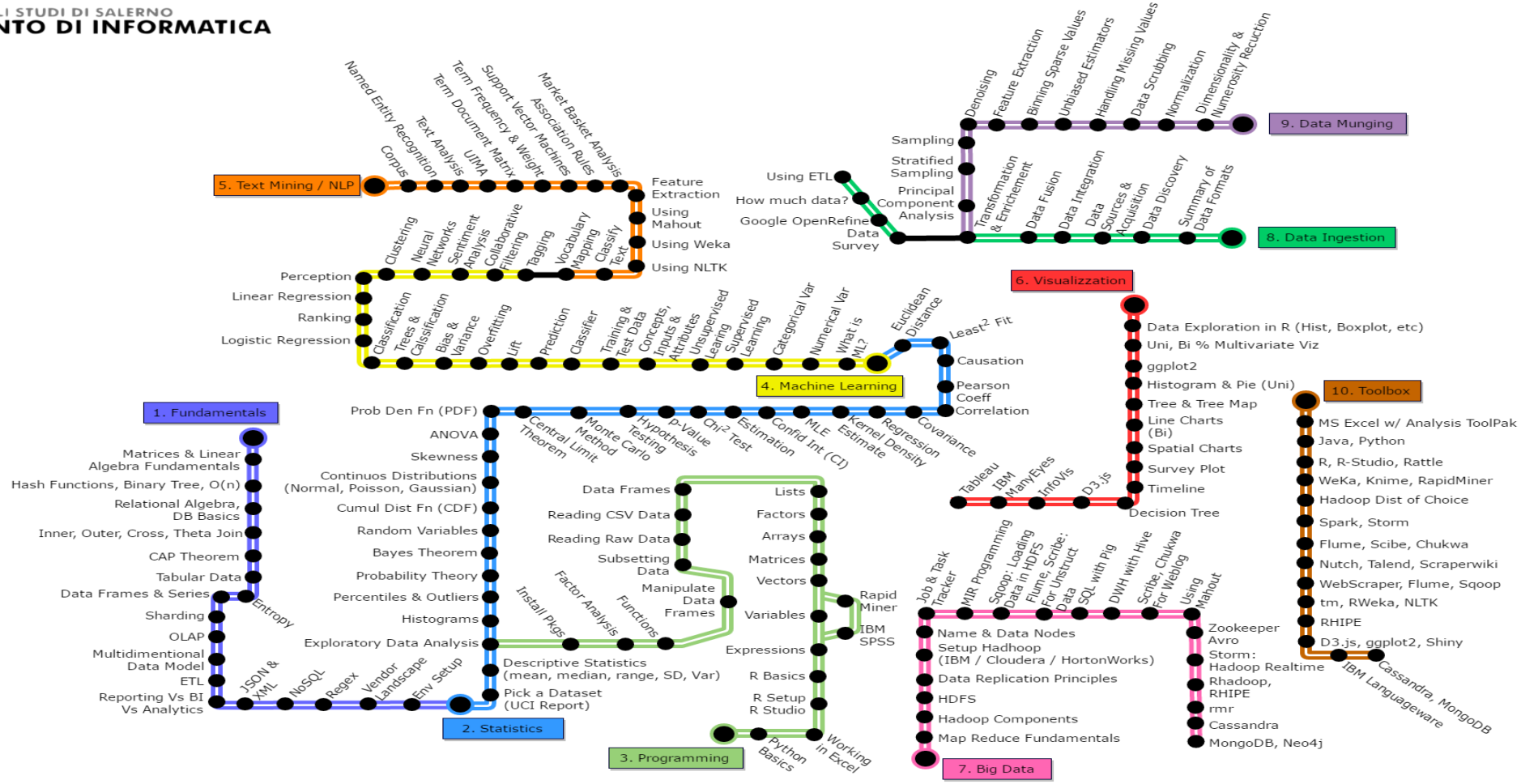
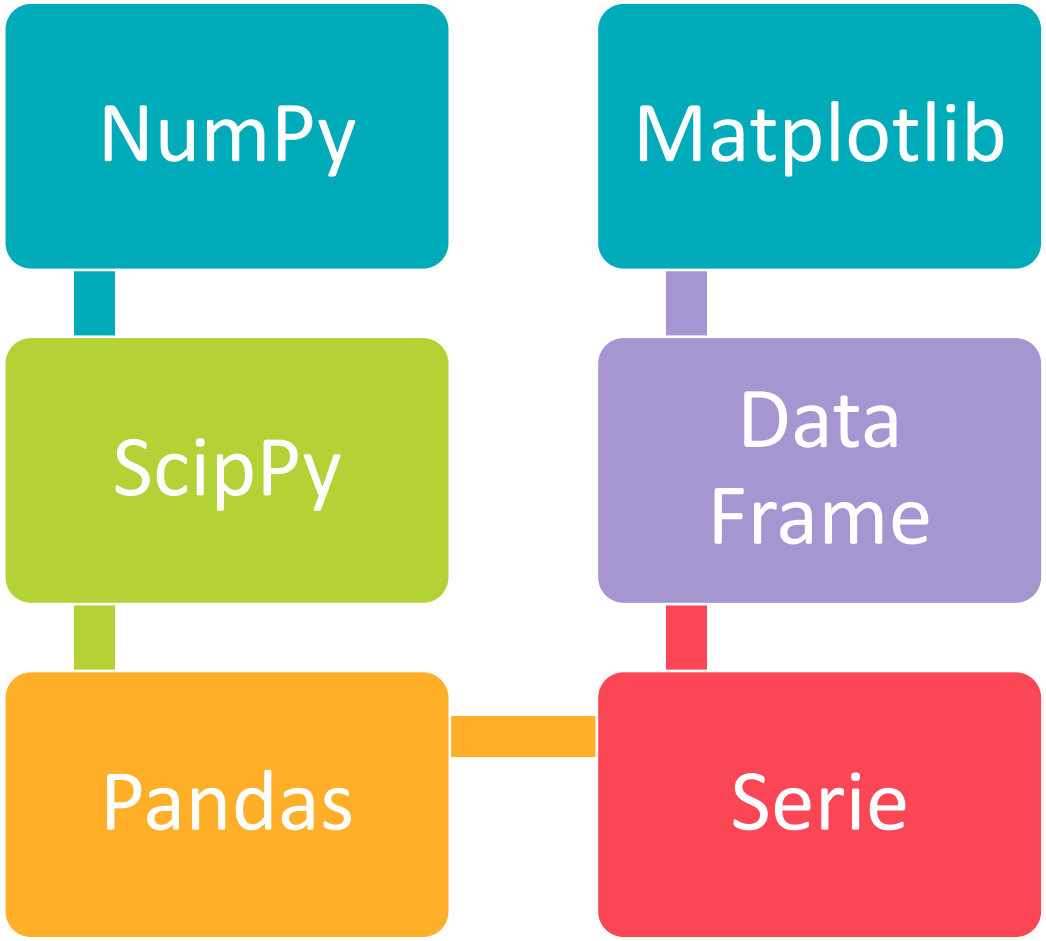


Fondamenti di Data Science e Machine Learning

Python Libraries

Prof. Giuseppe Polese, a.a. 2024-25

Outline



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**

```
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

```
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

```
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

```
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()**

```
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**

```
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
```

Code: 28. SciPy_versione.py

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: (0.3333333333333337, 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.

```
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

Pandas

3

Pandas

- **Pandas** è una libreria open source che consente di eseguire la manipolazione dei dati in Python
- La libreria **Pandas** è costruita su NumPy, il che significa che **Pandas** ha bisogno di NumPy per funzionare
- **Pandas** fornisce un modo semplice per creare, manipolare e gestire i dati
- **Pandas** è anche una soluzione elegante per i dati di serie temporali

Perchè usare Pandas?

