Python for Data processing

Lecture 2:

Jupyter, Arrays, tensors and computations - Part II

Kosta Rozen

This lecture

- plotting intermezzo
- advanced NumPy
- efficient NumPy

Matplotlib: plotting with Python

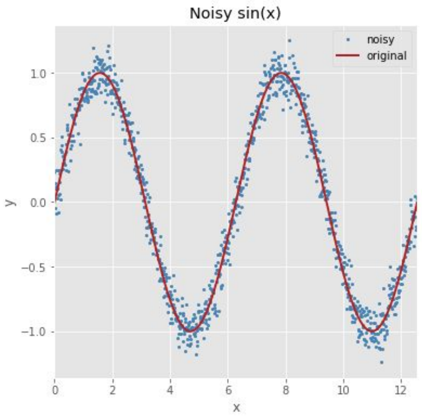
Matplotlib

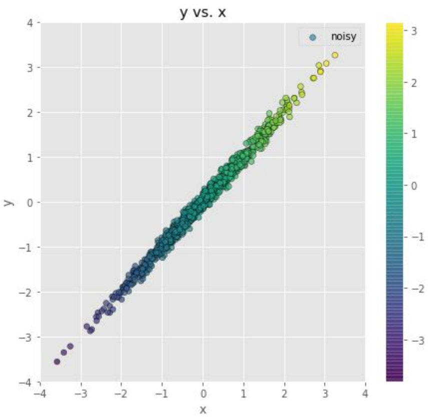
Plotting library for Python:

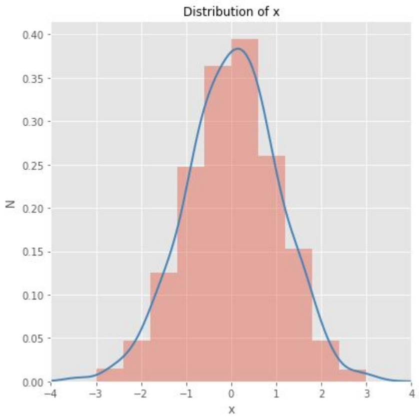
- relies on NumPy arrays (we'll see, that it nicely works with Pandas as well)
- a lot of plotting options, output formats and UI toolkits
- publication ready images
- low level

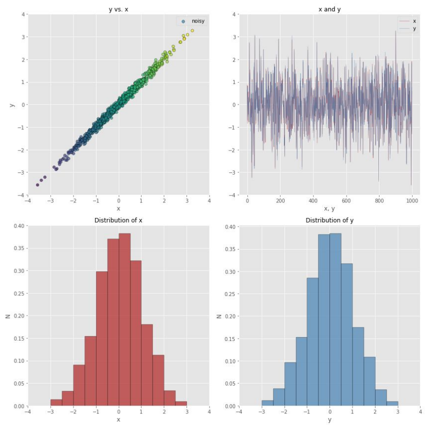
We will start using it when learning PyTorch

Matplotlib


Noisy sin(x)
noisy
original
x
y


y vs. x
noisy
x
y


Distribution of x
x
N


y vs. x
x and y
Distribution of x
Distribution of y

Matplotlib figures

It all starts with **Figure** (explicitly or implicitly)

Figure:

- can have size
- multiple subplots
- other properties

→let's try it out!

Matplotlib plots: line

plt.plot to plot simple y(x) for two (or single!) arrays:

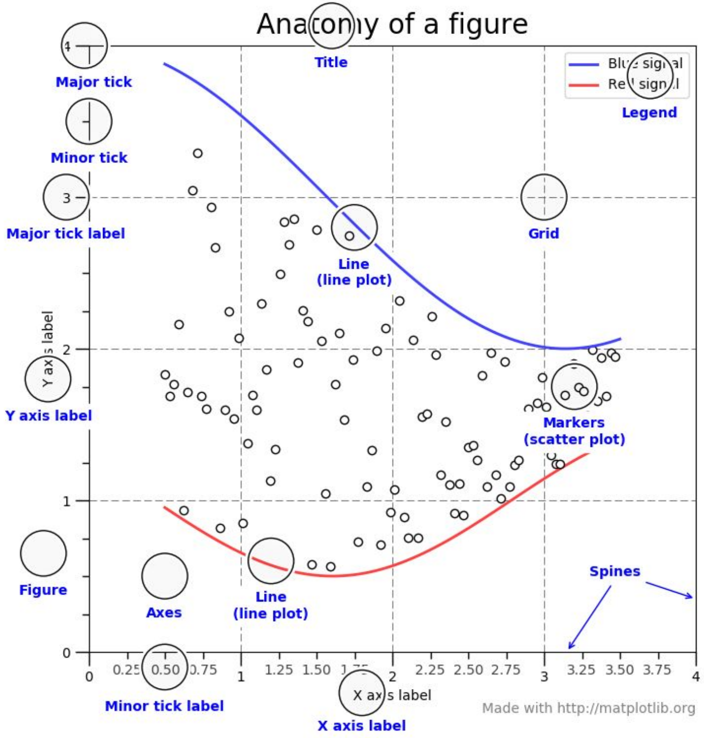
- you can change line appearance
- plot multiple lines on a single figure (axes)

