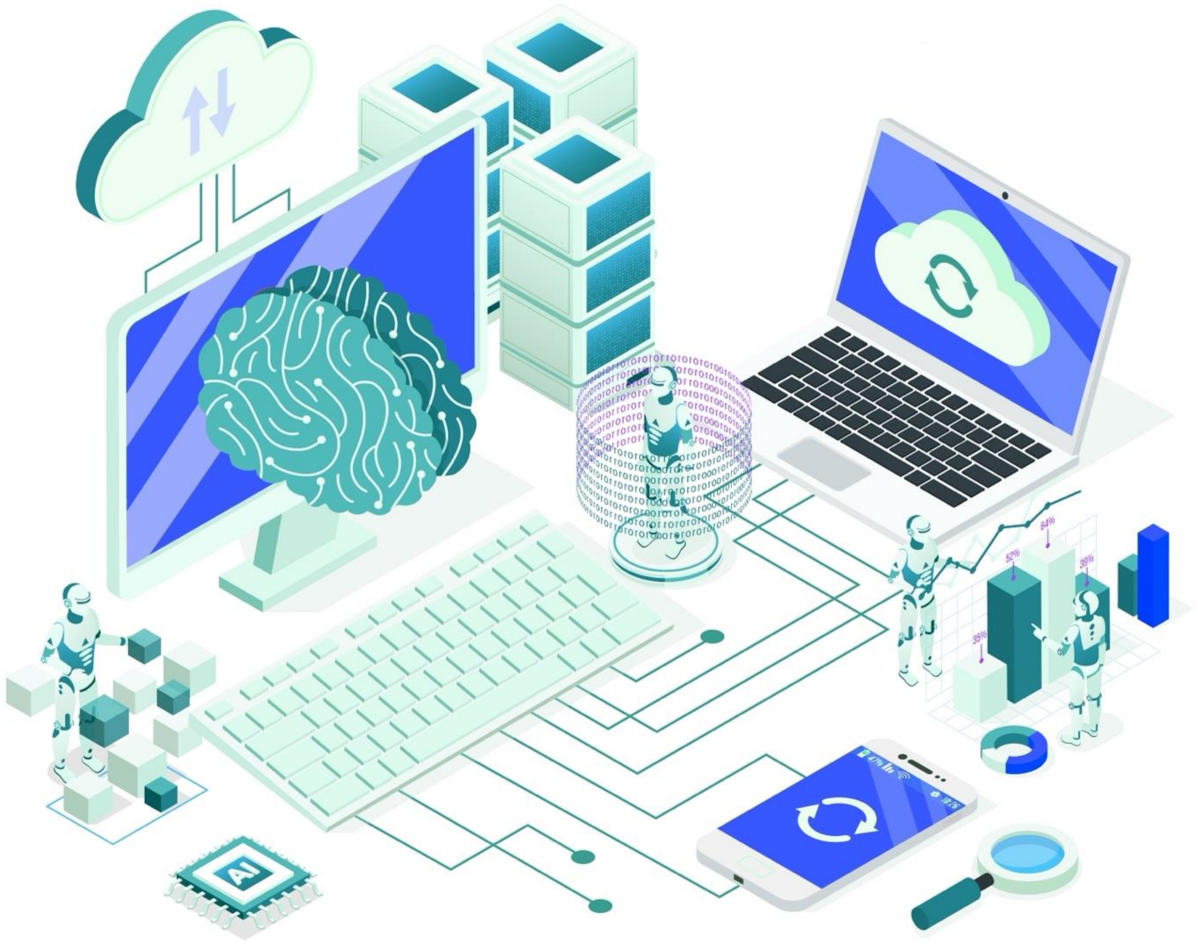
Machine Learning

Librerie Python



Prof. Giuseppe Polese Prof.ssa Loredana Caruccio

Outline

- NumPy
- ScipPy
- Pandas
- Serie
- Data Frame
- Matplotlib

NumPy

NumPy

- **NumPy** è il pacchetto fondamentale per il calcolo scientifico con **Python**
- Contiene:
  - *Un potente oggetto: matrice N-dimensionale*
  - *Funzioni sofisticate (broadcasting)*
  - *Strumenti per l'integrazione di codice C/C++ e Fortran*
  - *Utili capacità di algebra lineare, trasformata di Fourier e numeri casuali*

- **NumPy** può anche essere utilizzato come un efficiente **contenitore multidimensionale** di dati generici e possono essere definiti tipi di dati arbitrari.

