Numpy - Introduction

Marcin Wierzbiński

UNIWERSYTET OTWARTY
UNIWERSYTETU WARSZAWSKIEGO

Numpy

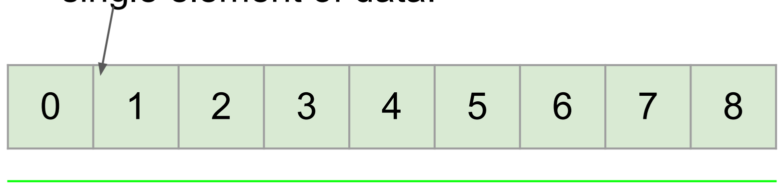
Numpy is a Python C extension library for array-oriented computing.

- Mathematical operations on arrays are up to 50x faster than iterating over native Python lists using loops.
- The efficiency gains are primarily due to NumPy storing array elements in an ordered single location within memory, eliminating redundancies by having all elements be the same type and making full use of modern CPUs.
- The efficiency advantages become particularly apparent when operating on arrays with thousands or millions of elements, which are pretty standard within data science.
- Indexing syntax for easily accessing portions of data within an array.
- It contains built-in functions that improve quality of life when working with arrays and math, such as functions for linear algebra, array transformations, and matrix math.

- NumPy is suited to many applications:
  - Image processing
  - Signal processing
  - Linear algebra

numpy array

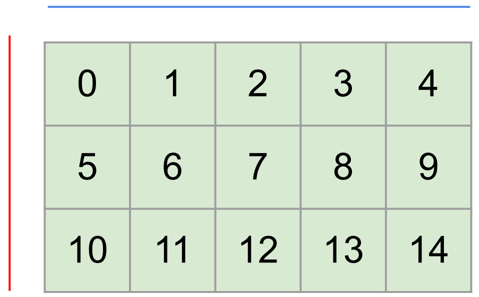
single element of data:



one dimensional array (vector)

shape: (9,)

`arr.shape`



two dimensional array
(matrix)

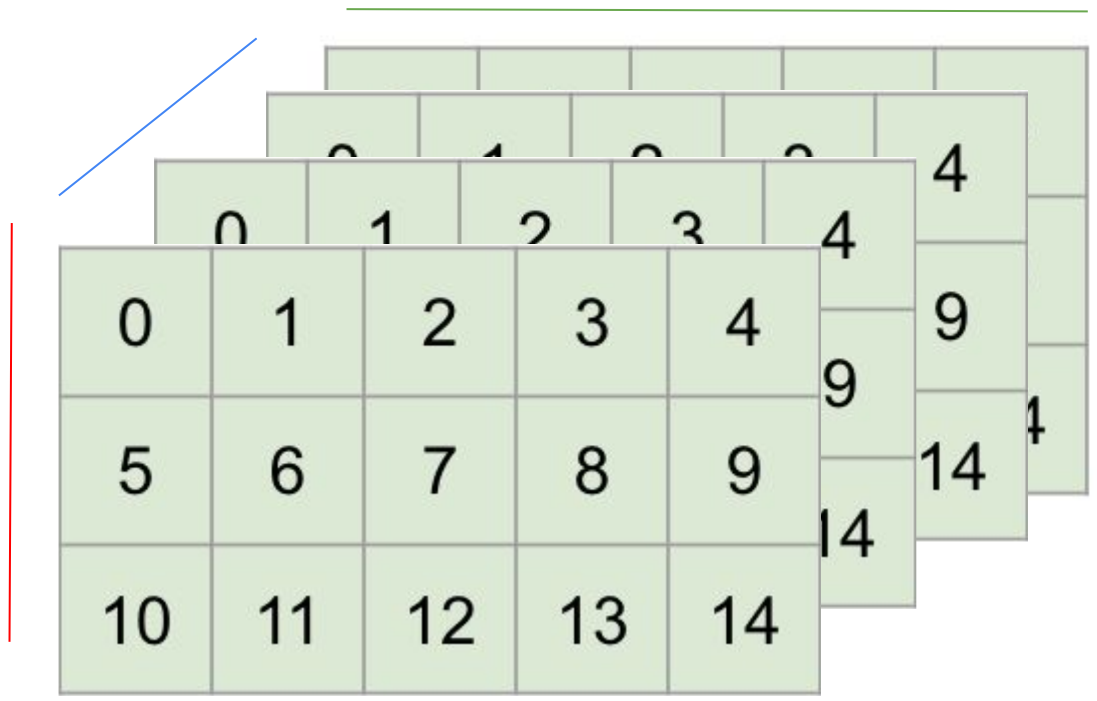
shape: (3,5)

