

Week2

May 16, 2020

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 [ ]: %matplotlib notebook
```

```
In [1]: import matplotlib as mpl
        mpl.get_backend()
```

```
Out[1]: 'module://ipykernel.pylab.backend_inline'
```

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

```
In [5]: # because the default is the line style '-',
        # nothing will be shown if we only pass in one point (3,2)
        plt.plot(3, 2)
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x7efd2c7c3c88>]
```

```
In [6]: # 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[6]: [<matplotlib.lines.Line2D at 0x7efd2c409eb8>]
```

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

```
In [8]: # 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.

```

In [9]: %%html
        <img src='test.png' />

<IPython.core.display.HTML object>

```

```

In [10]: # 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])

```

```

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

```

```

In [11]: # 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')

```

```

Out[11]: [<matplotlib.lines.Line2D at 0x7efd2787f978>]

```

```

In [12]: # get current axes
ax = plt.gca()
# get all the child objects the axes contains
ax.get_children()

```

```
Out[12]: [<matplotlib.lines.Line2D at 0x7efd2787f7f0>,
<matplotlib.lines.Line2D at 0x7efd278a6b38>,
<matplotlib.lines.Line2D at 0x7efd2787f978>,
<matplotlib.spines.Spine at 0x7efd278af550>,
<matplotlib.spines.Spine at 0x7efd278af748>,
<matplotlib.spines.Spine at 0x7efd278af940>,
<matplotlib.spines.Spine at 0x7efd278afb38>,
<matplotlib.axis.XAxis at 0x7efd278afcf8>,
<matplotlib.axis.YAxis at 0x7efd278bf470>,
<matplotlib.text.Text at 0x7efd278d6860>,
<matplotlib.text.Text at 0x7efd278d6588>,
<matplotlib.text.Text at 0x7efd278d66a0>,
<matplotlib.patches.Rectangle at 0x7efd278d6048>]
```

2 Scatterplots

```
In [13]: 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
```

```
Out[13]: <matplotlib.collections.PathCollection at 0x7efd250f3518>
```

```
In [14]: 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)
```

```
Out[14]: <matplotlib.collections.PathCollection at 0x7efd25067ba8>
```

```
In [15]: # 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 [16]: # 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 [17]: plt.figure()
# plot a data series 'Tall students' in red using the first two elements of each tuple
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 each tuple
plt.scatter(x[2:], y[2:], s=100, c='blue', label='Short students')

Out[17]: <matplotlib.collections.PathCollection at 0x7efd2509a898>

In [18]: # 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[18]: <matplotlib.text.Text at 0x7efd228b0b00>

