Python for Data processing

Lecture 1:
Jupyter, Arrays, tensors and computations - Part I

Kosta Rozen

Lecturer

Kosta Rozen

- Product Analytics Lead @ Waze
- Data Scientist & Analytics Lead @ Intel
- Lecturer @ BGU
- M.Sc. in Industrial Engineering
- Scrum Master
- PMP (Project Management Professional)
- Y-Data Alumni

Our TA

Anatoly Bardukov

applied math from HSE

senior full stack Developer at **Nvidia**

author at [Industrial Machine Learning (Coursera)]

Course logistics

Each week:

- lecture slides (released on Friday, lecture based)
- Jupyter notebooks (released on Friday, topic based)
- one graded assignment (released on Friday, deadline on Saturday in a week)
- one or more optional materials (released during a week)

+ online discussions (Slack)
+ office hours, Q's during lectures, quizzes

Course logistics

Graded assignments:

- we run them with Papermill
- they should not fail
- partially autograded
- you have one week

Course logistics

Study groups (pairs, for both Python and Probability courses)**:**

- <u>Oct ??:</u> form study groups (if you don't, we assign you randomly)
- <u>Oct ??:</u> tell us, if randomly assigned group is not working for you (logistics, whatever)
- do homework together, discuss, have fun

Why Python?

Why Python?

Python is:

- simple enough
- flexible
- general purpose
- has huge ecosystem for DS and ML

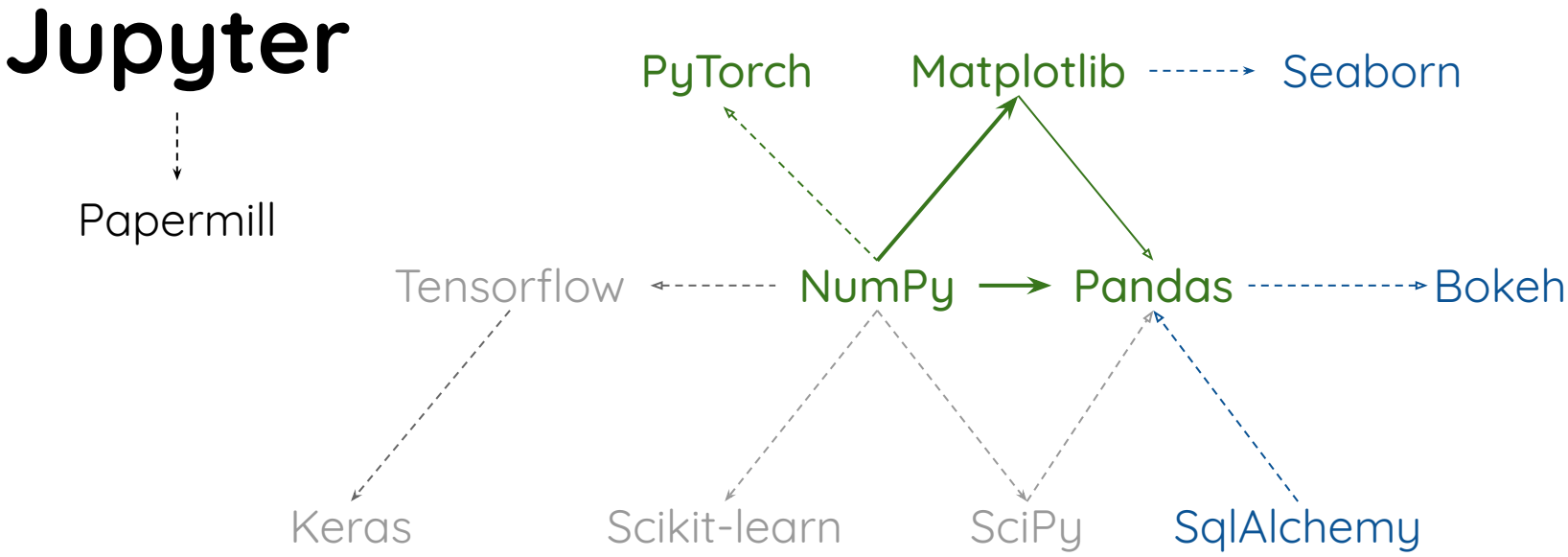
Why Python?