Versione NumPy

- Come controllare la versione di **NumPy**

```
import numpy as np

print("NumPy version:{}".format(np.__version__))
```

```
NumPy version:1.15.4

Process finished with exit code 0
```

Code: 1. NumPy_version.py

Creare un array

- Come creare un array **NumPy**

```
import numpy as np
#Define simple list in Python
list = [0,1,2,3,4]
#Convert list in NumPy array
arr = np.array(list)
#Print content of arr and his type
print ("arr content: {}\n".format(arr))
print ("arr type: {}".format(type(arr)))
```

```
arr content: [0 1 2 3 4]

arr type: <class 'numpy.ndarray'>
```

Code: 2. NumPy_array.py

Operazioni Matematiche (1)

- Operazioni matematiche con gli array **NumPy**

```
import numpy as np
#Define simple list in Python
list = [0,1,2,3,4]
#Convert list in NumPy array
arr = np.array(list)
#Print content of arr
print ("arr content: {}".format(arr))
#Add 4 to all the arr items
arr = arr + 4
#Print arr after sum operation
print ("arr content after add operation (+4): {}".format(arr))
```

Code: 3. NumPy_operazioni_matematiche_1.py

Operazioni Matematiche (2)

- Operazioni matematiche con gli array **NumPy**

```
#Subtract 2 to all the items in arr
arr = arr - 2
#Print arr after subtraction operation
print ("arr content after subtraction operation (-2): {}".format(arr))
#Multiply 4 to all the arr items
arr = arr * 4
#Print arr after multiplication operation
print ("arr content after multiplication operation (*4):
{}".format(arr))
#Divide by 2 all the arr items
arr = arr / 2
#Print arr after division operation
print ("arr content after division operation (/2): {}".format(arr))
```

Code: 4. NumPy_operazioni_matematiche_2.py

Forma e Tipo (1)

```
import numpy as np
#Define simple list in Python
list = [0,1,2,3,4]
#Convert list in NumPy array
arr = np.array(list)
#Print content of arr
print ("arr content: {}".format(arr))
#Print arr shape and type
print ("arr shape : {}".format(arr.shape))
print ("arr type: {}",format(arr.dtype))
```

```
arr content: [0 1 2 3 4]
arr shape : (5,)
arr type: {} int32
arr1 shape : (3,)
arr1 type: {} float64
```

Code: 5. NumPy_forma_tipo_1.py

Forma e Tipo (2)

```
import numpy as np
#Convert list 2d in NumPy array
#Define simple 2d list in Python
list2 = [[5,6,7],[8,9,10]]
arr_2 = np.array(list2)
#Print content of arr_2
print ("arr_2 content: {}".format(arr_2))
#Print arr_2 shape and type
print ("arr_2 shape:
{}".format(arr_2.shape))
print ("arr_2 type:
{}".format(arr_2.dtype))
```

```
arr_2 content: [[ 5  6  7]
 [ 8  9 10]]
arr_2 shape: (2, 3)
arr_2 type: int32
```

Code: 6. NumPy_forma_tipo_2.py

Cambiare tipo (1)

```python
import numpy as np
#Define simple 2d list in Python
list2 = [[0,1,2],[3,4,5],[6,7,8]]
#Convert list2 in NumPy array 2d
arr_2 = np.array(list2)
#Print content of arr_2
print ("arr_2 content: {}".format(arr_2))
#Print arr_2 shape and type
print ("arr_2 shape : {}".format(arr_2.shape))
print ("arr_2 type: {}",format(arr_2.dtype))
```

```
arr_2 content: [[0 1 2]
 [3 4 5]
 [6 7 8]]
arr_2 shape : (3, 3)
arr_2 type: {} int32
```

Types in ***NumPy:***
'float', 'int', 'bool', 'str' and 'object'

