Python: Data

Adam Boult (www.bou.lt)

April 29, 2025

Contents

Preface

This is a live document, and is full of gaps, mistakes, typos etc.

Part I

Basic scientific libraries

Chapter 1

NumPy

1.1 Introduction

1.1.1 Introduction

1.1.2 Creating NumPy arrays

```
import numpy as np
a = np.array([0,1,2])
b = np.array([3,4,5])
```

Can do basic element-wise operations on these.

```
import numpy as np
a = np.array([0,1,2])
b = np.array([3,4,5])
sum = a+b
minux = a-b
element_wise_product = a*b
divided = a/d
dot_product_scalar = a@b
```

You don't need to use the symbols.

```
import numpy as np
a = np.array([0,1,2])
b = np.array([3,4,5])
sum = np.add(a, b)
minus = np.subtract(a,b)
element_wise_product = np.multiply(a,b)
divided = np.divide(a, b)
dot_product_scalar = np.dot(a,b)
```

Can also define ones like this

```
np.eye(n)
np.ones(i, j, k, ...)
np.full(i, j, k, ...)
np.rand(i, j, k, ...)
np.zeroes(i, j, k, ..)
```

1.1.3 Multi-dimensional arrays

```
import numpy as np
A = np.array([[0,1],[2,3]])
transposed = A.T
determinant = np.linalg.det(A)
inverse = np.linalg.inv(A)
eigenvalues, eigenvectors = np.linalg.eig(A)
```

Note that it's .shape, not .shape()

```
import numpy as np
A = np.array([[0,1],[2,3]])
A.shape
```

We can multiply two matrices together. Note that using * would be elementwise, and probably not what is wanted.

```
import numpy as np
A = np.array([[0,1],[2,3]])
B = np.array([[4,5],[6,7]])
A@B
```

1.1.4 dtypes

```
import numpy as np
A = np.array([[0,1],[2,3]], dtype = np.int32)
```

1.1.5 Solving linear matrix equations

```
a = np.array([[1, 2], [3, 5]])
b = np.array([1, 2])
x = np.linalg.solve(a, b)
array([-1.,  1.])
```

Chapter 2

numba

2.1 Introduction

2.1.1 Introduction

jit compilation for numpy?

Chapter 3

Matplotlib

3.1 Introduction

3.1.1 Introduction

designed around numpy but can use math module

Chapter 4

SciPy

4.1 Statistics

4.1.1 Introduction

Can do stats, but stats models uses and builds on SciPy.stats, and is generally a better choice to use.

4.2 Integration

4.2.1 Introduction

4.3 Optimisation

4.3.1 Introduction

scipy.optimize.fsolve(func, x0)

x0 is starting guess

4.4 Fast Fourier transforms

4.4.1 Introduction

Part II

Other scientific libraries

Chapter 5

pandas

5.1 Introduction

5.1.1 Introduction

built on numpy.

get pandas size

sort pandas

pandas: subsetting tables: .loc[] + selecting series from df + selecting row from df selecting element from series/row

pandas: apply

missing value, joins

pandas data structures

Chapter 6

seaborn

6.1 Introduction

6.1.1 Introduction

Works better with pandas natively than matplotlib?

Abstraction around matplotlib.

Chapter 7

sklearn

7.1 Introduction

7.1.1 Introduction

uses numpy, matplotlib/plotly, pandas, scipy

7.1.2 xgboost

7.1.3 Dimensionality reduction including PCA

7.1.4 Model selection including grid search and cross-validation

7.1.5 k-means clustering

7.1.6 Support vector machines (SVMs)

7.1.7 Regressions

7.1.8 trees, forests, gradient boost trees, xgboost

Chapter 8

biopython

8.1 Introduction

8.1.1 Introduction

Builds on numpy.

Chapter 9

plotly

9.1 Introduction

9.1.1 Introduction

Chapter 10

statsmodels

10.1 Introduction and datasets

10.1.1 Introduction

builds on numpy, pandas, matplotlib, scipy

10.2 OLS

10.2.1 Introduction

```
import statsmodels.api as sm

results = sm.OLS(y, X).fit()

print(results.summary())
```

10.2.2 Robust standard errors

10.3 Time series

10.3.1 Introduction

Part III

Tensor libraries

Chapter 11

tensorflow and keras

11.1 Introduction

11.1.1 Introduction

keras is "front end" for tensorflow

XLA (Accelerated linear algebra)

Gradient accumulation?

Chapter 12

PyTorch

12.1 Introduction

12.1.1 SORT

basic creation and training of deep nn. section on convolution. section on transformers. also for keras and tensorflow. and jax.

12.1.2 From NumPy to PyTorch

Basic functions are the same.

```
import numpy as np
import torch
a_np = np.array([[0,1],[2,3]])
b_np = np.array([[4,5],[6,7]])
a_torch = torch.tensor([[0,1],[2,3]])
b_torch = torch.tensor([[4,5],[6,7]])

a+b
a-b
a*b
a/b
a@b
```

Tensors can also be defined using NumPy arrays

```
tensor = torch.from_numpy(a_np)
```

Where @ is elementwise multiplication and @ is matrix multiplication.

As with NumPy we can define arrays like this.

```
torch.eye(n)
torch.ones(i, j, k, ...)
torch.full(i, j, k, ...)
torch.rand(i, j, k, ...)
torch.zeroes(i, j, k, ..)
```

We can also define arrays of specific dtypes, as in NumPy.

```
a_torch = torch.tensor([[0,1],[2,3]], dtype =torch.float32)
```

12.1.3 Attaching CUDA cores to PyTorch tensors

If tensors are on CUDA cores then the CUDA cores will be used rather than the CPU.

Can check if CUDA is available:

```
torch.cuda.is_available()
```

Can move a tensor to the GPU.

```
tensor_cpu = torch.randn(3,3)
tensor_gpu = tensor_cpu.cuda()
# or tensor_gpu = tensor.cpu.to("cuda")
tensor_back_to_cpu = tensor_gpu.cpu()
# or tensor_back_to_cpu = tensor_gpu.to("cpu")
```

The .cuda() function can take an integer if there are multiple GPUs available.

Whole models can also be moved to the GPU. model.cuda()

We can be dynamic

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
tensor = torch.randn(3, 3).to(device)  # This will be on GPU if CUDA is available, otherwise
model = SomeNeuralNetwork().to(device)  # Move model to the same device
```

12.2 Neural networks in PyTorch

12.2.1 Defining feedforward networks

nn.Module

nn.sequential

nn.Linear

nn.ReLU

nn.Softmax

12.2.2 Backpropagation

12.2.3 Data

torch.utils.data.DataLoader

torch.utils.data.Dataset

torchvision (images)

torchtext

datasets available

torchaudio

12.2.4 PyTorch and autograd

12.2.5 JIT compiling on PyTorch

12.2.6 XLA and TPUs with PyTorch

12.2.7 fast.ai

fast.ai is a thing that sits on top of pytorch

Chapter 13

jax

13.1 Introduction

13.1.1 Introduction