UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Corso di Laurea in Ingegneria Clinica e Biomedica
Informatica Medica I

IL LINGUAGGIO PYTHON PER L'ANALISI DEI DATI: ARRAY E DATA VISUALIZATION

Prof. Sara Renata Francesca Marceglia

Librerie Python per l'analisi dei dati

Python toolboxes/libraries per data science:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Python toolboxes/libraries per data visualization:

- matplotlib
- Seaborn

Le librerie mettono a disposizione METODI e quindi vanno importate e nominate

Importare le librerie

```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

I nomi delle librerie vengono poi utilizati per chiamare i metodi messi a disposizione dalla libreria stessa

Python Libraries for Data Science

NumPy:

- Introduce gli oggetti che permettono la gestione di array multimensionalie matrici (Matlab e R like) e le funzioni che permettono di effettuare operazioni matematiche e statistiche su di essi

- Permette la vettorizzazione di operazioni matematiche e quindi migliora la performance in termini di tempo di esecuzione rispetto all'uso delle liste

- Diverse librerie python sono costruite su NumPy

Link: http://www.numpy.org/

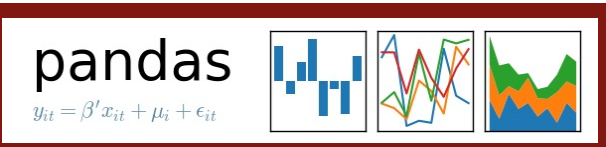
Python Libraries for Data Science

SciPy:

- Mette a disposizione gli algoritmi dell'algebra lineare, equzioni differenziali, integrazione numerica, ottimizzazione, statistica, etc
- Parte dello SciPy Stack
- Costruita su Numpy

Link: https://www.scipy.org/scipylib/

Python Libraries for Data Science



Pandas:

- Introduce le strutture dati (e le relative funzioni) per lavorare con dati in formato tabellare(simile alle Series e Data Frames in R)
- Mette a disposizione i metodi per la manipolaizone dei dati: filtraggio, segmentazione, modifica della struttura, etc.
- Permette la gestione dei dati mancanti

Link: http://pandas.pydata.org/

Python Libraries for Data Science

SciKit-Learn:

- Mette a disposizione algoritmi di machine learning (alcuni sono già disponibili in NumPy): classification, regression, clustering, model validation etc.

- Basato su NumPy, SciPy e matplotlib

Link: http://scikit-learn.org/

Python Libraries for Data Science

matplotlib:

- Libreria di grafica python 2D
- Contiene un insieme di funzioni simile a MATLAB
- line plots, scatter plots, barcharts, istogrammi, pie charts etc.

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- Basata su matplotlib
- Fornisce un'interfaccia di alto livello per ottenere dei grafici di buona qualità
- Simile alla libreria ggplot2 in R

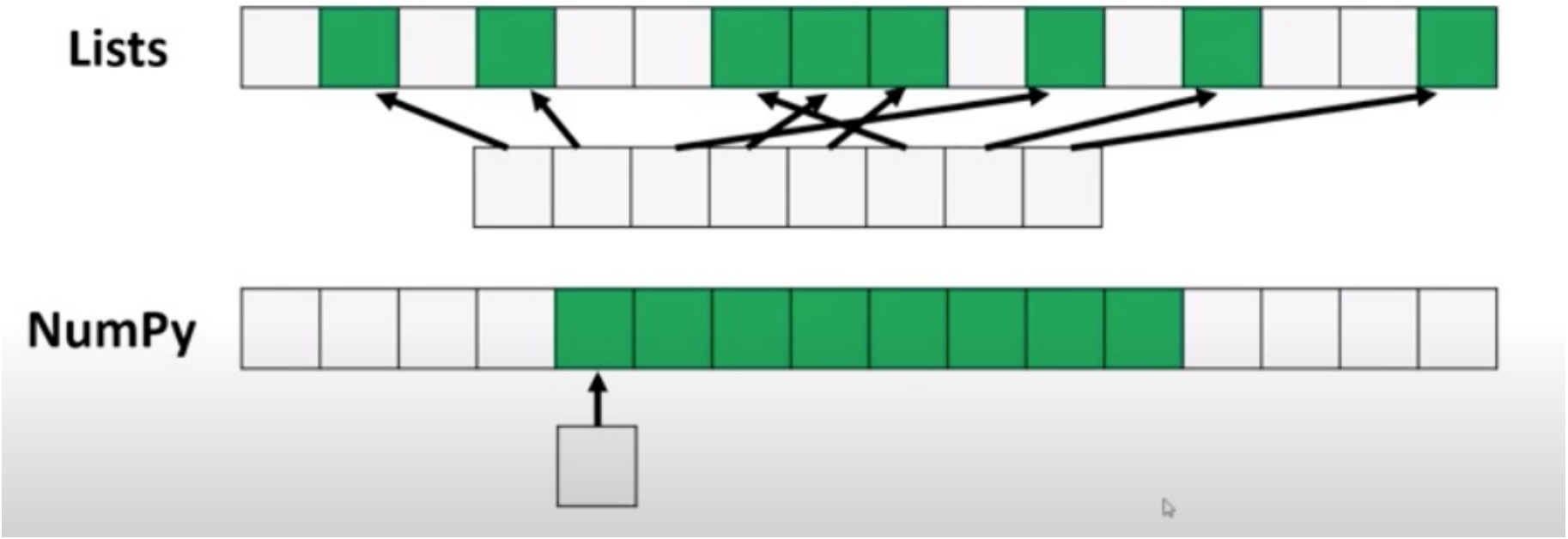
Link: https://seaborn.pydata.org/

Numpy: caratteristiche

- L'utilizzo di funzioni e metodi che agiscono su vettori e matrici evita l'utilizzo di loop (in particolare ciclo for che risulta molto lento su array e matrici NumPy)
- Nasce per fornire alte prestazioni
- In NumPy gli array sono gestiti in memoria in modo molto più efficiente rispetto alle liste
- Permette la gestione di file di grandi dimensioni
- NumPy ha varie parti scritte in C, cosa che rende un codice NumPy più veloce dell'analogo in Python puro

Numpy: caratteristiche

- Gli array sono memorizzati sempre in celle di memoria contigue e hanno meno metadati, in quanto il tipo di dato degli array è fissato → le operazioni di I/O di array sono significativamente più veloci



Organizzazione di NumPy

Sub-Packages	Purpose	Comments
core	basic objects	all names exported to numpy
lib	Addintional utilities	all names exported to numpy
linalg	Basic linear algebra	LinearAlgebra derived from Numeric
fft	Discrete Fourier transforms	FFT derived from Numeric
random	Random number generators	RandomArray derived from Numeric
distutils	Enhanced build and distribution	improvements built on standard distutils
testing	unit-testing	utility functions useful for testing
f2py	Automatic wrapping of Fortran code	a useful utility needed by SciPy

Array in NumPy: Metodo np.array

- Il metodo “array” è messo a disposizione dal modulko NumPy → deve essere chiamato a partire dall’alias del modulo stesso

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

- Gli array possono essere create a partire da altri tipi di dato