Code: 7. NumPy_cambiare_tipo_1.py

Cambiare tipo (2)

```
#Convert integer list2 in NumPy array of float items
arr_2f = np.array(list2, dtype='float')
#Print content of arr_2f
print ("arr_2f content: {}".format(arr_2f))
#Print arr_2f shape and type
print ("arr_2f shape : {}".format(arr_2f.shape))
print ("arr_2f type: {}",format(arr_2f.dtype))
```

```
arr_2f content: [[0. 1. 2.]
 [3. 4. 5.]
 [6. 7. 8.]]
arr_2f shape : (3, 3)
arr_2f type: {} float64
```

*Types in **NumPy**:*
'float', 'int', 'bool', 'str' and 'object'

Code: 8. NumPy_cambiare_tipo_2.py

Cambiare tipo (3)

```
#Change arr_2f items to integer type through use
of astype() function
arr_2i=arr_2f.astype('int')
#Print content of arr_2i
print ("arr_2i content: {}".format(arr_2i))
#Print arr_2i shape and type
print ("arr_2i shape : {}".format(arr_2i.shape))
print ("arr_2i type: {}",format(arr_2i.dtype))
```

```
arr_2i content: [[0 1 2]
 [3 4 5]
 [6 7 8]]
arr_2i shape : (3, 3)
arr_2i type: {} int32
```

*Types in **NumPy**:*
'float', 'int', 'bool', 'str' and 'object'

Code: 9. NumPy_cambiare_tipo_3.py

Array Booleano

- Creare un array **NumPy** Booleano

```
import numpy as np
#Create a boolean array
arr_2b = np.array([1,0,1],dtype='bool')
#Print content of arr2_b
print("arr_2b content: {}".format(arr_2b))
#Print arr_2b shape and type
print ("arr_2b shape: {}".format(arr_2b.shape))
print ("arr_2b type: {}".format(arr_2b.dtype))
```

```
arr_2b content: [ True False  True]
arr_2b shape: (3,)
arr_2b type: bool
```

Code: 10. NumPy_Array_Booleano.py

Creare un oggetto

- Creare un oggetto **NumPy**

```python
import numpy as np
#Create an object array to hold numbers as well as strings
arr_obj = np.array([1, 'a'], dtype='object')
#Print content of arr_obj
print("arr_obj content: {}".format(arr_obj))
#Print arr_obj shape and type
print("arr_obj shape: {}".format(arr_obj.shape))
print("arr_obj type: {}".format(arr_obj.dtype))
```

```
arr_obj content: [1 'a']
arr_obj shape: (2,)
arr_obj type: object
```

Code: 11. NumPy_creare_oggetto.py

Array 2D vs 3D (1)

- Array **2D** vs Array **3D** con **NumPy**

```
import numpy as np
#Create 2d NumPy array
arr_2d = np.array([(1,2,3),(4,5,6)])
#Print content of arr_2d
print("arr_2d content: {}".format(arr_2d))
#Print arr_2d shape and type
print ("arr_2d shape:
{}".format(arr_2d.shape))
print ("arr_2d type: {}".format(arr_2d.dtype)
```

```
arr_2d content: [[1 2 3]
 [4 5 6]]
arr_2d shape: (2, 3)
arr_2d tyoe: int32
```

Code: 12. NumPy_array_2D_3D_1.py

Array 2D vs 3D (2)

- Array **2D** vs Array **3D** con **NumPy**

```
#Create 3d NumPy array
arr_3d = np.array([[[1, 2,3],[4, 5, 6]],[[7, 8,9],[10, 11, 12]]])
#Print content of arr_3d
print("arr_3d content: {}".format(arr_3d))
#Print arr_3d shape and type
print ("arr_3d shape: {}".format(arr_3d.shape))
print ("arr_3d type: {}".format(arr_3d.dtype))
```

```
arr_3d content: [[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]
arr_3d shape: (2, 2, 3)
arr_3d tyoe: int32
```

Code: 13. NumPy_array_2D_3D_2.py

zeros() e ones() (1)