`arr.shape`

… and so on

3D array (tensor)

Shape: (3,5,4)



Array creation

Explicitly from a list of values

```
import numpy as np
np.array([1,2,3,4])
```

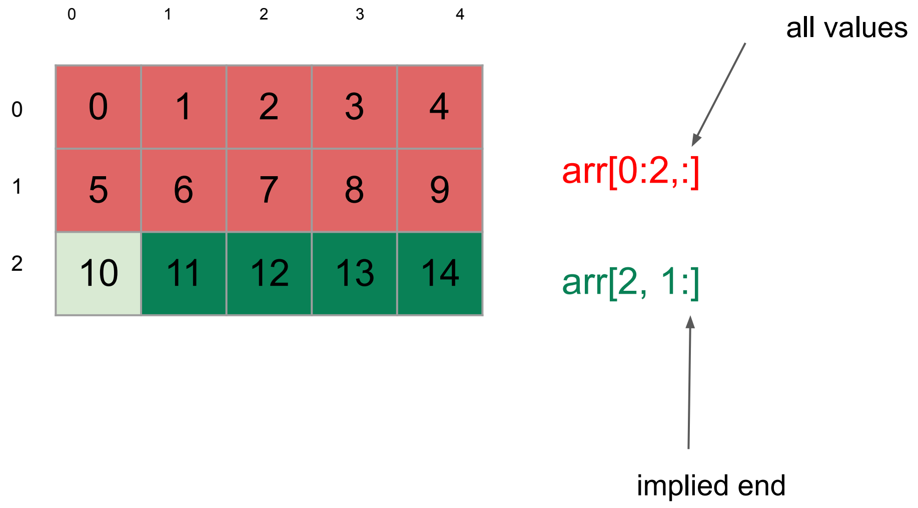
As a range of values

```
np.arange(10)
```

By specifying the number of elements

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

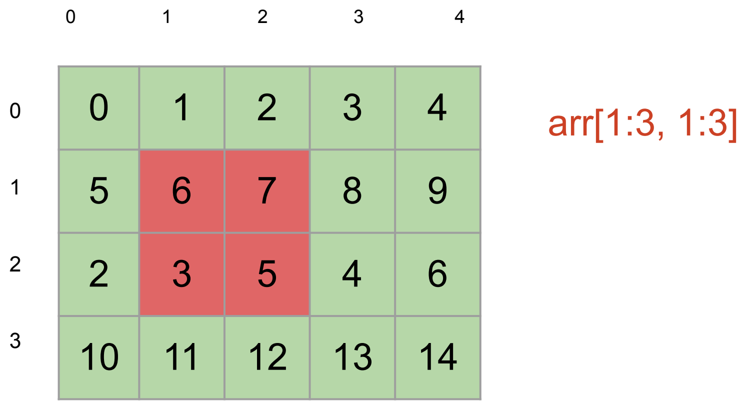
Indexing and Slicing



	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14

arr[0:2,:]

arr[2, 1:]



	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9
2	2	3	5	4	6
3	10	11	12	13	14

arr[1:3, 1:3]

