Week2

May 19, 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.

- The architecture to accomplish this is logically separated into three layers, which can be viewed as a stack. Each layer that sits above another layer knows how to talk to the layer below it, but the lower layer is not aware of the layers above it. The three layers from bottom to top are: backend, artist, and scripting.

Figure: The whole figure. The figure keeps track of all the child Axes, a smattering of 'special' artists (titles, figure legends, etc), and the canvas. (Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you). A figure can contain any number of Axes, but will typically have at least one.`It's convenient to create the axes together with the figure, but you can also add axes later on, allowing for more complex axes layouts.` fig = plt.figure() # an empty figure with no Axes fig, ax = plt.subplots() # a figure with a single Axes fig, axs = plt.subplots(2, 2) # a figure with a 2x2 grid of Axes

Axes: This is what you think of as 'a plot', it is the region of the image with the data space. A given figure can contain many Axes, but a given Axes object can only be in one Figure. The Axes contains two (or three in the case of 3D) Axis objects (be aware of the difference between Axes and Axis) which take care of the data limits (the data limits can also be controlled via the axes.Axes.set_xlim() and axes.Axes.set_ylim() methods). Each Axes has a title (set via set_title()), an x-label (set via set_xlabel()), and a y-label set via set_ylabel()).

The Axes class and its member functions are the primary entry point to working with the OO interface.

Axis These are the number-line-like objects. They take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). The location of the ticks is determined by a Locator object and the ticklabel strings are formatted by a Formatter. The combination of the correct Locator and Formatter gives very fine control over the tick locations and labels.

Artist Basically everything you can see on the figure is an artist (even the Figure, Axes, and Axis objects). This includes Text objects, Line2D objects, collections objects, Patch objects ... (you get the idea). When the figure is rendered, all of the artists are drawn to the canvas. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved from one to another.

```
In [2]: %matplotlib notebook
```

```
In [3]: import matplotlib as mpl
        mpl.get_backend()
```

```
Out[3]: 'nbAgg'
```

1.0.1 The object-oriented interface and the pyplot interface

As noted above, there are essentially two ways to use Matplotlib:

- Explicitly create figures and axes, and call methods on them (the "object-oriented (OO) style").
- Rely on pyplot to automatically create and manage the figures and axes, and use pyplot functions for plotting.

```
In [4]: import matplotlib.pyplot as plt                #matplotlib.pyplot is a state-
        plt.plot?
```

So one can do (OO-style)

```
In [5]: import numpy as np
        x = np.linspace(0, 2, 100)

        # Note that even in the OO-style, we use `.pyplot.figure` to create the fig
        fig, ax = plt.subplots()  # Create a figure and an axes.
        ax.plot(x, x, label='linear')  # Plot some data on the axes.
        ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
        ax.plot(x, x**3, label='cubic')  # ... and some more.
        ax.set_xlabel('x label')  # Add an x-label to the axes.
        ax.set_ylabel('y label')  # Add a y-label to the axes.
        ax.set_title("Simple Plot")  # Add a title to the axes.
        ax.legend()  # Add a legend.
```

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

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

```
Out[5]: <matplotlib.legend.Legend at 0x7f18485ef588>
```

or (pyplot-style)

```
In [6]: x = np.linspace(0, 2, 100)

        plt.plot(x, x, label='linear')  # Plot some data on the (implicit) axes.
        plt.plot(x, x**2, label='quadratic')  # etc.
        plt.plot(x, x**3, label='cubic')
        plt.xlabel('x label')
        plt.ylabel('y label')
        plt.title("Simple Plot")
        plt.legend()
```

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


<IPython.core.display.HTML object>


Out[6]: <matplotlib.legend.Legend at 0x7f1831358e10>

In [7]: # 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[7]: [<matplotlib.lines.Line2D at 0x7f18312a99e8>]

In [8]: # 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, '.')

