

# Parallel Computing with Joblib and Plotting with Seaborn.

LINCS Python Academy

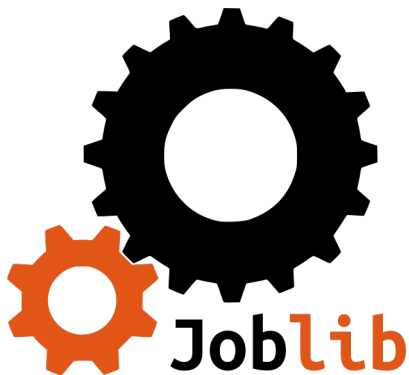
Quentin Lutz

30-10-2019

# 1. Joblib

# 1. Joblib

## An overview

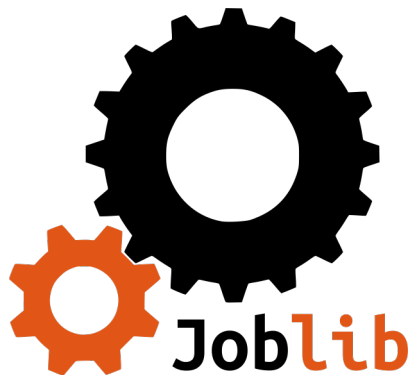


Joblib provides 3 main functionalities:

- Caching
- Saving and loading Python objects (like Pickle)
- Parallel computing

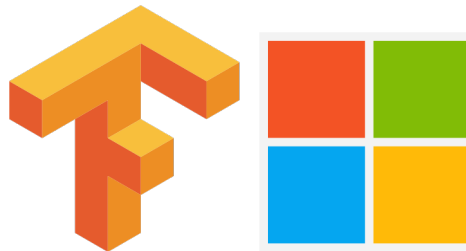
# 1. Joblib

## An overview



Joblib provides 3 main functionalities:

- Caching
- Saving and loading Python objects (like Pickle)
- **Parallel computing**



# 1. Joblib

## Parallel computing

What is parallel computing?

**Many calculations** are carried out **simultaneously**

Why use it?

Your CPU is able to run multiple processes at once.

However, unless dedicated packages are involved, a Python program will only use one.

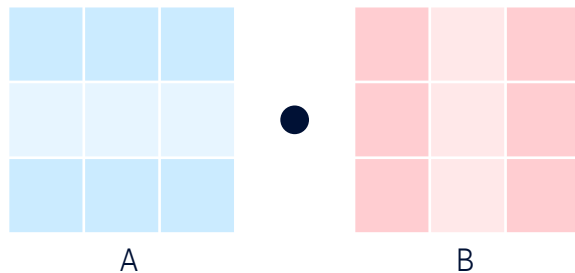
Why not **make the most out of the resources** at your disposal?

# 1. Joblib

## Parallel computing

An example: **Dot product** of matrices

Multiplying two random 3000x3000 matrices:



Results in:

NumPy dot	// NumPy dot with 5 workers	// NumPy dot with 10 workers	// NumPy dot with 30 workers
41.2s	10.1s	6.21s	5.58s

# 1. Joblib

## Pros & Cons of Joblib

### Pros:

- Ease of use
- Somewhat tolerant shared memory access
- Easy debugging

### Cons:

- Lack of actual control
- Lesser performance than even other Python packages (e.g. Numba)

Basically: Parallel computing **for dummies**

# 1. Joblib

## Syntax

Say you want to run multiple executions of some function in parallel:

```
from joblib import Parallel, delayed
```

```
Parallel(n_jobs=n_jobs)(delayed(function)(argument) for argument in arguments)
```

The maximum number  
of processes to  
run at once

Your function

The values to evaluate  
the function at

This returns the list [ function(argument[0]), ... ,function(argument[-1]) ].

It is thus equivalent to [function(argument) for argument in arguments]



# 1. Joblib

## Syntax

For instance:

```
square = lambda x : x**2
```

```
Parallel(n_jobs=2)(delayed(square)(i) for i in range(10))
```

Yields:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# 1. Joblib

## Practical cases

Some considerations must be made in the case of functions with several inputs:

Take the built-in `pow` function whose signature is `pow(x, y, z=None)`. The previous example can be written as:

```
Parallel(n_jobs=2)(delayed(pow)(i,2) for i in range(10))
```

However, note that even if an argument is fixed, its value is copied `n_jobs` times (which can prove troublesome for large objects).

