx, '__class__', None):
```

```
TypeError: object of type 'map' has no len()
```

```
plt.figure()
observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[D]')
observation_dates = list(map(pd.to_datetime, observation_dates)) # convert the map to a list to get rid of the error
plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data, '-o')
```

```
<IPython.core.display.Javascript object>
```




```
[<matplotlib.lines.Line2D at 0x7fbe1c5df438>,
 <matplotlib.lines.Line2D at 0x7fbe1c6123c8>]
```

```
x = plt.gca().xaxis

# rotate the tick labels for the x axis
for item in x.get_ticklabels():
    item.set_rotation(45)
```

```
# adjust the subplot so the text doesn't run off the image
plt.subplots_adjust(bottom=0.25)
```

```
ax = plt.gca()
ax.set_xlabel('Date')
ax.set_ylabel('Units')
ax.set_title('Exponential vs. Linear performance')
```

```
<matplotlib.text.Text at 0x7fbe1c5e62b0>
```

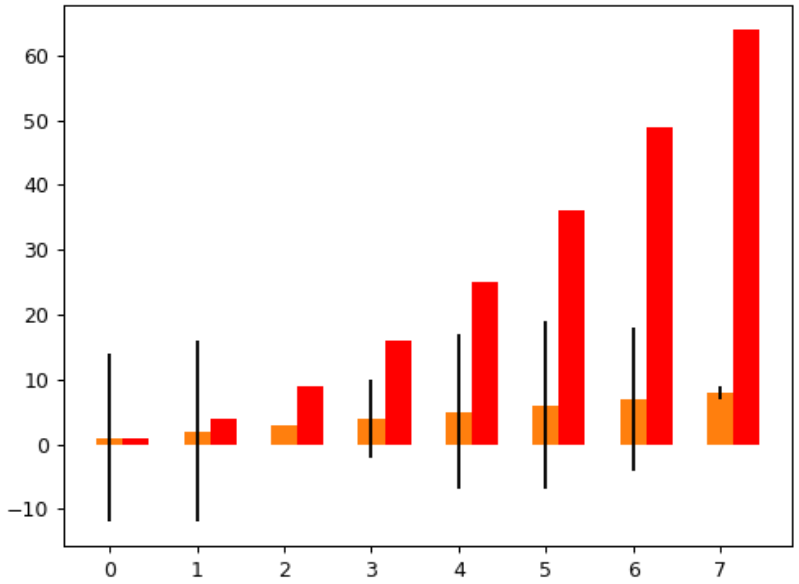
```
# you can add mathematical expressions in any text element
ax.set_title("Exponential ($x^2$) vs. Linear ($x$) performance")
```

```
<matplotlib.text.Text at 0x7fbe1c5e62b0>
```

Bar Charts

```
plt.figure()
xvals = range(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3)
```

```
<IPython.core.display.Javascript object>
```




```
<Container object of 8 artists>
```

```
new_xvals = []

# plot another set of bars, adjusting the new xvals to make up for the first set of bars plotted
for item in xvals:
    new_xvals.append(item+0.3)

plt.bar(new_xvals, exponential_data, width = 0.3 ,color='red')
```

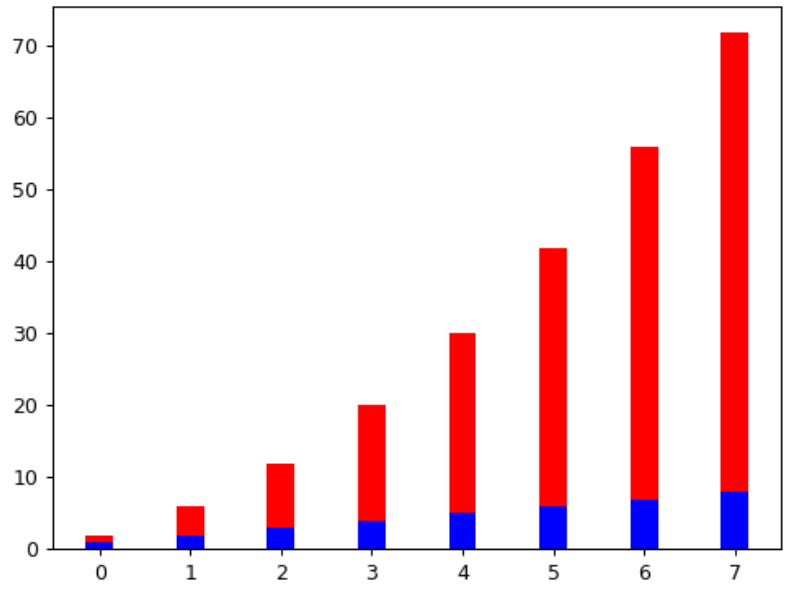
```
<Container object of 8 artists>
```

```
from random import randint
linear_err = [randint(0,15) for x in range(len(linear_data))]

# This will plot a new set of bars with errorbars using the list of random error values
plt.bar(xvals, linear_data, width = 0.3, yerr=linear_err)
```

```
<Container object of 8 artists>
```

```
# stacked bar charts are also possible
plt.figure()
xvals = range(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3, color='b')
plt.bar(xvals, exponential_data, width = 0.3, bottom=linear_data, color='r')
```

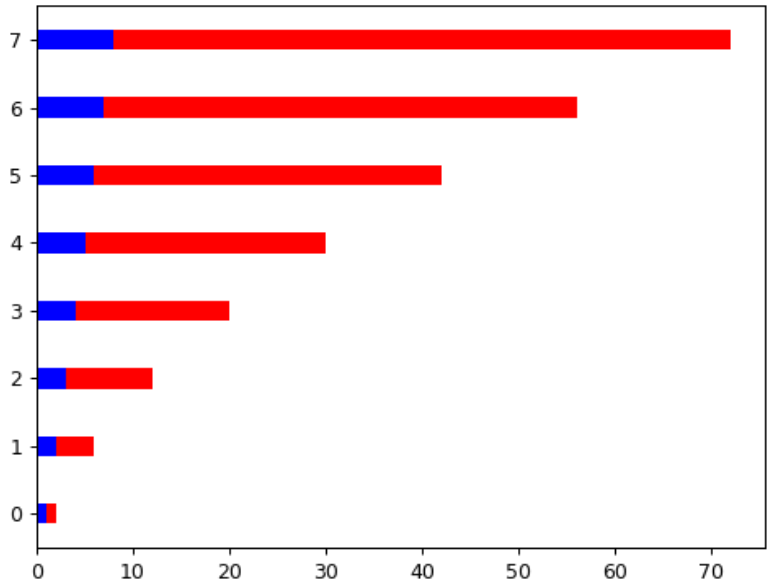
```
<IPython.core.display.Javascript object>
```




```
<Container object of 8 artists>
```

```
# or use barh for horizontal bar charts
plt.figure()
xvals = range(len(linear_data))
plt.barh(xvals, linear_data, height = 0.3, color='b')
plt.barh(xvals, exponential_data, height = 0.3, left=linear_data, color='r')
```

```
<IPython.core.display.Javascript object>
```




```
<Container object of 8 artists>
```