- **numpy.zeros()** e **numpy.ones()** per creare un array pieno di zeri o uno

```
numpy.zeros(shape, dtype, order)
Return a new array of given shape and type, filled with zeros
Parameters:
        shape -> int of tuple of ints ( dimension of array like
        (2X2) or 2)
        dtype -> data-type, optional (the desired data type like
        numpy.int8, etc.) default is numpy.float64
        order -> {C,F}, optional default is 'C' (whether to store
        multi-dimensional data in row-major (C-style) or
        column-major (Fortran-style) order in memory)
Returns:
        out -> ndarray (array of all zeros)
```

zeros() e ones() (2)

- **numpy.zeros()** e **numpy.ones()** per creare un array pieno di zeri o uno

numpy.ones(*shape, dtype, order****)***
Return a new array of given shape and type, filled with ones
Parameters:
    shape -> int of tuple of ints *(dimension of array like (2X2) or 2)*
    dtype -> data-type, optional *(the desired data type like* ***numpy.int8****, etc.) default is* ***numpy.float64***
    order -> {C,F}, optional default is 'C' *(whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory)*
Returns:
    out -> ndarray *(array of all ones)*

zeros() e ones() (3)

- **numpy.zeros()** e **numpy.ones()** per creare un array piena di zeri o uno

```python
import numpy as np
#Create 2d NumPy array af all zeros
arr_zeros =
np.zeros(shape=(2,2),dtype=np.int8,order='C')
#Print content of arr_zeros
print ("arr_zeros content: {}".format(arr_zeros))
#Print arr_zeros shape and type
print ("arr_zeros shape: {}".format(arr_zeros.shape))
print ("arr_zeros type: {}".format(arr_zeros.dtype)
```

```
arr_zeros content: [[0 0]
 [0 0]]
arr_zeros shape: (2, 2)
arr_zeros type: int8
```

Code: 14. NumPy_array_zeros.py

zeros() e ones() (3)

- **numpy.zeros()** e **numpy.ones()** per creare un array piena di zeri o uno

```
#Create 2d NumPy array af all ones
arr_ones = np.ones(shape=(2,2),dtype=np.int8,order='C')
#Print content of arr_ones
print ("arr_ones content: {}".format(arr_ones))
#Print arr_ones shape and type
print ("arr_ones shape: {}".format(arr_ones.shape))
print ("arr_ones type: {}".format(arr_ones.dtype))
```

```
arr_ones content: [[1 1]
 [1 1]]
arr_ones shape: (2, 2)
arr_ones type: int8
```

Code: 15. NumPy_array_ones.py

Reshape() (1)

- **numpy.reshape()** per rimodellare i dati da larghi a lunghi

```
import numpy as np
#Create 2d NumPy array
arr_2d = np.array([(1,2,3), (4,5,6)])
#Print content of arr_2d
print ("arr_2d content: {}".format(arr_2d))
#Print arr_2d shape and type
print ("arr_2d shape: {}".format(arr_2d.shape))
print ("arr_2d type: {}".format(arr_2d.dtype))
```

```
arr_2d content: [[1 2 3]
 [4 5 6]]
arr_2d shape: (2, 3)
arr_2d type: int32
```

Code: 16. NumPy_array_reshape_1.py

Reshape() (2)

- **numpy.reshape()** per rimodellare i dati da larghi a lunghi

```
arr_2dr = arr_2d.reshape(3,2)
#Print content of arr_2dr
print ("arr_2dr content: {}".format(arr_2dr))
#Print arr_2dr shape and type
print ("arr_2dr shape: {}".format(arr_2dr.shape))
print ("arr_2dr type: {}".format(arr_2dr.dtype))
```

```
arr_2dr content: [[1 2]
 [3 4]
 [5 6]]
arr_2dr shape: (3, 2)
arr_2dr type: int32
```

Code: 17. NumPy_array_reshape_2.py

flatten() (1)

- **numpy.flatten()** per appiattire l'array

```python
import numpy as np
#Create 2d NumPy array
arr_2d = np.array([(1,2,3), (4,5,6)])
#Print content of arr_2d
print ("arr_2d content: {}".format(arr_2d))
#Print arr_2d shape and type
print ("arr_2d shape: {}".format(arr_2d.shape))
print ("arr_2d type: {}".format(arr_2d.dtype))
```

```
arr_2d content: [[1 2 3]
 [4 5 6]]
arr_2d shape: (2, 3)
arr_2d type: int32
```