```
>>>a=np.array(lista)
>>>b=np.array(tupla)
```

Caratteristiche degli array

- Tipo di dato definito: nella divhiarazione dell'array è possibile definire il tipo di dato

  >>> a = np.array([1,2,3,4,5,6,7], *datatype*)

- Poichè il datatype è messo a disposizione da NumPy e non è nativo di Python, se si vuole utilizzare uno specifico datatype deve essere chiamato come metodo di np

  np.datatype

- Allocazione di memoria contigua
- Gestione più veloce rispetto a una Python List
- Si possono applicare operatori matematici e di algebra lineare
- Sono analoghi agli array di Matlab

Tipi di dato

Type	Description
bool	Boolean
int	Platforma integer
int8	Byte (-128,127)
int16	Integer (-32768,32767)
int32	Integer (-2147483648, 2147483647)
int64	Integer (-9223372036854775808, 9223372036854775807)
uint8	unsigned integer (0,255)
uint16	unsigned integer (0,65535)
uint32	unsigned integer (0,4294967295)
uint64	unsigned integer (0,18446744073709551615)
float	float64
float32	Single precision float
float64	Double precision float
complex	complex128
complex64	Complessi con 2 32-bits float
complex128	Complessi con 2 64-bits float

Esempio:

a=np.array([1,2,3,4,5,6,7],**np.int8**)

ATTRIBUTI DELLA CLASSE ARRAY

ATTRIBUTO	SIGNIFICATO
a.dtype	Ritorna il tipo di dato dell'array
a.flat[i]	Iteratore monodimensionale. Ritorna il contenuto dell'array nella posizione i-esima, valutata come se l'array fosse monodimensionale
a.nbytes	Ritorna il numero di bytes occupati dall'array
a.Itemsize	Ritorna l'occupazione di memoria di ogni singolo elemento dell'array
a.ndim	Numero di dimensioni dell'array
a.size	Numero totale di elementi dell'array
a.shape	Ritorna il numero di elementi in ogni dimensione dell'array (forma dell'array)

METODI DI CREAZIONE DEGLI ARRAY

Metodi del modulo Numpy (accessibili come **np.XXX**)

METODO	DESCRIZIONE
np.ones(m,n)	Restituisce una matrice di 1 di dimensione m x n
np.zeros(m,n)	Restituisce una matrice di 0 di dimensione m x n
np.full (m,n,k)	Restituisce una matrice di numeri k di dimensione m x n
np.full_like (np.array,k)	Restituisce una matrice di numeri k di dimensione uguale a np.array
np.identity(m)	Restituisce una matrice identità (quadrata) m x m
np.random.rand(m,n)	Restituisce una matrice di numeri casuali di dimensione m x n
np.repeat(np.array,n, axis = k)	Ripete n volte ciascun element di np.array lungo la dimensione k
np.arange(start,stop,step)	Genera un array di numeri da “start” a “stop” con un passo di “step”

ACCESSO AGLI ELEMENTI

- L'accesso ai singoli elementi si effettua come per le liste

