NumPy Matplotlib SciPy

For ML Techniques

Agenda

NumPy

- Vectors
- Matrices
- NumPy arrays
- Sampling from Distributions

SciPy

- Unconstrained optimisation
- Constrained optimisation

Matplotlib

- Plotting simple curves
- Bar plots, histograms, scatter plots
- Contour plots, 3D plots

Process

Sequence

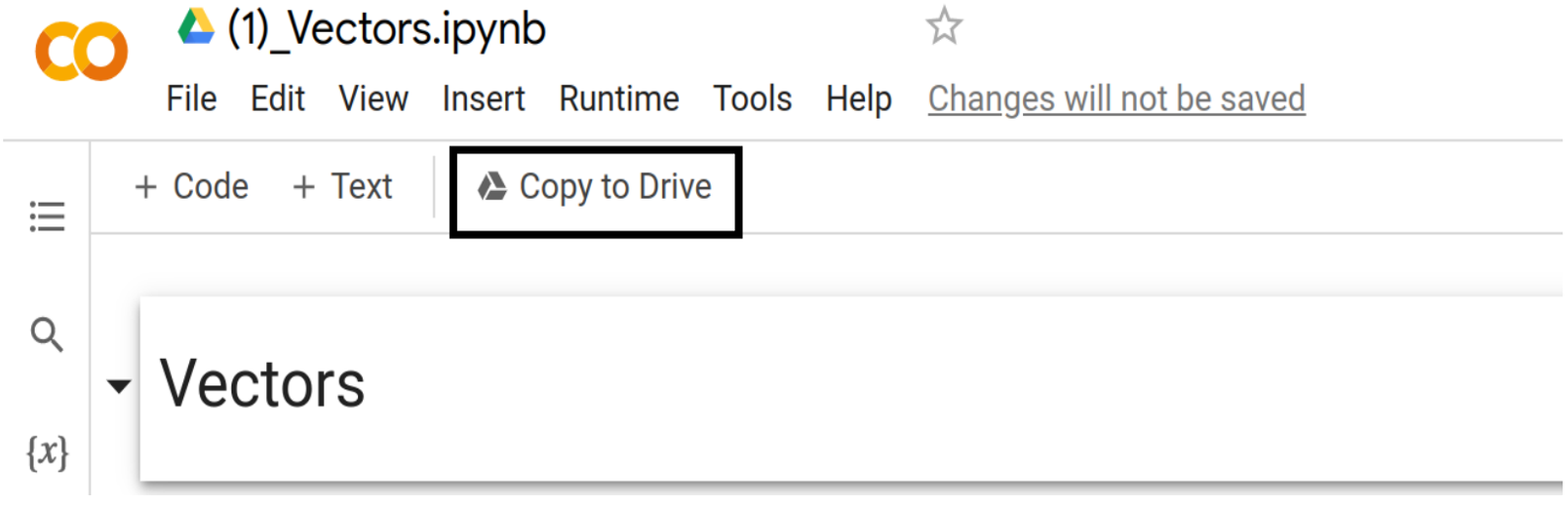
(1) NumPy
(2) Matplotlib
(3) NumPy
(4) Matplotlib
(5) NumPy
(6) Matplotlib, SciPy

Type of Session

- Code with instructor
- Code demo + QA

Colabs and Slides

- http://tiny.cc/mlt_workshop
- We will go from (1) to (6)
- Open colab and copy it to your drive
- Slides will be uploaded tonight



NumPy, Matplotlib, SciPy

NumPy	• Created in 2005 • Open source • Current version: 1.23.0	Linear Algebra • Vectors • Matrices
Matplotlib	• Created in 2003 • Open source • Current version: 3.6.2	Visualising • data • models
SciPy	• Created in 2001 • Open source • Current version: 1.9.3	Optimisation

NumPy - still relevant?

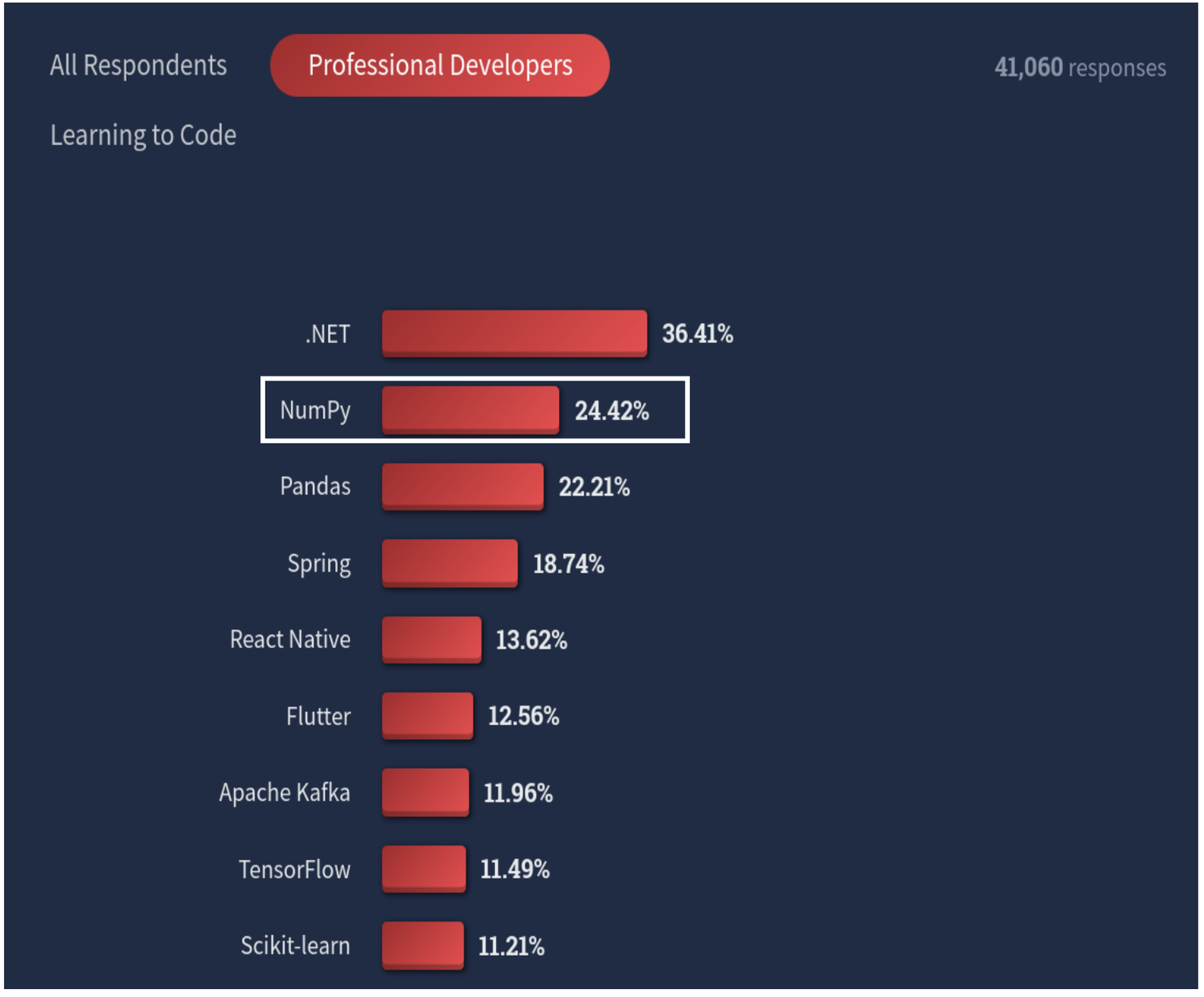
Source:

[Stack Overflow Developer Survey]

NumPy - still relevant?

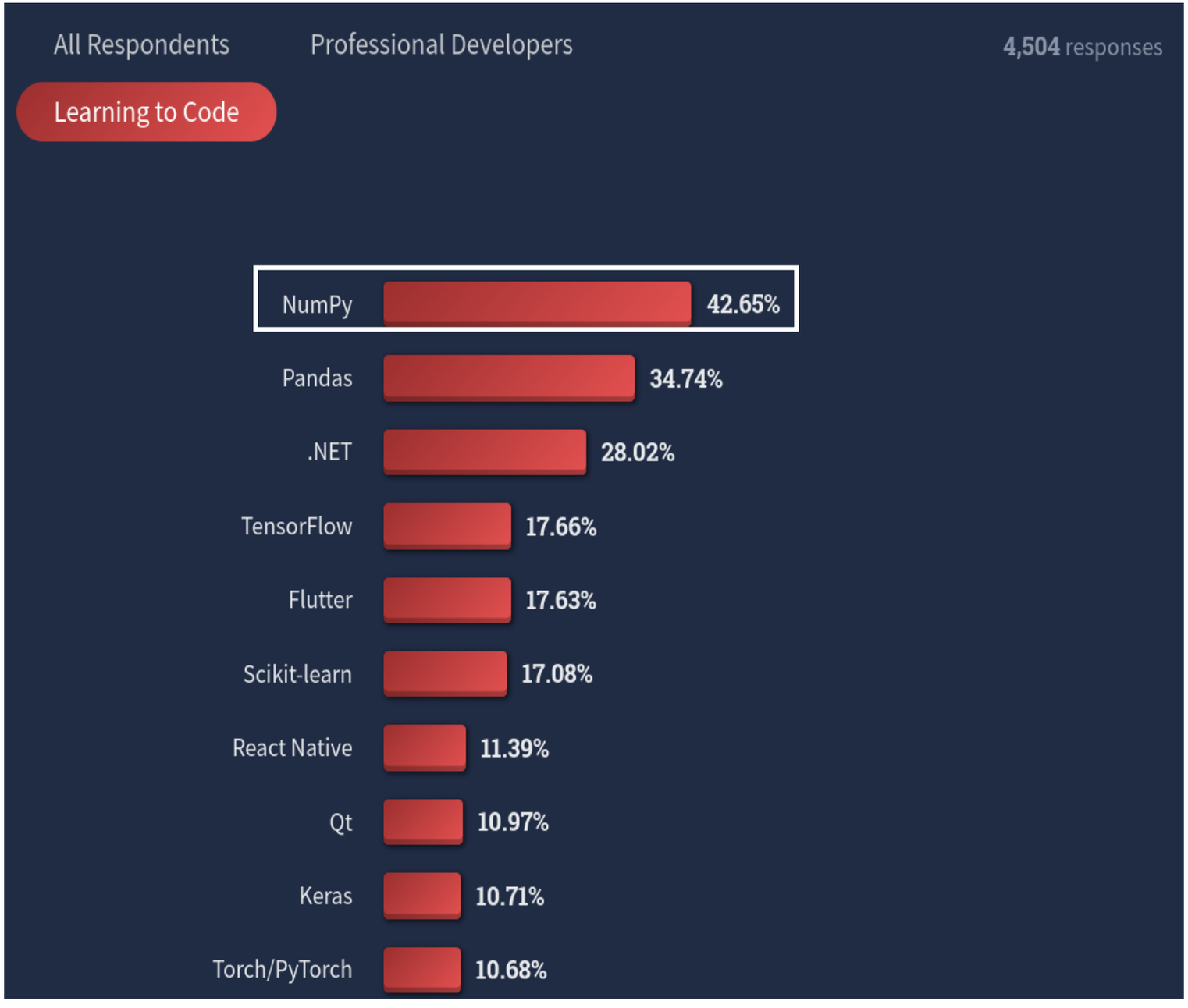
Most Popular Technologies

Source:
[Stack Overflow Developer Survey]



NumPy - still relevant?