Hint:

- use predefined styling

→let's try it out!

matplotlib figures: elements



Matplotlib plots: scatter

plt.scatter to plot simple (x, y) pairs:

- you can change markers appearance
- point-wise color and size

→let's try it out!

Matplotlib plots: histogram

plt.hist to plot distribution of x:

- select bin size and number of bins
- stacked histograms

→let's try it out!

Matplotlib plots: box plots

plt.boxplot to get another view of variable(-s) distribution

→let's try it out!

Matplotlib figures: subplots

Each figure can contain multiple plots:

- use plt.subplot(rows, columns, plot_number)
- or ax = fig.add_subplot(rows, columns, plot_number)

→let's try it out!

Seaborn

Stylish plotting:

- based on matplotlib
- many additional types of plots
- styling

→let's try it out!

NumPy from inside

ndarray from inside

NumPy array is a **container**:

```c
typedef struct PyArrayObject {
        PyObject_HEAD
        char *data; /* Block of memory */
        PyArray_Descr *descr; /* Data type descriptor */
        /* Indexing scheme */
        int nd;
        npy_intp *dimensions;
        npy_intp *strides;
        /* Other stuff */
        PyObject *base;
        int flags;
        PyObject *weakreflist;
} PyArrayObject;
```

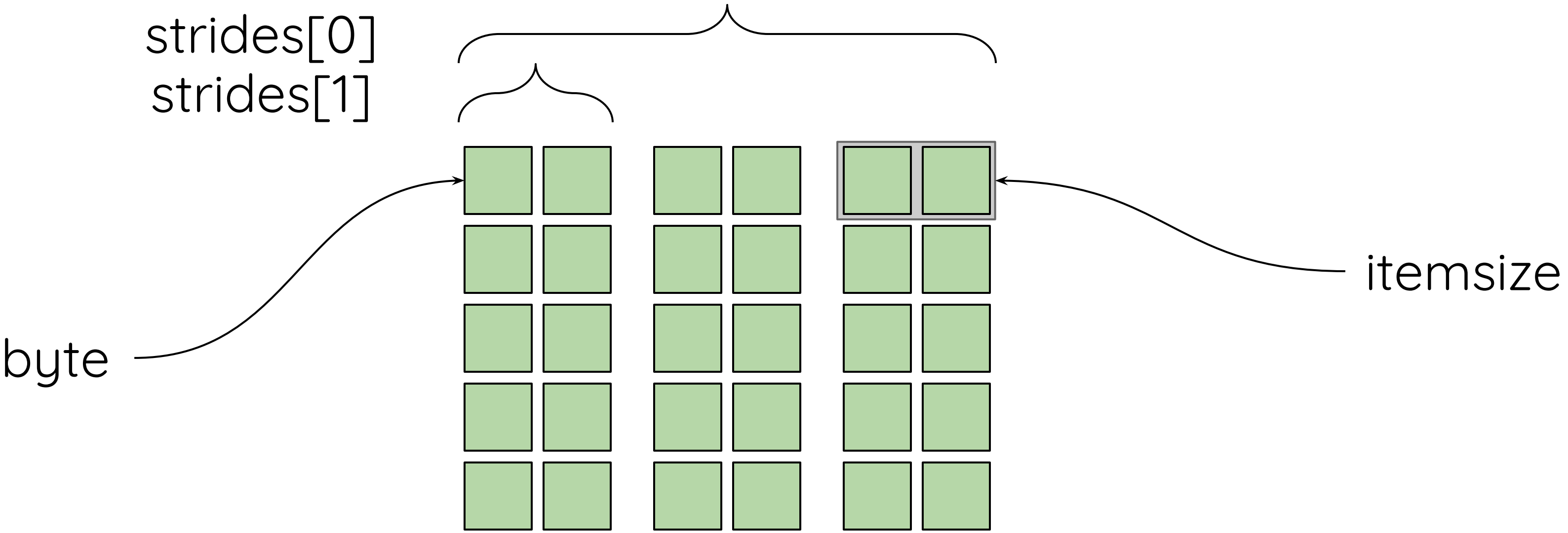
ndarray from inside

NumPy:

- stores data as a flat chunk of memory
- have indexing scheme on top of that (dimensions and type)
- knows how to step through the memory
- knows the origin

→let's try it out!

ndarray from inside



→let's try it out!

shape = (5, 3) (**elements**)
strides = (6, 2) (**bytes**)
(i, j) = i * strides[0] + j*strides[1]

Consequence #1: cache effects

Data is read from memory in chunks, not element by element

↓

Memory layout may impact performance

→let's try it out!

Consequence #2: copies

Copies are costly

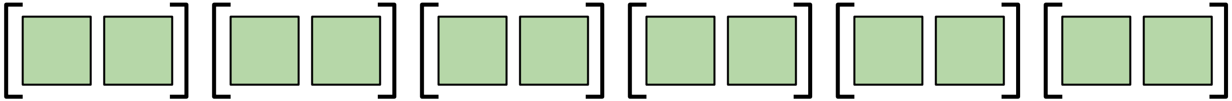
↓

Look at inplace operations

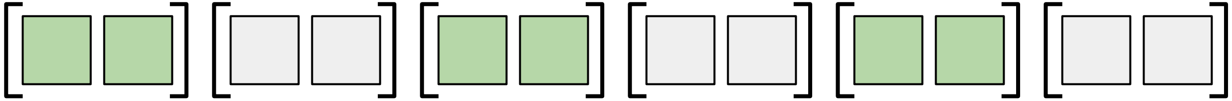
→let's try it out!

View vs. copy

`arr`$^{\texttt{np.int16}}$



`arr[::2]`



`arr`$^{\texttt{np.int16}}$

shape = (6,) **strides** = (2,)

(i) = i * 2

`arr[::2]`

shape = (3,) **strides** = (4,)

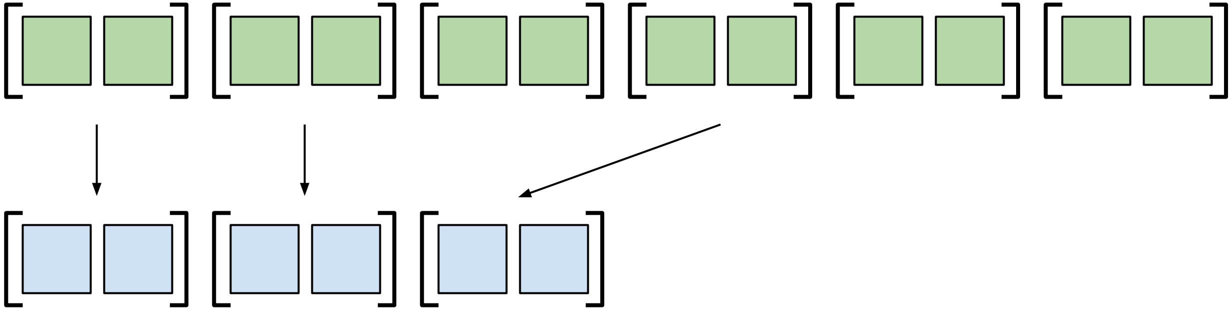
(i) = i * 4

`arr[::2]`
is a **view**

View vs. copy

`arr`$^{\texttt{np.int16}}$

`arr[[T, T, F, T, F, F]]`
is a **copy**



`arr`$^{\texttt{np.int16}}$

shape = (6,) **strides** = (2,)
(i) = i * 2

`arr[::2]`

shape = (3,) **strides** = (2,)
(i) = i * 2

`arr[::2]`
is a **view**

Broadcasting

What if input arrays have **different shapes**?

- should we reshape them to common shape before applying some **ufunc**? **No.**
- if possible, ufunc adds missing dimensions and loop through them with stride=0

→let's try it out!

Efficient NumPy

- use indexing wisely
- avoid copies whenever possible
- use inplace operations whenever possible
- use broadcasting whenever possible
- avoid loops
- **vectorize**

Still slow?

What if the code is so complex, it **gains little** from all the remedies above?

We have tools for that also.

Cython, Numba → optional assignment

What we’ve learned

- basic and advanced NumPy
- some plotting

Next time

- PyTorch: tensor (and deep learning) framework

Assignment

- more NumPy: broadcasting, etc.

questions?