But it's interpreted! **Isn't it slow?**

Short answer is: **No. It's ok.**

Long answer is: **No, it's not slow,** cause all the heavy lifting is done in **C/C++/Fortran** under the hood. Thank you, Python C API!

Python ecosystem for ML



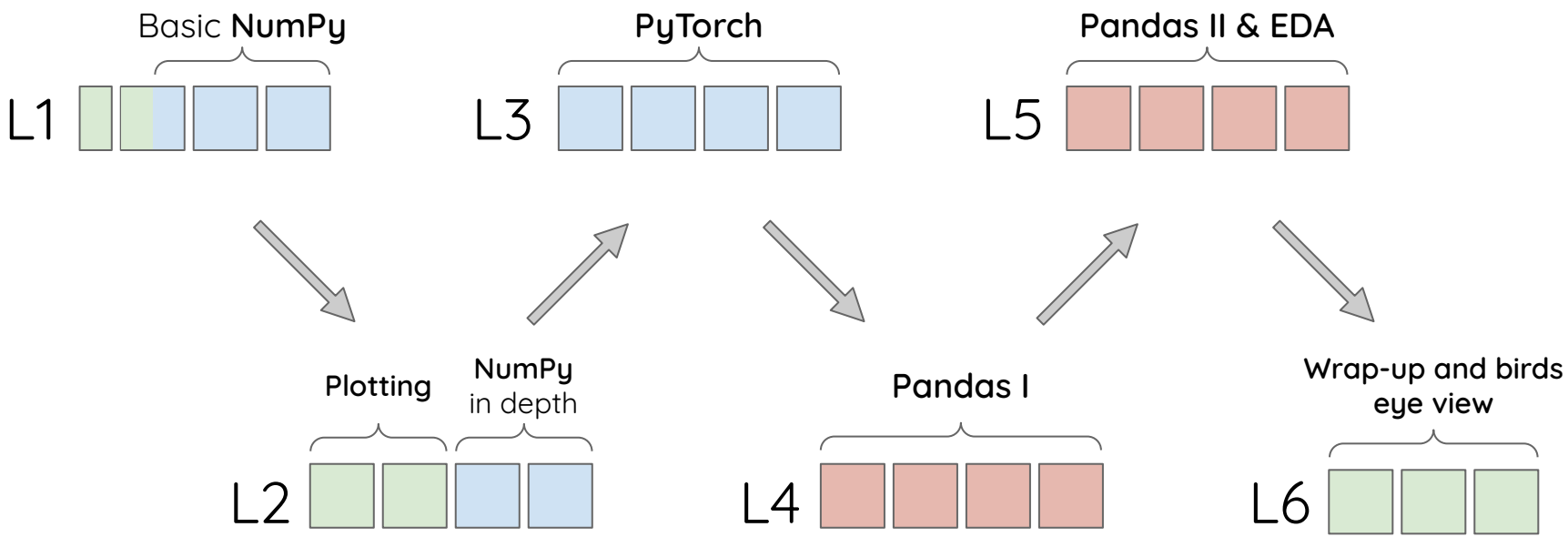
Syllabus

Main parts of the course are:

- NumPy (+ PyTorch as a topping)
- Matplotlib (+ Seaborn and Bokeh)
- Pandas
- basics of **exploratory data analysis**
- tools for **reproducibility**, project structuring and more

Syllabus

1 unit, about 50 minutes



→let’s try it out!

(i.e. we’re going to switch to notebook, terminal or whatever)

Resources worth reading

Python for Data Analysis by Wes McKinney

PyData YouTube channel

From Python to Numpy by Nicolas P. Rougier

Scientific Computing in Python by Sebastian Raschka

...and there will be more along the way.