Most Popular Technologies



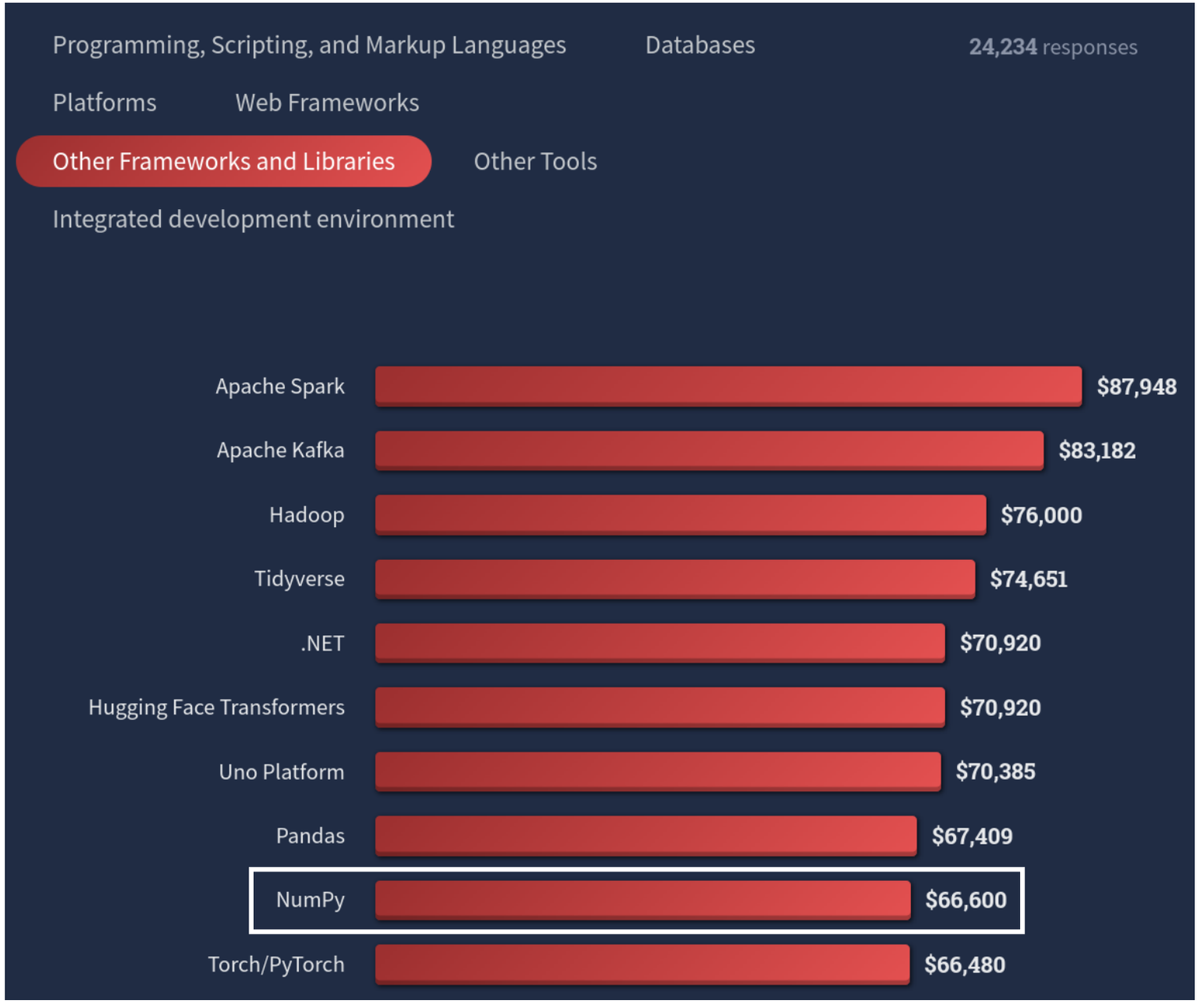
Source:

[Stack Overflow Developer Survey]

NumPy - still relevant?

Top Paying Technologies

Source:
[Stack Overflow Developer Survey]



Vectors

Vectors

$$\mathbf{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

sample

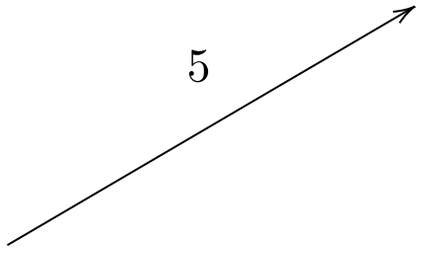
example

---

practical

Vectors

$$\mathbf{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$



sample

feature-vector

example

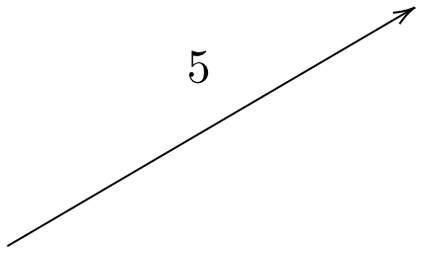
---

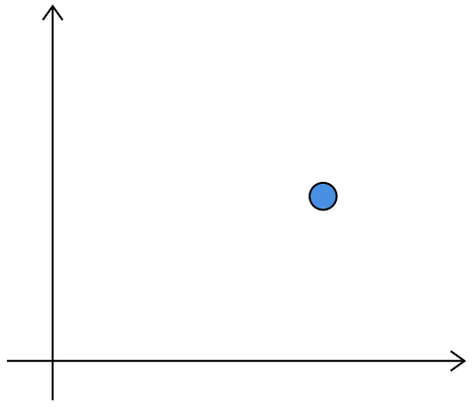
practical

abstract

Vectors

$$\mathbf{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$





sample

feature-vector

data-point

example

---

practical

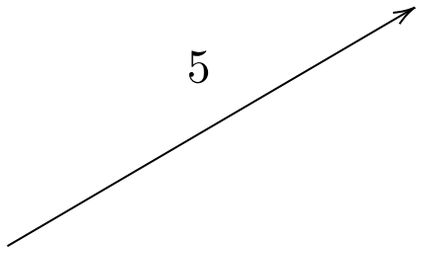
abstract

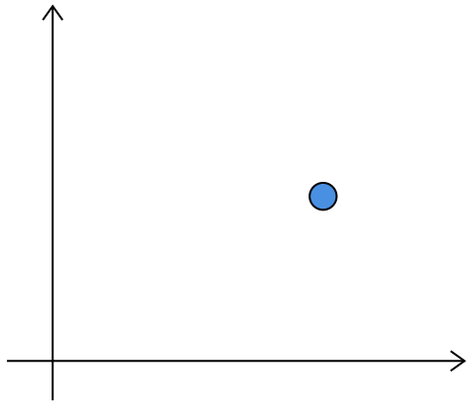
geometric

algebraic

Vectors

$$\mathbf{x} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$





```
np.array([3, 4])
```