# 1. Joblib

## Practical cases

In case you need multiple parameters to vary, one possible work-around is to generate the list of tuples of arguments to pass. For instance to fill in:

$i^k :$

$i \backslash k$	2	3
1		
2		

Use:

```
from itertools import product
```

```
Parallel(n_jobs=2)(delayed(pow)(i,k) for k, i in product(range(2,4), range(1,3)))
```

Which returns [1, 4, 1, 8]

# 1. Joblib

## Practical cases

Back to the dot product example:

To multiply matrices A and B in NumPy, you could write (assuming the relevant imports have been made):

```
C = dot(A, B)
```

To use Joblib, you would write:

```
C = vstack(Parallel(n_jobs=n_jobs)(delayed(dot)(A_block, B) for A_block in split(A, n_jobs)))
```

# 1. Joblib

## Debugging

When an error is encountered, the corresponding exception is raised.

To get the actual error trace for that exception, one may set `n_jobs` to 1 to disable parallel computing.

## Performance

Using the default settings, performance is at its poorest but the risk of incompatibilities with the original Python code is lower.

Refer to the documentation for further improvements (especially when using compiled extensions overriding the Python Global Interpreter Lock).

# 1. Joblib

## Takeaway

Joblib :

- is an easy-to-use package for parallel computing
- mostly requires no changes to the code to parallelize
- has suboptimal performance when using default parameters in exchange of increased coverage

# References

- <https://joblib.readthedocs.io>
- <https://dzone.com/articles/code-parallelization-joblib>
- <https://buildmedia.readthedocs.org/media/pdf/joblib/latest/joblib.pdf>

## 2. Seaborn



## 2. Seaborn

### An overview

Seaborn is a **data visualization library based on Matplotlib**. It is part of the PyData community (which includes Pandas and Numba).

It features:

- (arguably) **better visuals**
- enhanced **interoperability with Pandas**

## 2. Seaborn

### An overview

Because Seaborn based on Matplotlib, most of the PyPlot syntax is still valid.

In particular, the Figure/Axes duality and the endless Warnings remain but are not necessarily encountered.








In terms of functionalities, some existing plots of Matplotlib have no equivalent in Seaborn (notably 3D plots).

However, it makes it easier to plot a range of common functions over DataFrame-based data.

## 2. Seaborn

### Some Pandas-related features

Say you have a tidy DataFrame containing your data in the observation x variables format:

		Force	PV	Défense	Vitesse	Spécial	Type
Nom							
Bulbizarre		45	49	49	45	65	Plante
Herbizarre		60	62	63	60	80	Plante
Florizarre		80	82	83	80	100	Plante
Salamèche		39	52	43	65	50	Feu
Reptincel		58	64	58	80	65	Feu
Dracaufeu		78	84	78	100	85	Feu
Carapuce		44	48	65	43	50	Eau