- Gestisce facilmente i dati mancanti
- Utilizza **Series** per la struttura dati unidimensionale e **DataFrame** per la struttura dati multidimensionale
- Fornisce un modo efficiente per suddividere i dati
- Fornisce un modo flessibile per unire, concatenare o rimodellare i dati
- Include un potente strumento di serie temporali con cui lavorare

Versione Pandas

- Come controllare la versione **Pandas**

```
import pandas as pd

print("Pandas version:{}".format(pd.__version__))
```

```
Pandas version: 0.23.4
```

Code: 37. Pandas_version.py

Che cos'è un DataFrame?

- Un **DataFrame** è una matrice bidimensionale, con assi etichettati (righe e colonne), e si definisce come un modo per archiviare i dati
- Il **DataFrame** è ben noto agli statistici e ad altri professionisti dei dati
  - Un **DataFrame** è un dato tabellare, con righe per memorizzare le informazioni e colonne per denominare le informazioni
- Ad esempio, il **prezzo** può essere il nome di una colonna e 2, 3, 4 i valori del prezzo

Che cos'è una Series?

- Una **serie** è una struttura dati unidimensionale
- Può avere qualsiasi struttura di dati come intero, **float** e **string**
- È utile quando si desidera eseguire calcoli o restituire una matrice unidimensionale
- Una **serie** non può avere più colonne

Definizione di Series

- Le **serie** hanno un parametro
  - I dati possono essere un elenco, un dizionario o un valore scalare

```
import pandas as pd
#Definition of serie of floats
serie_1 = pd.Series([1., 2., 3.])
#Print content of serie_1 with index
print ("serie_1:")
print (serie_1)
#Definition of serie of floats
serie_2 = pd.Series([1., 2., 3.], index=['a', 'b', 'c'])
#Print content of serie_2
print("serie_2:")
print(serie_2)
```

```
serie_1:
0    1.0
1    2.0
2    3.0
dtype: float64
serie_2:
a    1.0
b    2.0
c    3.0
dtype: float64
```

Code: 38. Pandas_series.py

Creare un Data Frame (1)

- È possibile convertire un array **NumPy** in un **DataFrame** pandas con i

pandas.DataFrame(); e viceversa utilizzando **numpy.array()**

```
import numpy as np
import pandas as pd
#Define 2d NumPy array
arr_2d = [(1,2),(3,4)]
#Convert 2d Numpy array to pandas Data frame
df_arr_2d = pd.DataFrame(arr_2d)
#Print content of df_arr_2d
print("Data Frame:")
print(df_arr_2d)
```

```
Data Frame:
   0  1
0  1  2
1  3  4
```

Code: 39. Pandas_dataFrame_1.py

Creare un Data Frame (2)

- È possibile convertire un array **NumPy** in un **DataFrame** pandas con i

pandas.DataFrame(); e viceversa utilizzando **numpy.array()**

```
#Convert pandas Data frame to 2d Numpy array
arr = np.array(df_arr_2d)
#Print content of arr
print("Numpy array:")
print(arr)
#Define dictionary dic
dic = {'Name': ["John", "Smith"], 'Age': [30, 40]}
#Convert dictionary to pandas Data Frame
df_dic= pd.DataFrame(dic)
#Print content of df_dic
print("Data Frame with dictionary:")
print(df_dic)
```

```
Numpy array:
[[1 2]
 [3 4]]
Data Frame with dictionary:
    Name  Age
0   John   30
1  Smith   40
```

Code: 40. Pandas_dataFrame_2.py

Range dei Dati

- **Pandas** ha una comoda API per creare un intervallo di date
- **pandas.data_range(date,period,frequency):**
  - Il primo parametro è la data di inizio
  - Il secondo parametro è il numero di periodi (facoltativo se è specificata la data di fine)
  - l'ultimo parametro è la frequenza: giorno: 'D,' mese: 'M' e anno: 'Y'

Data_range()

- **pandas.data_range(date,period,frequency)**

```
import pandas as pd
#Create date using pandas.data_range()
dates_d = pd.date_range('20300101', periods=6, freq='D')
#Print content of dates_d
print('Day:', dates_d)
```

```
Day: DatetimeIndex(['2030-01-01', '2030-01-02', '2030-01-03', '2030-01-04',
                    '2030-01-05', '2030-01-06'],
                   dtype='datetime64[ns]', freq='D')
```

Code: 41. Pandas_data_range.py

Controllo dei dati (1)

- È possibile controllare la testa o la coda del dataset con **head()** o **tail()**

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
# Use dates_m as an index for the data frame
# Each row will be given a "name" or an index, corresponding to a date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Use method tail()
print ("tail:")
print(df.tail(1))
#Use method head()
print("head:")
print(df.head(1))
```

Code: 42. Pandas_controllo_dati_1.py

Controllo dei dati (2)

- È possibile controllare la testa o la coda del dataset con **head()** o **tail()**

```
Data Frame:
                   A         B         C         D
2030-01-31 -0.335212  1.082534  0.468492 -0.249135
2030-02-28  1.246830 -0.566842  0.290878  1.033448
2030-03-31  0.268166 -0.163397 -0.076707 -1.586657
2030-04-30  0.355040 -1.689920  1.640998 -0.722774
2030-05-31  0.497199 -0.273865  0.041032 -0.914333
2030-06-30  1.367342 -1.113279  1.091821 -1.077072

head:
                   A         B         C         D
2030-01-31 -0.335212  1.082534  0.468492 -0.249135

tail:
                   A         B         C         D
2030-06-30  1.367342 -1.113279  1.091821 -1.077072
```

Code: 42. Pandas_controllo_dati.py

Controllo dei dati (3)

- Una pratica eccellente per avere un indizio sui dati è usare **describe()**
  - Fornisce **counts**, **mean**, **std**, **min**, **max** e **percentage** del set di dati

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
# Use dates_m as an index for the data frame
# Each row will be given a "name" or an index, corresponding to a date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Use method describe()
print ("dataset clues:")
print(df.describe())
```

Code: 43. Pandas_controllo_dati_2.py

Controllo dei dati (4)

- Una pratica eccellente per avere un indizio sui dati è usare **describe()**
  - Fornisce **counts**, **mean**, **std**, **min**, **max** e **percentage** del set di dati

Data Frame:

	A	B	C	D
2030-01-31	-0.519106	1.785377	-0.173451	0.530234
2030-02-28	-0.379216	-0.267312	0.009829	-0.417903
2030-03-31	-1.479765	1.050694	1.147479	0.346487
2030-04-30	1.121995	-0.162147	0.836656	-0.165249
2030-05-31	0.211253	-0.318291	2.075698	-1.923282
2030-06-30	-0.140122	0.822066	-0.864400	-2.764445

dataset clues:

	A	B	C	D
count	6.000000	6.000000	6.000000	6.000000
mean	-0.197493	0.485065	0.505302	-0.732360
std	0.859313	0.866562	1.055520	1.321077
min	-1.479765	-0.318291	-0.864400	-2.764445
25%	-0.484134	-0.241021	-0.127631	-1.546937
50%	-0.259669	0.329959	0.423242	-0.291576
75%	0.123410	0.993537	1.069773	0.218553
max	1.121995	1.785377	2.075698	0.530234

Code: 43. Pandas_controllo_dati.py

Slice(Parte) dei dati (1)

- È possibile utilizzare il nome della colonna per **estrarre** i dati in una determinata colonna

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
# Use dates_m as an index for the data frame
# Each row will be given a "name" or an index, corresponding to a date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Print column using name
print ("first column:")
print(df['A'])
```

Code: 44. Pandas_slice_dati_1.py

Slice(Parte) dei dati (2)

- È possibile utilizzare il nome della colonna per **estrarre** i dati in una determinata colonna