sample

example

feature-vector

data-point

array

---

practical

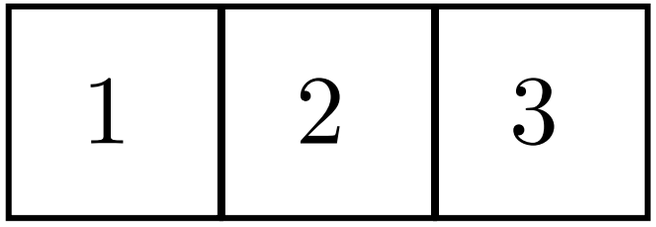
abstract

geometric

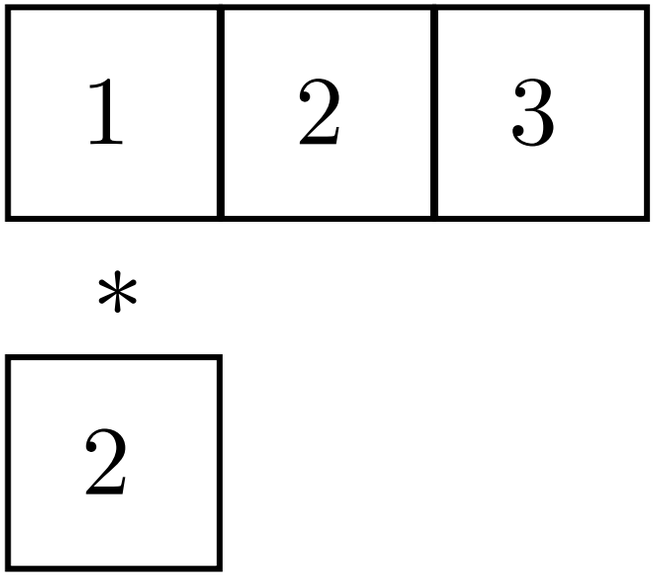
algebraic

computational

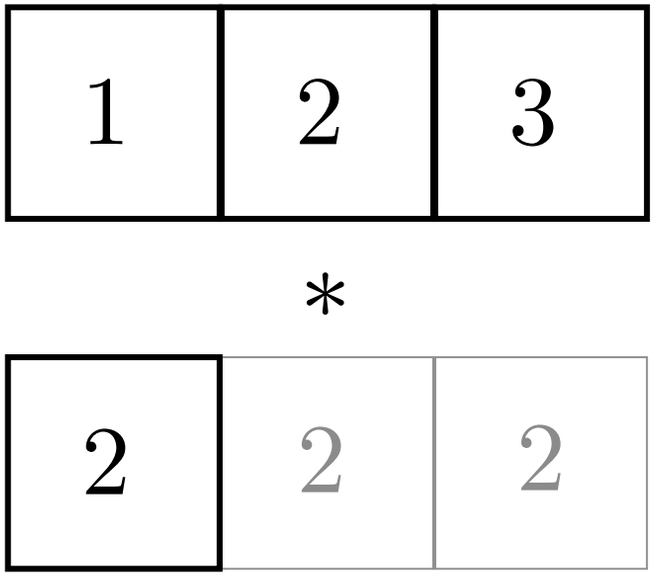
Vector | NumPy Array



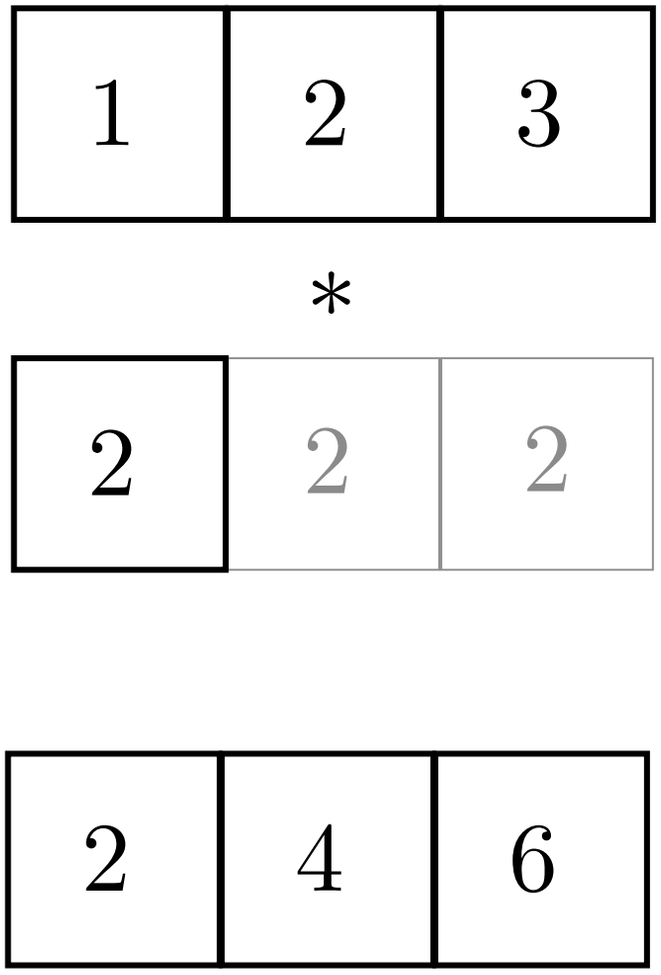
Broadcasting



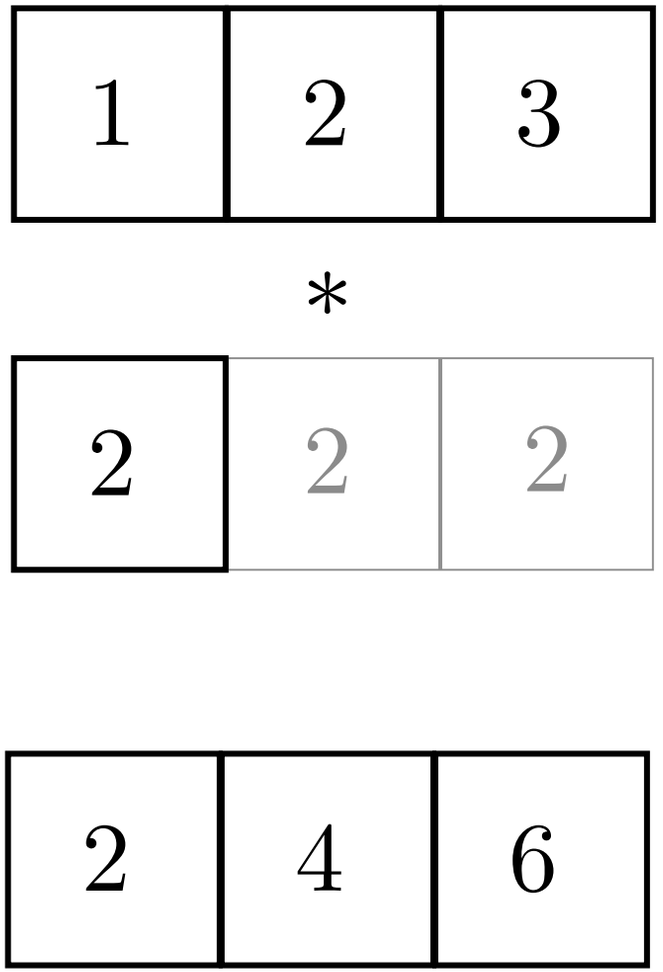
Broadcasting



Broadcasting

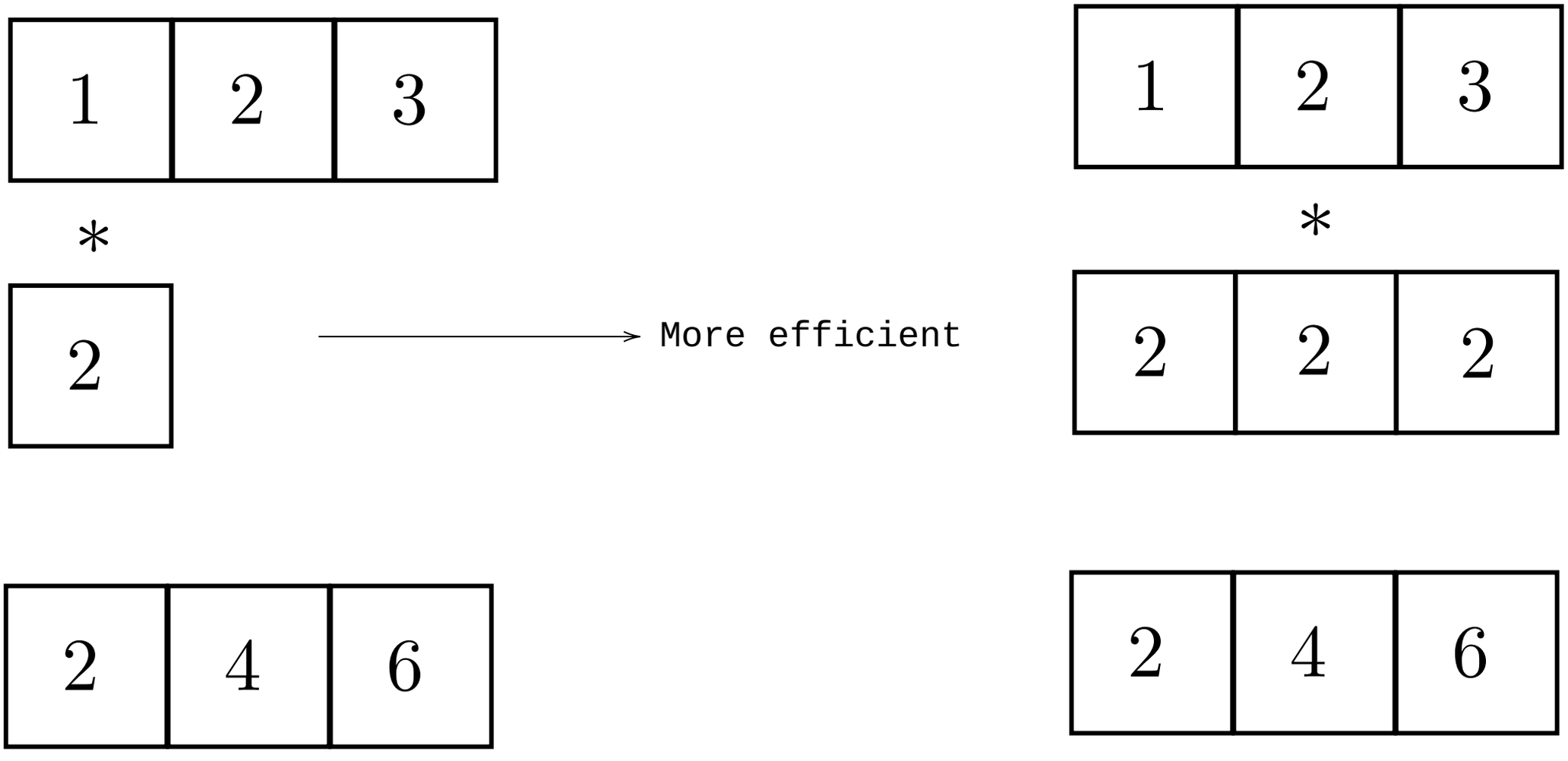


Broadcasting



- Conceptual explanation
- Not exactly what happens in NumPy

Broadcasting

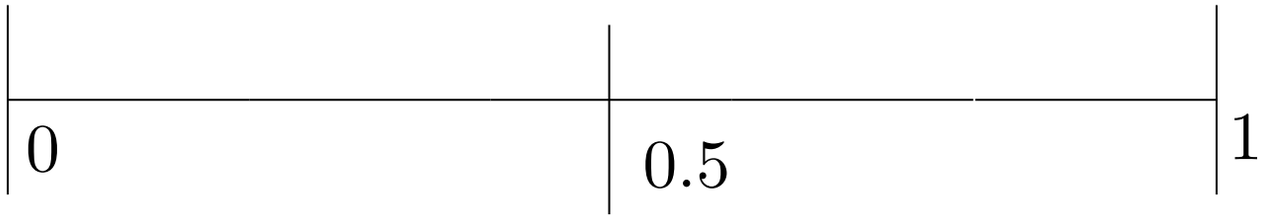


Math → NumPy (Vectors)