## 2. Seaborn

### Some Pandas-related features

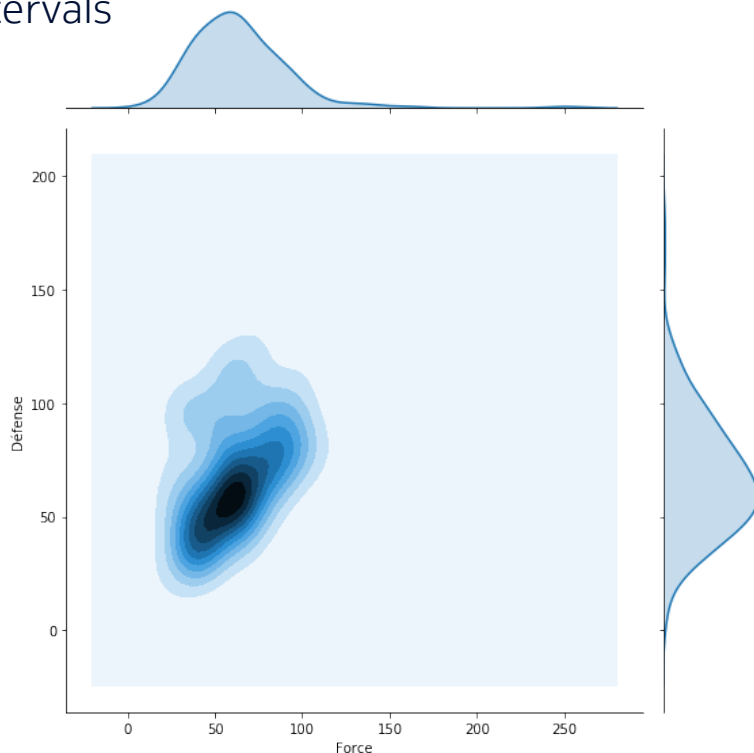
Some computations can be handled by Seaborn. Those include polynomial regression, kernel density estimates and confidence intervals

Here we ran:

```
sns.jointplot(x="Force",y="Défense",  
kind="kde", data=POKEDEX, height=8);
```

Used to set the figure size  
without the PyPlot syntax

Where POKEDEx is the previous DataFrame.



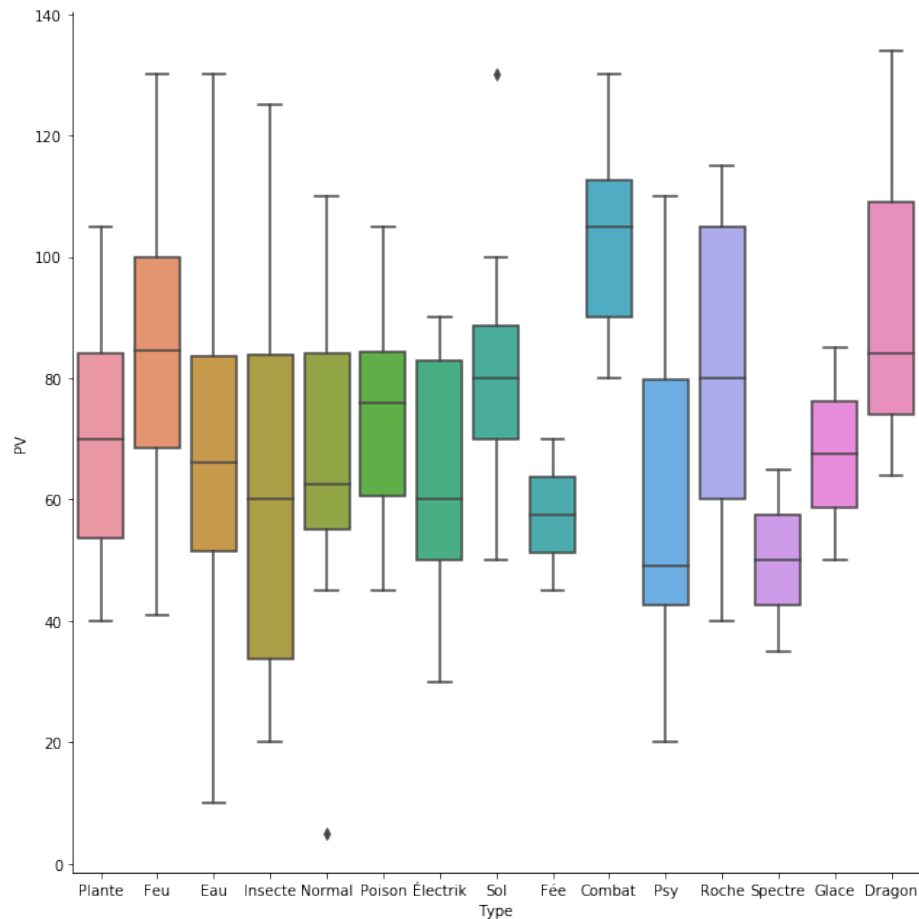
## 2. Seaborn

### Some Pandas-related features

Category plots make it possible to work on multiple subsets of the DataFrame depending on a given criteria.