```
Data Frame:
                   A         B         C         D
2030-01-31 -1.270479 -0.619234 -0.519960  0.943866
2030-02-28  1.262097  0.927393  1.065476 -1.353677
2030-03-31  0.771563  0.394064 -0.426624 -0.391374
2030-04-30 -1.826723 -1.749747  0.835737  0.368412
2030-05-31 -1.165080 -0.356957 -1.041455  0.031599
2030-06-30 -0.276490 -0.238573 -1.574928 -0.634111
first colums:
2030-01-31   -1.270479
2030-02-28    1.262097
2030-03-31    0.771563
2030-04-30   -1.826723
2030-05-31   -1.165080
2030-06-30   -0.276490
Freq: M, Name: A, dtype: float64
```

Code: 44. Pandas_slice_dati_1.py

Slice(Parte) dei dati (3)

- Per selezionare più colonne, è necessario utilizzare due volte la parentesi, [[..,..]]
  - La prima coppia di parentesi indica che si desidera selezionare le colonne
  - La seconda coppia di parentesi indica quali colonne si desidera restituire

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
# Use dates_m as an index for the data frame
# Each row will be given a "name" or an index, corresponding to a date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Print column using name
print ("first two colums:")
print(df[['A','B']])
```

Code: 45. Pandas_slice_dati_2.py

Slice(Parte) dei dati (4)

- Per selezionare più colonne, è necessario utilizzare due volte la parentesi, [[..,..]]
  - La prima coppia di parentesi indica che si desidera selezionare le colonne
  - La seconda coppia di parentesi indica quali colonne si desidera restituire

```
Data Frame:
                   A         B         C         D
2030-01-31 -1.124070  1.170906 -0.762710  0.371729
2030-02-28 -0.124767  0.046144  0.798957  0.166812
2030-03-31 -0.295791 -0.348615 -0.240716 -0.142574
2030-04-30  1.108678  1.259876  0.881061 -1.021890
2030-05-31  0.452371  0.725347 -0.894437 -0.022574
2030-06-30 -0.985186  0.032864  0.601079 -1.107904
first two colums:
                   A         B
2030-01-31 -1.124070  1.170906
2030-02-28 -0.124767  0.046144
2030-03-31 -0.295791 -0.348615
2030-04-30  1.108678  1.259876
2030-05-31  0.452371  0.725347
2030-06-30 -0.985186  0.032864
```

Code: 45. Pandas_slice_dati_2.py

Slice(Parte) dei dati (5)

- È possibile suddividere le righe

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
# Use dates_m as an index for the data frame
# Each row will be given a "name" or an index, corresponding to a
date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Print column using name
print ("first three rows:")
print(df[0:3])
```

Code: 46. Pandas_slice_dati_3.py

Slice(Parte) dei dati (6)

- È possibile suddividere le righe

```
Data Frame:
                   A         B         C         D
2030-01-31 -0.447362 -1.275389 -0.334829 -0.655417
2030-02-28  0.438954  0.483098 -0.508109 -1.591590
2030-03-31 -0.012803 -0.205406  0.722621 -1.032293
2030-04-30  0.744750 -1.544801  0.863565 -0.416415
2030-05-31  0.477229 -0.449814  1.087759 -0.023196
2030-06-30  0.260484  1.331865 -0.551913  0.310608
first three rows:
                   A         B         C         D
2030-01-31 -0.447362 -1.275389 -0.334829 -0.655417
2030-02-28  0.438954  0.483098 -0.508109 -1.591590
2030-03-31 -0.012803 -0.205406  0.722621 -1.032293
```

Code: 46. Pandas_slice_dati_3.py

loc[] function (1)

- La funzione **loc[]** viene utilizzata per selezionare le colonne in base a nomi e criteri di selezione per i dati

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
# Use dates_m as an index for the data frame
# Each row will be given a "name" or an index, corresponding to a date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Print rows with attribute A > 0
print ("rows with attribute A > 0:")
print(df.loc[df['A'] > 0])
```

Code: 47. Pandas_loc.py

loc[] function (2)

- La funzione **loc[]** viene utilizzata per selezionare le colonne in base a nomi e criteri di selezione per i dati

```
Data Frame:
                   A         B         C         D
2030-01-31 -0.898388 -0.008761  0.690172  1.129059
2030-02-28  1.462090 -0.942333  0.820440 -0.327735
2030-03-31 -2.302688  1.107672  1.127441  0.241374
2030-04-30  0.478878  0.058691 -2.833534  0.518210
2030-05-31  0.723496 -0.735732  0.958378 -1.143461
2030-06-30 -0.723879 -0.674280 -0.314789  0.539755
rows with attribute A > 0:
                   A         B         C         D
2030-02-28  1.462090 -0.942333  0.820440 -0.327735
2030-04-30  0.478878  0.058691 -2.833534  0.518210
2030-05-31  0.723496 -0.735732  0.958378 -1.143461
```

Code: 47. Pandas_loc.py

Eliminare una colonna (1)

- È possibile eliminare le colonne utilizzando **pandas pd.drop()**

```
import numpy as np
import pandas as pd
#Create date using pandas.data_range()
dates_m = pd.date_range('20300101', periods=6, freq='M')
#Create a random sequence. The sequence has 4 columns and 6 rows
random = np.random.randn(6,4)
#Use dates_m as an index for the data frame
#Each row will be given a "name" or an index, corresponding to a date.
df = pd.DataFrame(random, index=dates_m,columns=list('ABCD'))
#Print content of df
print("Data Frame:")
print(df)
#Drop colums A and C from df
df_1=df.drop(columns=['A', 'C'])
#Print content of df_1
print("Data Frame after drop:")
print(df_1)
```

Code: 48. Pandas_drop.py

Eliminare una colonna (2)

- È possibile eliminare le colonne utilizzando **pandas pd.drop()**

```
Data Frame:
                   A         B         C         D
2030-01-31 -0.463147 -0.908536 -2.006566  0.250268
2030-02-28  0.633078  0.129734  0.200273  0.074995
2030-03-31  0.289186  0.984765 -0.857520  0.053942
2030-04-30  2.148310  0.073121 -1.705877  0.862170
2030-05-31  0.613084 -0.013009  0.293692 -0.991823
2030-06-30  0.345148 -0.973410 -0.418604 -0.651093
Data Frame after drop:
                   B         D
2030-01-31 -0.908536  0.250268
2030-02-28  0.129734  0.074995
2030-03-31  0.984765  0.053942
2030-04-30  0.073121  0.862170
2030-05-31 -0.013009 -0.991823
2030-06-30 -0.973410 -0.651093
```

Code: 48. Pandas_drop.py

Concatenazione (1)

- È possibile concatenare due dataframe utilizzando **pandas.concat()**

```
import numpy as np
import pandas as pd
#Define first Data Frame
df1 = pd.DataFrame({'name': ['John', 'Smith','Paul'],'Age': ['25', '30',
'50']},index=[0, 1, 2])
#Print content of df1
print("Data Frame 1:")
print(df1)
#Define second Data Frame
df2 = pd.DataFrame({'name': ['Adam', 'Smith' ],'Age': ['26', '11']},index=[3, 4])
#Print content of df2
print("Data Frame 2:")
print(df2)
df_concat = pd.concat([df1,df2])
#Print content of df_concat
print("Data Frame concat:")
print(df_concat)
```

Code: 49. Pandas_concat.py

Concatenazione (2)