This lecture

- environment review: **Jupyter**
- high-performance Python arrays: **NumPy**
- creating and indexing arrays, linear algebra and more

Jupyter and other tools

Jupyter

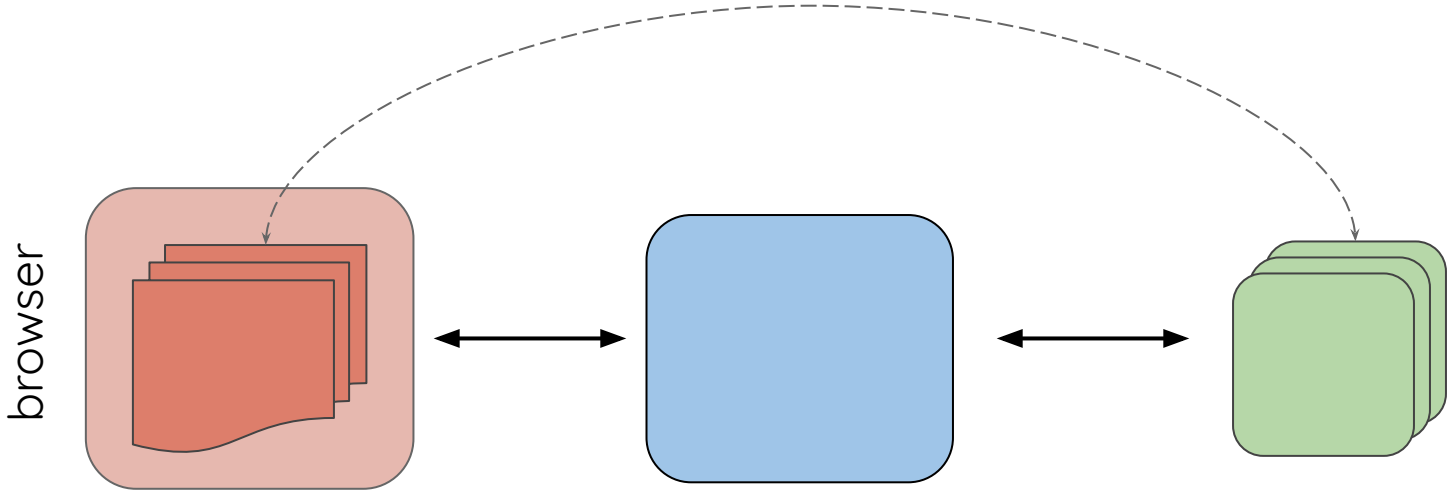
web-based interactive environment

extremely suitable for exploration

originate in IPython project, but is largely **language agnostic now**

→let's try it out!

Jupyter



notebooks

Jupyter
server

kernels

Hosted notebook envs

Hosted:

- Google Colab
- Yandex DataSphere
- Nextjournal
- Vertex AI (GCP)
- more

Jupyter: a bit of safety

Jupyter server, when running on a cloud/remote machine **should not be open** to the outside world.

Use password and **https** or **ssh** tunneling.

NumPy: basics of high performance arrays

Why NumPy?

Pure Python:

- is slow (everything works through Python interpreter)
- lacks strong numerical infrastructure (**math** module? really?)

NumPy:

- fast (it's C/C++/Fortran and battle tested BLAS etc. implementations)
- a lot of routines for virtually any generic use case

ndarray

- core data structure in **numpy**
- container with **known number** of elements of the **same** (known) **size**
- supports **indexing** and **vectorized** operations
- allows to **share** data
- **fundamental** for most other numerical packages (Pandas, Matplotlib, SkLearn, etc.)

Creating arrays: naive

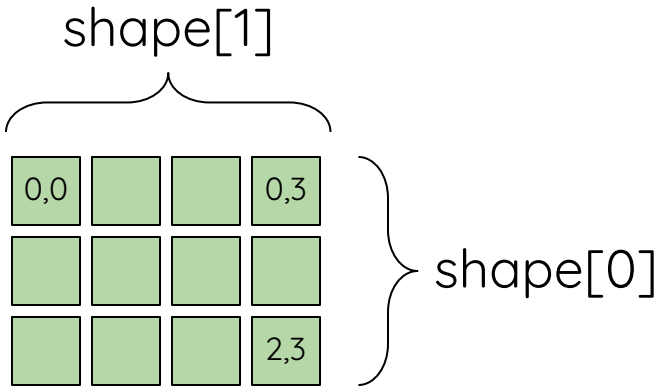
- let's use **np.ndarray** directly!
- Or, better, let's create an array from Python sequence

→let's try it out!

