M2 EMINES Datasciece
2021/2022

Mathurin Massias

Lab n° 1 : Introduction: Python, Numpy, Pandas

Important preliminary remarks

We are use `jupyter notebook` for all practical sessions. Some important points to remember :

- Loading -

```
import math                          # import a package
import numpy as np                   # import a package under an alias
from sklearn import linear_model     # import a submodule
from os import mkdir                 # import a peculiar function
```

- Using standard help -

When facing a difficulty, you are strongly encouraged to refer to the online documentation of pandas, numpy, etc. It should become a reflex to look for the answer in the doc or on stackoverflow.

```
linear_model.LinearRegression?     # to get some help on the LinearRegression object
```

- Package versions -

```
print(np.__version__)                  # to get a package version
```

- Figures -

```
import matplotlib.pyplot as plt  # import the plt function of package matplotlib
%matplotlib inline               # tell Jupyter to plot figures inside notebook
```

- Evaluation -

The Lab is done by pairs. Send your Lab by email to mathurin.massias@gmail.com with the file name constructed as `fistname1_lastname1_firstname2_lastname2.ipynb`, where you have substituted your first and last names.

Strings

1) From a string containing all the alphabet letters, generate the string `cfilorux` using *slicing*. Do the same for the strings `vxz` and `zxvt`. Don't type the alphabet yourself, use the `string` module.
2) Declare a string variable `"  XHEC DataScience for Business "`. Make it all lowercase. Remove spaces at the beginning and at the end, but not between words. Replace all e's with E's.
3) Display the number π with 9 decimal digits. Don't cast a number to string, and don't use `round` : use Python's string formatting instead (either the `format` method, either the `%` operator).
4) Count the number of occurrences of each character in the string `s="HelLo WorLd!!"`. (in real life, you should use a `collections.Counter` ; here, you are asked to code the method yourself). Output a dictionary that to each character associates the number of occurrences in this string. In this question, we consider that lower and upper case characters are the same (e.g. your dictionary should note have both a L and a l entry).

Fast computations with numpy ; basic plots.

Hint : useful functions : `np.arange, np.allclose, np.all, np.linalg.norm` for instance. The whole `numpy.linalg` module contains interesting Linear Algebra functions. `numpy.random` contains functions to generate arrays of (pseudo-) random numbers.

In all this section, unless asked explicitly, you **cannot use** for/while loops (they are slow in native python).

5) Compute `0.1 + 100 - 100`. Using `==`, check if it is equal to `0.1`. Comment. Compare the two floating point numbers again with an appropriate numpy function.
6) Create a list (resp. an array) containing all square numbers from 1, 4, ... to 121, using a for loop (resp. only numpy).
7) Create an array containing integers from 2 to 14 by step of 3 (2, 5, 8, ...). Create an array with 15 equispaced valued from 0 to 1 included. Use numpy built-in functions.
8) Compute $2\prod_{k=1}^{\infty} \frac{4k^2}{4k^2-1}$ (approximate ∞ by a large number n) using a for loop. Propose a version without loop, using only `numpy` (see `numpy`). Measure the time taken by both versions using `time.time()`. You should display the results with a relevant number of significant digits, e.g. not 0.002487976589749873 seconds. Use the ipython magic `%timeit` again to measure time of one version. Why is it better than `time.time` ?
9) (row and column vectors, aka numpy only knows 1D arrays) Compute the dot product of `np.arange(5)` and `np.ones(5)`. What is the shape of `np.arange(5)` ? How many dimensions does this array have ? What is the shape of its transpose (use `.T`) ? What does transposing 1D arrays do ?
10) What does reshape do in `M = np.arange(12).reshape(2, 6)` ? What does `M[:, ::3]` do ? What happens when you do `np.arange(3) * np.arange(4)[:, None]` (this powerful tool is known as broadcasting)
11) Create a random matrix $M \in \mathbb{R}^{5\times 6}$ with coefficients taken uniformly (and independently) in $[-1, 1]$. Substract to each even column of `M` (say `M[:, 0]` is even), twice the value of the following (uneven) column.
12) Replace the negative values in M by 0. Compute the mean of each row of M. Substract to each row of M, its mean.
13) Create a random matrix $M \in \mathbb{R}^{5\times 10}$ with coefficients taken uniformly (and independently) in $[-1, 1]$. Test whether $G = M^\top M$ is symmetric semi-definite positive, and that its eigenvalues are strictly positive. Compute the rank of G. Compute the Euclidean norm of G. Compute the operator norm of G (aka spectral norm, aka Schatten 2-norm). Compute the standard deviation of each column of G.
14) Plot the functions $x \mapsto x^d$ on the interval $[-1, 2]$ for $d \in \{2, 3, 4\}$ with a decent resolution. Put a xlabel, a ylabel, legend the 3 curves with $d = 2$, $d = 3$, $d = 4$ respectively. Put a title.

Numpy broadcasting

In this exercise, you can use neither lists nor for loops. You should use only numpy's operation, which are fast. An introduction to broadcasting is available here : https://numpy.org/doc/stable/user/basics.broadcasting.html.

15) Create an array with integers 1, 3, ..., 19. Subtract its mean to it. Observe than you can thus subtract a number to an array, even though they do not have the same shape. Create an array with 3 lines and 4 columns, such that `arr[i, j] = 4 * i + j` (it thus contains integers from 0 to 11). `reshape` will help.
16) Now, we subtract vectors to 2D arrays, using broadcasting. Take the previous (3, 4) array, and subtract its column wise mean to it (easy). Subtract its row wise mean to it (technically more challenging the first time, you can add an axis to a 1D array with `arr[:, None]`).
17) Using broadcasting and `np.arange`, create an array of shape (3, 5) such that `arr[i, j] = i * j`.

Value and reference types

18) Create a variable `a` equal to 1000. Check the address in memory of the variable with `a`. Create a second variable `b` equal to 1000. What is the address in memory of `b`? Create a third variable `c` equal to `a`. Do `a += 1`. How does it affect the values of the three variables? Why?

19) Do the same with `a = np.array([0, 1])`. What is going on?

20) Create two variables equal to 1. Using `==`, are they equal? Using `is`, are they equal? Do the same when they are equal to 1000. Medit. What is going on?