$\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$

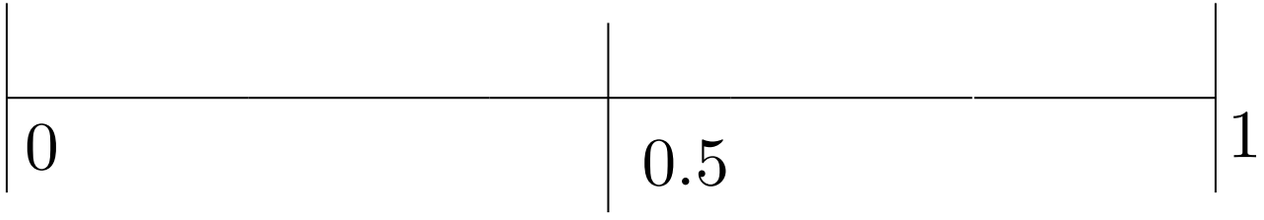
$\mathbf{x} + \mathbf{y}$	`x + y`
$\mathbf{x} \odot \mathbf{y}$	`x * y`
$\mathbf{x}^T\mathbf{y}$ or $\mathbf{x} \cdot \mathbf{y}$	`x @ y`
$a \cdot \mathbf{x} + b \cdot \mathbf{1}$	`a * x + b`
$\mathbf{0}$	`np.zeros(d)`
$\mathbf{1}$	`np.ones(d)`
$\|\|\mathbf{x}\|\|_p$	`np.linalg.norm(x, p)`

Linspace

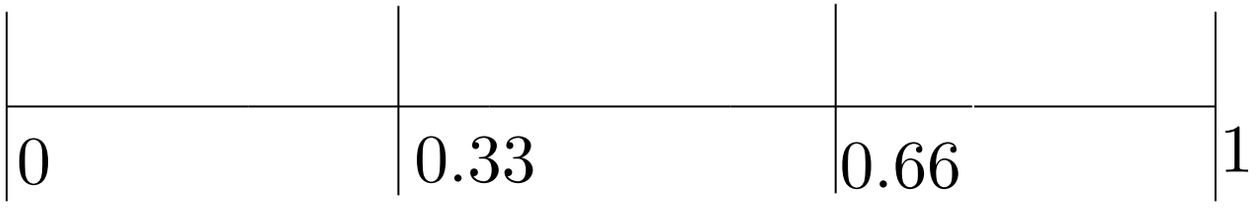



```
np.linspace(0, 1, 3)
```

Linspace

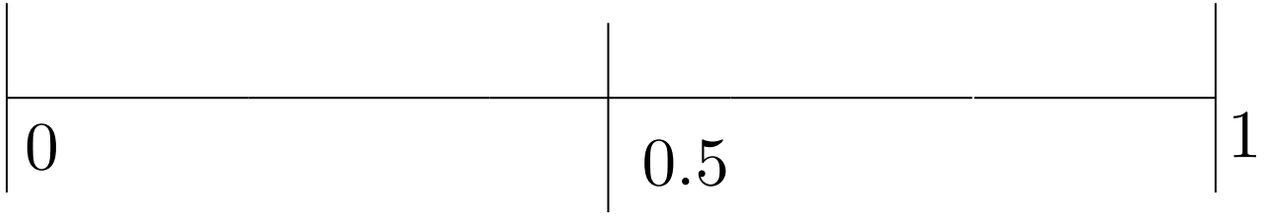



```
np.linspace(0, 1, 3)
```



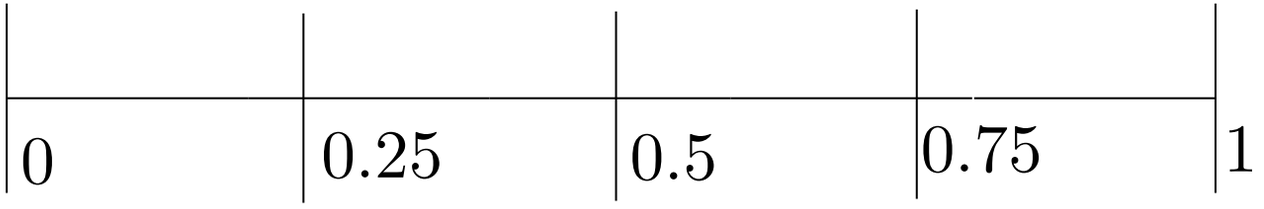

```
np.linspace(0, 1, 4)
```

Linspace



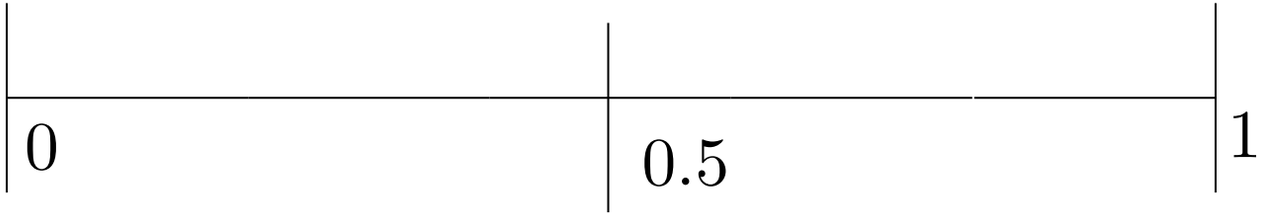

```
np.linspace(0, 1, 3)
```


```
np.linspace(0, 1, 4)
```



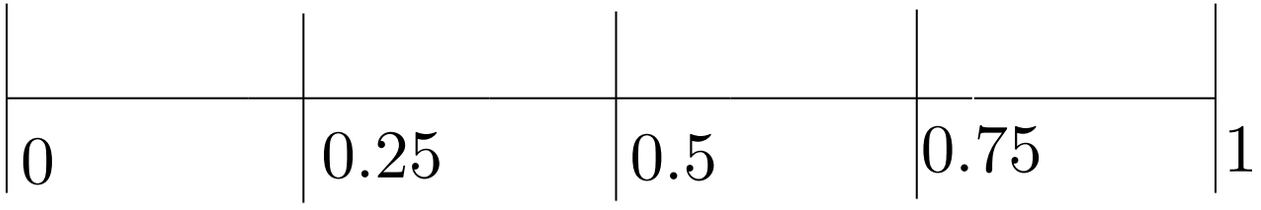

```
np.linspace(0, 1, 5)
```

Linspace

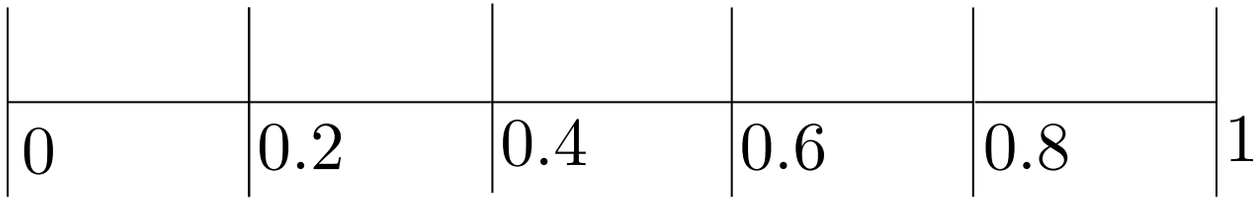



```
np.linspace(0, 1, 3)
```


```
np.linspace(0, 1, 4)
```

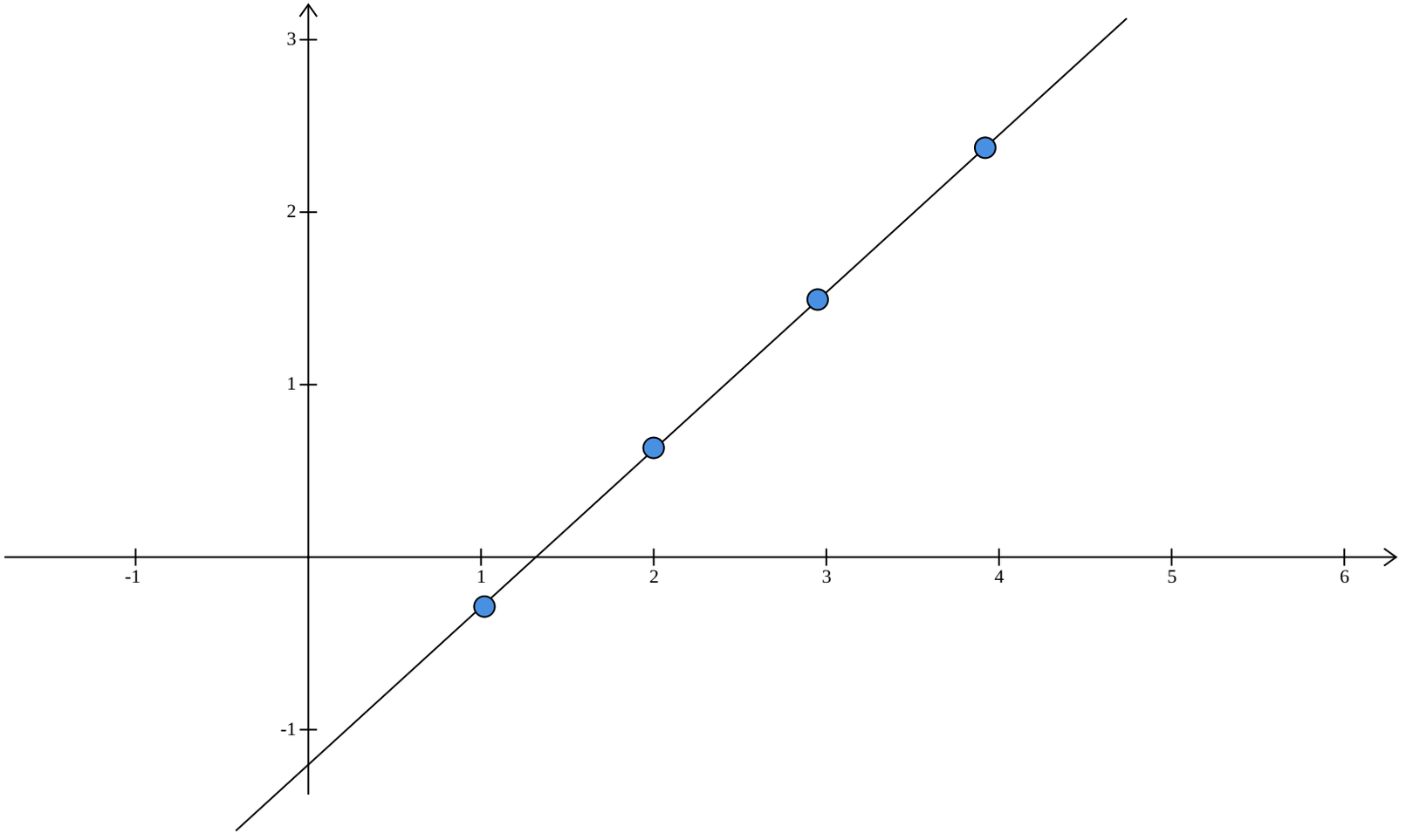



```
np.linspace(0, 1, 5)
```




```
np.linspace(0, 1, 6)
```

