‹ (../06-visualization-ggplot-python/)

Python for ecologists (../)

Data Ingest & Visualization - Matplotlib & Pandas

› (../08-working-with-sql/)

Overview

Teaching: 20 min
Exercises: 25 min
Questions
- What other tools can I use to create plots apart from ggplot?
- Why should I use Python to create plots?

Objectives
- Import the pyplot toolbox to create figures in Python.

Putting it all together

Up to this point, we have walked through tasks that are often involved in handling and processing data using the workshop ready cleaned
files that we have provided. In this wrap-up exercise, we will perform many of the same tasks with real data sets. This lesson also covers data visualization.

As opposed to the previous ones, this lesson does not give step-by-step directions to each of the tasks. Use the lesson materials you've already gone through as well as the Python documentation to help you along.

1. Obtain data

There are many repositories online from which you can obtain data. We are providing you with one data file to use with these exercises, but feel free to use any data that is relevant to your research. The file `bouldercreek_09_2013.txt` contains stream discharge data, summarized at 15 15 minute intervals (in cubic feet per second) for a streamgage on Boulder Creek at North 75th Street (USGS gage06730200) for 1-30 September 2013. If you'd like to use this dataset, please find it in the data folder.

2. Clean up your data and open it using Python and Pandas

To begin, import your data file into Python using Pandas. Did it fail? Your data file probably has a header that Pandas does not recognize as part of the data table. Remove this header, but do not simply delete it in a text editor! Use either a shell script or Python to do this - you wouldn't want to do it by hand if you had many files to process.

If you are still having trouble importing the data as a table using Pandas, check the documentation. You can open the docstring in an ipython notebook using a question mark. For example:

```
import pandas as pd
pd.read_csv?
```

Look through the function arguments to see if there is a default value that is different from what your file requires (Hint: the problem is most likely the delimiter or separator. Common delimiters are `','` for comma, `' '` for space, and `'\t'` for tab).

Create a DataFrame that includes only the values of the data that are useful to you. In the streamgage file, those values might be the date, time, and discharge measurements. Convert any measurements in imperial units into SI units. You can also change the name of the columns in the DataFrame like this:

```
    df = pd.DataFrame({'1stcolumn':[100,200], '2ndcolumn':[10,20]}) # this just creates
a DataFrame for the example!
    print('With the old column names:\n') # the \n makes a new line, so it's easier to
see
    print(df)

    df.columns = ['FirstColumn','SecondColumn'] # rename the columns!
    print('\n\nWith the new column names:\n')
    print(df)

    With the old column names:

       1stcolumn  2ndcolumn
    0        100         10
    1        200         20


    With the new column names:

       FirstColumn  SecondColumn
    0          100            10
    1          200            20
```

3. Make a line plot of your data

Matplotlib is a Python library that can be used to visualize data. The toolbox `matplotlib.pyplot` is a collection of functions that make matplotlib work like MATLAB. In most cases, this is all that you will need to use, but there are many other useful tools in matplotlib that you should explore.

We will cover a few basic commands for formatting plots in this lesson. A great resource for help styling your figures is the matplotlib gallery (http://matplotlib.org/gallery.html), which includes plots in many different styles and the source code that creates them. The simplest of plots is the 2 dimensional line plot. These examples walk through the basic commands for making line plots using pyplots.

Challenge - Lots of plots

Make a variety of line plots from your data. If you are using the streamgage data, these could include (1) a hydrograph of the entire month of September 2013, (2) the discharge record for the week of the 2013 Front Range flood (September 9 through 15), (3) discharge vs. time of day, for every day in the record in one figure (Hint: use loops to combine strings and give every line a different style and color), and (4) minimum, maximum, and mean daily discharge values. Add axis labels, titles, and legends to your figures. Make at least one figure with multiple plots using the function `subplot()`.

Using pyplot:

First, import the pyplot toolbox:

```
    import matplotlib.pyplot as plt
```

By default, matplotlib will create the figure in a separate window. When using ipython notebooks, we can make figures appear in-line within the notebook by writing:

```
    %matplotlib inline
```

We can start by plotting the values of a list of numbers (matplotlib can handle many types of numeric data, including numpy arrays and pandas DataFrames - we are just using a list as an example!):

```
    list_numbers = [1.5, 4, 2.2, 5.7]
    plt.plot(list_numbers)
    plt.show()
```

The command `plt.show()` prompts Python to display the figure. Without it, it creates an object in memory but doesn't produce a visible plot. The ipython notebooks (if using `%matplotlib inline`) will automatically show you the figure even if you don't write `plt.show()`, but get in the habit of including this command!