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Out[8]: [<matplotlib.lines.Line2D at 0x7f182eaba0f0>]
```

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

1.0.2 Backends

A lot of documentation on the website and in the mailing lists refers to the "backend" and many new users are confused by this term. Matplotlib targets many different use cases and output formats. To support all of these use cases, matplotlib can target different outputs, and each of these capabilities is called a backend; the "frontend" is the user facing code, i.e., the plotting code, whereas the "backend" does all the hard work behind-the-scenes to make the figure. There are two types of backends: user interface backends (for use in pygtk, wxpython, tkinter, qt4, or macosx; also referred to as "interactive backends") and hardcopy backends to make image files (PNG, SVG, PDF, PS; also referred to as "non-interactive backends").

Selecting a backend:- There are three ways to configure your backend: - The rcParams["backend"] (default: 'agg') parameter in your matplotlibrc file - The MPLBACKEND environment variable - The function matplotlib.use()

```
In [9]: # 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 [10]: %%html
         <img src='test.png' />
```

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

```
In [11]: # 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[11]: [0, 6, 0, 10]
```

```
In [12]: # 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[12]: [<matplotlib.lines.Line2D at 0x7f181c83db38>]
```

```
In [13]: # get current axes
         ax = plt.gca()
         # get all the child objects the axes contains
         ax.get_children()
```

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


<IPython.core.display.HTML object>


Out[13]: [<matplotlib.spines.Spine at 0x7f181c7cf5c0>,
          <matplotlib.spines.Spine at 0x7f181c882828>,
          <matplotlib.spines.Spine at 0x7f181c882400>,
          <matplotlib.spines.Spine at 0x7f181c87db70>,
          <matplotlib.axis.XAxis at 0x7f181c87d630>,
          <matplotlib.axis.YAxis at 0x7f181c7f1828>,
          <matplotlib.text.Text at 0x7f181c7a3e10>,
          <matplotlib.text.Text at 0x7f181c7a3e80>,
          <matplotlib.text.Text at 0x7f181c7a3ef0>,
          <matplotlib.patches.Rectangle at 0x7f181c7a3f28>]
```

2 Scatterplots

```
In [14]: 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[14]: <matplotlib.collections.PathCollection at 0x7f181c6dfd30>

In [18]: 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[18]: <matplotlib.collections.PathCollection at 0x7f181c5439b0>

In [25]: # 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 [26]:
```
# use zip to convert 5 tuples with 2 elements each to 2 tuples with 5 elem
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,
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 [34]:
```
plt.figure()
# plot a data series 'Tall students' in red using the first two elements o
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
plt.scatter(x[2:], y[2:], s=100, c='blue', label='Short students')
```

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

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

Out[34]:
```
<matplotlib.collections.PathCollection at 0x7f181c1a0ef0>
```

In [35]:
```
# 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[35]:
```
<matplotlib.text.Text at 0x7f181c1eca90>
```

In [36]:
```
# add a legend (uses the labels from plt.scatter)
plt.legend()
```

Out[36]:
```
<matplotlib.legend.Legend at 0x7f181c15ba20>
```

In [38]:
```
# add the legend to loc=4 (the lower right hand corner), also gets rid of
plt.legend(loc=4, frameon=False, title='Legend')
```

```
Out[38]: <matplotlib.legend.Legend at 0x7f181c1b65c0>

In [39]: # get children from current axes (the legend is the second to last item in
         plt.gca().get_children()

Out[39]: [<matplotlib.collections.PathCollection at 0x7f181c1a0550>,
          <matplotlib.collections.PathCollection at 0x7f181c1a0ef0>,
          <matplotlib.spines.Spine at 0x7f181c249518>,
          <matplotlib.spines.Spine at 0x7f181c2c0128>,
          <matplotlib.spines.Spine at 0x7f181c2c0710>,
          <matplotlib.spines.Spine at 0x7f181c2c02b0>,
          <matplotlib.axis.XAxis at 0x7f181c2bc710>,
          <matplotlib.axis.YAxis at 0x7f181c23e828>,
          <matplotlib.text.Text at 0x7f181c1eca90>,
          <matplotlib.text.Text at 0x7f181c1ecb00>,
          <matplotlib.text.Text at 0x7f181c1ecb70>,
          <matplotlib.legend.Legend at 0x7f181c1b65c0>,
          <matplotlib.patches.Rectangle at 0x7f181c1ecba8>]

In [40]: # get the legend from the current axes
         legend = plt.gca().get_children()[-2]