Code: 18. NumPy_array_flatten_1.py

flatten() (2)

- **numpy.flatten()** per appiattire l'array

```
arr_1d = arr_2d.flatten()
#Print content of arr_1d
print ("arr_1d content: {}".format(arr_1d))
#Print arr_1d shape and type
print ("arr_1d shape: {}".format(arr_1d.shape))
print ("arr_1d type: {}".format(arr_1d.dtype))
```

```
arr_1d content: [1 2 3 4 5 6]
arr_1d shape: (6,)
arr_1d type: int32
```

Code: 19. NumPy_array_flatten_2.py

hstack() e vstack()

- **numpy.hstack() e numpy.vstack()** per aggiungere dati orizzontalmente e verticalmente

```python
import numpy as np
#Create two 2d NumPy array
arr_1 = np.array([1,2,3])
arr_2 = np.array([4,5,6])
#Print content of arr_1
print ("arr_1 content: {}".format(arr_1))
#Print arr_1 shape and type
print ("arr_1 shape: {}".format(arr_1.shape))
print ("arr_1 type: {}".format(arr_1.dtype))
#Print content of arr_2
print ("arr_2 content: {}".format(arr_2))
#Print arr_2 shape and type
print ("arr_2 shape: {}".format(arr_2.shape))
print ("arr_2 type: {}".format(arr_2.dtype))
happ = np.hstack((arr_1,arr_2))
print ("Horizontal Append: {}".format(happ))
vapp = np.vstack((arr_1,arr_2))
print ("Vertical Append: {}".format(vapp))
```

```
arr_1 content: [1 2 3]
arr_1 shape: (3,)
arr_1 type: int32
arr_2 content: [4 5 6]
arr_2 shape: (3,)
arr_2 type: int32
Horizontal Append: [1 2 3 4 5 6]
Vertical Append: [[1 2 3]
 [4 5 6]]
```

Code: 20. NumPy_array_hstack_vstack.py

asarray() (1)

- **numpy.asarray()**
  - Se si desidera modificare il valore della matrice, non è possibile. Il motivo è che non è possibile cambiare una copia
  - La matrice è immutabile. È possibile utilizzare **numpy.asarray()** se si desidera aggiungere modifiche nella matrice originale.

```
numpy.asarray(a, dtype=None, order=None)
Convert the input to an array
Parameters:
          a -> array_like (input data, in any form that can be
          converted to an array. This includes lists, lists of
          tuples, tuples, tuples of tuples, tuples of lists and
          ndarrays)
          dtype -> data-type, optional (by default, the data-type is
          inferred from the input data)
          order -> {C,F}, optional default is 'C' (whether to use row-major
          (C-style) or column-major (Fortran-style) memory representation)
Returns:
          out -> ndarray (array interpretation of a. No copy is
          performed if the input is already an ndarray with
          matching dtype and order. If a is a subclass of ndarray, a base
          class ndarray is returned)
```

asarray() (2)

- **numpy.asarray()**

```python
import numpy as np
#Create matrix of all ones
A = np.matrix(np.ones((4,4)))
#Print content of A
print ("A content: {}".format(A))
#The change is made on the third line because the indexing starts at 0
np.asarray(A)[2] = 2
#Print content A after the change
print("A content after the change: {}".format(A))
```

```
A content: [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
A content after the change: [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [2. 2. 2. 2.]
 [1. 1. 1. 1.]]
```

Code: 21. NumPy_asarray.py

arange() (1)

- **numpy.arange()** per la creazione di valori uniformemente distanziati all'interno di un determinato intervallo. Ad esempio, se si desidera creare valori da 1 a 10

```
numpy.arange(start, stop, step, dtype=None)
Return evenly spaced values within a given interval.Values are
generated within the half-open interval [start, stop) (in other
words, the interval including start but excluding stop)

Parameters:
        start -> number, optional (start of interval. The interval
        includes this value. The default start value is 0)
        stop -> number (end of interval)
        step -> number, optional (spacing between values. For any
        output out, this is the distance between two adjacent
        values, out[i+1] - out[i]. The default step size is 1)
        dtype -> dtype (the type of the output array. If dtype is
not     given, infer the data type from the other input arguments)
Returns:
        out -> ndarray (array of evenly spaced values)
```

arange() (2)

