Week2

May 13, 2018

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 0x24da8b39b70>]
```

```
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 0x24da8c41470>]
```

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 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 [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 0x24da9e2ae10>]
```

```
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 0x24da9e2abe0>,
          <matplotlib.lines.Line2D at 0x24da8c61ac8>,
          <matplotlib.lines.Line2D at 0x24da9e2ae10>,
          <matplotlib.spines.Spine at 0x24da9df2a90>,
          <matplotlib.spines.Spine at 0x24da9df2dd8>,
          <matplotlib.spines.Spine at 0x24da9df2da0>,
          <matplotlib.spines.Spine at 0x24da8c65f98>,
          <matplotlib.axis.XAxis at 0x24da9dfc160>,
          <matplotlib.axis.YAxis at 0x24da9709978>,
          Text(0.5,1,''),
          Text(0,1,''),
          Text(1,1,''),
          <matplotlib.patches.Rectangle at 0x24da9e25cf8>]
```

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 child objects
```

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

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


Out[11]: <matplotlib.collections.PathCollection at 0x24daa528748>

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 0x24daac0cda0>

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),
         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
         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 0x24dab30e160>
```

```
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 0x7fe5180afc88>
```

```
In [16]: # add a legend (uses the labels from plt.scatter)
         plt.legend()
```

```
Out[16]: <matplotlib.legend.Legend at 0x24dab9bdb70>
```

```
In [17]: # add the legend to loc=4 (the lower right hand corner), also gets rid of the frame a
         plt.legend(loc=4, frameon=False, title='Legend')
```

```
Out[17]: <matplotlib.legend.Legend at 0x24dab9cd2e8>
```

```
In [18]: # get children from current axes (the legend is the second to last item in this list)
         plt.gca().get_children()
```

```
Out[18]: [<matplotlib.collections.PathCollection at 0x24dab301da0>,
          <matplotlib.collections.PathCollection at 0x24dab30e160>,
          <matplotlib.spines.Spine at 0x24dab2d4240>,
          <matplotlib.spines.Spine at 0x24dab2d4358>,
          <matplotlib.spines.Spine at 0x24dab2d4470>,
          <matplotlib.spines.Spine at 0x24dab2d4588>,
          <matplotlib.axis.XAxis at 0x24dab2d4668>,
          <matplotlib.axis.YAxis at 0x24dab2dbe48>,
          Text(0.5,1,''),
          Text(0,1,''),
          Text(1,1,''),
          <matplotlib.legend.Legend at 0x24dab9cd2e8>,
          <matplotlib.patches.Rectangle at 0x24dab2d4a58>]
```

```
In [23]: # get the legend from the current axes
         legend = plt.gca().get_children()[-2]
```

```
In [20]: # you can use get_children to navigate through the child artists
         legend.get_children()[0].get_children()[1].get_children()[0].get_children()
```

```
Out[20]: [<matplotlib.offsetbox.HPacker at 0x24dab9cdeb8>,
          <matplotlib.offsetbox.HPacker at 0x24dab9cdef0>]
```

```
In [21]: # 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 0x0000024DAB9CD4A8>
        <matplotlib.offsetbox.TextArea object at 0x0000024DAB9DD2E8>
            Text(0,0,'None')
        <matplotlib.offsetbox.HPacker object at 0x0000024DAB9DD358>
            <matplotlib.offsetbox.VPacker object at 0x0000024DAB9DD4E0>
                <matplotlib.offsetbox.HPacker object at 0x0000024DAB9DD208>
                    <matplotlib.offsetbox.DrawingArea object at 0x0000024DAB9E8550>
                        <matplotlib.collections.PathCollection object at 0x0000024DAB9E8630>
                    <matplotlib.offsetbox.TextArea object at 0x0000024DAB9E8390>
                        Text(0,0,'Tall students')
                <matplotlib.offsetbox.HPacker object at 0x0000024DAB9DD3C8>
                    <matplotlib.offsetbox.DrawingArea object at 0x0000024DAB9E87B8>
```

```
                        <matplotlib.collections.PathCollection object at 0x0000024DAB9E8898>
                    <matplotlib.offsetbox.TextArea object at 0x0000024DAB9E8668>
                        Text(0,0,'Short students')
    FancyBboxPatch(0,0;1x1)
```