- È possibile concatenare due dataframe utilizzando **pandas.concat()**

```
Data Frame 1:
    name Age
0   John  25
1  Smith  30
2   Paul  50
Data Frame 2:
    name Age
3   Adam  26
4  Smith  11
```

```
Data Frame concat:
    name Age
0   John  25
1  Smith  30
2   Paul  50
3   Adam  26
4  Smith  11
```

Code: 49. Pandas_concat.py

Drop_duplicates (1)

- Se un set di dati contiene informazioni duplicate **drop_duplicates()** è una funzione semplice per escludere righe duplicate

```
import numpy as np
import pandas as pd
#Define first Data Frame
df1 = pd.DataFrame({'name': ['John', 'Smith','Paul'],'Age': ['25', '30', '50']},index=[0, 1, 2])
#Print content of df1
print("Data Frame 1:"); print(df1)
#Define second Data Frame
df2 = pd.DataFrame({'name': ['Adam', 'Smith' ],'Age': ['26', '11']},index=[3, 4])
#Print content of df2
print("Data Frame 2:");print(df2)
df_concat = pd.concat([df1,df2])
#Print content of df_concat
print("Data Frame concat:"); print(df_concat)
#Drop duplicate rows with drop_duplicates()
df_concat_nd = df_concat.drop_duplicates('name')
#Print content of df_concat_nd
print ("Data Frame without duplicate names:"); print(df_concat_nd)
```

Code: 50. Pandas_duplicates.py

Drop_duplicates (2)

- Se un set di dati contiene informazioni duplicate **drop_duplicates()** è una funzione semplice per escludere righe duplicate

```
Data Frame 1:
    name Age
0   John  25
1  Smith  30
2   Paul  50
Data Frame 2:
    name Age
3   Adam  26
4  Smith  11
```

```
Data Frame concat:
    name Age
0   John  25
1  Smith  30
2   Paul  50
3   Adam  26
4  Smith  11
Data Frame without duplicate names:
    name Age
0   John  25
1  Smith  30
2   Paul  50
3   Adam  26
```

Code: 50. Pandas_duplicates.py

Sort_values() (1)

- È possibile ordinare i valori con **sort_values()**

```
import numpy as np
import pandas as pd
#Define first Data Frame
df1 = pd.DataFrame({'name': ['John', 'Smith','Paul'],'Age': ['25', '30', '50’]},
index=[0, 1, 2])
#Define second Data Frame
#Print content of df1
print("Data Frame 1:");print(df1)
df2 = pd.DataFrame({'name': ['Adam', 'Smith' ],'Age': ['26', '11']},index=[3, 4])
#Print content of df2
print("Data Frame 2:");print(df2)
df_concat = pd.concat([df1,df2])
#Print content of df_concat
print("Data Frame concat:");print(df_concat)
#Sorting values by Age
df_concat_st = df_concat.sort_values('Age')
#Print content of df_concat_st
print("Data Frame sorting by Age: ");print(df_concat_st)
```

Code: 51. Pandas_sort.py

Sort_values() (2)

- È possibile ordinare i valori con **sort_values()**

```
Data Frame 1:
    name Age
0   John  25
1  Smith  30
2   Paul  50
Data Frame 2:
    name Age
3   Adam  26
4  Smith  11
```

```
Data Frame concat:
    name Age
0   John  25
1  Smith  30
2   Paul  50
3   Adam  26
4  Smith  11
Data Frame sorting by Age:
    name Age
4  Smith  11
0   John  25
3   Adam  26
1  Smith  30
2   Paul  50
```

Code: 51. Pandas_sort.py

Rinomina: modifica dell'indice (1)

- È possibile utilizzare **rename()** per rinominare una colonna in Pandas
  - Il primo valore è il nome della colonna corrente e il secondo valore è il nuovo nome della colonna

```
import numpy as np
import pandas as pd
#Define first Data Frame
df1 = pd.DataFrame({'name': ['John', 'Smith','Paul'],'Age': ['25', '30', '50']},index=[0, 1, 2])
#Define second Data Frame
#Print content of df1
print("Data Frame 1:"); print(df1)
df2 = pd.DataFrame({'name': ['Adam', 'Smith' ],'Age': ['26', '11']},index=[3, 4])
#Print content of df2
print("Data Frame 2:"); print(df2)
df_concat = pd.concat([df1,df2])
#Print content of df_concat
print("Data Frame concat:"); print(df_concat)
#Rename Data Frame
df_concat_rename = df_concat.rename(columns={"name": "Surname", "Age": "Age_ppl"})
#Print content of df_concat_remane
print("Data Frame after remane: ");print(df_concat_rename)
```

Code: 52. Pandas_rename_index.py

Rinomina: modifica dell'indice (2)

- È possibile utilizzare **rename()** per rinominare una colonna in Pandas
  - Il primo valore è il nome della colonna corrente e il secondo valore è il nuovo nome della colonna

```
Data Frame 1:
    name Age
0   John  25
1  Smith  30
2   Paul  50
Data Frame 2:
    name Age
3   Adam  26
4  Smith  11
Data Frame concat:
    name Age
0   John  25
1  Smith  30
2   Paul  50
3   Adam  26
4  Smith  11
```

```
Data Frame after remane:
  Surname Age_ppl
0    John      25
1   Smith      30
2    Paul      50
3    Adam      26
4   Smith      11
```

Code: 52. Pandas_rename_index.py

Importare un CSV (1)

- Per importare un set di dati CSV, è possibile utilizzare l'oggetto **pandas.read_csv()**
- *pandas.read_csv(filepath_or_buffer, sep, names, index_col, skipinitialspace)*
  - **filepath_or_buffer**: percorso o URL con i dati
  - **sep**: Definire il delimitatore da utilizzare
  - ***names***: assegnare un nome alle colonne. Se il set di dati ha dieci colonne, è necessario passare dieci nomi
  - **index_col**: Se True, la prima colonna viene utilizzata come indice di riga
  - **skipinitialspace**: ignora gli spazi dopo il delimitatore

Importare un CSV (2)

- **pandas.read_csv()**

```
import pandas as pd
#Define attributes of CSV file
COLUMNS = ['age','workclass', 'fnlwgt', 'education', 'education_num','marital',
'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss',
'hours_week', 'native_country', 'label']

#Define path of CSV file
PATH = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
df_train = pd.read_csv(PATH, skipinitialspace=True, names = COLUMNS, index_col=False)

#Print shape of df_train
print("df_train shape: {} ".format(df_train.shape))
```

```
df_train shape: (32561, 15)

Process finished with exit code 0
```

Code: 53. Pandas_import_csv.py

Group By (1)

- Un modo semplice per visualizzare i dati consiste nell'utilizzare il metodo **groupby**
  - Questo metodo può aiutarti a riepilogare i dati per gruppo
- Di seguito è riportato un elenco di metodi disponibili con **groupby**:
  - **conteggio**: count()
  - **min**: min()
  - **max**: max()
  - **media**: media()
  - **Mediana**: Mediana()
  - **Deviazione standard**: sdt()
  - Ecc..