```
>>>a=np.array ([[1,2,3],[4,5,6],[7,8,9],[1',11,12]])
>>>array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12]])
>>>a[1][2]
6
>>>a[1,2]
6
```

Attenzione: si può utilizzare anche la notazione negative (l'elemento –i corrisponde all'elemento che si trova in posizione **(a.size)-i**

- L'accesso a righe/colonne si effettua con l'operatore ":"
  - **a[i,:]** → accesso arra riga i-esima
  - **a[:,i]** → accesso alla colonna i-esima
- L'accesso a range si effettua con la notazione "i:j"
  - **a[i:j,k]** → accesso agli elementi da i a j-1 della colonna k
  - Si può aggiungere anche uno step **"i:j:step"** per accedere agli elementi da i a j-1 saltando di «step»

METODI DI MANIPOLAZIONE DELLA CLASSE ARRAY

METODO	DESCRIZIONE
a.fill (k)	Riempie tutto l'array con il numero k
a.sort(axis = k)	Ordina l'array a secondo l'asse k
a.transpose()	Restituisce la matrice trasposta
a.reshape((m,n))	Restituisce un array che contiene gli stessi elementi di a ma configurati in una matrice m x n (attenzione: fiunziona solo se il numero degli elementi è compatibile con la nuova matrice)
np.vstack([a1,a2])	Restituisce una matrice in cui gli array v1 e v2 sono messi in colonna (attenzione: devono essere compatibili dimensionalmente)
np.hstack([a1,a2])	Restituisce una matrice in cui gli array v1 e v2 sono messi in riga (attenzione: devono essere compatibili dimensionalmente)

ASSEGNAZIONE E COPIA, GENERAZIONE DA FILE

- Come per le liste, la creazione di un array mediante uguaglianza con un altro array determina il fatto che i due array siano uguali anche come oggetti → la modifica dell'uno si riversa anche sull'altro

- È meglio utilizzare il metodo «copy» che è lo stesso delle liste

  b=a.copy

- Gli array possono anche essere caricati da file di testo

  New_array = np.genfromtxt('nomeFile.txt', delimiter = '*tipoDelimiter*')

  Dove tipoDelimitatore può essere «,» «;» «\t»

FANCY INDEXING

- La selezione di un array può essere effettuata attraverso un array di indici