Here we ran:

```
sns.catplot(x="Type", y="PV", kind="box",  
data=POKEDEX, height=9);
```



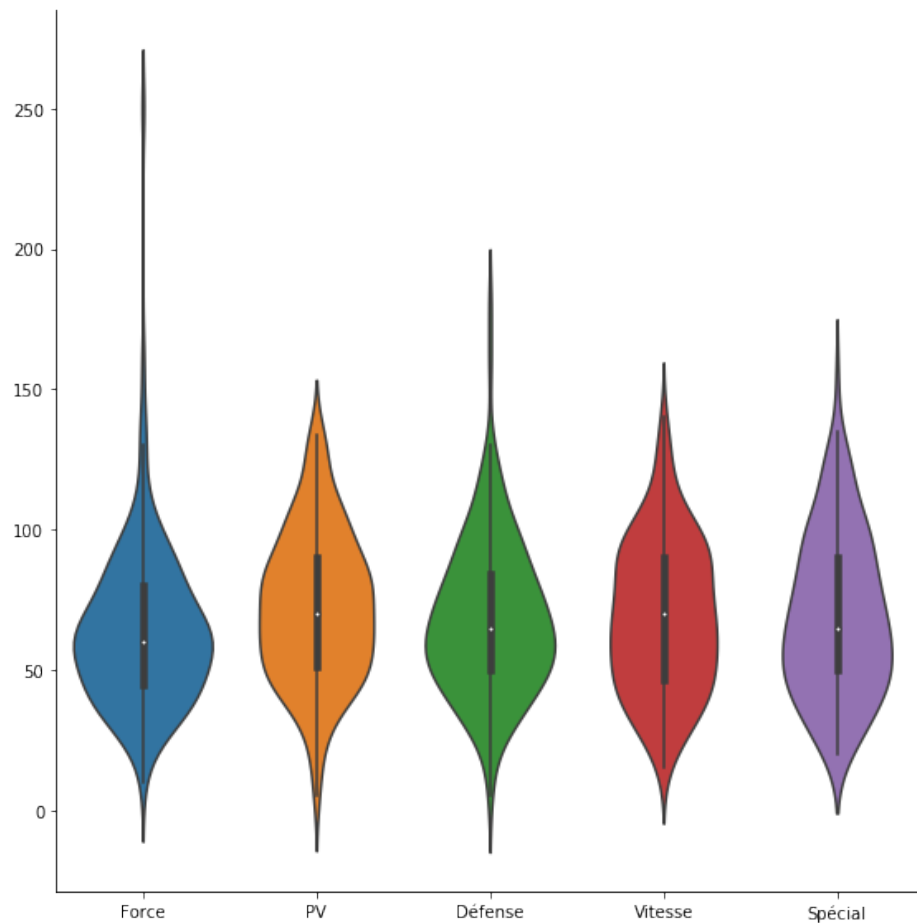
## 2. Seaborn

### Some Pandas-related features

By default, category plots give an overview of the repartition of numerical variables in the DataFrame.