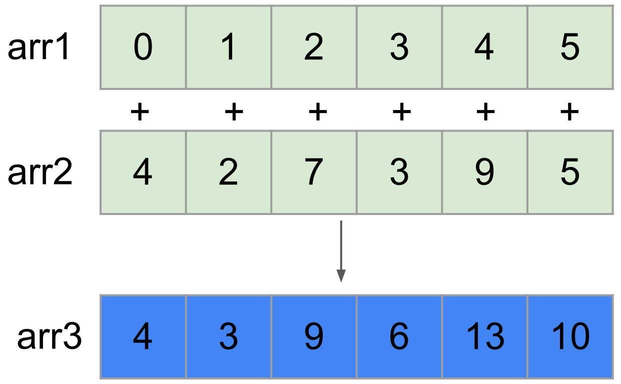
Array views

Simple assignments do not make copies of arrays. Slicing operations do not make copies either; they return views on the original array.

```
import numpy as np

a = np.arange(10)
b = a[3:7]
>> array([3, 4, 5, 6])
b[:]= 0
a
>> array([0, 1, 2, 0, 0, 0, 0, 7, 8, 9])
```

Universal function

Numpy ufuncs are functions that operate element-wise on one or more arrays



arr3 = arr1 + arr2

Numpy has many build-in ufuncs

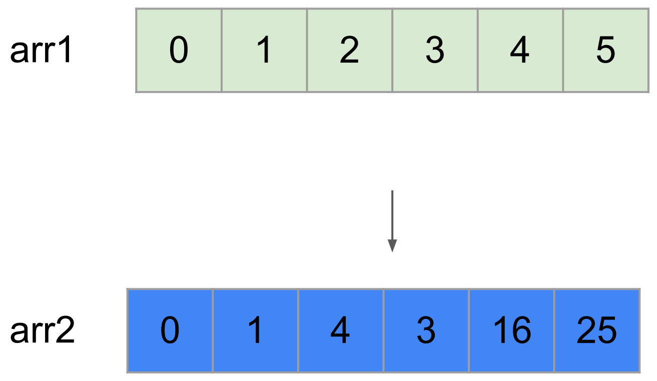
comparison: <, <=, ==, !=, >=, >

arithmetic: +, -, *, /, **

exponential: exp, log, log10, power, sqrt

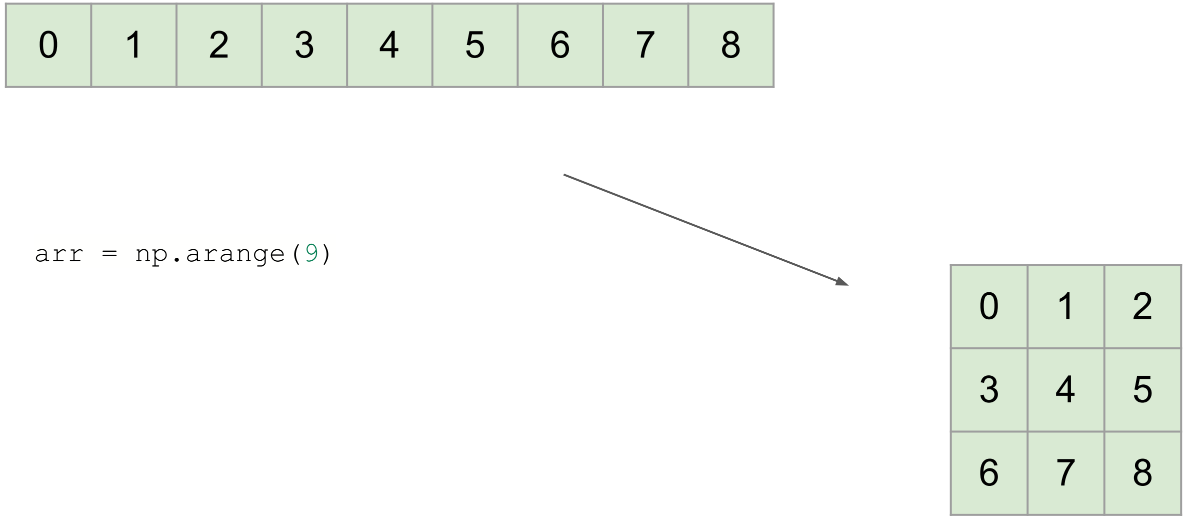
trigonometric: sin, cos, tan, arcsin

logical operations: and, not, or