Group By (2)

- All'interno di **groupby()**, è possibile utilizzare la colonna a cui si desidera applicare il metodo

```
import pandas as pd
#Define attributes of CSV file
COLUMNS = ['age','workclass', 'fnlwgt', 'education', 'education_num',
        'marital', 'occupation', 'relationship', 'race', 'sex','capital_gain',
'capital_loss', 'hours_week', 'native_country','label']

#Define path of CSV file
PATH = "https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data"
df_train = pd.read_csv(PATH, skipinitialspace=True, names = COLUMNS,
index_col=False)

#Application of groupby() function
df_gb = df_train.groupby(['label']).mean()

#Print content of df_gb
print ("Data Frame after groupby:")
print(df_gb)
```

```
Data Frame after groupby:
                age        fnlwgt     ...      capital_loss  hours_week
label                                 ...
<=50K  36.783738  190340.86517     ...         53.142921   38.840210
>50K   44.249841  188005.00000     ...        195.001530   45.473026

[2 rows x 6 columns]
```

Code: 54. Pandas_group_by_1.py

Group By (3)

- All'interno di **groupby()**, è possibile utilizzare la colonna a cui si desidera applicare il metodo

```
import pandas as pd
#Define attributes of CSV file
COLUMNS = ['age','workclass', 'fnlwgt', 'education', 'education_num', 'marital',
'occupation', 'relationship', 'race', 'sex', 'capital_gain', 'capital_loss',
'hours_week', 'native_country',       'label']
#Define path of CSV file
PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
df_train = pd.read_csv(PATH, skipinitialspace=True, names = COLUMNS,
index_col=False)
#Application of groupby() function
df_gb = df_train.groupby(['label'])['age'].min()
#Print content of df_gb
print ("Data Frame after groupby:")
print(df_gb)
```

```
Data Frame after groupby:
label
<=50K    17
>50K     19
Name: age, dtype: int64
```

Code: 55. Pandas_group_by_2.py

Group By (4)

- All'interno di **groupby()**, è possibile utilizzare la colonna a cui si desidera applicare il metodo

```
import pandas as pd
#Define attributes of CSV file
COLUMNS = ['age','workclass', 'fnlwgt', 'education',
'education_num', 'marital', 'occupation', 'relationship', 'race',
'sex','capital_gain', 'capital_loss', 'hours_week',
'native_country', 'label']
#Define path of CSV file
PATH = "https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data"
df_train = pd.read_csv(PATH, skipinitialspace=True, names = COLUMNS,
index_col=False)
#Application of groupby() function
df_gb = df_train.groupby(['label', 'marital'])['capital_gain'].max()
#Print content of df_gb
print ("Data Frame after groupby:")
print(df_gb)
```

```
Data Frame after groupby:
label  marital
<=50K  Divorced                34095
       Married-AF-spouse        2653
       Married-civ-spouse      41310
       Married-spouse-absent    6849
       Never-married           34095
       Separated                7443
       Widowed                  6849
>50K   Divorced                99999
       Married-AF-spouse        7298
       Married-civ-spouse      99999
       Married-spouse-absent   99999
       Never-married           99999
       Separated               99999
       Widowed                 99999
Name: capital_gain, dtype: int64

Process finished with exit code 0
```

Code: 56. Pandas_group_by_3.py

Matplotlib

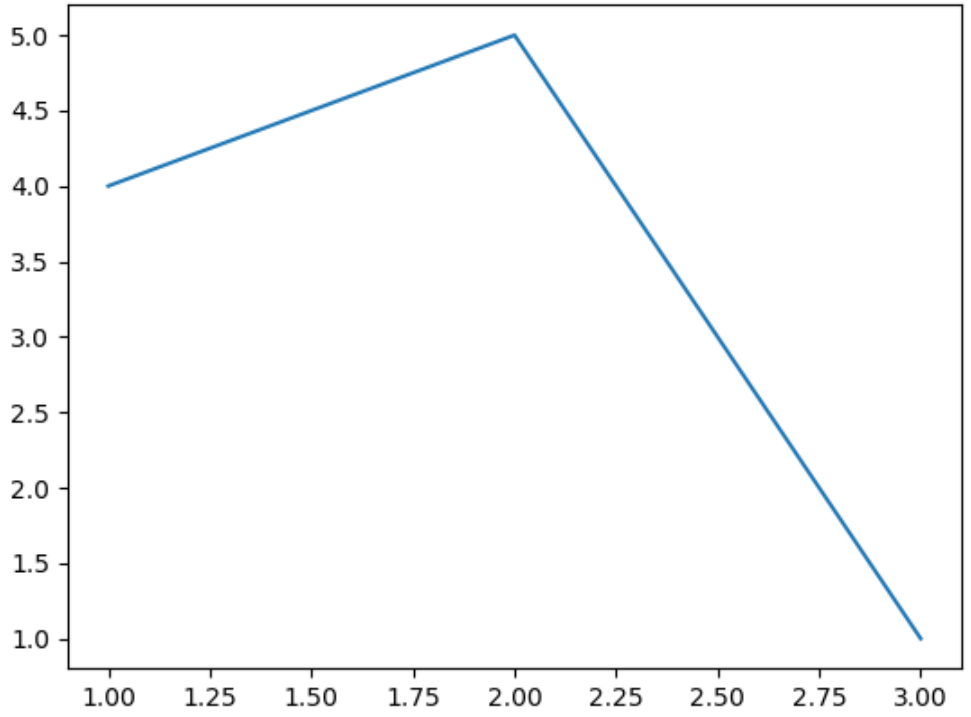
4

Matplotlib

- **matplotlib.pyplot** è una libreria di plotting usata per la grafica 2D in Python
- **matplot.pyplot** è una raccolta di funzioni di stile di comando che fanno funzionare **matplotlib** come MATLAB
- Ogni funzione **pyplot** apporta alcune modifiche a una figura
  - Crea una figura
  - Crea un'area di plottaggio in una figura
  - Traccia alcune linee in un'area di plottaggio
  - decora la trama con etichette
  - ecc

plot()

```
from matplotlib import pyplot as plt
#The first vector represents x axis, the second represents y axes
plt.plot([1,2,3],[4,5,1])
#Showing what we plotted
plt.show()
```



Code: 57. Matplotlib_plot.py

title(),xlabel() e ylabel() (1)

- **pyplot.title(), pyplot.xlabel()** and **pyplot.ylabel()**

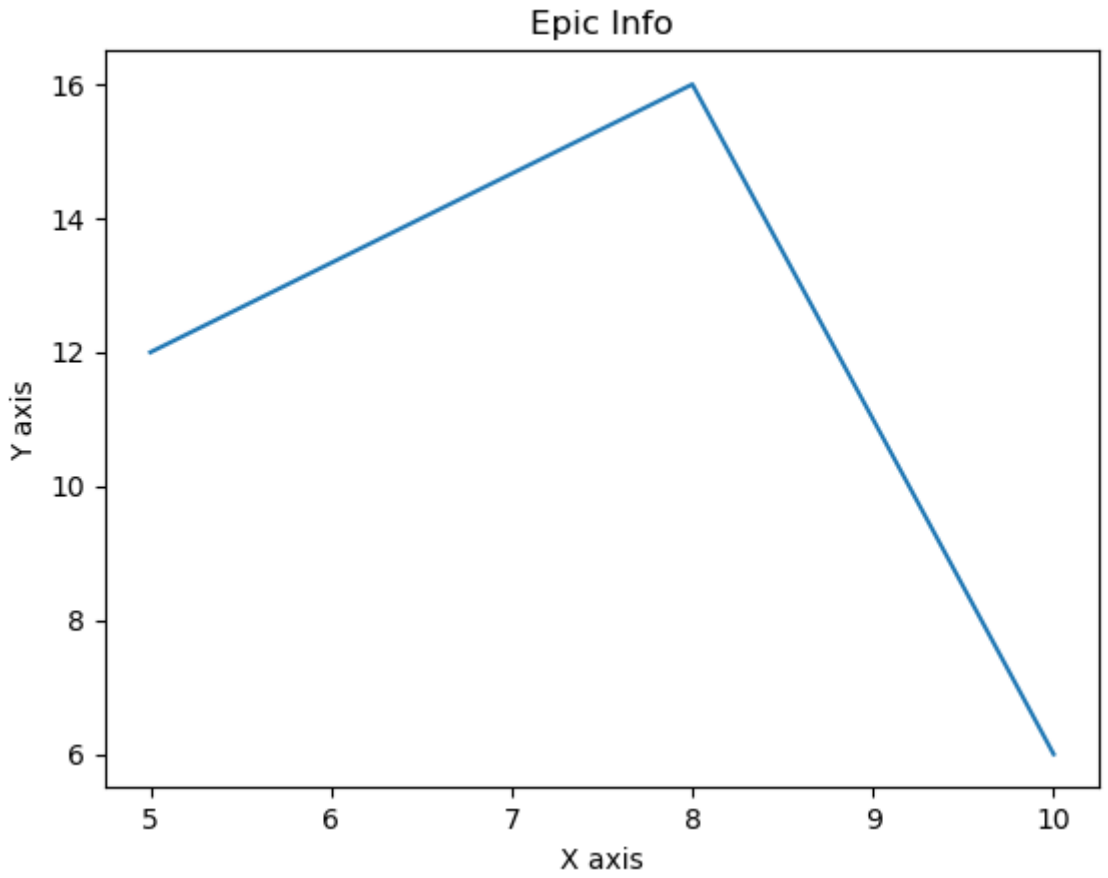
```
from matplotlib import pyplot as plt

x = [5,8,10]
y = [12,16,6]
#Use plot() function with two vector defined above
plt.plot(x,y)
#Definition of label for Title
plt.title('Epic Info')
#Definition of label for y axis
plt.ylabel('Y axis')
#Definition of label for x axis
plt.xlabel('X axis')
#Showing what we plotted
plt.show()
```

