W4 Tuesday & Wednesday - Plotting and graphics

The focus of the next two days will be twofold: on the one hand we will investigate one of Python's most valuable modules: Numpy. The second objective is to use Numpy to investigate some properties of pictures, particularly pictures that are made up of pixels. We are going to apply some simple mathematic operations on numpy arrays, by extracting features of the images and by manipulating them. Finally, we will encounter some very simple plotting features in Python. A slightly more elaborate example is given in the script 'Iris.py'. Just run that script/notebook after you've finished working on the flower images, the main objective is to demonstrate a few features of plotting in Python so that you are aware of what is possible.

Numpy

Numpy is a library outside of the Python 'standard library', but it is arguably one of the most widely used. It is at the heart of every scientific computing and ploting module available in Python, including, for instance, BioPython. Numpy allows you to do very powerful matrix operations very efficiently, and without having to do too much programming. Matrix or vector operations are at the heart of many scientific analysis algorithms, in physics, engineering, chemistry, but also in biology.

Task 1: getting familiar with numpy.

Open the **'W4D1 Basic Numpy'** notebook.

READ THIS CAREFULLY!
You open the notebook by: first download the W4D1.zip file, expand it somewhere convenient, and then navigate to that directory from a terminal.
If your working directory is that W4D1 directory, type:

```
jupyter notebook
```

This will open a new notebook browser in your web browser. Activate the **'W4D1 Basic Numpy'** notebook by double-clicking. Go through it, try running fields – if you want with changed parameters or fields -, and generally try to understand what the various operations, etc., mean.

Then, at the end of the notebook, create a new empty cell and continue working in that (every subtask should be done in a separate, new cell. You can add cells by pushing the '+' symbol in the top of your notebook. You can move cells up or down using the arrow symbols).

a) Create a numpy array '`x`', based on the following list: `[[67,98,202],[43,2,6],[12,99,100]`. Have a look at Cells #2 and #3 for basic syntax.
Produce the square of the matrix (name `x_square`). Look at cell #12 for basic syntax. What is the value in row 3, column 2? Cell #18 (a.o.) shows basic syntax.

b) multiply the array using the vector '`a`' from the Notebook and call that matrix '`ax`'. In which 'direction' does the multiplication occur?

c) Now do the same, but on a transposed matrix '`x`'. Call the array '`ax_t`'. See cell #11 for syntax. (Hint: if you are not certain what 'transpose' means, try it out!).

d) Now create a 3-dimensional array by stacking '`x`', '`x_square`', '`ax`' and '`ax_t`'. Use the '`np.dstack`' method for that:

```
np.stack((x,x_square,ax,ax_t))
```

Note that this function takes only one argument: a tuple containing the arrays to stack. Look at cell #6 for syntax. Name the resulting array '`x3d`'. What is the shape of the array (use `x3d.shape` to find out. Btw, note that 'shape' is not a function, but a property. Functions or methods require parentheses)

e) By applying '`dir(x3d)`' you can find out which methods can be applied to your array. Calculate sum, mean, standard deviation, minimum value and maximum value of the matrix. See cell #14 for an example.

f) Retrieve all values of '`x3d`' larger than 200. Cells #21 and #22 provide further information.

g) Flatten the matrix. Cell #15 provides an example. How long is the vector? Make a ***pseudocode*** loop-in-loop-in-loop that would make a similar flattened matrix (vector). (You can make real Python code for this, that's up to you, but the main thing is that you understand the principle).

Flowers

Sunflowers and pansies

For this exercise you can use much of the code provided in the **'Flowers'** notebook. In fact, we have prepared a new notebook, called **'Flowers exercises'**, that has all the functions you need already arranged. Make sure to 'activate' the functions before you require them.

All of the exercises are designed to be done in a purely interactive environment, and each exercise should never require more than 5 or 6 lines of code, most only 2 or 3. No loops should be necessary, but if any of you is adventurous enough to make more complicated code, please go ahead!

What is important is to keep your eye on the various functions. It will be pointed out when you need one of them, but also have a look at what it does.

Each question should be done in a single cell. If you want to make a new cell, press the '+' symbol at the top of your notebook. You can move cells up or down using the arrows.

If we ask for a value(s) of an array, or another value that you need to calculate, make sure to explicitly 'print' the values or objects because otherwise they will not always be displayed when running the cells.