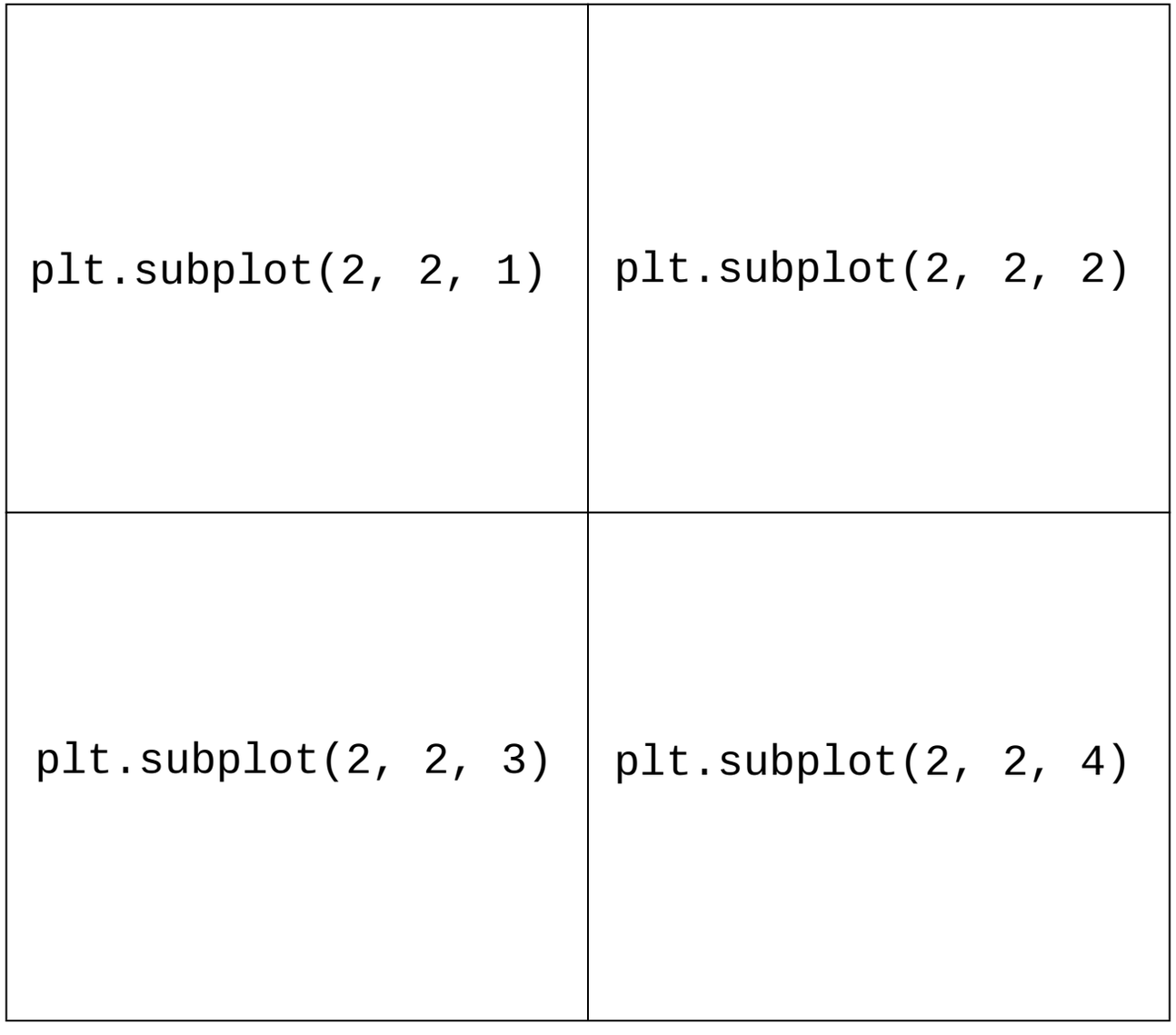
Plotting in Matplotlib



$[x_1, x_2, x_3, x_4]$

$[y_1, y_2, y_3, y_4]$

Subplots


plt.subplot(2, 2, 1)
plt.subplot(2, 2, 2)
plt.subplot(2, 2, 3)
plt.subplot(2, 2, 4)

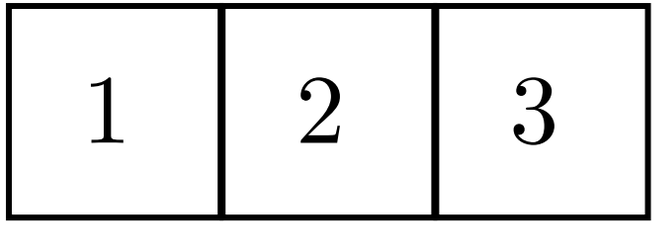
Math → NumPy (Matrices)

$\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$

$\mathbf{I} \in \mathbb{R}^{d \times d}$

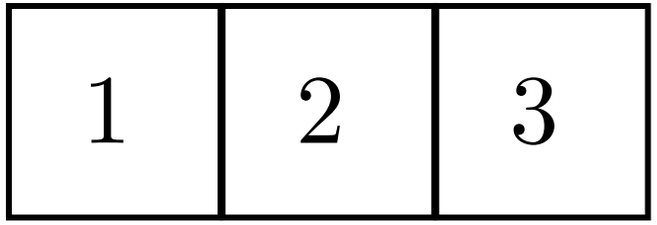
$\mathbf{A} + \mathbf{B}$	`A + B`
$\mathbf{A} \odot \mathbf{B}$	`A * B`
$\mathbf{AB}$	`A @ B`
$\mathbf{A}^T$	`A.T`
$\mathbf{I}$	`np.eye(d)`
$\text{diag}(a_1, \cdots, a_d)$	`np.diag(a_1,...,a_d)`
$\mathbf{0}$	`np.zeros((m, n))`
$\mathbf{1}$	`np.ones((m, n))`

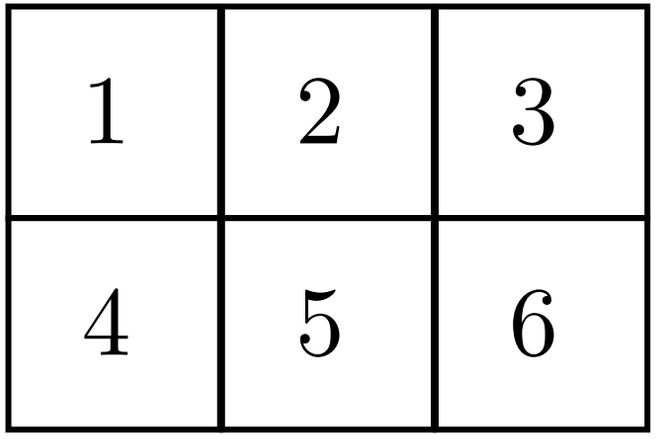
n-Dimensional Arrays




```
shape = (3, )
ndim = 1
row
```

n-Dimensional Arrays

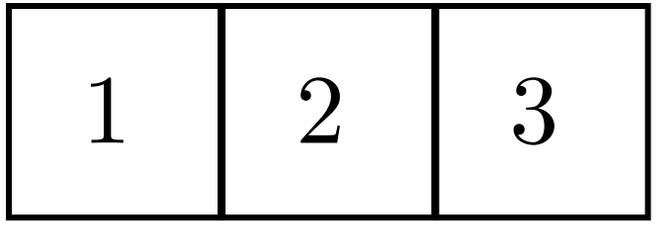




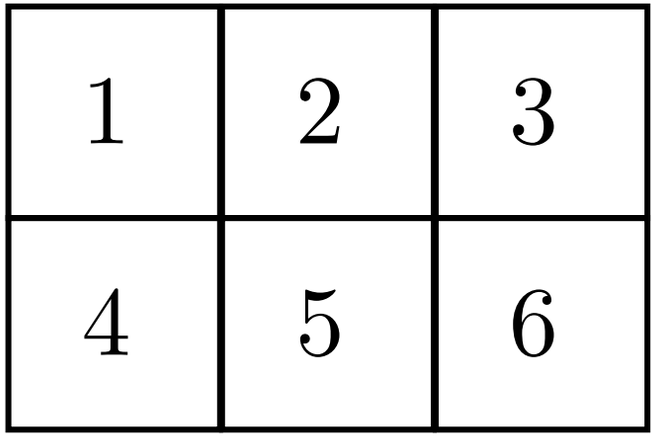

```
shape = (3, )
ndim = 1
row
```

```
shape = (2, 3)
ndim = 2
row x col
```

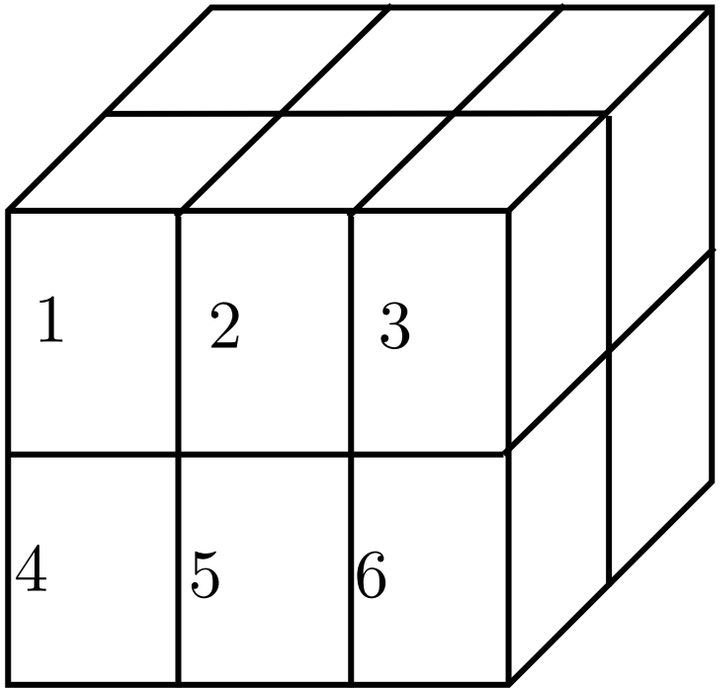
n-Dimensional Arrays




```
shape = (3, )
ndim = 1
row
```