In [46]: # you can use get_children to navigate through the child artists
         legend.get_children()[0].get_children()[1].get_children()[0].get_children

Out[46]: [<matplotlib.offsetbox.HPacker at 0x7f181c1c9cf8>,
          <matplotlib.offsetbox.HPacker at 0x7f181c1c9240>]

In [47]: # 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 0x7f181c175358>
        <matplotlib.offsetbox.TextArea object at 0x7f181c2ef940>
            Text(0,0,'None')
        <matplotlib.offsetbox.HPacker object at 0x7f181c2a8d30>
            <matplotlib.offsetbox.VPacker object at 0x7f181c2a8320>
                <matplotlib.offsetbox.HPacker object at 0x7f181c477e48>
                    <matplotlib.offsetbox.DrawingArea object at 0x7f181c1750f0>
```

```
                        <matplotlib.collections.PathCollection object at 0x7f181c2b
                    <matplotlib.offsetbox.TextArea object at 0x7f181c2a89e8>
                        Text(0,0,'Tall students')
                <matplotlib.offsetbox.HPacker object at 0x7f181c4da8d0>
                    <matplotlib.offsetbox.DrawingArea object at 0x7f181c1b6198>
                        <matplotlib.collections.PathCollection object at 0x7f181c1a
                    <matplotlib.offsetbox.TextArea object at 0x7f181c1c9e10>
                        Text(0,0,'Short students')
    FancyBboxPatch(0,0;1x1)
```

3 Line Plots

```
In [48]: 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[48]: [<matplotlib.lines.Line2D at 0x7f181c0fca20>,
          <matplotlib.lines.Line2D at 0x7f181c0fcba8>]

In [49]: # plot another series with a dashed red line
         plt.plot([22,44,55], '--r')

Out[49]: [<matplotlib.lines.Line2D at 0x7f181c131518>]

In [50]: 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[50]: <matplotlib.legend.Legend at 0x7f181c1196d8>

In [56]: # 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[56]: <matplotlib.collections.PolyCollection at 0x7f181c1313c8>
```

Let's try working with dates!

```
In [57]: plt.figure()

         observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime6

         plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponer

<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Out[57]: [<matplotlib.lines.Line2D at 0x7f1815f032e8>,
          <matplotlib.lines.Line2D at 0x7f1815f03470>]
```

Let's try using pandas

```
In [58]: import pandas as pd

         plt.figure()
         observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime6
         observation_dates = map(pd.to_datetime, observation_dates) # trying to plo
         plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponer

<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
        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-58-31d150774667> in <module>()
      4 observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='dateti
      5 observation_dates = map(pd.to_datetime, observation_dates) # trying to
----> 6 plt.plot(observation_dates, linear_data, '-o',  observation_dates, expo


    /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(sel
   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 [59]: plt.figure()
         observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime6
         observation_dates = list(map(pd.to_datetime, observation_dates)) # convert
         plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponen
```

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

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

```
Out[59]: [<matplotlib.lines.Line2D at 0x7f180e89b3c8>,
          <matplotlib.lines.Line2D at 0x7f180e866e48>]
```

```
In [60]: x = plt.gca().xaxis

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

In [61]: # adjust the subplot so the text doesn't run off the image
         plt.subplots_adjust(bottom=0.25)

In [62]: ax = plt.gca()
         ax.set_xlabel('Date')
         ax.set_ylabel('Units')
         ax.set_title('Exponential vs. Linear performance')

Out[62]: <matplotlib.text.Text at 0x7f180e83e1d0>

In [63]: # you can add mathematical expressions in any text element
         ax.set_title("Exponential ($x^2$) vs. Linear ($x$) performance")

Out[63]: <matplotlib.text.Text at 0x7f180e83e1d0>
```

4 Bar Charts

```
In [67]: 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[67]: <Container object of 8 artists>

In [68]: new_xvals = []

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

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

Out[68]: <Container object of 8 artists>

In [69]: 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 randon
         plt.bar(xvals, linear_data, width = 0.3, yerr=linear_err)
```

```
Out[69]: <Container object of 8 artists>

In [70]: # 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[70]: <Container object of 8 artists>

In [71]: # 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[71]: <Container object of 8 artists>

In [ ]:
```