Note: the **RED** cell numbers refer to the 'Original Flowers' notebook. The **BLUE** function names refer to functions found in the 'Flowers-exercises' notebook.

Task 2: Images and arrays

a) Load in the sunflower image:

image_sunflower_0076.png.
Plot the image. What are the dimensions of this image?

For pointers how to do this, have a look at the original 'Flowers' notebook; specifically cells #2 (loading an image), #4 (shape of an image), #5 (plotting), and #6 (showing). Arrange everything in a single cell. When you do '`plt.show()`', you have to close the image before you can continue.

b) What are the pixel values of the pixel at position [50,50], and the one at position [400,230]?

Pointer*: cell #7 of the original 'Flowers' notebook*

Check at this website whether the colors correspond to what you think they should be:
http://www.rapidtables.com/web/color/RGB_Color.htm

c) Isolate the 'blue' channel, and display it. Does that pattern make sense?

Pointer: cell #15 of the original 'Flowers' notebook. Be sure for plotting to include the appropriate color mapping: 'plt.imshow(blue, cmap = cm.Blues)'.

d) - i Calculate the grayscale value of pixels [400,10] and [470,10] by hand.

Pointer: just return the RGB pixel value for these pixels as done in b), and take the average.

- ii Create a grayscale image of the sunflower based on pixel averages, and display it.

Pointer: you can use the 'RGB_to_grayscale' function for that. It takes an RGB image as input, and an optional parameter 'method'. The function returns a grayscale image.

```
gray = RGB_to_grayscale(sunflower,method='pixelaverage')
```

The 'gray' image can be displayed similar to cell #12 of the original 'Flowers' notebook (again, beware of this peculiar color mapping thing):

```
plt.imshow(gray, cmap = cm.Greys_r)
plt.show()
```

- iii What are the dimensions of the grayscale image?

- iv Retrieve the pixel values of the grayscale image at locations: [400,10], [470,10]. Does that make sense when you consider the original and the grayscale images? What value would a black pixel be? And a white pixel?

e) Again, create a grayscale image of the sunflower, but now with weighted averages. You can do the same as in d), but now supply "method='weightedaverage'". What are the values of these two pixels now?

(bonus: if you look at the function 'RGB_to_grayscale', what would happen if you make a type and instead supply "method='weigtdavrge'"?

Task 3: Deriving features from image data: color histograms

a) Plot color histograms for all three channels. Explain the pattern. Remember: yellow is composed of green and red and has little to no blue. And what about that 'blue spike'?

Pointer: take the function 'plot_RGB_histograms', which takes only one variable: an image.
```
plot_RGB_histograms(image)
```
Note that you can save images using the 'disk' symbol in the plot window. You need to close the plot window to continue.

b) Crop the image (remove 100 pixels on all sides), and plot again. Now again make a color histogram plot. Explain the differences of this histogram with the previous one, based on that cropped image.

Pointer: use the 'crop_sides' function, which takes an image as input, and optional parameter of number of pixels to crop on all sides (default: 100). It returns a cropped image.
```
croppedsunflower = crop_sides(sunflower)
```
Making the histogram on the cropped image is done in the same way as in a).

c) Concatenate the color histograms into a single 'feature vector'. How long is that vector? What is the highest value in the vector? Does that match with the plotted color histograms?

Pointer: use the 'hist_all_colors' function. It takes an image as input, and returns a vector.
```
sunflower_hist = hist_all_colors(sunflower)
```

d) now load the picture of a pansy (image_pansy_0179.png). Remember that the images are inside your working directory INSIDE the folder 'Flowers'. Make sure to adjust the path accordingly!

Again, plot the color histogram. What are obvious differences between the histogram of the sunflower and that of the pansy?

Pointer: again use the `plot_RGB_histograms` function.

e) Calculate the Euclidean distance between the arrays `[1,2,3,4]` and `[2,2,5,3]`.
You could do this by turning these into numpy arrays and use the function '`euclid_dist`' that is given, which takes two numpy 1-d arrays (vectors). But you could in fact do this easily by hand too: take the sum of the square of the differences between the two vectors, and then take the square root of that.

https://en.wikipedia.org/wiki/Euclidean_distance

f) Calculate the Euclidean distance between the sunflower and the pansy histograms.