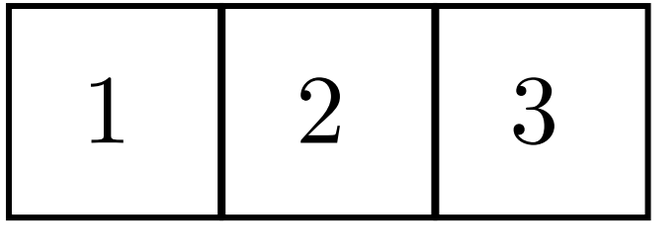


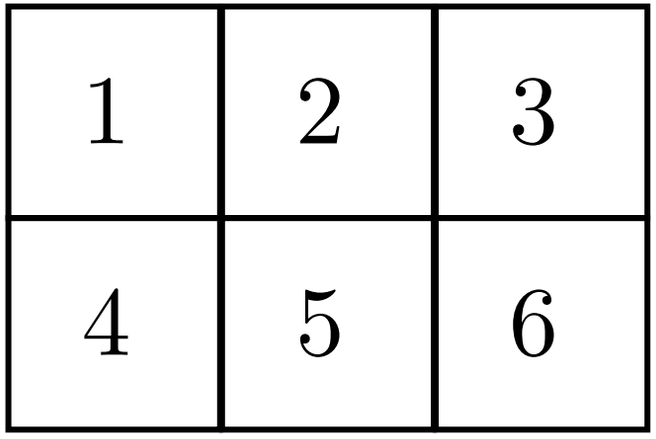

```
shape = (2, 3)
ndim = 2
row x col
```

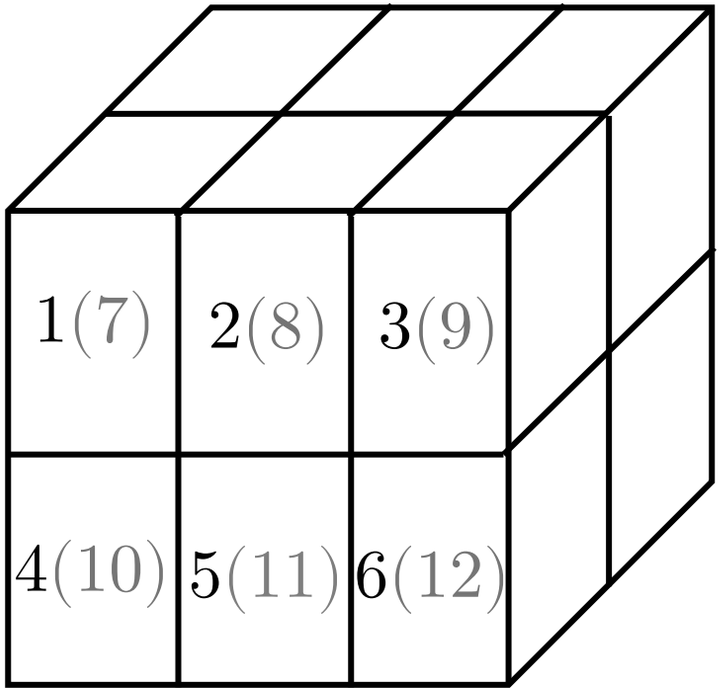



```
shape = (2, 2, 3)
ndim = 3
depth X row x col
```

n-Dimensional Arrays








```
shape = (3, )
ndim = 1
row
```

```
shape = (2, 3)
ndim = 2
row x col
```

```
shape = (2, 2, 3)
ndim = 3
depth X row x col
```

n-Dimensional Arrays: Use Cases

ndim	shape	Object

n-Dimensional Arrays: Use Cases

ndim	shape	Object
1	100	Feature-vector

n-Dimensional Arrays: Use Cases

ndim	**shape**	**Object**
1	100	Feature-vector
2	10 x 10	Grayscale image

n-Dimensional Arrays: Use Cases

ndim	shape	Object
1	100	Feature-vector
2	10 x 10	Grayscale image
3	3 x 10 x 10	RGB image

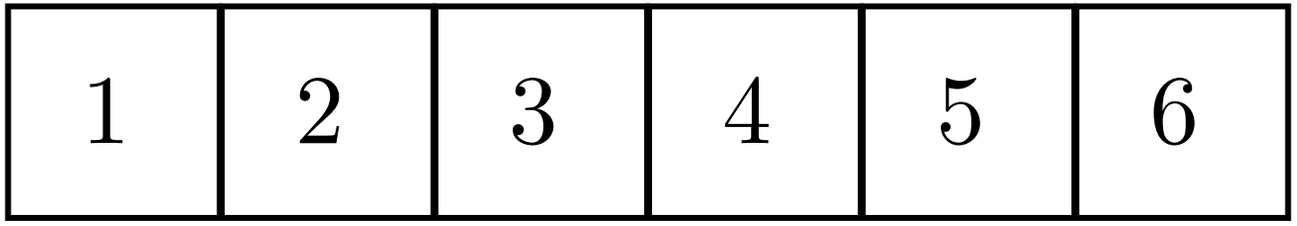
n-Dimensional Arrays: Use Cases

ndim	shape	Object
1	100	Feature-vector
2	10 x 10	Grayscale image
3	3 x 10 x 10	RGB image
4	50 x 3 x 10 x 10	50 RGB images

n-Dimensional Arrays: Use Cases

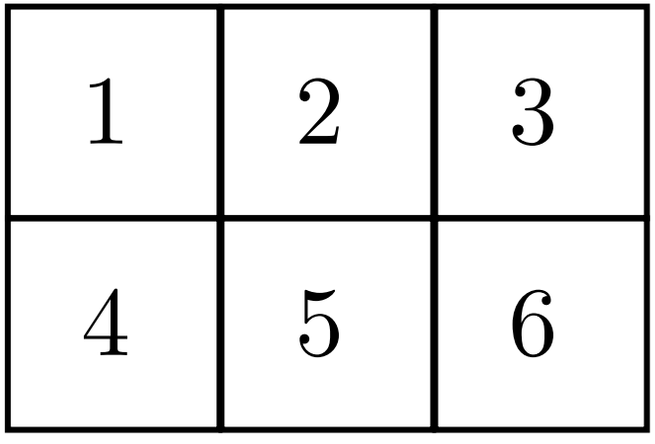
ndim	shape	Object
1	100	Feature-vector
2	10 x 10	Grayscale image
3	3 x 10 x 10	RGB image
4	50 x 3 x 10 x 10	50 RGB images
5	100 x 50 x 3 x 10 x 10	100 videos 50 frames each Each frame is a 10 x 10 RGB image

Reshape

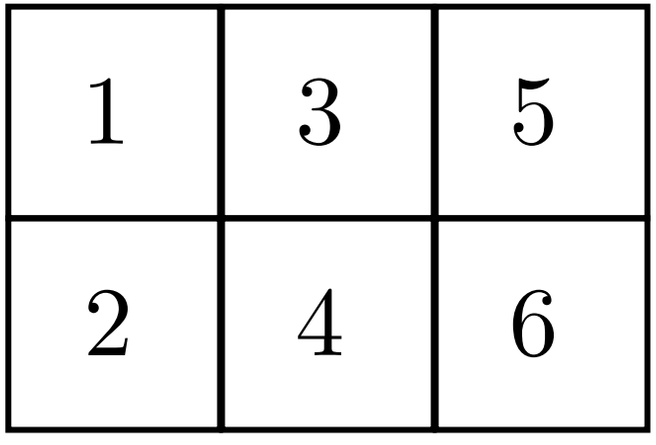


(6,)

default



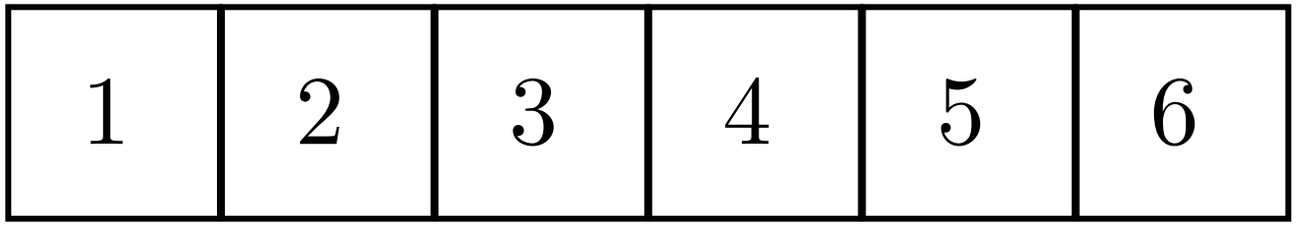
(2, 3)



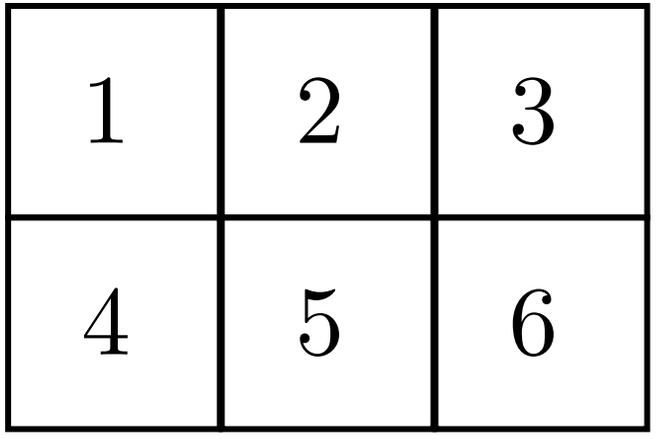
(2, 3)