arr2 = arr1 ** 2

reshape

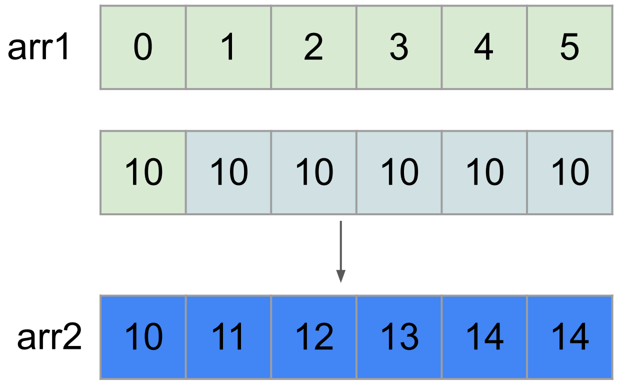


```
arr = np.arange(9)
```

```
arr.reshape(3,3)
```

broadcasting

A key feature of Numpy, where arrays with different but compatible shapes can be used as arguments to unfuncs.



arr2 = arr1 + 10

In this case an array scalar is broadcast to an array with shape (5,)

General Broadcasting Rules

Two dimensions are compatible when

1. they are equal, or
2. one of them is 1.

If these conditions are not met, a ValueError: operands could not be broadcast together exception is thrown, indicating that the arrays have incompatible shapes.

Image (3d array): (256, 256, 3)

Scale (1d array): 3

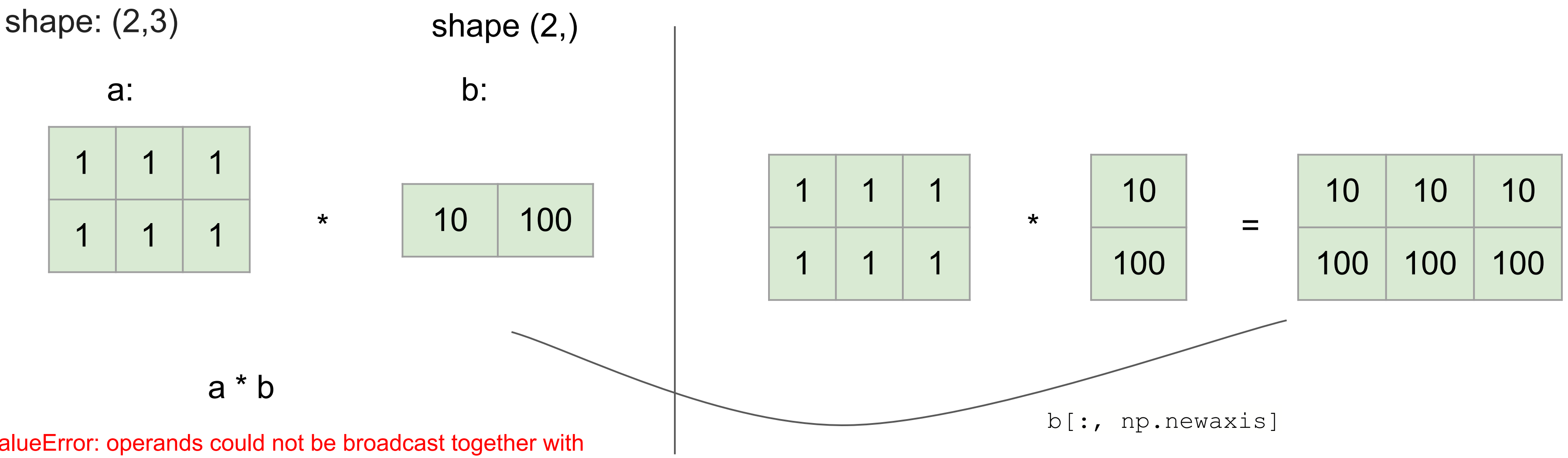
Result (3d array): (256, 256, 3)