3 Line Plots

```
In [24]: 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[24]: [<matplotlib.lines.Line2D at 0x24daba14550>,
          <matplotlib.lines.Line2D at 0x24daba14a20>]
```

```
In [25]: # plot another series with a dashed red line
         plt.plot([22,44,55], '--r')
```

```
Out[25]: [<matplotlib.lines.Line2D at 0x24dab322160>]
```

```
In [26]: 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
         plt.legend(['Baseline', 'Competition', 'Us'])
```

```
Out[26]: <matplotlib.legend.Legend at 0x24dac0c5e48>
```

```
In [27]: # 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[27]: <matplotlib.collections.PolyCollection at 0x24dac0dd710>
```

Let's try working with dates!

```
In [28]: 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,
```

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


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

```
Out[28]: [<matplotlib.lines.Line2D at 0x24dac126b00>,
          <matplotlib.lines.Line2D at 0x24dac126cf8>]
```

Let's try using pandas

```
In [29]: 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 wil
         plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data,
```

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


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

```
        ---------------------------------------------------------------------------

        AttributeError                            Traceback (most recent call last)

        C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\units.py
        143                 # get_converter
    --> 144                 if not np.all(xravel.mask):
        145                     # some elements are not masked


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


    During handling of the above exception, another exception occurred:


        RuntimeError                              Traceback (most recent call last)
    <ipython-input-29-d8577b79c140> 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 w
----> 6 plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\pyplot.py
   3259                           mplDeprecation)
   3260        try:
-> 3261            ret = ax.plot(*args, **kwargs)
   3262        finally:
   3263            ax._hold = washold


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\__init__
   1715                         warnings.warn(msg % (label_namer, func.__name__),
   1716                                       RuntimeWarning, stacklevel=2)
-> 1717                return func(ax, *args, **kwargs)
   1718            pre_doc = inner.__doc__
   1719            if pre_doc is None:


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\axes\_axe
   1370            kwargs = cbook.normalize_kwargs(kwargs, _alias_map)
   1371
-> 1372            for line in self._get_lines(*args, **kwargs):
   1373                self.add_line(line)
   1374                lines.append(line)


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\axes\_bas
    402                    this += args[0],
    403                    args = args[1:]
--> 404                for seg in self._plot_args(this, kwargs):
    405                    yield seg
    406


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\axes\_bas
    382                x, y = index_of(tup[-1])
    383
--> 384            x, y = self._xy_from_xy(x, y)
    385
    386            if self.command == 'plot':


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\axes\_bas
    214        def _xy_from_xy(self, x, y):
```

```
    215         if self.axes.xaxis is not None and self.axes.yaxis is not None:
--> 216             bx = self.axes.xaxis.update_units(x)
    217             by = self.axes.yaxis.update_units(y)
    218


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\axis.py i
   1430         """
   1431
-> 1432         converter = munits.registry.get_converter(data)
   1433         if converter is None:
   1434             return False


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\units.py
    155                 if (not isinstance(next_item, np.ndarray) or
    156                     next_item.shape != x.shape):
--> 157                     converter = self.get_converter(next_item)
    158                 return converter
    159


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\units.py
    160         if converter is None:
    161             try:
--> 162                 thisx = safe_first_element(x)
    163             except (TypeError, StopIteration):
    164                 pass


    C:\Users\apday\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib\cbook\__i
   2309         except TypeError:
   2310             pass
-> 2311         raise RuntimeError("matplotlib does not support generators "
   2312                            "as input")
   2313     return next(iter(obj))


    RuntimeError: matplotlib does not support generators as input
```

```
In [30]: 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
         plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponential_data,
```