If you provide the `plot()` function with only one list of numbers, it assumes that it is a sequence of y-values and plots them against their index (the first value in the list is plotted at `x=0`, the second at `x=1`, etc). If the function `plot()` receives two lists, it assumes the first one is the x-values and the second the y-values. The line connecting the points will follow the list in order:

```
    plt.plot([6.8, 4.3, 3.2, 8.1], list_numbers)
    plt.show()
```

A third, optional argument in `plot()` is a string of characters that indicates the line type and color for the plot. The default value is a continuous blue line. For example, we can make the line red (`'r'`), with circles at every data point (`'o'`), and a dot-dash pattern (`'-.'`). Look through the matplotlib gallery for more examples.

```
    plt.plot([6.8, 4.3, 3.2, 8.1], list_numbers, 'ro-.')
    plt.axis([0,10,0,6])
    plt.show()
```

The command `plt.axis()` sets the limits of the axes from a list of `[xmin, xmax, ymin, ymax]` values (the square brackets are needed because the argument for the function `axis()` is one list of values, not four separate numbers!). The functions `xlabel()` and `ylabel()` will label the axes, and `title()` will write a title above the figure.

A single figure can include multiple lines, and they can be plotted using the same `plt.plot()` command by adding more pairs of x values and y values (and optionally line styles):

```
    import numpy as np

    # create a numpy array between 0 and 10, with values evenly spaced every 0.5
    t = np.arange(0., 10., 0.5)

    # red dashes with no symbols, blue squares with a solid line, and green triangles w
ith a dotted line
    plt.plot(t, t, 'r--', t, t**2, 'bs-', t, t**3, 'g^:')

    plt.xlabel('This is the x axis')
    plt.ylabel('This is the y axis')
    plt.title('This is the figure title')

    plt.show()
```

We can include a legend by adding the optional keyword argument `label=''` in `plot()`. Caution: We cannot add labels to multiple lines that are plotted simultaneously by the `plt.plot()` command like we did above because Python won't know to which line to assign the value of the argument label. Multiple lines can also be plotted in the

same figure by calling the `plot()` function several times:

```
    # red dashes with no symbols, blue squares with a solid line, and green triangles w
ith a dotted line
    plt.plot(t, t, 'r--', label='linear')
    plt.plot(t, t**2, 'bs-', label='square')
    plt.plot(t, t**3, 'g^:', label='cubic')

    plt.legend(loc='upper left', shadow=True, fontsize='x-large')

    plt.xlabel('This is the x axis')
    plt.ylabel('This is the y axis')
    plt.title('This is the figure title')

    plt.show()
```

The function `legend()` adds a legend to the figure, and the optional keyword arguments change its style. By default [typing just `plt.legend()`], the legend is on the upper right corner and has no shadow.

Like MATLAB, pyplot is stateful; it keeps track of the current figure and plotting area, and any plotting functions are directed to those axes. To make more than one figure, we use the command `plt.figure()` with an increasing figure number inside the parentheses:

```
    # this is the first figure
    plt.figure(1)
    plt.plot(t, t, 'r--', label='linear')

    plt.legend(loc='upper left', shadow=True, fontsize='x-large')
    plt.title('This is figure 1')

    plt.show()

    # this is a second figure
    plt.figure(2)
    plt.plot(t, t**2, 'bs-', label='square')

    plt.legend(loc='upper left', shadow=True, fontsize='x-large')
    plt.title('This is figure 2')

    plt.show()
```

A single figure can also include multiple plots in a grid pattern. The `subplot()` command especifies the number of rows, the number of columns, and the number of the space in the grid that particular plot is occupying:

```
    plt.figure(1)

    plt.subplot(2,2,1) # two row, two columns, position 1
    plt.plot(t, t, 'r--', label='linear')

    plt.subplot(2,2,2) # two row, two columns, position 2
    plt.plot(t, t**2, 'bs-', label='square')

    plt.subplot(2,2,3) # two row, two columns, position 3
    plt.plot(t, t**3, 'g^:', label='cubic')

    plt.show()
```

4. Make other types of plots:

Matplotlib can make many other types of plots in much the same way that it makes 2 dimensional line plots. Look through the examples in http://matplotlib.org/users/screenshots.html and try a few of them (click on the “Source code” link and copy and paste into a new cell in ipython notebook or save as a text file with a `.py` extension and run in the command line).

Challenge - Final Plot

Display your data using one or more plot types from the example gallery. Which ones to choose will depend on the content of your own data file. If you are using the streamgage file, you could make a histogram of the number of days with a given mean discharge, use bar plots to display daily discharge statistics, or explore the different ways matplotlib can handle dates and times for figures.

Key Points

‹ (../06-visualization-ggplot-python/)

› (../08-working-with-sql/)

Source (https://github.com/datacarpentry/python-ecology-lesson/) / Contributing (https://github.com/datacarpentry/python-ecology-lesson/blob/gh-pages/CONTRIBUTING.md) / Cite (https://github.com/datacarpentry/python-ecology-lesson/blob/gh-pages/CITATION) / Contact ()