A (4d array): (8, 1, 6, 1)

B (3d array): (7, 1, 5)

Result (4d array): (8, 7, 6, 5)

The dimensions do not match



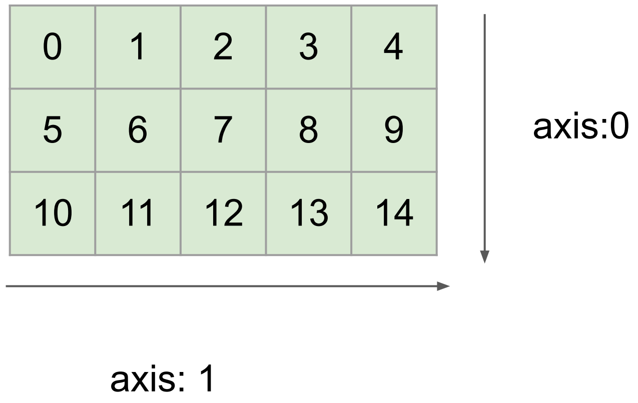
a * b

```
b[:, np.newaxis]
```

ValueError: operands could not be broadcast together with shapes (2,3) (2,)

Axis

Array method reductions take an optional axis parameter that specifies over which axes to reduce,



```
b = np.arange(15).reshape((3,5))
b.sum(axis=1)
>>> array([10, 35, 60])

b.sum(axis=0)
>>> array([15, 18, 21, 24, 27])

b.sum()
>>> 105
```

```
b.sum()
b.mean()
b.min()
b.max()
b.var()
b.std()
```

Performance pandas vs numpy

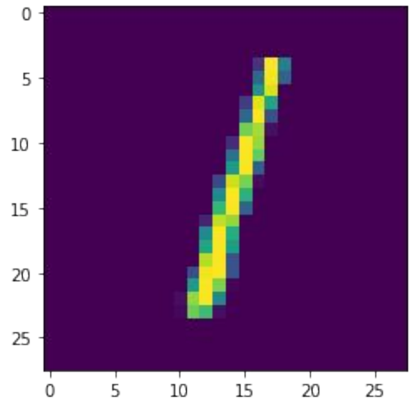
```python
import pandas as pd

df = pd.read_csv('tips.csv')
df.head()
arr = df.to_numpy()

%timeit df['total_bill'].mean()
```

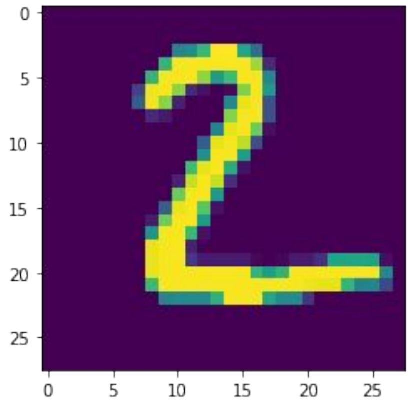
```
73.8 µs ± 18.4 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

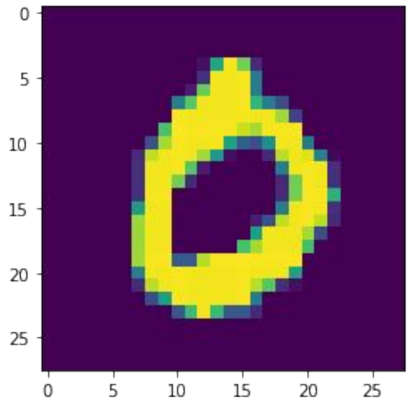
```python
%timeit arr[:,1].mean()
```

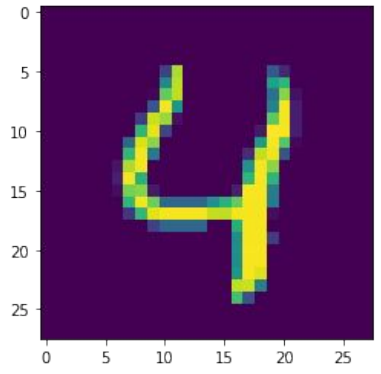
```
16.8 µs ± 4.2 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Mnist dataset







