

Week2

May 28, 2019

1 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.

```
In [1]: %matplotlib notebook
```

```
In [2]: import matplotlib as mpl
        mpl.get_backend()
```

```
Out[2]: 'nbAgg'
```

```
In [3]: import matplotlib.pyplot as plt
        plt.plot?
```

```
In [4]: # 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>
```

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

```
Out[4]: [<matplotlib.lines.Line2D at 0x7f6d1b053128>]
```

```
In [5]: # 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, '.')
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x7f6d3c3ff160>]
```

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

```
In [6]: # First let's set the backend without using mpl.use() from the scripting la
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.

```
In [7]: %%html
<img src='test.png' />

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

```
In [8]: # 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>
```

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

```
Out[8]: [0, 6, 0, 10]
```

```
In [9]: # 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>

<IPython.core.display.HTML object>

```
Out[9]: [<matplotlib.lines.Line2D at 0x7f6d005b1f98>]
```

```
In [10]: # get current axes
ax = plt.gca()
# get all the child objects the axes contains
ax.get_children()
```

```
Out[10]: [<matplotlib.lines.Line2D at 0x7f6d005b1630>,
<matplotlib.lines.Line2D at 0x7f6d1b092d68>,
<matplotlib.lines.Line2D at 0x7f6d005b1f98>,
<matplotlib.spines.Spine at 0x7f6d11886f28>,
<matplotlib.spines.Spine at 0x7f6d1b03ecc0>,
<matplotlib.spines.Spine at 0x7f6d1b00ba90>,
<matplotlib.spines.Spine at 0x7f6d18830978>,
<matplotlib.axis.XAxis at 0x7f6d18830d68>,
<matplotlib.axis.YAxis at 0x7f6d11891390>,
<matplotlib.text.Text at 0x7f6d11846080>,
<matplotlib.text.Text at 0x7f6d118460f0>,
<matplotlib.text.Text at 0x7f6d11846160>,
<matplotlib.patches.Rectangle at 0x7f6d11846198>]
```

2 Scatterplots

```
In [11]: 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 ch
```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

Out[11]: <matplotlib.collections.PathCollection at 0x7f6d0054d978>

In [12]: **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>

<IPython.core.display.HTML object>

Out[12]: <matplotlib.collections.PathCollection at 0x7f6d00473898>

In [13]: *# 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)
```

In [14]: *# 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)

In [15]: 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>

<IPython.core.display.HTML object>

Out[15]: <matplotlib.collections.PathCollection at 0x7f6d0041ce10>

In [16]: # 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')

Out[16]: <matplotlib.text.Text at 0x7f6d0042e780>

In [17]: # add a legend (uses the labels from plt.scatter)
plt.legend()

Out[17]: <matplotlib.legend.Legend at 0x7f6d1887c080>

In [18]: # add the legend to loc=4 (the lower right hand corner), also gets rid of the legend title
plt.legend(loc=4, frameon=False, title='Legend')

Out[18]: <matplotlib.legend.Legend at 0x7f6d00377e10>

In [19]: # get children from current axes (the legend is the second to last item in the list)
plt.gca().get_children()

```

```
Out[19]: [<matplotlib.collections.PathCollection at 0x7f6d0041c2b0>,
<matplotlib.collections.PathCollection at 0x7f6d0041ce10>,
<matplotlib.spines.Spine at 0x7f6d004521d0>,
<matplotlib.spines.Spine at 0x7f6d004523c8>,
<matplotlib.spines.Spine at 0x7f6d004525c0>,
<matplotlib.spines.Spine at 0x7f6d004527b8>,
<matplotlib.axis.XAxis at 0x7f6d00452978>,
<matplotlib.axis.YAxis at 0x7f6d0045a438>,
<matplotlib.text.Text at 0x7f6d0042e780>,
<matplotlib.text.Text at 0x7f6d0042e5c0>,
<matplotlib.text.Text at 0x7f6d0042e358>,
<matplotlib.legend.Legend at 0x7f6d00377e10>,
<matplotlib.patches.Rectangle at 0x7f6d0042e390>]
```

```
In [20]: # get the legend from the current axes
legend = plt.gca().get_children()[-2]
```

```
In [21]: # you can use get_children to navigate through the child artists
legend.get_children()[0].get_children()[1].get_children()[0].get_children()
```

```
Out[21]: [<matplotlib.offsetbox.HPacker at 0x7f6d0038e4a8>,
<matplotlib.offsetbox.HPacker at 0x7f6d0038e438>]
```

```
In [22]: # 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 u
rec_gc(plt.legend())
```

Legend