```
>>> z=np.arange(20,80,5)
 array([20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75])
>>>z[[3,5,7]]
 array([35, 45, 55])
```

Passo una lista che contiene gli indici degli elementi che voglio selezionare

- La selezione di un array può essere effettuata attraverso una maschera booleana

```
>>> z>50
 array([False, False, False, False, False, False, False,  True,  True,
True,  True,  True])
>>>z[z>50]
 array([55, 60, 65, 70, 75])
```

Passo come lista di indici la maschera booleana che ho ottenuto cercando I valori >50

OPERAZONI NUMERICHE

```
>>> a = np.array([1, 2, 3, 4])
>>> a + 1                    # somma di uno scalare
array([2, 3, 4, 5])
>>> a – 2                    # sottrazione di uno scalare
array([-1, 0, 1, 2])
>>> 3*a                      # moltiplicazione per uno scalare
array([3, 6, 9, 12])
>>> 2**a                     # potenza (base scalare)
array([2, 4, 8, 16])
>>> a = np.array([1., 2., 3., 4.])
>>> a/2                      # divisione per uno scalare
array([0.5, 1., 1.5, 2.])
```

Tutte le operazioni sono effettuate secondo la logica elementwise (un elemento alla volta)

OPERATORI ARITMETICI

- Anche gli operatori aritmetici tra array operano secondo la logica elementwise → gli elementi vengono utilizzati uno alla volta a coppie
- È necessario che le matrici siano di uguale shape

```
>>> z1 = np.full(12,7)
array([7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7])

>>>z+z1
 array([27, 32, 37, 42, 47, 52, 57, 62, 67, 72, 77, 82])
```

CONFRONTO TRA ARRAY

- Gli operatori logici di confronto operano elementwise e ritornano una maschera booleana che riporta il risultato del confronto

```
>>>z/10==z1
array([False, False, False, False, False, False, False, False, False,
       False,  True, False])
```

- Per operare un confront tra array si usa il metodo np.array_equal

```
np.array_equal(z,z1)
```

ALTRE OPERAZIONI

```
>>> b = np.sin(a)
>>> b = np.arcsin(a)
>>> b = np.sinh(a)
>>> b = a**2.5  # power function
>>> b = np.log(a)
>>> b = np.exp(a)
>>> b = np.sqrt(a)
```

Operazioni trigonometriche, logaritmi, esponenziali etc → **Operano tutte elementwise**

OPERAZIONI STATISTICHE DESCRITTIVE

OPERAZIONE	DESCRIZIONE
np.mean(a)	Media dell'array a
np.median(a)	Mediana dell'array a
np.std(a)	Deviazione standard dell'array a
a.min()	Ritorna il minimo di a
a.max()	Ritorna il massimo di a
a.argmin()	Ritorna l'idice del minimo di a
a.argmax()	Ritorna l'indice del massimo di a
np.corrcoef(a1,a2)	Matrice di correlazione tra a1 e a2

Per ottenere tutte le statistiche descrittive di un array si può utilizzare la **funzione describe del modulo stats di SciPy:**

