Beginner's Python Cheat Sheet — matplotlib

What is matplotlib?

Data visualization involves exploring data through visual representations. The matplotlib package helps you make visually appealing representations of the data you're working with. matplotlib is extremely flexible; these examples will help you get started with a few simple visualizations.

Installing matplotlib

matplotlib runs on all systems, but setup is slightly different depending on your OS. If the minimal instructions here don't work for you, see the more detailed instructions at http://ehmatthes.github.io/pcc/. You should also consider installing the Anaconda distrubution of Python from https://continuum.io/downloads/, which includes matplotlib.

matplotlib on Linux

```
$ sudo apt-get install python3-matplotlib
```

matplotlib on OS X
Start a terminal session and enter `import matplotlib` *to see if it's already installed on your system. If not, try this command:*

```
$ pip install --user matplotlib
```

matplotlib on Windows
You first need to install Visual Studio, which you can do from https://dev.windows.com/. The Community edition is free. Then go to https://pypi.python.org/pypi/matplotlib/ or http://www.lfd.uic.edu/~gohlke/pythonlibs/#matplotlib and download an appropriate installer file.

Line graphs and scatter plots

Making a line graph

```
import matplotlib.pyplot as plt

x_values = [0, 1, 2, 3, 4, 5]
squares = [0, 1, 4, 9, 16, 25]
plt.plot(x_values, squares)
plt.show()
```

Line graphs and scatter plots (cont.)

Making a scatter plot
The scatter() function takes a list of x values and a list of y values, and a variety of optional arguments. The `s=10` *argument controls the size of each point.*

```
import matplotlib.pyplot as plt

x_values = list(range(1000))
squares = [x**2 for x in x_values]

plt.scatter(x_values, squares, s=10)
plt.show()
```

Customizing plots

Plots can be customized in a wide variety of ways. Just about any element of a plot can be customized.

Adding titles and labels, and scaling axes

```
import matplotlib.pyplot as plt

x_values = list(range(1000))
squares = [x**2 for x in x_values]
plt.scatter(x_values, squares, s=10)

plt.title("Square Numbers", fontsize=24)
plt.xlabel("Value", fontsize=18)
plt.ylabel("Square of Value", fontsize=18)
plt.tick_params(axis='both', which='major',
                labelsize=14)
plt.axis([0, 1100, 0, 1100000])

plt.show()
```

Using a colormap
A colormap varies the point colors from one shade to another, based on a certain value for each point. The value used to determine the color of each point is passed to the `c` *argument, and the* `cmap` *argument specifies which colormap to use.*
    The `edgecolor='none'` *argument removes the black outline from each point.*

```
plt.scatter(x_values, squares, c=squares,
            cmap=plt.cm.Blues, edgecolor='none',
            s=10)
```
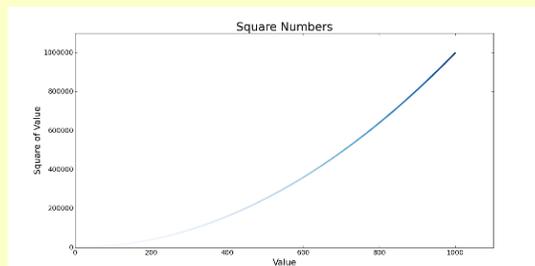

Customizing plots (cont.)

Emphasizing points
You can plot as much data as you want on one plot. Here we re-plot the first and last points larger to emphasize them.

```
import matplotlib.pyplot as plt

x_values = list(range(1000))
squares = [x**2 for x in x_values]
plt.scatter(x_values, squares, c=squares,
            cmap=plt.cm.Blues, edgecolor='none',
            s=10)

plt.scatter(x_values[0], squares[0], c='green',
            edgecolor='none', s=100)
plt.scatter(x_values[-1], squares[-1], c='red',
            edgecolor='none', s=100)

plt.title("Square Numbers", fontsize=24)
--snip--
```

Removing axes
You can customize or remove axes entirely. Here's how to access each axis, and hide it.

```
plt.axes().get_xaxis().set_visible(False)
plt.axes().get_yaxis().set_visible(False)
```

Setting a custom figure size
You can make your plot as big or small as you want. Before plotting your data, add the following code. The `dpi` *argument is optional; if you don't know your system's resolution you can omit the argument and adjust the* `figsize` *argument accordingly.*

```
plt.figure(dpi=128, figsize=(10, 6))
```

Saving a plot
The matplotlib viewer has an interactive save button, but you can also save your visualizations programmatically. To do so, replace `plt.show()` *with* `plt.savefig()`. *The* `bbox_inches='tight'` *argument trims extra whitespace from the plot.*
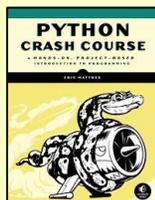
```
plt.savefig('squares.png', bbox_inches='tight')
```

Online resources

The matplotlib gallery and documentation are at http://matplotlib.org/. Be sure to visit the examples, gallery, and pyplot links.