Pointer: You should still have the vectorized color histogram of the sunflower, from c). Make another one for the pansy and use these two vectors to calculate the euclidean distance between the two images using the '`euclid_dist`' function.
```
pansy_hist = hist_all_colors(pansy)
euclid_dist(sunflower_hist,pansy_hist)
```

g) You could extend this by making color histograms for other pictures, such as for a second pansy and/or a second sunflower. If you would calculate the

Euclidean distance between two pansies, would that value be smaller or larger than any of the pansies to a sunflower?

Pointer: you can use the '`euclid_dist_between_images`' function, it takes two images as input.

Further considerations: You could do the same by cropping the images so that you 'zoom in' on the flower more and leave more of the background out. Color histograms, for instance in conjunction with localizing items in a picture, are among the many features that can be used to do automated image analysis, such as, e.g., a plant classification system. Have a look at the last 2 cells of the original 'Flowers' notebook.

If you are interested in image manipulation, Python is a fantastic language to start. It has a very powerful image analysis library (openCV, where 'CV' stands for 'Computer Vision'). For some great free tutorials where you can explore the power of Python for image analysis, check out this website:
http://www.pyimagesearch.com/

Iris data plotting

Task 4 (optional): Run the Jupyter notebook for 'Iris data'.

This is just a little demo of plotting in Python. We will use very simple plotting commands today and later in the week to make simple plots and histograms. You can, however, tweak many parameters to make very complicated and beautiful graphics of your research. The way that you have to 'build up' graphs to get a nice picture seems very tedious. Some prefer R for plotting because it seemingly is less difficult, especially since there are libraries that promise to make beautiful pictures easily. However, in every single case, making publishing-quality plots takes a bit of effort. The reward? Once you can script it, you can do it as many times as you like!!!

Run the notebook before embarking on these questions. Note that you need to close the plotting windows when you want to continue.

a) Describe the structure of the iris dataset. See cells #3-6. Tip: you can also ask to just show the entire 'iris' dataset in one go!

b) Identify the part where the correlation coefficients, and slope an intercept of the fitted lines were calculated. Hint: these are numpy methods. Describe them based on information retrieved from the help function (note the absence of parentheses and the presence of the question mark):

```
np.method?
```

c) [Challenge] Copy the last cell (use the 'copy cell' and 'past below' icons at the top of your notebook. Modify the copied cell. Make the canvas you are plotting on bigger: from 6x8 to 6x12, so that it can include a third plot.
Then, add a third plot, at the bottom that plots petal length vs sepal length. Note that this requires: 1) an array that captures the first and third column of the data, call that X3. Note that you can not only slice, but actually retrieve specific columns by providing a list of index numbers. 2) You need to again define 'FeatureX' and 'FeatureY' to contain 'sepal length' and 'petal length' respectively.

A few final words on notebooks

Jupyter/iPython notebooks are a great way to bridge the gap between completely ‘interactive’ programming, e.g. in Idle or iPython, and ‘full text’ Python scripts, edited in a generic text editor (gEdit, Nano) or a professional IDE (Integrated Development Environment), such as Pycharm.

The advantage is that you can still interactively check how code behaves, while at the same time being able to organize the code in larger chunks, which are easy to edit if you make a mistake, and easier to organize.

Like iPython and Idle, Jupyter notebooks give you all the advantages of autocompletion and hints. Make use of that!

Making notebooks is a great way to document your analysis, and in fact in some of the data analytics community notebooks are the standard way of documenting analyses. Jupyter does support other scripting languages that are often used in data analytics, such as R and Julia (hence the name: Ju-Pyt-R).

Notebooks start becoming a bit cumbersome to use once your script evolves beyond 100-200 lines (depending on structure). Although the good thing is that to keep things manageable in a notebook, organizing your code in subroutines/functions (using ‘def’) is an extra good idea (well, honestly, it is a good idea ALWAYS).

Another downside is that it is not possible to provide arguments, which requires you to hard-code, e.g., input and output paths, or any other parameter you might like to vary between analyses.

Note that there are other valuable tools to document your scripts. One is code management systems such as Git. The scope of Git is a bit different – you document code, not your analysis. Git is an extremely valuable tool. Remember the name.