Code: 58. Matplotlib_title_xlabel_ylabel.py

title(),xlabel() e ylabel() (2)

- **pyplot.title(), pyplot.xlabel()** and **pyplot.ylabel()**

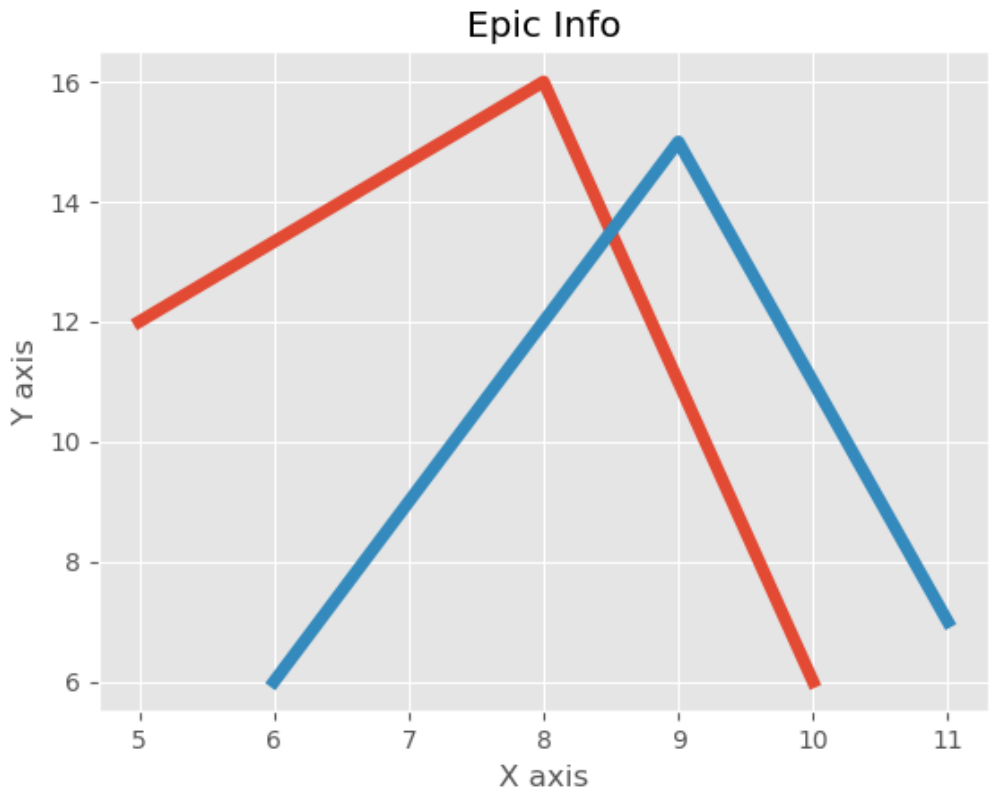


Code: 58. Matplotlib_title_xlabel_ylabel.py

Visualizzazione di più linee (1)

- **pyplot.plot()**

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
#Define first two vectors of points
x = [5,8,10]
y = [12,16,6]
#Define second two vectors of points
x2 = [6,9,11]
y2 = [6,15,7]
#Plot first line
plt.plot(x,y,linewidth=5)
#Plot second line
plt.plot(x2,y2,linewidth=5)
#Definition of label for Title
plt.title('Epic Info')
#Definition of label for y axis
plt.ylabel('Y axis')
#Definition of label for x axis
plt.xlabel('X axis')
#Showing what we plotted
plt.show()
```



Code: 59. Matplotlib_plot_multilines_1.py

Visualizzazione di più linee (2)

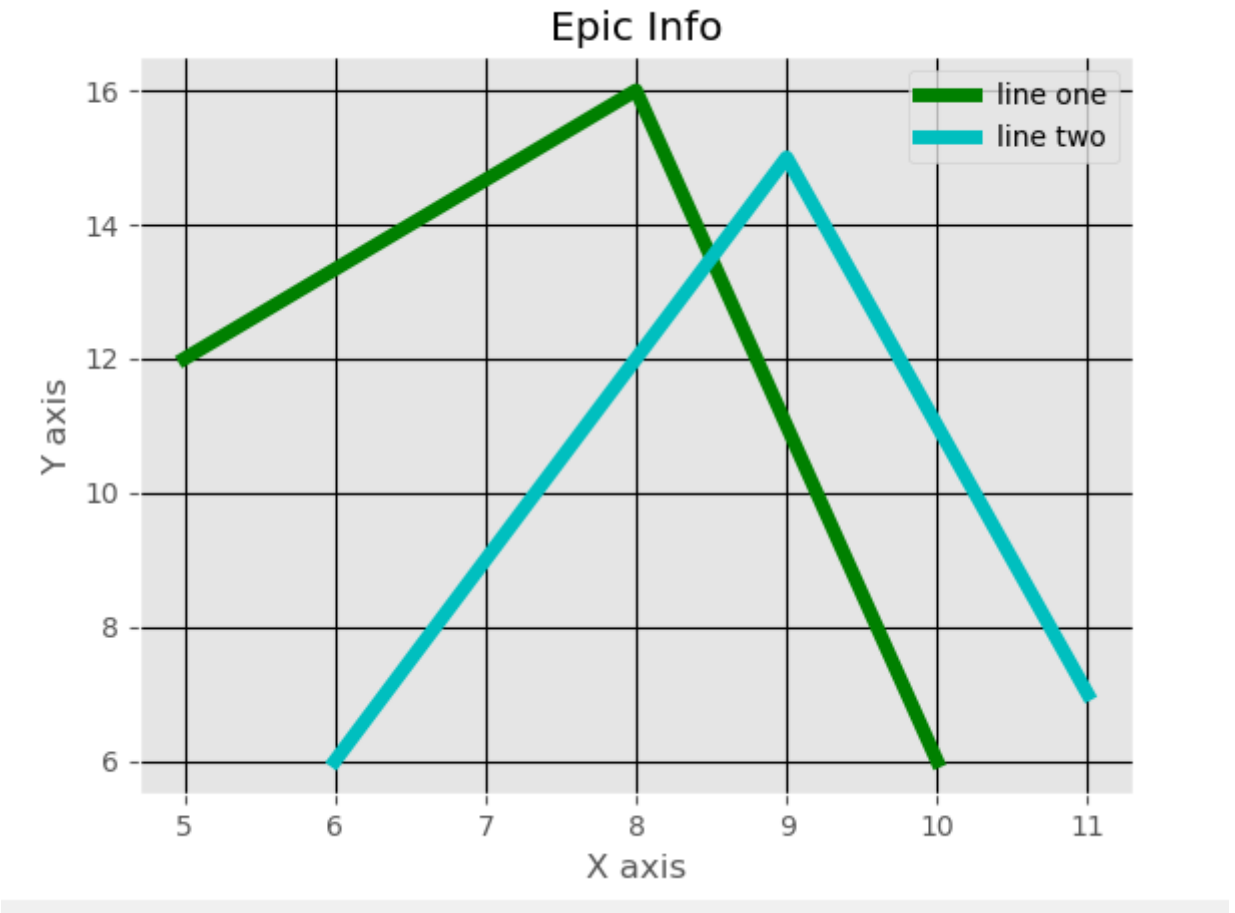
- **pyplot.plot()**

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot’)
#Define first two vectors of points
x = [5,8,10]; y = [12,16,6]
#Define second two vectors of points
x2 = [6,9,11];y2 = [6,15,7]
#Define first line with color green and label 'line one'
plt.plot(x,y,'g',label='line one', linewidth=5)
#Define second line with color celestial and label 'line two'
plt.plot(x2,y2,'c',label='line two',linewidth=5)
#Define of label for Title
plt.title('Epic Info')
#Define of label for y axis
plt.ylabel('Y axis')
#Define of label for x axis
plt.xlabel('X axis')
#Insert legend
plt.legend()
#Define grid with color specified by 'k'
plt.grid(True,color='k’);plt.show()
```