Here we ran:

```
sns.catplot(data=POKEDEX, kind='violin',  
height=8);
```

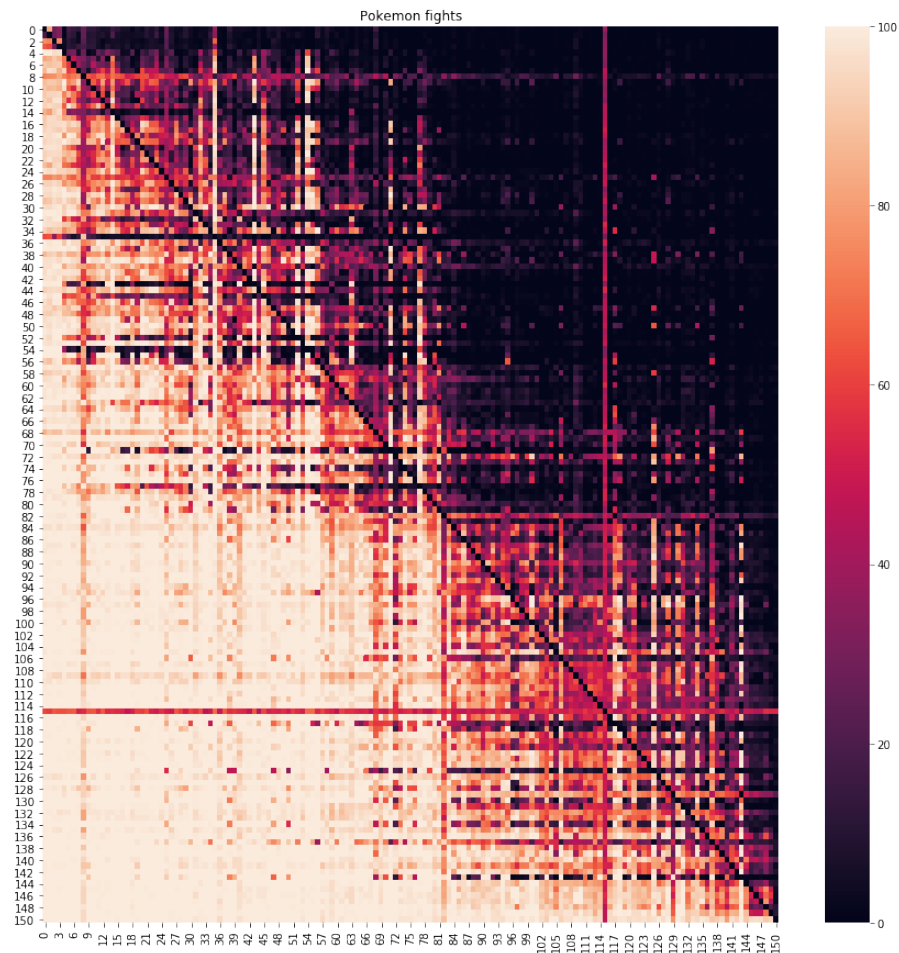


## 2. Seaborn

### Some Pandas-related features

However, it is not always that easy to change parameters such as the size of the figure. For instance, when making a heatmap, you need to use the fig/ax syntax from PyPlot:

```
f, ax = plt.subplots(figsize=(15, 15))  
ax.set_title('Pokemon fights')  
sns.heatmap(mat, linewidths=0, ax=ax);
```



## 2. Seaborn

### Understanding Seaborn plots

Seaborn has many plot types to offer. Some such as catplot are actually aggregations of multiple plots. For example:

```
sns.catplot(x="Type", y="PV", kind="box", data=POKEDEX, size=9);
```

Actually uses the boxplot function which returns an ax object whereas catplot returns a figure-level object.

Thus, **all Seaborn plots are not created equal.**

Note that, apart from heatmap and clustermap, all the plot types can be used without the fig/ax syntax.



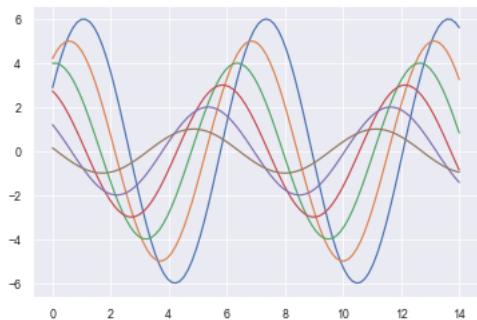
## 2. Seaborn

### Understanding Seaborn plots

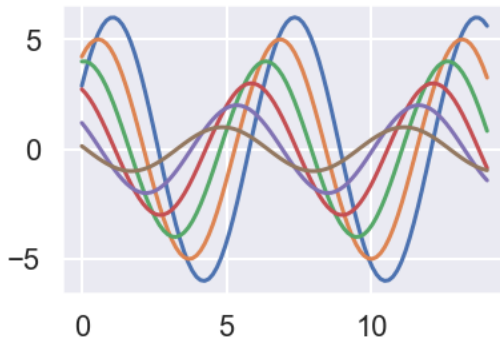
Some aesthetics functions have been simplified in Seaborn using `seaborn.set_xxx` functions that apply to all plots (even Matplotlib ones to some extent).