Reshape

default

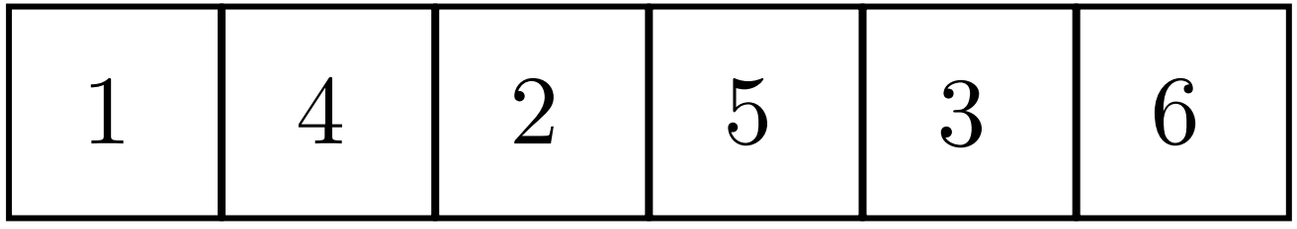


(6,)

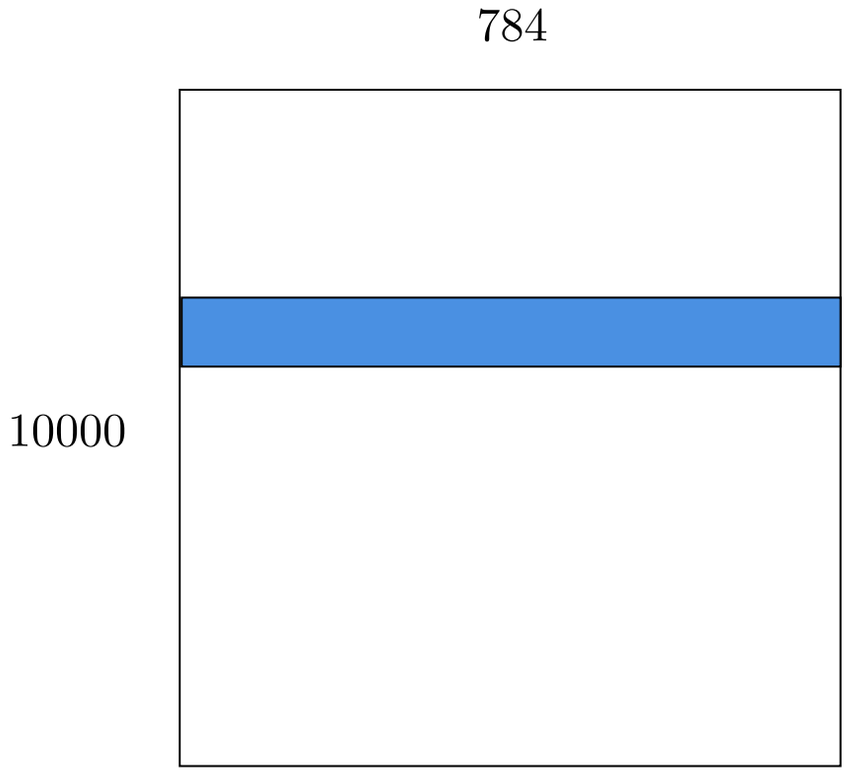


(2, 3)

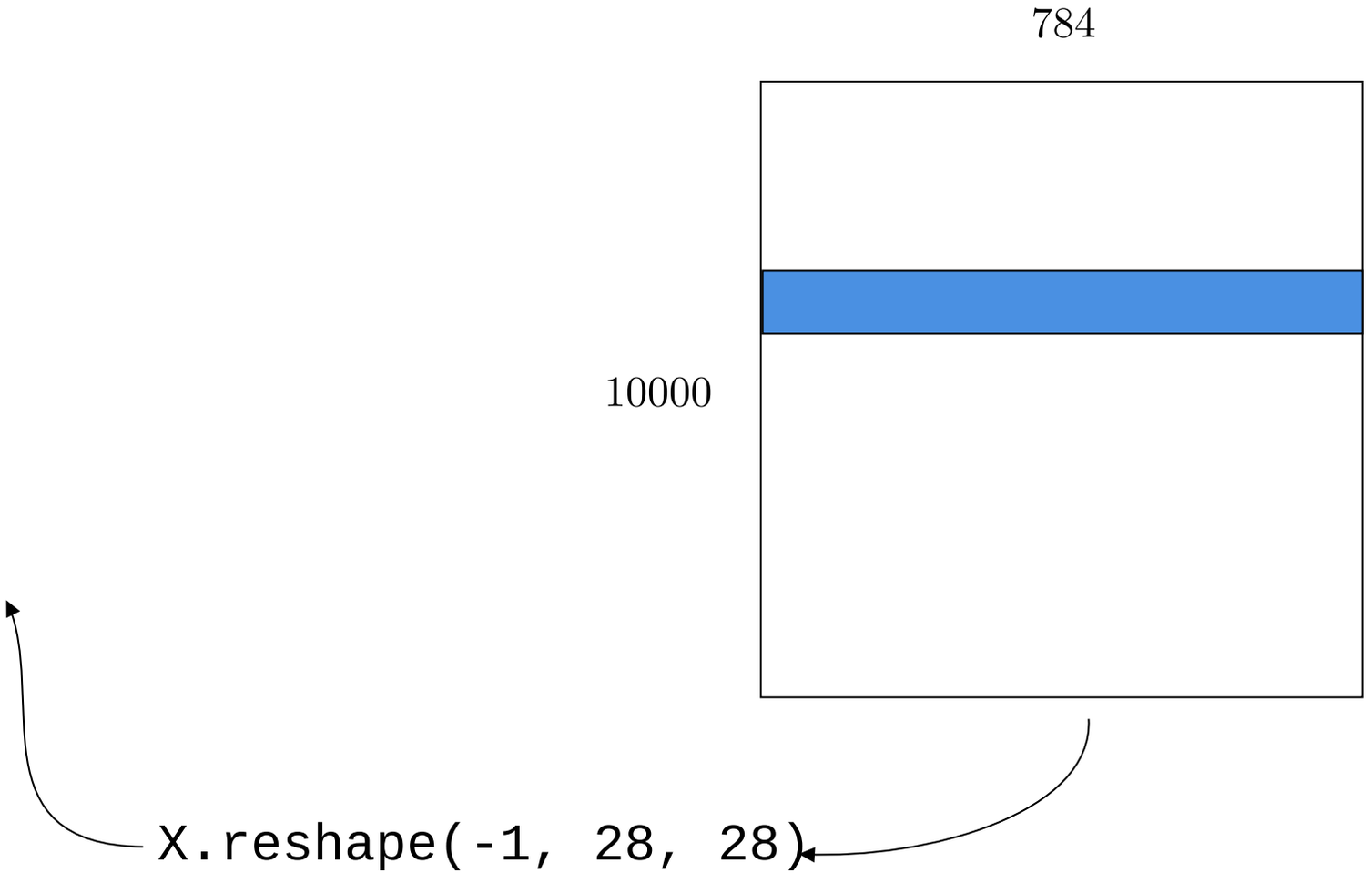
(6,)



