Scientific Programming – NumPy, Pandas, Visualization

Part I: NumPy exercises:

1) Assume you have a gene expression matrix containing expression levels for N genes (in N matrix rows) in M samples (in M matrix columns). The samples can be separated in two distinct groups or contidions.

In WeBeep (materials → exercises) you will find an example file named "`GSE16237_series_matrix-cleaned-4noamp_vs_4amp.tsv`" extracted from a publicly available data set for gene expression levels from neuroblastoma tumors. The file contains only a subset of the available samples and does not report sample IDs nor gene IDs (to keep this exercise as simple as possible).

Once downloaded, you can read the file as a NumPy array using the function "`np.fromfile()`":

```
ndat = np.fromfile("GSE16237_series_matrix-cleaned-4noamp_vs_4amp.tsv",
                   dtype=float, count=-1, sep='\t').reshape(20464,8)
```

As you can guess from the "`reshape`" transformation, the file contains expression levels for 20,464 genes in 8 tumors. The **first 4 samples (columns)** are for stage 4 tumors which do not harbor an amplification of the MYCN gene, the **last 4 samples (columns)** are from stage 4 tumors which instead do harbor an *amplification of the MYCN gene* (massive increase in copy number of that gene, a frequent event in neuroblastomas).

Tasks:

a) Perform a (dangerously simplified) column-wise normalization of the expression levels by dividing all expression levels of one sample (one column) by the maximum expression level of that column, such that for each column/sample expression levels range from 0 to 1. [This is not how such data would be normalized, but for the sake of this exercise we will consider it a sufficient approximation although it is not …]

b) Then, for each of the two conditions (non-amplified stage 4 neuroblastoma versus MYCN-amplified stage 4 neuroblastoma), i.e., independently for the first 4 columns and the last 4 columns, identify the row (=position in the file) of the gene with the highest mean expression across the four samples of that condition.

Hint: for finding the position (index) of a maximum value in a NumPy array, you can use the function "`np.argmax()`"!

2) Assume you have a 1-dimenstional NumPy array of experimental observations/measurements, where each observervation value is >= 0. You can easily construct an arbitrary example vector.
Immagine that you want to compute `log10` for those observations which are >1, but want to set 0 for all those which are smaller (so you don't get negative values or even worse, `-inf` when the observation was `0`). Can you do this with a single command?

3) The following may be useful for some applications in linear algebra:
Find out how to create NumPy quadratic matrices (same number of rows and columns) which have all elements set to 0 except the values on the diagonal using one single command/function call. Create …

(a) A 4x4 matrix which has all diagonal values set to 1, the rest to 0.
(b) A 4x4 matrix which has the diagonal values ranging from 1 to 4, all other values set to 0.

4) Creation of N-dimensional arrays

a) Create a 2-dimensional array with 4 rows and 25 columns containing the natural numbers from 1 to 100. Use a simple command that avoids listing all 100 numbers.
b) Create the same array without specifying the number of columns in your command!
c) Place the same natural numbers in a 3-dimensional array (choose the necessary number of rows, columns and layers). After having created the array, how can you obtain the number of dimensions, the shape and the content data type from the array object itself?
d) Copy the array from c) into a new array and modify the copy without modifying the original array.
e) Compute the sum of all differences between the array from c) and the array from d). Try using the standard command "`sum()`" and the NumPy version "`np.sum()`". Do you observe the same behavior?
f) The array multiplication with "`a*b`" of two equally shaped matrices `a` and `b` multiplies element-wise the corresponding elements, so it is not a true matrix multiplication. Find out how a true matrix multiplication can be performed in NumPy and perform it by squaring the matrix obtained from a). What shape does the output have?

5) The NumPy function "`np.random.random(N)`" generates N random numbers in the interval [0.0,1.0), i.e., without the upper boundary 1. Use a simple command to check that for N=100 indeed all returned random numbers are smaller than 1 AND at least 0.
Unfortunately performing a logical AND on NumPy arrays does not work with the standard bitwise AND operator ("&"). Can you find out how to perform a logical AND operation on two NumPy arrays?

Part II: Pandas exercises

6) Create two Pandas series containing the same four integers: 1,2,3,4; one Series with a defined index and a Series name and one without. Analyze them both.
How do they look like, can you note the differences?
Can you compare whether the two Series are identical?

7) Take one of the two Series created above. Let's say you named it "`a`".

a) What is the difference (if any) between executing "`a[1]`" and "`a[1:2]`" (where the second expression takes the 0-based positions from 1 to 2-1, so from 1 to 1)?
b) Now create a third Series ("`c`") with the same for integers (1,2,3,4) and use as index the same integers but in inverted order (4,3,2,1). What do you get now for "`c[1]`" and "`c[1:2]`"? Can you explain the observation?
c) Now try "`c.loc[1:2]`". You should get an empty Series. Why is that? How can you fix this? How about "`c.iloc[1:2]`". Do you get what you expect?

8) PLAY with Pandas Series! Try adding elements, removing elements, overwriting elements, etc! Try missing values! Try whatever comes into your mind and see what happens. And: read some documentation; as usual there is more to discover than what we discussed in the lecture …!

9) Pandas DataFrame: Create the first DataFrame "`genes`" that we saw in the lectures (slide 60), so that you have:

```
       geneID chrom protein
TP53     7157    17     p53
AKT 1     207    14    Akt1
RB1      5925    13     pRb
```

a) Add two new columns/data items: "`start_pos`" and "`end_pos`" (look up the genes or just invent some positions). Inspect the DataFrame afterwards. Do you get what you expected?
b) Add a derived data item "`length`" determined from the chromosomal positions added in a)
c) Use single command/expression to get the start position of all genes which have a chromosome number of <=14. Try to get the list of starting positions both as a Pandas Series and as a NumPy array!

10) PLAY with DataFrames! Drop rows and columns!
Try to write the data to a CSV file, etc! Read some documentation and then write the data also as TSV!
Concatenate the DataFrame with itself!
Define a second DataFrame using the same geneIDs (but partly different genes) and containing different information (e.g., gene description, biotype, etc); then merge/join the two DataFrames in different ways (inner, outer, left, right)

Part III: Seaborn exercises

11) Practice some Seaborn, you can find great tutorials online, for example:
https://www.kaggle.com/code/saikiranvepamani/seaborn-exercises
https://elitedatascience.com/python-seaborn-tutorial

Read some documentation! There's always more to discover!!!