```
import scipy.stats as spt
spt.describe(a)
```

OPERAZIONI DI ALGEBRA LINEARE

- NumPy mette a disposizione anche operatori per algebra lineare
- Esiste un oggetto di tipo matrix che eredita da array e che mette a disposizione alcune operazioni, ad esempio:
  - .T trasposta
  - .H coniugata trasposta
  - .I inversa

- Rimanendo con gli oggetti array, il prodotto matriciale si ottiene con la funzione “matmul”

np.matmul(a1,a2)

a1.shape = m x n, a2.shape = n x h → risultato m x h

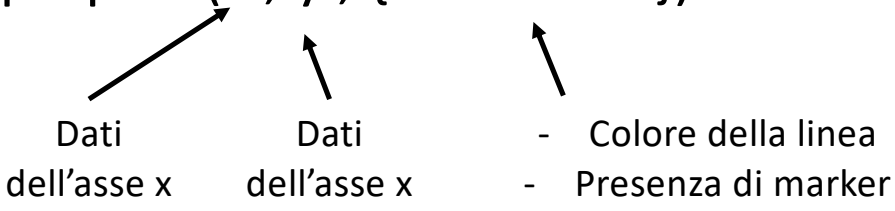
VISUALIZZAZIONE DEI DATI: GRAFICI A LINEA

- Grafici semplici → messi a disposizione dal modulo pyplot di matplotlib