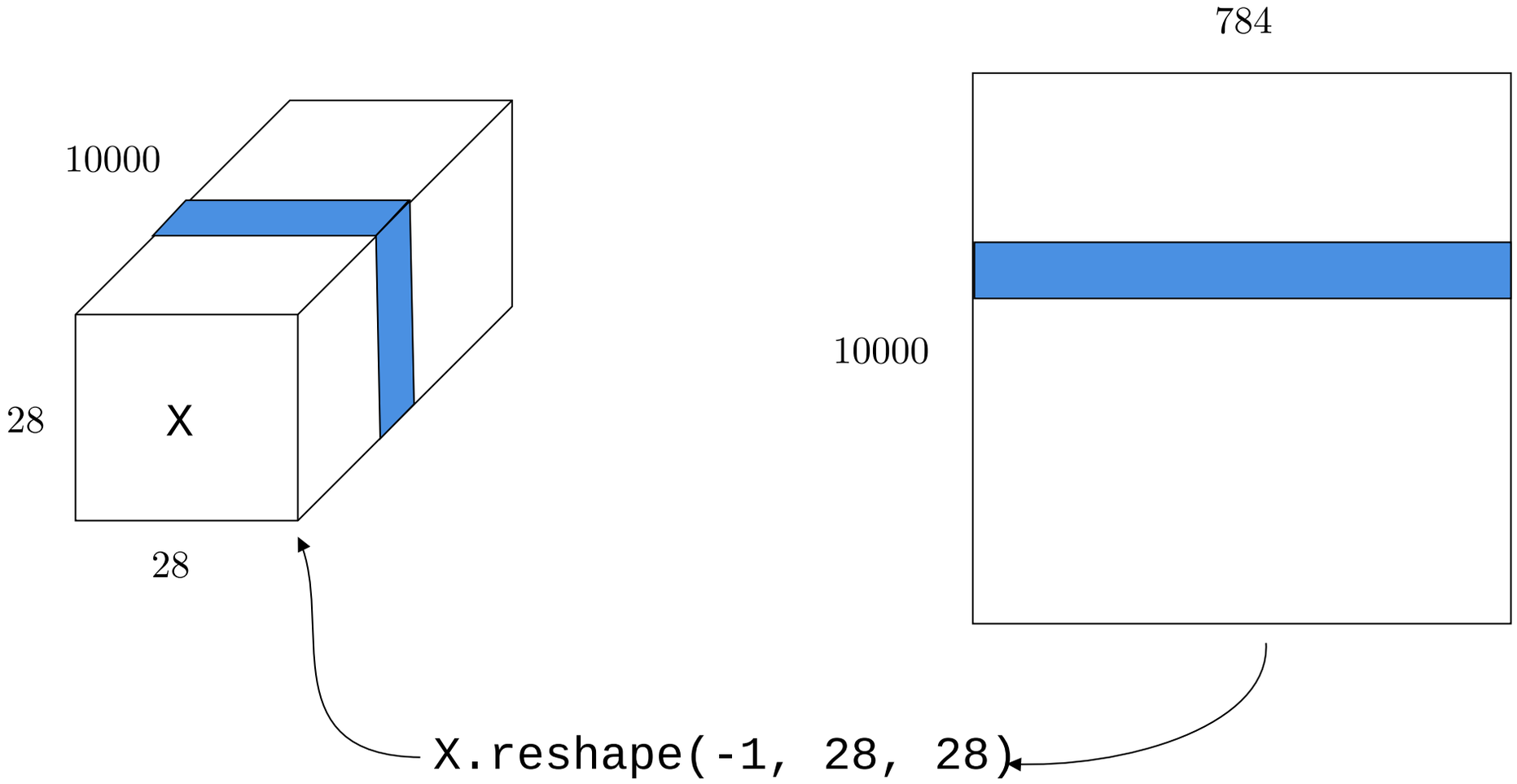
Reshape: Use Case



Reshape: Use Case



Reshape: Use Case



Matrix-vector Addition: Row vector

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

$$= \begin{bmatrix} 2 & 4 & 6 \\ 5 & 7 & 9 \end{bmatrix}$$

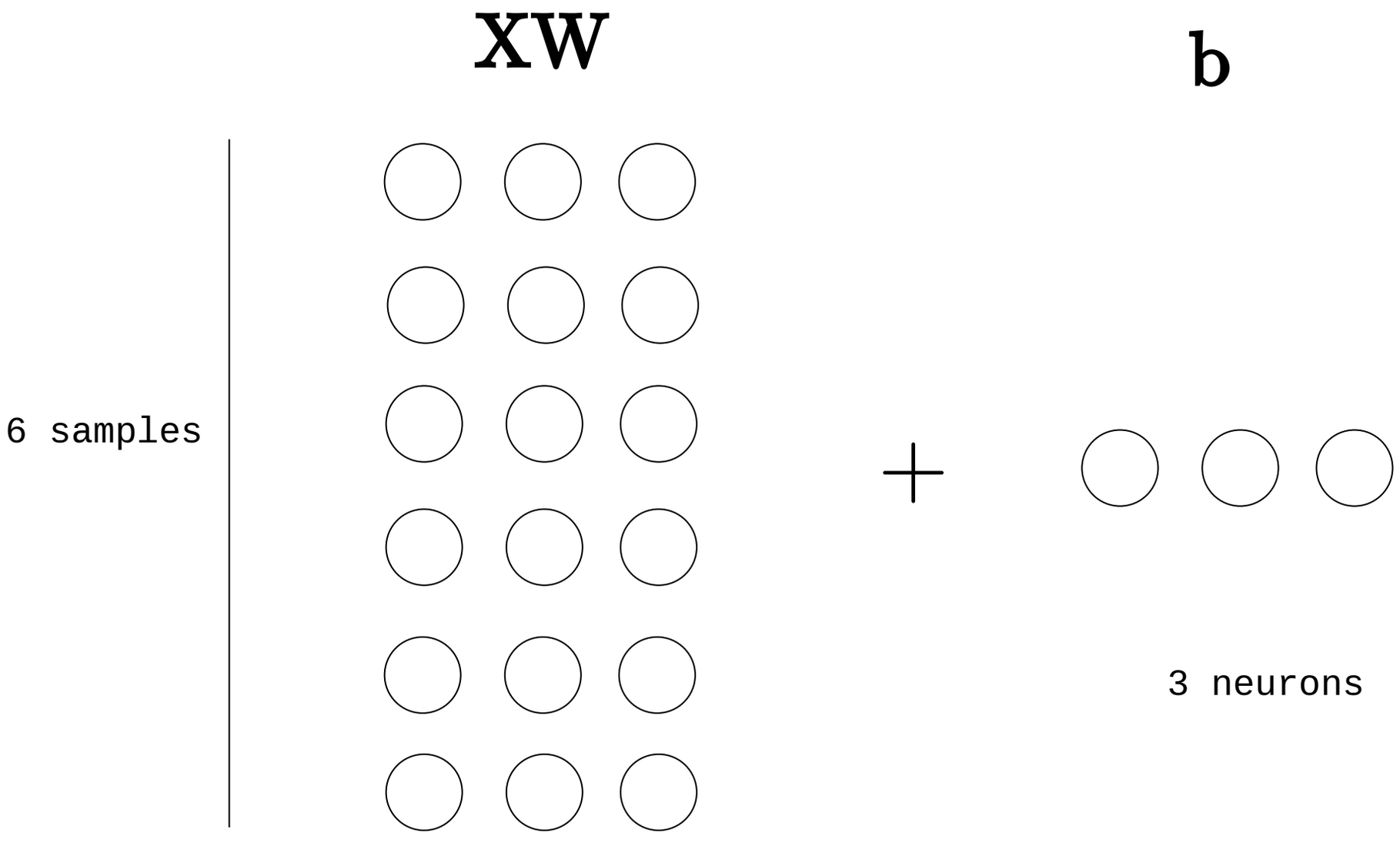
Matrix-vector Addition: Column vector

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} \overset{?}{=}$$

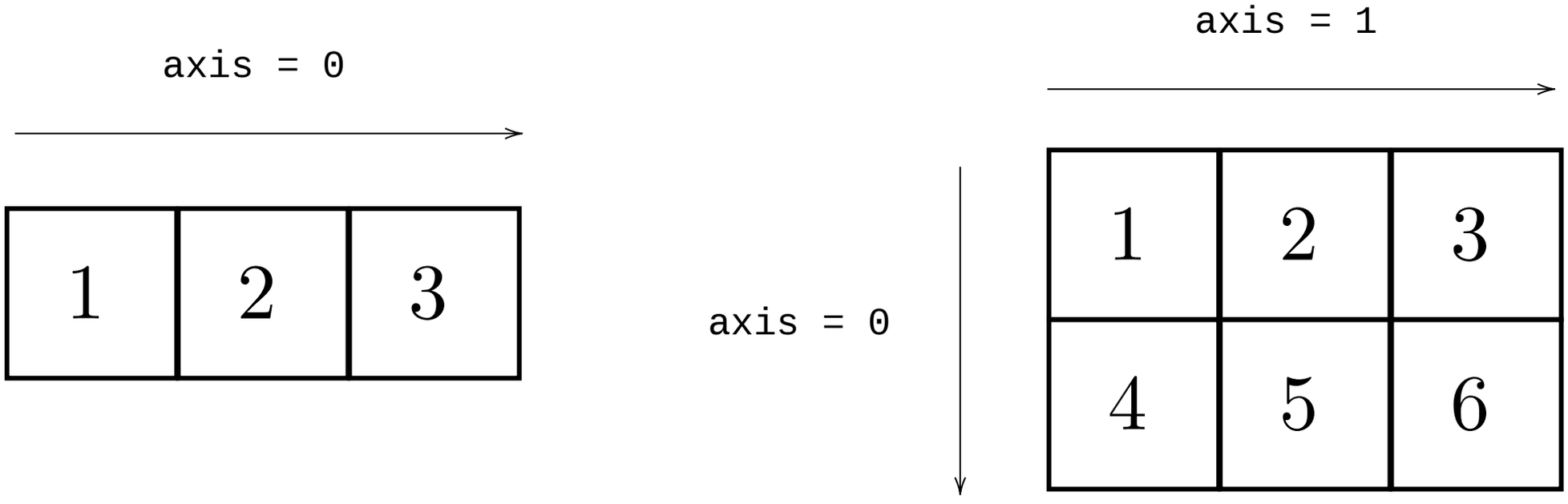
Matrix-vector Addition: Column vector

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 4 \\ 6 & 7 & 8 \end{bmatrix}$$

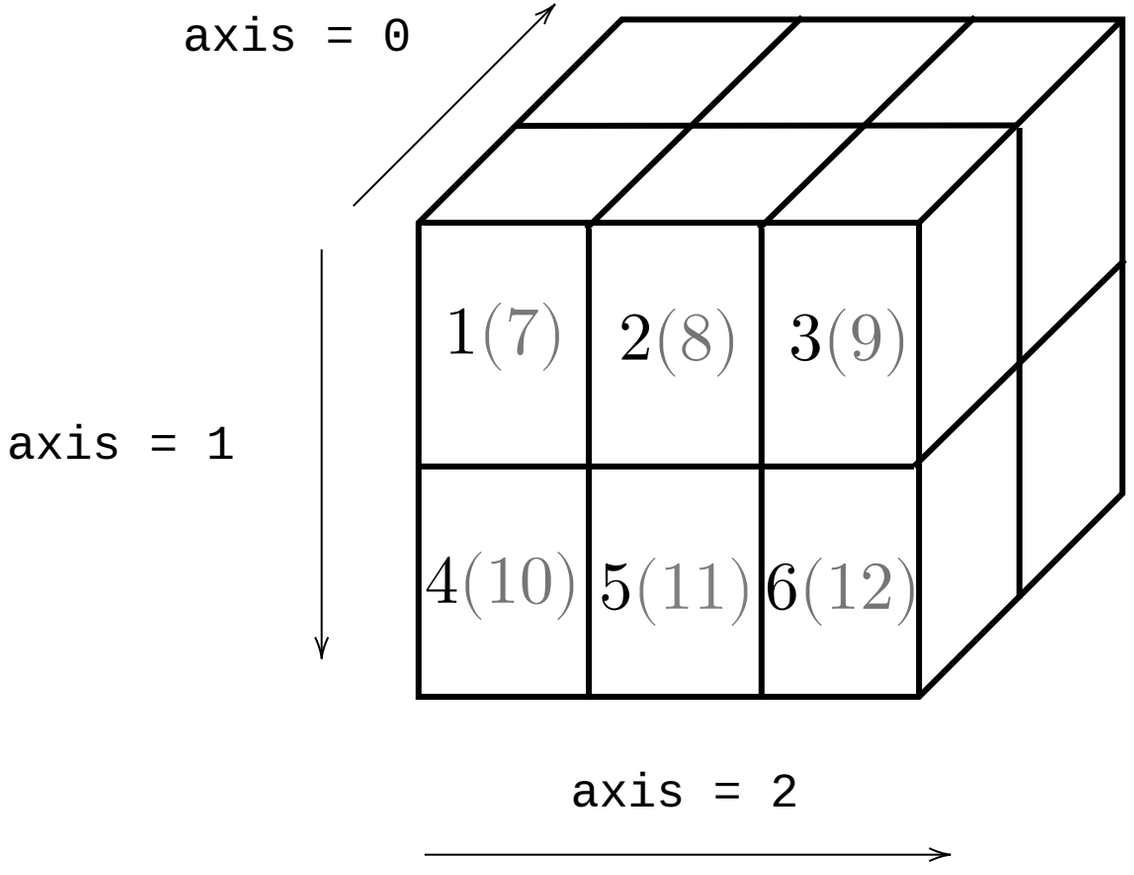
Matrix-vector Addition: Use Case



Axis



Axis



Math and NumPy

NumPy is math done using Python