Python Crash Course

Covers Python 3 and Python 2

nostarchpress.com/pythoncrashcourse

Multiple plots

You can make as many plots as you want on one figure. When you make multiple plots, you can emphasize relationships in the data. For example you can fill the space between two sets of data.

Plotting two sets of data
Here we use `plt.scatter()` *twice to plot square numbers and cubes on the same figure.*

```python
import matplotlib.pyplot as plt

x_values = list(range(11))
squares = [x**2 for x in x_values]
cubes = [x**3 for x in x_values]

plt.scatter(x_values, squares, c='blue',
        edgecolor='none', s=20)
plt.scatter(x_values, cubes, c='red',
        edgecolor='none', s=20)

plt.axis([0, 11, 0, 1100])
plt.show()
```

Filling the space between data sets
The `fill_between()` *method fills the space between two data sets. It takes a series of x-values and two series of y-values. It also takes a* `facecolor` *to use for the fill, and an optional* `alpha` *argument that controls the color's transparency.*

```python
plt.fill_between(x_values, cubes, squares,
        facecolor='blue', alpha=0.25)
```

Working with dates and times

Many interesting data sets have a date or time as the x-value. Python's datetime module helps you work with this kind of data.

Generating the current date
The `datetime.now()` *function returns a datetime object representing the current date and time.*

```python
from datetime import datetime as dt

today = dt.now()
date_string = dt.strftime(today, '%m/%d/%Y')
print(date_string)
```

Generating a specific date
You can also generate a datetime object for any date and time you want. The positional order of arguments is year, month, and day. The hour, minute, second, and microsecond arguments are optional.

```python
from datetime import datetime as dt

new_years = dt(2017, 1, 1)
fall_equinox = dt(year=2016, month=9, day=22)
```

Working with dates and times (cont.)

Datetime formatting arguments
The `strftime()` *function generates a formatted string from a datetime object, and the* `strptime()` *function genereates a datetime object from a string. The following codes let you work with dates exactly as you need to.*

%A	Weekday name, such as Monday
%B	Month name, such as January
%m	Month, as a number (01 to 12)
%d	Day of the month, as a number (01 to 31)
%Y	Four-digit year, such as 2016
%y	Two-digit year, such as 16
%H	Hour, in 24-hour format (00 to 23)
%I	Hour, in 12-hour format (01 to 12)
%p	AM or PM
%M	Minutes (00 to 59)
%S	Seconds (00 to 61)

Converting a string to a datetime object

```python
new_years = dt.strptime('1/1/2017', '%m/%d/%Y')
```

Converting a datetime object to a string

```python
ny_string = dt.strftime(new_years, '%B %d, %Y')
print(ny_string)
```

Plotting high temperatures
The following code creates a list of dates and a corresponding list of high temperatures. It then plots the high temperatures, with the date labels displayed in a specific format.

```python
from datetime import datetime as dt

import matplotlib.pyplot as plt
from matplotlib import dates as mdates

dates = [
    dt(2016, 6, 21), dt(2016, 6, 22),
    dt(2016, 6, 23), dt(2016, 6, 24),
    ]

highs = [57, 68, 64, 59]

fig = plt.figure(dpi=128, figsize=(10,6))
plt.plot(dates, highs, c='red')
plt.title("Daily High Temps", fontsize=24)
plt.ylabel("Temp (F)", fontsize=16)

x_axis = plt.axes().get_xaxis()
x_axis.set_major_formatter(
        mdates.DateFormatter('%B %d %Y')
        )
fig.autofmt_xdate()

plt.show()
```

Multiple plots in one figure

You can include as many individual graphs in one figure as you want. This is useful, for example, when comparing related datasets.

Sharing an x-axis
The following code plots a set of squares and a set of cubes on two separate graphs that share a common x-axis.

The `plt.subplots()` *function returns a figure object and a tuple of axes. Each set of axes corresponds to a separate plot in the figure. The first two arguments control the number of rows and columns generated in the figure.*

```python
import matplotlib.pyplot as plt

x_vals = list(range(11))
squares = [x**2 for x in x_vals]
cubes = [x**3 for x in x_vals]

fig, axarr = plt.subplots(2, 1, sharex=True)

axarr[0].scatter(x_vals, squares)
axarr[0].set_title('Squares')

axarr[1].scatter(x_vals, cubes, c='red')
axarr[1].set_title('Cubes')

plt.show()
```
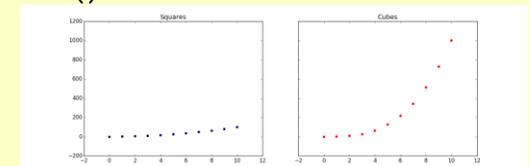
Sharing a y-axis
To share a y-axis, we use the `sharey=True` *argument.*

```python
import matplotlib.pyplot as plt

x_vals = list(range(11))
squares = [x**2 for x in x_vals]
cubes = [x**3 for x in x_vals]

fig, axarr = plt.subplots(1, 2, sharey=True)

axarr[0].scatter(x_vals, squares)
axarr[0].set_title('Squares')

axarr[1].scatter(x_vals, cubes, c='red')
axarr[1].set_title('Cubes')

plt.show()
```