```
<matplotlib.offsetbox.VPacker object at 0x7f6d003aa400>
<matplotlib.offsetbox.TextArea object at 0x7f6d003aa198>
  Text(0,0,'None')
<matplotlib.offsetbox.HPacker object at 0x7f6d00377fd0>
  <matplotlib.offsetbox.VPacker object at 0x7f6d00377780>
    <matplotlib.offsetbox.HPacker object at 0x7f6d003aa0b8>
      <matplotlib.offsetbox.DrawingArea object at 0x7f6d0039fba8>
        <matplotlib.collections.PathCollection object at 0x7f6d0039fba8>
        <matplotlib.offsetbox.TextArea object at 0x7f6d00377828>
          Text(0,0,'Tall students')
      <matplotlib.offsetbox.HPacker object at 0x7f6d003aa160>
        <matplotlib.offsetbox.DrawingArea object at 0x7f6d0039fe80>
```

```
<matplotlib.collections.PathCollection object at 0x7f6d003a...>
<matplotlib.offsetbox.TextArea object at 0x7f6d0039fcc0>
Text(0,0,'Short students')
FancyBboxPatch(0,0;1x1)
```

3 Line Plots

```
In [23]: 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>
```

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

```
Out[23]: [<matplotlib.lines.Line2D at 0x7f6d00302fd0>,
<matplotlib.lines.Line2D at 0x7f6d0030a1d0>]
```

```
In [24]: # plot another series with a dashed red line
plt.plot([22,44,55], '--r')
```

```
Out[24]: [<matplotlib.lines.Line2D at 0x7f6d002b8f98>]
```

```
In [25]: 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
plt.legend(['Baseline', 'Competition', 'Us'])
```

```
Out[25]: <matplotlib.legend.Legend at 0x7f6d002d0860>
```

```
In [26]: # 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)
```

```
Out[26]: <matplotlib.collections.PolyCollection at 0x7f6d002705f8>
```

Let's try working with dates!

```

In [27]: plt.figure()

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

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

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

Out[27]: [<matplotlib.lines.Line2D at 0x7f6d00248e48>,
          <matplotlib.lines.Line2D at 0x7f6d00248fd0>]

```

Let's try using pandas

```

In [28]: import pandas as pd

        plt.figure()
        observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[ns]')
        observation_dates = map(pd.to_datetime, observation_dates) # trying to plot with pandas
        plt.plot(observation_dates, linear_data, '-o', observation_dates, exponential_data, '-o')

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```

```

AttributeError                                Traceback (most recent call last)

/opt/conda/lib/python3.6/site-packages/matplotlib/units.py in get_converter(unit)
    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[ns]', freq='D')
      5 observation_dates = map(pd.to_datetime, observation_dates) # trying to
----> 6 plt.plot(observation_dates, linear_data, '-o', observation_dates, exponential_data)

/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/axes/_base.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/_base.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_argument(self, remaining, 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, remaining, 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(s
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
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
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()

```

In [29]: plt.figure()
        observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[ns]', freq='D')
        observation_dates = list(map(pd.to_datetime, observation_dates)) # convert to datetime
        plt.plot(observation_dates, linear_data, '-o', observation_dates, exponen

```

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

```

Out[29]: [<matplotlib.lines.Line2D at 0x7f6cf4a61be0>,
         <matplotlib.lines.Line2D at 0x7f6cf4a1a438>]

```

```

In [30]: x = plt.gca().xaxis

```

```

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

In [31]: # adjust the subplot so the text doesn't run off the image
plt.subplots_adjust(bottom=0.25)

In [32]: ax = plt.gca()
ax.set_xlabel('Date')
ax.set_ylabel('Units')
ax.set_title('Exponential vs. Linear performance')

Out[32]: <matplotlib.text.Text at 0x7f6cf4a68a58>

In [33]: # you can add mathematical expressions in any text element
ax.set_title("Exponential ( $x^2$ ) vs. Linear ( $x$ ) performance")

Out[33]: <matplotlib.text.Text at 0x7f6cf4a68a58>

```

4 Bar Charts

```

In [34]: plt.figure()
xvals = range(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3)

<IPython.core.display.Javascript object>

<IPython.core.display.HTML object>

Out[34]: <Container object of 8 artists>

In [35]: new_xvals = []

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

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

Out[35]: <Container object of 8 artists>

In [ ]: 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
plt.bar(xvals, linear_data, width = 0.3, yerr=linear_err)

```

```

In [36]: # 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>

<IPython.core.display.HTML object>

Out[36]: <Container object of 8 artists>

In [37]: # 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>

<IPython.core.display.HTML object>

Out[37]: <Container object of 8 artists>

In [ ]:

```