Array: basic properties

- shape: `arr.shape`
- type of elements: `arr.dtype, arr.itemsize`
- number of dimensions: `arr.ndim, ar.size`

Python list vs. ndarray



len = 5
element size = ?

shape = (3, 4) (**elements**)
ndim = 2
element size = fixed

Creating arrays: advanced

We rarely need to create arrays from Python sequences (and **np.ndarray** should be avoided altogether)

Instead we need:

- arrays of **specific structure or type**
- arrays, filled with some **numeric pattern**

→let's try it out!

Array: basic indexing

NumPy arrays support slicing syntax:

- **a[0, 1]** is ok
- **a[0, :3]** is ok
- **a[1:, :3]** is ok
- **a[0, :-9]** and even this is also ok

→let's try it out!

Array: boolean and fancy indexing

Basic indexing may be insufficient: (and for large arrays it usually is)

- boolean: **a[boolean_mask]**
- fancy: **a[int_idx]** (remember about **np.where**)

→let's try it out!

Array: view vs. copy

Basic indexing returns a **view**(same memory is used), fancy and boolean indexing return a **new array**.

But you can mix them.

And it's a bit different for **setting** values.

→let's try it out!

Array: changing shape

Sometimes we need to **change array shape**:

- flat vector to row or column vector
- row vector to column vector
- transpose

```
arr.reshape, np.expand_dims, arr.T
arr.flatten, arr.ravel
```

Array: changing type

Sometimes we need to **change array type**:

- to create integer mask
- to reduce memory consumption
- to conform with external API

Array: stack

Sometimes we need to **combine** multiple arrays:

- to create a matrix from several vectors
- to combine results from different sources

→let's try it out!

Array: universal functions

Fast, vectorized functions, operating element-wise.

- **unary: np.sum**, **np.mean** and so on
- **binary: np.maximum**, **np.logical_and** and so on

Array: universal functions

ufuncs support common arguments:

- **axis**: operate over this axis
- **where**: masking
- **keepdims**: do not drop reduced dimensions

NumPy: basics of linear algebra

Linear algebra: basics

Linear algebra works on vectors (1D), matrices (2D) and tensors (>3D).

Typical operations:

- dot-product of vector and matrix
- matrix operations: invert, get eigenvalues
- decompositions

Linear algebra: np.linalg

Entry point for linear algebra operations:

- **np.linalg.inv**, **np.linalg.det**, **np.linalg.trace**
- **np.linalg.eig**
- matrix decompositions

→let's try it out!

Linear algebra: eigenvalues and eigenvectors

The simplest possible decomposition for square matrices

Plays huge role in many algorithms
(often in extended ways)

→let's try it out!

Reading and writing data with NumPy

Reading and writing NumPy arrays

Array can be saved to a file with **np.save**:

- binary format
- read with **np.load**

→let's try it out!

Reading and writing NumPy arrays

Multiple array can be saved to a single file with **np.savez**:

- it's zip, but uncompressed
- use **np.load** to read it (return dict-like object, no data is actually read)

→let's try it out!

Reading and writing text files

np.loadtxt and **np.savetxt**: to read text files (mostly CSV)

But **Pandas** is much better in this!

→let's try it out!

Other formats and options

Natively or through scipy.io:

- binary data (from files)
- **mat** files
- WAV files

Using 3rd party packages:

- HDF5 (**h5py**)
- images (**skimage**, **opencv**)

What we've learned

- creating and indexing arrays
- changing array properties
- calculate with arrays
- basics of linear algebra operations
- I/O

Assignment

- Exploring NumPy: array creation, indexing
- `ufunc`'s

questions?