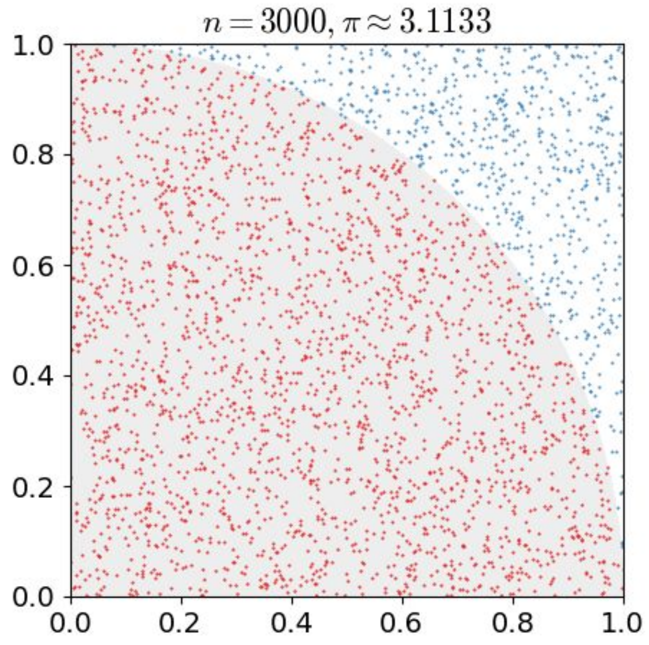


The MNIST is a large database of handwritten digits on image (28 x 28) pixels that is commonly contains 60,000 training images and 10,000 testing images.

