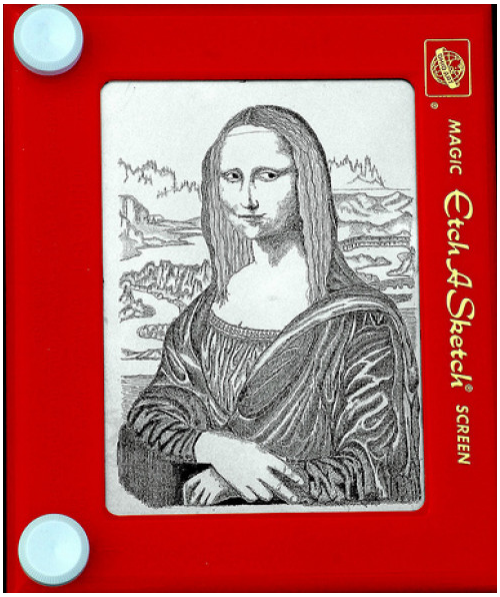
Python #3
Plotting and Visualizing Data

Location: https://people.sc.fsu.edu/~jburkardt/classes/python_2022/python03/python03.pdf

Freely adapted from the Python lessons at https://software-carpentry.org/



Data visualization

- *The eye can see patterns that are hidden in large datasets;*
- *To visualize data, we need functions from the* `matplotlib.pyplot` *library;*
- *We will use* `import matplotlib.pyplot as plt` *to simplify our work;*
- *The simplest command is* `plt.plot(x,y)`*, which plots (x,y) data;*
- *Many commands are available to annotate this basic plot;*
- *Multiple* `plt()` *commands can draw on the same plot;*
- *Complicated data, such as z(x,y), can be displayed with a "heat map";*

1 Getting ready to plot

In any Python session where you want to make plots, you will first need to get access to a plotting library. We will be using the most popular library, known as `matplotlib`. We will see that this library includes many functions that are similar to ones in the MATLAB programming language. If you are familiar with how to make a MATLAB plot, then you can sometimes very simply translate each MATLAB plot command to a corresponding `matplotlib()` function.

First, we must use an `import` statement to get access to the library. Because the full name of the library is so long, we really must come up with an abbreviation. By convention, this plotting library is abbreviated as `plt`. While we are at it, we will also set up the shorthand name `np` for the `numpy` library. Both of these shorthand names are extremely common in Python programming, and are used almost all the time.

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
```

2 A first plot

We are going to start with a plot of n pairs of values (x, y). Usually, such data comes as a list. For this example, however, we will assume that we have a formula $y = f(x)$ and an interva of interest $a \leq x \leq b$ over which we want to display the function. In that case, for any value of n, we can simply evaluate $y(x)$ at n equally spaced values of x. Then we can either display the points as dots, or connect the dots to suggest the shape of the full function.

To do this easily, we need a simple way of generating n points in the interval $[a, b]$. MATLAB users may be familiar with the `x=linspace(a,b,n)` command, which does exactly this. It's no coincidence that this useful function is also available in `numpy`, with the same format.

We will suppose that our functional relationship is $y = x^2 + sin(53x)$. Of course, that's the mathematical version. In Python, this will read `y = x**2 + np.sin ( 53*x )`.

Given a set of n pairs of (x, y) data, the simplest plot command is then `plt.plot(x,y)`. Let's see what happens if we put all these ideas together.

```
>>> import matplotlib.pyplot as plt  # only need to state this once
>>> import numpy as np               # only need to state this once
>>> n = 11
>>> x = np.linspace ( -2.0, 2.0, n )
>>> y = x**2 + np.sin ( 53.0 * x )
>>> plt.plot ( x, y )                # makes the plot, but doesn't show it.
>>> plt.show ( )                     # shows the plot
```

When you are working interactively, a call to plt.show() not only displays the plot, but won't let you proceed until you have closed the plot window.

Although our function is obviously smooth, our plot is not. That is presumably because we are not using enough sample points n. Let's go to $n = 21$. But we will also specify that we want to display the data points as well as the lines that connect them. We do this by adding a format statement to the `plot()` command, namely the string 'o-'. The `o` means that each data point should be marked by a small open circle. The `-` indicates that we should connect the data points. When we don't specify a format statement, then by default this is equivalent to `'-'`, that is, just connect the data points, which is what happened in our first plot. If we specify a format statement, then the connect lines are only drawn if the string includes the `'-'` character.