Code: 60. Matplotlib_plot_multilines_2.py

plottaggio di più linee (2)

- **pyplot.plot()**

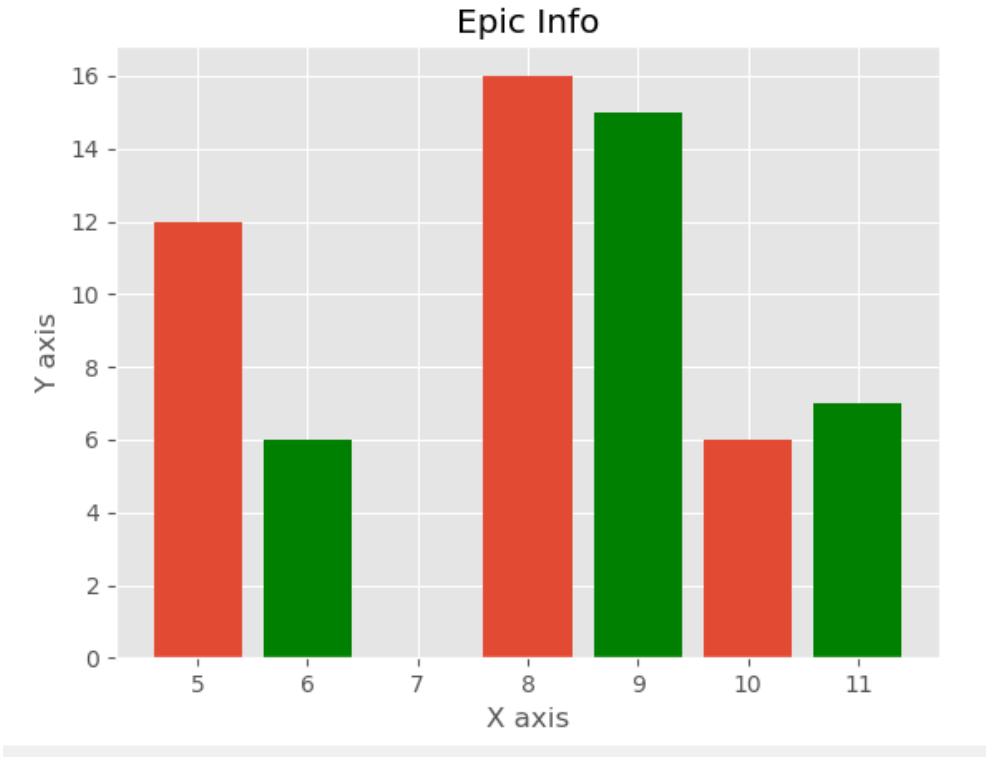


Code: 60. Matplotlib_plot_multilines_2.py

plottaggio di barre

- **pyplot.bar()**

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
#Define first two vectors of points
x = [5,8,10]; y = [12,16,6]
#Define second two vectors of points
x2 = [6,9,11]; y2 = [6,15,7]
#Plot first bar
plt.bar(x, y, align='center')
#Plot second bar
plt.bar(x2, y2, color='g', align='center')
#Definition of label for Title
plt.title('Epic Info')
#Definition of label for y axis
plt.ylabel('Y axis')
#Definition of label for x axis
plt.xlabel('X axis'); plt.show()
```

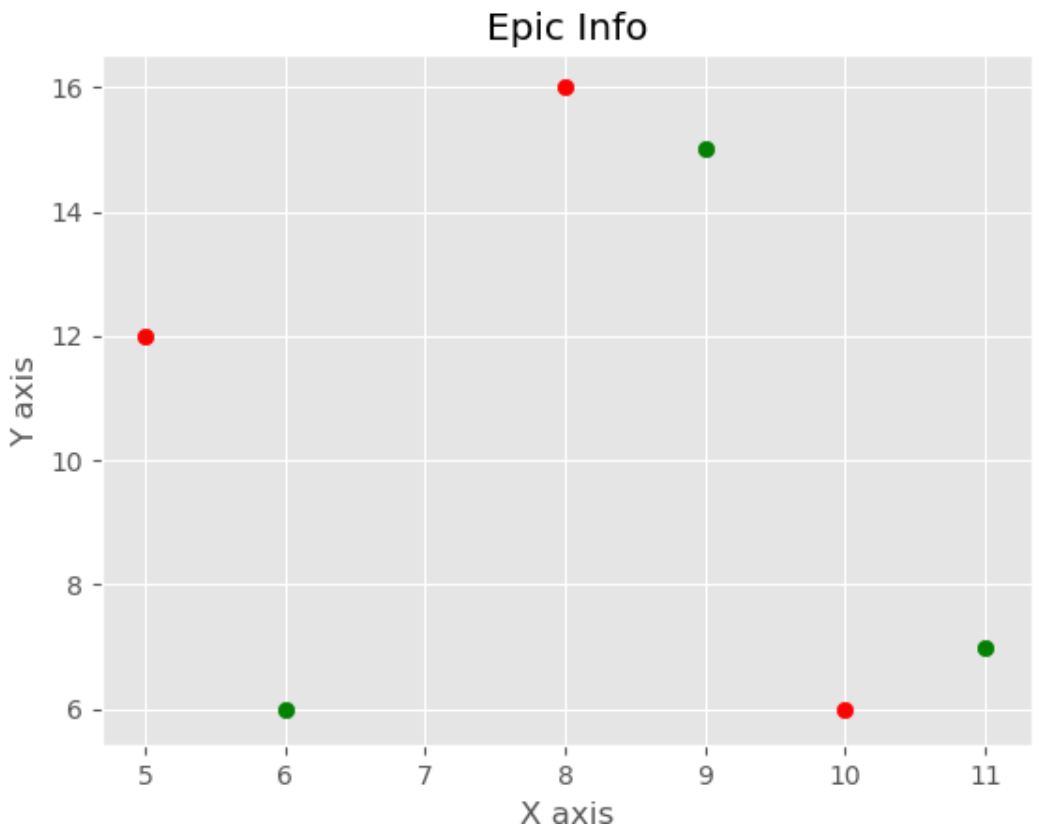


Code: 61. Matplotlib_plot_bars.py

plottaggio di punti

- **pyplot.scatter()**