For instance, the font size and scaling on the axis can be changed simply using the following:

```
sns.set_context("paper")
```



```
sns.set_context("poster")
```



## 2. Seaborn

### Understanding Seaborn plots

Extensive personalization of a plot such as combining different graphs requires the fig/ax syntax:

```
spectre = POKEDDEX.query("Type == 'Spectre'")
```

```
combat = POKEDDEX.query("Type == 'Combat'")
```

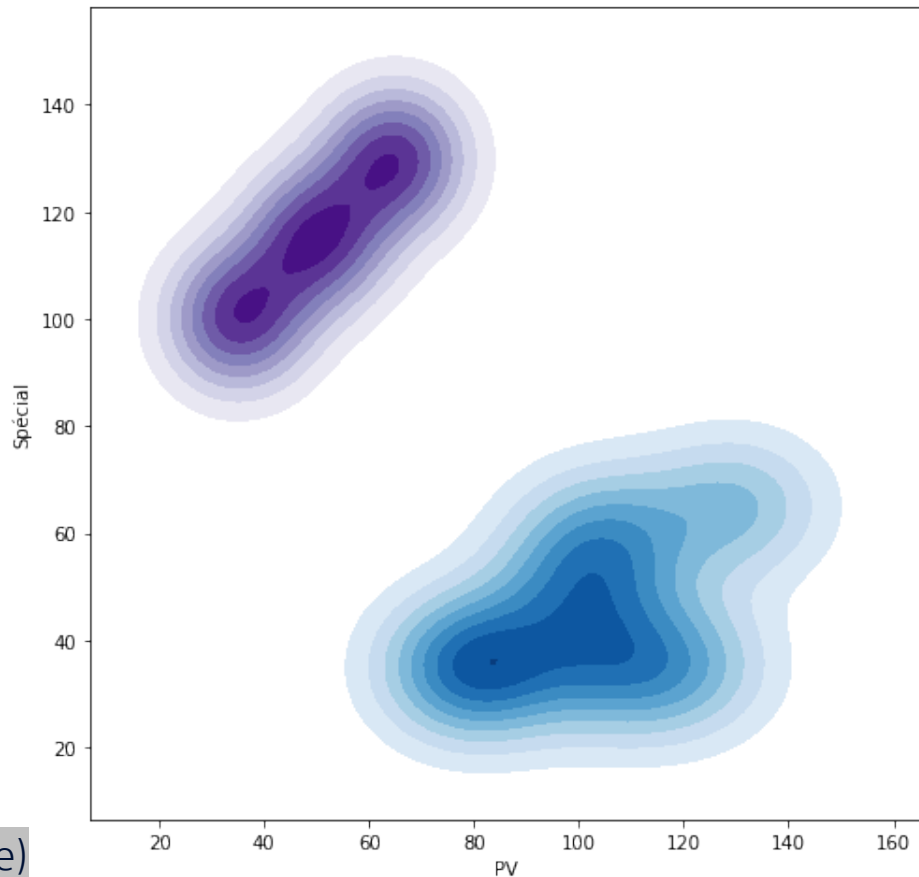
```
f, ax = plt.subplots(figsize=(8, 8))
```

```
ax = sns.kdeplot(spectre.PV, spectre.Spécial,
```

```
cmap="Purples", shade=True,  
shade_lowest=False)
```

```
ax = sns.kdeplot(combat.PV, combat.Spécial,
```

```
cmap="Blues", shade=True, shade_lowest=False)
```



## 2. Seaborn

### Takeaway

Seaborn :

- is built on top of Matplotlib and is not meant to supersede it but rather to offer an alternative to it when working with DataFrames
- may indeed offer simple APIs for some basic operations
- will usually require the use of PyPlot fig/ax objects or other Matplotlib APIs for thorough tweaking

# References

- <https://seaborn.pydata.org/tutorial.html>
- <https://seaborn.pydata.org/api.html>
- <https://elitedatascience.com/python-seaborn-tutorial>