- **numpy.arange()** per la creazione di valori uniformemente distanziati all'interno di un determinato intervallo. Ad esempio, se si desidera creare valori da 1 a 10

```
import numpy as np
#Create array with numpy.arange() of 10 elements
arr = np.arange(1, 11)
#Print content of arr
print ("arr content: {}".format(arr))
#Create array with numpy.arange() of 4 elements and step = 4
arr_1 = np.arange(1, 14,4)
#Print content of arr_1
print ("arr_1 content (step = 4): {}".format(arr_1))
```

```
arr content: [ 1  2  3  4  5  6  7  8  9 10]
arr_1 content (step = 4): [ 1  5  9 13]
```

Code: 22. NumPy_arange.py

linspace() (1)

- **numpy.linspace()** restituisce numeri con spaziatura uniforme su un intervallo specificato

```
numpy.linspace(start, stop, num, endpoint)
Returns num evenly spaced samples, calculated over the interval
[start, stop]. The endpoint of the interval can optionally be excluded

Parameters:
        start -> array_like (the starting value of the sequence)
        stop -> array_like (the end value of the sequence,
        unless endpoint is set to False)
        num -> int, optional (number of samples to generate. Default is
        50. Must be non-negative)
        endpoint -> bool, optional (If True, stop is the last sample.
        Otherwise, it is not included. Default is True)
Returns:
        out -> ndarray (there are num equally spaced samples in the closed
        interval [start, stop] or the half-open interval [start, stop,
        depending on whether endpoint is True or False )
```

linspace() (2)

- **numpy.linspace()** restituisce numeri con spaziatura uniforme su un intervallo specificato

```
import numpy as np
#Create array with numpy.linspace() of 10 elements
arr = np.linspace(1.0, 5.0, num=10)
#Print content of arr
print ("arr content: {}".format(arr))
#Create array with numpy.linspace() of 5 with endpoint=False
arr_1 = np.linspace(1.0, 5.0, num=5, endpoint=False)
#Print content of arr_1
print ("arr_1 content: {}".format(arr_1))
```

```
arr content: [1.         1.44444444 1.88888889 2.33333333 2.77777778 3.22222222
 3.66666667 4.11111111 4.55555556 5.        ]
arr_1 content: [1.  1.8 2.6 3.4 4.2]

Process finished with exit code 0
```

Code: 23. NumPy_linspace.py

Indexing

- **Indicizzazione** nella libreria **NumPy**

```python
import numpy as np
#Create 2d NumPy array
arr_2d = np.array([(1,2,3), (4,5,6)])
#Print content of arr_2d
print ("arr_2d content: {}".format(arr_2d))
#Print first row of arr_2d
print ("first row of arr_2d: {}".format(arr_2d[0]))
#Print second row of arr_2d
print ("second row of arr_2d: {}".format(arr_2d[1]))
#Print first column of arr_2d
print ("first column of arr_2d: {}".format(arr_2d[:,0]))
#Print second column of arr_2d
print ("second column of arr_2d: {}".format(arr_2d[:,1]))
#Print third column of arr_2d
print ("third column of arr_2d: {}".format(arr_2d[:,2]))
#Print first two elements of the second row of arr_2d
print ("first two elements of arr_2d: {}".format(arr_2d[1,:2]))
```

```
arr_2d content: [[1 2 3]
 [4 5 6]]
first row of arr_2d: [1 2 3]
second row of arr_2d: [4 5 6]
first column of arr_2d: [1 4]
second column of arr_2d: [2 5]
third column of arr_2d: [3 6]
first two elements of arr_2d: [4 5]
```

Code: 25. NumPy_indicizzazione.py

Funzioni statistiche (1)

- **Funzioni statistiche** nella libreria **NumPy**

Funzioni	NumPy
Min	numpy.min()
Max	numpy.max()
Media	numpy.mean()
Mediana	numpy.median()
Deviazione Standard	numpy.std()

Funzioni statistiche (2)