```
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
#Define first two vectors of points
x = [5,8,10];y = [12,16,6]
#Define second two vectors of points
x2 = [6,9,11];y2 = [6,15,7]
#Plot first scatter
plt.scatter(x, y, color='r')
#Plot second scatter
plt.scatter(x2, y2, color='g')
#Definition of label for Title
plt.title('Epic Info')
#Definition of label for y axis
plt.ylabel('Y axis')
#Definition of label for x axis
plt.xlabel('X axis');plt.show()
```



Code: 62. Matplotlib_plot_points.py

plottaggio di DataFrame (1)

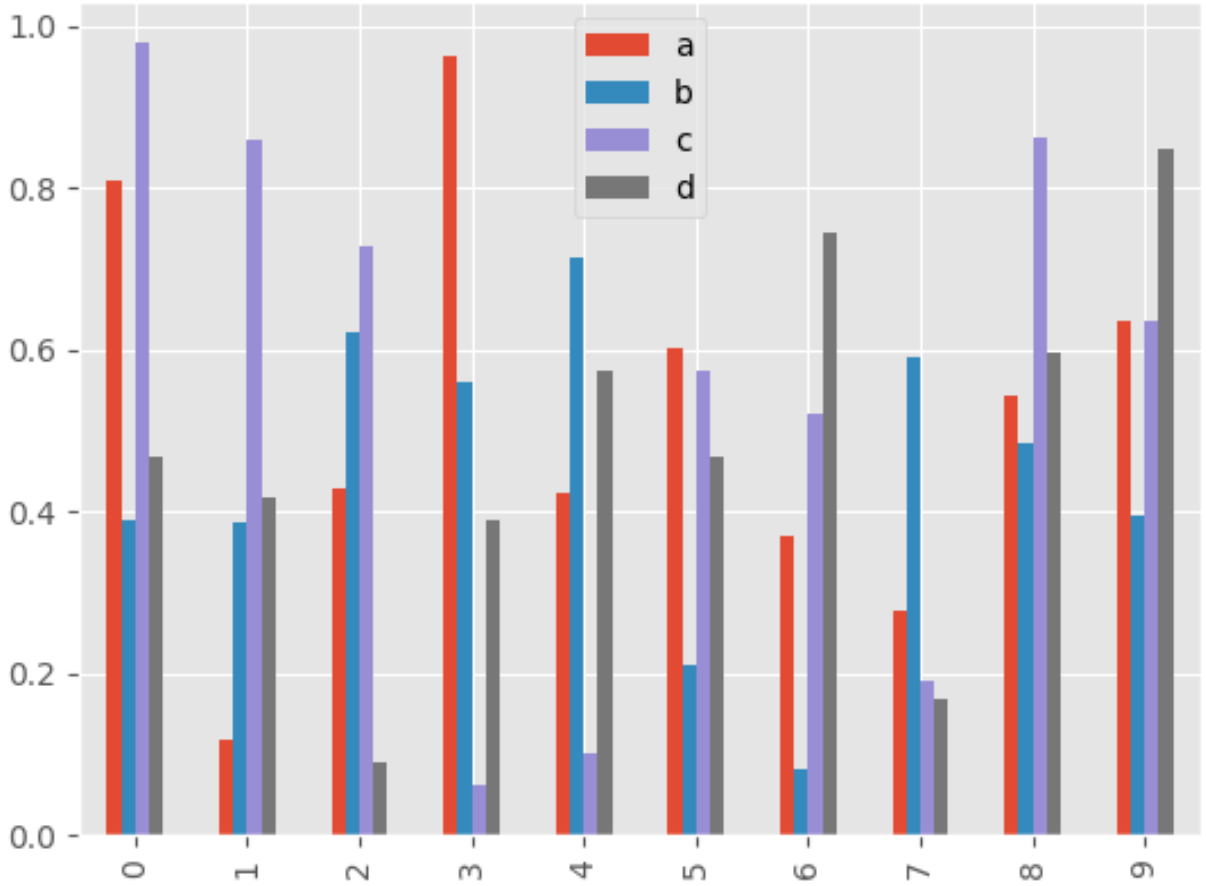
- **pandas.plot()**

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib import style
style.use('ggplot')
#Define Data Frame
df2 = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
#Plot Data Frame
df2.plot(kind='bar')
plt.show()
```

Code: 63. Matplotlib_Pandas_dataframe.py

plottaggio di DataFrame (2)

- **pandas.plot()**



Code: 63. Matplotlib_Pandas_dataframe.py

plottaggio di grafici a torta (1)

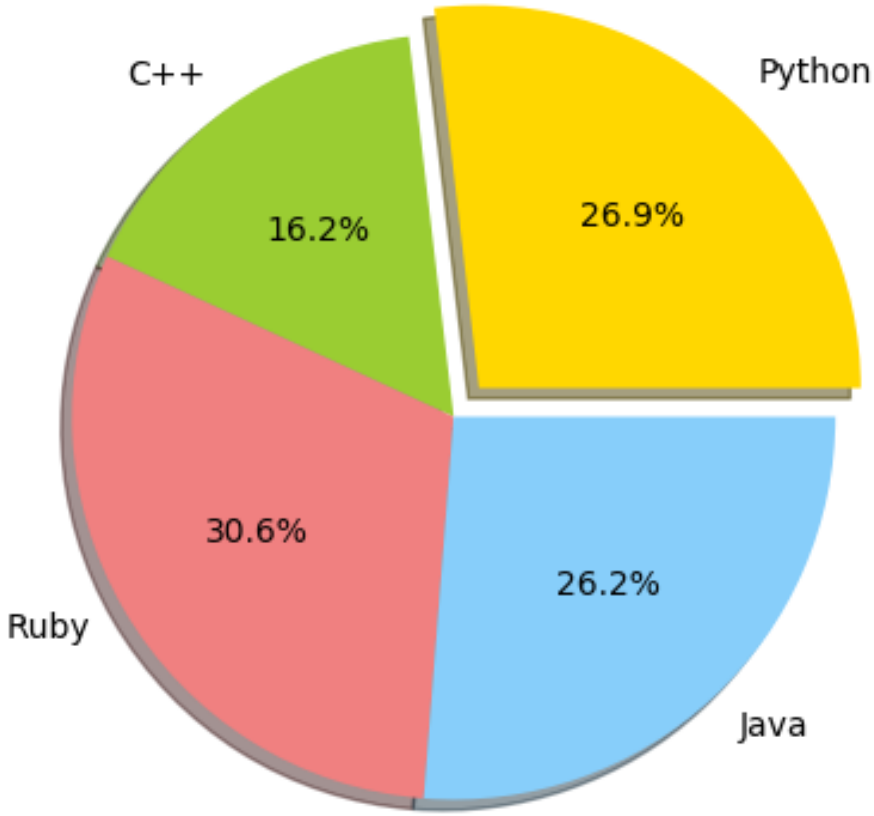
- **pyplot.pie()**

```
import matplotlib.pyplot as plt
# Data to plot
labels = 'Python', 'C++', 'Ruby', 'Java'
sizes = [215, 130, 245, 210]
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
# explode 1st slice
explode = (0.1, 0, 0, 0)
# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%', shadow=True)
plt.axis('equal')
plt.show()
```

Code: 64. Matplotlib_pie.py

plottaggio di grafici a torta (2)

- **pandas.plot()**



Code: 64. Matplotlib_pie.py