```
import matplotlib.pyplot as plt
```

- Metodo plt.plot permette la visualizzazione di vari tipi di grafico

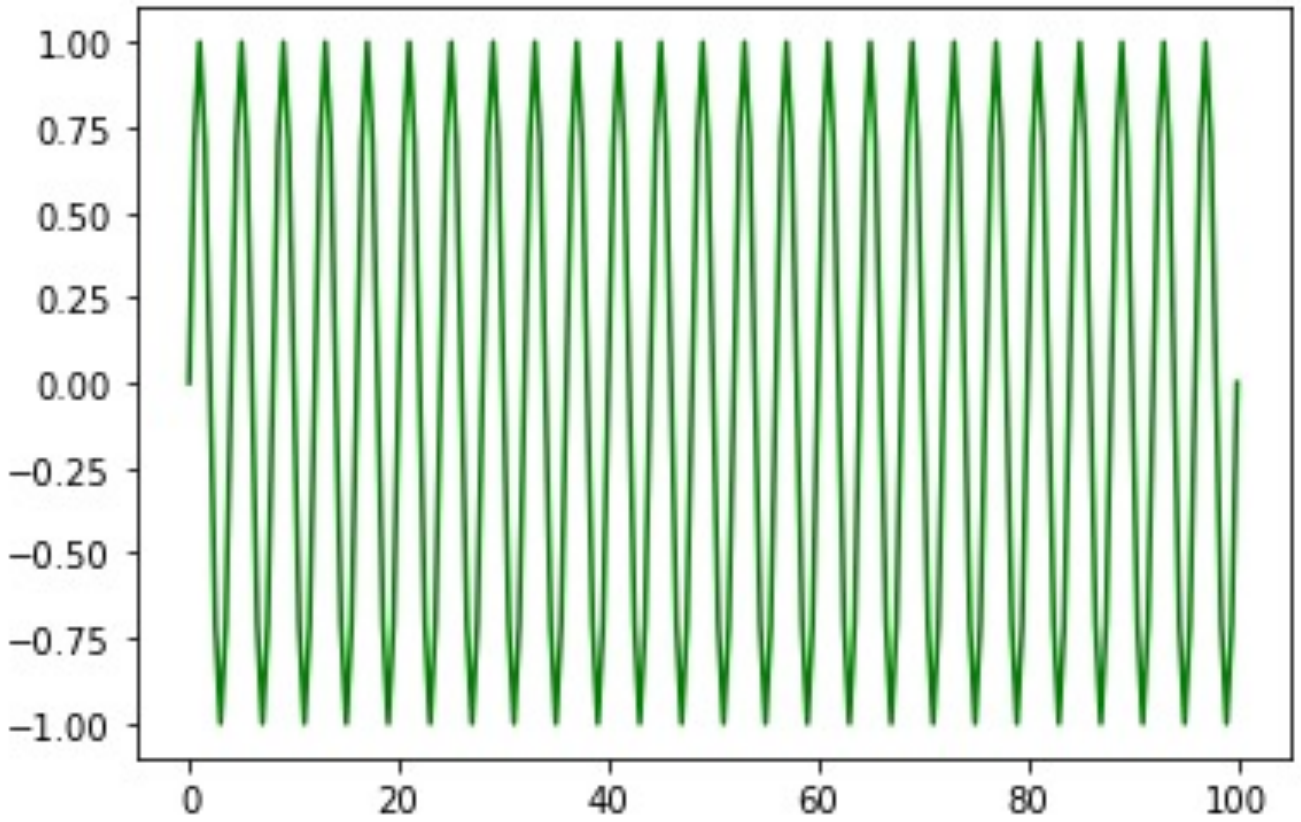
```
plt.plot (x , y , {descrittori})
```



- La visualizzazione viene attuata dal metodo plt.show() che va chiamato subito dopo il metodo plot

ESEMPIO

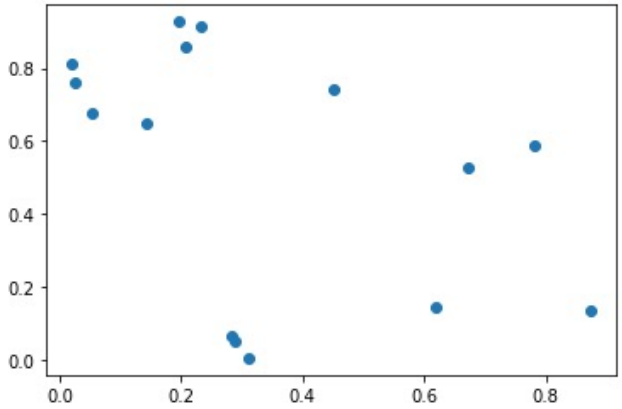
```
t = np.linspace(0,100,201)
y=np.sin(np.pi/2*t)
plt.plot(t,y,'green')
plt.show()
```



VISUALIZZAZIONE DEI DATI: SCATTER PLOT E ISTOGRAMMI

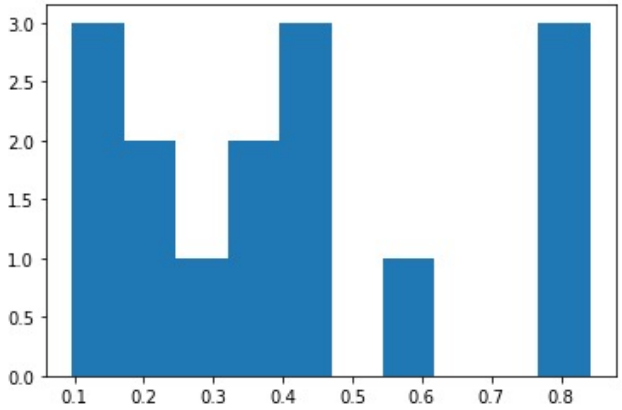
- Scatterplot → metodo plt.scatter (x,y, {descrittori})

```
a=np.random.rand(15,1)
b=np.random.rand(15,1)
plt.scatter(a,b)
plt.show()
```



- Istogrammi → plt.hist(x, bins = nbins) dove nbins è il numero di suddivisioni che voglio rappresentare

```
plt.hist(a,bins=10)
plt.show()
```



SUBPLOT

```
t = np.linspace(0,100,201)
y=np.sin(np.pi/2*t)
y2=np.sin(np.pi/8*t)
```

```
ax = plt.subplot()
ax.plot(t,y)
ax.plot(t,y2,'red')
plt.show()
```