```

```

In [19]: # add a legend (uses the labels from plt.scatter)
plt.legend()

Out[19]: <matplotlib.legend.Legend at 0x7efd2c41d518>

In [20]: # add the legend to loc=4 (the lower right hand corner), also gets rid of
plt.legend(loc=4, frameon=False, title='Legend')

Out[20]: <matplotlib.legend.Legend at 0x7efd228e2a20>

In [21]: # get children from current axes (the legend is the second to last item in
plt.gca().get_children()

Out[21]: [<matplotlib.collections.PathCollection at 0x7efd228f8b70>,
<matplotlib.collections.PathCollection at 0x7efd2509a898>,
<matplotlib.spines.Spine at 0x7efd228f17f0>,
<matplotlib.spines.Spine at 0x7efd228f19e8>,
<matplotlib.spines.Spine at 0x7efd228f1be0>,
<matplotlib.spines.Spine at 0x7efd228f1dd8>,
<matplotlib.axis.XAxis at 0x7efd228f1f98>,
<matplotlib.axis.YAxis at 0x7efd22907668>,
<matplotlib.text.Text at 0x7efd228b0b00>,
<matplotlib.text.Text at 0x7efd228b0b70>,
<matplotlib.text.Text at 0x7efd228b0be0>,
<matplotlib.legend.Legend at 0x7efd228e2a20>,
<matplotlib.patches.Rectangle at 0x7efd228b0c18>]

In [22]: # get the legend from the current axes
legend = plt.gca().get_children()[-2]

In [23]: # you can use get_children to navigate through the child artists
legend.get_children()[0].get_children()[1].get_children()[0].get_children()

Out[23]: [<matplotlib.offsetbox.HPacker at 0x7efd22869780>,
<matplotlib.offsetbox.HPacker at 0x7efd22869860>]

In [24]: # 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 0x7efd2287e400>
  <matplotlib.offsetbox.TextArea object at 0x7efd2287e320>
    Text(0,0,'None')
  <matplotlib.offsetbox.HPacker object at 0x7efd22877780>
    <matplotlib.offsetbox.VPacker object at 0x7efd228777b8>
      <matplotlib.offsetbox.HPacker object at 0x7efd2287e278>
        <matplotlib.offsetbox.DrawingArea object at 0x7efd22877b00>
          <matplotlib.collections.PathCollection object at 0x7efd228777f0>
            <matplotlib.offsetbox.TextArea object at 0x7efd228777f0>
              Text(0,0,'Tall students')
            <matplotlib.offsetbox.HPacker object at 0x7efd2287e2e8>
              <matplotlib.offsetbox.DrawingArea object at 0x7efd2287e080>
                <matplotlib.collections.PathCollection object at 0x7efd228777d8>
                  <matplotlib.offsetbox.TextArea object at 0x7efd22877d68>
                    Text(0,0,'Short students')
  FancyBboxPatch(0,0;1x1)
```

3 Line Plots

In [25]: `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')
```

Out[25]: [`<matplotlib.lines.Line2D at 0x7efd1457a198>`,
`<matplotlib.lines.Line2D at 0x7efd1457a320>`]

In [26]: `# plot another series with a dashed red line`
`plt.plot([22,44,55], '--r')`

Out[26]: [`<matplotlib.lines.Line2D at 0x7efd160b4908>`]

```
In [27]: 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[27]: `<matplotlib.legend.Legend at 0x7efd14581978>`

In [28]: `# 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 [28]: <matplotlib.collections.PolyCollection at 0x7efd145817f0>

Let's try working with dates!

In [29]: 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')
```

Out [29]: [<matplotlib.lines.Line2D at 0x7efd14502dd8>,
<matplotlib.lines.Line2D at 0x7efd14502f60>]

Let's try using pandas

In [30]: 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')
```

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-30-31d150774667> in <module>()
      4 observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64[ns]')
      5 observation_dates = map(pd.to_datetime, observation_dates) # trying to plot with pandas
----> 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,
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,
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,
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_ne
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_ar
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
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 [ ]: 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, exponential_data, '-o')

```

```

In [31]: x = plt.gca().xaxis

```

```

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

```

```

In [32]: # adjust the subplot so the text doesn't run off the image
        plt.subplots_adjust(bottom=0.25)

```

```

In [33]: ax = plt.gca()
        ax.set_xlabel('Date')
        ax.set_ylabel('Units')
        ax.set_title('Exponential vs. Linear performance')

```

```

Out[33]: <matplotlib.text.Text at 0x7efd098bc978>

```

```
In [34]: # you can add mathematical expressions in any text element
ax.set_title("Exponential ( $x^2$ ) vs. Linear ( $x$ ) performance")
```

```
Out[34]: <matplotlib.text.Text at 0x7efd098bc978>
```

4 Bar Charts

```
In [35]: plt.figure()
xvals = range(len(linear_data))
plt.bar(xvals, linear_data, width = 0.3)
```

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

```
In [36]: 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[36]: <Container object of 8 artists>
```

```
In [37]: 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)
```

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

```
In [38]: # 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')
```

```
Out[38]: <Container object of 8 artists>
```

```
In [39]: # 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')
```

```
Out[39]: <Container object of 8 artists>
```

```
In [ ]:
```