- **Funzioni statistiche** nella libreria **NumPy**

```
import numpy as np
#Generate random number from normal distribution
normal_array = np.random.normal(5, 0.5, 10)
#Print content of normal_array
print ("normal_array content: {}".format(normal_array))
#Min
print("normal_array min: {}".format(np.min(normal_array)))
#Max
print("normal_array max: {}".format(np.max(normal_array))
```

```
normal_array content: [5.57274138 5.28203603 5.36255524 4.53879078 4.10160406 5.23754758
 4.74041197 5.10384769 4.59438864 4.96687754]
normal_array min: 4.101604059492985
normal_array max: 5.572741379606414
```

Code: 26. NumPy_funzioni_statistiche_1.py

Funzioni statistiche (2)

- **Funzioni statistiche** nella libreria **NumPy**

```
#Mean
print("normal_array mean: {}".format(np.mean(normal_array)))
#Median
print("normal_array median: {}".format(np.median(normal_array)))
#Standard deviation
print("normal_array standard deviation: {}".format(np.std(normal_array)))
```

```
normal_array mean: 4.950080090756345
normal_array median: 5.035362613162279
normal_array standard deviation: 0.42826970013338816
```

Code: 27. NumPy_funzioni_statistiche_2.py

SciPy

2

SciPy

- **SciPy** è una libreria open source basata su Python, utilizzata in matematica, calcolo scientifico, ingegneria e calcolo tecnico.
- **SciPy** contiene una varietà di sotto-pacchetti che aiutano a risolvere il problema più comune relativo al calcolo scientifico
- **SciPy** è la libreria scientifica più utilizzata seconda solo alla GNU Scientific Library per C / C ++ o Matlab
- Facile da usare e da capire, nonché potenza di calcolo veloce
- Può operare su una matrice di libreria **NumPy**

NumPy vs SciPy

- **NumPy:**
  - è scritto in C e utilizzato per il calcolo matematico o numerico
  - è più veloce di altre librerie Python
  - **NumPy** è la libreria più utile per Data Science per eseguire calcoli di base
  - **NumPy** contiene il tipo di dati array, che esegue le operazioni più basilari come l'ordinamento, la modellatura, l'indicizzazione, ecc.
- **SciPy:**
  - è costruito in cima al **NumPy**
  - è una versione completa di Linear Algebra mentre **Numpy** contiene solo poche funzionalità
  - la maggior parte delle nuove funzionalità di Data Science sono disponibili in **SciPy** anziché in **NumPy**

Versione SciPy

- Come controllare la versione **SciPy**

```
import scipy as sp
print("SciPy version:{}".format(sp.__version__))
```

```
SciPy version: 1.1.0
```

Cubic root

- La funzione **Radice cubica** trova la radice cubica dei valori

```
from scipy.special import cbrt
#Find cubic root of 27 & 64 using cbrt() function
cubic_root = cbrt([27, 64])
#Print content of cubic_root
print("cubic roots: {}".format(cubic_root))
```

```
cubic roots: [3. 4.]
```

Code: 29. SciPy_cubic.py

Funzione esponenziale

- La **funzione esponenziale** calcola il risultato di 10^x

```
from scipy.special import exp10
#Define exp10 function and pass value in its
exp_values = exp10([1,10])
#Print content of exp_values
print("exponential values: {}".format(exp_values))
```

```
exponential values: [1.e+01 1.e+10]
```

Code: 30. SciPy_Funzione_esponenziale.py

Permutazioni e combinazioni (1)

- **SciPy** fornisce anche funzionalità per calcolare permutazioni e combinazioni

```
scipy.special.comb(N, k, exact, repetition)
The number of combinations of N things taken k at a time

Parameters:
        N -> int, ndarray (number of things)
        k -> int, ndarray (number of elements taken)
        exact -> bool, optional (if exact is False, then floating
        point precision is used, otherwise exact long integer
        is computed)
        repetition -> bool, optional (if repetition is True, then
        the number of combinations with repetition is computed)
Returns:
        out -> int, float, ndarray (the total number of
        combinations)
```

Permutazioni e combinazioni (2)