```
>>> n = 21
>>> x = np.linspace ( -2.0, 2.0, n )  # because n changed, we have to recompute x
>>> y = x**2 + np.sin ( 53.0 * x )    # and then y
>>> plt.plot ( x, y, '-o' )
>>> plt.show ( )
```

You may be surprised that this plot looks just as ragged as the first one. Then again, if you think about it, $\sin(53x)$ is going to oscillate many times over the interval. Maybe we need to boost n significantly. While we're at it, we can skip drawing the connecting lines, and we can use a dot instead of an open circle to display the data. Let's also use the `plt.xlabel()` and `plt.ylabel()` functions to label the axes:

```
>>> n = 101
>>> x = np.linspace ( -2.0, 2.0, n )
>>> y = x**2 + np.sin ( 53.0 * x )
>>> plt.plot ( x, y, '.' )
```

```
>>> plt.xlabel ( '<--- X axis --->' )
>>> plt.ylabel ( '<--- Y = F(X) --->' )
>>> plt.show ( )
```

In some ways, this plot is even worse! It seems as though we have three separate functions. Again, this is because n is not large enough to get a clear image. Let's go for broke with 1001 points. Since we expect this plot to come out nicely, we will opt to draw the connecting lines, and not the data points. we will also "decorate" the plot, by adding grid lines and a title:

```
>>> n = 1001
>>> x = np.linspace ( -2.0, 2.0, n )
>>> y = x**2 + np.sin ( 53.0 * x )
>>> plt.plot ( x, y, '-' )
>>> plt.grid ( True )
>>> plt.title ( '1001 data points' )
>>> plt.show ( )
```

3 Two curves on one plot

If you are used to plotting in MATLAB, you probably know that it can be tricky to show two curves on the same plot; the obvious approach would be to call `plot()` twice, but in MATLAB, the second call starts by erasing the output of the first call. In contrast, with `matplotlib`, you can make several calls to `plt.plot()`, with the single image displaying each of your curves on the same grid.

Consider the following example, where we have also used the numpy function `np.pi`

```
>>> n = 101
>>> x = np.linspace ( 0.0, 2.0 * np.pi, n )
>>> y1 = np.sin ( x )
>>> y2 = np.cos ( x )
>>> plt.plot ( x, y1, 'r-' )  # Draw this curve in Red
>>> plt.plot ( x, y2, 'g-' )  # Draw this curve in Green
>>> plt.grid ( True )
>>> plt.title ( 'Trig fun' )
>>> plt.show ( )
```

4 Erasing a plot; saving a plot to a file

In some cases, you may find that, somehow, data from a previous plot is showing up on your new plot. This can happen especially if you didn't call `plt.show()` to show (and then erase!) the previous plot. In that case, calling `plt.clf()` will clear the previous figure, that is, give you a blank figure in which your next plot can appear.

In some cases, you may want to save a copy of your plot to a file. There are many file types to choose from, including `.eps`, `.gif`, `.jpg`, `.pdf`, `.png`, and `.svg`. (Depending on your installation, some of these might not be available.) Once you have issued your plot commands, but **before** your `plt.show()` command, you can save your plot to a file by the command `plt.savefig ( file.ext )` where `.ext` should actually be one of the file types listed above. Let's plot a function and save it as `humps.png`. Our figure will also include x and y axes that we draw ourselves, in black (a color whose symbol is "k").

```
>>> n = 101
>>> x = np.linspace ( 0.0, 2.0, n )
>>> y = 1.0 / ( ( x - 0.3 )**2 + 0.01 ) + 1.0 / ( ( x - 0.9 )**2 + 0.04 ) - 6.0
>>> plt.plot ( x, y, 'r-' )  # Draw this curve in Red
>>> plt.plot ( [0.0, 2.0], [0.0,0.0], 'k-' )  # Black line from (0,0) to (2,0)
>>> plt.plot ( [0.0, 0.0], [0.0,100.0], 'k-' )  # Black line from (0,0) to (0,100)
```

```
>>> plt.grid ( True )
>>> plt.title ( 'y = humps(x)' )
>>> plt.savefig ( 'humps.png' )
>>> plt.show ( )
```

5 Grouping plots

Especially if your plots are going to be part of a report, or if you want to emphasize a comparison between two plots, it can be useful to be able to create several plots that are grouped together into a single figure.

In this case, technically, `matplotlib`, thinks of a **figure** as the whole graphic object it is working on, whereas the individual plots that make up the figure are each called by the name **axes**. In order to keep track of all the players in this drama, we must begin with a `figure()` statement that defines a name for the figure, and then use an `add_subplot()` command to define an identifying name for each plot to the figure.

Now when we want to define what goes into a specific subplot, we need to preface the command with the subplot name, rather than with the usual `plt.` preface. Doing labels and grids also is handled somewhat differently, as the following example will suggest:

```
>>>  x = np.linspace ( 0.0, 5.0, 101 )
>>>  y1 = np.cos(x)+5*np.cos(1.6*x)-2*np.cos(2*x)+5*np.cos(4.5*x)+7*np.cos(9*x)
>>>  y2 = - np.sin(x)-8*np.sin(1.6*x)+4*np.sin(2*x)-22.5*np.sin(4.5*x)-63*np.sin(9*x)
>>>  fig = plt.figure ( )

>>>  axes1 = fig.add_subplot ( 1, 2, 1 )  # on a (1x2) layout, this if the first subplot
>>>  axes1.plot ( x, y1, 'r-' )
>>>  axes1.set_title ( 'y1=f(x)' )
>>>  axes1.grid ( True )

>>>  axes2 = fig.add_subplot ( 1, 2, 2 )  # and this is the second subplot
>>>  axes2.plot ( x, y2, 'b-' )
>>>  axes2.set_title ( 'y2=dfdx(x)' )
>>>  axes2.grid ( True )

>>>  plt.show ( )
```

If we want instead to create a figure which holds a table of three rows and two columns, then we used commands like `axes1 = fig.add_subplot ( 3, 2, 1 )` through `axes6 = fig.add_subplot ( 3, 2, 6 )`. The third index in the `subplot()` function refers to the subplots position, which will be numbered from left to right columns, and then from top to bottom rows, as suggested by this diagram:

```
+----------+----------+----------+
| Subplot1 | Subplot2 | Subplot3 |
+----------+----------+----------+
| Subplot4 | Subplot5 | Subplot6 |
+----------+----------+----------+
```

(Why we're suddenly counting starting at 1 instead of 0 is a mystery to me!)

6 Group plot of the medical data

Now we want to turn our graphic skills to the medical records, which we saw in the previous discussion, stored in a file named `inflammation-01.csv`. If you do not have a copy of this file, it can be downloaded from the class website. Another way to get it is to go to

`https://people.sc.fsu.edu/~jburkardt/classes/python_2022/python_2022.html`

then choose `datasets`, which will give you a menu that includes the file we are interested in.

Recall from the previous discussion how to read information from this file into a Python variable which we might call `data`. Recall that the data is a table, with each of the 60 rows representing information for a given patient, and each column representing measurements taken on one of 40 days.

We want to compute, for each of the 40 days (= row = axis 0), the maximum, average, and minimum inflammation values. Then we want to display these quantities in a single figure, as a horizontal row of three plots. Here we go!

```
>>> data = np.loadtxt ( fname = 'inflammation-01.csv', delimiter = ',' )
>>> y1 = np.max ( data, axis = 0 )
>>> y2 = np.mean ( data, axis = 0 )
>>> y3 = np.min ( data, axis = 0 )

>>> fig = plt.figure ( )

>>> axes1 = fig.add_subplot ( 1, 3, 1 )  # on a (1x3) layout, this if the first subplot
>>> axes1.plot ( y1, 'r-' )
>>> axes1.set_title ( 'Maximum inflammation' )
>>> axes1.grid ( True )

>>> axes2 = fig.add_subplot ( 1, 3, 2 )  # and this is the second subplot
>>> axes2.plot ( y2, 'b-' )
>>> axes2.set_title ( 'Mean inflammation' )
>>> axes2.grid ( True )

>>> axes3 = fig.add_subplot ( 1, 3, 3 )  # and this is the third subplot
>>> axes3.plot ( y3, 'g-' )
>>> axes3.set_title ( 'Minimum inflammation' )
>>> axes3.grid ( True )

>>> plt.show ( )
```

Notice one peculiar thing: in our `plot()` commands, we are only giving a y value, with no x information. That's OK, because in such a case, `matplotlib` simply uses the index of each item as its x value; in this case, we always have $0 \leq x < 40$.

Another peculiar thing is that, if this data is really from a medical study, then it is very unlikely that the mean value would behave in such a regular fashion; even the maximum and minimum functions would be expected to be much more irregular than we are seeing. Did our researcher Dr Maverick really make these measurements, or is something else going on?