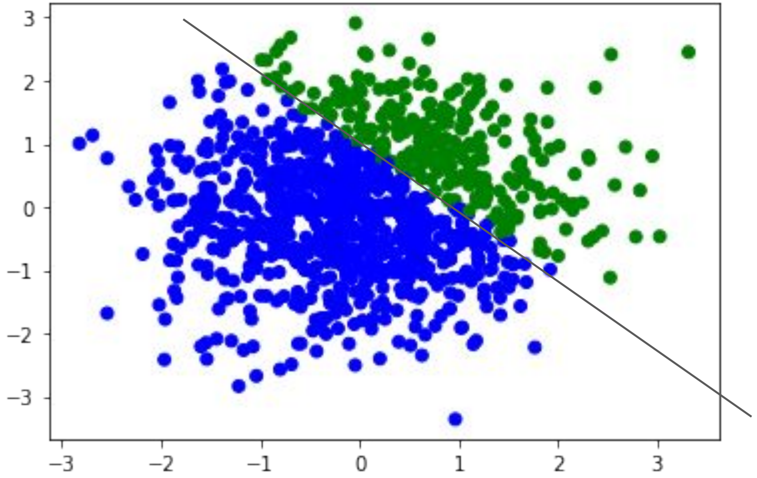
Example: estimator of pi

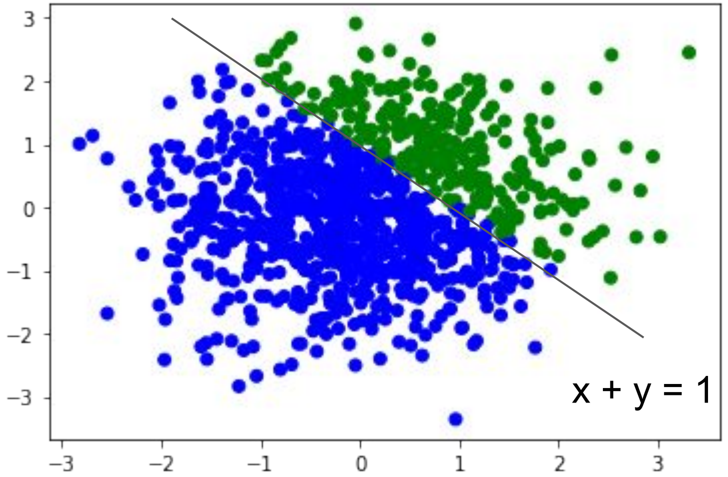
Monte Carlo method repeats random sampling to obtain some numerical results.

1. Draw a square, then inscribe a quadrant within it
2. Uniformly scatter a given number of points over the square
3. Count the number of points inside the quadrant, i.e. having a distance from the origin of less than 1
4. The ratio of the inside-count and the total-sample-count is an estimate of the ratio of the two areas, *π*/4. Multiply the result by 4 to estimate *π*.



Binary classification

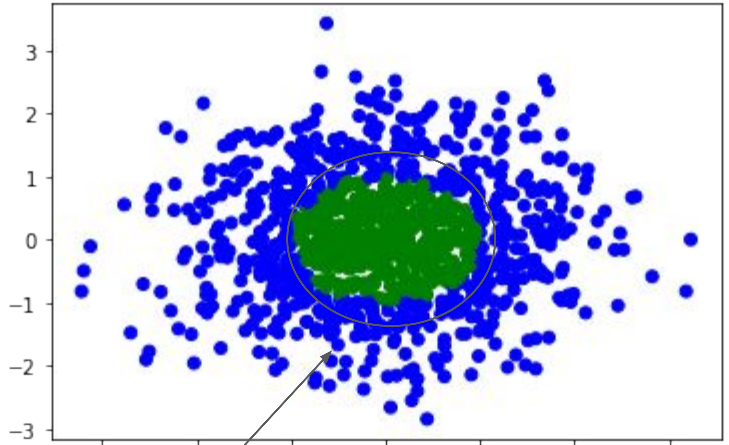


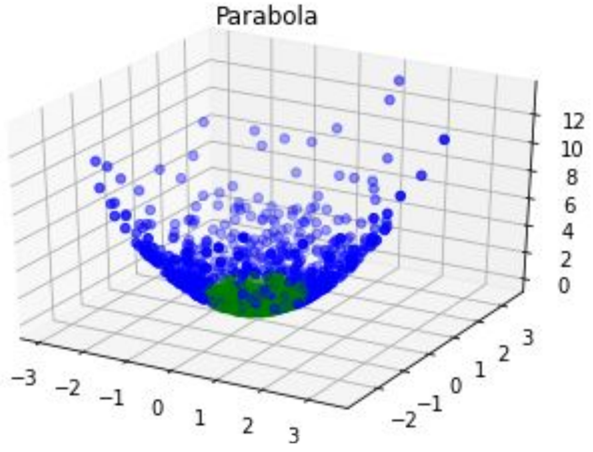


If x + y => 1: then ‘green’
If x + y < 1; then ‘blue’

Example: graph of function

parabolic function:





z = x^{2} + y^{2}

x^{2} + y^{2} = 1

Referencje:

- https://www.analyticsvidhya.com/blog/2021/05/image-processing-using-numpy-with-practical-implementation-and-code/
- https://en.wikipedia.org/wiki/Monte_Carlo_method
-