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

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


Out[30]: [<matplotlib.lines.Line2D at 0x24dae67ba20>,
          <matplotlib.lines.Line2D at 0x24dae691208>]
```

```
In [32]: 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]: Text(0.5,1,'Exponential vs. Linear performance')
```

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

Out[33]: Text(0.5,1,'Exponential ($x^2$) vs. Linear ($x$) performance')
```

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 [37]: new_xvals = []

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

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

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

```
In [35]: 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 valu
         plt.bar(xvals, linear_data, width = 0.3, yerr=linear_err)

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

```
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 [40]: # 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[40]: <Container object of 8 artists>
```

4.1 Dejunkifying a Plot

```
In [90]: import matplotlib.pyplot as plt
         import numpy as np

         plt.figure()

         languages =['Python', 'SQL', 'Java', 'C++', 'JavaScript']
         pos = np.arange(len(languages))
         popularity = [56, 39, 34, 34, 29]

         bars = plt.bar(pos, popularity, align='center', alpha=0.7)
         plt.xticks(pos, languages)
         plt.ylabel('% Popularity')
         plt.title('Top 5 Languages for Math & Data \nby % popularity on Stack Overflow', alpha
<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Out[90]: Text(0.5,1,'Top 5 Languages for Math & Data \nby % popularity on Stack Overflow')

In [93]: #TODO: remove all the ticks (both axes), and tick labels on the Y axis

         plt.tick_params(
             axis='both',          # changes apply to the x-axis
             which='both',      # both major and minor ticks are affected
             bottom='off',      # ticks along the bottom edge are off
             top='off',         # ticks along the top edge are off
             right='off',      # ticks along the bottom edge are off
             left='off',         # ticks along the top edge are off
             labelleft='off')

         #sample answer
         #plt.tick_params(top='off', bottom='off', left='off', right='off', labelleft='off', l

In [91]: # TODO: remove the frame of the chart
         # https://matplotlib.org/api/spines_api.html

         for spine in plt.gca().spines.values():
             spine.set_visible(False)

In [92]: '''Task: Change the bar colors to be less bright blue, make one bar,
         the python bar, a contrasting color, soften all labels by turning grey.'''

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

Out[92]: [<matplotlib.patches.Rectangle at 0x24db3373b00>,
          <matplotlib.patches.Rectangle at 0x24db3373908>,
          <matplotlib.patches.Rectangle at 0x24db337c160>,
          <matplotlib.patches.Rectangle at 0x24db337c4e0>,
          <matplotlib.patches.Rectangle at 0x24db337c860>,
          <matplotlib.spines.Spine at 0x24db333b860>,
          <matplotlib.spines.Spine at 0x24db333be80>,
          <matplotlib.spines.Spine at 0x24db333bf98>,
          <matplotlib.spines.Spine at 0x24db33460f0>,
          <matplotlib.axis.XAxis at 0x24db33461d0>,
          <matplotlib.axis.YAxis at 0x24db334a940>,
          Text(0.5,1,'Top 5 Languages for Math & Data \nby % popularity on Stack Overflow'),
          Text(0,1,''),
          Text(1,1,''),
          <matplotlib.patches.Rectangle at 0x24db3360ac8>]
```

```
In [94]: ax.get_children()[0].set_color('r')
         ax.get_children()[1].set_color('xkcd:sky blue')
         ax.get_children()[2].set_color('xkcd:sky blue')
         ax.get_children()[3].set_color('xkcd:sky blue')
         ax.get_children()[4].set_color('xkcd:sky blue')
```

```
In [95]: # TODO: direct label each bar with Y axis values
         for bar in bars:
             plt.gca().text(bar.get_x() + bar.get_width()/2, bar.get_height() - 5, str(int(bar
                          ha='center', color='w', fontsize=11)
```