1_PlottingBasics

April 12, 2017

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 [ ]:
```
import matplotlib as mpl
```

In [ ]:
```
import matplotlib.pyplot as plt
```

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

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

2 Scatterplots

In [ ]:
```
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 chil
```

In [ ]:
```
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)
```

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

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

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

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

```
In [ ]: # add the legend to loc=4 (the lower right hand corner), also gets rid of t
        plt.legend(loc=4, frameon=False, title='Legend')
```

3 Line Plots

```
In [ ]: 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')

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

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

In [ ]: # 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)
```

Let's try working with dates!

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

        observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64

        plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponent
```

Let's try using pandas

```
In [ ]: plt.figure()
        observation_dates = np.arange('2017-01-01', '2017-01-09', dtype='datetime64
        observation_dates = list(map(pd.to_datetime, observation_dates)) # convert
        plt.plot(observation_dates, linear_data, '-o',  observation_dates, exponent

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

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

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

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

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

4 Bar Charts

```
In [ ]: plt.figure()
        xvals = range(len(linear_data))
        plt.bar(xvals, linear_data, width = 0.6)

In [ ]: new_xvals = []

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

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

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 [ ]: # 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'

In [ ]: # 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'

In [ ]:
```