- **SciPy** fornisce anche funzionalità per calcolare **permutazioni** e **combinazioni**

```
from scipy.special import comb
#Find combinations of 5, 2 values using comb(N, k)
com = comb(5, 2, exact = False, repetition=True)
#Print content of com
print("combination value: {}".format(com))
```

```
combination value: 15.0

Process finished with exit code 0
```

Code: 31. SciPy_permutazioni_combinazioni.py

Permutazioni e combinazioni (3)

- **SciPy** fornisce anche funzionalità per calcolare **permutazioni** e **combinazioni**

```
from scipy.special import perm
#Find permutation of 5, 2 using perm (N, k) function
per = perm(5, 2, exact = True)
#Print content of per
print("permutation value: {}".format(per))
```

```
permutation value: 20

Process finished with exit code 0
```

Code: 31. SciPy_permutazioni_combinazioni.py

Determinante della matrice

- **Calcolo** del **determinante** di una matrice bidimensionale

```
from scipy import linalg
import numpy as np
#Define 2d NumPy array
arr_2d = np.array([ (4,5), (3,2) ])
#Print content of arr_2
print ("arr_2d content: {}".format(arr_2d))
#Pass values to det() function
det = linalg.det(arr_2d)
#Print content of det
print ("matrix determinant: {}".format(det))
```

```
arr_2d content: [[4 5]
 [3 2]]
matrix determinant: -7.0
```

Code: 32. SciPy_determinante.py

Inversa della matrice

- **Calcolo dell'inversa** di qualsiasi **matrice** quadrata

```
from scipy import linalg
import numpy as np
#Define 2d NumPy array
arr_2d = np.array([ [4,5], [3,2] ])
#Print content of arr_2d
print ("arr_2d content: {}".format(arr_2d))
#Pass value to function inv()
inv = linalg.inv( arr_2d )
#Print content of inv
print ("inverse matrix : {}".format(inv))
```

```
arr_2d content: [[4 5]
 [3 2]]
inverse matrix : [[-0.28571429  0.71428571]
 [ 0.42857143 -0.57142857]]
```

Code: 33. SciPy_inversa.py

Autovalori e autovettori

- **Autovalori** e **autovettori** che possono essere facilmente risolti usando **SciPy**

```
from scipy import linalg
import numpy as np
#Define 2d NumPy array
arr_2d = np.array([[5,4],[6,3]])
#Pass value into function eig()
eg_val, eg_vect = linalg.eig(arr_2d)
#Print content of eg_val
print("eigenvalues: {}".format(eg_val))
#Print content of eg_vect
print("eigenvectors: {}".format(eg_vect))
```

```
eigenvalues: [ 9.+0.j -1.+0.j]
eigenvectors: [[ 0.70710678 -0.5547002 ]
 [ 0.70710678  0.83205029]]
```

Code: 34. SciPy_autovalori_autovettori.py

Integrazione numerica (1)

- La libreria **scipy.integrate** ha a disposizione le funzioni pe il calcolo degli integrali singolo, doppio, triplo, multiplo, quadrata gaussiana, Romberg, trapezoidale e regole di Simpson.

```
from scipy import integrate
# Take f(x) function as f
f = lambda x : x**2
#Single integration with a = 0 & b = 1
integration = integrate.quad(f, 0 , 1)
#Print content of integration
print("integration:{}".format(integration))
```

```
integration: (0.33333333333333337, 3.700743415417189e-15)
```

Code: 35. SciPy_Integrazione_numerica_1.py

Integrazione numerica (2)

- La libreria **scipy.integrate** ha a disposizioni integrazioni singole, doppie, triple, multiple, quadrata gaussiana, Romberg, trapezoidale e regole di Simpson.

```python
from scipy import integrate
#Import square root function from math lib
from math import sqrt
# set  function f(x)
f = lambda x, y : 64 *x*y
# Lower limit of second integral
p = lambda x : 0
# Upper limit of first integral
q = lambda y : sqrt(1 - 2*y**2)
# Perform double integration
integration = integrate.dblquad(f , 0 , 2/4,  p, q)
#Print content of integration
print("integration: {}".format(integration))
```

```
integration: (3.0, 9.657432734515774e-14)
```

Code: 36. SciPy_Integrazione_numerica_2.py